**UNIVERSITE DE LIEGE**
**Faculté des Sciences Appliquées**
**Département d’Electricité, Electronique et Informatique**
**Institut Montefiore**

# Near optimal closed-loop control Application to electric power systems

Dissertation présentée en vue de l’obtention
du grade de Docteur en Sciences Appliquées

par

**Damien ERNST**

Année Académique 2002-2003

# Preface

In 1996, still an undergraduate student, I started doing research in summer. At that time I had to study and improve a method developed in Liège, aiming to analyze transient stability phenomena that may occur on an electric power system [PERV00]. In December 1999, after more than one year spent as a PhD student, I went to Stockholm in order to start a research collaboration with the Royal Institute of Technology (KTH). The main motivation of the collaboration was to use the tools developed in Liège in the context of transient stability in order to explore how useful they could be to the design of the control function of a Flexible AC Transmission System (FACTS) [GAPE01]. About the same period, in Liège, a friend of mine was working on reinforcement learning methods. By friendship we decided to work together in order to apply the methods he knew well to a power system emergency defence scheme I was familiar with [EBZ$^+$98, DEW00]. I rapidly realized the tremendous interests that such methods could also offer to control the FACTS device I studied at KTH as well as to solve many other power system control problems. From there on, I decided to concentrate my research effort essentially on this topic. This dissertation is the result of this research effort.

# Acknowledgments

First of all, I would like to express my deepest gratitude and appreciation to my supervisor, Professor Louis Wehenkel, for his constant support and guidance throughout this thesis.
I would like to extend my warmest thanks to Professor Mania Pavella for introducing me several years ago in the fascinating world of research and for her support, valuable suggestions and perpetual enthusiasm for research.
Many thanks to the staff of the Electrical Engineering & Computer Science Department of the University of Liège for providing a stimulating and friendly atmosphere for study and research and help in many respects.
A special thanks to Dr. Thierry Van Cutsem for providing the Simulink models of the four-machine power system.
I am very grateful to the FNRS (Fonds National de la Recherche Scientifique) for providing the financial support for my PhD.
Finally, I would like to express my deepest gratitude and personal thanks to those closest to me. In particular, I would like to thank my parents for teaching me the value of education, their support and encouragement.

Damien Ernst
Liège
January 2003

# Summary

The work reported in this thesis is motivated by the practical problem of electric power systems control. Electric power systems are large-scale, complex non-linear systems which pose a number of challenging control problems, some of which are not yet solved in a fully satisfactory way. Interpreting the word “control” in the large sense, these problems encompass a multitude of issues as diverse as energy market regulation, system expansion, maintenance scheduling, power system operation, and automatic real-time control. Often these problems are coupled and most of them show up in different shapes and intensities in the various interconnected and isolated power systems that exist around the world.

The strategy we adopt to tackle such problems consists in explicitly formulating them as *optimal control problems* and in searching for an optimal *closed-loop control policy*. Unfortunately, optimal solutions of most power system control problems are impossible to determine exactly. Therefore, we use approximation techniques in order to find *near* optimal control policies. The solution of the original control problem is thus restated as the determination of an approximation architecture so as to approximate “at best” the optimal control policy of the original problem.

From a fundamental point of view our work is based on the combination of two intimately related frameworks. The first one concerns the theory of Markov Decision Processes (MDPs) and the Dynamic Programming (DP) paradigm. The second one, known as Reinforcement Learning (RL), studies the design of algorithms to determine optimal control policies from interaction with a system. To solve a practical control problem, our approach then consists in three successive steps : (i) define, in an ad hoc way, the states of a finite MDP that is used to determine the control policy; (ii) determine an optimal control policy for this MDP using information gathered (by observation, simulation, or calculation) from the original control problem; (iii) extrapolate this policy to the original problem.

Along these ideas, we have studied two categories of approaches, to handle respectively the case where an accurate and tractable analytical characterization of the original problem is available, and the more difficult - but very relevant - case where information can only be gathered from interaction with the original system (or with a black-box simulation model of the latter). In both cases, we have developed and compared various approximation architectures and various types of algorithms to determine their optimal parameters.

Among the numerous electric power system control problems mentioned above, we have focused on those related to automatic real-time control of Flexible Alternative Current Transmission System (FACTS) devices to damp electric power oscillations. Control strategies for such a FACTS device have been determined and assessed under various idealistic and realistic conditions, and in the context of different power system models ranging from two to more than sixty state variables. The thesis ends with a synthetic discussion of power system control problems and the advocacy of the developed framework.

# Abbreviations and nomenclature

*Abbreviations used (alphabetical order) :*

| | |
|---|---|
| AVR | Automatic Voltage Regulator |
| AGC | Automatic Generation Control |
| DP | Dynamic Programming |
| FACTS | Flexible Alternative Current Transmission System |
| FD | finite differences |
| Hz | Hertz |
| KF | Kalman Filter |
| kV | Kilovolt |
| LSE | Least Square Estimation |
| $LT$ | Learning Time (used in the graphics only) |
| MDP | Markov Decision Process |
| MDP$^{\delta}$ | Markov Decision Process that approximates the initial control problem |
| MLE | Maximum Likehood Estimation |
| MW | Megawatt |
| OMIB | One-Machine Infinite Bus |
| PSS | Power System Stabilizer |
| RL | Reinforcement Learning |
| SA | Stochastic Approximation |
| $SC$ | Abbreviation used for the word score (used in the graphics only) |
| SPS | System Protection Scheme (or Special Protection Scheme) |
| TCSC | Thyristor Controlled Series Capacitor |
| WLSE | Weighted Least Square Estimation |

*Nomenclature :*

| | |
|---|---|
| $x_{t+1} = f(x_t, u_t, w_t)$ | generic equation of a discrete-time stochastic system |
| $x_{t+1} = f(x_t, u_t)$ | generic equation of a discrete-time deterministic system |
| $\dot{x} = f(x, u, w)$ | generic equation of a continuous-time stochastic system |

| | |
|---|---|
| $\dot{x} = f(x,u)$ | generic equation of a continuous-time deterministic system |
| $x$ | state of the system |
| $u$ | control variable or action |
| $w$ | random disturbance |
| $x_t$ | state of the system at time $t$ |
| $u_t$ | action taken at time $t$ |
| $w_t$ | value of the random disturbance at time $t$ |
| $X$ | state space |
| $D$ | disturbance space |
| $C$ | control space |
| $U(x)$ | set of actions that can be taken in state $x$ |
| $U'(x)$ | $= U(x) \setminus \{u \in U(x) \|\ (x,u)$ has no yet been visited$\}$ |
| $X \times U$ | $\{(x,u) \| x \in X$ and $u \in U(x)\}$ |
| $\Pi$ | set of control policies |
| $\pi$ | control policy |
| $\mu$ | deterministic stationary policy |
| $\mu^p$ | randomized stationary policy |
| $V$ | value function or maximum expected return |
| $J^*$ | value function or maximum expected return ($J^* = V$) |
| $Q$ | $Q$-function of the system |
| $\tilde{Q}$ | approximate $Q$-function |
| $J^\pi$ | expected return associated to the policy $\pi$ |
| $J^{\mu^p}$ | expected return associated to the policy $\mu^p$ |
| $J^\mu$ | expected return associated to the policy $\mu$ |
| $p(x' \| x,u)$ | probability to reach state $x'$ after taking action $u$ in state $x$ |
| $r(x,u,w)$ | reward function |
| $r(x,u)$ | average reward obtained by taking action $u$ in state $x$ |
| $r_t$ | $r(x_t, u_t, w_t)$ |
| $R$ | $\sum_{k=0}^{\infty} \gamma^k r_k$ |
| $R_t$ | $\sum_{k=t}^{\infty} \gamma^k r_k$ |
| $B_r$ | bound on the rewards |
| $^\delta$ | suffix used to identify an element belonging to a MDP$^\delta$ |
| $x^\delta$ | state of a MDP$^\delta$ |
| $X^\delta$ | finite set of all $x^\delta$ |
| $C^\delta$ | subset of $C$ that intervenes in the MDP$^\delta$ definition |
| $U^\delta(x^\delta)$ | set of actions that can be taken in $x^\delta$ |
| $X^\delta \times U^\delta$ | $\{(x^\delta, u) \| x^\delta \in X^\delta$ and $u \in U^\delta(x^\delta)\}$ |
| $V^\delta$ | value function of the MDP$^\delta$ |
| $Q^\delta$ | $Q$-function of the MDP$^\delta$ |

| | |
|---|---|
| $W(x, x^\delta)$ | "probability" for state $x \in X$ to belong to $x^\delta \in X^\delta$ |
| $\alpha$ | degree of correction used in non-model based techniques |
| $\beta$ | memory factor used in model based techniques |
| $\delta$ | temporal-difference factor |
| $\epsilon$ | used to define the degree of randomness of the $\epsilon$-Greedy policy |
| $\lambda$ | trace decay factor |
| $\alpha_r$ | degree of correction used in the estimate of the reward |
| $\alpha_p$ | degree of correction used in the estimate of the transition probabilities |
| $N(u)$ | number of times $u$ has been taken |
| $N(x, u)$ | number of times $u$ has been taken while being in $x$ |
| $N(x', x, u)$ | number of times $x'$ has been reached while taking action $u$ in $x$ |
| $N^\delta(x^\delta, u)$ | number of times action $u$ has been taken while being in $x^\delta$ |
| $N^\delta(x^{\delta'}, x^\delta, u)$ | number of times $x^{\delta'}$ has been reached while taking action $u$ in $x^\delta$ |
| $(x, u)$ | state-action pair composed by the state $x$ and the action $u$ |
| $e(x, u)$ | eligibility trace for the state-action pair $(x, u)$ |
| $e^\delta(x^\delta, u)$ | eligibility trace for the state-action pair $(x^\delta, u)$ |
| $P_{uniform}(x)$ | symbolizes a uniform probability distribution of $x$ on $X$ |
| $P_0(x)$ | probability that $x$ is the initial state of the system |
| $P_{learning}(x, u)$ | probability distribution of the $(x, u)$ visited during the learning |
| $S$ | pseudo state space of the system |
| $s$ | pseudo-state of the system |
| $\mathcal{H}_t$ | $(x_0, \cdots, x_{t-1}, u_0, \cdots, u_{t-1}, w_0, \cdots, w_{t-1})$ |
| $I_B$ | indicator function of a set $B$. $I_B(x) = 1$ if $x \in B$ and 0 otherwise |
| $\delta(x, x')$ | $\delta(x, x') = 1$ if $x = x'$ and 0 otherwise |
| $\underset{a}{E}[f]$ | the expectation of $f$ with respect to the distribution of the random variable $a$ |

# Contents

**1 Introduction** **23**

1.1 Power system control and reinforcement learning . . . . . . . . . 23

1.2 Framework used to present RL methods . . . . . . . . . . . . . . 24

1.3 Outline of the thesis . . . . . . . . . . . . . . . . . . . . . . . . . 25

1.3.1 Structure of the main body . . . . . . . . . . . . . . . . . 25

1.3.2 About the appendices . . . . . . . . . . . . . . . . . . . . 27

1.4 Main contributions of the thesis . . . . . . . . . . . . . . . . . . . 27

1.4.1 Reinforcement learning and dynamic programming . . . . 27

1.4.2 Power system control . . . . . . . . . . . . . . . . . . . . 29

**2 Dynamic programming** **31**

2.1 Dynamic programming : the basics . . . . . . . . . . . . . . . . 32

2.1.1 Finite spaces . . . . . . . . . . . . . . . . . . . . . . . . . 32

2.1.2 Infinite state spaces . . . . . . . . . . . . . . . . . . . . . 39

2.1.3 Remarks . . . . . . . . . . . . . . . . . . . . . . . . . . . 39

2.2 Computation of $V$, $J^{\mu}$, $J^{\mu^p}$ . . . . . . . . . . . . . . . . . . . . . 40

2.3 Finite state and control spaces : computational methods . . . . . 44

2.3.1 Markov decision processes . . . . . . . . . . . . . . . . . 44

2.3.2 Definition of the MDP structure from the system dynamics and reward function . . . . . . . . . . . . . . . . . . . . 45

2.3.3 MDP structure computation . . . . . . . . . . . . . . . . 46

2.3.4 Rewriting of the mappings using a MDP structure . . . . . 47

2.3.5 Computation of $J^{\mu}$ . . . . . . . . . . . . . . . . . . . . . 47

2.3.6 Value iteration algorithm . . . . . . . . . . . . . . . . . . 48

2.3.7 Gauss-Seidel version of the value iteration algorithm . . . 53

2.3.8 Policy iteration algorithm . . . . . . . . . . . . . . . . . . 54

2.3.9 The modified policy iteration algorithms . . . . . . . . . . 55

2.3.10 Randomized stationary policy evaluation . . . . . . . . . 56

2.4 Infinite state spaces . . . . . . . . . . 57
2.5 Summary . . . . . . . . . . 58

**3 Approximate solutions for DP problems 59**
3.1 Introduction . . . . . . . . . . 60
3.2 Aggregation technique . . . . . . . . . . 61
3.2.1 Approximate MDP . . . . . . . . . . 61
3.2.2 Extension of the MDP solution to the original problem . . 62
3.2.3 Equivalence between $Q$ and $\tilde{Q}$ . . . . . . . . . . 64
3.2.4 Equivalence conditions used to define the MDP$^\delta$ . . . . . 65
3.2.5 Difficulties to compute the MDP$^\delta$ structure . . . . . . . . 66
3.2.6 Continuous state spaces . . . . . . . . . . 67
3.3 Representative states technique . . . . . . . . . . 69
3.3.1 Approximate MDP . . . . . . . . . . 69
3.3.2 Extension of the MDP solution to the original problem . . 71
3.3.3 Equivalence between $Q$ and $\tilde{Q}$ . . . . . . . . . . 71
3.3.4 Equivalence conditions used to define the MDP$^\delta$ . . . . . 73
3.3.5 Difficulties to compute the MDP$^\delta$ structure . . . . . . . . 74
3.3.6 Triangulation technique . . . . . . . . . . 75
3.3.7 Other methods to determine $W$ . . . . . . . . . . 80
3.4 Quality of the approximate policy computed . . . . . . . . . . 80
3.5 Bellman error method . . . . . . . . . . 85
3.6 Summary . . . . . . . . . . 86

**4 A simple FACTS control problem 87**
4.1 The OMIB power system . . . . . . . . . . 88
4.1.1 The system dynamics . . . . . . . . . . 88
4.1.2 State space trimming . . . . . . . . . . 89
4.2 Optimal control problem formulation . . . . . . . . . . 91
4.2.1 Discrete-time system dynamics . . . . . . . . . . 91
4.2.2 Reward definition . . . . . . . . . . 91
4.3 Optimal control problem solution . . . . . . . . . . 92
4.3.1 Aggregation technique . . . . . . . . . . 92
4.3.2 Representative states technique . . . . . . . . . . 98
4.4 State and control space discretization variants . . . . . . . . . . 100
4.4.1 About the different scores . . . . . . . . . . 103
4.4.2 About the technique used . . . . . . . . . . 103
4.4.3 About the grid used . . . . . . . . . . 103
4.4.4 About the control set used . . . . . . . . . . 104

4.5 On the combined use of multiple MDP$^{\delta}$ . . . . . 104
4.6 Robustness of the control law . . . . . 106
4.6.1 Robustness to a parameter change . . . . . 106
4.6.2 Robustness to noise . . . . . 107
4.6.3 Robustness to modeling . . . . . 107
4.7 Summary . . . . . 110

**5 Reinforcement learning** **111**
5.1 The generic reinforcement learning method . . . . . 112
5.2 Action selection . . . . . 113
5.3 Learning : model based techniques . . . . . 115
5.3.1 Introduction . . . . . 115
5.3.2 Kalman Filter like algorithm . . . . . 115
5.3.3 Stochastic Approximation algorithm . . . . . 117
5.3.4 Resolution of the MDP . . . . . 118
5.4 Learning : non-model based techniques . . . . . 124
5.4.1 Introduction . . . . . 124
5.4.2 $Q$-learning . . . . . 125
5.4.3 $Q(\lambda)$ . . . . . 126
5.5 Learning modules : a unified view . . . . . 131
5.6 Summary . . . . . 132

**6 Reinforcement learning in infinite state spaces** **133**
6.1 Overview . . . . . 134
6.1.1 Model based techniques : general scheme . . . . . 134
6.1.2 Non-model based techniques : general scheme . . . . . 135
6.1.3 Action selection . . . . . 138
6.2 About convergence properties . . . . . 138
6.3 Model based aggregation technique . . . . . 143
6.3.1 Kalman Filter like algorithm . . . . . 143
6.3.2 Stochastic Approximation algorithm . . . . . 145
6.4 Non-model based aggregation technique . . . . . 147
6.4.1 $Q$-learning algorithm . . . . . 148
6.4.2 $Q(\lambda)$ algorithm . . . . . 149
6.5 Aggregation technique : a unified view . . . . . 150
6.6 Model based representative states technique . . . . . 153
6.6.1 Kalman Filter like algorithm . . . . . 153
6.6.2 Stochastic Approximation algorithm . . . . . 156
6.7 Non-model based representative states technique . . . . . 160

6.7.1 $Q$-learning algorithm . . . . . . . . . . . . . . . . . . . . 161
6.7.2 $Q(\lambda)$ algorithm . . . . . . . . . . . . . . . . . . . . . . . 163
6.8 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . 166

**7 Revisit of a simple FACTS control problem 169**
7.1 Reinforcement learning and the aggregation technique . . . . . . 170
7.1.1 Model based techniques . . . . . . . . . . . . . . . . . . . 170
7.1.2 Model based vs non-model based RL algorithm . . . . . . 176
7.2 Representative states technique . . . . . . . . . . . . . . . . . . . 177
7.2.1 Procedure . . . . . . . . . . . . . . . . . . . . . . . . . . 177
7.2.2 Resulting score curves . . . . . . . . . . . . . . . . . . . 178
7.2.3 Comparison with the aggregation technique . . . . . . . . 180
7.3 On the combined use of multiple MDP$^\delta$ . . . . . . . . . . . . . . 180
7.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . 181

**8 Dynamic programming in continuous-time 185**
8.1 Hamilton-Jacobi-Bellman equation . . . . . . . . . . . . . . . . . 186
8.2 Convergent numerical approximation scheme . . . . . . . . . . . 190
8.3 Learning of the control problem solution . . . . . . . . . . . . . 194
8.4 The simple FACTS control problem . . . . . . . . . . . . . . . . 195
8.4.1 Rewards, decay factor and reinforcement functional . . . . 195
8.4.2 The HJB equation for the simple FACTS control problem . 196
8.4.3 Discretization of the HJB equation for the OMIB case . . 197
8.5 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . 198

**9 Real world conditions 199**
9.1 Problems met when controlling a real world system . . . . . . . . 199
9.1.1 Policy used when starting to interact with a real system . . 199
9.1.2 Non observable rewards . . . . . . . . . . . . . . . . . . 201
9.1.3 Partially observable system state . . . . . . . . . . . . . 201
9.1.4 Curse of dimensionality . . . . . . . . . . . . . . . . . . 202
9.1.5 Non-stationarity of the system . . . . . . . . . . . . . . . 203
9.2 A real world power system control problem . . . . . . . . . . . . 204
9.2.1 The four-machine power system . . . . . . . . . . . . . . 204
9.2.2 Control problem definition . . . . . . . . . . . . . . . . . 205
9.2.3 Reinforcement learning algorithm used in real conditions 207
9.2.4 Case I : control of the stable equilibrium point . . . . . . . 211
9.2.5 Case II : damping of self-sustained power oscillations . . 212
9.2.6 Case III : adaptation to time-varying conditions . . . . . . 216

9.2.7 Case IV : damping of oscillations caused by short-circuits 217
9.3 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . 224

**10 Power system control and RL 227**
10.1 Power system control : a high priority . . . . . . . . . . . . . . . 227
10.2 Power system control : a generic description . . . . . . . . . . . . 228
10.3 Design of control schemes : a systematic approach . . . . . . . . 231
10.4 Reinforcement learning as a tool to design agents . . . . . . . . . 233
10.4.1 Two modes of application . . . . . . . . . . . . . . . . . . 233
10.4.2 Usefulness of RL methods to power system control . . . . 235
10.4.3 Multi-agent systems . . . . . . . . . . . . . . . . . . . . . 236
10.5 Some problems already tackled by RL methods . . . . . . . . . . 236
10.5.1 Continuously acting controllers . . . . . . . . . . . . . . . 237
10.5.2 System Protection Schemes . . . . . . . . . . . . . . . . . 237
10.6 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . 239

**11 Conclusions and future prospects 241**
11.1 Closure . . . . . . . . . . . . . . . . . . . . . . . . . . . . 241
11.2 Problems, solutions and future research prospects . . . . . . . . . 242

**A Linear approximation functions 247**
A.1 Weighted Least Square Estimation (WLSE) . . . . . . . . . . . . 247
A.1.1 Problem statement . . . . . . . . . . . . . . . . . . . . . . 247
A.1.2 Resolution of the WLSE problem : pseudo-inverse . . . . 248
A.1.3 Kalman filter like algorithm . . . . . . . . . . . . . . . . 248
A.1.4 Infinite set of values $(\mathbf{A} \to \infty)$ . . . . . . . . . . . . . . 249
A.2 Minimization of an integral of a continuum of cost functions . . . 250
A.2.1 Problem statement . . . . . . . . . . . . . . . . . . . . . . 250
A.2.2 Iterative algorithm . . . . . . . . . . . . . . . . . . . . . . 251
A.2.3 LSE and stochastic approximation algorithm . . . . . . . 252
A.3 Proofs and additional explanations . . . . . . . . . . . . . . . . 252
A.3.1 Estimation of the transition probabilities of the MDP$^\delta$ . . 252
A.3.2 MDP structure estimation by solving WLSE problems . . 253
A.3.3 MDP structure estimation by minimizing an integral of a continuum of cost functions . . . . . . . . . . . . . . . . 256

**B Triangulation of an $n$-cube 259**
B.1 Definitions . . . . . . . . . . . . . . . . . . . . . . . . . . . 259
B.2 Triangulation of the $n$-cube into $n!$ simplices . . . . . . . . . . . 260

B.3 Triangulation of an $n$-cube into $s(n)$ simplices . . . . . . . . . . . 260
B.4 Triangulation quality and control law quality . . . . . . . . . . . . 262

**C Algorithms built under contraction assumptions 263**
C.1 Contraction mapping . . . . . . . . . . . . . . . . . . . . . . . . 263
C.2 Description of algorithmic models . . . . . . . . . . . . . . . . . . 264
C.2.1 All elements of $R$ are refreshed . . . . . . . . . . . . . . . 264
C.2.2 One element of $R$ is refreshed . . . . . . . . . . . . . . . . 264
C.2.3 One element of $R$ is refreshed and noise introduction . . 265
C.3 $T$ is a contraction mapping : consequences . . . . . . . . . . . . . 266
C.4 Introduction of the $H$ mapping and consequences . . . . . . . . . 267
C.5 Q-learning convergence proof . . . . . . . . . . . . . . . . . . . . 268
C.6 Aggregation technique : convergence to the MDP$^\delta$ solution . . . 269
C.6.1 Convergence conditions . . . . . . . . . . . . . . . . . . . . 269
C.6.2 Proof . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 270
C.7 Aggregation technique : convergence . . . . . . . . . . . . . . . . 272
C.7.1 Convergence conditions . . . . . . . . . . . . . . . . . . . . 272
C.7.2 Proof . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 273
C.8 Representative states technique : convergence . . . . . . . . . . . 273
C.8.1 Convergence conditions . . . . . . . . . . . . . . . . . . . . 273
C.8.2 Proof . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 274

# List of Figures

2.1 Representation of $T^k J(x)$ . . . . . . . . . . . . . . . . . . . . 43
2.2 The value iteration algorithm . . . . . . . . . . . . . . . . . . . . . . 49
2.3 The value iteration algorithm : $Q$-function computation . . . . . . 51
2.4 The Gauss-Seidel version of the value iteration algorithm : $Q$-function computation . . . . . . . . . . . . . . . . . . . . . . . . . 53
2.5 The Policy Iteration algorithm . . . . . . . . . . . . . . . . . . . . 54
2.6 Randomized stationary policy evaluation . . . . . . . . . . . . . . . 56

3.1 Approximate solutions for dynamic programming . . . . . . . . . 60
3.2 Aggregation technique : numerical estimation of the MDP$^\delta$ structure 67
3.3 Discretization of the state space (three-dimensional representation) 68
3.4 Approximation of the $Q$-function for different state space discretizations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
3.5 State space represented by a finite number of states . . . . . . . . 70
3.6 Representative states technique : numerical estimation of the MDP$^\delta$ structure . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 74
3.7 Triangulation of the state space. Two-dimensional representation. 75
3.8 Triangulation technique used to define the function $W$. The states selected from $X$ to represent $X^\delta$ are 1, $1.875$, 3 and $x^t$. . . . . . . . 77
3.9 Numerical estimation of the approximate $Q$-function. $\#X^\delta = 22$. 78
3.10 Approximate $Q$-function computed for different sizes of $X^\delta$ with $N_{w_i} = 1000$ . . . . . . . . . . . . . . . . . . . . . . . . . . . 79
3.11 Boundary and triangulation . . . . . . . . . . . . . . . . . . . . . 79
3.12 Numerical estimation of the score of a policy $\mu$ . . . . . . . . . . 84

4.1 The one-machine infinite bus (OMIB) system . . . . . . . . . . . 88
4.2 Uncontrolled system . . . . . . . . . . . . . . . . . . . . . . . . 90
4.3 Discretization and triangulation of the state space . . . . . . . . . 93
4.4 $50 \times 50$ grid and $\sum_{i=1}^{m-1} N_i = 10000$ . . . . . . . . . . . . . . . . 95

4.5 $50 \times 50$ grid and $\sum_{i=1}^{m-1} N_i = 100000$ . . . . . . . . . . . . . . . 96
4.6 $25 \times 25$ grid and $\sum_{i=1}^{m-1} N_i = 10000$ . . . . . . . . . . . . . . . 97
4.7 Score obtained as a function of $\sum_{i=1}^{m-1} N_i$ for different state space discretizations. . . . . . . . . . . . . . . . . . . . . . . . . . 98
4.8 Representative states technique. $50 \times 50$ grid used . . . . . . . . 99
4.9 Representative states technique. $25 \times 25$ grid used . . . . . . . . 100
4.10 Phase plane . . . . . . . . . . . . . . . . . . . . . . . . . . . 101
4.11 Rapid changes of the control variable value around the equilibrium point . . . . . . . . . . . . . . . . . . . . . . . . . . . . 104

5.1 Generic reinforcement learning method . . . . . . . . . . . . . . 112
5.2 *Action selection* : $\epsilon$-Greedy policy . . . . . . . . . . . . . . . . . 114
5.3 *Learning* : model based technique . . . . . . . . . . . . . . . . . 115
5.4 *Estimation of the MDP structure* : Kalman Filter like algorithm . 116
5.5 *Estimation of the MDP structure* : Stochastic Approximation algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . 117
5.6 *Resolution of the MDP :* Prioritized sweeping algorithm . . . . . 121
5.7 $Q$-function estimated during the learning process . . . . . . . . . 122
5.8 Exploration-exploitation tradeoff . . . . . . . . . . . . . . . . . . 122
5.9 Score curves for different variants of the *Learning* module. . . . . 123
5.10 *Learning* : non-model based technique . . . . . . . . . . . . . . 124
5.11 *$Q$-function correction* : $Q$-learning algorithm . . . . . . . . . . . 125
5.12 $Q(\lambda)$ : $Q$-function correction . . . . . . . . . . . . . . . . . . . 127
5.13 *$Q$-function correction* : Watkins's $Q(\lambda)$ algorithm . . . . . . . . 128
5.14 $Q$-function estimated during the learning process . . . . . . . . . 129
5.15 Four-state control problem : score curves . . . . . . . . . . . . . 130
5.16 The $\alpha_k$ terms satisfy the convergence conditions . . . . . . . . . . 130

6.1 *Learning* : non-model based technique . . . . . . . . . . . . . . 134
6.2 *Learning* : model based technique . . . . . . . . . . . . . . . . . 134
6.3 *Action selection* : $\epsilon$-Greedy policy . . . . . . . . . . . . . . . . . 139
6.4 Aggregation technique : trajectory in a discretized continuous state space . . . . . . . . . . . . . . . . . . . . . . . . . . . 140
6.5 Representative states technique : trajectory in a triangulated state space . . . . . . . . . . . . . . . . . . . . . . . . . . . 142
6.6 *Estimation of the MDP$^\delta$ structure* : Kalman Filter like algorithm used with the aggregation technique . . . . . . . . . . . . . . . 143
6.7 *Estimation of the MDP$^\delta$ structure* : Stochastic Approximation algorithm used with the aggregation technique . . . . . . . . . . . 146

6.8 $Q^{\delta}$*-function correction* : $Q$-learning algorithm used with the aggregation technique . . . . . . . . . . . . . . . . . . . . . . . . . 148
6.9 $Q^{\delta}$*-function correction* : Watkins's $Q(\lambda)$ algorithm used with the aggregation technique . . . . . . . . . . . . . . . . . . . . . . 150
6.10 Representation of $\tilde{Q}$ at different stages of the learning process . . 151
6.11 Score curves . . . . . . . . . . . . . . . . . . . . . . . . . . . 152
6.12 *Estimation of the MDP*$^{\delta}$ *structure* : Kalman Filter like algorithm used with the representative states technique . . . . . . . . . . . 154
6.13 *Estimation of the MDP*$^{\delta}$ *structure* : Stochastic Approximation algorithm used with the representative states technique . . . . . . . 157
6.14 $Q^{\delta}$*-function correction* : $Q$-learning used with the representative states technique . . . . . . . . . . . . . . . . . . . . . . . . . 161
6.15 $Q^{\delta}$*-function correction* : Watkins's $Q(\lambda)$ used with the representative states technique . . . . . . . . . . . . . . . . . . . . . . . 162
6.16 Representation of $\tilde{Q}$ at different stages of the learning process . . 163
6.17 Score curves for different variants of the *Learning* module. . . . . 164
6.18 Score curves . . . . . . . . . . . . . . . . . . . . . . . . . . . 165
6.19 Kalman Filter like algorithm vs Stochastic Approximation algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 166

7.1 Control law at different stages of the learning process . . . . . . . 174
7.2 Score curves for RL algorithms used with the aggregation technique 175
7.3 Score curves for RL algorithms used with the aggregation technique 176
7.4 Score curves for RL algorithms used with the representative states technique . . . . . . . . . . . . . . . . . . . . . . . . . . . . 178
7.5 Problem with the representative states technique and the boundary 179
7.6 Comparison between RL algorithms used either with the representative states technique or the aggregation technique . . . . . . . . 180
7.7 Score obtained as a function of learning time $LT$ . . . . . . . . . 182

8.1 Trajectories in the state space . . . . . . . . . . . . . . . . . . . . 187
8.2 Value function and solutions of the HJB equation . . . . . . . . . 189
8.3 Discretization of the state space . . . . . . . . . . . . . . . . . . 191
8.4 Discretization of HJB . . . . . . . . . . . . . . . . . . . . . . . 194
8.5 Approximation of the optimal stationary policy computed by using a finite differences method to discretize HJB equation . . . . . . . 197

9.1 A four-machine power system . . . . . . . . . . . . . . . . . . 204

9.2 The system operates in steady-state conditions. The RL algorithm takes control of the TCSC at $t = 5\,s$. . . . . . . . . . . . . . . . 211
9.3 Representation of the self-sustained electric power oscillations . . 212
9.4 After ten minutes of control . . . . . . . . . . . . . . . . . . . . . 213
9.5 After one hour of control . . . . . . . . . . . . . . . . . . . . . . 213
9.6 After five hours of control . . . . . . . . . . . . . . . . . . . . . 214
9.7 $\overline{R_t}$ as a function of the learning time . . . . . . . . . . . . . . . 215
9.8 $\overline{R_t}$ as a function of the learning time . . . . . . . . . . . . . . . 216
9.9 Representation of the electric power oscillations caused by a 60 $ms$ three-phase short-circuit at bus $\#10$ . . . . . . . . . . . . . . . 217
9.10 Representation of $P_e - t$ at different stages of the learning process 218
9.11 Number of times the loss of stability has been observed . . . . . . 220
9.12 $R_t$ obtained after convergence of the RL algorithm compared to $R_t$ obtained while using a fixed capacitor. $\Delta P_e = 5$ MW . . . . . . 221
9.13 Representation of $P_e - t$ after convergence of the algorithm. $\Delta P_e =$ 5 MW . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 221
9.14 Representation of $u - t$ after convergence of the algorithm. $\Delta P_e =$ 5 MW . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 222
9.15 $R_t$ obtained after convergence of the RL algorithm compared to $R_t$ obtained while using a fixed capacitor. $\Delta P_e = 2$ MW . . . . . . 222
9.16 Representation of $P_e - t$ after convergence of the RL algorithm. $\Delta P_e = 2$ MW . . . . . . . . . . . . . . . . . . . . . . . . . . 223
9.17 Representation of $u - t$ after convergence of the RL algorithm. $\Delta P_e = 2$ MW . . . . . . . . . . . . . . . . . . . . . . . . . . 223

10.1 Power system and control : observations, agent, action, device and physical influence . . . . . . . . . . . . . . . . . . . . . . . . . 229
10.2 Two modes of application . . . . . . . . . . . . . . . . . . . . . 234

B.1 Cube triangulation. Taken from [Epp]. . . . . . . . . . . . . . . . 261

# List of Tables

2.1 Value iteration algorithm : illustration on the four-state control problem . . . 52
2.2 $Q$-function representation . . . 55
2.3 $\epsilon$-Greedy policy . . . 57

3.1 Solution of the MDP$^\delta$ . . . 63
3.2 Transition probabilities computed by using the triangulation technique . . . 77
3.3 Score obtained for different values of $\epsilon$ . . . 82
3.4 Score obtained for different discretizations of the state space . . . 84
3.5 Score obtained for different sizes of the disturbance space subsets 85

4.1 Scores obtained for different ways to compute the MDP$^\delta$ . . . 102
4.2 Combined use of multiple MDP$^\delta$ . . . 106
4.3 Robustness of the control law to a parameter change . . . 107
4.4 Robustness of the control law to noise . . . 108
4.5 Robustness with respect to system modeling . . . 109

7.1 Score for different $P_0(x)$ as a function of the learning time . . . 172

B.1 Comparison between the minimum number of $n$-simplices required to triangulate the $n$-cube and $n!$ . . . 261
B.2 Score obtained for different triangulations. The OMIB power system with AVR. $\Delta E = 0.2$ . . . 262

# Chapter 1

# Introduction

This research is motivated by our interest to solve practical electric power systems control problems. To tackle such problems we formulate them as *optimal control problems* and search for an optimal *closed-loop control policy* i.e., a control policy which uses real-time measurements to decide which control action to take with the objective of maximizing some performance index. Among the approaches to solve optimal control problems, we focus on *Reinforcement Learning,* which concerns the development of algorithms to approximate optimal control policies in an automatic way.

In the following sections we first provide an intuitive presentation of reinforcement learning and of the dynamic programming ideas on which reinforcement learning is based. Then we outline the logical organization of this work and provide some reading guidelines. We conclude this introduction with a statement of our main personal contributions.

## 1.1 Power system control and reinforcement learning

Learning from interaction with an environment is probably the most natural form of learning. Humans acquire an immense source of knowledge from interaction with their environment. When they hold a conversation, drive a car, or play tennis, they are acutely aware of the effect they have on their environment and try to analyze these effects in order to improve their behavior.

In this work, we explore how a *computational approach* to learning from interaction can be applied to control power systems. The computational approach considered is called *reinforcement learning* (RL). Reinforcement learning is learning from interaction what to do i.e., how to map situations to actions, so as to maximize a numerical reward signal. In this framework the learning agent is not told explicitly which actions to take but instead must discover himself which actions yield the highest rewards by trying out various possible actions and by observing their consequences.

The interests in reinforcement learning methods for power system control are huge. Indeed, they offer the possibility to design intelligent agents able to learn an appropriate control strategy from interaction with the power system[1].

At least two types of applications are possible. One is to use these RL driven agents in a simulation environment and to exploit the control strategy they have learned to control the real power system. The other application consists in using these agents directly on the real power system. Although this latter application is the most attractive one, it is also the most difficult to achieve. Indeed, one must guarantee that during his interaction with the power system the RL driven agent will not, notably by lack of knowledge, adopt a behavior that can jeopardize the system integrity.

The aim of this thesis is to explore these different applications and to set up reinforcement learning algorithms able to match power system control requirements.

Although the reinforcement learning methods developed in this work can be used with a vast panel of power system control problems, in the examples we focus mainly on control problems involving a Thyristor Controlled Series Capacitor (TCSC) device. This device is built by using power electronics and has roughly the ability to modify the line impedance. The reinforcement learning algorithms are able to control it (i.e., to impose on the TCSC ways of modifying the line impedance) and to gather information about the power system dynamics. They will have to learn from this interaction an appropriate control strategy for the TCSC device.

## 1.2 Framework used to present RL methods

Reinforcement learning methods will be presented in this thesis as a way of learning the (approximate) solution of optimal control problems.

The optimal control problems that we consider are multi-stage decision problems. The outcome of each decision is not considered as fully predictable but can be anticipated to some extent by knowing the system dynamics and a reward function. Each decision results in some immediate reward but also influences the context in which future decisions are to be made and therefore affects the reward incurred in future stages. We are interested in decision making policies that maximize the total

[1] Applications of reinforcement learning to power system control are more extensively discussed in chapter 10.

reward over a number of stages. **Dynamic Programming** is used to provide the mathematical formalization of the tradeoff between immediate and future rewards. Among the different types of optimal control problems, we consider mainly discrete-time control problems with infinite horizon. Although such problems do not exist in practice since every physical process is bounded in time, their analysis is elegant and their optimal policies are simple. Among the different types of infinite horizon problems that have been studied in the literature we consider, notably for convergence reasons, only the discounted problems for which the rewards importance decays exponentially with their time of appearance. As we will observe in the applications, this type of optimal control problem yields control policies that are easily implementable and are able to control the power system correctly.

## 1.3 Outline of the thesis

### 1.3.1 Structure of the main body

Reinforcement learning methods do not use any analytical knowledge of the elements composing the optimal control problem[2] to learn its solution. They consider these elements as **unknown** and compensate for this absence of knowledge by the possibility to learn by interacting with the system. Rather than focusing directly on the reinforcement learning methods, we will first present methods that solve optimal control problems when the system dynamics and the reward function are known.

Many variables that describe the power system dynamics being continuous, we will have to deal with state spaces composed of an **infinite** number of states. Control problems with infinite state spaces being more difficult to solve than control problems with finite state spaces, we will focus first on these latter.

More specifically, we will basically concentrate on the following four problems.

- *Problem I* : resolution of an optimal control problem with **known** system dynamics and reward function and **finite** state space.

- *Problem II* : resolution of an optimal control problem with **known** system dynamics and reward function and **infinite** state space.

- *Problem III* : resolution of an optimal control problem with **unknown** system dynamics and reward function and **finite** state space.

[2]These elements are the system dynamics and the reward function.

- *Problem IV* : resolution of an optimal control problem with **unknown** system dynamics and reward function and **infinite** state space.

These four problems are treated in the following way.

- *Problem I.* To solve this problem we use classical dynamic programming algorithms. These algorithms are described in chapter 2. At the beginning of this chapter we define also the type of optimal control problem used in the thesis (except for chapter 8) and characterize its solution.

- *Problem II.* Generally speaking, it is not possible to compute exactly the solution of infinite state space control problems. Therefore, we propose some strategies to determine approximate solutions. We focus mainly on strategies that consist in defining from the infinite state space control problem knowledge a finite state space control problem able to catch its main features. This finite state space control problem is solved and its solution is extended to the initial control problem. We investigate two different such strategies in chapter 3.

- *Problem III.* To compensate for the absence of knowledge of the system dynamics and the reward function, one has the possibility to interact with the system. Two approaches are possible. The first reconstructs the control problem structure and solves it. The second approach learns directly the control problem solution without reconstructing its structure. Both approaches are described in chapter 5.

- *Problem IV.* The absence of knowledge of the system dynamics and the reward function is compensated by the possibility to interact with the system. We propose two different approaches. Roughly speaking, both approaches consist in determining the solution of a finite state space control problem and in extending its solution to the initial control problem (procedure similar to the one used to solve *Problem II*). One approach reconstructs the finite state space control problem structure and solves it while the other learns directly its solution. Chapter 6 is dedicated to this *Problem IV*.

The theoretical derivations of solutions for these four different problems are provided respectively in chapters 2, 3, 5, 6, together with illustrative (academic) examples. Chapters 4 and 7 complement this material with an in-depth empirical analysis of the various algorithms proposed by simulations carried out on a simple but still representative power system control problem.

Chapter 8 is a short incursion into the field of continuous-time optimal control problems. The theory underlying the Hamilton-Jacobi-Bellman equation is first outlined and a numerical discretization technique to solve it is described. This algorithm is then applied to the test problem used in chapter 4, and the resulting continuous-time control policy is compared with the previously obtained discrete-time one. Analogies are drawn between the discretization techniques used in this chapter and those used in chapters 3 and 6.
In chapter 9, we describe several additional problems (partial observability, curse of dimensionality, time variance of the system, $\cdots$) met when applying reinforcement learning methods to control real world systems and discuss some strategies that can be adopted to solve these problems. An application of reinforcement learning methods to control a FACTS device installed on a four-machine power system is also carried out in this chapter.
In chapter 10 we highlight the main challenges in power system control and discuss how some (or many) of these challenges could be met by using reinforcement learning methods.
The last chapter of the thesis is devoted to conclusions and discussions about open issues and directions for future research.

### 1.3.2 About the appendices

We have removed from the main text of the thesis some detailed derivations which are not strictly necessary for its understanding.
For the sake of completeness these derivations are collated in three appendices dealing respectively with linear approximation, triangulation, and algorithms based on contraction assumptions. The interested reader may have a quick look at these appendices before starting to read the main text, or refer to them on the fly.

## 1.4 Main contributions of the thesis

We discuss separately our contributions to the field of reinforcement learning and dynamic programming and our contributions in the context of electric power systems control.

### 1.4.1 Reinforcement learning and dynamic programming

The theoretical part of this thesis is the synthesis of our own study during the last two years and classical material found in the literature on dynamic programming and reinforcement learning (or neuro-dynamic programming). Starting from the basic results in finite Markov decision problems, we provide a unified framework to formulate and study a broad class of continuous state space optimal control and reinforcement learning algorithms. Below, we enumerate some specific technical contributions that we believe are new and hopefully interesting.

- *The representative states technique.* We introduce and study this discretization technique to transform a continuous state space control problem into a finite Markov Decision Process (MDP). The use of this technique, already known in the context of continuous-time optimal control problems (e.g. [Mun00]), is new in the context of discrete-time optimal control problems.

- *Model based reinforcement learning algorithms.* We combine two discretization techniques (the representative states technique and the aggregation technique) with model based RL algorithms to treat continuous state space problems. The way the finite MDPs are defined as well as the way their solution is extended to the initial control problem are specific to this thesis (chapter 6). While most recent work in RL focuses on non-model based techniques, we found out that model based ones provide better performances in the context of our applications.

- *Equivalence conditions.* We introduce theoretical equivalence conditions between solutions based on finite MDP based approximation architectures and the solution of the original continuous state space control problem and show how these equivalence conditions can be used as a framework to elaborate algorithms to reconstruct the approximating MDP structure either from the knowledge of system dynamics and reward functions or from samples along system trajectories (chapters 3 and 6).

- *MDP estimation algorithms.* We introduce and study reinforcement learning algorithms that learn the structure of a finite MDP aimed to catch the main features of the infinite state space control problem (chapter 6). Two types of algorithms are introduced to estimate the finite MDP structure. One is based on the Kalman filter algorithm and the other on a stochastic approximation algorithm.

- *Unification of model based and non-model based algorithms.* We provide a unified view and a systematic theoretical study of model based and non-model based reinforcement learning algorithms (chapters 5 and 6).

- *Implementations, empirical studies and applications.* We first provide a systematic comparison of various algorithms on a simple FACTS control problem (chapters 4, 7, 8). Then we apply RL techniques to real-world control problems with continuous state spaces in various conditions (chapter 9).

We mention also some work not reported in this thesis on the development of adaptive discretization methods for reinforcement learning [BEW03].

### 1.4.2 Power system control

The practical part of the thesis is the result of personal work and collaborations with other colleagues during the last three years. We have implemented ourselves all the algorithms used in this context, except for some general purpose power system simulation software that was already available. Our main contributions in this context are the following.

- *Reinforcement learning methods for power system control.* Reference [DEW00] is one of the earliest attempts to use reinforcement learning in power systems. It is the first attempt to develop an automatic generation shedding system by such an approach. This application is briefly reported in chapter 10.
- *TCSC control by reinforcement learning.* References [Ern01, EW02] are the first publications on the control of such a device by reinforcement learning. These publications partly overlap with the material reported in chapters 7 and 9 of this work.
- *General framework for power system control.* In chapter 10 we propose a general framework for the development of power system control agents based on reinforcement learning.

Finally, reference [EGW03] reports on work not reported in this thesis dealing with the application of reinforcement learning for power system control .

**Chapter 2**

# Dynamic programming

*In this chapter, dynamic programming is used to handle discrete-time optimal control problems with infinite time horizon. Although these infinite time horizon control problems do not exist in practice because physical processes are bounded in time, their analysis is elegant and the implementation of the optimal policies computed is simple because they are stationary. Among the different types of infinite horizon problems we consider only the discounted problems for which the rewards are weighted so that the importance of the rewards decays exponentially with their time of appearance. Furthermore we will suppose that the rewards are bounded.*
*In this chapter we successively*

- *define the elements the optimal control problem is made of and introduce the Dynamic Programming (DP) equation as a way of characterizing the control problem solution;*

- *present iterative methods to solve the DP equation and to evaluate the return of some particular policies;*

- *explain how to transform the control problem into a Markov Decision Process (MDP);*

- *present some algorithms (value iteration algorithm, policy iteration algorithm, $\cdots$) to solve the optimal control problem when the state space and the control space are finite.*

*For more information about dynamic programming, and for the proofs which we omit, the reader may refer to [Ber00] and [Ber95].*

## 2.1 Dynamic programming : the basics

In this section we describe the main elements that make up a dynamic programming problem and give some important results of the dynamic programming theory. For the sake of simplicity we consider first control problems for which the state space, the control space and the disturbance space are finite.

### 2.1.1 Finite spaces

#### The system dynamics

One of the elements that will make up the dynamic programming (DP) problem is a discrete-time dynamical system described by the generic equation :

$$x_{t+1} = f(x_t, u_t, w_t), \quad t = 0, 1, \cdots \tag{2.1}$$

where for all $t$, the state $x_t$ is an element of a finite space $X$, the control $u_t$ is an element of a finite space $C$ and the random disturbance $w_t$ is an element of a finite space $D$. The control $u_t$ is constrained to take values on a non-empty subset $U(x_t)$ of $C$ which depends on the current state $x_t$. The random variables $w_t$ have identical statistics and are characterized by the probability distribution $P_w(.|x_t, u_t)$ defined on $D$, where $P_w(w_t|x_t, u_t)$ represents the probabilities of occurrence of $w_t$, when the current state and control are $x_t$ and $u_t$, respectively. The probability of $w_t$ may depend explicitly on $x_t$ and $u_t$ but not on values of prior disturbances $w_{t-1}, \cdots, w_0$.

#### The rewards

Another element that intervenes in the definition of the DP problem is a *reward function* $r : X \times C \times D \to \mathbb{R}$ and its associated *discount factor* $\gamma$ with $0 < \gamma \leq 1$ that weights the rewards. In this work we consider *discounted problems*, that is problems for which[1] $\gamma < 1$. The rewards we consider are supposed to be bounded, that is there exists a value $B_r$ such that $|r(x, u, w)| < B_r$ for all $x \in X$, $u \in U(x)$ and $w \in D$.

[1] Although most of the time we will consider $\gamma$ as constant, optimal control problems for which $\gamma$ depends on $x$ and $u$ can be treated similarly. Such problems are encountered in chapter 8.

The *return over $N$-stages* is denoted by $R^N$ and is equal to the *sum of the weighted rewards observed* over the $N$-stages defined as follows[2] :

$$R^N = \sum_{t=0}^{N-1} \gamma^t r(x_t, u_t, w_t). \tag{2.2}$$

We mainly focus on the case where $N \to \infty$. In this context we talk about *return* rather than return over $\infty$-stages and denote it by $R$. One should notice that a reward obtained after time $t \neq 0$ has less importance on the return than a same magnitude reward observed at $t = 0$. Indeed, it intervenes in the return definition multiplied by a factor $\gamma^t < 1$.

**Admissible control policies**

The objective of the controller is to maximize in some sense (to be defined below) the overall reward $R$. The control policy represents the method used by the controller to select at each time $t$ a control $u_t$ constrained to stay in $u_t \in U(x_t)$.
At time $t$, when choosing the value of $u_t$, the controller has at its disposal the full history of the states already met and of the actions already selected at previous time steps ($t = 0, \dots, t-1$), as well as the value of the present state $x_t$. We denote this history by

$$h_t = (x_0, \cdots, x_t, u_0, \cdots, u_{t-1}).$$

Based on this information the controller may select actions either in a deterministic or in a stochastic way.
To formalize these notions we define the set of all admissible histories at time $t$ by

$$H_t = \{(x_0, \cdots, x_t, u_0, \cdots, u_{t-1}) | x_k \in X, u_j \in U(x_j), k \in \{1, \cdots, t\}, j \in \{1, \cdots, t-1\}\}$$

and an admissible control policy by a sequence $\pi = \{\pi_1, \pi_2, \cdots\}$ of conditional probability distributions $\pi_t(u_t|h_t)$, such that for every $h_t \in H_t$, $\sum_{u \in U(x_t)} \pi_t(u|h_t) = 1$.
We denote by $\Pi$ the set of all admissible control policies. A particular element of $\Pi$ defines a particular method to generate a sequence $\{u_t\}$ of actions $u_t \in U(x_t)$ such that for every history $h_t$ and $t = 0, 1, \cdots$ the distribution of $u_t$ is $\pi_t(.|h_t)$.

[2]The value $r(x_t, u_t, w_t)$ is also denoted by $r_t$.

### Controlled system trajectories

Once a control policy is adopted the process evolution is well defined. Trajectories of the system are generated according to the following mechanism.

- At time $t = 0$ an initial state $x_0$ is chosen according to a probability distribution $P_0(.)$, and the history is initialized to $h_0 = (x_0)$.
- At time $t$ the controller selects an action $u_t \in U(x_t)$ according to the given distribution $\pi_t(.|h_t)$.
- The disturbance $w_t$ is generated according to the probability distribution $P_w(.|x_t, u_t)$.
- The value of the reward function is determined and equal to $r_t = r(x_t, u_t, w_t)$.
- The next state is generated according to the system equation

  $$x_{t+1} = f(x_t, u_t, w_t)$$

  and the history is updated by appending the control action taken $u_t$ and the new state reached $x_{t+1}$ to the value of the previous history $h_t$.

This mechanism defines for every finite time $t$, a joint probability distribution over the variables $(x_0, \ldots, x_t, u_0, \ldots, u_t, w_0, \ldots, w_t)$ i.e., a discrete-time discrete-space stochastic process.

### Markov policies

The class $\Pi$ of control policies is the most general class of control policies that one can envisage in order to optimize system behavior. We now introduce some simpler, so-called *Markov* classes of control policies, which choose the control action $u_t$ only on the basis of the state at time $t$:

- *Randomized Markov policies.* A policy is a randomized Markov policy if it corresponds to a policy that selects at time $t$ the action $u_t$ according to a probability distribution $\mu_t^p(.|x_t)$ with $\sum_{u \in U(x_t)} \mu_t^p(u|x_t) = 1$. Such policies are denoted for brevity by $\mu_t^p$. These policies use only from the history $h_t$ the value of the current state to determine $u_t$.
- *Deterministic (nonrandomized) Markov policies.* A policy is a deterministic Markov policy if it corresponds to a policy that selects at time $t$ the control $u_t = \mu_t(x_t)$ where $\mu_t : X \to C$ with $\mu_t(x) \in U(x)$ $\forall x \in X$.

- *Randomized stationary (Markov) policies.* A policy is a randomized stationary policy if it corresponds to a policy that selects at time $t$ the action $u_t$ according to a probability distribution $\mu^p(.|x_t)$ with $\sum_{u \in U(x_t)} \mu^p(u|x_t) = 1$. Such policies are denoted for brevity by $\mu^p$.

- *Deterministic stationary (Markov) policies.* A policy is a deterministic stationary policy if it corresponds to a policy that selects at time $t$ the control $u_t = \mu(x_t)$ where $\mu : X \to C$ with $\mu(x_t) \in U(x_t)\ \forall x \in X$. Such policies choose while being in a state $x$ always the same action and are denoted for brevity by $\mu$.

One can observe that these Markov classes of policies are particular cases of the general policies $\Pi$. They are much simpler to describe, study and implement, and, as we will see soon, they are sufficiently general to solve our optimal control problem.

**The expected return over $N$-stages of a policy**

Among the system trajectories of length $N$, let us focus on the subset of trajectories starting from a particular initial condition $x_0 = x$. Then, the *expected return over $N$-stages of a policy* $\pi$ acting on a system starting from state $x$, is defined by the following expression :

$$J_N^\pi(x) \quad = \underset{\substack{(x_0,\cdots,x_{N-1},\\ u_0,\cdots,u_{N-1},\\ w_0,\cdots,w_{N-1})}}{E}[\sum_{t=0}^{N-1} \gamma^t r(x_t, u_t, w_t)|x_0 = x] \tag{2.3}$$

where the expectation is taken over all system trajectories according to the conditional probability distribution

$$P(x_0, \ldots, x_{N-1}, u_0, \ldots, u_{N-1}, w_0, \ldots, w_{N-1}|x_0 = x).$$

Notice that this conditional probability distribution can be derived by recursion from the different elements defining our control problem and the policy $\pi$, and assuming that $P_0(x_0) = \delta(x, x_0)$. In order to highlight this, we set

$$\mathcal{H}_t = (x_0, \cdots, x_t, u_0, \cdots, u_t, w_0, \cdots, w_t) \quad t \in \{0, \cdots, N-1\}.$$

We therefore have for all $t \in \{1, \cdots, N-1\}$ [3]:

$$P(\mathcal{H}_t) = P(\mathcal{H}_{t-1}) I_{\{x_t\}}(f(x_{t-1}, u_{t-1}, w_{t-1}))\pi_t(u_t|h_t)P_w(w_t|x_t, u_t) \tag{2.4}$$

[3] $I_B$ is the indicator function of a set $B$. $I_B(x) = 1$ if $x \in B$ and 0 otherwise.

with $P(\mathcal{H}_0) = \delta(x, x_0)\pi_0(u_0|x_0)P_w(w_0|x_0, u_0)$.
Expression (2.4) provides a way of computing $P(\mathcal{H}_{N-1})$ necessary to evaluate (2.3).

