Eindhoven University of Technology
Department of Biomedical Engineering
Computational Biology

An object-oriented framework to model glucose and insulin dynamics

Master Thesis

Author:
R.P.G. Bossink

Supervisors:
prof.dr. P.A.J. Hilbers
prof.dr.ir. N.A.W. van Riel

Committee:
prof.dr. P.A.J. Hilbers
prof.dr.ir. N.A.W. van Riel
dr.ir. S. Loerakker

Eindhoven, October 2019 - October 2020

Abstract

In patients with diabetes the glycemic regulatory system malfunctions. These regulatory systems are responsible for mediating the energy supply in the human body. To study the underlying mechanisms of diabetes, multiple dynamic models representing the glycemic regulatory systems have been developed. One of these models is the Eindhoven diabetes education simulator (eDES), created by Eindhoven University of Technology. Over time, the eDES evolved and grew into multiple versions with their own specific abilities and distinct features, making each version suitable for specific data sets of interest. Initially, the eDES model is used for educating diabetes patients about their disease. However, currently the model is used as a simulation tool for scientific research. We set out to accomplish multiple goals: 1. Create a platform where multiple eDES versions can be accessed. 2. Create a flexible and object-oriented program. 3. Develop the model in a modular and extendable fashion.

Multiple versions of eDES are condensed into a single platform created in Python (eDESpy). Merging the different versions into one program, enables easy accessibility of the different models in one location, along with the specific benefits of these models. To increase functionality and usability, the code was equipped with a new intuitive Graphical User Interface (GUI). This GUI enables facile adjustments of patient data, parameter sets, along with different nutrition and medication. Additionally, the GUI provides graphical visibility of fluxes, states and patient data. The input information - patient parameters, nutrition and medication - are stored in a newly created SQL database (DB). With this DB, it is no longer necessary to manually insert the previously mentioned patient information into the code, all information can be stored and retrieved instantly from the DB. The extensions of the GUI and SQL DB, enable the previously impossible comparison of multiple patients during a single simulation. Furthermore, the eDESpy code is constructed in a modular fashion resulting in an easily adaptable and extendable model. The eDESpy model is created with multiple compartments. Each compartment contains code describing specific organ functionality, metabolic processes, and processes like administration of medication or nutritional consumption. These compartments can be (de)activated, with use of the GUI. This is a useful new feature to simulate organ failure or certain diseases. Moreover, this feature shows the desired flexibility and modularity of the program.

It is known that physical exercise (PE) can significantly influence the glycemic regulatory system and have a positive impact on the overall health of diabetic patients. Modelling PE could provide new insights in the response of the glycemic system. The addition of PE modules shows the enhanced extendability of eDESpy and increases the overall completeness of the model. Therefore, two new PE modules are added onto the model.

In the DB, patient glucose and insulin data is saved. However, patient specific parameters, which are necessary to compute the model, were not available. Therefore, a parameter estimation is carried out using the eDESpy model, fitting the model for each patient to blood glucose and insulin data accordingly, creating a patient specific model. These personalized parameter sets can be referred to as the concept of virtual patients (VP). All VPs are stored in the created SQL DB, generating a virtual patient population (VPP). With this VPP, in silico tests can be done. An example of these tests is the clustering of diabetic patients into specific groups. Patient clustering could lead to new insights. In this research is experimented with clustering algorithms to separate different types of diabetes patients based on their estimated parameter sets.

Acknowledgements

After a year of hard work, this thesis is finished. Looking back I learned a lot of new things from many people. Without those people this research would not have been possible, therefore, I would like to express my gratitude to them.

First, I would like to thank my main supervisor Peter Hilbers for his support and open discussions. Peter helped me shape the eDESpy model and adviced me on programming with Python. Next, I want to thank Natal van Riel, for his constructive feedback and sharing many interesting articles. The two of you kept challenging me and inspired me to keep extending and improving the possibilities of the eDESpy model.

I would like to express my gratitude to the graduation committee, Peter Hilbers, Natal van Riel and Sandra Loerakker. Thank you for reading and grading my thesis. I hope you found the thesis interesting and I look forward to discuss my findings with you.

Furthermore, I want to thank everyone from the cBio group for the warm welcome I received! In particular Shauna O'Donovan and Gizem Aktas, who were also working on the eDES project. With them I could discuss the details of the eDES MATLAB versions. Additionally, I want to thank my neighbours in the cBio lab, David Lao Martil and Weizhou Xing, for the nice conversations.

Additionally I want to thank my family and girlfriend for their support and believe. In particular, I want to thank my dad for asking critical questions and his advice on how to approach certain programming difficulties.

Finally I would like to thank all of my friends, from both Wijchen and Eindhoven for their mental support. In particular Stijn Hofstraat for providing me with desired feedback on my writing. I am very grateful to all the people surrounding me.

Cheers,
Robbert

Table of Contents

Chapter 1

Introduction

Diabetes mellitus is a metabolic disease in which the complex regulatory systems that mediate the energy supply in the human body malfunction. The disease is noncommunicable and predominantly existing in high income countries. However, prevalence of diabetes is firmly increasing allover the world, this increase is most noticeably in middle-income countries [1]. Estimated 425 million people, 8.8 percent of the global population, suffer from diabetes. In 2016, 1.6 million people died because of complications associated with diabetes, making diabetes one of the leading causes of death worldwide [2]. According to the World Health Organization the number of diabetes patients will rise to 629 million in 2045. Furthermore, the short-term and long-term consequences of diabetes have severe impact on the patient's quality of life, for example the worldwide increase of diabetic retinopathy [3][4]. Diabetic retinopathy, is a leading cause of vision-loss and the fifth most common cause of preventable blindness world wide [5]. The economical aspect of diabetes is extensive and costs can be separated in two categories. Category one, direct medical costs, such as hospital based services, medication and supplies. Category two, indirect medical costs, in the form of reduced productivity and absence at work. In 2017, the average total costs of a diabetic patient in the United States was estimated to be approximately 14,200 dollars a year [6]. The high morbidity rates, decrease in quality of life and increasing costs stress the relevance and urgency for additional diabetes research.

1.1 Glucose and insulin dynamics

1.1.1 Glucose metabolism

Carbohydrates are the main energy source of the human body. In the process of digestion sugar complexes are reduced to monosaccharides. Digestion starts in the mouth with the enzyme salivary α-amylase, this enzyme is produced by the salivary glands. With a hydrolyses reaction, α-amylase cleaves the carbohydrates α-1,4 bonds, resulting in shorter saccharide chains [7]. In the stomach the salivary α-amylases are inhibited by the acidic environment. The majority of the carbohydrate digestion process takes place in the small-intestines. In the duodenum, additional α-amylase is added through pancreatic secretion [8]. In order to cross the cell membranes of the intestinal epithelial cells, the sugar molecules have to be monosaccharides. Brush border hydrolases are bounded to the lumenal plasma membrane of absorptive enterocytes. These brush border hydrolases are able to dismantle specific glycosidic bindings, into monosaccharides. An example is sucrase, which can separate sucrose into two monosaccharides, fructose and glucose. Through active transport, glucose is absorbed into enterocytes, with use of the SGLUT-1 sodium-dependent hexose transporter. GLUT-2, another hexose transporter, transports glucose to the basolateral membrane [7]. Through passive diffusion the glucose is transported into the bloodstream. Glucose is the main energy form in the human body [9].

When glucose is adopted from the small intestines into the blood stream, there are multiple options: Glucose can be converted to glycogen for energy storage (glycogenesis). This happens during hyperglycemia, when the levels of glucose in the blood are elevated. Alternatively, glucose is transported through the blood stream to supply additional energy to the peripheral tissue [10]. Situations in which the concentration of glucose in the blood is too low, are called hypoglycemia. The human brain and other tissues are in constant need of glucose, therefore, hypoglycemia is dangerous and can result in tissue damage [11]. During hypoglycemia, due to glycogenolysis, stored glycogen is converted to glucose to increase blood-glucose levels [9]. A schematic overview of these processes is displayed in Figure 1.1.

As previously stated, glucose is the main form of energy in the human body. However, to free its chemical energy, glucose needs to be converted into adenosine triphosphate (ATP). This process is called cellular respiration, which consists of glycolysis, the citric acid cycle and the electron transport chain. This is the cell's most efficient method of producing ATP [7]. In addition, human cells have other less efficient cycles to produce ATP. These cycles typically require different energy sources instead of glucose. An example of an energy source associated with diabetic ketoacidosis are fatty acids, this will be elucidated later [12].

Glycogenesis takes predominantly place in the liver and skeletal muscles when blood-glucose levels are sufficiently high [10], as can be seen in Figure 1.1. During glycogenesis, hexokinase phosphorilates glucose into glucose-6-phosphate. Next, phospho-glucomutase adjusts the molecule into a glucose-1-phosphate. UDP-glucose pyrophosphorylase transfers the molecule into an UDP-glucose. Glycogenin is able to create short glycogen chains, due to the formation of α(1-4) bonds. Glycogen synthase, followed by glycogen branching enzymes, are able to form glycogen. Glycogen is a branched polymer consisting of many glucose residues, and therefore efficient in the storage of energy [7]. In the absence of glucose, cells need to prevent metabolic starvation and therefore need to exploit alternative energy sources. In this metabolic starvation state, the cell initiates lipolysis and fatty acids are released from adipose tissue. Furthermore, there is no longer inhibition of fatty acid transport into the mitochondria. Inside the mitochondria beta oxidation is occurring. Beta oxidation reduces the fatty acid carbon chain into two carbon units, these small carbon units are used by the citric acid cycle. However, due to the significant amount of carbon units, the carbon units can fuse into acidic ketone bodies.

During hypoglycemia or in case the body needs additional energy, glycogenolysis and gluconeogenesis takes place, see Figure 1.1. Both processes are initiated by glucagon and are able through a cascade to endogenously produce glucose, as will be discussed in the next Section. In glycogenolysis, previously stored glycogen is converted back to glucose. Glycogenolysis starts with the enzyme glycogen phoshorylase, that phosphorylases the α(1-4) bonds, resulting in glucose-1-phosphate. However, for the metabolism process a glucose-6-phosphate is needed. Therefore, phosphoglucomutase is able to exchange the phosphate from the 1-carbon to the 6-carbon, providing glucose-6-phosphate [7, 13]. Next the glucose-6-phosphate needs to be dephosphorylated for release into the blood stream. In the liver this is done by glucose-6-phosphatase, a transmembrane enzyme in the endoplasmic reticulum. At this point the glucose can be transported into the blood stream [14]. Furthermore, there is the possibility of generating glucose through gluconeogenesis, which occurs during a fasting state. Gluconeogenesis converts non-carbohydrate carbon substances, such as lipids, amino-acids and pyruvates, into glucose [7]. The process of gluconeogenesis takes mostly place in the liver and kidneys. The enzyme catalyzed pathway is comparable with the reversed reaction of glycolysis. The amounts of glucose produced by gluconeogenesis are low compared to glycogenolysis. However, after depletion of the glycogen storage, gluconeogenesis can last for days [9, 15].

1.1.2 Endocrine regulation of blood-sugar homeostasis

In the previous section the processes to acquire, store and release glucose are explained. This section will elaborate on endocrine processes to regulate glucose dynamics. Blood-sugar levels are controlled by multiple hormones and neuropeptides, which are released by the pancreas, liver, intestine, adipose and muscle tissue [9]. An overview of these processes is shown in Figure 1.1.

The pancreas possesses endocrine glands, called the islets of Langerhans. These islands contain multiple secretory cells, only the for this research relevant secretory α- and β-cells are further discussed. α-Cells produce glucagon, while β-cells secrete insulin, proinsulin, c-peptide and amylin. Insulin is the most important hormone in glucose metabolism, whereas glucagon is an antagonist of insulin. [9].

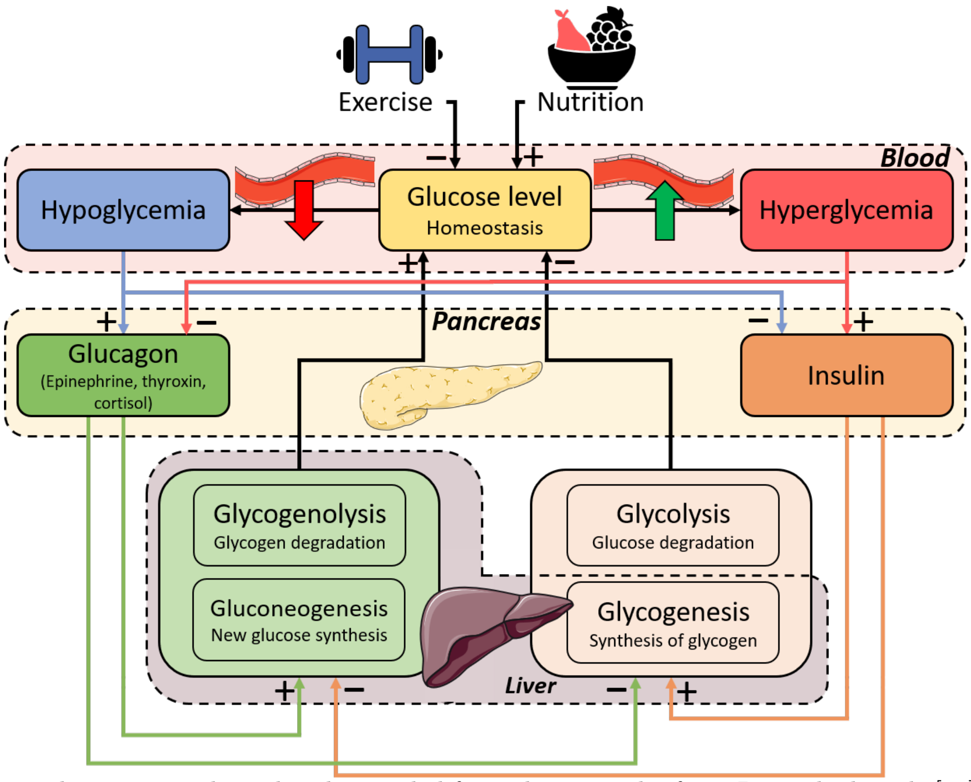


Figure 1.1: This Figure, adapted and extended from the research of van Rozendaal et al. [16], provides a schematic overview of the hormones and processes that regulate and influence glucose homeostasis. Stimulation of processes is indicated with a plus sign. Whereas, the inhibition of processes is visualized with a minus sign. To improve the readability of the figure the fluxes are shown with different colors, to stress the origin of the flux. The organs where the processes, conditions, or hormones, are located or produced, are shown in the figure.

The α- and β-cells are able to assess the amount of glucose in the bloodstream, and secrete glucagon and insulin accordingly to the anabolic or catabolic state. Therefore, when the glucose levels in the blood decrease, the α-cells are triggered to secrete additional glucagon into the blood stream. This increase of glucagon leads to the degradation of glycogen to glucose, resulting in elevated blood-glucose levels [17]. Oppositely, if blood sugar levels are high, additional insulin is synthesized and released in the bloodstream. During the synthesis of insulin equimolair amounts of c-peptide are produced and released. This peptide has a longer half-life compared to insulin, providing a more stable test window, which will recur in Chapter 2.3 [18]. The increasing concentration of insulin in the blood, results in decreasing blood glucose levels. Insulin induces increased uptake of glucose in insulin-dependent tissues, such as muscle and adipose tissue. Naturally, non-insulin dependent tissues, e.g. brain tissue and erythrocytes, remain unaffected by the described insulin increase [19]. Glucose is consistently transferred to most tissues via the GLUT-1 transporter. Whereas insulin-dependent tissues, e.g. muscle and adipose tissue, use the GLUT-4 proteins for rapidly boosting the glucose concentration in the cell if necessary [7, 14]. Insulin dependent cells have transmembrane insulin receptors (IR). These receptors are part of an extensive class of tyrosine kinase receptors. When insulin binds to the receptor, protein cascades are activated. The involved cascades are: translocation of the previously discussed glucose transporter (GLUT-4), resulting in an increased influx of glucose; synthesis of glycogen; glycolysis and the synthesis of fatty acids [7].

Conclusively, the endocrine glucose metabolic system is regulated through multiple communication mechanisms. In healthy people the previously explained carbohydrate metabolic system is in homeostasis. However, these circumstances do not apply on patients with diabetes as will be discussed in Section 1.2.2 [9].

1.1.3 Glucose metabolism during physical exercise

It is widely known that regular physical exercise (PE) has a positive effect on the overall condition of the human body. PE decreases the risk of obtaining disorders like: cancer, hypertension, obesity, cardiovascular, metabolic diseases and more [20]. Besides the fact that PE makes humans more vital, PE also improves mental health significantly [21]. It is relevant for humans to remain active, in order to stay healthy. Even when patients are diagnosed with a disease, increasing activity, could reduce and even minimize the complications of certain diseases. In diabetes type 2 patients it is perceived that an active lifestyle could significantly reduce the severeness of the disease [1]. Being active increases glucose tolerance along with insulin sensitivity. During PE glucose uptake is possible without the presence of insulin [22]. Diabetic patients can use this to lower their bloodsugar levels, without using additional medication.

During PE the demand of glucose in the contracting skeletal muscles cells is increasing. The sympathetic nervous system is activated, and multiple hormonal processes are induced. However, only the for this study relevant processes are mentioned. Epinephrine is released and stimulates the following processes: glycogenolysis in the liver and muscle tissue, an increase of the heart rate, and vasodilation in the liver and skeletal muscles. Resulting in an increased blood flow towards the muscle. Therefore, elevating the delivery of glucose into the skeletal muscle tissue and decreasing the blood glucose levels. To increase blood glucose levels the hormone glucagon is released, inducing gluconeogenesis and glycogenolysis in the liver, as can be seen in Figure 1.1. Furthermore, glucagon stimulates lipolysis in the adipose tissue. During PE, the synthesis of glycogen is inhibited. The focus shifts from storage, to the release of energy [23]. Glucose transport into the cell is enhanced by increasing the amount of GLUT4 transporters in the cell membrane. Often, the glucose transport via transmembrane proteins, is considered the limiting factor of the fastest possible glucose uptake [24]. When a subject regularly exercises the number of capillaries in the muscle increase, providing better blood flow towards the muscle. According to Borghouts et al. [25], up to two hours post exercise glucose uptake by insulin independent tissues is elevated, whereas, insulin sensitivity can be increased up to 16 hours post exercise.

1.2 Diabetes

1.2.1 General

In this chapter diabetes mellitus will be further discussed. As previously mentioned, it is estimated that 463 million people, 9.3 percent of the world population suffers from diabetes mellitus. Whereas in 2016, 1.6 million people deceased of the consequences of diabetes [1].

When the metabolic system (described in chapter 1.1) is not functioning properly, a metabolic disorder can be diagnosed. Diabetes mellitus is a specified group of metabolic disorders, concerning the glycemic metabolic system [26]. Diabetes is characterized by insulin resistance or deficiency, leading to a disequilibrium of the glycemic metabolic system. Diabetes mellitus can distinguished in two types (other types of diabetes are beyond the scope of this research). Type 1 diabetes mellitus (DM1), which is insulin dependent, and type 2 diabetes mellitus (DM2), which is non insulin dependent.

1.2.2 Type I versus type II diabetes

First DM1 is explained. Type 1 diabetes is mostly affecting young people and adolescents. The percentage of patients with DM1 is roughly around 10% of the patients diagnosed with diabetes. However, this percentage severely varies in different countries [27]. DM1 is an autoimmune disease, the pancreatic β-cells are destroyed through a T-cell mediated inflammatory response [28]. Therefore, due to absence of the β-cells, the production and secretion of insulin is no longer possible. Due to insulin deficiency insulin dependent tissues are unable to access their required amount of glucose, whereas the blood sugar levels remain elevated. Additionally, glycogenesis is obstructed by the absence of insulin. Gluconeogenesis, as an alternative source of energy is initiated, leading to diabetic ketoacidosis [7, 29]. Without insulin treatment, acidosis and dehydration result in damaged tissues and could eventually lead to a patients death [9]. Although DM1 mostly occurs in young people, it is possible for the autoimmune disease to develop later in life. This is called, latent autoimmune diabetes in adults (LADA) [29, 30].

In DM2 the β-cells have a decreased response to elevated blood-glucose levels. Furthermore, these non insulin dependent diabetes patients have increased insulin resistance. The combination of insulin resistance and decreased insulin release, results in elevated blood-sugar levels. Although, the importance of the two problems varies in each individual [9]. DM2, is in general related to decreased physical activity, genetics, diet, and excess weight [28][10]. Additionally, excess of dietary sugars, can induce insulin resistance, which could lead to DM2 development. Approximately 90% of the diabetic patients suffer from DM2. Therefore, DM2 is the most common type of diabetes [28, 31]. Patients with DM2, develop similar symptoms compared to patients with DM1. However, generally the symptoms of type 2 diabetes are less obvious and severe [31]. Antecedent to fully developed diabetes, a patient can be diagnosed with prediabetes. Prediabetes is an intermediate state of hyperglycemia, the glycemic parameters are elevated compared to normal values. However, the values are under the diabetes threshold [32].

1.2.3 Symptoms

Most symptoms of DM1 and DM2 are similar. However, the impact of these symptoms will differ per patient, type of diabetes, and if the disease is treated. Symptoms of diabetes are excessive thirst, frequent urination, nausea, sudden weight loss, infections, slow-healing wounds, and blurred vision, and fatigue [31]. The causes of these symptoms will be further discussed, starting with excessive thirst and frequent urination. Kidneys contribute to glucose homeostasis. in cases of prolonged hyperglycemia, renal glucose excretion increases equivalent to the increase of the blood glucose levels [33]. When the blood sugar levels exceed the renal threshold, homeostasis of the glycemic regulatory system is no longer possible, resulting in glucose excretion through the urine [34]. This process of glucose excretion in the urine is called glycosuria, this does not occur in healthy individuals. The excess glucose in the kidneys small tubules causes osmotic pressure, leading to retention of water. Furthermore, the osmolarity of the blood is likewise increasing and draining water from the interstitial space. The body is no longer able to reabsorb the water, leading to decreased extracellular fluid and increased urination [29, 35]. Over time there is the possibility of developing diabetic nephropathy, in which the kidneys are unable to function accordingly [36].

Meanwhile during the hyperglycemia, cells are in need of alternative energy sources, one of these sources is lipolysis. As explained in Section 1.1.1, this leads to generation of ketone bodies, resulting in increased acidity of the blood (diabetic ketoacidosis). The ketone bodies are reduced to acetone, departing the body in a gas state through the lungs. Therefore the breath of a patient with diabetic ketoacidosis will have a "fruity" smell. Ketoacidosis can also occur in patients with medication as a result of physical stress, e.g. during an infection. Ketoacidosis can lead to nausea and worsen the dehydration. Eventually if the acidosis is untreated it can become fatal for the patient [29].

Elevated blood glucose levels can damage the capillaries, this is called ischemia. High glucose levels reduce the vasodilator nitric oxide in the blood vessels. Deficiency of nitric oxide causes elevated blood pressure and reduces the vessels quality and diameter [37]. Resulting in limited blood supply to nerves and other tissues, preventing them from obtaining essential nutrients and oxygen. Diabetic peripheral neuropathy is a complication in which the peripheral nerves are damaged [29, 38]. First, only minor complications will occur in the extremities, due to reduced blood flow. These symptoms are tingling, cold or numb hands or feet. However, in a later stadium the consequences of peripheral neuropathy will become much severe. Patients are more vulnerable to infections and the infections become more serious, as the immune system is impaired [39]. Furthermore, peripheral neuropathy, ischemia and infections, can critically damage the cells in the foot. Eventually necrosis of tissue will occur and a diabetic foot ulcer is developed. Due to permanent damage, eventually amputation is required. It is estimated that 15% of all diabetic patients are troubled with diabetic foot ulcers [40]. Similarly, due to damaged capillaries in the, retina, brain and heart, chances increase for developing diabetic retinopathy, strokes and heart attacks [5, 41]

1.2.4 Diagnosis

Multiple tools are accessible to detect hyperglycemia. With these tools diabetes mellitus can be diagnosed. The possible diagnoses are: normal glucose tolerance (normal), pre-diabetes and diabetes. The threshold values for diagnosing diabetes, concerning the different tests are displayed in Table 1.1. These values are based on the findings of the World Health Organization [42, 43], as well as, standards of the American Diabetes Association [44].

The main key to diagnose diabetes are the blood glucose levels. These values can be measured under different circumstances and give an indication of the glycemic status of the patient. The glycemic values can be tested in fasting state, randomly or after an oral glucose tolerance test (OGTT). The fasting blood glucose test is executed after a night without nutrition (8-12 hours). These test results indicate the basal glucose levels during a fasting state. Furthermore, there is the possibility of a random blood test, if the glycemic values exceed the threshold value displayed in Table 1.1 it suggests the patient has diabetes. During an OGTT, in fasting state, the patient consumes a solution with 75 g glucose. The course of the blood glucose levels are measured for two hours, providing an impression on the bodies reaction to glucose [29].

One of the tests recommended by the American Diabetes Association is the glycated hemoglobin (HbA1c) test. As result of a non-enzymatic reaction, unbounded glucose molecules in the blood stream, are able to bind with the hemoglobin molecules in the erythrocytes. The glycated hemoglobin provides a useful indicator for the average plasma glucose concentration over the last two to three months [45]. This time span of two to three months is due to the life expectancy of the erythrocytes. Glycated hemoglobin used as a biomarker, specifies the percentage of the hemoglobin that is bounded to glucose. With this percentage the overall glycemic values can be calculated. When executed correctly, the advantage of this test is the indication of the average blood glucose levels for a prolonged period of time. However, this test is only reliable for patients without blood diseases or iron-deficiency anemia [29, 45].

Additionally, measuring c-peptide can be used as a biomarker for endogenous insulin production. As explained in Section 1.1.2, during synthesis of insulin equimolar amounts of c-peptide are produced. Therapeutically administered insulin does not increase the c-peptide levels in the blood or urine. Therefore, it provides an explicit understanding of the patients endogenous insulin production [46]. Furthermore, c-peptide has a significant longer half-life time, compared to insulin, providing a more stable testing window [18].

Furthermore, urine can be analyzed to confirm if glucose is secreted through the urine. As previously explained in Section 1.2.3, glycosuria is an indicator for diabetes mellitus. Analysis of antibodies can be used for distinguishing type 1 and type 2 patients. If the patients tests positive on the antibodies inducing the inflammatory response of the pancreatic β-cells, the patient is diagnosed with DM1.

Diagnosis:	normal	pre-diabetes	diabetes
Fasting plasma glucose [$mmol/l$]	< 5.6	$5.6 \leq to < 7.0$	$7.0 \leq$
Oral glucose tolerance test [$mmol/l$]	< 7.8	$7.8 \leq to < 11.1$	$11.1 \leq$
Random plasma glucose sample [$mmol/l$]			$11.1 \leq$
HbA1c [%]	< 5.7	$5.7 \leq to < 6.5$	$6.5 \leq$

Table 1.1: This Table, adapted from the American Diabetes Association [44] & data from the World Heath Organisation [43], displays the threshold values for multiple diagnostic diabetes tests. The possible diagnoses are a normal glucose tolerance (normal), pre-diabetes and diabetes. The considered measurements are the: HbA1c, fasting plasma glucose, random plasma glucose and oral glucose tolerance test (OGTT).

1.2.5 Treatments

As explained in section 1.1.2, the main problem of diabetes is the absence or ineffectiveness of insulin to regulate the glycemic regulatory system. Therefore, therapeutically administered insulin can aid in balancing the homeostasis of the system. DM1 patients always require insulin therapy, due to the absence of insulin production. Whereas for patients with DM2, increase of insulin sensitivity, is often of more importance, compared to additional insulin administration. The administration of insulin is done via bolus injection, or continuous with use of an insulin pump into the subcutaneous tissue. Insulin requirements are patient dependent and can change during different phases in life [29]. Various types of insulin can be distinguished based on their acting time, however, this research focuses solely on short and long-acting insulin. Short-acting insulin is usually administered 0 till 15 minutes before the meal. This type of insulin works very rapid, intended to reduce the postprandial glucose peak. Obviously, long-acting insulin works for a prolonged period of time. During fasting or interprandial periods, long-acting insulin is able to provide the required basal insulin levels [47]. Often a mixture of both short- and long-acting insulin is prescribed according to the patient.

However, insulin therapy only deals with the consequences of DM1, the β-cells are still deficient. To cope with this, clinically viable transplantation strategies are being developed. The strategies differ from complete pancreas transplantation and pancreatic islet transportation, to β-cell derived therapies. Essentially these techniques are combined with immune modulators, dealing with the autoimmune response in DM1 patients [48]. These treatments look promising, however, currently DM1 patients are still reliant on insulin therapy [49].

Metformin is a medicine commonly used for DM2 patients. According to the American Diabetes Association Metformin affects the blood sugar levels on multiple fronts: intestinal glucose absorption is reduced; insulin sensitivity is increased; inhibition of gluconeogenesis; and reduction of hepatic glucose production [44, 50]. The exact mechanisms behind Metformin are not completely understood, however, their positive effects on reducing blood glucose levels are evident [51].

Self-care is very important for patients with diabetes. Part of this self-care is monitoring glucose, these glucose results can aid the patient in making educated decisions about nutrition, medication or activity. Additionally, there is also the possibility of constant glucose monitoring, using a small sensor under the skin. As previously mentioned an unhealthy lifestyle of obesity and physical inactivity increases the chance of developing diabetes type 2 [52]. However, according to the World Health Organisation, abrupt changes towards a healthy lifestyle can "reverse" DM2 [1]. By any means, applying to both types of diabetes, a positive or negative lifestyle has a complementary effect on the severeness of the disease [53]. Moreover, diabetes patients have to predominantly administer the medication themselves. Therefore, it is very important to educate the patients about the disease and how to cope with it. Sugarvita, developed by the Technical University Eindhoven (TU/e) and Màxima Medical Centre, is such an educational tool to teach diabetic patients about their disease [54]. eDES is initially intended as the mathematical model behind Sugarvita.

1.3 Virtual patients

In silico research is considered to be a very rich tool for biomedical research [55]. According to Morrision et al. [56] "computational modeling is one tool to support faster more efficient regulatory approvals without sacrificing patient safety or the confidence in regulatory decisions." It could provide evidence-based advise to aid in the optimization of metabolic control in diabetes care [57]. Moreover, this optimization could be personalized to individual patients. By doing this virtual patients (VP) are created which could be used for in silico testing. According to the research of Salzsieder et al. [57], in silico diabetes research has many advantages. An analysis of the glycemic control can be executed, resulting in the weak-points of the patients glycemic system. When these weak-points are known, therapeutic adjustments to improve the glycemic control can be tested. Eventually, this could lead to personalized therapy recommendations [57].

In this research is, with actual patient data describing glucose and insulin dynamics over time, a personalized parameter set estimated. These parameters sets are unique and could be regarded as a metabolic fingerprint. Although this metabolic fingerprint is unique, patients with resembling metabolic fingerprints can be clustered into likewise patient groups. This could be useful for diagnosing, or the exchange of information or parameters between patients. E.g. it could be hypothesised that in a specific patient cluster the efficiency of certain type of medication is higher. Conclusive, links between patients could be found, that could not be found by simply looking at the patients medical records or simulation results. Clustering these VPs, into patient groups, with similar qualifications, could even accelerate the learning process. Whereas, gaps of missing patient information could be exchanged between patients in the same cluster. Furthermore, medication schemes could be tested in silico on an entire patient cluster.

1.4 Previous research and current situation

Multiple researches concerning the Eindhoven Diabetes Education Simulator (eDES) are conducted, in this chapter the current state of the research will be discussed. The model of eDES is initially based on the model of Dalla Man et al. [58], which is considered one of the most complete metabolic diabetes models. These models use parameters describing the exact specifications of the simulation. However, the eDES models have few parameter sets which are manually entered into the code of the program (hardcoded). The first version of the model, eDES 1.0, focuses mainly on the postprandial glucose and insulin dynamics. This version of the model is extended to eDES 1.1 by van Rozendaal et al. [16]. The 1.1 version is considered the core of the eDES models [59]. The next model is eDES 2.0, created by Maas et al. [54]. The reason for version 2.0 instead of 1.2 is the addition of c-peptide. As explained in Section 1.1.2 and 1.2.4, the measurement of c-peptide is used to determine the endogenous insulin production. Furthermore, c-peptide has a longer half-life time compared to insulin, providing a more stable test window for measurements [46]. The exact differences between the models are precisely described in Section 2.3. Both models have similar state compartments in which the glucose and insulin levels are monitored, the green boxes in Figures 2.1 and 2.2. However, there are some minor differences. Whereas, in the eDES 1.1 model, insulin can be temporary stored in the interstitial fluid, as can be seen in Section 2.1. In eDES 2.0, this is an insulin outflux leaving the system (see Section 2.3, Figure 2.2). The addition of the c-peptide flux, combined with an insulin sink instead of interstitial fluid compartment, is what separates eDES 2.0 from the previous versions of the model. The previously mentioned models are created in MATLAB and structured differently.

1.5 Aim and outline of this study

This research is composed out of multiple goals: 1. Create a platform where multiple eDES versions can be accessed. 2. Create a flexible and object-oriented program. 3. Develop the model in a modular and extendable fashion. To achieve these goals multiple steps are made.

Firstly, the existing eDES model is transferred from MATLAB to Python (eDESpy) to create a platform. The eDESpy model is covering multiple meals and different types of medication. A model managing single meals could be able to describe a more precisely trajectory of postprandial glucose and insulin dynamics. Whereas, models able to manage multiple meals, can include the amplifying effects which multiple meals have on each other. In this research those models are accommodated into a single program, which enables the operation of the models in the same way and simultaneously.

Furthermore, in Python this model is recreated in a new modular fashion. With this modular model, expansion of the currently existing models is becoming more accessible. The eDESpy model is created with multiple compartments. Each compartment contains code describing specific organ functionality, metabolic processes, and processes like administration of medication or nutritional consumption. Compartments in the eDESpy model can be switched on or off if necessary. Additionally a database (DB) behind the eDESpy model is created to counter the previously described stiffness of the hardcoded MATLAB models. This DB must be able to store: patient information, medication, nutrition, parameters, and patient data. With this new DB it is no longer necessary to manually type this information directly into the code.

On top of the eDESpy platform a Graphical User Interface (GUI) is build, to improve the usability and flexibility of the eDESpy model. In this GUI different versions of the model can be chosen and patients can be selected from the DB. Furthermore, parameter and event sets can be adjusted and stored into the DB. The GUI is a workbench like tool enabling visualization of differences in the simulations. In the GUI compartments can be (de)activated and the efficiency of these compartments can be adjusted. The fluxes and the output of the model are displayed in the GUI. In general, the GUI makes the features of the eDESpy model accessible at once, enabling the comparison different simulations.

To show the extendability of the model, PE modules are added to the model. In Section 1.1.3, is explained that PE has a significant influence on the glycemic regulatory system. Modelling PE could provide new insights in glucose and insulin dynamics and be a valuable extension of the model. Since, likewise medicinal use and nutrition, PE could be a part of the lifestyle of a diabetic patient.

The model parameters are patient specific and necessary to compute the simulation. With the patient data, consisting of points describing blood glucose and insulin levels over time, it is possible to estimate a patient specific parameter set. Therefore, parameter estimations are executed for each patient in the data set, to obtain personalized patient models. These personalized parameter sets, which could be regarded as a "metabolic fingerprint", are stored in the DB, creating a virtual patient population (VPP). It is hypothesised that these patients can be clustered based on similarities in their unique metabolic fingerprint, as can be seen in Section 4. In this research a clustering algorithm is tested to investigate the possibility to distinguish the patients. Clustering patients could be useful for the exchange of missing parameters between patients, or recognizing diverging (diabetic) disorders.

