Artificial intelligence is the science of making machines do things that would require intelligence if done by men.

– Marvin Minsky, 1968.

University of Alberta

MODEL-FREE INTELLIGENT DIABETES MANAGEMENT USING MACHINE LEARNING

by

Meysam Bastani

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

Spring 2014
Edmonton, Alberta

To my parents
for their love and support

Abstract

Each patient with Type-1 diabetes must decide how much insulin to inject before each meal to maintain an acceptable level of blood glucose. The actual injection dose is based on a formula that takes current blood glucose level and the meal size into consideration. While following this insulin regimen, the patient records their insulin injections, blood glucose readings, meal sizes and potentially other information in a diabetes diary. During clinical visits, the diabetologist analyzes these records to decide how best to adjust the patient's insulin formula.

This research provides methods from supervised learning and reinforcement learning that automatically adjust this formula using data from a patient's diabetes diary. These methods are evaluated on twenty *in-silico* patients, achieving a performance that is often comparable to that of an expert diabetologist. Our experimental results demonstrate that both supervised learning and reinforcement learning methods appear effective in helping to manage diabetes.

Acknowledgements

The completion of this dissertation could not have been possible without the guidance, support and encouragement of my supervisor, my friends, and my beloved family. I wish to express my sincere appreciation to Russell Greiner, my supervisor, for his wisdom, patience, and advice throughout my research. I am very grateful for all I have learned from him.

I would like to thank Edmond A. Ryan, for his endless help in my quest to understand the biology of diabetes, and for agreeing to help us as our expert diabetologist throughout this research. Many thanks to the other members of my committee, David Wishart and Michael Bowling for their constructive feedback.

I have benefited from the insight and advice of several other professors. Specifically, I owe Richard S. Sutton a big thank you for helping and encouraging me to learn and use reinforcement learning from the beginning of my research. Many thanks to Susan A. Murphy for her helpful ideas, our numerous discussions on actor-critic methods during her visit to UofA, as well as sharing her algorithm with me.

I thank the students and staff at the AICML. In particular, credit goes to Leslie Acker, Bret Hoehn, and Nasimeh Asgarian for creating an enjoyable environment for work and research. I would also like to thank many of the RLAI lab members for their productive discussions whether it was during the tea-time talks or ACPG meetings or in Patrick Pilarski's or Joseph Modayil's office. My special thanks to Katherine Chen for her time and helpful feedback on my dissertation.

I am thankful for the support of my dear friends Kasra Nikooyeh and Shima Khatibisepehr whom I consider family. I owe the same gratitude to my CS friends who made my stay in Edmonton a unique and delightful experience: Yavar Naddaf, Babak Damavandi, Saman Vaisipour, Maysam Heydari, Siamak Ravanbakhsh, Reihaneh Rabbani, Arash Afkanpour, Sheehan Khan, and many other great friends.

I am indebted to my parents for their never-ending support and dedication, this thesis is dedicated to them. My eternal gratitude goes to my wonderful wife, Fatemeh Miri Disfany, for her unconditional love and support. She has always been there for me through thick and thin. She has earned this as much as I have.

This research has been financially supported by the Alberta Ingenuity Centre for Machine Learning (AICML).

Table of Contents

List of Tables

List of Figures

List of Symbols

Other		
$\mathbb{N}$	natural numbers: $\{1, 2, \cdots\}$	
$\mathbb{N}_0$	natural numbers with zero: $\{0, 1, 2, \cdots\}$	
$\mathbb{R}$	real numbers	
$\varphi \in \mathbb{R}^{(n_\varphi)}$	vector over n_φ features	
$\theta \in \mathbb{R}^{(n_\theta)}$	policy parameters	
$W \in \mathbb{R}^{(n_W)}, \omega \in \mathbb{R}^{(n_\omega)}$	weight vectors	
$\sim$	drawn from distribution	
$\approx$	approximation	
$[A, B]$	column-wise concatenation of two matrices: $A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{m \times n'}$	
$[A; B]$	row-wise concatenation of two matrices: $A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{m' \times n}$	
$\hat{e}_i$	i-th unit vector in the Cartesian coordinate system	
$U(a, b)$	uniform distribution on $[a, b]$	
$N(\mu, \sigma)$	Gaussian distribution	
$N(\sigma)$	Gaussian distribution with $\mu = 0$	
$TN(\mu, \sigma, a, b)$	truncated Gaussian distribution	Section 4.1.5
$\theta^\dagger$	T1DM optimal policy parameters	Section 4.2
${}^p\theta^\dagger$	T1DM optimal policy parameters for patient p	Section 4.2
MDP		
$\mathcal{S}$:	state space	Section 2.2.3
$\mathcal{A}$:	action space	Section 2.2.3
$\mathcal{P}$: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$	transition probability matrix	Section 2.2.3
$\mathcal{R}$	reward probability matrix	Section 2.2.3
$t \in \mathbb{N}$	time-step	Section 2.2.3
$\tau_t \in \mathbb{R}$	the actual time at time-step t	Section 2.2.5
$\tilde{R}$	mean-reward	Section 2.2.5
R^+	accumulative-reward	Section 2.2.5
π	policy	Section 2.2.6
J	reinforcement learning objective function	Section 2.2.9
V^π	value function of policy π	Section 2.2.10
V^*	optimal value function	Section 2.2.10
Q^π	action value function of policy π	Section 2.2.10
Q^*	optimal action value function	Section 2.2.10
$\pi^Q_{(greedy)}$	greedy policy based on Q	Section 2.2.11
$\gamma \in [0, 1]$	discount factor	Section 4.3.1
$\alpha \in [0, 1]$	step-size/learning-rate	Section 4.3.1

Chapter 1

Introduction

1.1 Motivation

Diabetes mellitus is increasingly responsible for significant levels of mortality and health complications. According to estimates from the International Diabetes Federation (IDF), as of 2012, more than 371 million people worldwide have diabetes [37] with 50% of this population still remaining undiagnosed. A prevalence map of diabetes as well as more detailed information about the percentage of undiagnosed cases within the population can be seen in Figure 1.1. The prevalence rate of diabetes has been rising rapidly in the last decade, especially in North America and Europe. This is replicated in Canada for the years 2001 to 2007 – see Figure 1.2.

This chronic disease imposes a large economic burden on healthcare systems, especially in North American countries. In 2010, 11.6% of the worldwide healthcare expenditure (about $376 billion) was spent to prevent and treat diabetes and its complications; about half of these costs occurred in North America [84]. In addition to these direct costs, diabetes also imposes significant indirect social costs to the economy due to earnings losses, lost work days, restricted activity days, lower productivity, etc. The American Diabetes Association estimates that in 2007, the US economy lost around $58 billion in indirect costs (about half of the direct health care expenditure on diabetes during the same year) [84].

Diabetes is defined in simple terms as a disease that causes a high blood glucose level in patients. Currently there is no evident cure for this disease [33]. Treatment consists of regulating a patient's blood glucose level to stay within a specific range. There are two major sub-types of diabetes mellitus: *type-1* and *type-2*. This dissertation focuses on type-1 diabetes treatment which affect about 10 percent of the diabetic population. According to Tao et al. [71], about 25 percent of the total diabetic costs is related to indirect costs of the type-1 diabetic population; this share for direct medical costs is about 7 percent[1].

In order to keep their blood glucose level in an acceptable range, type-1 diabetic patients must inject insulin several times during a day. The amount of each injection is specified by a formula (or

[1] Early mortality costs are ignored in these estimates.

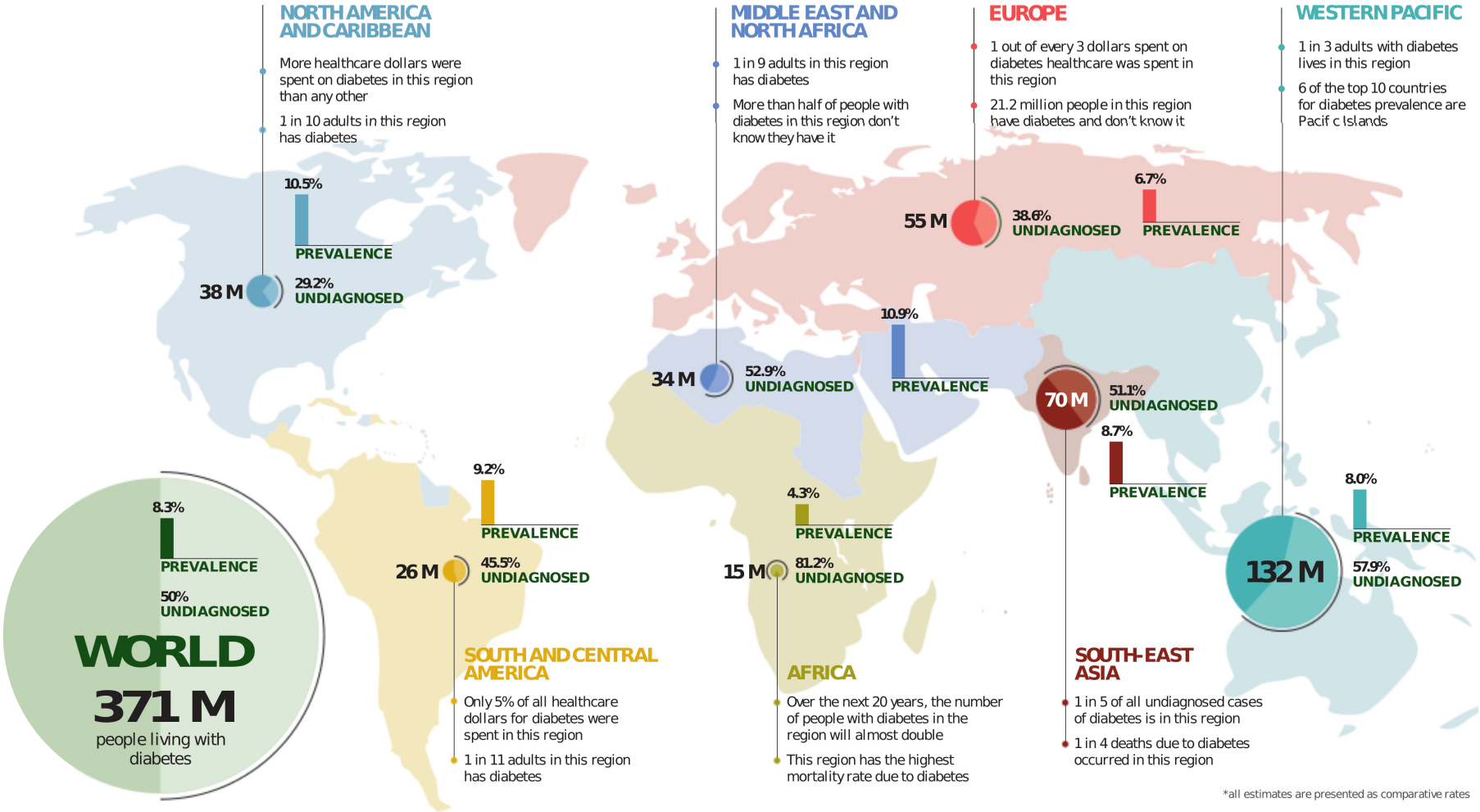


Figure 1.1: The prevalence map of diabetes in 2012; Reprinted from Diabetes Atlas 2012 [37].

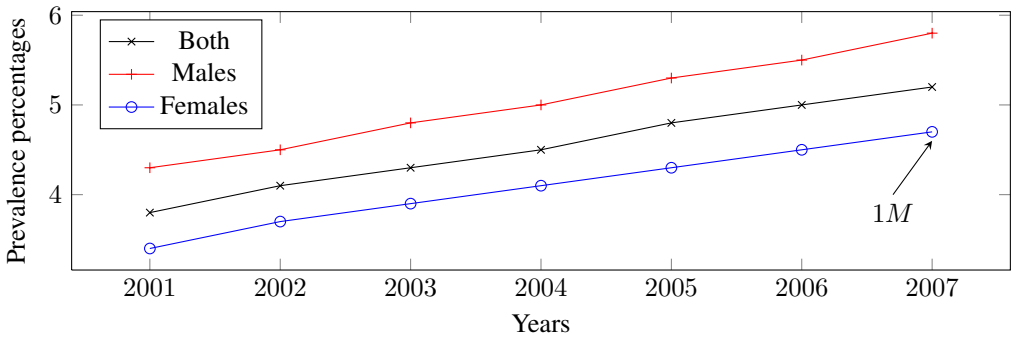


Figure 1.2: Age-standardized prevalence percentages of diagnosed diabetes among people aged one year and older, by sex, Canada, 2000-2001 to 2006-2007. Age-standardized to 1991 Canadian population [30]. The population of females diagnosed diabetes in 2007 is about one million (indicated by arrow).

a table), which we call a *(treatment) policy*. This policy is prescribed by the patient's diabetologist[2], is specific to each patient, and needs to be adjusted regularly. Patients are instructed how to make policy changes themselves but many lack confidence to make the changes. Accordingly, patients are encouraged to visit their diabetologist every 3-6 months [3], which corresponds to 2-4 policy updates per year. Such frequent policy adjustments are only possible if there is a sufficient number of diabetologists available –*e.g.,* in the developed world, where there are on average more than 2 physicians per 1000 people [77]. In emerging nations (*e.g.,* countries with fewer than one physician per 1000 individuals) the treatment policy updates would be far less frequent – perhaps 1 per 2 years or in some cases never!

Automation of the policy adjustment process would help patients manage their diabetes without frequent in-person diabetologist supervision. Patients might be able to get the same level of care (or better) and the diabetologists may be able to manage larger patient populations. Policy adjustment automation could be even more beneficial in rural areas and emerging nations. This automation system could be implemented on a cell-phone, personal computer, or a dedicated device that suggests a policy adjustment to the patient. The system might also have the ability to warn the patient's diabetologist or caregiver that the patient needs medical attention.

Contributions

The main contributions of this dissertation are as follows:

- Formalizing the diabetes treatment policy adjustment problem.
- Showing that, given access to the optimal standard treatment-policies for some patients, a supervised learning method can produce a regressor that can effectively control the blood glucose of other patients.
- Showing that a reinforcement learning method can effectively control the blood glucose of a

[2]In this dissertation, the term diabetologist is taking to include the medical doctor, the nurses, the dietitians and the general diabetes care team. The diabetes care team essentially includes the patient; however, in this dissertation by diabetologist we mainly refer to the external resources and not the patient.

patient.

Here, a method is considered effective if it can control the blood glucose level with an acceptable performance. To assess the effectiveness of blood glucose control, we define a score function in Section 1.3.6.

After formalizing the policy adjustment problem, we employed the following three methods. One method uses a supervised learning approach to automate policy adjustments based on population data. The other two methods use reinforcement learning algorithms to dynamically tailor a treatment policy to individual patients. Our experimental results, on twenty simulated patients, show that some of these methods are as effective as an expert diabetologist in controlling a patients' blood glucose levels.

1.2 Outline

The remaining sections of this chapter offer background information on diabetes (Section 1.3) and discuss the existing literature on automated diabetes management (Section 1.4). Chapter 2 introduces the foundations of the machine learning technologies that are used in our methods. Chapter 3 explains a type-1 diabetes simulator platform that we used to evaluate our systems. Our formulation of the diabetes blood glucose control problem, our supervised policy adjustment algorithm, as well as two reinforcement learning policy adjustment algorithms are presented in Chapter 4. Chapter 5 presents the experimental results obtained from each of the algorithms on the simulated patients. Conclusions from the study, and a number of possible future research directions, are presented in Chapter 6. Appendix A provides additional information about each *in-silico* patient. Additional experimental results are provided in Appendix B and Appendix C.

1.3 Biological background

In this section we provide a brief history of diabetes, describe how diabetes affects blood glucose levels, and introduce current strategies for diabetes treatment. Readers who are interested in the history of diabetes will find a brief discussion below (Section 1.3.1); others may skip directly to Section 1.3.2.

1.3.1 History of diabetes

Polyuric diseases have been described for over 3500 years. Aretaeus gave the first description of what we now know as type-1 diabetes in the second century A.D.: "Diabetes is a dreadful affliction, not very frequent among men, being a melting down of the flesh and limbs into urine". It is clear from his description why he called the disease "diabetes", the Greek word for a syphon [72].

Avicenna (980–1037), a Persian physician and philosopher, provided a comprehensive list of the symptoms for diabetes in one of his books, *The Canon of Medicine*. He pointed out that diabetic

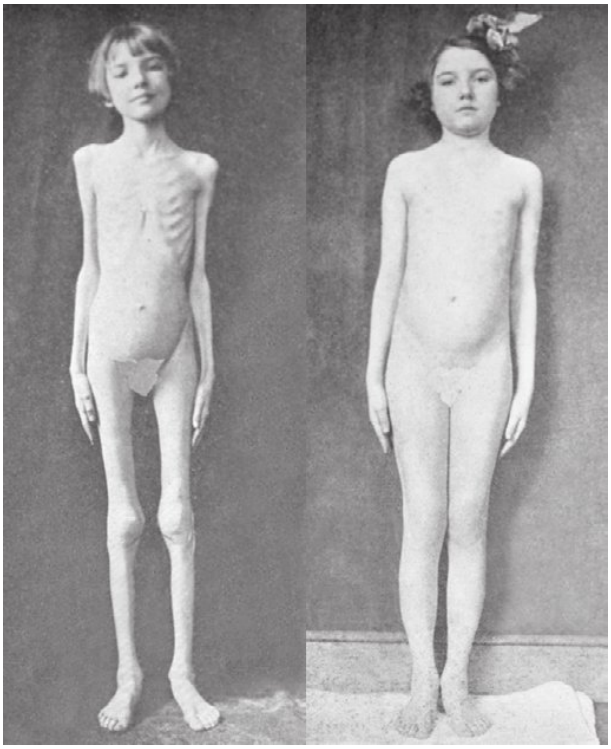


Figure 1.3: Effects of Insulin Therapy. Adopted from [53], the photographs show a young girl who underwent insulin treatment in a study by Geyelin [35] in 1922. The left figure shows the young girl with insulin-deficient diabetes before treatment with insulin, and the right figure, after successful treatment.

urine, when it has evaporated, leaves a sweet residue that looks like honey [52, 72, 33]. Later, the adjective "*mellitus*", the Latin word for *honey*, was added to the name by Edinburgh physician William Cullen (1710–1790) [52].

In 1815, the French chemist Michel Chevreul (1786–1889) identified the sugar in diabetic urine as glucose [14]. Later in the 1840s, Claude Bernard (1813–1878) proved that glucose is normally present in blood, and showed that it is stored in the liver (as glycogen) [48].

In 1889, an accident happened in a laboratory where Oskar Minkowski was working. After a house-trained dog had a pancreatectomy, it was no longer continent of its urine. After further analysis of the dog, Minkowski claimed that the pancreatectomy had caused severe diabetes in the dog [52]. In 1893, Gustave Laguesse proposed that pancreatic islets, described by Paul Langerhans in 1869, are involved in the production of an internal secretion that regulates glucose metabolism. At that time, it was still questionable whether there is a true glucose-lowering role associated with the internal secretion of pancreatic islets, which in 1909, Jean de Meyer named *"insulin"* (from the Latin for *island*) [33].

Finally, in 1921, insulin was successfully extracted from pancreatic islets by Banting, Best, Macleod and Collip in Macleod's lab at the University of Toronto in Canada. The first clinical use of insulin for diabetes treatment was conducted in 1922. Figure 1.3 shows the effects of insulin treatment on a young adolescent girl [35, 53]. The effectiveness of such treatments meant the discovery of insulin was a significant milestone in the history of diabetes.

Over the past century, there have been other breakthroughs, including modifications of insulin to

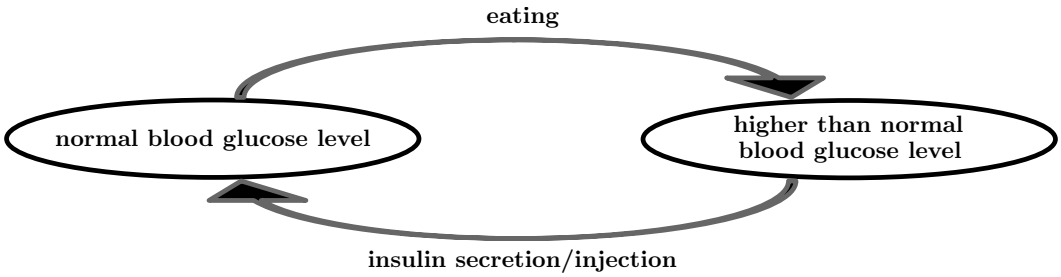


Figure 1.4: Glucose-Insulin equilibrium. When a person consumes food, the carbohydrates in the food break down into glucose in the stomach, and are absorbed into the blood stream. To bring the blood glucose levels back down to normal, in a healthy individual, the pancreas secretes insulin into the blood stream. The presence of insulin reduces the concentration of glucose in blood stream by allowing its disposal into cells. A type-1 diabetic, however, is incapable of producing insulin. Maintaining normal blood glucose levels requires patient intervention by injecting insulin.

increase its biological half-life, glucose monitoring, and insulin pumps. One of the most recent and significant breakthroughs was the the successful transplantation of islets of Langerhans cells from the pancreases of recently deceased people to the pancreas of diabetic patients by a team from the University of Alberta in Canada [61]. Despite all of these efforts to cure diabetes, success has still eluded the scientific community in this goal. Until a cure is achieved it is important to maintain glucose control and prevent diabetes complications. For this, a glucose-insulin equilibrium has to be maintained.

1.3.2 Glucose-insulin equilibrium

When we eat food, our digestive system breaks the carbohydrates down to glucose. The absorption of glucose in the intestine increases its concentration in the blood stream, which puts the body into a state of *hyperglycemia* (state of high blood glucose). Glucose, the key source of energy in human body, needs insulin for its routine disposal into cells. In a healthy individual, the pancreas produces insulin, which allows muscle and fat cells to absorb glucose from blood stream (Figure 1.4). Consequently the blood glucose level decreases back to the normal level – about 5.5 mmol/L (3.5–6.5 mmol/L)[3] [52].

Other mechanisms operate when the blood glucose goes below its normal value – that is, when the body enters a state of *hypoglycemia*. However, as standard diabetes management does not involve these mechanisms, we do not discuss them in this dissertation[4].

1.3.3 Types of diabetes

The vast majority of diabetic patients can be divided into two categories: *type-1* and *type-2*. In patients with *type-1* diabetes, the patient essentially does not produce insulin. This type begins with

[3] All the values referring to blood glucose in this dissertation are in SI units (mmol/L). To convert these values to mg/dl, the prevalent unit being used in United States and a few other countries, multiply the value in SI unit by 18. For instance, 4 mmol/L is equal to $4 \times 18 = 72$ mg/dL.

[4] For a preliminary introduction to blood glucose regulation in body, see [38].

the selective autoimmune destruction of insulin-producing pancreatic cells (β-cells), which later leads to the inability of the pancreas to secrete insulin. This insulin deficiency affects the glucose-insulin equilibrium and causes the blood glucose of the patients to rise. The body cannot use the glucose, so it senses starvation and then breaks down body fats in an uncontrolled manner, giving rise to ketoacidosis, which eventually leads to death. Type-1 diabetic patients without insulin will die in days to weeks. This type afflicts about 10% of diabetic patients.

Far more prevalent than type-1 is *type-2* diabetes, where patients may still produce normal amounts of insulin, yet their cells cannot utilize glucose efficiently. Sometimes, in addition to insulin resistance, some degree of insulin deficiency is also involved, where the amount of insulin secreted by the pancreas is lower than normal. This type of diabetes is usually associated with obesity or high BMI. The most important factors associated with the increased prevalence of type-2 diabetes are believed to be a high-fat, high-calorie diet and a sedentary lifestyle [53]. Patients with type-2 diabetes will not die if not given insulin, as they have enough of their own insulin to prevent ketoacidosis.

This dissertation focuses on ways to help type-1 diabetic patients manage their diabetes. Wherever we mention diabetes we are referring only to type-1 diabetes unless otherwise specified.

1.3.4 Current means of diabetes management

Type-1 diabetic patients regulate their blood glucose by injecting insulin multiple times during the day. There are two major types of insulin: *bolus insulin* and *basal insulin*. Bolus insulin is a rapid-acting insulin injected at meal times to balance the short-term burst of glucose associated with eating food. Basal insulin is a long-acting insulin with a slow release time. It provides the body with the background insulin that is needed for normal cell functions during the day.

The treatment policy, as mentioned above, determines the amount of each insulin injection – bolus and basal. The part of policy that deals with basal injections is usually in the form of fixed dosage at specific times, or in the form of a low dose insulin infusion in the background using an insulin pump; see Section 4.1.2.

As mentioned earlier, ideally a patient maintains a diabetes record or diary, in which she records the amounts of carbohydrates in each meal, blood glucose readings, insulin injections, and the time-stamp of each event. The patient's diabetologist then uses this diary (plus some other information about the patient, *e.g.,* Age, Weight, Gender, BMI[5], etc.) to improve the treatment policy to attain steady glucose control (Figure 1.5). High glucose concentrations in blood over the long term increase the risk of long term complications (chronic problems for internal organs) and hence needs to be avoided. However, low blood glucose values carries the immediate risk of passing out and coma, and thus has to be avoided at all costs.

Various options are available to a patient on how to take insulin injections and how to record his or her blood glucose readings; see Section 4.1. Next we discuss the standard form of policy that

[5]BMI stands for body mass index.

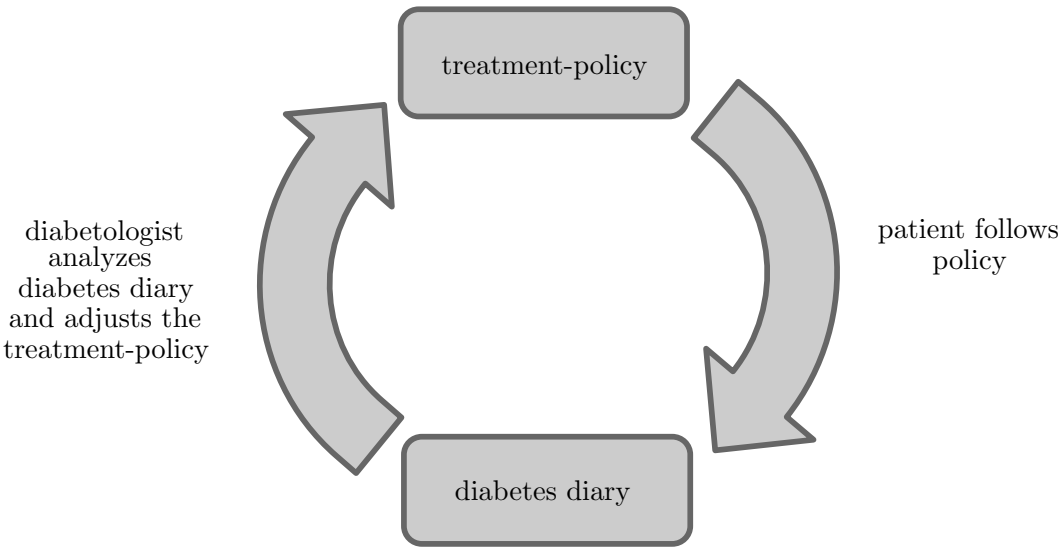


Figure 1.5: The standard policy adjustment (treatment update) cycle of a diabetic patient.

diabetologists use.

1.3.5 Standard treatment-policy formula

Most patients compute how much bolus insulin to inject per meal based on the following treatment-policy formula[6]:

$$Inj = \frac{bg - C_{target}}{CF} + \frac{carbs}{CR} \tag{1.1}$$

where:

Inj:	Bolus injection dose
bg:	Current blood glucose
C_{target}:	Target blood glucose (constant)
$carbs$:	the size of the meal (in grams of carbohydrate)
CR:	Carbohydrate-ratio
CF:	Correction-factor

In Equation 1.1, CR and CF are known as *policy parameters*. We also refer to the pair of parameters as $\theta = [\theta_1, \theta_2] = [CR, CF] \in \mathbb{R}^2$. The target blood glucose constant, C_{target}, is usually selected by the diabetologist and remains constant. For the purpose of this dissertation we are going to fix the target blood glucose constant to C_{target} = 6 mmol/L and not consider adjusting it.

An example can clarify how this policy is used. Assume that the patient's diabetologist has prescribed a correction-factor of 2 mmol/L/U and a carbohydrate-ratio of 10 gr/U ($\theta = [CR, CF] = [2, 10]$). Imagine the patient's blood glucose is bg = 8 mmol/L and she is about to consume $carbs$ = 50 grams of carbohydrates. According to the treatment policy, the patient should inject 6 units of bolus insulin for the current meal.

[6]This formula may be modified if the diabetic patient is planning to do exercise or any other activity that consumes glucose. In such a situation, the patient will need to inject less insulin than computed by Equation 1.1.

$$Inj = \frac{(8-6)}{2} + \frac{50}{10} = 1 + 5 = 6$$

We refer to the Equation 1.1 as *standard policy*. Note that both parameters in this policy are within a typical range of $CR \in [CR^{min}, CR^{max}] = [3, 30]$, $CF \in [CF^{min}, CF^{max}] = [.4, 2.8]$. In a real-world scenario, a diabetic patient, might have different policies for different meals (*e.g.,* one for lunch, and one for breakfast and supper), but in this thesis, to simplify the problem, we assume the policy is the same for all meals.

Among the components of the policy adjustment process, we thus far have described the form of the standard policy that patients use. Of course, in each iteration of the cycle, the diabetologist wants to improve on the current policies. This requires a benchmark (performance measure), which provides the quality of a given policy. This score can be used to compare different policies, and it helps the diabetologist to evaluate the adjustment of the policy. In the next section we provide more details about this concept.

1.3.6 Performance measures

One of the challenges in diabetes management is the lack of a gold standard quantitative measure to evaluate the performance of the treatment policy. In the following section we introduce the currently accepted performance measure that is used by diabetologists. Later we introduce our own performance measure.

A1C (the standard performance measure)

Hemoglobin A1C, also known as HbA1C, glycated hemoglobin, or more commonly now, simply A1C, is a specific form of hemoglobin, which is formed by the glycosylation of hemoglobin – an irreversible attachment of glucose to hemoglobin. It was distinguished from other forms of hemoglobin for the first time by Huisman and Meyering in 1958 using chromatography [36]. Ten years later, Bookchin and Gallop realized that A1C is a glycoprotein [9]. Rahbar and coworkers in 1969 noticed that levels of A1C are higher in diabetic patients [55]; and in 1976 Cerami et al. [39] suggested that A1C can be used to measure the quality of glucose control in diabetic patients [79], as it shows the percentage of hemoglobin that is glycated, which corresponds to the average blood sugar level for the past two to three months. The A1C test has been considered the most reliable method for assessing chronic glycemic control and has become the worldwide standard –*e.g.,* is used by the FDA to evaluate new diabetic drugs [3, 46, 59].

A1C is correlated with average blood glucose

Nathan et al. (2008) sought to define the mathematical relationship between A1C and average blood glucose. Their study included 500 subjects of various types of patients (non- diabetic, type-1, and type-2 diabetic patients). They obtained 2700 glucose samples from each patient over the three month duration of the study.

Using linear regression, they expressed average blood glucose as a function of A1C levels for most patients with type-1 and type-2 diabetes ($R^2 = 0.84$):

$$Average\ BG(mmol/L) = 1.59 \times A1C - 2.59$$

Our performance measure

Although the A1C measure is informative about the average level of glucose in blood and relates to long-term complications, it cannot provide any information about variation in blood glucose levels. These variations could reflect the instability in the blood glucose and possibly hypoglycemia. For example, a patient could have a normal A1C but be repeatedly passing out with hypoglycemia, as she often has hyperglycemia to compensate. Having a more informative performance measure is critically important for automated policy adjustment. We need a performance measure that incorporates some aspects of blood glucose variation, especially the contrast in the consequences of having high blood glucose vs having low.

To come up with a good performance measure, we first need to determine what is expected of an ideal controller. An ideal controller must keep the blood glucose level in an acceptable range, which for a diabetic patient is considered to be between four and eight mmol/L [33]. When the blood glucose is within this range, it is best to maintain it as close to six mmol/L as possible; when it is outside, it is desirable to get it back in the range.

We therefore need to identify an appropriate penalty to assign if the policy is not performing well by letting the blood glucose level go outside of the specified range. If this happens, patients will enter into either the hyper- or the hypoglycemic states, each of which lead to biological complications for the patient.

When in a state of *hypoglycemia*, in which the blood glucose drops below four mmol/L, a patient develops various symptoms. Since glucose is vital for brain function, there will be neuroglycopenic[7] symptoms, such as confusion, weakness, drowsiness, tiredness, blurred vision, etc. In addition, as the body attempts to address this, it will quickly produce symptoms to help counter this, which include autonomic symptoms such as sweating, trembling, warmness, pounding heart, anxiety, and shivering [32]. A more serious case is acute hypoglycemia, when the blood glucose level drops significantly below four mmol/L in a short time. This sudden drop can cause unconsciousness, seizure, and in rare cases, permanent brain damage or death.

In contrast, *hyperglycemia*, where the glucose is higher than normal, is typically not associated with any acute symptoms unless very severe. However, chronic hyperglycemia can cause various complications within two to ten years. The severity of the complications depends on how high the average blood glucose level is. These complications may occur even at blood glucose levels slightly above normal. The common complications usually affect vital organs by causing chronic

[7] Lack of glucose in brain.

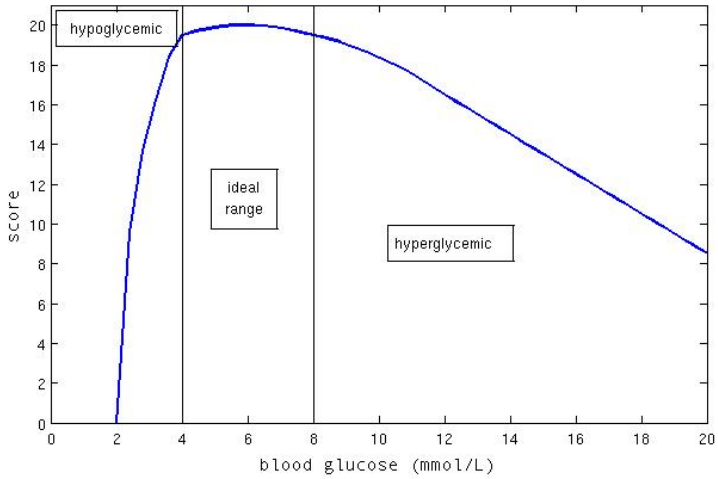


Figure 1.6: Our devised score function, BGSF.

problems that affect nerves, kidneys, eyes (retina) and increased risk of coronary heart disease or other damages such as diabetic foot (ulcers) [1, 17].

While some score functions have been used on continuous blood glucose readings (*e.g.,* M-value [60]), there exists no gold standard, or generally-accepted score function[8]. Therefore, we decided to define a score function that matched our objectives.

We let the normal level of blood glucose, six mmol/L, have the highest score. Knowing that hypoglycemia can cause immediate and severe problems, our proposed score function assigns exponentially lower scores for blood glucose levels lower than four mmol/L. For Hyperglycemia, however, as the effects are chronic, the proposed score of blood glucose values above eight mmol/L linearly decreases. Our blood glucose score function, *BGSF*, is shown in Figure 1.6.

This score function evaluates a single blood glucose reading, but it can also be used to evaluate a treatment policy over a period of time (as we see next). More details will be provided in Section 2.2.5 and Section 4.2.5.

1.3.7 Performance surface of the standard policy

In order to better understand the standard treatment-policy, we examined variations in the performance of this policy with respect to its parameters. To achieve this goal, we sampled around 1600 policies with different parameter settings from the $CR \times CF$ space and evaluated their performance using the average BGSF score of the blood glucose readings over 9 days[9]. Using this information for patient 2 (one of our *in-silico* patients), we provide the contour plot of the performance surface in Figure 1.7. This surface is created by fitting a curve (linear interpolation [25]) to the sampled points for patient 2. In this plot, we indicate regions in parameter space that correspond to policies

[8]For information about the score functions used on discrete readings (*e.g.,* a few samples per day), see [58] and for the ones that has been used on continuous readings see Fig.4 of [24]

[9]Refer to mean-reward in Section 2.2.5 for more detail about this averaged score.

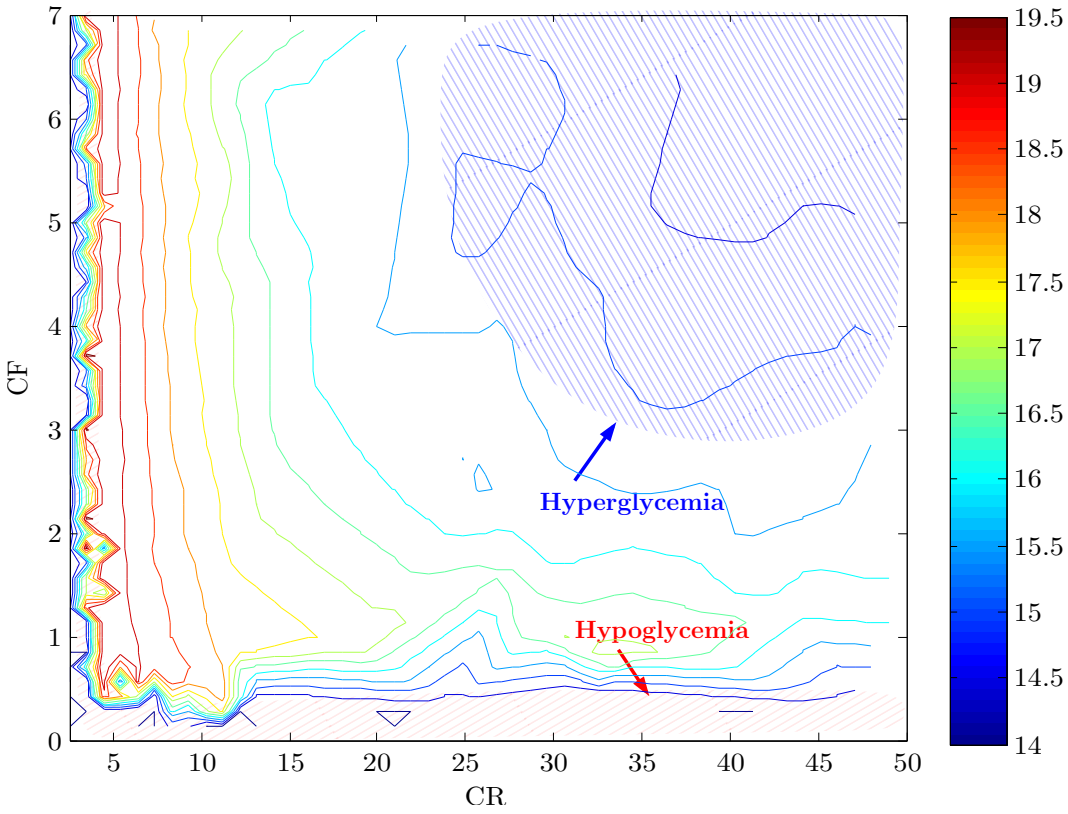


Figure 1.7: Contour plot corresponding to the performance surface of the standard policy with respect to policy parameters CR and CF. The policy performance is calculated by averaging the BGSF scores of the blood glucose readings over 9 days of following the policy. The two hatched regions show the policy parameter settings that would cause hypoglycemia (low blood glucose) or hyperglycemia (high blood glucose). This contour plot is generated for patient 2 (PID=2); the contour plots for other patients can be found in Appendix C.

that lead to hypoglycemia and hyperglycemia.

A three-dimensional plot of the same surface is given in Figure 1.8. To highlight the more informative parts of the plot, all performance values below 14 are replaced with the value 14. Performances below this threshold roughly correspond to average blood glucose levels under 4 or over 13, both of which are undesirable.

We can make some observations about the shape of the performance surface[10]. First, we see that the optimal (highest) performance region of the surface is independent of the CF value. This expansion implies that, when the CR parameter is in proximity of its optimal values, the performance of the standard policy is insensitive to CF. This insensitivity can be explained by the fact that the correction factor, as its name suggests, is important when there is a need for correction (that is, when the blood glucose is either too high or too low), and it becomes almost irrelevant once the bolus peaks are handled with the correct carbohydrate ratio (and thus the blood glucose is in the correct range). Second, we observe the high sensitivity of performance with respect to the CR parameter –*i.e.,* the

[10]Note that the surface plot for our other (*in-silico*) patients looks different – in particular, for patient 3 (in Appendix C) we see that the surface has a highly irregular shape.

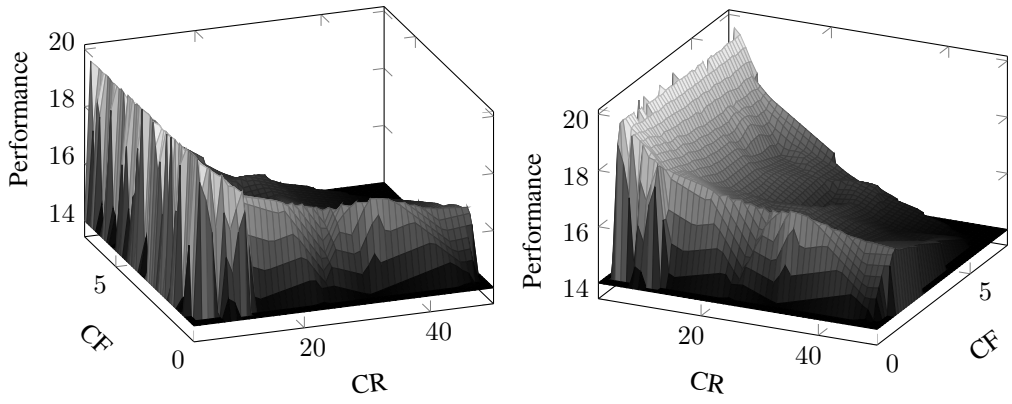


Figure 1.8: Performance surface of standard policy with respect to its parameters. Both of the pictures exhibit the same surface but from different view points. Policy performance is calculated by averaging the BGSF scores of the blood glucose readings over 9 days. Note that there are sharp cliffs around the low values of CR and CF. This performance surface belongs to patient-2 (PID=2).

CR values of 4 to 7 seem to have around optimal performance (about 19), but a CR value of 2 leads to performances below 2. The final notable observation is the quick drop in performance of the policy when the CR parameter takes smaller than optimal values. On one side of the optimal ridge are policies that perform well; on the other side, and not very far away, we have policies that lead to hypoglycemia. The proximity of the optimal policies to the devastating ones suggests that the search for optimal policies should be done cautiously.

1.4 Related work

Various studies have investigated ways to utilize computer-based systems to control the blood glucose level of diabetic patients. These methods can be mainly categorized into *open-loop* and *closed-loop* control systems [27, 70].

Following the definitions provided in [27, 10], we call a diabetes management system "open-loop", when the patient (or diabetologist) is involved in the decision making related to each insulin injection and it is the patient's responsibility to decide the right dosage of insulin to inject[11]. Regardless of the injection scenario or glucose monitoring system used, as long as a patient is making the decision about the injections, it is categorized as an open-loop system[12].

In contrast, a closed-loop blood glucose controller (also known as *artificial pancreas*) is a system that tries to mimic the function of the pancreas [27, 70]. It tries to inject the right amount of insulin at the right time such that the fluctuations in levels of blood glucose of a patient imitate that of a healthy individual. To achieve this goal, the controller needs live feedback of the current blood

[11] Note that this dosage might be calculated based on her treatment policy.

[12] This definition of an open-loop system in diabetes literature might rather seem different from the definition of open-loop controller in control theory. Maybe *patient-mediated closed-loop system* or *open-loop intermittent feedback control* would be a more precise name. Although there is no direct feedback loop in these systems, these systems rely on an indirect feedback of the blood glucose values (collected by patient).

glucose values. For this reason, closed-loop controllers usually need a continuous blood glucose monitoring (CGM) system in conjunction with an insulin pump. Thus, one of the drawbacks of using a closed-loop (diabetes) control system is its financial burden. The cost for the combination of a CGM and an insulin pump is currently about $10,000 to startup plus the ongoing operating cost of about $300 per month. Furthermore, it is usually harder to validate and test closed-loop systems; as the patients should be under the vigilant supervision of a medical team to prevent potential life threatening situations should the system malfunction.

Definition of Model-based Methods

Open-loop and closed-loop systems can be further categorized as methods that are either *model-based* or not; see Figure 1.9. A method is categorized as model-based if it relies on a mathematical model of the physiological processes involved in the glucose-insulin interaction. Before we get into the details of the control algorithms, we briefly describe the common practice method used to create such mathematical models.

Mathematical models of the glucose-insulin interaction

Many different approaches have been used to create mathematical models for glucose-insulin interaction dynamics in the body. The prevalent approach is to create models based on the current physiological knowledge of the organs and bio-mechanisms involved in regulation of glucose, insulin, and possibly other hormones [43, 5]. Bolie's model [8] and Bergman's *minimal model* [6] are two examples of such models.

These models are only as good as our knowledge about the physiology of the human body. It might be impossible to create a precise model of these mechanisms due to an insufficient understanding of the biological mechanisms involved, or simply because we cannot calibrate the model when a specific substance in the body cannot be measured (due to technical or ethical issues). For instance, Bolie's model considers the glucose-insulin regulatory system consisting of liver, pancreas and peripheral tissues ignoring some very important organs such as kidneys.

On the bright side, the more we learn about the human glucose regulatory systems, the more accurate our models become[13]. Thus far, the *Type-1 Diabetes Mellitus Simulator (T1DMS)* system, described in Chapter 3, is one of the most comprehensive attempts that incorporates physiological knowledge of the glucose regulatory system into a model.

1.4.1 Open-loop dosage adjustment

Several open-loop diabetes control algorithms use clinically obtained rules to perform their decision-making tasks. These systems are devised based on clinical experience. Other methods prefer to employ mathematical models to assist dosage control.

[13] Of course, this depends on our ability to incorporate some of the novel knowledge into the models.

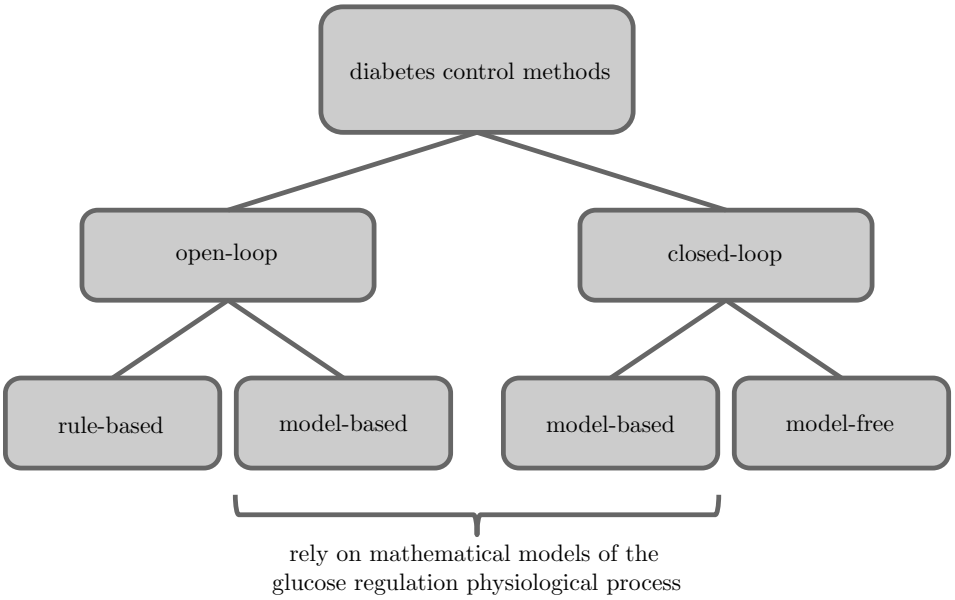


Figure 1.9: Categorization of computer-based diabetes control systems.

Scenario: Hyperglycemia (high blood sugar) not explained by diet/exercise/insulin
•If fasting blood glucose on rising is greater than 5 mmol/L for 2 days in a row, increase your evening long-acting insulin by 1-2 U. •If blood glucose 2 h after breakfast is greater than 6.7 mmol/L AND blood glucose before lunch is greater than 5.8 mg/dl for 2 days in a row, increase your morning short-acting insulin by 1-2 U. •If blood sugar 2 h after breakfast is greater than 6.7 mmol/L, but less than 5.8 mmol/L before lunch, consult your doctor or clinician. •If blood glucose 2 h after lunch is greater than 6.7 mmol/L, AND blood glucose before supper is greater than 5.8 mmol/L for 2 days in a row, increase your morning long-acting insulin by 1-2 U. •If blood glucose 2 h after lunch is consistently greater than 6.7 mmol/L AND blood glucose before supper is less than 5.8 mmol/L, consult your doctor or clinician.

Table 1.1: An example of the rules used in a rule-based algorithm for insulin regimen adjustment; adapted from [64].

Open-loop rule-based algorithms

Rule-based algorithms (Figure 1.9) usually find policy adjustments directly from the patient's condition. This is usually done by explicitly defining rules and/or formulas to adjust the insulin dosage by consulting a diabetologist [42]. These rules can be used directly or in a knowledge-based system.

An example rule-based system is Skyler et al. [64]. See Table 1.1 for some of their rules for a specific scenario.

On one hand, rule-based systems help patients adjust their treatment policy when the rules are correct and do not contradict each other. On the other hand, their functionality is limited by the size and broadness of their rules. Some of the rules (Table 1.1) explicitly require input from the diabetologist. Furthermore, no adaptation occurs, and these rules are fixed unless updated by an expert physician.

Knowledge-based systems, also known as expert systems, are more sophisticated versions of these algorithms. Expert systems are systems that attempt to act the way experts think the expert should act. An expert system consists of two main components: a knowledge base and an inference-core [29]. The knowledge base in this case is a set of rules that are written by an expert. The inference core is a reasoning backbone that can deduce facts (given the current situation of the patient) that are not stored in the knowledge-base but can be deduced from the other rules that are stored and also patient information.

One such expert system is *ExDiabeta* developed by Siemens. This system recommends the required insulin dosage adjustment, based on blood glucose readings that are measured during the day (7 measurements in total). The knowledge base for this system is contained in about 300 rules. For more information about this system and similar systems see [42].

Rule-based algorithms suffer from some fundamental issues. These methods rely on expert physicians to add more rules and resolve contradictions in the rule set. More importantly, if such a method fails in a situation, it always fails in the same situation, unless a physician revises the rules by hand, a very time-consuming process that might benefit from automation.

Open-loop model-based approaches

Model-based approaches differ from the rule-based systems by relying on mathematical models of the glucose-insulin interaction. The open-loop model-based methods (Figure 1.9) are further categorized into two groups: manual and automatic. In the first group, the methods use the model to build a simulator for a specific patient; this simulator is then used by the diabetologist to search for the best dosage adjustment. The diabetologist examines multiple dosage adjustments on the simulator and evaluates their performance using her professional knowledge. The best performing dosage adjustment is then used to update the patient's treatment policy [41, 12] [14]. In contrast, the methods in the second group (automatic), use optimization techniques on the mathematical model

[14] For more examples, see Section 3.3.1 of [42].

Control technique	Reference	Notes
Full state feedback	Shimoda et al. [63]	If fails in situation s, always fails in s
Adaptive Control	Pagurek et al. [49]	Adapts (learns from failures)
Model Predictive Control	Andreassen et al. [4]	

Table 1.2: A few examples of model-based control techniques that has been used in diabetes literature. For a comprehensive list see [70, 13, 43].

to come up with the optimal[15] dose adjustment for the patient (*e.g.,* [34]). Note that these systems provide suggestions to the diabetologist (or to the patient).