**The (infinite horizon) expected return**

The *expected return over an infinite number of stages (or expected return) of a policy* $\pi$ is given by[4] :

$$J^{\pi}(x) = \lim_{N\to\infty} J_N^{\pi} = \lim_{N\to\infty} E[\sum_{t=0}^{N-1} \gamma^t r(x_t, u_t, w_t)|x_0 = x]. \tag{2.5}$$

**The maximum expected return over $N$-stages**

The upper bound of the expected return over $N$-stages is denoted by $J_N^*$ and is given by the following expression :

$$J_N^*(x) = \sup_{\pi\in\Pi} E[\sum_{t=0}^{N-1} \gamma^t r(x_t, u_t, w_t)|x_0 = x]. \tag{2.6}$$

One can show that under the assumption of finite (or even countable) state space, control space and disturbance space, there always exists at least one policy in $\pi^* \in \Pi$ such that $J_N^{\pi^*}(x) = J_N^*(x)$. Hence, $J_N^*(x)$ represents indeed the best expected return over $N$-stages that can be obtained by a policy $\pi \in \Pi$. In order to stress this fact, we will use "max" operator instead of the "sup" in the subsequent derivations, whenever this is justified.
Observe that while $J_N^*(x)$ is unique, the policy $\pi$ which realizes that maximum is not necessarily unique. Thus the maximum return is uniquely determined, but there may be several optimal policies which yield this return.

**The maximum expected return (over an infinite number of stages)**

The maximum expected return (or value function) is the limit of the maximum expected return over $N$-stages when $N \to \infty$. It is denoted by $J^*$(or $V$) and is defined as follows[4] :

$$J^*(x) = V(x) = \lim_{N\to\infty} \max_{\pi\in\Pi} E[\sum_{t=0}^{N-1} \gamma^t r(x_t, u_t, w_t)|x_0 = x]. \tag{2.7}$$

[4]We do not have any problem with the limit because it is an infinite sum of numbers that are bounded in absolute value by the decreasing geometric expression $\gamma^t B_r$.

### Sufficiency of Markov policies

Below we state three important results, which plainly justify to restrict our attention to the classes of Markov policies introduced above.

**Finite horizon return equivalence:** Given an initial probability distribution on $x_0$, for every $N$ and for every possible policy in $\Pi$ there exists a *Markov policy* which yields the same expected return over $N$-stages. Thus, in terms of the expected returns that can be obtained, the class of Markov policies is sufficiently large to cover all possible behaviors.

**Finite horizon optimality equivalence :** For every $N$ there exists a *deterministic Markov policy* whose expected return over $N$-stages is equal to $J_N^*(x)$ $\forall x \in X$. Therefore, even if the class of policies is restricted to the set of deterministic Markov policies, one still has the guarantee to find in it at least one policy whose return equals the maximum expected return over $N$-stages of all admissible policies.

**Infinite horizon optimality equivalence :** There exists a *deterministic stationary (Markov) policy* whose return is equal to $V$ everywhere on $X$.

A deterministic stationary policy whose expected return is equal to the maximum expected return everywhere is said to be an *optimal stationary policy* [5] and is denoted by $\mu^*$. *When facing an optimal control problem, our main objective will be to compute an optimal stationary policy.*

### The Dynamic Programming equation

One remarkable property of the maximum expected return is that it satisfies the equation[6] :

$$V(x) \quad = \quad \max_{u \in U(x)} \underset{w}{E}[r(x,u,w) + \gamma V(f(x,u,w))] \tag{2.8}$$

known as the **DP equation** or the **Bellman equation** and that it is the *unique* solution of this equation.
The knowledge of $V$ can be used to compute an optimal stationary policy. Indeed, it can be shown that among the stationary policies all and only those which satisfy

[5]We will often call abusively a policy *the* optimal stationary policy even if there may exist many optimal stationary policies.

[6]The expectation is computed by using $P(w) = P_w(w|x,u)$.

the following expression are optimal :

$$\mu^*(x) \quad = \quad \arg\max_{u \in U(x)} \underset{w}{E}[r(x,u,w) + \gamma V(f(x,u,w))]. \tag{2.9}$$

We remark that the system dynamics and the reward function have to be known to deduce $\mu^*$ from $V$.

**The $Q$-function**

We use the notation $X \times U$ to represent the set of all possible state-action pairs $(x,u)$. Therefore, we have $X \times U = \{(x,u) | x \in X \text{ and } u \in U(x)\}$.
We define the $Q$-function $Q : X \times U \to \mathbb{R}$ as follows :

$$Q(x,u) = \underset{w}{E}[r(x,u,w) + \gamma V(f(x,u,w))], \tag{2.10}$$

which implies :

$$V(x) = \max_{u \in U(x)} Q(x,u), \tag{2.11}$$

and therefore :

$$Q(x,u) = \underset{w}{E}[r(x,u,w) + \gamma \max_{u' \in U(f(x,u,w))} Q(f(x,u,w),u')]. \tag{2.12}$$

$Q(x,u)$ represents the value of the expected return obtained by starting from state $x$ when the first action taken is $u$ and when an optimal policy is followed in the aftermath. $Q(x,u)$ is referred to as the value of the $Q$-function for the state-action pair $(x,u)$.
The $Q$-function knowledge allows to directly compute an (actually all) optimal stationary policy(ies) through the following expression :

$$\mu^*(x) \quad = \quad \arg\max_{u \in U(x)} Q(x,u). \tag{2.13}$$

*Notice that in practice, the determination of an optimal stationary policy will most of the time be realized from the Q-function.*

**Characterization of the return of stationary policies**

Expected returns of a deterministic stationary policy and of a randomized stationary policy can be characterized by similar expressions as the one that characterizes the value function. Indeed, it can be shown that the expected return $J^{\mu}$ of a deterministic stationary policy is the unique solution of[7]

$$J^{\mu}(x) \quad = \quad \underset{w}{E}[r(x,\mu(x),w) + \gamma J^{\mu}(f(x,\mu(x),w))] \tag{2.14}$$

and that the expected return $J^{\mu^p}$ of a randomized stationary policy $\mu^p$ is the unique solution of[8]

$$J^{\mu^p}(x) \quad = \quad \underset{(w,u)}{E}[r(x,u,w) + \gamma J^{\mu}(f(x,u,w))]. \tag{2.15}$$

### 2.1.2 Infinite state spaces

We will consider later in this work optimal control problems for which the state space ($X$) and/or the disturbance space ($D$) and/or the control space ($C$) are infinite. The elements that make up the dynamic programming problem (the system dynamics, the reward function, the policy) can be defined similarly when the spaces are not finite anymore.
Concerning the results of the dynamic programming theory given in section 2.1.1, they remain valid if $C$ is a compact space and if some "continuity conditions" on $r(x,u,w)$ and $f(x,u,w)$ are satisfied[9]. Therefore if these conditions are satisfied one can still state among other things that :

- there exists a deterministic stationary policy whose return is equal to the maximum expected return
- the value function is the unique solution of the DP equation
- equation (2.9) is a necessary and sufficient condition characterizing all optimal stationary policies.

### 2.1.3 Remarks

#### Terminal states

During the control process a terminal state can be reached (a state for which the control process stops when it reaches it). The handling of systems with terminal

[7]Expression (2.14) is an immediate consequence of the Bellman equation by choosing $U(x) = \{\mu(x)\}$.

[8]The mathematical expectation is computed by using $P(w,u) = P_w(w|x,u)\mu^p(u|x)$.

[9]For a rigorous formulation of the conditions, the reader may refer to [HLL96] and [HLL99]. These conditions are supposed to be satisfied in all the optimal control problems we treat.

states does not differ in principle from that of systems that do not have any terminal states. Indeed a terminal state can be seen as a regular state in which the system is stuck and for which all the future rewards obtained in the aftermath are zero. Denoting by $x^t$ a terminal state, we thus have :

$$x^t = f(x^t, u, w) \tag{2.16}$$

$$r(x^t, u, w) = 0 \tag{2.17}$$

If such a state is reached after $T$ stages then the return obtained can be written :

$$R = \sum_{t=0}^{T-1} \gamma^t r(x_t, u_t, w_t). \tag{2.18}$$

Such terminal states will be used to model undesirable conditions that the controller should try to avoid reaching in any case. In our applications we will consider, for example, that a terminal state is reached when the power system leaves its stability domain. We will see in the applications how to penalize transitions toward such terminal states.

### Continuous-time dynamics

Although we assume discrete-time dynamics for the system, in this work we will consider many systems having a continuous underlying dynamics described by :

$$\dot{x} = f(x, u, w). \tag{2.19}$$

Discrete-time control on such a system means that at instant $t$ the controller observes the state of the system and sets *the evolution* of the control variable $u$ over the time interval $[t, t+1]$. Usually $u$ is set to be constant (see for example [Oga95]) over this interval but as it will be illustrated in chapter 3 having $u$ varying on the time interval can also have some interest.

## 2.2 Computation of $V$, $J^{\mu}$, $J^{\mu^p}$

The value function, the expected return of a deterministic stationary policy and the expected return of a randomized stationary policy are respectively the unique solutions of equations (2.8), (2.14) and (2.15). We describe hereafter iterative methods to solve these equations. Practical implementations of these iterative methods when the state space and the control space are finite are further detailed in section 2.3.

**Computation of $V$**

For any bounded function $J : X \to \mathbb{R}$, we consider the function obtained by applying the dynamic programming mapping ($T$) to $J$, and we denote it by :

$$(TJ)(x) = \max_{u \in U(x)} \mathop{E}_{w}[r(x,u,w) + \gamma J(f(x,u,w))] \quad \forall x \in X. \tag{2.20}$$

Since $(TJ)(.)$ is itself a bounded function defined on the state space $X$, we view $T$ as a mapping that transforms the bounded function $J$ on $X$ into a bounded function $TJ$ on $X$. Notice that $V$ has earlier been defined as a fixed point of this mapping. Actually, this $T$ mapping is a *contraction mapping*[10]. Indeed it can be shown that for any pair of bounded functions $J, \overline{J} : X \to \mathbb{R}$ we have :

$$\|TJ - T\overline{J}\|_{\infty} \leq \gamma \|J - \overline{J}\|_{\infty}.$$

$T$ being a *contraction mapping* it has a unique fixed point $V$ [11], and moreover the sequence of functions $T^N J$ converges (in $\infty$-norm) toward $V$, whatever the bounded function $J$ used as starting point.[12]
Consequently, the maximum expected return satisfies :

$$V(x) = \lim_{N \to \infty} (T^N J)(x) \quad \forall x \in X. \tag{2.21}$$

Expression (2.21) can be used to compute the maximum expected return, starting with an arbitrary bounded function as initial guess.

**Computation of $T^{\mu}$**

We introduce the $T^{\mu}$ mapping that transforms any bounded function $J : X \to \mathbb{R}$ into :

$$(T^{\mu}J)(x) = \mathop{E}_{w}[r(x,\mu(x),w) + \gamma J(f(x,\mu(x),w))] \quad \forall x \in X. \tag{2.22}$$

This mapping is also a *contraction mapping* whose unique fixed point is $J^{\mu}$. If applied an infinite number of times to any bounded function it converges to $J^{\mu}$. Consequently :

$$J^{\mu}(x) = \lim_{N \to \infty} (T^{\mu N} J)(x) \quad \forall x \in X. \tag{2.23}$$

[10]See section C.1 for the definition of a contraction mapping and the description of its properties.

[11]hence $V$ is indeed uniquely defined as the fixed point of this mapping

[12]Notice that the set of bounded real-valued functions defined on an arbitrary set $X$ is a Banach space.

**Computation of $\mathbf{J}^{\mu^{\mathbf{p}}}$**

We introduce the mapping $T^{\mu^p}$ that transforms any bounded function $J : X \to \mathbb{R}$ into :

$$(T^{\mu^p})J(x) = \underset{(w,u)}{E}[r(x,u,w) + \gamma J(f(x,u,w))] \quad \forall x \in X. \tag{2.24}$$

This mapping is also a *contraction mapping* whose unique fixed point is $J^{\mu^p}$. If applied an infinite number of times to any bounded function it converges to $J^{\mu^p}$. Consequently :

$$J^{\mu^p}(x) = \lim_{N\to\infty}(T^{\mu^p N}J)(x) \quad \forall x \in X. \tag{2.25}$$

**Example 2.2.1** We illustrate hereafter the use of expression (2.21) to compute the maximum expected return.
Suppose the system described by the discrete-time equation :

$$x_{t+1} = x_t + u_t \tag{2.26}$$

where the state space $X = \{x \in \mathbb{R} | x \in [1,3]\} \cup \{x^t\}$ ($x^t$ represents the terminal state of the system that is reached if $x$ goes outside the interval $[1,3]$), the control space $C = \{-1,1\}$ and $U(x) = C\,\forall x \in X \setminus \{x^t\}$. The reward function $r(x,u,w)$ is equal to 0 if $f(x,u) \in [1,3]$ and if $x = x^t$, 3 if $f(x,u) < 1$ and 5 if $f(x,u) > 3$. The decay factor $\gamma = 0.5$.
We start the computation with a function $J$ equal to zero on $X$ : $J(x) = 0\,\forall x \in X$.
Computation of $TJ(x)$ :

$$TJ(x) = \begin{cases} 3 & \text{if } x \in [1,2[ \\ 0 & \text{if } x = 2 \\ 5 & \text{if } x \in ]2,3] \\ 0 & \text{if } x = x^t \end{cases}$$

Computation of $T^2J(x)$ :

$$T^2J(x) = \begin{cases} 3 & \text{if } x \in [1,2[ \\ 2.5 & \text{if } x = 2 \\ 5 & \text{if } x \in ]2,3] \\ 0 & \text{if } x = x^t \end{cases}$$

Computation of $T^3J(x)$ :

$$T^3J(x) = \begin{cases} 3 & \text{if } x \in [1,2[ \\ 2.5 & \text{if } x = 2 \\ 5 & \text{if } x \in ]2,3] \\ 0 & \text{if } x = x^t \end{cases}$$

We notice that $T^3J = T^2J$ and therefore we have $V = T^2J(x)$. The computation process is represented on figure 2.1.

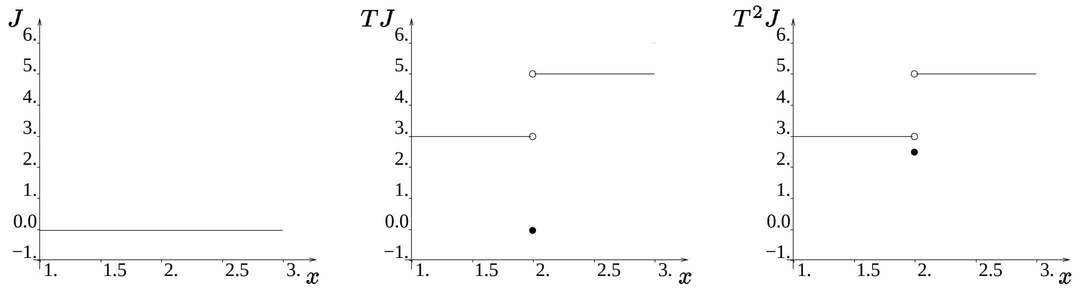


Figure 2.1: Representation of $T^kJ(x)$

The $Q$-function can be computed by using equation (2.10) :

$$Q(x,-1) = \begin{cases} 3 & \text{if } x \in [1,2[ \\ 1.5 & \text{if } x \in [2,3[ \\ 1.25 & \text{if } x = 3 \\ 0 & \text{if } x = x^t \end{cases}$$

$$Q(x,1) = \begin{cases} 1.25 & \text{if } x = 1 \\ 2.5 & \text{if } x \in ]1,2] \\ 5 & \text{if } x \in ]2,3] \\ 0 & \text{if } x = x^t \end{cases}$$

$Q(x,-1) > Q(x,1)\,\forall x \in [1,2[$ and $Q(x,-1) < Q(x,1)$ otherwise. From expression (2.13) we deduce the optimal stationary policy : $\mu^*(x) = -1$ if $x \in [1,2[$ and 1 otherwise. Although the system was composed of an infinite number of states, we managed to compute the optimal stationary policy and the maximum expected return. But usually problems with infinite state spaces cannot be solved so easily. Consider for instance a system dynamics slightly modified through the introduction of a noise factor :

$$x_{t+1} = x_t + u_t + w_t$$

where $w_t$ is drawn according to the standard Gaussian distribution and $C = [-1, 1]$. The computation of the value function and the optimal stationary policy by using expression (2.21) becomes much more difficult. In chapter 3 we provide methods to compute numerically an approximate solution of such optimal control problems.

## 2.3 Finite state and control spaces : computational methods

In this section we suppose that the state space ($X$) and the control space ($C$) are finite and we

- define the notion of a Markov Decision Process (MDP)
- explain how to compute the structure of the MDP that the optimal control problem corresponds to
- reformulate the $T$, $T^{\mu}$ and $T^{\mu^p}$ mappings by using the MDP structure to represent the system dynamics and reward function
- deduce from this new formulation a method to compute $J^{\mu}$
- introduce the *value iteration algorithm*, the *Gauss-Seidel version of the value iteration algorithm* and the *policy iteration algorithm* as a means of solving the optimal control problem.

### 2.3.1 Markov decision processes

A Markov Decision Process (MDP) is defined through the following objects : a state space $X$, a control space $C$, sets $U(x)$ of available actions at states $x \in X$, transition probabilities $p(x'|x, u)$, $\forall x, x' \in X$, $u \in U(x)$ and a reward function $r(x, u)$ denoting the one-stage reward obtained by using action $u$ in state $x$ [FS02]. MDPs can be seen as a particular type of the discrete-time optimal control problem stated in section 2.1 where the system dynamics is expressed using a transition probabilities formulation and where the reward does not depend on the disturbance parameter $w$ anymore.

### 2.3.2 Definition of the MDP structure from the system dynamics and reward function

If the state space is finite, it means that $X$ is now described by the finite set[13] :

$$X = \{x_1, x_2, \cdots, x_n\}. \tag{2.27}$$

The expected reward obtained by choosing action $u$ in state $x_i$ is given by :

$$r(x_i, u) \quad = \quad \underset{w}{E}[r(x_i, u, w)], \tag{2.28}$$

where the expectation is taken with respect to the conditional probability distribution $P_w(\cdot|x_i, u)$.
The transition probabilities can be computed from the system equation (2.1) and the known distribution of $P_w(.|x, u)$ of the input disturbance $w_t$. More precisely, we have

$$p(x_j|x_i, u) \quad = \quad \underset{w}{E}[I_{\{x_j\}}(f(x_i, u, w))], \tag{2.29}$$

where the expectation is taken with respect to the conditional probability distribution $P_w(\cdot|x_i, u)$.
Equations (2.28) and (2.29) define the structure of the Markov Decision Process equivalent to the original optimal control problem, in the sense that the expected return of any control policy applied to the original problem is equal to its expected return defined for this MDP in the following way :

$$J_N^\pi(x) = \underset{\substack{(x_0,\cdots,x_{N-1},\\ u_0,\cdots,u_{N-1})}}{E}[\sum_{t=0}^{N-1} \gamma^t r(x_t, u_t)|x_0 = x]. \tag{2.30}$$

**Example 2.3.1** In this example we introduce a control problem having a finite number of states. This control problem is used in the sequel to illustrate different types of algorithms and is referred to as the *four-state control problem.*
The system dynamics is described by the discrete-time equation :

$$x_{t+1} = x_t + u_t + w_t \tag{2.31}$$

where $X = \{1, 2, 3, x^t\}$, $C = \{-1, 1\}$, $U(x) = C\, \forall x \in X \setminus \{x^t\}$ and $D = \{-1, 0, 1\}$. If the value of $x_{t+1}$ computed by equation (2.31) is different from 1, 2 or 3 then the terminal

[13]The symbol $x_i$ ($i \in \mathbb{N}$) is used to denote an element of the finite state space $X$ as well as the state of the system at time $i$. Nevertheless, this notation will be used in a context that will remove any ambiguity.

state $x^t$ is reached. The probabilities of occurrence of $w_t$ are : $P_w(w_t = -1|x_t, u_t) = 0.25$, $P_w(w_t = 0|x_t, u_t) = 0.5$ and $P_w(w_t = 1|x_t, u_t) = 0.25$. The reward obtained is zero everywhere ($r(x_t, u_t, w_t) = 0$) except when the terminal state is reached. It equals $3$ if $x_t + u_t + w_t < 1$ and $5$ if $x_t + u_t + w_t > 3$. The decay factor $\gamma$ is taken equal to $0.5$.
By using equations (2.28) and (2.29), it is possible to compute the structure of the Markov Decision Process the control problem corresponds to.
By way of example we detail the computation of $r(2,1)$, that is the average reward obtained when taking $u = 1$ in state $x = 2$.

$$\begin{aligned} r(2,1) &= \sum_{w\in D} P_w(w|2,1)r(2,1,w) \\ &= P_w(-1|2,1)r(2,1,-1) + P_w(0|2,1)r(2,1,0) + P_w(1|2,1)r(2,1,1) \\ &= 0.25 * 0 + 0.5 * 0 + 0.25 * 5 \\ &= 1.25 \end{aligned}$$

By proceeding similarly we can compute the other terms $r(x,u)$. The results of the computation are : $r(1,-1) = 2.25$, $r(2,-1) = 0.75$, $r(3,-1) = 0$, $r(1,1) = 0$, $r(2,1) = 1.25$ and $r(3,1) = 3.75$.
To illustrate how the transition probabilities of the MDP are computed we detail the computation of $p(x^t|2,1)$, that is the probability to reach the terminal state when taking $u = 1$ in state $x = 2$.

$$\begin{aligned} p(x^t|2,1) &= \sum_{w\in D} P_w(w|2,1)I_{\{x^t\}}(f(2,1,w)) \\ &= P_w(-1|2,1)I_{\{x^t\}}(2) + P_w(0|2,1)I_{\{x^t\}}(3) + \\ &\quad P_w(1|2,1)I_{\{x^t\}}(x^t) \\ &= 0.25 * 0 + 0.5 * 0 + 0.25 * 1 \\ &= 0.25 \end{aligned}$$

By proceeding similarly we obtain : $p(1|1,1) = 0.25$, $p(2|1,1) = 0.5$, $p(3|1,1) = 0.25$, $p(x^t|1,1) = 0$, $p(1|2,1) = 0$, $p(2|2,1) = 0.25$, $p(3|2,1) = 0.5$, $p(x^t|2,1) = 0.25$, $p(1|3,1) = 0$, $p(2|3,1) = 0$, $p(3|3,1) = 0.25$, $p(x^t|3,1) = 0.75$, $p(1|1,-1) = 0.25$, $p(2|1,-1) = 0$, $p(3|1,-1) = 0$, $p(x^t|1,-1) = 0.75$, $p(1|2,-1) = 0.5$, $p(2|2,-1) = 0.25$, $p(3|2,-1) = 0$, $p(x^t|2,-1) = 0.25$, $p(1|3,-1) = 0.25$, $p(2|3,-1) = 0.5$, $p(3|3,-1) = 0.25$ and $p(x^t|3,-1) = 0$.

### 2.3.3 MDP structure computation

Equations (2.28) and (2.29) can be rewritten as follows :

$$r(x_i, u) = \arg\min_{a\in\mathbb{R}} \underset{w}{E}[(a - r(x_i,u,w))^2], \tag{2.32}$$

$$p(x_j|x_i, u) = \arg\min_{a\in\mathbb{R}} \underset{w}{E}[(a - I_{\{x_j\}}(f(x_i,u,w)))^2]. \tag{2.33}$$

In chapter 5, equations similar to (2.32) and (2.33) will be used to reconstruct the MDP structure from observations gathered by interacting with the system. In this context the expectation operator will be replaced :

- in the case of equation (2.32) by a sum on the rewards observed while taking action $u$ in state $x_i$
- in the case of equation (2.33) by a sum on the states reached while taking action $u$ in state $x_i$.

### 2.3.4 Rewriting of the mappings using a MDP structure

By using the MDP structure to rewrite the mapping $T$, we obtain :

$$(TJ)(x_i) = \max_{u \in U(x_i)} [r(x_i, u) + \gamma \sum_{x_j \in X} p(x_j | x_i, u) J(x_j)] \quad \forall x_i \in X. \tag{2.34}$$

Expression (2.34) can be easily evaluated for each $x_i \in X$.
Similarly we can rewrite the mapping $T^{\mu}$ as follows :

$$(T^{\mu}J)(x_i) = [r(x_i, \mu(x_i)) + \gamma \sum_{x_j \in X} p(x_j | x_i, \mu(x_i)) J(x_j)] \quad \forall x_i \in X. \tag{2.35}$$

The $T^{\mu^p}$ mapping can also be rewritten :

$$\begin{aligned}(T^{\mu^p}J)(x_i) \quad = \quad & \sum_{u \in U(x_i)} \mu^p(u|x_i)[r(x_i, u) \\ & + \gamma \sum_{x_j \in X} p(x_j | x_i, u) J(x_j)] \ \forall x_i \in X. \end{aligned} \tag{2.36}$$

### 2.3.5 Computation of $J^{\mu}$

There exists a straightforward way to estimate the value of $J^{\mu}$. Indeed if we consider the $n$-dimensional vectors

$$J = \begin{bmatrix} J(x_1) \\ J(x_2) \\ \vdots \\ J(x_n) \end{bmatrix}, \quad J^{\mu} = \begin{bmatrix} J^{\mu}(x_1) \\ J^{\mu}(x_2) \\ \vdots \\ J^{\mu}(x_n) \end{bmatrix}, \quad T^{\mu}J = \begin{bmatrix} T^{\mu}J(x_1) \\ T^{\mu}J(x_2) \\ \vdots \\ T^{\mu}J(x_n) \end{bmatrix}, \tag{2.37}$$

denote by $P^\mu$ the $n \times n$ transition probability matrix

$$P^\mu = \begin{pmatrix} p(x_1|x_1,\mu(x_1)) & \cdots & p(x_n|x_1,\mu(x_1)) \\ \vdots & \ddots & \vdots \\ p(x_1|x_n,\mu(x_n)) & \cdots & p(x_n|x_n,\mu(x_n)) \end{pmatrix} \tag{2.38}$$

and by $r^\mu$ the $n$-dimensional reward vector

$$r^\mu = \begin{pmatrix} r(x_1,\mu(x_1)) \\ \vdots \\ r(x_n,\mu(x_n)) \end{pmatrix} \tag{2.39}$$

then equation (2.35) can be rewritten in matrix form as follows

$$T^\mu J = r^\mu + \gamma P^\mu J. \tag{2.40}$$

The expected return of the policy $\mu$ is the unique solution of this equation. Hence we can write :

$$J^\mu = T^\mu J^\mu = r^\mu + \gamma P^\mu J^\mu. \tag{2.41}$$

This latter equation can be viewed as a system of $n$ equations with $n$ unknowns, which can also be written as

$$(I - \gamma P^\mu) J^\mu = r^\mu \tag{2.42}$$

or, equivalently,

$$J^\mu = (I - \gamma P^\mu)^{-1} r^\mu \tag{2.43}$$

where $I$ denotes the $n \times n$ identity matrix. The invertibility of the matrix is guaranteed since the DP equation has a unique solution for any reward function. Equation (2.43) represents a direct way to compute the expected return of a policy $\mu$.

### 2.3.6 Value iteration algorithm

The value iteration algorithm computes the value function through the use of expression (2.21). The algorithm is detailed in figure 2.2. It can also be used to compute the expected return of a policy ($J^\mu$) by considering that $U(x) = \{\mu(x)\}$.

Figure 2.2: The value iteration algorithm

```
Initialize arbitrarily $J(x_i)$ for all $x_i \in X$
Repeat indefinitely
    $errorMax = 0$
    Repeat once for all $x_i \in X$
        $J'(x_i) \leftarrow \max\limits_{u \in U(x_i)} [r(x_i, u) + \gamma \sum_{x_j \in X} p(x_j|x_i, u) J(x_j)]$
        $errorMax = \max(errorMax, |J'(x_i) - J(x_i)|)$
    $J \leftarrow J'$
    If $errorMax < \epsilon$ then stop the algorithm and use $J$ as an approximation of
    $V$
```

**Error bounds on $V$**

The parameter $\epsilon$ triggers the stopping of the algorithm. A too large value of $\epsilon$ can lead to important errors in the computation of $V$ while a too small value can cause the algorithm to iterate too many times. Notice that the precision of $V$ so computed is well controlled independently of the initial condition of the iterative algorithm. Indeed, it can be shown that for every initial vector $J$, state $x_i$ and iteration $k$ of the algorithm we have

$$\begin{aligned} (T^k J)(x_i) + \underline{c}_k &\leq (T^{k+1} J)(x_i) + \underline{c}_{k+1} \\ &\leq V(x_i) \\ &\leq (T^{k+1} J)(x_i) + \overline{c}_{k+1} \\ &\leq (T^k J)(x_i) + \overline{c}_k \end{aligned} \tag{2.44}$$

where the error bounds $\underline{c}_k$ and $\overline{c}_k$ are defined as :

$$\begin{aligned} \underline{c}_k &= \frac{\gamma}{1-\gamma} \min_{x_i \in X} [(T^k J)(x_i) - (T^{k-1} J)(x_i)] \\ \overline{c}_k &= \frac{\gamma}{1-\gamma} \max_{x_i \in X} [(T^k J)(x_i) - (T^{k-1} J)(x_i)]. \end{aligned}$$

By expression (2.44) one can see that $T^k J$ belongs to the interval :

$$[V - \overline{c}_k, V - \underline{c}_k]. \tag{2.45}$$

Therefore, the approximation of $V$ computed by using the value iteration algorithm lies necessarily in the interval :

$$[V - \frac{\gamma}{1-\gamma}\epsilon, V + \frac{\gamma}{1-\gamma}\epsilon]. \tag{2.46}$$

**Convergence to the optimal policy**

Let us denote by $J_\epsilon$ the estimate of $V$ returned by the value iteration algorithm. In order to compute an estimate of the optimal stationary policy from the estimated value function we have first to compute $TJ_\epsilon$ and then determine a policy $\mu_\epsilon$ satisfying the equation

$$T^{\mu_\epsilon} J = TJ_\epsilon. \tag{2.47}$$

Notice that the use of equation (2.47) to compute an estimate of the optimal stationary policy is equivalent to introducing the estimate of the value function computed by using the value iteration algorithm into the right side of equation (2.9) and compute for each state $x$ of the system the value of the control variable that maximizes it.
It can be shown that we have the following bound on the suboptimality of $\mu_\epsilon$ so computed :

$$\begin{aligned}\max_{x_i \in X}[V(x_i) - J^{\mu_\epsilon}(x_i)] \leq \; & \tfrac{\gamma}{1-\gamma}\left(\max_{x_i \in X}[(TJ_\epsilon)(x_i) - J_\epsilon(x_i)]\right. \\ & \left. - \min_{x_i \in X}[(TJ_\epsilon)(x_i) - J_\epsilon(x_i)]\right),\end{aligned} \tag{2.48}$$

and hence

$$\max_{x_i \in X}[V(x_i) - J^{\mu_\epsilon}(x_i)] \;\; \leq \;\; \frac{2\gamma^2\epsilon}{1-\gamma}. \tag{2.49}$$

From this inequality we can deduce that the computation of the optimal stationary policy by using the value iteration algorithm with a sufficiently small value of $\epsilon$ leads to the exact solution. Indeed, let us consider the set of all non-optimal stationary policies $\mu'$ and let us consider their optimality gap $|V - J^{\mu'}|_\infty$ with respect to the optimal policies. Since the number of possible stationary policies is finite ($X$

and $C$ being finite), the minimum gap of these suboptimal policies is bounded away from zero, say equal to $\delta > 0$. Under these conditions, using $\epsilon$ such that $\frac{2\gamma^2\epsilon}{1-\gamma} < \delta$, necessarily implies that

$$|V - J^{\mu_\epsilon}|_\infty = 0 \text{ and hence that } \mu_\epsilon = \mu^*.$$

### $Q$-function value iteration

The use of the value iteration algorithm requires, at the end of the estimation of the value function, an additional step to compute the optimal stationary policy. One way to avoid this step is to modify the value iteration algorithm in order to get an estimate of the $Q$-function rather than an estimate of $V$. The operations that are required in this respect to deduce the optimal stationary policy from the $Q$-function estimate consist only in computing for each $x$ of the system the value of $u \in U(x)$ that maximizes $Q(x, u)$. This version of the value iteration algorithm is represented in figure 2.3[14].

Figure 2.3: The value iteration algorithm : $Q$-function computation

```
Initialize arbitrarily $Q(x_i, u)$ for all $x_i \in X$ and for all $u \in U(x_i)$
Repeat indefinitely
    $errorMax = 0$
    Repeat once for all $x_i \in X$ and for all $u \in U(x_i)$
        $Q'(x_i, u) \leftarrow [r(x_i, u) + \gamma \sum_{x_j \in X} p(x_j|x_i, u) \max_{u' \in U(x_j)} Q(x_j, u')]$
        $errorMax = \max(errorMax, |Q'(x_i, u) - Q(x_i, u)|)$
    If $errorMax < \epsilon$ then stop the algorithm
    $Q \leftarrow Q'$
```

**Example 2.3.2** This example illustrates the value iteration algorithm on the four-state control problem detailed in example 2.3.1. Many of the concepts introduced in this section such as the error bounds and the estimation of the stationary policy suboptimality are also illustrated.

[14]Proof of convergence of this algorithm is given in section C.4.

Table 2.1 represents the values of $J$ obtained at different stages of the value iteration algorithm when $J$ is initialized to 0 everywhere.

Table 2.1: Value iteration algorithm : illustration on the four-state control problem

| $k$ | $(T^kJ)(1)$ | $(T^kJ)(2)$ | $(T^kJ)(3)$ | $(T^kJ)(x^t)$ | $\overline{c}_k$ | $\underline{c}_k$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | | |
| 1 | 2.25 | 1.25 | 3.75 | 0 | 3.75 | 0 |
| 2 | 2.53125 | 2.34375 | 4.21875 | 0 | 1.09375 | 0 |
| 3 | 2.56641 | 2.59766 | 4.27734 | 0 | 0.253906 | 0 |
| 4 | 2.5708 | 2.64404 | 4.28467 | 0 | 0.0463867 | 0 |
| 5 | 2.57135 | 2.65167 | 4.28558 | 0 | 0.00762939 | 0 |
| $\infty$ | 2.57143 | 2.65306 | 4.28571 | 0 | 0 | 0 |

Suppose that the value iteration algorithm is stopped when $k = 4$. By using expression (2.44), we can compute the interval that $V(1)$ belongs to. We obtain

$$\begin{array}{rc} & 2.57135 + \frac{0.5}{1-0.5}(0. - 0.) < V(1) < 2.57135 + \frac{0.5}{1-0.5}(2.64404 - 2.59766) \\ \Rightarrow & 2.57137 < V(1) < 2.61773 \end{array}$$

Of course, this result is consistent with the value of $V(1) = 2.57143$ computed by using an arbitrarily large number of iterations.
By using equation (2.47) it is possible to get an approximation $\mu$ of the optimal stationary policy from $T^4J$. This approximation must satisfy the equation $T^\mu J = T(T^4)J$. By way of example we detail the computation of $\mu(1)$. We have

$$\begin{aligned} T^{\mu(1)=1}(1) &= r(1,1) + \gamma * (p(1|1,1) * J(1) + p(2|1,1) * J(2) + \\ &\quad p(3|1,1) * J(3) + p(x^t|1,1) * J(x^t)) \\ &= 0 + 0.5 * (0.25 * 2.5708 + 0.5 * 2.64404 + 0.25 * 4.28467 + 0 * 0) \\ &= 1.51794 \\ T^{\mu(1)=-1}(1) &= r(1,-1) + \gamma * (p(1|1,-1) * J(1) + p(2|1,-1) * J(2) + \\ &\quad p(3|1,-1) * J(3) + p(x^t|1,-1) * J(x^t)) \\ &= 2.25 + 0.5 * (0.25 * 2.5708 + 0. * 2.64404 + 0. * 4.28467 + 0.75 * 0.) \\ &= 2.57135. \end{aligned}$$

Therefore one must have $\mu(1) = -1$. If we proceed similarly for $\mu(2)$ and $\mu(3)$ we obtain $\mu(2) = 1$ and $\mu(3) = 1$. The bound on the policy suboptimality can be computed through the use of expression (2.48) :

$$\begin{array}{rc} & \max_{x_i \in X}[V(x_i) - J^\mu(x_i)] < \frac{0.5}{1-0.5}(2.65167 - 2.64404) + \frac{0.5}{1-0.5}(0 - 0) \\ \Rightarrow & \max_{x_i \in X}[V(x_i) - J^\mu(x_i)] < 0.00763. \end{array}$$

### 2.3.7 Gauss-Seidel version of the value iteration algorithm

The value iteration algorithm implies the handling of two vectors $J$ and $J'$ where $J'$ stores the new value of the estimate of the value function. Only when the new estimate of the value function has been evaluated for all the states of the system, it replaces the old estimate $J$ in further computations.
Another type of algorithm known as the Gauss-Seidel version of the value iteration algorithm uses the new estimate of the maximum expected return of a state immediately to compute new estimates of the maximum expected return for the other states. This algorithm is described in figure 2.4[15].
The convergence of the Gauss-Seidel algorithm can be substantially faster than that of the value iteration algorithm (see [BT89]). But the comparison is sometimes misleading because the ordinary methods are usually used in combination with error bound methods ($\overline{c}_k$ and $\underline{c}_k$, see equation (2.44)).

Figure 2.4: The Gauss-Seidel version of the value iteration algorithm : $Q$-function computation

```
Initialize arbitrarily $Q(x_i, u)$ for all $x_i \in X$ and $u \in U(x_i)$
Repeat indefinitely
    $errorMax = 0$
    Repeat once for all $x_i \in X$ and for all $u \in U(x_i)$
        $oldValue = Q(x_i, u)$
        $Q(x_i, u) \leftarrow [r(x_i, u) + \gamma \sum_{x_j \in X} p(x_j|x_i, u) \max_{u' \in U(x_j)} Q(x_j, u')]$
        $errorMax = \max(errorMax, |Q(x_i, u) - oldValue|)$
    If $errorMax < \epsilon$ then stop the algorithm
```

There is also a more flexible form of the Gauss-Seidel method which selects states in arbitrary order to update their maximum expected return. This method maintains an approximation on $J$ to the optimal vector $V$, and at each iteration it selects and replaces $J(x_i)$ by $TJ(x_i)$. At each iteration, the choice of state $x_i$ is arbitrary except that all states must be selected an infinite number of times. This method is

[15]Proof of convergence of this algorithm is given in section C.4.

an example of an *asynchronous fixed point iteration* method [Ber81].

### 2.3.8 Policy iteration algorithm

The policy iteration algorithm (proposed first in [Bel57]) generates a sequence of stationary policies, each with improved expected return over the previous one.
Given the stationary policy $\mu$ and the corresponding expected return $J^{\mu}$, an improved policy $\mu'$ is computed by maximizing the DP equation corresponding to $J^{\mu}$, that is $T^{\mu'}J^{\mu} = TJ^{\mu}$ (it can be shown that $J^{\mu'}(x) \geq J^{\mu}(x)\ \forall x$ with the inequality becoming strict for at least one $x \in X$ if $\mu$ is not optimal), and repeating the process until the optimal stationary policy is reached. This algorithm is known to converge in a finite number of iterations to the optimal stationary policy and its tabular version is given in figure 2.5. The main weakness of this algorithm is that it requires the solving of the equations $(I - \gamma P^{\mu})J^{\mu} = r^{\mu}$. The dimension of this system is equal to the number of states and thus if the state space is large, this algorithm does not stay attractive anymore.

Figure 2.5: The Policy Iteration algorithm

```
Initialize arbitrarily a stationary policy $\mu(x_i)$ for all $x_i \in X$
Repeat
    Given the stationary policy $\mu$, compute $J^{\mu}$ by solving the linear system of equations $(I - \gamma P^{\mu})J^{\mu} = r^{\mu}$
    If $J^{\mu} = TJ^{\mu}$ then stop the algorithm
    Obtain a new stationary policy $\mu'$ satisfying $T^{\mu'}J^{\mu} = TJ^{\mu}$
    $\mu \leftarrow \mu'$
```

**Example 2.3.3** This example is aimed to illustrate the use of the policy iteration algorithm. This algorithm is used to compute the optimal stationary policy of the four-state control problem described in example 2.3.1.
We start with an initial policy $\mu^0$ such that $\mu^0(x) = 1\,\forall x \in X$. We compute the average rewards associated to this policy and the transition probability matrix and deduce from their knowledge the value of $J^{\mu^0}$ by using equation (2.43).

$$r^{\mu^0} = \begin{pmatrix} 0 \\ 1.25 \\ 3.75 \\ 0 \end{pmatrix}, \quad P^{\mu^0} = \begin{pmatrix} 0.25 & 0.5 & 0.25 & 0. \\ 0. & 0.25 & 0.5 & 0.25 \\ 0. & 0. & 0.25 & 0.75 \\ 0. & 0. & 0. & 1. \end{pmatrix}, \quad J^{\mu^0} = \begin{pmatrix} 1.37026 \\ 2.65306 \\ 4.28571 \\ 0 \end{pmatrix}.$$

The new policy $\mu^1$ obtained by solving the equation $T^{\mu^1} J^{\mu^0} = TJ^{\mu^0}$ is such that $\mu^1(1) = -1$, $\mu^1(2) = 1$ and $\mu^1(3) = 1$.
The computation of the average rewards and the transition probability matrix corresponding to $\mu^1$ allows the determination of $J^{\mu^1}$ :

$$r^{\mu^1} = \begin{pmatrix} 2.25 \\ 1.25 \\ 3.75 \\ 0 \end{pmatrix}, \quad P^{\mu^1} = \begin{pmatrix} 0.25 & 0. & 0. & 0.75 \\ 0. & 0.25 & 0.5 & 0.25 \\ 0. & 0. & 0.25 & 0.75 \\ 0. & 0. & 0. & 1. \end{pmatrix}, \quad J^{\mu^1} = \begin{pmatrix} 2.57143 \\ 2.65306 \\ 4.28571 \\ 0 \end{pmatrix}.$$

The computation of $\mu^2$ by solving the equation $T^{\mu^2} J^{\mu^1} = TJ^{\mu^1}$ leads to a policy that is equal to $\mu^1$ everywhere and the iterative process can stop. We have $\mu^1 = \mu^*$ and $J^{\mu^1} = V$. From the knowledge of the value function $V$ we can easily compute the $Q$-function by using equation (2.10). The $Q$-function is represented in table 2.2. Remark that $\mu^*(x)$ is indeed equal to $\underset{u \in U(x)}{\arg\max} Q(x, u)$.

Table 2.2: $Q$-function representation

| $x$ | $Q(x, -1)$ | $Q(x, 1)$ |
|---|---|---|
| 1 | 2.57143 | 1.52041 |
| 2 | 1.72449 | 2.65306 |
| 3 | 1.52041 | 4.28571 |

### 2.3.9 The modified policy iteration algorithms

When the number of states is really large it becomes time consuming to solve the linear system $(I - \gamma P^\mu) J^\mu = r^\mu$ at each stage of the policy iteration algorithm. One way to get around this difficulty is to solve the linear system iteratively by using for example the value iteration algorithm. The algorithm that uses this strategy is called the *modified policy iteration algorithm* [PS78, PS82]. In fact we may even consider solving the system $(I - \gamma P^\mu) J^\mu = r^\mu$ only approximately by using a limited number of value iterations. However, one can show that in this latter case the modified policy iteration algorithm loses the property to converge in a finite number of iterations because the system is not solved perfectly. If the number of value iterations is only one, then this type of algorithm behaves exactly like the value iteration algorithm.

### 2.3.10 Randomized stationary policy evaluation

The algorithm represented in figure 2.6 is used to compute the expected return of a randomized stationary policy. It consists of a Gauss-Seidel implementation of the algorithm inspired by the $T^{\mu^p}$ mapping (equation (2.36)).

Figure 2.6: Randomized stationary policy evaluation

```
Initialize arbitrarily $J(x_i)$ for all $x_i \in X$
Repeat
    $errorMax = 0$
    Repeat once for all $x_i \in X$
        $oldValue = J(x_i)$
        $J(x_i) \leftarrow \sum_{u \in U(x_i)} \mu^p(u|x)[r(x_i,u) + \gamma \sum_{x_j \in X} p(x_j|x_i,u)J(x_j)]$
        $errorMax = \max(errorMax, |J(x_i) - oldValue|)$
    If $errorMax < \epsilon$ then stop the algorithm
```

**Example 2.3.4** This example illustrates the computation of the expected return of a randomized stationary policy on the four-state control problem described in example 2.3.1. We have already computed the optimal stationary policy of this control problem by using two algorithms : the value iteration algorithm (example 2.3.2) and the policy iteration algorithm (example 2.3.3). The optimal stationary policy is : $\mu^*(1) = -1$, $\mu^*(2) = 1$ and $\mu^*(3) = 1$.
The randomized stationary policy we consider is known as the $\epsilon$-Greedy policy ($0 \leq \epsilon \leq 1$). This policy consists in selecting, while being in $x$, $u$ equal to $\mu^*(x)$ with a probability of $1 - \epsilon$. Otherwise the value of $u$ is chosen randomly with the same probability among the elements of $U(x)$.
Therefore we have $\epsilon - \text{Greedy}(\mu^*(x)|x) = 1 - \epsilon + \frac{\epsilon}{\#U(x)}$ and $\epsilon - \text{Greedy}(u|x) = \frac{\epsilon}{\#U(x)}$ if $u \neq \mu^*(x)$. If we use the algorithm described in figure 2.6 to compute the expected return of the $\epsilon$-Greedy policy for different values of $\epsilon$, we obtain the results gathered in table 2.3.

If $\epsilon = 0$ then $J^{\epsilon-\text{Greedy}} = V$. Observe that the larger the value of $\epsilon$ the smaller the expected return of the $\epsilon$-Greedy policy ($J^{\epsilon_1-\text{Greedy}} < J^{\epsilon_2-\text{Greedy}}$ if $\epsilon_1 > \epsilon_2$).

Table 2.3: $\epsilon$-Greedy policy

| $\epsilon$ | $J^{\epsilon-\text{Greedy}}(1)$ | $J^{\epsilon-\text{Greedy}}(2)$ | $J^{\epsilon-\text{Greedy}}(2)$ |
|---|---|---|---|
| 0. | 2.57143 | 2.65306 | 4.28571 |
| 0.1 | 2.50884 | 2.55571 | 4.12586 |
| 0.2 | 2.44117 | 2.46005 | 3.9623 |
| 1. | 1.71494 | 1.74712 | 2.51494 |

## 2.4 Infinite state spaces

Generally speaking if the state space is infinite, the optimal stationary policy cannot be computed exactly anymore.
However if the disturbance space and the control space remain finite, it is still possible to compute exactly the optimal stationary policy pointwise i.e., *for each* $x \in X$. To achieve this we proceed as follows :

- we determine which states can be reached from a state $x$ in a maximum number of $N$ stages and denote by $A$ this reachability set
- we determine all the stationary policies that can be defined on $A$
- we compute the expected return over $N$-stages

$$J_N(x) = \underset{\mathcal{H}_{N-1}}{E}[\sum_{t=0}^{N-1} \gamma^t r(x_t, \mu(x_t), w_t)|x_0 = x]$$

  for each of these policies (it requires the computation of $J_N(x)$ for a maximum of $\#C^{\#A}$ policies, the maximum being reached if $\#U(x) = \#C \, \forall x \in A$)
- we compute the maximum of $J_N(x)$ over all the possible policies and denote it by $J_N^*(x)$.

It can be shown that if all the policies for which the expected return over $N$-stages is higher or equal to $J_N^* - 2\frac{\gamma^N}{1-\gamma}B_r$ associate to $x$ the same control variable value, then they coincide for $x$ with $\mu^*$. Therefore by choosing $N$ large enough, it is possible to determine the optimal action to be associated with a state $x$ and to have an estimation of the maximum expected return by using the expression :

$$J_N^*(x) - \frac{\gamma^N}{1-\gamma}B_r < V(x) < J_N^*(x) + \frac{\gamma^N}{1-\gamma}B_r.$$

The main drawback of this method is that it can require huge amounts of computation and of memory, especially if the value of $\gamma$ is close to 1. Some smart implementations exist (see [CCPBS98]) but the computational burden involved remains high and most of the time out of reach of today's computer capabilities.

## 2.5 Summary

*In this chapter we have presented the main results of the dynamic programming theory that will be used in the subsequent chapters. We have focused on the particular case of infinite-time horizon control problems with bounded rewards and strict exponential decay. Among the results presented, there were some implementable algorithms that allow us to solve such control problems under the condition that their state spaces and control spaces were finite. Unfortunately, similar algorithms do not exist for infinite state space control problems. Strategies to solve at least partially these infinite state space control problems will be addressed in the next chapter.*

# Chapter 3

# Approximate solutions for DP problems

*In the preceding chapter various algorithms have been proposed to solve the dynamic programming problem in the context of finite state and control spaces. While the finiteness of the control space is not too restrictive in practice, these algorithms are not applicable to infinite state spaces, and moreover, even for finite state spaces, if the number of states is large they tend to become inefficient. In this chapter we propose a way to avoid these difficulties by constructing approximate solutions to control problems with infinite (or finite, but very large) state spaces. Essentially, this approach consists in three steps :*

1. *associate to the original control problem defined in an infinite state space a finite Markov decision process using a manageable number of states;*

2. *use the algorithms of the preceding chapter to define an optimal control policy for the reduced problem;*

3. *extend this control policy to the original problem.*

*We describe two different methods to define a finite Markov decision process based respectively on the idea of aggregate states and of representative states, and we provide also a way to evaluate the quality of the approximate policy obtained.*

*The resulting methods are illustrated by academic examples, and in chapter 4 they are evaluated in more detail in the context of a simple but realistic electric power system control problem.*

## 3.1 Introduction

This chapter is devoted to the resolution of discrete-time optimal control problems related to systems having an infinite (or too large) number of states. Rather than looking for exact solutions that are most of the time impossible to determine exactly, we present some techniques that are able to provide some good approximate solutions.
These techniques consist in constructing a finite MDP from the infinite state space control problem elements (notably the system dynamics and the reward function) and in exploiting the finite MDP solution in order to get an approximation of the infinite state space control problem solution. The procedure is summarized in figure 3.1. These techniques provide a good approximation of the control problem solution if the finite MDP is able to catch the control problem main features and if its solution is correctly extended.

Figure 3.1: Approximate solutions for dynamic programming

```
Construct a finite MDP$^\delta$ from the initial control problem.
Solve the MDP$^\delta$.
Extend the solution of the MDP$^\delta$ in order to get an approximate solution to the initial problem.
```

We present two techniques for solving approximately these infinite state space control problems. The first one is inspired from the aggregation methods that can be found in the DP literature [Ber75, HL86, Whi78, Whi79]. We will name it the *aggregation technique*. The second one is more tailor-made for problems with continuous state spaces. It consists roughly in selecting a finite number of “representative”states from the infinite state space and in associating to each of these states a state of the finite MDP. The transition probabilities and the rewards that define the MDP structure are computed both by using the rewards and system dynamics knowledge at these selected states and geometrical or distance considerations (see [SK00, Mun00] for related work). This technique will be referred to as the *representative states technique*.
In order to differentiate the symbols related to the MDP that approximates the control problem to the symbols related to this control problem itself, we use the suffix $\delta$. (For more information the reader may refer to the nomenclature at the beginning of the thesis.) For example, the term MDP$^\delta$ refers to the finite MDP that is used to

catch the main characteristics of the original control problem.

## 3.2 Aggregation technique

### 3.2.1 Approximate MDP

The procedure consists in lumping together states of the original system into a finite number $m$ of disjoint subsets $X_1, X_2, \cdots, X_m$ such that $X = X_1 \cup X_2 \cup \cdots \cup X_m$. Each of these subsets is aggregated into a single state that can be seen as an aggregate state. The $m$ states obtained from the aggregation process form the sought MDP$^\delta$. The MDP$^\delta$ is solved and the results obtained for each aggregate state are extended to all the states composing it. We denote the set of these aggregate states by $X^\delta$.
Let us denote by $x_1^\delta, x_2^\delta, \cdots, x_m^\delta$ ($X^\delta = \{x_1^\delta, x_2^\delta, \cdots, x_m^\delta\}$) the aggregate states that correspond to the subset $X_1, X_2, \cdots, X_m$, and for each $i$ let us denote by $p_i(x)$ a positive weight (or density[1]) function defined on $X_i$, such that $\int_{X_i} dp_i(x) = 1$.
Then the transition probabilities of the MDP$^\delta$ and their associated rewards are defined according to the equations[2] :

$$\begin{aligned}
r^\delta(x_i^\delta, u) &= \underset{(x,w)}{E}[r(x,u,w)|x \in X_i, p_i(x)] && (3.1)\\
&= \int_{X_i}\left[\int_D r(x,u,w)dP_w(w|x,u)\right]dp_i(x) && (3.2)\\
p^\delta(x_j^\delta|x_i^\delta, u) &= \underset{(x,w)}{E}[I_{X_j}(f(x,u,w))|x \in X_i, p_i(x)] && (3.3)\\
&= \int_{X_i}\left[\int_D I_{X_j}(f(x,u,w))dP_w(w|x,u)\right]dp_i(x) && (3.4)
\end{aligned}$$

which can be rewritten if the original state space $X$ is finite :

$$\begin{aligned}
r^\delta(x_j^\delta, u) &= \sum_{x \in X_i} p_i(x) r(x,u) && (3.5)\\
p^\delta(x_j^\delta|x_i^\delta, u) &= \sum_{x \in X_i} p_i(x) \sum_{x' \in X_j} p(x'|x,u) && (3.6)
\end{aligned}$$

[1] If the original state space is countable $p_i(x)$ would be a probability mass function, otherwise it is a probability density function; notice that we do not suppose that $p_i(x)$ denotes the true conditional probability $P(x|x \in X_i)$.