Chapter 2

Mathematical model

In the following chapter, the mathematical equations from the eDES 1.1 and eDES 2.0 model are explained. These equations are included into the eDESpy model. Therefore, it is useful to elaborate upon their origin and function. The complete set of differential equations, can be found in Appendix B.1. All parameters, constants and variables with their function and dimension are stored in the Appendix Tables B.1, B.2 and B.3. It is important to understand the mathematics for recreating, validating, and extending, the model. The extension of the model with PE modules is explained in Section 5. The mathematical conventions as introduced by van Rozendaal et al. [16] are used in this paper and displayed in Table 2.1.

symbol	**notation**	**interpretation**
	italic	variables and quantities
	roman	descriptive terms
X	upper case	concentration or mass of component X
x	lower case	flux of component X
$x^{\cdots}$	superscript	compartment or anatomical annotation
$x_{\ldots}$	subscript	further description
x_{in}^{Y}		appearance rate from compartment Y
x_{out}^{Y}		disappearance rate towards compartment Y

Table 2.1: A table containing the mathematical conventions used in this paper, as introduced by the research of van Rozendaal et al. [16]

2.1 Mathematics of a healthy glucose metabolic system

In the next section the mathematical model, for glucose metabolism in healthy people, will be further explained. In this chapter is elaborated on the mathematics used in eDESpy 1.1. The overall concept of eDESpy 1.1 is in agreement with eDESpy 2.0. However, there are some minor differences between the models which are discussed in Section 2.3.

The "healthy" part of eDESpy model can be regarded as the backbone of the program. The eDESpy model is based on the MATLAB model van Rozendaal et al. [16], which is partly based on the model of Dalla Man et al. [58]. The glucose regulatory system described in Chapter 1.1 consists of complex physiological processes. Based on these processes a simplified mathematical system is composed. This system covers the essential processes and is merged into one metabolic system. The physiological compartments of a metabolic system can be simplified and translated into four mathematical compartments. These state compartments are: the glucose mass in the gut, glucose in the blood plasma, insulin in the blood plasma, and insulin in the interstitial fluid. These are the main components of the system and are shown in the green boxes in Figure 2.1. Furthermore, all reaction velocity parameters ($k_{1,2,..,n}$) combined with the location (flux) on which they impact the model are also displayed in the same Figure.

Glucose dynamics in the gut. The first compartment, described by Equation 2.1, represents the glucose absorption in the gut. After the uptake of carbohydrate rich nutrition, the digestion process starts, see Section 1.1.1. After consumption the food enters the digestive tract. The inflow of carbohydrates through the nutrients into the gut is depicted by m_G^{gut}. The outflow of carbohydrates from the small intestines to the blood plasma is depicted as m_G^{pl}.

$$\frac{\mathrm{d}M_G^{\mathrm{gut}}}{\mathrm{d}t} = m_G^{\mathrm{gut}} - m_G^{pl} \tag{2.1}$$

$$m_G^{\mathrm{gut}} = D^{\mathrm{meal}} \sigma k_1^\sigma t^{\sigma-1} e^{-(k_1 t)^\sigma} e^{-(k_{18} t)} \tag{2.2}$$

Following will be shown a dissection regarding the construction of the differential equation concerning the glucose in the gut. The equation consists of two terms, Equation 2.2 describes the glucose influx from the gut into the stomach, Equation 2.3 calculates the outflux of glucose from the gut to the plasma. Equation 2.2 is based on the gastric emptying model of Elashoff et al. [60], k_1 and k_{18} are the velocity constants for the reaction. The shape factor of the equation is resembled by σ, which is displayed in Figure 2.1. The effect of differences in σ is shown in Appendix B, Figure B.1. D^{meal} is the total amount of carbohydrates present in the stomach. The exponential decay function describing the gastric emptying rate can be separated into two components: k_1 a component with a faster- and k_{18} a component with a slower-gastric emptying rate. Due to these multiple rates of gastric emptying, complex meals and OGTTs can be simulated.

$$m_G^{\mathrm{pl}} = k_2 M_G^{\mathrm{gut}} \tag{2.3}$$

In the intestines glucose is absorbed by the enterocytes and released into the bloodstream (m_G^{pl}). Equation 2.3 shows the outflow of glucose to the blood plasma which is linear dependent to the glucose mass (M_G^{gut}) in the gut.

Glucose dynamics in the blood plasma. Glucose concentration in the blood plasma increases due to the exogenous glucose production, along with, the endogenous glucose production. The exogenous glucose production rises with the uptake of the nutrition from the gut (g^{gut}). Whereas the endogenous glucose production by the liver (g^{liv}) is initiated by glycogen, as explained in Section 1.1.1. Combined are these two fluxes ($g^{gut} + g^{liv}$), responsible for the increase of the glucose in the blood plasma. Additionally, glucose is constantly devoured by human tissues. As is explained in Section 1.1.1, there are two possibilities for the uptake of glucose: glucose uptake by non insulin independent tissues ($g^{non \cdot it}$), which are erythrocytes and brain cells; or glucose uptake by insulin dependent tissues (g^{it}) e.g. liver, adipose or muscle tissue. Additionally, as can be seen coloured red in Equation 2.4, there is the possibility of renal secretion of glucose (g_{th}^{ren}). However, this will not occur in healthy subjects. Renal secretion is further discussed in Section 2.2.

$$\frac{\mathrm{d}G^{\mathrm{pl}}}{\mathrm{d}t} = g^{\mathrm{gut}} + g^{\mathrm{liv}} - g^{\mathrm{non \cdot it}} - g^{\mathrm{it}} - g_{th}^{\mathrm{ren}} \tag{2.4}$$

Next, the terms of the differential equation describing the glucose concentration in the blood plasma will be elaborated. The exogenous glucose increase (g^{gut}), displayed in Equation 2.5 depends on the mass of glucose in the gut (M_G^{gut}). Whereas, f is an unit conversion factor, v_G is the glucose distribution volume in the plasma and M^b resembles the patients body weight.

$$g^{\mathrm{gut}} = k_2 \frac{f}{v_G M^{\mathrm{b}}} M_G^{\mathrm{gut}} \tag{2.5}$$

The equation concerning the endogenous glucose production (EGP) of the liver is described by Equation 2.6, which is based on the work of Dalla Man et al. [58]. In this equation there is a term for the basal hepatic glucose production (g_b^{liv}). Additionally, the negative terms are able to reduce the EGP of the liver. Again k_9 and k_{10} are velocity parameters. If the plasma glucose levels are elevated ($G^{\text{pl}} > G_b^{\text{pl}}$) the overall hepatic glucose production will be reduced. Vice versa if the plasma glucose levels are lower compared to the basal glucose levels ($G^{\text{pl}} < G_b^{\text{pl}}$), hepatic glucose production is increased. However, g_b^{liv} also depends on the insulin concentration in the interstitial fluid compartment (I^{rem}). Combined with the β-cell sensitivity to glucose the EGP can be suppressed.

$$g^{\text{liv}} = g_b^{\text{liv}} - k_9 \left(G^{\text{pl}} - G_b^{\text{pl}} \right) - k_{10} \beta I^{\text{rem}} \tag{2.6}$$

The two terms, g^{it} and $g^{non \cdot it}$, resembling the utilization of glucose remain. Firstly, the utilization of glucose by insulin dependent tissues is discussed. Gottesman et al. [61], has shown that Michaelis-Menten kinetics are sufficient in describing the balance between the utilization and the blood plasma glucose concentration. Subsequently, the research of Dalla Man et al. [58], Lehmann et al. [62], with adaptations by van Rozendaal et al. [16], leaded to the following equation (Equation 2.7) describing the utilization of glucose:

$$g^{non \cdot it} + g^{it} = k_3 \Gamma_0 \frac{G^{\text{pl}}}{(K_{\text{M}} + G^{\text{pl}})} + k_4 \beta I^{\text{rem}} \frac{G^{\text{pl}}}{(K_{\text{M}} + G^{\text{pl}})} = \left(k_3 \Gamma_0 + k_4 \beta I^{\text{rem}} \right) \frac{G^{\text{pl}}}{(K_{\text{M}} + G^{\text{pl}})} \tag{2.7}$$

In Equation 2.7, K_{M} is the Michaelis-Menten constant responsible for the tissues uptake of glucose. Again, k_3 and k_4 are the reaction rate velocity parameters which can be found in the flowchart in Figure 2.1. Noticeably, the term containing the interstitial insulin concentration (I^{rem}) and β-cell sensitivity to glucose (β), is responsible for the glucose uptake by the insulin dependent tissues. Whereas, the other term, containing the glucose conversion factor (Γ_0), covers the non-insulin mediated glucose uptake.

Insulin dynamics in the blood plasma. Insulin is produced endogenously by the β-cells, which are located in the islets of Langerhans (pancreas), see Section 1.1.2. In healthy subjects this is the only possibility for the system to increase insulin blood plasma levels (i^{pnc}). Furthermore, there is a decreasing flux (i^{rem}), as can be seen in Equation 2.8. i^{rem} Transports insulin towards the remote compartment, which is explained later. The in red displayed short-acting (i^{sa}) and long-acting insulin (i^{la}), will be further elaborated upon in section 2.2.

$$\frac{\mathrm{d}I^{\text{pl}}}{\mathrm{d}t} = i^{\text{pnc}} - i^{\text{rem}} + i^{\text{sa}} + i^{\text{la}} \tag{2.8}$$

The β-cell response for the synthesis of insulin, is modelled with use of a proportional integral derivative controller, as proposed by Steil et al. [63]. This controller describes the production of insulin in multiple phases, resulting in different types of insulin release, clustered in one equation. Therefore, the release of the endogenous produced insulin in the bloodstream is described by the following equation:

$$i^{\text{pnc}} = \beta^{-1} \left(k_{12} \left(G^{\text{pl}} - G_b^{\text{pl}} \right) + \frac{k_{13}}{\tau_{\text{i}}} \int \left(G^{\text{pl}} - G_b^{\text{pl}} \right) \mathrm{d}t + (k_{14} \tau_{\text{d}}) \frac{\mathrm{d}G^{\text{pl}}}{\mathrm{d}t} \right) \tag{2.9}$$

In Equation 2.9, τ_{i} is the integral and τ_{d} is the derivative time constant. β Represents the glucose sensitivity of the β-cells, whereas, k_{12}, k_{13} and k_{14} are the reaction speed velocity parameters. The removal of insulin from the blood plasma (i^{rem}) depends on the uptake of insulin by the liver, combined with the diffusion of insulin towards the interstitial fluid. If the amount of insulin in the blood plasma exceeds the basal insulin level (I_b^{pl}), the removal of insulin increases:

$$i^{\text{rem}} = k_5 \left(I^{\text{pl}} - I_b^{\text{pl}} \right) \tag{2.10}$$

Insulin dynamics in the remote compartment. The remote compartment, which is often referred to as the interstitium or interstitial fluid, can be regarded as a buffer zone for insulin. Whereas glucose can be experimentally measured in the interstitial fluid, for insulin this is much harder and less reliable [64]. The concentration of insulin in the remote compartment increases with the influx of insulin from the blood plasma (i^{pl}). When insulin interacts with the receptors of the insulin dependent tissues (i^{it}), the concentration of insulin in the interstitial fluid will reduce.

$$\frac{\mathrm{d}I^{\mathrm{rem}}}{\mathrm{d}t} = i^{\mathrm{pl}} - i^{\mathrm{it}} \tag{2.11}$$

The equation describing the inflow of insulin into the interstitium (Equation 2.12), works in similar fashion as Equation 2.10. However, the velocity parameter k_6 differs due to the possibility of insulin removal by the liver.

$$i^{\mathrm{pl}} = k_6 \left(I^{\mathrm{pl}} - I_b^{\mathrm{pl}} \right) \tag{2.12}$$

The insulin in the interstitial fluid could be taken up by the insulin dependent tissues (i^{it}). Whereas, the removal increases linearly if the concentration of of insulin in the interstitial fluid (I^{rem}) increases, as can be seen in Equation 2.13.

$$i^{\mathrm{it}} = k_7 I^{\mathrm{rem}} \tag{2.13}$$

2.2 Mathematics of the glucose metabolic system in patients with diabetes

In the previous section, the mathematical equations describing the metabolic system of a healthy person are elucidated. Certainly there is significant overlap between a healthy persons metabolic system and the system of a diabetes patient. However, some adjustments and the additions for medication are necessary. These differences are explained in this section.

As explained in Section 1.2.1, the most important difference between a healthy subject and a DM1 patient, is the incapability to produce endogenous insulin (i^{pnc}). To model this, the parameters involved in the insulin production (k_{12}, k_{13} and k_{14} in Equation 2.9), along with the basal insulin level (I_b^{pl}) are set to zero. For the modeling of DM2 patients, reduced β-cell sensitivity to glucose is implemented (depicted as β in the equations). Whereas the basal insulin levels are comparable to the basal levels of healthy individuals, the parameters k_{12} and k_{14} (in Equation 2.9) are reduced to describe the lower insulin production.

Renal secretion of glucose. The part of the model concerning the absorption of glucose from the gut, is identical in diabetes patients compared to healthy patients. For the plasma glucose levels the equation differs due to the possibility of renal secretion of glucose (g_{th}^{ren}). Renal secretion only occurs in prolonged uncontrolled hyperglycemic situations. Only if the plasma glucose levels exceed the threshold, the g_{th}^{ren} term is activated as can be seen in Equation 2.14. Equation 2.15, is the complete equation concerning the glucose plasma concentration in patients with diabetes.

$$g_{th}^{\mathrm{ren}} = \begin{cases} k_8 \left(G^{\mathrm{pl}} - G_{th}^{\mathrm{pl}} \right) & \text{if } G^{\mathrm{pl}} > G_{th}^{\mathrm{pl}} \\ 0 & \text{if } G^{\mathrm{pl}} \leq G_{th}^{\mathrm{pl}} \end{cases} \tag{2.14}$$

$$\frac{\mathrm{d}G^{\mathrm{pl}}}{\mathrm{d}t} = g^{\mathrm{liv}} + g^{\mathrm{gut}} - g^{\mathrm{non\cdot it}} - g^{\mathrm{it}} - g_{th}^{\mathrm{ren}} \tag{2.15}$$

However, in patients with DM1 and in some cases DM2, there is the necessity to exogenous administer insulin in order to increase the insulin plasma levels. As explained in Section 1.2.5, multiple forms of medication or insulin administration are possible. However this model is limited to only short-acting (i^{sa}) and long-acting insulin (i^{la}). These medicinal terms are added to Equation 2.16:

$$\frac{\mathrm{d}I^{\mathrm{pl}}}{\mathrm{d}t} = i^{\mathrm{pnc}} - i^{\mathrm{rem}} + i^{\mathrm{sa}} + i^{\mathrm{la}} \tag{2.16}$$

Short-acting insulin. Modeling short-acting insulin dynamics is based on the work of Shimoda et al. [65], who proposed the use of two subcutaneous compartments to delay the inflow of insulin into the blood plasma. After injection, the insulin is present in the subcutaneous compartment near the injection site (U_I^{sc1}), as can be seen in Equation 2.17. U^{sa} Represents the injected units of short acting insulin.

$$\frac{\mathrm{d}U_I^{\mathrm{sc1}}}{\mathrm{d}t} = U^{\mathrm{sa}} - k_{16}U_I^{\mathrm{sc1}} \tag{2.17}$$

From this injection site, the insulin will diffuse towards the subcutaneous tissue proximal to the blood plasma (U_I^{sc2}) resulting in the desired delay (Equation 2.18). From the compartment proximal to the plasma the influx of short-acting insulin into the blood plasma (i^{sa}) occurs.

$$\frac{\mathrm{d}U_I^{\mathrm{sc2}}}{\mathrm{d}t} = k_{16}U_I^{\mathrm{sc1}} - k_{17}U_I^{\mathrm{sc2}} \tag{2.18}$$

$$i^{\mathrm{sa}} = \frac{k_{15}}{v_I M^{\mathrm{b}}} U_I^{\mathrm{sc2}} \tag{2.19}$$

In Equation 2.19, v_I is the insulin distribution volume in the plasma, M^{b} covers the body mass of the patient. This influx of short-acting insulin is added to the equation for insulin in blood plasma (Equation 2.16).

Long-acting insulin The administration of long-acting insulin is modelled accordingly to the study of Berger et al. [66]. Due to the slow release of long-acting insulin, the medication is modelled with use of the half-time absorption ($t_{0.5}$) of the injected insulin type. The half-time absorption is calculated in Equation 2.20, in which U^{la} represents the total amount of injected units long-acting insulin. Furthermore, a and b are the medication specific dose shape factors regarding the absorption of insulin.

$$t_{0.5} = aU^{\mathrm{la}} + b \tag{2.20}$$

The increase of insulin in the blood plasma by the administration of long-acting insulin (i^{la}) is described by the equation below (Equation 2.21). In which h is a shape factor characterized by time. The insulin distribution volume is depicted as v_I.

$$i^{\mathrm{la}} = \frac{h t^{h-1} t_{0.5}^{h}}{v_I M^{\mathrm{b}} \left(t_{0.5} + t^{h}\right)^2} U^{\mathrm{la}} \tag{2.21}$$

Subsequently, the influx of long-acting insulin is added to the equation covering the insulin concentration in the blood plasma (Equation 2.16).

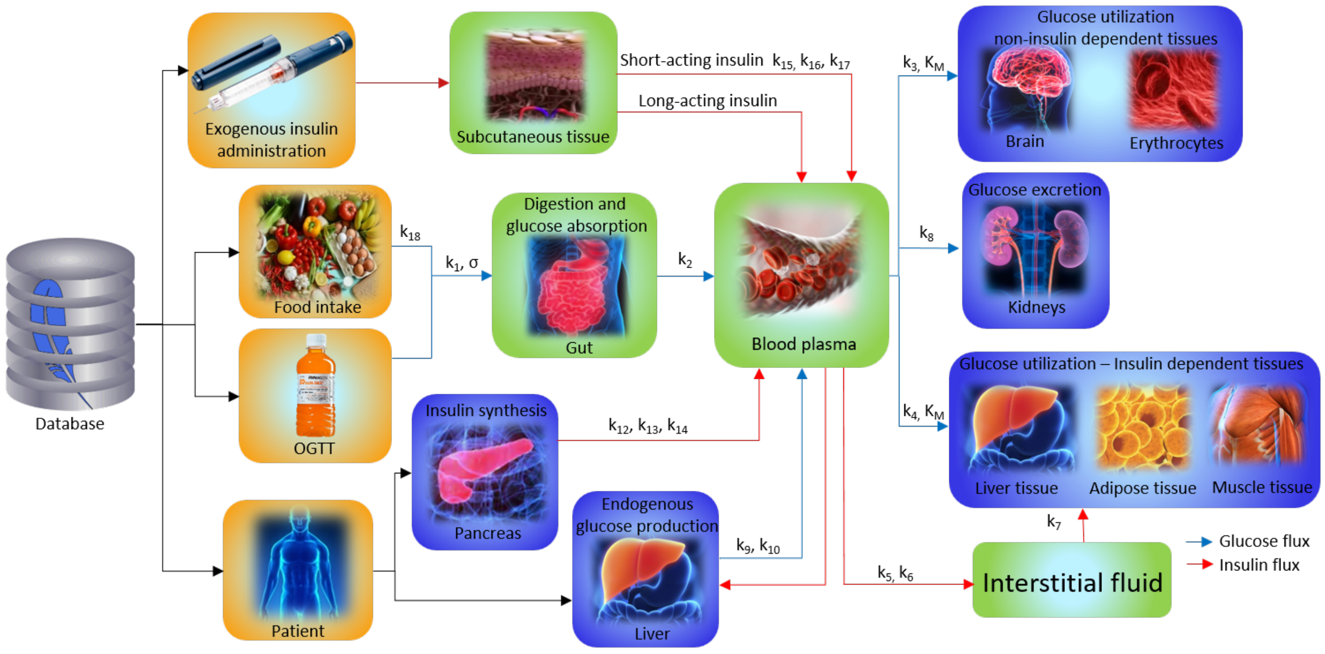


Figure 2.1: This flowchart is showing the compartments and fluxes of the model. The blue arrows represent the glucose fluxes, whereas the red arrows represent the insulin fluxes. Furthermore, the parameters responsible for these fluxes are displayed on the arrows. In the orange boxes are the inputs from the DB displayed. In the blue boxes organs are displayed presenting the in- and out- fluxes of the system. The state compartments of the system are shown in green boxes.

2.3 Differences between model versions

Overall the eDES 1.1 and eDES 2.0 models are very similar. Therefore, only significant differences between the models are discussed in this section, again the complete sets of differential equations can be found in Appendix B. Furthermore, the corresponding parameters that differ for each model can be found in Appendix Table B.1. Notice that the numbers and therefore the function of the parameters k_i are not identical in the different model versions. Minor variations between equations and their notation are not discussed to focus on the important differences. The compartments, fluxes and their corresponding parameters can be found in the flowchart in Figure 2.2.

Removal of the insulin interstitial fluid compartment. In eDES 2.0, the remote compartment regarding the insulin in the interstitial fluid is removed. According to Maas et al. [67], this is since real insulin levels are unknown and cannot be measured. Furthermore, by decreasing the number of estimated parameters, it could increase the accuracy of the remaining parameters. Adjusting the model is done by replacing the I^{rem} term in different ways. In the equation describing the glucose produced by the liver (Equation 2.22), the I^{rem} is replaced with $I^{\text{pl}} - I_b^{\text{pl}}$ leading to the following equation:

$$g^{\text{liv}} = g_b^{\text{liv}} - k_3 \left(G^{\text{pl}} - G_b^{\text{pl}} \right) - k_4 \beta (I^{\text{pl}} - I_b^{\text{pl}}) \tag{2.22}$$

Additionally, the insulin used by the insulin dependent tissues, can no longer be obtained from the remote compartment. Therefore, the equation regarding the glucose uptake by insulin dependent tissues (g^{it}), is modified to secrete the insulin in the plasma I^{pl}:

$$g^{it} = k_5 \beta I^{\text{pl}} \frac{G^{\text{pl}}}{(K_{\text{M}} + G^{\text{pl}})} \tag{2.23}$$

Insulin in the blood plasma is removed from the system by the uptake from the interstitial fluid (i^{it}) and leaves the system. Therefore, in model 2.0 the interstitial fluid is a sink for insulin (see Figure 2.2) and no longer a compartment, as could be seen in model 1.1. Furthermore, in the 2.0 model a term for insulin degradation by the liver (i^{liv}) is introduced. Providing the following equation for insulin in the blood plasma:

$$\frac{\mathrm{d}I^{\mathrm{pl}}}{\mathrm{d}t} = i^{\mathrm{pnc}} - i^{\mathrm{if}} - i^{\mathrm{liv}} + i^{\mathrm{sa}} + i^{\mathrm{la}} \tag{2.24}$$

The by the liver degraded insulin is calculated with Equation 2.25. τ_{i} Represents the integral time constant, the parameter k is adjusted to a constant (c_3) which is evaluated later in this chapter.

$$i^{\mathrm{liv}}(t) = k_7 \frac{G_b^{\mathrm{pl}}}{\beta \tau_{\mathrm{i}} I_b^{\mathrm{pl}}} I^{\mathrm{pl}}(t) \tag{2.25}$$

Addition of c-peptide. Arguably one of the most significant differences between the models is the extension of c-peptide to the model. As discussed in Section 1.1.2, the advantages of adopting c-peptide are: a more stable test window combined with a better understanding of the endogenously produced insulin and the parameters involved. Furthermore, once the c-peptide is produced it does not interact with glucose or insulin and is only degraded by the liver. A remote compartment for the c-peptide (Y) exists. The c-peptide dynamics are significantly less complicated compared to the insulin dynamics of the model. The c-peptide compartment is based on the work of van Cauter et al. [68] and the equations are displayed below:

$$\frac{\mathrm{d}C^{\mathrm{pl}}}{\mathrm{d}t} = c^{\mathrm{pnc}} - c^{\mathrm{liv}} + c^{Y} \tag{2.26}$$

$$c^{\mathrm{pnc}} = \frac{v_I \alpha i^{\mathrm{pnc}}}{v_C} \tag{2.27}$$

$$c^{\mathrm{liv}} = \left(k_{12} + k_{14}\right) C^{\mathrm{pl}} \tag{2.28}$$

$$c^{Y} = k_{13} Y \tag{2.29}$$

$$\frac{\mathrm{d}Y}{\mathrm{d}t} = k_{12} C^{\mathrm{pl}} - k_{13} Y \tag{2.30}$$

In Equation 2.26, the total amount of c-peptide in the blood plasma is calculated. The by the pancreas secreted c-peptide (c^{pnc}) is calculated in Equation 2.27, where v_I and v_C represent the distribution of insulin and c-peptide in the plasma. α Is the unit conversion factor for insulin to c-peptide. Furthermore, Equation 2.27 connects pancreatic c-peptide secretion, with the secretion of insulin. The remaining equations of: c-peptide produced by the liver (c^{liv} Equation 2.28), c-peptide flow to the second compartment (c^Y Equation 2.29), and the c-peptide concentration in the remote compartment (Y Equation 2.30); depending on similar, previously discussed, straightforward kinetics and need no further explanation. All c-peptide fluxes are displayed with green arrows in the flowchart in Figure 2.2.

Hypoglycemic response. As explained in Section 1.1.1, as hypoglycemia can occur in patients with DM1. The body will quickly react to this hypoglycemia with extra secretion of glucose as response to glucagon. According to Maas et al. [67], this process can be mimicked without actually including glucagon. This is done with use of an if statement that recognizes if the plasma glucose level exceeds the basal level, or not ($G^{\mathrm{pl}} > G_b^{\mathrm{pl}} \vee G^{\mathrm{pl}} \leq G_b^{\mathrm{pl}}$). Additional, one of the two different velocity parameters (k_3 $\vee$ k_{10}) is accordingly multiplied with the ($G^{\mathrm{pl}} - G_b^{\mathrm{pl}}$) term in the equation regarding the basal hepatic glucose production. Resulting in an imitation of a reaction to hypoglycemia. The statement along with the corresponding equations are displayed below:

$$g^{\mathrm{liv}} = \begin{cases} g_b^{\mathrm{liv}} - k_3 \left(G^{\mathrm{pl}} - G_b^{\mathrm{pl}}\right) - k_4 \beta \left(I^{\mathrm{pl}} - I_b^{\mathrm{pl}}\right), & \text{if } G^{\mathrm{pl}} > G_b^{\mathrm{pl}} \\ g_b^{\mathrm{liv}} - k_{10} \left(G^{\mathrm{pl}} - G_b^{\mathrm{pl}}\right) - k_4 \beta \left(I^{\mathrm{pl}} - I_b^{pl}\right), & \text{if } G^{\mathrm{pl}} \leq G_b^{\mathrm{pl}} \end{cases} \tag{2.31}$$

Basal levels. In the 2.0 model, are besides the already existing basal levels, steady state levels introduced. In healthy subjects are their basal and steady state levels identical; however, this is not the case for patients with diabetes. The body strives to agree with the basal levels, whereas, the steady state levels are reached without the exogenous impact of medication. Especially in patients with DM1, which have insulin steady state levels approaching zero, mathematics could conflict with the physiological reality of the system. Since dividing by a number close to zero, could provide unrealistic high values for the uptake of glucose by non insulin dependent tissues. Regardless if the subject is healthy or a patient, the desired glucose uptake by non insulin dependent tissues should not differ extremely. Therefore, the constant c_2, Equation 2.32, is introduced to reproduce the insulin independent glucose uptake for healthy subject.

$$c_2 = g_b^{\text{liv}} \left(\frac{K_{M,\text{healthy}} + G_{b,\text{healthy}}^{\text{pl}}}{G_{b,\text{healthy}}^{\text{pl}}} \right) \tag{2.32}$$

Resulting in a physiologically more acceptable equation, describing the total uptake of glucose by non-insulin dependent tissues:

$$g^{\text{non-it}} = c_2 \frac{G^{\text{pl}}}{K_M + G^{\text{pl}}} \tag{2.33}$$

Equivalently, the equation describing insulin clearance by the liver from the plasma (Equation 2.25), is adjusted. I_b^{pl} can become zero for a DM1 patient. To circumvent division by zero the multiplication factor c_3, adjusted to a healthy subject, is introduced:

$$c_3 = k_{7,\text{ healthy}} \frac{G_{b,\text{ healthy}}^{\text{pl}}}{\beta \tau_{\text{i}} I_{b,\text{ health}}^{\text{pl}}} \tag{2.34}$$

The rate constant of insulin clearance (c_3) from Equation 2.34, is multiplied by the insulin concentration in the plasma to calculate the insulin clearance by the liver ($i^{\text{liv}} = c_3 * I^{\text{pl}}$).

Degradation of long acting insulin. In the older version of the model, degradation of the long-acting insulin at the injection site is not taken into account. Therefore, all injected long-acting insulin is used in this situation. To compensate for this effect an additional parameter (k_e), that differs per insulin brand, is added to the model. Resulting in the final equation for long-acting insulin:

$$i^{\text{la}} = (1 - k_e) \frac{h \left(t_{0.5}\right)^h t^{h-1}}{\left(\left(t_{0.5}\right)^h + t^h\right)^2} \frac{1}{v_I M^{\text{b}}} U^{\text{la}} \tag{2.35}$$

With U^{la}, the units of administered long-acting insulin. h Is the absorption time characteristic and $t_{0.5}$ the half-life time, both differ per insulin brand.

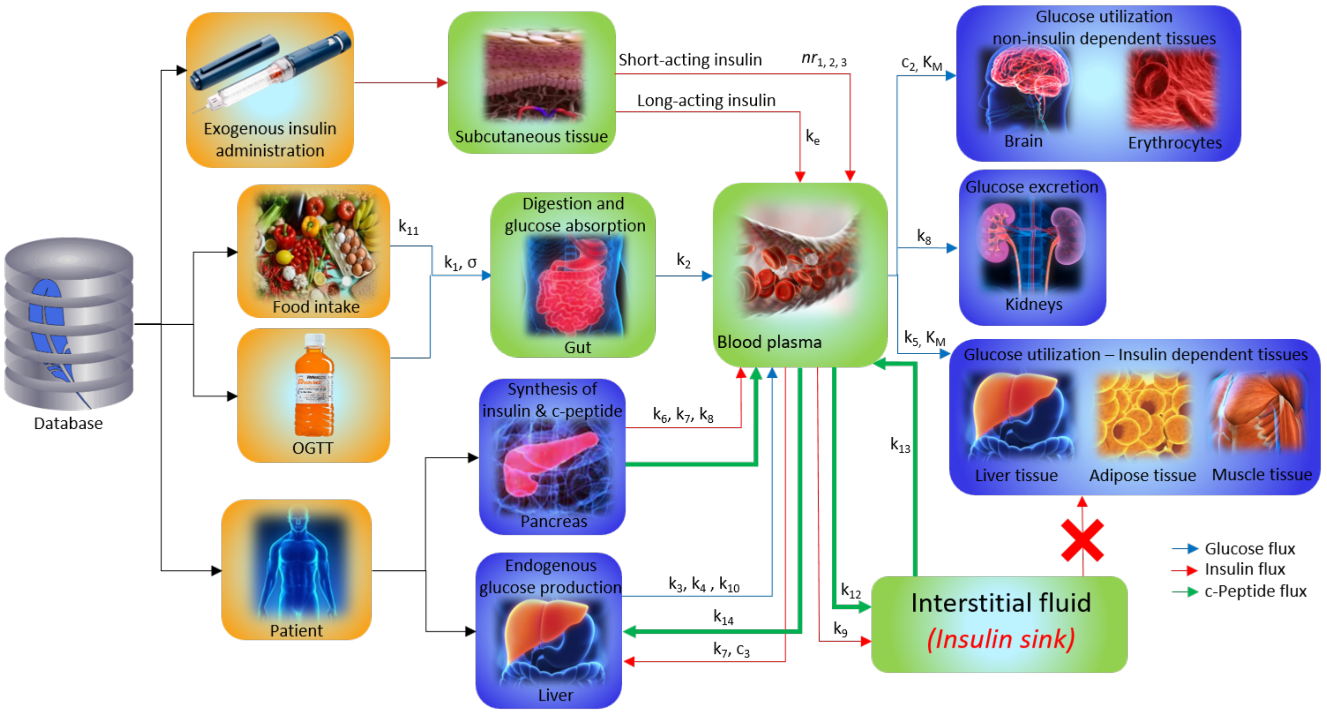


Figure 2.2: In this figure, the compartments and fluxes are displayed. The blue arrows represent the glucose flux, the red arrows represent the insulin flux, whereas the green arrow represent the newly added c-peptide. Additionally, the parameters responsible for these fluxes are displayed next to its corresponding arrow. In the orange boxes are the inputs from the DB displayed. The blue boxes displayed the organs responsible for the in- and out- fluxes of the system. In green boxes are the state compartments of the system displayed. The c-peptide arrows are displayed bigger to stress the differences of the 2.0 model. The in model 1.1 existing insulin flux towards the insulin dependent tissues is marked with a cross. Whereas, this flux is no-longer present in eDESpy 2.0.

Chapter 3

Construction of the model

3.0.1 Model structure

Although the mathematical equations are alike, the composition of the eDESpy model differs from the previously created eDES versions. The eDES MATLAB versions are built in a more hard-coded way, to adjust parameters and constants the source code needs to be adjusted and the executable has to be recompiled. Statements and the equations that are activated by those statements, are placed in the same script. This reduces the flexibility of the model in expanding, or (partially) (de)activating compartments. In this section the construction of the eDESpy model is explained. The new eDESpy model is made as modular as possible, without losing functionality. This is done to make the existing models more operational and accessible. In Figure 3.1 the different sections of the model are shown.

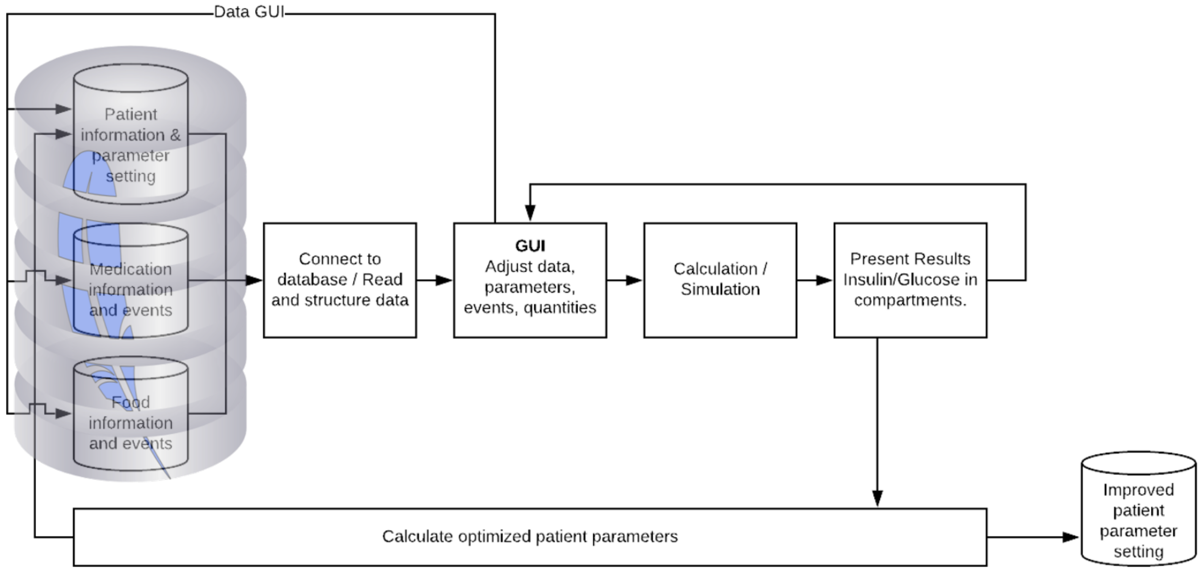

Data GUI
Patient information & parameter setting
Medication information and events
Food information and events
Connect to database / Read and structure data
GUI
Adjust data, parameters, events, quantities
Calculation / Simulation
Present Results Insulin/Glucose in compartments.
Calculate optimized patient parameters
Improved patient parameter setting

Figure 3.1: Multiple compartments of the *eDESpy* model. At the left hand side, through the transparent SQLite logo, is the information retrieved from the DB shown. The program will connect with the DB and structure the data. Furthermore, a GUI is used in which the data, parameters, and events are shown and can be adjusted. The simulation is computed and the results are displayed in the GUI, afterwards the output and parameters can be saved in the DB.