1.4.2 Closed-loop blood glucose control

We can categorize closed-loop methods into two classes: *model-free* versus *model-based* [13]; see Figure 1.9. Closed-loop model-based control methods, very similar to open-loop model-based methods (mentioned in Section 1.4.1), are based on mathematical models of glucose regulatory systems in the human body. Various researchers have attempted to solve diabetes problems using model-based control techniques. However, only a few diabetes studies have tried to use a model-free approach. Note that a model-free approach is the focus of our research.

Closed-loop model-based control

Model-Based Control

Model-based control methods apply algorithms from control theory to the mathematical model of the glucose-insulin regulatory system [13]. The details of these algorithms need a control theory foundation and are outside of the scope of this dissertation. Therefore, we will confine our introduction to a very short list of model-based control techniques that have been used in diabetes literature with an example for each in Table 1.2; for more information about the applications of model-based control algorithms in diabetes, see [70, 13, 43].

Closed-loop model-free control

As can be inferred from the name, model-free diabetes control algorithms are not dependent on a physiological model and instead rely solely on experimental data. The two main classes of model-free control methods in the closed-loop diabetes literature are *proportional-integral-derivative* (PDI) and *curve-fitting-based control* [13, 43].

Model-Free Control

Curve-Fitting

In the algorithms that use curve-fitting for glucose control, the usual approach is to first gather experimental data by observing the effects of insulin (as well as glucose) on blood glucose levels for a diabetic patient. Thereafter, the goal is to fit a response curve to the data for both glucose and insulin to be used for the blood glucose control. For instance, the curve-fitting controller by Albisser et al. [2] utilizes a system capable of intravenous injection of glucose and insulin separately. Insulin and glucose response curves are manually selected by an expert from a family of sigmoidal response

[15] Like any other optimization method, these methods need to have an objective function, which is a measure of the performance of the adjusted treatment (see Section 1.3.6).

curves based on patient's clinical characteristics (*e.g.,* body weight, daily insulin requirements). These curves specify the minute-by-minute insulin and glucose infusion rates based on current blood glucose and its rate of change. The main problem with curve-fitting control algorithms is that a delay in the blood glucose readings can make the system unstable and put it out of synchronization [13].

PDI

The second category of model-free control methods in the literature are Proportional-Derivative-Integral (PDI) methods, a generic method borrowed from control theory. In the PDI approach, an error value is calculated based on the difference between the blood glucose signal value and a desirable target blood glucose value. PDI consists of three main components, all of which play a role in selecting the (next) insulin injection through a weighted sum of their output signals. The proportional ("P") component is the error itself (the difference). The derivative component ("D") is the derivative of the error signal (the slope of the difference). Finally, the integral ("I") term is an integrated version of the error signal over a period of time (sum of differences).

Solving a PDI involves finding the right weight for each component. Finding these weights is not an easy problem but there are black-box solvers that can be borrowed from control theory (*e.g.,* Matlab's PDI tuner) to help. An instance of using PDIs in diabetes literature is the work of Palazzo and Viti [50] [16]. Similar to the curve-fitting approaches, a disadvantage of PDI methods is that they are sensitive to delays in blood glucose readings.

1.4.3 Model-free, or model-based, that is the question

Model-free methods, in either open-loop or closed-loop systems, have some advantages over their model-based counterparts (Figure 1.9). One problem with the model-based methods is that they need an accurate model, and building such a model usually requires special and costly physiological studies of the processes involved in glucose regulation. However, model-free control algorithms rely solely on the "simple" type of data that patients normally collect. Another problem with the model-based methods is their dependence on a patient specific model. Thus any modification to a component of the mathematical model will require the parameters of the model-based method to be re-calculated.

Flexibility

Due to recent advancements in technology, glucose meters and insulin injection devices have become cheaper and provide more functionality. Glucose meters now have memories that can store days worth of glucose reading logs [15]. Furthermore, various cell-phone tracking applications are available to help patients keep a record of their blood glucose values, insulin injections, exercise, and diet [51, 85]. More advanced devices such as insulin pumps and continuous glucose meters make it even easier to maintain the patient's computer-readable diabetes diary. Model-free approaches have the ability to utilize this huge amount of "simple" information, while model-based methods usually need more "advanced" measurements (*e.g.,* the concentration of blood glucose in the hepatic vein). Given the above factors, we believe model-free methods are an important avenue of research that

Vast Amount of Dara

[16] For more examples, see Chee and Fernando [13] and Lunze et al. [43]

has not been exploited to its full advantage by the diabetes community.

Scarcity of Model-Free Methods

Our decision to employ a model-free approach to tackle the diabetes dosage adjustment problem is mainly motivated by the ever-increasing amounts of available patient data coupled with the power of model-free approaches to utilize it. In addition, the scarcity of the model-free research in the literature of diabetes control has further encouraged us to examine the possibility of using machine learning techniques in this context.

One of the important steps in a study to develop a model-free approach is the evaluation phase. Such studies could be on either real patients or on *in-silico* patients. An obvious major disadvantage of clinical studies (involving real patients) is their cost and length of study. Considering all the advancements that have been made in the modelling of *in-silico* patients, we decided to use an insulin-glucose mathematical model to evaluate our work, acknowledging the need for later clinical evaluation.

To select the mathematical model for our *in-silico* patients, we did a simple search for implementations of a type-1 diabetes simulator and filtered studies by their number of citations and the availability of their simulators for public use. We ended up with two options: Lehmann et al. [41]'s *AIDA* and Dalla Man et al. [21]'s *T1DMS* simulators.

Why T1DMS?

Our first choice for type-1 diabetes simulation was AIDA, primarily because it was available at no cost. However, after careful deliberation we ruled out this choice mainly because we could not obtain the stand-alone version from the authors. Using the AIDA web-interface[17] was not an option as well because it was not capable of handling the large number of simulation requests that we required. In contrast, for T1DMS, we were able to purchase its stand alone version. Also, this simulator has a more sophisticated model of the glucose-insulin mechanism (compared to AIDA). We will provide details about T1DMS in Chapter 3.

[17] Available at http://www.2aida.net/aida/options.htm.

Chapter 2

Machine learning foundations

Machine learning has been used to tackle various problems, from playing games like chess and backgammon to helping physicians in diagnosing cancer. In this dissertation we use two different techniques from the machine learning field, *supervised learning* and *reinforcement learning*, to deal with the diabetes dosage-adjustment/control problem. Simply put, supervised learning involves teaching through the use of examples, while reinforcement learning involves teaching through the use of encouragement/punishment. These methods have their own specialities and are designed to deal with specific forms of problems, which will be discussed further when they are introduced. Section 2.1 first describes supervised learning and later, Section 2.2 introduces reinforcement learning.

2.1 Supervised learning

The goal of supervised learning is to approximate the true relationship between x and y, given a training set that has sample pairs of x and y. In Section 4.2 we use supervised learning to build a mapping from patient data, to best treatment policy parameters (of that patient).

Formally, a training set of size N: $(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)$ is given where each of the pairs in the training set is an example pairs of input and output, where the outputs are generated by an unknown function $y = f(x)$. The goal of supervised learning is to find a function h (called a $predictor$) that approximates the true function f, $h \approx f$ [57].

If the output value is $y \in \mathbb{R}$, then the supervised learning problem is called *regression* [57]. In regression, we call $x \in \mathbb{R}^d$ the input feature vector and y output variable. We can stack up the input feature vectors and outputs in two matrices X ($N \times d$) and Y ($N \times 1$).

A Simple Example

A simple example of regression is to predict the A1C of a patient based on his averaged blood glucose (ABG). Let us assume that the duration of the study is three months and that 1000 patients are involved in this study, for whom the ABG's are calculated by averaging the blood glucose over this period. Also assume that we have measured the patients' A1C on the last day of the study. The set of input feature vectors X is column vector of 1000 ABG readings (a matrix of size 1000×1) and Y is another column vector containing the A1C measures of patients (output feature vectors).

Each row of X and Y corresponds to one patient. Regression can help us train a predictor on this training set (X and Y) to find an h function. Later we can use this function h on the averaged blood glucose readings of a new patient to estimate her A1C measure. In order to estimate our potential error in estimating the A1C of this new patient, we can use cross-validation, a technique that will be discussed in Section 2.1.2.

2.1.1 Support vector regression (SVR)

Various methods have been created to address the regression problem. One of the most significant of these is *Support Vector Regression*(SVR) [74].

For simplicity we explain the SVR problem in brief for a linear predictor. In this case, the predictor has the following form:

$$\hat{y} = h(x) = xW + b \quad \text{where: } W \in \mathbb{R}^{d\times 1}, b \in \mathbb{R}, x \in \mathbb{R}^{1\times d}$$

or in matrix form :

$$\hat{Y} = h(X) = XW + \vec{1}b \quad \text{where: } W \in \mathbb{R}^{d\times 1}, b \in \mathbb{R}, X \in \mathbb{R}^{N\times d}$$

To calculate the best W,b for approximating the response Y from X, SVR solves the following optimization:

$$\min_{W,b,\vec{\xi}} \frac{1}{2}||W||^2 + C\cdot(\vec{1}^\top\vec{\xi} + \vec{1}^\top\vec{\xi^*}) \quad \text{subject to: } \begin{cases} \vec{\xi} + \epsilon & \geq & Y - XW - b \\ \vec{\xi^*} + \epsilon & \geq & -Y + XW + b \\ \vec{\xi}, \vec{\xi^*} & \geq & \vec{0} \end{cases}$$

In this optimization, SVR uses the soft margin loss function and the slack variables ξ, ξ^*. Note that this loss function penalizes a prediction only if its error is more than ϵ; see Figure 2.1. This loss function and the use of slack variables provide a feasible optimization that can be solved efficiently.

Various formulations and implementations of SVRs exist that are based on the same idea but have minor differences. For example, another formulation of the SVR supports the use of *kernel*s (dual formulation) [65]. A kernel is a matrix whose (i,j) entities shows the similarity between two feature vectors i and j; the simplest example is a dot product ($XX^\top$) kernel. For a comprehensive introduction to support vector regression, and support vector machines and kernels in general, refer to [74, 62].

In our experiments we use the *SVMTorch II* implementation of the SVR [16], which is optimized for large training sets (*i.e.,* large N).

2.1.2 Cross-validation

Evaluation of a machine learning method, L, usually involves two phases: the *training phase* and the *test phase*. The datasets used in these two phases are denoted respectively as D_T and D_E (datasets that could be equal – *i.e.,* D_T = D_E). The output of the training-phase of the learner L is a predictor

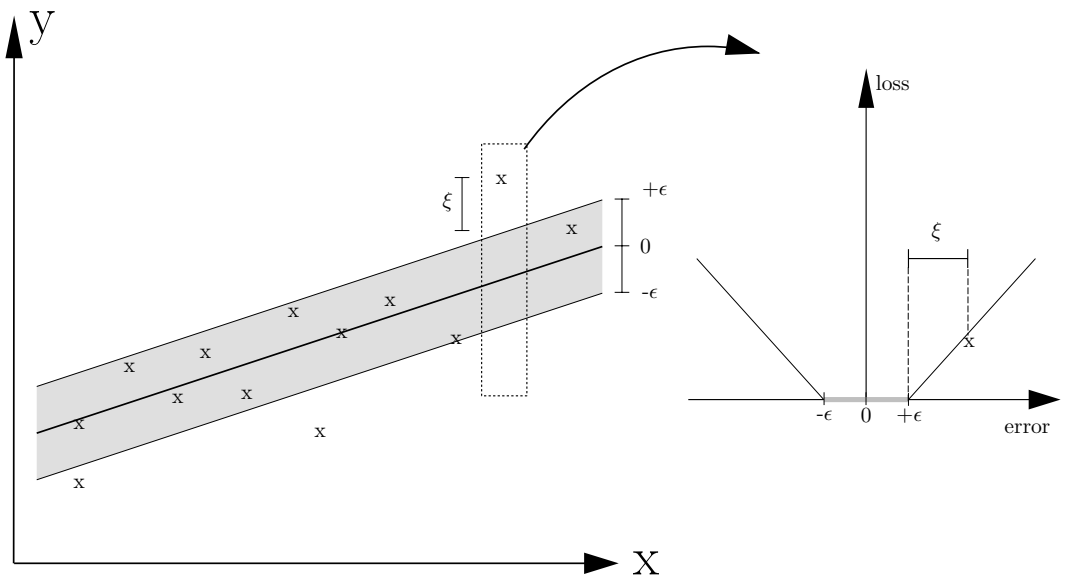


Figure 2.1: This figure visualizes the soft margin loss for a linear SVR. Adapted from [65]

P that is obtained from *training data* (D_T). During the test-phase the predictor P is evaluated on *testing-data* (D_E) and reports the resulting score (*e.g.,* accuracy, error, etc.).

One way of evaluating the performance of a machine learning (prediction) method is *cross-validation* [82]. This method is especially useful when the amount of data is limited. In k-fold cross-validation we first partition the available training data, D, into k disjoint subsets (or folds) $D_1, D_2, \cdots, D_k$ where $D = D_1 \cup D_2 \cup \cdots \cup D_k$ and $\forall\ 1 \leq i, j \leq k\ ,\ i \neq j\ :\ D_i \cap D_j = \emptyset$. In the cross-validation process we evaluate a machine learning method k times. The test data for fold-i is: $D_E^i = D_i$ and the train data for this fold is: $D_T^i = D \setminus D_i$. After evaluating L on all of the k folds, we can aggregate the evaluation scores over folds and report the desired statistics – *e.g.,* average score.

2.2 Reinforcement learning (RL)

"The idea that we learn by interacting with our environment is probably the first to occur to us when we think about the nature of learning. When an infant plays, waves its arms, or looks about, it has no explicit teacher, but it does have a direct sensorimotor connection to its environment. Exercising this connection produces a wealth of information about cause and effect, about the consequences of actions, and about what to do in order to achieve goals" [67].

The above paragraph introduced the form of learning that well explains the main idea behind *reinforcement learning* (RL); learning by interacting with the environment. This is in contrast to supervised learning, which needs an explicit teacher.

Reinforcement learning is usually used to solve sequential decision-making problems. The diabetes control problem is a good example of this sort of problem.

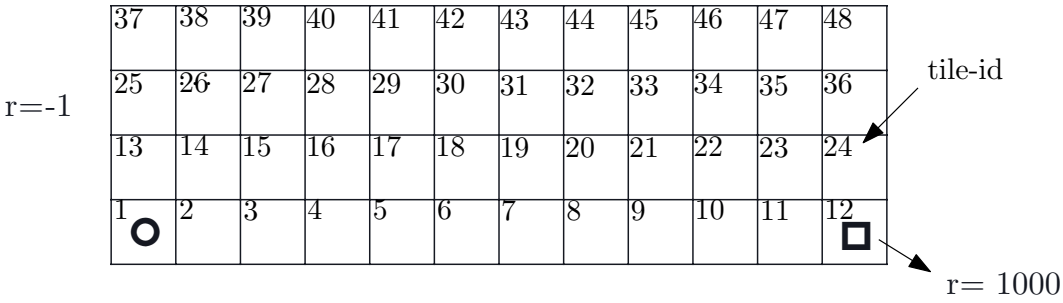


Figure 2.2: Grid world.

2.2.1 Sequential decision-making problems

Many problems can be formulated as a *sequential decision-making problem*. We represent a sequential decision-making problem using two major entities: *agent* and *environment*. These two entities interact with each other in a sequence of decision-making events. It is the agent's responsibility to make decisions by performing an action. The environment, on the other hand, is anything that surrounds the agent and can interact with the agent. In the following we introduce sequential decision-making problems with the help of a simple example.

2.2.2 Grid world

Grid world [67], is a simple environment consisting of a N by M grid of tiles. Two tiles in this world are special, goal (□) and start (○). In every step the agent should choose an action, to move: Up (U), Down (D), Left (L), and Right (R). All of the tiles are labelled with unique *tile-ID*s. The *state* of the agent in the environment is the tile-id of the tile it currently occupies. There is a -1 *reward* assigned to every tile except the goal tile, which has a reward of 1000. The agent receives the reward of a tile as soon as it enters that tile. The goal of the agent is to accumulate the highest reward. The world starts with the agent in the ○ state and ends when the agent is in the □ state (see Figure 2.2).

The grid-world is a very simple sequential decision-making problem. The environment dynamics in this example are very simple; at every state, the environment changes the state of the agent according to its action. For example, if the agent is in state 3, after it has chosen action U (Up), the environment will transit it from state 3 to 15.

A *policy* is a mapping from states to actions that tells the agent what action it should take in its current state (tile-id). Here, one wants to provide the agent with a policy that guides it through the grid from start to goal by passing the least number of intermediate tiles. This policy is called the optimal policy as it helps the agent to get the most cumulative reward (Figure 2.3).

The dynamics of the environment is not always as simple as the above version of the grid world. We can change the grid world dynamics to make it more complex by adding a lock on the goal tile and then putting its key in another tile (lets assume that is tile 22). The agent now needs to pass through the key tile to be able to enter the goal state. Thus the dynamics of the system (specifically

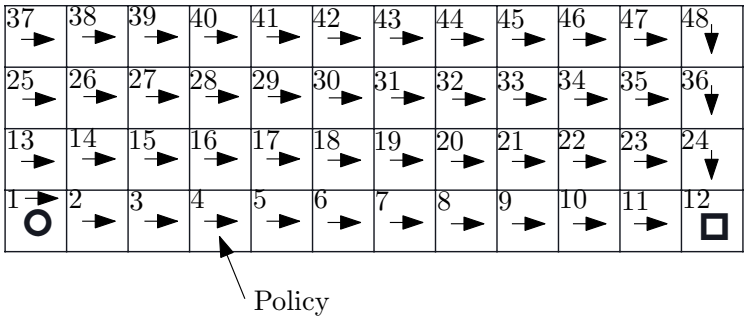


Figure 2.3: Optimal policy in the Grid world. The arrow in each tile shows the action that agent should take to arrive at goal (G) achieving the highest accumulating reward.

for the states leading to goal) cannot be represented as a function of the agent's current tile and action alone. This is because, it is essential to know if the agent is in possession of the key or not. Another way of making sure that the agent is able to unlock the goal is by knowing if the agent has passed through state 22, which requires having the history of agent's transitions in the grid world.

Markov Property

In this context, if any transition in an environment is independent of the past given the current state, we say the Markov property holds [67]. For example in the grid world with the lock, the Markov property does not hold if state (or the agent's observation of it) only contains the tile-ID variable. This is because the agent's transition to the goal state not only depends on the agent's tile-ID (location) but also on its possession of the key. However, the Markov property would hold if a Boolean variable showing this possession is added to the state variables[1].

A sequential decision process is called a *Markov Decision Process* if the Markov property holds for its transitions.

In the standard reinforcement learning framework, it is usually assumed that the environment consists of a Markov Decision Process (MDP) with a finite number of states and actions[2].

2.2.3 Markov decision process (MDP)

An MDP is defined by quadruple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where:

- $\mathcal{S} = \{s_1, s_2, \cdots, s_{|\mathcal{S}|}\}$ are *states*
- $\mathcal{A} = \{a_1, a_2, \cdots, a_{|\mathcal{A}|}\}$ are *actions*
- $\mathcal{P}(s, a, s') = Pr(S_{t+1} = s' | S_t = s, A_t = a)$ are state transition probabilities, characterizing the distribution over the next state.
- $\mathcal{R}(s, a, s') = E[R_{t+1} | S_t = s, A_t = a, S_{t+1} = s']$ is the *reward function*, that determines the expected reward associated with a state transition accomplished by a specific action.

[1] This Boolean variable would be true if the key is in possession of the agent and false otherwise.
[2] For a comprehensive introduction to MDP see [54]

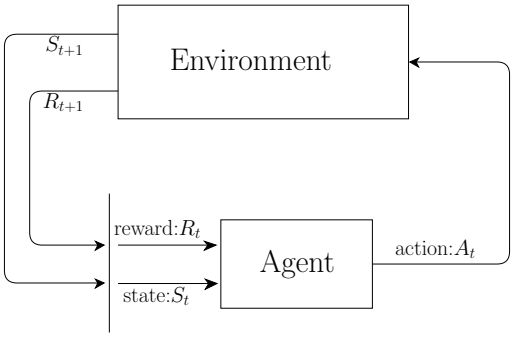


Figure 2.4: Conventional reinforcement learning setup. In every decision-making time-step t, the agent observes a state $S_t \in \mathcal{S}$ from the environment, then decides to perform the action $A_t \in \mathcal{A}$. The environment receives this action from the agent and emits back a reward $R_{t+1} \in \mathbb{R}$ value, and the agent arrives at the new state S_{t+1}

.

- $\mathcal{R}(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s')\mathcal{R}(s,a,s') = E[R_{t+1}|S_t = s, A_t = a]$, is another similar form of reward function.

As in the grid world example, in every decision-making time-step $t \in \mathbb{N}_0$, the agent first receives information from the environment that shows its current state $S_t \in \mathcal{S}$. The agent then decides to perform an action $A_t \in \mathcal{A}$ based on its policy. The environment receives this action from the agent and emits back a reward $R_{t+1} \in \mathbb{R}$ value, at the same time, the agent receives information about its next state, at time-step $t+1$, S_{t+1}(see Figure 2.4).

Diabetes MDP

In the diabetes world, if we ignore the basal injections for simplicity, we can think of meals as decision points. Likewise, the state can be viewed as the current blood glucose (and the meal that the patient is going to eat –as it will also have an effect on the patient's next state). The amount of insulin that the patient is going to inject is considered as an action. For the rewards we need a measure that shows how well the blood glucose is controlled between one decision point (meal) and the next. The transitions can be thought as the way the body of the patient behaves after having a meal and taking the injection.

An example will help to clarify what we mean by *transition* in the diabetes world. Let us assume that the patient's current blood glucose is at a normal level, and that the patient is going to eat a normal meal. If the patient takes her normal insulin injection, then at the time of her next meal, her blood glucose level will be around the normal value. Whereas, if she takes more insulin, she will probably end up with lower blood glucose at her next meal.

Continuing vs Episodic

There are many differences between the grid world and the diabetes world. One of these is the continuing nature of the diabetes problem. In RL, tasks are usually either *episodic* or *continuing*. An episodic task, as in the case of the grid world, is a task in which the interaction between agent and environment consists naturally of multiple sequences (each of which is called an *episode*). For example, an episode in the grid world as mentioned starts at the S tile, and ends at the G tile. In contrast, the diabetes problem, which is a continuing task, is a task in which the agent-environment

interaction is like a single chain of events and does not have natural episodes (it does not end as long as the patient is diabetic and alive).

Another difference is the type of information that the agent receives about the environment. Depending on the type of information that an agent receives from the environment, we can have a fully observable environment or a partially observable environment (in this case we have a partially observable MDP or a POMDP). In a fully observable environment, the agent receives a Markov state S_t at every time-step; otherwise it only gets a partial observation of the true state. In the diabetes world, the true Markov state probably includes many variables representing various factors in the patient's body, ranging from her blood pressure to the concentration of various hormones in her system and her brain activity. Not all of the above factors are observable (measurable), thus in reality the patient only receives a partial observation of the current state of her body.

MDP vs POMDP

Although we only have partial observations of the true state of the patient's body, we might still have enough information in our observations to guide the patient to maintain her blood glucose within an acceptable range. For instance, the state space that diabetologists use consists only of current blood glucose and the amount of carbohydrates that the patient is going to eat, and this state is informative enough to be used by patients in real life. Although the diabetes world is a non-Markovian environment, we assume that we have full observations of the state and thus have an MDP.

In the Grid World example we explained that the Markov property holds if the future is independent of the past given the current state. Here is a formal definition: the Markov property holds if and only if:

$$Pr(S_{t+1} = s', R_{t+1} = r|s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, ..., r_2, s_1, a_1) = Pr(S_{t+1} = s', R_{t+1} = r|s_t, a_t)$$

Thus far we have employed the grid-world example to introduce sequential decision-making and MDP. Next we motivate the use of RL in the diabetes domain using two more examples. The first example is a control problem in which the goal is to balance a pole on a cart. This example is a good analog for balancing blood glucose in diabetes. In the second example, we illustrate one of the subtleties of the diabetes world.

2.2.4 Two additional RL examples and their similarity to the diabetes task

One of the commonly used examples in the reinforcement learning literature is the *pole-balancing* problem. Pole-balancing is a good example of a control task, where the goal is to keep the value of a parameter (angel of the pole) within a specific range. In that sense, the task of controlling the blood glucose level within a certain range is very similar to the pole-balancing problem.

The goal of the pole-balancing problem is to balance a pole (an inverted pendulum) that is connected to a cart by a pivot. The cart can only move along a horizontal rail, and the pole is only free to move about the horizontal axis of the pivot. The state of the system is defined by four real

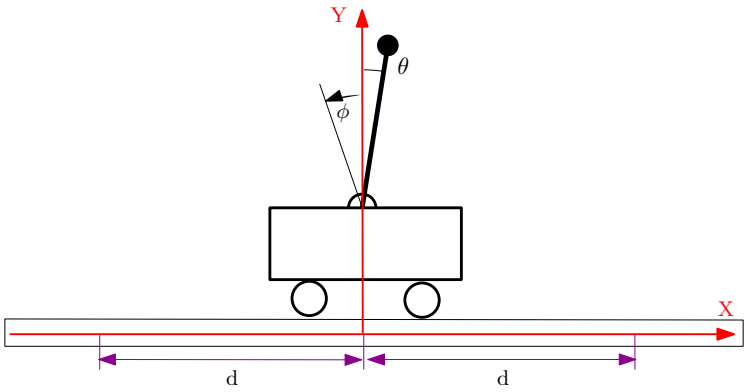


Figure 2.5: The pole-balancing reinforcement learning problem, adapted from [67].

values: the angle of the pole θ, the angular velocity of the pole $\dot{\theta}$, the position of the cart relative to the centre of the track x and the velocity of the cart $\dot{x}$. The output of the control system (action) is a forward or backward force to the cart. The constraints of the system are: (1) the pole must remain upright with a deviation angle no more that $\pm\phi$ degrees (from vertical axis) and (2) the cart must remain within $\pm d$ of origin; see Figure 2.5. The reward for violation of the constraint is -1 and otherwise 0. Once the constraints are violated, the state is reset to a random valid state (which does not violate the system constraints).

There exists an analogy between the blood glucose control and pole-balancing example, in which the angle of the pole is analogous to the blood glucose level. There are also some clear differences between the two tasks. First, the dynamics of the diabetes task are more complex –*i.e.,* the number of parameters that impact the blood glucose level is larger– and many things can affect the blood glucose level of the patient (*e.g.,* the temperature of the environment, stress, etc.). In addition, there is a difference between actions that can be taken. In the diabetes control task, the only possible action for the controller is to inject insulin, which decreases the blood glucose level. In contrast, in the pole-balancing problem, the available actions are pushing left or right. Put another way, in the diabetes task, our actions are one-sided. Having only a one-sided action available is analogous to a version of the pole-balancing problem in which the cart is on a hill and its controller can only push the cart to the left. In this analogy, eating meals in diabetes task is very similar to the push-to-right action in pole-balancing, in that these respective controllers are incapable of taking either of these actions.

Another example of a reinforcement learning problem that can clearly show some of the burdens involved in glucose control is the *cliff-walk* problem [67]. The schematics of this problem can be seen in Figure 2.6

The cliff-walk world is very similar to the grid world. However, in the cliff-walk world there exists a "*cliff*", consisting of a few tiles with very negative rewards. The agent's goal is to arrive at the goal state without falling off the cliff (entering the cliff states). The cliff state(s) in this environment is very similar to hypoglycemia state in the diabetes problem. The closer one can get

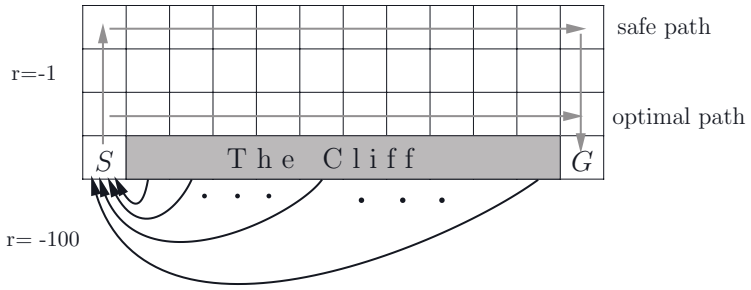


Figure 2.6: The Cliff-Walk problem adapted from [67]. Reward for each transition on the grid is -1 except on the cliff tiles where it is -100. The starting state is denoted by S and the terminal goal state by G.

to the cliff, the greater the reward will be. Yet one must be careful not to get too close to the cliff, because falling off the cliff, like succumbing to hypoglycemia, leads to devastating consequences.

Thus far, we have used some examples to familiarize the reader with reinforcement learning and to motivate its application to diabetes management. In the following section, we first formalize the concepts that we provided in the previous examples, and then introduce some additional reinforcement learning concepts. These concepts are the building blocks of our reinforcement learning methods, which will be introduced in Section 4.3.

We start by providing a formal description of the reinforcement reward in the diabetes problem.

2.2.5 Reward

Reward is defined for blood glucose measurements over a specific time interval. It shows how well blood glucose has been controlled in this time interval (*e.g.,* from breakfast to lunch).

We define two different forms of rewards based on our previously defined score function :

- mean-reward (over time):
$$\tilde{R}_{t+1} = \frac{\int_{\tau_t}^{\tau_{t+1}} score(BG_\tau)\, d\tau}{\tau_{t+1} - \tau_t}$$
- cumulative-reward:
$$R^{+}_{t+1} = \int_{\tau_t}^{\tau_{t+1}} score(BG_\tau)\, d\tau$$

where τ is the actual time (*e.g.,* 11am) at time-step t (*e.g.,* 3rd meal).

2.2.6 Policy

In the grid world example we described the policy concept and provided an optimal policy for that problem. Here we formalize the definition of the policy.

We can have two different forms of policy:

- Stochastic policy: $\pi(a, s) = Pr(A_t = a|S_t = s)$. This form of policy assigns a probability to each action given the current state.

Objective type/name	Objective formula
Discounted Sum	$J^{\gamma} \triangleq \sum_{t=0}^{\infty} \gamma^t R_{t+1}, \quad 0 \leq \gamma < 1$
Finite Horizon	$J^{T} \triangleq \sum_{t=0}^{T} R_{t+1}$
Average Reward	$J^{\infty} \triangleq \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T} R_{t+1}$

Table 2.1: Various forms of reinforcement learning objectives.

- Deterministic policy: Very similar to a stochastic policy, but the probability of selecting one of the actions is one and the probability for any other action is zero.

A good example of a deterministic policy, in the diabetes world, is the standard treatment policy that we introduced in Section 1.3.5. Note that each action is the amount to inject (Inj), which is deterministically specified by the policy parameters and user state.

In the following we formalize the goal of reinforcement learning.

2.2.7 Goal

The goal of reinforcement learning is to find a policy that selects actions in such a way that an objective function is maximized; this objective function is based on future rewards (will be discussed in Section 2.2.9). To accomplish this goal, an inexperienced agent must train its policy $\pi(a, s) = Pr(A_t = a|S_t = s)$, in a process that usually needs both to *exploit* its current knowledge about the environment (to acquire good rewards), and to *explore* the environment to learn about unidentified opportunities.

2.2.8 Epsilon-soft policy

As mentioned above, a reinforcement learning (RL) agent usually tries to explore the world to learn about the environment and potentially achieve a desirable performance. One of the very common exploration techniques that is used in the RL literature is ϵ-soft policy [78]. This technique creates an ϵ-soft policy, $\pi_{(\epsilon\text{-soft})}$ based on policy π by adding some exploration. At each step, an ϵ-soft policy selects actions at random with probability ϵ (exploration) and selects the actions from policy π with probability $1 - \epsilon$ (exploitation).

2.2.9 Objective function

The objective function, denoted by J, can have different meanings and forms. It may refer to the discounted sum of future rewards, the average of all future rewards, or the sum of the future rewards up to a finite horizon (see Table 2.1). Each of these formulations would have its own optimizing policy which might be well different from the others.

Based on the objective function, one can define a measure of goodness for states or state-action pairs, which will be discussed next.

2.2.10 State/state-action value function

State value function, or simply *value function*, $V^{\pi}(s)$, is defined as the expected objective (J) when the agent starts at state s and follows policy π. Similarly, *state-action value function*, or simply *action value function*, $Q^{\pi}(s,a)$, is defined as the expected objective (J) when agent follows policy π after taking action a at state s. Note that these definitions are based on the expectation on trajectories generated by policy π, which is denoted by E_{π}.

$$V^{\pi}(s) \triangleq E_{\pi}[J|S_1 = s]$$

$$Q^{\pi}(s,a) \triangleq E_{\pi}[J|S_1 = s, A_1 = a]$$

There exists a unique optimal value for each of these functions denoted by:

$$V^{*}(s) = \max_{\pi} V^{\pi}(s),\ \forall s \in \mathcal{S}$$

$$Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a),\ \forall s \in \mathcal{S}, a \in \mathcal{A}.$$

As in the case of Q^{*}, we can define $\pi^{*}(s,a) = \operatorname{argmax}_{\pi} Q^{\pi}(s,a)$. This optimal policy is the policy whose corresponding action value function ($Q^{\pi^{*}}$) would be Q^{*}. Note that this policy is not necessarily unique.

Next we discuss different ways that the value function can be represented.

Function approximation

In problems with discrete and small state spaces, one can represent the value function with an array or matrix, which contains the state-value $V(s)$ of each state ($s \in \mathcal{S}$). This representation method is known as the *tabular* representation. In most real-life problems, however, the state space is usually very large or continuous. In these problems, it is not possible or feasible to use a tabular representation, since the number of elements in the table of these value functions could be very large (or in the case of a continuous state space, infinite).

One way of dealing with the above issue is to use function approximation. The goal of function approximation is to find a more compact representation that approximates the value function. In function approximation one first needs to define a set of features $\varphi(S)$ for state S. Once this feature vector is defined one can approximate the value function with a predictor. As an example, in linear approximation, the value function can be approximated using a weight vector ω as: $V(S) \approx \varphi(S)^{\top}\omega$.

Discretization (or *tiling*) is one of the linear function approximation methods most commonly used in RL literature [67]. This method gets its name from the fact that it discretizes the state space into disjoint areas, which are called *tile*s. Using this method, an approximate state is assigned to each tile (tile index) and this state is the representative of all the actual states that fall into the same tile; see Figure 2.7. In this method a feature vector is defined using a binary vector and the tile index

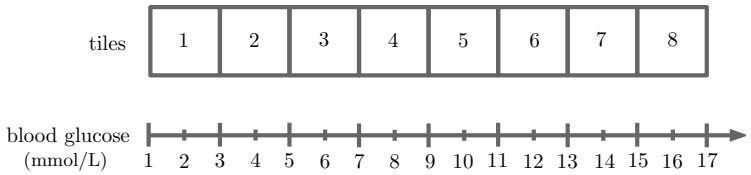


Figure 2.7: An example of discretization on blood glucose.

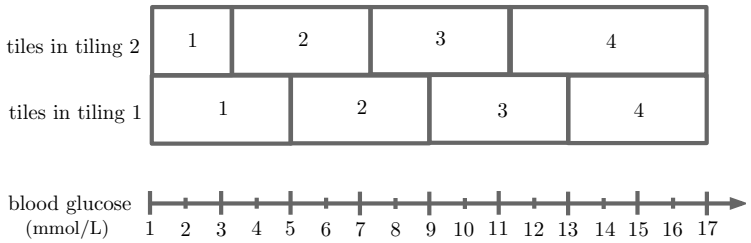


Figure 2.8: An example of tile coding with 2-tilings on blood glucose.

i. Given that the number of tiles is n, we create a binary vector with n elements, where the i-th feature in this vector is one and the rest of the values are zero.

$$\varphi_{\text{discretization}}(S) = [\overbrace{0 \cdots 0}^{i-1} 1 \overbrace{0 \cdots 0}^{n-i}]$$

For example, in Figure 2.7, we discretize the blood glucose values of interval $[1, 17]$ into $n = 8$ tiles. In the example presented in this figure, all blood glucose values between 1 and 3 mmol/L fall into the tile 1, and all are represented by feature vector [1,0,0,0,0,0,0,0]. Similarly blood glucose values of 10 and 10.5 are located in tile 5 and consequently have a feature vector of [0,0,0,0,1,0,0,0], where the 5th bit is one.

By adjusting the sizes of the tiles, we can obtain the desired compactness in the representation. Furthermore we can obtain generalization over the state space using *tile-coding* (that is, through the use of more than one tiling) with larger tiles. In tile-coding, different tilings have different offsets (so that the tilings overlap). Figure 2.8 is an example that shows how two tilings can be used. In this example, we have four tiles in each tiling. The tile-coded feature vector of blood glucose of 12 mmol/L in this case is [$\overbrace{0,0,1,0}^{\text{tiling-1 features}}$, $\overbrace{0,0,0,1}^{\text{tiling-2 features}}$], as the value of 12 mmol/L for blood glucose is in tile 3 of the first tiling, and in tile 4 of the second tiling (tiling 2). For more information on tile-coding see [67]. A very similar argument can be used to address the action space. Consequently we can also use function approximation to approximate the action value function $Q(s, a)$.

In the following we describe two commonly used classes of reinforcement learning algorithms that help an agent to find an optimal policy. We use one method from each of these categories in Section 4.3 to address the diabetes problem.

2.2.11 Value-based reinforcement learning methods

These methods are based on *value functions*. If we have access to Q^* then a simple form of optimal policy can be obtained from the following greedy policy (substitute Q with Q^*):

$$\pi_{(greedy)}(s,a) = \begin{cases} 1, & \text{if } a = \underset{a'}{\text{argmax}}\, Q(s,a') \\ 0, & \text{otherwise} \end{cases}.$$

Value-based methods consist of many iterations of the following two phases: *policy evaluation* and *policy improvement*. In the policy evaluation phase (*e.g.,* at iteration i), the goal is to come up with an estimate of the current policy's action value function (Q^{π_i}). In contrast, in the policy improvement phase of iteration i, the goal is to come up with a slightly better policy by using $\pi_{(greedy)}$ of Q^{π_i}. As these iterations proceed, under certain conditions Q^i converges to Q^* [67].

In model-free RL, the policy evaluation phase is usually done by sampling trajectory/trajectories of (state, action, reward, state) quadruples achieved by following the policy that is going to be evaluated. One of the standard methods for policy evaluation estimation is *Temporal Difference learning* (TD-learning) [66]. TD-learning is a general method for solving prediction problems that uses previous predictions as targets for next predictions during the learning process (this is called *bootstrapping*).

One of the value-based RL algorithms that we utilize is Sarsa. This algorithm will be introduced in Section 4.3.

2.2.12 Policy-based methods

In contrast to value-based methods, policy-based methods do not directly use the (action-)value functions to select actions [67]. These methods might still calculate the value function or action-value function, but these might only affect the policy indirectly (*e.g.,* to update the policy). Actor-critic methods are one of the methods in this category; these methods are discussed in Section 4.3.2.

In the next chapter we introduce our experimental platform, the type-1 diabetes simulator.

Chapter 3

T1DM Simulator

In this chapter we introduce our diabetes evaluation platform. We examine and evaluate our algorithms on *in-silico* patients simulated by Type-1 Diabetes Mellitus Simulator (T1DMS). We described the reasons behind selecting this specific simulator for our purposes in Section 1.4.3. In the following we briefly introduce the T1DM-simulator.

T1DMS is a commercial software created by "The Epsilon Group"[1] to simulate the metabolic system of type-1 diabetic patients. This simulator is implemented in Simulink®/Matlab® and uses the mathematical model of Dalla Man et al. [21]. This *in-silico* model is the first (and so far the only) FDA-accepted tool to substitute animal trials in pre-clinical testing of new treatment strategies in type-1 diabetic patients. T1DMS provides a set of *in-silico* patients that can be used to measure and compare the effectiveness of different proposed treatments.

The science behind T1DM simulator is the result of more than a decade of research done by Dr. Claudio Cobelli and his team from the University of Padova, Italy, and of their collaboration with researchers at the University of Virginia, USA [28, 11, 75, 47, 19, 20, 73, 22, 21, 40, 31].

The T1DM simulator is a Matlab® program consisting of multiple subsystems. These subsystem are mostly implemented in Simulink. Each of the subsystems has its own compartments and potentially its related differential equations. As T1DMS is not an open source software, we do not have access to most of the implementations of its subsystems.

3.1 Prerequisites to simulation

In order to use the T1DM simulator, we need to specify the following options [31]:

1. The simulation period: how long the patient should be simulated (e.g. 5 days).
2. The patient-ID (PID) of the *in-silico* patient which is going to be simulated. The simulator comes with 30 *in-silico* patients. A list of patients and their parameters is provided in Appendix A.

[1] http://tegvirginia.com/

3. The meal information for the whole simulation period, consisting of exact meal amounts and their associated times[2].

4. The injection controller block.

Figure 3.1 shows the main view of the T1DMS model in Simulink. This view shows various blocks of the simulator. The block that is indicated by a large light-blue vertical rectangle in the middle is the heart of the simulator. This block is a black box[3] that models the human body. The rest of the blocks are white boxes[4] and are responsible for simulating the glucose sensor, insulin pump, and (most importantly) the controller.

The controller block, the orange horizontal box in the bottom of the figure, is responsible for decision-making related to control of the blood glucose level of the simulated patient. This is the block that decides about the time and amount of insulin injections. It is our job to fill the contents of this block (with our algorithms).

Figure 3.1 shows that the controller receives a number of input signals and produces output signals accordingly (all links in Simulink are in the form of signals –that is, of time varying variables). The input signals consist of (1) the current glucose reading that is supplied to the controller by the glucose-sensor block (this block simulates the noise in the glucose meter readings); (2) the current insulin injection (previously advised pump rates); (3) time-of-day, the current simulated time of day for the *in-silico* patient (e.g. 18.5 would be the signal's value at 6:30pm); (4) the meal announcement signals, including one that specifies meal times and another that shows the size of the meal. The outputs of the controller block are (1) the suggested basal pump rate and (2) the suggested bolus pump rate.

A more complex version of the controller can also be implemented. This version offers the option of advising users about carbohydrate consumption by providing a signal showing the grams of carbohydrates per minute for food intake. We did not use this option but it will be discussed in Chapter 6.

The controller can process its input signals once every minute[5], and produces a subsequent output signal. In other words, the controller measures the information about the state of the patient (blood glucose, meal consumption, and time-of-day) and then recommends an appropriate action (injection).

3.2 Complications, modifications, and add-ons

In the process of using T1DMS we faced several problems that needed special solutions. Here we provide a list of these issues and our proposed solutions.

[2] Another meal-related information is the meal duration, which we fixed to 20 minutes in our experiments
[3] Black box blocks are blocks that we do not have access to their implementation
[4] White box refers to the blocks that their implementation is available to us
[5] This is the time scale that is observable to the *in-silico* patient.

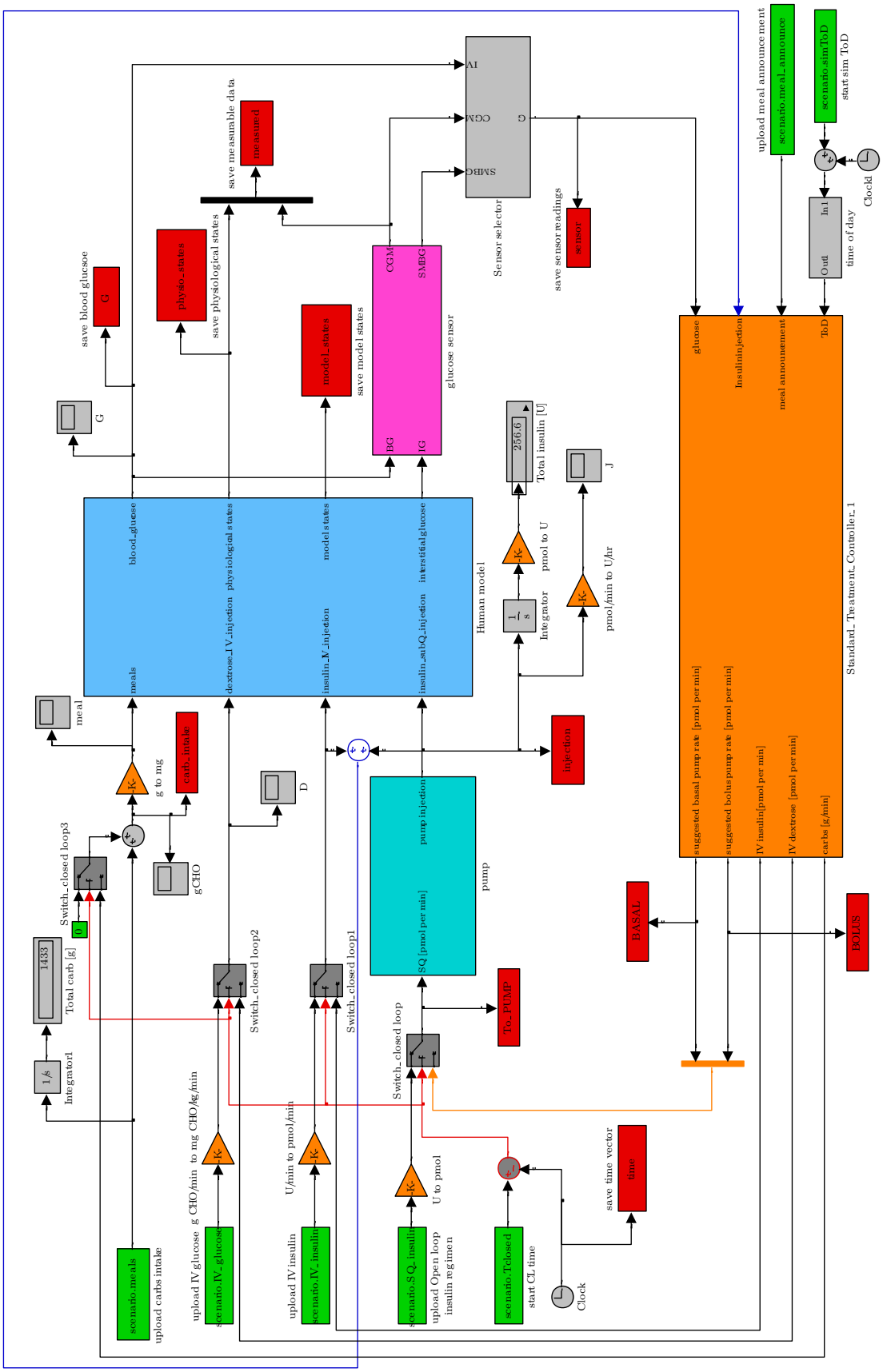


Figure 3.1: Simulink view of the glucose-insulin simulator system of the T1DMS adapted from [31]

.

3.2.1 Hypoglycemia

The first issue that we had with the T1DMS was the lack of any information about hypoglycemia symptoms. No such information is provided by T1DMS, either at run time or after the simulation. If the blood glucose of the patient goes very low, then the simulator gets into a state in which none of the actions suggested by the controller, nor the meal signals, will change the state of the *in-silico* patient. This could be an issue for the learning algorithms (as one would expect the blood glucose level to rise after eating a meal). Furthermore, the simulator can stay in this terminal state for a very long time, which wastes CPU time. The way we dealt with this issue was to command the Simulink to stop the simulator if the blood glucose readings got below a certain threshold (2 mmol/L).

3.2.2 Finite simulation time

There is a limit to the simulation period. This limit is basically posed by the hardware memory limit of the workstation that runs the simulator. This limit in our system was about 100 simulation days.

3.2.3 Continuation

When the simulator stops, either due to the Hypoglycemia (see above) or because the simulation period is over, it is not possible to continue simulation of the patient past this point (*e.g.,* in another run)[6]. Therefore whenever a continuous run is desired, *e.g.,* during a learning task, if the simulator stops, we restart it and assume that the second simulation is a continuation of the first. Although these restarting events happen rarely (*e.g.,* once in 100 days), one needs to be careful about handling them in the learning algorithm.

3.2.4 History buffers

In order to provide some information about the history of events to the controller we needed to add various buffers inside the controller module. These buffers store historical information related to the three following signals: sensor blood glucose readings, insulin injections, and meal signals.

3.2.5 Glucose sensor and noise

As mentioned above, the simulator has a few blocks that simulate the continuous glucose sensor. These blocks are capable of simulating various types of sensors: intravenous (IV), sub-cutaneous, and Self-Monitoring of Blood Glucose (SMBG). We use the sub-cutaneous sensor, which is the standard type of continuous blood glucose sensor that diabetic patients use in real-life. The simulator adds some noise to the blood glucose signal that the controller receives (to simulate the real life noise in blood glucose sensors). To reduce the noise in blood glucose readings, we pre-processed

[6]The simulator's internal state variables are only accessible at the end of simulation, and these variables are read-only at the beginning of the simulation. In order to make sure that we could not initiate the simulation from the end of another run, we contacted the T1DMS developers. They told us that in order to have full access to these (internal) variables, one needs access to the protected code (which is not available to us). Additionally changing these variables might necessitate further verification of the resulting model. Due to these issues we decided not to follow this path.

the blood glucose signal inside our controller by applying a moving-average filter (over a window of 100 minutes).

Chapter 4

Methods

This chapter presents our formulation of the diabetes control problem and the methods we used to address this problem. Section 4.1 describes the problem and Section 4.2 presents T1DC, our algorithm based on supervised learning. Finally, Section 4.3 provides details about two reinforcement learning algorithms that we use to automate the diabetes treatment-policy adjustment.

4.1 Formulation of the problem

This section introduces our formulation of the diabetes blood glucose control problem. First, some concepts are introduced, including meal and injection scenarios as well as patient options for blood glucose measurement and insulin injection. Section 4.1.4 then defines the blood glucose controller concept. This dissertation uses the vocabulary used in T1DMS [31] to name these concepts so that they correspond with the terms used by the software to simulate a patient with a specific patient-ID[1] (PID). Finally, Section 4.1.5 details our experimental settings.

4.1.1 Meal scenario

To be able to manage diabetes more easily, diabetic patients try to eat very similar meals every day. In real life, achieving this goal is not easy (*e.g.,* it is almost impossible to eat your usual meal at a party). In contrast, for *in-silico* patients, it is easy to assume that similar meals are consumed every day.

A *daily meal plan* specifies the size and time of daily meals. For example, the daily meal plan of a simulated patient can be as follows: three meals per day at 8am, 12am and 7pm, with the associated carbohydrate consumptions of 30, 50 and 70 grams. To make the daily meal plan more realistic, we may add some stochasticity to the time and the size of each meal. For example, in a more natural scenario, a lunch contains about 50 grams of carbohydrate and is eaten sometime around noon. Formally, this stochasticity can be added using probability distributions to create a generative model for lunch. An example is when the patient's lunch time is normally distributed

[1] List of *in-silico* patients can be found in Appendix A.

around noon, (*e.g.,* $N(\mu = 12\text{am}, \sigma = 30 \text{ minutes})$), and consists of about 60 grams of carbohydrate (*e.g.,* normally distributed $N(\mu = 60, \sigma = 10)$). Similar generative models can be used for breakfast and supper. Details of the actual daily meal plan that we used in our experiments will be discussed in Section 4.1.5. This daily meal plan can be used to generate the sequence of meals that an (*in-silico*) patient consumes over a period of time (*e.g.,* k-days). We call this randomly generated sequence the *meal scenario*.

4.1.2 Injection scenario

An *injection scenario* describes how a diabetic patient takes insulin through insulin injections. As explained in Chapter 1, type-1 diabetic patients use two different types of insulin: Basal and Bolus. Each of these types has its own injection scenario:

Bolus injection scenario

For the short-acting insulin, two different types of injection scenarios exist: **standard** and **continuous**. In the standard injection scenario, injections happen prior to each meal, while in the continuous injection scenario insulin injections can happen at any moment (*e.g.,* once every few minutes). The latter scenario can be used only by patients who are using an insulin-pump (at the time of writing this dissertation, an insulin-pump costs about $7000).

Basal injection scenario

Basal injection scenario is also categorized into two types: **standard** and **continuous**. In the standard basal injection scenario, long-acting insulin is used for basal injections. However, in the continuous basal injection scenario, continuous infusion of rapid-acting insulin is used.