[2] Equation (3.3) can also be written as follows: $p^\delta(x_j^\delta|x_i^\delta, u) = P(f(x,u,w) \in X_j | x \in X_i, p_i(x))$.

where $p(x'|x,u)$ and $r(x,u)$ represent respectively the original control problem system dynamics and reward function expressed by using a MDP formulation (see section 2.3).
Equation (3.3) represents the probability for the next state to belong to $X_j$ after taking action $u$ assuming that the current state $x$ is randomly selected in $X_i$ according to the probability distribution $p_i(x)$. Similarly, equation (3.1) represents the average reward obtained after taking action $u$ while being in the set $X_i$ with a probability distribution on $x$ equal to $p_i(x)$. The terms $p_i(x)$ can also been interpreted as the weight a state $x \in X_i$ of the system has in the aggregation process i.e., how much it influences the characteristics of the aggregate state $x_i^\delta$.
Concerning the control variable $u$, we limit its value to a finite number of possibilities to make the computation of the whole MDP$^\delta$ structure feasible. We thus constraint $u$ to belong to a finite set $C^\delta \subset C$ with $U^\delta(x^\delta) \subset C^\delta$ and $U^\delta(x_i^\delta) \subset U(x)\, \forall x \in X_i$. $U^\delta(x)$ will be used to refer to the set $U^\delta(x_i^\delta)$ such that $x \in X_i$.
The set of all possible state-action pairs $\{(x^\delta, u)|x^\delta \in X^\delta \text{ and } u \in U^\delta(x^\delta)\}$ is denoted by $X^\delta \times U^\delta$.

### 3.2.2 Extension of the MDP solution to the original problem

Once the MDP$^\delta$ is built and solved, its solution (characterized by the $Q^\delta$-function) is extended to the original control problem.
The strategy chosen consists in extending trivially the $Q^\delta$ values of the aggregate states to all the states they are composed of. We denote by $\tilde{Q}$ the approximation of the $Q$-function that has been obtained. We thus have :

$$\tilde{Q}(x,u) \quad = \quad Q^\delta(x_i^\delta, u) \quad \text{if}\, x \in X_i \quad \forall x \in X, \forall\, u \in U^\delta(x). \tag{3.7}$$

The approximation of the optimal stationary policy is deduced from $\tilde{Q}$ :

$$\tilde{\mu}^*(x) \quad = \quad \underset{u \in U^\delta(x)}{\arg\max} \tilde{Q}(x,u) \quad \forall x \in X. \tag{3.8}$$

Notice that if $U^\delta(x) \neq U(x)$ then $\tilde{Q}(x,u)$ is not defined for some values of $u \in U(x)$. But since we deduce from $\tilde{Q}$ the approximation of the optimal stationary policy, this implies that the search for the optimal action to be taken while being in state $x$ is restricted to $U^\delta(x)$.
Obviously, the policy computed by using equation (3.8) depends on the way the state space has been partitioned, on the choice of the probability distributions $p_i(x)$ and on the control sets $U^\delta(x_i^\delta)$ chosen. The ideal choice for the $X_i$, $p_i$ and $U^\delta(x_i^\delta)$ would be the one which would lead to a good approximation of the optimal stationary policy without requiring a too vast amount of computation. We do not provide

any algorithm that could help to do this choice automatically but we stress that a good insight into the original control problem can already give some clues.

**Example 3.2.1** To illustrate the aggregation technique we use the four-state control problem described in example 2.3.1.
The state space of the system is equal to $\{1, 2, 3, x^t\}$. We lump together these states to obtain three disjoint subsets $X_1$, $X_2$, $X_3$ with $X_1 = \{1, 2\}$, $X_2 = \{3\}$ and $X_3 = \{x^t\}$. The value of the different $p_i(x)$ are chosen as follows : $p_1(1) = 0.5$, $p_1(2) = 0.5$, $p_2(3) = 1$ and $p_3(x^t) = 1$. Concerning the control sets, we take $C^\delta = C$ and $U^\delta(x_i^\delta) = U(x) \forall\, x \in X_i$.
The state space of the system being finite, we can use equations (3.5) and (3.6) to compute the MDP$^\delta$ structure. We detail hereafter the computation of $p^\delta(x_1^\delta|x_1^\delta, 1)$, $p^\delta(x_2^\delta|x_1^\delta, 1)$, $p^\delta(x_3^\delta|x_1^\delta, 1)$ and $r^\delta(x_1^\delta, 1)$.

$$\begin{aligned}
p^\delta(x_1^\delta|x_1^\delta, 1) &= p_1(1)(p(1|1,1) + p(2|1,1)) + p_1(2)(p(1|2,1) + p(2|2,1)) \\
&= 0.5 * (0.25 + 0.5) + 0.5 * (0 + 0.25) \\
&= 0.5 \\
p^\delta(x_2^\delta|x_1^\delta, 1) &= p_1(1)p(3|1,1) + p_1(2)p(3|2,1) \\
&= 0.5 * 0.25 + 0.5 * 0.5 \\
&= 0.375 \\
p^\delta(x_3^\delta|x_1^\delta, 1) &= p_1(1)p(x^t|1,1) + p_1(2)p(x^t|2,1) \\
&= 0.5 * 0 + 0.5 * 0.25 \\
&= 0.125 \\
r^\delta(x_1^\delta, 1) &= p_1(1)r(1,1) + p_1(2)r(2,1) \\
&= 0.5 * 0 + 0.5 * 1.25 \\
&= 0.625
\end{aligned}$$

Table 3.1: Solution of the MDP$^\delta$

| $x^\delta$ | $Q^\delta(x^\delta, -1)$ | $Q^\delta(x^\delta, 1)$ |
|---|---|---|
| $x_1^\delta$ | 2 | 1.92857 |
| $x_2^\delta$ | 1.28571 | 4.28571 |
| $x_3^\delta$ | / | / |

Once the MDP$^\delta$ structure is determined, its $Q^\delta$-function can be computed. The $Q^\delta$-function is given in table 3.1.

We can use expression (3.7) to compute $\tilde{Q}$ and expression (3.8) to get the approximation of the optimal stationary policy. We obtain $\tilde{\mu}^*(1) = -1$, $\tilde{\mu}^*(2) = -1$ and $\tilde{\mu}^*(3) = 1$. This policy is different from the optimal stationary one. Indeed it has been shown previously that $\mu^*(1) = -1, \mu^*(2) = 1$ and $\mu^*(3) = 1$.

### 3.2.3 Equivalence between $Q$ and $\tilde{Q}$

We have seen how to construct a MDP$^\delta$ from the original control problem (equations (3.1) and (3.3)) and how to extend the solution of the MDP$^\delta$ to get an approximation of the $Q$-function (equation (3.7)), and from it an approximation of the optimal stationary policy (equation (3.8)).
We describe hereafter some sufficient conditions on the original control problem and on the MDP$^\delta$ structure in order to have an equivalence between the approximate solution and the real solution, that is in order to make $\tilde{Q}$ coincide with $Q$. These conditions of equivalence between $Q$ and $\tilde{Q}$ are used in chapter 6 to justify the algorithms we introduce to reconstruct the MDP$^\delta$ structure from interaction with the system.

#### Sufficient equivalence conditions

The equivalence conditions are expressed by the following three equations :

$$r^\delta(x_i^\delta, u) = \underset{w}{E}[r(x, u, w)], \quad \forall i \in \{1, \cdots, m\}, \forall u \in U^\delta(x_i^\delta),\ \forall x \in X_i \tag{3.9}$$

$$p^\delta(x_j^\delta | x_i^\delta, u) = \underset{w}{E}[I_{X_j}(f(x, u, w))],\ \forall i, j \in \{1, \cdots, m\}, \forall u \in U^\delta(x_i^\delta),\ \forall x \in X_i \tag{3.10}$$

$$U(x) = U^\delta(x_i^\delta), \quad \forall i \in \{1, \cdots, m\}, \forall x \in X_i. \tag{3.11}$$

- Condition (3.9) actually means that the average reward obtained while taking an action $u$ in $x$ has to be identical for all $x$ belonging to a same subset.
- Condition (3.10) actually means that for all $x$ belonging to a same subset, the probability to arrive in a subset $X_j$ while taking action $u$ has to be the same.
- Condition (3.11), on the other hand, implies that the whole original control set has to be used to reconstruct the MDP$^\delta$ structure ($\tilde{Q}$ will therefore be defined $\forall x \in X$ and $\forall u \in U(x)$).

**Proof**

We prove hereafter that if conditions (3.9), (3.10) and (3.11) are satisfied then $\tilde{Q}$ coincides with $Q$ everywhere.
The DP equation for the MDP$^\delta$ can be written as follows :

$$\begin{array}{rcl} Q^\delta(x_i^\delta, u) & = & r^\delta(x_i^\delta, u) + \gamma \sum_{j=1}^{m} p^\delta(x_j^\delta | x_i^\delta, u) \max_{u' \in U^\delta(x_j^\delta)} Q^\delta(x_j^\delta, u') \\ \text{which holds} & & \forall i \in \{1, \cdots, m\}, \, \forall u \in U^\delta(x_i^\delta). \end{array}$$

By using conditions (3.9) and (3.10) this latter expression implies that :

$$\begin{array}{rcl} Q^\delta(x_i^\delta, u) & = & \underset{w}{E}[r(x,u,w)] + \gamma \sum_{j=1}^{m} \underset{w}{E}[I_{X_j}(f(x,u,w)) \max_{u' \in U^\delta(x_j^\delta)} Q^\delta(x_j^\delta, u')] \\ \text{which holds} & & \forall i \in \{1, \cdots, m\}, \, \forall x \in X_i, \, \forall u \in U^\delta(x_i^\delta). \end{array}$$

By using condition (3.11) we can state that :

$$\begin{array}{rcl} Q^\delta(x_i^\delta, u) & = & \underset{w}{E}[r(x,u,w)] + \gamma \sum_{j=1}^{m} \underset{w}{E}[I_{X_j}(f(x,u,w)) \max_{u' \in U(f(x,u,w))} Q^\delta(x_j^\delta, u')] \\ \text{which holds} & & \forall i \in \{1, \cdots, m\}, \, \forall x \in X_i, \, \forall u \in U(x). \end{array}$$

Using equation (3.7) we can write :

$$\begin{array}{rcl} \tilde{Q}(x, u) & = & \underset{w}{E}[r(x,u,w) + \gamma \max_{u' \in U(f(x,u,w))} \tilde{Q}(f(x,u,w), u')] \\ \text{which holds} & & \forall x \in X, \, \forall u \in U(x). \end{array}$$

$\tilde{Q}$ satisfies the DP equation of the original control problem everywhere. The solution of this equation being unique, $\tilde{Q}$ coincides with $Q$ when conditions (3.9), (3.10) and (3.11) are satisfied. **QED**

### 3.2.4 Equivalence conditions used to define the MDP$^\delta$

Equivalence conditions (3.9) and (3.10) suggest a way to build the MDP$^\delta$ structure different from the one defined by equations (3.1) and (3.3). Indeed, one may reasonably suppose that the “less” these equivalence conditions are violated, the better the approximation of the $Q$-function will be.

Therefore one could for example compute $r^\delta(x_i^\delta, u)$ and $p^\delta(x_j^\delta|x_i^\delta, u)$ as follows :

$$r^\delta(x_i^\delta, u) = \underset{a \in \mathbb{R}}{\arg\min} \sum_{x \in \mathcal{F}_i} (a - \underset{w}{E}[r(x, u, w)])^2 \tag{3.12}$$

$$p^\delta(x_j^\delta|x_i^\delta, u) = \underset{a \in \mathbb{R}}{\arg\min} \sum_{x \in \mathcal{F}_i} (a - \underset{w}{E}[I_{X_j}(f(x, u, w))])^2 \tag{3.13}$$

with $\mathcal{F}_i$ being a set of states selected from $X_i$.
By noting that :

$$\underset{a \in \mathbb{R}}{\arg\min} \underset{x}{E}\left[(a - \underset{w}{E}[I_{X_j}(f(x, u, w))])^2 | x \in X_i, p_i(x)\right] = \underset{(x,w)}{E}\left[I_{X_j}(f(x, u, w)) | x \in X_i, p_i(x)\right]$$

and

$$\underset{a \in \mathbb{R}}{\arg\min} \underset{x}{E}\left[(a - \underset{w}{E}[r(x, u, w)])^2 | x \in X_i, p_i(x)\right] = \underset{(x,w)}{E}\left[r(x, u, w) | x \in X_i, p_i(x)\right],$$

equations (3.1) and (3.3) that we have used to define the MDP$^\delta$ structure can be seen as a particular case of equations (3.12) and (3.13) where the summation is replaced by the expectation with respect to $p_i(x)$.
Equations similar to equations (3.12) and (3.13) will be used in chapter 6 to build the MDP$^\delta$ structure from interaction with the system.

### 3.2.5 Difficulties to compute the MDP$^\delta$ structure

Computation of the MDP$^\delta$ structure by using expressions (3.1) and (3.3) cannot be achieved most of the time if the state space is infinite due to the impossibility of computing exactly the right hand side of these expressions. One way to circumvent these difficulties is to estimate numerically these terms. The main idea of such numerical schemes is to modify both the $p_i(x)$ and the $P_w(w|x, u)$ such that they respectively become different from 0 for only a finite number of $x \in X_i$ and a finite number of $w \in D$.
One way to guarantee the numerical scheme convergence i.e., that it converges to the MDP$^\delta$ structure defined by equations (3.1) and (3.3) when the amount of computation increases, is to choose the $x$ and the $w$ that intervene in the building process according to the $p_i(x)$ and $P_w(w|x, u)$ probability distributions themselves.

The tabular version of a convergent numerical scheme is given in figure 3.2. If $N_i \to \infty$ and $N_{w_i} \to \infty$ $\forall i \in \{1, \cdots, m\}$ then this numerical scheme converges to the exact solution. The choice of $w$ according to $P_w(w|x, u)$ in order to compute $f(x, u, w)$ and $r(x, u, w)$ is strictly equivalent to simulate the system for one time step starting from $x$ with the control variable value $u$ and to observe the next state and the reward obtained.

Figure 3.2: Aggregation technique : numerical estimation of the MDP$^\delta$ structure

```
Initialize $N^\delta(x^{\delta'}, x^\delta, u)$ and $r^\delta(x^\delta, u)$ to 0 $\forall x^\delta, x^{\delta'} \in X^\delta$ and $\forall u \in U^\delta(x^\delta)$
Repeat $N_i$ times for each $i \in \{1, \cdots, m\}$
    Choose a state $x \in X_i$ according to the probability distribution $p_i(x)$
    Repeat $N_{w_i}$ times for each $u \in U^\delta(x_i^\delta)$
        Choose $w$ according to $P_w(w|x, u)$ and compute $x' = f(x, u, w)$ and $r = r(x, u, w)$.
        Determine to which $x^{\delta'}$ the state $x'$ corresponds
        $N^\delta(x^{\delta'}, x_i^\delta, u) \leftarrow N^\delta(x^{\delta'}, x_i^\delta, u) + 1$
        $r^\delta(x_i^\delta, u) \leftarrow r^\delta(x_i^\delta, u) + \frac{r - r^\delta(x_i^\delta, u)}{\sum_{x^\delta \in X^\delta} N^\delta(x^\delta, x_i^\delta, u)}$
$p^\delta(x_j^\delta | x_i^\delta, u) \leftarrow \frac{N^\delta(x_j^\delta, x_i^\delta, u)}{\sum_{x^\delta \in X^\delta} N^\delta(x^\delta, x_i^\delta, u)}$ $\forall i, j \in \{1, \cdots, m\}$ and $\forall u \in U^\delta(x_i^\delta)$
```

### 3.2.6 Continuous state spaces

In this work, the aggregation technique is used for systems with continuous state spaces. The process of dividing a continuous state space into a finite number of disjoint subsets is known as *discretization of the state space*.

On each *discretized region* of the state space the approximate $Q$-function is then considered to be constant as well as the approximation of the optimal stationary policy computed from it.

Figure 3.3 represents the discretization of a three-dimensional continuous state space. One can reasonably suppose that the smaller the discretized regions are, the better the optimal control problem solution is approximated.

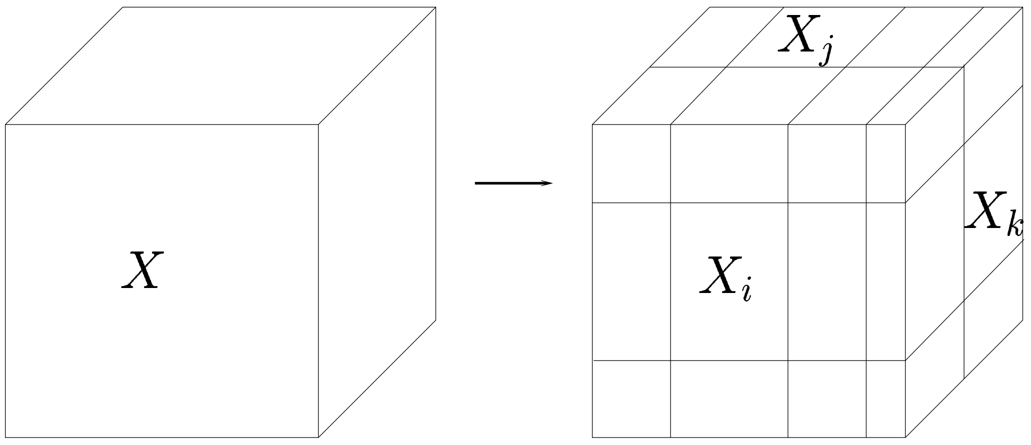


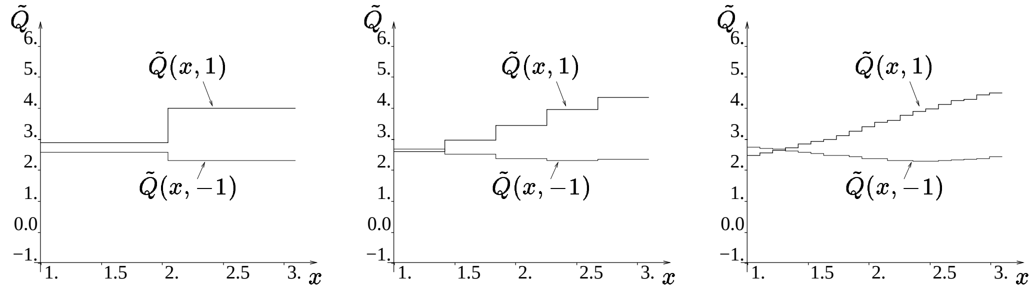
Figure 3.3: Discretization of the state space (three-dimensional representation)

**Example 3.2.2** In order to illustrate the aggregation technique on a system having a continuous state space we consider the control problem described at the end of example 2.2.1. Its characteristics are recalled hereafter.
The system dynamics is described by the discrete-time equation :

$$x_{t+1} = x_t + u_t + w_t.$$

The state space $X = [1,3] \cup x^t$ (the terminal state $x^t$ is reached if $x$ goes outside the interval $[1,3]$), the control space $C = [-1,1]$, $U(x) = C \, \forall x \in X \setminus \{x^t\}$ and the noise factor $w_t$ is drawn according to the standard Gaussian distribution.
The reward $r(x,u,w)$ is equal to $0$ if $x+u+w \in [1,3]$, $3$ if $x+u+w < 1$ and $5$ if $x+u+w > 3$. The decay factor $\gamma = 0.5$.
We study the results obtained by using the aggregation technique for three different state space discretizations.
The first discretization consists in partitioning the state space into 3 disjoint subsets ($X_1 = [1,2[$, $X_2 = [2,3]$ and $X_3 = \{x^t\}$), the second into 6 disjoint subsets ($X_i = [1+0.4*(i-1), 1.4+0.4*(i-1)[$ with $i \in \{1,2,3,4\}$, $X_5 = [2.6,3]$, $X_6 = \{x^t\}$) and the last one into 21 disjoint subsets $X_i = [1+0.1*(i-1), 1.1+0.1*(i-1)[$ with $i \in \{1,2,\cdots,19\}$, $X_{20} = [2.9,3]$, $X_{21} = \{x^t\}$). The $p_i(x)$ are such that the probability to draw each $x$ of $X_i$ is the same.
The control space being infinite, we consider the finite subset $C^\delta = \{-1,1\} \subset C$ to construct the MDP$^\delta$ and choose $U^\delta(x^\delta) = C^\delta$ for all $x^\delta \in X^\delta \setminus \{x^\delta_{\#X^\delta}\}$ where $x^\delta_{\#X^\delta}$ denotes the state of $X^\delta$ that corresponds to $x^t$.
To estimate the MDP$^\delta$ structure, we use the algorithm described in figure 3.2 with $N_i = 1000$ and $N_{w_i} = 1000$. These values are large enough to ensure that the MDP$^\delta$ structure computed does not differ much from the one defined by equations (3.1) and (3.3).

Once the MDP$^\delta$ is solved i.e., once the $Q^\delta(x^\delta, u)$ values are computed, the solution obtained is extended to the original control problem by using equation (3.7). Each of the approximate $Q$-functions is represented in figure 3.4.
From the approximate $Q$-function, we can deduce the approximate optimal stationary policy by using equation (3.8). For example we can see in figure 3.4a that the policy obtained when $\#X^\delta = 3$ consists in taking $u = 1$ everywhere while the policy obtained when $\#X^\delta = 6$ consists in taking $u = 1$ only on the interval $[1.4, 3]$. If $\#X^\delta = 21$ the policy is still different and consists in choosing $u = -1$ on the interval $[1, 1.3[$ and $u = 1$ on $[1.3, 3]$. It is important to be able to estimate the quality of the policy i.e., how close the approximate policy is to the optimal one, in order to determine which state space discretization has led to the best policy. This topic is addressed in section 3.4.



(a) $\#X^\delta = 3$ (b) $\#X^\delta = 6$ (c) $\#X^\delta = 21$

Figure 3.4: Approximation of the $Q$-function for different state space discretizations

## 3.3 Representative states technique

### 3.3.1 Approximate MDP

The representative states technique consists in selecting a finite number of states from $X$ and in defining between these states a MDP$^\delta$ structure. The MDP$^\delta$ structure definition is based on the knowledge of the system dynamics, the reward function and a notion of "distance" between the system states.
The $m$ states of $X^\delta = \{x_1^\delta, x_2^\delta, \cdots, x_m^\delta\}$ can be seen both as states of the MDP$^\delta$ and states of $X$. To compute the MDP$^\delta$ structure we introduce the function $W : X \times X^\delta \to [0, 1]$ such that $\sum_{x^\delta \in X^\delta} W(x, x^\delta) = 1$, $\forall x \in X$ and $W(x^\delta, x^\delta) = 1\, \forall x^\delta \in X^\delta$. The value of $W(x, x^\delta)$ can be seen as the probability for $x$ to belong to $x^\delta$ or "how close" $x$ is from $x^\delta$. Some of these concepts are illustrated on figure 3.5.

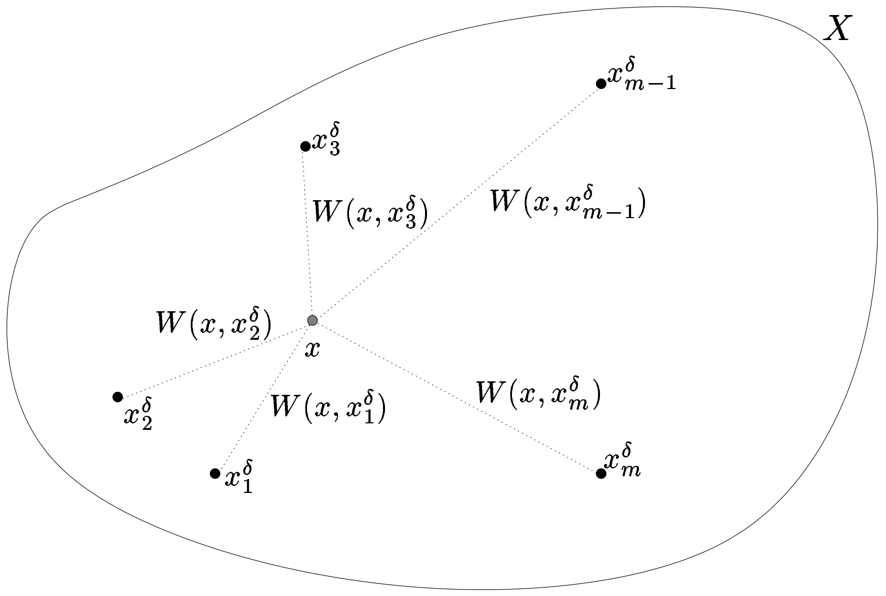


Figure 3.5: State space represented by a finite number of states

In the representative states technique, the transition probabilities and the rewards of the MDP$^\delta$ are defined by the following relations :

$$r^\delta(x^\delta_i, u) = \underset{w}{E}[r(x^\delta_i, u, w)] \tag{3.14}$$

$$p^\delta(x^\delta_j | x^\delta_i, u) = \underset{w}{E}[W(f(x^\delta_i, u, w), x^\delta_j)] \tag{3.15}$$

Concerning the possible values for the control variable $u$ used in the MDP$^\delta$ structure computation, we limit them again to a finite set $C^\delta \subset C$. The condition imposed on $U^\delta(x^\delta)$ is to be a subset of $C^\delta$ and to be such that $U^\delta(x^\delta) \subset U(x^\delta)$.[3]

We then denote by $U^\delta(x)$ the set that is composed of the intersection of the sets $U^\delta(x^\delta)$ where $x^\delta \in X^\delta$ is such that $W(x, x^\delta) \neq 0$.

The set of all possible state-action pairs $\{(x^\delta, u) | x^\delta \in X^\delta \text{ and } u \in U^\delta(x^\delta)\}$ is denoted by $X^\delta \times U^\delta$.

[3]The terms $r^\delta(x^\delta, u)$, $p^\delta(x^{\delta\prime} | x^\delta, u)$ and $Q^\delta(x^\delta, u)$ are sometimes written for some $u \notin U^\delta(x^\delta)$ even if they are not defined for these values of the control variable. This has been done in order to ease the notations. When confronted with such terms one should simply consider them as equal to zero.

### 3.3.2 Extension of the MDP solution to the original problem

Once the finite MDP$^\delta$ structure is computed, one can solve it in order to compute its $Q^\delta$-function. From its knowledge we define the approximate $Q$-function as follows :

$$\tilde{Q}(x,u) = \sum_{x^\delta \in X^\delta} W(x,x^\delta) Q^\delta(x^\delta,u) \quad \forall x \in X, \forall u \in U^\delta(x) \tag{3.16}$$

The approximate optimal stationary policy is deduced from $\tilde{Q}$ by :

$$\tilde{\mu}^*(x) = \arg\max_{u \in U^\delta(x)} \tilde{Q}(x,u) \quad \forall x \in X. \tag{3.17}$$

If $U^\delta(x) \neq U(x)$ then $\tilde{Q}(x,u)$ is not defined for some values of $u \in U(x)$. Since the approximate optimal stationary policy is deduced from $\tilde{Q}$, the search for the optimal action to be taken while being in state $x$ is limited to $U^\delta(x)$.
Obviously, the resulting approximate solution depends on the way the function $W$ is chosen, on the states of $X$ selected to represent $X^\delta$ and on the $U^\delta(x^\delta)$ subsets.

### 3.3.3 Equivalence between $Q$ and $\tilde{Q}$

We have seen how to construct a MDP$^\delta$ from the original control problem knowledge (equations (3.14) and (3.15)) and how to extend the solution of the MDP$^\delta$ to get an approximation of the $Q$-function (equation (3.16)), and from it an approximation of the optimal stationary policy (equation (3.17)).
We describe hereafter some sufficient conditions on the original control problem and on the MDP$^\delta$ structure so as to have an equivalence between the approximate solution and the real solution, that is in order to make $\tilde{Q}$ coincide with $Q$. These equivalence conditions are used in chapter 6 to justify the algorithms we introduce to reconstruct the MDP$^\delta$ structure from interaction with the system.

#### Sufficient equivalence conditions

The equivalence conditions are expressed by the following four equations :

$$\sum_{x^\delta \in X^\delta} W(x,x^\delta) r^\delta(x^\delta,u) = \underset{w}{E}[r(x,u,w)], \qquad \forall x \in X, \forall u \in U^\delta(x) \tag{3.18}$$

$$\sum_{x^\delta \in X^\delta} W(x,x^\delta) p^\delta(x^{\delta\prime}|x^\delta,u) = \underset{w}{E}[W(f(x,u,w),x^{\delta\prime})], \qquad \forall x^{\delta\prime} \in X^\delta, \tag{3.19}$$

$$\forall x \in X, \forall u \in U^\delta(x)$$

$$U(x) = U^\delta(x), \qquad \forall x \in X \tag{3.20}$$

$$\max_{u \in U^\delta(x)} \sum_{x^\delta \in X^\delta} W(x, x^\delta) Q^\delta(x^\delta, u) = \sum_{x^\delta \in X^\delta} W(x, x^\delta) \max_{u \in U^\delta(x)} Q^\delta(x^\delta, u), \ \forall x \in X \tag{3.21}$$

**Proof**

We prove hereafter that if conditions (3.18), (3.19), (3.20) and (3.21) are satisfied then $\tilde{Q}$ coincides with $Q$ everywhere.
The DP equation for the MDP$^\delta$ can be written as follows :

$$\begin{array}{rcl} Q^\delta(x^\delta, u) & = & r^\delta(x^\delta, u) + \gamma \sum_{x^{\delta\prime} \in X^\delta} p^\delta(x^{\delta\prime} | x^\delta, u) \max_{u' \in U^\delta(x^{\delta\prime})} Q^\delta(x^{\delta\prime}, u') \\ \text{which holds} & & \forall x^\delta \in X^\delta, \, \forall u \in U^\delta(x^\delta) \end{array}$$

We can state that :

$$\begin{array}{rcl} \sum_{x^\delta \in X^\delta} W(x, x^\delta) Q^\delta(x^\delta, u) & = & \sum_{x^\delta \in X^\delta} W(x, x^\delta) [r^\delta(x^\delta, u) + \\ & & \gamma \sum_{x^{\delta\prime} \in X^\delta} p^\delta(x^{\delta\prime} | x^\delta, u) \max_{u' \in U^\delta(x^{\delta\prime})} Q^\delta(x^{\delta\prime}, u')] \\ \text{which holds} & & \forall x \in X, \, \forall u \in U^\delta(x) \end{array}$$

By using conditions (3.18) and (3.19) we can write :

$$\begin{array}{rcl} \sum_{x^\delta \in X^\delta} W(x, x^\delta) Q^\delta(x^\delta, u) & = & \underset{w}{E}[r(x, u, w)] + \\ & & \gamma \sum_{x^{\delta\prime} \in X^\delta} \underset{w}{E}[W(f(x, u, w), x^{\delta\prime}) \max_{u' \in U^\delta(x^{\delta\prime})} Q^\delta(x^{\delta\prime}, u')] \\ \text{which holds} & & \forall x \in X, \, \forall u \in U^\delta(x) \end{array}$$

Conditions (3.20) and (3.21) and equation (3.16) imply that :

$$\begin{array}{rcl} \tilde{Q}(x, u) & = & \underset{w}{E}[r(x, u, w) + \gamma \max_{u' \in U(f(x, u, w))} \tilde{Q}(f(x, u, w), u')] \\ \text{which holds} & & \forall x \in X, \, \forall u \in U(x) \end{array}$$

$\tilde{Q}$ satisfies the DP equation of the original control problem everywhere. The solution of this equation being unique, $\tilde{Q}$ coincides with $Q$ when conditions (3.18), (3.19), (3.20) and (3.21) are satisfied.

### 3.3.4 Equivalence conditions used to define the MDP$^\delta$

Equivalence conditions (3.18) and (3.19) suggest a way to build the MDP$^\delta$ structure different from the one defined by equations (3.14) and (3.15). Indeed, one may reasonably suppose that the "less" these equivalence conditions are violated, the better the approximation of the $Q$-function will be.

Therefore one could for example select from $X$ a finite set of states $\mathcal{F}$ and determine the $r^\delta(x_i^\delta, u)$ terms ($\forall i \in \{1, \cdots, m\}$) by computing the vector $a \in \mathbb{R}^m$ that minimizes :

$$\sum_{x \in \mathcal{F}} \Big(\sum_{i=1}^{m} W(x, x_i^\delta) a(i) - \underset{w}{E}[r(x, u, w)]\Big)^2 \tag{3.22}$$

and by equating $r^\delta(x_i^\delta, u)$ with $a(i)$.

Similarly, one could define the $p^\delta(x_j^\delta | x_i^\delta, u)$ terms ($\forall i \in \{1, \cdots, m\}$) by computing the vector $a \in \mathbb{R}^m$ that minimizes :

$$\sum_{x \in \mathcal{F}} \Big(\sum_{i=1}^{m} W(x, x_i^\delta) a(i) - \underset{w}{E}[W(f(x, u, w), x_j^\delta)]\Big)^2 \tag{3.23}$$

and by equating $p^\delta(x_j^\delta | x_i^\delta, u)$ with $a(i)$.

Notice that the determination of all the terms $p^\delta(.|., u)$ requires minimizing $m$ expressions of the type (3.23). One can show that if the solution of each minimization problem is unique, then we can guarantee that $\sum_{j=1}^{m} p^\delta(x_j^\delta | x_i^\delta, u) = 1$ (section A.3.1). However, one cannot guarantee that $|p^\delta(x_j^\delta | x_i^\delta, u)| \leq 1$ by using such a method. In practice, one can solve this problem by truncating and renormalizing these values.

If $\mathcal{F}$ is equal to $X^\delta$, then it is straightforward to see that the computation of the MDP$^\delta$ structure by minimizing expressions (3.22) and (3.23) leads to the same MDP$^\delta$ structure as the one computed by using expressions (3.14) and (3.15).

Equations similar to equations (3.22) and (3.23) will be used in chapter 6 to build the MDP$^\delta$ structure from interaction with the system.

### 3.3.5 Difficulties to compute the MDP$^\delta$ structure

Computation of the MDP$^\delta$ structure implies the ability to evaluate the right hand sides of equations (3.14) and (3.15). This evaluation cannot be achieved most of the time if the disturbance space $D$ is infinite. To circumvent this difficulty we can estimate these right hand sides numerically. The main idea of such numerical schemes is to modify the $P_w(w|x,u)$ probability distributions such that they become different from zero for only a finite number of $w \in D$.

When the amount of computation increases, one way to guarantee the numerical scheme convergence i.e., it converges to the exact computation of the MDP$^\delta$ structure (as defined by equations (3.14) and (3.15)), is to choose the value of $w$ that intervenes in the building process according to the $P_w(w|x,u)$ probability distributions themselves. The tabular version of a convergent numerical scheme is given in figure 3.6. If the system is deterministic then the value of $N_{w_i}$ can be chosen equal to 1. If the system is stochastic, then $N_{w_i}$ must become infinitely large in order to ensure that the numerical scheme converges to the MDP$^\delta$ structure.

Figure 3.6: Representative states technique : numerical estimation of the MDP$^\delta$ structure

```
Initialize $N^\delta(x^{\delta'}, x^\delta, u)$ and $r^\delta(x^\delta, u)$ to 0, $\forall x^\delta, x^{\delta'} \in X^\delta$ and $\forall u \in U^\delta(x^\delta)$
Do for each $i \in \{1, \cdots, m\}$
    Repeat $N_{w_i}$ times for each $u \in U^\delta(x_i^\delta)$
        Choose $w$ according to $P_w(w|x_i^\delta, u)$ and compute $x' = f(x_i^\delta, u, w)$ and $r = r(x_i^\delta, u, w)$.
        Do for all $x^{\delta'} \in X^\delta$
            $N^\delta(x^{\delta'}, x_i^\delta, u) \leftarrow N^\delta(x^{\delta'}, x_i^\delta, u) + W(x', x^{\delta'})$
        $r^\delta(x_i^\delta, u) \leftarrow r^\delta(x_i^\delta, u) + \frac{r - r^\delta(x_i^\delta, u)}{\sum_{x^\delta \in X^\delta} N^\delta(x^\delta, x_i^\delta, u)}$
$p^\delta(x_j^\delta|x_i^\delta, u) \leftarrow \frac{N^\delta(x_j^\delta, x_i^\delta, u)}{\sum_{x^\delta \in X^\delta} N^\delta(x^\delta, x_i^\delta, u)}$ $\forall i, j \in \{1, \cdots, m\}$ and $\forall u \in U^\delta(x_i^\delta)$
```

### 3.3.6 Triangulation technique

The triangulation technique may be used to define the function $W$ over continuous state spaces. We suppose here that $X$ is a bounded subset of $\mathbb{R}^n$.

The triangulation technique consists in partitioning the state space $X \subset \mathbb{R}^n$ into simplices and in considering that the set $X^\delta$ is composed by the vertices of these simplices. The function $W(x, x^\delta)$ is computed by assuming that it is zero for all elements $x^\delta \in X^\delta$ which do not correspond to a vertex of the $n$-simplex to which $x$ belongs. It implies that at most $n+1$ states $x^\delta \in X^\delta$ are such that $W(x, x^\delta) \neq 0$ .

If the set $\{v_1, v_2, \cdots, v_{n+1}\} \subset X^\delta$ contains the vertices of an $n$-simplex to which $x$ belongs, then $W(x, v_1), W(x, v_2), \cdots, W(x, v_{n+1})$ are computed by solving the linear system[4] :

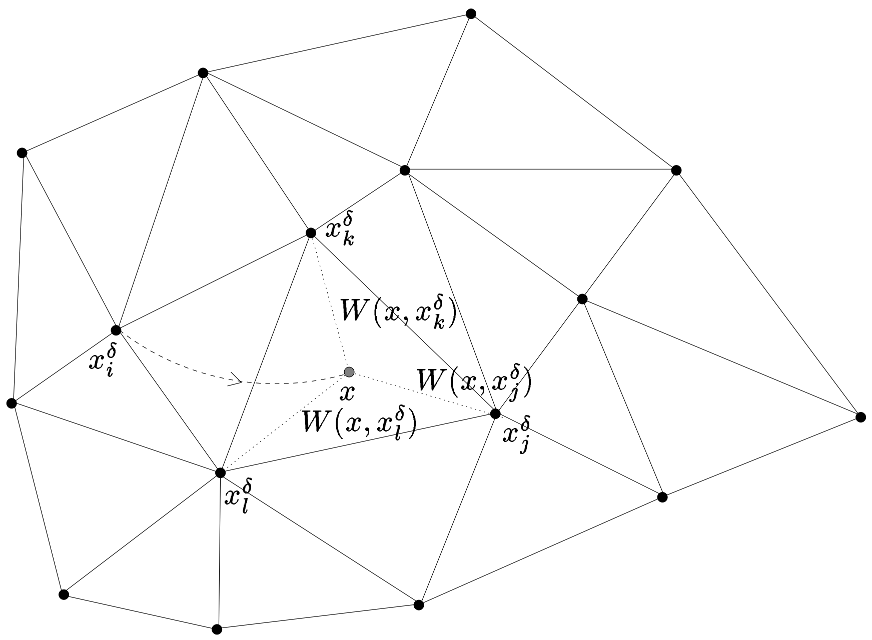


Figure 3.7: Triangulation of the state space. Two-dimensional representation.

[4] $W(x, v_1), W(x, v_2), \cdots, W(x, v_{n+1})$ represent the barycentric coordinates of $x$ in the simplex.

$$\begin{cases} \sum_{i=1}^{n+1}(x(1) - v_i(1))W(x, v_i) = 0 \\ \sum_{i=1}^{n+1}(x(2) - v_i(2))W(x, v_i) = 0 \\ \vdots \\ \sum_{i=1}^{n+1}(x(n) - v_i(n))W(x, v_i) = 0 \\ \sum_{i=1}^{n+1} W(x, v_i) = 1 \end{cases} \tag{3.24}$$

where $v(k)$ and $x(k)$ represent respectively the value of $v$ and $x$ according to the $k$th dimension.[5]
Remark that the triangulation technique is invariant with respect to a linear transformation of the state space and therefore frees itself from the tuning of parameters linked to the scaling of the variables that define a state.
In most cases we will apply the triangulation technique in the following way :

1. we first partition the state space into a finite number of $n$-rectangles by using a *regular* grid;

2. we then partition each $n$-rectangle into simplices such that the vertices of the simplices coincide with the vertices of the $n$-rectangle. The partitioning of an $n$-rectangle into simplices is discussed in appendix B.

**Example 3.3.1** This example illustrates the use of the triangulation technique. The illustration is carried out on the control problem described in example 2.2.1.
We select in $X$ a subset of states that are used to determine $X^\delta$. The set chosen is $\{1, 1.875, 3, x^t\}$. To this set corresponds the set $X^\delta = \{x_1^\delta, x_2^\delta, x_3^\delta, x_4^\delta\}$. $C^\delta$ is chosen equal to $C$ ($\{-1, 1\}$) and $U^\delta(x^\delta)$ equal to $C^\delta$ $\forall x^\delta \in X^\delta \setminus \{x_4^\delta\}$. The computation of the MDP$^\delta$ structure is done by using equations (3.14) and (3.15). By way of example we detail the computation of $p^\delta(x_3^\delta|x_2^\delta, 1)$. By using equation (3.15) we can write :

$$\begin{aligned} p^\delta(x_3^\delta|x_2^\delta, 1) &= E[W(f(x_2^\delta, 1), x_3^\delta)] \\ &= W(1.875 + 1, x_3^\delta) \\ &= W(2.875, x_3^\delta) \end{aligned}$$

$W(2.875, x_3^\delta)$ is determined by using the triangulation technique. The set of the vertices of the 1-simplex that encloses $2.875$ is composed of two elements $x_2^\delta$ and $x_3^\delta$. Using equation (3.24), we can write :

$$\begin{cases} (2.875 - 1.875)W(2.875, x_2^\delta) + (2.875 - 3)W(2.875, x_3^\delta) = 0 \\ W(2.875, x_2^\delta) + W(2.875, x_3^\delta) = 1 \end{cases}$$

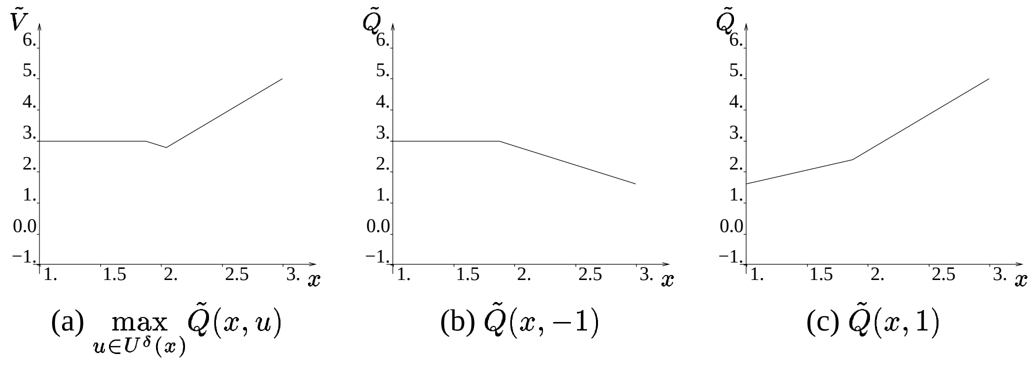
[5]If $x$ belongs to the intersection of several simplices this definition will provide the same result whatever the one chosen among these latter.

By solving this system we obtain $W(2.875, x_2^\delta) = \frac{1}{9}$ and $W(2.875, x_3^\delta) = \frac{8}{9}$. Therefore we have $p(x_3^\delta|x_2^\delta, 1) = \frac{8}{9}$.
The transition probabilities $p^\delta(.|x_2^\delta, .)$ are given in table 3.2.

Table 3.2: Transition probabilities computed by using the triangulation technique

| $x$ | $p^\delta(x^\delta\|x_2^\delta, 1)$ | $p^\delta(x^\delta\|x_2^\delta, -1)$ |
|---|---|---|
| $x_1^\delta$ | 0 | 0 |
| $x_2^\delta$ | $\frac{1}{9}$ | 0 |
| $x_3^\delta$ | $\frac{8}{9}$ | 0 |
| $x_4^\delta$ | 0 | 1 |

Once the MDP$^\delta$ structure is computed and solved, we can compute the value of the approximate $Q$-function. The results are represented on figure 3.8. By using equation (3.17)



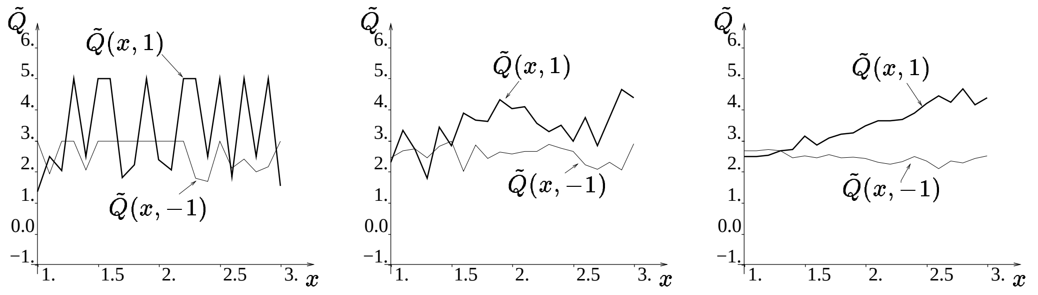
(a) $\max_{u \in U^\delta(x)} \tilde{Q}(x, u)$ (b) $\tilde{Q}(x, -1)$ (c) $\tilde{Q}(x, 1)$

Figure 3.8: Triangulation technique used to define the function $W$. The states selected from $X$ to represent $X^\delta$ are 1, 1.875, 3 and $x^t$.

it is possible to compute an approximation of the optimal stationary policy. We obtain a control strategy that consists in taking $u = -1$ on the interval $[1, 2.04688[$ and $u = 1$ on the interval $]2.04688, 3]$. At $x = 2.04688$ the value of the control variable value can be either $-1$ or 1 ($\tilde{Q}(2.04688, -1) = \tilde{Q}(2.04688, 1)$). The approximate optimal stationary policy differs from the exact one (computed in example 2.2.1) on the interval $[2, 2.04688]$.

**Example 3.3.2** We illustrate the representative states technique on the control problem described in example 3.2.2.
$1, 1.1, 1.2, \cdots, 3$ and $x^t$ are the representative states. $\#X^\delta$ is equal to 22. $x_1^\delta$ corresponds to the state 1, $x_2^\delta$ to 1.1, $\cdots$, $x_{21}^\delta$ to 3 and $x_{22}^\delta$ to $x^t$.
The control space being infinite, we consider a finite subset $C^\delta = \{-1, 1\}$ of it to construct the MDP$^\delta$ structure and choose $U^\delta(x^\delta) = C^\delta$ for all $x^\delta \in X^\delta \setminus \{x_{22}^\delta\}$.

The function $W$ is defined by using the triangulation technique.
The disturbance space being infinite, we use the algorithm described in figure 3.6 to compute the MDP$^\delta$ structure. Once the structure is computed we solve the MDP$^\delta$ and compute the approximate $Q$-function by using equation (3.16).
The results obtained for different values of the term $N_{w_i}$ used in figure 3.6 are represented on figure 3.9.
To each of these approximate $Q$-functions corresponds an approximate optimal stationary policy (equation 3.17). In example 3.3.1 we were able to compare the approximate optimal policy computed with the real one. But for this control problem we do not know the optimal policy. Therefore we cannot determine by comparison with the optimal policy which one among the three policies is the best.
In section 3.4 we introduce an algorithm that attributes to each policy a score value, which assesses how close a policy is to the optimal one.



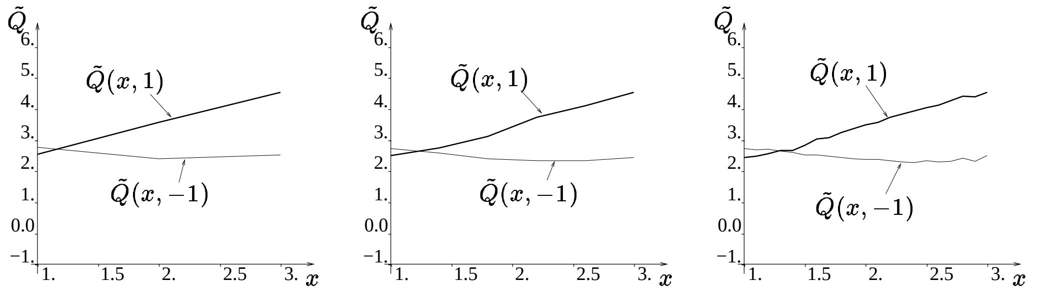
(a) $N_{w_i} = 1$ (b) $N_{w_i} = 10$ (c) $N_{w_i} = 100$

Figure 3.9: Numerical estimation of the approximate $Q$-function. $\#X^\delta = 22$.

Note that on figure 3.10 we have drawn the $Q$-function obtained for different $X^\delta$ when choosing $N_{w_i}$ equal to 1000. This figure has to be compared with figure 3.4.

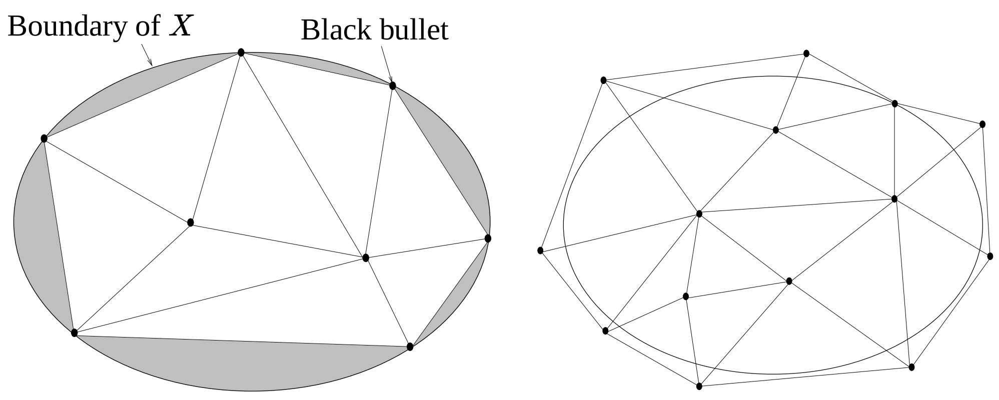
## Problem with the boundary

The triangulation of the state space is not always possible because it requires that the state space $X$ is contained in a finite number of simplices with their vertices inside $X$.
A two-dimensional illustration of the problems of a strictly convex boundary is carried out in figure 3.11a. The grayish areas represent the regions of $X$ that are not recovered by any simplices whose vertices are represented by the black bullets. By increasing the number of black bullets or by better arranging them one can decrease the surface of these areas, which, however, will never disappear completely if $\#X^\delta$ remains finite.



(a) $\#X^\delta = 4$ (b) $\#X^\delta = 7$ (c) $\#X^\delta = 22$

Figure 3.10: Approximate $Q$-function computed for different sizes of $X^\delta$ with $N_{w_i} = 1000$



(a) $X^\delta \subset X$ (b) $X^\delta \not\subset X$

Figure 3.11: Boundary and triangulation

Two strategies can be used to overcome this problem. One consists in mixing the aggregation technique with the representative states technique while the other one consists in choosing the vertices of the simplices outside $X$ in order to ensure that the simplices entirely cover $X$. This latter strategy makes sense if the system dynamics and reward function can still be defined for states close to the boundary but outside $X$. Such a procedure is illustrated on figure 3.11b and is used notably in chapter 4.

### 3.3.7 Other methods to determine $W$

The only method we have described to define the function $W$ is the triangulation technique. The main advantage of this technique is that once the triangulation of the state space is realized, the value of $W$ is determined without having to tune any parameters so as to weight one direction of the state space according to another.
But many other functions $W$ could also be used. For example if $d(x, x')$ is a distance defined on $X$, we could consider the function $W$ defined by the expression :

$$W(x, x^\delta) = \begin{cases} \frac{\frac{1}{d(x,x^\delta)}}{\sum_{x^{\delta'} \in X^\delta} \frac{1}{d(x,x^{\delta'})}} & \text{if} \quad x \neq x_i^\delta \quad \forall i \in \{1, 2, \cdots, m\} \\ 1 & \text{if} \quad x = x^\delta \\ 0 & \text{otherwise} \end{cases}$$

or any other expression that satisfies $W(x, x^\delta) \in [0, 1]\ \forall x \in X\ \forall x^\delta \in X^\delta$ and $\sum_{x^\delta \in X^\delta} W(x, x^\delta) = 1\ \forall x \in X$ and gives an idea of "how close" $x$ is to $x^\delta$.

## 3.4 Quality of the approximate policy computed

As stated in chapter 2, the control problem objective is to find the stationary policy that maximizes the expected value of the return whatever the initial state is i.e., to find the policy $\mu^*$ that maximizes $\forall x \in X$

$$J^\mu(x) = \lim_{N \to \infty} \underset{\mathcal{H}_{N-1}}{E} [\sum_{t=0}^{N-1} \gamma^t r(x_t, \mu(x_t), w_t) | x_0 = x]. \tag{3.25}$$

The better a policy fulfills this objective, the better it approximates the optimal policy. We could thus evaluate a policy $\mu$ by computing the norm

$$\|J^\mu - J^{\mu^*}\|_\infty$$

or any other "norm" measuring the distance of a policy from the optimal one. Unfortunately, in most practical applications, the optimal policy is unknown beforehand and this criterion is therefore not usable. In other words, we need to define a criterion able to compare the quality of two approximate policies without resorting to the knowledge of the true optimal policy or value function.
Of course, we can say that the policy $\mu_1$ better approximates the optimal policy than the policy $\mu_2$ if $J^{\mu_1}(x) \geq J^{\mu_2}(x) \, \forall x \in X$. Unfortunately the use of such a criterion to compare two policies is most of the time not helpful because if the inequality does not hold for just one $x \in X$, nothing can be concluded. We therefore decided to design a more pragmatic criterion to compare the policies' quality. It consists in associating to a policy $\mu$ a *score* defined by the expression :

$$\text{score of } \mu \quad = \quad \lim_{N\to\infty} \underset{\mathcal{H}_{N-1}}{E} [\sum_{t=0}^{N-1} \gamma^t r(x_t, \mu(x_t), w_t) | P_0(x_0)] \tag{3.26}$$

where $P_0(x_0)$ is the probability (or weight) put on the initial state $x_0$ ($P_0(.)$ is defined on $X$). According to this criterion $\mu_1$ is better than $\mu_2$ if the score of $\mu_1$ is higher than the one of $\mu_2$.
Obviously, the score computed for each policy and therefore also the result of the comparison depend on $P_0(.)$.
In particular, if $P_0(.)$ is different from zero for only one element $x$ of $X$ then the comparison between two policies $\mu_1$ and $\mu_2$ amounts to comparing the value of $J^{\mu_1}(x)$ and $J^{\mu_2}(x)$.
If $P_0(.)$ represents a uniform probability distribution, then the comparison between $\mu_1$ and $\mu_2$ amounts to comparing the average of $J^{\mu_1}$ and $J^{\mu_2}$ over the whole state space.
Notice however that whatever the chosen probability distribution $P_0(.)$, no policy can have a score greater than the optimal one.
In the simulations of chapter 4 we will see that to evaluate the quality of policies it is useful to consider several probability distributions $P_0(.)$ in order to concentrate the comparisons on various regions of interest of the state space.