Database. Starting with the left side of Figure 3.1, there is this cylindrical shape with a blue feather on it. This is the logo of SQLiteStudio, a DB management system using SQLite, the engine of the DB is embedded into the program. The parameter settings (parameters), medication plus nutriment (events), and patient information (constants), are stored in tables. The patients and its corresponding parameters and events are linked through a patient id. Furthermore, in the DB it is optional to switch between patients, parameters and events, those will or will be not be loaded into the program.

Connect to database. Next, there is this box responsible for reading and structuring the data. In this connection the DB is read and the data is loaded from the proper tables. In addition, this part of the model is responsible for assigning the retrieved data to the corresponding object, e.g. a patient. For additional information on the DB and the connection with Python see Section 3.2.2.

Graphical User Interface. Briefly, the GUI is the control center of the model. This window is a tool in which capabilities of eDESpy can be operated. Examples are: the opportunity to choose which eDES version is used; appoint a patient that is evaluated; adjust and estimate the parameters; and decide which compartments of the model are used. Furthermore, the visualization of the graphs and results takes place in the GUI. This will be explained further in Chapter 3.3. In addition, there is the possibility to safe calculated or adjusted parameters/events onto the DB.

Calculation. The next part is responsible for the computations of the model. This part loops through the specified events (medication, activity and meals) and calculates the states of the model accordingly with use of the corresponding equations. Naturally, when different models or compartments are used this will affect this part of the program.

Optimization. The model can fit parameters to a patient, creating a patient specific model. The optimization is discussed in Chapter 4. These parameters are stored into the DB, creating a VPP.

3.1 Object-oriented programming

While developing the program, the code is constructed as much object-oriented as possible. Python is a language that supports object-oriented programming (OOP). In this section the concept of OOP, and its importance is explained. The four pillars of OOP are encapsulation, abstraction, inheritance and polymorphism. However, the for this research important concepts are abstraction and inheritance. Abstraction reduces complexity, along with easier adjustment of the code, without impacting other parts of the program. Inheritance means that objects can inherent properties and methods; therefore, similar methods can be used for multiple objects which reduces redundant code.

Starting with the class, this is a template of an object defined with variables/attributes and functions/-methods that characterize the object. In the example code in Listing 3.1, the general object is an Event. The class *Event* is an upper level or general class. Type is an example of an attribute referring to the object Event. The instance variables are defined in the *__init__* definition. Whenever a class is called upon, the initial definition is immediately executed. The instance variable defined inside a method belongs only to its current class instance. Class variables, which are used less often, are shared by all instances of a class and defined within the class but outside its methods. In Listing 3.1, *units = 0* is an example of a class variable [69]. In the Event class are three different methods constructed: *timeString*, which provides the time in a digital notation; *timeMin*, returning the time as the total amount of minutes; and *toString* which returns a string providing information about the time, description and units. The lower level class *Med(Event)* inherits all the attributes, along with the defined methods from the upper level class *Events*. The lower level classes contain the specific differences in attributes distinguishing them from other classes. E.g. for medication the type of administration is relevant; however, this information is irrelevant for nutrition. With abstraction, the focus is on the important information retrieved from the DB. To extract an attribute from an object, for example the starting time from the object *Event*, the following construction is used: *Event.start_time*.

```
class Event():
    units = 0
    def __init__(self, start_time, end_time, e_id, Type, description):
        self.start_time = start_time
        self.end_time = end_time
        self.e_id = e_id            #event id
        self.Type = Type            #Type of event: medication, nutrition, activity
        self.description = description

    def timeString(self):
        return "{0:02d}:{1:02d}".format(int(self.start_time / 60), self.start_time % 60)
    def timeMin(self):
        return self.start_time
    def toString(self):
        return self.timeString() + " " + self.description + ": " + str(self.units)

class Med(Event):
    def __init__(self, e_id, Type, start_time, end_time, units, administration,
    description):
        Event.__init__(self, start_time, e_id, Type, end_time, description)
        self.units = units
        self.administration = administration
```

Listing 3.1: An example of object-oriented code from eDESpy, the lower level class *Med*, inherits attributes and methods from the general class *Event*.

3.1.1 Computation of the model

In this section the process on how the eDESpy model executes the computation is explained. The programming language used for creating the model is Python 3.7. Python is chosen since it is an object based programming language and has a wide range of freely available applications and extensions [70]. Alongside, the Integrated Development Environment (IDE) Spyder is chosen. Predominantly since Spyder is a scientific development environment. Spyder can be found in the Anaconda Navigator interface [71]. Certain additional libraries are used to improve the functionality of the Python script. From the standard library of Python *Math* is used in mathematical functions, an example is the natural exponent in the ODEs. *NumPy* is a library containing multi-dimensional array and matrix data structures [72], used for creating and operating lists and arrays. *SciPy* is a library used for technical and scientific computing, in this research *SciPy* is mostly used for solving differential equations [73]. Furthermore, *Matplotlib* an extension of *NumPy* is used to produce the plots [74]. *PyQt5* is a package assisting with the creation of a GUI, see Section 3.3.

The concept of the model is a continuous simulation, in which the system state is changed continuously over time. The model is based on a set of differential equations, defining the rates of change of state variables. These state variables represent the amount of glucose and insulin in the state compartments, which changes over time. However, the simulation is able to handle multiple events. Types of events are: consummation of nutrition, administration of medication and participate in PE. These events can overlap, or can be initiated simultaneously. The lasting effects of all events on the continuous simulation are calculated in each time step. This will be explained further in the following paragraphs.

To run the program, without usage of the GUI, *Mainfile.py* is executed. Initially, the time span of the simulation is set to 24 hour, the proper version and DB are chosen. Next, the data is retrieved from the DB and stored in lists containing objects. The exact process is further discussed in Section 3.2.2. The patients parameters are stored under the object *"p"*, constants *"c"*. The nutrition, medication and PE are stored in a list named *"EL"*. The list of events is sorted by the starting time of the object, assuring the event list (EL) is in the chronological order. The initial values "x_0" are introduced; along with the efficiency factors and compartments which are further discussed in Section 3.1.2. When the program is computed in its original state, all compartments are turned on and the efficiency is set to a normal 100%. All preliminary work is done, the next step is calculating the output of the states x.

The functionality of multiple events computed in a single simulation is possible due to the *CalcEvents* function, in *CalculateEvents.py*. The methodology used is that the ordinary differential equations are computed over multiple small time periods, with divergent inputs. Moreover, initial state variables of the new event, are equal to final state variables of the previous event. In the *CalculateEvents.py* file all occurring events are separately computed, whereafter the results are merged in a coinciding timeline.

With *NumPy* an array is arranged from time point 0, until the starting time of the first actual event. During this pre-event time, a "dummy event" is executed to start the simulation; however, this dummy event has no influence on the simulation and is only used for practical purposes. When the starting point of the first real event is reached, the *NumPy* arrays containing the time between this event and the next event, (or end of the simulation depending on the amount of events,) is created. These *NumPy* arrays are chronologically inserted in the *odeint()* solver, as can be seen in Listing 3.2. The *odeint* function, from the scientific Python package (*SciPy*), is capable of solving a system of ordinary differential equations.

The outputs are concatenated in the ”xList”, merging the output of the handled events. By giving *odeint* the input function *”model.calculate_states”*, the correct class of the model in *CalculateStates.py*, regarding the model version is implemented.

```python
import DATA.Event_operations as Event_operations
def CalcEvents(model, x0, p, c, EL, stepsize, duration, fun_id, efficiency_factor):

    # Dummy event:
    E_dummy = [Meal(0, 'meal', 1, 1, 0, '', '')]

    # Time & computation before first event:
    t= np.arange(0, EL[0].start_time, stepsize)
    x = odeint(model.calculate_states, x0, t, args=(p, c, 0, E_dummy, fun_id,
    efficiency_factor))
    xList = x

    for i in range(Event_operations.LengthEventList(EL)):
        # Final event time:
        if i == Event_operations.LengthEventList(EL) - 1:
            t = np.arange(0, duration - EL[i].start_time, stepsize)
        # Starting & intermediate time events:
        else:
            t = np.arange(0, EL[i+1].start_time - EL[i].start_time, stepsize)

        x = odeint(model.calculate_states,xList[-1],t, args=(p, c, i, EL, fun_id,
    efficiency_factor))
        xList = np.concatenate((xList, x), axis = 0)
    return xList
```

Listing 3.2: The code from the *CalcEvents* function is displayed, which loops through the instances in the EL. The time span in between instances is stored in a numpy array. Before the first real event, a fictional dummy event is introduced. This is for practical purposes and does not influence the solution. The *model* input parameter is responsible for running the class of the correct version, eDESpy 1.1 or 2.0. The *odeint* solver from *SciPy* calculates the solutions between the events. These solutions are stored in *xList*.

In the calculated states function, all statements inducing the different actions of the models are defined. For example, when a meal is consumed this part of the model needs to impact the system and not before. However, when a new event starts, the old event cannot be neglected. From the *CalcEvents* function, the EL and the current loop index (i), are input for the classes (*eDESpy_11 or eDESpy_20*), in the *CalculateStates.py* file. Additionally, the indexes of the occurring events in the *EL* list, are stored in new lists (e.g. *iMeal* is a list with all the indexes on which a meal occurs). By creating these index lists as can be seen in Listing 3.3, the program implements the indexes on which a certain event occurs, and recognizes if there should be operated upon that event. If this is the case, the program continues to the events corresponding function in *equations_11.py* or *equations_20.py* (depending on the desired version). If necessary, this equation loops through a for loop containing all likewise previous events, see Listing 3.4. The effect of all likewise types of events, at a the current point in time, is summated and returned.

```python
# CalculateStates.py:
class eDESpy_11():
    def calculate_states(x,t,p,c,i, EL, fun_id, efficiency_factor):
    iMeal = Event_operations.Index_Events(EL)[0]     #Nutrion
    iINSsa = Event_operations.Index_Events(EL)[1]    #Short-acting insulin
    iINSla = Event_operations.Index_Events(EL)[2]    #Long-acting insulin
    iAct = Event_operations.Index_Events(EL)[4]      #Activities
    # When there are no events "meal":
    if not iMeal:
        mgmeal = 0

    # A meal occurs:
    elif min(iMeal) <= i and fun_id[0] == True:
        mgmeal = Equations.MgMeal(x,t,p,i, EL, iMeal, fun_id[10])
    else:
        mgmeal = 0
```

Listing 3.3: In this Listing the class *eDESpy_11* (from *CalculateStates.py*) is displayed. The *eDESpy_11* class assures that the equations are executed on the correct moment.

Glucose gut. This definition contains the in Section 2.1 explained equation, responsible for calculating the total amount of glucose (Equation 2.2). The reason why it is necessary to, for example, compute the effect of the meal using a loop is explained using Figure 3.2. Two meals are consumed both containing 40.000 mg carbohydrates. The first meal is takes place at 8:00, whereas, the second meal starts at 9:00 (point 1 in Figure 3.2). If the lasting effect of the meal at 8:00 would not be taken into account, the glucose mass in the gut would follow the blue line in Figure 3.2 and not exceed line 2. In this situation the system only recognizes the starting glucose mass in the gut at point 1. Obviously, this is incorrect since there is still glucose in the gut that needs to be digested. Therefore this looping system is necessary to achieve the expected maximum (line 3 in Figure 3.2). This is done likewise for the additions of medication, which are explained later. After the calculation of the total glucose in the gut, the transfer towards the absorbed glucose into the plasma is made (Equation 2.3). In *CalculatedStates.py* these results are subtracted from each other, as is shown in Equation 2.1, resulting in the final glucose mass in the gut.

```python

# Equations.py
def MgMeal(x,t,p,i, EL, iMeal, OGTT):
    MgMeal = []
    len_iMeal = sum(indexMeal <= i for indexMeal in iMeal)
    for k in iMeal[:len_iMeal]:

        #Calculate total mgmeal according to previous meals
        if k <= i:
            D = EL[k].units
            tcurrent = EL[i].start_time
            tprevious = EL[k].start_time

            if OGTT == True:
                p.k18 = 0

            t2 = t + (tcurrent-tprevious)
            mgmeal = (p.k1*t2)**(p.sigma-1) * p.sigma * p.k1 * \
            math.exp(-(p.k1*t2)**p.sigma) * math.exp(-(p.k18*t2)) * D
            MgMeal = np.append(MgMeal, mgmeal)

    mgmeal = sum(MgMeal)
    return mgmeal

    def calc_mgpl(k2, Mg):
        mgpl = k2 * Mg
        return mgpl
```

Listing 3.4: In this Listing the definition *MgMeal* (from *Equations_11.py*) is shown. This code shows the example in which *MgMeal* is called upon. Firstly, the code in *MgMeal* checks whether it is the first meal, if this is not the case it calculates the lasting effect of all previous meals on the system and returns the sum of the total meals. *calc_mgpl* Is a definition responsible for calculating the outflux of glucose mass in the gut.

Glucose in the blood plasma. The computation of the total amount of glucose in the blood plasma is quite straightforward. All terms of the Equation 2.15 are determined. The equation is solved and the plasma glucose levels are returned. This definition will be more extensively discussed in Chapter 5. For the corresponding code please see file *Equations_11.py* or *Equations_20.py*.

Short-acting insulin. Two different types of administration are possible for short-acting insulin. A bolus or continuous administration. The definition in Listing 3.6 verifies the type of administration, and administers the indicated amount of insulin. The administration time for a bolus is short ($t < 1$), compared to the lasting continuous administration. The continuous administration is necessary for maintaining the basal insulin levels during insulin pump therapy. The two Equations 2.17 and 2.18 providing the insulin concentration in the two subcutaneous compartments are calculated. This is crucial

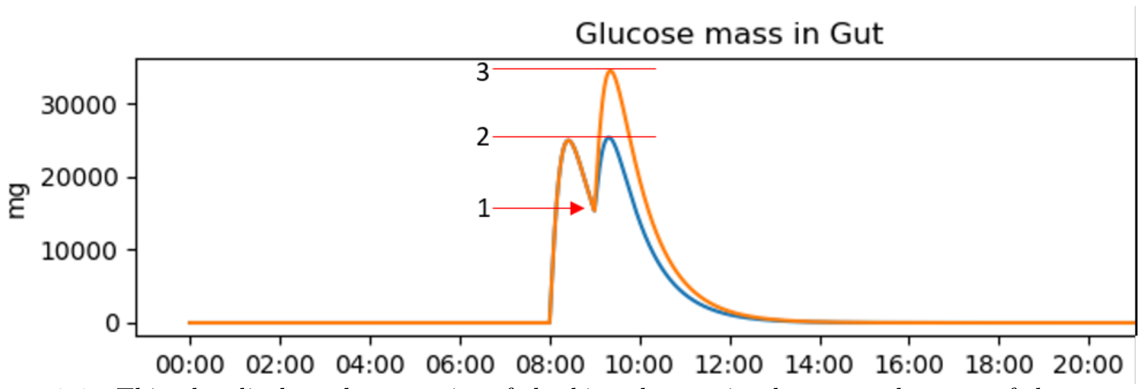


Figure 3.2: This plot displays the necessity of checking the previously occurred events of the system. The blue line visualizes the situation when no measures are taken. While the orange line, displays the current situation while taking into account the previous meals. Two meals of both 40.000 mg carbohydrates are consumed on 8:00 and 9:00. At point 1, the second meal is consumed, however the digestion of the first meal is still in progress. Therefore, without taking into account the previous meal (represented by the blue line), only the starting value of the glucose mass in the gut is correct. However the maximum glucose mass in the gut reached would not exceed line 2. Obviously, this is not correct. By using a for loop, as displayed in Listing 3.4, the expected summation of both meals is reached.

for the desired delay in insulin release. Where after the resulting short-acting insulin affecting the blood plasma insulin concentration, from Equation 2.19, is computed.

```

# Equations.py
def Short_acting_Insulin(x,t,p,c,i, EL):
    Uisc1 = x[4]
    Uisc2 = x[5]

    # Different administration, Bolus and Continuous:
    if EL[i].type =='isa_bolus':
        if EL[i].administration == 'bolus':
            if EL[i].units > 0 and t<1:
                usa = EL[i].units
            else:
                usa = 0
        else:
            usa = 0
    else:
        usa = 0

    if EL[i].type == 'isa_basal':
        if EL[i].administration == 'continuous':
            usa = EL[i].units

    duisc1dt = usa - (k16*Uisc1);
    duisc2dt = (k16*Uisc1) - (k17*Uisc2);
    isa = (p.k15/(c.vi*c.Mb)) * Uisc2;
    return duisc1dt, duisc2dt, isa, Ula
```

Listing 3.5: This is the code providing the administration of short-acting insulin. Two types of insulin administration are possible, an injection (bolus), or continuous insulin administration to maintain basal levels.

Long-acting insulin. Multiple brands of long acting insulin exist. These types all have a different specifications. Therefore different parameters are assigned to different insulin brands. The insulin brand name is stored under ”description” in the object long acting insulin. The *Medicine_Types.py* file contains two definitions. One definition regarding the 1.1 version medicine specifications (*medicine_specs_11*), and a definition for eDESpy version 2.0 (*medicine_specs_20*). Exact functions of the parameters can be found in Table B.1, in the Appendix. As previously explained is the for loop necessary to screen for similar previous long-acting insulin events, to calculate their lasting effect on the system. The equation regarding the half dose absorption time (Equation 2.20), and the total increase of long-acting insulin (Equation 2.21) in the plasma, are calculated.

```python

# Equations.py
def Long_acting_Insulin(x,t,c,i, EL, iINSla):
    iLa = []
    len_iINSla = sum(index <= i for index in iINSla)
    for k in iINSla[:len_iINSla]:
        Ula = EL[k].units

        # Different types of long acting insulin:
#         [h, a, b] = Medicine_Types.medicine_specs(EL[k].description)
        h, a, b = Medicine_Types.medicine_specs_11(EL[k].description)
        tcurrent = EL[i].start_time
        tprevious = EL[k].start_time
        t2 = t + (tcurrent-tprevious)

        # Time of half dose absorption [min]:
        t05 = (a*Ula) + b
        ila = (((h * (t2**(h-1)) * (t05**h) * Ula) / \
                ((t05**h) + (t2**h))**2)) * (1 / (c.vi*c.Mb))
        iLa = np.append(iLa, ila)

    ila = sum(iLa)
    return ila
```

Listing 3.6: In this Listing the definition *Long_acting_Insulin* is shown. Parameters "*h*", "*a*" and "*b*" differ per insulin brand. The corresponding parameters are retrieved from the *medicine_specs_11* definition. Lasting effects of the previous administrations are calculated, along with, the effect of the current event. The definition returns the total effect of the long-acting insulin on the plasma levels.

Insulin in blood plasma. The definition *Insulin_in_Plasma* in *Equations_11.py* calculates the plasma insulin concentration. Exogenous and endogenous insulin inclusion is taken into account, along with the uptake and removal of insulin. Remaining plasma glucose concentration, as calculated by Equation 2.16, is returned.

Insulin in interstitial fluid. The function computing the insulin concentration in the remote compartment is *Insulin_in_Interstitium* (in *Equations_11.py*). The insulin outflow from the plasma compartment towards the interstitial fluid, Equation 2.12, is computed. Additionally, the removal of the insulin from the interstitial fluid, Equation 2.13, is calculated. Subtracting these terms, as can be seen in Equation 2.11, results in the insulin concentration in the interstitial fluid.

Summarizing, *CalculateEvents.py* loops through the entire EL. *CalculateStates* holds all statements of the eDESpy model and is divided in an eDESpy_11 and eDESpy_20 class. In these classes the corresponding mathematical equations of the model are selected in *equations_11.py* and *equations_20.py*.

3.1.2 Model compartments

One of the demands of the eDESpy model is flexibility. To achieve this increased flexibility, the model is divided into multiple definitions (compartments), resembling different types of organ function, metabolic dynamics, medication administration, and the uptake of nutrition. In this section the compartments and the adjustable efficiency of compartments/fluxes are discussed. Switching certain compartments on or off could provide better insights in the effects of the processes of the model. Furthermore, by turning compartments (partly) off, diseases and other malfunctions could be imitated. For partial deactivation of compartments an efficiency factor is created. When the program is used under normal circumstances, the compartments are turned on, whereas the efficiency factors are set to 100%. The model contains predefined segment setting like: *"without medication"* or *"OGTT"*.

In Table 3.1 the compartments provided with a switch and in some cases efficiency factors are displayed. These efficiency factors and compartments can be adjusted in the *mainfile.py*, along with the *switch.py* file. However, this improvement excels more in combination with the use of the GUI. Whereas in the

Description:	Compartment (on):	Efficiency:
Nutrition	fun_id[0] = True	None
Glucose absorption from the gut to the blood plasma	fun_id[1] = True	efficiency_factor[0]
Endogenous glucose production by the liver	fun_id[2] = True	efficiency_factor[1]
Isnulin/c-peptide synthesis by the pancreas	fun_id[3] = True	efficiency_factor[2]
Medicinal short-acting insulin	fun_id[4] = True	None
Medicinal long-acting insulin	fun_id[5] = True	None
Interstitial fluid compartment	fun_id[6] = True	None
Glucose utilization by insulin dependent tissues	fun_id[7] = True	efficiency_factor[3]
Glucose utilization by non-insulin dependent tissues	fun_id[8] = True	efficiency_factor[4]
Glucose excretion by kidneys	fun_id[9] = True	efficiency_factor[5]
Oral Glucose Tolerance Test (OGTT)	fun_id[10] = True	None
c-Peptide flux to plasma	fun_id[11] = True	None
Hypoglycemic response	fun_id[12] = True	None
Physical exercise M_1	fun_id[15] = True	None
Physical exercise M_2	fun_id[14] = True	None

Table 3.1: In this table all compartment which can be turned on and off, are displayed. Furthermore, the efficiency factors are displayed, these are used to reduce or increase the ability of an organ or process. By doing this the effects of certain diseases could be mimicked.

GUI it is possible to recreate different circumstances and compare them with another.

The mechanism behind the on/off switch of the compartments is visible in Listing 3.3, and in this example located on line 13. The statement is only activated if a meal consumed combined with $fun\_id[0] == True$. Another example stressing the utility of this feature is shown in Listing 3.7. In Section 2.3 Equation 2.31, an additional equation describing the hypoglycemia response is added to the model version 2.0. When the plasma glucose level is lower than the basal glucose level, another statement containing a different reaction velocity parameter is activated. To visualize the effect of this adjustment, the statement can turned off by setting the function id to False ($fun\_id[12] == False$). The results are displayed in Figure 3.3, a single meal at 8:00 am is consumed. It can be seen that deactivating the statement provides more natural flow (the orange line), combined with a physiological expected glucose dip.

```python
def Glucose_in_Plasma_2(x, p, c, c1, c2, efficiency_factor, fun_id):
    Mg = x[0]
    Gpl = x[1]
    Ipl = x[2]

    if not fun_id[12]:
       gliv = c.gbliv - p.k3 * (Gpl-c.Gplb) - p.k4 * c.beta * (Ipl-c.Iplb) *
    efficiency_factor[1]
    else:
        if Gpl - c.Gplb > 0:
            gliv = c.gbliv - p.k3 * (Gpl-c.Gplb) - p.k4 * c.beta * (Ipl-c.Iplb) *
    efficiency_factor[1]
        else:
            gliv = c.gbliv - p.k10 * (Gpl-c.Gplb) - p.k4 * c.beta * (Ipl-c.Iplb) *
    efficiency_factor[1]

    if not fun_id[2]:
        gliv = 0

    ggut = p.k2 * (c.fc / (c.vg * c.Mb)) * Mg * efficiency_factor[0]
    if not fun_id[1]:
        ggut = 0
```

Listing 3.7: This code displays the mechanics behind the compartments and efficiency. The example used to illustrate this feature is the definition *Glucose_in_Plasma_2* (from *equations_20.py*). When the identity of the function (*fun_id*), is set to False, the compartment is turned off. These deactivated compartments are no longer able to influence the system. The efficiency factor (*efficiency_factor[0]*), is able to increase or reduce the efficiency of certain organs or procedures.

Next the mechanics behind the efficiency factor are explained. In the GUI or *switch.py* file, the option exists to decrease the functionality of organs and tissues. However, it is important to notice that using the efficiency is an addition to the already loaded model. Therefore, if a patient with DM2 has an initial insulin effectiveness of 80%, and the efficiency is set to 75%, the final insulin effectiveness will be 60%. The green line in Figure 3.3 shows a reduced 50% reduced glucose absorption from the gut to the blood plasma, along with a deactivated hypoglycemic response statement.

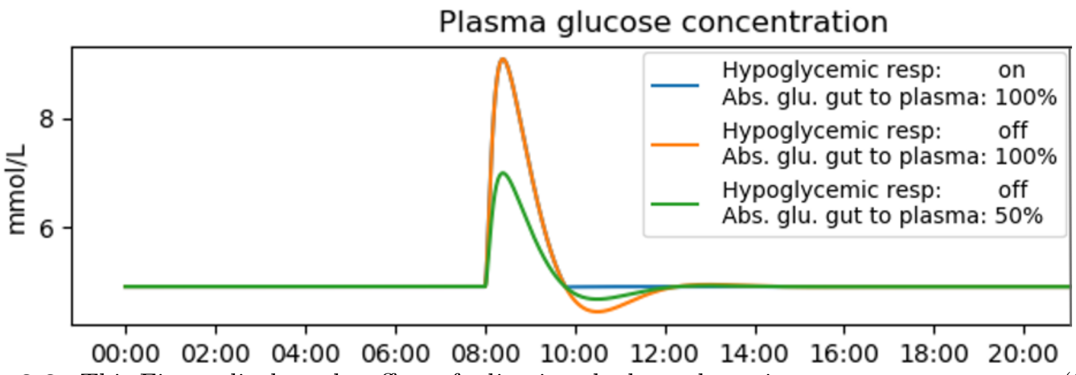


Figure 3.3: This Figure displays the effect of adjusting the hypoglycemic response compartment (from version 2.0). Whereas the blue line displays the normal plasma glucose concentration after a meal at 8:00. Next, the compartment of the hypoglycemic response is turned off, resulting in the orange line. Followed by a decrease of the absorption rate of 50%, resulting in the green line.

3.2 Database

The DB is a useful tool to increase the usability of the model. Patient information (constants), patient parameters (necessary to compute the simulation), nutrition and medication, should all be stored in a DB to improve accessibility of the data. A DB enables the possibility to retrieve data and information fast, compared to manually adjusting the program, as is done in previous eDES versions. In this section, the DB will be considered in more detail. Starting with discussing the software used, followed by the build-up and content of the DB. Thereafter, the basic basal SQL operations useful for controlling the eDESpy DB are explained. If additional information is required on how to build a SQL DB, please read the book by Grant Allen and Mike Owens, the Definitive Guide to SQLite [75].

3.2.1 SQLite

SQL, Structured Query Language, is a programming language used for managing structured data. The data is systematically organized, for easier data management. The DB management system used is SQLite, a C-language library. The code of the SQL DB engine is public and free to use for commercial and private purposes. SQLite does not require any server processes and the information is stored in a single disk file. The databse software is regarded very reliable and is one of the most used DB systems [75]. SQLite enables access to the DB and the possibility to view and adjust the data. The SQLite DB is a relational DB, storing the information in tables [76]. In this research the decision is made to use SQLiteStudio, due to personal preferences; however, other SQLite software packages can be used. SQLiteStudio can be downloaded free of charge from: *www.sqlitestudio.pl.*

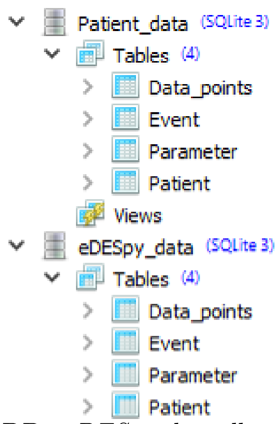

Patient_data (SQLite 3)
Tables (4)
Data_points
Event
Parameter
Patient
Views
eDESpy_data (SQLite 3)
Tables (4)
Data_points
Event
Parameter
Patient

Figure 3.4: The structure of the DB. *eDESpy_data.db* stores fictional and non fictional data and events from the GUI. *Patient_data* contains information about the actual events of the patients and their corresponding data points. As can bee seen, the structure of both DBs is identical.

3.2.2 The eDESpy database

In the eDESpy model there are two DBs. One DB is for usage of the regular model (*eDESpy.db*), the other DB is for the storage of patient data (*Patient_data.db*). This separation is necessary to assure the patient data is not accidentally polluted by operations from the model. The actual patient data is used for estimating the parameters. Both DBs can be accessed and adjusted by the model. Therefore, structure of both DBs is identical. Creating these DBs identically has a significant advantage. Particularly, actual events from the patient data or fictional events stored by the GUI, can be retrieved from the DB using the same code. Similarly, different versions of eDESpy all connect to the same DB. This concept is explained later.

The structure of the DBs can be seen in Figure 3.4, it exists of four tables: Event, Parameters, Patient and Data_points. These DB tables are described in Figure 3.5. First, the focus is on the initial three tables. The Event table stores all information regarding nutrition, medication and physical activities. The structure and all data columns of the event table are shown in Figure 3.5. The "Parameter" table stores all parameter sets; whereas, the "Patient" table stores constants and additional information about the patient. The "Data_points" table obviously contains actual measured data from the patients, which are used for the parameter estimation.

If it is desired to add or remove a parameter from the existing model, the structure of the table must be adjusted. In SubFigure 3.5a, the add button, which adds an extra column to the table, is marked with green. The remove button is colored red. However, to recognize this column change, the *read_db_tables* method in Listing 3.8, needs to be modified to posses the same number of rows compared to the DB table. This is necessary, since this table is direct input of a class creating an object, the input of the corresponding class should also be corrected. Thereafter, in the class the name of the newly created instance should be coupled to the object by using *self.name*.

In the DB, each DB row can be activated by setting the boolean "active" to true (active = 1), if lines need to be deactivated the boolean must be set to false (active = 0). The code (from *db_connect.py*) responsible for creating a connection and obtaining the desired rows with information, is displayed in Listing 3.8. The input of the function create_connection, is the name of the DB. When this connection is created all "activated" data from the DB is retrieved, using the *read_db_table.py* function. The string in line 12 is SQL code which is executed from Python, returning the activated (active = 1) rows.

```
1 def create_connection(database):
2     conn = None
3     try:
4         conn = sqlite3.connect(database)
5         curr = conn.cursor()
6     except Error as e:
7         print(e)
8     return conn, curr
```

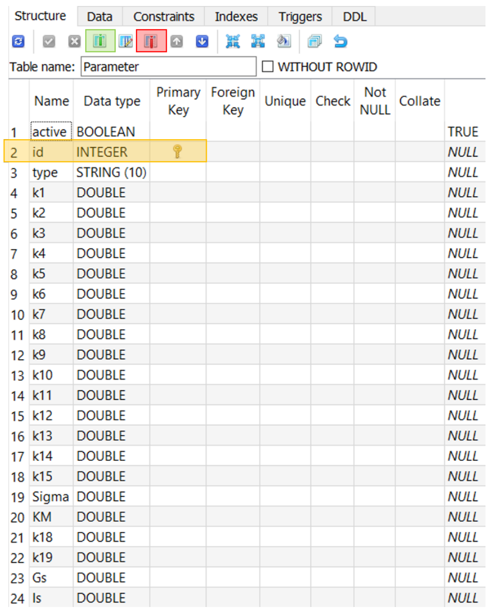


Table name: Parameter ☐ WITHOUT ROWID

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	
1	active	BOOLEAN							TRUE
2	id	INTEGER	🔑						*NULL*
3	type	STRING (10)							*NULL*
4	k1	DOUBLE							*NULL*
5	k2	DOUBLE							*NULL*
6	k3	DOUBLE							*NULL*
7	k4	DOUBLE							*NULL*
8	k5	DOUBLE							*NULL*
9	k6	DOUBLE							*NULL*
10	k7	DOUBLE							*NULL*
11	k8	DOUBLE							*NULL*
12	k9	DOUBLE							*NULL*
13	k10	DOUBLE							*NULL*
14	k11	DOUBLE							*NULL*
15	k12	DOUBLE							*NULL*
16	k13	DOUBLE							*NULL*
17	k14	DOUBLE							*NULL*
18	k15	DOUBLE							*NULL*
19	Sigma	DOUBLE							*NULL*
20	KM	DOUBLE							*NULL*
21	k18	DOUBLE							*NULL*
22	k19	DOUBLE							*NULL*
23	Gs	DOUBLE							*NULL*
24	Is	DOUBLE							*NULL*

(a) Structure of the table containing the Parameters.

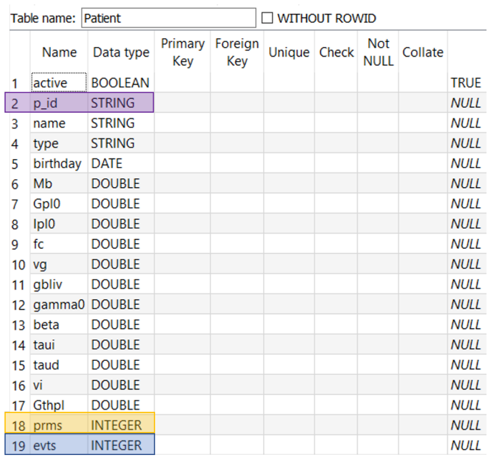


Table name: Patient ☐ WITHOUT ROWID

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	
1	active	BOOLEAN							TRUE
2	p_id	STRING							*NULL*
3	name	STRING							*NULL*
4	type	STRING							*NULL*
5	birthday	DATE							*NULL*
6	Mb	DOUBLE							*NULL*
7	Gpl0	DOUBLE							*NULL*
8	Ipl0	DOUBLE							*NULL*
9	fc	DOUBLE							*NULL*
10	vg	DOUBLE							*NULL*
11	gbliv	DOUBLE							*NULL*
12	gamma0	DOUBLE							*NULL*
13	beta	DOUBLE							*NULL*
14	taui	DOUBLE							*NULL*
15	taud	DOUBLE							*NULL*
16	vi	DOUBLE							*NULL*
17	Gthpl	DOUBLE							*NULL*
18	prms	INTEGER							*NULL*
19	evts	INTEGER							*NULL*

(b) Structure of the table containing information about the patient and the corresponding constants.

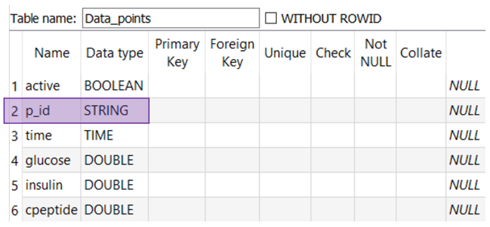


Table name: Data_points ☐ WITHOUT ROWID

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	
1	active	BOOLEAN							*NULL*
2	p_id	STRING							*NULL*
3	time	TIME							*NULL*
4	glucose	DOUBLE							*NULL*
5	insulin	DOUBLE							*NULL*
6	cpeptide	DOUBLE							*NULL*

(c) Structure of the table containing the measured data points, necessary for the parameter estimation.

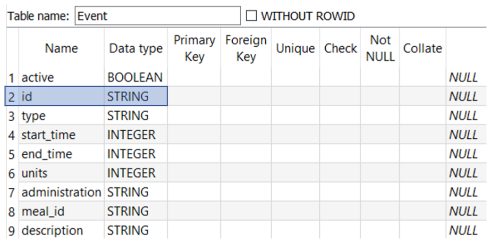


Table name: Event ☐ WITHOUT ROWID

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	
1	active	BOOLEAN							*NULL*
2	id	STRING							*NULL*
3	type	STRING							*NULL*
4	start_time	INTEGER							*NULL*
5	end_time	INTEGER							*NULL*
6	units	INTEGER							*NULL*
7	administration	STRING							*NULL*
8	meal_id	STRING							*NULL*
9	description	STRING							*NULL*

(d) Structure of the table containing the information of all occurring events: physical activities, medication and nutrition.