4.1.3 Glucose monitoring

Glucose monitoring can be done in two ways:

- Discrete monitoring
- Continuous monitoring

In discrete monitoring, patients sample their blood glucose level a few times a day, usually two to eight times. To measure the blood glucose level, patients prick their skin, usually around the fingertip, with a lancet and then put the resulting small drop of blood on a test strip. The test strip is then inserted into a blood glucose meter, which measures the level of blood glucose. These samples are usually gathered right before meals and sometimes after meals (e.g. 2 hours after the meals).

In continuous glucose monitoring, patients use a *Continuous Glucose Meter* (CGM) device [52]. This device has a glucose sensor that is placed under the skin and measures blood glucose every minute or every few minutes. These readings can be easily exported to a computer.

4.1.4 Controller

A controller is a device or algorithm that monitors and controls a dynamical system [81]. In this thesis, the controller is controlling the blood glucose level by deciding the amount of insulin injections needed at any moment[2].

Inputs to the controller can be temporal features such as:

- History of previous blood glucose readings [3]
- History of previous meals (time, duration, size)
- History of previous insulin injections (time, amount)

and/or other clinical information about the patient (e.g. body weight, body mass index (BMI), age, gender, etc.)[4].

As the controller controls the blood glucose level by specifying the amount to inject, it needs to be defined in conjunction with an *Injection scenario*.

4.1.5 Experimental setup

As mentioned in Chapter 1, we use the T1DM simulator, described in Chapter 3, for all of our experiments. The first part of our experimental setup describes the inputs to T1DMS (introduced in Section 3.1). The first input is the meal information for the duration of the simulation. Using the approach that is discussed in Section 4.1.1, we generate a meal scenario based on the following daily-meal-plan:

Meal	Time	Size (grams of carbohydrate)
Breakfast	$TN(7, .5, 6, 8)$	$TN(40, 10, 20, 60)$
Lunch	$TN(12, .5, 11, 13)$	$TN(50, 10, 30, 80)$
Supper	$TN(18, .5, 17, 19)$	$TN(70, 10, 50, 90)$

where TN denotes a truncated Gaussian distribution, which is a bounded version of Gaussian distribution: $TN(\mu, \sigma, a, b)$ defines a truncated Gaussian distribution with mean μ and standard deviation σ, whose values are bounded to interval $[a, b]$ (a Gaussian distribution conditioned on the random variable being in the interval) [80].

The second input is the injection scenario, in which we use **continuous basal** in conjunction with **standard bolus** injections. For glucose monitoring we use a continuous glucose meter. While this setup may look arbitrary at first, there is a reasoning behind this choice. We could not choose standard basal injection as T1DMS is unable to simulate long-acting insulin. This forced us to use the insulin pump for basal injections. However, we still had the choice of using a standard scenario for bolus injections.

[2] Remember that there exist a specific controller module in the T1DM simulator with the same role (see Chapter 3).
[3] For more details refer to Section 4.1.3.
[4] Of course if the inputs to the controller are instantaneous readings, we can still assume that temporal data can be extracted from this instantaneous data within the controller itself (*e.g.,* using a buffer).

This raises an important question: why would we want to use a standard scenario for bolus injections, given that we are using a continuous scenario (insulin pump) for basal injections? Fortunately, according to our diabetologist[5], if a patient is using the pump solely for basal injections, it is not hard to come up with the right settings for the standard basal scenario. Thus, our decision to combine continuous basal and standard bolus injection scenarios enables us to switch back to the full standard injection scenario if this is desired at a future time (*e.g.,* with a more sophisticated simulator or in a clinical study).

Our choice of a continuous blood glucose monitor is based partly on the fact that it helps us to define a performance measure. While the CGM does not cost as much as an insulin pump, it provides an immense amount of information about the variations and complex patterns of blood glucose. Although this amount of raw information might be overwhelming for the human brain to analyze[6], this certainly is not the case with a computer.

We simplified the diabetes control problem in various ways. In this thesis, we address only the bolus injection control problem and leave the basal injection control to be addressed in a future study. For the basal injections, we used the optimal basal infusion rates that are provided by the creators of T1DMS. Second, we had reason to believed that children's diets as well as their sensitivity to blood glucose fluctuations might significantly differ from those of adults and adolescents [26]. Thus, to further simplify our problem, we removed children from our study, and used only 20 *in-silico* patients that model adolescents and adults.

Apart from the above simplifications, due to the limited speed of the simulator it was not possible to extensively tune some of the parameters for the methods that we introduce in this chapter. The extensive tuning becomes even less feasible if we have multiple parameters to tune; for example there are three parameters in one of our reinforcement learning algorithms. As a result, we only tried a handful of randomly picked parameters. Consequently there is room for future comprehensive studies to address the parameter tunings.

Here we explain the setting in which each algorithm works. T1DC is very similar to the policy treatment adjustment cycle that is introduced in Section 1.3.4. T1DC is a supervised learning method that automates the treatment adjustment cycles. T1DC is trained using data from a population of patients, to learn the effects of a given policy on the diabetes-diary. After the training, T1DC can suggest a policy adjustment for a new patient by observing her diabetes diary. T1DC is more like an open-loop diabetes control system.

The reinforcement learning methods are meant to be used in a different setting. Our RL methods are more similar to the closed-loop diabetes control systems. The health of all of our *in-silico* patients is very important to the algorithm (even during the learning phase[7]), we want them to have

[5]Here and below, this refers to Dr. Edmond A. Ryan (Department of Medicine, University of Alberta, Edmonton, Canada)

[6]For example, three months' worth of minute-by-minute readings is about 130,000 blood glucose readings.

[7]This is in contrast to the T1DC's case, where we are only concerned about the patients in the test phase – *i.e.,* there is no penalty for going hyper- or hypo-glycemic during the training phase.

the least number of hypoglycemia events possible. If such an event happens, the patient is assumed to be disconnected from our system, being taken care of by a specialist, until her health situation gets back to normal and she is reconnected to our RL system. We simulate this process by stopping the simulation and restarting it. In the following section, we discuss our first treatment adjustment algorithm.

4.2 Supervised algorithm, T1DC

The T1DC algorithm uses supervised learning to predict the best policy parameters. The policy is assumed to be in standard policy form – Equation 1.1. Policy Form

T1DC uses extracted temporal features (from blood glucose readings) of a patient to predict the treatment policy parameters that would work best for her. The main assumption is that these temporal features contain information that helps to describe the hidden physiological properties of patients, by specifying the intrinsic properties of the patient's body and her blood glucose regulatory system (the properties include such factors as liver properties, sensitivity to insulin, the total volume of the blood in the patient's body, etc.). If we had access to all of these hidden properties of our *in-silico* patients, we could use them as input to our policy, and also to the learning process that would allow us to identify the appropriate policy. But even if we did have access to those values, we would not want to use them; because in real life, these hidden properties are very hard to measure without extensive testing of the patient, and are generally inaccessible. However, we would still use easily obtainable clinical information about the patients (such as, body-weight, blood glucose readings, age, etc.).

To be able to apply T1DC on new patients, we need to train the algorithm based on information acquired from a fixed set of patients (training set). During the training phase, a learner uses the training set and returns a predictor. At this point T1DC can be used in an iterative policy adjustment cycle for a new patient: the patient follows an initial policy, π_{θ_0}, for k-days, meaning that we simulate the patient for this period of time, using injections based on π_{θ_0}. Then we extract some features (called *temporal features*) from the diabetes diary of the patient (data that are observed over these k-days). The input features of the predictor consist of these temporal features as well as some clinical features. The output of the predictor is the estimated optimal policy parameters θ_1. When the predictor provides an output, the above cycle can be repeated by starting with the policy from the newly estimated policy parameters, π_{θ_1} (instead of starting with π_{θ_0}).

We should note that the T1DC algorithm is not directly concerned with blood glucose control performance, but only tries to predict the optimal policy parameters as accurately as it can (on its training patients). T1DC assumes that the optimal parameters will provide the best possible blood glucose control, and the closer the predicted parameters get to the optimal parameters, the better the blood glucose control performance will be. Yet, in practice, we do not care about the parameters as much as we care about the actual performance of the predicted policies in action. In our experimental

results (Section 5.1.2), we provide both of these evaluations and discuss this difference further.

Given the above overview of the algorithm, we are ready to provide the details of the T1DC algorithm. First we introduce the input and output feature vectors, and then we discuss the learning and testing phases.

Optimal policy parameters

The prediction label of the T1DC is the optimal policy parameter(s) $\theta^{\dagger}$. As we are using supervised learning, we have to have access to the optimal policy parameters. This information is available for each of the *in-silico* patients, as it has been provided by the creators of the T1DM simulator. We denote patient p's optimal policy parameters by ${}^{p}\theta^{\dagger}$.

The input features of the T1DC algorithm are $\varphi = [\varphi_{\odot}, \varphi_{\text{patient}}, \theta]$, where:

- $\varphi_{\odot}$: Temporal features (described next)
- φ_{patient}: Clinical features (body weight and age for this *in-silico* patient)
- θ: Current policy parameters

4.2.1 Temporal features

After simulating k-days of patient life by following policy π_{θ_0}, we first form the ζ_t (row) vector, which is formed using the blood glucose readings obtained at each of the meal times during the simulation [8]:

$$\begin{aligned}\zeta_t = \; & [bg_{(\tau_t - 4 \text{ hr})}, bg_{(\tau_t - 3 \text{ hr})}, bg_{(\tau_t - 2 \text{ hr})}, bg_{(\tau_t - 1 \text{ hr})}, bg_{(\tau_t)}, \\ & bg_{(\tau_t + 1 \text{ hr})}, bg_{(\tau_t + 2 \text{ hr})}, bg_{(\tau_t + 3 \text{ hr})}, bg_{(\tau_t + 4 \text{ hr})}, bg_{(\tau_{t+1})}]\end{aligned}$$

Note that $bg_{(\tau_t - 1 \text{ hr})}$ is the blood glucose level at an hour before t-th meal, and the last feature, $bg_{(\tau_{t+1})}$, is the blood glucose level at next meal and (by definition) is not available for the last t; thus ζ_t is formed for all of the meals except the last one. For example if $k = 3$, then we will have eight ($\overbrace{3}^{\text{meals}} \times \overbrace{3}^{\text{days}} - \overbrace{1}^{\text{last meal}} = 8$) ζ_t vectors , $t \in \{1, 2, \cdots, 8\}$. ζ_1 corresponds to the breakfast of the first day, ζ_2 is related to first day's lunch, etc. Then we form ζ, by stacking up ζ_ts[9]:

$$\zeta = [\zeta_1; \zeta_2; ...; \zeta_8]$$

In this example, ζ is a 8×10 matrix.

The *temporal feature vector* is then formed by aggregating some statistics over all ζ_t vectors:

$$\varphi_{\odot} = [mean(\zeta),\ std(\zeta),\ min(\zeta)]$$

[8] Note that here $t \in \mathbb{N}$ is the meal-index, while $\tau_t \in \mathbb{N}$ is the actual meal-time from the beginning of the simulation (in minutes).

[9] We are using Matlab®'s notation for concatenation. If $A = [1, 2, 3]$ and $B = [3, 4, 5]$ are two row vectors then $C = [A; B]$ is a 2×3 matrix, with A being its first row, and B its second row.

where :

$$mean(\zeta) = \frac{1}{N} \cdot \sum_{t=1}^{N} \zeta_t \tag{4.1}$$

$$std(\zeta) = \sqrt{\frac{1}{N} \sum_{t=1}^{N} \big(\zeta_t - mean(\zeta)\big)^2} \tag{4.2}$$

$$min(\zeta) = \Lambda \in \mathbb{R}^{1\times 10} : \ \Lambda_j = \min_i \zeta_{(i,j)} \tag{4.3}$$

Note that in Equation 4.1 and Equation 4.2 operations are all element-wise. In Equation 4.3 $\zeta_{(i,j)}$ refers to the element in the ζ matrix that is in the i-th row and j-th column; this equation is calculating the minimum along columns (the $min(\zeta)$ would have the same size as the ζ_t –*i.e.,* 1×10). In the above example, N is eight.

4.2.2 Training phase

To generate samples for patient p_T, who is in the training-set ($p_T \in P_T$), we generate 300 random policy parameters ${}^{p_T}\theta^i$, $i \in \{1, 2, \ldots, 300\}$. For each of ${}^{p_T}\theta^i$ policy parameters, we run the simulator for k days and extract temporal features $\varphi_{\circledcirc}^{{}^{p_T}\theta^i}$. We can train a predictor by having the temporal features, policy parameters, and clinical features (as input features), and optimal policy parameters of patients ${}^{p_T}\theta^\dagger$ (as output features); see Figure 4.1. We use two separate support vector regressors, one for each of the policy parameters (CR and CF). As mentioned in Section 2.1.1, we use the SVM-Torch [16] implementation of support vector regression. The following parameter setting is used for SVM-Torch: $C = 1000$; Gaussian Kernel with $\sigma = 5$; and $\epsilon = .5$ for CR and $\epsilon = .1$ for CF predictor. We did not do an extensive tuning of the parameters. We used a large C (regularization constant) as we had only 20 patients, and we tuned σ and ϵ on one of the patients (PID=1) and used these for all patients.

4.2.3 Testing phase, iterative use

Let us consider patient p_t from the set of test patients ($p_t \in P_t$). We choose a random initial policy for this patient, ${}^{p_t}\theta^0$, and then use this policy to simulate p_t for k days. Then we extract the temporal features from the simulated data and form the $\varphi_{\circledcirc}^{{}^{p_t}\theta^0}$. Given the temporal features, we can create the complete input feature vector $\varphi^{{}^{p_t}\theta^0} = [\varphi_{\circledcirc}^{{}^{p_t}\theta^0}, \varphi_{\text{🧍}}^{p_t}, \ {}^{p_t}\theta^0]$ and use the predictor. The predictor provides us $\widehat{{}^{p_t}\theta^{\dagger_1}}$, which will be our next policy parameters to follow (*e.g.,* ${}^{p_t}\theta^1 = \widehat{{}^{p_t}\theta^{\dagger_1}}$). We can further continue the policy update for more iterations; see Figure 4.1.

4.2.4 Further details

If policy parameter prediction errors are negative ($\widehat{\theta} > \theta^\dagger$ e.g. $\widehat{CR} > CR^\dagger$) then the amount of bolus insulin injection will be higher than the optimal dosage. This type of error can potentially cause hypoglycemia. As explained in Section 1.3, hypoglycemia can have serious complications

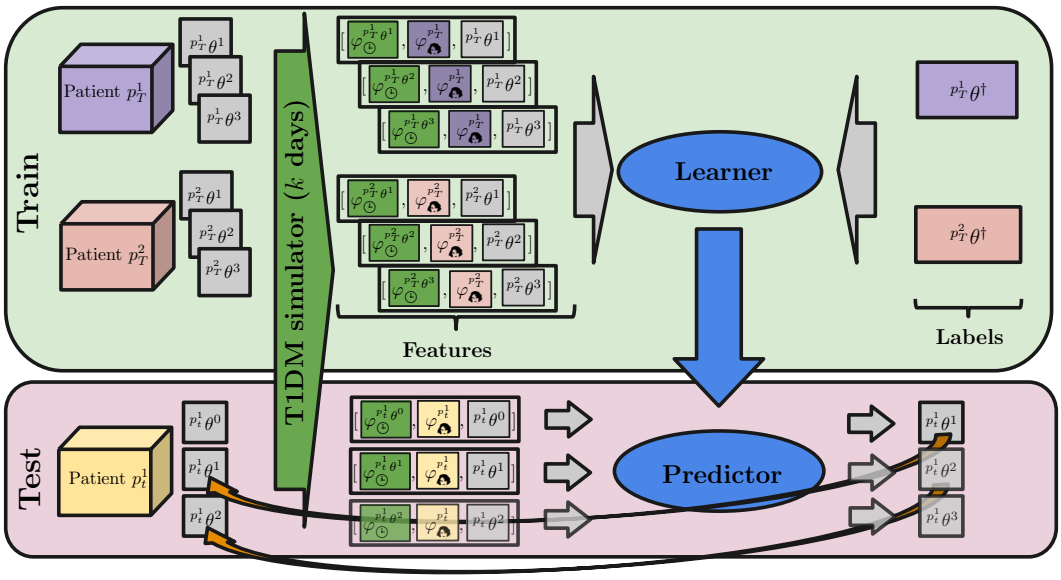


Figure 4.1: T1DC algorithm: In the training phase a learner and predictor is trained based on two patients, p_T^1 and p_T^2, using three sample policies for each patient. In the testing phase the learned predictor is used to do three iterations of policy update on patient p_t^1.

and it is crucial that we prevent it. This suggests the prediction loss assigned to positive errors should be higher than that assigned to negative errors.

To address the above issue, instead of changing the predictor (which is discussed in Section 6.2.6), we slightly modified the iterative phase in the algorithm. In each iteration (*e.g.,* i-th iteration) when the next policy parameters are calculated by the predictor ($\widehat{\theta}_i$), we compare them with the previous policy parameters (θ^i), and if the changes in the parameters are negative ($\Delta\theta_i = \widehat{\theta}_i - \theta^i < 0$), then instead of using $\widehat{\theta}_i$ as the next policy parameters, we use $\theta^{i+1} = \theta^i + \alpha\Delta\theta_i$, where $0 < \alpha \leq 1$ (in our implementation $\alpha = 0.7$). We should emphasize that this modification is only applied to the iterative process if $\Delta\theta_i < 0$. We provide empirical evidence to support this modification in Section 5.1.1.

4.2.5 Initial policy

We used two different means of initializing the policy:

- Safe initialization
- Smart initialization

In safe initialization, we use the safest policy for the population. This means that the policy parameters are chosen as θ^{max} (introduced in Section 1.3.5). Note that this policy is safe in the sense that it involves injecting a very small amount of insulin (as θ_1 and θ_2 are denominators in the standard policy formula).

In smart initialization, we use the formula suggested by Wolpert [83] which uses the patient's body weight to initialize the θ (we use his 1800 rule for sensitivity and 500 rule for carbohydrate

ratio).

After consulting with our expert diabetologist, we realized that a reasonable period of time for the policy to show its effect on the glucose reading patterns is about one to two weeks. Based on this suggestion we set $k = 9$ in the T1DC algorithm.

Evaluation over one hundred days (EOH)

To evaluate the performance of T1DC policy, the policy is followed for 100 days and then the mean-reward of the resulting blood glucose values is calculated. This mean-reward is used as our evaluation criterion. Using this measure we can compare two controllers. The larger the mean-reward, the better the controller. The blood glucose values used in the above evaluation are the actual blood glucose values provided by the simulator (not the noisy values given to the learning algorithm; see Section 3.2.5) provided by the simulator[10].

4.3 Reinforcement learning algorithms

In this section we describe two reinforcement learning (RL) algorithms and detail how we utilized them to address the diabetes control problem.

The following sections describe the temporal difference control algorithm Sarsa (Section 4.3.1), and Policy-Gradient (Section 4.3.2) and Actor-Critic (Section 4.3.2) methods; and discuss how we utilized them.

4.3.1 Sarsa: on-policy temporal difference control

The Sarsa algorithm [56] is an on-policy temporal difference (TD) control algorithm. It uses TD-learning to predict the action-value function (Section 2.2.10) by observing transitions from one state-action pair to the next. Sarsa's update rule for the Q function, in a discounted reward setting (J^γ –see Section 2.2.7), is given by:

$$\delta_{t+1} = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \tag{4.4}$$

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \delta_{t+1}, \tag{4.5}$$

where $\alpha \in [0, 1]$ (learning rate) and $\gamma \in [0, 1]$ (discount factor) are parameters of the algorithm, and $A_t = a_t, S_t = s_t, S_{t+1} = s_{t+1}, A_{t+1} = a_{t+1}, R_{t+1} = r_{t+1}$, and $A_{t+1} \sim \pi(S_{t+1}, \cdot)$, $S_{t+1} \sim \mathcal{P}(S_t, A_t, \cdot), R_{t+1} \sim \mathcal{R}(S_t, A_t)$ –see Figure 2.4.

Sarsa is created by combining the above update rule with an ϵ-*greedy* policy, which is an ϵ-soft greedy policy based on the action value function; see Algorithm 1. This algorithm is executed in each state and the Q is updated after each step.

[10]Remark: During the EOH process, we ignore any hypoglycemic event that occurs and continue to follow the policy until the end of 100 days of simulation. This way, if blood glucose is low, then the mean reward will be low, and if the patient suffers a severe hypoglycemia event then the blood glucose value will stay low; consequently, the mean reward will be much lower.

Algorithm 1 The Sarsa algorithm

function Sarsa(S, A, R, S', A', Q)

Input: S is the previous state, A is the previous action chosen (*e.g.,* by ϵ-greedy policy), R is the immediate reward received when transitioning to S', where action A' is chosen. Q is the current estimation of the action value function.

1: $\delta = R + \gamma \cdot Q(S', A') - Q(S, A)$
2: $Q(S, A) = Q(S, A) + \alpha \cdot \delta$
3: **return** Q

One can use linear function approximation for the action value function in the Sarsa algorithm to obtain Algorithm 2.

Algorithm 2 The Sarsa algorithm with linear function approximation

function SarsaLinFuncApprox(S, A, R, S', A', θ)

Input: S is the last state, A is the last action chosen (*e.g.,* by ϵ-greedy policy), R is the immediate reward received when transitioning to S', where action A' is chosen. $\theta \in \mathbb{R}^d$ is the parameter vector of the linear function approximation.

1: $\delta = R + \gamma \cdot \theta^\top \varphi(S', A') - \theta^\top \varphi(S, A)$
2: $\theta = \theta + \alpha \cdot \delta$
3: **return** θ

Implementation details

We implemented Sarsa with linear function approximation, which is outlined in Algorithm 2. Mean-reward (Section 2.2.5) is used as the reward signal for Sarsa. In the following, we describe our action space, action value function approximation, and the exploration methods used in our experiments.

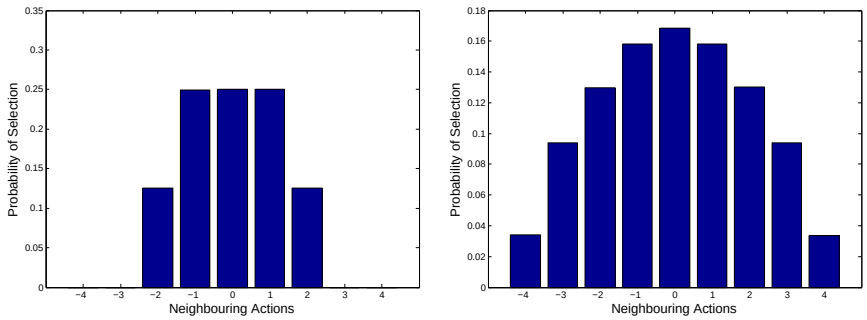
It is more natural to consider the insulin injection amount as a continuous variable, but the Sarsa algorithm requires actions to be discrete. Thus, we discretized the insulin injection range into bins. We used 20 units as the maximum amount of insulin injection[11].

The state in our experiments usually consists of current blood glucose value and size of meal that is going to be consumed (in grams of carbohydrate). As we did in the case of action, we discretized these continuous variables.

To present our action value function, we either simply discretize the states/actions (similar to tabular representation –Section 2.2.10) or we use tile-coding. In the next chapter, a table containing this information accompanies each result. If the number of the tilings in the accompanied table is one, then the discretization is used, otherwise the tile-coding representation is used. In both of the aforementioned representations, the state-action value function Q is initialized pessimistically (with zeros).

Tile-coding on action space is done by considering action as a new dimension in the state space (and then using the regular tile-coding on the space-action pair). In this case, to find the best performing action at a given state, our policy evaluates all of the possible actions on the joint tile-coding

[11] This matches the maximum injection dosage that the optimal T1DMS policies would advise on our 20 *in-silico* patients.



(a) UN exploration. (b) TG exploration.

Figure 4.2: Comparison between our two local exploration methods. The zero on the horizontal-axis corresponds to the candidate action.

(of states and actions) and selects the action that provides the highest Q value.

Due to some properties of the diabetes problem, we do not use the standard exploration methods in our RL implementations. In the diabetes control domain, exploration is risky, as some of the actions may harm the patient. Common exploration methods, such as ϵ-greedy, are problematic because they have a non-zero chance of selecting any action. We avoid the risk of taking a harmful action by only allowing exploration in a limited neighbourhood around the best action. We use two exploration methods, uniform (UN) exploration, and truncated Gaussian (TG) exploration.

In UN-exploration, the action is initialized greedily based on Q: $a_{(temp)} = \underset{a}{\text{argmax}}\, Q(s,a)$. Then a uniform random number from range $[-2, 2]$ is added to $a_{(temp)}$. Thereafter, $a_{(temp)}$ is rounded to the nearest integer and then it is truncated to the acceptable action range. The resulting number is the action –that is, the amount of insulin that is injected. TG-exploration is done very similarly, except the random number is selected from a truncated Gaussian distribution rather than from a uniform distribution. This truncated Gaussian distribution has the following form: $TN(\mu = 0, \sigma = 2, -4, 4)$, where the last two parameters define the truncation interval [-4,4]. The above process essentially provides a local exploration around the best candidate action, by selecting the neighbouring actions of the candidate action ($\underset{a}{\text{argmax}}\, Q(s,a)$) with some probability; see Figure 4.2.

More details about the parameter settings and function approximation are provided in Section 5.2.

4.3.2 Actor-critic methods

Actor-Critic (AC) methods are one of the policy-based methods that are introduced in Chapter 1. In an actor-critic method, two main components work closely together. The first component is the *actor*, which is responsible for making actions and improving the policy. The second component is the *critic*. It is usually a state-value function that helps the learning process by evaluating (criticizing) the current policy. The critic evaluates the current policy based on effect of its selected action on current state. Using the state-value function, the critic can estimate whether the actor's choice of

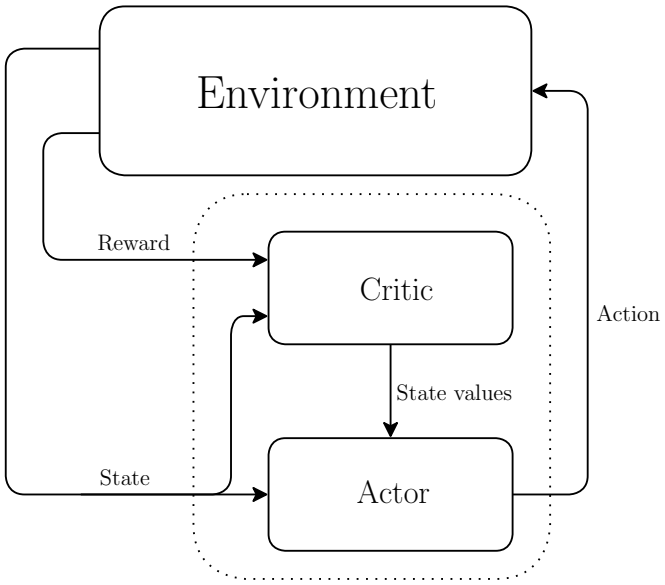


Figure 4.3: Schematic of an actor-critic agent (dotted box) in interaction with the environment; adapted from [69].

action is leading the agent to a better state or not [67, 69]; see Figure 4.3.

The separation of actor and critic provides some advantages that makes actor-critic methods favourable for our task. One of these advantages is their ability to select actions more efficiently if the action space is continuous or large. In the mentioned value-based RL methods, choosing the best action needs a search through action space to find the action that has the highest action-value. This process becomes infeasible if the number of actions grows large. In contrast, if we have an explicit form of policy in the actor, action selection might become easier and more feasible. For example, selecting actions using the standard policy can be done by a simple calculation. This calculation is based on policy parameters and current state, and involves no search.

In actor-critic methods the policy is usually selected to be stochastic. This stochasticity not only enables exploration, but it also provides the ability to deal with continuous actions for free (when it is combined with the policy-gradient methods). In the following section, we introduce policy-gradient methods, one of the methods that can improve the policy and can thus be used as an actor [69].

Policy-gradient methods

Policy-gradient methods perform a gradient ascent on the performance surface induced by a policy class. Let us assume that we are considering the class of linear policies that are parametrized by θ, π_θ, where $\theta \in \mathbb{R}^{d_\theta}$. Let us also assume that the objective function is the average-reward $J^\infty(\pi)$. In this setting, the goal of the policy-gradient method is to find the policy parameters θ^* that maximizes the performance of the policy –*i.e.,* that maximizes the objective function $J^\infty(\pi)$.

$$\theta^* = \underset{\theta}{\text{argmax}}\ J^{\infty}(\pi_\theta)$$

This is usually done by stochastic gradient ascent – that is, by updating θ approximately proportional to the performance gradient:

$$\theta_{t+1} \approx \theta_t + \alpha_\theta \nabla_\theta J^{\infty}(\pi)$$

In order to do the stochastic gradient ascent, we need to reformulate $J^{\infty}(\pi)$ to be able to calculate the gradient with the help of the policy gradient theorem. First we make an assumption that facilitates the reformulation, then we describe the reformulation and move on to the theorem.

Stationary Distribution

We assume that $d^{\pi}(S) = \lim_{t\to\infty} Pr(S_t = s|S_0 = s_0, \pi)$, the stationary distribution of states under π, exists and is independent of S_0 for all policies [12]. The performance of a policy, $J^{\infty}(\pi)$ can then be written as:

Performance of Policy

$$J^{\infty}(\pi) = \sum_s d^{\pi}(s) \sum_a \pi(s,a) R(s,a) \tag{4.6}$$

Simply put, if all of the states of the MDP (that represents the environment) are reachable using the current policy π, then d is a probability distribution over all of the states and shows how probable it is that the agent is in a specific state at any time, when following the policy π. Now, let us describe Equation 4.6. The left-hand side $J^{\infty}(\pi)$, which was defined (in Table 2.1) as $\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} R_{t+1}$, is the expected average reward gained by following policy π. On the right-hand side, according to the definition of d, an agent is in state s with probability $d^{\pi}(s)$, and in that state the agent has a probability $\pi(s,a)$ of taking action a (as the policy π is being followed). By definition, the expected reward that the agent will receive by performing action a in state s is $R(s,a)$. Thus the right-hand-side is a weighted sum over all states and a weighted sum over actions, of the expected reward of doing those actions in those states. This is essentially the expected reward that an agent will get if it follows policy π; thus the right-hand-side is equal to the left-hand side[13].

Policy Gradient Theorem

The policy gradient theorem [68] uses this formulation of the performance and proves that :

$$\nabla_\theta J^{\infty}(\pi) = \sum_s d^{\pi}(s) \sum_a \pi(s,a) \Big(\nabla_\theta \ln \pi(a,s) \cdot \Big(Q^{\pi}(s,a) - V^{\pi}(s) \Big) \Big)$$

Note that there is no term of form $\nabla_\theta d^{\pi}(s)$ in the gradient. As we are the ones who choose the form of policy (and as a result, its derivative), this theorem paves the way to calculating the gradient and to optimizing the performance of the policy from samples gathered during the execution of a policy.

Policy gradient as an actor

Policy gradient method can be used to implement an actor in an actor-critic method. For a few reasons we chose the policy of the actor (of our actor-critic method) based on the standard policy.

[12] This puts some restriction on MDPs involved, regarding the chain of states that result from following any policy, *i.e.,* Markov chain must be ergodic; see [44].

[13] This simple explanation is meant to provide some insight into the reformulation; for a formal description see [68].

Before we provide the reasons underlying our decision, we first remind the reader that the standard policy (introduced in Section 1.3.5) has the following form:

$$\pi_\theta(a,s) = \pi_{(CF,CR)}(a, \overbrace{\quad}^{[bg,\, carbs]}_{s}) = \begin{cases} 1 & \text{if } a = \dfrac{1}{CF} \cdot (bg - C_{target}) + \dfrac{1}{CR} \cdot carbs \\ 0 & \text{otherwise} \end{cases}$$

where :

C_{target} : target blood glucose value constant

CF : Correction Factor (sensitivity to insulin)

CR : Carbohydrate Ratio

There are three reasons for choosing this form of policy. First, the standard policy proved its capabilities in practice. Second, it is easy to interpret the evolving parameters during and after optimization (parameters have biological meanings –*i.e.,* sensitivity of blood glucose to carbohydrate consumption, and to insulin injection). Last, it is a form of policy that patients are familiar with, which could be an advantage for a potential clinical study. In the following we describe the policy for our actor-critic implementation.

We created our policy by first bounding the range of parameters using a logistic function. This bounding is used to ensure that the policy parameters remain in a sensible range at all times. As mentioned in Section 1.3.5, this sensible range for policy parameters is assumed to be:

$$CR \in [3, 30] \rightarrow 1/CR \in [.03, .3]$$

$$CF \in [.4, 2.8] \rightarrow 1/CF \in [.35, 2.5]$$

Incorporating the range for the parameters, our stochastic policy is defined as:

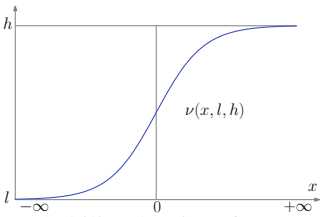
$$\pi_\theta(a,s) = N(\mu_\theta(s), \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} exp\Big(-\frac{(a-\mu_\theta(s))^2}{2\sigma^2}\Big)$$

where :

$\mu_\theta(s) = (bg - C_{target}) \cdot \nu(\theta_1, 0.35, 2.5) + carbs \cdot \nu(\theta_2, 0.03, .3)$

$\nu(x,l,h)$: logistic function bounded by (l,h) *i.e.,* $l + \dfrac{h-l}{1+e^{-x}}$

σ : exploration rate



The compatible features $\nabla_\theta \ln \pi(a,s)$ can be calculated based on the probability density of our policy as:

$$\nabla_\theta \ln \pi(a,s) = \frac{\nabla_\theta \pi_\theta(a,s)}{\pi(a,s)} = \frac{a - \mu_\theta(s)}{\sigma^2} \times \frac{\partial \mu_\theta}{\partial \theta}$$

$$\mu_\theta(s) = (bg - C_{target}) \cdot \left(0.35 + \frac{2.5 - 0.35}{1 + e^{-\theta_1}}\right) + carbs \cdot \left(0.03 + \frac{.3 - 0.03}{1 + e^{-\theta_2}}\right)$$

$$\frac{\partial \mu_\theta}{\partial \theta_1} = (bg - C_{target}) \frac{2.5 - 0.35}{(1 + e^{-\theta_1})^2} e^{-\theta_1}$$

$$\frac{\partial \mu_\theta}{\partial \theta_2} = carbs \cdot \frac{.3 - 0.03}{(1 + e^{-\theta_2})^2} e^{-\theta_2}$$

Implementation details

Here we first describe an actor-critic algorithm that aims to maximize the long term average-reward J^∞, then we describe a different version that we utilized in our experiments.

Algorithm 3 shows the actor-critic algorithm, which is slightly different from the original algorithm [7] in its J^∞ update rule (the first two steps of the algorithm). This version uses a more stable rule that was proposed to us by Richard S. Sutton and had previously been used in [23].

Algorithm 3 Average-Reward Actor-Critic algorithm with function approximation (MDP).

function ActorCritic($S_t, A_t, R_{t+1}, S_{t+1}, J_t^\infty, \omega_t, \theta_t$)

Input: S_t is the previous state, A_t is the last action chosen by policy π, R_{t+1} is the immediate reward received when transitioning to S_{t+1}; J_t^∞, ω_t, θ_t are the current estimations of average reward, value-function weights, and best policy parameters (respectively).

1: $\delta_t = R_{t+1} - J_t^\infty + \omega_t^\top \varphi(S_{t+1}) - \omega_t^\top \varphi(S_t)$
2: $J_{t+1}^\infty = J_t^\infty + \alpha_j \, \delta_t$
3: $\omega_{t+1} = \omega_t + \alpha_\omega \delta_t \varphi(S_t)$
4: $\theta_{t+1} = \theta_t + \alpha_\theta \, \delta_t \, \nabla_\theta \ln \pi(A_t, S_t)$
5: **return** $(J_{t+1}^\infty, \omega_{t+1}, \theta_{t+1})$

For the actor-critic methods, we used accumulative-reward, R^+ (introduced in Section 2.2.5). When we use R^+, the reward signal that the algorithm receives depends on the length of the reward's time-window. As the time-window between various meals can be different from day to day, the Markovian assumption would not hold unless we have a time-related feature in our state representation. Adding an extra feature to the state enlarges the state space and makes the critic's job harder (it has to learn a more complex function), which can potentially slow down the learning pace. In order to solve this issue without adding a time-related feature to our state features, we use SMDP formulation of the above actor-critic. Semi-Markov Decision Processes (SMDPs) are very similar to MDPs, but in SMDPs, the time window between time-steps can vary for each time-step (*e.g.*, $(\tau_{t_2} - \tau_{t_1}) \neq (\tau_{t_3} - \tau_{t_2})$). The full description of SMDPs and their average-reward formulation can be found in [54]. The average-reward formulation of the SMDP aims to maximize:

$$J_{SMDP}^\infty(\pi) = \lim_{T \to \infty} \frac{E_\pi[\sum_{t=0}^T R_{t+1}]}{E_\pi[\sum_{t=0}^T (\tau_{t+1} - \tau_t)]} = \lim_{T \to \infty} \frac{E_\pi[\sum_{t=0}^T R_{t+1}]}{E_\pi[\tau_T - \tau_0]}$$

Although this might look different from J^∞[14], they are very similar. J_{SMDP}^∞ is the average reward over time, whereas J^∞ is average reward over the time-steps. Their difference is only a nor-

[14] J^∞ is defined in Section 2.2.5 as $J^\infty = \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T} R_{t+1}$.

malization factor (the length of the simulation divided by the number of meals). Note that J^{∞}_{SMDP} is analogous to A1C, as both are averaged over time, one on blood glucose (A1C), and the other on scores of blood glucose values. The SMDP actor-critic algorithm (Algorithm 4) is very similar to the MDP actor-critic in Algorithm 3, differing only in step 1 of the algorithm (the TD-error calculation). We used this algorithm for our actor-critic. For further information about this algorithm and details about its derivation, see [45].

Algorithm 4 SMDP Average-Reward Actor-Critic algorithm with function approximation (SAC) [45].

function ActorCritic($S_t, A_t, R_{t+1}, S_{t+1}, J^{\infty}_t, \omega_t, \theta_t$)

Input: S_t is the previous state, A_t is the last action chosen by policy π, R_{t+1} is the immediate reward received when transitioning to S_{t+1}; J^{∞}_t, ω_t, θ_t are the current estimations of average reward, value-function weights, and best policy parameters (respectively).

1: $\delta_t = R_{t+1} - J^{\infty}_t(\tau_{t+1} - \tau_t) + \omega_t^{\top}\varphi(S_{t+1}) - \omega_t^{\top}\varphi(S_t)$
2: $J^{\infty}_{t+1} = J^{\infty}_t + \alpha_j\, \delta_t$
3: $\omega_{t+1} = \omega_t + \alpha_{\omega}\delta_t\varphi(S_t)$
4: $\theta_{t+1} = \theta_t + \alpha_{\theta}\, \delta_t\, \nabla_{\theta} \ln \pi(A_t, S_t)$
5: **return** $(J^{\infty}_{t+1}, \omega_{t+1}, \theta_{t+1})$

More details about the parameter setting of the algorithm and function approximation that is used in the critic is presented with the results.

Chapter 5

Experimental results and discussion

The previous chapter introduced the three algorithms that are used in our experiments and Section 4.1.5 provided details about our experimental setup. In this chapter, we discuss the experimental results obtained from those algorithms. In the following, we present and discuss the results from the T1DC algorithm (in Section 5.1), and the reinforcement learning algorithms (in Section 5.2). Finally, we provide a summary of our results and further discuss them (in Section 5.3).

5.1 T1DC results

This section describes the results of experiments using T1DC. We perform two different sets of analysis on these results. In the first analysis, we report T1DC's prediction error using cross-validation. For the second part, we compare the performance of the policies proposed for an *in-silico* patient by T1DC with that of the policies provided by a diabetologist.

5.1.1 T1DC prediction error

T1DC prediction error is calculated in a leave-one-out cross-validation setting (introduced in Section 2.1.2). The training data corresponding to each patient are obtained from $k = 9$ days of simulations of random policy parameters θ. For each patient, 300 random samples of θ are drawn from the inverse uniform distribution[1] in interval $[0.8 \times \theta^{\dagger}, \theta^{max}]$, where $\theta^{\dagger}$ is the optimal policy parameters[2] and θ^{max} is the maximum value the policy parameters are allowed to take[3].

Our rationale to use a minimum range of $0.8 \times \theta^{\dagger}$, as opposed to θ^{min}, is to avoid the complications of handling severe hypoglycemia. An example of such a complication occurs when we use close-to-θ^{min} policy-parameters to obtain sample training data. In such cases, the simulator usually stops after the first few meals due to severe hypoglycemia and does not make it to day 9. Consequently, the resulting temporal features are not complete and it is difficult to use these data for training purposes.

[1] Inverse uniform distribution is used because the parameters are in the denominator of the policy formula. The policy formula is introduced in Section 1.3.5.

[2] Optimal policy parameters are introduced in Section 4.2.

[3] $\theta^{max} = [CR^{max}, CF^{max}]$, where CR^{max} and CF^{max} are introduced in Section 1.3.5.

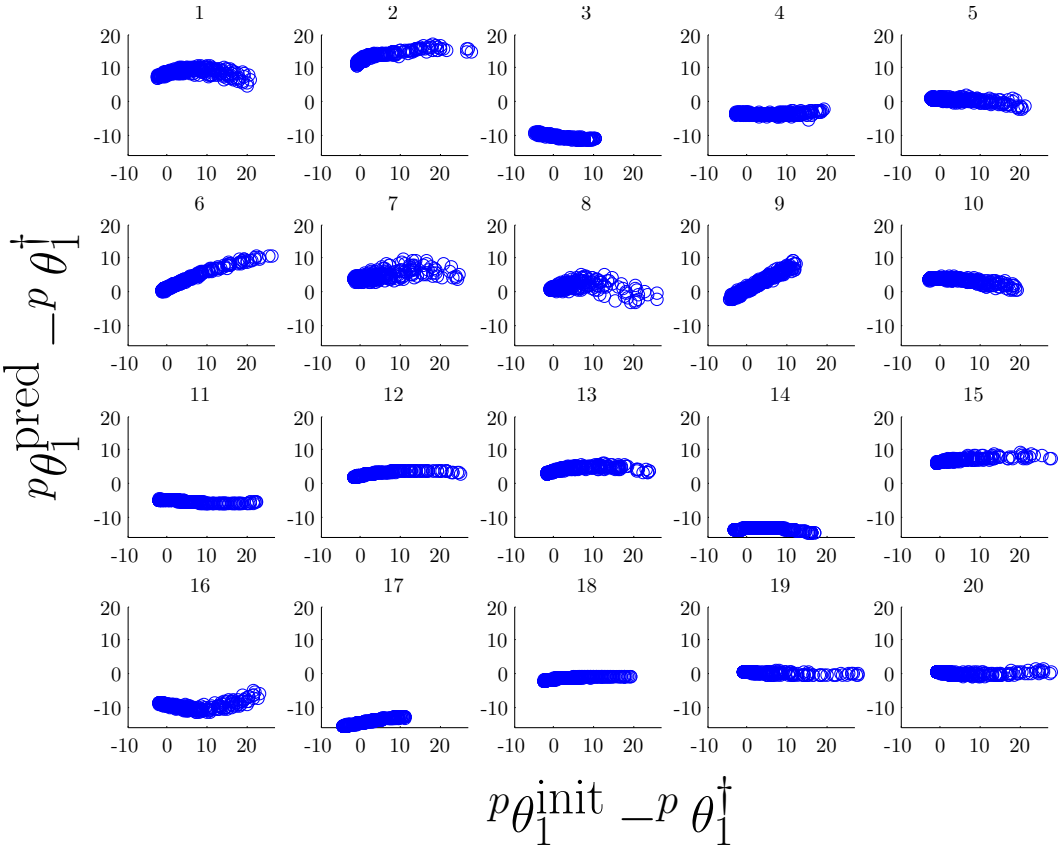


Figure 5.1: Distance of policy predicted parameter θ_1^{pred} (CR) from optimal parameter $\theta_1^{\dagger}$ after one-iteration of T1DC as a function of the distance of the initial policy from the optimal policy (horizontal axis). The former is the same as prediction error, and is calculated for all of the patients using leave-one-out cross-validation. The patient-ID (PID) is denoted by the superscript p to the left of θ.

In the first set of T1DC experiments, our goal is to analyze the effects of initial policy on the predicted policy in one iteration of T1DC. To study this effect, we plot the prediction error after one iteration (${}^p\theta_1^{\text{pred}}$-${}^p\theta_1^\dagger$) versus the distance of the initial policy (${}^p\theta_1^{\text{init}}$) from the optimal policy(${}^p\theta_1^\dagger$), which is ${}^p\theta_1^{\text{init}}$-${}^p\theta_1^\dagger$ (initial error).

Figure 5.1 shows the results of this experiment. For simplicity we only plotted results corresponding to θ_1 (CR). Performing the same analysis on θ_2 (CF) produced similar results. Each graph in this figure deals with a specific patient (whose id, p, appears above). For each point (x,y) shown, the x-value is the initial error of the θ_1 value (CR) – *i.e.,* the distance between this CR value and the optimal value $\theta_1^\dagger$, and the y-value is the prediction error of the CR value after one iteration – *i.e.,* the distance between the predicted CR value and the optimal value.

In this experiment, the desired predictor will have a prediction error near zero, independent of the initial policy. In other words, the ideal predictor will output nearly optimal policy parameters, using the data that are gathered by an arbitrary initial policy. Patient 20's plot is a good example of such desired qualities. Prediction errors for this patient are almost constant and close to zero across different initial values of θ_1. A less desirable predictor would not yield the optimal results in the first iteration, but would produce slightly more accurate predictions in each iteration. This trend is observable in patient 6's prediction results. For this patient, the predictions get more accurate with more iterations. In this case, in general we want the $|y| < |x|$, as this means that the error decreases with each iteration of T1DC.

To further clarify the results shown by patient 6's plot in Figure 5.1, we provide a step-by-step example of running T1DC on this patient. Knowing that the optimal parameter for patient-6 is ${}^6\theta_1^\dagger = 7$, given the initial parameter ${}^6\theta_1^0 = 35$, it is possible to infer the predicted policy parameter from the plot using the following procedure. Consider the data point on the far (top) right of the plot, with coordinates $[x = 28; y = 10]$, whose x coordinate shows ${}^6\theta_1^0 - {}^6\theta_1^\dagger = 35 - 7$ and whose y coordinate shows the error corresponding to the next iteration, 10. This means that the next parameter will be ${}^6\theta_1^1 = \text{next-error} + {}^6\theta_1^\dagger = 10 + 7 = 17$. If we followed this process based on the estimation of errors in the plot, the value of ${}^6\theta_1$ in following iterations would be about ${}^6\theta_1^2 = 12$, ${}^6\theta_1^3 = 9.5$, and eventually it would converge to ${}^6\theta_1^\dagger$.

In contrast to above examples, some of the plots shown in Figure 5.1 do not show any of the desirable trends. Patient-17 is a clear example of such a case, with ${}^{17}\theta_1^\dagger = 22$. Using the initial policy parameter of 30 (${}^{17}\theta_1^0$), the prediction error will be approximately -15, which results in ${}^{17}\theta_1^1 = {}^{17}\theta_1^\dagger - 15 = 7$. This value for the parameter is noticeably lower than the optimal value, which leads to a larger-than-optimal dose of insulin injection, and it could potentially lead to hypoglycemia. In Section 4.2.4 we suggested a modification to the T1DC's testing phase that would alleviate this kind of dramatic drop in the policy parameters.

In this experiment 52% of the prediction errors are less than the initial errors and 74% of the prediction errors are not farther than $.2 \times \theta^\dagger$ from the initial errors.

5.1.2 Comparing the performance of T1DC to that of an expert diabetologist

In the previous subsection, we analyzed the T1DC prediction errors. In evaluating the T1DC, what we are really concerned about is not how accurate T1DC predicts the parameters, but about the actual performance of the obtained policies. In this regard, we compare the performance of the policies proposed by T1DC with that of the polices that our expert diabetologist provided. The performance of each policy will be evaluated using evaluation over one hundred days (EOH)[4].

To be able to perform a fair comparison between T1DC and our expert diabetologist, we must provide both with the same initial knowledge about patients. As explained in Section 4.1.5, T1DC has access to the optimal basal values, which is not accessible to the diabetologist. Therefore, we design two experimental settings for the diabetologist. In the first experiment, the diabetologist has access to the optimal basal-infusion-rates of the patients, while in the second experiment, the diabetologist not only has to adjust the CR and CF, but also has to come up with the basal-infusion rates. Using these two experiments we can see how this additional information (knowing the optimal basal) can help the diabetologist in the policy adjustment process. Apart from optimal basal rates, in our experiments the diabetologist and T1DC both have access to similar patient information and similar amounts of data.

This set of experiments follows the same procedure described in T1DC's testing-phase (Section 4.2.3): the controller, either the diabetologist or the T1DC predictor, starts with an initial policy and the simulator runs this policy for $k = 9$ days. The controller then uses the information that has been gathered during these $k = 9$ days and updates the policy for the next iteration. Note that when T1DC is being used on a patient, it used the predictor that was trained on other patients; refer to cross-validation in Section 2.1.2.

Following the above process, each controller produces a sequence of policy parameters for each patient. The length of this sequence in our experiments is four (four update iterations). Each policy that is created using these parameters is then evaluated using the EOH method. To aggregate the performance of a controller for each iteration, we take the average of the EOHs over all patients except the patient-1, as we tune the parameters of the T1DC on this patient[5]. The results of this comparison is given in Figure 5.2.

This figure shows the evaluated performance of five different policy iteration sequences. Two of these sequences are obtained using the T1DC algorithm – one using the safe initialization of policy parameters, the other using smart initialization[6]. Two other sequences are obtained with the help of our expert diabetologist. The last sequence, which is shown by the dashed-line at the top of the figure, depicts the performance of the optimal T1DM policy for the patients. Error bars in the plot show standard deviation of performance over the 19 patients.

Some observations can be made about this figure. First, we see a significant difference between

[4] EOH is introduced in Section 4.2.5.
[5] Adding patient-1 does not cause any significant change to the plot.
[6] These initializations are introduced in Section 4.2.5.