**Example 3.4.1** The control problem used is the four-state control problem defined in example 2.3.1.
We are going to evaluate the score obtained by an $\epsilon$-Greedy policy on this control problem. The $\epsilon$-Greedy policy has been introduced in example 2.3.4 and the expected return of this policy $J^{\epsilon-\text{Greedy}}$ is given in table 2.3 for different values of $\epsilon$.
The initial probability distribution that intervenes in the score computation is defined as follows : $P_0(1) = \frac{1}{3}$, $P_0(2) = \frac{1}{3}$, $P_0(3) = \frac{1}{3}$ and $P_0(x^t) = 0$. The detail of the score

computation for the $\epsilon$-greedy policy with $\epsilon = 0.1$ is :

$$\begin{aligned}
\text{score of } \epsilon\text{-Greedy} &= P_0(1) * J^{\epsilon-\text{Greedy}}(1) + P_0(2) * J^{\epsilon-\text{Greedy}}(2) + \\
&\quad P_0(3) * J^{\epsilon-\text{Greedy}}(3) + P_0(x^t) * J^{\epsilon-\text{Greedy}}(x^t) \\
&= \frac{1}{3} * 2.50884 + \frac{1}{3} * 2.55571 + \frac{1}{3} * 4.12586 + 0 * 0 \\
&= 3.06347
\end{aligned}$$

The scores obtained for different values of $\epsilon$ are gathered in table 3.3. The larger the value of $\epsilon$ the smaller the score.

Table 3.3: Score obtained for different values of $\epsilon$

| $\epsilon$ | score |
|---|---|
| 0. | 3.17006 |
| 0.1 | 3.06347 |
| 0.2 | 2.95450 |
| 1. | 1.99233 |

**Example 3.4.2** The control problem used is the four-state control problem defined in example 2.3.1.
In the examples 2.3.2 and 2.3.3 we saw how to compute the optimal stationary policy of the control problem and obtained : $\mu^*(1) = -1$, $\mu^*(2) = 1$ and $\mu^*(3) = 1$. The maximum expected return i.e., the expected return that corresponds to the optimal stationary policy, is : $J^{\mu^*}(1) = 2.57143$, $J^{\mu^*}(2) = 2.65306$ and $J^{\mu^*}(3) = 4.28571$.
If we define $P_0(x_0)$ as follows : $P_0(1) = \frac{1}{3}, P_0(2) = \frac{1}{3}, P_0(3) = \frac{1}{3}, P_0(x^t) = 0$ and use equation (3.26) to compute the score corresponding to the optimal policy, we obtain :

$$\begin{aligned}
\text{score of } \mu^* &= P_0(1) * J^{\mu^*}(1) + P_0(2) * J^{\mu^*}(2) + P_0(3) * J^{\mu^*}(3) \\
&= \frac{1}{3} * 2.57143 + \frac{1}{3} * 2.65306 + \frac{1}{3} * 4.28571 \\
&= 3.17006
\end{aligned}$$

The four-state control problem has also been used in example 3.2.1 to illustrate the aggregation technique. The policy $\mu$ obtained by using the aggregation technique was : $\mu(1) = 1$, $\mu(2) = 1$ and $\mu(3) = 1$. With the value iteration algorithm (figure 2.2) to compute the expected return of this policy, we obtain : $J^{\mu}(1) = 1.37026$, $J^{\mu}(2) = 2.65306$ and $J^{\mu}(3) = 4.28571$. By proceeding similarly we can compute the score of $\mu$. We get a value of $2.76967$ which is smaller than the score corresponding to the optimal policy.

## Numerical estimation of the score

If the state space is infinite the score of a policy $\mu$ as defined by equation (3.26) can most of the time be estimated only numerically. The numerical estimation of the score implies among others to be able to estimate the expected return $J^{\mu}(x)$ of a policy $\mu$. To estimate $J^{\mu}(x)$, we run simulations starting from $x_0 = 0$ with the policy $\mu$ and observe at each stage the reward obtained. Although computation of the expected return requires simulating the system during an infinite number of stages (if no terminal state is reached), we truncate the estimation of the expected return to its first $k$ terms knowing that the answer obtained will stand in the interval

$$[J^{\mu}(x) - \frac{\gamma^k}{1-\gamma}B_r, J^{\mu}(x) + \frac{\gamma^k}{1-\gamma}B_r] \tag{3.27}$$

where $B_r$ represents the bound on the rewards. An algorithm that allows the numerical estimation of the score is represented on figure 3.12. The *end of the episode* designates either the fact that the system is stuck in a terminal state or that the maximum number of steps in a simulation has been reached.

**Example 3.4.3** Hereafter we use the control problem detailed in example 3.2.2. There we have computed approximate optimal policies by using the aggregation technique for different discretizations of the state space. We have already mentioned that we were unable to decide which discretization led to the best approximation of the optimal policy. Now, we are going to use equation (3.26) to compute a score value for these different policies in order to determine which one is the best.
The probability distribution on the initial states is chosen such that $P_0(.)$ represents a uniform probability distribution on $X \setminus \{x^t\}$ and that $P_0(x^t) = 0$.
The state space being infinite, we use the algorithm given in figure 3.12 to compute the scores. The number of simulations is chosen equal to $5000$ and each simulation is run until the terminal state $x^t$ is reached. The results obtained are represented in table 3.4. The finer the discretization, the better the score. Although we cannot compare these scores to the score obtained by the optimal stationary policy (because we do not know this policy), we can still compare them to the score obtained by a policy that would consist in choosing actions at random i.e., while being in a state $x$ the action taken is drawn randomly in the set $\{-1, 1\}$. This random policy gives a score of $2.82$, well below the score of the three policies obtained by using the aggregation technique.
In example 3.3.2, we have computed by using the representative states technique different policies corresponding to different sizes of the disturbance space subsets used to compute the MDP$^{\delta}$ structure (different $N_{w_i}$). The scores of these different policies (computed by using the algorithm depicted in table 3.12 with the same probability distribution on the initial states and the same algorithm parameters as detailed before) are given in table 3.5. The larger the disturbance space subsets, the better the policy obtained (according to the score criteria).

Figure 3.12: Numerical estimation of the score of a policy $\mu$

```
Initialize $score$ and $numberSimulations$ to zero
Repeat until the maximum number of simulations is reached
    Choose $x$ according to a given probability distribution
    Initialize $scoreSimulation$ to zero
    Initialize $decay$ to 1
    Initialize the state of the system to $x$
    Repeat until the end of an episode
        Apply action $\mu(x)$ on the system
        Simulate the system for one step
        Observe the reward $r$ obtained and the state $x'$ reached
        $scoreSimulation \leftarrow scoreSimulation + decay * r$
        $decay \leftarrow \gamma * decay$
        $x \leftarrow x'$
    $numberSimulations \leftarrow numberSimulations + 1$
    $score \leftarrow score + \frac{1}{numberSimulations}(scoreSimulation - score)$
```

Table 3.4: Score obtained for different discretizations of the state space

| $\#X^{\delta}$ | score |
|---|---|
| 3 | 3.42 |
| 6 | 3.49 |
| 21 | 3.51 |

Table 3.5: Score obtained for different sizes of the disturbance space subsets

| $N_{w_i}$ | score |
|---|---|
| 1 | 3.03 |
| 10 | 3.48 |
| 100 | 3.51 |
| 1000 | 3.52 |

## 3.5 Bellman error method

The aggregation technique and the representative states technique consist in :

- determining an approximation architecture that allows one to compute $\tilde{Q}(x,u)$ from a parameter vector[6] $a$, each component of this vector corresponding to the value of the $Q^{\delta}$-function evaluated at an element of $X^{\delta} \times U^{\delta}$ (equation (3.7) for the aggregation technique and equation (3.16) for the representative states technique)

- computing the optimal value $a^*$ of the parameter vector $a$ by solving the DP equation of the MDP$^{\delta}$ structure.

On the other hand, to solve this problem one could rather select from $X \times U$ a finite set $\mathcal{F}$ of state-action pairs and solve the following optimization problem :

$$\min_{a}\Big(\sum_{(x,u)\in\mathcal{F}} (\tilde{Q}(x,u,a) - \underset{w}{E}[r(x,u,w) + \gamma \max_{u\in U(f(x,u,w))} \tilde{Q}(f(x,u,w),u,a)])^2\Big) \quad (3.28)$$

where we have used the notation $\tilde{Q}(x,u,a)$ rather than $\tilde{Q}(x,u)$ in order to stress the dependence on the parameter vector $a$.
Solving this optimization problem is equivalent to minimizing the error in the Bellman equation. Methods that consist in minimizing the error in the Bellman equation are referred to as the *Bellman error methods* [SS85, Bai95]. If $\tilde{Q}$ can represent exactly the $Q$-function, expression (3.28) is equal to zero whatever the finite set $\mathcal{F}$.
We have introduced the Bellman error method here in order to compare its rationale with the other approaches described in this chapter. A technique inspired from this method will be introduced in chapter 6 in the context of reinforcement learning in infinite state spaces.

[6] The parameter vector $a$ represents the parameters of the approximation architecture.

## 3.6 Summary

*We have described two techniques to compute approximations of the optimal stationary policy for control problems with infinite state spaces.*
*We recall that both techniques consist in defining from the original control problem a finite Markov Decision Process that catches its main characteristics and in using the MDP$^\delta$ solution to approximate the optimal stationary policy of the original control problem. These techniques suppose, like the DP techniques used to solve control problems with finite state spaces, that the system dynamics and the reward function are known.*
*We will provide a detailed case study comparing these methods in the next chapter, in the context of a simple but representative power system control problem.*
*In the subsequent chapters we will see how control problems can be solved by adapting these techniques to situations where the system dynamics and the reward function are not known explicitly anymore but can be observed along various system trajectories.*

# Chapter 4

# A simple FACTS control problem

*In this chapter we*

- *describe the characteristics of a power system composed of one machine connected to an infinite bus through a transmission line and controlled by a Flexible Alternative Current Transmission System (FACTS) device and explain the electrical power oscillations damping problem;*

- *formulate the control problem as a discrete-time optimal control problem such that the electrical power oscillations are well damped if the optimal stationary policy is used to control the FACTS;*

- *explain how to apply the computational strategies detailed in the previous chapter in order to solve the control problem, compare the different results obtained and discuss the policy robustness.*

*We notice that this FACTS control problem is rather simplified with respect to real world conditions. In particular, we suppose full observability of the system state and neglect in the dynamics most of the existing control loops installed on real power systems (prime-mover control, voltage control and power system stabilizers). Last but not least, most real power systems are composed of many more than just the two machines considered here (the infinite bus is an idealization of a very large machine representing in a simplified way the remaining system). Nevertheless, we will see that this problem is sufficiently rich to encounter the purpose of this chapter, which is to show how such a practical problem can be formulated as an infinite horizon optimal control problem and to study and compare the main features of the different algorithms proposed in the previous chapters.*

## 4.1 The OMIB power system

### 4.1.1 The system dynamics

The type of FACTS used is a TCSC (Thyristor Controlled Series Capacitor) which can be considered as a variable reactance (more often a capacitance) in series with a transmission line . We denote by $X_{\text{FACTS}}$ an equivalent 60 Hz steady state reactance of the TCSC. A negative value of $X_{\text{FACTS}}$ means that the FACTS is acting like a capacitance. In the present study we neglect the time constant of the FACTS device and hence consider that $X_{\text{FACTS}}$ represents the control variable of the problem and is deemed to be able to vary infinitely rapidly[1].
The simplified power system model on which the FACTS is installed consists of a single synchronous generator (machine) connected to an infinite inertia machine through a transmission line. This simplified power system will be referred to as an OMIB (One-Machine Infinite Bus) system.
Figure 4.1 sketches the structure of this power system where the symbol $X_{\text{system}}$ embeds the generator transient reactance, the transformer reactance and the line reactance.

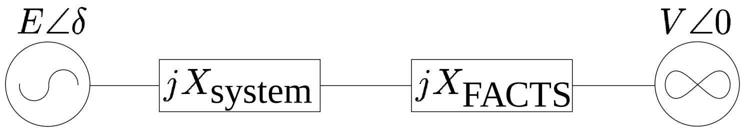


Figure 4.1: The one-machine infinite bus (OMIB) system

The system dynamics is described by the following equations :

$$\dot{\delta} = \omega \tag{4.1}$$

$$\dot{\omega} = \frac{P_m - P_e}{M} \tag{4.2}$$

where $P_m$ is the mechanical power of the machine, $P_e$ its electrical power, $M$ its inertia, $\delta$ its rotor angle (w.r.t. a reference synchronous with the infinite machine), and $\omega$ its rotor speed. $P_m$ is assumed constant whereas $P_e$ varies with $\delta$ according to the following expression :

$$P_e = \frac{EV}{X_{\text{system}} + X_{\text{FACTS}}} \sin\delta. \tag{4.3}$$

[1]The TCSC model considered here is much simplified. More complex models exist. In this respect, we advice the reader to refer to [HG00] and references therein.

$E$ is the terminal voltage of the machine and $V$ the voltage of the infinite bus system.
The values of all the system parameters expressed in per unit are chosen as follows : $E = 1$, $V = 1$, $P_m = 1$, $M = \frac{2H}{w_b} = \frac{2*6}{2*\pi*60} = 0.03183$, and $X_{\text{system}} = 0.4$.
The line reactance has been chosen equal to $0.18$ (it corresponds more or less to the reactance of a $300$ km, $250$ kV line), the transformer reactance to $0.02$ and the machine transient reactance to $0.2$. The value of the FACTS reactance $X_{\text{FACTS}}$ belongs to the interval $[-0.04, 0.]$, corresponding to a maximum of 10 percent of $X_{\text{system}}$ which can be compensated.
Note that often a term $-\frac{D}{M}\omega$ is added to the right hand part of equation (4.2), in order to represent the natural damping of the power system. This term has been deliberately removed here ($D = 0$) so as to emphasize that the damping is produced only by the FACTS itself.
While this power system model is extremely simplified, it is still representative of the main physical features of the practical problem of power system electromechanical oscillations damping. Moreover, the parameters are adjusted to represent realistic orders of magnitude.

### 4.1.2 State space trimming

#### Limitation of the control problem to the stability domain

Although the system dynamics (equations (4.1) and (4.2)) is defined whatever the value of $\delta$ or $\omega$, we limit the control problem state space to the region of the plane $(\delta, \omega)$ that represents the stability domain of the uncontrolled OMIB system ($X_{\text{FACTS}} = 0$). The separatrix is represented on figure 4.2a.

#### Analytical characterization of the stability domain

The states $(\delta, \omega)$ that are inside the stability domain satisfy the inequality

$$\frac{1}{2}M\omega^2 - P_m\delta - \frac{EV}{X}\cos(\delta) \leq -0.438788,$$

(see [Pai89] for more information), the equality occurring for the points that lie on the boundary. Inside this domain, the value of $\delta$ (expressed in $rad$) always lies in the interval $[-0.972, 2.730]$ while the value of $\omega$ (expressed in $rad/s$) belongs to $[-11.927, 11.927]$. This also implies that the value of $P_e$ is always in the interval $[-2.2944, 2.7777]$.

### Stable equilibrium point

Inside the stability domain (represented on figure 4.2a) the uncontrolled system has just one stable equilibrium point defined by

$$(\delta_e, \omega_e) = (\arcsin \frac{X_{\text{system}} P_m}{EV}, 0) = (0.411, 0),$$

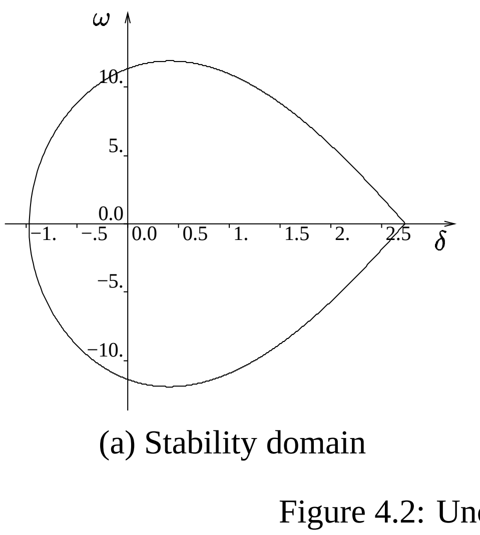
to which corresponds an electrical power transmitted in the line equal to $P_m$.

### Trajectories

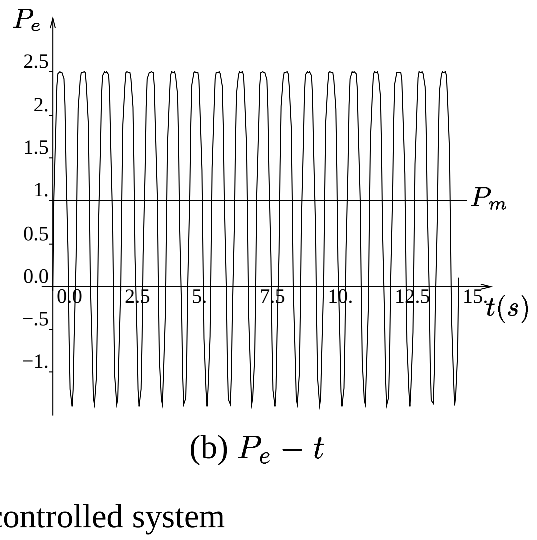
All the trajectories of this uncontrolled system that start inside the stability region remain in it and are closed. This implies among other things that the electrical power transmitted in the line is not damped (see equation (4.3)) and oscillates indefinitely if the starting point of a trajectory is different from the stable equilibrium point. The evolution of this electrical power over a period of $15\,s$ when the the initial angle value ($\delta$) is 0 and the initial speed value ($\omega$) is $8\,rad/s$ is represented on figure 4.2b. Note that we can say that the electrical power transmitted in the line oscillates around the value $P_m$ because it can be shown that

$$\lim_{T\to\infty} \frac{1}{T} \int_0^T P_e(t)dt = P_m$$

if the system stays indefinitely in the stability region.



(a) Stability domain



(b) $P_e - t$

Figure 4.2: Uncontrolled system

## 4.2 Optimal control problem formulation

In chapter 2 we have seen that the two main elements that intervene in the definition of a DP problem are the system dynamics and the reward function (with its associated decay factor $\gamma$).

### 4.2.1 Discrete-time system dynamics

The system dynamics of the power system is described by equations (4.1) and (4.2). This system has continuous-time dynamics (where the control variable is $X_{\text{FACTS}}$). Discrete-time control on such a system means that at instant $t$ the controller observes the state of the system and sets the evolution of the control variable $X_{\text{FACTS}}$ over the time interval $[t, t+1]$. The infinite state space $X$ is composed of the stability region represented on figure 4.2a plus a terminal state that is reached if the observed state has left the stability domain. The time between $t$ and $t+1$ is chosen equal to $0.050\,s$. This time discretization, the bounds on $X_{\text{FACTS}}$ and the fact that $X_{\text{FACTS}}$ can change infinitely rapidly imply that the control space $C$ (the set of all $u$) of the discrete-time control problem is composed of all the functions defined on an interval $[0, 0.050]$ that take their values on $[-0.04, 0]$ (the interval in which $X_{\text{FACTS}}$ is constrained to stay). Moreover, we have $U(x) = C\,\forall x \in X$ for this control problem and a disturbance space $D$ which is empty due to the fact that the system dynamics is deterministic.

### 4.2.2 Reward definition

Our aim is to define a reward function $r(x, u, w)$ and a decay factor $\gamma$ such that the stationary policy $\mu^*$ that maximizes

$$\lim_{N\to\infty} \underset{\mathcal{H}_{N-1}}{E} [\sum_{t=0}^{N-1} \gamma^t r(x_t, \mu(x_t), w_t) | x_0 = x]$$

also leads to the damping of the electrical power oscillations.
We have chosen (electrical powers are expressed in per unit) :

$$r(x_t, u_t, w_t) = \begin{cases} -|P_{e_{t+1}} - P_m| & \text{if} \quad x_{t+1} \neq x^t \\ -100 & \text{if} \quad x_{t+1} = x^t \end{cases}$$

and $\gamma = 0.98$.
The choice of $-|P_{e_{t+1}} - P_m|$ as reward if $x_{t+1}$ is still in the stability domain implies that the closer the instantaneous electrical power is to its value at the equilibrium

point, the larger the reward obtained. Therefore the policy that maximizes the sum of the rewards will also drive the system into regions of the state space where oscillations of the electrical power are small and especially to the equilibrium point where they vanish.
The value of $\gamma$ ($\gamma = 0.98$) has been chosen close to 1 in order to ensure that the rewards importance does not decay too rapidly with time, in comparison with the time constant of the system oscillations. With this value, $\gamma^t$ reaches a value of 10% after 114 time steps i.e., after 5.7 seconds of real time, which is about 6 to 7 times larger than the natural oscillation period of the system (see figure 4.2b).
If the system goes outside the stability domain, the reward gotten is equal to a large negative value ($-100$). This choice intends to guarantee that the optimal control policy does not drive the system outside the stability domain because the return obtained would be very low. We notice that this value of reward is (by far) larger in absolute value than the maximal value of $|P_{e_{t+1}} - P_m|$ in all our simulations. Hence the bound on the rewards $B_r$ is also equal to 100.

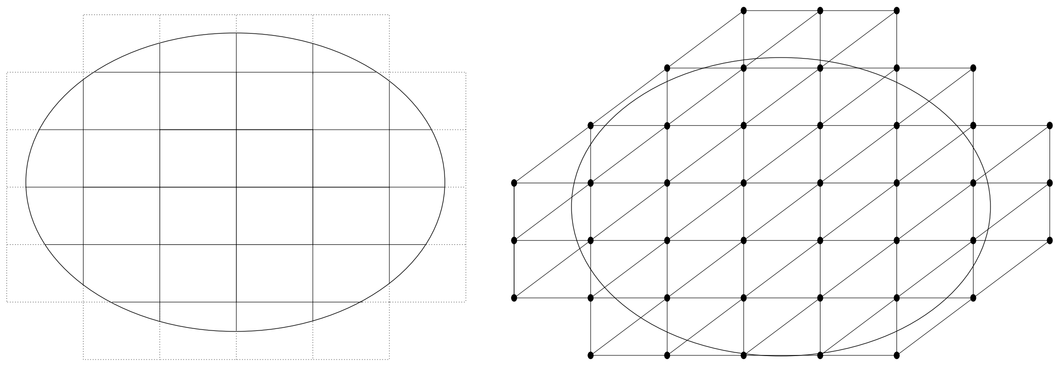
## 4.3 Optimal control problem solution

The FACTS control problem has an infinite state space which implies that we are not able to compute the exact optimal stationary policy but just an approximation of it. The computation of the approximate optimal policy will be done by using the aggregation technique and the representative states technique explained respectively in sections 3.2 and 3.3.

### 4.3.1 Aggregation technique

#### State space discretization

The first phase of the aggregation technique consists in partitioning the state space $X$ into $m$ different disjoint subsets $X_1, X_2, \cdots X_m$. If we take $X_m$ equal to $\{x^t\}$ then the subsets $X_1, X_2, \cdots, X_{m-1}$ must cover the stability domain represented on figure 4.2a. In order to partition (see section 3.2.6) this continuous domain into $m-1$ disjoint subsets, we use a regular grid. The grid structure we use is represented on figure 4.3a. The subsets of the state space that are defined by this kind of grid have the same shape except for the ones that intersect with the boundary. The discretization step of this grid according to the variable $\delta$ is, in a first stage, chosen equal to one fiftieth of the difference between the maximum value of $\delta$ and the minimum value of $\delta$ on $X \setminus \{x^t\}$ ($\frac{2.730-(-0.972)}{50} = 0.07404$). Its discretization step according to the variable $\omega$ is, in a first stage, chosen equal to one fiftieth of the difference between the maximum value of $\omega$ and the minimum value of $\omega$ on $X \setminus \{x^t\}$ ($\frac{11.927-(-11.927)}{50} = 0.47708$). In the sequel we call it the $50 \times 50$ grid; this discretization will be compared both with coarser and finer ones in order to assess the impact of this parameter on the results.



(a) Discretization (aggregation technique) (b) Triangulation (representative states technique)

Figure 4.3: Discretization and triangulation of the state space

The second phase of the aggregation technique consists in associating to each subset $X_i$ a probability distribution $p_i(.)$. The probability distribution $p_i(.)$ is simply chosen equal to a uniform probability distribution on $X_i$, no other choice being justified a priori.

### Control space discretization

In a first stage, we limit the control space $C^\delta$ to two elements, both of which consider constant $X_{\text{FACTS}}$ values on the time interval $[t, t+1]$. But for one element of $C^\delta$ this value is equal to 0 (the FACTS is acting at the minimum level of its capacity) while for the other it is equal to $-0.04$ (the FACTS is acting at full range of its capacity). Denoting by its value an element of the control space that corresponds to a constant value of $X_{\text{FACTS}}$ over the interval $[t, t+1]$, we write : $C^\delta = \{-0.04, 0\}$. Moreover we take $U^\delta(x) = C^\delta \; \forall x \in X \setminus \{x^t\}$.

Subsequently, we will assess the effect of using a richer set of possible control actions.

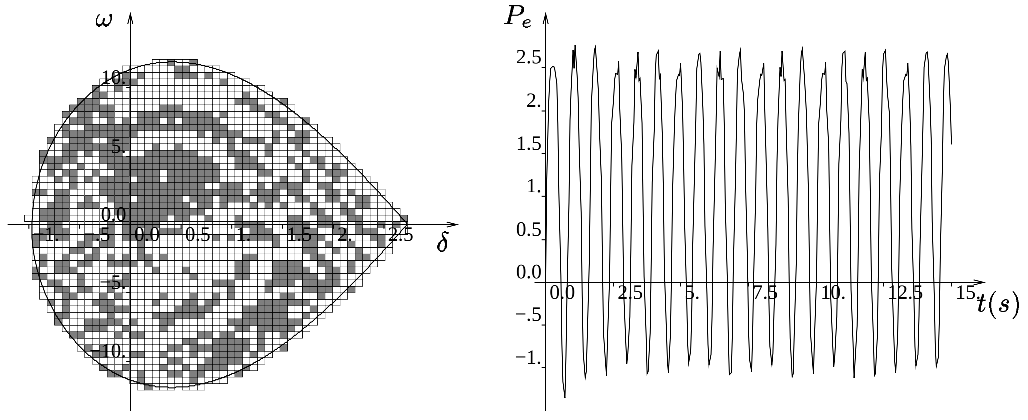
### MDP$^\delta$ structure estimation

The elements $p_i$, $X_i$ and $U^\delta$ having been chosen, we can compute the structure of the MDP$^\delta$ defined by the equations (3.1) and (3.3). Since the state space is infinite, we must evaluate these expressions numerically. To this end, we use the algorithm detailed in figure 3.2. Notice that since the original system is deterministic, the values of the $N_{w_i}$ terms can be taken equal to 1. On the other hand, we decided to take the values of $N_i$ approximately proportional to the size of the subset $X_i$ to which they correspond (therefore the subsets of $X$ which intersect with the boundary of the stability region will have a value of $N_i$ smaller than the subsets which do not intersect with the stability boundary (see figure 4.3)), and such that $\sum_{i=1}^{m-1} N_i = N_T$. Actually, the procedure that we have used to generate $N_i$ points in each cell slightly departs from the scheme defined in figure 3.2. Indeed, rather than sampling separately $N_i$ points in each cell, we sample the total number of points ($\sum_{i=1}^{m-1} N_i$) uniformly inside the complete stability region and then allocate each point to the region it falls in. In principle, this introduces some more variance in the estimation procedure, but for the quite large number of points considered in the present simulations, we have found that this does not lead to any significant differences.
We start by using a value of $N_T = 10000$, leading to a rather small value of $N_i \simeq 6$. Subsequently we will increase these numbers to see how they affect the quality of the resulting control policy.

### Control policy determination

We compute the $Q^\delta$-function of the MDP$^\delta$ by using the Gauss-Seidel version of the value iteration algorithm with $\epsilon = 0.0001$ (figure 2.4). Then, using equations (3.7) and (3.8) in order to compute respectively an approximation of the $Q$-function and from it an approximation of the optimal stationary policy, we obtain the policy represented on figure 4.4a. The grayish areas represent regions of the state space in which the policy consists in choosing $X_{\text{FACTS}}$ equal to $-0.04$ while the white areas designate regions where the control variable value is equal to $0$.
The scattering of white and gray areas seems rather random and is obviously not likely to represent a very efficient policy. We have simulated the power system by using this control law (which means that each $0.050\,s$ we observe $(\delta, \omega)$ and use the control law in order to set the value of $X_{\text{FACTS}}$ over the next $0.050\,s$) in order to see what happens in terms of damping. For example, figure 4.4b represents the evolution of the electrical power observed over a period of $15\,s$ when starting the simulation with $(\delta, \omega) = (0, 8)$. We indeed clearly see that the power system oscillations are not damped. We will shortly see that this is because the aggregation

technique has produced a policy that is unable to approximate sufficiently well the optimal stationary policy.



(a) Policy computed (b) $P_e - t$

Figure 4.4: $50 \times 50$ grid and $\sum_{i=1}^{m-1} N_i = 10000$

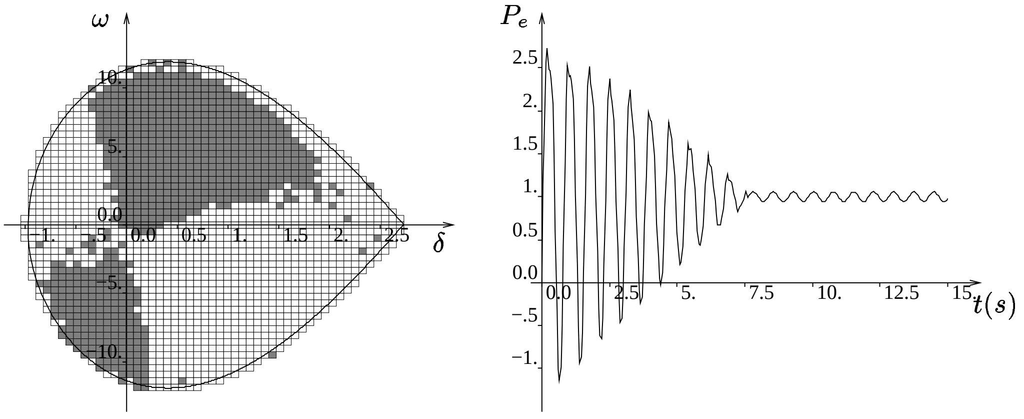
## Policy score evaluation

We can estimate the score of this policy (see section 3.4). The score computation requires choosing a probability distribution on the initial states. We take it equal to a uniform probability distribution on $X \setminus \{x^t\}$. Since the state space is infinite, the score evaluation i.e., the evaluation of expression (3.26), is done numerically by using the algorithm described on figure 3.12. The *maximum number of steps* parameter of this algorithm is chosen equal to 1000 which implies that the estimation of the return $R(x_0)$ lies in the interval

$$[R(x_0) - \frac{0.98^{1000}}{1-0.98}100, R(x_0) + \frac{0.98^{1000}}{1-0.98}] = [R(x_0) - 8.41 * 10^{-6}, R(x_0) + 8.41 * 10^{-6}].$$

For these estimations, the *maximum number of simulations* parameter is taken equal to $25000$. These parameters have been tuned in order to ensure that expression (3.26) is approached with a sufficient accuracy and they are used each time we compute the score for the OMIB system whatever the initial probability distribution may be.
The policy represented on figure 4.4 gives a score value of $-62.257$. If we proceed to the score computation for a policy that chooses always $X_{\text{FACTS}} = -0.04$, we

obtain a score value equal to $-58.7692$. A policy for which $X_{\text{FACTS}}$ is always equal to 0 everywhere gives a score of $-59.0311$ while a completely random policy (a policy for which, while being in a state $x$ we choose arbitrarily a control variable value equal to 0 or $-0.04$) gives a score value of $-66.0511$. We can observe that the policy computed via the aggregation technique has a worse score than the ones which choose constant $X_{\text{FACTS}}$ value.



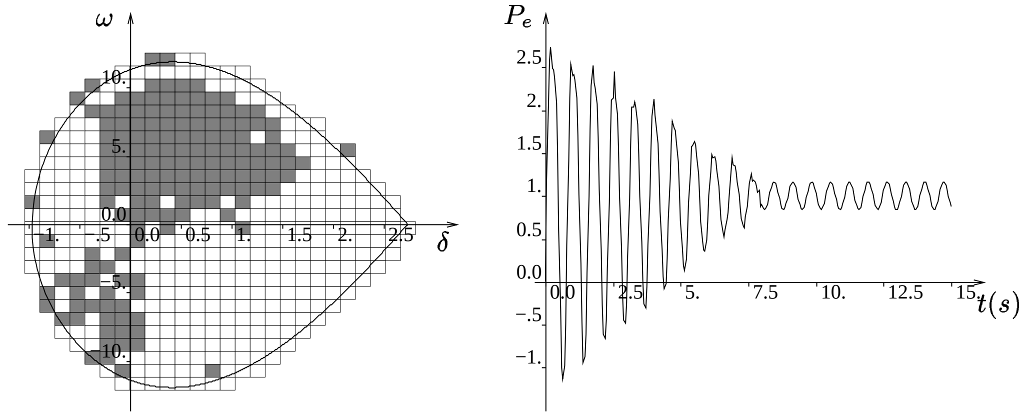
(a) Policy computed (b) $P_e - t$

Figure 4.5: $50 \times 50$ grid and $\sum_{i=1}^{m-1} N_i = 100000$

## Analysis of results and variations

One may wonder why the aggregation technique has provided such poor results. Actually, we found that the problem comes from the way the $p_i(.)$ probability distributions are approximated by the numerical estimation scheme. Indeed, if we take $\sum_{i=1}^{m-1} N_i = 100000$ rather than 10000 and perform all the computations again, we obtain the stationary policy represented on figure 4.5a that gives much better results in terms of electrical power oscillations damping. The evolution of the electrical power when using this policy and starting from $(\delta, \omega) = (0, 8)$ is represented on figure 4.5b. The score is now equal to $-45.8295$.

Another choice of the subsets $X_i$, of the $p_i$ or of the $U^\delta$ control sets would have led to another MDP$^\delta$ structure and therefore to another approximation of the optimal stationary policy. For instance, if we modify the way the state space is discretized by choosing this time a $25 \times 25$ grid and do all the computations with $\sum_{i=1}^{m-1} N_i = 10000$, we obtain the stationary policy represented on figure 4.6a. Its score is equal to $-47.8721$. The evolution of the electrical power corresponding to a simulation starting from the state $(\delta, \omega) = (0, 8)$ when using the policy just computed is represented on figure 4.6b. At the sight of the result obtained, it can seem surprising that the number of points ($\sum_{i=1}^{m-1} N_i$) used to compute the MDP$^\delta$ structure is sufficient while it was not when a finer grid was used (see figure 4.4).



(a) Policy computed (b) $P_e - t$

Figure 4.6: $25 \times 25$ grid and $\sum_{i=1}^{m-1} N_i = 10000$

The relation that exists between the score obtained by the control policy and the number of points chosen in $X \setminus \{x^t\}$ to compute the MDP$^\delta$ structure is illustrated on figure 4.7 for three different discretizations of the state space (done by a $25 \times 25$ grid, a $50 \times 50$ grid and a $100 \times 100$ grid). The score of the policy increases when the number of points used to compute the MDP$^\delta$ structure increases i.e., when the $p_i(.)$ are better approximated by the numerical scheme. We can also note that for a same score value, the finer the grid, the more points needed to compute the MDP$^\delta$ structure. As the number of points increases, the grid that represents the finest discretization of the state space is finally the one that offers the best policy, even if the scores obtained are then close.

We note that this relation between the number of cells in the discretized model and the number of samples that have to be simulated to yield a good policy is directly related to the so-called *bias-variance tradeoff* well known in estimation theory. Indeed, the finer the grid, the better the policy obtained in asymptotic conditions, but also the larger the number of samples needed to estimate the parameters of the MDP$^\delta$ with sufficient accuracy. We observe that in the present application, the refinement of the grid leads to a rather small reduction of bias and a direct increase in variance.

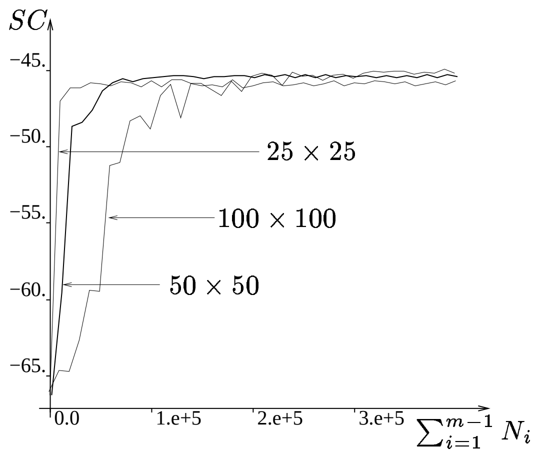


Figure 4.7: Score obtained as a function of $\sum_{i=1}^{m-1} N_i$ for different state space discretizations.

### 4.3.2 Representative states technique

The use of the representative states technique pursues two main objectives :

1. to select a finite number of representative states from the original control problem
2. to define the function $W$.

#### Interpolation scheme and control space

Here, the selected representative states will be located at the intersection of the lines that define a regular grid.

The function $W$ is defined by using the triangulation technique. Due to the convexity of the state space boundary, the triangulation will recover $X \setminus \{x^t\}$ completely only if some of the representative states are outside $X$ (section 3.3.6). On figure 4.3b we have drawn a schematic representation of the state space triangulation.

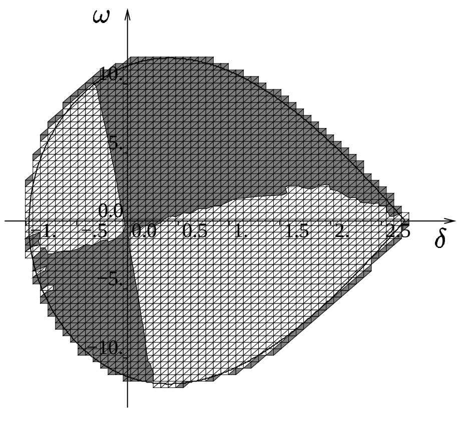
The regular grid used is the $50 \times 50$ grid described in the previous paragraph and the control set $C^\delta$ is chosen equal to $\{-0.04, 0.\}$ with $U^\delta(x^\delta) = C^\delta \; \forall x^\delta \in X^\delta \setminus \{x^t\}$.

### MDP$^\delta$ computation

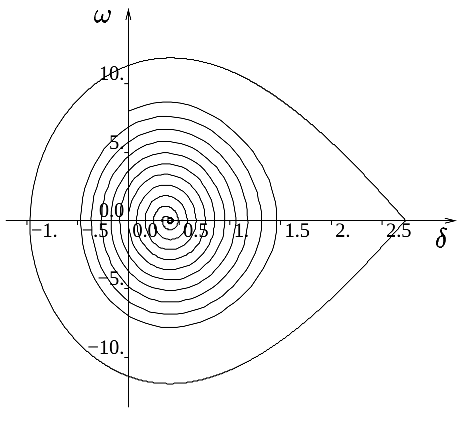
Computation of the MDP$^\delta$ structure can be done by evaluating the expressions (3.14) and (3.15). Notice that since the OMIB system dynamics is deterministic these expressions can be computed exactly as opposed to the aggregation technique where they could only be approximated through sampling. The use of the algorithm described on figure 3.6 with $N_{w_i} = 1$ thus leads to the exact computation of the MDP$^\delta$ structure.

### Control policy computation and score

Once the MDP$^\delta$ is solved (the Gauss-Seidel version of the value iteration algorithm has been used with $\epsilon = 0.0001$ (figure 2.4) to solve this control problem), we can use equation (3.16) to compute an approximation of the $Q$-function. From it we deduce an approximation of the optimal stationary policy by using expression (3.17).



(a) Policy computed



(b) Trajectory in the phase plane

Figure 4.8: Representative states technique. $50 \times 50$ grid used

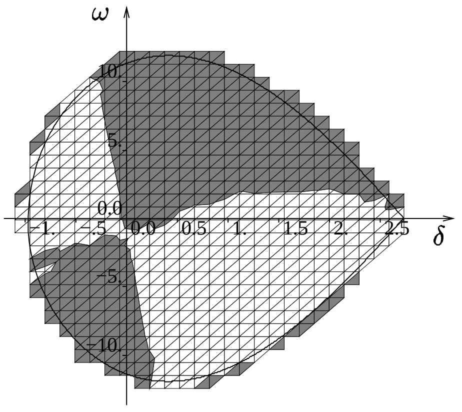
The policy obtained is represented on figure 4.8a where the grayish areas represent regions of the state space where the approximate optimal control policy consists in choosing $u = -0.04$ while the white areas are regions where $u$ is chosen equal to zero.

The score obtained for this policy by choosing $P_0(.)$ equal to a uniform probability distribution on $X \setminus \{x^t\}$ is equal to $-45.1323$.

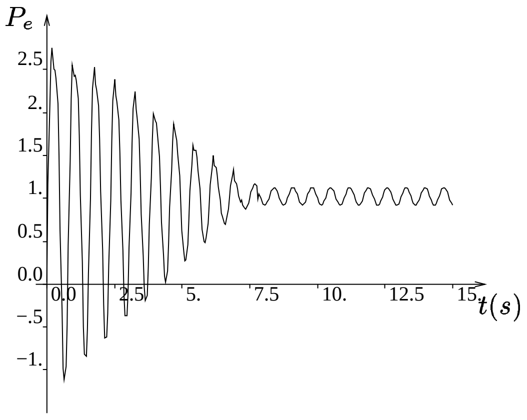
The trajectory obtained by simulating the system from $(\delta, \omega) = (0, 8)$ when this policy is used is represented on figure 4.8b. The system "gets closer" to the equilibrium point of the system, a sign that the electrical power oscillations are well damped. The number of elements comprised in $X^\delta$ is equal to $1748$, $1747$ being represented on figure 4.8a plus the terminal state $x^t$.

### Variations

If we use the same computational scheme but this time with a $25 \times 25$ grid, we obtain the policy represented on figure 4.9a whose score is $-45.2602$. The evolution of the electrical power transmitted in the line when using this policy and when $x_0 = (0, 8)$ is represented on figure 4.9b.



(a) Policy computed



(b) $P_e - t$

Figure 4.9: Representative states technique. $25 \times 25$ grid used

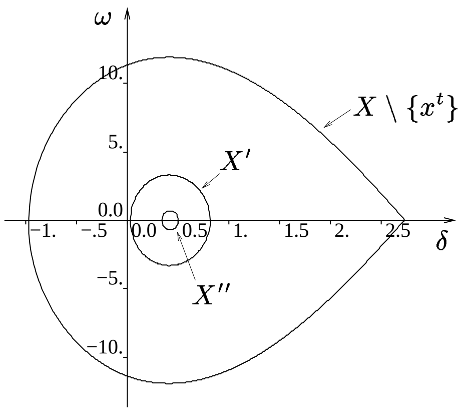
## 4.4 State and control space discretization variants

In section 4.3 we have seen how to compute approximate optimal policies for the FACTS control problem. In this section we study systematically the influence of the technique used (aggregation or representative states technique), the grid used and the control set $C^\delta$ on the policies obtained.

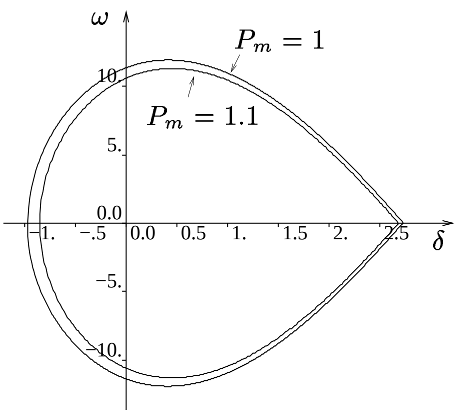
We consider the $25 \times 25$, $50 \times 50$ and $100 \times 100$ grids and the following three control sets :

$$\begin{aligned}
C_1^\delta &= \{-0.04, 0.\} \\
C_2^\delta &= \{-0.04, -0.03, -0.02, -0.01, 0.\} \\
C_3^\delta &= \{-0.04, -0.03, -0.03, -0.01, 0., \\
&\quad \{X_{\text{FACTS}}(t) = -0.01 + 0.01\frac{t}{0.050}, \; t \in [0, 0.050]\}, \\
&\quad \{X_{\text{FACTS}}(t) = -0.01\frac{t}{0.050}, \; t \in [0, 0.050]\}\}
\end{aligned}$$

with $U(x^\delta)$ equal to the corresponding control set $\forall x^\delta \in X^\delta \setminus \{x^t\}$. Note that two elements of $C_3^\delta$ do not correspond to a constant value of $X_{\text{FACTS}}$ on the interval $[t, t+1]$.
To each policy we attribute three different scores corresponding to three different probability distributions on the initial states (equation (3.26)).
These probability distributions differ by the initial states selected in the different subsets of $X$ ($X \setminus \{x^t\}$, $X'$ and $X''$) represented on figure 4.10a.



(a) Three different subset of $X$ used for the score computation



(b) Stability domain and $P_m$

Figure 4.10: Phase plane

$X \setminus \{x^t\}$ represents the stability domain of the uncontrolled OMIB system, $X'$ encloses the states of $X$ that satisfy the equation

$$\frac{1}{2}M\omega^2 - P_m\delta - \frac{EV}{X}\cos(\delta) \leq -2.528$$

and $X''$ encloses the states of $X$ that satisfy

$$\frac{1}{2}M\omega^2 - P_m\delta - \frac{EV}{X}\cos(\delta) \leq -2.695.$$

These three sets contain the stable equilibrium point of the uncontrolled OMIB system and their boundary corresponds to a trajectory of the uncontrolled system ($X_{\text{FACTS}} = 0$).

Each of these three scores can highlight certain features of the policy evaluated. For example if a policy scores well in the domain $X''$, it means that the policy is able to limit the magnitude of the residual oscillations observed on figures and 4.5b, 4.6b and 4.9b.

In order to obtain good results with the aggregation technique, it is necessary to approximate accurately the $p_i$ distributions. This has been achieved by choosing $\sum_{i=1}^{m-1} N_i = 1,000,000$ (section 4.3).

The different scores computed are represented in table 4.1, which suggests the following observations.

Table 4.1: Scores obtained for different ways to compute the MDP$^\delta$

| score | Aggregation | | | Representative states | | |
|---|---|---|---|---|---|---|
| $C_1^\delta$ | $25 \times 25$ | $50 \times 50$ | $100 \times 100$ | $25 \times 25$ | $50 \times 50$ | $100 \times 100$ |
| $X \setminus \{x^t\}$ | -45.7783 | -45.4948 | -45.1414 | -45.2602 | -45.1323 | -44.9836 |
| $X'$ | -7.7821 | -7.7084 | -7.2964 | -8.5559 | -7.5203 | -7.6974 |
| $X''$ | -4.3010 | -1.9856 | -1.3261 | -3.2941 | -1.7097 | -2.1120 |
| $C_2^\delta$ | $25 \times 25$ | $50 \times 50$ | $100 \times 100$ | $25 \times 25$ | $50 \times 50$ | $100 \times 100$ |
| $X \setminus \{x^t\}$ | -45.5320 | -45.1381 | -45.1156 | -45.2254 | -44.9398 | -45.0051 |
| $X'$ | -8.0571 | -7.4946 | -7.0881 | -7.8403 | -7.3256 | -7.1877 |
| $X''$ | -2.4281 | -1.5511 | -1.1113 | -2.1668 | -1.3313 | -1.1646 |
| $C_3^\delta$ | $25 \times 25$ | $50 \times 50$ | $100 \times 100$ | $25 \times 25$ | $50 \times 50$ | $100 \times 100$ |
| $X \setminus \{x^t\}$ | -45.8621 | -45.3057 | -45.3147 | -45.2238 | -44.9575 | -44.9396 |
| $X'$ | -8.4662 | -7.5461 | -7.2803 | -8.5865 | -7.1309 | -7.0877 |
| $X''$ | -2.8150 | -1.6641 | -1.0763 | -3.2834 | -0.9528 | -0.8580 |

### 4.4.1 About the different scores

The smaller the domain used to compute the score, the larger its interval of variation. Indeed we have respectively for the scores corresponding to the domains $X \setminus \{x^t\}$, $X'$ and $X''$ the intervals $[-45.8621, -44.9396]$, $[-8.5559, -7.0877]$ and $[-4.3010, -0.8580]$. The best scores are provided by the representative states technique with a $100 \times 100$ grid and the $C_3^\delta$ control set. The worst are achieved by using a $25 \times 25$ grid (with the aggregation technique for the scores that correspond to the domains $X \setminus \{x^t\}$ and $X''$ and the representative states technique for the score that corresponds to $X'$).

### 4.4.2 About the technique used

It is difficult to assess exactly which technique gives the best results. If we use a $50 \times 50$ grid we always obtain better results with the representative states technique. However this observation does not hold true anymore with a $25 \times 25$ or a $100 \times 100$ grid. Nevertheless, due to the fact that the best results for the three scores are always obtained with the representative states technique, this one may be preferred. The preference can be strengthened if we take into account the larger amount of computation necessary to compute a policy with the aggregation technique.

### 4.4.3 About the grid used

For the aggregation techniques, the finer the grid, the better the scores are. Although this tendency is still verified for the representative states technique if we limit the comparison to the $25 \times 25$ and the $50 \times 50$ grids, this does not hold true anymore if we consider the $100 \times 100$ grid. Sometimes the $50 \times 50$ grid gives better results.

One may wonder why sometimes the results are worse with a finer grid. It seems reasonable to assume that the finer the grid is, the better the scores should be. Figure 4.11 answers this question. It shows the evolution of the electrical power when using the policy computed by using a $100 \times 100$ grid, the representative states technique and $C_1^\delta$ as control set. The residual electrical power oscillations do not have the same aspect as the ones observed on figure 4.5b, 4.6b and 4.9b. In this case the residual oscillations have a harsh aspect caused by rapid changes of $X_{\text{FACTS}}$ when the system moves around its equilibrium point. This phenomenon is linked to some "misfortune" in the way the grid is placed on the phase plane. For example by shifting the grid of 0.01 to the right in the $\delta$ direction we obtain a policy with a score related to $X'$ equal to $-7.3082$ and a score related to $X''$ equal to $-1.3096$. These scores are higher than the corresponding ones obtained with the $50 \times 50$ grid.

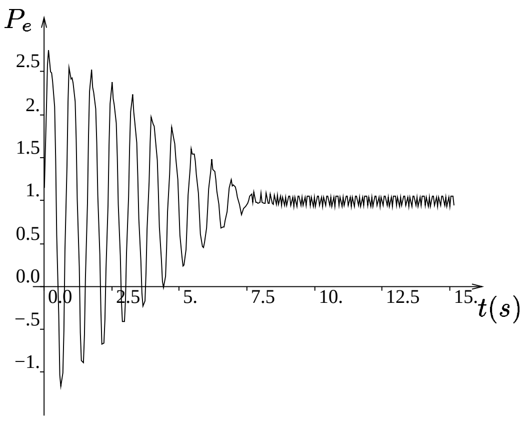


Figure 4.11: Rapid changes of the control variable value around the equilibrium point

### 4.4.4 About the control set used

In general, the control set $C_2'^{\delta}$ offers better results than $C_1'^{\delta}$. The most striking improvements correspond to the scores related to $X''$ for which small variations of $X_{\text{FACTS}}$ are needed to smooth small oscillations of the system. Although the best scores are obtained for $C_3'^{\delta}$, the use of this control set sometimes deteriorates the quality of the result observed especially when the grid is coarse. The finer the grid the better the influence a complex control set tends to have on the scores computed.

## 4.5 On the combined use of multiple MDP$^{\delta}$

We report here on some preliminary investigations that we have carried out in order to further improve performances. The idea, inspired from the so-called ensemble techniques used in automatic learning [Die00], consists in exploiting in parallel several MDP$^{\delta}$ structures and combining the optimal decisions provided by these models into a single one using a majority vote. This works as follows when the aggregation technique is used[2] :

- choose a reference grid;
- for this choice, create $n$ approximation structures MDP$_i^{\delta}$, $\forall i = 1, \ldots, n$, by shifting the reference grid in the state-space by a randomly drawn offset

[2]It is straightforward to extend the procedure to the representative states technique.

vector;

- draw a sample $N_T$ of states in the original state-space (in the same way as previously described);
- for each MDP$^\delta_i$, use this sample of states (together with the computed successors and rewards associated to each admissible control value) in order to estimate the transition probabilities and reward function of this MDP$^\delta_i$;
- for each MDP$^\delta_i$ thus obtained, determine the $Q^\delta$-function by dynamic programming and deduce from it the corresponding control policy $\tilde{\mu}^*_i$ for the original problem;
- define the "team control policy" by majority vote on the individual policies i.e., by

$$\tilde{\mu}^*_T(x) = \arg\max_{u \in U^\delta(x)} \sum_{i=1}^{n} I_{\{u\}}(\tilde{\mu}^*_i(x)). \tag{4.4}$$

This approach is liable to reduce variance with respect to a single model and it is also liable to reduce the bias introduced by the discretization grid. Notice that this scheme evokes also a multi-agent control scheme where several agents propose their control decision to a higher level control agent, which uses this information to decide about the actual control to be applied to the system.

Table 4.2 shows the assessment of this technique in identical conditions to those used in the preceding section, with three different grid sizes and both approximation techniques, in the case of the $C^\delta_1$ control set. In these simulations, the random offset is drawn with a uniform distribution over one grid cell located at the origin of the state space. The upper part of the table recalls the scores of table 4.1; the lower part shows the scores obtained with a team of 10 control agents.