Figure 3.5: The build-up of different tables in the DB are displayed. The in Sub-figure 3.5a highlighted buttons can remove or add a column. The other transparent colored boxes shows the connection between different tables. When the integers in similarly colored boxes are identical, the model couples the parameters, data points or events with the corresponding patient.

```

def read_db_table(curr, t):

    curr.execute("SELECT * FROM {} WHERE active = TRUE".format(t))
    rows = curr.fetchall()
    return rows
```

Listing 3.8: This fragment of code, from *db_connect.py*, is able to make a connection from Python to the SQL DB. It fetches all activated rows from the DB which are now accessible in Python.

Next, the data must be linked to objects via the corresponding classes. This is done in *db_connect.py*, by the *read_db_tables* function. E.g. each activated row from the table "Parameter" is injected into the class Parameters, resulting in an object containing all parameters for one patient. The code responsible for this operation is shown (reduced in size), in Listing 3.9. This is done similarly for the "patient" and "value" table, containing the constants and measured data of the patient. The classes "Parameters" and "Patient" are created in the *Patients.py* file. For "Events" the approach is slightly different, since different events are possible. In *db_connect.py* first the nature of the event is determined, next the row is send to *Events.py*, to its corresponding class: Med, Meal or Exc. The returned objects are stored in the lists and are ready for the model to use.

```
# Method from the db_connect.py file:
def read_db_tables(database):
    pa = []
    c, t = create_connection(database)
    r = read_db_table(t, "Parameter")
    for row in r:
        pm.append(Parameters(row[1], row[2], row[3], ... row[23]))

# Class from the Patients.py file:
class Parameters:
    def __init__(self, p_id, Type, k1, ... Is):
        self.id = p_id
        self.Type = Type
        self.k1 = k1
        ...
        self.Is = Is
```

Listing 3.9: The first definition, *read_db_tables*, is from the *db_connect.py* file. The obtained active rows, shredded into individual data pieces. Subsequently, these pieces are injected into the class *Parameters*. Where the information is linked to the object "*Parameters*". All objects *Parameters* are stored in a list *pm*.

Additionally, links are created between the tables to ensure proper alignment. The patient table has a "*prms*" and "*evts*" column. When in agreement with the "id" integer from the "Parameter" or "Event" table, the Python program recognizes this and links the parameters and events to the correct patient. In Figure 3.5 are these linking integers marked with the same coloured transparent boxes. In the GUI or main file, the patients are chosen for analyzing. Through the linkage of the event_id and parameter_id with the appropriate patient, all elements necessary from the DB are coupled with the patient and available in Python.

Besides retrieving information from the DB, it is also necessary to import and export information from eDESpy into the DB. Examples are: newly created event lists or the adjusted/estimated parameters. In Listing 3.10, is the code responsible for saving the event list in the DB displayed. Saving the parameters is done is a similar fashion. Starting with creating a connection to the db and the correct table. Every event set has an id number, these numbers are ascending and the next unique number is found. Next, for every event in the event list, the SQL code is executed. This code links the events to the corresponding columns in the DB. After committing, the data is saved in the DB.

```
    c, t = create_connection(database)
    if tbl == "Event":
        max_ev_id = 0
        for ev in evs:
            if int(ev.e_id) > max_ev_id:
                max_ev_id = int(ev.e_id)
        max_ev_id = max_ev_id + 1
        for ev in evl:
            c.execute("INSERT INTO Event (active, id, type, start_time, end_time, units,
             administration, meal_id, description) VALUES('{}', '{}', '{}', '{}',
      '{}', '{}', '{}', '{}', '{}')".format( 1, ev.e_id, ev.type, ev.start_time, ev.
    end_time, ev.units, admtxt, mealtxt, ev.description))
    c.commit()
```

Listing 3.10: This is code from the method *append_db_event* in *db_connect.py*. The connection with the proper table from the DB is created. Following, the next event id is determined, this is an unique id belonging to the new event set. Following, the data is stored in the corresponding column using SQL code. This data is committed and saved in the DB.

When adding an extra parameter, event property, or constant, to the model minor adjustments must be made. As is shown in Subfigure 3.5a, a new column must be created in the table. In the (to the table) corresponding method, as can be seen in Listing 3.9, an extra row needs to be added. In the equivalent class, the new attribute is created. The new attribute is added to the object and can now be used in Python. However, to store extra attributes into the DB, the appending method is adapted. In the execute, which can be seen in Listing 3.10, the new column name of the table must be added. In the same line of code, the new attribute is inserted in *.format(...)*, as well as, a set of braces ('{}') in *VALUES*. Next, a few commands necessary to preform four basic operations to the DB are discussed. First, query command 1 in Listing 3.11, is explained. With *UPDATE* the changes are loaded into the table, in this case the table "Patient". *SET*, defines the change. *WHERE*, searches the instances in the table where the condition is met. In this example p_id ='*p001*', is correct. Query command 2, is an example deactivating every instance in which $p\_id \neq$'*p001*'. The third query command in Listing 3.11, selects all (*SELECT **) instances from the table event, where the type equals '*meal*'. The final command inserts all instances from the table *Patient_data.Data_points*, into *eDESpy_data.Data_points*. This operation is useful for copying or backing-up data. As previously mentioned, SQL is considered a quite straightforward programming language to learn [75]. Sufficient sources elucidating SQL are available [76]. In a future model, these basic operation could be implemented into the Python program, reducing the requirement of using SQL commands.

```

/* Query command 1: */
UPDATE Patient
SET active = 1
WHERE p_id='p001'

/* Query command 2: */
UPDATE Patient
SET active = 0
WHERE p_id<>'p001'

/* Query command 3: */
SELECT *
FROM event
WHERE type = 'meal'

/* Query command 4: */
INSERT INTO eDESpy_data.Data_points
SELECT * FROM Patient_data.Data_points
```

Listing 3.11: SQL code, displaying four basic commands useful for operating the DB. The first statement activates the patient in table Patient with p_id ='*p001*'. Query command 2, deactivates every patient with a different patient id than '*p001*'. The third operation selects all events from table *event* with the type '*meal*'. Query command 4, inserts an entire table into another table.

3.3 Graphical User Interface

A GUI, significantly increases the operationabillity of a program. Whereas, with the new GUI differences in the simulations can be executed without having to make adjustments in the code. Additionally, this could make the model more accessible for people with limited programming skills. In this section the creation, use and extension of the GUI are analyzed. First the software used to create the GUI, *PyQt5* and *Qt designer*, along with the creation of the GUI are examined in Section 3.3. Followed by a section showing the functionality of the GUI.

3.3.1 GUI software

Qt designer is an application programming interface, useful for composing the GUI. Whereas the software enables to form objects and widgets, which can be adjusted to the desired preferences [77]. *Qt designer* can be downloaded freely from: *https://build-system.fman.io/qt-designer-download*, please select the right file for your operating system. *PyQt5* is installed by running the following line in the Python console or command prompt: "*pip install PyQt5*". In *Qt designer* widgets and features can be added, selected, named and aligned. The design can be easily adjusted without the necessity to program this by hand. Furthermore, Qt designer contains a preview function, which enables to preview the created window, without having to run the code in Python. *Qt designer* works by converting the created GUI into Extensible Markup Language (XML) code [78]. This XML code is stored in a user interface (*.ui*) file, in this situation *eDESpy.ui*. XML is a markup language useful for storage of layout settings and formatting. *PyQt5* is a GUI and cross platform toolkit, it functions as a XML parser [79]. How both platforms (XML and Python) are connected is shown in listing 3.12.

```python
# code from ControlCenter.py
from PyQt5.uic import loadUi
class myWindow(QMainWindow, object):
    def __init__(self):
        super(self.__class__, self).__init__()
        loadUi('./GUI/eDESpy_v2.ui', self)
        ...
        self.pushBtn_RunSim.clicked.connect(self.on_pushBtn_RunSim_Clicked)

    def on_pushBtn_RunSim_Clicked(self):
        ...
```

Listing 3.12: This code connects the with *Qt designer* created XML GUI layout, while using *PyQt5*. Furthermore, the correct approach for connecting an attribute (in this case *self.pushBtn_RunSim*) with a function in the Python code, is shown.

After connecting the XML code to the Python code, it is necessary to connect the in *Qt design* created widgets with the handles in Python. Due to the fact that the XML code is connected to the super class, the code recognizes the created attribute with the "self" keyword followed by the name of the attribute. Different widgets or attributes have different methods which can be applied to them. E.g. a push button contains the method, *isChecked()*, which returns Boolean state of button. To execute a function after a button is pressed, the button needs to be connected to the function in the initial method, to initialize the object's state. An example of connecting a push button (*RunSim*), is shown in listing 3.12. This is the basic concept of creating a connection between the XML and the Python code.

The *mplWidgets* in the GUI, in which the plots are displayed, are handled in the *PlotWidget.py* file. This file consist of classes and definitions necessary to operate the multiple plots accordingly. Different axis and layouts can be chosen from the functions, as well as, the option to clear and add lines in the plots.

To start the GUI the *eDESpy_GUI.py* file must be executed. This file is responsible for setting the correct path along with defining the application and window. When executed the window will open and the GUI is ready for usage. In the control center, all additional widgets and files necessary to run the model are imported.

3.3.2 Functionality and operation

In this section the functions of the GUI are discussed and the practical use of the GUI is explained. The functions of the GUI are extensive, from visualizing patient data to estimating new parameter sets.

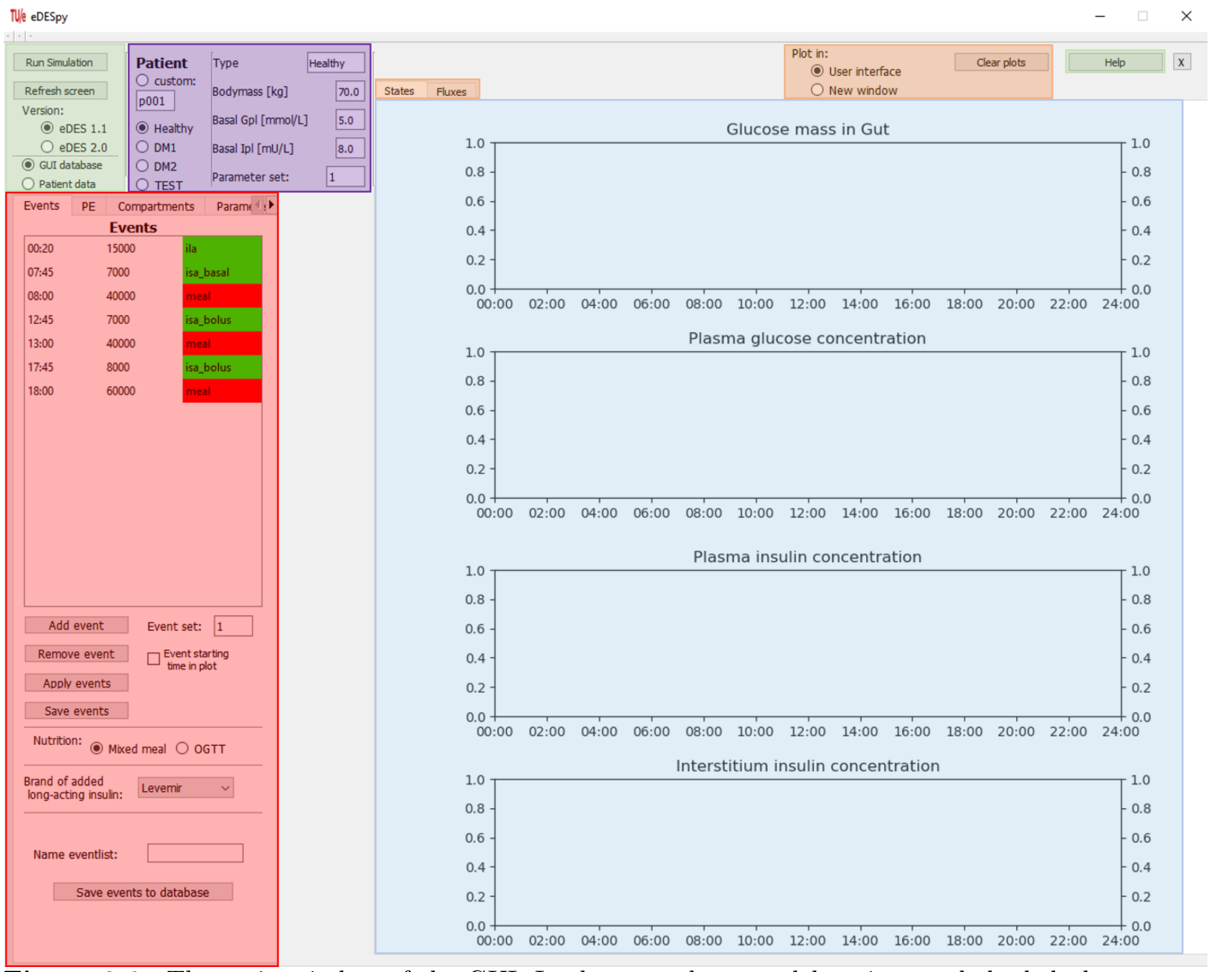
Installation settings. To install eDESpy the following steps must be taken:
1. Download and unpack the *eDESpy.zip* file.
2. The SQLite studio can be download, if desired, to operate directly in the DB, please see Section 3.2 for additional information.
3. Download Python (*www.python.org/downloads*). Or if desired download a Python integrated development environment (*https://www.anaconda.com/products/individual*).
4. Direct the path in *eDESpy_GUI.py*, to the location where *eDESpy.zip* is unpacked (E.g. *pth = "C:/Users/../../../eDESpy/"*). 5. Run the *eDESpy_GUI.py* file.

Main window of the GUI. When the *eDESpy_GUI.py* is computed, the window shown in Figure 3.6, appears. The colored boxes show different sections of the GUI. The green box (in Figure 3.6), contains the most powerful operations. The model version, along with the desired DB, can be chosen. Furthermore, with the *Run simulation* button, the simulation is computed. When changes are applied, press this button to recompute the simulation. Additionally, in the green box on the right of the window, a help button is located. When pressed, a file with additional information about the program is opened. The patient information is located in the purple square. With the radio buttons: Healthy, DM1, DM2, TEST; predefined subjects are chosen. Selecting the *"TEST"* subject, provides simulation with the patient and parameter set last worked on. With *"custom"*, a patient is retrieved from the DB by its corresponding patient id. In the line edits next to the radio buttons, information about the patient and the corresponding parameter set is displayed. It is also possible to adjust the parameter set in this text box. If the simulation is computed, the plots highlighted in blue, display the output in the state compartments of the simulation. If the version of the model is changed, the plots automatically change alongside to the correct state compartments. The fluxes of the output are shown by pressing the fluxes tap, located in the orange box. Additionally, the decision can be made to plot the graphs in new windows or clear the plots in the GUI. An important tabular is located in the red box in Figure 3.6. This tabular consist of six tabs. Each tab contains different functionalities of the model, which will be elaborated next.

Event list (EL). The section of the GUI regarding the event list, is located in the first tab. As previously explained, the event list consists of all of actions executed by a subject: consuming nutrition, applying medication and PE. In Figure 3.7, the event tab is showed, the table storing the event list is marked in red. The starting time, amount of carbohydrates/units and the type of event are displayed. The backgrounds of the types are colored different to obtain a quick overview of the EL. All events can be adjusted directly in the table. A complete event set can be retrieved from the DB using the *"Event set"* (accentuated with green in Figure 3.7). *"Add event"*, provides an extra row in the table, which can be filled in. To remove a row, select the event that needs to be dismissed and click the *"Remove event"* button. After finishing modifications to the EL, pressing *"Apply events"* runs the simulation with the newly created EL. An overview of the EL is printed in the Python console. *"Save events"* stores the created EL into the DB. In the purple box in Figure 3.7, the type of long-acting insulin, along with mixed meal or OGTT can be chosen. The checkbox marked with an orange square, shows the starting times of the events in the plot.

Compartments. This feature is explained in Section 3.1.2. In Figure 3.8, the tab containing the compartments is displayed. Different compartments of the model can be switched on/off by selecting the check boxes. Additionally, there is the option to adjust the efficiency. With these options it is possible to simulate certain diseases or deviations where organs, processes or medication in the body malfunction. The text box marked with red, in Figure 3.8, contains information on how to operate the compartments and efficiency. When the everything is set correctly, the *"Set compartments and efficiency"* button (highlighted in green) is clicked. An overview of the set compartments is printed in the Python console.

Parameters. The tab in which the parameters are shown is displayed in Figure 3.9. In this tab (3.9), all parameters can be adjusted as desired. When the mouse cursor hovers over the parameter name, a description of the parameter appears, as can be seen highlighted in red. The functions of the parameters along with the shown description change if the version of the model is adjusted. The value of each parameter can be adjusted by hand, with the slider or by adjusting the percentage, as is accentuated in



Version: [✓ eDES 1.1] eDES 2.0

[✓ GUI database] Patient data

Patient

custom: p001

[✓ Healthy] DM1 DM2 TEST

Type	Healthy
Bodymass [kg]	70.0
Basal Gpl [mmol/L]	5.0
Basal Ipl [mU/L]	8.0
Parameter set:	1

Plot in: [✓ User interface] New window

Events | PE | Compartments | Parame

Events

00:20	15000	ila
07:45	7000	isa_basal
08:00	40000	meal
12:45	7000	isa_bolus
13:00	40000	meal
17:45	8000	isa_bolus
18:00	60000	meal

Event set: 1

Event starting time in plot

Nutrition: [✓ Mixed meal] OGTT

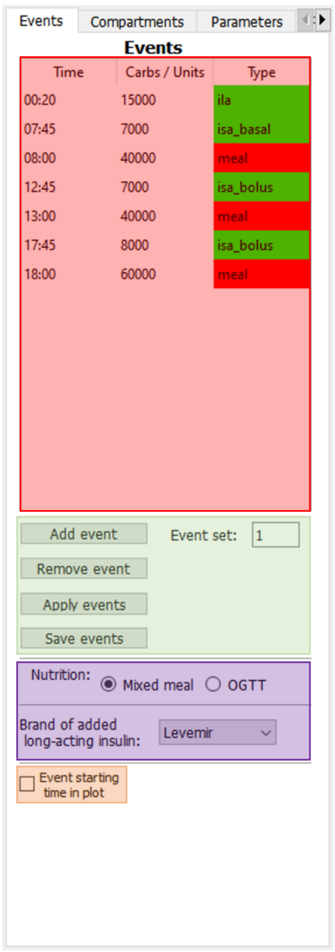
Brand of added long-acting insulin: Levemir

Name eventlist:

Figure 3.6: The main window of the GUI. In the green box, model options and the help button are displayed. In the purple box the patient and the according patient information can be selected. The blue box contains the location of the canvases in which the plots are displayed. Whereas the options in the orange box can be used for operating the plots. In the red box is a tabular displayed. The current tab is "Events", most of the functionalities of the model can be found in these tabs.

purple in Figure 3.9. When the slider is moved, the changed percentage is adjusted likewise. Pressing the button *"Parameter help"*, prints a list in the Python console with all parameters, description and value. The *"Restore parameters"* button, restores the initial parameter set from the DB. To observe the effect of the modified parameters, press *"Apply parameters"*, and the simulation is computed with the new parameter set. After naming the parameter set, it can be stored in the DB by pressing *"Save parameters"*. Subsequently, if the parameters are estimated in the GUI, the obtained values are displayed in this tap. Therefore, the storage of estimated parameter sets is done identically.

Physical Exercise. In this section only the controls of the PE modules in the GUI are explained, for additional information on the PE modules, see Chapter 5. In Figure 3.10, is the tab displayed necessary for the PE module. A text box containing information on how to use the PE module is highlighted in red. The possibility exists to choose between method 1 (M_1) and method 2 (M_2). The specific exercise information of M_1, can be inserted in the purple accentuated area. The necessary input is the slow down percentage, power level, starting time, warming up, steady exercise, and slow down time, along with the duration of the afterburn effect. There can be chosen between different the units of the power level: mg glucose burned, kilo calories and Joules. When everything is filled in correctly the specification are saved with the *"Apply specifications"*, and the simulation can be computed. If the *"Specified exc plot"* button is pressed, an intensity overview of the PE pops-up. An example of a PE intensity overview can be seen in Section 5, in Figure 5.2. If the second method is active, a start time and amount of carbohydrates are necessary input. Therefore, these activities are stored in the DB with type: *"activity m2"*. Adding a M_2 activity is done identically as adding medication or nutrition, which can be seen in Figure 3.7.



Events

Time	Carbs / Units	Type
00:20	15000	ila
07:45	7000	isa_basal
08:00	40000	meal
12:45	7000	isa_bolus
13:00	40000	meal
17:45	8000	isa_bolus
18:00	60000	meal

Event set: 1

Nutrition: [✓ Mixed meal] OGTT

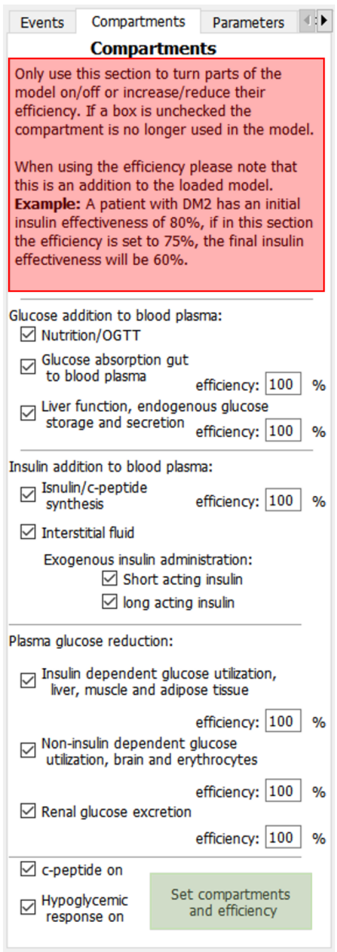
Brand of added long-acting insulin: Levemir

☐ Event starting time in plot

Figure 3.7: In red a table containing the event list, which can be adjusted directly in the table. The possibility exists to remove and add events, and save or retrieve event lists from the DB. There is the opportunity to choose the type of nutrition, as well as, the brand of long acting insulin.

Fluxes. The plots in the main window of the GUI (see Figure 3.6) are located in a tabular (accentuated with blue). It shows the calculated output in the state compartments, if the tab *"flux"* is clicked, the fluxes of the system are shown. Adjustments to these fluxes can be made in the tab shown in Figure 3.11. The plots containing the fluxes, can be operated and cleared independently from another. In the green box in Figure 3.6, the flux or the velocity (derivative) of the flux can be selected if desired. In the drop down box, highlighted in red, different types of fluxes can be selected. Furthermore, there is the opportunity to show the legend, with the number of the simulation or the name of the flux.

Data and parameter estimation. A parameter estimation can be done using the GUI, by selecting the tab in Figure 3.12. However, not all functionalities which exist in the Python scripts regarding the parameter estimation (see Section 3.4), are built into the GUI. Since this would be very time consuming, with little extra benefit. The data from estimations outside the GUI can be imported and used. Nonetheless, the possibility to execute a parameter estimation inside the GUI and play with the results exists. Firstly, assure that the correct DB is connected, this can be done in the main window (green area in Figure 3.6). In the in Figure 3.12 with orange highlighted area, there is the opportunity to plot the patient data in a line or scatter plot. In the green area, there is the possibility to use the events from the DB, corresponding with the patient, or an in the GUI adapted event list. Furthermore, the method used by the optimizer can be chosen and the computation can be started. Notice that the optimizer used (and its name), must corresponding with the Python *SciPy* package.



Events | Compartments | Parameters

Compartments

Only use this section to turn parts of the model on/off or increase/reduce their efficiency. If a box is unchecked the compartment is no longer used in the model.

When using the efficiency please note that this is an addition to the loaded model.
Example: A patient with DM2 has an initial insulin effectiveness of 80%, if in this section the efficiency is set to 75%, the final insulin effectiveness will be 60%.

Glucose addition to blood plasma:

- ☑ Nutrition/OGTT
- ☑ Glucose absorption gut to blood plasma — efficiency: 100 %
- ☑ Liver function, endogenous glucose storage and secretion — efficiency: 100 %

Insulin addition to blood plasma:

- ☑ Isnulin/c-peptide synthesis — efficiency: 100 %
- ☑ Interstitial fluid

Exogenous insulin administration:

- ☑ Short acting insulin
- ☑ long acting insulin

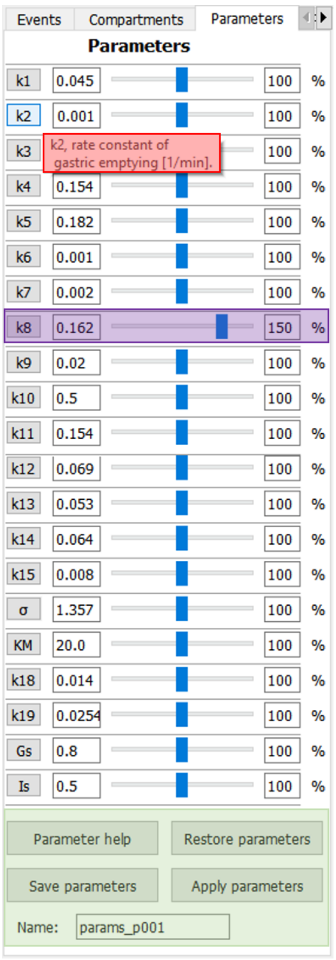
Plasma glucose reduction:

- ☑ Insulin dependent glucose utilization, liver, muscle and adipose tissue — efficiency: 100 %
- ☑ Non-insulin dependent glucose utilization, brain and erythrocytes — efficiency: 100 %
- ☑ Renal glucose excretion — efficiency: 100 %

- ☑ c-peptide on
- ☑ Hypoglycemic response on

Set compartments and efficiency

Figure 3.8: The tab containing the compartment setting. By (un)checking the boxes, compartments can be (de)activated. The efficiency of compartments can be adjusted. After finishing, the *"Set compartments and efficiency"* applies the changes to the model. The text box highlighted with red, contains information about operation of the compartments.



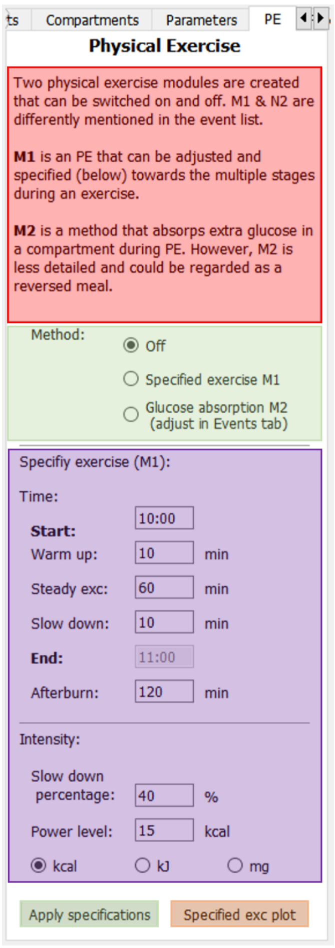
Parameters

k1	0.045	100 %
k2	0.001	100 %
k3		100 %
k4	0.154	100 %
k5	0.182	100 %
k6	0.001	100 %
k7	0.002	100 %
k8	0.162	150 %
k9	0.02	100 %
k10	0.5	100 %
k11	0.154	100 %
k12	0.069	100 %
k13	0.053	100 %
k14	0.064	100 %
k15	0.008	100 %
σ	1.357	100 %
KM	20.0	100 %
k18	0.014	100 %
k19	0.0254	100 %
Gs	0.8	100 %
Is	0.5	100 %

k2, rate constant of gastric emptying [1/min].

Name: params_p001

Figure 3.9: In this figure, the tab containing the parameters and their information is shown. If the mouse cursor is hovered over the name of the parameter, a parameter description appears, as is highlighted in red. All parameter value can be adjusted by typing, moving the slider, and changing the percentage (accentuated in purple). The help function prints the parameter description and the corresponding values of all parameters in the Python console. Furthermore, changed parameters can be applied to the model and saved in the DB. However, the original parameter values can be restored from the DB.


ts | Compartments | Parameters | PE

Physical Exercise

Two physical exercise modules are created that can be switched on and off. M1 & N2 are differently mentioned in the event list.

M1 is an PE that can be adjusted and specified (below) towards the multiple stages during an exercise.

M2 is a method that absorps extra glucose in a compartment during PE. However, M2 is less detailed and could be regarded as a reversed meal.

Method: [✓ Off] Specified exercise M1 Glucose absorption M2 (adjust in Events tab)

Specifiy exercise (M1):

Time:

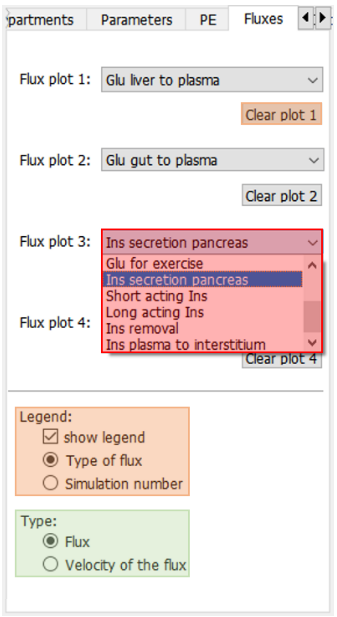
Start:	10:00	
Warm up:	10	min
Steady exc:	60	min
Slow down:	10	min
End:	11:00	
Afterburn:	120	min

Intensity:

Slow down percentage:	40	%
Power level:	15	kcal

[✓ kcal] kJ mg

Figure 3.10: Information about the PE is stored in this tab. The options for activating different PE modules is highlighted in green. The text box (red), contains extra information about how to use the PE modules. In the purple area, the exact specifications for a M_1 PE can be inserted. After insertion, the specifications must be applied to see their effect. The *"Specified exc plot"* button, provides a plot of the intensity course of the PE.


partments | Parameters | PE | Fluxes

Flux plot 1: Glu liver to plasma

Flux plot 2: Glu gut to plasma

Flux plot 3: Ins secretion pancreas

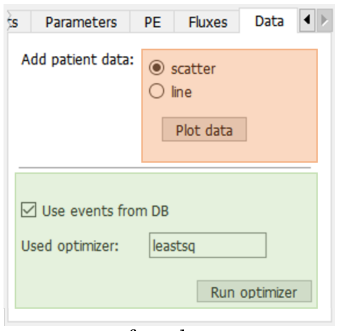
Glu for exercise
Ins secretion pancreas
Short acting Ins
Long acting Ins
Ins removal
Ins plasma to interstitium

Flux plot 4:

Legend: [✓ show legend] [✓ Type of flux] Simulation number

Type: [✓ Flux] Velocity of the flux

Figure 3.11: This is the tab responsible for operating the plots of the fluxes. Each flux can be plotted independently, see the with red accentuated drop down box. There is an option to choose between the flux or the derivative of the flux (green). Additionally, the plots can be individually cleared and different legends can be activated.



Add patient data: [✓ scatter] line

[✓ Use events from DB]

Used optimizer: leastsq

Figure 3.12: In this figure the tab necessary for the parameter estimation is displayed. The orange box shows the possibilities for plotting the patient data. In the green box are options for the optimizer located, there can be decided to use events from the DB, and choose the method of the optimizer.

3.4 Model calibration

To fit the simulation accordingly, parameters can be adjusted to obtain the desired shape of the simulation. These parameters can change per patient, whereas, each patient is different. However, most of these parameter sets are lacking and have to be estimated, to successfully calibrate the model. In the DB patient data containing blood glucose and insulin data over a period of time (approximately 12 hours) is stored. By adjusting the parameters the model can be fitted onto this patient data, providing a patient specific model. In this section we discuss the estimation of the parameters. The differential equations described in Chapter 2, contain parameters and based on the available data, these parameters are estimated. After successfully calculating the parameters, the model should fit the provided data, resulting in a calibrated model. The theory behind the used techniques is elucidated first.

Used patient data. For the parameter estimation patient data from the research of Maas et al. [67] is used. The patients in the data set are all connected to the Máxima Medical Centre. The data set contains ten DM1 and forty-three DM2 patients. Forty-seven measurements are taken for each patient. The glucose, insulin and c-peptide, concentrations in the blood are calculated. The patients in the data set are labeled with a patient id, the patients $p001 - p012$ are diagnosed with type 1 diabetes. For the DM1 patients, only blood glucose levels are available. All patients are diagnosed with the disease for at least one year and more than 18 and younger than 85 years old. All patients signed an informed consent and underwent a physical examination. Different patient groups are created, depending on the differences between the patients. These patient groups can be accessed easily in the SQLite DB. Before the measurement, the patients were asked to be (if possible) in a fasting state. An additional MATLAB file is created to extract the patient data from MATLAB and convert it towards the SQL DB. Not-a-Number (NaN), or zero values, in the patient data are deactivated. The eDESpy model measures insulin medication in milli-units (mU). In the data set, insulin medication is measured in units (U). Therefore, these values are converted to mU.

Maximum Likelihood Estimation (MLE). A useful tool for estimating a parameter set is the Maximum Likelihood Estimation (MLE). In this method values of the parameters are estimated, increasing the likeliness that the model fits the provided data. When the simulated output and the data increase in similarity, the likelihood increases too. Therefore, the likelihood function is maximized to achieve the best data approximation, by selecting the best suitable parameter values ($\hat{\theta}$), creating the optimal parameter set ($\hat{\boldsymbol{\theta}}$) [80].

The concept of maximization of the likelihood is equivalent to minimization of the objective function. Calculating the error, which is the difference between the model and data, provides an indication on how the model with its current parameter set matches the data. In this research the used objective function, shown in Equation 3.1, calculates the error, for each parameter set ($\boldsymbol{\theta}$).

$$SSE(\theta) = \sum_{i=1}^{N} (y_i(\theta) - \hat{y}_i)^2 \tag{3.1}$$

In this equation N is the number of data points, y_i is the calculated output, with use of parameter set ($\boldsymbol{\theta}$). The corresponding data point is $\hat{y}_i$. In Equation 3.2, the solution with the smallest error is found. Resulting in an optimal parameter set $\hat{\boldsymbol{\theta}}$, consisting of a vector containing the most likely parameters, see Equation 3.3.

$$\hat{\theta} = \arg\left(\min\left(SSR(\theta)\right)\right) \tag{3.2}$$

$$\hat{\boldsymbol{\theta}} = \begin{bmatrix} \hat{\theta}_1 \\ \hat{\theta}_2 \\ \vdots \\ \hat{\theta}_n \end{bmatrix} \tag{3.3}$$

The error is calculated separately for glucose, in the gut, plus glucose and insulin in the blood plasma. These values are concatenated and processed into the algorithm. For DM1 patients, no insulin measurements were available. Therefore the error of the insulin, in the cases of DM1 patients ($p001 - p012$), is not taken into account. The first definition in Listing 3.13 (*find_nearest*), finds points in the simulation output, nearest to the time points of the actual patient data. The points of the patient data, along with the corresponding points of the computed output, can be inserted in Equation 3.1. Following, the errors are calculated, as can be seen in Listing 3.13. In this research the least-squares fitting method ('*leastsq*'), from the *scipy* optimizer is used [73].

```python
# Calculating the SSE, from param_est.py

#-----------------------Find nearest DP---------------------------
def find_nearest(array, value):
    array = np.asarray(array)
    idx = (np.abs(array - value)).argmin()
    return idx

#------------------------Calc sum squared errors-----------------
def calculate_SSE(prm, timeDP, glucoseDP, insulinDP, mg_gut_DP, p_Type):
    par = Patients.Parameters(prm['k1'].value, prm['k2'].value, ...
                                                 prm['k19'].value)
    # run eDESpy, with new parameters.
    x = CalculateEvents.CalcEvents(model, x0, par, c, events, stepsize, duration, fun_id
    , efficiency_factor)

    # Find nearest indices.
    indices = [find_nearest(time, timeDP[i]) for i in range(len(timeDP))]

    # Calculate the errors.
    #DM1 patients (without insulin datapoints)
    if p_Type == 'DM1':
        SE_mg_gut = np.square(np.subtract(np.divide(x[indices,0],max(mg_gut_DP)), np.
    divide(mg_gut_DP, max(mg_gut_DP))))
        SE_glucose = np.square(np.subtract(np.divide(x[indices,1],max(glucoseDP)), np.
    divide(glucoseDP, max(glucoseDP))))
        All = np.concatenate((SE_glucose, SE_mg_gut))
    #DM2 patients (with insulin datapoints)
    else:
        SE_mg_gut = np.square(np.subtract(np.divide(x[indices,0],max(mg_gut_DP)), np.
    divide(mg_gut_DP, max(mg_gut_DP))))
        SE_glucose = np.square(np.subtract(np.divide(x[indices,1],max(glucoseDP)), np.
    divide(glucoseDP, max(glucoseDP))))
        SE_insulin = (np.square(np.subtract(np.divide(x[indices,2],max(insulinDP)), np.
    divide(insulinDP,max(insulinDP)))))
        All = np.concatenate((SE_glucose, SE_insulin, SE_mg_gut))

    # Removing all NaN.
    All = All[~np.isnan(All)]

    return All
```

Listing 3.13: In this listing a part of the computation of the SSE is displayed. The first method, *find_nearest*, finds the points which are closest to the patient data. The *calculate_SSE* definition, executes the computation of the SSE, as shown in equation 3.1. For patients with DM1, the insulin error is not computed, due to absence of insulin data.