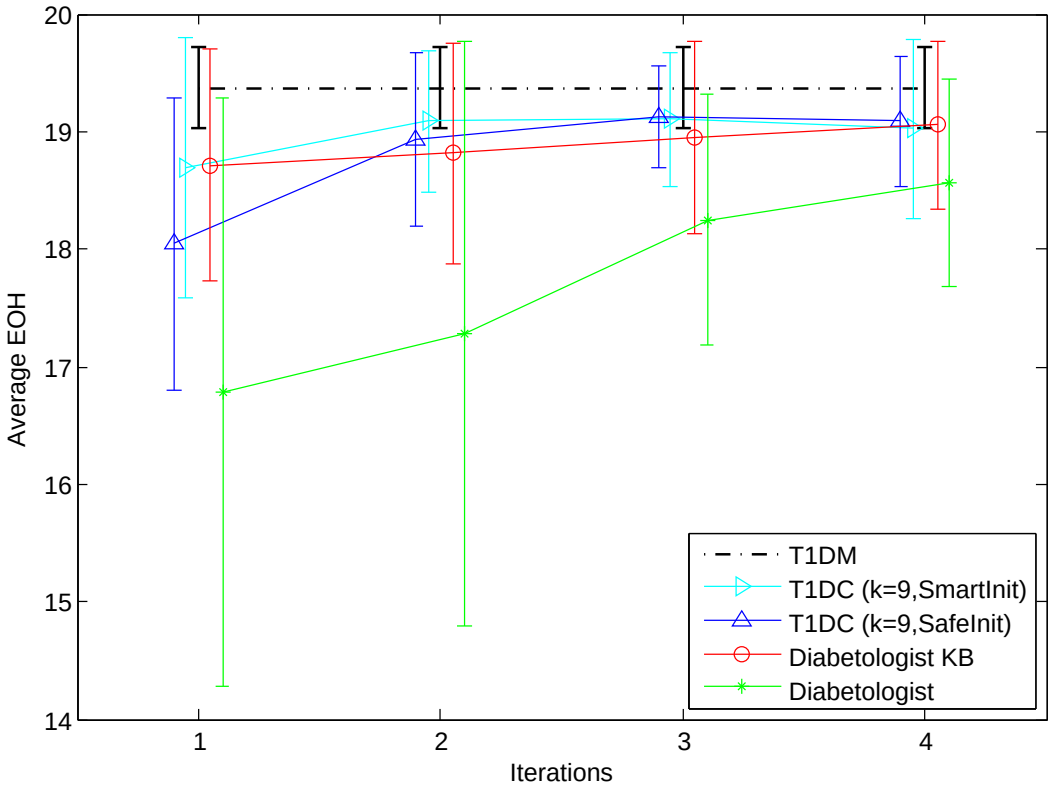


Figure 5.2: Comparison of average control performance of T1DC and diabetologist policies. The control performance is measured by averaging the evaluation over one hundred days(EOH) criterion over 19 patients (PID $\in \{2, \cdots, 20\}$). In the legend, KB stands for Knowing Basal.

the performance of the two diabetologist iteration sequences, which suggests that knowing the basal value significantly improved the performance of the diabetologist. This difference is significant in all of the iterations (based on paired t-test, p-value < 0.05). Second, for T1DC results, we see that smart initialization seems to provide a better starting performance, but it did not affect the performance of the other iterations. Finally we observe that the performance of the T1DC is comparable to the performance of the diabetologist (when diabetologist does know the basal values).

5.2 Reinforcement learning results

In this section, we present results and discussion for reinforcement learning methods. Section 5.2.1 first explains the naming convention used in this section. Section 5.2.2 and Section 5.2.3 then discuss the experimental results obtained from reinforcement learning algorithms.

5.2.1 Naming convention

Result and Setting Identifier

In order to easily distinguish the results, every result in this section is identified by a result identifier, which is the letter 'R' followed by a number, *e.g.,* R1, R2, etc. A result's identifier is also used to refer to the parameter setting that produces this result. Note that due to stochasticity in the algorithm and the simulator, the same parameter settings could produce different results in two runs. This is why we used the same parameter setting for several runs, each with its own identifier. Hence, two or more different identifiers can refer to same parameter setting. For example if R1 and R2 are two different runs of the same algorithm with the same parameters, then the settings for R1 and R2 are the same.

5.2.2 Sarsa

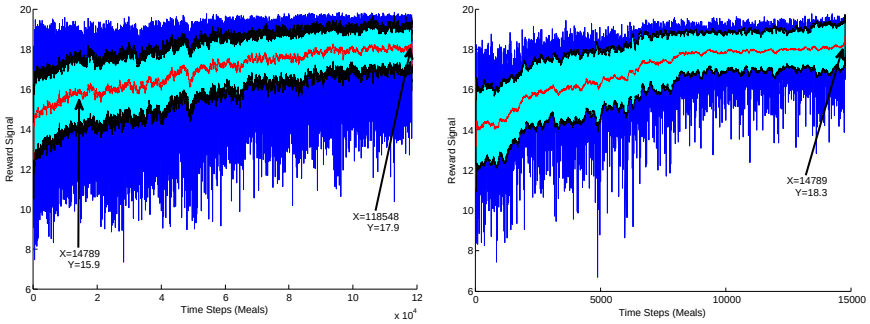
In this section we discuss results of the Sarsa algorithm, which was introduced in Section 4.3.1.

Our first attempts to run Sarsa were naive and involved discretizing features and feeding them to the algorithm. The outcome was not very good. In the second round of trials, we achieved substantial improvement by applying common practices, such as tile-coding and parameter-tuning. In the following we provide details about the settings of both runs and explain why one worked better than the other.

To emphasize the importance of tuning Sarsa, we provide a comparative analysis between Sarsa with naive settings and Sarsa with enhanced settings. Figure 5.3 shows the performance of two single runs of Sarsa with two different settings, R10 (left) and R84 (right). The y-axis in this figure denotes the average historical rewards ($\tilde{R}$) over the previous 200 time-steps. It is clear that R10 is slower than R84 – *i.e.,* the performance (historical rewards) achieved by R84 after about 15,000 meals[7] is not achieved by R10 even after 85,000 meals.

Below are the settings used to achieve R10 and R84 – panel (a) and (b) of Figure 5.3:

[7] As we model three meals per day, this means observing 15,000 meals would take more than 13 years.



(a) (b)

Figure 5.3: Comparing the performance of the Sarsa algorithm in two different settings. Blue is the actual reward signal, red is the average reward signal in a window of 200 meals, cyan is the standard deviation around the average. It can be seen that (a) did not achieve the performance of (b), despite using almost 7 times more samples. Note that the scale of the horizontal axis is different in (a) and (b).

R10	State Tilings #	Q-state	State Feature Range	State Tiles #	Sarsa-Parameters
	1	[bg,carbs]	[1,15],[0,100]	[15, 10]	$\gamma = .99, \alpha = .3$
	Action Tilings #	Actions	Action Range	Action Tiles #	Exploration
	1	[inj]	[1,20]	20	UN-exploration
R84	State Tilings #	Q-state	State Feature Range	State Tiles #	Sarsa-Parameters
	100	[bg,carbs]	[1,20],[0,100]	[8,4]	$\gamma = .1, \alpha = .001$
	Action Tilings #	Actions	Action Range	Action Tiles #	Exploration
	100	[inj]	[1,20]	4	UN-exploration

The main issue with the Sarsa-R10 settings is that there are $15 \times 10 \times 20 = 3000$[8] elements in the Q matrix and the algorithm only updates one of them at each meal. In this setting no generalization is performed over state space or action space. This makes the learning process very slow and inefficient. The setting we used in R84 varies greatly from the R10 setting, as R84 can generalize over both state and action spaces. In R84, we used tile coding with large tiles to generalize over state-action space. In addition, in this setting, we used a smaller γ.

Thus far we have analyzed the importance of the settings used for Sarsa. In the following, we provide a more detailed analysis of the effects of employing exploration methods as well as using generalization over actions on the performance of Sarsa. Exploration methods can take one of the two following forms: TG-exploration or UN-exploration (see Section 4.3.1). Regarding the actions, we either generalize over them (GE), or use discretized actions (NO –*i.e.,* no generalization-over-actions). Figure 5.4 shows three single runs for each of the four possible combinations of the above options. This figure is generated for patient 2 (PID=2).

These results suggest that the combination of TG-exploration and GE results in the best performance – see R88, R90 and R94 in the figure. The runs corresponding to UN+GE setting, R84, R86 and R92, show a very similar performance and arguably stand as the second-best option.

[8] As Q is represented in a tabular form, its size can be calculated by multiplying the number of possible states and actions. State space has two dimensions (bg, carb) that are discretized into 15 and 10 tiles respectively and action space is also divided into 20 tiles. Thus, the size of Q would be $15 \times 10 \times 20$.

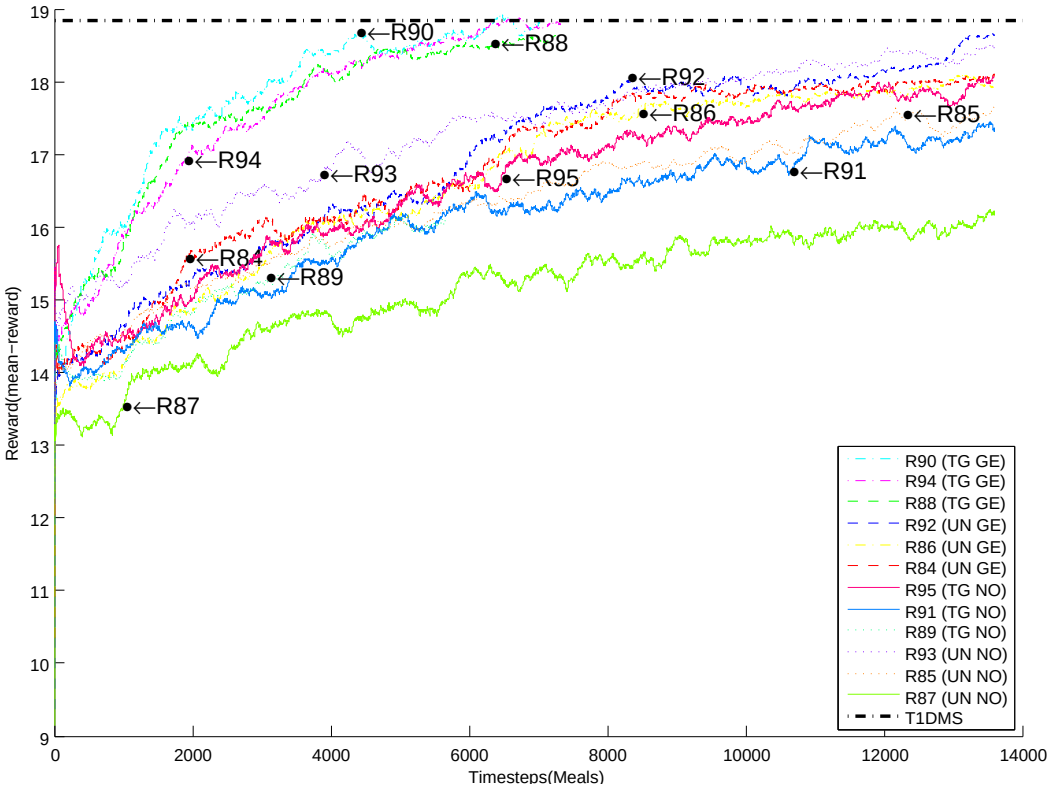


Figure 5.4: The effects of generalization over actions and different explorations on the performance of Sarsa. Each curve represents a single run (labelled with its result-id). The legend shows the exploration type for each result (either TG or UN) as well as if the Sarsa method used generalization (GE) over actions or not (NO). The y-axis shows the historical average reward signal over a window of 100 meals.

Unlike two above settings, the results of UN+NO settings are scattered and show a fair amount of variation from one run to another[9]. This stochasticity in the results could be due to the lack of action generalization, and to the combination of restricted exploration of UN-exploration (in comparison to TG-exploration) with pessimistic optimization. If the algorithm incidentally picks a few small actions at the beginning, this combination might cause the algorithm to get stuck in choosing small actions, and might consequently limit the performance. The selection of small actions may be due to poor exploration, which in our combination, would lead to larger Q-values for these (small) actions. As the Q-values of these small actions get larger and larger, it becomes even harder for the exploration method to induce the algorithm to pick larger actions. Hence this Sarsa will remain trapped in this situation until Q-values of these actions converge to their true values. Thereafter, even with the poor exploration, slightly larger actions will be selected and the algorithm will learn that these larger actions have larger Q-values[10].

A closer look at Sarsa results in Figure 5.4 suggests that this algorithm is capable of learning a near-optimal policy. This figure shows that Sarsa achieves an acceptable performance, which is comparable to the performance of the optimal policy. One drawback of using Sarsa (with our offered settings) is its speed. Our best performing Sarsa is still very slowly and takes more than 6 years of *in-silico* patient life to achieve a near optimal performance. Sarsa's laggard pace makes it impractical for real-life practice.

After observing that setting $\gamma = .1$ improves Sarsa's performance in some of our initial (not reported) experiments, we decided to evaluate Sarsa's performance with respect to γ. The effect of γ on learning can be seen in Figure 5.5, which shows that Sarsa with $\gamma = 0$ can achieve near optimal performance. This result implies that our formulation of the diabetes control problem can be solved using a one-step look-ahead approach – *i.e.,* where actions only affect the next state and have negligible effect on future rewards.

To explain this situation, recall that we are only considering the short-acting insulin, which is mostly active in the first 4 hours [76]. However, in our meal scenario, meals are about 5 hours apart. Therefore, it is reasonable to speculate that the action will not affect future rewards. Note that the problem will not be a one-step look-ahead problem if one wants to use a more realistic meal scenario where the patient may have one or two snacks between the main meals, or when the main meal times vary dramatically.

5.2.3 SAC

As mentioned in Section 4.3.2, we implemented Algorithm 4 (SAC) for our actor-critic method. The first step towards running this algorithm was to choose appropriate learning rates. So we started by looking at similar actor-critic algorithms in the reinforcement learning literature. Despite the fact

[9] This is pointed out to us by Micheal Bowling during one of our meetings.

[10] Unfortunately due to time constraints, this theory could not be validated in this dissertation; this will need to be done in a future study.

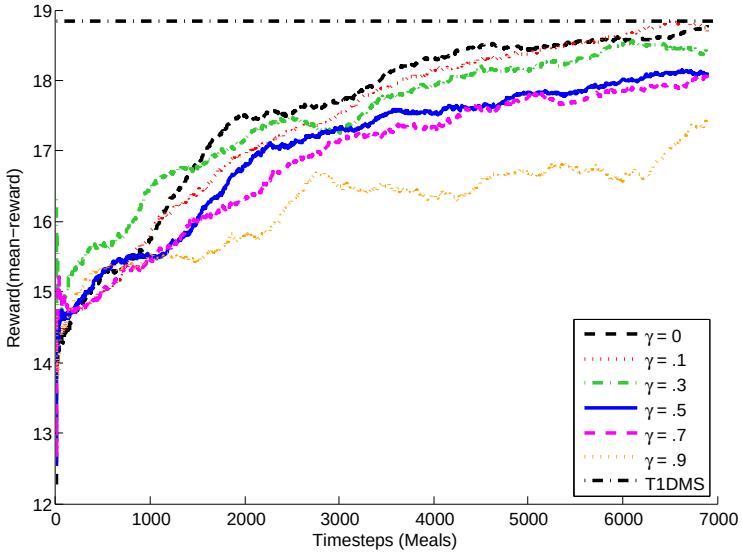


Figure 5.5: The effects of γ parameter on the performance of Sarsa. Each curve represents a single run.

that SAC is formulated on SMDPs, and the actor-critic algorithm of Degris et al. [23] is formulated on MDPs, they are still fairly similar. Therefore, we decided that this work (Degris et al. [23]) is a good candidate for our purpose. Using this paper, we tried to approximate the relative scale of our parameters towards each other (*e.g.,* α_j is 10 times smaller than α_ω, etc.).

Based on the estimated relative scale, we tried various parameter settings on two of the patients (PID=1,2), to determine good values for the parameters $\alpha_j, \alpha_\omega, \alpha_\theta$; this led to $\alpha_j = 10^{-5}$, $\alpha_\omega = 10^{-4}$, $\alpha_\theta = (10^{-5},\ 5 \times 10^{-5})$. The exploration parameter (σ) was set to 0.5. As mentioned earlier, due to the slowness of the simulator, comprehensive parameter selection needs to be addressed in a future study and the provided parameters should not considered optimal.

In addition to tuning the above-mentioned parameters, we needed to initialize SAC's variables. We initialize the average reward, J^∞ using the first observed reward. The other important set of variables is the starting policy parameters, which we initialized using the smart-initialization method (Section 4.2.5). To represent the state value function of the SAC's critic, we used the following features: current blood glucose, blood glucose an hour before meal, the size of the meal to be eaten (in grams of carbohydrates), and the meal-category (breakfast/lunch/supper). This state-value function is approximated using discretization or tile-coding.

We summarize the results of SAC algorithm in Figure 5.6, Figure 5.7, Figure 5.8 and Figure 5.9, which depict the evolution of different variables during the learning process. The first two figures in the series are obtained using discretization, and the last two are obtained using tile-coding. Each figure consists of 8 plots where each plot is indexed by its row (starting at 1 from the top row) and column (starting at 1 from the left column); for example, top-left plot is plot(1,1) and the bottom-middle plot is plot(3,2). Corresponding information for each plot is presented below. This

information is similar across all four figures.

- Plot(1,1) shows blood glucose readings at meal time. The average blood glucose value over a window of 100 meals is plotted in red. The desirable range of blood glucose values is shown by the green dashed lines.
- Plot(1,2) shows the first parameter of the policy, CF (correction factor). The red dashed line shows the optimal CF parameter and is provided in the T1DMS package.
- Plot(1,3) shows the other parameter of the policy, CR (carbohydrate ratio). As with the plot(1,2), the red dashed line shows the optimal CR parameter.
- Plot(2,1) shows the evolution of the average reward J^{∞}.
- Plot(2,2) shows the mean-reward corresponding to the time span between two consecutive meals. The averaged mean-reward over a window of 100 (pair-of) time spans is plotted in red. Standard deviation around this average is shown in cyan. The black dashed line in this plot shows mean-reward performance obtained by optimal T1DMS policy for the patient.
- Plot(2,3) shows the raw bolus insulin injections suggested by SAC. Average injections over a window of 100 meals is shown in red. Standard deviation around this mean is shown in cyan.
- Plot(3,1) shows the distribution of blood glucose values measured at meals between the 400th and 500th meals (a period corresponding roughly to the fifth month of learning). The green dashed lines show desirable range of blood glucose values.
- Plot(3,2) shows the distribution of blood glucose values measured for the last 100 meals (the last month of the 33 month learning process). The green dashed lines show the desirable range of blood glucose values.
- More information about the patient is provided in the bottom right corner of the figure: Patient-ID[11], CR^* and CF^* show the optimal standard policy parameters ($\theta^{\dagger}$). Reward^* is the average mean-reward if the optimal policy is followed.

The first figure in the series, Figure 5.6, shows results of SAC on patient 2 (PID=2). In this figure, Plot(1,3) shows the evolution of CR. Note that the smart-initialization suggested a higher-than-optimal value for the initial CR parameter. Plot(1,1) and Plot(3,1) show that blood glucose is poorly controlled in the first 500 meals. However, after about 1000 meals, when the CR parameter has almost converged to the optimal CR value, the blood glucose levels seem to be close to the acceptable range. Plot(2,2) shows a similar pattern for the mean-reward, where the mean-rewards get very close to optimal value after the first 1000 meals. Plot(3,2) shows histogram of blood glucose values over the last 100 meals; these values are in the acceptable range most of the time. This plot

[11] The patient-ID (PID) can be used to access more information about the patient in Appendix A.

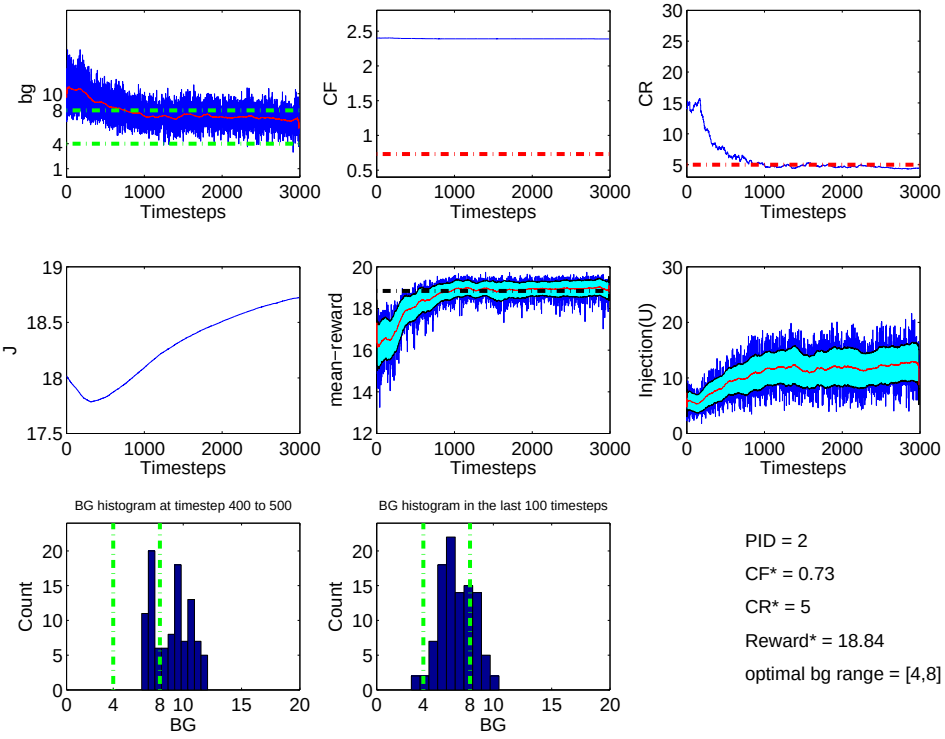


Figure 5.6: The evolution of variables during the SAC learning phase on patient 2 (PID=2). See text for more information.

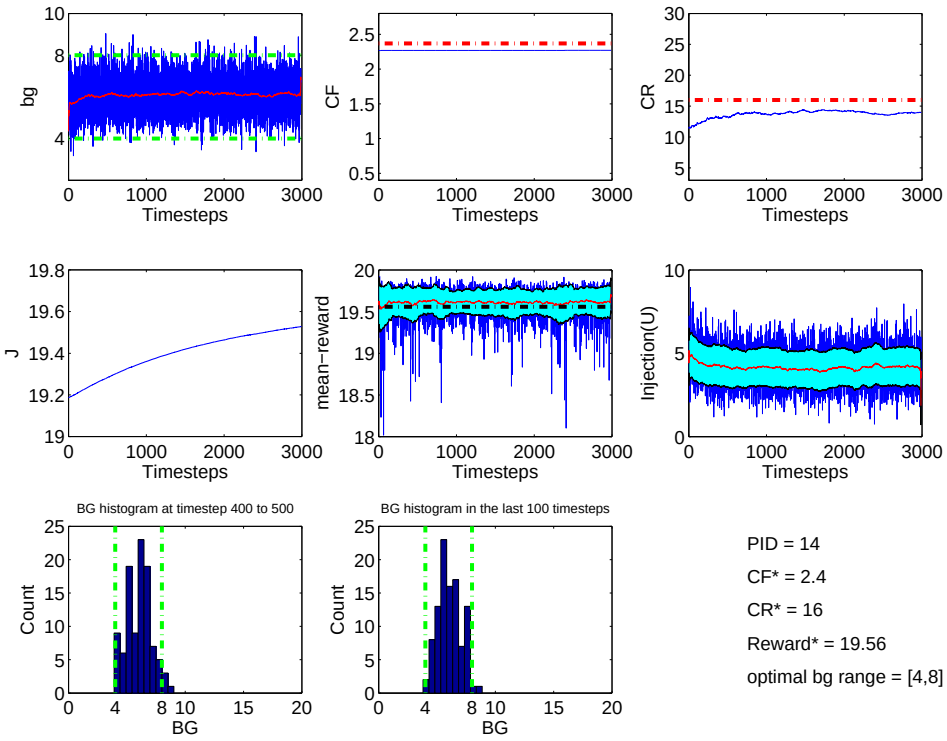


Figure 5.7: The evolution of variables during the SAC learning phase on patient 14 (PID=14). See text for more information.

shows a notable improvement over the histogram of the 5th month (Plot(3,1)). Plot(1,2) shows that, unlike other variables, the performance of SAC does not depend on the value of CF. This observation is in agreement with results provided in Section 1.3.7.

Next, we will discuss results for patient 14 (PID=14) shown in Figure 5.7. These results are in agreement with patient 2's results in showing the effectiveness of the SAC algorithm in controlling blood glucose. In this figure, Plot(1,3) shows that the smart initialization suggested a lower-than-optimal value for the initial CR parameter. We can see an increase in the CR parameter in the first hundred meals, which led to a better control of the blood glucose (see the slight increase of the blood glucose readings during the first hundred meals). Accordingly, Plot(3,1) shows that blood glucose levels are acceptable during the 5th month. Finally, Plot(1,2) shows that the CF parameter is initiated very close to its optimal value and has not changed much during the learning phase.

The effect of tile coding on performance of SAC

As observed in Section 5.2.2, tile coding improved Sarsa's performance. This observation motivated us to try to improve SAC's performance using a similar approach. For this purpose, we used a configuration similar to Sarsa's tile coding: 8 tiles for each blood-glucose feature, 3 tiles for meal size, 3 tiles for meal type, and 100 tilings. Figure 5.8 and Figure 5.9 demonstrate results obtained

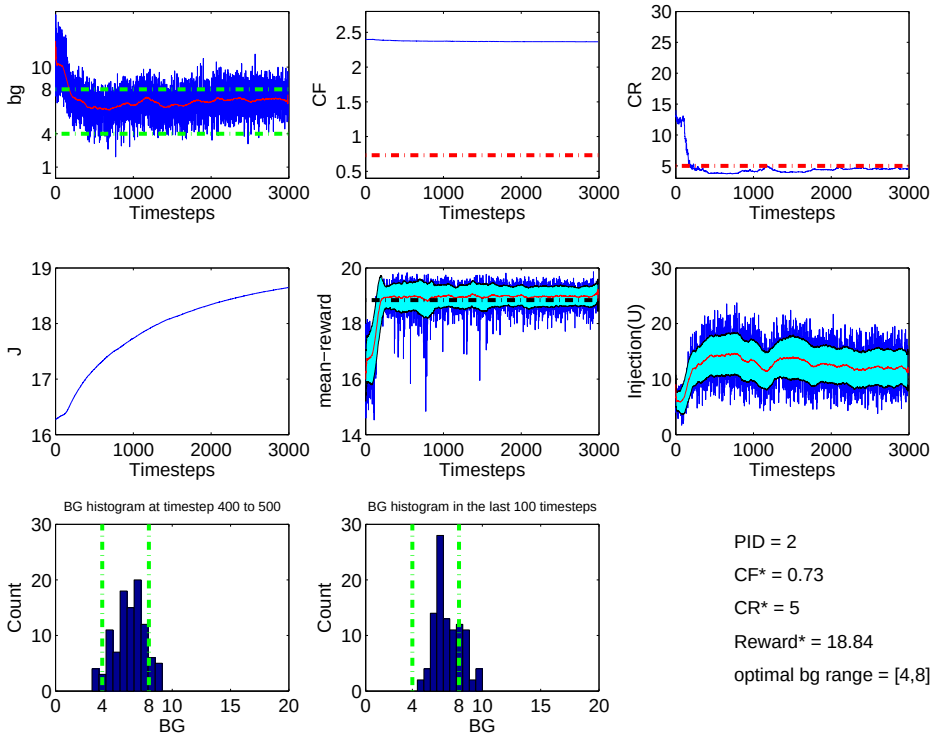


Figure 5.8: Evolution of different variables during the learning phase of SAC with tile coding. Here the algorithm was run on patient 2 (PID=2). See text for more information.

from running this version of SAC on patient 2 and patient 14 respectively. The results for patient 2 (Figure 5.8) show that this version of the SAC converges after about 200 meals to the optimal policy. This is 5 times faster than the convergence rate of the previous version of the SAC shown in Figure 5.7. Results for other patients show a similar performance improvement when the SAC algorithm is used with tile coding; see Appendix B.

5.3 Summary and further discussion

In this chapter we illustrated the results of our supervised and reinforcement learning algorithms. Here we provide a summary and further discussion of the results.

5.3.1 T1DC

We used two different methods to evaluate the T1DC algorithm. First, we analyzed the performance of the T1DC algorithm by looking at its prediction error, and later we compared its performance with that of an expert diabetologist. Although it generated prediction errors for some of the patients, the performance of the policy sequence obtained by T1DC was comparable to that of a diabetologist.

One important feature of the T1DC algorithm is its ability to generalize over a population. De-

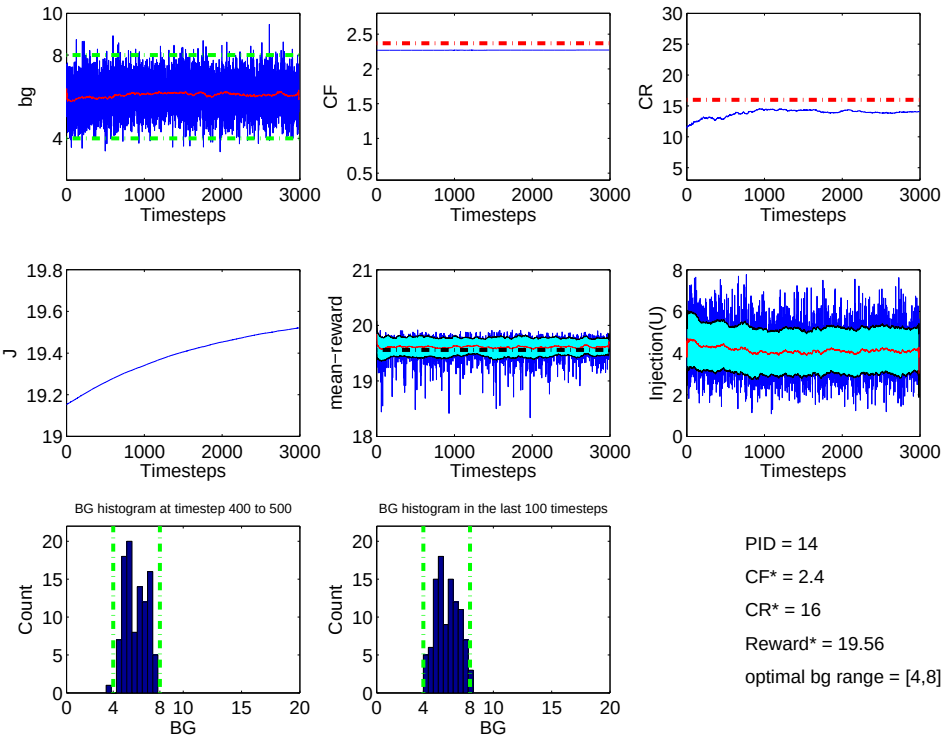


Figure 5.9: Evolution of different variables during the learning phase of SAC with tile coding. Here the algorithm was run on patient 14 (PID=14). See text for more information.

spite the small size of the patient population (20 patients), the T1DC algorithm was able to generalize its predictions across the patients and managed to achieve an acceptable performance within a short period of time (36 days –four iterations of 9 days). We anticipate this performance would be enhanced as the size of training set increases, making this method an appropriate tool for large clinical studies. Yet, one should be aware of the complications associated with using this method. For instance, it is not possible to use T1DC's current training method in a clinical study. In the current method the training data is obtained from some random policies, and in real life, these random policies cannot be tried on patients due to potential dangers. One way to address this issue might be to devise a clinical trial in which a small amount of noise is added to the policy that each patient is following. Although the sampled policies would not cover the whole space of policies, they might still cover the space sufficiently for T1DC's generalization purposes. More importantly, the optimal policy parameters (the labels of the T1DC algorithm), are not available for such a clinical study.

5.3.2 Sarsa

We have tested the Sarsa algorithm with various parameter and approximation settings. The results suggested the importance of two factors in the performance of Sarsa: (1) the exploration method and (2) generalization over state-action space. Our best performing Sarsa is obtained when TG-exploration is combined with generalization on the state-action space.

One of the main challenges of using Sarsa (and SAC) in this domain was finding an appropriate exploration method. To limit the possibility of harming the patient by taking an exploratory action, we limited the exploration to the neighbourhood of the best candidate action. The combination of this exploration method with pessimistic initialization of state-action value function helped us to prevent hypoglycemia as much as possible. Another challenge was selecting good values for the learning-rate and discount factor parameters. As explained earlier, due to the slowness of the simulator it was not feasible to search extensively for the right parameters. Despite this, our results suggests that Sarsa works best when γ is small. Finally, finding an appropriate discretization of actions was also a challenge in running Sarsa.

Although we did not address the discretization issue in our experiments, we note that discretization must be in accordance with the exploration method. We further discuss this using an example with the following assumptions. Assume that the Sarsa algorithm is running with the optimal state-action value function. Consider that, for a hypothetical patient, the optimal dosage of insulin corresponding to a specific size of meal is 10 units and that injections of 14 units of insulin or more causes hypoglycemia. Also assume that the range of available actions is [0,20]. If the size of bins for discretization of actions is 1 unit, then it is safe to use an exploration that selects 3 neighbouring actions (of the best candidate action) on either side of the action[12]. However if the size of bins are 2, then the same exploration may cause hypoglycemia. This is the case because during the exploration,

[12]The actions that are not away from the optimal action more than 3 actions.

the 3rd farthest neighbour might be selected. This would lead to an injection of six units more insulin than the optimal dosage (3 bins $\times$ 2 units per bin = 6), which, based on our assumption, would cause hypoglycemia.

5.3.3 SAC

We demonstrated the results of our implementation of the SAC algorithm, using four instances (from the complete set of results that are given in Appendix B). These results showed compelling evidence that the performance of a policy using parameters found by SAC are comparable to the performance of optimal policies. For some patients (*e.g.,* PID=5), this performance surpassed the performance of the T1DMS's optimal policy according to our criteria.

5.3.4 Additional discussion

Each of the above algorithms have their own advantages and disadvantages. On one hand, the T1DC algorithm can generalize over patient's population, based on sampled data from each patient. This generalization ability helps the T1DC algorithm to improve the policy efficiently in a short period of time (in our case, 36 days). On the other hand, the T1DC predictor cannot further adapt to a patient once its learning phase is over. When T1DC's learning phase is over, the predictor is fixed and the T1DC does not benefit from the additional data that are gathered during the test phase. By contrast, reinforcement learning (RL) methods learn continually from new experience. Another disadvantage of T1DC, is that it requires knowing the optimal policy parameters for training patients, while RL methods do not need this information.

In contrast to the T1DC algorithm, which naturally generalizes over patient population, it is hard for the Sarsa algorithm to transfer the knowledge that it has learned from one patient to another. We did not pursue this goal, but one way of incorporating this knowledge might be through the initiation of the action-value function using prior knowledge about a patient (and patients in general). For example, one can initialize the Q values corresponding to the actions that match the smart-initialization policy with higher values.

Among the reinforcement learning algorithms, SAC showed a better overall performance, which is not surprising for a few reasons. First, SAC has the knowledge about the structure of an effective policy (the standard policy). Second, SAC starts from a reasonable policy setting (smart-initialization) while Sarsa starts from scratch ($Q = 0$). Finally, SAC is designed for a continuous action space, which matches our task, while Sarsa is designed for discrete actions (which might cause the problems that are discussed in Section 5.3.2).

We should note that comparison between T1DC and the two reinforcement learning methods would not be fair due to the significant difference in their training data. T1DC achieves an acceptable performance (comparable to the performance of an expert diabetologist) faster than SAC, but when it was tested on one patient it had already seen 2700 days of data on each of the other 19 *in-*

silico patients (*e.g.,* 153,900 meals). Furthermore, the T1DC's learner has access to optimal policy parameters for its training patients.

Finally, we anticipate it may be possible to benefit from all of the above advantages by using T1DC for a few iterations and then initializing the SAC policy parameters using T1DC's suggested policy parameters.

Chapter 6

Conclusion and future work

This dissertation demonstrated the application of three machine learning techniques in automating type-1 diabetes management. As illustrated in Chapter 1, the management process requires a type-1 diabetic patient to follow a treatment policy to maintain her blood glucose level within an acceptable range. Currently, the patient's diabetologist adjusts patient's treatment-policy to enhance her blood glucose control. In this research we used machine learning techniques to automatically adjust the parameters of the meal-related component of patient's treatment policy.

We implemented three machine learning algorithms, consisting of T1DC, which is a supervised learning method, and Sarsa and SAC, which are reinforcement learning algorithms, and tested them on twenty *in-silico* patients. These *in-silico* patients were simulated using a commercial type-1 diabetes simulator called T1DMS. Our experimental results indicated that the methods we employed were able to adjust the treatment-policy of our *in-silico* patients with a performance level that was often comparable to that of an expert diabetologist.

6.1 Contributions

Contribution

During the course of this dissertation, we formalized the diabetes treatment policy adjustment problem and devised the following tools to solve this problem:

- Given that we have access to the optimal standard treatment-policies for some patients, we built a supervised learning tool that allows us to effectively control the blood glucose of other patients.
- We built a reinforcement learning tool to effectively control the blood glucose of a patient.

We defined a plausible score function to assess the effectiveness of our provided tools[1]. Our empirical results for the T1DC method (Section 5.1) showed that the performance of the policy sequence obtained using this supervised learning tool is comparable to that of an expert diabetologist.

[1]Note that our framework is independent of the specific score function, which can be replaced by another score function to address the policy adjustment problem. But we did not try that here.

Furthermore, the results of SAC and Sarsa (provided in Section 5.2) showed that these reinforcement learning tools can be used to effectively adjust the treatment policy of *in-silico* patients.

6.2 Future Work

In this dissertation we only scratched the surface of the type-1 diabetes control problem, leaving many avenues for future work. The next sections provide a selection of topics for further research, including many that have already been introduced in earlier chapters.

6.2.1 Meal consumption actions

In a real-life hypoglycemia event, a typical diabetes patient immediately consumes some form of sugar (*e.g.,* sugar-tablet, candy, etc.) to prevent the blood glucose level from further decreasing [52]. As mentioned in the description of the T1DMS simulator (Chapter 3), it is possible to implement a controller that can simulate the above corrective behaviour of the patient. This extension can alleviate some of the hypoglycemia-related issues discussed in Section 3.2 – that is stopping and restarting the simulator if patient's glucose level drops below a certain threshold. This corrective measure can be considered as a new action for the controller or it can be implemented as a rescue signal. In the case of the rescue signal, upon the occurrence of a hypoglycemia event, the signal would induce the controller to feed the patient with a fixed amount of sugar and punish the learning algorithm (of the controller) and then continue the learning process. This modification would make the learning task more similar to the real-life scenario but it also adds complexities that need to be addressed (*e.g.,* what is the right way of penalizing this action).

6.2.2 Extending the application to type-2 diabetes

As mentioned in Chapter 1, the bodies of the patients with type-2 diabetes can still produce some quantity of insulin. Thus, management of type-2 diabetes is not as hard as that of type-1 diabetes. In severe cases of type-2 diabetes, where patients need to inject insulin, the control problem becomes very similar to the type-1 diabetes problem. It would be interesting to investigate whether our algorithms would work in this slightly modified domain with some tuning. A commercial simulator of type-2 diabetes is under development by developers of T1DMS, and it could be a potential platform for future analysis.

6.2.3 Other score functions

A critical part of a reinforcement learning problem is the definition of a reasonable reward function. This reward function characterizes the goal of the algorithm. Our formulation of reward in the diabetes problem (as explained in Section 2.2.5) is defined based on a score function. One possibility for future work is to analyze the properties of various score functions proposed in the diabetes literature (*e.g.,* see Fig.4 of [24]) and their effect on the performance of blood glucose control.

6.2.4 Reinforcement learning exploration methods

As demonstrated in Chapter 5, one of the factors that can influence the performance of a reinforcement learning algorithms is its exploration method. Some properties of diabetes management (*e.g.,* the risk of hypoglycemia) make it essential to come up with a customized exploration method, which considers the risk associated with each exploration event. We dealt with the risk issue by restricting exploration to a local neighbourhood. But a more rigorous study of this issue is of great interest. As a possible first step towards this customization we could change the Gaussian stochastic policy of our actor-critic method to an asymmetric distribution.

6.2.5 Dynamic learning rates

It might be of interest to improve the performance of the reinforcement learning algorithms by dynamically changing their learning rates through the learning process (*e.g.,* see [18]). In simple terms, this is intuitive because when we get more confident in our estimates (of state-value function, action-value function, etc.) we do not want to change them significantly.

6.2.6 T1DC improvements

Asymmetric prediction loss (for T1DC)

In the T1DC algorithm, the loss function of the support vector regression method (as shown in Figure 2.1) penalizes positive and negative errors similarly. However, in our application, the positive errors are less important than the negative errors (as explained in Section 5.1.1). Using an asymmetric loss may improve performance of the T1DC algorithm.

Check for improvement

One way of making sure that T1DC policy updates are actually improving current policy is to consider the evolution of the mean-reward for each policy over its k-days. Using this information in the updates could prevent a decrease in the performance of T1DC during the iterative process.

6.2.7 Warm-start

One of the factors that affect the performance of the reinforcement learning algorithms (as discussed in Section 5.3) is their initialization. In real-life situations, when a new patient wants to try a hypothetical sophisticated controller, there might be years' worth of information from a diabetes diary available for her. One possibility for future work is to find ways to leverage this available information for better initialization of the controller.

Bibliography

[1] S.I. Ahmad. *Diabetes: An Old Disease, a New Insight*. Springer-Verlag New York Incorporated, 2013. ISBN 9781461454403.

[2] A.M. Albisser, B.S. Leibel, T.G. Ewart, Z. Davidovac, C.K. Botz, W. Zingg, H. Schipper, and R. Gander. Clinical control of diabetes by the artificial pancreas. *Diabetes*, 23(5):397–404, 1974.

[3] American Diabetes Association. Standards of medical care in diabetes–2013. *Diabetes Care*, 36(Supplement_1):S11–S66, December 2012. ISSN 0149-5992, 1935-5548. doi: 10.2337/dc13-S011. URL http://care.diabetesjournals.org/content/36/Supplement_1/S11.full.

[4] S. Andreassen, J.J. Benn, R. Hovorka, K.G. Olesen, and E.R. Carson. A probabilistic approach to glucose prediction and insulin dose adjustment: description of metabolic model and pilot evaluation study. *Computer Methods and Programs in Biomedicine*, 41(3):153–165, 1994.

[5] N.P. Balakrishnan, G.P. Rangaiah, and L. Samavedham. Review and analysis of blood glucose (bg) models for type 1 diabetic patients. *Industrial & Engineering Chemistry Research*, 50 (21):12041–12066, 2011.

[6] R.N. Bergman, Y.Z. Ider, C.R. Bowden, and C. Cobelli. Quantitative estimation of insulin sensitivity. *American Journal of Physiology-Endocrinology And Metabolism*, 236(6):E667, 1979.

[7] S. Bhatnagar, R.S. Sutton, M. Ghavamzadeh, and M. Lee. Natural actor–critic algorithms. *Automatica*, 45(11):2471–2482, 2009.

[8] V.W. Bolie. Coefficients of normal blood glucose regulation. *Journal of Applied Physiology*, 16(5):783–788, 1961.

[9] R.M. Bookchin and P.M. Gallop. Structure of hemoglobin AIc: nature of the n-terminal chain blocking group. *Biochemical and Biophysical Research Communications*, 32(1):86–93, July 1968. ISSN 0006-291X. doi: 10.1016/0006-291X(68)90430-0. URL http://www.sciencedirect.com/science/article/pii/0006291X68904300.

[10] E.R. Carson and T. Deutsch. A spectrum of approaches for controlling diabetes. *Control Systems, IEEE*, 12(6):25–31, 1992.

[11] A. Caumo and C. Cobelli. Hepatic glucose production during the labeled IVGTT: estimation by deconvolution with a new minimal model. *The American Journal of Physiology*, 264(5 Pt 1):E829–41, May 1993. LR: 20061115; JID: 0370511; 0 (Blood Glucose); 11061-68-0 (Insulin); 50-99-7 (Glucose); ppublish.

[12] S.C. Chao and A.M. Albisser. The diabetes simulator. *Decision support for patient management*, 1989.

[13] F. Chee and T. Fernando. *Closed-Loop Control of Blood Glucose*. Lecture Notes in Control and Information Sciences. Springer, 2007. ISBN 9783540740308. URL http://books.google.ca/books?id=NJe-NdZzgGMC.

[14] M.E. Chevreul. Note sur le sucre du diabte. *Ann Chim (Paris)*, 95:319–320, 1815.

[15] S.F. Clarke and J.R. Foster. A history of blood glucose meters and their role in self-monitoring of diabetes mellitus. *British Journal of Biomedical Science*, 69(2):83–93, 2012.

[16] R. Collobert and S. Bengio. Svmtorch: Support vector machines for large-scale regression problems. *The Journal of Machine Learning Research*, 1:143–160, 2001.

[17] M.A. Creager, T.F. Lüscher, F. Cosentino, and J.A. Beckman. Diabetes and vascular disease pathophysiology, clinical consequences, and medical therapy: part i. *Circulation*, 108(12): 1527–1532, 2003.

[18] W. Dabney and A.G. Barto. Adaptive step-size for online temporal difference learning. In *AAAI*, 2012.

[19] C. Dalla Man, A. Caumo, R. Basu, R. Rizza, G. Toffolo, and C. Cobelli. Minimal model estimation of glucose absorption and insulin sensitivity from oral test: validation with a tracer method. *American journal of physiology. Endocrinology and metabolism*, 287(4):E637–43, October 2004. ISSN 0193-1849. doi: 10.1152/ajpendo.00319.2003. URL http://www.ncbi.nlm.nih.gov/pubmed/15138152.

[20] C. Dalla Man, G. Toffolo, R. Basu, R. a Rizza, and C. Cobelli. A model of glucose production during a meal. *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference*, 1(0):5647–50, January 2006. ISSN 1557-170X. doi: 10.1109/IEMBS.2006.260809. URL http://www.ncbi.nlm.nih.gov/pubmed/17946713.

[21] C. Dalla Man, R. a Rizza, and C. Cobelli. Meal simulation model of the glucose-insulin system. *IEEE transactions on bio-medical engineering*, 54(10):1740–9, October 2007. ISSN 0018-9294. doi: 10.1109/TBME.2007.893506. URL http://www.ncbi.nlm.nih.gov/pubmed/17946394.

[22] C. DallaMan, Raimondo D.M., Rizza R.A., and C. Cobelli. GIM, simulation software of meal glucose-insulin model. *Journal of diabetes science and technology*, 1(3):323–330, May 2007. LR: 20100928; JID: 101306166; OID: NLM: PMC2769591; ppublish.

[23] T. Degris, P.M. Pilarski, and R.S. Sutton. Model-free reinforcement learning with continuous action in practice. In *American Control Conference (ACC), 2012*, pages 2177–2182. IEEE, 2012.

[24] L. Desborough, C. Palerm, and S. Monirabbasi. Glycemic health metric determination and application, April 25 2013. US Patent 20,130,102,867.

[25] Matlab® developers. Curve fitting toolbox, interpolants. URL http://www.mathworks.com/help/curvefit/interpolants.html.

[26] D. Elleri, J.M. Allen, D.B. Dunger, and R. Hovorkea. Closed-loop in children with type 1 diabetes: Specific challenges. *Diabetes research and clinical practice*, 93:S131–S135, 2011.

[27] T.G. Farmer, T.F. Edgar, and N.A. Peppas. The future of open- and closed-loop insulin delivery systems. *Journal of Pharmacy and Pharmacology*, 60(1):1–13, 2008. ISSN 2042-7158. doi: 10.1211/jpp.60.1.0001. URL http://dx.doi.org/10.1211/jpp.60.1.0001.

[28] E. Ferrannini and C. Cobelli. The kinetics of insulin in man. i. general aspects. *Diabetes Metab Rev*, 3(2):335–63, 1987. ISSN 0742-4221. URL http://www.biomedsearch.com/nih/kinetics-insulin-in-man-General/3552526.html.

[29] M. Gondran. *An introduction to expert systems*. McGraw-Hill, Inc., 1986.

[30] Public Health Agency of Canada Government of Canada. Report from the national diabetes surveillance system: Diabetes in canada, 2009. http://www.phac-aspc.gc.ca/publicat/2009/ndssdic-snsddac-09/index-eng.php#toc, February 2009. URL http://www.phac-aspc.gc.ca/publicat/2009/ndssdic-snsddac-09/index-eng.php#toc. Presents the findings from the National Diabetes Surveillance System (NDSS) for 2009. Indicates future plans for the NDSS.

[31] The Epsilon Group. *The UVa / Padova T1DM Simulator T1DMS Distributed Version User Guide*, 2011.

[32] D.A. Hepburn, I.J. Deary, B.M. Frier, A.W. Patrick, J.D. Quinn, and B.M. Fisher. Symptoms of acute insulin-induced hypoglycemia in humans with and without iddm: factor-analysis approach. *Diabetes Care*, 14(11):949–957, 1991.

[33] R.I.G. Holt, C. Cockram, A. Flyvbjerg, and B.J. Goldstein. *Textbook of Diabetes*. Wiley, 2011. ISBN 9781444348064. URL http://books.google.ca/books?id=mGV4wu8AkPwC.

[34] R. Hovorka, S. Svacina, E.R. Carson, C.D. Williams, and P.H. Sonksen. A consultation system for insulin therapy. *Computer methods and programs in biomedicine*, 32(3):303–310, 1990.

[35] Geyelin H.R., Harrop G., Murray M.F., and Corwin E. The use of insulin in juvenile diabetes. *Journal of Metabolic Research*, 2:767–792, 1922.

[36] T.H.J. Huisman, E.A. Martis, and A. Dozy. Chromatography of hemoglobin types on carboxymethylcellulose. *The Journal of laboratory and clinical medicine*, 52(2):312–327, 1958.

[37] IDF. IDF diabetes atlas update 2012, 2012. URL http://www.idf.org/diabetesatlas/5e/Update2012.

[38] P. James and McFadden R. Understanding the processes behind the regulation of blood glucose. *Nursing times*, 100(16):56–58, April 2004. ISSN 0954-7762. URL http://www.nursingtimes.net/Journals/2012/12/04/i/c/v/040420Understanding-the-processes-behind-the-regulation-of-blood-glucose.pdf. PMID: 15132068.

[39] R.J. Koenig, C.M. Peterson, R.L. Jones, C. Saudek, M. Lehrman, and A. Cerami. Correlation of glucose regulation and hemoglobin AIc in diabetes mellitus. *New England Journal of Medicine*, 295(8):417–420, 1976. ISSN 0028-4793. doi: 10.1056/NEJM197608192950804. URL http://www.nejm.org/doi/full/10.1056/NEJM197608192950804. PMID: 934240.

[40] B.P. Kovatchev, M. Breton, C.D. Man, and C. Cobelli. In silico preclinical trials: A proof of concept in closed-loop control of type 1 diabetes. *Journal of diabetes science and technology*, 3(1):44–55, 2009.

[41] E.D. Lehmann, T. Deutsch, E.R. Carson, and P.H. Sönksen. Aida: an interactive diabetes advisor. *Computer methods and programs in biomedicine*, 41(3):183–203, 1994.

[42] E.D. y Lehmann and T. Deutsch. Application of computers in diabetes care-a review. ii. computers for decision support and education. *Informatics for Health and Social Care*, 20(4):303–329, 1995.

[43] K. Lunze, T. Singh, M. Walter, M.D. Brendel, and S. Leonhardt. Blood glucose control algorithms for type 1 diabetic patients: A methodological review. *Biomedical Signal Processing and Control*, 2012.

[44] S.P. Meyn, R.L. Tweedie, and P.W. Glynn. *Markov chains and stochastic stability*, volume 2. Cambridge University Press Cambridge, 2009.

[45] S.A. Murphy. personal communication, dec 2012.

[46] D.M. Nathan, J. Kuenen, R. Borg, H. Zheng, D. Schoemfeld, and R.J. Heine. Translating the A1C Assay Into Estimated Average Glucose Values. *Diabetes*, 31(8):1–6, 2008. doi: 10.2337/dc07-0545.

[47] G. Nucci and C. Cobelli. Models of subcutaneous insulin kinetics. A critical review. *Computer methods and programs in biomedicine*, 62(3):249–57, July 2000. ISSN 0169-2607. URL http://www.ncbi.nlm.nih.gov/pubmed/10837910.

[48] J.M.D. Olmsted. Claude bernard, 1813-1878; a pioneer in the study of carbohydrate metabolism. *Diabetes*, 2(2):162–164, April 1953. ISSN 0012-1797. PMID: 13033650.

[49] B. Pagurek, J.S. Riordon, and S. Mahmoud. Adaptive control of the human glucose-regulatory system. *Medical and biological engineering*, 10(6):752–761, 1972.

[50] P. Palazzo and V. Viti. A new glucose-clamp algorithm-theoretical considerations and computer simulations. *Biomedical Engineering, IEEE Transactions on*, 37(5):535–543, 1990.