We observe that the performance is increased significantly in the regions close to the stable equilibrium point (regions $X'$ and $X''$) in all cases. All in all, the best results are obtained with a rather coarse $25 \times 25$ grid and the representative states technique. The global performances (scores averaged over the set $X \setminus \{x^t\}$) are also improved, except for the $100 \times 100$ grids where they are very slightly decreased.

Notice that for the aggregation technique these simulations have been carried out for very large samples ($N_T = 1,000,000$) and hence do not really highlight the variance reduction capability of this approach. This latter will be illustrated in chapter 7 by comparing the scores obtained for growing sample sizes.

Table 4.2: Combined use of multiple MDP$^\delta$

| | Aggregation | | | Representative states | | |
|---|---|---|---|---|---|---|
| Classical | $25 \times 25$ | $50 \times 50$ | $100 \times 100$ | $25 \times 25$ | $50 \times 50$ | $100 \times 100$ |
| $X \setminus \{x^t\}$ | -45.7783 | -45.4948 | -45.1414 | -45.2602 | -45.1323 | -44.9836 |
| $X'$ | -7.7821 | -7.7084 | -7.2964 | -8.55596 | -7.5203 | -7.6974 |
| $X''$ | -4.3010 | -1.9856 | -1.3261 | -3.2941 | -1.7097 | -2.1120 |
| $n = 10$ | $25 \times 25$ | $50 \times 50$ | $100 \times 100$ | $25 \times 25$ | $50 \times 50$ | $100 \times 100$ |
| $X \setminus \{x^t\}$ | -45.2853 | -45.1383 | -45.1562 | -44.9469 | -45.0006 | -45.0394 |
| $X'$ | -7.5978 | -7.4607 | -7.1205 | -7.2080 | -7.2091 | -7.2255 |
| $X''$ | -1.4207 | -1.5430 | -0.9828 | -1.1780 | -1.1360 | -1.2338 |

## 4.6 Robustness of the control law

In this section we discuss the robustness of the control law i.e., how the stationary policy computed behaves when the original system dynamics (described by equations (4.1) and (4.2)) changes. The robustness of the control law is studied for three different types of modifications of the system dynamics : a change in a parameter value, an introduction of noise and an addition of an Automatic Voltage Regulator (AVR) to the synchronous machine (this modification altering the order of the system dynamics). The control law whose robustness is studied uses the representative states technique with a $50 \times 50$ grid and $C^\delta = \{-0.04, 0.\}$ (section 4.3). For each modification of the system dynamics we compare the score of this policy (referred to as the *original policy*) with the score of a policy computed by taking into account the new system dynamics (referred to as the *tailor made policy*).

### 4.6.1 Robustness to a parameter change

The parameter we change in the original system dynamics is the mechanical power $P_m$ produced by the generator. We increase it by 10 percent, that is we take $P_m$ equal to 1.1 (note that an increase in the generator mechanical power modifies the stability domain of the system as represented on figure 4.10b).
The scores obtained with the original policy (for three different probability distributions on the initial states (section 4.4) by noting that this time $X \setminus \{x^t\}$ is chosen equal to the stability domain of the uncontrolled power system for which $P_m = 1.1$) are gathered in the second column of table 4.3. If we use the representative states

technique with a $50 \times 50$ grid and a control space $C^{\prime\delta} = \{-0.04, 0.\}$ (section 4.3) to compute a new control law by taking into account the fact that the mechanical power of the generator has been modified, we obtain the scores represented in the third column of table 4.3. Although the score values of the *tailor made policy* are better, the difference is not so important, a sign that the control law robustness to a modification of the generator production is good.

Table 4.3: Robustness of the control law to a parameter change

| score | Original policy | Tailor made policy |
|---|---|---|
| $X \setminus \{x^t\}$ | -43.5461 | -43.4572 |
| $X'$ | -6.96496 | -6.80716 |
| $X''$ | -1.61503 | -1.42858 |

## 4.6.2 Robustness to noise

The noise is introduced in the original system dynamics by considering that the voltage $V$ of the infinite bus system is not constant anymore but expressed by the equation : $V = 1 + w$ where $w$ is a noise factor. The value of $w$ is drawn according to a Gaussian distribution (with a zero mean and 0.2 standard deviation) and refreshed in the integration process each 0.01 $s$ . The scores obtained with the original policy are represented in the second column of table 4.4 while those obtained with a tailor made policy are gathered in the third column. This tailor made policy is computed by using the representative states technique with a $50 \times 50$ regular grid and a control set $C^{\delta} = \{-0.04, 0.\}$ (section 4.3). Because the system dynamics is stochastic, the parameters of the MDP$^{\delta}$ structure could not be computed exactly and were estimated by using the algorithm explained in figure 3.6 with $N_{w_i}$ equal to 100.
The robustness of the control law to noise is even more impressive than its robustness to a change of the mechanical power of the synchronous machine.

## 4.6.3 Robustness to modeling

In order to test the robustness of the control law to an increase in the system dynamics order, we add an AVR to the original OMIB system. This AVR modifies the field voltage of the synchronous machine (represented by $E$) in order to ensure that the voltage at the middle of the electrical distance between $E$ and $V$ tends to reach

Table 4.4: Robustness of the control law to noise

| score | Original policy | Tailor made policy |
|---|---|---|
| $X \setminus \{x^t\}$ | -47.2678 | -47.2436 |
| $X'$ | -8.96947 | -8.90631 |
| $X''$ | -3.24162 | -3.06327 |

a reference value $E_{\text{AVR}}$ (the FACTS is supposed to be installed after the middle of the electrical distance between $E$ and $V$; i.e., just after the $\frac{X_{\text{system}}}{2}$ reactance). The system dynamics can be described by the set of equations :

$$\dot{\delta} = \omega \tag{4.5}$$

$$\dot{\omega} = \frac{P_m - P_e}{M} \tag{4.6}$$

$$\dot{E} = \frac{E_{\text{AVR}} - V_{\text{middle of the electrical distance}}}{T_{\text{AVR}}} \tag{4.7}$$

where $E_{\text{AVR}}$ represents the AVR voltage reference, $T_{\text{AVR}}$ the AVR time constant and $V_{\text{middle of the electrical distance}}$ the voltage at the middle of the electrical distance between $E$ and $V$, just before the FACTS.
$V_{\text{middle of the electrical distance}}$ is equal to

$$\sqrt{\frac{((1 - \frac{0.5X_{\text{system}}}{X_{\text{system}} + X_{\text{FACTS}}})E\cos\delta + \frac{0.5X_{\text{system}}}{X_{\text{system}} + X_{\text{FACTS}}}V)^2 +}{((1 - \frac{0.5X_{\text{system}}}{X_{\text{system}} + X_{\text{FACTS}}})E\sin\delta)^2}} \tag{4.8}$$

and we have chosen $T_{\text{AVR}} = 0.040$ and $E_{\text{AVR}} = 1$. Besides, we suppose that $E$ belongs to the interval $[0.9, 1.1]$ which means that there is a limitation on the synchronous machine field voltage.
The use of the original policy on this system implies that the value of the control variable depends only on the value of $(\delta, \omega)$ and not on the newly created state variable $E$. The score values computed for this original policy (the initial states used for the scores computation have a value of $E = 1$ with their values of $\delta$ and $\omega$ chosen as previously) are gathered in the second column of table 4.5. These score values are surprisingly good compared to the ones computed for the system without AVR (table 4.1). This is due to the fact that the AVR produces some damping itself. Indeed, the policy that would consist in choosing the value of the control variable

at random among $\{0, -0.04\}$ gives a score corresponding to $X \setminus \{x^t\}$ equal to $-56.0455$ while it was $-66.0511$ without AVR.
In order to compute the tailor made control law, we define a new state space equal to

$$\{(\delta, \omega, E) \in \mathbb{R}^3 | (\delta, \omega) \in \text{ stability domain of the original system and} E \in [0.9, 1.1]\} \cup \{x^t\}$$

and from this new state space we select the representative states. The $(\delta, \omega)$ values of the representative states correspond to the values chosen to design the original control law. Their $E$ values belong to the 11 elements set $\{0.9, 0.92, \cdots, 1.08, 1.1\}$. The representative states are at the intersection of the lines that define a regular grid in $\mathbb{R}^3$ having a discretization step according to the state-variable $E$ (denoted by $\Delta E$) equal to $0.02$.
Concerning the control set $C^{\delta}$, we choose it equal to $\{-0.04, 0\}$.
Once the triangulation is done by using the algorithm described in appendix B, we can compute the approximation of the optimal stationary policy. By evaluating this policy we obtain the scores represented in the third column of table 4.5. And these scores are surprisingly worse than the ones obtained with the original policy. This is caused by a too coarse $n$-grid according to the state variable $E$. Indeed, by decreasing $\Delta E$ the trend can be reversed (as illustrated in table 4.5).
Remark that table 4.5 shows also the excellent robustness of the original control law to the addition of an AVR.

Table 4.5: Robustness with respect to system modeling

| score | Original policy | Tailor made policy | | |
|---|---|---|---|---|
| | | $\Delta E = 0.02$ | $\Delta E = 0.01$ | $\Delta E = 0.005$ |
| $X \setminus \{x^t\}$ | -38.0874 | -38.6561 | -38.1497 | -37.9342 |
| $X'$ | -5.9032 | -6.4278 | -6.0341 | -5.8917 |
| $X''$ | -1.7608 | -2.3210 | -2.0147 | -1.7530 |

It may be interesting to note that with $\Delta E$ equal to $0.005$ the MDP$^{\delta}$ numbers $71587$ states ($1746 * 41 + 1$) whereas the MDP$^{\delta}$ corresponding to the original control policy is composed of $1747$ states. This exponential increase in $\#X^{\delta}$ is the main limitation to the application of the aggregation or the representative states techniques to power systems modeled with a large number of state variables. Nevertheless, the high robustness of the control law suggests that we could still design the control policy for a less complex system and extend the results to the large system.

## 4.7 Summary

*In this chapter we have investigated in detail the application of the two techniques introduced in chapter 3 to compute approximations of the optimal stationary policy for control problems with infinite state spaces in the context of a simple FACTS control problem.*

*We found that both techniques allow us to solve this non-trivial control problem in a quite satisfactory way. The aggregation technique however intrinsically presents a bias variance tradeoff which was not observed in the representative states technique.*

*Obviously, these discretization techniques lead in both cases to a combinatorial explosion of the computing effort with the size of the system state space. However, the resulting control policies are found to be quite robust with respect to modeling errors. In particular, we observe that using a reduced order system model to tune the control policy could be an approach to limit this computational problem in practice. Another approach, not investigated in the present thesis would consist of using adaptive multidimensional discretization techniques such as regression trees or nearest-neighbor like methods [BEW03].*

*The main limitation of the approaches investigated in this chapter is related to the fact that they need an explicit analytical description of the system dynamic model and reward function (more precisely, they need to be able to sample these models at an arbitrary point in the state space and for an arbitrary value of the control input).*

*Together, these two difficulties somewhat reduce the usefulness of these techniques to simple power system control problems for which a good enough analytical simulation model is available. Nevertheless, given the flexibility of the framework and the robustness of the resulting control policies, we believe that a significant number of power system control problems could be solved quite efficiently in this way.*

*In the following chapters of this thesis, we will study how to solve control problems when only the system trajectory can be observed through some particular measurements i.e., without using an analytical description of system dynamics and reward functions. The resulting extensions of the DP based techniques, called reinforcement learning, are applicable in a much larger range of conditions. In particular, in chapter 7 we will use the "simple FACTS control problem" to assess these reinforcement learning algorithms and in chapter 9 we will consider a full fledged multi-machine power system control problem, while relaxing completely observability requirements and time-invariance assumptions. Finally, in chapter 10 we will provide a discussion of the different practical schemes that could be envisaged in order to apply these techniques to almost any power system control problem.*

# Chapter 5

# Reinforcement learning

*In chapter 2 we have seen how solutions of optimal control problems with finite state spaces and control spaces could be computed. The procedure described there consisted first in defining from the reward function and the system dynamics knowledge a MDP equivalent to the initial control problem and in solving it by using classical DP algorithms such as the value iteration or the policy iteration algorithm. In this chapter, we describe how to solve such control problems but without assuming that the system dynamics and the reward function are known explicitly. To compensate for this lack of information on the control problem, we only suppose that the system can be controlled according to any policy and that we are able to observe at each step of the control process the current state and the reward obtained.*

*The algorithms that determine the policy used for controlling the system and use the observations to learn information about the control problem solution are referred to as reinforcement learning algorithms. Their description is the purpose of this and the next chapter of this thesis. The present chapter focuses on finite state and action spaces, while the next chapter focuses on the extension of these ideas to infinite state spaces, according to an approach similar to that of chapter 3.*

*For a more intuitive description of the reinforcement learning algorithms, the reader may refer to [SB98]. We also kindly advise him to consult [BT96] for a deeper mathematical analysis, especially concerning the convergence issues of the non-model based algorithms.*

Figure 5.1: Generic reinforcement learning method

```
Repeat indefinitely
    $t = 0$
    Observe state $x_t$
    *Action Selection :* take action $u_t \in U(x_t)$
    Repeat until the end of the episode
        Observe state $x_{t+1}$ and the reward value $r_t$
        *Action Selection :* take action $u_{t+1} \in U(x_{t+1})$
        *Learning*
        $t \leftarrow t + 1$
```

## 5.1 The generic reinforcement learning method

This chapter is devoted to the computation of the solution of discrete-time optimal control problems (as defined in chapter 2) for which the system dynamics and reward function are unknown. To overcome this absence of knowledge we suppose that the system can be controlled according to any policy and that at each time step $t + 1$ we can observe the state $x_{t+1}$ of the system and the value of the reward $r_t = r(x_t, u_t, w_t)$.

The *reinforcement learning methods* aim at solving such kind of problems. These methods decide on the choice of the action $u_t$ for all $t$ and use the observations made in order to learn the control problem solution. A generic version of a reinforcement learning method is represented on figure 5.1[1].

Two modules compose a reinforcement learning algorithm. One is the *Action Selection* module that provides the algorithm with a method to decide which action to take while being in a state and the other is a *Learning* module that uses the observa-

[1]In this generic version of the RL methods we have considered that the period during which the reinforcement learning algorithms interact with the system is partitioned into episodes. An episode begins at $t = 0$ and ends when the control process is stopped. We have considered in this generic algorithm that when the control process is stopped it is restarted again, in some fashion, which yields a new episode.

tions done on the system to learn information about the control problem solution.
The *Action Selection* module design amounts to choosing a policy. This policy is often asked to meet two characteristics that are usually incompatible. One is to be able to use at best the information already obtained about the control problem solution and the other to guarantee that the speed at which the information is learned is maximal.
There are basically two families of *Learning* modules, that is two families of methods, which allow to use the observations done on the system in order to learn the control problem solution. They are known as the model based and the non-model based methods. The model based methods reconstruct the MDP structure that corresponds to the control problem and solve it by using DP algorithms (to provide an estimate of the $Q$-function) while the non-model based techniques directly learn its $Q$-function without reconstructing any model.
The *Action selection* section (section 5.2) presented in this chapter actually holds valid for systems with both finite and infinite state spaces, contrary to the *Learning* sections that, in the present chapter, suppose that the number of system states is finite.

## 5.2 Action selection

We will see in the next two sections that the *Learning* modules provide an estimate of the $Q$-function of the control problem, which is refreshed each time new observations are done on the system. In order to have convergence of the estimated $Q$-function to the exact $Q$-function of the control problem, all these *Learning* modules require that for each $x \in X$, each action $u \in U(x)$ is taken an infinite number of times. Any policy that achieves this is able to drive the *Learning* module to a correct estimation of the $Q$-function and by using equation (2.13) to determine the optimal policy too.
But the choice of the policy may not be just realized in order to guarantee the correct estimation of the $Q$-function. Indeed, the choice of the policy may also be motivated by the fact that it can use the information gathered by the learning process in order to obtain at each instant $t$ the best possible return

$$R_t = \sum_{k=t}^{\infty} \gamma^{k-t} r(x_k, u_k, w_k).$$

This implies that the policy should exploit the current estimate of the $Q$-function to decide which action to take. This can be done by taking, while being in state $x$, the

Figure 5.2: *Action selection* : $\epsilon$-Greedy policy

```
**Input :** Current state $x$ of the system
Choose a number at random in the interval $[0,1]$ with a uniform probability.
If the number is inferior to $\epsilon$ then choose the control variable value randomly in $U(x)$. Otherwise choose the control variable value randomly in $\{u \in U(x) | Q(x,u) = \max_{u' \in U(x)} Q(x,u')\}$.
```

action $u$ that satisfies

$$u = \arg\max_{u' \in U(x)} \hat{Q}(x, u'),$$

where $\hat{Q}$ denotes the current estimate of the $Q$-function. Unfortunately, with such a strategy the *Learning* module is not ensured to get sufficient information to converge to the right solution. While being in a state $x$, it is possible that the policy will always consist in triggering the same action, without even evaluating the effect of the others.

Thus, an ideal policy would be a policy that causes the fastest convergence of the *Learning* module to the exact solution of the control problem while ensuring at the same time that the return obtained is the best. Unfortunately, the more you achieve one of these objectives the less the other is met. In the reinforcement learning literature this is known as the *exploration-exploitation* tradeoff. It assumes that the more exploration you do, e.g. the more often the action is taken at random, the worse the return you obtain is and vice versa. There exists a vast literature about this tradeoff and we kindly advise the reader to consult for example [Meu99], [Meu96] or [ACBF02] and references therein in order to obtain more accurate information about it.

The policy we use in the applications is the so-called $\epsilon$-Greedy policy ($\epsilon \in [0,1]$). It consists in selecting with a $(1-\epsilon)$ probability an action that maximizes $Q(x,u)$ and with a probability $\epsilon$ an action taken at random among $U(x)$. The larger the value of $\epsilon$ is, the more exploration and the less exploitation the policy does. When the policy chooses in state $x$ an action $u$ that maximizes $\hat{Q}(x,.)$, we usually say that the policy has chosen a *greedy action* while if the action chosen does not maximize $\hat{Q}(x,.)$ we say that a *non-greedy action* has been taken.

The $\epsilon$-Greedy policy has already been used in examples 2.3.4 and 3.4.1 and its tabular version is given in figure 5.2. Moreover the tradeoff between exploration and exploitation is illustrated in example 5.3.1 on the four-state control problem.

Figure 5.3: *Learning* : model based technique

---

*Estimation of the MDP structure*
*Resolution of the MDP*

---

## 5.3 Learning : model based techniques

### 5.3.1 Introduction

The model based methods use the observations done on the system in order to estimate the structure of the MDP that corresponds to the control problem i.e., in order to estimate the $r(x, u)$ and $p(x'|x, u)$ values $\forall x, x' \in X$ and $\forall u \in U(x)$, and to solve this MDP in order to get an estimate of the $Q$-function of the system and therefrom an estimate of the optimal policy. The procedure is sketched on figure 5.3 where we have divided the *Learning* module into two distinct modules. One that is responsible for the estimation of the MDP structure and the other for the MDP resolution.
We will describe two algorithms that allow estimating the MDP structure from the observations done on the system. One will be referred to as the Kalman Filter like algorithm[2] (section 5.3.2) and the other as the Stochastic Approximation algorithm (section 5.3.3)[3]. These algorithms have been named according to the numerical methods they use (described in appendix A) to carry out the MDP structure estimation.
The resolution of the MDP can be based for example on the value iteration or the policy iteration algorithms explained in chapter 2. Nevertheless the use of such algorithms in this context requires some specific attention, this will be the subject of section 5.3.4.

### 5.3.2 Kalman Filter like algorithm

The Kalman Filter like algorithm is represented on figure 5.4. At each stage $t + 1$ of the control process, the algorithm uses the values of $x_t$, $u_t$, $r_t$ and $x_{t+1}$ to refresh

[2]This estimation is also known as the Maximum Likelihood Estimation (MLE) of the MDP structure.

[3]In section 6.6, when describing *Learning* modules for infinite state space control problems, we will see that an algorithm coming from the same family as the Stochastic Approximation algorithm described here implies less computational burden than an algorithm coming from the same family as the Kalman Filter like algorithm presented here.

Figure 5.4: *Estimation of the MDP structure* : Kalman Filter like algorithm

```
Input : $x_t$, $u_t$, $r_t$ and $x_{t+1}$
$H(x_t,u_t) \leftarrow \beta H(x_t,u_t)+1$
$r(x_t,u_t) \leftarrow r(x_t,u_t)+\frac{1}{H(x_t,u_t)}(r_t - r(x_t,u_t))$
Repeat for all $x \in X$
    $p(x|x_t,u_t) \leftarrow p(x|x_t,u_t)+\frac{1}{H(x_t,u_t)}(I_{\{x\}}(x_{t+1}) - p(x|x_t,u_t))$
```

the MDP structure. If $\beta = 1$ the function $H(x,u)$ defined on $X \times U$ represents the number of times the action $u$ has been triggered while being in state $x$. The value of this function is initialized to 0 everywhere at the beginning of the learning. If $\beta < 1$, more weight is given to the last observations in the MDP structure estimation. This feature has been added in order to ensure that the reinforcement learning algorithm can act in a time-varying environment[4]. For time-invariant systems, $\beta$ should obviously be chosen equal to 1.
Let $N(x,u)$ be the number of times the particular state-action pair $(x,u)$ has been visited[5]. Let $w(i)$ be the value of the random variable $w$ the $i$th time the state-action pair $(x,u)$ has been visited. Each $w(i)$ is drawn independently according to $P_w(.|x,u)$.
It can be shown that the value of $r(x,u)$ estimated by the algorithm minimizes

$$\sum_{i=1}^{N(x,u)} \beta^{N(x,u)-i}(r(x,u) - r(x,u,w(i)))^2 \quad . \tag{5.1}$$

Similarly it can be shown that the value of $p(x'|x,u)$ computed minimizes

$$\sum_{i=1}^{N(x,u)} \beta^{N(x,u)-i}(p(x'|x,u) - I_{\{x'\}}(f(x,u,w(i))))^2 \quad . \tag{5.2}$$

If $\beta = 1$, the estimate of the MDP structure converges to the right MDP structure i.e., the $r(x,u)$ and $p(x'|x,u)$ terms estimated by the algorithm converge respectively to $\underset{w}{E}[r(x,u,w)]$ and $\underset{w}{E}[I_{\{x'\}}(f(x,u,w))]$ (equations (2.28) and(2.29)), if each state-action pair is visited an infinite number of times.

[4]Time-varying systems will not be encountered in this chapter but will in chapter 9.

[5]By “a visit of a state-action pair $(x,u)$”, we mean that the action $u$ has been selected while being in state $x$.

Figure 5.5: *Estimation of the MDP structure* : Stochastic Approximation algorithm

```
**Input :** $x_t$, $u_t$, $r_t$ and $x_{t+1}$
$r(x_t, u_t) \leftarrow r(x_t, u_t) + \alpha_r(r_t - r(x_t, u_t))$
Repeat for all $x \in X$
    $p(x|x_t, u_t) \leftarrow p(x|x_t, u_t) + \alpha_p(I_{\{x\}}(x_{\{t+1\}}) - p(x|x_t, u_t))$
```

In section A.3.2 we provide a deeper insight into the way the MDP structure is estimated by the Kalman Filter like algorithm and highlight notably that this estimation procedure is a particular case of the general iterative algorithm described in section A.1.

### 5.3.3 Stochastic Approximation algorithm

The Stochastic Approximation algorithm used to estimate the MDP structure is represented on figure 5.5. At each stage $t + 1$ of the control process, this algorithm uses the knowledge of $x_t$, $u_t$, $r_t$ and $x_{t+1}$ to refresh the MDP structure estimation.

For this algorithm to hold valid, one has to initialize the $r(x, u)$ and $p(x'|x, u)$ values $\forall x', x \in X$ and $\forall u \in U(x)$ at the beginning of the learning. These values can be initialized almost arbitrarily since they just have to satisfy two conditions : $0 \leq p(x'|x, u) \leq 1$ and $\sum_{x' \in X} p(x'|x, u) = 1$.

In order to have a first guess on these values that stands as close as possible to the solution, we will, when a state-action pair $(x_t, u_t)$ is met for the first time, initialize $r(x_t, u_t)$ to $r_t$ and $p(x|x_t, u_t)$ to $I_{\{x\}}(x_{t+1})$ $\forall x \in X$.

#### Convergence conditions

It can be shown that the MDP structure estimated by the Stochastic Approximation algorithm converges to the right MDP structure if each state-action pair $(x, u)$ is visited an infinite number of times and if the parameters $\alpha_{r_k}(x, u)$ ($\alpha_{r_k}(x, u)$ represents the value of $\alpha_r$ the $k$th time the state-action pair $(x, u)$ has been visited) and $\alpha_{p_k}(x,u)$ satisfy (appendix A) :

$$\sum_{k=1}^{\infty} \alpha_{\{r,p\}_k}(x,u) \quad = \quad \infty \tag{5.3}$$

$$\sum_{k=1}^{\infty} \alpha^2_{\{r,p\}_k}(x,u) \quad < \quad \infty \quad . \tag{5.4}$$

We remark that if we store during the learning for each state-action pair $(x,u)$ the value $N(x,u)$ that represents the number of times the action $u$ has been taken while being in state $x$ and if we take $\alpha_r$ and $\alpha_p$ equal to

$$(\frac{1}{N(x,u)})^{\theta} \tag{5.5}$$

with $0.5 < \theta \leq 1$ then equations (5.3) and (5.4) are satisfied. If $\theta = 1$, the Stochastic Approximation algorithm behaves like the Kalman Filter like algorithm with $\beta = 1$.
In section A.3.3 we provide a deeper insight into the way the MDP structure is estimated by the Stochastic Approximation algorithm and highlight notably that the estimation is based on the iterative algorithm described in section A.2.

**Obtaining the same estimation as with the Kalman Filter like algorithm**

Suppose that we keep in a set all the four-tuples $(x_t, u_t, r_t, x_{t+1})$ obtained during the learning. Suppose that each element of this set has the same probability to be selected. If we repeat an infinite number of times the sequence of operations that consists in selecting an element of the set and in using it as input of the Stochastic Approximation algorithm, we will converge to an estimate of the MDP structure that is identical to the one obtained by using a Kalman Filter like algorithm for which $\beta = 1$ (see section A.2.3 for more details).

### 5.3.4 Resolution of the MDP

The resolution of the MDP might be done by using the DP programming algorithms explained in chapter 2 (value iteration, policy iteration, $\cdots$). Each time the MDP structure is refreshed, one of these algorithms can be run in order to get a new estimate of the $Q$-function. But such a strategy has two main disadvantages :

- it requires the structure of the MDP to be completely initialized (the $p(x'|x,u)$ and $r(x,u)$ to be initialized $\forall x, x' \in X$ and $u \in U(x)$ at the beginning of the learning)

- it can lead to high computational burden since it implies solving the DP problem completely at each stage of the learning process.

In this section we describe some modifications to the Gauss-Seidel version of the value iteration algorithm (described in figure 2.4) in order to overcome these difficulties. Similar modifications could be done to the other DP algorithms but are not detailed here.

### Using the old values of the $Q$-function as the starting point of the DP algorithm

At each stage of the learning, the structure of the MDP is modified only slightly. It can seem reasonable to say that a minor change in the MDP structure induces only a minor change in the $Q$-function. In this way of thinking, rather than to initialize the $Q(x,u)$ values to arbitrary values at the beginning of the algorithm, we initialize them to the values of the $Q$-function previously computed in order to speed up the convergence of the DP algorithm.

### Pair $(x,u)$ not yet visited : no modification of $Q(x,u)$

The key part of the Gauss-Seidel version of the value iteration algorithm detailed in figure 2.4 is to realize DP iterations i.e., operations of the type :

$$Q(x,u) \leftarrow [r(x,u) + \gamma \sum_{x' \in X} p(x'|x,u) \max_{u' \in U(x')} Q(x',u')]. \tag{5.6}$$

A DP iteration with a state-action pair $(x,u)$ that has not yet been visited implies initializing $r(x,u)$ and $p(x'|x,u)\, \forall x' \in X$ to arbitrary values. We rather do not realize any DP iteration on a state-action pair that has not yet been visited and keep constant its corresponding $Q(x,u)$ value (the value $Q(x,u)$ was initialized to).

### Pair $(x',u')$ not yet visited : modification of the DP iteration

Some of the state-action pairs $(x',u')$ that intervene in the DP iteration (see equation (5.6)) may not have been visited yet and thus their $Q$ values are equal to the value to which they were initialized at the beginning of the learning process. These initial values can be quite distant from their real values which can, through equation (5.6), lead to a really bad estimate of the $Q$-function in some other states.
In most situations, this problem has no real impact on the speed of convergence. Nevertheless, a temporary wrong value of $Q$ can strongly influence, for some control problems, the way the reinforcement learning algorithm interacts with the system while choosing a greedy action. This will be illustrated in section 9.2.7 where

we will have to modify the DP iteration in order to obtain a good behavior of the reinforcement learning algorithm. An appropriate modification of the DP iteration consists in redefining it as follows :

$$Q(x,u) \leftarrow [r(x,u) + \gamma \sum_{x' \in X} p(x'|x,u) \max_{u' \in U'(x')} Q(x',u')] \tag{5.7}$$

where $U'(x') = U(x') \setminus \{u' \in U(x') | \, (x',u')$ has not yet been visited$\}$ if the state $x'$ has already been visited and $U'(x') = U(x')$ otherwise.

**Further reduction of computational burden**

Despite the modifications brought to the Gauss-Seidel version of the value iteration algorithm, the computational burden can still be too heavy especially if the RL algorithm interacts with a real system for which a time constraint on the computation duration is associated to each time interval $[t+1, t+2]$.
One strategy to alleviate the computational burden is to solve the MDP only partially, that is to do only a few DP iterations each time the MDP structure is refreshed. Such a strategy is similar to the one adopted by the *asynchronous fixed point iteration* algorithm explained in chapter 2. In this algorithm the DP iteration locations were chosen at random. We propose here a smarter approach to the choice of the state-action pairs on which to realize a DP iteration. It consists in executing first a DP iteration on the state-action pair $(x_t, u_t)$ for which $p(x'|x_t, u_t)$ and $r(x_t, u_t)$ have just been refreshed, then on its predecessors[6], then on the predecessors of the predecessors and so on. This strategy can be particularly efficient if the transition probability matrices are sparse.
The tabular version of such an algorithm that we have named *Prioritized Sweeping algorithm* (by analogy with the name of a model based reinforcement learning method that was using a similar algorithm to solve the MDP (see [MA93])) is represented on figure 5.6. On this figure, $iter_{max}$ indicates the maximum number of DP iterations that are allowed to be done each time the MDP structure is refreshed. On the other hand, $\sigma$ is a small value aimed to reject for further DP iteration the state-action pairs that precede a state whose $Q(x,u)$ value is deemed to have been too slightly modified to cause any significant changes in the $Q$ values of its predecessors.

[6]Predecessors of a state $x'$ (or of a state-action pair $(x',u')$) are meant to be the state-action pairs $(x,u)$ such that the next state observed after taking action $u$ in state $x$ has already been, during the learning process, $x'$.

Figure 5.6: *Resolution of the MDP :* Prioritized sweeping algorithm

```
**Input :** $x_t$ and $u_t$
Empty *Queue*
Put state-action pair $(x_t, u_t)$ in *Queue* and set $iter = 0$
Repeat until *Queue* is empty or $iter = iter_{max}$
    Take the state-action pair $(x, u)$ from the head of *Queue*
    $\delta \leftarrow r(x,u) + \gamma \sum_{x' \in X} p(x'|x,u) \max_{u' \in U(x')} Q(x',u') - Q(x,u)$
    $Q(x,u) \leftarrow Q(x,u) + \delta$
    $iter \leftarrow iter + 1$
    If $(\delta > \sigma)$ then put all state-action pairs that precede $x$ at the end of *Queue*
```

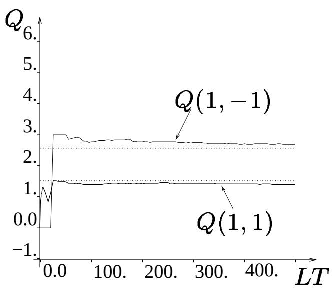
**Example 5.3.1** In this example we apply the model based reinforcement learning algorithms to the *four-state control problem* defined in example 2.3.1.

The policy used is an $\epsilon$-Greedy policy with $\epsilon = 0.1$ (*Action selection* module). The structure of the MDP is reconstructed by using the Kalman Filter like algorithm (*Estimation of the MDP structure* module) and the estimated MDP is solved at each stage of the control process by using the Gauss-Seidel version of value iteration algorithm (*Resolution of the MDP* module). An episode ends only when the terminal state $x^t$ is reached. The initial state of each episode is chosen at random among $X \setminus \{x^t\}$ and the $Q$-function is initialized to 0 everywhere at the beginning of the learning.

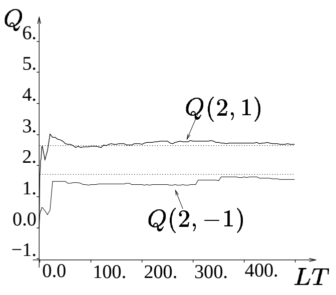
In such conditions, figures 5.7a, 5.7b and 5.7c represent the evolution of the $Q$-function estimated. The symbol $LT$ is used for Learning Time that is measured in number of episodes[7]. The dotted lines represent the value of the exact $Q$-function, that is the $Q$-function computed while knowing the system dynamics and the reward function. We can see that the estimate of the $Q$-function converges to its exact value. Note that the number of episodes necessary to get an estimate of the $Q$-function that gives by equation (2.13) a correct estimate of the optimal stationary policy is small. Indeed this occurs when $Q(1,-1) > Q(1,1)$, $Q(2,-1) < Q(2,1)$ and $Q(3,-1) < Q(3,1)$ which happens after

[7]The number of episodes is not necessarily the most adequate measure of the learning time. Indeed the length of an episode depends on the policy which itself depends here on the $Q$-function estimated at each instant and therefore also on the *Learning* module used. A better measure would have been the number of transitions from $t$ to $t+1$. However in the RL literature the learning time is often characterized in terms of episodes.

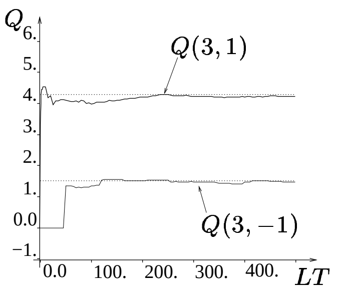
only 22 episodes.



(a) State 1



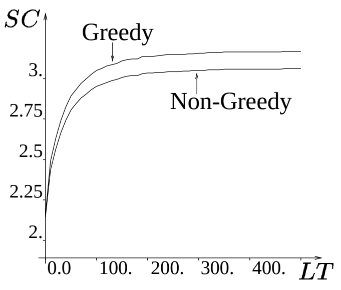
(b) State 1



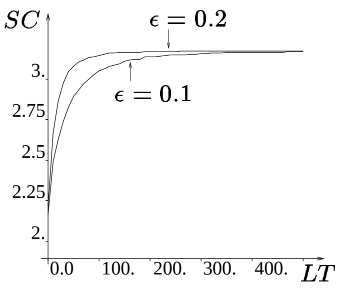
(c) State 3

Figure 5.7: $Q$-function estimated during the learning process

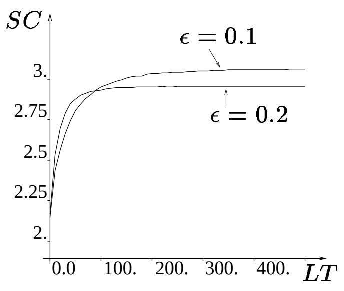
If we compute the score (see equation (3.26) with $P_0(1) = \frac{1}{3}$, $P_0(2) = \frac{1}{3}$, $P_0(3) = \frac{1}{3}$ and $P_0(x^t) = 0$) obtained by evaluating after each episode the estimate of the optimal policy obtained from the estimate of the $Q$-function, we get the curve labeled "Greedy", represented on figure 5.8a[8]. The "Non-Greedy" curve represents the score corresponding to the $\epsilon$-Greedy policy. This curve illustrates better the return observed during the learning process. The larger the value of $\epsilon$, the larger the difference between these two curves will be.



(a) $\epsilon = 0.1$



(b) Greedy



(c) Non-Greedy

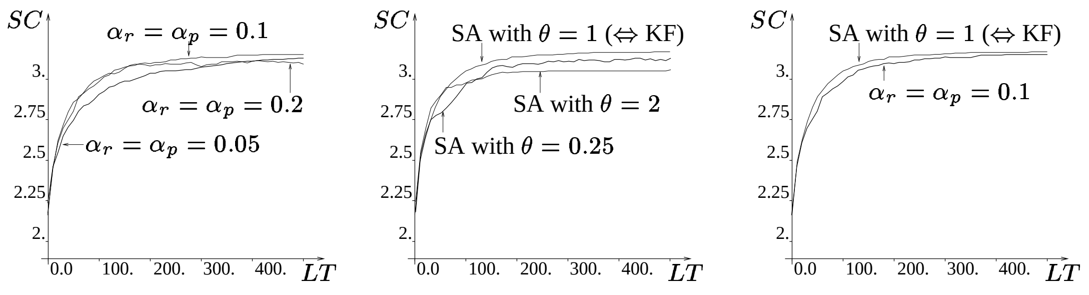
Figure 5.8: Exploration-exploitation tradeoff

We can also use the four-state control problem to illustrate the *exploration-exploitation* tradeoff. Suppose that we have another reinforcement learning algorithm, similar to the

[8]Note that due to the fact that the scores computed can have a high variance (the initial states are chosen at random, the action taken while being in $x$ is chosen randomly the first time $x$ is met, the use of an $\epsilon$-Greedy policy,$\cdots$) the learning process has been repeated 200 times and the scores obtained have been averaged.

one just used in this example ($\epsilon$-Greedy policy, Kalman Filter like algorithm, value iteration algorithm, same initialization of the $Q$-function), except that the $\epsilon$ factor of the $\epsilon$-Greedy policy is now equal to $0.2$. The greedy scores (which assess the goodness of the learned policy) computed for the two reinforcement learning algorithms are represented on figure 5.8b. The larger the value of $\epsilon$ (i.e. the more exploration), the better the score of the policy computed. By the end of the learning process, the two curves meet since that both reinforcement learning algorithms have learned a policy that coincides with the optimal policy. Figure 5.8c represents the non-greedy scores (which assess the goodness of the return obtained during the learning process since they correspond to the $\epsilon$-Greedy policies used during the learning) computed for the two reinforcement learning algorithms. At the beginning of the learning, the larger the exploration factor $\epsilon$, the larger the non-greedy score obtained. But after less than one hundred episodes the tendency is inversed and the tradeoff between exploration and exploitation appears.

On figure 5.9a we have represented the score curves obtained while using this time a Stochastic Approximation algorithm in order to estimate the MDP structure. The coefficients $\alpha_r$ and $\alpha_p$ that represent the amount of correction done at each iteration have been chosen constant. With such a choice, equation (5.4) is not satisfied anymore, and this prevents us from stating anything about the MDP structure that will be estimated by the algorithm. We can observe that with a choice of $\alpha_r = \alpha_p = 0.05$, the learning of the structure is slow but the result obtained after $500$ episodes is quite good. With $\alpha_r$ and $\alpha_p$ chosen equal to $0.2$, the score is perhaps better at the beginning but as the learning process proceeds, this advantage fades in favor of Stochastic Approximation algorithms using smaller values of $\alpha_r$ and $\alpha_p$. Among the three variants represented, the choice $\alpha_r = \alpha_p = 0.1$ seems to realize at best the compromise between the learning speed and the quality of the estimate obtained at the end of the learning period.



(a) $\alpha_r$ and $\alpha_p$ constant (b) $\alpha_r$ and $\alpha_p$ non constant (c) Constant vs non constant

Figure 5.9: Score curves for different variants of the *Learning* module.

On figure 5.9b, score curves are represented when the parameters $\alpha_r$ and $\alpha_p$ used in the Stochastic Approximation algorithm satisfy an equation of the type (5.5). As aforementioned, if $\theta$ is chosen equal to 1 in this equation, the Stochastic Approximation algorithm is equivalent to the Kalman Filter like algorithm used with $\beta = 1$. Observe that when $\theta$

Figure 5.10: *Learning* : non-model based technique

---

*$Q$-function correction*

---

is chosen equal to $2$, bad results are obtained by the learning. Indeed if $\theta$ is larger than $1$, equation (5.3) is not satisfied anymore : the decrease rate of $\alpha_r$ and $\alpha_p$ is too fast to allow for a right estimate of the MDP structure. The value of $\theta = 0.25$ leads to the violation of equation (5.4) : $\alpha_r$ and $\alpha_p$ decrease too slowly to ensure the convergence. It must be noticed that the corresponding score curve is less attractive than the one obtained by using the Kalman Filter like algorithm.

On figure 5.9c, we have compared the two strategies, $\alpha_r$ and $\alpha_p$ constant vs non constant; observe that the best result is obtained when $\alpha_r$ and $\alpha_p$ satisfy indeed equations (5.3) and (5.4).

## 5.4 Learning : non-model based techniques

### 5.4.1 Introduction

Contrary to the model based methods, non-model based methods directly learn the value of the $Q$-function of the system without reconstructing any model of the control problem. The type of non-model based method we use in this work belongs to the *temporal difference* class of methods. With respect to other classes of non-model based methods[9], its two main characteristics are to correct the value of the $Q$-function at each stage $t+1$ of the control process and to base the correction of $Q(x,u)$ not only on the observations done but also on the value of the $Q$-function for other state-action pairs than $(x,u)$. The *Learning* module for this type of non-model based methods is sketched on figure 5.10 and reduced to a single purpose : a *$Q$-function correction*. In this section we describe two types of *$Q$-function correction* modules. One, known as $Q$-learning, corrects at time $t+1$ only the value of $Q(x_t,u_t)$ while the other, known as $Q(\lambda)$, performs corrections at time $t+1$ not only on $Q(x_t,u_t)$ but also on $Q(x_{t-1},u_{t-1})$, $Q(x_{t-2},u_{t-2})$, $Q(x_{t-3},u_{t-3})$, $\cdots$.

Figure 5.11: *$Q$-function correction* : $Q$-learning algorithm

```
**Input :** $x_t$, $u_t$, $r_t$ and $x_{t+1}$
$\delta \leftarrow (r_t + \gamma \max_{u \in U(x_{t+1})} Q(x_{t+1}, u)) - Q(x_t, u_t)$
$Q(x_t, u_t) \leftarrow Q(x_t, u_t) + \alpha\delta$
```

### 5.4.2 $Q$-learning

The $Q$-learning algorithm was first introduced in [Wat89] and is sketched on figure 5.11. At time $t+1$, it uses the four-tuple $(x_t, u_t, r_t, x_{t+1})$ as input and updates $Q(x_t, u_t)$. The $Q(x_t, u_t)$ update is done through the computation of the *temporal difference*

$$\delta(x_t, u_t) \quad = \quad (r_t + \gamma \max_{u \in U(x_{t+1})} Q(x_{t+1}, u)) - Q(x_t, u_t) \tag{5.8}$$

that depends on the four-tuple $(x_t, u_t, r_t, x_{t+1})$ and on the $Q$ values in state $x_t$ and $x_{t+1}$. The temporal difference $\delta$ represents the difference between an estimate

$$r(x_t, u_t, w_t) + \gamma \max_{u \in U(x_{t+1})} Q(x_{t+1}, u)$$

of $Q(x_t, u_t)$ based on the observations in $t+1$, and the current estimate. In this sense the temporal difference provides an indication as whether the current estimate should be raised or lowered.
Once the temporal difference $\delta$ is computed, the current estimate of $Q(x_t, u_t)$ is refreshed by adding $\alpha\delta$ ($\alpha \geq 0$), $\alpha$ being the stepsize used for updating $Q(x_t, u_t)$. Let $\alpha_k$ be the value of $\alpha$ at the $k$th stage of the algorithm (i.e. the $k+1$th time the $Q$-function is updated), and let us define $\alpha_k(x, u)$ :

- as being equal to $\alpha_k$ if at the $k$th stage of the algorithm, $(x, u)$ coincides with $(x_t, u_t)$ (i.e. if $Q(x, u)$ is updated at the $k$th stage of the algorithm)
- as being equal to 0 if at the $k$th stage of the algorithm, $(x, u)$ does not coincide with $(x_t, u_t)$.

[9]There exists another class of non-model based reinforcement learning methods known as Monte-Carlo methods that update the value of $Q(x, u)$ only at the end of an episode without using the estimate of the $Q$-function for other state-action pairs than $(x, u)$. These methods are not used in this work; for more information the reader may refer to [Rub81], [SS96] or [BD94].

The convergence of the $Q$-learning algorithm (proved in [WD92]) occurs if the $\alpha_k(x,u)$ are nonnegative and satisfy (for every state-action pair $(x,u)$) :

$$\sum_{k=0}^{\infty} \alpha_k(x,u) \to \infty, \tag{5.9}$$

$$\sum_{k=0}^{\infty} \alpha_k^2(x,u) < \infty. \tag{5.10}$$

The proof is detailed in section C.5. Note that satisfaction of expressions (5.9) and (5.10) implies that every state-action pair has to be visited an infinite number of times.

Remark that in order to use this algorithm, one has to initialize the $Q$-function at the beginning of the learning process. Note also that if the system is deterministic and the $Q$-function correctly estimated then the temporal difference $\delta$ would always be equal to zero.

If $\#U(x) = 1, \forall x \in X \setminus \{x^t\}$, that is if the $Q$-learning algorithm is used to evaluate a stationary policy $\mu$, then the $Q$-learning algorithm is referred to as the TD(0) algorithm.

One of the main difficulties encountered with the $Q$-learning algorithm is linked to the $\alpha_k(x,u)$ values choice. Indeed, even if it is easy to choose these values such that conditions (5.9) and (5.10) are satisfied, one observes that when they decrease too fast (while still satisfying conditions (5.9) and (5.10)), unaffordable learning times may be required to converge to a "good" solution. This is why the $\alpha$ value used inside the $Q$-learning algorithm is often chosen constant in examples treated in the literature.

### 5.4.3 $Q(\lambda)$

#### $Q(\lambda)$ algorithm : principle

The $Q$-learning algorithm corrects at time $t+1$ only $Q(x_t,u_t)$, leaving unchanged the other components of the estimated $Q$-function. On the other hand a model based algorithm refreshes at time $t+1$ the estimates of $r(x_t,u_t)$ and $p(.|x_t,u_t)$ and then solves (at least partially) the MDP structure which may lead to a modification of the $Q$-function not only for $(x_t,u_t)$, but also for the predecessors of $(x_t,u_t)$, the predecessors of the predecessors, $\cdots$. The $Q(\lambda)$ algorithm has an intermediate behavior since it corrects at time $t+1$ the $Q$-function not only for $(x_t,u_t)$ but also for $(x_{t-1},u_{t-1})$, $(x_{t-2},u_{t-2})$, $\cdots$. The corrections are realized by backpropagating at each $t+1$ the temporal-difference factor $\delta(x_t,u_t)$ along the system trajectory.

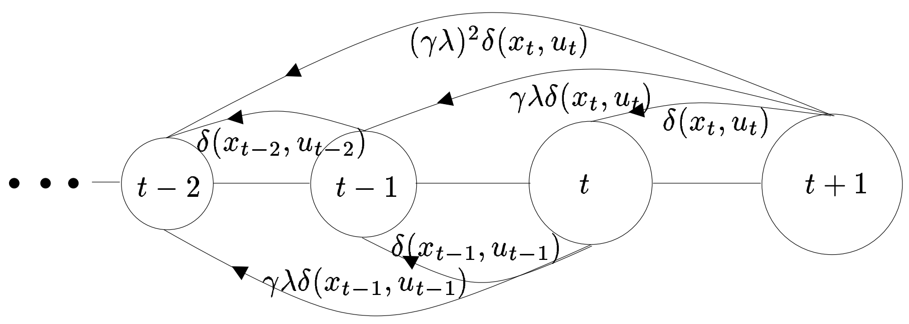

$(\gamma\lambda)^2\delta(x_t,u_t)$
$\gamma\lambda\delta(x_t,u_t)$
$\delta(x_t,u_t)$
$\delta(x_{t-2},u_{t-2})$
$\bullet\bullet\bullet$
$t-2$
$t-1$
$t$
$t+1$
$\delta(x_{t-1},u_{t-1})$
$\gamma\lambda\delta(x_{t-1},u_{t-1})$

Figure 5.12: $Q(\lambda)$ : $Q$-function correction

**Intuitive sketch of Q$(\lambda)$ based on Q-learning**

Let us first recall that the $Q$-learning algorithm consists in computing at time $t+1$ of the control process a factor $\delta(x_t,u_t)$ defined by equation (5.8) and to refresh the estimate of $Q(x_t,u_t)$ by adding the value of $\alpha\delta(x_t,u_t)$.

Suppose now that the stepsize $\alpha$ is chosen equal to 1, and suppose that we already know at time 1, 2, $\cdots$, $t$ what the value of $\delta(x_t,u_t)$ will be. Suppose also that if $Q(x_i,u_i)$ is increased by a certain value, then $\max\limits_{u\in U(x_i)} Q(x_i,u)$ increases by the same value.

Under these assumptions one could modify the value of $\delta(x_{t-1},u_{t-1})$ at time $t$ of the control process by adding a factor $\gamma\delta(x_t,u_t)$ in order to anticipate the variation that will occur in $t+1$ on $Q(x_t,u_t)$. And by proceeding recursively, we can see that a way of anticipating this variation on $Q(x_t,u_t)$ is to add to each $\delta(x_{t-j},u_{t-j})$ a factor $\gamma^j\delta(x_t,u_t)$ ($j\in\{1,\cdots,t\}$).

Such an intuitive reasoning leads to the $Q(\lambda)$ algorithm which adopts a backward view in order to compensate for the impossibility to predict at time $t-j+1$ ($\forall j\in\{1,\cdots,t\}$) the value of $\delta(x_t,u_t)$. This algorithm is sketched on figure 5.12.

Hence, the $Q(\lambda)$ algorithm corrects at time $t+1$ not only $Q(x_t,u_t)$ by a factor $\alpha\delta(x_t,u_t)$, but also the $Q$ values of the preceding state-action pairs $Q(x_{t-j},u_{t-j})$ by a factor $\alpha(\gamma\lambda)^j\delta(x_t,u_t)$ ($j\in\{1,\cdots,t\}$) taking into account both the discount factor $\gamma$ and a trace-decay factor $\lambda$.

The *trace-decay parameter* $\lambda$ ($0\leq\lambda\leq 1$) has been introduced in order to possibly down-weight the amount of correction done on the state-action pairs preceding $(x_t,u_t)$, but it is usually chosen close or equal to 1. Note that if $\lambda=0$ then the $Q(\lambda)$ algorithm is identical to $Q$-learning.

### Watkins's $Q(\lambda)$ algorithm

There exist many variants of the $Q(\lambda)$ algorithm. One of them is known as Watkins's $Q(\lambda)$ algorithm whose tabular version is represented in figure 5.13. The function $e(x,u)$ stores what is known as the *eligibility trace*. The higher the value of the eligibility trace for a state-action pair, the greater the amount of correction realized on its corresponding $Q$ value. The particularity of this Watkins's version of the $Q(\lambda)$ algorithm is that it sets the eligibility trace to 0 each time a non-greedy action is taken. The reason for this is that when a non-greedy action is taken in $t+1$ one cannot suppose anymore that

> *"if $Q(x_{t+1},u_{t+1})$ is increased by a certain value, then $\max\limits_{u\in U(x_{t+1})} Q(x_{t+1},u)$ is also increased by the same value"*

which was assumed to motivate the corrections that the $Q(\lambda)$ algorithm does along the trajectory.
Concerning the initialization of the algorithm depicted in figure 5.13, the value of the $Q$-function must be initialized at the beginning of the learning and $e(x,u)$ must be set equal to 0 $\forall (x,u) \in X \times U$ at the beginning of each episode.

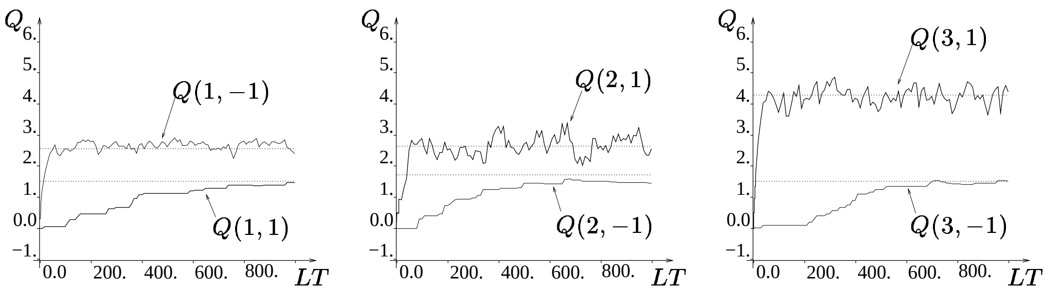
Figure 5.13: *$Q$-function correction* : Watkins's $Q(\lambda)$ algorithm

```
Input : $x_t$, $u_t$, $r_t$, $x_{t+1}$ and $u_{t+1}$
$\delta \leftarrow (r_t + \gamma \max\limits_{u\in U(x_{t+1})} Q(x_{t+1},u)) - Q(x_t,u_t)$
$e(x_t,u_t) \leftarrow e(x_t,u_t) + 1$
Do $\forall (x,u) \in X \times U$
    $Q(x,u) \leftarrow Q(x,u) + \alpha\delta e(x,u)$
    If $Q(x_{t+1},u_{t+1}) = \max\limits_{u'\in U(x_{t+1})} Q(x_{t+1},u')$ then $e(x,u) \leftarrow \gamma\lambda e(x,u)$, else
    $e(x,u) = 0$
```

### Convergence issues

Convergence of the $Q(\lambda)$ algorithm is not guaranteed except if $\#U(x) = 1 \forall x \in X \setminus \{x^t\}$ and if $\alpha_k$ is satisfying expressions (5.9) and (5.10). In this particular case the control policy is given beforehand, and the algorithm is known as TD($\lambda$) [JJS94].

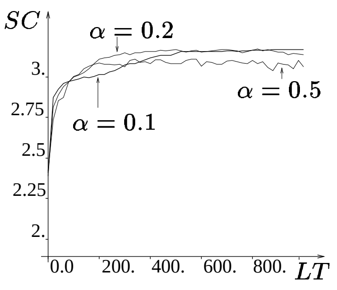
**Example 5.4.1** We illustrate in this example an application of a non-model based reinforcement learning algorithm on the *four-state control problem*. Model based algorithms were also applied to this control problem in example 5.3.1.