Obviously, it is necessary to reduce the output of the objective function as much as possible. By adjustments to the parameters ($\theta_{1,2,..,n}$), the optimizer searches for parameter sets with a lower error (Equation 3.2). Most optimization algorithms use strategies to find the solution as quickly as possible, without having to try all possible changes in parameter values. MLE algorithms find a numerical solution of the minimized objective function in the parameter space. Commonly, the minimization of complex mathematical processes is a nonlinear least square problem. If the algorithm finds a minima, the possibility exists that it is a local minima. A local minimum, in contrary to a global minimum, is a minimum that is not the lowest possible solution [81]. Therefore, different algorithms and initial parameter values could be used to increase the chance of finding a global minimum [82].

Latin Hypercube Sampling. To increase the chance of finding a global minimum, multiple iterations should be computed using diverse initial parameter sets $\boldsymbol{\theta}$. Latin Hypercube Sampling (LHS) is used to obtain initial parameter samples over the entire parameter space [83]. LHS is an example of a computational algorithm which belongs to the category of Monte Carlo sampling methods [84]. Monte Carlo like sampling methods, repeatedly use random number generation to find, in this case, initial parameter values. The sample values are distributed according to a probability density function. When random sampling is used, the algorithm does not take into account any of the previously provided samples. Multiple iterations are necessary to increase the possibility of covering the entire parameter space. Latin Hyper cube sampling assures that over multidimensional axis no two points are on a similar axis. This reduces the amount of necessary sample points to cover the entire parameter space, and therefore, it reduces the chance of missing a global minima [81]. For implementing LHS into the model, the Surrogate modelling Toolbox (SMT) is used. SMT is an open-source package from Python, containing multiple sampling methods [85]. Due to the boundaries of the parameter space, which are mainly between 0.001 and 10, the output of the LHS is converted to a logarithmic scale, using a scaling function. For every (not fixed) parameter the initial value is stored in an array. The number of arrays (or rows in a Matrix form) created, is indicated with N.

Parameter estimation. The parameter estimation is an iterative process, necessary to reduce the error of the fitted model on the patient data. The code regarding the parameter estimation can be found in *param_est.py* and *eDESpy_param_est_all.py.* After the computation the newly optimized parameter sets are stored into the DB. These parameter sets ($\hat{\boldsymbol{\theta}}_{p001,\,p002,\,..,\,p082}$) correspond with individual patients. These parameter sets are unique and patient specific, therefore they can be regarded as VPs. These VPs are analyzed in Chapter 4. In this research, the lowest error is considered leading in deciding the best fit.

Initially, only a single meal was estimated and by visualizing the fit in the GUI, it could be observed that the data fit was quite weak. The glucose mass in the gut deviated significantly from the expected values, subsequently leading to a poor glucose and insulin data fit. However, the glucose mass in the gut is based on the model of Elashoff et al. [60], and should not deviate to physiologically unacceptable values after the estimation. To counter this problem, there is experimented with fixating digestion parameters and the addition of fictional data points, based on the previous eDES models. The fictional data points describe the expected glucose course in the gut. This is done by selecting data points from the simulated output, using a predefined parameter set ($\boldsymbol{\theta}_{Healthy}$, from the research of van Rozendaal et al. [16]) which is able to physiologically acceptable describe the glucose dynamics in the gut. Besides, the glucose and insulin error in the blood plasma, the error of glucose in the gut can also be calculated. All errors are appended to the minimization algorithm, as can be seen in Listing 3.13.

The results of a newly estimated parameter set ($\hat{\boldsymbol{\theta}}_{p021}$), a DM2 patient is shown in Figure 3.13. In Table 3.2, additional information concerning the parameter estimation is shown. LHS is used, with $N = 50$, to induce different initial parameter values. The minimizing method used is least squares. The parameter set containing the lowest residual, the total difference between an observed and fitted values, is chosen to be the best fit. For each patient an estimated parameter set ($\hat{\boldsymbol{\theta}}_{p0..}$) is stored in the DB. In Figure 3.13 it can be seen that the simulation with the newly fitted parameters (the orange line), fit the data points (green dots), significantly better than the initial parameters from the older eDES 1.1 version. This increased data fit is the effect of the calibration of the parameters on the patient data.

Patient id:	Method:	LHS:	N:	Data points:	Variables:	Residual:	Computation time:
$p021$	leastsq	yes	50	141	15	0.6334	2 : 07 : 19

Table 3.2: Additional information about the parameter estimation, executed by eDESpy. The output of the simulation is shown in Figure 3.13 (orange).

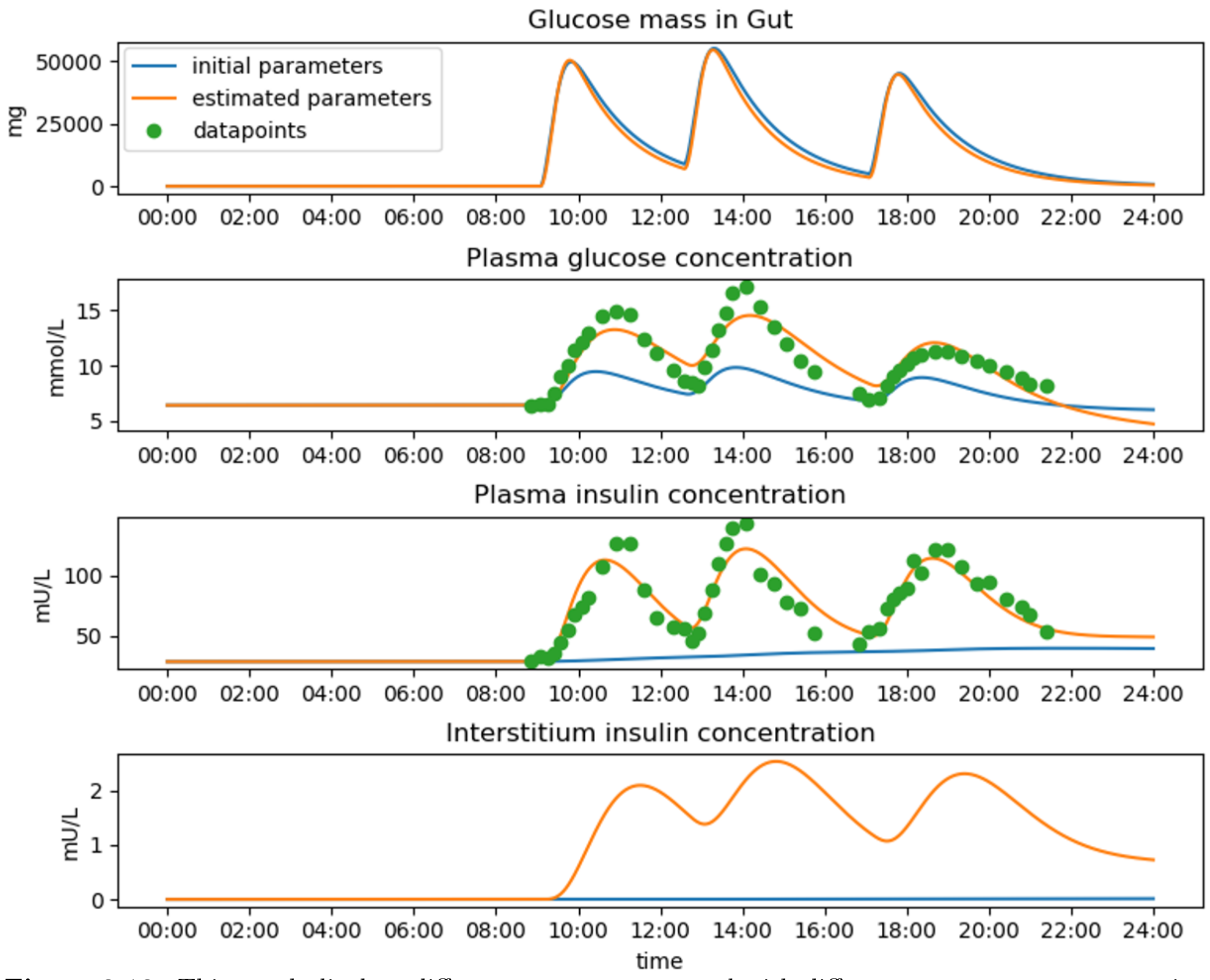


Figure 3.13: This graph displays different outputs, computed with different parameter set, concerning a DM2 patient. The output of an initial parameter set ($\boldsymbol{\theta}_{p021}$), from the research of van Rozendaal et al. [16], is depicted with a blue line. The data points, from the research of Maas et al. [67], are shown in green. Whereas, the output of the simulation, with a by the eDESpy model estimated parameter set, $\hat{\boldsymbol{\theta}}_{p021}$, is shown in orange.

3.5 Model validation

In this section the created Python model is compared with the previous MATLAB versions. This is necessary to assure that the eDESpy model is working accurately compared to its previous versions. The validation of the model is done by comparing the output results. This is done by extracting the output from the MATLAB simulation. This output is loaded into Python and plotted against the output of the eDESpy model. Regardless the differences in used software and program structure, when executed with identical input, the output should not differ. First the credibility of eDESpy 1.1 will be confirmed. The input for both models is shown in Table 3.3. The output of both summations is plotted in Figure 3.14. As can be seen, the results from both simulation, MATLAB and Python, coincide with no difference between the lines.

Name:	Start time:	Carbohydrates [mg]:	Units [-]	Administration
Levemir	00:20		15000	bolus
NovoRapid	07:45		7000	bolus
Breakfast	08:00	40000		
NovoRapid	12:45		7000	bolus
Lunch	13:00	40000		
NovoRapid	17:45		8000	bolus
Dinner	18:00	60000		

Table 3.3: This table displays the list of events which are used for validation of eDESpy 1.1. The events consists of nutrition and medication. The name of the meal or medication is mentioned, followed by the time of consumption, amount of carbohydrates or units, and the type of medication administration. The same set of events is used in the research of van Rozendaal et al. [16].

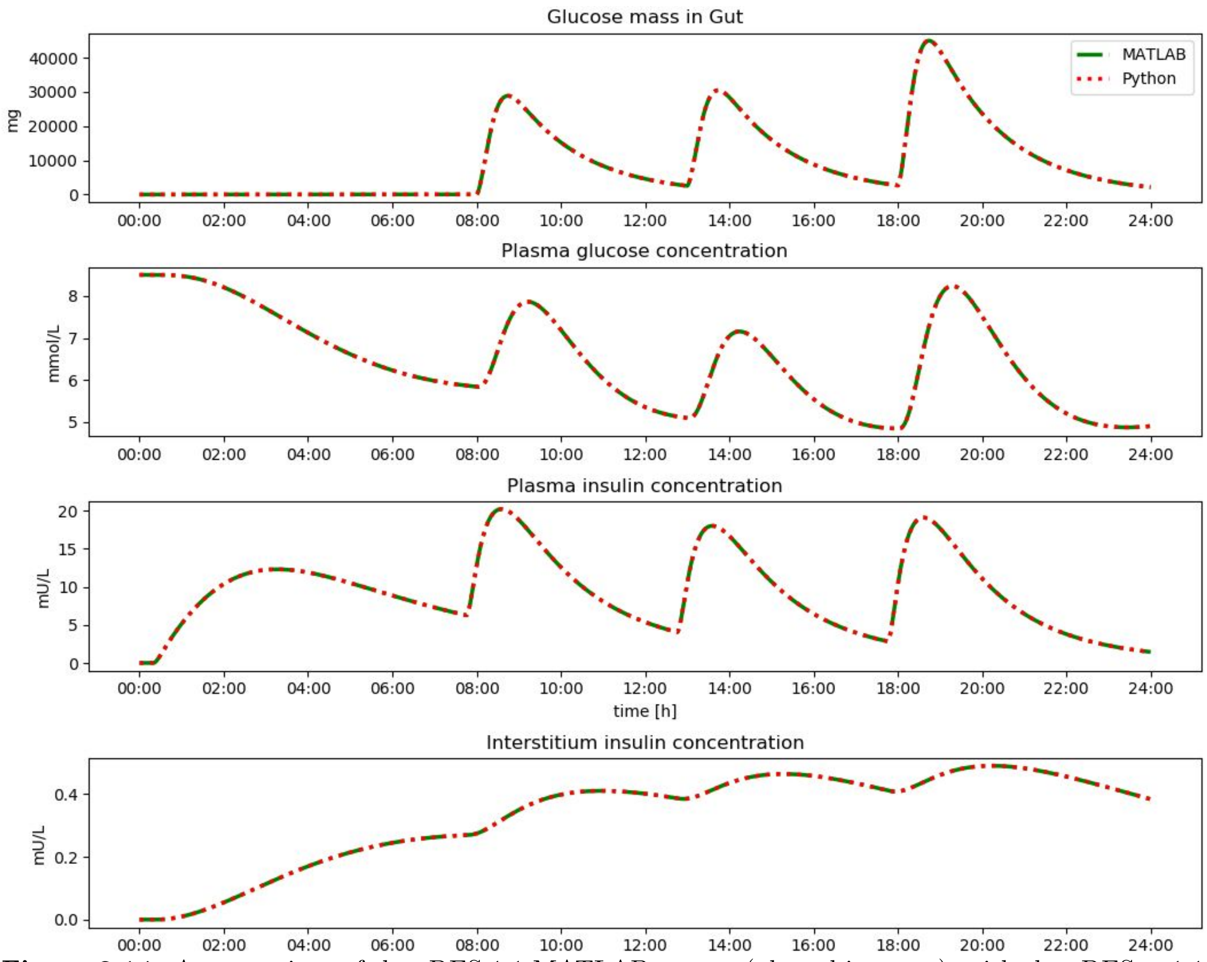


Figure 3.14: A comparison of the eDES 1.1 MATLAB output (plotted in green), with the eDESpy 1.1 (plotted in red) output. The input of both models is identical and is displayed in Table 3.3. As can be seen in this graph are both outputs in perfect agreement.

Additionally the 2.0 part of the eDESpy model is validated. This is done with a single meal, since the MATLAB version currently only supports single meal simulations. The meal, which contains 75 grams of carbohydrates, is consumed after one hour. In Figure 3.15 is shown that both plots are perfectly aligned. With these results it can be concluded that both models, eDESpy 1.1 and eDESpy 2.0, are in agreement, and therefore validated with their MATLAB counterparts.

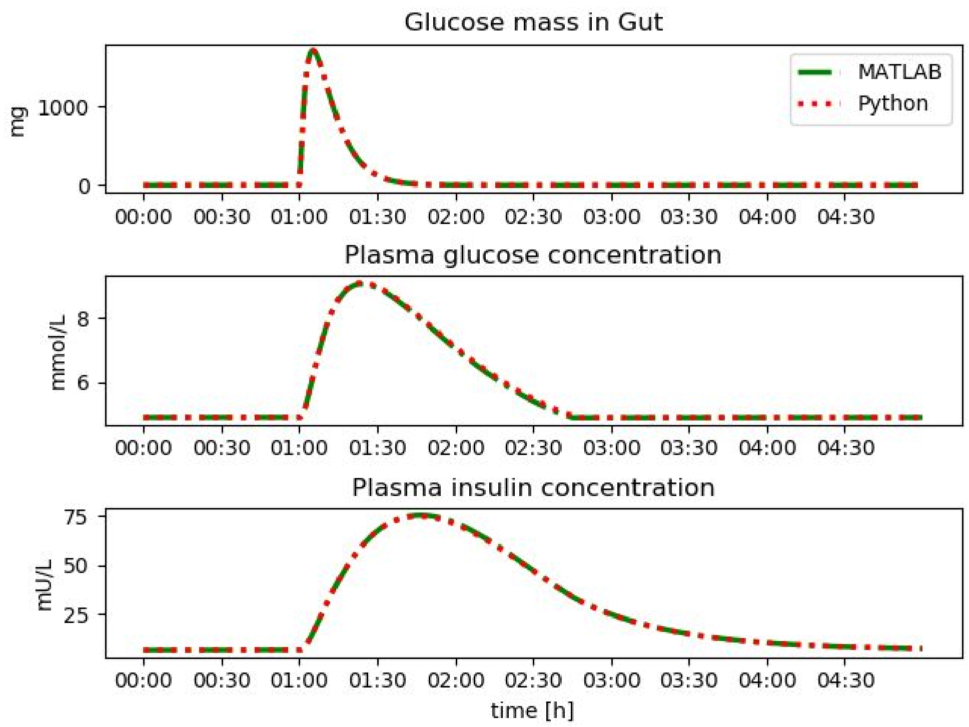


Figure 3.15: Validation of the eDESpy 2.0 model. Both the MATLAB and Python model are executed with the same input. The MATLAB output is plotted in green, whereas the Python output is plotted in red. After one hour a single meal (75 g carbohydrates) is consumed, the output of both models is identical.

Chapter 4

Virtual patients

As mentioned in Section 1.3, the usage of VPs could be an effective instrument for biomedical research. When the VPs are clustered into specific groups, new links between patients could be found. Furthermore, VPs provide a safe in silico experimenting environment [56]. In silico research, could provide insights in the conduct of a diabetic glycemic regulatory system [57]. These VPs are based on their individually estimated parameter sets $\hat{\boldsymbol{\theta}}$, which provide an unique metabolic fingerprint. Furthermore, clustering VPs could aid in exchange of missing parameters between VPs, or specifying the condition of their disease.

From the patient glucose and insulin data using a parameter estimation, parameters set are derived. These parameter set are unique, and are regarded as a VP. With these subjects, new simulations and tests can be computed in the eDESpy model. The complete group of patients is stored into a VPP, VPs can be clustered into groups according to their "metabolic fingerprint", see Section 1.3 [86]. To create the clusters in which the VPs are stored, the k-means method is used. The concept works with cluster centering and minimizing the total distance of all data points, in an assigned cluster [87, 88]. Multiple iterations are used with relocated cluster centers to find the optimum. At the lowest possible total variance, the optimum reached. If the optimum is found the clusters will no longer change. Since the standard k-means clustering methods work with optimizing the Euclidean distance between center and data points points, a logarithmic scale is introduced. By implementing a logarithmic scale, the distribution ratio between different sized parameters is evenly divided. To compute the k-means, the *sklearn.cluster* package is used [89]. All estimated parameters are retrieved from the DB and stored into arrays, as can be seen in Listing 4.1. If there are any zero values in the array, these values are set to a negligible small number. This is done for mathematical purposes, necessary to implement logarithmic scaling. The computation of the k-means can be seen in listing 4.1. All parameters are retrieved from the DB and stored into an array. This array is converted to a logarithmic scale, and the by clustering package can be used.

```python
# from k_means.py

from sklearn.cluster import KMeans

Patient_id_list = ['p001 t1', ... , 'p083 t1']
database = "eDESpy_data.db"
aps, evs, pdata, prm = read_db_tables(database)

def get_param(mp):
    for ps in prm:
        if ps.Type == mp:
            return ps
par_list = [get_param(Patient_id_list[i]) for i in range(len(Patient_id_list))]

def create_param_array(p):
    return [p.k1, p.k2, ... , p.sigma, p.KM]

par_array = np.array([create_param_array(par_list[i]) for i in range(len(par_list))])
par_array[par_array==0] = 1e-10          # Remove zero values from the array
par_array_log = np.log10(par_array)      # Convert to logarithmic scale.
```

```

#Compute k-means:
kmeans = KMeans(n_clusters=10, random_state=0).fit(par_array)

# Elbow method
def calc_kmeans(N):
    kmeans = KMeans(n_clusters=N, random_state=0).fit(par_array)
    return kmeans.inertia_

Cluster_list = list(range(1,10))
SSE_list = [calc_kmeans(Cluster_list[i]) for i in range(len(Cluster_list))]
```

Listing 4.1: The code used for k-means clustering. First the parameter sets are retrieved from the DB. All parameters are stored into arrays and converted to a logartimic scale, zero values are stored as a negligible number close to zero. Below that the code necessary for the elbow method is displayed. A function is created to calculate the total SSE (*kmeans.inertia_*) of the clustering function, changing the amount of clusters, and storing the SSE into a list.

To examine the desired amount of clusters the elbow method is used [87, 88]. The *calc_kmeans* definition, returns the total sum of squared errors (SSE). With the SSE is referred to the sum of squared distances of samples to their closest cluster center. Computing this definition with different amount of clusters ($N = 1$ till $N = 10$), as input, provides a list of different SSE. As a consequence of different amount of clusters. In Figure 4.1, the amount of clusters is plotted against the total SSE. The location of the "elbow", where the graph makes the strongest kink, points out the most applicable amount of clusters. In Figure 4.1, this location ($N = 2$) is marked with a red dot.

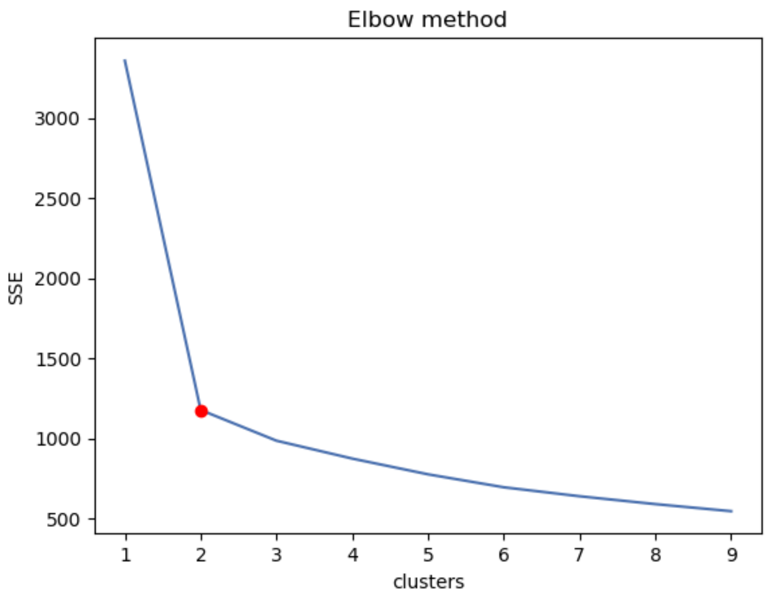


Figure 4.1: In this figure the SSE is plotted against the number of clusters. With use of the elbow method, the best applicable amount of clusters is determined, indicated with a red dot.

Primarily, all estimated parameters ($\hat{\theta}_{1,\,2,\,..,\,15}$) are taken into account. The amount of patients is quite low with respect to the relatively large amount of parameters. However, when more patients are available, using all parameters is the most favorable way. This is the most favorable way for the reason that the complete parameter set represents the metabolic fingerprint of a VP. The complete estimated parameter sets of all patients ($\hat{\boldsymbol{\theta}}_{p001,\,p002,\,..,\,p082}$) are displayed, using a logarithmic scale, in a heat map in Appendix Table D.1. Another heat map contains the deviation of each parameter towards the mean value of that parameter, see Appendix Table D.2. This heat map provides an overview regarding the deviation from the mean. The patient data exists of patients with type 1 and 2 diabetes, the k-means program is tested by the recognition of separation of these two groups. In Figure 4.1, the results of the elbow method are displayed. Resulting in the expected amount of clusters, $N = 2$. First, the clustering is done by involving the complete parameter set (Figure 4.2, left histogram). The accuracy of this clustering is 100%. Next,

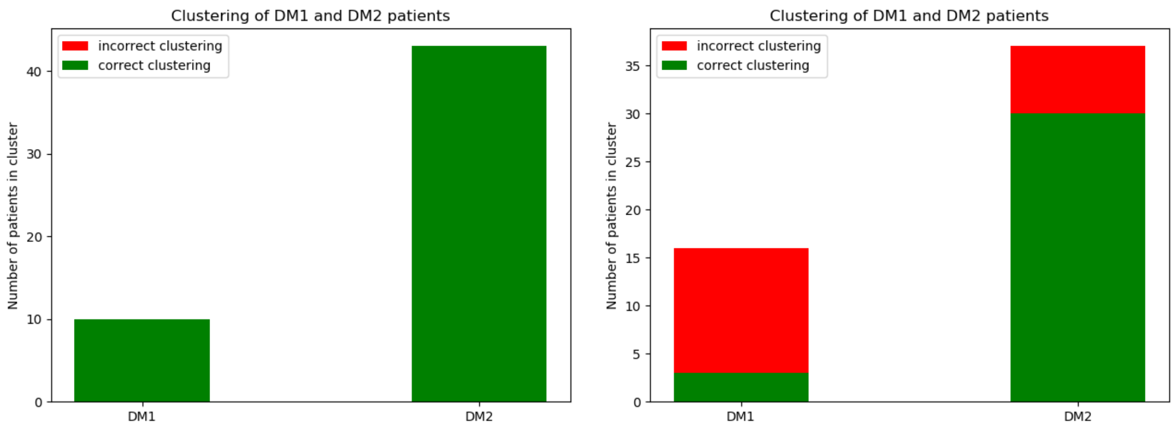


Figure 4.2: In this histogram the clustering of the type 1 and 2 patients is shown. In the Figure on the left, all parameters are used for clustering. In the right figure, parameters (p_{12}, p_{13} and p_{14}), are excluded from the clustering algorithm. Correctly clustered patients are displayed in green, whereas, incorrect clustering is displayed in green.

typically diverging parameters between DM1 and DM2 patients (p_{12}, p_{13} and p_{14}) are excluded from the clustering algorithm, resulting in the Histogram on the right 4.2.

According to literature [30], 10% of the type 2 diagnosed patients have non-diagnosed latent autoimmune diabetes in adults (LADA). Which is a form of type 1 diabetes [29]. According to Maas, four DM2 patients had immeasurable c-peptide levels (*p039*, *p044*, *p048* and *p062*). Which could be due to LADA, this matches the non-diagnosed LADA statistics from literature. Next will be investigated if these patients can be clustered into the same VP group. When the clustering algorithm is computed the clusters in Table 4.1 are derived. A green background displays the correctly clustered patients, whereas, a red background displays the falsely clustered patients.

Table 4.1: This table shows the clustering of the total VPP, using five clusters. The cells with a green background are successfully clustered, whereas, the patient with a red background is unsuccessfully clustered. The patients with no background color are DM2 patients, which are stored in DM2 clusters.

cluster 1, LADA	**cluster 2, DM1**	**cluster 3, DM2**	**cluster 4, DM2**	**cluster 5, DM2**
p039	p001	p015	p013	p014
p044	p003	p020	p016	p017
p048	p004	p021	p018	p019
p062	p005	p023	p022	p024
p079	p006	p031	p028	p030
	p007	p041	p029	p036
	p008	p058	p037	p040
	p010	p059	p043	p042
	p011	p061	p046	p054
	p012	p069	p049	p072
		p071	p050	
			p053	
			p057	
			p074	
			p076	
			p080	
			p082	

Chapter 5

Extending the model with physical exercise

It is known that PE has a significant impact on the glycemic regulatory system, as explained in Section 1.1.3 and 5.1. The reason why PE is added to the eDESpy model is that PE, just like nutrition and medication, could be a part of the lifestyle of a patient with diabetes. Modelling PE aids in the completeness of the model, and could provide new insights in the effects of PE on the glucose and insulin dynamics of a diabetes patient. Furthermore, in this section the extendability of the model, which is one of the requirements of eDESpy, is studied by the addition of a PE module. First the theory combined with the mathematics behind the expansion is explained, next there is elaborated on the implementation of PE into the model.

5.1 Theoretics behind implementing physical exercise

PE has a great impact on the metabolic system. The health benefits of activity are widely recognized. PE is considered a useful tool for managing diabetes and other metabolic diseases. However, diabetes patients should be careful during and after PE, as there is a risk of hypo or hyperglycemia [90]. As explained in Section 1.1.3, multiple processes are initiated during PE. The effects of these processes are modelled into the mathematical equations of the eDESpy model. However, the simulation of activity is quite challenging, since the effects of PE change according to the type of PE, intensity and duration. Physical characteristics of the subjects, e.g. gender, body weight, fitness status and age, influences the metabolic system significantly [91]. For simplification purposes differences in aerobic and anaerobic are not taken into account. Only the course of carbohydrates as source of energy are modelled, since eDESpy is limited to glucose dynamics.

The main effects of PE that need to be simulated, are shown in Figure 5.1. The glucose utilization during and post PE increases. Hepatic glucose production elevates, since glycogenolysis and gluconeogenesis increases and glycogenesis decreases. Furthermore, the plasma insulin levels decrease (see Figure 5.1). Additionally, increased insulin and β-cell sensitivity occurs. During PE the first glucose energy source is dietary glucose, followed by glycogen for short term energy needs. Gluconeogenesis becomes more dominant during long term PE [15].

One of the difficulties of modelling PE is determining its intensity. According to Breton et al. [92], heart rate is a valid reference point to measure PE intensity. Whereas the heart rate is highly correlated to the oxygen consumption. Another method to measure the intensity of an activity is using of the lactate threshold. Furthermore, measuring the subjects maximal oxygen uptake (VO_{2max}), is a method to refer to the vitality level of the subject [93]. Work rate in Watts is another approach of expressing energy consumption during PE [94, 15]. However, to implement these reference points detailed data sets including heart, work rate, or VO_{2max}, are needed. Unfortunately, these data sets are rare, especially for diabetes patients. In this research, it is chosen to measure the PE intensity using burned carbohydrates (mg), Joules, kilocalories, or different intensity levels. However, the model can be easily modified to work with VO_{2max} or heart rate.

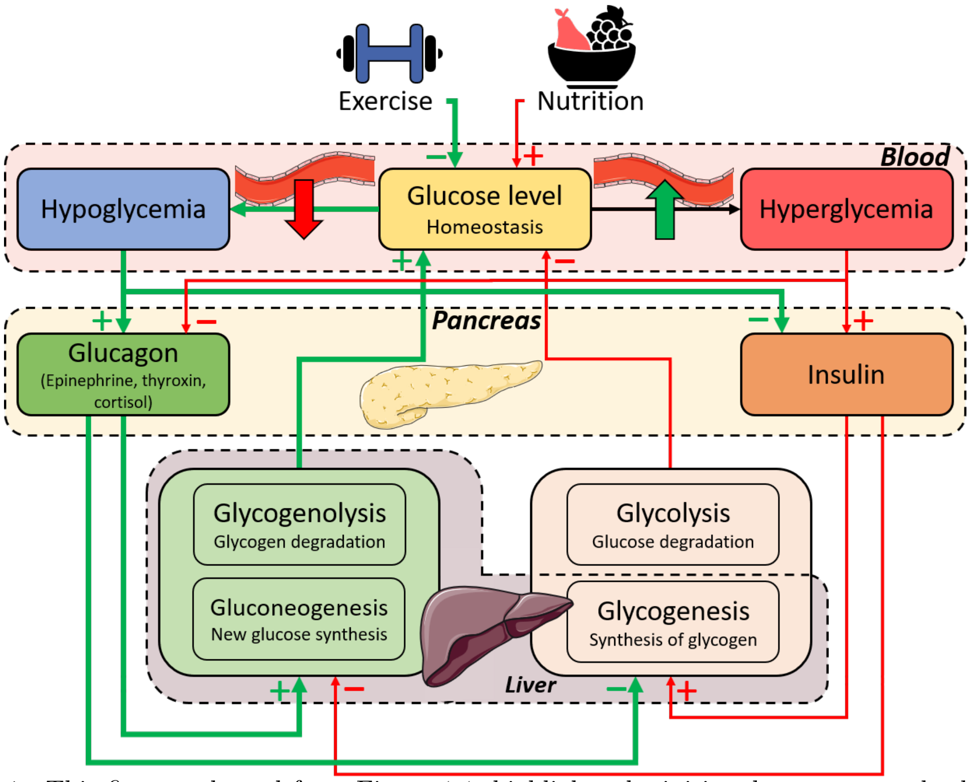


Figure 5.1: This figure, adapted from Figure 1.1, highlights the initiated processes and released hormones during PE (green arrows). Stimulation of processes is indicated with a plus sign. Whereas, the inhibition of processes is visualized with a minus sign. The processes induced during hyperglycemia, which are considered less relevant during a PE, are displayed with the red arrows.

Increased glucose utilization. When a subject starts exercising the percentage of carbohydrate metabolism to fat metabolism is circa $49 - 51\%$. After 100 minutes of PE the ratio is reduced to approximately 39% carbohydrate metabolism and 61% fat oxidation [15]. According to Wahren et al. [95], during forty minutes of light PE glucose uptake increases 7 fold, whereas during moderate and heavy PE the glucose uptake is able to increase 10-20 times. However, sources frequently conflict about exact increments during PE [95, 96]. These conflicts in literature could be due to high person-to-person differences.

Hepatic glucose production. As proposed by Parker et al. [97], PE could be separated into a short- and long term effect [98]. Due to elevated hepatic glucose production (glycogenolysis), the g^{liv} flux increases significantly. However, glycogen storage is limited, therefore this is a short term effect. Hepatic gluconeogenesis is initiated producing glucose in lesser amounts [97].

Increased sensitivity. During PE the muscle's sensitivity towards insulin can increase. This increment in insulin sensitivity has an acute and long term effect. This long term effect is accomplished by overall vitality and training programs [15, 90]. This insulin enhancement can lead to restoration or super-compensation of glycogen [24]. Furthermore, it is shown that PE can improve pancreatic β-cell functioning [99].

5.2 Addition of physical exercise

Creating the PE module is done by testing different approaches. Initially, the in the previous section discussed physiological effects are mimicked. The actual physiological complexity needs to be simplified in order to successfully adapt PE into the model. Afterwards the complexity can be increased.

The first approach was adjusting the parameters. Increasing glucose utilization of the muscles and increasing the hepatic glucose production, mimicking the processes of glycogenolysis and gluconeogenesis.

A temporary parameter increase, is used, the design and results can be found in Appendix C. Furthermore, there is experimented with increasing the β-cell and insulin sensitivity. The code is still available for future research, however, in the current model it is switched off. Since these attempts did not result in the desired behaviour of the system.

While creating the PE module two methods were promising. Due to the modularity of the model, both modules could be implemented into the eDESpy program. In the first method (M_1), the PE itself is exactly described. This makes the PE event, and the different stages of the PE, specific and adjustable. In the second method (M_2), the glucose usage is modelled based on the gastric emptying model of Elashoff et al. [60]. In both cases a new compartment is created to store the extra glucose used during PE. In the GUI can the different methods be activated, see Figure 3.10.