[51] K. Patrick, W.G. Griswold, F. Raab, and S.S. Intille. Health and the mobile phone. *American journal of preventive medicine*, 35(2):177, 2008.

[52] W.A. Petit, W.A.J. Petit, and C.A. Adamec. *The Encyclopedia of Diabetes*. Facts on File Library of Health and Living. Facts On File, Incorporated, 2010. ISBN 9780816079483. URL http://books.google.ca/books?id=0-H-QQAACAAJ.

[53] K.S. Polonsky. The past 200 years in diabetes. *New England Journal of Medicine*, 367(14): 1332–1340, 2012. ISSN 0028-4793. doi: 10.1056/NEJMra1110560. URL http://www.nejm.org/doi/full/10.1056/NEJMra1110560. PMID: 23034021.

[54] M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994. ISBN 0471619779.

[55] S. Rahbar, O. Blumenfeld, and H.M. Ranney. Studies of an unusual hemoglobin in patients with diabetes mellitus. *Biochemical and Biophysical Research Communications*, 36(5):838–843, August 1969. ISSN 0006-291X. doi: 10.1016/0006-291X(69)90685-8. URL http://www.sciencedirect.com/science/article/pii/0006291X69906858.

[56] G.A. Rummery and M. Niranjan. *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering, 1994.

[57] S.J. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Prentice hall Englewood Cliffs, 2010.

[58] E.A. Ryan, T. Shandro, K. Green, B.W. Paty, P.A. Senior, D. Bigam, A.M.J. Shapiro, and MC. Vantyghem. Assessment of the severity of hypoglycemia and glycemic lability in type 1 diabetic subjects undergoing islet transplantation. *Diabetes*, 53(4):955–962, 2004.

[59] C.D. Saudek, R.L. Derr, and R.R. Kalyani. Assessing glycemia in diabetes using self-monitoring blood glucose and hemoglobin a1c. *The Journal of the American Medical Association*, 295(14):1688–1697, April 2006. ISSN 0098-7484. doi: 10.1001/jama.295.14.1688. URL http://dx.doi.org/10.1001/jama.295.14.1688.

[60] J. Schlichtkrull, O. Munck, and M. Jersild. The m-value, an index of blood-sugar control in diabetics. *Acta Medica Scandinavica*, 177(1):95–102, 1965. ISSN 0954-6820. doi: 10.1111/j.0954-6820.1965.tb01810.x. URL http://dx.doi.org/10.1111/j.0954-6820.1965.tb01810.x.

[61] A.M.J. Shapiro, J.R.T. Lakey, E.A. Ryan, G.S. Korbutt, E. Toth, G.L. Warnock, N.M. Kneteman, and R.V. Rajotte. Islet transplantation in seven patients with type 1 diabetes mellitus using a glucocorticoid-free immunosuppressive regimen. *New England Journal of Medicine*, 343(4):230–238, 2000.

[62] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004. ISBN 9780521813976. URL http://books.google.ca/books?id=9i0vg12lti4C.

[63] S. Shimoda, K. Nishida, M. Sakakida, Y. Konno, K. Ichinose, M. Uehara, T. Nowak, and M. Shichiri. Closed-loop subcutaneous insulin infusion algorithm with a short-acting insulin analog for long-term clinical application of a wearable artificial endocrine pancreas. *Frontiers of medical and biological engineering: the international journal of the Japan Society of Medical Electronics and Biological Engineering*, 8(3):197, 1997.

[64] J.S. Skyler, D.L. Skyler, D.E. Seigler, and M.J. O'Sullivan. Algorithms for adjustment of insulin dosage by patients who monitor blood glucose. *Diabetes Care*, 4(2):311–318, 1981.

[65] A.J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004.

[66] R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3 (1):9–44, 1988.

[67] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. A Bradford book. A Bradford Book, 1998. ISBN 9780262193986. URL http://books.google.ca/books?id=CAFR6IBF4xYC.

[68] R.S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12(22), 2000.

[69] C. Szepesvári. Algorithms for reinforcement learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 4(1):1–103, 2010.

[70] D. Takahashi, Y. Xiao, and F. Hu. A survey of insulin-dependent diabetes-part ii: control methods. *International journal of telemedicine and applications*, 2008:4, 2008.

[71] B. Tao, M. Pietropaolo, M. Atkinson, D. Schatz, and D. Taylor. Estimating the cost of type 1 diabetes in the us: a propensity score matching method. *PLoS One*, 5(7):e11501, 2010.

[72] R. Tattersall. *Diabetes: The Biography*. Biographies of Disease. OUP Oxford, 2009. ISBN 9780199541362. URL http://books.google.ca/books?id=UBuVrdNBWKkC.

[73] G. Toffolo, M. Campioni, R. Basu, R. a Rizza, and C. Cobelli. A minimal model of insulin secretion and kinetics to assess hepatic insulin extraction. *American journal of physiology. Endocrinology and metabolism*, 290(1):E169–E176, January 2006. ISSN 0193-1849. doi: 10.1152/ajpendo.00473.2004. URL http://www.ncbi.nlm.nih.gov/pubmed/16144811.

[74] V. Vapnik. *The nature of statistical learning theory*. springer, 1999.

[75] P. Vicini, A. Caumo, and C. Cobelli. The hot IVGTT two-compartment minimal model: indexes of glucose effectiveness and insulin sensitivity. *The American Journal of Physiology*, 273(5 Pt 1):E1024–32, Nov 1997.

[76] J. Walsh. *Using insulin: everything you need for success with insulin*. Torrey Pines Press, 2003.

[77] WHO. Health system statistics: Human resources for health, by who region, 1995-2004, 2005. URL http://www.who.int/healthinfo/statistics/14whostat2005graph_humanresources.jpg.

[78] M. Wiering and M. Van Otterlo. *Reinforcement Learning: State-of-the-art*, volume 12. Springer, 2012.

[79] Wikipedia. Glycated hemoglobin — wikipedia, the free encyclopedia, 2013. URL https://en.wikipedia.org/w/index.php?title=Glycated_hemoglobin. Online; accessed 2013-06-04.

[80] Wikipedia. Truncated normal distribution — wikipedia, the free encyclopedia, 2013. URL https://en.wikipedia.org/wiki/Truncated_normal_distribution. Online; accessed 2013-06-04.

[81] Wikipedia. Controller (control theory) — wikipedia, the free encyclopedia, 2013. URL https://en.wikipedia.org/wiki/Controller_%28control_theory%29. Online; accessed 2013-07-04.

[82] I.H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[83] H. Wolpert. *Smart Pumping: A Practical Approach to the Insulin Pump*. McGraw-Hill Companies,Incorporated, 2002. ISBN 9781580401258. URL http://books.google.ca/books?id=mxkCAAAACAAJ.

[84] P. Zhang, X. Zhang, J.B. Brown, D. Vistisen, R.A. Sicree, J. Shaw, and G.A. Nichols. Economic impact of diabetes. *Diabetes Atlas, IDF*, 4, 2010.

[85] F. Zhou, HI Yang, J.M.R. Álamo, J.S. Wong, and C.K. Chang. *Mobile personal health care system for patients with diabetes*. Springer, 2010.

Appendix A

List of In-Silico Patients

Patient-Type	Patient-id(PID)	Body-Weight(Kg)	Age	Optimal-basal	Optimal-Sensitivity(CF)	Optimal-CHO(CR)
Adolescent	1	68.7	18	0.84	0.5	12
Adolescent	2	51	19	0.92	0.5	15
Adolescent	3	44.8	15	0.65	0.5	15
Adolescent	4	49.6	17	0.87	0.5	15
Adolescent	5	47	16	0.72	0.5	15
Adolescent	6	45.5	14	0.88	0.5	15
Adolescent	7	37.9	16	0.78	0.4	17
Adolescent	8	41.2	14	0.56	0.4	17
Adolescent	9	43.9	19	0.61	0.4	17
Adolescent	10	47.4	17	0.79	0.4	17
Adult	11	102.3	61	1.27	0.7	10
Adult	12	111.1	65	1.37	0.7	10
Adult	13	81.6	27	1.43	0.6	13
Adult	14	63	66	0.89	0.5	14
Adult	15	94	52	1.18	0.6	10
Adult	16	66	26	1.72	0.5	12
Adult	17	91.2	35	1.37	0.6	10
Adult	18	102.8	48	1.14	0.7	10
Adult	19	74.6	68	1.13	0.5	13
Adult	20	73.9	68	1.02	0.5	12
Child	21	34.6	9	0.39	0.4	17
Child	22	28.5	9	0.4	0.3	20
Child	23	41.2	8	0.29	0.3	20
Child	24	35.5	12	0.49	0.3	20
Child	25	37.8	10	0.52	0.3	20
Child	26	41	8	0.4	0.3	20
Child	27	45.5	9	0.47	0.4	20
Child	28	23.7	10	0.34	0.3	20
Child	29	35.5	7	0.39	0.3	20
Child	30	35.2	12	0.4	0.3	20