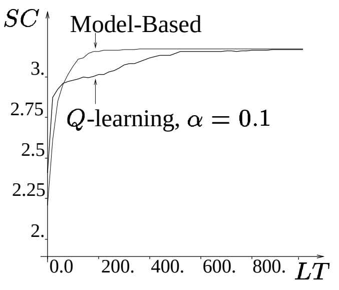
(a) State 1 (b) State 2 (c) State 3

Figure 5.14: $Q$-function estimated during the learning process

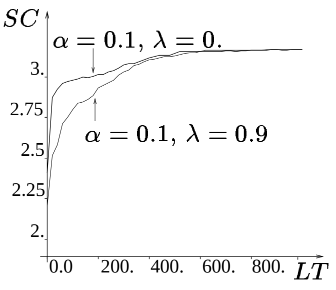
The policy used is an $\epsilon$-Greedy policy (*Action selection* module) and the algorithm used to learn the $Q$-function is the $Q$-learning algorithm (*Q-function correction* module). The value of the parameter $\alpha$ is chosen constant and equal to $0.1$. At the beginning of the learning we initialize the value of the $Q$-function to zero everywhere. An episode ends only when a terminal state $x^t$ is reached and the initial state of each episode is chosen at random in $X \setminus \{x^t\}$. Under such conditions, figures 5.14a, 5.14b and 5.14c represent the $Q$-function after each episode of the learning process. Because $\alpha$ does not satisfy equation (5.10), the values of the $Q(x,u)$ terms won't converge to their exact values represented by the dotted lines on the figures. Even if the $Q$-function does not converge to the exact value, the estimate of the optimal policy that we can deduce from it is correct after only $5$ learning episodes. Indeed after $5$ episodes we have already obtained $Q(1,-1) > Q(1,1)$, $Q(2,-1) < Q(2,1)$ and $Q(3,-1) < Q(3,1)$.
Figure 5.15a represents the score evolution (computed in the same conditions as in example 5.3.1) of the estimate of the optimal policy for different values of $\alpha$. At the end of the learning (which occurs after $1000$ episodes), the best score is obtained for the smallest value of $\alpha$, that is the value which "violates" the least equation (5.10). Nevertheless a too small $\alpha$ value could slow down the learning process even if eventually it offers better results.
Figure 5.15b compares the performances of a model based algorithm ($\epsilon$-Greedy policy with $\epsilon = 0.1$, Kalman Filter like algorithm, Gauss-Seidel value iteration) and a non-model based algorithm ($\epsilon$-Greedy policy with $\epsilon = 0.1$, $Q$-learning, $\alpha = 0.1$). We obtain better performances for the model based algorithm except at the very beginning of the learning. In fact, we will see throughout this work that model based algorithms offer consistently better



(a) Different values of $\alpha$



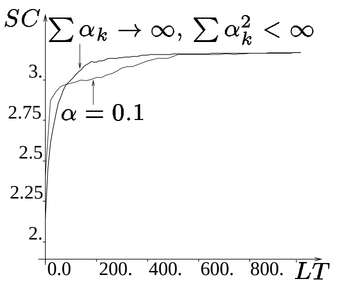
(b) MB vs NMB



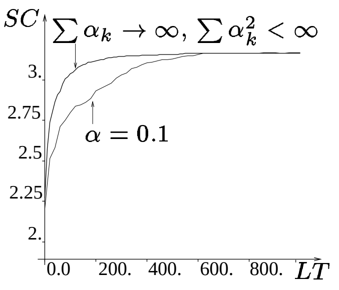
(c) $\lambda = 0$ vs $\lambda = 0.9$

Figure 5.15: Four-state control problem : score curves

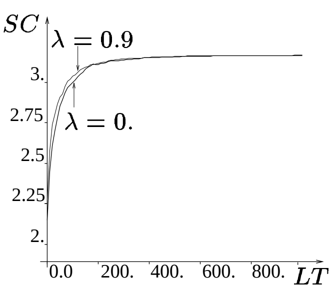
performances (for a comparison between the performances of model based and non-model based algorithms see [AS97] ).
Figure 5.15c compares the performances of the Watkins's $Q(\lambda)$ algorithm ($\alpha = 0.1$, $\lambda = 0.9$) and the $Q$-learning algorithm ($\alpha = 0.1$). The $Q$-learning score curve being above the Watkins's $Q(\lambda)$ score curve, we can conclude that the use of the eligibility trace did not speed up the learning.



(a) $\lambda = 0$



(b) $\lambda = 0.9$



(c) $\lambda = 0$ vs $\lambda = 0.9$

Figure 5.16: The $\alpha_k$ terms satisfy the convergence conditions

So far $\alpha$ has been chosen constant. One may wonder whether the results improve significantly when conditions (5.9) and (5.10) are satisfied. If in the $Q$-learning algorithm we choose $\alpha = \frac{1}{N(x_t, u_t)}$ ($N(x_t, u_t)$ representing the number of times the state-action pair $(x_t, u_t)$ has been visited) we obtain the score curve sketched on figure 5.16a and labeled "$\sum \alpha_k \to \infty$, $\sum \alpha_k^2 < \infty$". On the same graphic we have also drawn the score curve corresponding to $\alpha$ constant. As we can observe, the results are better (except perhaps at the very beginning of the learning) when conditions (5.9) and (5.10) are indeed satisfied. Moreover, when these two conditions are satisfied the performances of the Watkin's $Q(\lambda)$ algorithm strongly increase (figure 5.16b) and, contrary to what was observed when $\alpha$ was chosen constant, the use of the eligibility trace does not slow down the learning anymore (figure 5.16c).

## 5.5 Learning modules : a unified view

Suppose that we have kept trace of all the four-tuples $(x_t, u_t, r_t, x_{t+1})$ obtained while interacting (in some arbitrary way) with the system. Suppose also that each state-action pair has been visited at least once during this interaction.
We propose three different ways of computing from this trace an estimate of the $Q$-function.

1. Each element of the trace is used a single time as input to the Kalman Filter like model updating algorithm (figure 5.4). After that, the $Q$-function is estimated by solving the MDP structure thus computed.

2. We repeat an "infinite" number of times the sequence of operations, which consists first in selecting at random an element of the trace and then in using it as input to the Stochastic Approximation algorithm (figure 5.5). The $Q$-function is estimated by solving the MDP structure thus computed.

3. Suppose that each element of the trace has the same probability to be selected. We repeat an infinite number of times the sequence of operations, which consists first in selecting an element of the trace and then in using it as input of the $Q$-learning algorithm (figure 5.11).

We claim that the $Q$-function estimated by each of these three procedures is the same. We already mentioned previously that under such conditions the Kalman Filter like algorithm and the Stochastic Approximation algorithm led to the same estimate of the MDP structure. Thanks to the DP equation solution uniqueness, the $Q$-function estimated by the two corresponding ways will also be the same.
It is straightforward to see that the $Q$-learning algorithm will also lead to the same estimate of the $Q$-function. Indeed, when an element $(x_t, u_t, r_t, x_{t+1})$ of the trace is used as input of this algorithm, one can consider :

- $r_t$ as being drawn independently from a probability distribution having as average the value of $r(x_t, u_t)$ estimated by the Kalman Filter like algorithm

- $x_{t+1}$ as being drawn independently according to the probabilities $p(.|x_t, u_t)$ estimated by the Kalman Filter like algorithm and everything happens for the $Q$-learning algorithm as if it were interacting with a finite state space control problem having as MDP structure the one estimated by the Kalman Filter like algorithm. The convergence property of $Q$-learning implies that the $Q$-function obtained corresponds indeed to this MDP structure.

## 5.6 Summary

*In this chapter we have described several reinforcement learning methods that allow resolving discrete-time optimal control problems for which the system dynamics and the reward function are unknown. To compensate for the absence of knowledge of the system dynamics and the reward function, the reinforcement learning algorithms control the system and observe at each stage of the control process the state of the system and the reward obtained.*

*We have first discussed what control strategy these algorithms should use in order to ensure that the information learned about the control problem is already well exploited but without preventing further learning (exploration-exploitation tradeoff). Then we have presented two families of methods that are able to solve, by learning, the control problem. One (the model based family) consists in reconstructing the MDP structure that corresponds to the control problem and in solving this MDP by using classical DP algorithms. The other (the non-model based family) learns directly the $Q$-function of the control problem (or equivalently the $Q$-function of the MDP that corresponds to the control problem). For these two families of methods we have described several types of algorithms and discussed their convergence to the exact solution.*

*All these algorithms hold valid only if the state space is finite. Next chapter is devoted to the description of reinforcement learning algorithms in infinite state spaces. For the model based algorithms the procedure will consist in learning the structure of a MDP$^\delta$ (the Markov Decision Process that approximates the initial control problem (along the ideas of chapter 3)) while the non-model based algorithms will directly learn the $Q^\delta$-function of a MDP$^\delta$ without explicitly reconstructing its structure.*

# Chapter 6

# Reinforcement learning in infinite state spaces

*In this chapter we introduce "Learning" modules able to act when the state space is infinite, by combining the two MDP$^{\delta}$ approximation schemes introduced in chapter 3 with the four reinforcement learning algorithms of chapter 5. This combination leads to two categories of "Learning" modules. One will reconstruct a MDP$^{\delta}$ structure and solve it in order to compute a $Q^{\delta}$-function (model based technique) while the other will learn directly a $Q^{\delta}$-function (non-model based technique). We describe the eight algorithms and characterize their convergence conditions in order to highlight their main properties and relationships.*

*This rather lengthy and technical chapter is the core of the theoretical part of this thesis. We have organized it in three parts, so as to make its reading a little easier. The first part (sections 6.1 and 6.2) provides a synthetic overview of the main principles that are applied in order to formulate algorithms and characterize their convergence properties. The second part (sections 6.3 to 6.5) concern the detailed derivation of the learning algorithms in the context of the aggregation technique, and the third part (sections 6.6 and 6.7) does a similar job with respect to the representative states technique. Some of the related mathematical details and convergence proofs are collated in appendices A and C.*

*The described algorithms are illustrated in the present chapter on an academic problem. In chapter 7 we provide an empirical comparison of most of these combinations in the context of a real-world though simple FACTS control problem already introduced in chapter 4.*

Figure 6.1: *Learning* : non-model based technique

```
$Q^\delta$-function correction
```

Figure 6.2: *Learning* : model based technique

```
Estimation of the MDP$^\delta$ structure
Resolution of the MDP$^\delta$
```

## 6.1 Overview

In chapter 3 two families of techniques were introduced to compute approximate solutions for infinite state space control problems, namely *the aggregation technique* and *the representative states technique*. In the following sections we discuss how to design the *Learning* modules described in figure 6.1 and 6.2 for each one of these techniques.
The design consists in keeping the same approximation architectures as those used in chapter 3 (section 3.5) in order to compute the discrete $Q^\delta$-function from interaction with the system, and extend it to the infinite state space. Before discussing in detail the different schemes, we first provide a quick overview of their main ideas.

### 6.1.1 Model based techniques : general scheme

In this section we make some preliminary comments concerning the model based algorithms that will be discussed in detail in sections 6.3 and 6.6.

#### MDP$^\delta$ structure reconstruction

Model based algorithms aim at reconstructing an MDP$^\delta$ structure such that the $\tilde{Q}$ -function deduced from the MDP$^\delta$ solution is able to approximate well the $Q$-function of the original control problem.
When the system dynamics and the reward function were known (chapter 3), we presented a way to compute the MDP$^\delta$ structure (equations (3.1) and (3.3) for the aggregation technique and equations (3.14) and (3.15) for the representative states technique. We explained in sections 3.2.4 and 3.3.4 that this way consisted in

adopting an MDP$^\delta$ structure that “violates the least” the equivalence conditions between the MDP$^\delta$ and the initial control problem.
The procedure we will use in this chapter to design the model based algorithms follows the same philosophy. The MDP$^\delta$ structure will be refreshed each time new observations are available in order to “satisfy at best” the equivalence conditions between the MDP$^\delta$ and the original control problem.

**Resolution of the MDP$^\delta$**

Concerning the *Resolution of the MDP$^\delta$* modules that are part of the model based *Learning* modules, we will not describe them in this chapter, since the considerations of section 5.3.4 are still applicable here. Indeed, the only difference is that here it is the MDP$^\delta$ rather than the MDP representing the initial control problem which is solved iteratively and repeatedly during the reinforcement learning protocol.
The only remark that must perhaps complement section 5.3.4, in the present context, concerns the Prioritized sweeping algorithm described on figure 5.6. Recall that this algorithm takes as input the state-action pair $(x, u)$ for which the values of $p(x'|x, u)$ and $r(x, u)$ have been refreshed by the *Estimation of the MDP structure* module at the current iteration, and introduces it into the *Queue* at the beginning of the algorithm. In the context of the representative states technique, this needs to be slightly modified since at each iteration several state-action pairs of the MDP$^\delta$ are refreshed by the *Estimation of the MDP$^\delta$ structure* module. To adapt the Prioritized sweeping algorithm to this observation we simply use as input of the algorithm all the state-action pairs $(x^\delta, u)$ for which the values of $r^\delta(x^\delta, u)$ and $p^\delta(x^{\delta\prime}|x^\delta, u)$ have been refreshed at the current iteration, and put all these state-action pairs on the *Queue*.

### 6.1.2 Non-model based techniques : general scheme

Below we explain the generic procedures we will follow to design non-model based algorithms (in section 6.4 for the aggregation technique, and in section 6.7 for the representative states techniques).
This procedure is inspired from the $Q$-learning algorithm and the $Q(\lambda)$ algorithm introduced in chapter 5 for a tabular representation of the $Q$-function. Here we view the approximate $\tilde{Q}$-function induced by either of the two approximation schemes as a function depending on a finite dimensional parameter vector $a$. In our context, this parameter vector corresponds to the values of the $Q^\delta$-function of the finite MDP$^\delta$ implicitly used by these schemes. The function $\tilde{Q}(x, u, a)$ is itself defined

by the way the $Q^\delta$-function is extrapolated over the original state space (section 3.5).

## $Q$-learning and advantage updating [BT96] [SB98]

Recall the form of the $Q$-learning algorithm in the case of a finite state space (see section 5.4.2) : at time $t+1$ the observed four-tuple $(x_t, u_t, r_t, x_{t+1})$[1] is used to compute the temporal difference

$$\delta(x_t, u_t) \quad = \quad r_t + \gamma \max_{u \in U(x_{t+1})} Q(x_{t+1}, u) - Q(x_t, u_t), \tag{6.1}$$

and to update the value of the tabular function $Q(\cdot, \cdot)$ at the last visited state-action pair, according to :

$$Q(x_t, u_t) \leftarrow Q(x_t, u_t) + \alpha \delta(x_t, u_t). \tag{6.2}$$

We remind that this algorithm converges to the optimal $Q$-function under appropriate conditions on the stepsizes $\alpha$ (equations (5.9) and (5.10)).
Let us extend this algorithm to the case where a parametric $Q$-function of the form $\tilde{Q}(x, u, a)$ is used. Equation (6.2) provides us with a desired update for $\tilde{Q}(x_t, u_t, a)$. Since the only way to change $\tilde{Q}$ is to change the parameter vector $a$, we compute the gradient $\frac{\partial \tilde{Q}(x_t, u_t, a)}{\partial a}$ of this function with respect to the parameter vector. Indeed, this latter indicates in which way $a$ should be updated so that $\tilde{Q}(x, u, a)$ moves in the desired direction. Based on this argument, the following iteration is suggested :

$$a \leftarrow a + \alpha \delta(x_t, u_t) \frac{\partial \tilde{Q}(x_t, u_t, a)}{\partial a}. \tag{6.3}$$

This iteration is known as *advantage updating* and algorithms based on it are frequently used. Nevertheless, to our knowledge theoretical justifications of these algorithms and convergence results are quite limited.
Notice that the particular case of a tabular $Q$-function would correspond to a parameter vector with as many components as entries in the table, i.e. one component per state-action pair. Moreover, in that case, equation (6.3) actually degenerates into equation (6.2).

[1]where $w_t$ is drawn independently from $P_w(.|x_t, u_t)$, and $r_t = r(x_t, u_t, w_t)$ and $x_{t+1} = f(x_t, u_t, w_t)$

### $Q(\lambda)$ **and advantage updating [BT96] [SB98]**

The extension uses the same philosophy, but adapted to the $Q(\lambda)$ algorithm. The latter uses the four-tuple $(x_t, u_t, r_t, x_{t+1})$ to compute the temporal difference and to update $Q(x_{t-j}, u_{t-j})$ ($\forall j \in \{0, \cdots, t\}$) according to :

$$Q(x_{t-j}, u_{t-j}) \quad \leftarrow \quad Q(x_{t-j}, u_{t-j}) + \alpha\delta(x_t, u_t)(\gamma\lambda)^j. \tag{6.4}$$

With an approximate $Q$-function of the form $\tilde{Q}(x, u, a)$ we could then compute $\frac{\partial\tilde{Q}(x_{t-j}, u_{t-j}, a)}{\partial a}$ for each value of $j$, and use these gradients to update $a$ in the following way :

$$a \quad \leftarrow \quad a + \alpha\delta(x_t, u_t)\sum_{j=0}^{t}\frac{\partial\tilde{Q}(x_{t-j}, u_{t-j}, a)}{\partial a}(\gamma\lambda)^j. \tag{6.5}$$

We observe that this method requires computing at an iteration at time $t+1$, $t+1$ times an expression of the type $\frac{\partial\tilde{Q}(x,u,a)}{\partial a}$. One strategy commonly used to avoid this is to replace the value of $\frac{\partial\tilde{Q}(x_{t-j}, u_{t-j}, a)}{\partial a}$ ($j \neq 0$) in the iteration by its value computed at time[2] $t - j + 1$ . This leads to the following algorithm :

$$\theta \quad \leftarrow \quad (\gamma\lambda)\theta + \frac{\partial\tilde{Q}(x_t, u_t, a)}{\partial a} \tag{6.6}$$

$$a \quad \leftarrow \quad a + \alpha\theta(r_t + \gamma \max_{u \in U(x_{t+1})} \tilde{Q}(x_{t+1}, u, a) - \tilde{Q}(x_t, u_t, a)) \tag{6.7}$$

where $\theta$ is a vector of the same dimensionality as the parameter vector $a$ which plays the role of the eligibility trace used in the tabular version of the $Q(\lambda)$ algorithm. Note that $\theta$ is re-initialized to 0 at the beginning of each learning episode.

### **Bellman error methods and RL**

When the system dynamics and the reward function are known, one strategy commonly used to compute the parameters of the approximation architecture consists in minimizing the error in the Bellman equation. This approach known as Bellman error methods has been briefly discussed in section 3.5.
When the system dynamics and the reward function are unknown, the Bellman error methods can still suggest a way of determining the parameters of the approximation architecture based on a sample $\mathcal{F}$ of four-tuples encountered during the learning.

[2]These two values may differ because the value of $a$ at time $t - j + 1$ is not equal to the value of $a$ at time $t + 1$.

The adaptation of equation (3.28) is straightforward and leads to the following optimization problem :

$$\min_a \left( \sum_{(x_t,u_t,r_t,x_{t+1})\in\mathcal{F}} (\tilde{Q}(x_t,u_t,a) - [r_t + \gamma \max_{u\in U(x_{t+1})} \tilde{Q}(x_{t+1},u,a)])^2 \right), \qquad (6.8)$$

where $\mathcal{F}$ is the set of four-tuples $(x_t, u_t, r_t, x_{t+1})$ already observed at a certain iteration of the reinforcement learning protocol.

Strictly speaking, this optimization problem should be solved at each step $t+1$ of each episode and taking into account the full set of four-tuples already encountered during the current and all past episodes. Obviously an efficient implementation of this idea would call for incremental formulas allowing to maintain an approximation of the gradient of equation (6.8), each time a new four-tuple is included into $\mathcal{F}$. We did not investigate this type of algorithm but believe that the parallel between equation (6.8) and the $Q$-learning algorithm (equation (6.3)) may suggest further ideas.

Notice that if $\tilde{Q}$ can represent the $Q$-function of the original problem exactly and if each state-action pair has been visited an infinite number of times during the learning, it can be shown that the function $\tilde{Q}$ obtained by solving (6.8) is equal to the real $Q$-function.

### 6.1.3 Action selection

Before closing this section, it should be noted that the approximation of the $Q$-function ($\tilde{Q}$) that will be finally obtained by using the *Learning* modules with the aggregation or the representative states technique will only be defined for all pairs $(x, u)$ such that $x \in X$ and $u \in U^\delta(x)$ (see equations (3.7) and (3.16)). The *Action Selection* modules that use the approximate $Q$-function ($\tilde{Q}$) in order to select an action $u \in U(x)$ will not hold valid anymore if $U(x) \neq U^\delta(x)$. This will notably be the case for the $\epsilon$-Greedy policy defined in figure 5.2. Therefore, when mentioning that an $\epsilon$-Greedy policy is used in a reinforcement learning algorithm used with an infinite state space control problem we will consider the version of the $\epsilon$-Greedy algorithm sketched on figure 6.3.

## 6.2 About convergence properties

In the following sections we will discuss in detail various combinations of algorithms and approximation schemes and investigate their asymptotic convergence properties when the number of time-samples and/or episodes grows to infinity. In

Figure 6.3: *Action selection* : $\epsilon$-Greedy policy

```
**Input :** Current state $x$ of the system
Choose a number at random in the interval $[0, 1]$ with a uniform probability.
If the number is inferior to $\epsilon$ then choose the control variable value randomly in $U^\delta(x)$. Otherwise choose the control variable value randomly in $\{u \in U^\delta(x) | \tilde{Q}(x,u) = \max_{u' \in U^\delta(x)} \tilde{Q}(x,u')\}$.
```

order to clarify and simplify the comparisons of these various properties we will use a unified scheme to state such convergence properties. Below, we first explain our notations for convergence properties, then discuss why it is necessary in the current context to impose restrictive conditions in order to satisfy them.

**Definition of a convergence property**

A convergence property of a *Learning* module is defined through the following elements :

- conditions on the way the sampling is realized (referred to as *Input*). These are basically conditions on the way the four-tuples $(x_t, u_t, r_t, x_{t+1})$ used as input of the learning algorithm are generated
- conditions on the values of the parameters inside the algorithm (referred to as *Algorithm*)
- conditions on the control problem (referred to as *Control problem*). These are basically conditions on the system dynamics and the reward function
- the solution to which it converges if the above three families of conditions are fulfilled (referred to as *Solution*).

By way of example we have, for the algorithm defined on figure 5.4 and entitled "*Estimation of the MDP structure* : Kalman Filter like algorithm" the following convergence property :

$$\begin{cases} \textit{Input} : \text{each state-action pair is visited an infinite number of times} \\ \textit{Algorithm} : \beta = 1 \\ \textit{Control problem} : / \\ \textit{Solution} : \text{converge to the MDP structure defined by equations (2.28) and (2.29).} \end{cases}$$

**On the requirement for restrictive sampling conditions**

We will see later in this chapter that most of the convergence properties of the *Learning* modules can only be established if some rather restrictive conditions are done on the way the sampling is realized.
In order to provide some intuitive clues about this necessity, let us first consider the case of the aggregation technique. Figure 6.4 represents a trajectory in a discretized state space. We focus on the state $x_t \in X_i$. At time $t+1$ the four-tuple $(x_t, u_t, r_t, x_{t+1})$ will be used in the *Learning* module either to update $p^\delta(.|x_i^\delta, u_t)$ and $r^\delta(x_i^\delta, u_t)$ or to refresh directly the value $Q^\delta(x_i^\delta, u_t)$.

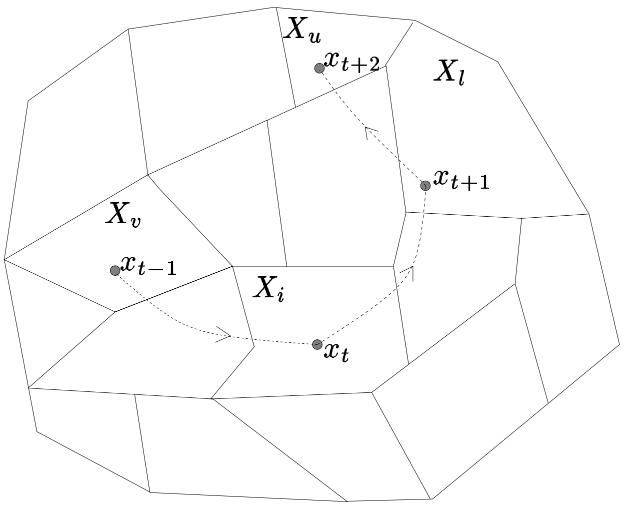


Figure 6.4: Aggregation technique : trajectory in a discretized continuous state space

We remind that in chapter 3 states in a particular region $X_i$ were sampled directly (and independently of everything else) through a probability distribution $p_i(x)$ and controls were sampled (actually screened) independently in $U(x)$ for each such state. Notice that this sampling scheme was implementable in the context of chapter 3 thanks to the direct access to the system dynamics $f(x, u, w)$, and led to various convergence properties.
In the context of reinforcement learning the situation is quite different; indeed, the state-action pairs encountered during a particular episode necessarily follow a system trajectory (see figure 6.4) and the control values depend on the current value of the $\tilde{Q}$-function i.e., on all past state-action pairs encountered during the learning protocol. This introduces a complex statistical dependency among successive

state-action pairs, and makes it difficult if not impossible, to prove general convergence properties without supplementary assumptions on the way the samples are eventually generated by an algorithm.

On the other hand, if in some context we assume that *each pair* $(x_t, u_t)$ *is drawn independently from a certain probability distribution*[3] (which occurs for example if we use a probabilistic policy that does not vary during the learning[4] and one-step episodes with the initial state of each episode being drawn independently from a probability distribution), then some convergence properties can be recovered.

At this point, note that for one type of algorithm presented in this chapter (namely the model based Kalman filter like algorithm), this condition can be relaxed, because the algorithm reconstructs the MDP$^\delta$ structure by giving to each observation the same weight, no matter the order in which these observations take place. Therefore, in this case some convergence properties can be found under a less restrictive assumption, namely that the *infinite sample* composed of all the state-action pairs visited can be assumed to have been generated by a probability distribution[5].

Let us now consider the case where the RL algorithms are used with the representative states technique. Figure 6.5 represents a trajectory in a triangulated state space. As with the aggregation technique, convergence properties will be formulated under the assumption that the state-action pairs $(x_t, u_t)$ visited during the learning are drawn independently from a probability distribution. Moreover another problem, specific to the proposed non-model based algorithms, also occurs. Indeed, we will see that in this particular context, we are able to prove convergence properties only

[3]Let us clarify what we understand by this sentence. By context, we mean a stochastic mechanism which produces sequences $(x_t, u_t), t = 1, 2, \dots$ that are used by a reinforcement learning algorithm. We say that this sequence *is assumed to be drawn independently from a certain probability measure $P$ defined over $X \times U$*, if any finite subset of these random variables $(X_{t_1}, U_{t_1}), \dots, (X_{t_n}, U_{t_n}), \forall n \in \mathbb{N}$ are mutually independent and if each one is distributed according to $P$. Under these conditions one can possibly show that some convergence property holds with *probability 1*, which means that the set of sequences generated by the mechanism which would lead to the violation of this convergence property has probability 0.

[4]An $\epsilon$-Greedy policy is a probabilistic policy that varies during the learning because the function $\tilde{Q}$ it uses to select an action varies.

[5]Let us also clarify this statement. We say that a particular sample $(x_t, u_t), t = 1, 2, \dots$ can be *assumed to have been generated by a probability distribution*, if there exists a probability measure defined over $X \times U$, such that for every measurable subset $S \subset X \times U$ it is true that

$$\lim_{N \to \infty} \frac{\sum_{t=0}^{N-1} I_S(x_t, u_t)}{N} = P(S).$$

Note that this statement concerns a particular sample, and not the mechanism which is used to generate this sample. Under these conditions, one can possibly show that some convergence property holds *for this particular sample*.

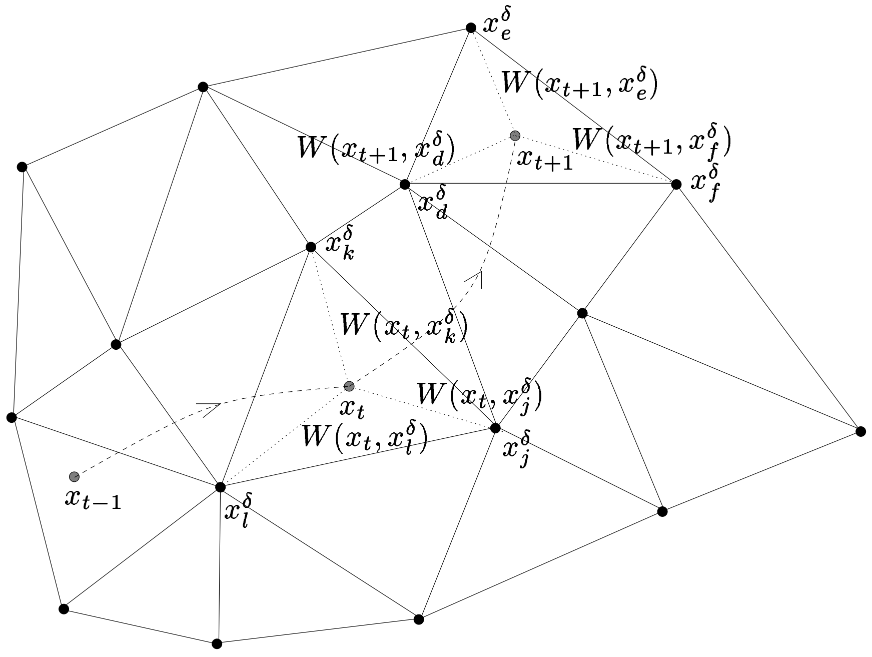


Figure 6.5: Representative states technique : trajectory in a triangulated state space

if we furthermore assume that each state-action pair $(x_t, u_t)$ visited is such that $x_t$ coincides with an element of $X^\delta$ (that is with a black bullet of figure 6.5).

**Violation of restrictive sampling conditions in the applications**

Even if some restrictive conditions on the way of realizing the learning are required in order to have convergence properties, they will be most of the time violated in the applications considered in this work. Violation of these restrictive conditions does not mean that the algorithms do not converge anymore but, rather, that we are *unable to characterize the solution* to which they are going to converge.

**Interest of convergence conditions**

Since the convergence conditions that we will state generally make unrealistic assumptions on the sampling mechanism, their practical usefulness is questionable. Nevertheless, we found it interesting from a conceptual point of view to state these conditions, since we believe that they highlight theoretical features of the different algorithms and thereby improve our intuitive understanding of their main similarities and differences.

Figure 6.6: *Estimation of the MDP$^\delta$ structure* : Kalman Filter like algorithm used with the aggregation technique

```
Input : $x_t$, $u_t$, $r_t$ and $x_{t+1}$
If $u_t \notin U^\delta(x_t)$ then exit.
Determine to which $x_i^\delta$ $x_t$ corresponds.
$H(x_i^\delta, u_t) \leftarrow \beta H(x_i^\delta, u_t) + 1$
$r^\delta(x_i^\delta, u_t) \leftarrow r^\delta(x_i^\delta, u_t) + \frac{1}{H(x_i^\delta,u_t)}(r_t - r^\delta(x_i^\delta, u_t))$
Repeat once for all $j \in \{1, \cdots, m\}$
    $p^\delta(x_j^\delta|x_i^\delta, u_t) \leftarrow p^\delta(x_j^\delta|x_i^\delta, u_t) + \frac{1}{H(x_i^\delta,u_t)}(I_{X_j}(x_{t+1}) - p^\delta(x_j^\delta|x_i^\delta, u_t))$
```

## 6.3 Model based aggregation technique

In this section we use the aggregation technique to formulate two model based algorithms. One relies on a Kalman filter like algorithm to update the MDP$^\delta$ structure (see appendix A.1) while the other is based on a stochastic approximation algorithm (see appendix A.2). Both algorithms are inspired from the procedure outlined in section 6.1.1. Hence, they will try to determine

- a value for $r^\delta(x_i^\delta, u)$ that "stands close" to $\underset{w}{E}[r(x, u, w)]$, $\forall x \in X_i$
- a value for $p^\delta(x_j^\delta|x_i^\delta, u)$ that "stands close" to $\underset{w}{E}[I_{X_j}(f(x, u, w))]$, $\forall x \in X_i$.

### 6.3.1 Kalman Filter like algorithm

This algorithm is sketched on figure 6.6. It takes the four-tuple $(x_t, u_t, r_t, x_{t+1})$ as input, checks whether $u_t \in U^\delta(x_t)$ (if $u_t \notin U^\delta(x_t)$ then no structure update is done), identifies in which subset $X_i$ the state $x_t$ stands and then updates $r^\delta(x_i^\delta, u_t)$ and $p^\delta(.|x_i^\delta, u_t)$. We note that in the present case and if $\beta = 1$, the function $H(x^\delta, u)$ defined on $X^\delta \times U^\delta$ represents the number of times the action $u$ has been taken while being in the subset corresponding to $x^\delta$. The value of this function is initialized to 0 everywhere at the beginning of the learning. If $\beta < 1$, more weight is given to the more recent observations in the MDP$^\delta$ structure estimation.

## Rationale

Let $N^\delta(x_i^\delta, u)$ be the number of times the action $u$ ($u \in U^\delta(x_i^\delta)$) has been taken while being in $X_i$. Let $x(k)$ and $w(k)$ be respectively the value of $x_t$ and $w_t$ the $k$th time the action $u_t = u$ has been taken while being in $X_i$.
It can be shown that the value of $r^\delta(x_i^\delta, u)$ estimated by the algorithm minimizes

$$\sum_{k=1}^{N^\delta(x_i^\delta,u)} \beta^{N^\delta(x_i^\delta,u)-k}(r^\delta(x_i^\delta, u) - r(x(k), u, w(k)))^2.$$

Similarly it can be shown that the computed value of $p^\delta(x_j^\delta|x_i^\delta, u)$ minimizes

$$\sum_{k=1}^{N^\delta(x_i^\delta,u)} \beta^{N^\delta(x_i^\delta,u)-k}(p^\delta(x_j^\delta|x_i^\delta, u) - I_{X_j}(f(x(k), u, w(k))))^2.$$

Suppose now that the infinite sample composed of all the state-action pairs $(x, u)$ visited (with $u \in U^\delta(x)$) can be assumed to have been generated by a probability distribution $P_{learning}(x, u)$. If $\beta = 1$, then the values of $r^\delta(x_i^\delta, u)$ and $p^\delta(x_j^\delta|x_i^\delta, u)$ estimated by the algorithm minimize respectively

$$\underset{(x,w)}{E}[(r^\delta(x_i^\delta, u) - r(x, u, w))^2|u, x \in X_i, P_{learning}(x, u)]$$

and

$$\underset{(x,w)}{E}[(p^\delta(x_j^\delta|x_i^\delta, u) - I_{X_j}(f(x, u, w)))^2|u, x \in X_i, P_{learning}(x, u)].$$

## Convergence properties

The convergence properties of the algorithm described in figure 6.6 are[6] :

- *Input :* each state-action pair $(x^\delta, u) \in X^\delta \times U^\delta$ is visited an infinite number of times [7]
  *Algorithm :* $\beta = 1$
  *Control problem :* the equivalence conditions (3.9) and (3.10) are satisfied by the approximation architecture

[6]The following convergence properties are not an exhaustive list of the algorithm's convergence properties but the ones that give interesting insights about the algorithm behavior.

[7]By visit of a state-action pair $(x^\delta, u)$ when using the aggregation technique, we mean that the action $u$ has been triggered while being in the subset $X_i$ corresponding to $x^\delta$.

*Solution :* it converges *with probability 1* to the MDP$^\delta$ structure defined by equations (3.1) and (3.3)[8]

- *Input :* one-step episodes are used, with $P_0(x)$ given by :

$$P_0(x) = \begin{cases} c_1 p_1(x) & \text{if } x \in X_1 \\ c_2 p_2(x) & \text{if } x \in X_2 \\ \vdots & \\ c_m p_m(x) & \text{if } x \in X_m \end{cases}$$

with $c_i > 0$ and $\sum_{i=1}^m c_i = 1$, and the probability to select an action $u \in U^\delta(x_i^\delta)$ while being in any state of $X_i$ is the same and different from zero.
*Algorithm :* $\beta = 1$
*Control problem :* /
*Solution :* it converges *with probability 1* to the MDP$^\delta$ structure defined by equations (3.1) and (3.3)

- *Input :* the infinite sample composed of all the state-action pairs $(x, u)$ visited (with $u \in U^\delta(x)$) can be assumed to have been generated by the probability distribution $P_{learning}(x, u)$ and each state-action pair $(x^\delta, u) \in X^\delta \times U^\delta$ is visited an infinite number of times
*Algorithm :* $\beta = 1$
*Control problem :* /
*Solution :* it converges *for this sample* and *with probability 1* with respect to disturbance process to the MDP$^\delta$ structure defined as follows :

$$\begin{aligned} p^\delta(x_j^\delta | x_i^\delta, u) &= \underset{(x,w)}{E}[I_{X_j}(f(x,u,w)) | u, x \in X_i, P_{learning}(x,u)] \\ &= \underset{x}{E}[\underset{w}{E}[I_{X_j}(f(x,u,w))] | u, x \in X_i, P_{learning}(x,u)] \\ r^\delta(x_i^\delta, u) &= \underset{(x,w)}{E}[r(x,u,w) | u, x \in X_i, P_{learning}(x,u)] \\ &= \underset{x}{E}[\underset{w}{E}[r(x,u,w)] | u, x \in X_i, P_{learning}(x,u)]. \end{aligned}$$

### 6.3.2 Stochastic Approximation algorithm

*We have adopted a repetitive and partly redundant style here in order to ensure that each algorithm description is self-contained and thus can be read independently of the others.*

[8]Note that if there exists a MDP$^\delta$ structure that satisfies equations (3.9) and (3.10) then the MDP$^\delta$ structure defined by equations (3.1) and (3.3) does not depend on $p_i(.)$ anymore.

Figure 6.7: *Estimation of the MDP$^\delta$ structure* : Stochastic Approximation algorithm used with the aggregation technique

---

```
Input : $x_t$, $u_t$, $r_t$ and $x_{t+1}$
If $u_t \notin U^\delta(x_t)$ then exit.
Determine to which $x_i^\delta$ $x_t$ corresponds.
$r^\delta(x_i^\delta, u_t) \leftarrow r^\delta(x_i^\delta, u_t) + \alpha_r(r_t - r^\delta(x_i^\delta, u_t))$
Repeat once for all $j \in \{1, \cdots, m\}$
    $p(x_j^\delta|x_i^\delta, u_t) \leftarrow p(x_j^\delta|x_i^\delta, u_t) + \alpha_p(I_{X_j}(x_{t+1}) - p(x^\delta|x_i^\delta, u_t))$
```

---

This algorithm is represented on figure 6.7. At each $t+1$ it takes as input the four-tuple $(x_t, u_t, r_t, x_{t+1})$ and refreshes the MDP$^\delta$ structure if $u_t \in U^\delta(x_t)$. Notice that in the particular case for which $\alpha_r = \alpha_p = \frac{1}{N^\delta(x_i^\delta, u_t)}$ where $N^\delta(x_i^\delta, u_t)$ represents the number of times the action $u_t$ has been taken while being in $X_i$, this algorithm behaves exactly as the *Kalman Filter like algorithm* (with $\beta = 1$).

**Convergence properties**

Let $\alpha_{\{r,p\}_k}$ be the value of $\alpha_{\{r,p\}}$ at the $k$th stage of the algorithm.
We define $\alpha_{\{r,p\}_k}(x_i^\delta, u)$ $(\forall (x_i^\delta, u) \in X^\delta \times U^\delta)$ :

- as being equal to $\alpha_{\{r,p\}_k}$ if at the $k$th stage of the algorithm $u = u_t$ and $x_t \in X_i$
- as being equal to 0 if at the $k$th stage of the algorithm $u \neq u_t$ or $x_t \notin X_i$

and introduce the following condition :

$$\sum_{k=0}^{\infty} \alpha_{\{r,p\}_k}(x^\delta, u) = \infty, \quad \sum_{k=0}^{\infty} \alpha^2_{\{r,p\}_k}(x^\delta, u) < \infty \quad \forall (x^\delta, u) \in X^\delta \times U^\delta. \tag{6.9}$$

The convergence properties of the algorithm described in figure 6.7 are :

- *Input :* each state-action pair $(x^\delta, u) \in X^\delta \times U^\delta$ is visited an infinite number of times
  *Algorithm :* the $\alpha_{\{r,p\}_k}(x^\delta, u)$ are non negative and satisfy condition (6.9)[9]

[9]These conditions on the $\alpha_{\{r,p\}_k}(x^\delta, u)$ terms imply also that each state-action pair $(x^\delta, u) \in X^\delta \times U^\delta$ has to be visited an infinite number of times.

*Control problem :* the equivalence conditions (3.9) and (3.10) are satisfied
*Solution :* it converges *with probability 1* to the MDP$^\delta$ defined by equations (3.1) and (3.3)

- *Input :* one-step episodes are used, with $P_0(x)$ given by :

$$P_0(x) = \begin{cases} c_1 p_1(x) & \text{if } x \in X_1 \\ c_2 p_2(x) & \text{if } x \in X_2 \\ \vdots & \\ c_m p_m(x) & \text{if } x \in X_m \end{cases}$$

  with $c_i > 0$ and $\sum_{i=1}^m c_i = 1$, and if the probability to select an action $u \in U^\delta(x_i^\delta)$ while being in any state of $X_i$ is the same and different from zero
  *Algorithm :* the $\alpha_{\{r,p\}_k}(x^\delta, u)$ are non negative and satisfy condition (6.9)
  *Control problem :* /
  *Solution :* it converges *with probability 1* to the MDP$^\delta$ structure defined by equations (3.1) and (3.3)

- *Input :* one-step episodes are used, each state-action pair $(x_0, u_0)$ is drawn independently according to the probability distribution $P_0(x, u)$ and each state-action pair $(x^\delta, u) \in X^\delta \times U^\delta$ is visited an infinite number of times.
  *Algorithm :* the $\alpha_{\{r,p\}_k}(x^\delta, u)$ are non negative and satisfy condition (6.9)
  *Control problem :* /
  *Solution :* it converges *with probability 1* to the MDP$^\delta$ structure defined as follows :

$$\begin{aligned} r^\delta(x_i^\delta, u) &= \underset{(x,w)}{E}[r(x,u,w)|u, x \in X_i, P_0(x,u)] \\ &= \underset{x}{E}[\underset{w}{E}[r(x,u,w)]|u, x \in X_i, P_0(x,u)] \\ p^\delta(x_j^\delta | x_i^\delta, u) &= \underset{(x,w)}{E}[I_{X_j}(f(x,u,w))|u, x \in X_i, P_0(x,u)] \\ &= \underset{x}{E}[\underset{w}{E}[I_{X_j}(f(x,u,w))]|u, x \in X_i, P_0(x,u)]. \end{aligned}$$

## 6.4 Non-model based aggregation technique

This section presents algorithms which update the $Q^\delta$-function without explicitly reconstructing any MDP$^\delta$ structure. The value of the approximate $Q$-function will

Figure 6.8: $Q^\delta$*-function correction* : $Q$-learning algorithm used with the aggregation technique

```
Input : $x_t$, $u_t$, $r_t$ and $x_{t+1}$
If $u_t \notin U^\delta(x_t)$ then exit.
Determine to which $x_i^\delta$ the state $x_t$ corresponds
$\delta \leftarrow (r_t + \gamma \max_{u \in U^\delta(x_{t+1})} \tilde{Q}(x_{t+1}, u)) - \tilde{Q}(x_t, u_t)$
$Q^\delta(x_i^\delta, u_t) \leftarrow Q^\delta(x_i^\delta, u_t) + \alpha\delta$
```

be deduced from the $Q^\delta$-function by using equation (3.7) that we remind hereafter :

$$\tilde{Q}(x, u) \quad = \quad Q^\delta(x_i^\delta, u) \quad \text{if } x \in X_i \quad \forall x \in X, \forall\, u \in U^\delta(x). \tag{6.10}$$

### 6.4.1 $Q$-learning algorithm

The algorithm is represented on figure 6.8. At each $t+1$ it takes the four-tuple $(x_t, u_t, r_t, x_{t+1})$ as input and updates $Q^\delta(x_i^\delta, u_t)$ (with $x_t \in X_i$) leaving the other components of the $Q^\delta$-function unchanged.
This algorithm is an immediate consequence of the advantage updating method described in section 6.1.2 (equation (6.3)) by noting that according to equation (6.10) we have $\frac{\partial \tilde{Q}(x,u)}{\partial Q^\delta(x_i^\delta, u)} = 1$ if $x \in X_i$ and 0 otherwise.

#### Convergence properties

Let $\alpha_k$ be the value of $\alpha$ at the $k$th stage of the algorithm.
We define $\alpha_k(x_i^\delta, u)$ $(\forall (x_i^\delta, u) \in X^\delta)$ :

- as being equal to $\alpha_k$ if at the $k$th stage of the algorithm, $u = u_t$ and $x_t \in X_i$
- as being equal to 0 if at the $k$th stage of the algorithm, $u \neq u_t$ or $x_t \notin X_i$

and introduce the following condition :

$$\sum_{k=0}^{\infty} \alpha_k(x^\delta, u) = \infty, \quad \sum_{k=0}^{\infty} \alpha_k^2(x^\delta, u) < \infty \quad \forall (x^\delta, u) \in X^\delta \times U. \tag{6.11}$$

The convergence properties of the algorithm described in figure 6.8 are :

- *Input :* each state-action pair $(x^\delta, u) \in X^\delta \times U^\delta$ is visited an infinite number of times
  *Algorithm :* the $\alpha_k(x^\delta, u)$ are non negative and satisfy condition (6.11)
  *Control problem :* equivalence conditions (3.9) and (3.10) are satisfied
  *Solution :* it converges *with probability 1* to the $Q^\delta$-function corresponding to the MDP$^\delta$ whose structure is defined by equations (3.1) and (3.3)
  The proof is given in section C.6

- *Input :* one-step episodes are used, with $P_0(x)$ given by :

$$P_0(x) = \begin{cases} c_1 p_1(x) & \text{if } x \in X_1 \\ c_2 p_2(x) & \text{if } x \in X_2 \\ \vdots & \\ c_m p_m(x) & \text{if } x \in X_m \end{cases}$$

  with $c_i > 0$ and $\sum_{i=1}^{m} c_i = 1$, and the probability to select an action $u \in U^\delta(x_i^\delta)$ while being in any state of $X_i$ is the same and different from zero
  *Algorithm :* the $\alpha_k(x^\delta, u)$ are non negative and satisfy condition (6.11)
  *Control problem :* /
  *Solution :* it converges *with probability 1* to the $Q^\delta$-function corresponding to the MDP$^\delta$ defined by equations (3.1) and (3.3)
  The proof is given in section C.6

- *Input :* one-step episodes are used, each state-action pair $(x_0, u_0)$ is drawn independently according to the probability distribution $P_0(x, u)$ and each state-action pair $(x^\delta, u) \in X^\delta \times U^\delta$ is visited an infinite number of times
  *Algorithm :* the $\alpha_k(x^\delta, u)$ are non negative and satisfy condition (6.11)
  *Control problem :* /
  *Solution :* it converges *with probability 1* to the $Q^\delta$-function corresponding to the MDP$^\delta$ whose structure is defined as follows :

$$\begin{aligned} r^\delta(x_i^\delta, u) &= \underset{(x,w)}{E}[r(x,u,w) | u, x \in X_i, P_0(x,u)] \\ p^\delta(x_j^\delta | x_i^\delta, u) &= \underset{(x,w)}{E}[I_{X_j}(f(x,u,w)) | u, x \in X_i, P_0(x,u)]. \end{aligned}$$

  The proof is given in section C.7.

### 6.4.2 $Q(\lambda)$ algorithm

Iterations (6.6) and (6.7) used with $\tilde{Q}$ defined by using an aggregation technique lead to the algorithm described in figure 6.9 where the function $e$ represents the

Figure 6.9: $Q^\delta$*-function correction* : Watkins's $Q(\lambda)$ algorithm used with the aggregation technique

```
Input : $x_t$, $u_t$, $r_t$, $x_{t+1}$ and $u_{t+1}$
If $u_t \notin U^\delta(x_t)$ then exit.
Determine to which $x_i^\delta$ $x_t$ corresponds
$\delta \leftarrow (r_t + \gamma \max_{u \in U^\delta(x_{t+1})} \tilde{Q}(x_{t+1}, u)) - \tilde{Q}(x_t, u_t)$
$e(x_i^\delta, u_t) \leftarrow e(x_i^\delta, u_t) + 1$
Do $\forall (x^\delta, u) \in X^\delta \times U$
    $Q^\delta(x^\delta, u) \leftarrow Q^\delta(x^\delta, u) + \alpha\delta e(x^\delta, u)$
    If $\tilde{Q}(x_{t+1}, u_{t+1}) = \max_{u' \in U^\delta(x_{t+1})} \tilde{Q}(x_{t+1}, u')$ then $e(x^\delta, u) \leftarrow \gamma\lambda e(x^\delta, u)$, else
    $e(x^\delta, u) = 0$
```

eligibility trace[10]. This algorithm differs nonetheless from the one suggested by iterations (6.6) and (6.7) by setting the eligibility trace to zero each time a non-greedy action is chosen. This feature has been discussed in section 5.4.3 when introducing the $Q(\lambda)$ algorithm.

For this algorithm to hold valid, eligibility trace $e$ must be set to zero at the beginning of each episode.

Remark that if $\lambda = 0$ or if one-step episodes are used then the algorithm acts like the $Q$-learning algorithm.

## 6.5 Aggregation technique : a unified view

Suppose that we have kept in a set the four-tuples $(x_t, u_t, r_t, x_{t+1})$ obtained while interacting with the system. Suppose also that each action $u \in U^\delta(x_i^\delta)$ has been chosen at least once while being in $X_i$.

We propose three different ways of computing the $Q^\delta$-function from this set.

[10]Remark that with the aggregation technique the vector $\frac{\partial \tilde{Q}(x,u,a)}{\partial a}$ does not depend on $a$ anymore (its components can only be equal to 0 or 1) and therefore the algorithm described by iteration (6.5) modifies $a$ exactly like the algorithm described by iterations (6.6) and (6.7).

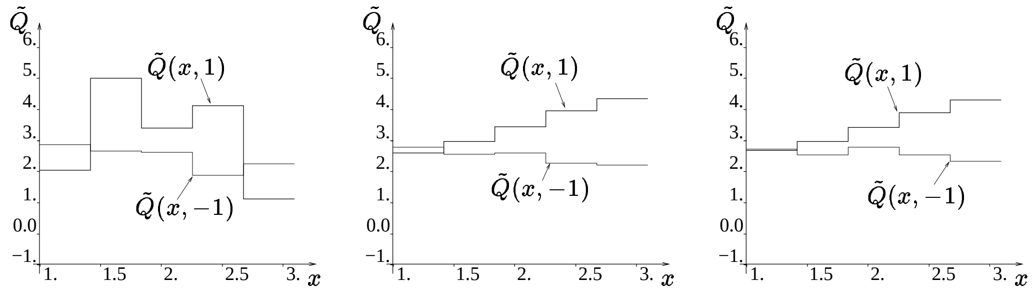
1. Each element of the set is used once as input of the *Kalman Filter like algorithm* (figure 6.6). The $Q^\delta$-function is estimated by solving the MDP$^\delta$ structure computed this way.

2. We suppose that each element of the set has the same probability to be selected. We repeat an infinite number of times the sequence of operations that consists first in selecting an element of the set and then in using it as input of the *Stochastic Approximation algorithm* (figure 6.7). The $Q^\delta$-function is estimated by solving the MDP$^\delta$ structure computed this way.

3. We suppose that each element of the set has the same probability to be selected. We repeat an infinite number of times the sequence of operations that consists first in selecting an element of the set and then in using it as input of the *$Q$-learning algorithm* (figure 6.8).

The $Q^\delta$-function estimated by each of these three procedures is the same. This can be shown by adopting the same reasoning as the one carried out in section 5.5.

**Example 6.5.1** We describe in this example an application of the reinforcement learning algorithms used with the aggregation technique on the infinite state space control problem defined in example 3.2.2. We recall that in example 3.2.2 we used the aggregation technique in order to compute $\tilde{Q}$ from the dynamics and reward function specification.



(a) 100 episodes (b) 500 episodes (c) 1000 episodes

Figure 6.10: Representation of $\tilde{Q}$ at different stages of the learning process

As in example 3.2.2, we partition the state space into six disjoint subsets $X_i = [1 + 0.4 * (i-1), 1.4 + 0.4 * (i-1)[$ with $i \in \{1,2,3,4\}$, $X_5 = [2.6, 3]$ and $X_6 = \{x^t\}$. The control space $C = [-1, 1]$ being infinite we use a finite subset of it to define $C^\delta$. This subset is chosen equal to $\{-1, 1\}$. $U^\delta(x^\delta)$ is chosen equal to $C^\delta$ everywhere on $X^\delta \setminus \{x_6^\delta\}$.
The type of *Action Selection* module we choose is an $\epsilon$-Greedy policy with $\epsilon = 0.1$. The *Learning* module used is a model based algorithm for which the *Estimation of the MDP*$^\delta$

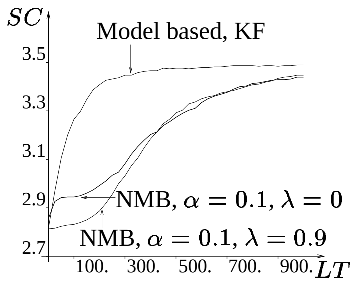
*structure* is done by using the Kalman Filter like algorithm. The *Resolution of the MDP*$^\delta$ module consists in solving at each time step $t+1$ the MDP$^\delta$ by using a Gauss-Seidel version of the value iteration algorithm.

The $Q^\delta$-function is initialized to zero everywhere at the beginning of the learning.

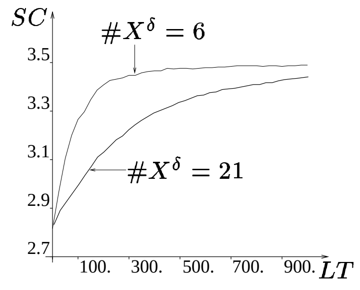
The initial state of each episode is chosen at random among $[1,3]$ and an episode ends only when the terminal state $x^t$ is reached.