5.2.1 Physical exercise module, method 1

Method 1 (M_1). In this method the focus is on describing the PE as exact as possible in different phases. When the PE is initiated, the body will first be in a warming-up phase, slightly increasing the glucose usage. After finishing the warming-up, there will be a steady glucose consumption during the PE. This steady glucose utilization level is called the power level of the PE. Although, the glucose utilization of this power level will fluctuate in vivo, for modelling purposes it is set to a constant utilization. After finishing the steady exercise phase, the slow-down is started. Subsequently, if the slow-down phase is completed the PE is finished remaining the afterburn effect, which will extinguish over time [15]. All phases can be seen in Figure 5.2. The time span of all phases can be adapted to personal preferences. Likewise, the power level and the slow-down percentage can be adjusted in the GUI, as can be seen in Figure 5.3. Regarding the power level, different energy units can be chosen: Joule, the SI unit for energy; calories, which is an unit often used in sports and nutrition [15]; and the amount of carbohydrates in grams (or mg). The conversion between grams glucose, kilo calories and kilo Joules, is done according to the following ratio: $1\,g\ glucose = 15.7\,kJ = 3.74\,kcal$ [100]. Enabling the use of different energy units, increases the usability of the PE module.

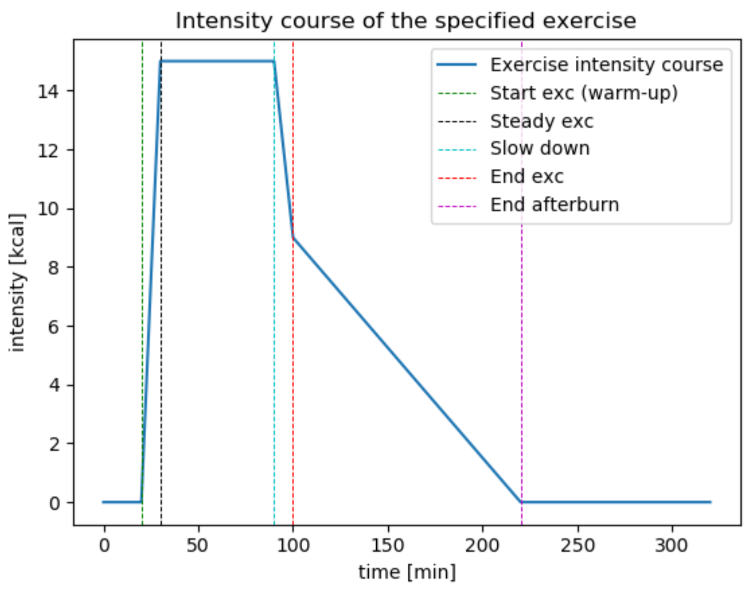
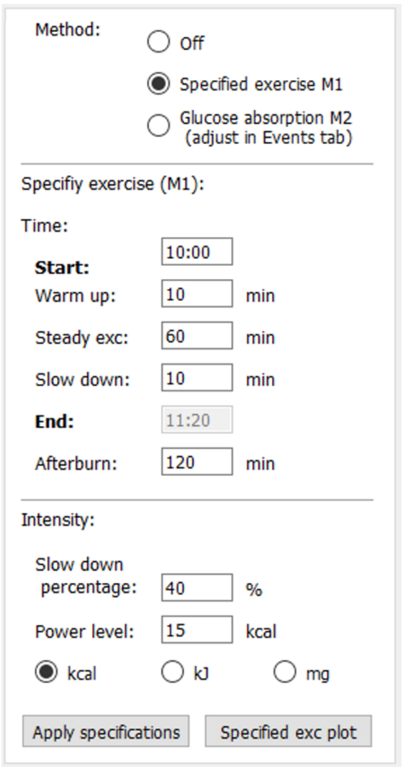


Figure 5.2: This plot displays the different phases during PE. It comprehends the warm-up, steady, slow-down and afterburn phase. The values can be found in Figure 5.3, where the specifications can be adjusted to the exact PE of the subject. On the x-axis is the time in minutes displayed, the y-axis shows the intensity in kJ, $kcal$ or mg glucose burned during the PE.

The intensity course of the specified PE in Figure 5.2, shows the amount of glucose utilized by the musculoskeletal system ($M_G^{\text{exc, M}_1}$), at a given point in time. This musculoskeletal system can be seen as an extra compartment, which is only activated during PE. Equation 5.1, converts $M_G^{\text{exc, M}_1}$, to a glucose flux moving from the blood plasma into the muscles, at the specific point in time. $M_G^{\text{exc, M}_1}$ can be


Method: ( ) Off [✓ Specified exercise M1] ( ) Glucose absorption M2 (adjust in Events tab)

Specifiy exercise (M1):

Time:

Start:	10:00	
Warm up:	10	min
Steady exc:	60	min
Slow down:	10	min
End:	11:20	
Afterburn:	120	min

Intensity:

Slow down percentage:	40	%
Power level:	15	kcal

[✓ kcal] ( ) kJ ( ) mg

Figure 5.3: A tab from the GUI in which the PE method can be activated. And in case of M_1 the activity can be precisely specified. The warm-up, start, steady exercise, slow-down and afterburn time can be filled in. The end time of the PE is than calculated. The power level and the percentage of slowing down intensity, can be inserted into the program. The percentage of slowing down is a reduction in intensity percentage from the steady exercise phase. If *"Specified exc plot"* is pressed, the window in Figure 5.2 appears.

inserted into Equation 5.1, which contains the unit conversion factor f, the glucose distribution volume in the blood plasma v_G and body mass M^{b}. The glucose absorption rate by the skeletal muscles during PE, is depicted as k_{19}.

$$g^{\mathrm{exc}} = k_{19} \frac{f}{v_G M^{\mathrm{b}}} M_G^{\mathrm{exc,\,M_{1\vee 2}}} \tag{5.1}$$

The g^{exc} is a new term reducing the plasma glucose levels during PE, as is shown in Equation 5.2. This is equal for both methods.

$$\frac{\mathrm{d}G^{\mathrm{pl}}}{\mathrm{d}t} = g^{\mathrm{liv}} + g^{\mathrm{gut}} - g^{\mathrm{non\cdot it}} - g^{\mathrm{it}} - g^{\mathrm{exc}} - g_{th}^{\mathrm{ren}} \tag{5.2}$$

Applying these additions to the system, provides results which are shown in Figure 5.4. The simulation is executed using a healthy subject and the event list is shown in Table 5.1. The precise specifications of the PE are shown in Figure 5.3.

The code activating the different methods, is displayed in Listing 5.1. When M_1 is activated, $M_G^{\mathrm{exc,\,M_1}}$ from Equation 5.1, is calculated with the code displayed in Listing 5.2. Next the changes to the blood glucose levels, are applied in Equation 5.2. This is for both methods similar. Therefore, regardless of the used method, the code in Listing 5.3 is executed.

```
#From CalculateStates.py
#Physical exercise statement, activating the different methods:
        if not iAct:  # Search for M2 PE
            mg_exc = 0
```

```python

        # A Method 2 exercise occurs:
        elif min(iAct) <= i and fun_id[14] == True:
            mg_exc = Equations.MgExc(x,t,p,i, EL,iAct)
        else:
            mg_exc = 0

        mgexc_pl = Equations.calc_mg_act_pl(p.k19, x[15])
        dMg_exc_dt = mg_exc - mgexc_pl

        # Specified exercise Method 1:
        if fun_id[15]:
            dMg_exc_dt = Equations.exc_funct(real_time, EL, i)
```

Listing 5.1: In this listing, the statements responsible for activating PE are shown. If M_1 is activated, the *exc_funct* method in Listing 5.2 is executed. Whereas, for M_2, Listing 5.4 will be executed.

```python
#From Equations_11.py
# Method 1 exercise function
def exc_funct(real_time, EL, i):
    rc = 0
    for j in range(i,len(EL)):
        if EL[j].type == 'activity m1':
            exc = EL[j]

            if real_time <= exc.start_time:
                MgExc = 0
                rc = 0
            if real_time >= exc.start_time and real_time < exc.start_time+exc.warm_up:
                rc = exc.power_level / exc.warm_up
                dt = real_time - exc.start_time
                MgExc = rc * dt
            if real_time >= exc.start_time+exc.warm_up and real_time < exc.start_time+exc.warm_up+exc.steady:
                rc = 0
                dt = real_time - exc.start_time - exc.warm_up
                MgExc = exc.power_level
            if real_time >= exc.start_time+exc.warm_up+exc.steady and real_time < exc.start_time+exc.warm_up+exc.steady+exc.slow_down:
                rc = -(exc.power_level*(1-exc.slow_down_perc)) / exc.slow_down
                dt = real_time - exc.start_time - exc.warm_up - exc.steady
                MgExc = exc.power_level + rc * dt
            if real_time >= exc.start_time+exc.warm_up+exc.steady+exc.slow_down and real_time < exc.start_time+exc.warm_up+exc.steady+exc.slow_down+exc.afterburn:
                dt = real_time - exc.start_time - exc.warm_up - exc.steady - exc.slow_down
                MgExc = (exc.power_level*exc.slow_down_perc) - (exc.power_level*exc.slow_down_perc)*dt/exc.afterburn
                rc = - exc.power_level/exc.afterburn*exc.slow_down_perc

            if real_time > exc.start_time+exc.warm_up+exc.steady+exc.afterburn+exc.slow_down:
                rc = 0
    return rc
```

Listing 5.2: This listing displays the specification of M_1. In this code the $M_G^{\text{exc, M}_1}$ is calculated. Which is visualized in Figure 5.2.

```python
def Glucose_in_Plasma(x,p,c,i, EL, .. , iAct):
    .
    .
    gexc = p.k19 * (c.fc / (c.vg * c.Mb)) * Mgexc
    dgpldt = gliv  + ggut - gnonit - git - gthren - gexc
```

Listing 5.3: For both PE methods, the effects on the plasma glucose levels induced by Equation 5.1, are calculated in this Listing.

Name:	Start time:	Carbohydrates [mg]:
Breakfast	08:00	40000
Activity M_1/M_2	10:00	
Lunch	13:00	40000
Dinner	18:00	60000

Table 5.1: A table containing the list of events used for the simulation of the M_1 PE, in Figure 5.4.

The results of the PE M_1 module can be seen in Figure 5.4. The plasma glucose and insulin levels decrease and the hepatic glucose production increases.

5.2.2 Physical exercise module, method 2

The second PE module, method 2 (M_2), is developed by using the extra created compartment as a storage for the increased glucose utilization. This compartment functions as the musculoskeletal system during PE, which is able to abruptly increase its uptake of glucose [15, 95]. With this compartment a glucose outflux from the blood plasma (g^{exc}) can be realized.

To enable dynamic PE changes in the system, a bypass from the blood plasma toward the skeletal muscles is created. The new compartment represents the additional need of glucose in the muscles to preform the PE. Following, the glucose will be absorbed from the plasma into the muscle compartment, using the rate constant of glucose uptake in muscles (k_{19}). Additionally, desired glucose uptake is lowered over time. This newly created method is based on an adjusted version of the gastric emptying model proposed by Elashoff et al. [60]. Physically, both processes are completely different. However, the modelling concept of both processes: a glucose plasma increase during nutritional processes; versus a decrease of blood glucose levels during PE; both with a delay due to the transportation of glucose between the biological compartments, is quite alike. Equation 5.3, is the created equation for glucose uptake in the skeletal muscles during PE. $m_G^{\text{exc, M}_2}$, is calculated in Equation 5.4. With shape factor σ, and exercise glucose utilization $D^{exc,\,M_2}$.

$$\frac{\mathrm{d}M_G^{\text{exc, M}_2}}{\mathrm{d}t} = m_G^{\text{exc, M}_2} - k_{19} m_G^{pl} \tag{5.3}$$

$$m_G^{\text{exc, M}_2} = \sigma k_{11}^{\sigma} t^{\sigma-1} e^{-(k_{11}t)^{\sigma}} D^{\text{exc, M}_2} \tag{5.4}$$

After substitution of $m_G^{\text{exc, M}_2}$, $M_G^{\text{exc, M}_2}$ is calculated. This term is inserted in Equation 5.1, obtaining g^{exc}, which is inserted in the final equation regarding the plasma glucose level (Equation 5.2).

$$\frac{\mathrm{d}M_G^{\text{exc, M}_2}}{\mathrm{d}t} = \sigma k_{11}^{\sigma} t^{\sigma-1} e^{-(k_{11}t)^{\sigma}} D^{\text{exc, M}_2} - k_{19} m_G^{pl}(t - \Delta t) \tag{5.5}$$

```python
#From Equations_11.py
# Method 2 exercise function
def MgExc(x,t,p,i, EL, iAct):
    MgExc = []
    len_iAct = sum(indexAct <= i for indexAct in iAct)
    for k in iAct[:len_iAct]:

        if k <= i:
            mg_carbs = EL[k].units
            tcurrent = EL[i].start_time
            tprevious = EL[k].start_time
            t2 = t + (tcurrent-tprevious)

            mg_act = mg_carbs * (p.k11*t2)**(p.sigma-1) * p.sigma * p.k11 * \
            math.exp(-(p.k11*t2)**p.sigma)

            MgExc = np.append(MgExc, mg_act)
    mgact = sum(MgExc)
    return mgact

# Apply Mgexc to plasma glucose
def calc_mg_act_pl(k19, Mg):
    mgpl = k19 * Mg
    return mgpl
```

Listing 5.4: This listing shows the computation of the PE M_2.

In Figure 5.5, the PE M_2 is applied to a healthy subject. The same event list, for executing M_1 is used, which can be found in Table 5.1). The consummation of glucose is set to $17\,g$, which produced the following results. When the PE starts, the plasma glucose and insulin are both lowered, and the hepatic glucose production is increased. In Figure 5.6, both methods are shown in the same figure for comparison.

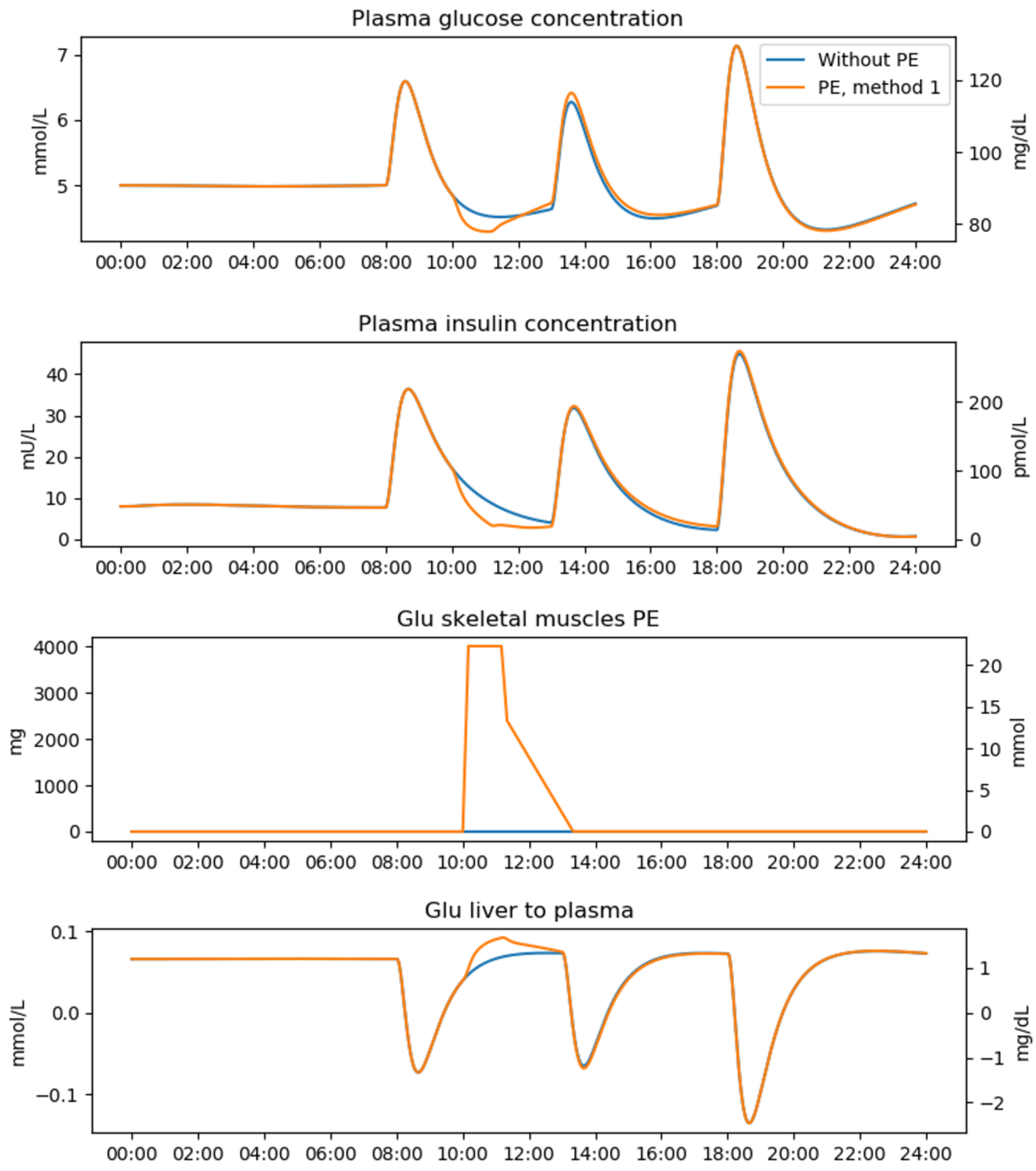


Figure 5.4: The output of the eDESpy simulation of a healthy subject. With a standard event list displayed in Table 3.3. However, a M_1 PE is added to the event list. The exact PE specifications are shown in the GUI in Figure 5.3. The blue line is the simulation without PE. Whereas the orange line is the output of the simulation with M_1 PE activated.

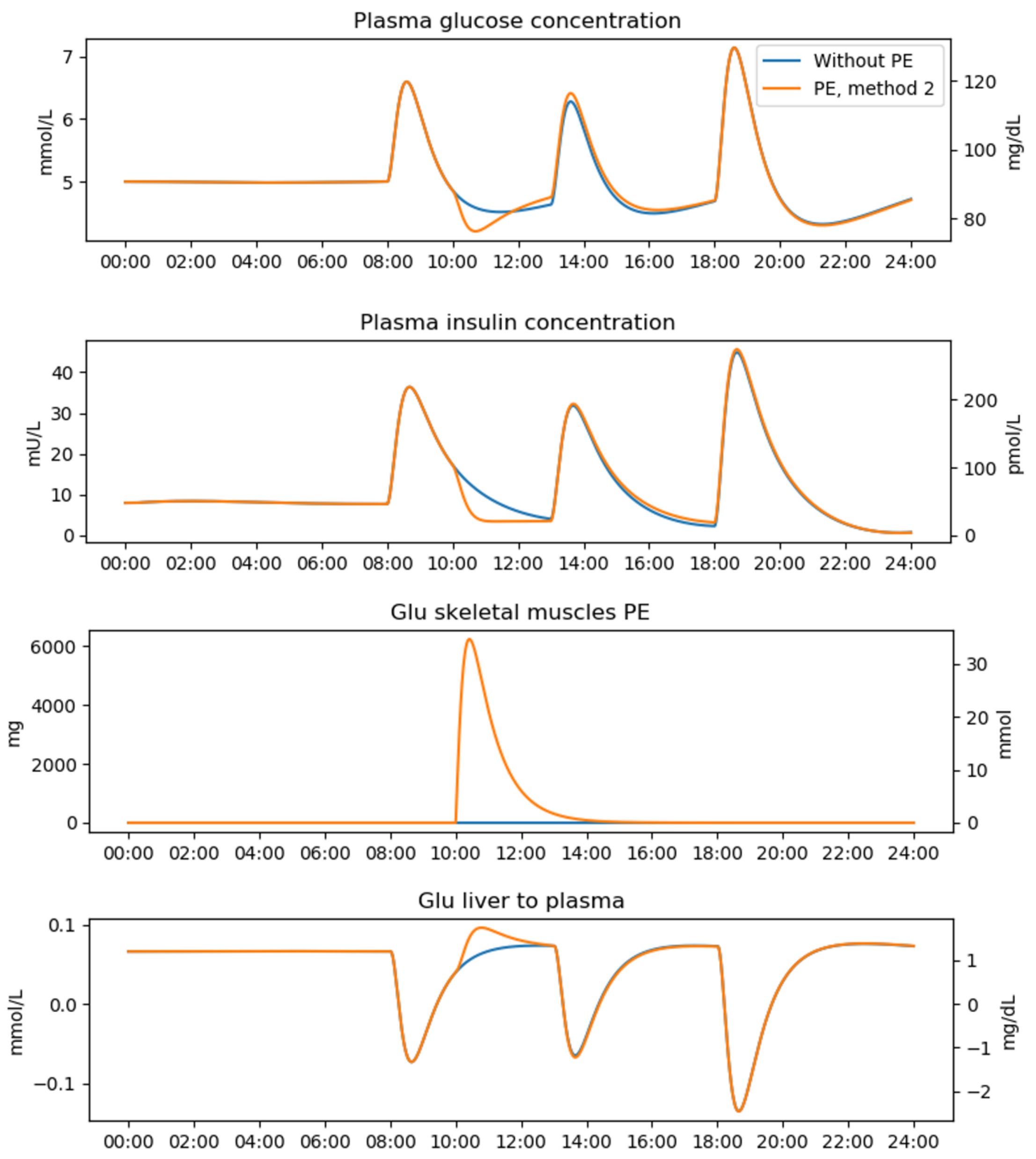


Figure 5.5: The output of the eDESpy simulation of a healthy subject. With a standard event list displayed in Table 5.1. However, a M_2 PE, with a consummation of around $10g$ glucose is added to the event list. The blue line is the simulation without PE, the orange line displays the output with PE M_2 activated.

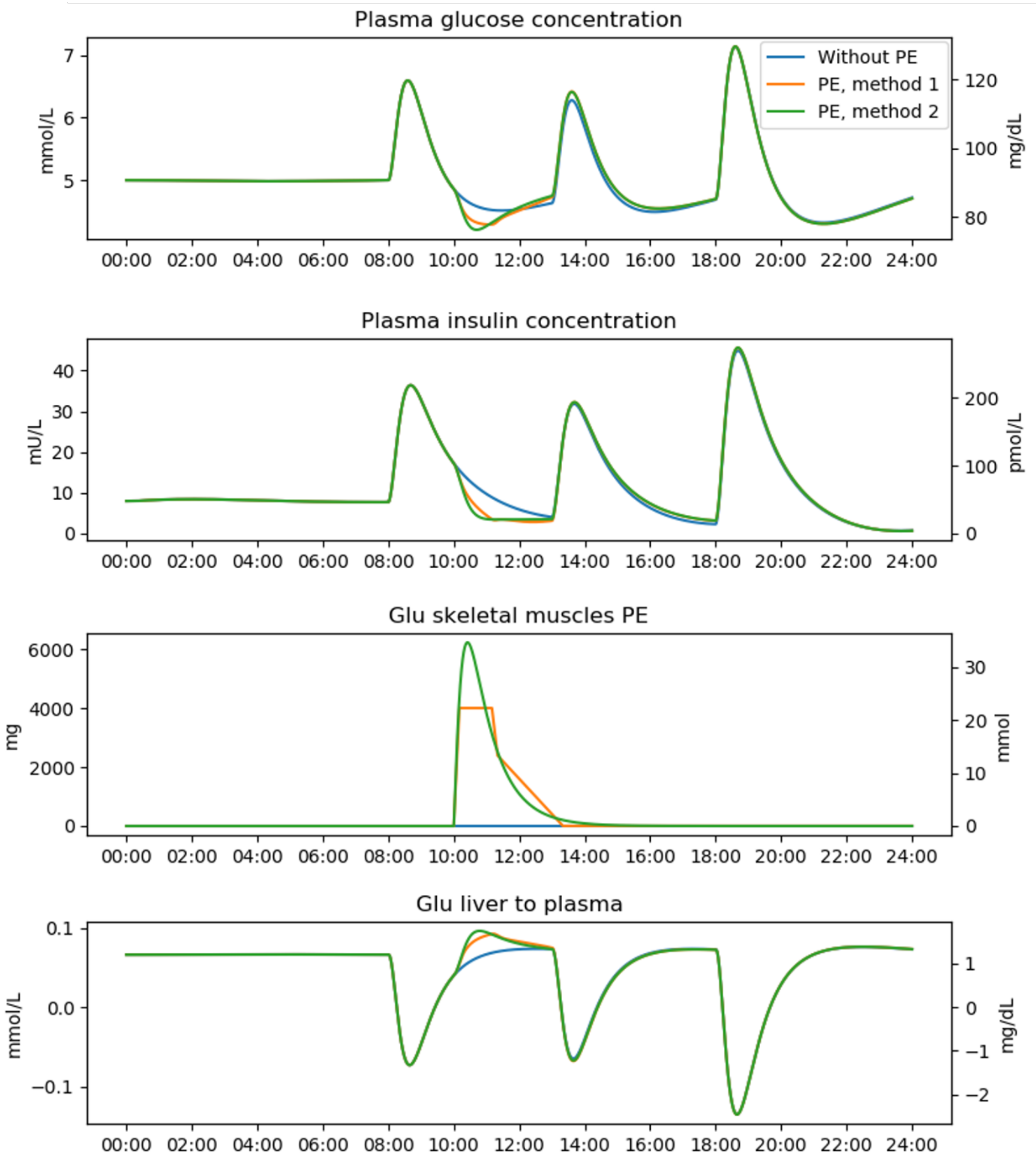


Figure 5.6: A comparison of the PE methods, no PE is shown with a blue line. PE M_1, an orange line, and PE M_2 is described with a green line. The event list in Table 5.1 is used. The specifications of the M_1 activity are shown in Figure 5.3.

Chapter 6

Conclusion and Discussion

6.1 Conclusion

In this research we set out to accomplish multiple goals: 1. Create a platform where multiple eDES versions can be accessed. 2. Create a flexible and object-oriented program. 3. Develop the model in a modular and extendable fashion.

The first goal is achieved and both models can be computed using the same program, providing an easily accessible platform (eDESpy). In Section 3.5, identical results between the Python and MATLAB versions are shown. Therefore, it can be concluded that the eDESpy model is validated successfully. Additionally, the eDESpy code is constructed in a more flexible and object-oriented way, compared to the previous more hardcoded eDES versions. This was the second requirement of the eDESpy model. Considered the interaction with the database (DB) and graphical user interface (GUI), it can be concluded that this requirement is achieved quite well. The created SQL DB, enables storage of events, parameter sets and patients. This DB is a useful extension of eDESpy, providing a structured archive of all necessary information regarding the model. Combining the DB with the GUI, provides accessible operation of multiple patients during different circumstances and events. Additionally, the GUI has shown to be an effective tool for operating the eDESpy model. The combination of the GUI and DB completely removes the necessity of hardcoded parameters, as was the case in previous eDES versions. To increase the flexibility even more, the eDESpy is created out of multiple compartments. Each compartment contains code describing specific organ functionality, metabolic processes, and processes like administration of medication or nutritional consumption. These compartments can be completely or partly, activated or deactivated. With the GUI, the user is able to simultaneously visualize changes between versions, patients, events, compartments, parameters, and more. The visualization of the previously mentioned components could be used to obtain a better understanding the eDESpy model, or imitate the effects of diseases or (partial) organ failure. The third goal is achieved by creating the model in the mentioned modular compartment structure, in which compartments can be disabled if desired. The extendability of the model is shown by creating two physical exercise (PE) modules. Both methods have particular advantages and can be computed simultaneously. Although the PE modules need to be calibrated, they are a promising start with the possibility to become a fine-tuned extension of the model.

Furthermore, for each patient in the provided patient data, an individual parameter set is estimated. These individual parameter sets are unique and can be regarded as a metabolic fingerprint. With these metabolic fingerprints, VPs are clustered into different patient groups, with similarities between their metabolic fingerprint. By combining these VPs, a virtual patient population (VPP) is composed. This VPP could be used for in silico research. The clustering algorithm is able to successfully cluster all DM1 patients. Furthermore, the k-means clustering algorithm has shown significant potential to recognize LADA patients, based on their metabolic fingerprint. Whereas, these patients were not labeled as LADA patients.

It can be concluded that the eDESpy model consisting of, a SQL database, a modular compartment structure, graphical user interface, physical exercise modules, virtual patient population, and clustering algorithm, has become a successful project with promising potentials.

6.2 Discussion

6.2.1 eDESpy

While, investigating the eDES model, a few weaknesses of the model occurred. There are physiologically incorrect possibilities in the system. If the simulation is stretched towards its physiologically acceptable limitations, malfunctions like negative concentration values or a negative pancreatic insulin flux can occur. Obviously, the pancreas is only able to produce, and not consume insulin. Therefore, it can be concluded that this is wrong by definition. Nevertheless, this does not mean that the model is not suitable for glucose and insulin modelling, it is a factor that should be taken into account. These complications could be improved in a followup version of the eDESpy model. Another possibility for improving the model, is adding an end time for a meal. Currently, a meal is consumed instantly, using a predefined digestion course. However, it could be useful to implement a carbohydrate consumption rate to show the difference between slow and fast eating patients. The DB and GUI are already preformed to consort with a meal end time. Therefore, if this is desired, it could be implemented into the model.

During the conversion of eDES from MATLAB to Python, the focus was on creating an object-oriented program. Therefore, the programming language Python is chosen for the platform since Python is an object based programming language and has a wide range of freely available applications and extensions [70]. The conversion started with reproducing the differential equations into separate definitions, which are all stored in an *equations.py* file. Following, definitions containing the equations are called upon from the *CalculateStates.py* file. This last file preserves an overview, regarding information on which definitions should be called upon and executed. In the previous eDES versions, all differential equations were computed in a single function. The segregation of functions, allowing all functions to be called upon independently, provides a significant improvement in flexibility. Programming the eDESpy model in this modular fashion, enabled the possibility to create switches to enable/disable compartments. Most switches are located in the *CalculateState.py* file, which regulates the overview of the program. Patients, constants, parameters and events in the eDESpy program are turned into objects. These objects can be easily accessed and stored in the DB. Furthermore, exchange and interaction between different objects is possible. Resulting in an accessible model, in which can be easily varied and exchanged between patients, parameter, and event sets. Object-oriented programming has proven to be very useful in the eDESpy project. Creating the model in an object-oriented fashion, enabled accessible expansion and adjustments of the code. Due to this type of programming, local changes could be modified without the necessity to adjust many other sections in the code.

As shown in Section 2.3, there are some differences between the 1.1 and 2.0 version of the model. In this research version 1.1 is used more often. The main differences are the addition of c-peptide and converting the interstitial fluid compartment into an insulin sink. As mentioned before, c-peptide has a longer half-life which provides a more stable test window compared to insulin [18]. Furthermore, c-peptide is unaffected by therapeutically administered insulin, providing insights in the endogenously produced insulin. Combined with the patient data, in which c-peptide concentration is measured, this could be a valuable addition. Although, the model is validated with the MATLAB version of the model (Figure 3.15), the complete 2.0 version contains a few limitations. In current research, this model is mostly used to compute the postprandial glucose and insulin dynamics, excluding medication, multiple meals and c-peptide. As can be seen in Section 2.3, the compartment of the interstitium, functions in eDES 2.0 as an insulin sink. This insulin sink is debatable. Whereas, using the interstitium as a potential buffer zone, provides a desired time delay in the insulin availability for the insulin dependent tissues. Furthermore, in eDES 2.0 a new statement describing a hypoglycemia response is introduced, as can be seen in Equation 2.31. This statement removes the physiologically expected plasma glucose dip after a meal. With eDESpy, the effects of this statement are visualized in Figure 3.3. It can be seen that deactivating this statement, provides the physiologically expected plasma glucose dip.

6.2.2 Database and GUI

The SQL DB is able to store all patient data, events, parameters and patients. The structure of the DB can be seen in Figure 3.5. Two DBs are created, *Patient_data.db* and *eDESpy_data.db*. Both DBs are identical, however, in the *Patient_data.db* the patient data used for estimating the parameters is stored. To prevent errors or indiscretion in the patient data, these DBs are separated. Whereas, the *eDESpy_data.db* DB is used to store in the GUI made adjustments to the events and parameters. The DB provides an overview of the data. Moreover, adjustments to the data can be made easily using the SQL program. If desired, individual rows in the DB can be (de)activated. This is useful to prevent mistakes, especially when working with multiple (older) parameter sets per patient. When a data row is deactivated the eDESpy program is unable to retrieve its containing information, assuring the user that undesired data is excluded from the simulation.

The GUI is a valuable tool for operating the model. The amount of effort to operate the functionalities of the model is significantly reduced. Events, patients and parameters can be easily adjusted, and multiple simulations can be computed simultaneously. Whereas, the different results are displayed in the interface, and can be compared. PE modules can be specified or turned on/off (see Figure 3.10), along with all different compartments and their efficiency, as displayed in Table 3.1. The GUI enables individual modifications of parameters, as can be seen in Figure 3.9. This feature enables the user to visualize the effect of parameters onto the model. Furthermore, the GUI enables the visualization of fluxes (see Figure 3.11), and embodies the possibility to execute a parameter estimation. The GUI is linked with the SciPy optimizing package, when the parameter estimation is executed, the parameters are loaded directly into the GUI. However, to use the complete functionality of the SciPy optimizing package, it is advised to use the additional Python files shown in Chapter 3.4 without the GUI. Whereas, the development of the GUI, is limited to the desired focus of the eDESpy model.

6.2.3 Physical exercise module

Modelling PE is quite a challenge. The reasons are that activities can be very diverse, and no typical PE exists. Duration, intensity, types of exercises, combined with the physical characteristics of the subject have divergent effects on the glycemic regulatory system [15, 91]. For the necessity of simplification differences in aerobic and anaerobic exercise are neglected. During PE the system must meet the following requirements: glucose utilization increases, plasma insulin levels decreases, and the hepatic glucose production is elevated, all within physiologically acceptable range (as described in Section 5.1, and shown in Figure 5.1). Multiple approaches are tested. First, is experimented with temporary adjustment of the parameters that should be affected during and after PE, as can be seen in Appendix C. However, this approach did not met the requirements needed to properly simulate the effects of PE on the glucose and insulin dynamics. Therefore, other solutions were investigated, leading to the development of two PE modules.

The created methods have both their advantages and disadvantages. In Method 1 (M_1), the course and specification of the PE is significantly more detailed. In Figure 5.2, is the intensity course of a PE displayed, which can be modified in the GUI as desired (see Figure 5.3). PE is divided into multiple phases of: warming up, steady PE, slowing down, and the afterburn effect; with different intensities. This is an advantage, since the impact of a PE is subjective. For example, if an elderly DM2 patient participates in an activity, the effects and the PE could differ completely compared to adolescent patient who participates in sports. Therefore, the possibility to adjust the different stages during the PE, including the afterburn effect, could provide new insights in the modelling of PE. M_1, uses a newly created compartment, mimicking skeletal muscles absorbing excess glucose only during PE. During PE M_1 (see Figure 5.4), the plasma glucose levels decrease, as expected. Plasma insulin levels decrease, and the newly created compartment, referring to the skeletal muscles, absorbs extra glucose. Hepatic glucose production increases during the PE, providing the blood with additional glucose.

The course of the PE in method 2 (M_2) is less specific compared to M_1. Whereas, in M_1 the different phases of a PE can be adjusted, M_2 only takes into account the amount of carbohydrates utilized during PE and the starting time. By adjusting the involved parameters the course of the PE could be influenced. However, in M_1 the type and specifications of the PE can be modified more easily. M_2 is based on a reversed version of the digestive system as proposed by Elashoff et al. [60]. Likewise M_1, the second method is using an extra compartment for temporary storage of the glucose essential during the PE. The inputs of M_2 are the starting time, combined with an indication of the carbohydrates consumed by the skeletal muscles during the PE. However, M_2 provides a more fluent function, which is closer to the actual expected physiological effects on the glycemic regulatory system. Compared to M_2, M_1 is more bulky. The contrast of both methods can be seen in Figure 5.6. The blue line corresponds with no PE, whereas the orange line resembles M_1, and the green line M_2.