Appendix B

Additional Results for SAC algorithm

B.1 SAC with tabular critic

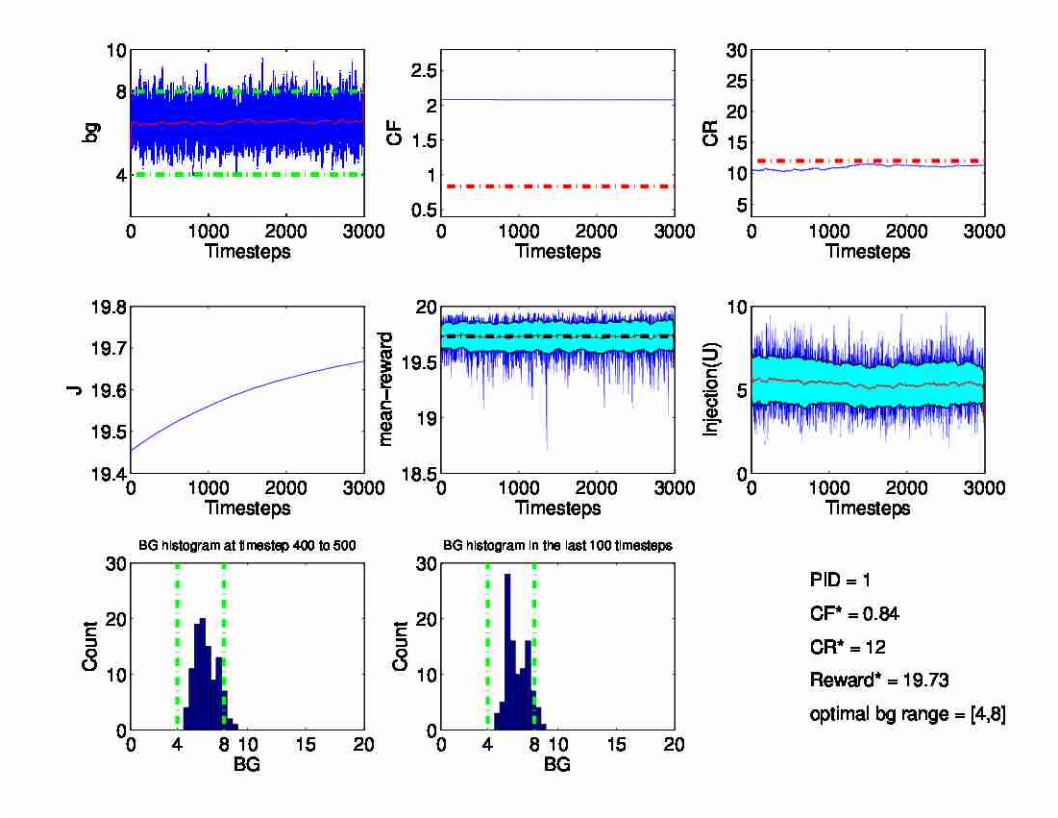


Figure B.1: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

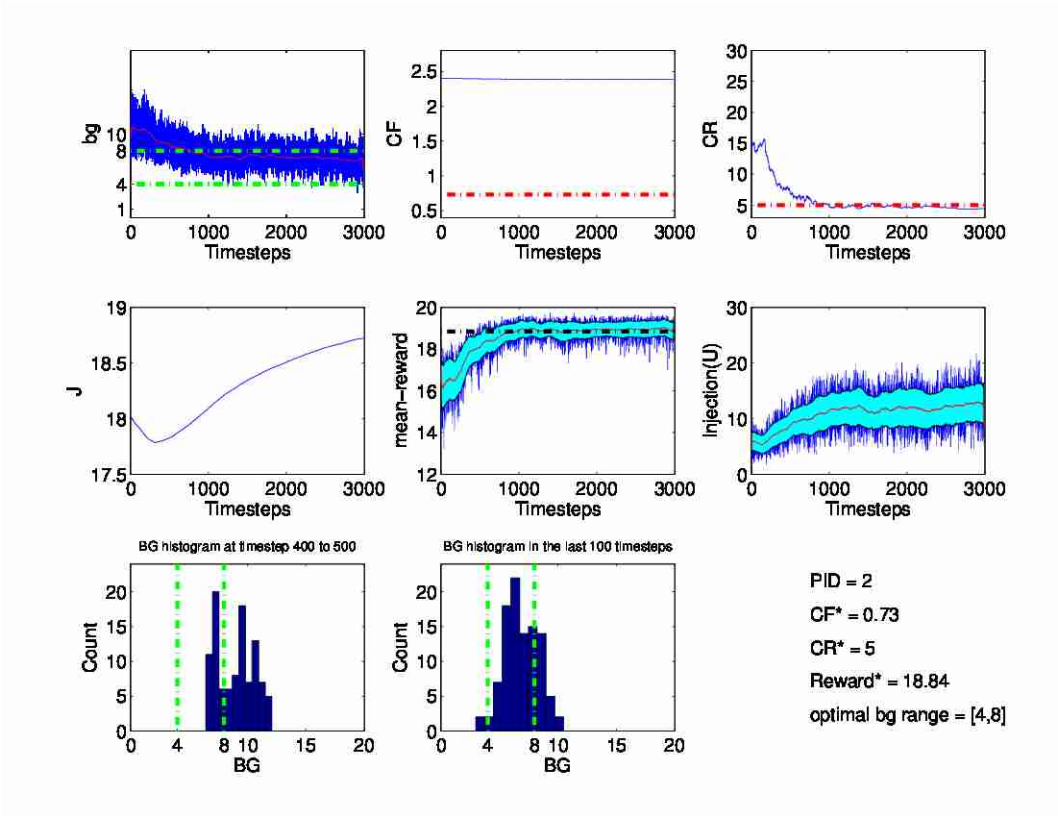


Figure B.2: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

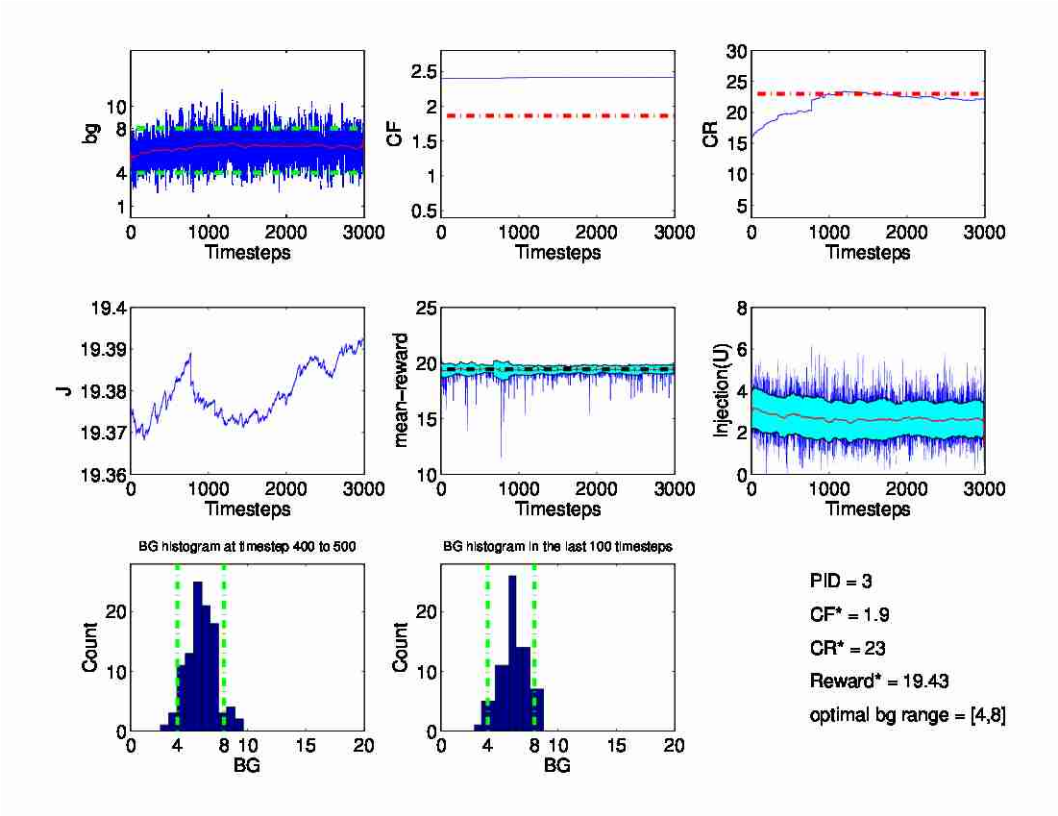


Figure B.3: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

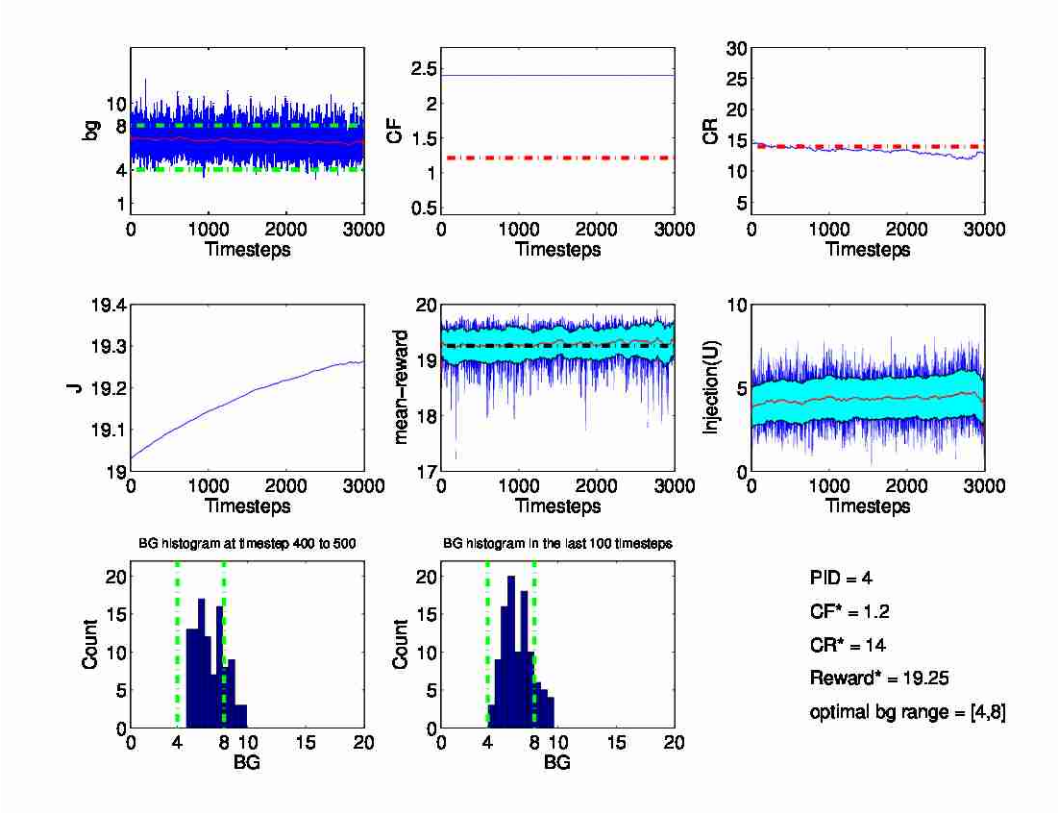


Figure B.4: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

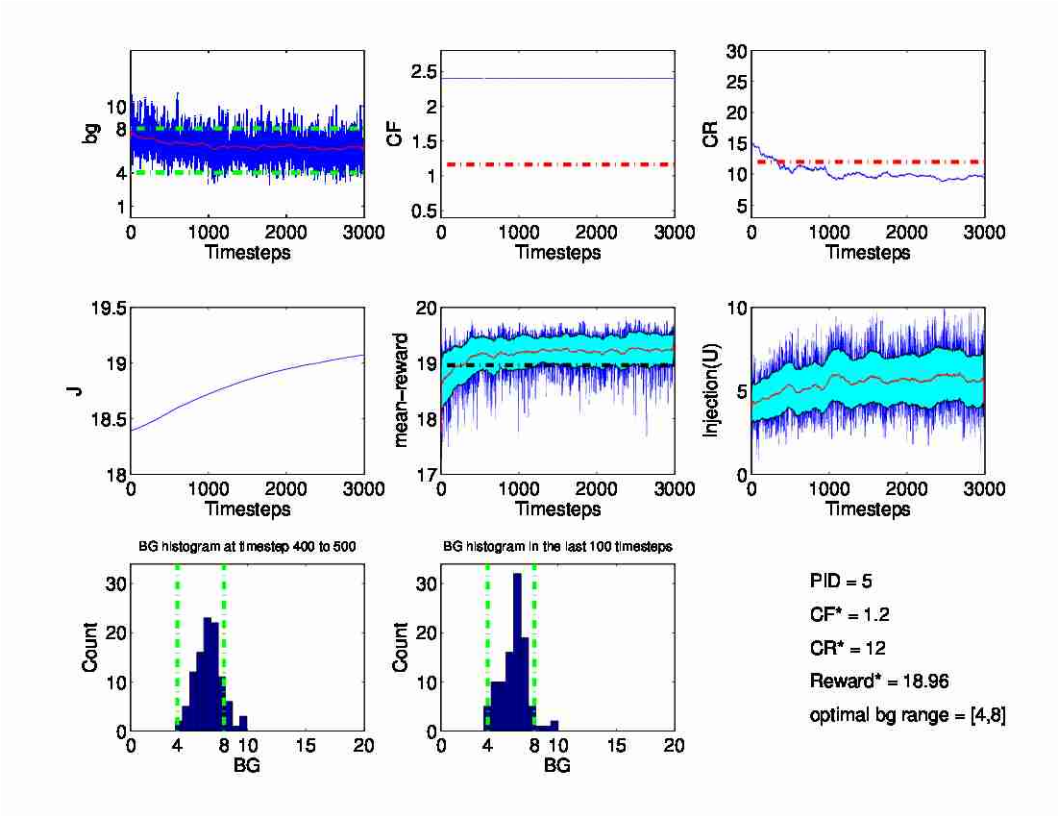


Figure B.5: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

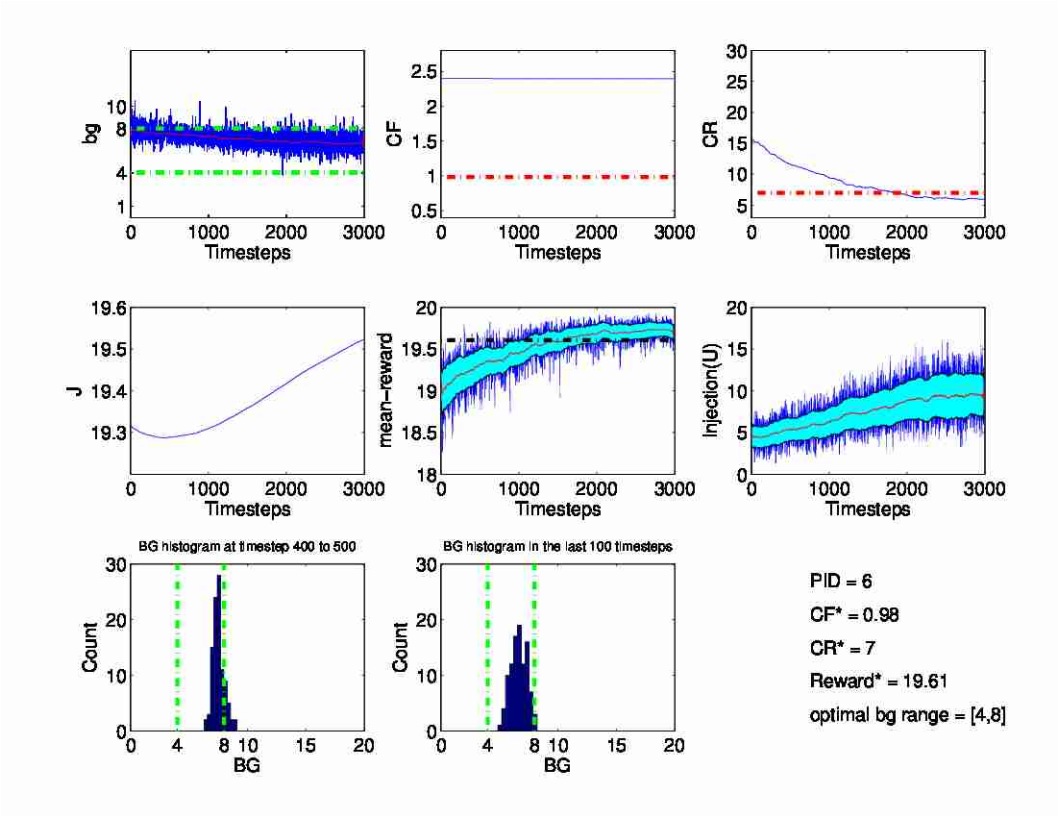


Figure B.6: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

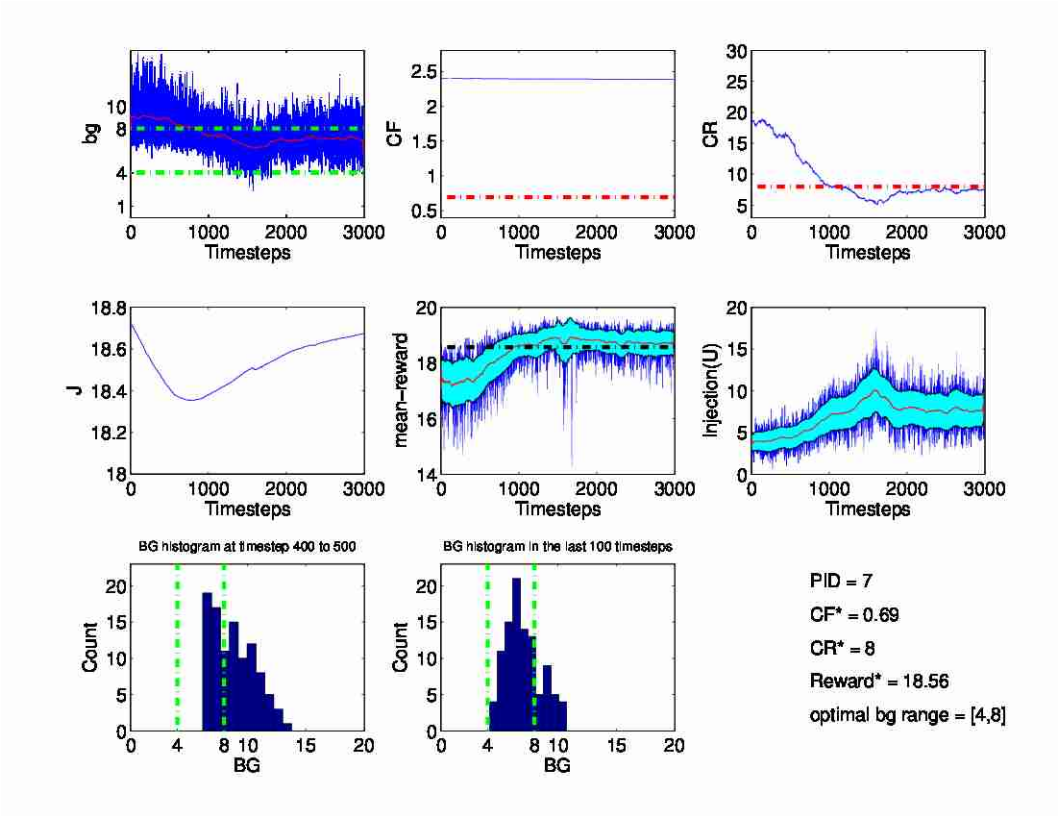


Figure B.7: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

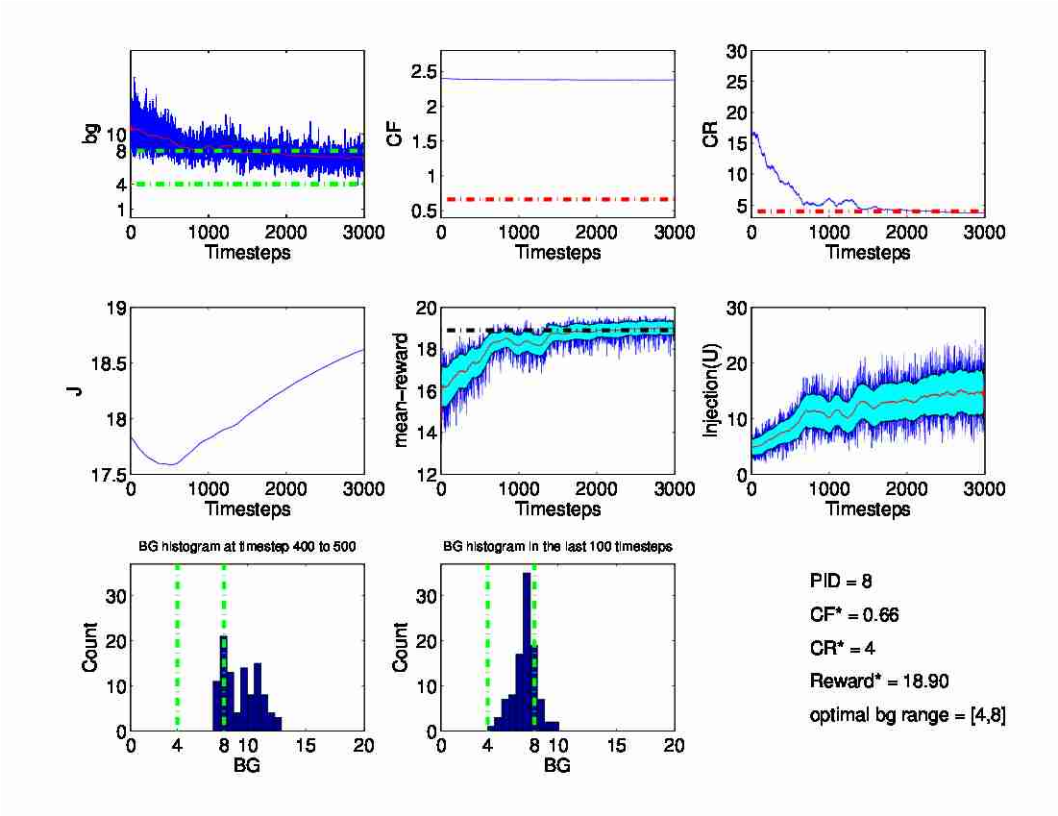


Figure B.8: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

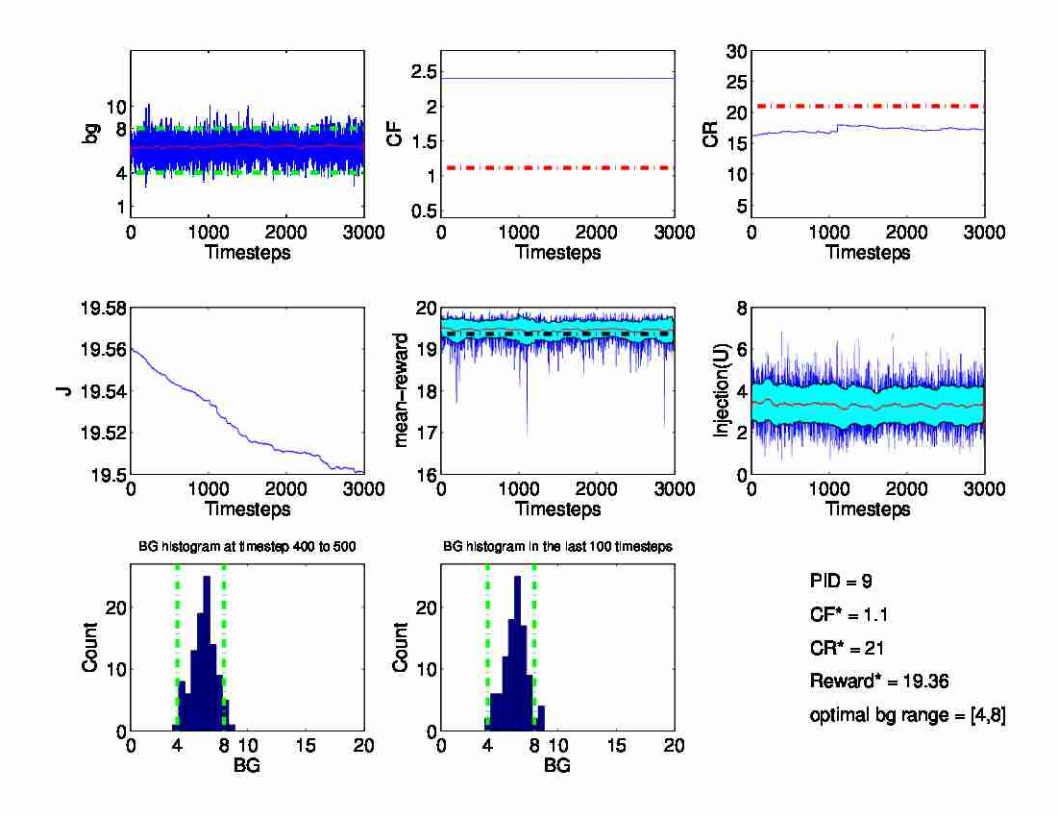


Figure B.9: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

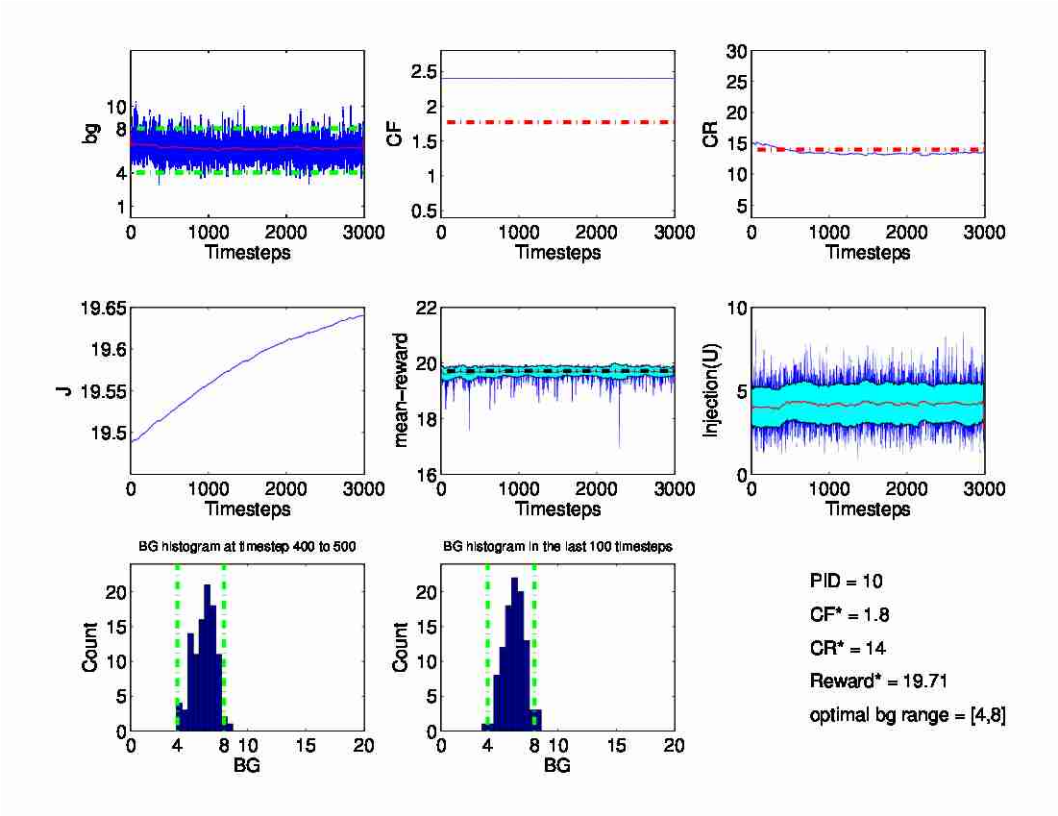


Figure B.10: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

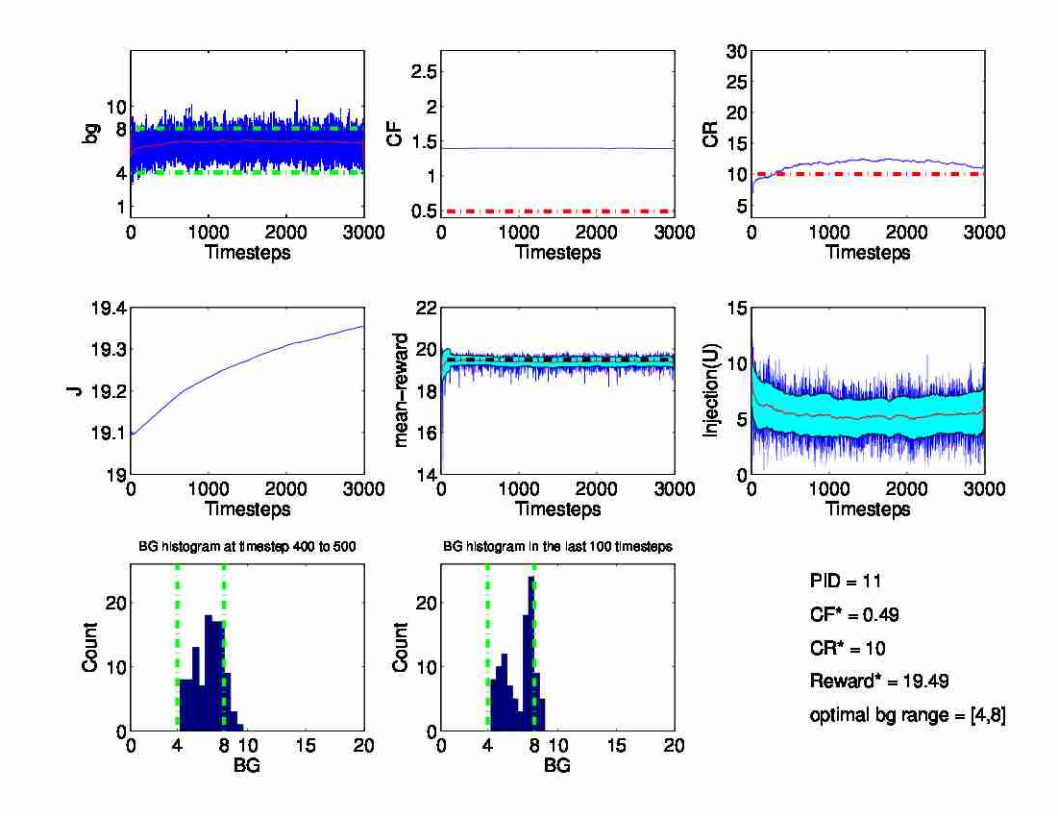


Figure B.11: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

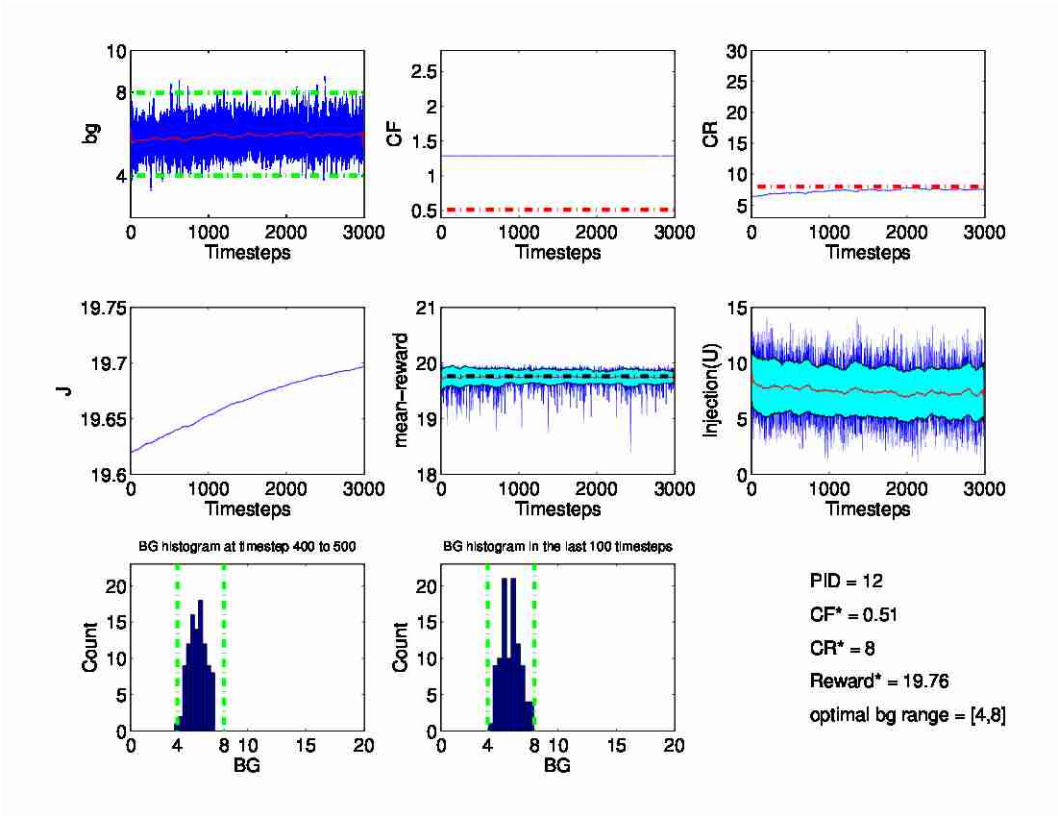


Figure B.12: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

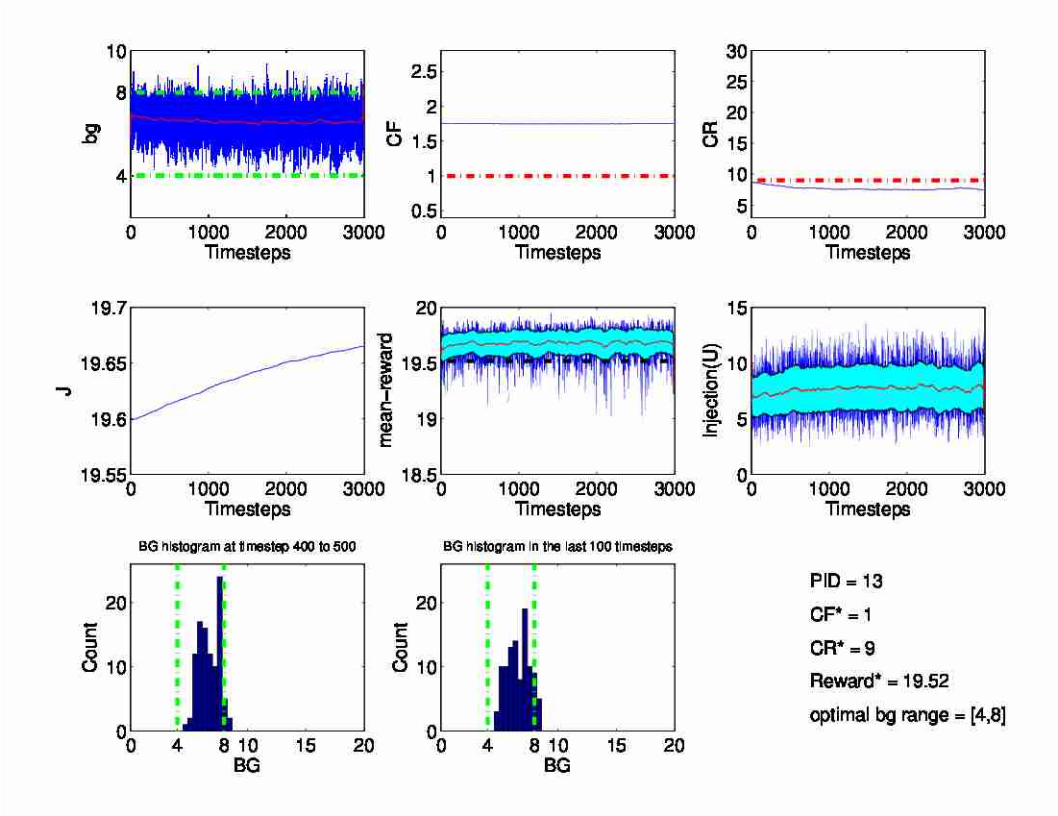


Figure B.13: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

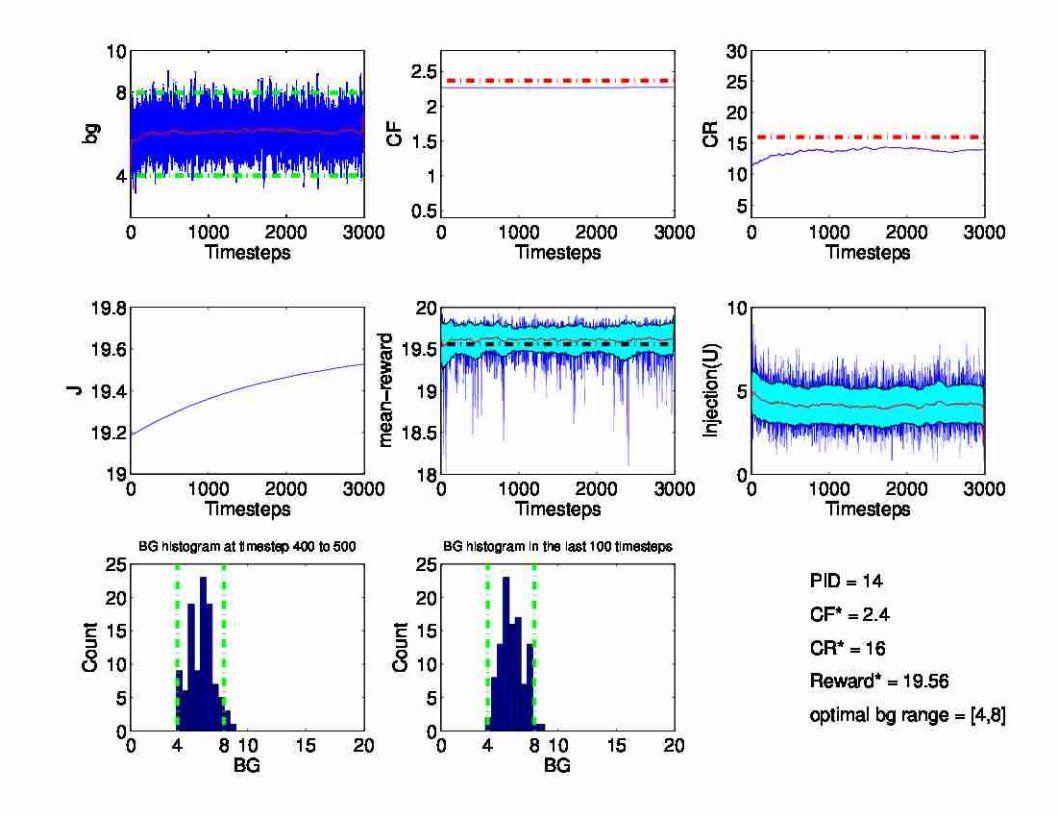


Figure B.14: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

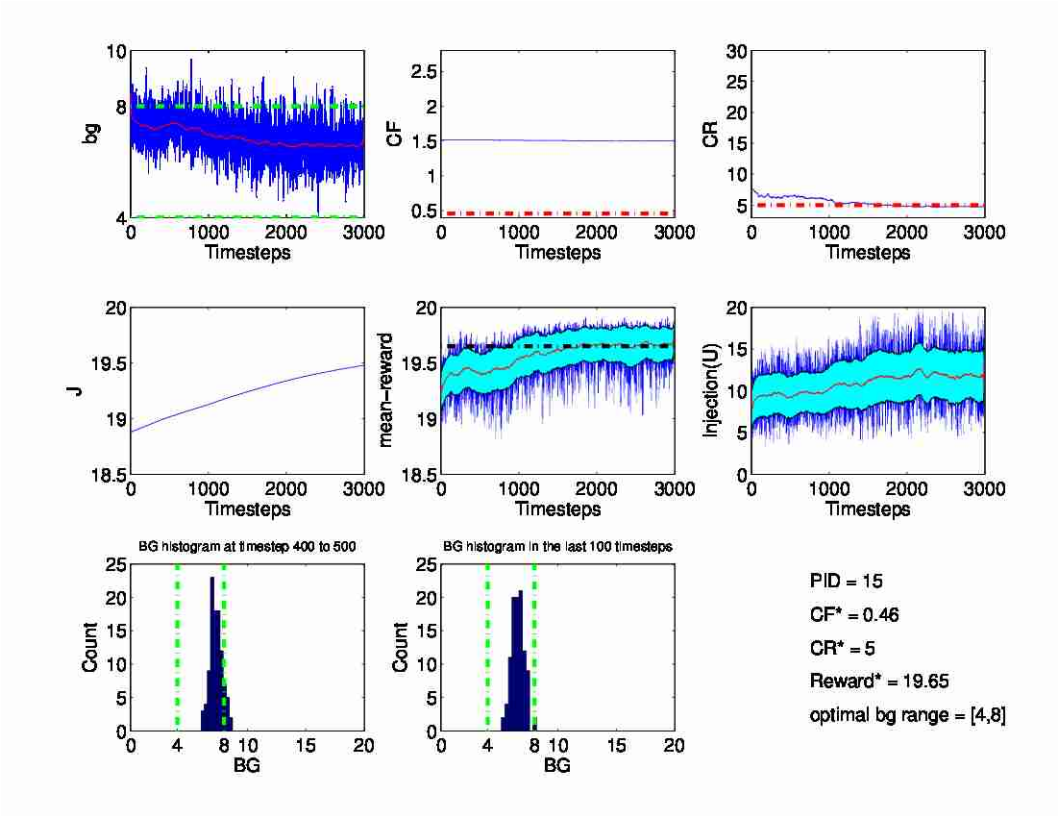


Figure B.15: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

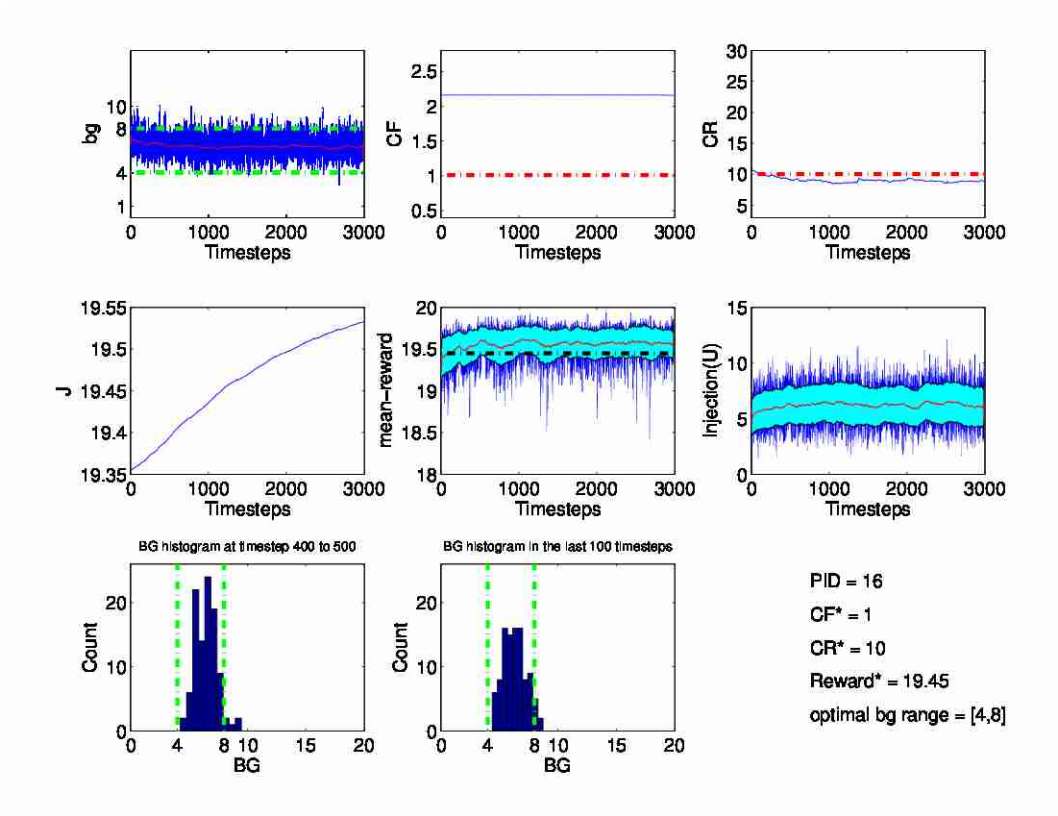


Figure B.16: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

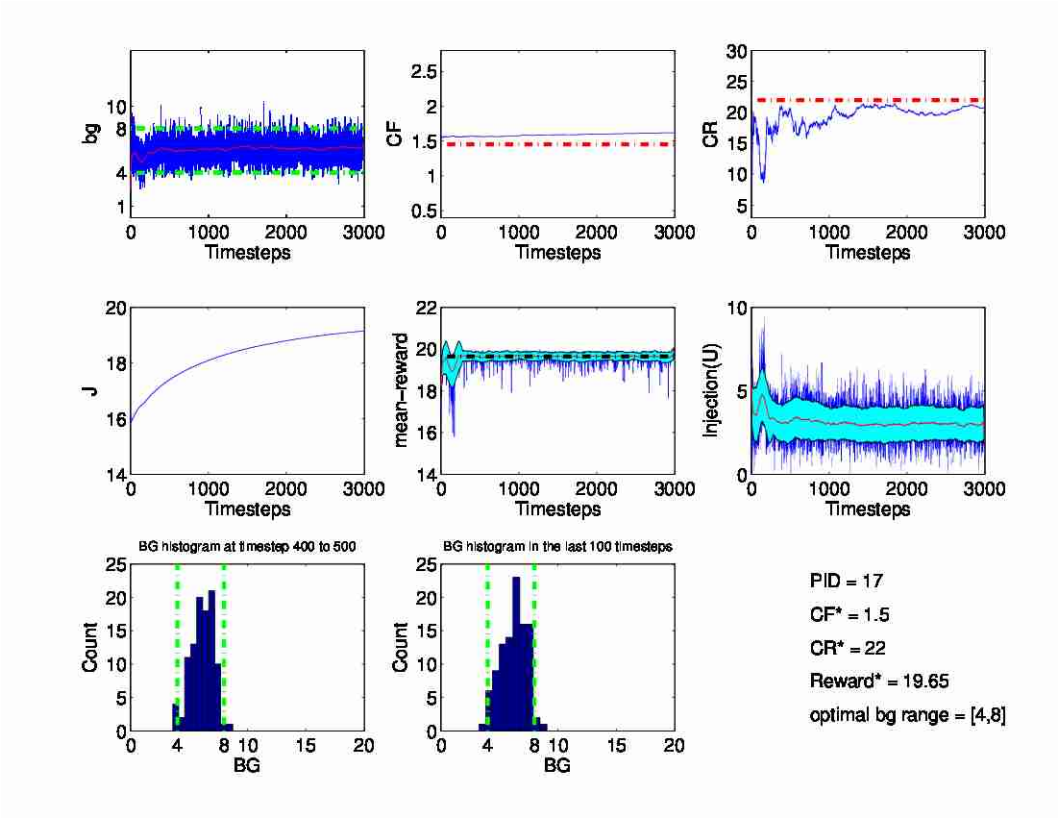


Figure B.17: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

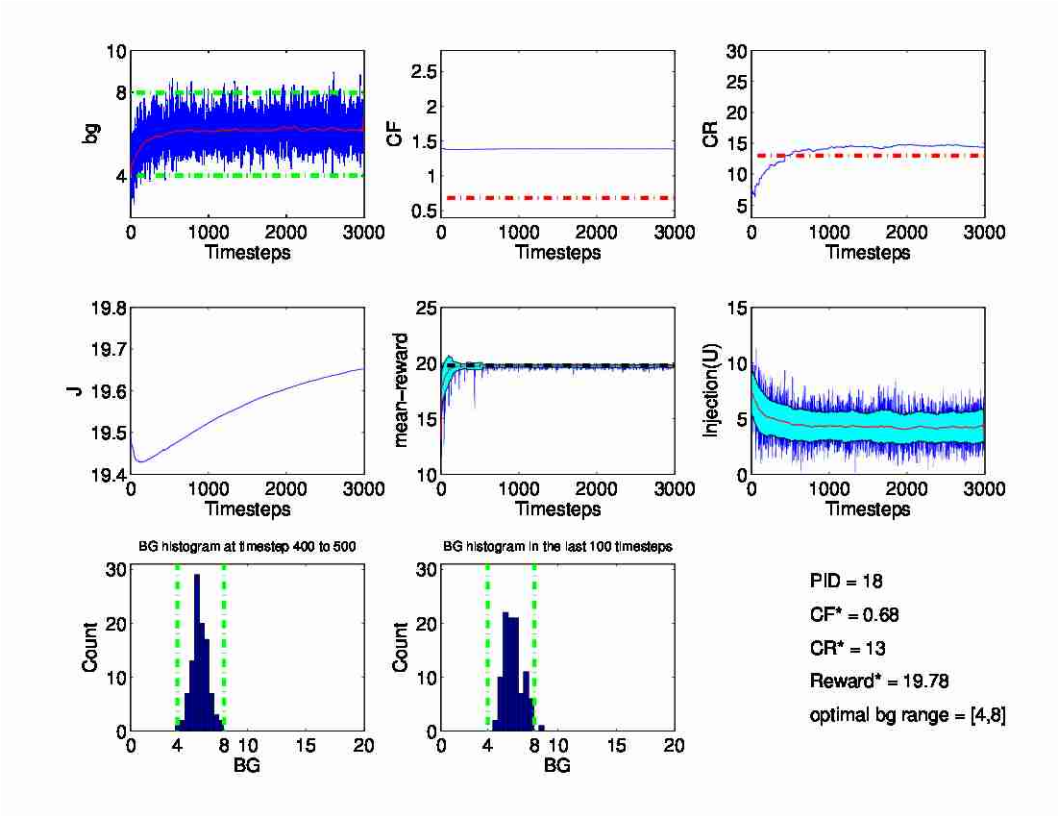


Figure B.18: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

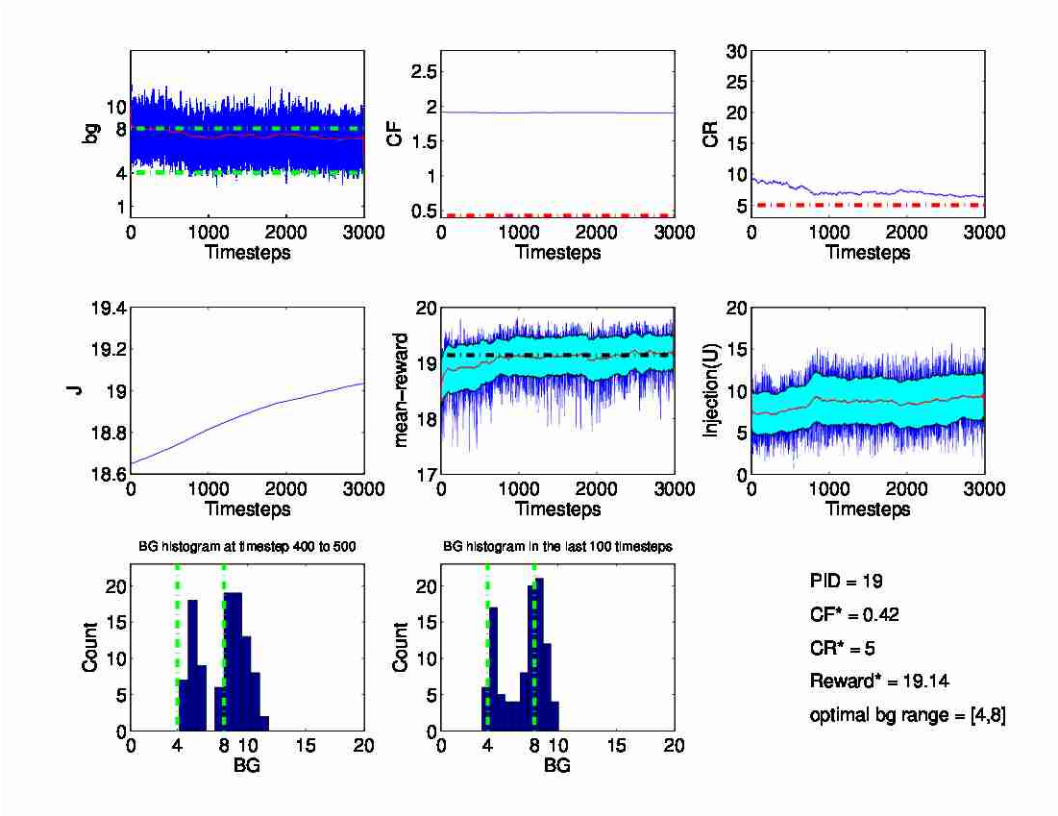


Figure B.19: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

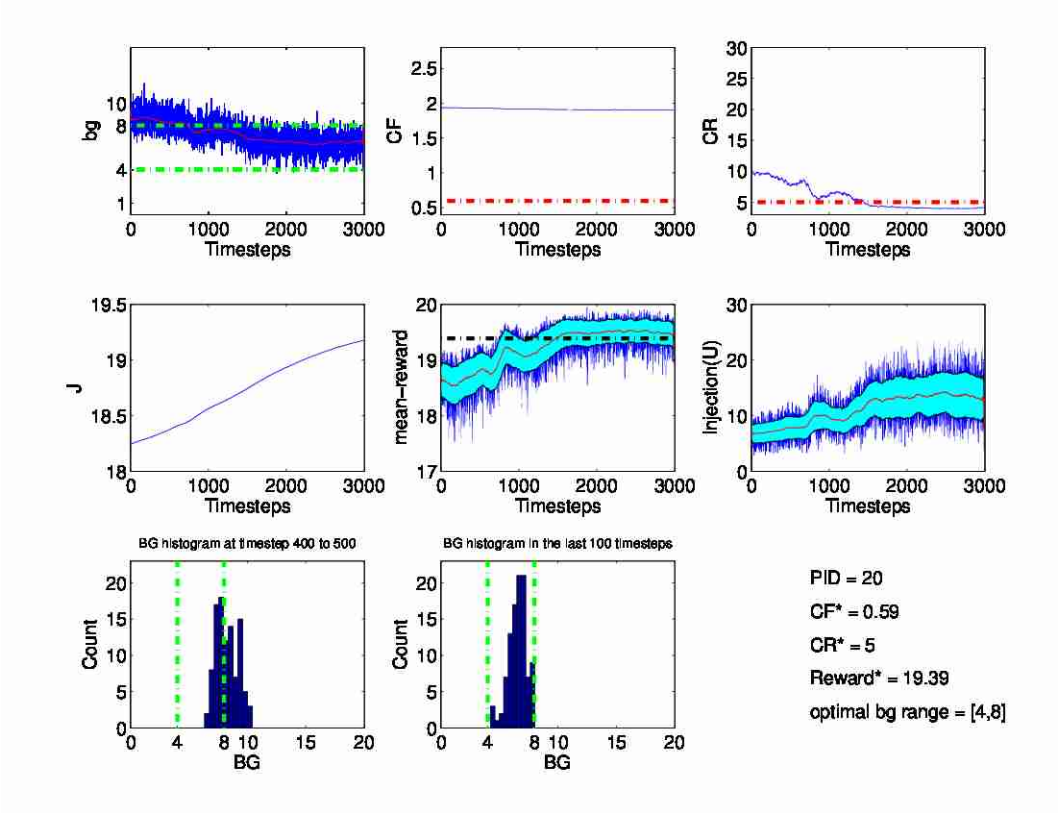


Figure B.20: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

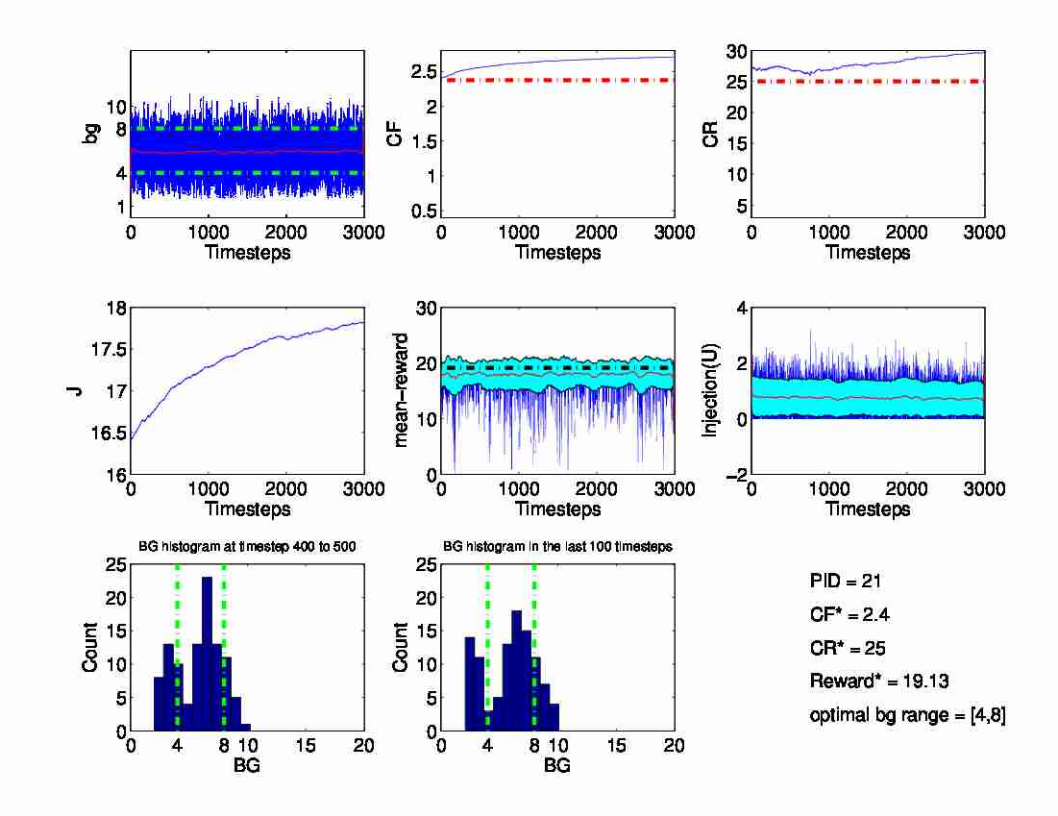


Figure B.21: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

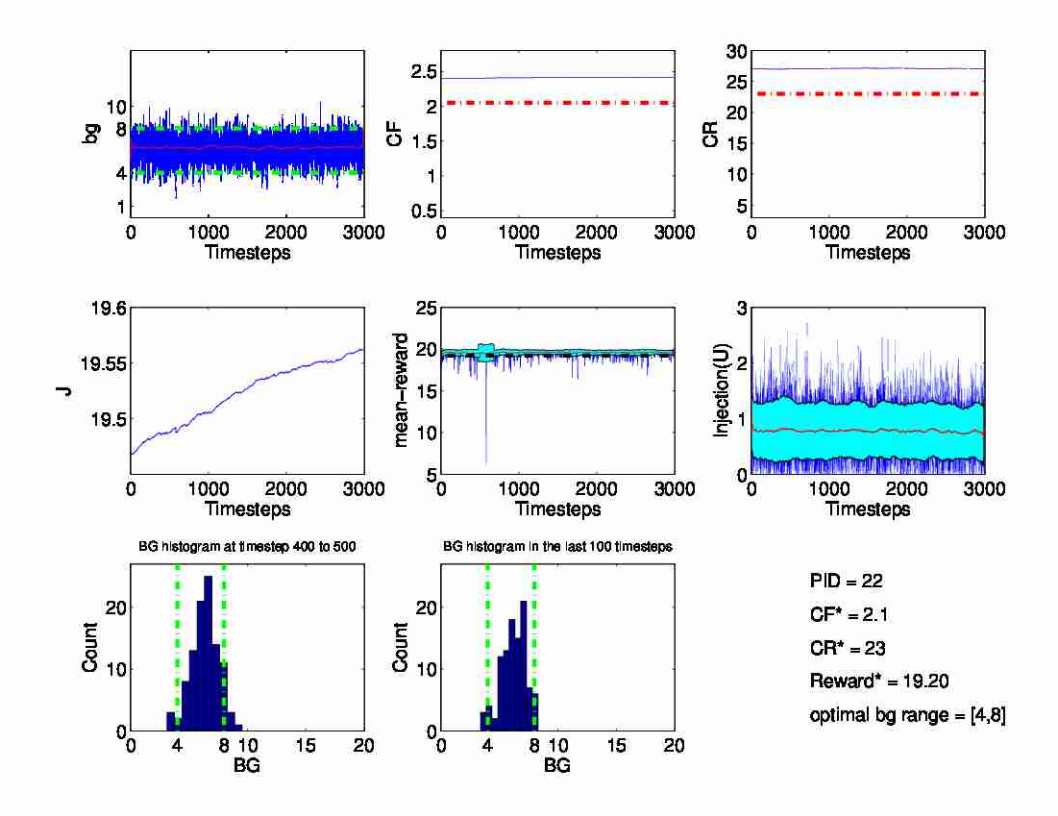


Figure B.22: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

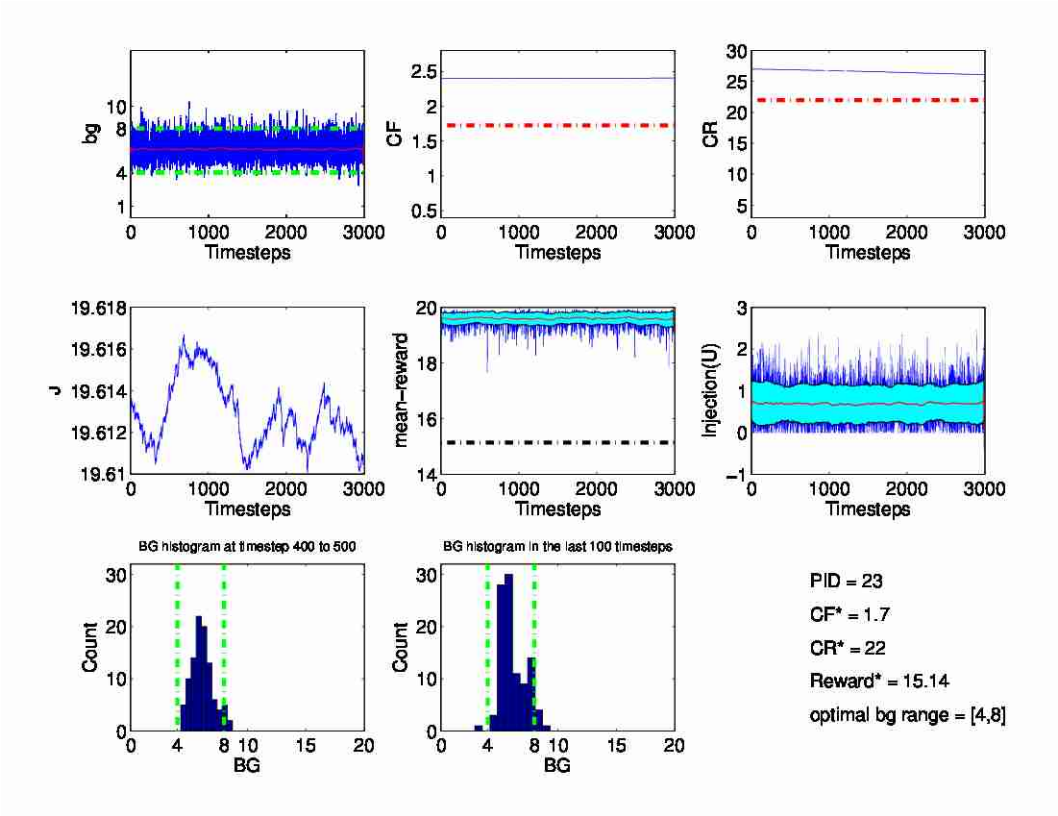


Figure B.23: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

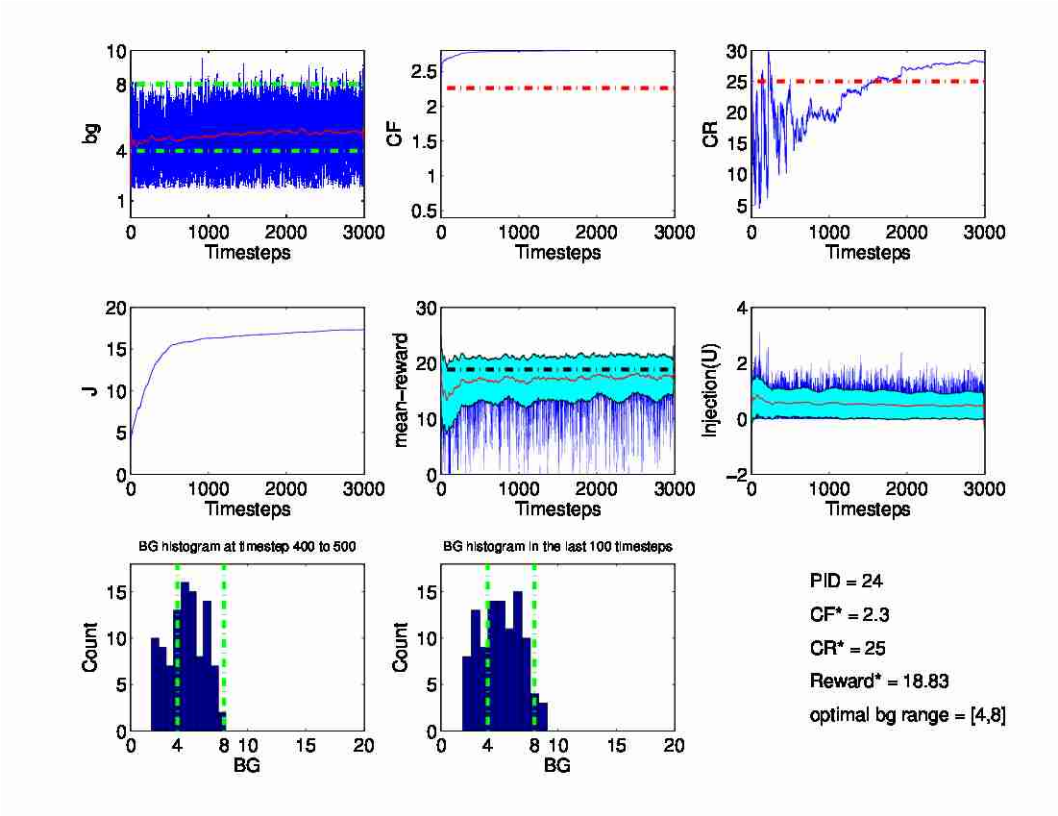


Figure B.24: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

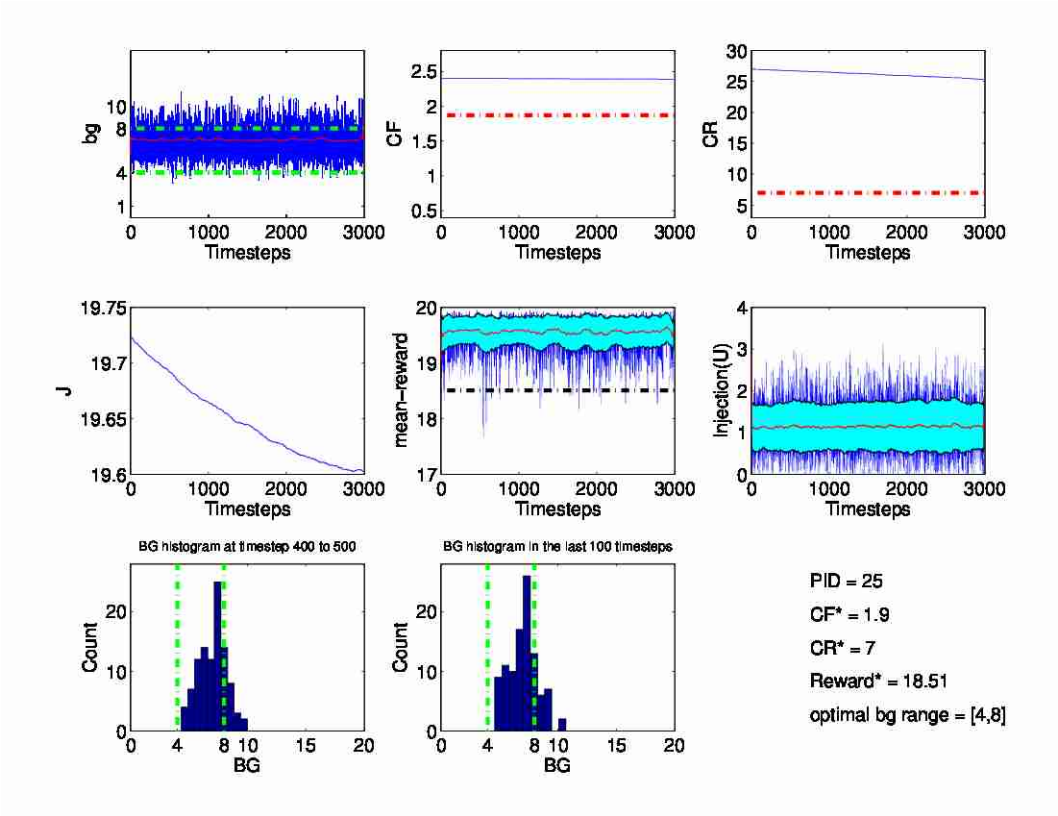


Figure B.25: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

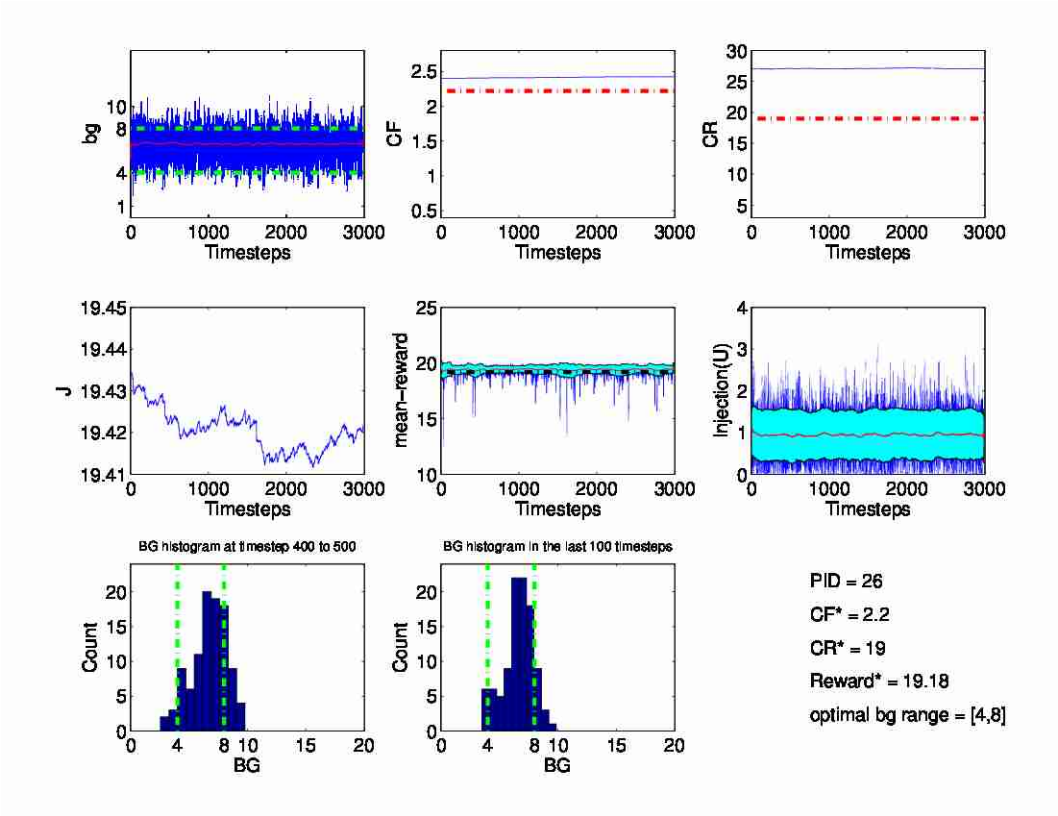


Figure B.26: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

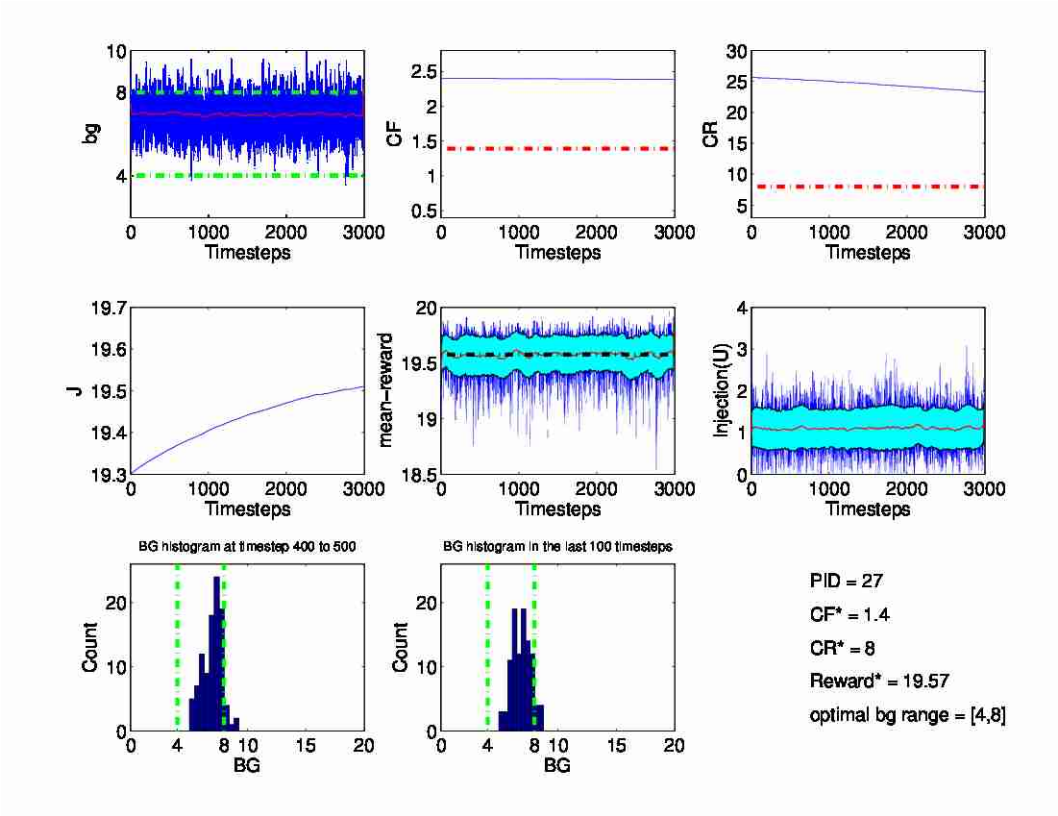


Figure B.27: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

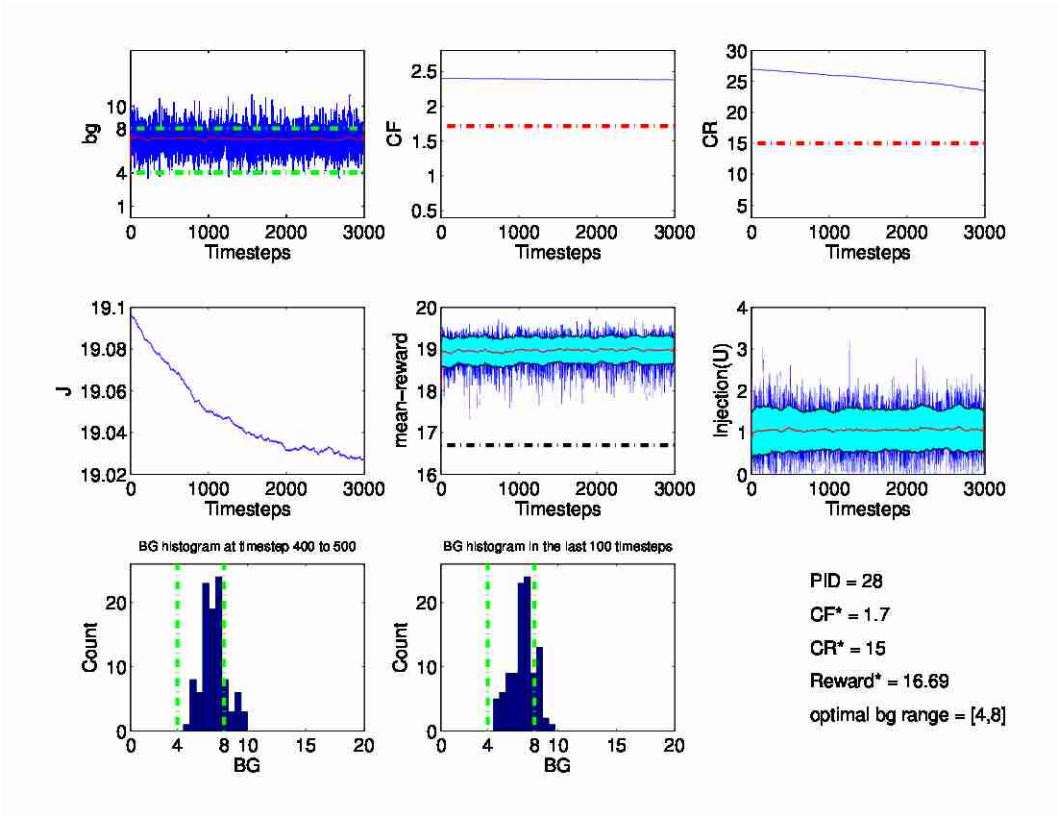


Figure B.28: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

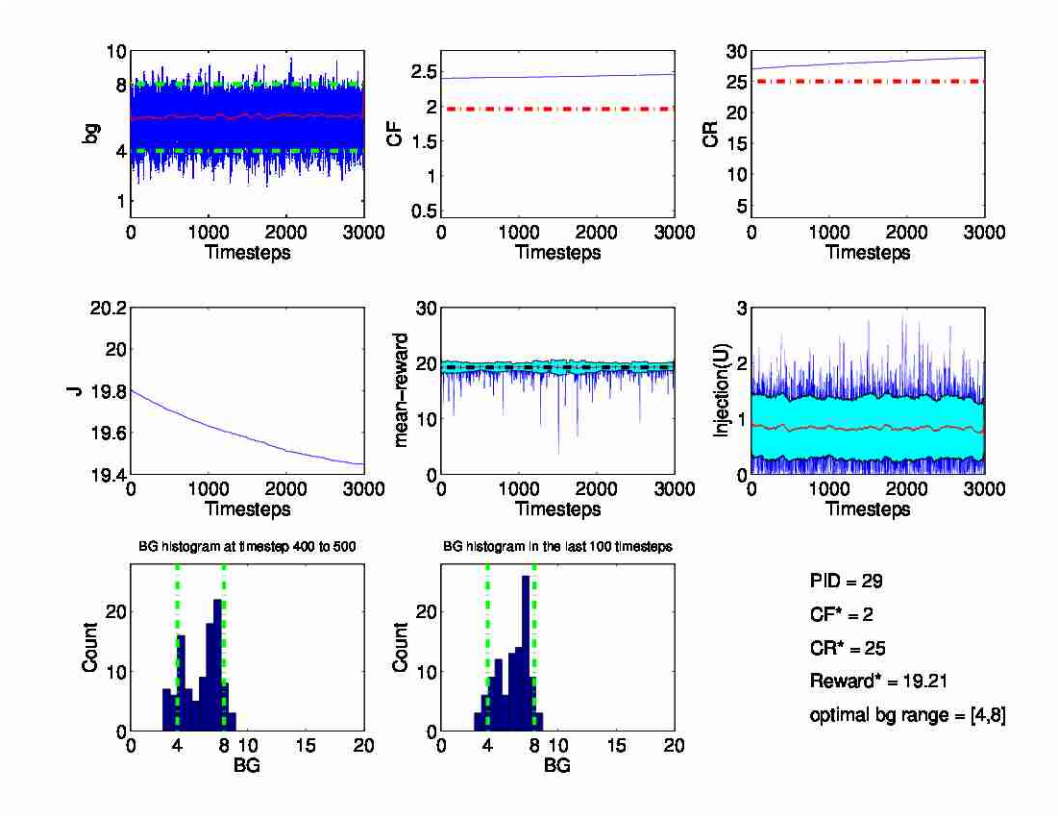


Figure B.29: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

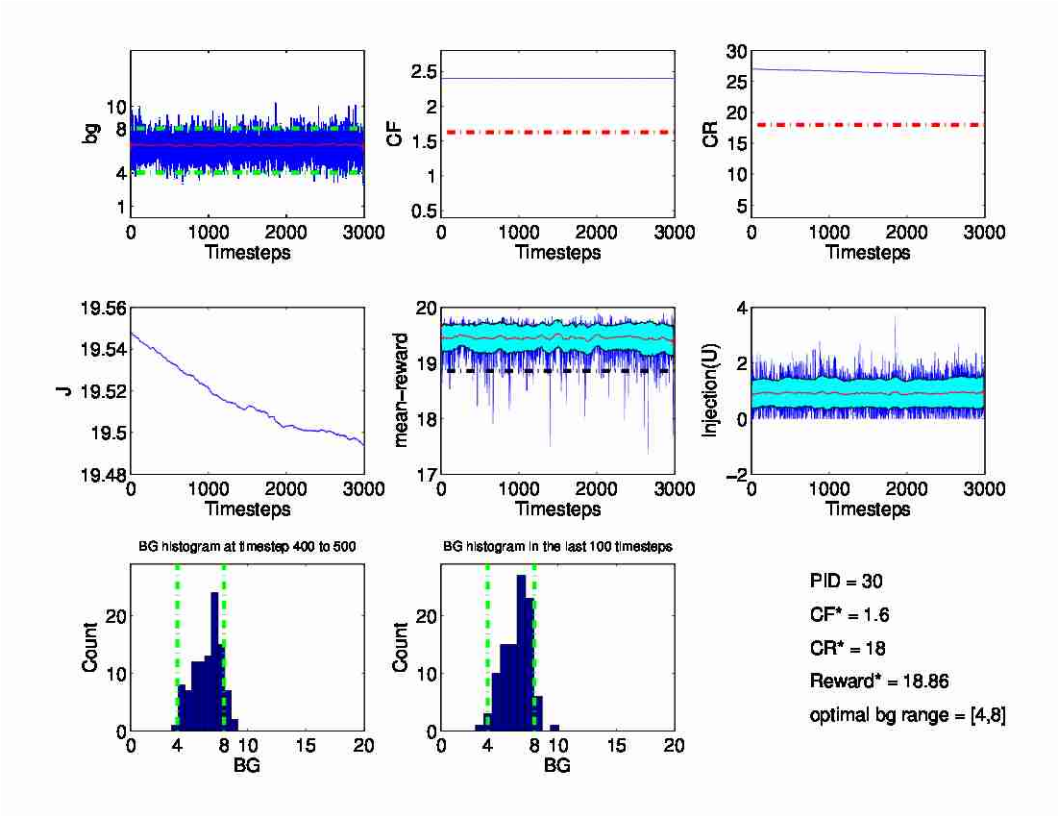


Figure B.30: The evolution of different variables during the learning phase of SAC (no-tile coding version). See Section 5.2.3 for more information.

B.2 SAC with tile-coding critic

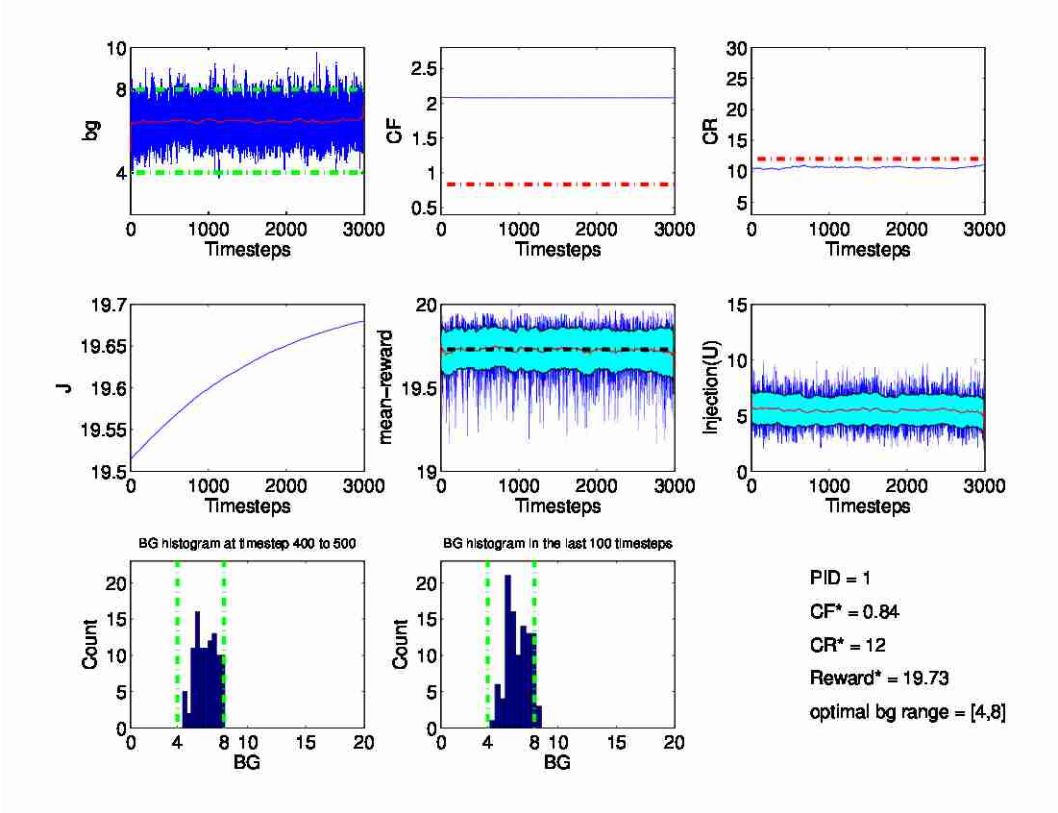


Figure B.31: The evolution of different variables during the learning phase of SAC (tile-coding version). See Section 5.2.3 for more information.

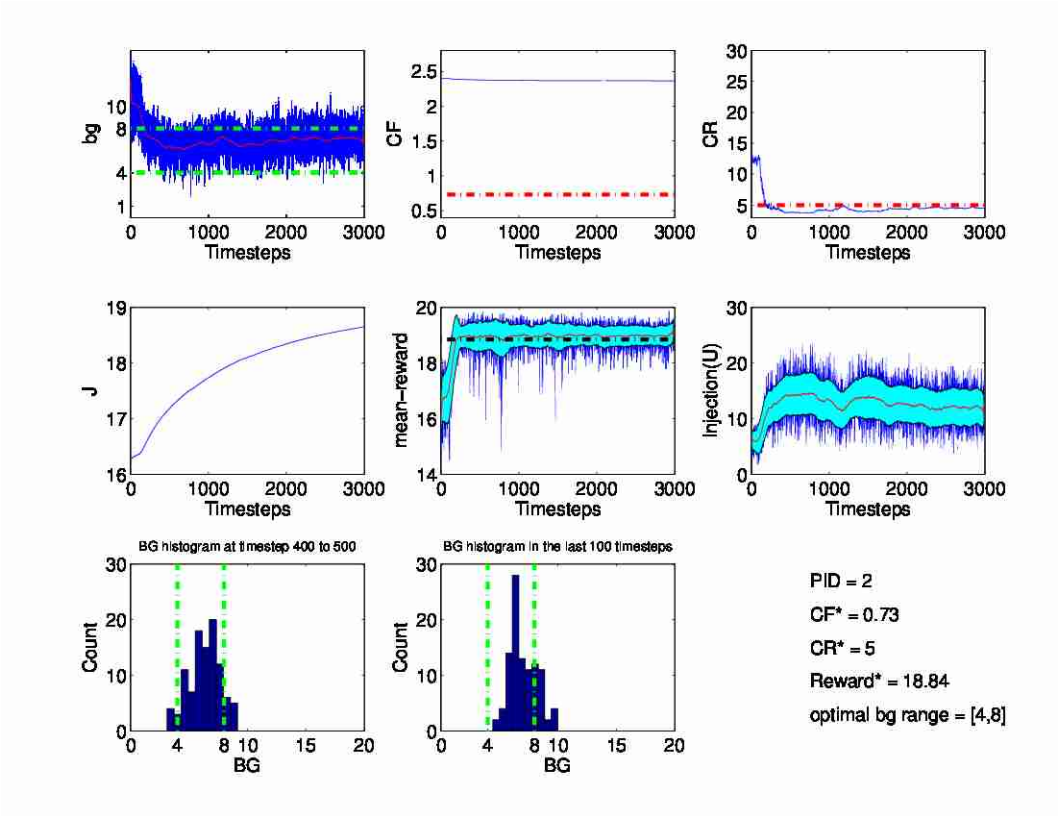


Figure B.32: The evolution of different variables during the learning phase of SAC (tile-coding version). See Section 5.2.3 for more information.

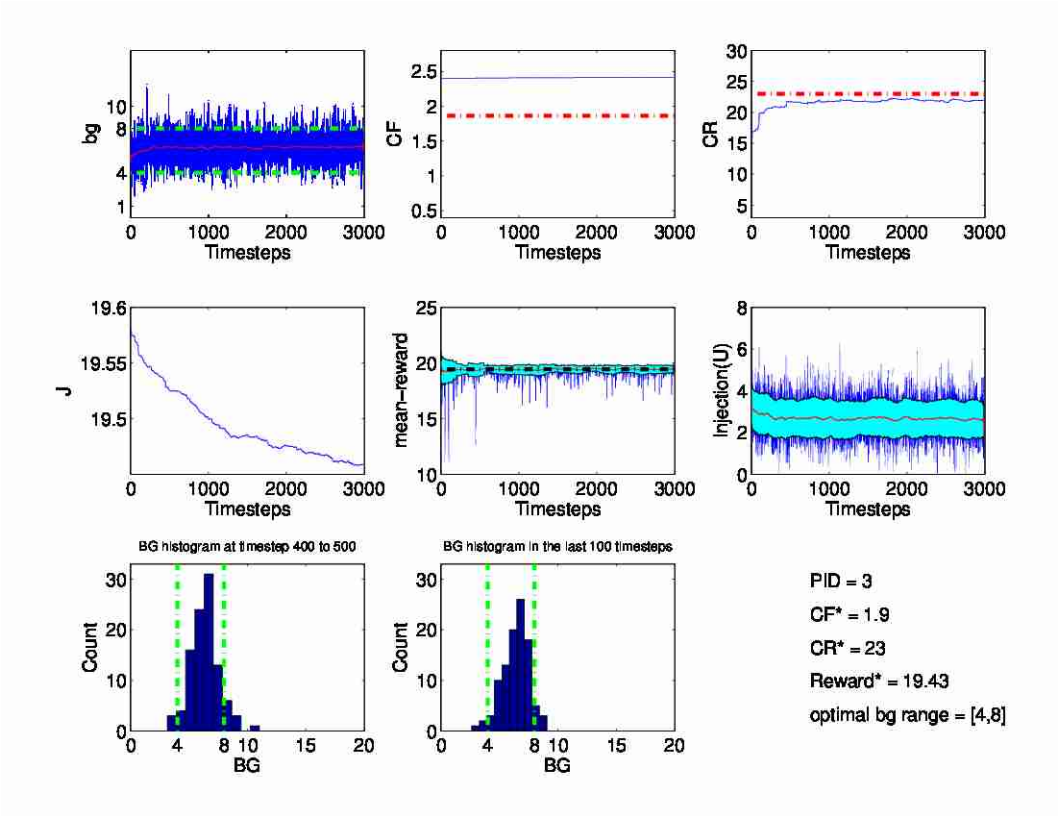


Figure B.33: The evolution of different variables during the learning phase of SAC (tile-coding version). See Section 5.2.3 for more information.

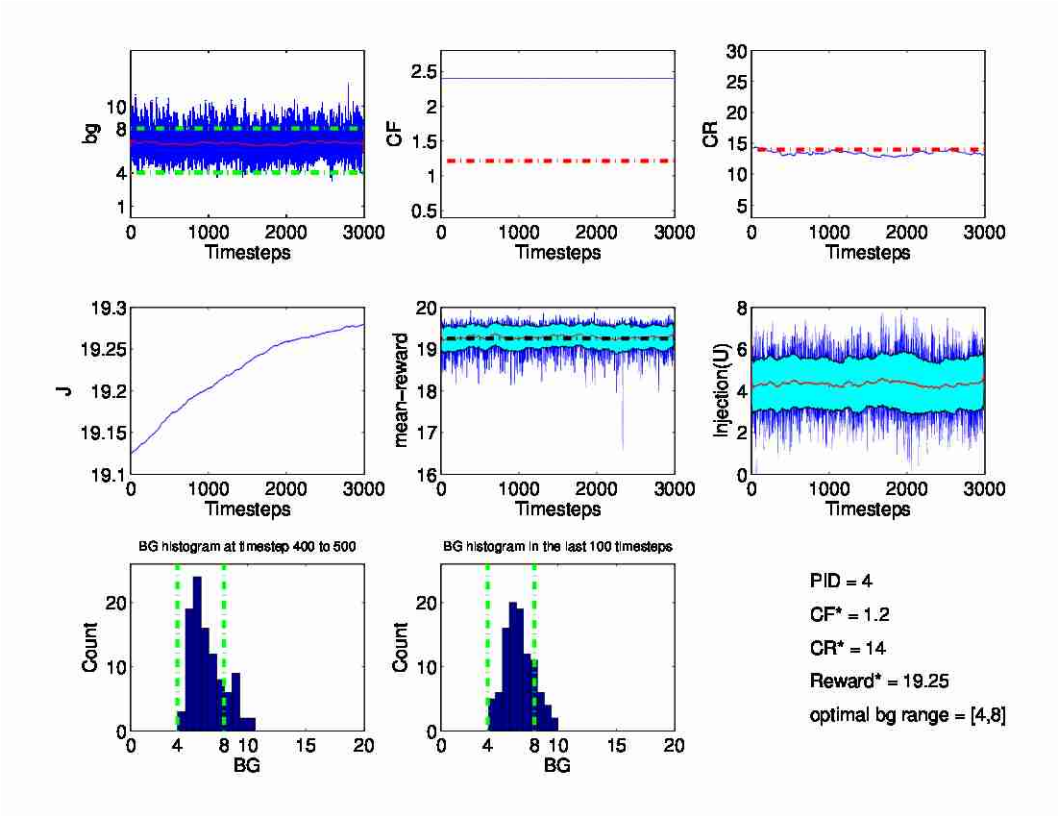


Figure B.34: The evolution of different variables during the learning phase of SAC (tile-coding version). See Section 5.2.3 for more information.