The $\tilde{Q}$ function obtained after 100 episodes is represented on figure 6.10a. According to this $\tilde{Q}$ function the optimal stationary policy consists in taking $u=-1$ on $[1,1.4[$ and $[2.6,3]$ and $u=1$ elsewhere. If we compare this result with the one computed for the same discretization of the state space and the same $C^\delta$ when the system dynamics and reward function were known (figure 3.4b) we see that the results are different. But after 500 episodes the results obtained in terms of approximation of the optimal policy are equivalent ($u=-1$ only on the interval $[1,1.4[$) (see figure 6.10b).

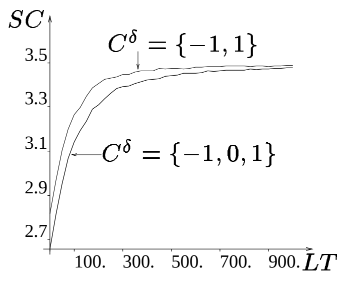
By comparing the $\tilde{Q}$ function obtained after 500 episodes and 1000 episodes (figure 6.10c), one can remark that the most significant variations of $\tilde{Q}$ occur where $\tilde{Q}(x,u)\neq\tilde{V}(x)$. In these regions the learning was slower and did not converge even after 500 episodes. This is because the $\epsilon$-Greedy policy chosen favors, while being in state $x$, the action $u$ that satisfies $\tilde{Q}(x,u)=\tilde{V}(x)$.



(a) MB vs NMB algorithms



(b) Influence of $\#X^\delta$



(c) Influence of $\#C^\delta$

Figure 6.11: Score curves

If we compute the score obtained by the policy after each episode of the learning (estimation of equation (3.26) while choosing $P_0(.)$ as a uniform probability distribution on $X\setminus\{x^t\}$ and $\mu=\epsilon$-Greedy policy with $\epsilon=0$) we obtain the curve drawn on figure 6.11a and labeled "Model based, KF"[11]. On the same figure are also drawn the score curves obtained when non-model based algorithms are used as *Learning* modules. As we can observe, the performances of the NMB algorithms are worse than those obtained by the MB

[11]The score curve drawn is in fact the average of the score curves obtained by 200 learning processes. This operation has been realized in order to ease the comparison between the different learning techniques that otherwise would be more difficult due to the harsh aspect of the score evolution especially at the beginning of the learning process.

algorithm. One can notice that the use of an eligibility trace ($\lambda = 0.97$) deteriorates the performances of the NMB algorithms, especially at the beginning of the learning.
On figure 6.11b we have drawn the score curves computed by the same reinforcement learning algorithm ($\epsilon$-Greedy policy with $\epsilon = 0.1$, MB with KF) but for two different discretizations of the state space. An increase in $\#X^{\delta}$ slows down the learning process. This is because a more complex MDP$^{\delta}$ structure requires a larger learning time to be reconstructed. However by the end a finer discretization of the state space can lead to better results. The same type of observations can be done concerning the influence of $C^{\delta}$ on the score curves. An increase in $\#C^{\delta}$ also increases the complexity of the MDP$^{\delta}$ structure. The influence of $C^{\delta}$ on the score curves is illustrated on figure 6.11c. Remark also on this figure that the starting points of the two score curves are different. This is due to the fact that at the beginning of the learning ($LT = 0$), $\tilde{Q}(x, u)$ is equal to zero everywhere. It implies the policy evaluated at $LT = 0$ is the randomized stationary policy that consists in choosing, while being in state $x$, all actions $u \in C^{\delta}$ with the same probability (see figure 6.3 with $\epsilon = 0$). Different $C^{\delta}$ lead to different randomized stationary policies, which explains why the starting points of the score curves are different.

## 6.6 Model based representative states technique

In this section we adapt the two model based algorithms to the representative states technique. Both algorithms are inspired from the procedure explained in section 6.1.1. They try to determine

- values for $r^{\delta}(x_1^{\delta}, u), \cdots, r^{\delta}(x_m^{\delta}, u)$ such that $\sum_{i=1}^{m} W(x, x_i^{\delta}) r^{\delta}(x_i^{\delta}, u)$ "stands close" to $\underset{w}{E}[r(x, u, w)]$, $\forall x \in X$

- values for $p^{\delta}(x^{\delta'}|x_1^{\delta}, u), \cdots, p^{\delta}(x^{\delta'}|x_m^{\delta}, u)$ such that $\sum_{i=1}^{m} W(x, x_i^{\delta}) p^{\delta}(x^{\delta'}|x_i^{\delta}, u)$ "stands close" to $\underset{w}{E}[W(f(x, u, w), x^{\delta'})]$, $\forall x \in X$.

### 6.6.1 Kalman Filter like algorithm

This algorithm is sketched on figure 6.12 [12].
It takes the four-tuple $(x_t, u_t, r_t, x_{t+1})$ as input, checks whether $u_t \in U^{\delta}(x_t)$ (if $u_t \notin U^{\delta}(x_t)$ then no structure update is done) and updates $r^{\delta}(., u_t)$ and $p^{\delta}(.|., u_t)$.

[12]The main drawback of this algorithm is linked to the high computational burden it requires since it inverses at each $t+1$ the $m \times m$ matrix $H(u_t)$. Even if the computational burden can be lightened by exploiting the fact that the matrices $H(u)$ are symmetrical and sparse ($H(u)_{ij}$ can only become different from zero if there exists a $x \in X$ such that $W(x, x_i^{\delta})$ and $W(x, x_j^{\delta})$ are both different from zero), they can still remain high. The Stochastic Approximation algorithm we present in section 6.6.2 allows also the MDP$^{\delta}$ structure estimation without requiring such high computational burden. However its convergence properties are lesser.

Figure 6.12: *Estimation of the MDP$^\delta$ structure* : Kalman Filter like algorithm used with the representative states technique

```
**Input :** $x_t$, $u_t$, $r_t$, $x_{t+1}$
If $u_t \notin U^\delta(x_t)$ then exit.
$y(i) = W(x_t, x_i^\delta) \quad \forall i \in \{1, \cdots, m\}$
$H(u_t) \leftarrow \beta H(u_t) + yy^T$
*Estimation of the rewards :*
$a(i) = r^\delta(x_i^\delta, u) \quad \forall i \in \{1, \cdots, m\}$
$a \leftarrow a + H(u_t)^{-1}(r_t - y^T a)y$
$r^\delta(x_i^\delta, u) \leftarrow a(i) \quad \forall i \in \{1, \cdots, m\}$
*Estimation of the transition probabilities :*
Do $\forall j = \{1, \cdots, m\}$
    $a(i) = p^\delta(x_j^\delta | x_i^\delta, u) \quad \forall i \in \{1, \cdots, m\}$
    $a \leftarrow a + H(u_t)^{-1}(W(x_{t+1}, x_j^\delta) - y^T a)y$
    $p^\delta(x_j^\delta | x_i^\delta, u) = a(i) \quad \forall i \in \{1, \cdots, m\}$
```

This algorithm is based on the Kalman filter like algorithm described in section A.1.3. For this algorithm to hold valid, one has to initialize $r^\delta(.,.)$ to arbitrary values at the beginning of the learning, $p^\delta(.|.,.)$ to values such that $\sum_{j=1}^{m} p^\delta(x_j^\delta|.,.) = 1$ and the $m \times m$ matrix $H(u)$ to $\delta I$ with $\delta > 0, \forall u \in C^\delta$ (such initialization of the $H(u)$ matrix is done in order to ensure its invertibility at each stage of the learning process).
Let $N(u)$ be the number of times the action $u$ ($u \in C^\delta$) has been taken during the learning. Let $x(k)$ and $w(k)$ be respectively the value of $x_t$ and $w_t$ the $k$th time the action $u = u_t$ has been taken.
Suppose that $\delta \to 0$. It can be shown that the values of $r^\delta(x_1^\delta, u), \cdots, r^\delta(x_m^\delta, u)$ estimated by the algorithm minimize

$$\sum_{k=1}^{N(u)} \beta^{N(u)-k} (\sum_{i=1}^{m} W(x(k), x_i^\delta) r^\delta(x_i^\delta, u) - r(x(k), u, w(k)))^2. \tag{6.12}$$

Similarly it can be shown that the computed values of $p^\delta(x^{\delta'}|x_1^\delta, u), \cdots, p^\delta(x^{\delta'}|x_m^\delta, u)$ minimize

$$\sum_{k=1}^{N(u)} \beta^{N(u)-k} (\sum_{i=1}^{m} W(x(k), x_i^\delta) p^\delta(x^{\delta'}|x_i^\delta, u) - W(f(x(k), u, w(k)), x^{\delta'}))^2. \tag{6.13}$$

One can guarantee that $\sum_{j=1}^{m} p^\delta(x_j^\delta|.,.) = 1$ at each stage of the learning (section A.3.1). However, one cannot state that $|p^\delta(.|.,.)| \leq 1$. So, before using these transition probabilities to compute the $Q^\delta$-function, one should "correct" them in order to ensure that $\sum_{j=1}^{m} p^\delta(x_j^\delta|.,.) = 1$ and $|p^\delta(.|.,.)| \leq 1$ are satisfied (the reader can refer to figure 6.13 for a way to correct these transition probabilities).
Suppose now that the infinite sample composed of all the state-action pairs $(x, u)$ visited (with $u \in U^\delta(x)$) can be assumed to have been generated by a probability distribution $P_{learning}(x, u)$. Then it can be shown that if $\beta = 1$, the estimated values of $r^\delta(x_1^\delta, u), \cdots, r^\delta(x_m^\delta, u)$ minimize

$$\underset{(x,w)}{E}[(\sum_{i=1}^{m} W(x, x_i^\delta) r^\delta(x_i^\delta, u) - r(x, u, w))^2 | u, P_{learning}(x, u)]$$

and that the computed values of $p^\delta(x^{\delta'}|x_1^\delta, u), \cdots, p^\delta(x^{\delta'}|x_m^\delta, u)$ minimize

$$\underset{(x,w)}{E}[(\sum_{i=1}^{m} W(x, x_i^\delta) p^\delta(x^{\delta'}|x_i^\delta, u) - W(f(x, u, w), x^{\delta'}))^2 | u, P_{learning}(x, u)].$$

### Convergence properties

We define for each $u \in C^\delta$ the $n \times n$ matrix $F(u)$ :

$$F(u)_{ij} = \underset{x}{E}[W(x,x_i^\delta)W(x,x_j^\delta)|u, P_{learning}(x,u)] \ \forall i,j \in \{1,\cdots,m\}, \tag{6.14}$$

the $n$-vector $r(u)$ :

$$r(u)(i) = \underset{(x,w)}{E}[W(x,x_i^\delta)r(x,u,w)|u, P_{learning}(x,u)] \quad \forall i \in \{1,\cdots,m\} \tag{6.15}$$

and the $n \times n$ matrix $P(u)$ :

$$P(u)_{ij} = \underset{(x,w)}{E}[W(x,x_i^\delta)W(f(x,u,w),x_j^\delta)|u, P_{learning}(x,u)] \ \forall i,j \in \{1,\cdots,m\} \tag{6.16}$$

The convergence properties of the algorithm described in figure 6.12 are :

- *Input :* one-step episodes starting from an element of $X^\delta$ are used and each state-action pair $(x^\delta, u) \in X^\delta \times U^\delta$ is visited an infinite number of times.
  *Algorithm :* $\beta = 1$
  *Control problem :* /
  *Solution :* it converges *with probability 1* to the MDP$^\delta$ structure defined by equations (3.14) and (3.15).

- *Input :* the infinite sample composed of all the state-action pairs $(x, u)$ visited (with $u \in U^\delta(x)$) can be assumed to have been generated by the probability distribution $P_{learning}(x, u)$, each action $u \in C^\delta$ has been triggered an infinite number of times and $F(u)$ (equation (6.14)) is invertible $\forall u \in C^\delta$.
  *Algorithm :* $\beta = 1$
  *Control problem :* /
  *Solution :* it converges *for this sample* and *with probability 1* with respect to the noise process to the MDP$^\delta$ structure defined as follows :

  $$r^\delta(x_i^\delta, u) = (F(u)^{-1}r(u))(i)$$
  $$p^\delta(x_j^\delta|x_i^\delta, u) = (F(u)^{-1}P(u))_{ij}$$

  where the $m \times m$ matrix $F(u)$, the $n$-vector $r(u)$ and the $m \times m$ matrix $P(u)$ are defined respectively by expressions (6.14), (6.15) and (6.16).

## 6.6.2 Stochastic Approximation algorithm

### Algorithm description and properties

The algorithm (based on the stochastic approximation algorithm described in section A.2) is sketched on figure 6.13. It takes the four-tuple $(x_t, u_t, r_t, x_{t+1})$ as

Figure 6.13: *Estimation of the MDP$^\delta$ structure* : Stochastic Approximation algorithm used with the representative states technique

```
**Input :** $x_t$, $u_t$, $r_t$, $x_{t+1}$
If $u_t \notin U^\delta(x_t)$ then exit.
$y(i) = W(x_t, x_i^\delta) \quad \forall i \in \{1, \cdots, m\}$
*Estimation of the rewards :*
$a(i) = r^\delta(x_i^\delta, u) \quad \forall i \in \{1, \cdots, m\}$
$a \leftarrow a + \alpha_r(r_t - y^T a)y$
$r^\delta(x_i^\delta, u) = a(i) \quad \forall i \in \{1, \cdots, m\}$
*Estimation of the transition probabilities :*
Do $\forall j = \{1, \cdots, m\}$
    $a(i) = p^\delta(x_j^\delta | x_i^\delta, u) \quad \forall i \in \{1, \cdots, m\}$
    $a \leftarrow a + \alpha_p(W(x_{t+1}, x_j^\delta) - y^T a)y$
    $p^\delta(x_j^\delta | x_i^\delta, u) = a(i) \quad \forall i \in \{1, \cdots, m\}$
*Correction of the transition probabilities ($p^\delta(.|., u_t)$ is constrained to stay in $[0, 1]$) :*
Do $\forall i = \{1, \cdots, m\}$
    $p^\delta(x_j^\delta | x_i^\delta, u_t) \leftarrow \max(0, \min(1, p^\delta(x_j^\delta | x_i^\delta, u_t))) \quad \forall j \in \{1, \cdots, m\}$
Do $\forall i = \{1, \cdots, m\}$
    $sum\_probability = \sum_{j=1}^{m} p^\delta(x_j^\delta | x_i^\delta, u_t)$
    $p^\delta(x_j^\delta | x_i^\delta, u_t) \leftarrow \frac{p^\delta(x_j^\delta | x_i^\delta, u_t)}{sum\_probability} \quad \forall j = \{1, \cdots, m\}$
```

input, checks whether $u_t \in U^\delta(x_t)$ (if $u_t \notin U^\delta(x_t)$ then no structure update is done) and updates $r^\delta(x^\delta, u_t)$ and $p^\delta(.|x^\delta, u_t)$ if $W(x_t, x^\delta) \neq 0$.
For this algorithm to hold valid, $r^\delta(.,.)$ and $p^\delta(.|.,.)$ must be initialized. To this end we proceed as follows : if $W(x_t, x^\delta) \neq 0$ and $r^\delta(x^\delta, u_t)$ has not yet been initialized then $r^\delta(x^\delta, u_t)$ and $p^\delta(x^{\delta'}|x^\delta, u_t)$ are initialized respectively to $r_t$ and $W(x_{t+1}, x^{\delta'})$. This initialization strategy is adopted in order to get the best possible initial guess of the MDP$^\delta$ structure.
We remark the similarity between this algorithm and the one described in figure 6.12. The only difference comes from the fact that the $H(u_t)^{-1}$ matrix is replaced by the scalar $\alpha_{\{r,p\}}$. It implies that the algorithm does not require inverting an $m \times m$ matrix at each $t+1$, and this alleviates considerably the computational burden.
Note that once the estimation of the MDP$^\delta$ structure is realized, the transition probabilities $p^\delta(.|., u_t)$ are corrected in order to ensure that they stay in the interval $[0, 1]$.

**Convergence properties**

Let $\alpha_{\{r,p\}_k}$ be the value of $\alpha_{\{r,p\}}$ at the $k$th stage of the algorithm.
We define $\alpha_{\{r,p\}_k}(u)$ ($\forall u \in C^\delta$) :

- as being equal to $\alpha_{\{r,p\}_k}$ if at the $k$th stage of the algorithm $u = u_t$
- as being equal to 0 if at the $k$th stage of the algorithm $u \neq u_t$

and introduce the following condition :

$$\sum_{k=0}^{\infty} \alpha_{\{r,p\}_k}(u) = \infty, \quad \sum_{k=0}^{\infty} \alpha^2_{\{r,p\}_k}(u) < \infty \quad \forall u \in C^\delta \tag{6.17}$$

We define for each $u \in C^\delta$ the $n \times n$ matrix $F(u)$ :

$$F(u)_{ij} = \underset{x}{E}[W(x, x_i^\delta)W(x, x_j^\delta)|u, P_0(x, u)] \; \forall i, j \in \{1, \cdots, m\}, \tag{6.18}$$

the $n$-vector $r(u)$ :

$$r(u)(i) = \underset{(x,w)}{E}[W(x, x_i^\delta)r(x, u, w)|u, P_0(x, u)] \quad \forall i \in \{1, \cdots, m\} \tag{6.19}$$

and the $n \times n$ matrix $P(u)$ :

$$P(u)_{ij} = \underset{(x,w)}{E}[W(x, x_i^\delta)W(f(x, u, w), x_j^\delta)|u, P_0(x, u)] \; \forall i, j \in \{1, \cdots, m\}. \tag{6.20}$$

The convergence properties of the algorithm described in figure 6.13 are[13] :

- *Input :* one-step episodes starting from an element of $X^\delta$ are used and each state-action pair $(x^\delta, u) \in X^\delta \times U^\delta$ is visited an infinite number of times.
  *Algorithm :* the $\alpha_{\{r,p\}_k}$ are non negative and satisfy condition (6.17).
  *Control problem :* /
  *Solution :* it converges *with probability 1* to the MDP$^\delta$ structure defined by equations (3.14) and (3.15).

- *Input :* one-step episodes are used, each state-action pair $(x_0, u_0)$ is drawn independently according to the probability distribution $P_0(x,u)$ and each action $u \in C^\delta$ is triggered an infinite number of times.
  *Algorithm :* the $\alpha_{\{r,p\}_k}$ are non negative and satisfy condition (6.17).
  *Control problem :* /
  *Solution :* it converges *with probability 1* to the MDP$^\delta$ structure defined as follows :

$$
\begin{aligned}
r^\delta(x_i^\delta, u) &= (F(u)^{-1} r(u))(i)\\
p^\delta(x_j^\delta | x_i^\delta, u) &= (F(u)^{-1} P(u))_{ij}
\end{aligned}
$$

  where the $m \times m$ matrix $F(u)$, the $n$-vector $r(u)$ and the $m \times m$ matrix $P(u)$ are defined respectively by expressions (6.18), (6.19) and (6.20).

### Values of $\alpha_r$ and $\alpha_p$

If $\alpha_{\{r,p\}}$ is chosen equal to

$$
(\frac{1}{N(u)})^\theta \tag{6.21}
$$

where $0.5 < \theta \leq 1$, then condition (6.17) is satisfied. The main disadvantage of this strategy is that the value of $\alpha_{\{r,p\}}$ is only linked to the number of times a control action has been chosen and not to the region of the state space visited. This choice implies in particular that when visiting for the first time a region of the state space, $\alpha_{\{r,p\}}$ can already be small, which may penalize the learning speed.

[13]Strictly speaking, we need to suppose that the algorithm is slightly modified such that the correction on the transition probabilities aiming to ensure that they stay in the interval $[0, 1]$ is not realized anymore.

To avoid this we define the function $N^\delta(x^\delta, u)$ that is initialized to zero everywhere on $X^\delta \times U^\delta$ at the beginning of the learning and updated each time a state-action pair $(x, u)$ is visited as follows :

$$N^\delta(x^\delta, u) \leftarrow N^\delta(x^\delta, u) + W(x, x^\delta) \quad \forall x^\delta \in X^\delta \tag{6.22}$$

By proceeding so, we may interpret the value of $N^\delta(x^\delta, u)$ as being the number of times the action $u$ has been taken while being in the state $x^\delta$. Then, if $\alpha_{\{r,p\}}$ is chosen for example equal to

$$\max(1, (\frac{1}{\sum_{x^\delta \in X^\delta} W(x, x^\delta) N^\delta(x^\delta, u)})^\theta), \tag{6.23}$$

it will depend not only on the number of times the action $u$ has been taken but also on the "number of times the region around $x$ has been visited".

**Obtaining the same estimation as with the Kalman Filter like algorithm**

Suppose that we keep in a set all the four-tuples $(x_t, u_t, r_t, x_{t+1})$ obtained during the learning. Suppose that each element of this set has the same probability to be selected. If we repeat an infinite number of times the sequence of operations that consists first in selecting an element of the set and then in using it as input of the *Stochastic Approximation algorithm* we will converge (if the MDP$^\delta$ structure that minimizes (6.12) and (6.13) is unique) to an estimate of the MDP$^\delta$ structure that is identical to the one obtained by using a *Kalman Filter like algorithm*. In the above procedure we suppose that the algorithm is slightly modified so that the correction on the transition probabilities terms aiming to ensure that they stay in the interval $[0, 1]$ is not realized anymore and that $\beta = 1$.

## 6.7 Non-model based representative states technique

This section presents algorithms which update the $Q^\delta$-function without explicitly reconstructing any MDP$^\delta$ structure. The value of the approximate $Q$-function will be deduced from the $Q^\delta$-function by using equation (3.16) that we remind hereafter :

$$\tilde{Q}(x, u) = \sum_{x^\delta \in X^\delta} W(x, x^\delta) Q^\delta(x^\delta, u) \quad \forall x \in X, \forall u \in U^\delta(x). \tag{6.24}$$

Figure 6.14: $Q^\delta$*-function correction* : $Q$-learning used with the representative states technique

```
**Input :** $x_t$, $u_t$, $r_t$ and $x_{t+1}$
If $u_t \notin U^\delta(x_t)$ then exit.
$\delta \leftarrow (r_t + \gamma \max_{u \in U^\delta(x_{t+1})} \tilde{Q}(x_{t+1}, u)) - \tilde{Q}(x_t, u_t)$
Repeat for all $x^\delta \in X^\delta$
    $Q^\delta(x^\delta, u_t) \leftarrow Q^\delta(x^\delta, u_t) + \alpha\delta W(x_t, x^\delta)$
```

## 6.7.1 $Q$-learning algorithm

The algorithm is described on figure 6.14. At each $t+1$ it takes the four-tuple $(x_t, u_t, r_t, x_{t+1})$ as input and updates $Q^\delta(x^\delta, u_t)$ for all $x^\delta \in X^\delta$ for which $W(x_t, x^\delta) \neq 0$ leaving the other components of the $Q^\delta$-function unchanged.
This algorithm is an immediate consequence of the advantage updating method described in section 6.1.2 (consider equation (6.3) and observe that according to equation (6.24) we have $\frac{\partial \tilde{Q}(x,u)}{\partial Q^\delta(x^\delta,u)} = W(x, x^\delta)$).

### Convergence properties

Let $\alpha_k$ be the value of $\alpha$ at the $k$th stage of the algorithm.
We define $\alpha_k(x_i^\delta, u)$ ($\forall (x_i^\delta, u) \in X^\delta$) :

- as being equal to $\alpha_k$ if at the $k$th stage of the algorithm, $u = u_t$ and $x_t = x_i^\delta$
- as being equal to 0 if at the $k$th stage of the algorithm, $u \neq u_t$ or $x_t \notin x_i^\delta$.

The convergence properties of the algorithm described in figure 6.14 are :

- *Input :* one-step episodes starting from an element of $X^\delta$ are used and each state-action pair $(x^\delta, u) \in X^\delta \times U^\delta$ is visited an infinite number of times.
  *Algorithm :* the $\alpha_k(x^\delta, u)$ are non negative and satisfy condition (6.11).
  *Control problem :* solution of the MDP$^\delta$ whose structure is defined by equations (3.14) and (3.15) satisfies the equivalence condition (3.21).
  *Solution :* it converges *with probability 1* to the $Q^\delta$-function corresponding to the MDP$^\delta$ defined by equations (3.14) and (3.15).
  The proof is given in section C.8.

Figure 6.15: *$Q^\delta$-function correction* : Watkins's $Q(\lambda)$ used with the representative states technique

```
**Input :** $x_t$, $u_t$, $r_t$, $x_{t+1}$ and $u_{t+1}$
If $u_t \notin U^\delta(x_t)$ then exit.
$\delta \leftarrow (r_t + \gamma \max\limits_{u \in U^\delta(x_{t+1})} \tilde{Q}(x_{t+1}, u)) - \tilde{Q}(x_t, u_t)$
Do $\forall x^\delta \in X^\delta$
    $e(x^\delta, u_t) \leftarrow e(x^\delta, u_t) + W(x_t, x^\delta)$
Do $\forall (x^\delta, u) \in X^\delta \times U^\delta$
    $Q^\delta(x^\delta, u) \leftarrow Q^\delta(x^\delta, u) + \alpha \delta e(x^\delta, u)$
    If $\tilde{Q}(x_{t+1}, u_{t+1}) = \max\limits_{u' \in U^\delta(x_{t+1})} \tilde{Q}(x_{t+1}, u')$ then $e(x^\delta, u) \leftarrow \gamma \lambda e(x^\delta, u)$, else
    $e(x^\delta, u) = 0$
```

- *Input :* one-step episodes starting from an element of $X^\delta$ are used and each state-action pair $(x^\delta, u) \in X^\delta \times U^\delta$ is visited an infinite number of times.
  *Algorithm :* the $\alpha_k(x^\delta, u)$ are non negative and satisfy condition (6.11).
  *Control problem : /*
  *Solution :* it converges *with probability 1* to the $Q^\delta$-function which terms are the unique solution of the system of equations :

$$Q^\delta(x_i^\delta, u) = r^\delta(x_i^\delta, u) + \gamma \max_{u' \in U^\delta(Reach(x_i^\delta, u))} \sum_{x_j^\delta \in X^\delta} p^\delta(x_j^\delta | x_i^\delta, u) Q^\delta(x_j^\delta, u') \ \forall (x_i^\delta, u) \in X^\delta \times U^\delta$$

  where $r^\delta(x_i^\delta, u)$ and $p^\delta(x_j^\delta | x_i^\delta, u)$ are defined respectively by equations (3.14) and (3.15) and $Reach(x_i^\delta, u)$ denotes an element of $X$ that can be reached when taking action $u$ in state $x_i^\delta$.
  The proof can be deduced from the reasoning done in section C.8 to prove the preceeding convergence property.

### 6.7.2 $Q(\lambda)$ algorithm

Iterations (6.6) and (6.7) used with $\tilde{Q}$ defined by using a representative states technique (equation (3.16)) lead to the algorithm defined in figure 6.15 where the function $e$ represents the eligibility trace[14]. This algorithm differs nonetheless slightly from the one suggested by iterations (6.6) and (6.7) by setting the eligibility trace to zero each time that a non-greedy action is chosen. This feature has been discussed when introducing the $Q(\lambda)$ algorithm (section 5.4.3).
For this algorithm to hold valid, one should initialize the eligibility trace $e$ to zero at the beginning of each episode.

**Example 6.7.1** We describe in this example an application of the reinforcement learning algorithms used with the representative states technique. The control problem considered here has been detailed in example 3.2.2. In example 3.3.2, we treated the same control problem by assuming that the reward function and the system dynamics were known and by using the representative states technique.

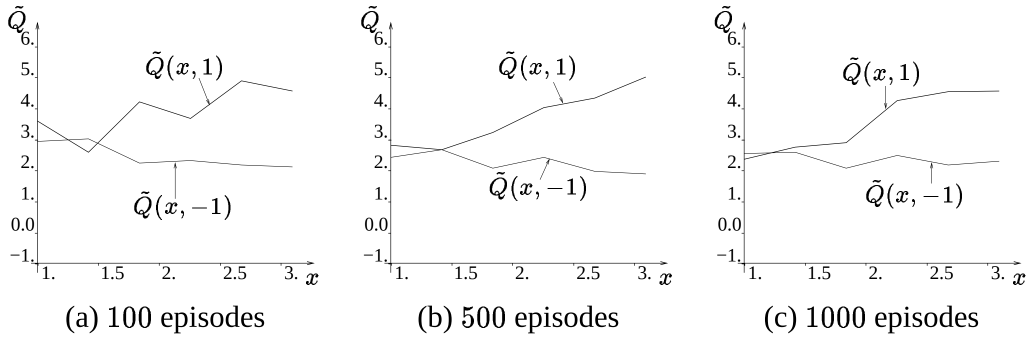
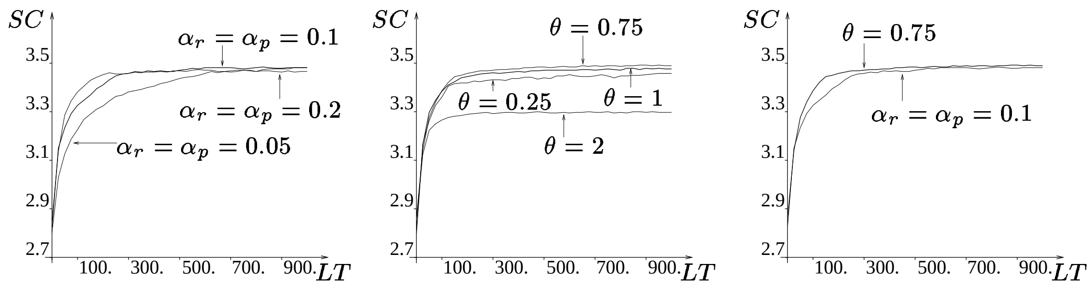


Figure 6.16: Representation of $\tilde{Q}$ at different stages of the learning process

The control problem state space is composed of $[1,3] \cup x^t$. We consider seven representative states : $1, 1.4, 1.8, 2.2, 2.6, 3$ and $x^t$. The control space being infinite we consider just a finite subset $C^\delta = \{-1,1\}$ and choose $U^\delta(x^\delta) = C^\delta$ for all $x^\delta \in X^\delta \setminus \{x^t\}$. The $W$ function is defined by using a triangulation technique.
The *Action Selection* module is composed of an $\epsilon$-Greedy policy with $\epsilon = 0.1$. As *Learning* module we use a model based algorithm for which the *Estimation of the MDP$^\delta$ structure* is done by using the Stochastic Approximation algorithm. The parameters $\alpha_r$ and $\alpha_p$ used to estimate the rewards and the transition probabilities (figure 6.13) are chosen both constant

[14]Remark that with the representative states technique the vector $\frac{\partial \tilde{Q}(x,u,a)}{\partial a}$ does not depend on $a$ anymore (its components can only be equal to $W(x,x_1^\delta)$, $W(x,x_2^\delta)$, $\cdots$, $W(x,x_m^\delta)$ or 0); therefore the algorithm described by iteration (6.5) modifies $a$ exactly like the algorithm described by iterations (6.6) and (6.7).

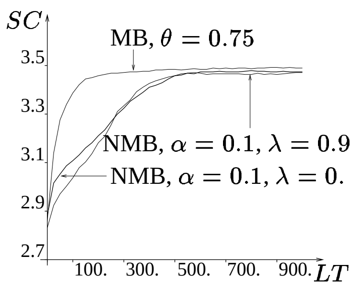
and equal to $0.1$. The *Resolution of the MDP$^\delta$* module consists in solving at each time step $t+1$ the MDP$^\delta$ by using a Gauss-Seidel version of the value iteration algorithm.
The initial state for each episode is chosen at random in the interval $[1,3]$ and an episode stops only when the terminal state $x^t$ is reached.
In such conditions the $\tilde{Q}$ functions we have obtained after $100$, $500$ and $1000$ episodes of learning are respectively represented on figures 6.16a, 6.16b and 6.16c. These representations can be compared to the one obtained by using the representative states technique when assuming that the reward function and the system dynamics are known (figure 3.10b). It is important to note that due to the fact that $\alpha_r$ and $\alpha_p$ were chosen constant, the estimation of the MDP$^\delta$ structure will not converge. This induces continual variations of the $Q^\delta$-function and thus also of $\tilde{Q}$.



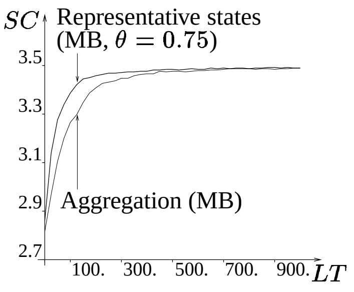
(a) $\alpha_r$ and $\alpha_p$ constant (b) $\alpha_r$ and $\alpha_p$ not constant (c) Constant vs not constant

Figure 6.17: Score curves for different variants of the *Learning* module.

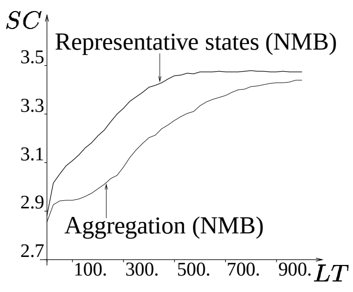
The influence of the values of $\alpha_r$ and $\alpha_p$ on the learning is illustrated on figure 6.17a. The larger their values ($\alpha_r = \alpha_p = 0.2$) the faster the learning is at the beginning. However the score obtained at the end of the learning period i.e., after $1000$ episodes, is slightly better if the values $\alpha_r$ and $\alpha_p$ are chosen small ($\alpha_r = \alpha_p = 0.05$). These observations motivate us to use values of $\alpha_r$ and $\alpha_p$ not constant anymore during the learning period but large at the beginning and decreasing with the learning time. For example, we represent on figure 6.17b score curves corresponding to $\alpha_r$ and $\alpha_p$ values defined by equation (6.23). The parameter $\theta$ indicates the rate of decrease of these values. A value of $\theta$ larger than $1$ (here $2$) leads to a very bad convergence of the algorithms because it drives $\alpha_r$ and $\alpha_p$ to decrease too rapidly. On the other hand when $\theta$ is chosen equal to a small value (see the curve for which $\theta = 0.25$) the decrease of $\alpha_r$ and $\alpha_p$ can not be sufficient to obtain the best performance of the *Learning* module. Among the four values of $\theta$ considered ($0.25$, $0.75$, $1.$ and $2$), the best results were obtained with $\theta = 0.75$. Figure 6.17c compares the performance of both strategies, $\alpha_r$ and $\alpha_p$ constant or decreasing with the learning time. The best results are observed when non-constant values are used for $\alpha_r$ and $\alpha_p$. Indeed, this allows both a fast learning at the beginning of the learning period and an excellent score value at its end.
We can also compare the performances of the model based and the non-model based algorithms used with the representative states technique. This is illustrated on figure 6.18a



(a) MB and NMB Representative states technique



(b) MB algorithms Representative states vs aggregation technique



(c) NMB algorithms Representative states vs aggregation technique

Figure 6.18: Score curves

where three score curves are displayed. The best score curve is once again obtained by using a model based technique. The performances of the two non-model based techniques described in this chapter are similar with however a slight advantage for the $Q$-learning algorithm (no eligibility trace) notably at the beginning of the learning.

In this example we have illustrated reinforcement learning algorithms used with the representative states technique; one may wonder whether the results thus obtained are better or worse than those obtained when the reinforcement learning algorithms are used with the aggregation technique. The comparison is difficult because for the aggregation technique the performances of the algorithms depend notably on the way $X$ is discretized and for the representative states technique on the states selected from $X$. However, in order to attempt some kind of answer, we have drawn on figure 6.18b the score curves obtained for model based reinforcement learning algorithms used with the aggregation technique (curve that was previously drawn on figure 6.11a and labeled "Model based, KF") and with a representative states technique (curve that was previously drawn on figure 6.17b and labeled "$\theta = 0.75$"). The performances of model based RL algorithms are better when used with the representative states technique than with the aggregation technique. Note that an opposite conclusion could be reached if the tuning of parameters $\alpha_r$ and $\alpha_p$ is inadequate. By drawing a similar graphic, related this time to the non-model based algorithm, we obtain the figure 6.18c. This figure also highlights the better performances of RL algorithms when used with the representative states technique.

**Example 6.7.2** In this example we focus on the Kalman Filter like algorithm used with the representative states technique (figure 6.12). The control problem, $X^\delta$, $C^\delta$, the *Action* module, the *Resolution of the MDP*$^\delta$ module and the learning conditions are chosen identical to those previously chosen in example 6.7.1.

The matrix $H(u)$ is initialized to $0.001I$ $\forall u \in C^\delta$ where $I$ is the $\#X^\delta \times \#X^\delta$ identity matrix. When used to compute the $Q^\delta$-function the estimated transition probabilities are

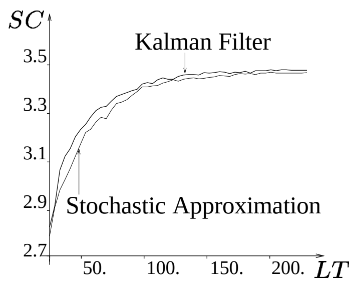


Figure 6.19: Kalman Filter like algorithm vs Stochastic Approximation algorithm

corrected in order to ensure that $0 \leq p^{\delta}(x^{\delta}_{j}|x^{\delta}_{i},u) \leq 1$ and $\sum_{j=1}^{m} p^{\delta}(x^{\delta}_{j}|x^{\delta}_{i},u) = 1$. The correction is carried out by using a similar procedure as the one used in figure 6.13.
By proceeding so we obtain the score curve labeled "Kalman Filter" represented on figure 6.19. The other score curve (labeled "Stochastic Approximation") drawn on this figure represents the score evolution when the Stochastic Approximation algorithm is used to estimate the MDP$^{\delta}$ structure (with $\alpha_{\{r,p\}}$ given by equation (6.23) and $\theta = 0.75$). The Kalman Filter like algorithm gives slightly better results.

## 6.8 Summary

*In this chapter we have seen that it is possible to compute approximations of the optimal stationary policy for systems with infinite state spaces when the system dynamics and the reward function are unknown. Broadly, the strategy used to this end consists in assuming that the initial infinite state space control problem could be represented more or less equivalently by a finite Markov Decision Process (MDP$^{\delta}$ in short) and in using reinforcement learning algorithms to learn either the structure of this MDP$^{\delta}$ (model based algorithms) or directly its $Q^{\delta}$-function (non-model based algorithms).*
*Eight algorithms that result from the combination of two different approximation architectures with two model based and two non-model based reinforcement learning strategies have been described and discussed using a unified framework. For most of these algorithms convergence properties have been stated and the intrinsic relationships among the different methods have been highlighted. We have also shown that it was possible to apply the Bellman error methods in this context, and have suggested some further directions to make them practically useful.*
*We would like to stress that the class of optimal control problems is a very rich set of problems and we do not believe that any of these algorithms could pretend to*

*be superior to the others in general. Rather, we observe that the paradigm of reinforcement learning offers a rich set of approaches among which we should choose the most appropriate one to a particular practical problem. While for a given problem it is not possible to decide a priori which of the presented approaches will be more efficient in terms of learning speed or quality of the solution obtained, it is always possible to select some method based on other a priori considerations such as the amount of computational resources available and the ease of implementation or the freedom one has in terms of state space sampling.*

*In the next chapter and in chapter 9 we will apply some of these algorithms to practical power system control problems in order to highlight their practical usefulness. We will see that with the developed methodology we are indeed able to compute from interaction with an infinite state space system a control law that perhaps does not converge to the optimal stationary policy but can still give a good approximation of it, with acceptable performance in a rather broad set of conditions.*

# Chapter 7

# Revisit of a simple FACTS control problem

*In this chapter we apply some of the reinforcement learning algorithms described in chapter 6 to the power system control problem studied in chapter 4. We consider both model based and non-model based algorithms combined with either the aggregation technique or the representative states technique.*

*The simulations are carried out using an ad hoc time domain simulation software that we developed on purpose for this research and which is coupled with our reinforcement learning software. The reinforcement learning algorithms are combined with an $\epsilon$-greedy control policy with a small value of $\epsilon$ which in principle would allow using these algorithms in interaction with a real power system, once a reasonable control policy has been tuned off-line during simulations.*

*The conditions under which these algorithms are used are kept identical to those used in chapter 4 (sampling period, power system model parameters, discretization grids) in order to allow us to compare the solution thus obtained with the previously obtained ones.*

*We would like to stress that the simulations provided here will be complemented in chapter 9 by much more extensive ones under more realistic conditions. Actually, chapter 9 aims to demonstrate the practical feasibility of these approaches in the context of power system control problems, while the simulations performed in the present chapter aim at providing more insight into the behavior of the different algorithms of chapter 6.*

## 7.1 Reinforcement learning and the aggregation technique

This section describes the application of the reinforcement learning algorithms combined with the aggregation technique and discusses the results obtained for the optimal control problem (discrete-time system model and reward definition) formulated in section 4.2.

The first part of the procedure aims to partition the state space $X$ into a finite number of disjoint subsets $X_i$ and to define the finite sets $C^\delta$ and $U^\delta(x)$ such that $C^\delta \subset C$ and $U^\delta(x) \subset U(x)$. We choose a $50 \times 50$ grid to partition $X \setminus \{x^t\}$ and to consider that $x^t$ is the only element of a subset $X_i$ (see chapter 4 for more details). Concerning the control sets, we have chosen them such that $C^\delta = \{-0.04, 0.\}$ and $U^\delta(x) = C^\delta$, $\forall x \in X \setminus \{x^t\}$.

The second part of the procedure consists in defining the content of the *Learning* module used in the reinforcement learning algorithm. Two families of *Learning* modules can be used. The first is related to the model based techniques and the second one to the non-model based techniques. We will use first a model based *Learning* module and compare later on the results thus obtained with those provided by non-model based techniques.

### 7.1.1 Model based techniques

#### MDP$^\delta$ structure learning

We have first to decide what kind of algorithm we are going to use as *Estimation of the MDP$^\delta$ structure* module. We have decided to choose the Kalman Filter like algorithm detailed in figure 6.6.

#### Dynamic programming algorithm

The use of a model based reinforcement learning algorithm implies defining a method that solves the MDP$^\delta$ (definition of the *Resolution of the MDP$^\delta$* module). If the MDP$^\delta$ is solved completely at each step of the learning by using for example the Gauss-Seidel version of the value iteration algorithm (see figure 2.4), the computational burden can become excessive. So we rather decide to use the Prioritized sweeping algorithm defined in figure 5.6 (with $iter_{max}$ equals to 10 and $\sigma = 0.0001$) to solve only partially the MDP$^\delta$ during an episode. At the end of an episode, we use the Gauss-Seidel version of value iteration algorithm to solve the MDP$^\delta$ completely. The value of the $Q^\delta$-function is initialized to zero everywhere.

## Action selection module

The third part of the procedure must result in the definition of the *Action Selection* module. The *Action Selection* module we choose is composed of an $\epsilon$-Greedy policy with $\epsilon = 0.01$. The $\epsilon$ value has intentionally been chosen small in order to proceed as if the RL algorithms were interacting with a real power system for which it would be important to exploit almost at its best the approximation of the optimal policy computed during the learning. An $\epsilon$-Greedy policy with a larger value of $\epsilon$ would tend to speed up the learning while reducing the use it does of the already learned control law.
Note that with an $\epsilon$-Greedy policy as *Action selection* module and since the $Q^\delta$-function is initialized to zero everywhere, the action taken is chosen at random in $C^\delta$ when the subset $X_i$ is visited for the first time (figure 6.3).

## Learning protocol

The learning protocol is defined by the way episodes are initialized and terminated, and by the sampling rate used by the learning algorithms and control module.
As in chapter 4, a sampling period of $50\,ms$ is used for the control module, and the reinforcement learning module is called at the same rate[1]. Note that this sampling period is not directly related to the time-step used by the underlying time-domain simulation program, although it obviously defines an upper bound on this latter.
Concerning an episode of the learning, the initial state of each episode is in our simulations chosen at random with a uniform probability distribution among the states of $X \setminus \{x^t\}$.
An episode is terminated either if a terminal condition is reached (the state of the system leaves the stability domain) or if a maximum number of steps is reached. In these simulations the maximum number of steps is limited to 1200, which corresponds to 60 seconds of real-time.

## Discussion of results

Table 7.1 summarizes the quality of the control policies obtained by the reinforcement learning method as a function of the number of episodes used (see figure

[1]In our presentation of reinforcement learning algorithms we have implicitly assumed that these rates are always identical. Nevertheless, most of these algorithms could easily be adapted so as to reduce the rate at which they are called with respect to the rate at which the control module is called. This could be useful in situations where real-time constraints are incompatible with the amount of computational resources available.

4.10a and section 4.4 for a reminder on how the scores are computed and for the geometrical representations of the domains $X \setminus \{x^t\}$, $X'$ and $X''$).

Table 7.1: Score for different $P_0(x)$ as a function of the learning time

| $LT$ | score | | |
|---|---|---|---|
| | $X \setminus \{x^t\}$ | $X'$ | $X''$ |
| 10 | -66.1966 | -25.1192 | -5.8464 |
| 500 | -49.0173 | -8.9007 | -1.3772 |
| 1000 | -46.8442 | -7.9074 | -1.3845 |
| 5000 | -46.5598 | -7.4852 | -1.3681 |
| 10000 | -46.4072 | -7.3692 | -1.3193 |
| 100000 | -45.2974 | -7.3159 | -1.3189 |

The different scores computed after 10, 500, 1000, 5000, 10000 and 100000 episodes yield the following observations and explanations.

- At first the learning of the control law is faster in the domain $X''$ than in $X \setminus \{x^t\}$. Indeed after 500 episodes the score related to $X''$ has almost converged contrary to the score related to $X \setminus \{x^t\}$. We explain this by the fact that the trajectories of the system tend to converge to this small $X''$ region centered on the stable equilibrium point of the system as soon as the learning module is able to learn something about how to stabilize the system. Therefore the subsets $X_i$ that belong to this region are visited much more often and the learning is faster there.

  Notice that this observation has also led us to limit an episode duration to a relatively small time (chosen here equal to 60 seconds) and to choose as initial state of an episode a state drawn at random in $X \setminus \{x^t\}$. If such a strategy was not used, for example if an episode ended only when a terminal state was reached, the states far from the stable equilibrium point of the system would be visited still more rarely and the learning in these areas would be even slower. Note that rather than defining time bounded episodes, one could use a value of $\epsilon$ much larger in the $\epsilon$-Greedy policy. Indeed, with such a choice the control law would be much more different from the optimal one and would even drive the system far from the stable equilibrium point of the system where bad rewards are observed.

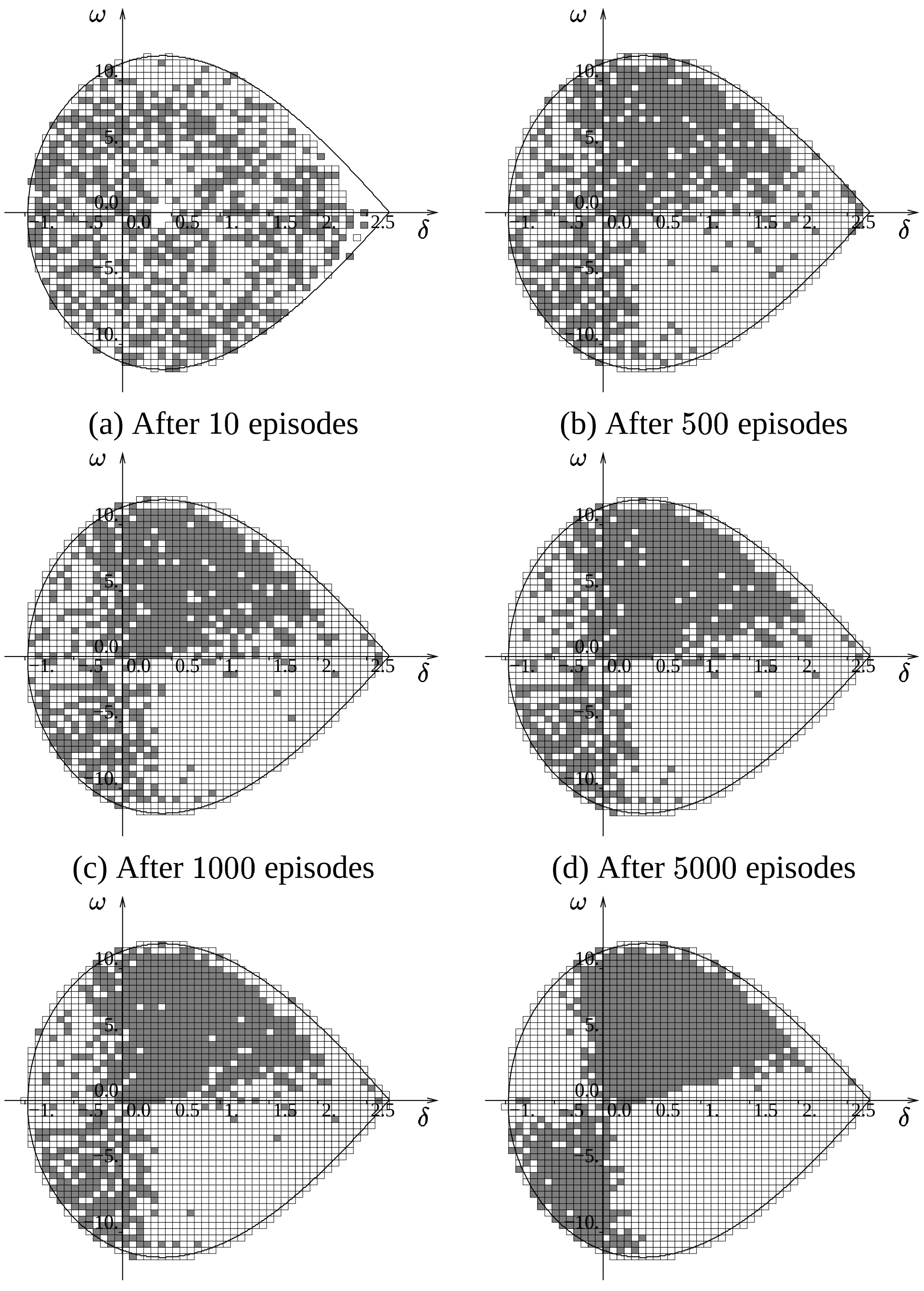
- The second important observation drawn from table 7.1 is the impressive

> quality of the control law observed by the end of the learning period i.e., after $100,000$ episodes. Comparing these scores to the results obtained with the aggregation technique where the system dynamics and the reward function were known (table 4.1) for which the three score values corresponding to the domains $X \setminus \{x^t\}$, $X'$ and $X''$ are respectively $-45.4948$, $-7.7084$ and $-1.9856$, one can see that the policy obtained after $100,000$ episodes by using the reinforcement learning technique is even better.
>
> The reason is probably linked to the high number of episodes used in the reinforcement learning procedure. Indeed, the $100,000$ episodes used can roughly be considered to represent approximately $1200*100,000 = 120,000,000$ one-step episodes. This has to be compared to the $1,000,000 * \#C^\delta = 2,000,000$ carefully chosen one-step episodes that have been used in chapter 3 to compute the MDP$^\delta$ structure. Of course for the same number of one-step episodes the procedure used in chapter 3 should be better, in particular because each action is used the same number of times in each subset $X_i$ and the number of times each subset is visited is independent of whether or not it is close to the stable equilibrium point of the OMIB system.

On figure 7.1 we have represented the control law obtained at different stages of the learning process. The gray tiles represent areas of the state space where the value of the control variable $u$ is chosen equal to $-0.04$ (the FACTS acts at full range of its capacitance) while the white areas represent areas of the state space in which the FACTS capacitance is chosen equal to $0$. Notice that in figure 7.1a, there exist areas of the state space $X$ without tiles. These areas correspond to regions of the state space that have not yet been visited during the learning and thus where nothing can be stated about the control variable value. As the learning process proceeds, one can observe that the organization of the tiles becomes better and better. These six figures can be compared to figure 4.5a where a similar control law was represented but computed by assuming that the system dynamics and the reward function were known.

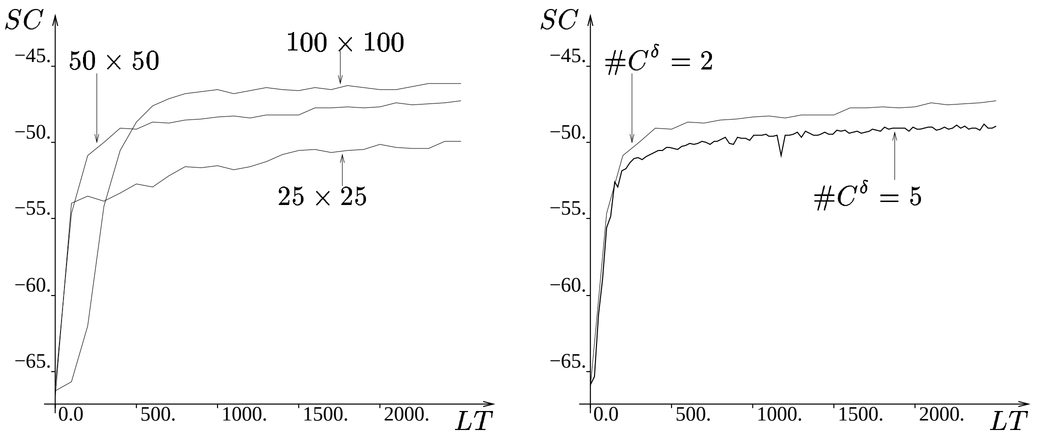
### Influence of the discretization grid size on convergence

One can wonder how the discretization of $X$ (the way the $X_i$ are chosen) influences the learning process. It must be noted that the number of states composing the MDP$^\delta$ will be higher if the $X_i$ are chosen smaller. Therefore the amount of information needed to reconstruct the MDP$^\delta$ structure will in principal also be higher. Thus we can expect that to a finer discretization of the state space will correspond a slower learning of the control law with the possibility that by the end the quality



(a) After 10 episodes
(b) After 500 episodes
(c) After 1000 episodes
(d) After 5000 episodes
(e) After 10000 episodes
(f) After 100000 episodes

Figure 7.1: Control law at different stages of the learning process



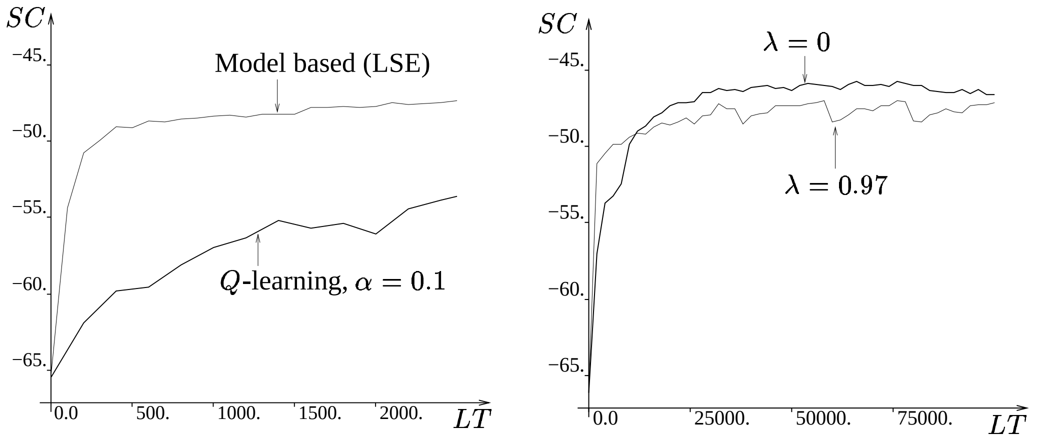
(a) Influence of the discretization (b) Influence of the control set

Figure 7.2: Score curves for RL algorithms used with the aggregation technique

of the control law observed will be higher.
The actual behavior is highlighted on figure 7.2a where the score curves are drawn for three different discretizations of the state space. These score curves can seem to be surprising in the sense that they illustrate that a finer discretization does not tend to penalize too much the learning speed. Indeed if one compares for example the $25 \times 25$ curve with the $50 \times 50$ curve, it is impossible to state firmly that even at the very beginning of the learning the rougher the discretization of the state space, the better the control law quality. But the comparison is in fact misleading due to the type of policy used ($\epsilon$-Greedy policy with a very small value of $\epsilon$) and the way the $Q^\delta$-function is initialized ($Q^\delta$-function is initialized to 0 everywhere). Since the rewards can only be negative, these two choices make the policy prefer an action that has never been triggered before while being in the subset $X_i$ (if during the learning $x_t \in X_i$ and if action $u$ has never been triggered before while being in $X_i$ then $\tilde{Q}(x, u) = 0$). This kind of policy behavior indirectly provides a lot of exploration in areas having a fine discretization and thus tends to favour, at least at the first stages of the learning process, the quality of the control law learned.