The in Section 5.1 discussed requirements for the PE modules, are met in both modules. The plasma glucose and insulin levels should decrease and the hepatic glucose production should increase. These effects can be seen in Figure 5.6. Therefore, it can be concluded that the foundation of the physiological processes occurring during PE, are present in the new PE modules. However, the precise linkage of burned calories and its exact effect on the system, should be calibrated per patient. This calibration can be done with sufficient (diabetes related) PE data. The PE related parameters could be re-estimated, resulting in a more physiologically accurate PE module.

6.2.4 Virtual patients

Parameter estimation. The VPs are created based on the estimated parameter sets ($\hat{\boldsymbol{\theta}}$), which are derived from the patient glucose and insulin data. While estimating these parameter sets some complications occurred. Regarding the patient data, sources conflict about the units in which the insulin is measured. A "read me" file attached to the patient data, states that the concentration is in $[nmol/L]$. Converting these values provide physiologically diverging insulin levels. However, in the research of Maas et al. [67], and the eDES 2.0 MATLAB files, all occurrences of insulin concentration, milli-units per liter ($[mU/L]$) are used. Including the appointing of the basal insulin levels. Therefore, the assumption is made that the insulin levels are in $[mU/L]$. Additionally, every patient is very different leading to significant differences in medication schemes, and administered medication. For example, short-acting insulin bolus, can differ up to ten times. Therefore, the different VPs in the DB should be approached with medication schemes that, to some extent, correspond with their original medication. Another complication is the exclusion of metformin in the previous eDES models. By excluding metformin, the model is unable to predict the by metformin induced changes in the glucose/insulin dynamics. Leading to possible miscalculations in the parameter estimation, which could be improved. Therefore, implementing metformin, could be a useful extension of the eDESpy model and will be considered later in the recommendations (Section 6.3).

The parameter estimation started with fitting single meal simulations. However, this did not provide the desired fit. Due to significant deviations of the glucose mass in the gut, which subsequently leads to a poor glucose and insulin data fit. In the work of Maas et al. [16] and van Rozendaal et al. [67], glucose digestion and absorption is modeled quite similarly, based on the work of Elashoff et al. [60]. The output of the eDESpy model, should produce equivalent physiologically acceptable results as the previous eDES versions. Therefore, there is experimented with fixating digestion parameters and the addition of fictional data points, based on the previous models. These fictional data points are calculated using the corresponding parameters from eDES 1.1 calculating the glucose mass in the gut. Enabling, calculation of glucose errors in the gut, exactly like the errors of glucose and insulin in the blood plasma are computed. This provided significantly more accurate results. Whereas, the model is still able to diverge by the possibility of adjusting its parameters, it is guided in a physiologically acceptable direction. Additionally, to obtain a likewise effect, the parameters concerning the glucose absorption could be set to a fixed value. However, this solution could more obviously restrict the flexibility of the computed parameter set, leaving less room for adjustments.

Virtual patient population. The establishment of a VPP is done successfully. This population is stored in the eDESpy DB. The metabolic fingerprints are based on the estimated parameter sets. However, it could be interesting to add (besides the parameters), certain constants to the clustering algorithm. Interesting examples are adding the weight of the patient, basal glucose/insulin levels, and age. Furthermore, due to the significant differences in a patients medication schedule, it could be interesting to add medication to the clustering algorithm.

Clustering. The clustering of patients is done by using k-means. However, the k-means method has a few disadvantages. The main disadvantage is the necessity of pre-specifying the number of clusters. Due to the high dimensional space, parameters in their clusters can not be visualised easily. Therefore, the used elbow method is a useful tool aiding in choosing the most suitable amount of clusters. Which resulted in a clear distribution between DM1 and DM2 patients. Obviously, this is expected due to deactivated parameters involved in β-cell activity, regarding the DM1 patients. Furthermore, there is the possibility of obtaining divergent results if the order of the data changes. Additionally, the k-means method is sensitive to outliers. However, all parameters are estimated between predefined boundaries. The heatmap in Figure D.1, displays the parameters in logarithmic scale. To visualize the deviation of each parameter towards the mean value of that parameter, another heatmap is created (Figure D.2).

Limited patient data was available for this research, making it hard to precisely cluster the patients into groups, using this many parameters. However, when the amount of VPs increase over time, the value of the clustering algorithm could increase. Whereas more VPs with similarities can be clustered, and hopefully more patterns between patient types can be found. In the left histogram 4.2, the DM1 and DM2 patients are separated based on their estimated parameter set. The preferable amount clusters is determined using the elbow method (Figure 4.1), and is set to two. The clustering has an accuracy of 100%, since the parameters p_{12}, p_{13} and p_{14}, in DM1 patients are set to zero. However, it clearly shows the capabilities of the clustering algorithm. Next the parameters which are close to zero in DM1 patients, are excluded from the k-means clustering algorithm, resulting in the right Histogram 4.2. The accuracy is reduced. However, this reduction is expected since the parameter differences are less deviant, and the typically diverging parameters between DM1 and DM2 patients are excluded. Besides this decrease in accuracy the clustering algorithm still looks promising.

Next, the clustering algorithm is tested to separate the Latent Autoimmune Diabetes in Adults (LADA), from the rest of the patients in the DB. According to Maas et al. [67], four patients were marked as LADA patients (*p039*, *p044*, *p048* and *p062*). These patients should be hoarded into the same cluster, based on their $\hat{\boldsymbol{\theta}}$. With five clusters, and 53 VPs, the clustering algorithm is computed. The results are shown in Table 4.1. The LADA cluster contains five patients. Meaning that one patient without LADA, is incorrectly assigned to the LADA cluster. Nonetheless, it is better to falsely include a patient, followed by additional research to verify if the patient has no LADA. Compared to missing a LADA patient, and excluding this patient from additional examinations or adaptations in medication. This clustering example, which also successfully clustered all DM1 patients, demonstrates that the algorithm looks very promising and is reasonably good in clustering VPs.

6.3 Recommendations

Currently, new versions and extensions of eDES are being developed in MATLAB. It would be very interesting to accommodate these new versions into the platform of eDESpy, whereas, the handles to achieve this are build into eDESpy. In future research, the eDESpy model could be improved by the addition of extra types of medication. Besides the different types of insulin, metformin usage of DM2 patients is also monitored in the used patient data set. Metformin lowers the blood glucose levels by decreasing the hepatic glucose production, since it restrains glycogenolysis and gluconeogenesis, and metformin increases glycogen synthesis. Additionally, metformin decreases glucose absorption in the gut and insulin sensitivity [44, 50, 51]. The by metformin induced changes in glucose dynamics, can currently not be anticipated by the eDES models. The setup required for the addition of this medication is already build into eDESpy. Therefore, if the control of metformin is desired, it can be added to the model.

At this point the eDESpy program is limited to glucose and insulin dynamics. However, the blood sugar levels are affected by many other hormones and processes. When looking for opportunities to improve the completeness of the model, adding other hormones besides insulin could be an opportunity. An example could be the introduction of glucagon to the system [101]; or the process of gluconeogenesis, which converts non-carbohydrate substances into glucose [7]. However, it could also be discussed that adding more hormones would unnecessarily overcomplicate the model, and a more simplified and robust model is preferred.

Furthermore, it would be interesting to expand the model by implementing stress or mental states into the model. Whereas, mental states significantly influence glucose dynamics, since it stimulates the release of various hormones [102, 103]. Simulating the consequences of a patients mental state could provide insights in the response of the glucose-insulin regulatory system. Furthermore, it would be an addition to the completeness of the eDESpy model.

Another functional improvement of the eDESpy model, could be recording the step by step operations executed in the GUI. By recording these steps and the details of the simulations, previously computed simulations could be loaded into the GUI. From this point the simulation could be continued. These recorded actions could be stored, and retrieved, from the SQL DB. This addition adds extra functionality to the eDESpy model.

Additionally, the quality of smartwatches and wearables increased significantly over the last decade [104]. Therefore, linking this opportunity to the eDESpy model could be a very valuable asset. By linking these technologies, the PE module could be specified with the exact moved distance, speed, and heart rate. This could provide opportunities to fine-tune the PE module, from the spectrum of a young athlete running a marathon, to an elderly person taking a walk around the park. During sports, the intensities of PE, can severely fluctuate. Measuring these fluctuations could improve the eDESpy PE complexity from simple constant pace exercises (e.g. walking, cycling, running), to more complicated sports. Furthermore, using a smartwatch able to measure the surrounding temperature could be a valuable addition to the PE module. Since the temperature can significantly influence the metabolism dynamics in the human body [15].

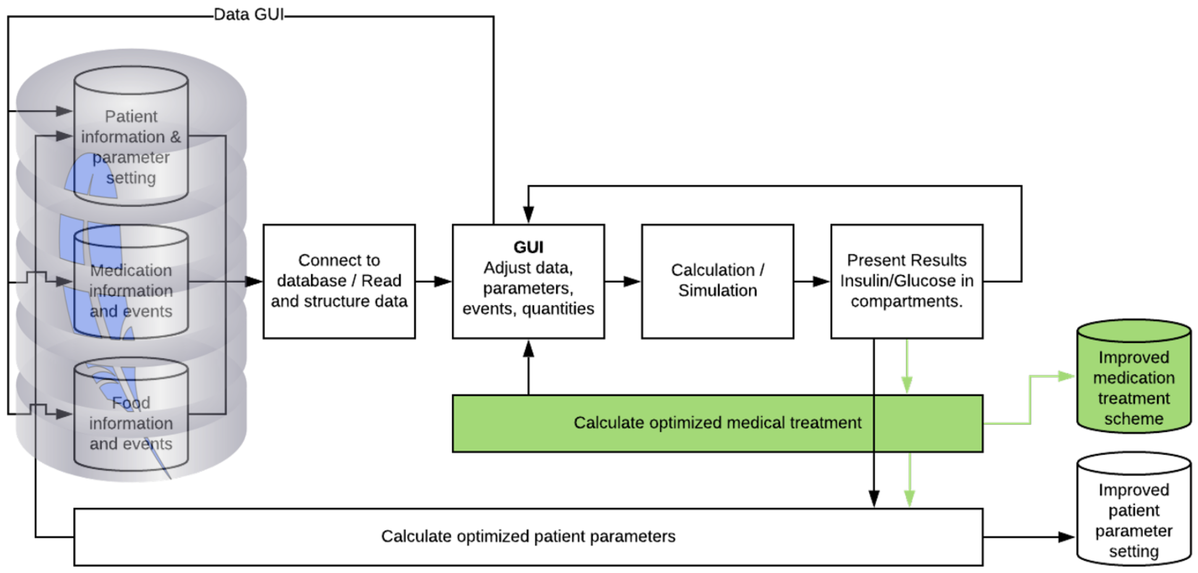

Data GUI
Patient information & parameter setting
Medication information and events
Food information and events
Connect to database / Read and structure data
GUI
Adjust data, parameters, events, quantities
Calculation / Simulation
Present Results Insulin/Glucose in compartments.
Improved medication treatment scheme
Calculate optimized medical treatment
Improved patient parameter setting
Calculate optimized patient parameters

Figure 6.1: An addition to the flowchart in 3.1. Displaying the different sections of the eDESpy model. The green boxes indicate where possible futuristic improvements, like an optimized medication scheme, should attach with the eDESpy program.

In Figure 3.1, the current structure of the eDESpy model is displayed. However, it would be fascinating to see if the eDESpy model is able to give predictions regarding the nutrition, medication or PE. Therefore, the program needs to know the desired glucose trend and possible options. By calculating the effects of certain options, the model could provide the patient with feedback regarding the cleverness of possible future events. It could even provide suggestions for an optimized medical treatment. Combined with the technology of real time continue glucose monitoring, many innovational opportunities arise. The updated structure of the eDESpy model, displayed in Figure 6.1, is showing the location where should be intervened in the eDESpy model, to realise these improvements. For this type of predictive medication modelling, it would be wise to look into the possibility of applying machine learning algorithms [105]. Although many improvements to the model must be made, the basic handles for these updates are already available.

Appendix A

List of abbreviations

ATP	adenosine triphosphate
DB	database
DM1	diabetes mellitus type 1
DM2	diabetes mellitus type 2
eDES	Eindhoven diabetic education simulator
eDESpy	Eindhoven diabetes simulator python
EGP	endogenous glucose production
EL	event list
GLUT4	glucose transporter type 4
GUI	graphical user interface
HbA1c	glycosylated hemoglobin
IDE	integrated development environment
LADA	latent autoimmune diabetes in adults
LHS	Latin hypercube sampling
M_1	method 1 (physical exercise)
M_2	method 1 (physical exercise)
MLE	maximum likelihood estimation
ODE	ordinary differential equation
OGTT	oral glucose tolerance test
OOP	object-oriented programming
PE	physical exercise
SQL	structured query language
SSE	sum of squared errors
VP	virtual patient
VPP	virtual patient population
XML	extensible markup language

Appendix B

Mathematical models

The complete set of mathematical equations of each model and the corresponding constants and parameters are stored in this section of the appendix. Notice that the equations in eDESpy 1.1 are extracted from the research of van Rozendaal et al. [16]; whereas, the equations of eDESpy 2.0 are obtained from the work of Maas et al. [67].

B.1 eDESpy 1.1

Glucose dynamics of the gut

$$\frac{\mathrm{d}M_G^{\mathrm{gut}}}{\mathrm{d}t} = m_G^{\mathrm{gut}} - m_G^{\mathrm{pl}} \tag{B.1}$$

$$m_G^{\mathrm{gut}} = \sigma k_1^{\sigma} t^{\sigma-1} e^{-(k_1 t)^{\sigma}} e^{-(k_{18} t)} D^{\mathrm{meal}} \tag{B.2}$$

$$m_G^{\mathrm{pl}} = k_2 M_G^{\mathrm{gut}} \tag{B.3}$$

Glucose dynamics in the blood plasma

$$\frac{\mathrm{d}G^{\mathrm{pl}}}{\mathrm{d}t} = g^{\mathrm{liv}} + g^{\mathrm{gut}} - g^{\mathrm{non-it}} - g^{\mathrm{it}} - g_{th}^{\mathrm{ren}} \tag{B.4}$$

$$g^{\mathrm{liv}} = g_0^{\mathrm{liv}} - k_9 \left(G^{\mathrm{pl}} - G_0^{\mathrm{pl}}\right) - k_{10} \beta I^{\mathrm{rem}} \tag{B.5}$$

$$g^{\mathrm{gut}} = k_2 \frac{f}{v_G M^{\mathrm{b}}} M_G^{\mathrm{gut}} \tag{B.6}$$

$$g^{\text{non-it}} = k_3 \Gamma_0 \frac{G^{\mathrm{pl}}}{K_{\mathrm{M}} + G^{\mathrm{pl}}} \tag{B.7}$$

$$g^{\mathrm{it}} = k_4 \beta I^{\mathrm{rem}} \frac{G^{\mathrm{pl}}}{K_{\mathrm{M}} + G^{\mathrm{pl}}} \tag{B.8}$$

$$g_{th}^{\mathrm{ren}} = \begin{cases} k_8 \left(G^{\mathrm{pl}} - G_{th}^{\mathrm{pl}}\right) & \text{if } G^{\mathrm{pl}} > G_{th}^{\mathrm{pl}} \\ 0 & \text{if } G^{\mathrm{pl}} \leq G_{th}^{\mathrm{pl}} \end{cases} \tag{B.9}$$

Insulin dynamics in the blood plasma

$$\frac{\mathrm{d}I^{\mathrm{pl}}}{\mathrm{d}t} = i^{\mathrm{pnc}} + i^{\mathrm{sa}} + i^{\mathrm{la}} - i^{\mathrm{rem}} \tag{B.10}$$

$$i^{\mathrm{pnc}} = \beta^{-1}\left(k_{12}\left(G^{\mathrm{pl}} - G_0^{\mathrm{pl}}\right) + \frac{k_{13}}{\tau_{\mathrm{i}}}\int\left(G^{\mathrm{pl}} - G_0^{\mathrm{pl}}\right)\mathrm{d}t + k_{14}\tau_{\mathrm{d}}\frac{\mathrm{d}G^{\mathrm{pl}}}{\mathrm{d}t}\right) \tag{B.11}$$

$$i^{\mathrm{sa}} = \frac{k_{15}}{v_I M^{\mathrm{b}}}U_I^{\mathrm{sc2}} \tag{B.12}$$

$$\frac{\mathrm{d}U_I^{\mathrm{sc1}}}{\mathrm{d}t} = u^{\mathrm{sa}} - k_{16}U_I^{\mathrm{sc1}} \tag{B.13}$$

$$\frac{\mathrm{d}U_I^{\mathrm{sc2}}}{\mathrm{d}t} = k_{16}U_I^{\mathrm{sc1}} - k_{17}U_I^{\mathrm{sc2}} \tag{B.14}$$

$$i^{\mathrm{la}} = \frac{h t_{0.5}^h t^{h-1}}{v_I M^{\mathrm{b}}\left(t_{0.5}^h + t^h\right)^2}U^{\mathrm{la}} \tag{B.15}$$

$$t_{0.5} = aU^{\mathrm{la}} + b \tag{B.16}$$

$$i^{\mathrm{rem}} = k_5\left(I^{\mathrm{pl}} - I_0^{\mathrm{pl}}\right) \tag{B.17}$$

Insulin dynamics in the remote compartment

$$\frac{\mathrm{d}I^{\mathrm{rem}}}{\mathrm{d}t} = i^{\mathrm{pl}} - i^{\mathrm{it}} \tag{B.18}$$

$$i^{\mathrm{pl}} = k_6\left(I^{\mathrm{pl}} - I_0^{\mathrm{pl}}\right) \tag{B.19}$$

$$i^{\mathrm{it}} = k_7 I^{\mathrm{rem}} \tag{B.20}$$

B.2 eDESpy 2.0

Glucose dynamics of the gut

$$\frac{\mathrm{d}M_G^{\mathrm{gut}}}{\mathrm{d}t} = m_G^{\mathrm{gut}} - m_G^{\mathrm{pl}} \tag{B.21}$$

$$m_G^{\mathrm{gut}} = \sigma k_1^{\sigma} t^{\sigma-1} e^{-(k_1 t)^{\sigma}} e^{-(k_{11} t)} D^{\mathrm{meal}} \tag{B.22}$$

$$m_G^{\mathrm{pl}} = k_2 M_G^{\mathrm{gut}} \tag{B.23}$$

Glucose dynamics in the blood plasma

$$\frac{\mathrm{d}G^{\mathrm{pl}}}{\mathrm{d}t} = g^{\mathrm{liv}} + g^{\mathrm{gut}} - g^{\mathrm{non\cdot it}} - g^{\mathrm{it}} - g_{th}^{\mathrm{ren}} \tag{B.24}$$

$$g^{\text{liv}} = \begin{cases} g_b^{\text{liv}} - k_3 \left(G^{\text{pl}} - G_b^{\text{pl}} \right) - k_4 \beta \left(I^{\text{pl}} - I_b^{\text{pl}} \right), & \text{if } G^{\text{pl}} > G_b^{\text{pl}} \\ g_b^{\text{liv}} - k_{10} \left(G^{\text{pl}} - G_b^{\text{pl}} \right) - k_4 \beta \left(I^{\text{pl}} - I_b^{pl} \right), & \text{if } G^{\text{pl}} \leq G_b^{\text{pl}} \end{cases} \tag{B.25}$$

$$g^{gut} = \frac{f}{v_G M^b} m_G^{pl} = k_2 \frac{f}{v_G M^b} M_G^{gut} \tag{B.26}$$

$$g^{\text{non-it}} = c_2 \frac{G^{\text{pl}}}{K_M + G^{\text{pl}}} \tag{B.27}$$

$$c_2 = g_b^{\text{liv}} \left(\frac{K_{M,\text{healthy}} + G_{b,\text{healthy}}^{\text{pl}}}{G_{b,\text{healthy}}^{\text{pl}}} \right) \tag{B.28}$$

$$g^{\text{it}} = k_5 \beta I^{\text{pl}} \frac{G^{\text{pl}}}{K_M + G^{\text{pl}}} \tag{B.29}$$

$$g^{\text{ren}} = \begin{cases} \frac{c_1}{v_G M^b} \left(G^{\text{pl}} - G_{th}\text{pl} \right), & \text{if } G^{\text{pl}} > G_{th}\text{pl} \\ 0, & \text{if } G^{\text{pl}} \leq G_{th}\text{pl} \end{cases} \tag{B.30}$$

Insulin dynamics in the blood plasma

$$\frac{\mathrm{d}I^{\text{pl}}}{\mathrm{d}t} = i^{\text{pnc}} + i^{\text{sa}} + i^{\text{la}} - i^{\text{liv}} - i^{\text{if}} \tag{B.31}$$

$$i^{\text{pnc}} = \beta^{-1} \left(k_6 \left(G^{\text{pl}} - G_b^{\text{pl}} \right) + \left(\frac{k_7}{\tau_{\text{i}}} \right) \int \left(G^{\text{pl}} - G_b^{\text{pl}} \right) \mathrm{d}t + \left(\frac{k_7}{\tau_{\text{i}}} \right) G_b^{\text{pl}} + \left(k_8 \tau_{\text{d}} \right) \frac{\mathrm{d}G^{\text{pl}}}{\mathrm{d}t} \right) \tag{B.32}$$

$$i^{\text{sa}} = \left(nr_1 - nr_3 \right) \frac{1}{v_I M^{\text{b}}} U_I^{\text{sc2}} \tag{B.33}$$

$$\frac{\mathrm{d}U_I^{\text{sc1}}}{\mathrm{d}t} = u^{\text{sa}} - nr_2 U_I^{\text{sc1}} \tag{B.34}$$

$$\frac{\mathrm{d}U_I^{\text{sc2}}}{\mathrm{d}t} = nr_2 U_I^{\text{sc1}} - nr_1 U_I^{\text{sc2}} \tag{B.35}$$

$$i^{\text{la}} = \left(1 - k_e \right) \frac{h \left(t_{0.5} \right)^h t^{h-1}}{\left(\left(t_{0.5} \right)^h + t^h \right)^2} \frac{1}{v_I M^{\text{b}}} U^{\text{la}} \tag{B.36}$$

$$t_{0.5} = a U^{\text{la}} + b \tag{B.37}$$

$$i^{\text{liv}} = c_3 I^{\text{pl}} \tag{B.38}$$

$$c_3 = k_{7,\text{ healthy}} \frac{G_{b,\text{ healthy}}^{\text{pl}}}{\beta \tau_{\text{i}} I_{b,\text{ health}}^{\text{pl}}} \tag{B.39}$$

$$i^{\text{if}} = k_9 \left(I^{\text{pl}} - I_b^{\text{pl}} \right) \tag{B.40}$$

C-peptide dynamics in the blood plasma

$$\frac{\mathrm{d}C^{\mathrm{pl}}}{\mathrm{d}t} = c^{\mathrm{pnc}} - c^{\mathrm{liv}} + c^{Y} \tag{B.41}$$

$$\begin{aligned} c^{\mathrm{pnc}} = \tfrac{v_I \alpha i^{\mathrm{pnc}}}{v_C} = \tfrac{v_I \alpha}{v_C \beta} \left(k_6 \left(G^{\mathrm{pl}} - G_b\mathrm{pl} \right) + \left(\tfrac{k_7}{\tau_{\mathrm{i}}} \right) \int \left(G^{\mathrm{pl}} - G_b^{\mathrm{pl}} \right) \mathrm{d}t \right. \\ \left. + \left(\tfrac{k_7}{\tau_{\mathrm{i}}} \right) G_b\mathrm{pl} + \tfrac{v_I \alpha}{v_C \beta} \left(k_8 \tau_{\mathrm{d}} \right) \tfrac{\mathrm{d}G^{\mathrm{pl}}}{\mathrm{d}t} \right. \end{aligned} \tag{B.42}$$

$$c^{\mathrm{liv}} = \left(k_{12} + k_{14} \right) C^{\mathrm{pl}} \tag{B.43}$$

$$c^{Y} = k_{13} Y \tag{B.44}$$

$$\frac{\mathrm{d}Y}{\mathrm{d}t} = k_{12} C^{\mathrm{pl}} - k_{13} Y \tag{B.45}$$

B.3 Differences in shape factor

The effects of a different shape factor (σ), on the glucose mass in the gut, are shown in Figure B.1. It can be seen that a lower σ, produces a lower more blunt peak. A higher σ produces a higher and more sharp peak.

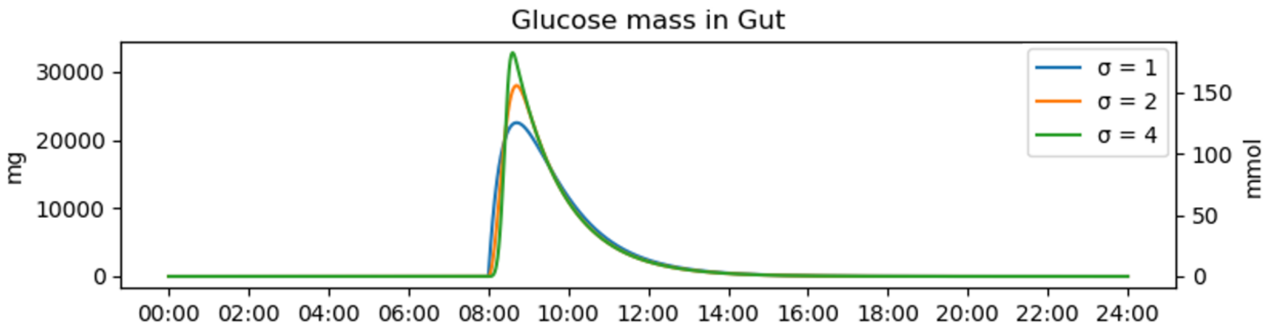


Figure B.1: This plot shows the effect of different σ on the glucose mass in the gut. A single meal is consumed at 8:00, by a standard DM2 patient ($\boldsymbol{\theta}_{DM2}$). Three different values for σ (σ = 1, 2, 4) are used.

Table B.1

Parameter	**Model 1.1**	**units**	**Model 2.0**	units
k_1	Rate constant of glucose appearance in gut (fast absorption)	[1/min]	Rate constant of glucose appearance in the gut	[1/min]
k_2	Rate constant of gastric emptying	[1/min]	Rate constant of gut emptying	[1/min]
k_3	-	-	Rate constant of ΔG suppression of EGP when $G^{pl} > G_b^{pl}$	[1/min]
k_4	Rate constant of insulin-dependent glucose uptake (muscle, liver, adipose tissue)	[1/min]	Rate constant of I^{rem} suppression of EGP (not for patients with diabetes type 1)	[1/min]
k_5	Rate constant of insulin outflow from plasma towards the interstitial fluid	[1/min]	Rate constant of insulin-dependent glucose uptake	[1/min]
k_6	Rate constant of insulin appearance into the interstitial fluid	[1/min]	Rate constant of ΔG-dependent insulin production	[1/min]
k_7	Rate constant of interstitial fluid insulin utilization	[1/min]	Rate constant of G-dependent insulin production	[1/min]
k_8	Rate constant of glomerular filtration	[1/min]	Rate constant of dG/dt-dependant insulin production	[1/min]
k_9	Rate constant of ΔG suppression of endogenous glucose production	[1/min]	Rate constant of insulin outflow from plasma to interstitial fluid	[1/min]
k_{10}	Rate constant of I^{rem} suppression of endogenous glucose production	[1/min]	Rate constant of ΔG suppression of EGP when $G^{pl} < G_b^{pl}$	[1/min]
k_{11}	-	-	Rate constant of glucose appearance in the gut	[1/min]
k_{12}	Rate constant of ΔG-dependent insulin production	[1/min]	Rate constant of c-peptide appearance in secondary compartment (Only used in case of c-peptide data)	[1/min]
k_{13}	Rate constant of G-dependent insulin production	[1/min]	Rate constant of c-peptide removal from secondary compartment (Only used in case of c-peptide data)	[1/min]
k_{14}	Rate constant of dG/dt-dependant insulin production	[1/min]	Rate constant of c-peptide clearance by the liver (Only used in case of c-peptide data)	[1/min]
k_{15}	Rate constant of short-acting insulin appearance in plasma	[1/min]	-	-
σ	Shape factor	[-]	Shape factor	[-]
K_M	Michaelis-Menten constant for glucose uptake by tissues	[mmol/L]	Michaelis-Menten constant for glucose uptake	[mmol/L]
k_{18}	Rate constant of glucose appearance in gut (slow absorption)	[1/min]	This parameter is not used in this version of eDESpy	
k_{19}	Exercise sensitivity	[-]	Exercise sensitivity	[-]

Table B.1: A table containing information about the function of all parameters in the 1.1 and 2.0 version of the eDES model. Depending on which version of eDESpy is used, the function of the parameters, and their units, can change. This table displays the function of the parameters for each version. All parameters and their values can be found in the SQL DBs, *eDESpy_data.db* and *Patient_data.db*.

Table B.2

Constants:	**Description:**	**Model 1.1** Value:	Units:	**Model 2.0** Value:	Units:
D^{meal}	consumed dose of carbohydrates	input	[mg]	input	[mg]
M^b	body Weight	input	[kg]	input	[kg]
G_b^{pl}	basal plasma glucose concentration	input	[mmol/L]	input	[mmol/L]
I_b^{pl}	basal plasma insulin concentration	input	[mU/L]	input	[mU/L]
u^{sa}	insulin infusion rate (after administration of short acting insulin)	input	[mU/min]	input	[mU/min]
U^{sa}	insulin injection dose (short acting insulin administration)	input	[mU]	-	-
U^{la}	insulin injection dose (long acting insulin administration)	input	[mU]	input	[mU]
f	unit conversion factor (inverse molecular weight of glucose)	0.0056	[mmol/mg]	0.005551	[mmol/mg]
v_G	glucose distribution volume in plasma	0.243	[L/kg]	$(17/70) = 0.243$	[L/kg]
β	beta cell sensitivity to glucose	0,5	[(mmol/L)/(mU/L)]	-	-
β	Unit conversion factor from glucose to insulin	-	-	1	[(mmol/L)/(mU/L)]
g_b^{liv}	basal endogenous glucose production	0.0661	[mmol/L/min]	0,043	[mmol/L/min]
Γ_0	glucose conversion factor (non-insulin mediated glucose uptake)	G_b^{pl}	[mmol/L]	-	-
g_{th}^{ren}	renal threshold of glucose	9	[mmol/L]	-	-
τ_i	integral time constant	31	[min]	31	[min]
τ_d	derivative time constant	3	[min]	3	[min]
v_I	insulin distribution volume in plasma	0,186	[L/kg]	$(13/70) = 0.186$	[L/kg]
a	dose shape factor 1, absorption long acting insulin	Levemir: 0.011 Insulatard: 0.018 Lantus: 0.029	[min/U]	Levemir: 0.00288 Lantus: 0.00755	[min/U]
b	dose shape factor 2, absorption long acting insulin	Levemir: 115 Insulatard: 100 Lantus: 250	[min]	Levemir: 0.00561 Lantus: 0.00162	[min]

Continued on next page

Table B.2 – *Continued from previous page*

Constants:	**Description:**	**Model 1.1** Value:	Units:	**Model 2.0** Value:	Units:
h	time characteristic shape factor of long acting insulin absorption	Levemir: 1.85 Insulatard: 1.81 Lantus: 1.73	[-]	Levemir: 1.79 Lantus: 1.14	[-]
$t_{0,5}$	time to absorb 50% of the injected insulin	Differs per insulin brand	[min]	Differs per insulin brand	[min]
k_e	long-acting insulin constant	-	-	Differs per insulin brand	[1/min]
v_c	c-peptide distribution volume in plasma	-	-	418/69,4	[L/kg]
α	unit conversion factor from insulin to c-peptide	-		$6,00 \cdot 10^{-3}$	[(nmol/L)/(mU/L)]
c_1	rate constant of glomerular filtration	-	-	0,1	[1/min]
c_2	rate constant of glucose uptake by the non-insulin dependent tissue	-	-	$2,27 \cdot 10^{-2}$	[1/min]
c_3	rate constant of insulin clearance by the liver	-	-	$1.53 \cdot 10^{-2}$	[1/min]
nr_1	rate constant of short acting insulin appearance in plasma	-	-	0.2483	[1/min]
nr_2	rate constant of short acting insulin appearance in subcutaneous compartment 1	-	-	$8,05 \cdot 10^{2}$	[1/min]
nr_3	rate constant of short acting insulin clearance from the plasma	-	-	0.1719	[1/min]

Table B.2: This table contains all constants used by the different eDES versions. These constants are extracted from the research of van Rozendaal et al. [16] and Maas et al. [67]. Extra information about the origin or deviation of certain can be found in their research.

Table B.3

Variable:	**Description:**	**Units:**
M_G^{gut}	glucose mass in gut	[mg]
G^{pl}	plasma glucose concentration	[mmol/L]
I^{pl}	plasma insulin concentration	[mU/L]
I^{rem}	interstitial insulin concentration	[mU/L]
U_I^{sc1}	insulin mass in subcutaneous compartment near the injection site	[mU]
U_I^{sc2}	insulin mass in subcutaneous compartment proximal to plasma	[mU]
C^{pl}	C-peptide in plasma	[nmol]
C	C-peptide in the secondary compartment c-peptide mass	[nmol]
t	time	[min]
m_G^{gut}	glucose appearance rate from stomach towards gut	[mg]
m_G^{pl}	glucose disappearance rate from gut towards plasma	[mg/min]
g^{liv}	endogenous glucose appearance rate from liver towards plasma	[mmol/L/min]
g^{gut}	exogenous glucose appearance rate from gut towards plasma	[mmol/L/min]
$g^{\text{non-it}}$	glucose disappearance rate from plasma towards non-insulin dependent tissues	[mmol/L/min]
g^{it}	glucose disappearance rate from plasma towards insulin dependent tissues	[mmol/L/min]
g_{th}^{ren}	glucose disappearance rate from plasma towards urine	[mmol/L/min]
i^{pnc}	appearance rate of endogenous insulin from the pancreas towards plasma	[mU/L/min]
i^{sa}	appearance rate of exogenous, short acting insulin from subcutaneous compartment towards plasma	[mU/L/min]
i^{la}	appearance rate of exogenous, long acting insulin from subcutaneous compartment towards plasma	[mU/L/min]
i^{rem}	insulin disappearance rate from plasma towards the interstitium and liver	[mU/L/min]
i^{pl}	insulin appearance rate from plasma towards the interstitium	[mU/L/min]
i^{it}	insulin disappearance rate from interstitium towards insulin dependent tissues	[mU/L/min]

Table B.3: A table with the overview of all variables used by the eDESpy model. The state variables are placed in the top rows of the tables and indicated with a capital letter.

Appendix C

Exercise module options

As is explained in Section 5, glucose utilization is modelled using an extra exercise compartment. However, different approaches which could be useful for future research are discussed next. None of the following discussed methods is currently activated in the model. However, these methods could could be interesting for future research.