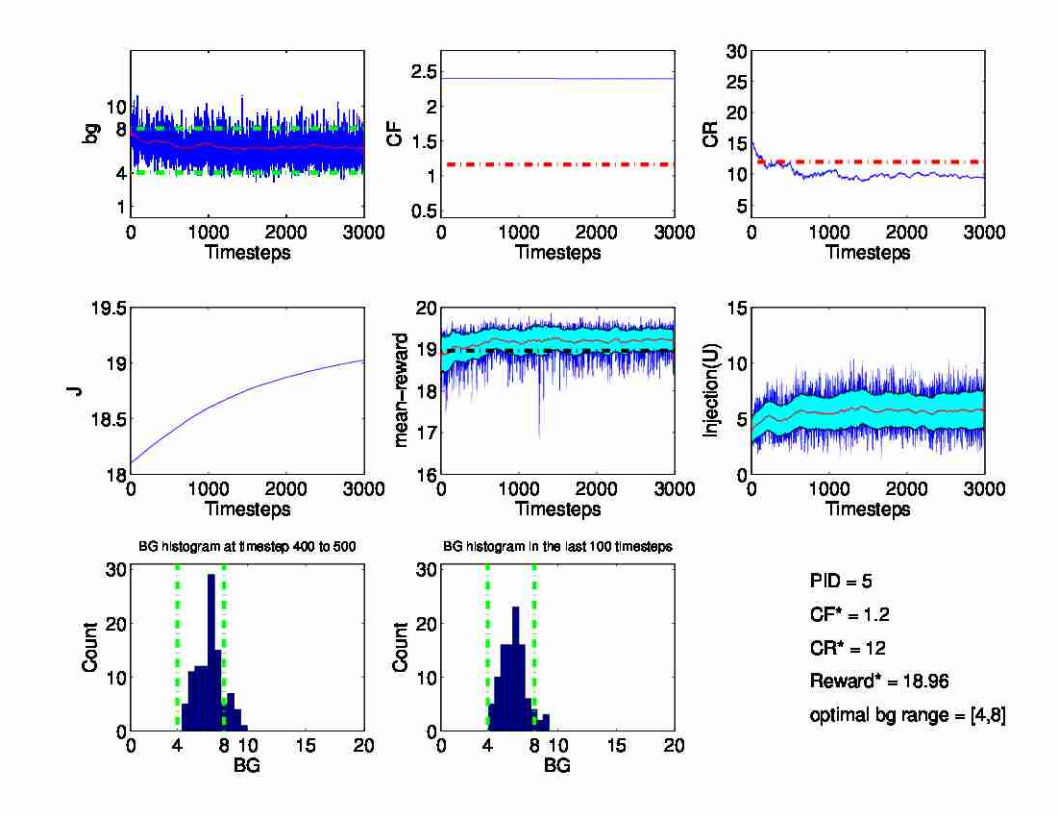


Figure B.35: The evolution of different variables during the learning phase of SAC (tile-coding version). See Section 5.2.3 for more information.

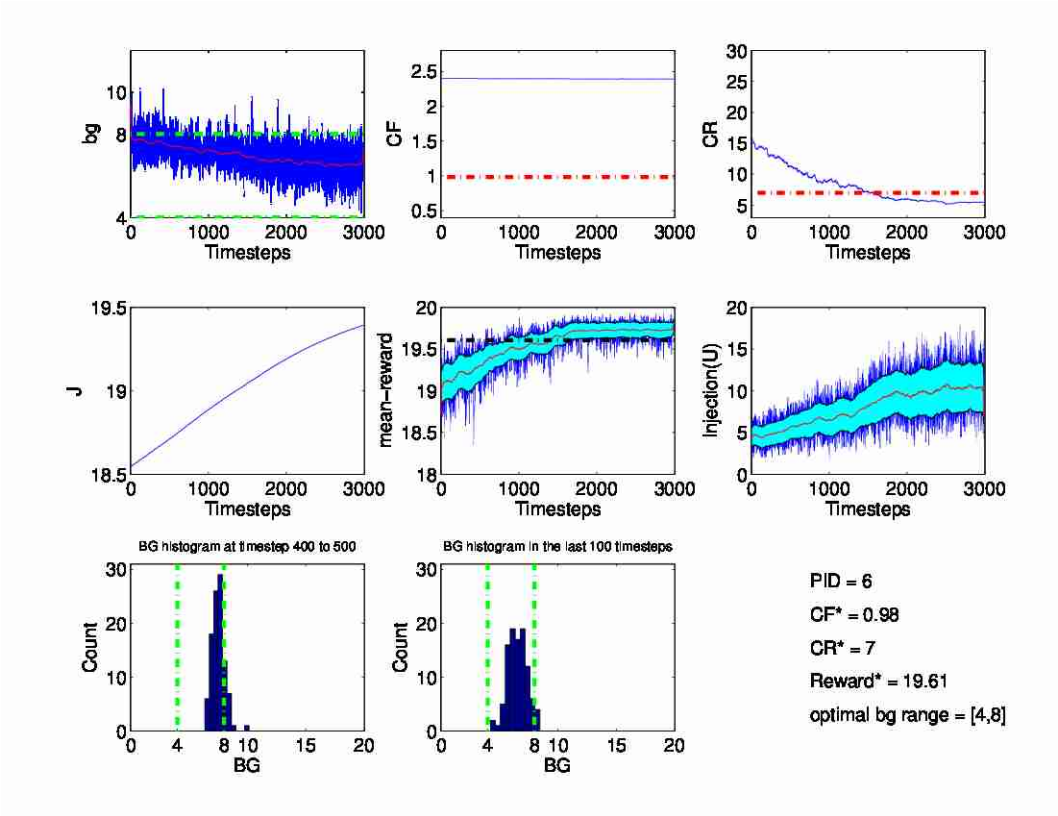


Figure B.36: The evolution of different variables during the learning phase of SAC (tile-coding version). See Section 5.2.3 for more information.

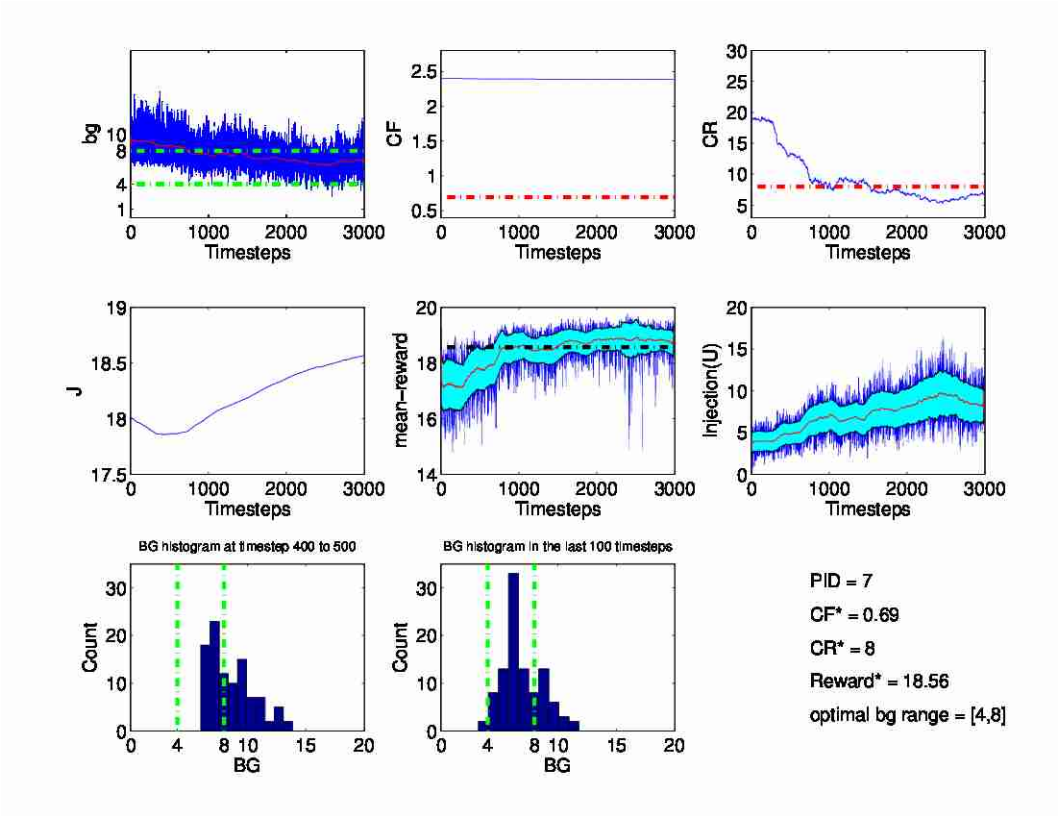


Figure B.37: The evolution of different variables during the learning phase of SAC (tile-coding version). See Section 5.2.3 for more information.

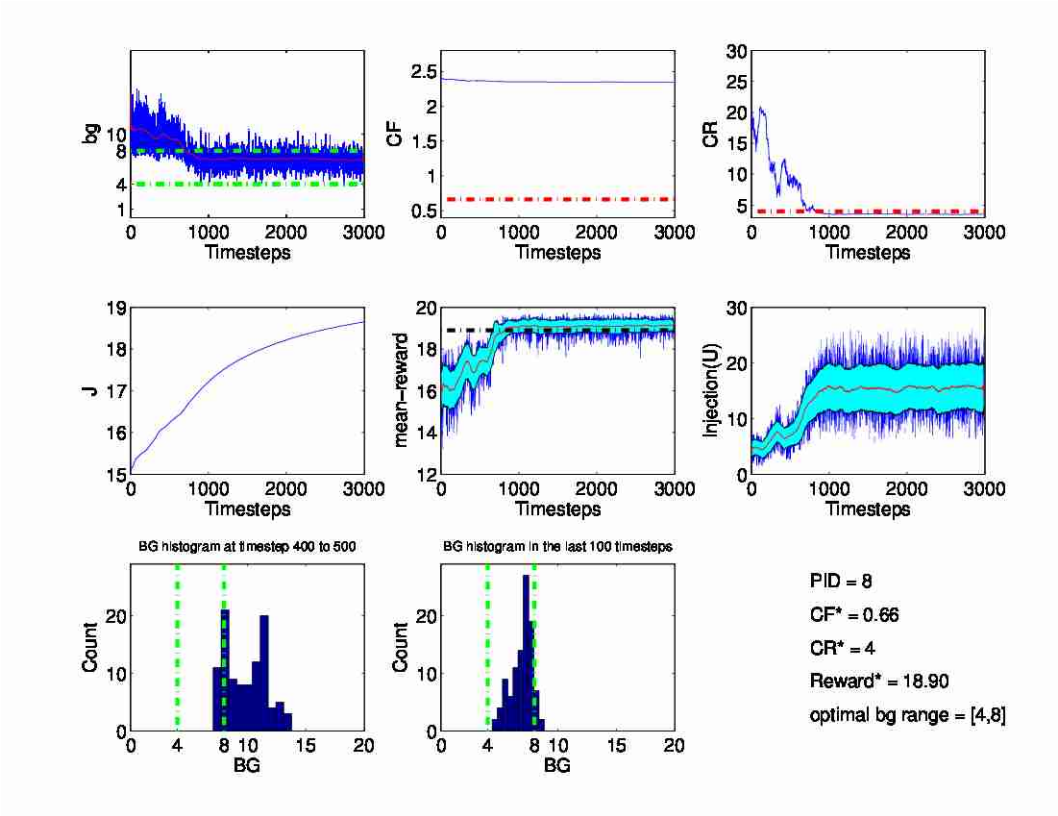


Figure B.38: The evolution of different variables during the learning phase of SAC (tile-coding version). See Section 5.2.3 for more information.

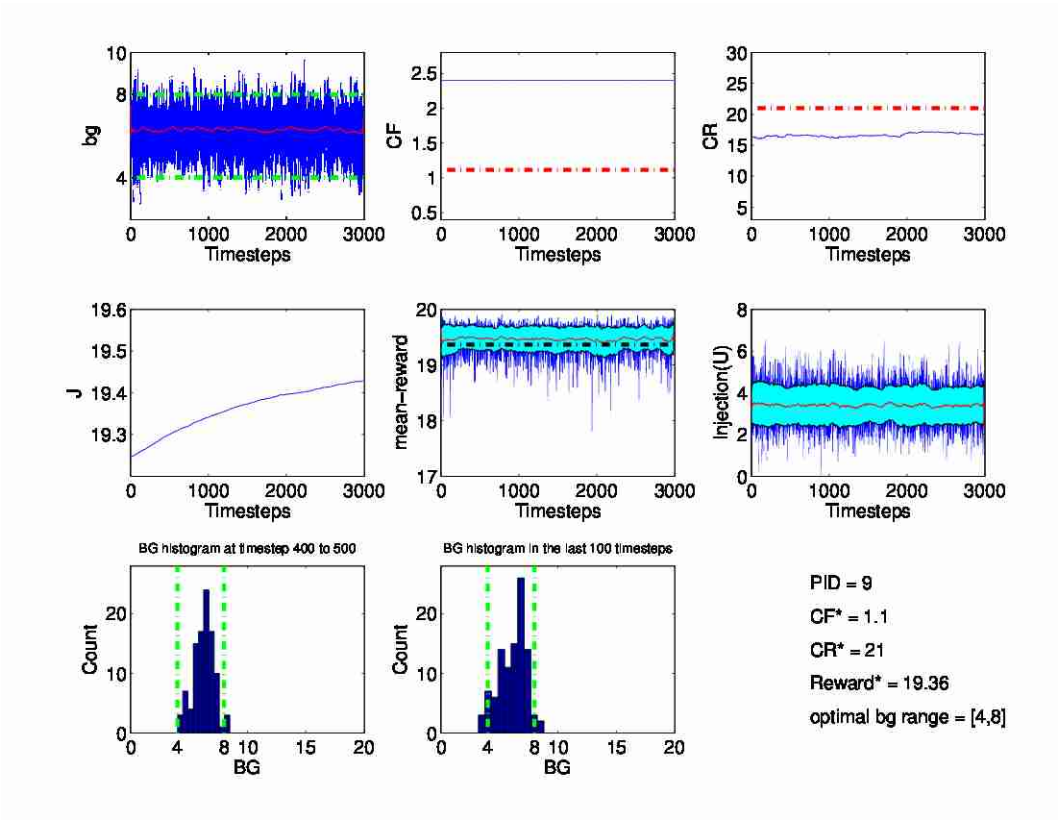


Figure B.39: The evolution of different variables during the learning phase of SAC (tile-coding version). See Section 5.2.3 for more information.

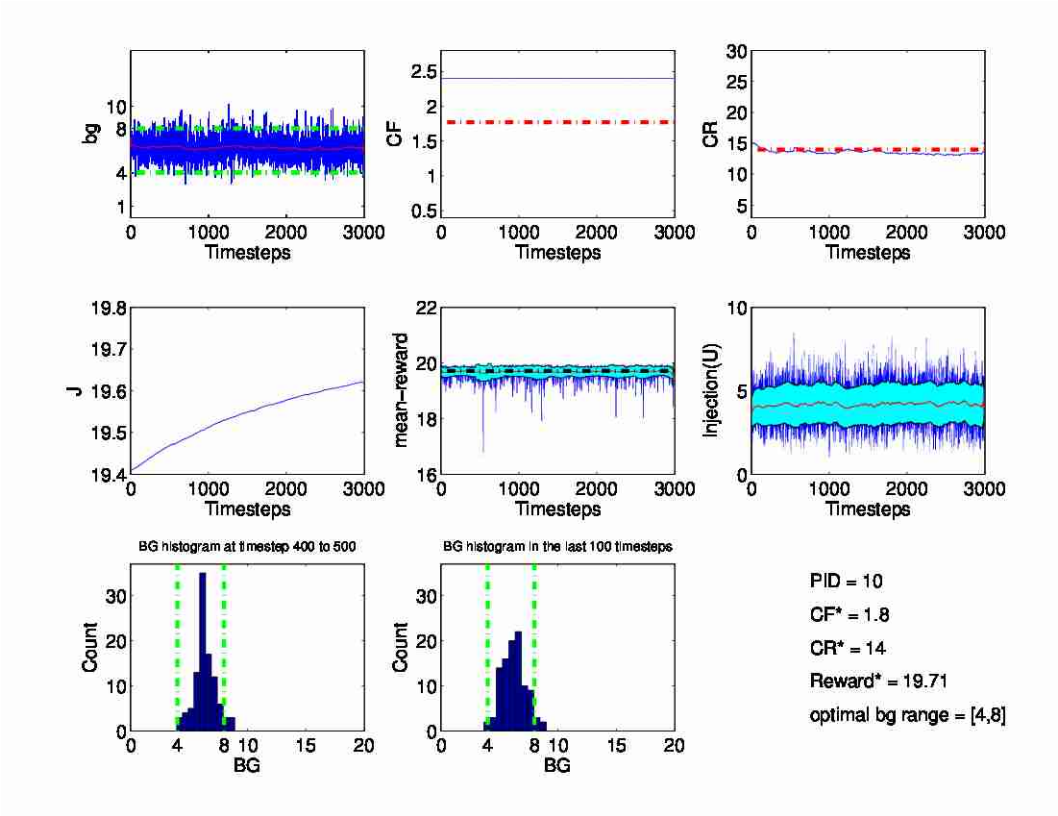


Figure B.40: The evolution of different variables during the learning phase of SAC (tile-coding version). See Section 5.2.3 for more information.

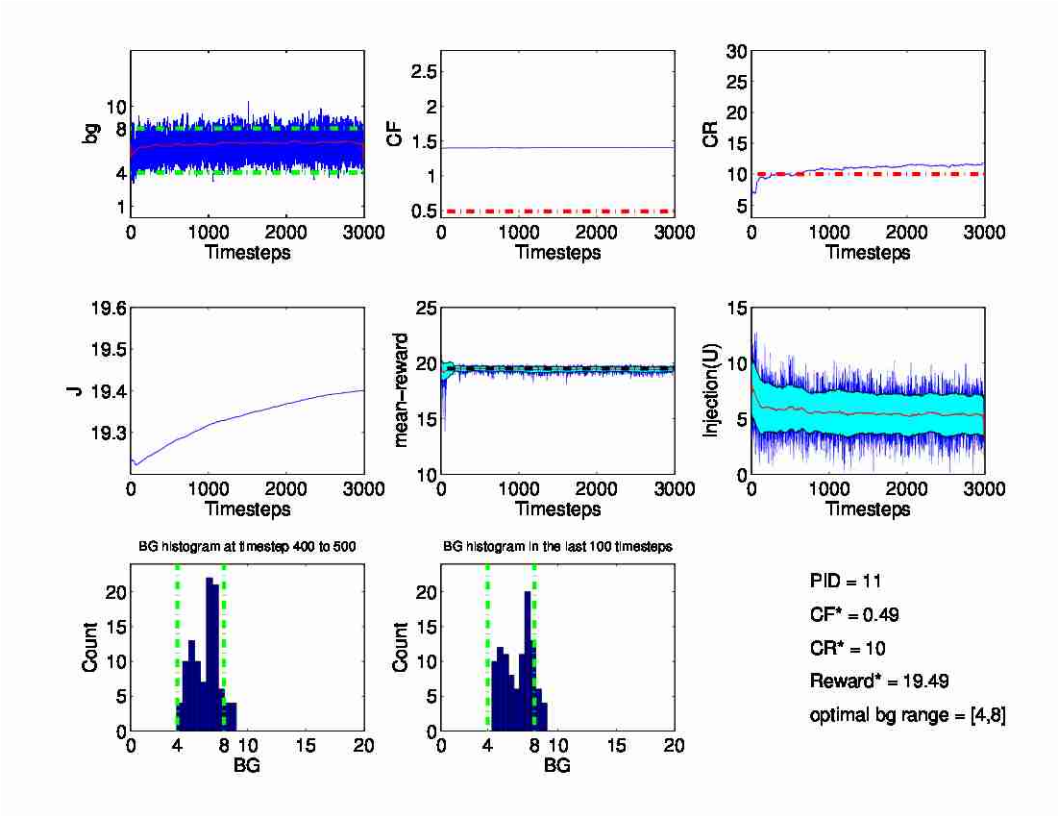


Figure B.41: The evolution of different variables during the learning phase of SAC (tile-coding version). See Section 5.2.3 for more information.

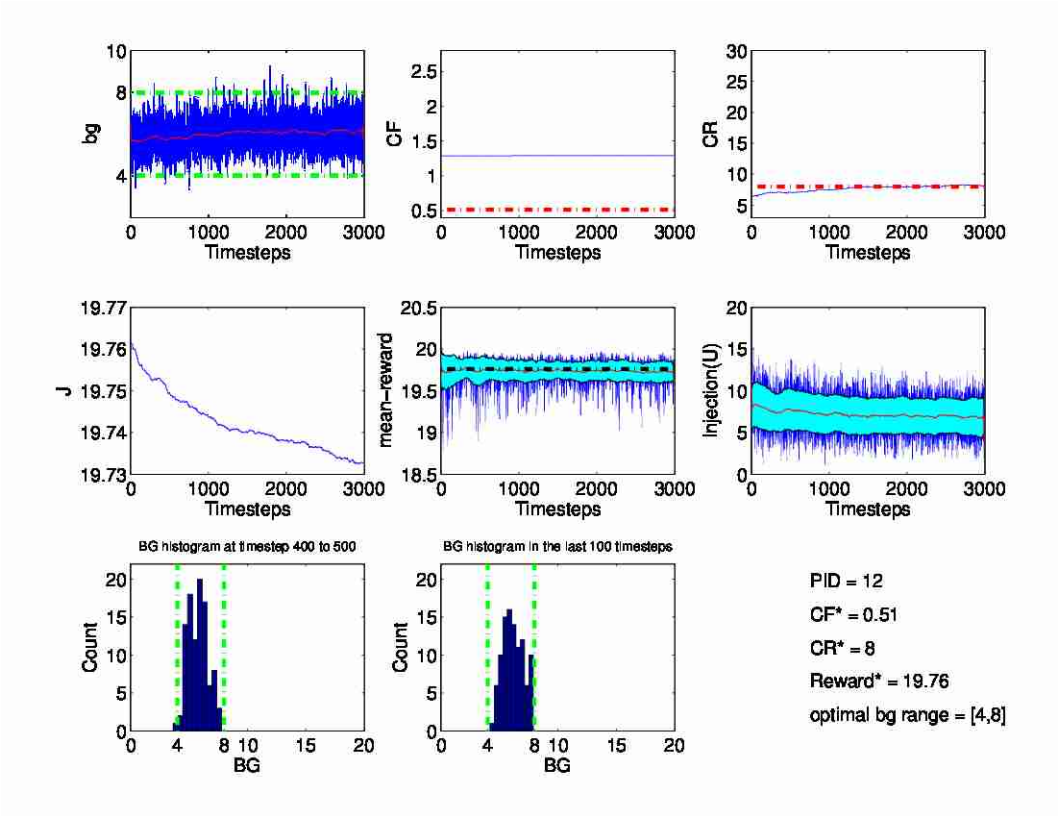


Figure B.42: The evolution of different variables during the learning phase of SAC (tile-coding version). See Section 5.2.3 for more information.

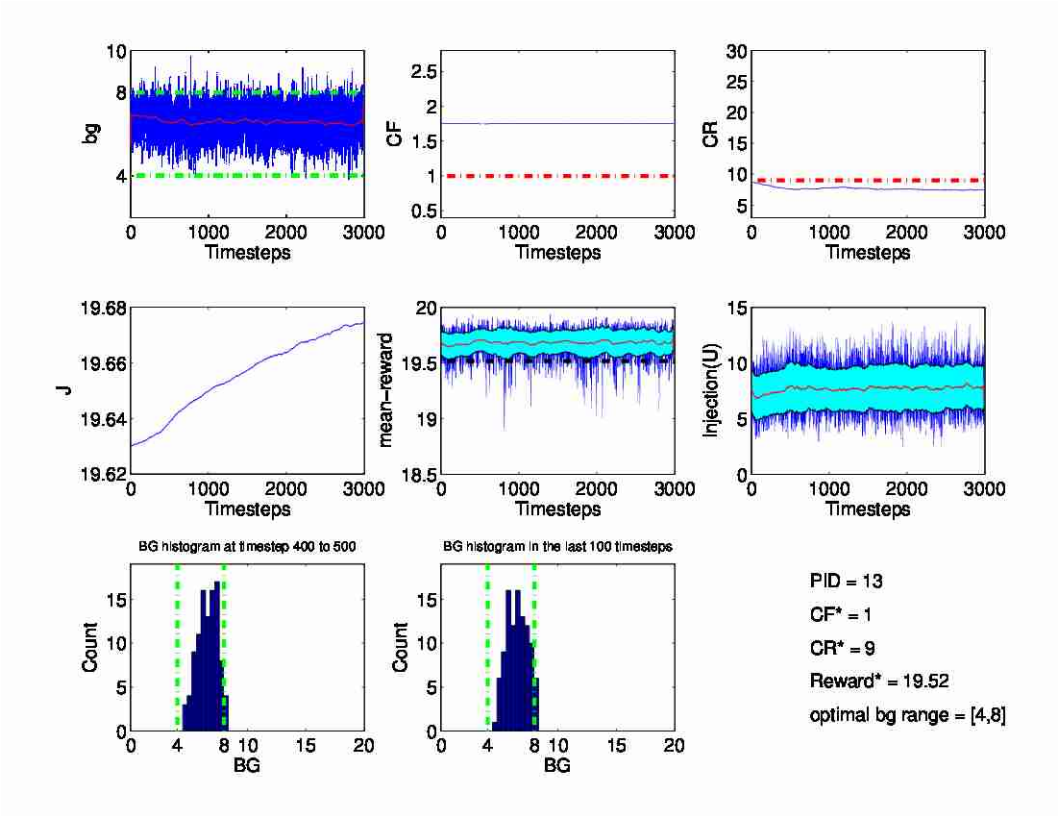


Figure B.43: The evolution of different variables during the learning phase of SAC (tile-coding version). See Section 5.2.3 for more information.

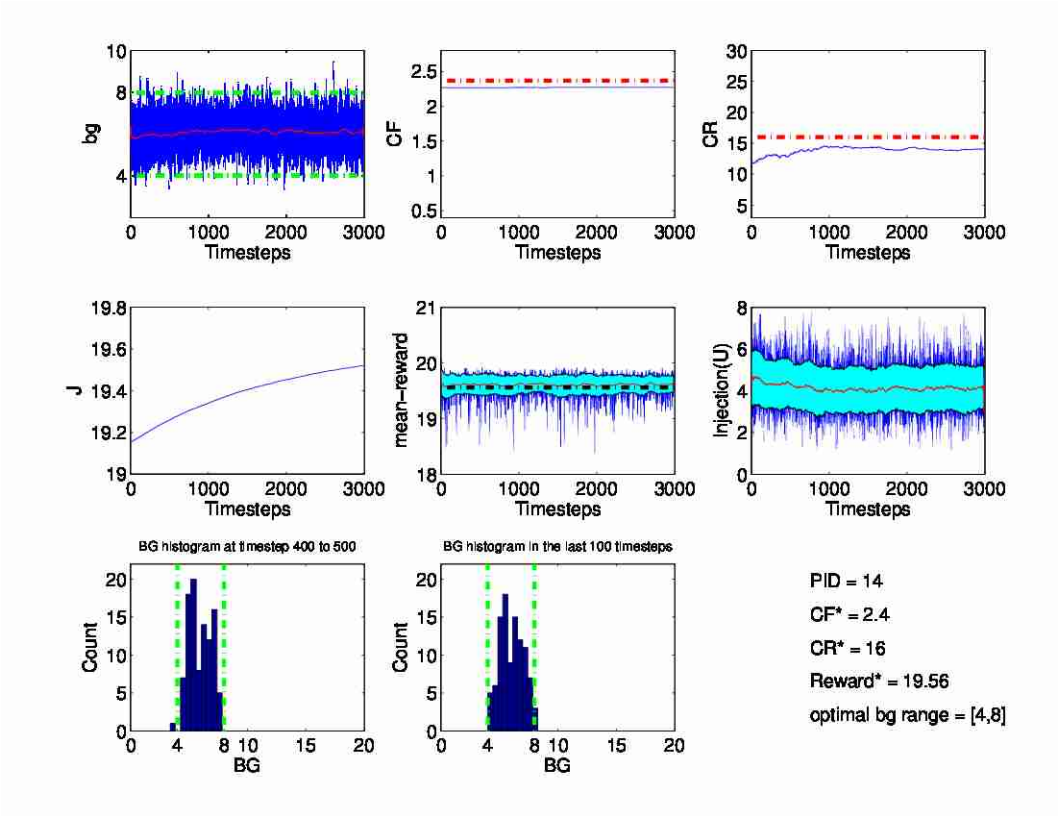


Figure B.44: The evolution of different variables during the learning phase of SAC (tile-coding version). See Section 5.2.3 for more information.

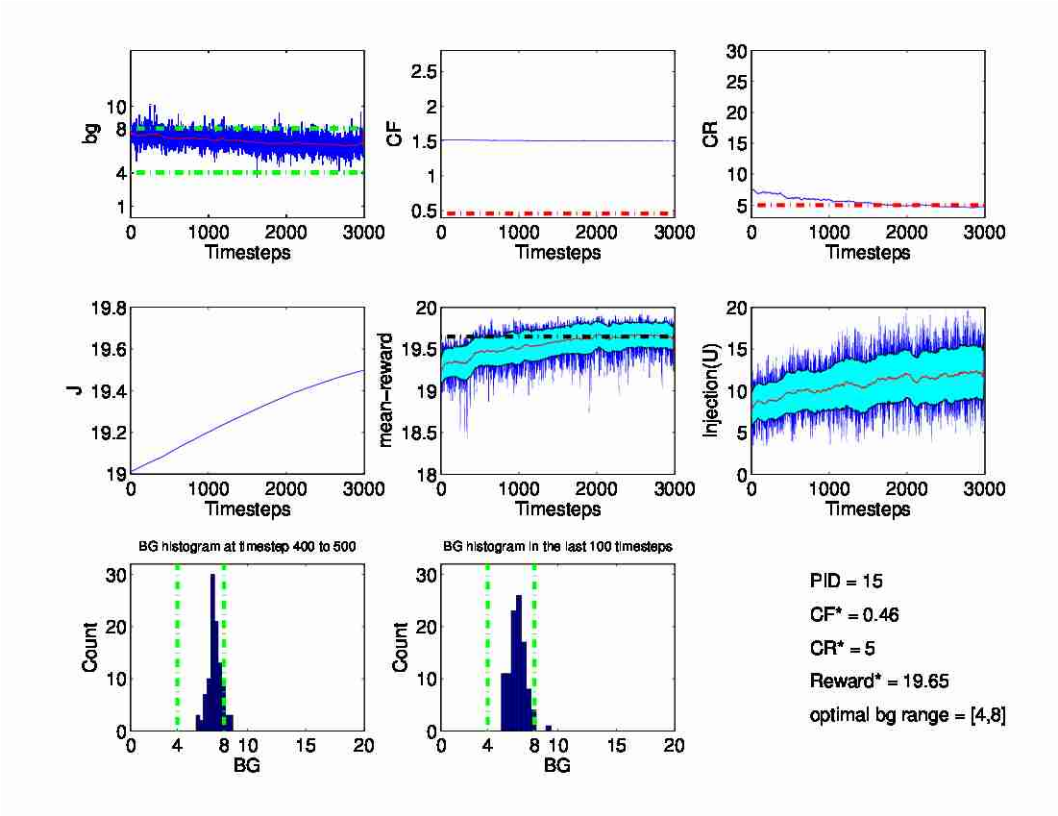


Figure B.45: The evolution of different variables during the learning phase of SAC (tile-coding version). See Section 5.2.3 for more information.

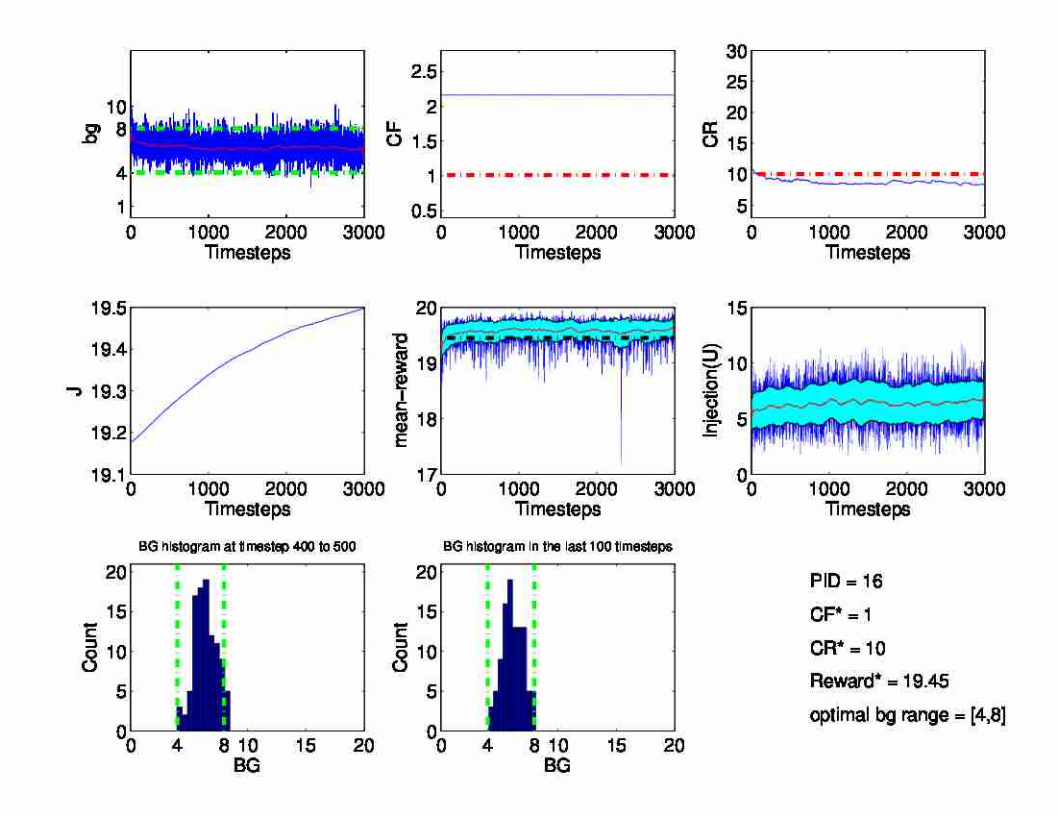


Figure B.46: The evolution of different variables during the learning phase of SAC (tile-coding version). See Section 5.2.3 for more information.

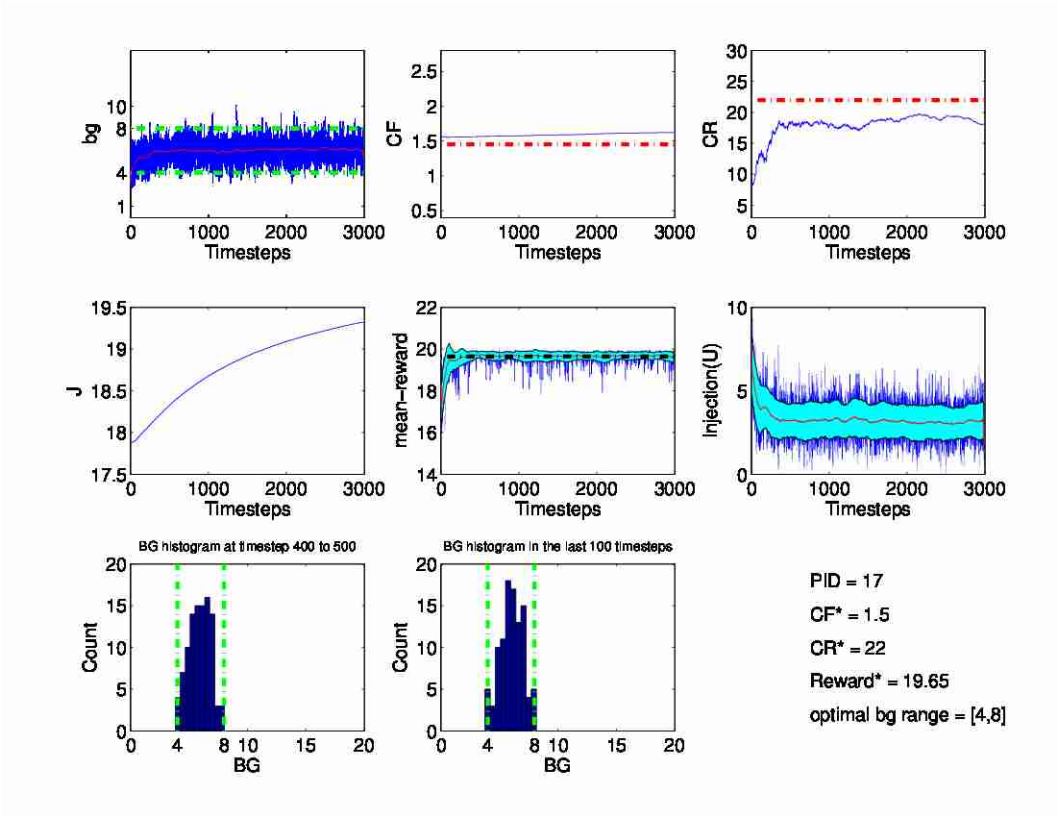


Figure B.47: The evolution of different variables during the learning phase of SAC (tile-coding version). See Section 5.2.3 for more information.

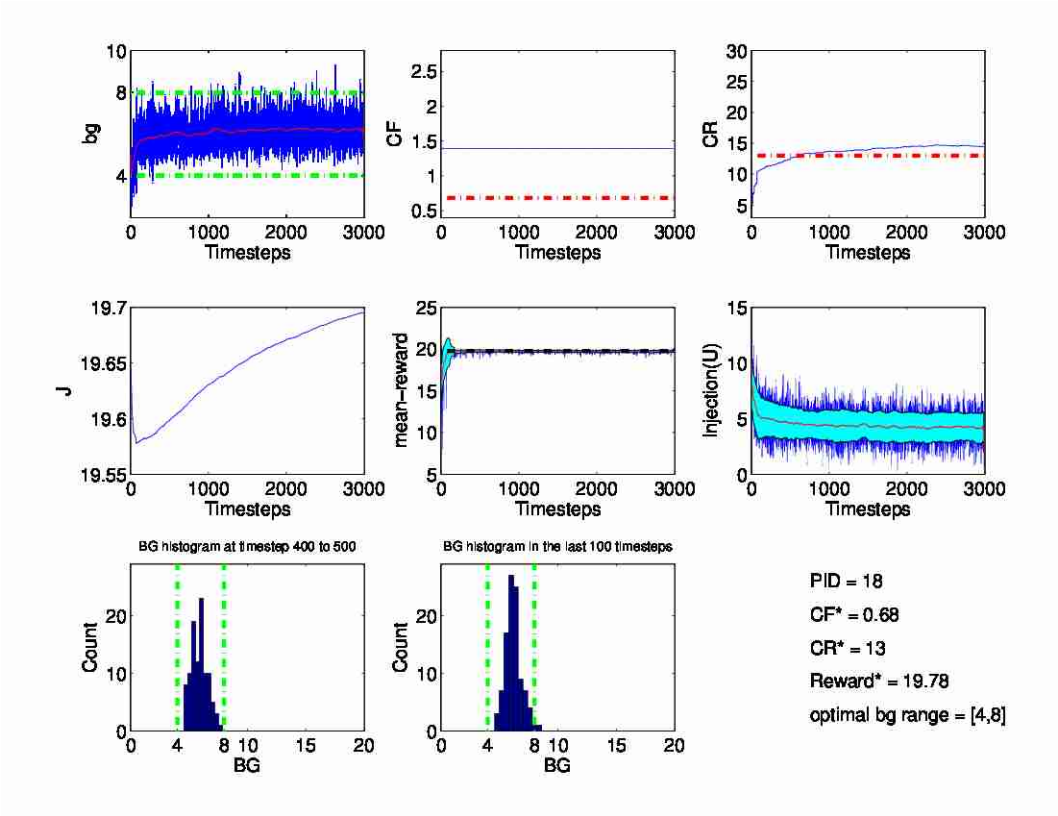


Figure B.48: The evolution of different variables during the learning phase of SAC (tile-coding version). See Section 5.2.3 for more information.

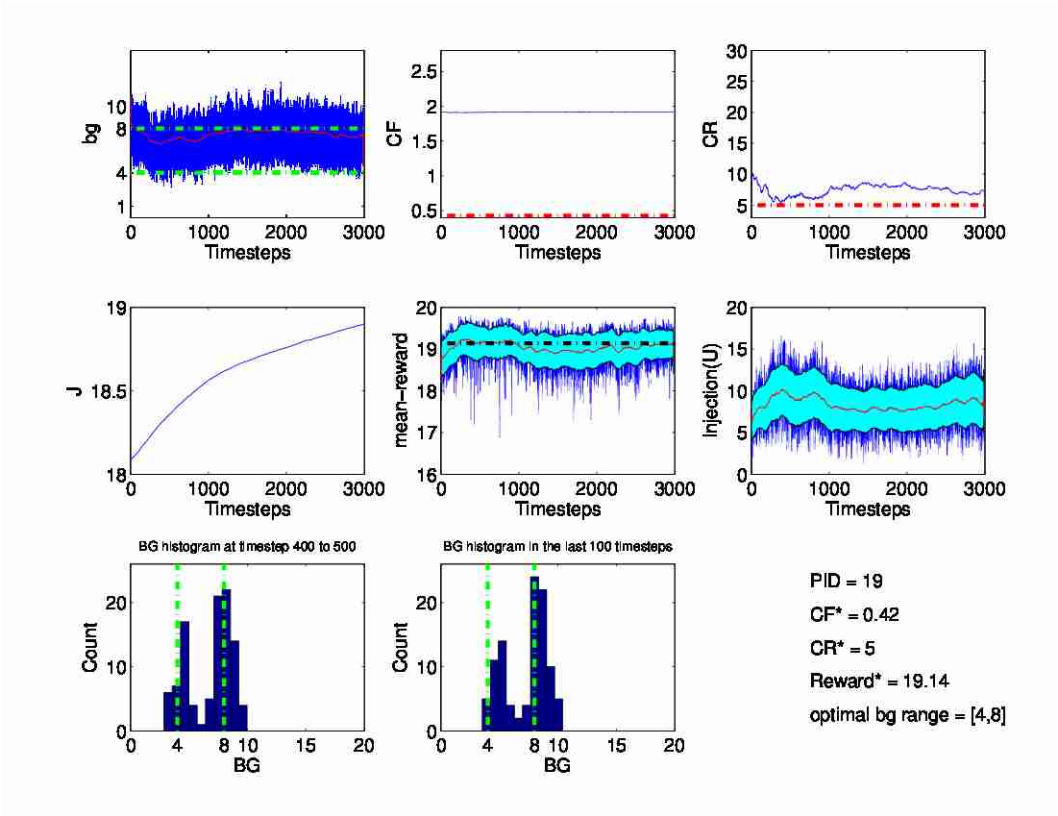


Figure B.49: The evolution of different variables during the learning phase of SAC (tile-coding version). See Section 5.2.3 for more information.

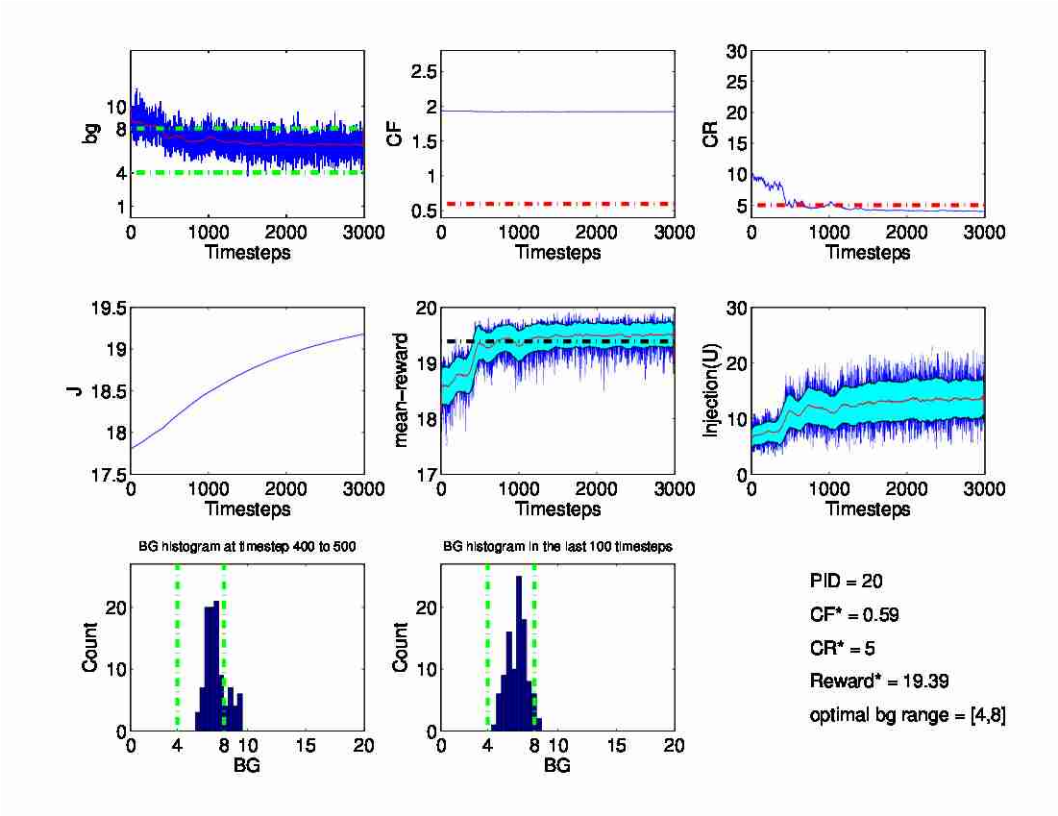


Figure B.50: The evolution of different variables during the learning phase of SAC (tile-coding version). See Section 5.2.3 for more information.

Appendix C

Performance contours

Here we provide the contour plot of the performance of the standard policy with respect to its parameters. See Section 1.3.7 for more details about the description of the plots. The surface is interpolated based on the 300 sampled points on the policy space. Note that the sampled parameters are selected from a restricted interval (the same interval that is used for generating training data for T1DC algorithm) ; see Section 5.1.1.

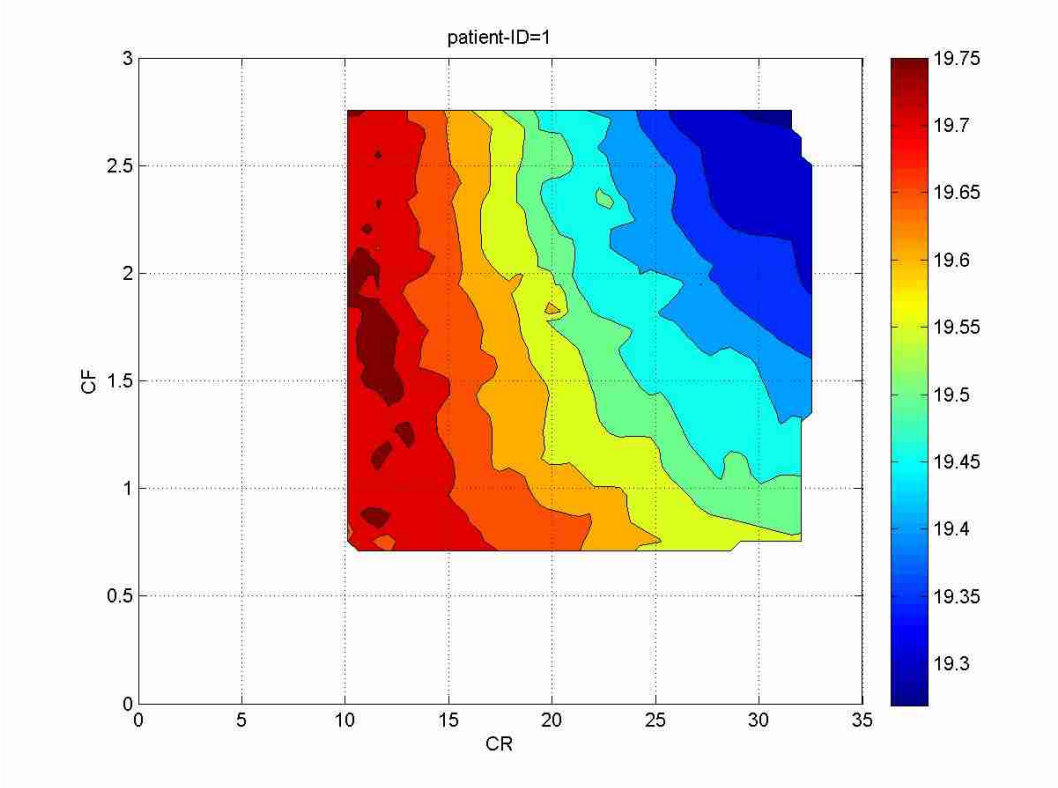


Figure C.1: Contour plot corresponding to the performance surface of the standard policy with respect to policy parameters CR and CF. The policy performance is calculated using mean-reward over 9-days of following the policy. See Section 1.3.7 for more details.

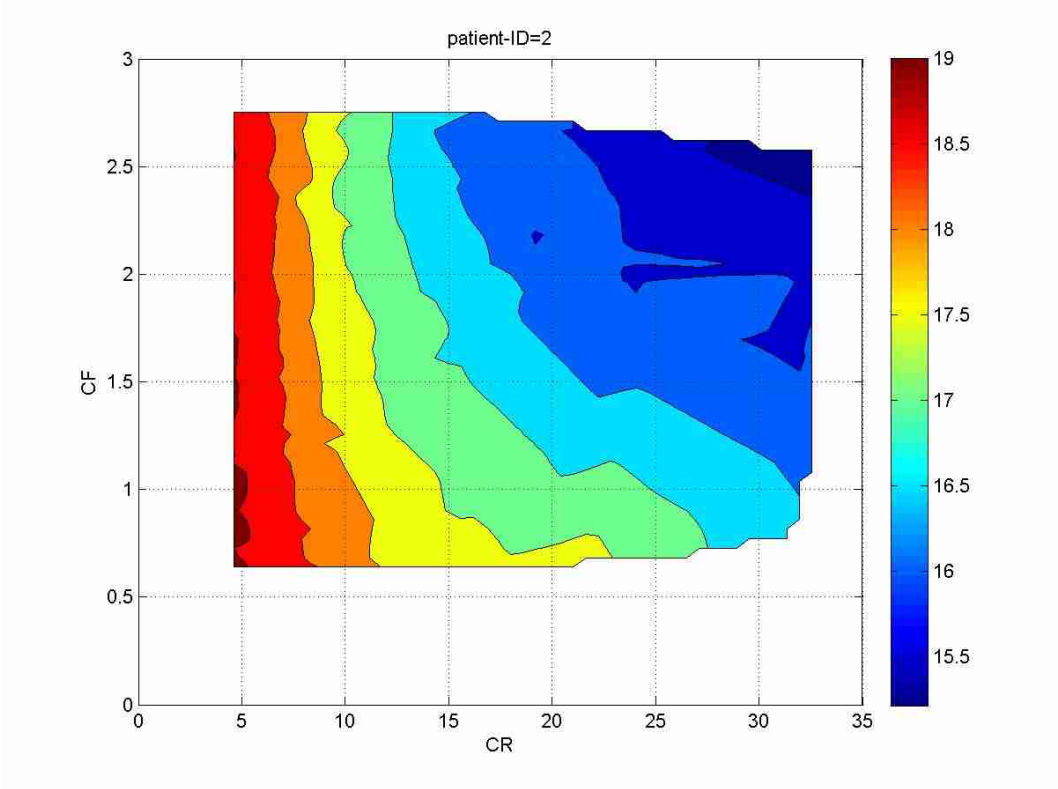


Figure C.2: Contour plot corresponding to the performance surface of the standard policy with respect to policy parameters CR and CF. The policy performance is calculated using mean-reward over 9-days of following the policy. See Section 1.3.7 for more details.