Glucose utilization. As explained is Section 5.1, during PE glucose utilization is increased. To enable this increase, a new parameter is created. The newly created parameter ($k_{exc,\,1}$) can impact the insulin dependent glucose utilization (C.1), non insulin dependent glucose utilization (C.2) or both. It is noticed that only influencing the non insulin dependent glucose utilization, provides the most desirable changes in the system. Wahren et al. [95] classified three different intensity levels, this is considered to be leading in the following examples.

$$g^{\mathrm{it}} = k_4 \beta I^{\mathrm{rem}} \frac{G^{\mathrm{pl}}}{K_{\mathrm{M}} + G^{\mathrm{pl}}} k_{exc,1} \tag{C.1}$$

$$g^{\mathrm{non-it}} = c_2 \frac{G^{\mathrm{pl}}}{K_M + G^{\mathrm{pl}}} k_{exc,1} \tag{C.2}$$

The $k_{exc,\,1}$ is created imitating the expected course of glucose increase. Multiple variations have been tried and a simple linear course of the new parameter is discussed. According to Wahren et al. [95], glucose utilization increases 7 fold under low intensity circumstances and 10 till 20 fold under middle to high intensity circumstances. Therefore, the glucose utilization multiplication factor $k_{exc,\,1}$, is created. Additionally, glucose utilization continues after finishing the exercise. This is called the afterburn [15]. In Figure C.1 both effects, of the exercise and the afterburn, are taken into account. The green vertical line corresponds with the ending time of the exercise, whereas, the red line marks the ending time of the afterburn. In the GUI, the exercise and afterburn time, can be easily adjusted. When the exercise is finished, $k_{exc,\,1}$ returns to its original value 1. Resulting in Figure C.1.

```python
# Increased glucose utilization, from Equations_11.py
def Kexc1(p,EL,real_time, iAct, i):
    afterburn = 60
    exc_time = 60

    fact_l = 7
    fact_m = 10
    fact_h = 20

    Kexc1 = 1
    len_iAct = sum(indexAct <= i for indexAct in iAct)
    for k in iAct[:len_iAct]:
        if k <= i:
```

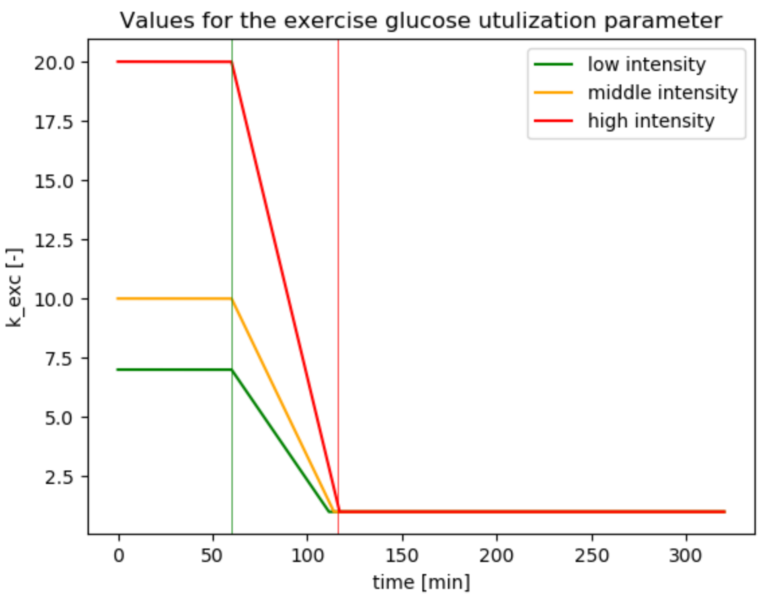


Figure C.1: This figure displays the course of the $k_{exc,1}$ parameter. This parameter is a multiplication factor describing the increase of glucose utilization. Three different intensity levels are constructed [95]. The vertical green line represents the ending time of the exercise, the red line corresponds with the ending time of the afterburn effect. Currently both, the exercise and afterburn effect, are set to 60 minutes. The code in listing C.2, provides the course of this parameter, which can be applied to the model.

```
            if real_time <= (EL[k].end_time) and (real_time - EL[k].start_time) > 0:
                if EL[k].units.lower() == 'low':
                    Kexc1 = fact_l
                if EL[k].units.lower() == 'med':
                    Kexc1 = fact_m
                if EL[k].units.lower() == 'high':
                    Kexc1 = fact_h
            if real_time > (real_time - EL[k].start_time):
                Kexc1 = -(fact_l)/(afterburn)*(real_time - EL[k].start_time)-(fact_l)/(
        afterburn)*-(real_time - EL[k].start_time)+fact_l
                Kexc1 = -(fact_m)/(afterburn)*(real_time - EL[k].start_time)-(fact_m)/(
        afterburn)*-(real_time - EL[k].start_time)+fact_m
                Kexc1 = -(fact_h)/(afterburn)*(real_time - EL[k].start_time)-(fact_h)/(
        afterburn)*-(real_time - EL[k].start_time)+fact_h
            if Kexc1 < 1:
                Kexc1 = 1
        else:
                Kexc1 = 1
    return Kexc1
```

Listing C.1: This method provides the multiplication parameter for the glucose utilization $k_{exc,1}$. When the exercise and afterburn effect is completed the parameter is set to $k_{exc,1} = 1$, returning the glucose utilization to its original state.

Hepatic glucose production. Although the model contains a hepatic glucose production feature, during exercise the dynamics change drastically. Therefore, additional adjustments to the hepatic glucose production could be useful. Modelling of hepatic glucose production consist of glycogenolysis and gluconeogenesis [97]. A parameter ($k_{exc,2}$), is created to describe the increase in hepatic glucose production. In Equation C.3, the original equation describing the hepatic glucose production, is multiplied with the newly created parameter $k_{exc,2}$. As can be seen in Figure C.2, an inverse Gaussian distribution function (Equation C.4) is used to describe this the hepatic glucose production. The first part of the distribution covers the significant glucose increase during glycogenolysis. Whereas over time gluconeogenesis, which is less productive, becomes the main hepatic glucose source. The inverse Gaussian distribution must be scaled accordingly to the literature and the model. Again three different intensity levels are build into the $k_{exc,2}$ parameter.

$$g^{\text{liv}} = g_0^{\text{liv}} - k_9 \left(G^{\text{pl}} - G_0^{\text{pl}} \right) - k_{10} \beta I^{\text{rem}} k_{exc,2} \tag{C.3}$$

$$k_{\text{exc},2} = \sqrt{\frac{\lambda}{2\pi x^3}} \exp \frac{\lambda (x-\mu)^2}{2\mu^2 x} \qquad \text{with } \mu = 150 \text{ and } \lambda = 60 \tag{C.4}$$

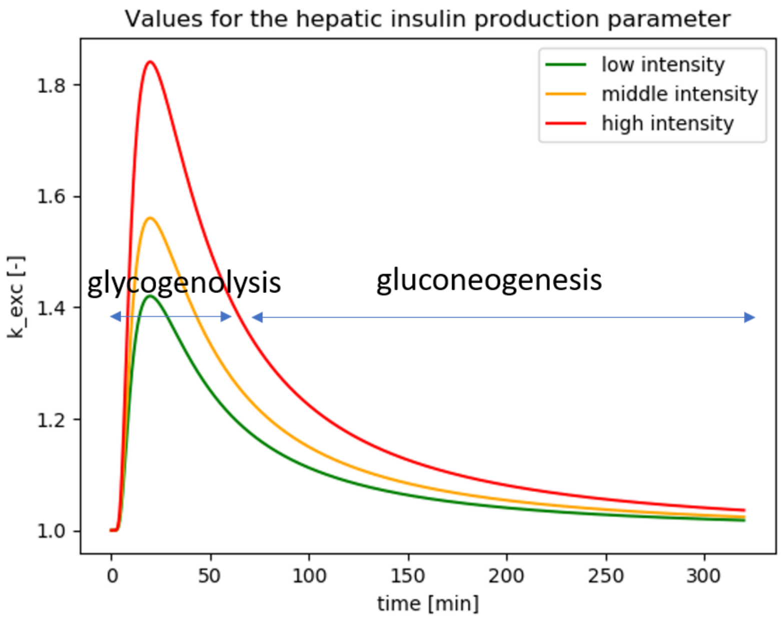


Figure C.2: In this figure is a possible course of the hepatic glucose production displayed. However, the course of this parameter should be fine-tuned according to the model and literature.

```python
# Hepatic glucose production, from Equations_11.py
def Kexc2(p,EL,real_time, iAct, i):

    afterburn = 60
    Lambda = 60
    mu = 150

    Kexc2 = 1
    len_iAct = sum(indexAct <= i for indexAct in iAct)
    for k in iAct[:len_iAct]:
        if k <= i:

            if real_time < (EL[k].end_time + afterburn) and (real_time - EL[k].
    start_time) > 0.00001:
                if EL[k].units.lower() == 'low':
                    Kexc2 = 50*math.sqrt(Lambda/(2*math.pi*(real_time - EL[k].start_time
    )**3))*math.exp(-Lambda*((real_time - EL[k].start_time)-mu)**2/(2*mu**2*(real_time -
     EL[k].start_time)))*0.75+0.9
                elif EL[k].units.lower() == 'med':
                    Kexc2 = 50*math.sqrt(Lambda/(2*math.pi*(real_time - EL[k].start_time
    )**3))*math.exp(-Lambda*((real_time - EL[k].start_time)-mu)**2/(2*mu**2*(real_time -
     EL[k].start_time)))*1+0.9
                elif EL[k].units.lower() == 'high':
                    Kexc2 = 50*math.sqrt(Lambda/(2*math.pi*(real_time - EL[k].start_time
    )**3))*math.exp(-Lambda*((real_time - EL[k].start_time)-mu)**2/(2*mu**2*(real_time -
     EL[k].start_time)))*1.5+0.9

            else:
                Kexc2 = 1
                if Kexc2 <=1:
                        Kexc2 = 1
    return Kexc2
```

Listing C.2: The $k_{exc,\,2}$ is calculated using an inverse Gaussian distribution, as can be seen in Equation C.4 and Figure C.2. $k_{exc,\,2}$ Increases during the exercise and decreases over time. When the effects of the exercise are dimmed out. The parameter is set to $k_{exc,\,2} = 1$, therefore it can no longer affect the system.

Increased β-cell sensitivity. During exercise the β-cell sensitivity increases [90, 99]. Different PE intensities are created. Low, middle and high exercise intensity. According to these exercise levels the exercise has no, medium or significant influence on the β-cell sensitivity. This effect can last for a prolonged period of time. Additionally this factor $k_{exc,\,3}$ could be increased when the fitness level of the subject increases. The newly created parameter impacts the pancreatic insulin production of DM2 patients as can be seen in the following equation:

$$i^{\text{pnc}} = \beta^{-1}\left(k_{12}\left(G^{\text{pl}} - G_0^{\text{pl}}\right) + \frac{k_{13}}{\tau_{\text{i}}}\int\left(G^{\text{pl}} - G_0^{\text{pl}}\right)\mathrm{d}t + k_{14}\tau_{\text{d}}\frac{\mathrm{d}G^{\text{pl}}}{\mathrm{d}t}k_{exc,3}\right) \tag{C.5}$$

Where:

$$\begin{aligned} k_{exc,3,l} &= 1.000, && \text{if intensity} = \text{low} \\ k_{exc,3,m} &= 1.002, && \text{if intensity} = \text{medium} \\ k_{exc,3,h} &= 1.005, && \text{if intensity} = \text{high} \end{aligned}$$

Additional research is necessary to fine-tune these $k_{exc,\,3}$ parameters, currently arbitrary values are chosen. If the parameters are re-estimated, this increase is already incorporated. Patients with DM1 have a pancreatic insulin production close to zero, therefore they are not affected by this addition. The definition responsible for increasing beta cell sensitivity is shown in Listing C.3.

```python
# Increased beta cell sensitivity, from Equations_11.py
def Kexc3(p,EL,real_time, iAct, i):
    Kexc3 = 1
    len_iAct = sum(indexAct <= i for indexAct in iAct)
    for k in iAct[:len_iAct]:
        if k <= i:
            if EL[k].units.lower() == 'low':
                Kexc3 = 1
            if EL[k].units.lower() == 'med':
                Kexc3 = 1.001
            if EL[k].units.lower() == 'high':
                Kexc3 = 1.002
        else:
            Kexc3 = 1

    return Kexc3
```

Listing C.3: The method above adjusts the β-cell sensitivity. If the intensity of the exercise is low, the $k_{exc,\,3}$ will not be affected.

As mentioned, these discussed methods are currently not active in the model. However, if additional extension of PE modules are desired, it could be interesting to review these concepts.

Appendix D

Patient clustering

All parameter estimated parameters values, for all VPs are displayed in a heatmap in logarithmic scale (Figure D.1). All exact values can be found in the SQL database, *eDESpy_data*. The heatmap in Figure D.2, shows the deviance from the mean parameter value.

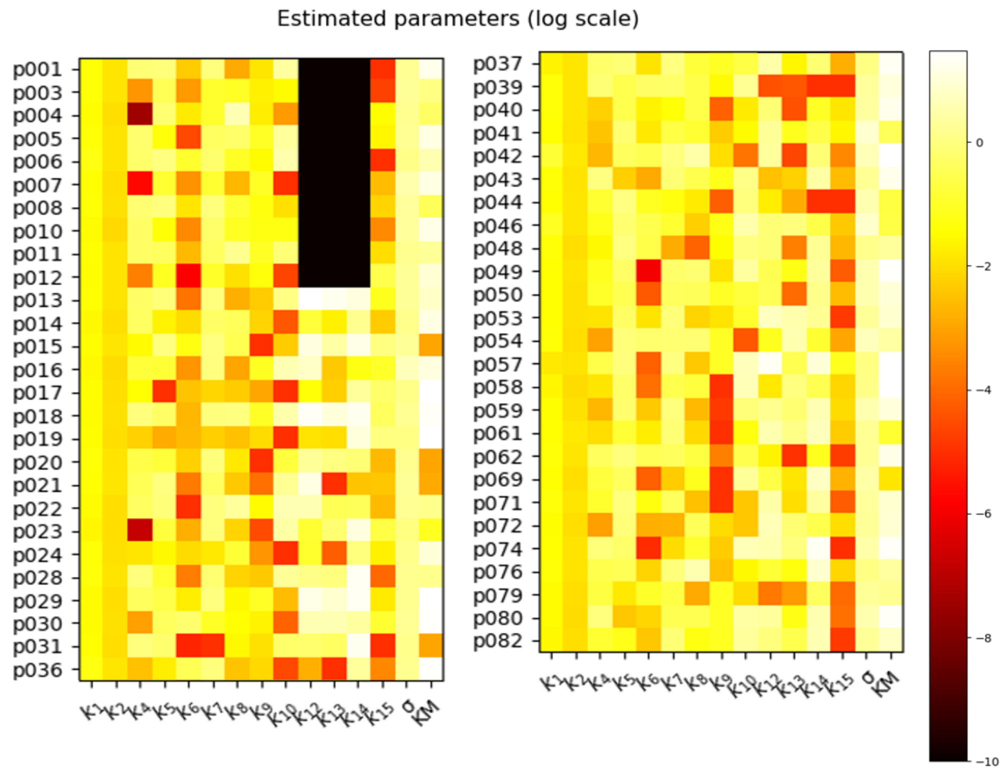


Figure D.1: This heat map displays the estimated parameters of all patients ($\hat{\boldsymbol{\theta}}_{p001,\, p002,\, ..\, ,\, p082}$), using a logarithmic scale.

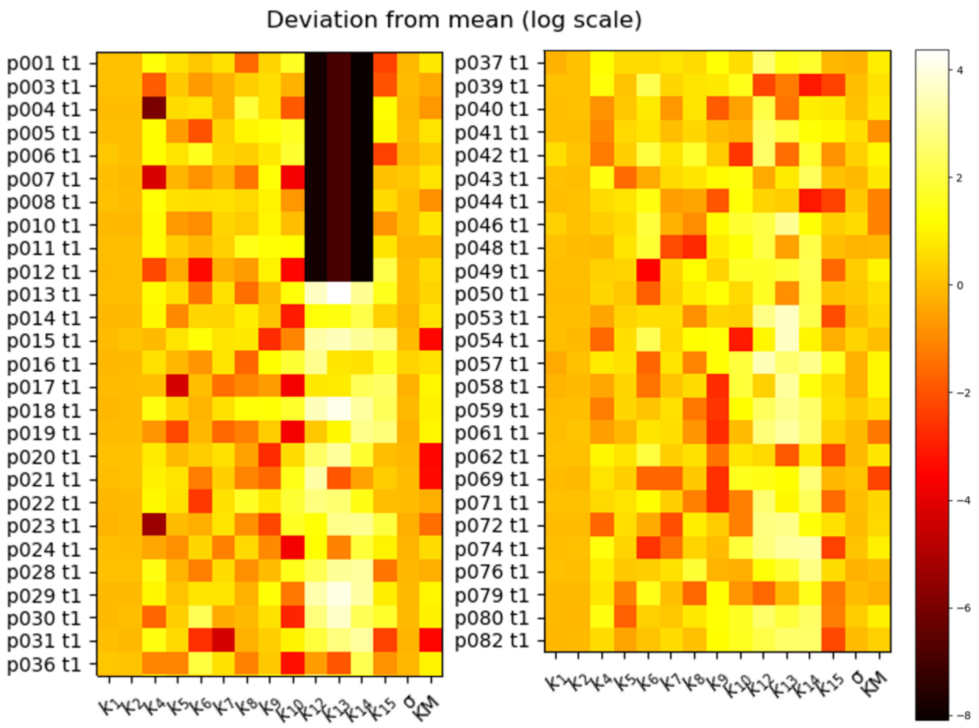


Figure D.2: In this heatmap, the deviation of all parameters (in logarithmic scale), towards the mean value of that certain parameter, is displayed.

Bibliography

[1] WHO Library Cataloguing. Global Report on Diabetes. *Isbn*, 978:6–86, 2016. 1, 4, 7

[2] Statista. Global Overview Diabetes. 2016. 1

[3] Xiyue Jing, Jiageng Chen, Yanan Dong, Duolan Han, Haozuo Zhao, Xuying Wang, Fei Gao, Changping Li, Zhuang Cui, Yuanyuan Liu, and Jun Ma. Related factors of quality of life of type 2 diabetes patients : a systematic review and meta-analysis. pages 1–14, 2018. 1

[4] World J Diabetes. World Journal of. 9358(4), 2017. 1

[5] Ryan Lee, Tien Y Wong, and Charumathi Sabanayagam. Epidemiology of diabetic retinopathy , diabetic macular edema and related vision loss. *Eye and Vision*, pages 1–25, 2015. 1, 5

[6] U S. Economic Costs of Diabetes in the. 41(May):917–928, 2018. 1

[7] Jeremy M Berg, John L Tymoczko, and Lubert Stryder. *Biochemistry seventh edition.* 2012. 1, 2, 3, 4, 64

[8] Kimie Date, Ayano Satoh, Kaoruko Iida, and Haruko Ogawa. Pancreatic ␣ -Amylase Controls Glucose Assimilation by Duodenal Retrieval through N -Glycan-specific Binding , Endocytosis , and Degradation *. 290(28):17439–17450, 2015. 1

[9] Walter F Boron and Emile L Boulpaep. Medical Physiology: a Cellular and Molecular Approach. pages 1074–1093, 2012. 1, 2, 3, 5

[10] Judith E. Fradkin and Griffin P. Rodgers. Diabetes research: A perspective from the National Institute of diabetes and digestive and kidney diseases. *Diabetes*, 62(2):320–326, 2013. 2, 5

[11] Pia V Röder, Bingbing Wu, Yixian Liu, and Weiping Han. Pancreatic regulation of glucose homeostasis. (November 2015), 2016. 2

[12] Authors Irl B Hirsch and Michael Emmett. Diabetic ketoacidosis and hyperosmolar hyperglycemic state in adults : Epidemiology and pathogenesis. pages 1–23, 2020. 2

[13] Suresh R Naik and Ganesh R Kokil. *Development and Discovery Avenues in Bioactive Natural Products for Glycemic Novel Therapeutics*, volume 39. Copyright &copy; 2013 Elsevier B.V. All rights reserved., 1 edition, 2013. 2

[14] Bruce Alberts, Alexander Johnson, and Julian Lewis. *Molecular biology of the cell.* 5 edition, 2008. 2, 3

[15] Scott K Powers and Howley T Edward. *Exercise Physiology Theory and Application to Fitness and Performance.* McGraw-Hill Education, 2015. 2, 50, 51, 52, 55, 62, 65, 75

[16] Yvonne van Rozendaal. Creating the core for the Eindhoven Diabetes Education Simulator : Model analysis , parameterization and extension for. (June), 2013. 3, 8, 10, 12, 42, 43, 44, 63, 67, 73

[17] María M Adeva-andany, Manuel González-lucán, Cristóbal Donapetry-garcía, Carlos Fernández-fernández, and Eva Ameneiros-rodríguez. Glycogen metabolism in humans , . *BBACLI*, 5:85–100, 2016. 3

[18] Emma Leighton, Christopher Ar, and Sainsbury Gregory. A Practical Review of C-Peptide Testing in Diabetes what is c-peptide and why might it be useful in clinical problems with c-peptide. *Diabetes Therapy*, 8(3):475–487, 2017. 3, 6, 61

[19] Pertti Ebeling, A. Koistinen Heikki, and Koivisto Veikko A. Insulin-independent glucose transport regulates insulin sensitivity. *Helsinki University Central Hospital*, 1994. 3

[20] Joseph A Knight. Knight - 2012 - Review Physical Inactivity Associated Diseases and Disorders. *Annals of clinical and laboratory science*, 42(3):320–337, 2012. 4

[21] Andréa Deslandes, Helena Moraes, and Camilia Ferreira. Exercise and mental health: many reasons to move. *Neuropsychobiology*, page 191-198, 2009. 4

[22] Richter Erik A., Thorkil. Ploug, and Henrik. Galbo. Increased Muscle Glucose Uptake After Exercise: No Need for Insulin During Exercise. *American Diabetes Association*, pages 1041–1048, 1985. 4

[23] Elaine N Marieb and Katja N Hoehn. *Human Anatomy Physiology*. Pearson Education Limited, 2014. 4

[24] Erik A Richter, Wim Derave, and Jørgen F P Wojtaszewski. Topical Review Glucose , exercise and insulin : emerging concepts. *Physiology*, pages 313–322, 2001. 4, 51

[25] LB Borghouts and HA Keizer. Exercise and insulin sensitivity: a review. *Int J Sports Med*, pages 1–12, 2009. 4

[26] Jean Weininger. Diabetes mellitus and metabolic disorders. pages 917–928, 2019. 4

[27] Centers For Disease Control and Pervention. National Diabetes Statistics Report. 2017. 4

[28] Akram T Kharroubi and Hisham M Darwish. Diabetes mellitus : The epidemic of the century. 6(6):850–867, 2015. 4, 5

[29] Richard I.G. Hold, Clive S. Cockram, and Barry J Goldstein. *Textbook of Diabetes*. 2010. 4, 5, 6, 7, 48

[30] RG Naik, BM Brooks-Worrell, and JP Palmer. Latent autoimmune diabetes in adults. *J Clin Endocrinol Metab*, pages 4635–4644, 2009. 5, 48

[31] IIDF. *IDF Diabetes Atlas Ninth edition 2019*. 2019. 5

[32] World Journal of Diabetes. *Prediabetes diagnosis and treatment*. 2015. 5

[33] Amanda Mather and Carol Pollock. Glucose handling by the kidney. *Kidney International*, 79:S1–S6, 2011. 5

[34] Klaus Rave, Leszek Nosek, John Posner, Tim Heise, and Kerstin Roggen. Renal glucose excretion as a function of blood glucose concentration in subjects with type 2 diabetes — results of a hyperglycaemic glucose clamp study. (April):2166–2171, 2006. 5

[35] Polly E. Parsons and Jeanine P. Wiener-Kronish. *Critical Care Secrets: Fifth Edition*. Elsevier Inc., 2013. 5

[36] Dove Press. Diabetic nephropathy – complications and treatment. pages 361–381, 2014. 5

[37] Zhang C, Hein TW, and Wang W. Constitutive expression of arginase in microvascular endothelial cells counteracts nitric oxide-mediated vasodilatory function. *FASEB J*, 2001. 5

[38] Claudia Hammi and Brent Yeung. Neuropathy. *StatPearls Publishing*, 2020. 5

[39] Juliana Casqueiro, Janine Casqueiro, and Cersio Alves. Infections in patients with diabetes mellitus: A review of pathogenesis. *Indian Journal of Endocrinology and Metabolism*, 2014. 5

[40] Sharad P Pendsey. Understanding diabetic foot. *International Journal of Diabetes in Developing Countries*, pages 75–79, 2010. 5

[41] Thomas R Einarson, Annabel Acs, Craig Ludwig, and Ulrik H Panton. Prevalence of cardiovascular disease in type 2 diabetes : a systematic literature review of scientific evidence from across the world in 2007 – 2017. *Cardiovascular Diabetology*, pages 1–19, 2018. 5

[42] A. A. Kazi and L. Blonde. *Classification of diabetes mellitus*, volume 21. 2001. 6

[43] Wolfgang Rathmann and Esther Jacobs. Screening for type 2 diabetes. *Diabetologe*, 16(1):87–96, 2020. 6

[44] C. Matthew Riddle. The journal of clinical and applied research and education, diabetes care, 2019. 6, 7, 64

[45] Shariq I Sherwani, Haseeb A Khan, Aishah Ekhzaimy, Afshan Masood, and Meena K Sakharkar. Significance of HbA1c Test in Diagnosis and Prognosis of Diabetic Patients. pages 95–104, 2016. 6

[46] A. G. Jones and A. T. Hattersley. The clinical utility of C-peptide measurement in the care of patients with diabetes. *Diabetic Medicine*, 30(7):803–817, 2013. 6, 8

[47] Bahendeka Silver, Kaushik Ramaiya, Swai Babu Andrew, Otieno Fredrick, Sarita Bajaj, Sanjay Kalra, Bavuma M Charlotte, Karigire Claudine, and Anthony Makhoba. EADSG Guidelines : Insulin Therapy in Diabetes EXECUTIVE SUMMARY AND. *Diabetes Therapy*, 9(2):449–492, 2018. 7

[48] Julie B Sneddon, Qizhi Tang, Peter Stock, Jeffrey A Bluestone, Shuvo Roy, and Tejal Desai. Perspective Stem Cell Therapies for Treating Diabetes : Progress and Remaining Challenges. *Stem Cell*, 22(6):810–823, 2018. 7

[49] P Gillard, B Keymeulen, and C Mathieu. Beta-cell transplantation in type 1 diabetic patients: a work in progress to cure. *Verhandelingen - Koninklijke Academie voor Geneeskunde van België*, 72:71–98, 01 2010. 7

[50] Dove Press. Metformin : a review of its potential indications. pages 2421–2429, 2017. 7, 64

[51] Graham Rena, D Grahame Hardie, and Ewan R Pearson. The mechanisms of action of metformin. pages 1577–1585, 2017. 7, 64

[52] Karla I Galaviz, K M Venkat Narayan, Felipe Lobelo, and Mary Beth Weber. Lifestyle and the Prevention of Type 2 Diabetes : A Status Report. (1518), 2018. 7

[53] P Hemachandra Reddy, United States, South West Campus, United States, United States, United States, United States, United States, and United States. HHS Public Access. 1(4):1–9, 2018. 7

[54] Anna Maas. *Playing with numbers.* Eindhoven, Technische Universiteit, 2017. 7, 8

[55] Marco Viceconti, Adriano Henney, and Edwin Morley-Fletcher. In silico clinical trials: how computer simulation will transform the biomedical industry. *International Journal of Clinical Trials*, 3(2):37, 2016. 7

[56] Tina M. Morrison, Pras Pathmanathan, Mariam Adwan, and Edward Margerrison. Advancing regulatory science with computational modeling for medical devices at the fda's office of science and engineering laboratories. *Frontiers in Medicine*, 5:241, 2018. 7, 46

[57] E. Salzsieder, L. Vogt, K. D. Kohnert, P. Heinke, and P. Augstein. Model-based Decision support in Diabetes Care. *Computer Methods and Programs in Biomedicine*, 102(2):206–218, 2011. 7, 46

[58] Chiara Dalla Man, Davide M Raimondo, Robert A Rizza, and Claudio Cobelli. {{GIM}}, simulation software of meal glucose{\textemdash}insulin model. *Journal of diabetes science and technology*, 1(3):323–330, 2007. 8, 10, 12

[59] Anne H. Maas, Yvonne J.W. Rozendaal, Carola Van Pul, Peter A.J. Hilbers, Ward J. Cottaar, Harm R. Haak, and Natal A.W. Van Riel. A physiology-based model describing heterogeneity in glucose metabolism: The core of the eindhoven diabetes education simulator (E-DES). *Journal of Diabetes Science and Technology*, 9(2):282–292, 2015. 8

[60] Janet D Elashoff, Terry J Reedy, and James H Meyer. Analysis of Gastric Emptying Data. *Gastroenterology*, 83(6):1306–1312, 1982. 11, 42, 52, 55, 62, 63

[61] I. Gottesman, L. Mandarino, C. Verdonk, R. Rizza, and J. Gerich. Insulin increases the maximum velocity for glucose uptake without altering the Michaelis constant in man. Evidence that insulin increases glucose uptake merely by providing additional transport sites. *Journal of Clinical Investigation*, 70(6):1310–1314, 1982. 12

[62] E. D. Lehmann and T. Deutsch. A physiological model of glucose-insulin interaction in type 1 diabetes mellitus. *Journal of Biomedical Engineering*, 14(3):235–242, 1992. 12

[63] G M Steil, A E Panteleon, and K Rebrin. Closed-loop insulin delivery — the path to physiological glucose control. 56:125–144, 2004. 12

[64] Werner Regittnig, Stefan Lindpointner, Stefan Korsatko, Dina Tutkur, Manfred Bodenlenz, and Thomas R. Pieber. Periodic extraction of interstitial fluid from the site of subcutaneous insulin infusion for the measurement of glucose: A novel single-port technique for the treatment of type 1 diabetes patients. *Diabetes Technology and Therapeutics*, 15(1):50–59, 2013. 13

[65] S Shimoda, K Nishida, and M Sakakida. Closed-loop subcutaneous insulin infusion algorithm with a short-acting insulin analog for long-term clinical application of a wearable artificial endocrine pancreas. *Front Med Biol Eng*, page 197-211, 1997. 14

[66] M Berger and D Rodbard. Computer simulation of plasma insulin and glucose dynamics after subcutaneous insulin injection. *Diabetes Care.*, page 725-736, 1989. 14

[67] Technische Universiteit Eindhoven. *Playing with numbers.* Number 2017. 2019. 15, 16, 40, 43, 63, 64, 67, 73

[68] van E Cauter, F Mestrez, and J Struis. Estimation of insulin secretion rates from C-peptide levels. Comparison of individual and standard kinetic parameters for C-peptide clearance. *Diabetes*, page 368-377, 1992. 16

[69] Dusty Phillips. *Python 3 Object Oriented Programming.* Packt Publishing, 2010. 20

[70] Guido van Rossum and Fred L Drake Jr. *Python tutorial.* Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995. 21, 61

[71] Pierre Raybout. *Spyder (from Anaconda Navigator).* Massachusetts Institute of Technology, 2009. 21

[72] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006. 21

[73] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed]. 21, 41

[74] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95, 2007. 21

[75] Mike Owens. *The definitive guide to SQLite.* Apress, 2006. 27, 31

[76] Richard D Hipp. Sqlite, 2020. 27, 31

[77] The Qt Company. Qt designer. 2020. 32

[78] Eve Maler, Jean Paoli, C. M. Sperberg-McQueen, François Yergeau, and Tim Bray. Extensible markup language (XML) 1.0 (third edition). first edition of a recommendation, W3C, February 2004. 32

[79] PyQT. Pyqt reference guide. 2012. 32

[80] Gabriele Lillacci and Mustafa Khammash. Parameter estimation and model selection in computational biology. *PLoS Computational Biology*, 6(3), 2010. 40

[81] Simeone Marino, Ian B Hogue, Christian J Ray, and Denise E Kirschner. *A methodology for perfoming gloabl*, volume 254. 2009. 41, 42

[82] JA Snyman. *Practical mathematical optimization: An introduction to basic optimization theory and classical and new gradient-based algorithms.* Springer Publishing, 2005. 41

[83] Attila Gábor and Julio R. Banga. Robust and efficient parameter estimation in dynamic models of biological systems. *BMC Systems Biology*, 9(1), 2015. 42

[84] M. D. McKay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 42(1):55–61, 2000. 42

[85] Mohamed Amine Bouhlel, John T. Hwang, Nathalie Bartoli, Rémi Lafage, Joseph Morlier, and Joaquim R. R. A. Martins. A python surrogate modeling framework with derivatives. *Advances in Engineering Software*, page 102662, 2019. 42

[86] RJ Allen, TR Rieger, and CJ Musante. Efficient generation and selection of virtual populations in quantitative systems pharmacology models. *CPT: Pharmacometrics & Systems Pharmacology*, 5(3):140–146, 2016. 46

[87] David Banks, Leanna House, Frederick R. McMorris, Phipps Arabie, and Wolfgang Gaul. *Classification, Clustering, and Data Mining Applications: Proceedings of the Meeting of the International Federation of Classification Societies (IFCS), ... Data Analysis, and Knowledge Organization).* Springer-Verlag, Berlin, Heidelberg, 2004. 46, 47

[88] Chunhui Yuan and Haitao Yang. Research on K-Value Selection Method of K-Means Clustering Algorithm. *J*, 2(2):226–235, 2019. 46, 47

[89] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 46

[90] M. Derouich and A. Boutayeb. The effect of physical exercise on the dynamics of glucose and insulin. *Journal of Biomechanics*, 35(7):911–917, 2002. 50, 51, 78

[91] Maria Concetta Palumbo, Micaela Morettini, Paolo Tieri, Fasma Diele, Massimo Sacchetti, and Filippo Castiglione. Personalizing physical exercise in a computational model of fuel homeostasis. *PLoS Computational Biology*, 14(4):1–23, 2018. 50, 62

[92] Marc D. Breton. Physical activity-the major unaccounted impediment to closed loop control. *Journal of Diabetes Science and Technology*, 2(1):169–174, 2008. 50

[93] O. Peter Adams. The impact of brief high-intensity exercise on blood glucose levels. *Diabetes, Metabolic Syndrome and Obesity: Targets and Therapy*, 6:113–122, 2013. 50

[94] Jaeyeon Kim, Gerald M. Saidel, and Marco E. Cabrera. Multi-scale computational model of fuel homeostasis during exercise: Effect of hormonal control. *Annals of Biomedical Engineering*, 35(1):69–90, 2007. 50

[95] J. Wahren, P. Felig, G. Ahlborg, and L. Jorfeldt. Glucose metabolism during leg exercise in man. *The Journal of clinical investigation*, 50(12):2715–2725, 1971. 51, 55, 75, 76

[96] Sahiin Kent. Muscle Glucose Metabolism during Exercise. *Annals of Medicine*, 22:185–189, 2009. 51

[97] Anirban Roy and Robert S. Parker. Mathematical Models of the Metabolic System in Health and in Diabetes: Dynamic Modeling of Exercise Effects on Plasma Glucose and Insulin Levels. *Journal of diabetes science and technology (Online)*, 1(3):338, 2007. 51, 76

[98] Physical activity into the meal glucose-insulin model of type 1 diabetes: In silico studies. *Journal of Diabetes Science and Technology*, 3(1):56–67, 2009. 51

[99] Kuang-Chung Shih and Ching-Fai Kwok. Exercise reduces body fat and improves insulin sensitivity and pancreatic β-cell function in overweight and obese male Taiwanese adolescents. *BMC Pediatrics*, 2018. 51, 78

[100] R.F. Schmidt, F. Lang, and M. Heckmann. *Physiologie des Menschen: mit Pathophysiologie.* Springer-Lehrbuch. Springer Berlin Heidelberg, 2007. 52

[101] Elin Nyman. Hierarchical modeling of diabetes, Department of Clinical and Experimental Medicine. 2009. 64

[102] RS Surwit, MS Schneider, and MN Feinglos. Stress and diabetes mellitus. *Diabetes Care*, 1992. 64

[103] Hilda Wong, Jaya Singh, Ryan M Go, Nancy Ahluwalia, and Michelle A Guerrero-Go. The Effects of Mental Stress on Non-insulin-dependent Diabetes: Determining the Relationship Between Catecholamine and Adrenergic Signals from Stress, Anxiety, and Depression on the Physiological Changes in the Pancreatic Hormone Secretion. *Cureus*, pages 1–8, 2019. 64

[104] Mesut Çiçek. Wearable technologies and its future applications. *International Journal of Electrical, Electronics and Data Communication*, 3:2320–2084, 05 2015. 65

[105] Xing Wei Wu, Heng Bo Yang, Rong Yuan, En Wu Long, and Rong Sheng Tong. Predictive models of medication non-adherence risks of patients with T2D based on multiple machine learning algorithms. *BMJ Open Diabetes Research and Care*, 8(1):1–11, 2020. 65