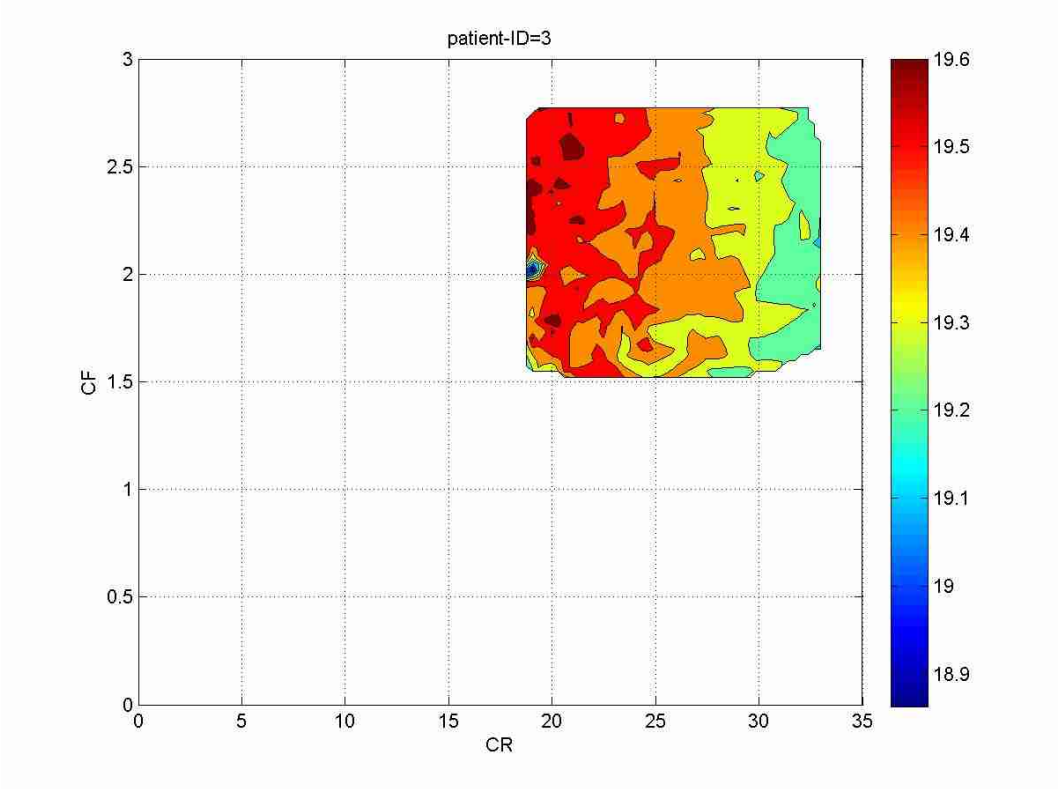


Figure C.3: Contour plot corresponding to the performance surface of the standard policy with respect to policy parameters CR and CF. The policy performance is calculated using mean-reward over 9-days of following the policy. See Section 1.3.7 for more details.

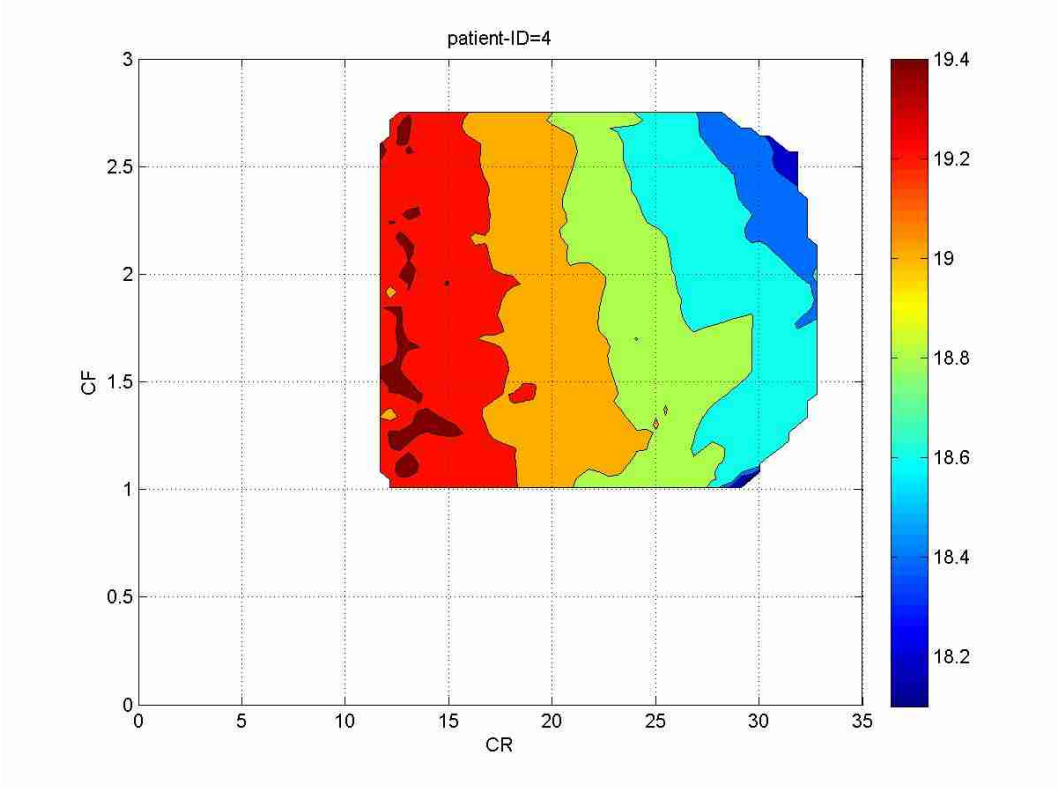


Figure C.4: Contour plot corresponding to the performance surface of the standard policy with respect to policy parameters CR and CF. The policy performance is calculated using mean-reward over 9-days of following the policy. See Section 1.3.7 for more details.

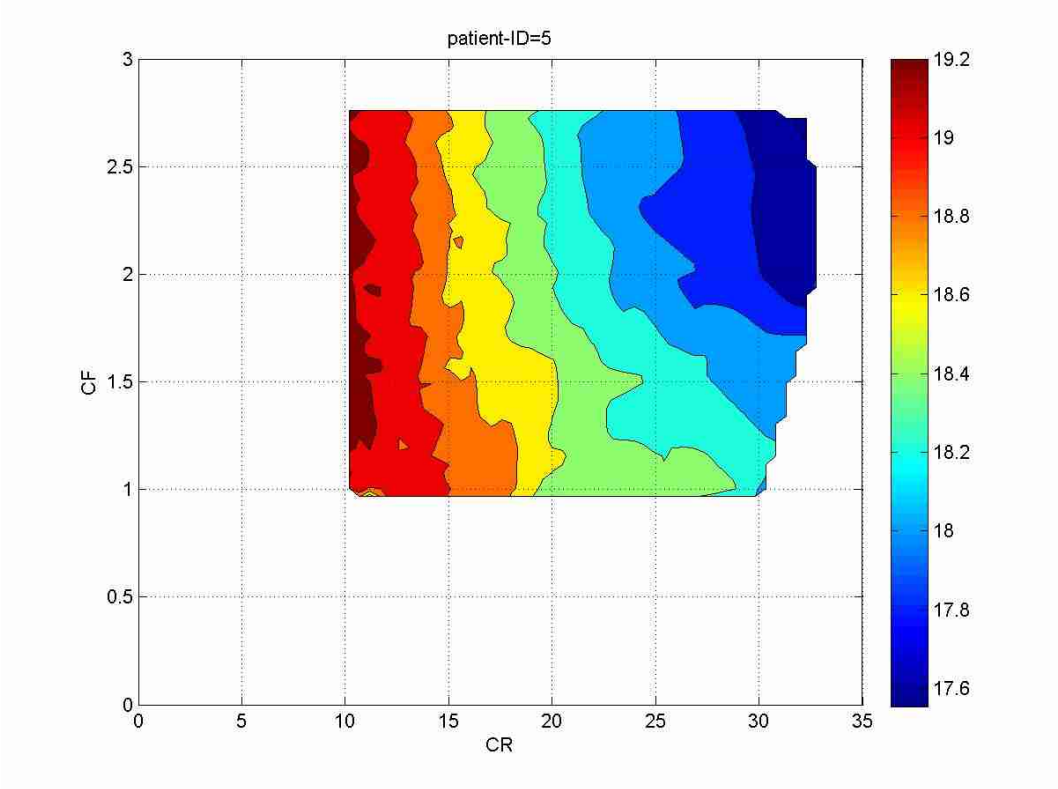


Figure C.5: Contour plot corresponding to the performance surface of the standard policy with respect to policy parameters CR and CF. The policy performance is calculated using mean-reward over 9-days of following the policy. See Section 1.3.7 for more details.

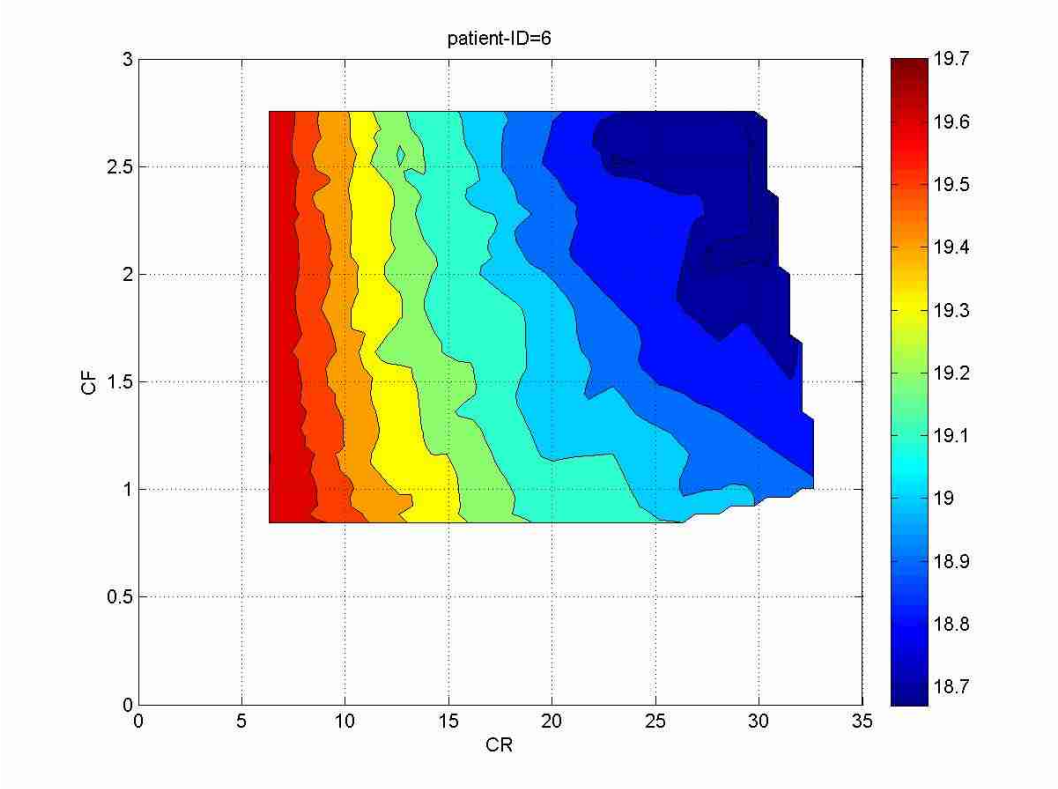


Figure C.6: Contour plot corresponding to the performance surface of the standard policy with respect to policy parameters CR and CF. The policy performance is calculated using mean-reward over 9-days of following the policy. See Section 1.3.7 for more details.

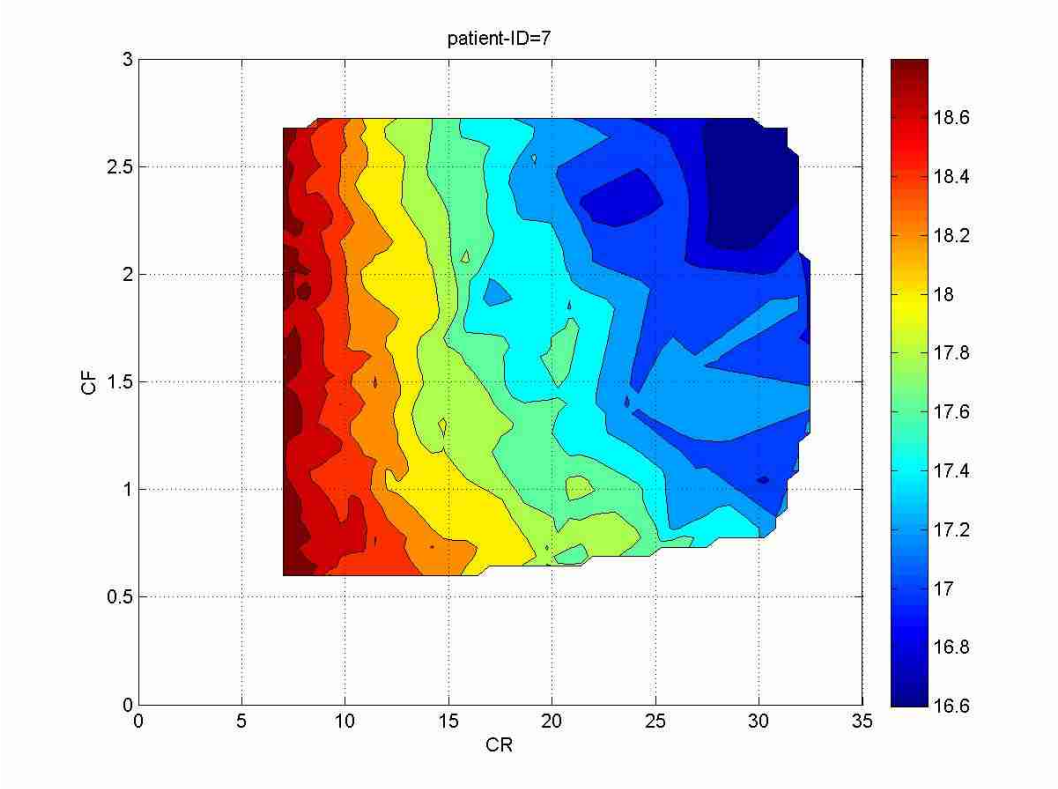


Figure C.7: Contour plot corresponding to the performance surface of the standard policy with respect to policy parameters CR and CF. The policy performance is calculated using mean-reward over 9-days of following the policy. See Section 1.3.7 for more details.

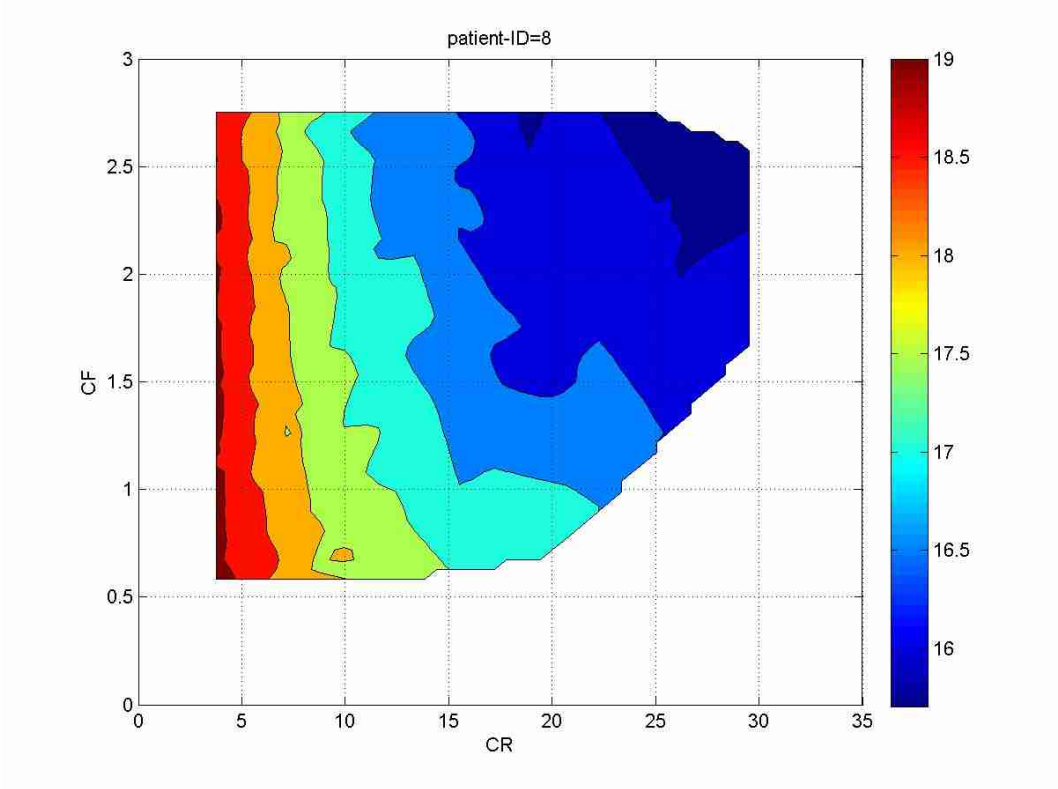


Figure C.8: Contour plot corresponding to the performance surface of the standard policy with respect to policy parameters CR and CF. The policy performance is calculated using mean-reward over 9-days of following the policy. See Section 1.3.7 for more details.

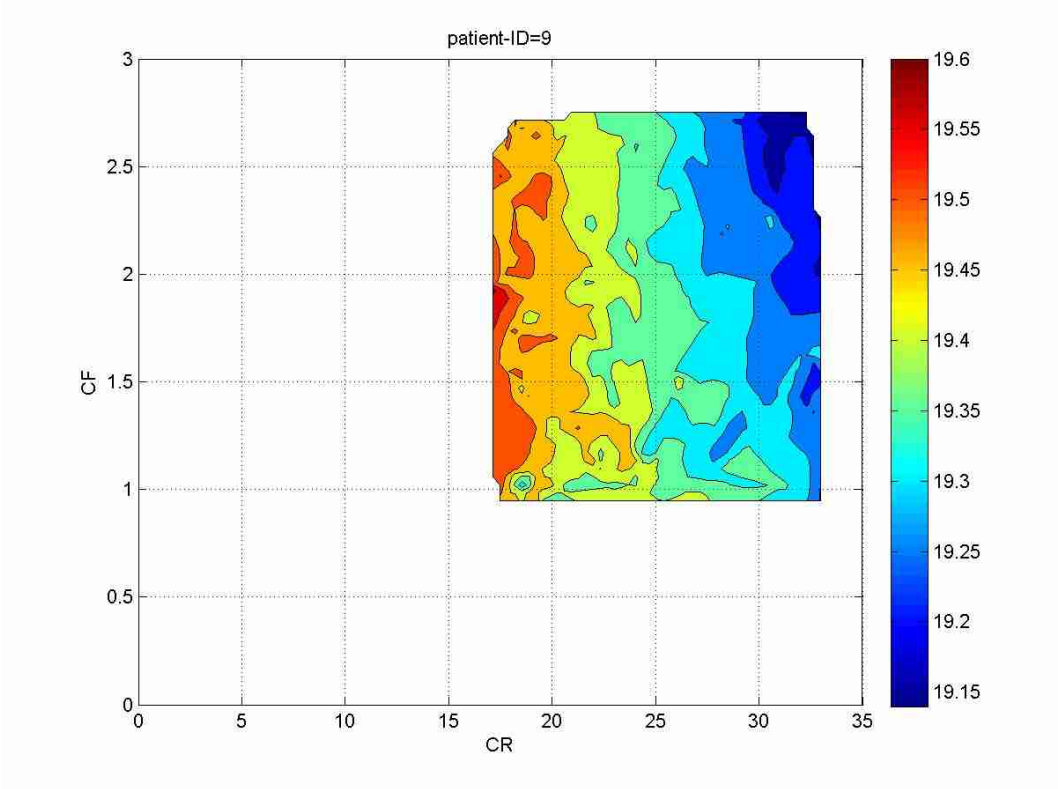


Figure C.9: Contour plot corresponding to the performance surface of the standard policy with respect to policy parameters CR and CF. The policy performance is calculated using mean-reward over 9-days of following the policy. See Section 1.3.7 for more details.

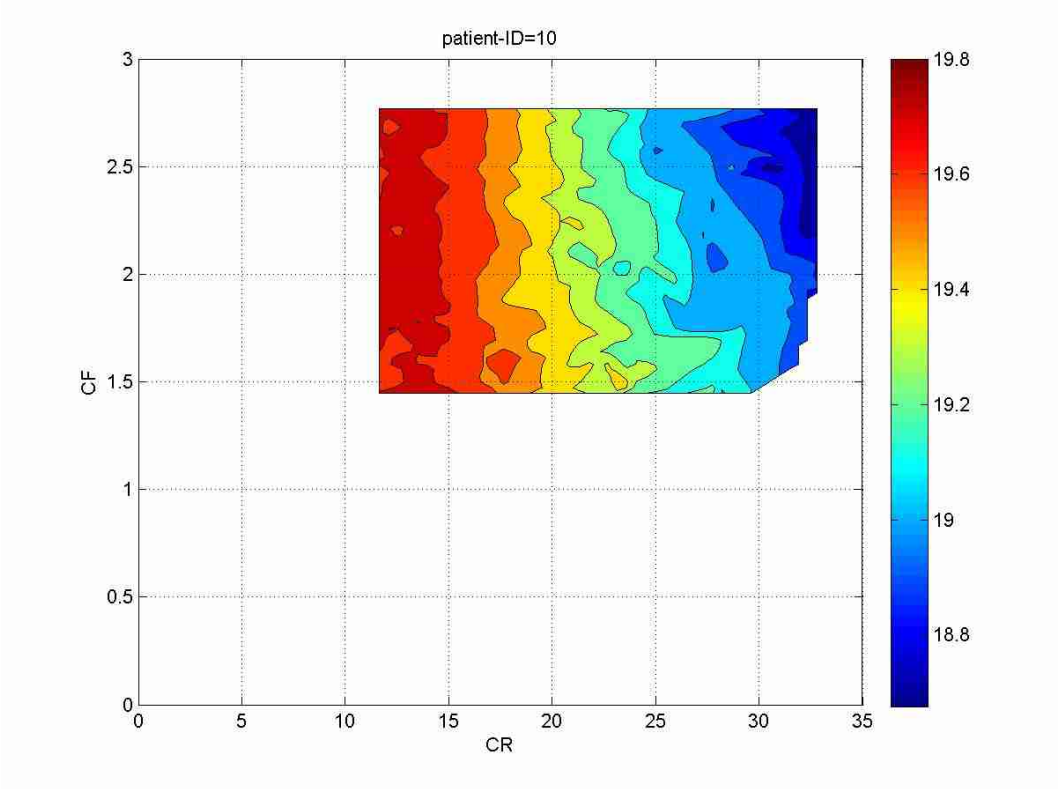


Figure C.10: Contour plot corresponding to the performance surface of the standard policy with respect to policy parameters CR and CF. The policy performance is calculated using mean-reward over 9-days of following the policy. See Section 1.3.7 for more details.

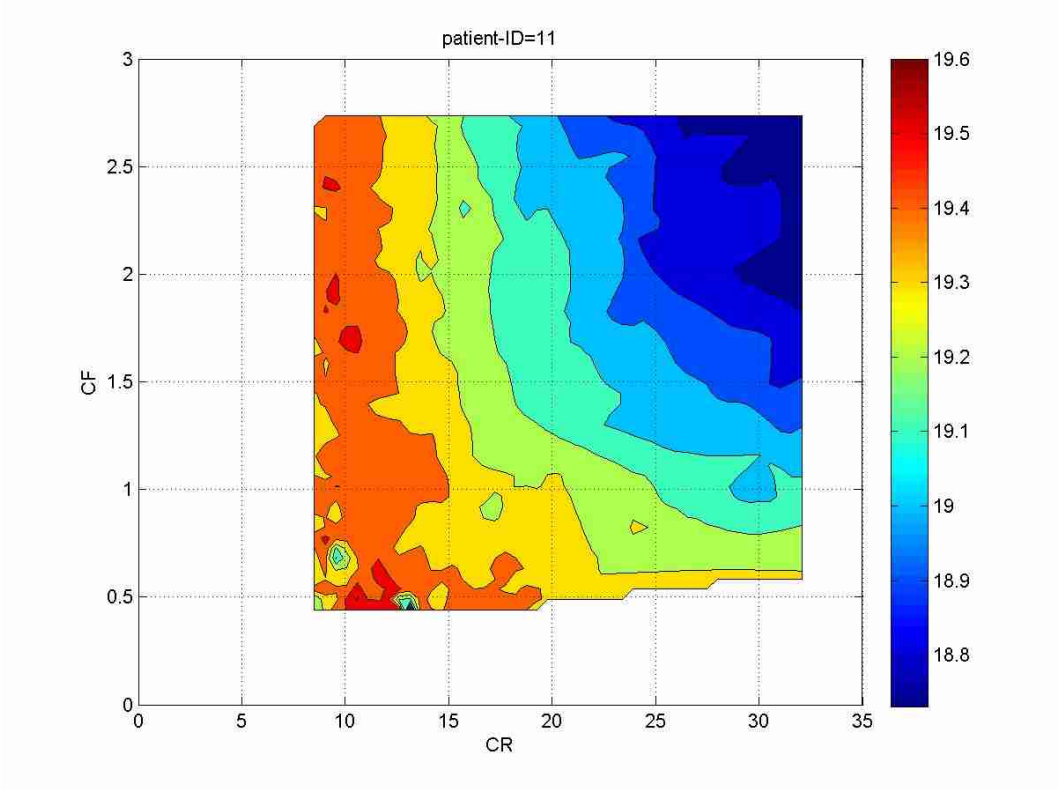


Figure C.11: Contour plot corresponding to the performance surface of the standard policy with respect to policy parameters CR and CF. The policy performance is calculated using mean-reward over 9-days of following the policy. See Section 1.3.7 for more details.

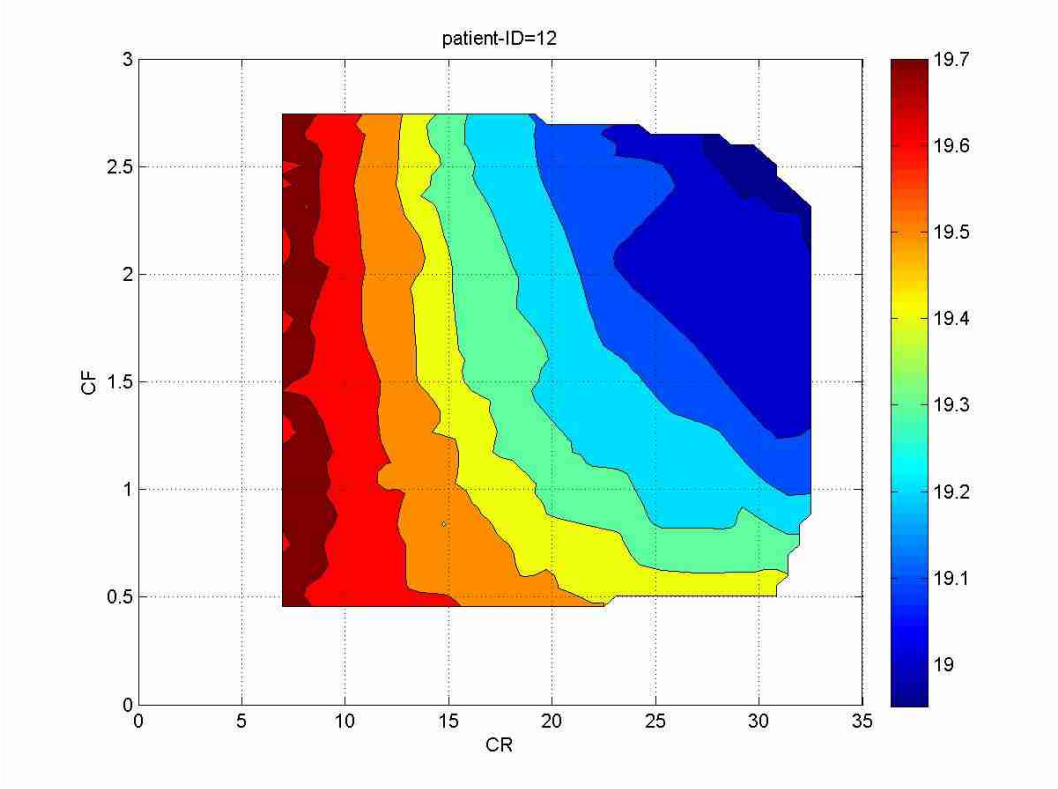


Figure C.12: Contour plot corresponding to the performance surface of the standard policy with respect to policy parameters CR and CF. The policy performance is calculated using mean-reward over 9-days of following the policy. See Section 1.3.7 for more details.

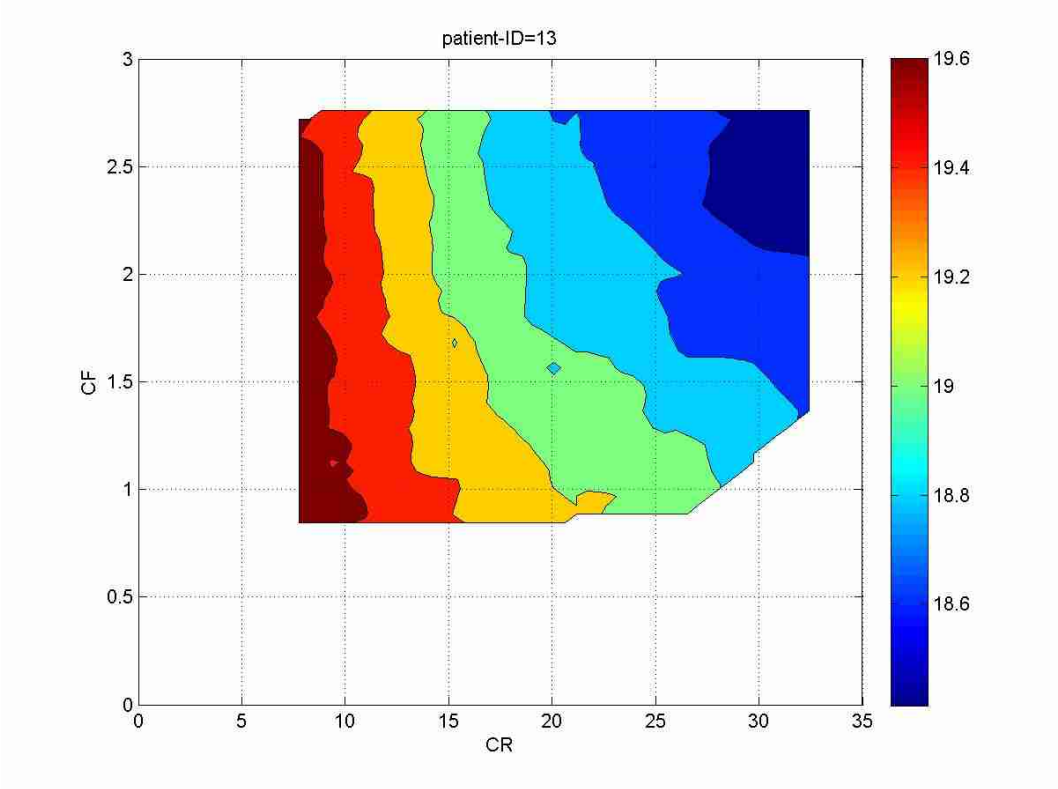


Figure C.13: Contour plot corresponding to the performance surface of the standard policy with respect to policy parameters CR and CF. The policy performance is calculated using mean-reward over 9-days of following the policy. See Section 1.3.7 for more details.

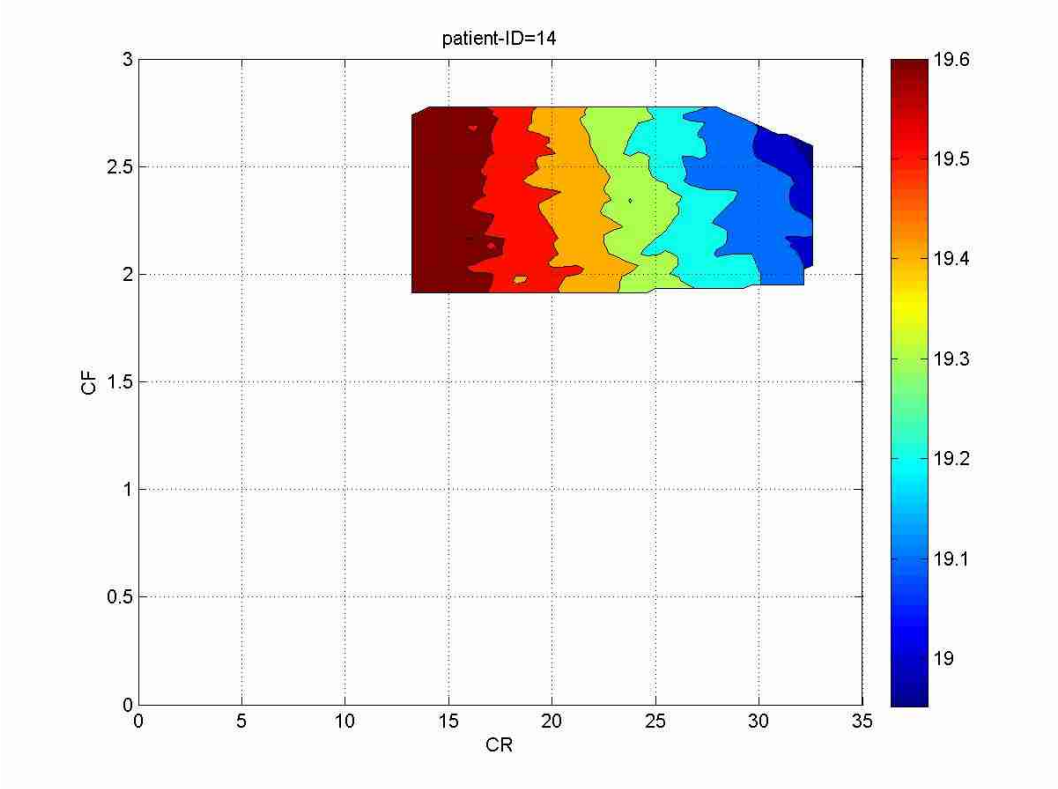


Figure C.14: Contour plot corresponding to the performance surface of the standard policy with respect to policy parameters CR and CF. The policy performance is calculated using mean-reward over 9-days of following the policy. See Section 1.3.7 for more details.

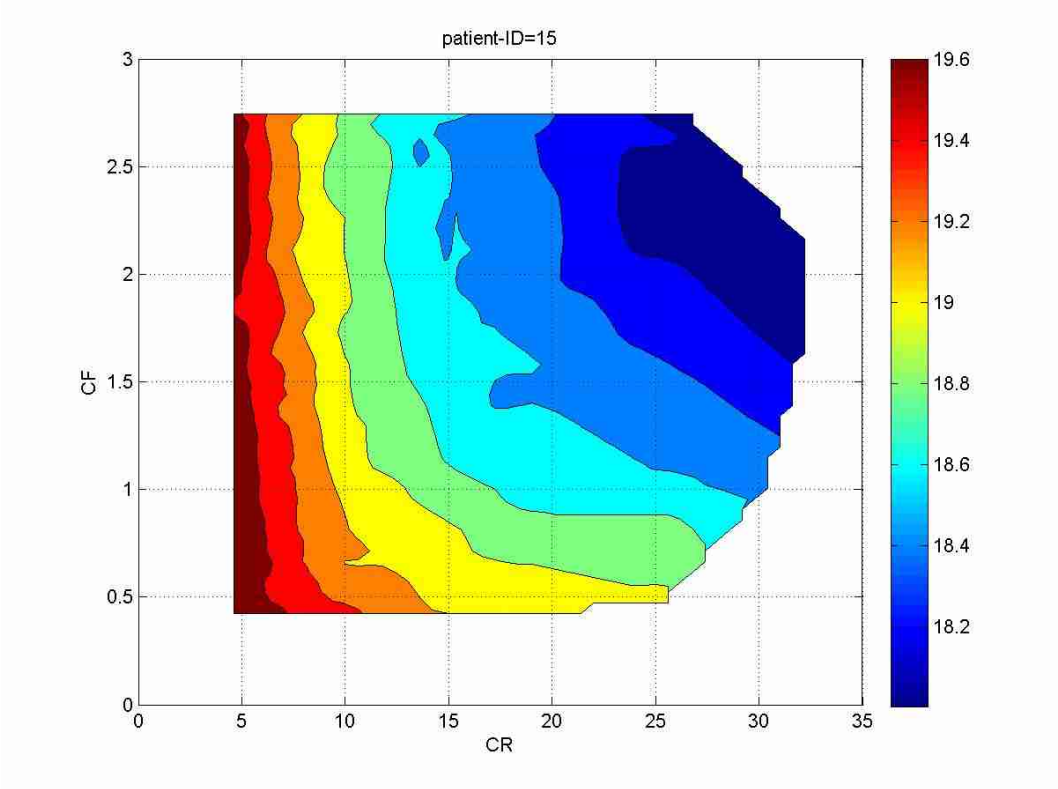


Figure C.15: Contour plot corresponding to the performance surface of the standard policy with respect to policy parameters CR and CF. The policy performance is calculated using mean-reward over 9-days of following the policy. See Section 1.3.7 for more details.

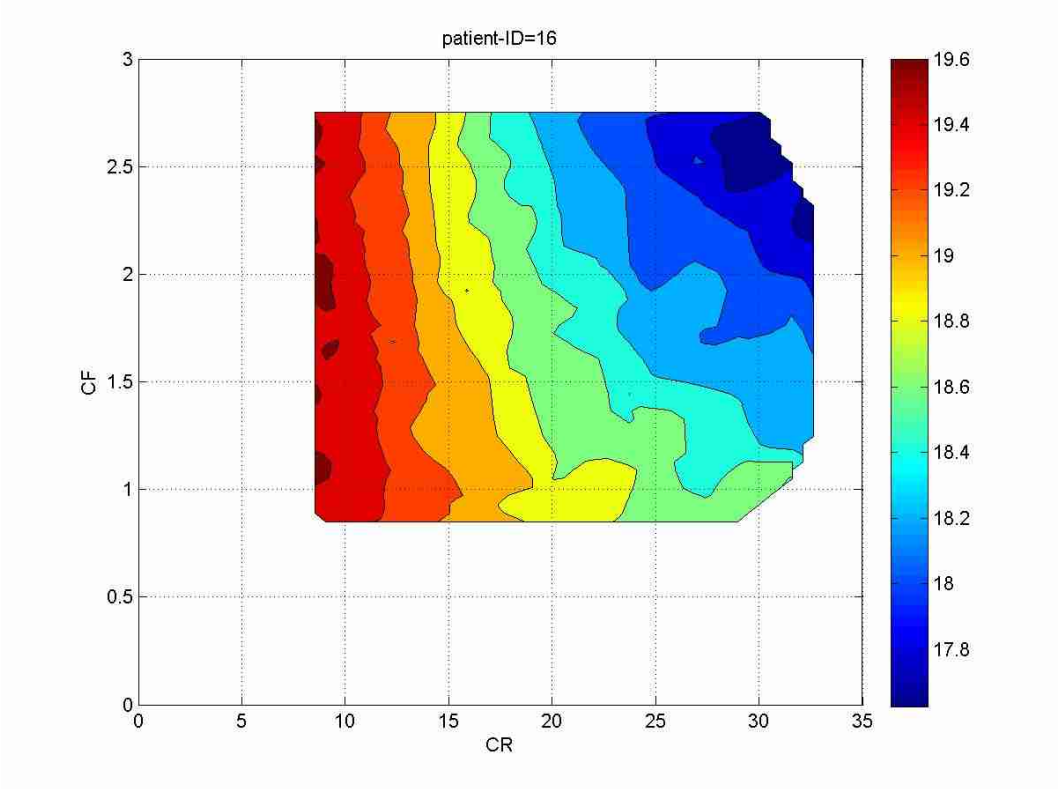


Figure C.16: Contour plot corresponding to the performance surface of the standard policy with respect to policy parameters CR and CF. The policy performance is calculated using mean-reward over 9-days of following the policy. See Section 1.3.7 for more details.

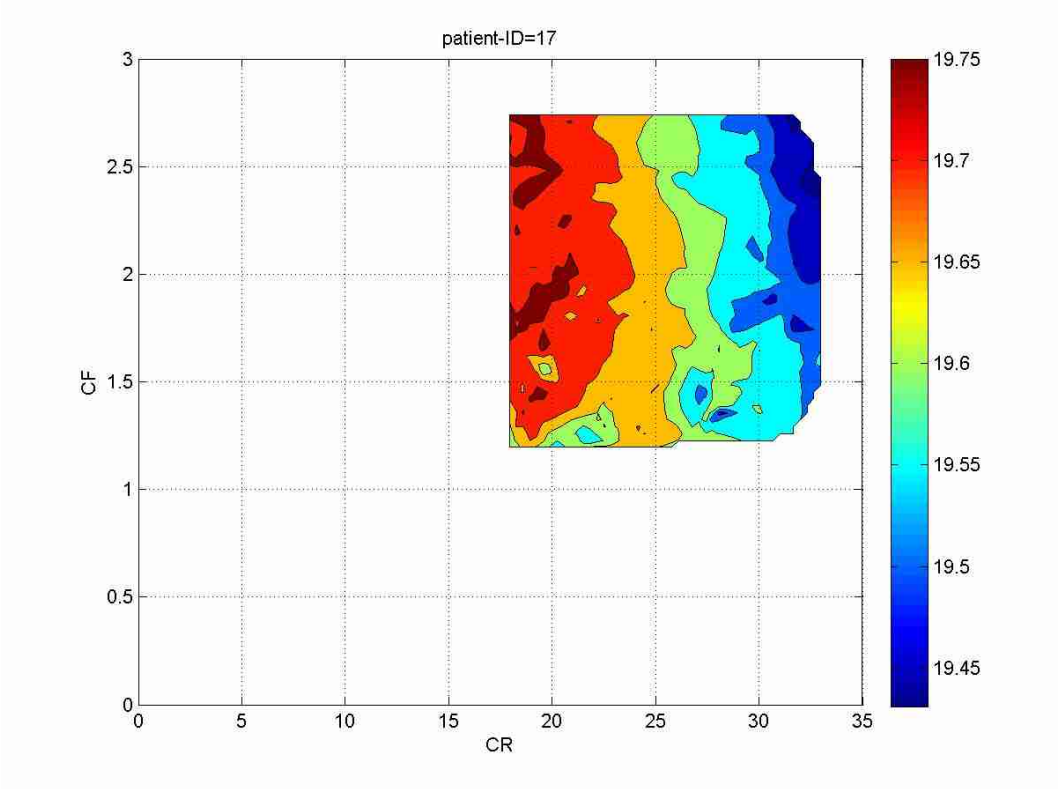


Figure C.17: Contour plot corresponding to the performance surface of the standard policy with respect to policy parameters CR and CF. The policy performance is calculated using mean-reward over 9-days of following the policy. See Section 1.3.7 for more details.

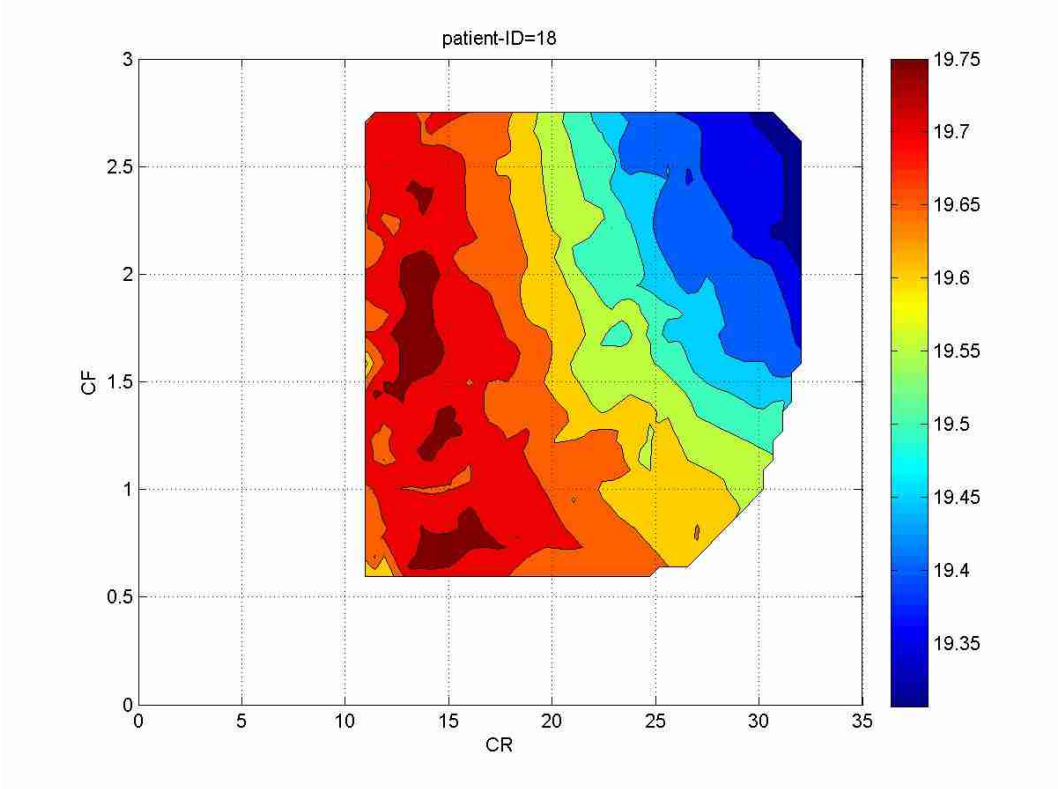


Figure C.18: Contour plot corresponding to the performance surface of the standard policy with respect to policy parameters CR and CF. The policy performance is calculated using mean-reward over 9-days of following the policy. See Section 1.3.7 for more details.

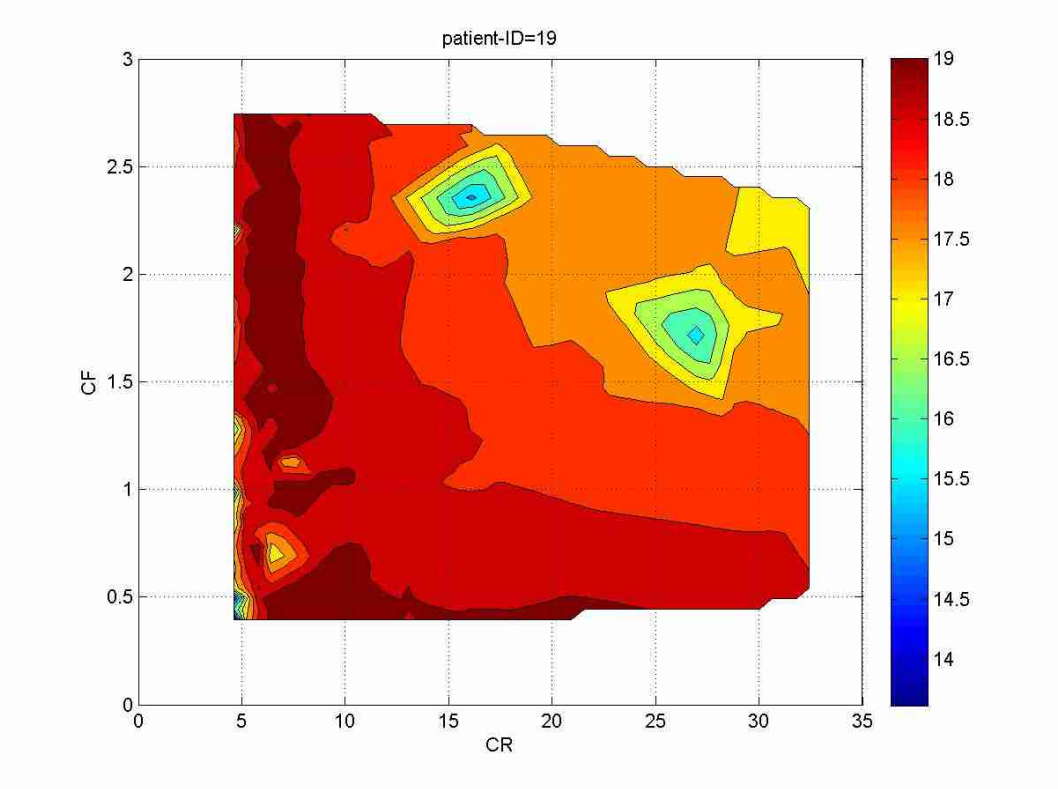


Figure C.19: Contour plot corresponding to the performance surface of the standard policy with respect to policy parameters CR and CF. The policy performance is calculated using mean-reward over 9-days of following the policy. See Section 1.3.7 for more details.

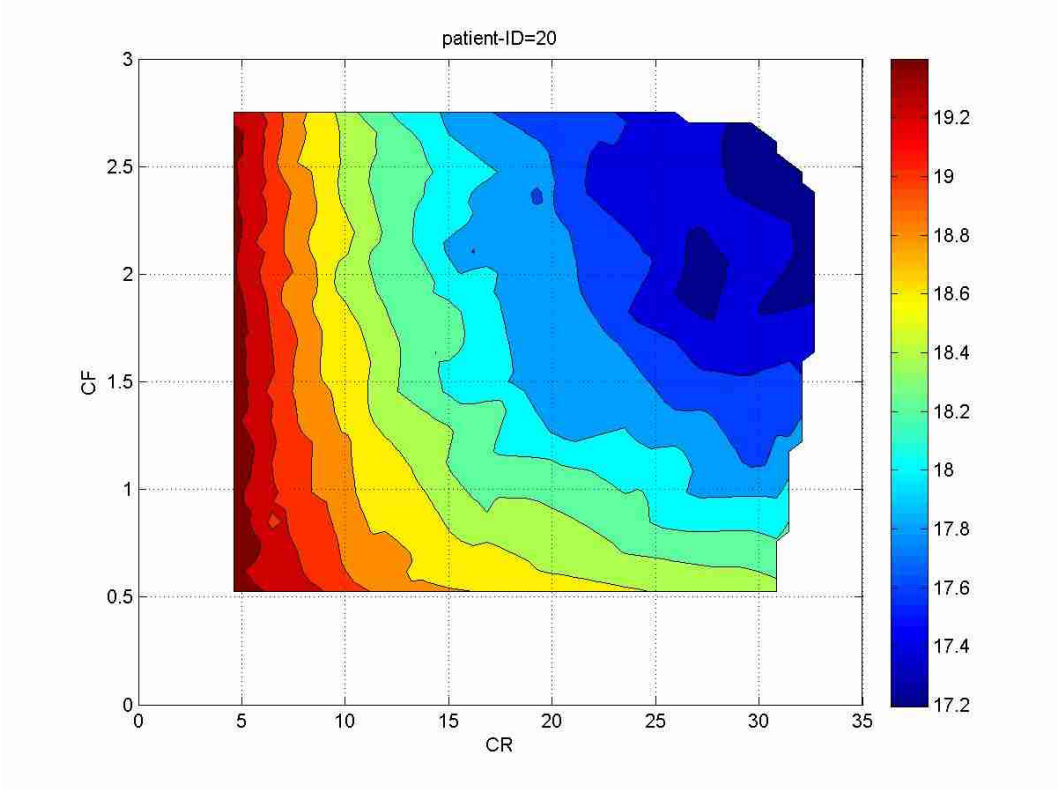


Figure C.20: Contour plot corresponding to the performance surface of the standard policy with respect to policy parameters CR and CF. The policy performance is calculated using mean-reward over 9-days of following the policy. See Section 1.3.7 for more details.