### Influence of the control space size

As far as the influence of the control set $C^\delta$ is concerned, figure 7.2 clearly shows that a larger control set is unfavourable to the learning speed of the algorithms because of the higher complexity it induces in the structure of the MDP$^\delta$. Nevertheless, it can be expected that after a certain learning time the results obtained with



(a) MB vs NMB (b) $Q$-learning vs $Q(\lambda)$

Figure 7.3: Score curves for RL algorithms used with the aggregation technique

a more complex control set should improve.

### 7.1.2 Model based vs non-model based RL algorithm

Till now, the type of *Learning* module we have used belonged to the model based techniques family, the family of techniques that reconstruct the structure of a MDP$^\delta$ in order to deduce from it the value of the $Q^\delta$-function. One can wonder how the non-model based techniques perform on the OMIB control problem. This is illustrated on figures 7.3a and 7.3b. Figure 7.3a shows that model based algorithms learn faster (as it was already the case in all the other examples treated in this work). Figure 7.3b illustrates the performances of a non-model based algorithm used with an eligibility trace. The score obtained at the beginning of the learning is better when an eligibility trace is used ($\lambda = 0.97$) even if the tendency reverses itself after approximately $10,000$ episodes. Concerning the score curve related to the use of an eligibility trace, it can also be observed that its evolution is less smooth than when no eligibility trace is used ($\lambda = 0$). This less smooth aspect is notably because the subsets of $X$ corresponding to the tiles that intersect with the stability boundary of the OMIB system represent smaller regions of the state space. This indeed implies that these subsets are less often visited (a phenomenon made even worse by the fact they stand far from the stable equilibrium point of the system) and that "their learning speed" is extremely slow. These slow learning speed regions can induce, when eventually visited, a large temporal difference $\delta$ (figure 6.8) even if the $Q$-function is already well approximated in the other regions of the state space. This large temporal-difference $\delta$ is used in the non-model based algorithm to correct the $Q^\delta$-function and its too large magnitude corrupts the already well estimated $Q^\delta$-function. This phenomenon is amplified by the use of the eligibility trace because the $\delta$ factor computed intervenes in the correction of many $Q^\delta(x^\delta, u)$ terms rather than one when $\lambda = 0$. Therefore it is not surprising to observe a non-smooth evolution of the score curve corresponding to $\lambda = 0.97$.

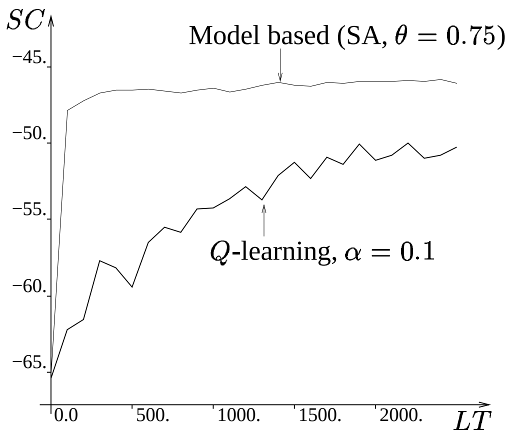
## 7.2 Representative states technique

In this section we describe application to the FACTS control problem of reinforcement learning algorithms combined with the representative states technique.

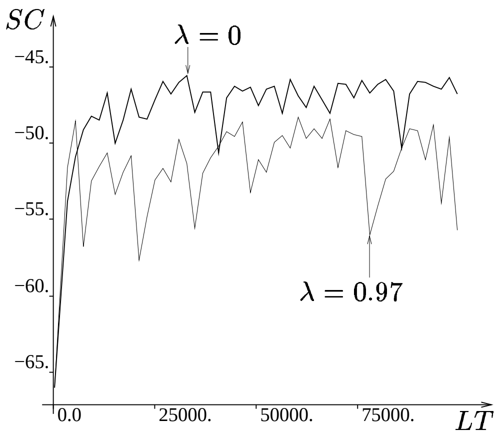
### 7.2.1 Procedure

We define $X^\delta$ by selecting from $X$ the states that stand at the center of the tiles defined by a $50 \times 50$ grid (plus of course the state $x^t$). The function $W$ is defined by using a triangulation technique. The triangulation of the state space is identical to the one described in figure 4.3b. We choose $C^\delta = \{-0.04, 0\}$ and $U^\delta(x) = C^\delta\, \forall x \in X \setminus \{x^t\}$.
The *Action Selection* module is composed of an $\epsilon$-Greedy policy with $\epsilon = 0.01$.
A learning episode lasts $60\, s$ and the initial state of each episode is chosen at random among $X \setminus \{x^t\}$.
The scores of the obtained control policies are evaluated in the same fashion as before.
As *Learning* module we adopt a model based algorithm such that the *Estimation of the MDP$^\delta$ structure* module is composed of the Stochastic Approximation algorithm defined on figure 6.13 (with $\alpha_r$ and $\alpha_p$ computed by using equation (6.23) with $\theta = 0.75$) and such that the *Resolution of the MDP$^\delta$* module is composed of a mix between the Prioritized sweeping algorithm and a Gauss-Seidel version of the value iteration algorithm (which is only used at the end of an episode).
Notice that the Stochastic Approximation algorithm rather than the Kalman Filter like algorithm is used here because in the context of the representative states technique the latter algorithm leads to much higher computational burden than in the aggregation technique. Indeed, its use on this OMIB control problem would require (with the choice of $X^\delta$ realized) inverting at each time step a $1891 \times 1891$ matrix which would lead to a (too) high computational burden.[2]

[2]It can be shown that the matrices $H(u)$(see figure 6.12) would remain sparse during the learning. Indeed $H(u)_{ij}$ (the term of the matrix $H(u)$ that stands at the intersection of the $i$th line and the $j$th



(a) MB vs NMB



(b) $Q$-learning vs $Q(\lambda)$

Figure 7.4: Score curves for RL algorithms used with the representative states technique

### 7.2.2 Resulting score curves

Under these learning conditions, we obtain the score curve represented on figure 7.4a and labeled "Model based (SA, $\theta = 0.75$)". On the same figure is also represented the score curve corresponding to a *Learning* module made up of a non-model based algorithm (curve labeled "$Q$-learning, $\alpha = 0.1$"). We notice that, once again, the model based algorithm offers better performances than the non-model based one.

Score curves corresponding to the non-model based algorithms used either with or without the eligibility trace are represented on figure 7.4b.

As for aggregation technique, the score curve related to the use of the eligibility trace ($\lambda = 0.97$) has a harsher evolution than the one that corresponds to $\lambda = 0$. This phenomenon has already been noticed but one can wonder why it becomes more important than with the aggregation technique and why the score evolution is not smooth anymore even when $\lambda = 0$ (see for example figure 7.6b where the score curves are represented for the aggregation and the representative states technique).

column) can become different from zero only if there exists a state $x \in X$ such that $W(x, x_i^\delta)$ and $W(x, x_j^\delta)$ are both different from zero. With the choice of $W$ realized in the present simulations, the number of terms per line (and column) that can be different from zero is at maximum $7$. Algorithms exploiting the symmetrical and sparse aspect of the matrices $H(u)$ could therefore be used to lighten the computational burden associated to the determination of their inverse. However such algorithms were not implemented in the framework of this research.

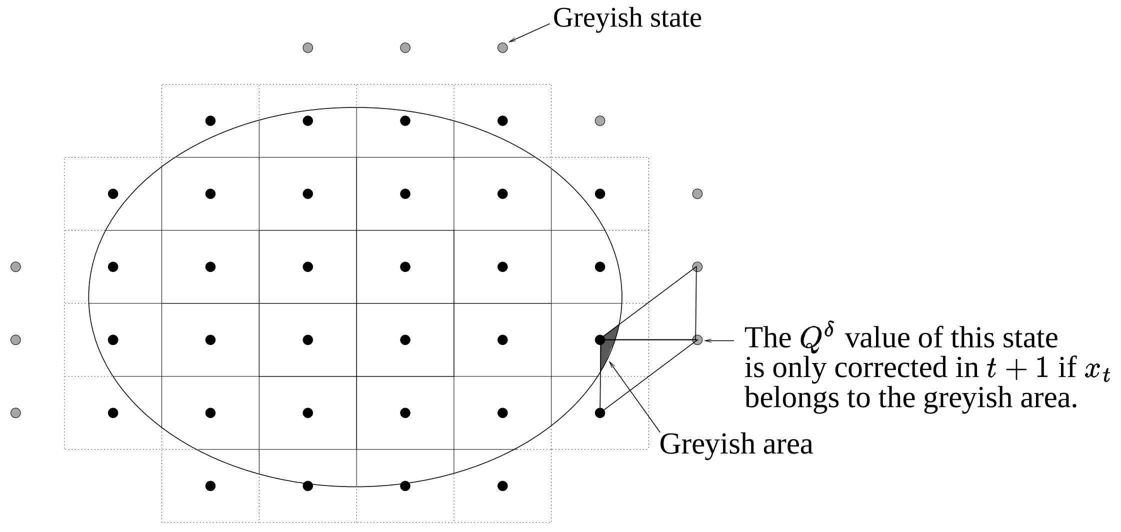
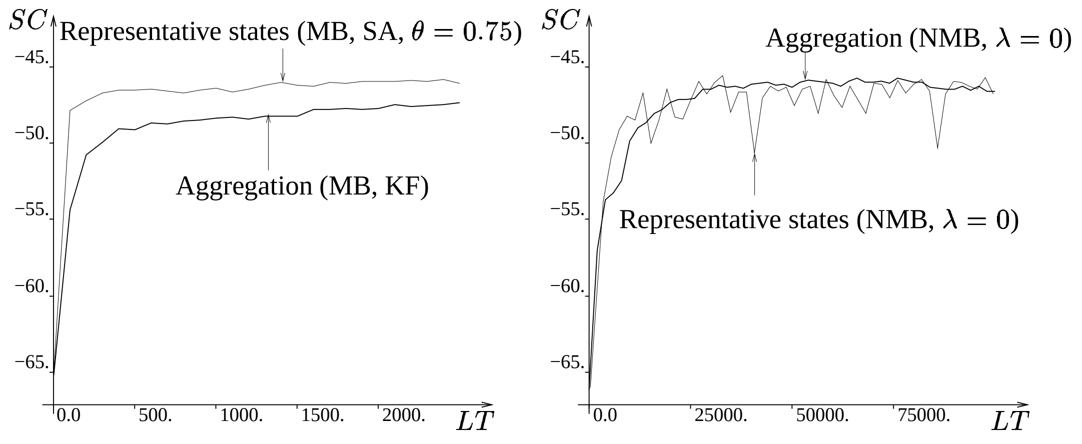

Greyish state
The $Q^\delta$ value of this state is only corrected in $t+1$ if $x_t$ belongs to the greyish area.
Greyish area

Figure 7.5: Problem with the representative states technique and the boundary

The explanation that we found is highlighted on figure 7.5, on which the tiles represent the subsets $X_i$ used with the aggregation technique while the bullets represent the states of $X$ that can be associated with elements of $X^\delta$ when the representative states technique is used. Note that some of these states are located outside $X$ because the simplices defined by triangulation technique have to cover the whole state space (section 3.3.6). Each black bullet can be associated with a subset $X_i$ while the grayish bullets cannot. These grayish bullets correspond to states of the MDP$^\delta$ defined by using the representative states technique while they do not have a corresponding role for the aggregation technique. The learning speed for these states is therefore extremely small because their $Q^\delta$ values are very rarely corrected (see the geometrical interpretation sketched on figure 7.5). But once they intervene in the learning process they corrupt the already well estimated $Q^\delta$ values of the other states and so foil the estimated optimal control law which leads to a sudden decrease of the score curve. We note that the number of such perturbing (grayish) states created when using a $50 \times 50$ grid is equal to 69, which seems small, compared to the total number of states of the MPD$^\delta$ (1891), but is obviously large enough to further destroy the quality of convergence of the non-model based methods applied to the simple FACTS control problem.



(a) Model based algorithms (b) Non-model based algorithms

Figure 7.6: Comparison between RL algorithms used either with the representative states technique or the aggregation technique

### 7.2.3 Comparison with the aggregation technique

Even if the representative states technique used with non-model based reinforcement learning algorithms offers less good results for the OMIB control problem than the aggregation technique due notably to some specific problems related to the boundary, one should not be discouraged to use the representative states technique with model based reinforcement learning algorithms.
Indeed, let us consider figure 7.6a which provides a synthetic comparison between model based techniques applied either with the aggregation or the representative states technique. We see that in terms of convergence speed, the performance of the representative states technique used with a model based reinforcement learning algorithm is really impressive. We believe that the results could be even better if a *Learning* module using the Kalman Filter like algorithm (figure 6.12) was used instead of the Stochastic Approximation algorithm (figure 6.13) thanks to the stronger convergence properties the Kalman Filter like algorithm has.

## 7.3 On the combined use of multiple MDP$^\delta$

Before concluding this chapter let us report on some preliminary investigations concerning the use of multiple MDP$^\delta$, already introduced in section 4.5. Recall that this idea consists in using in parallel a certain number of MDP$^\delta$ approximation architectures obtained by randomization of the origin of the discretization grid.

In the context of reinforcement learning, the use of this technique consists in running in parallel several learning agents (one for each version of the approximation architecture) which are all fed at each time-step with the four-tuple gathered from interaction with the system. In order to decide at each time-step how to control the system, a higher level agent would then apply the majority vote among the control decisions suggested by the lower level agents, together with an $\epsilon$-greedy strategy to ensure the desired degree of exploration.

Figure 7.7 shows the learning curves obtained by using this approach (with a team of 10 agents working in parallel) together with the aggregation technique (the Kalman Filter like algorithm is used to reconstruct the MDP$^\delta$ structure) and in the context of different grid sizes. Figures 4.7(a), (b), and (c) provide these curves respectively for the grid sizes of $25 \times 25$, $50 \times 50$, and $100 \times 100$. For the sake of comparison, we have also drawn on these figures the curves obtained with a single control agent, i.e. in the classical way (curves previously drawn on figure 7.2a). We observe that with the two coarser discretizations, the effect is to speed up the learning quite significantly. Figure 4.7d shows on a same diagram the three "10 MDP$^\delta$" curves, highlighting the fact that for the smaller grid sizes the "multi-agent" approach is able to provide a good compromise between learning speed and asymptotic behavior.

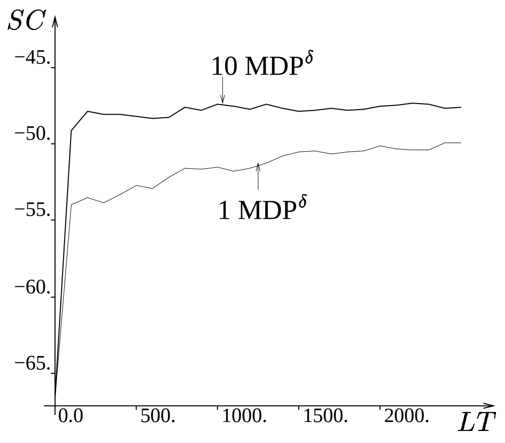
We believe that these improvements are mainly due to the variance reduction effect of the "randomizing and averaging combination" which is well known in the literature on automatic learning [Geu02]. In addition, let us recall that also in asymptotic regime the policy is improved by this technique in the regions close to the stable equilibrium point (see section 4.5). These observations suggest that the technique reduces the variance and at the same time the bias of the approximation architecture. Notice that such an improvement would be specially interesting in the context of higher dimensional applications where the variance will further increase due to the curse of dimensionality effect.

Although the investigations presented here are preliminary, they suggest that the combination of model-based reinforcement learning methods and ensemble methods used in automatic learning are a promising direction for further research.

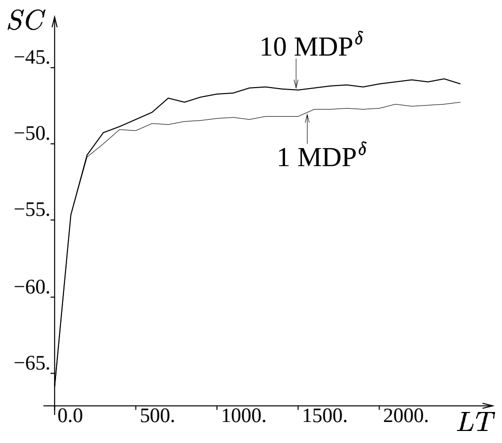
## 7.4 Summary

*In this chapter we have studied empirically the behavior of reinforcement learning algorithms for discrete-time control problems with continuous state spaces.*

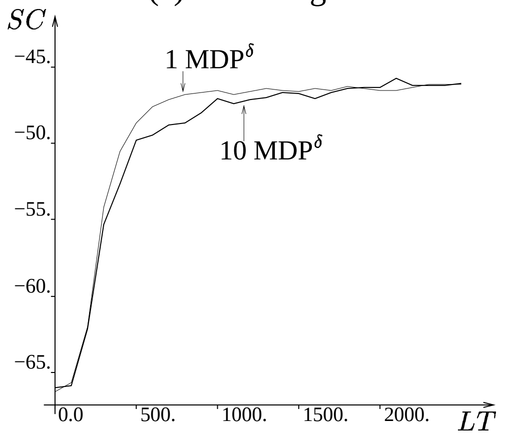
*This study has been carried out in the context of a simple, yet representative, power*



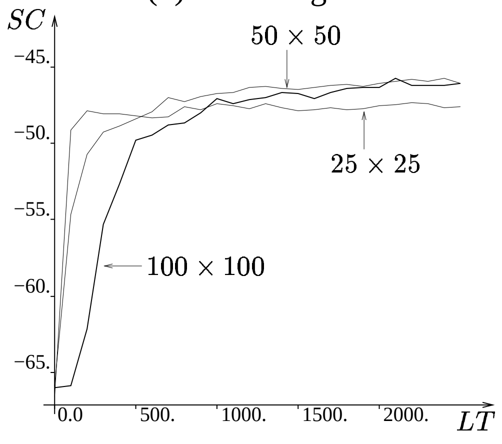
(a) $25 \times 25$ grid



(b) $50 \times 50$ grid



(c) $100 \times 100$ grid



(d) 10 MDP$^\delta$

Figure 7.7: Score obtained as a function of learning time $LT$

*system control problem. We believe that the use of this problem to study and develop reinforcement learning algorithms for power system control problems has many advantages over both purely academic problems and more large scale real power system problems. With respect to the former it allows us to exploit our physical understanding of this problem in order to interpret more effectively the obtained results. With respect to the latter, the small size of the system allows one to run more extensive simulations (in particular to evaluate scores of the resulting policies) and try out a larger number of learning algorithms variants. Also, the fact that the state space of our problem is two-dimensional allows one to show graphically the resulting policies, which is useful for quick appraisal of the results.*
*Among the findings of this chapter the main ones are as follows.*

- *Reinforcement learning algorithms allow indeed to determine by pure interaction a near optimal control policy for a sensible power system control problem.*

- *Model based techniques offer significantly better and faster convergence properties than non-model based ones.*

- *It is impossible to state firmly whether the aggregation technique is better than the representative states technique. The aggregation technique gave better results with non-model based algorithms while the representative states technique was performing better with model based algorithms.*

- *Convergence speed is strongly influenced by the way the system is controlled and by the complexity of the solution learned.*

*We can complement these findings with the robustness results already found in chapter 4, which still hold valid here since the type of control policy determined is essentially the same.*

# Chapter 8

# Dynamic programming in continuous-time

*This chapter is dedicated to the study of optimal control problems in the continuous-time case. Throughout this chapter we will stress that many of the ideas explained in previous chapters can be extended here although there exist some difficulties specific to the continuous-time case. We first define the type of continuous-time optimal control problem we use. Then we introduce the Hamilton-Jacobi-Bellman (HJB) equation used to characterize, at least partially, the value function. This equation corresponds to the Dynamic Programming (DP) equation introduced in the discrete-time case (chapter 2), but contrary to this DP equation many other functions than the value function can satisfy the HJB equation. We propose a way of solving this equation by discretizing the HJB equation space. The aim of the discretization is to define from the knowledge of the system dynamics and the rewards a finite Markov Decision Process and to solve it in order to extend its characteristics in terms of value function, Q-function and optimal control policy to the original system. To some extent it is a procedure similar to the one used in chapter 3 to solve discrete-time optimal control having an infinite number of states. We mention the convergence properties of this MDP and explain how its characteristics can be learned from interaction with the system. This last aspect is in some sense similar to chapter 6 content. Academic examples will be used to illustrate the different concepts and the FACTS control problem introduced in chapter 4 will be treated. In this chapter we limit ourselves to the deterministic case. The stochastic case can be found in [Mun97] where references to further work are given. Moreover, the content of this chapter is largely inspired from reference [Mun00] that the reader could consult for more information.*

## 8.1 Hamilton-Jacobi-Bellman equation

There exist two possible approaches for continuous-time optimal control problems. One is known as Pontryagin's *maximum principle* [PBGM62] and is not considered here. The other approach known as the *dynamic programming* approach [Bel57] is the one adopted. In this section we present the basic elements of the continuous-time optimal control problem considered, such as type of system dynamics, rewards, policy and reinforcement functional used. We then introduce the Hamilton-Jacobi-Bellman equation as a way of solving this continuous-time optimal control problem and illustrate these concepts on an academic example.

### The system dynamics

Let $x(t)$ be the *state of the system* at time $t$, which belongs to a closed and bounded subset $X \in \mathbb{R}^n$. We denote by $\partial X$ the boundary of this subset. The temporal evolution of the system depends on the current state and the *control action* $u(t) \in C$, where $C$, a closed subset of $\mathbb{R}$, is the *control space*; it is defined by the controlled differential equation :

$$\frac{dx(t)}{dt} = f(x(t), u(t)). \tag{8.1}$$

The function $f$ is considered to be Lipschitzian with respect to its first argument : there exists a (positive) constant $L_f$ such that :

$$\forall x, y \in X, \forall u \in C \quad \|f(x,u) - f(y,u)\| \leq L_f \|x - y\|. \tag{8.2}$$

For any initial state $x_0$ at time $t = 0$, the choice of a control function $u(t)$ leads, because the system dynamics (8.1) is deterministic, to a unique *trajectory* $x(t)$ (figure 8.1). For any system trajectory we denote by $T \geq 0$ the time at which the trajectory exits from the set $X$ i.e., such that $x(T) \in \partial X$ and $x(T) + \epsilon f(x(T), u(T)) \notin X$ for a sufficiently small $\epsilon$.

### Rewards

We define a *current reinforcement* $r(x,u) : X \times U \to \mathbb{R}$ and a *boundary reinforcement* $g(x) : \partial X \to \mathbb{R}$. The parameter $\gamma \in ]0,1[$ is the *discount factor* that weights short-term rewards more than long-terms rewards. Unlike the discrete case, in the continuous case we need to consider two different reinforcement functions : $r$ is obtained and accumulated as long as the trajectory remains in $X$, whereas $g$ occurs whenever the trajectory reaches the boundary of the state space and then exits (if it does). We suppose that these reinforcement functions are Lipschitzian according to the state variable in their corresponding domain of definition.

### Control policies

We define a policy $\pi$ such that $\pi : X \times \{t\} \to C$ and a stationary policy $\mu$ such that $\mu : X \to C$ .

### The reinforcement functional

We define the *reinforcement functional J* which depends of the initial state $x$ and the policy $\pi$ :

$$J^{\pi}(x) = \int_0^T \gamma^t r(x(t), \pi(x(t), t))dt + \gamma^T g(x(T))$$

where $T$ denotes the exit time of the trajectory, which is dependent on the initial state $x$ and the control policy $\pi$.
The *objective of the control problem* is to find the policy $\pi$ that maximizes the reinforcement functional for any initial state $x$.
We define the *value function* (denoted by $V$ or $J^*$), the maximum value of the reinforcement functional as a function of the initial state at time $t = 0$ :

$$V(x) \quad = \quad \sup_{\pi} J^{\pi}(x).$$

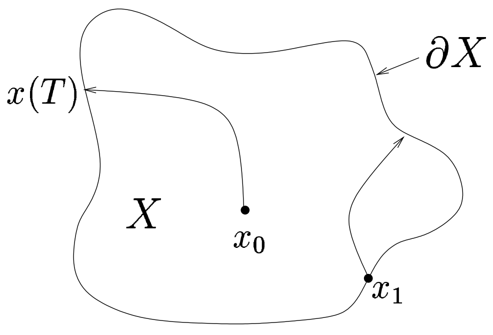


Figure 8.1: Trajectories in the state space

**The Hamilton-Jacobi-Bellman equation**

By using the dynamic programming principle one can prove that if the value function is *differentiable with respect to $x$*, it necessarily satisfies the Hamilton-Jacobi-Bellman (HJB) equation :

$$V(x)\ln\gamma + \max_{u\in C}[\frac{\partial V(x)}{\partial x}.f(x,u) + r(x,u)] = 0 \tag{8.3}$$

where $\frac{\partial V(x)}{\partial x}$ denotes the gradient of $V$ at $x$. Additionally it can be shown that $V$ must satisfy the following boundary condition :

$$V(x) \geq g(x) \quad \text{for} \quad x \in \partial X.$$

The boundary condition is an inequality (and not an equality) because at some boundary points (for example at $x_1 \in \partial X$ on figure 8.1) there may exist a control $u(t)$ such that the corresponding trajectory stays inside $X$ and whose reinforcement functional is strictly superior to the immediate boundary reinforcement $g(x_1)$.
An important result that can be drawn from the HJB equation is the stationarity of the optimal policy. Indeed, it can be shown from (8.3) that the optimal policy is given by :

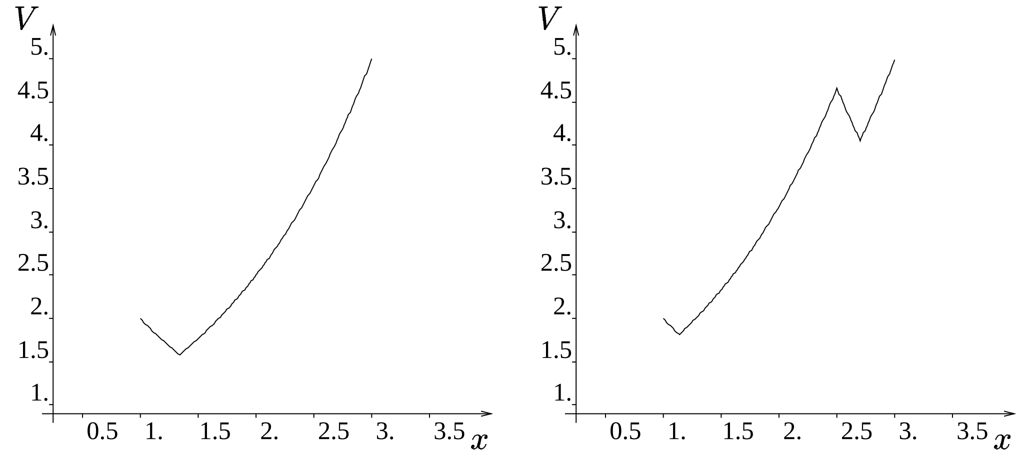
$$\mu^*(x) \in \arg\max_{u\in C}[\frac{\partial V(x)}{\partial x}.f(x,u) + r(x,u)] \tag{8.4}$$

which defines indeed a stationary policy.
Dynamic programming computes the value function in order to find the optimal control solution with a stationary policy.

**Properties of the value function**

One can show that if at any point of the boundary of the state space there exists a value of the control such that the corresponding trajectory is *not tangential* to the boundary then the value function is *continuous* (see [BP90] for the proof). In general however the value function is *not differentiable everywhere*. This means that we cannot expect to find solutions differentiable everywhere for the HJB equation. Now if we look for solutions differentiable almost everywhere for this equation, we find that many solutions other than the value function solve the HJB equation. Next example illustrates these concepts.

**Example 8.1.1** Consider the system dynamics $\frac{dx}{dt} = u$ with $X = [1,3]$, $\partial X = \{1,3\}$ and $C = \{-1,1\}$. The value of the current reinforcement $r$ is 0 everywhere on $X \times C$ and



(a) Value function (b) A solution of the HJB equation

Figure 8.2: Value function and solutions of the HJB equation

the boundary reinforcement is such that $g(1) = 2$, $g(3) = 5$. The decay factor $\gamma$ is chosen equal to $0.5$. An optimal policy thus corresponds to a policy which reaches one of the two exit points in minimum time. For each start state $x$ there are obviously only two candidate control strategies: either $u = 1, \forall t \geq 0$ or $u = -1, \forall t \geq 0$. Any other control policy would consist of wasting time (and reward). We thus can deduce that the value function is defined by :

$$\begin{aligned} V(x) &= \max\{g(1) * \gamma^{x-1}, g(3) * \gamma^{(3-x)}\} \\ \Rightarrow V(x) &= \begin{cases} 2 * 0.5^{x-1} & \text{if} \quad [1, 1.33904] \\ 5 * 0.5^{3-x} & \text{if} \quad ]1.33904, 3]. \end{cases} \end{aligned}$$

The value function is represented on figure 8.2a. At $x = 1.33904$ the function is not differentiable anymore. The optimal control variable value can be determined using equation (8.4) where the value function is differentiable. We find that the optimal control is equal to 1 when the derivative of $V$ is positive and equal to $-1$ when this derivative is negative. Thus the optimal control is $-1$ on $[1, 1.33904[$, 1 on $]1.33904, 3]$. It cannot be determined uniquely at $x = 1.33904$ by using equation (8.4) due to the non differentiability of the value function at this point. Actually, at this point the two possible values of $u$ are equivalent in terms of reward.
The HJB equation is given by :

$$V(x) \ln \gamma + \max\{\frac{\partial V(x)}{\partial x}, -\frac{\partial V(x)}{\partial x}\} \tag{8.5}$$

with the following boundary conditions : $V(1) \geq g(1)$ and $V(3) \geq g(3)$. The value function is not the unique function to satisfy this HJB equation almost everywhere with the corresponding boundary conditions. Indeed, figure 8.2b represents another continuous solution that satisfies the HJB equation almost everywhere with the boundary conditions $V(1) = g(1)$ and $V(3) = g(3)$. If you introduce the value of this function in equation (8.4), you will obtain a stationary policy that is different from the optimal stationary policy. Indeed, the control variable value computed is equal to $-1$ if $\frac{\partial V(x)}{\partial x} < 0$ and $+1$ if $\frac{\partial V(x)}{\partial x} > 0$, which obviously is a suboptimal control (at least in the part of the state space where it is different from the real optimal control function).

## 8.2 Convergent numerical approximation scheme

The main idea to approximate the HJB equation using a convergent numerical scheme is to discretize it into a Dynamic Programming (DP) equation for some Markov Decision Process, to solve the DP equation obtained and to extend the results computed to the initial control problem. It is roughly equivalent to the procedure adopted in chapter 3 to solve discrete-time optimal control problems with infinite state spaces since we define by discretizing the HJB equation a MDP that catches the initial control problem main characteristics.

In this section we explain how to discretize the HJB equation and how to solve the MDP obtained, and discuss the characteristics of the solution obtained.

### Discretization of the HJB equation

There exist different ways to discretize the HJB equation into a DP equation for some MDP. The method we present here is known as *finite differences* method [Kus90, KD92].

$X^\delta$, $C^\delta$, $x^\delta$ are used to represent respectively the state space of the MDP, the finite control space of the MDP with $C^\delta \subset C$ and a state of the MDP.

$\delta$ is used as suffix to specify whether we refer to an element of the initial control problem or an element of the MDP and to denote the discretization step.

As it was the case in chapter 3 with the representative states technique, there is a correspondence between each element of $X^\delta$ and an element of $\mathbb{R}^n$. This is illustrated on figure 8.3a where the white and black bullets can be associated to elements of $X^\delta$. Therefore we will also use $x^\delta$ to denote the state of $\mathbb{R}^n$ it corresponds to.

To discretize the HJB we proceed as follows. Let $e_1, e_2, \cdots, e_n$, be a basis of $\mathbb{R}^n$. The system dynamics is given by $f = (f_1, \cdots, f_n)$.

Let the positive and negative parts of $f_i$ be : $f_i^+ = \max(f_i, 0)$, $f_i^- = \max(-f_i, 0)$.

For any discretization step $\delta$, let us consider the regular grid[1] :

$$\delta Z^n = \{\delta \sum_{i=1}^{n} j_i e_i | j_i \in \mathbb{Z}, \forall i\}.$$

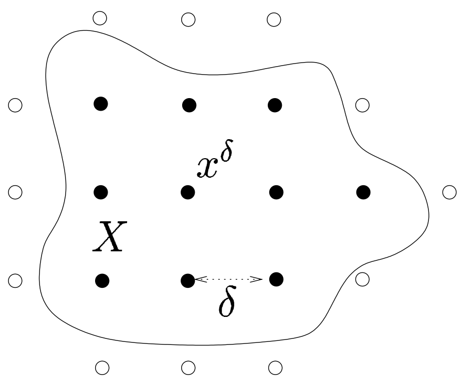
Let $\partial X^\delta$ denote the set of points defined by

$$\partial X^\delta = \{x^\delta \in \delta Z^n \setminus X \text{ such that at least one adjacent point } x^\delta \pm \delta e_i \in X\}$$

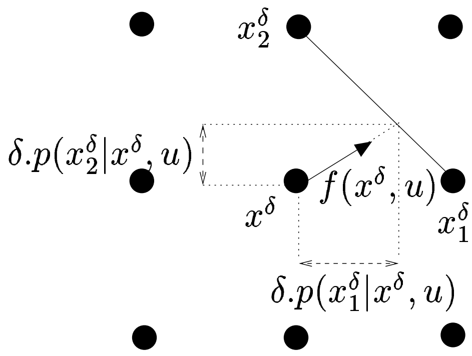
and let $X^\delta$ be defined by

$$X^\delta = (\delta Z^n \cap X) \cup \partial X^\delta.$$

Elements of $X^\delta \setminus \partial X^\delta$ can be interpreted as being black bullets of figure (8.3a) while elements of $\partial X^\delta$ as white bullets. $\partial X^\delta$ is called the *frontier* of $X^\delta$.



(a) Discretization of the state space



$x_2^\delta$
$\delta.p(x_2^\delta|x^\delta, u)$
$x^\delta$
$f(x^\delta, u)$
$x_1^\delta$
$\delta.p(x_1^\delta|x^\delta, u)$

(b) A geometrical interpretation

Figure 8.3: Discretization of the state space

The finite differences method consists in replacing the gradient $\frac{\partial V(x^\delta)}{\partial x}$ by the forward and backward difference quotients of $V$ at $x^\delta \in X^\delta \setminus \partial X^\delta$ in direction $e_i$ :

$$\begin{aligned}\Delta_i^+ V^\delta(x^\delta) &= \frac{1}{\delta}[V^\delta(x^\delta + \delta e_i) - V^\delta(x^\delta)]\\ \Delta_i^- V^\delta(x^\delta) &= \frac{1}{\delta}[V^\delta(x^\delta - \delta e_i) - V^\delta(x^\delta)].\end{aligned}$$

[1]The way the grid is defined is not well suited for some problems because it imposes the grid to contain the origin of the system and requires a same discretization step $\delta$ for each direction of the state space. These constraints can however be circumvented by first realizing a coordinates change on the system.

Thus the HJB equation can be approximated by the following equation :

$$V^\delta(x^\delta)\ln\gamma + \max_{u\in C^\delta}[\sum_{i=1}^{n}(\Delta_i^+ V^\delta(x^\delta).f_i^+(x^\delta,u) + \Delta_i^- V^\delta(x^\delta).f_i^-(x^\delta,u)) + r(x^\delta,u)] = 0.$$

Knowing that $(\Delta t \ln\gamma)$ is an approximation of $(\gamma^{\Delta t} - 1)$ as $\Delta t$ tends to 0, we deduce the following approximation equation[2] : for $x^\delta \in X^\delta \setminus \partial X^\delta$

$$V^\delta(x^\delta) = \max_{u\in C^\delta}[\frac{\delta}{||f(x^\delta,u)||_1} r(x^\delta,u) + \gamma^{\frac{\delta}{||f(x^\delta,u)||_1}} \sum_{x^{\delta'}\in X^\delta} p(x^{\delta'}|x^\delta,u)V^\delta(x^{\delta'})] \quad (8.6)$$

$$\text{with} \quad p(x^{\delta'}|x^\delta,u) = \begin{cases} \frac{f_i^\pm(x^\delta,u)}{||f(x^\delta,u)||_1} & \text{for } x^{\delta'} = x^\delta \pm \delta e_i \\ 0 & \text{otherwise} \end{cases} \quad (8.7)$$

which is a DP equation for a finite Markov Decision Process whose *state space* is $X^\delta$, control-space $C^\delta$ and probabilities transitions are $p(x^{\delta'}|x^\delta,u)$. A geometrical interpretation of these transition probabilities is carried out on figure 8.3b.
For the boundary condition, we define the terminal states' value function (assuming that $g(\cdot)$ is defined on $\partial X^\delta$) :

$$V^\delta(x^\delta) = g(x^\delta) \quad \text{for} \quad x^\delta \in \partial X^\delta. \quad (8.8)$$

The optimal stationary policy can be computed by introducing the $Q^\delta$-function :

$$Q^\delta(x^\delta,u) \quad = \quad \frac{\delta}{||f(x^\delta,u)||_1} r(x^\delta,u) + \gamma^{\frac{\delta}{||f(x^\delta,u)||_1}} \sum_{x^{\delta'}\in X^\delta} p(x^{\delta'}|x^\delta,u)\max_{u\in C^\delta} Q^\delta(x^{\delta'},u).$$

Therefore we have

$$\mu^{\delta*}(x^\delta) = \arg\max_{u\in C^\delta} Q^\delta(x^\delta,u). \quad (8.9)$$

The solution of equation (8.6) is discussed in the next paragraph.

[2]We denote by $||y||_1 = \sum_{i=1}^n |y_i|$ the 1-norm of any vector $y \in \mathbb{R}^n$.

## Solution of the MDP

We first define the $T^{FD}$ mapping by :

$$T^{FD}J(x^\delta) = \max_{u \in C^\delta}\left[\frac{\delta}{||f(x^\delta,u)||_1} r(x^\delta,u) + \gamma^{\frac{\delta}{||f(x^\delta,u)||_1}} \sum_{x^{\delta'} \in X^\delta} p(x^{\delta'}|x^\delta,u)J(x^{\delta'})\right]$$

and note that $V^\delta$ is a fixed point of the $T^{FD}$ mapping. Since $||f||_1$ is bounded from above[3] (by some value $M_f$), $T^{FD}$ is a *contraction mapping* (see appendix C). Indeed, it can be shown that :

$$||T^{FD}J_1 - T^{FD}J_2||_\infty \leq \gamma^{\frac{\delta}{M_f}} ||J_1 - J_2||_\infty. \tag{8.10}$$

Equation (8.10) guarantees that the $T^{FD}$ mapping has a unique fixed point. DP iterative methods (value iteration algorithm, Gauss-Seidel version of the value iteration algorithm, $\cdots$ ) can be used to compute this fixed point ($V^\delta$).

## Convergence to the real solution

Concerning the convergence of the finite differences approximation scheme, it can be shown that under certain hypotheses (notably concerning the way $C^\delta$ converges to $C$ as $\delta \to 0$) the solution $V^\delta$ satisfies (see [Mun00]) :

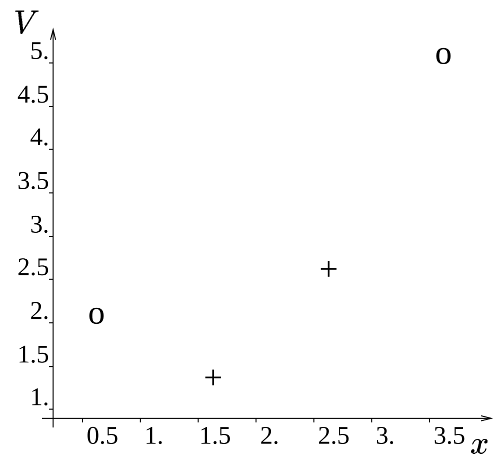
$$\lim_{\substack{\delta \downarrow 0 \\ x^\delta \to x}} V^\delta(x^\delta) = V(x) \text{ uniformly on any compact } \Omega \subset X \setminus \partial X. \tag{8.11}$$

**Example 8.2.1** We illustrate the use of the finite differences numerical scheme on the control problem described in example 8.1.1.
The set $X^\delta$ is composed of four states denoted by $x_1^\delta$, $x_2^\delta$, $x_3^\delta$ and $x_4^\delta$. They correspond to a value of $x$ respectively equal to $0.5$, $1.5$, $2.5$ and $3.5$. The discretization step $\delta$ is thus equal to $1$. $x_1^\delta$ and $x_2^\delta$ belong to $\partial X^\delta$. They are terminal states with a value function respectively equal to $2$ and $5$ ($V^\delta(x_1^\delta) = 2$ and $V^\delta(x_4^\delta) = 5$). The discretized control-space $C^\delta$ is chosen equal to $C$.
We can see that the term $\gamma^{\frac{\delta}{||f(x^\delta,u)||_1}}$ which intervenes in equation (8.6) is equal to $\gamma \quad , \forall x^\delta \in X^\delta$ and $\forall u \in C^\delta$.
If we use expression (8.7) to compute the transition probabilities we obtain the following values : $p(x_1^\delta|x_2^\delta,-1) = 1$, $p(x_3^\delta|x_2^\delta,1) = 1$, $p(x_2^\delta|x_3^\delta,-1) = 1$, $p(x_4^\delta|x_3^\delta,1) = 1$, the other terms being equal to $0$. The discretized HJB equation (8.6) becomes :

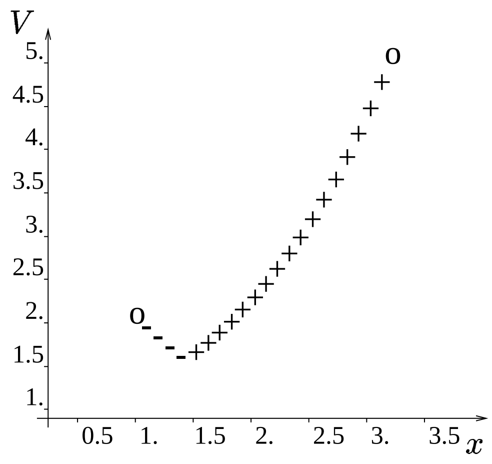
$$\begin{array}{rclcl} V^\delta(x_2^\delta) & = & \max(\gamma V^\delta(x_1^\delta), \gamma V^\delta(x_3^\delta)) & = & \max(1., 0.5V^\delta(x_3^\delta)) \\ V^\delta(x_3^\delta) & = & \max(\gamma V^\delta(x_2^\delta), \gamma V^\delta(x_4^\delta)) & = & \max(0.5V^\delta(x_2^\delta), 2.5). \end{array} \tag{8.12}$$

[3] $||f||_1$ is bounded from above because $f$ is Lipschitzian (equation 8.2) and $X$ is bounded.

By solving these two expressions we obtain $V^\delta(x_2^\delta) = 1.25$ and $V^\delta(x_3^\delta) = 2.5$. The optimal control variable value to associate to $x_2^\delta$ and $x_3^\delta$ can be found by computing the $Q^\delta$-function and by using equation (8.9). We have $\mu^*(x_2^\delta) = 1$ and $\mu^*(x_3^\delta) = 1$.
The results are represented on figure 8.4a while figure 8.4b represents the results obtained by choosing $X^\delta = \{0.9, 1, 1.1, \cdots, 2.9, 3, 3.1\}$ ($\#X^\delta = 23$ and $\delta = 0.1$). The $o$ symbols indicate the $V^\delta$ value of the states of $\partial X^\delta$, the $+$ symbols the $V^\delta$ value of the states of $X^\delta \setminus \partial X^\delta$ where the optimal control variable value computed is $+1$ and the $-$ symbols the $V^\delta$ value of the states of $X^\delta \setminus \partial X^\delta$ where the optimal control variable value computed is $-1$.
We can observe the close similarity there exists between this approximate value function and the exact one sketched on figure 8.2a.



(a) $\#X^\delta = 4$



(b) $\#X^\delta = 23$

Figure 8.4: Discretization of HJB

## 8.3 Learning of the control problem solution

In this section we will not discuss in detail about how to solve the continuous-time optimal control problem without having any knowledge about the system dynamics and the rewards i.e., just from interaction with the system. We refer the reader to [Mun97] and [Mun00] for a quite complete survey of this problem. Roughly speaking, there still exist two families of learning algorithms : model based algorithms that learn the structure of the MDP and compute from its knowledge the $Q^\delta$-function and non-model based algorithms that learn immediately the $Q^\delta$-function. All these algorithms require a sampling rate that has to be correlated with the size

of the discretization step $\delta$ and the system dynamics. It implies that a continuous monitoring of the system trajectory is necessary which makes difficult the practical feasibility of such learning algorithms.
From a theoretical point of view, it can be shown that the solution that would be learned for a discretization step $\delta$ does not necessarily converge to $V^\delta$ as defined by equation (8.6). However one can prove that if $\delta \to 0$ then the solution learned converges to the value function of the system. But such a learning scheme would require an infinite learning time and an infinite sampling rate.

## 8.4 The simple FACTS control problem

We treat hereafter the FACTS control problem introduced in chapter 4. The FACTS control problem is first stated as a continuous-time optimal control problem, which is then solved by using the finite differences numerical scheme described in section 8.2. We conclude by comparing the resulting control policy with the one obtained in chapter 4 using a discrete-time formulation.

### 8.4.1 Rewards, decay factor and reinforcement functional

We remind first the OMIB system dynamics from section 4.1.1 :

$$\begin{aligned} \dot{\delta} &= \omega && (8.13)\\ \dot{\omega} &= \frac{P_m - P_e}{M} && (8.14)\\ \text{with} \quad P_e &= \frac{EV}{X_{\text{system}} + u} \sin\delta && (8.15) \end{aligned}$$

where $u$ is the FACTS reactance. In the discrete-time case the reward used was $r_t = -|P_{e\,t+1} - P_m|$ when the system was standing in the stability domain , $-100$ if the system went outside the stability domain in which case a terminal state was reached. We were seeking for a stationary policy that was able to maximize the return $\sum_{t=0}^{\infty} \gamma^t r_t$ where $\gamma$ was taken equal to 0.98. The time between two successive steps $t$ and $t+1$ was chosen equal to $0.050\,s$.
The continuous case can be seen in some sense as the limit of the discrete-time case when the time between two successive steps $t$ and $t+1$ tends to 0. In this way of thinking we will look after a stationary policy that is able to maximize the functional :

$$J(x(0)) = -\int_0^T \beta^t |P_e(t) - P_m| dt + \beta^T g(x(T)) \tag{8.16}$$

where $T$ indicates the time for which the system goes outside the stability domain, if it does (the state space $X$ is defined as being the set of states that satisfy

$$\frac{1}{2}M\omega^2 - P_m\delta - \frac{EV}{X}\cos(\delta) \leq -0.438788$$

(see chapter 4)). The current reward equals $-|P_e(t) - P_m|$ and the boundary reward $g(x(T))$ is taken equal to $-100$.
Concerning the decay factor, that we have named $\beta$ in order to avoid any confusion with the discrete-time case, we take its value equal to $\gamma^{\text{number of steps / second}}$. We thus have : $\beta = 0.98^{20} \simeq 0.6676$, since the time step of $0.050\,s$ was used in the simulations of chapter 4. Such a strategy has been chosen in order to be able to compare results between the discrete-time case and the continuous-time case.

### 8.4.2 The HJB equation for the simple FACTS control problem

If we introduce the OMIB system dynamics and the current reward into the HJB equation we obtain :

$$V\ln\gamma + \max_{u\in C}[\frac{\partial V}{\partial\delta}\omega + \frac{\partial V}{\partial\omega}\frac{P_m - \frac{EV}{X_{\text{system}}+u}\sin\delta}{M} - |\frac{EV}{X_{\text{system}}+u}\sin\delta - P_m|]$$

with the boundary conditions :

$$V(x) \geq -100 \quad \text{for} \quad x \in \partial X. \tag{8.17}$$

We can use equation (8.4) in order to get some information about the optimal stationary policy of the system. We obtain :

$$\begin{aligned}\mu(x) &= \arg\max_{u\in C}[\frac{\partial V}{\partial\delta}\omega + \frac{\partial V}{\partial\omega}\frac{P_m - \frac{EV}{X_{\text{system}}+u}\sin\delta}{M} - |\frac{EV}{X_{\text{system}}+u}\sin\delta - P_m|] \\ &= \arg\max_{u\in C}[\frac{\partial V}{\partial\omega}\frac{\frac{-EV}{X_{\text{system}}+u}}{M}\sin\delta - |\frac{EV}{X_{\text{system}}+u}\sin\delta - P_m|].\end{aligned} \tag{8.18}$$

From equation (8.18) we can see that if $\sin\delta = 0$, that is if $\delta = 0$ ($\delta = k\pi$ with $k \neq 0$ does not belong to $X$) the optimal control variable value cannot be determined even if the value function $V$ is known. We can therefore expect a change in the optimal control variable value around the axis $\delta = 0$ (as it was the case with example 8.1.1). $V$ being unknown, it is difficult to extract other information about the optimal stationary policy by using equation (8.18).
In the next paragraph we discretize the HJB equation in order to compute an approximation of the optimal stationary policy.

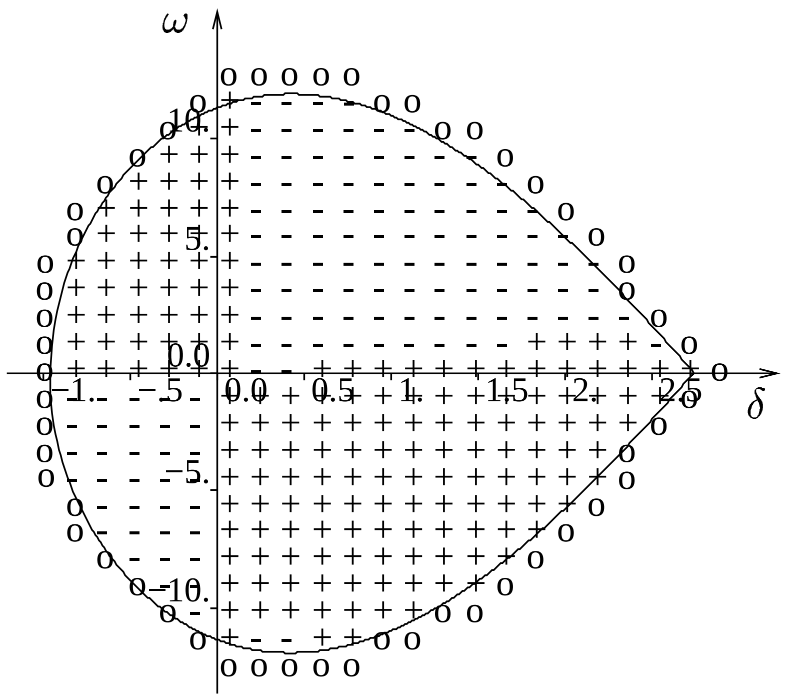


Figure 8.5: Approximation of the optimal stationary policy computed by using a finite differences method to discretize HJB equation

### 8.4.3 Discretization of the HJB equation for the OMIB case

If we discretize the HJB equation by using the finite differences method described in section 8.2 and compute for each $x^\delta$ the approximate optimal control variable value to be associated with ($C^\delta$ being chosen equal to $\{-0.04, 0\}$), we obtain the policy represented on figure 8.5. The $+$ symbols represent states of $X^\delta$ for which the value of $\underset{u \in C^\delta}{\arg\max} Q^\delta(x^\delta, u)$ is equal to 0, the $-$ symbols represent states of $X^\delta$ for which the value of $\underset{u \in C^\delta}{\arg\max} Q^\delta(x^\delta, u)$ is equal to $-0.04$ and the symbols *o* represent states of $X^\delta$ that belong to $\partial X^\delta$. If we compare this figure with a representation of the approximate optimal policy computed in the discrete time case (see for example figure 4.9a), we observe some strong similarities in the control laws obtained. This is not surprising since the continuous-time optimal control problem solved can be seen as the limit when $t \rightarrow 0$ of the discrete-time one. Trajectories of the system being clockwise, one can see that the switching occurs later with the continuous-time control law than with the discrete-time one.

## 8.5 Summary

*In this chapter we have described ways to solve some optimal control problems in the continuous-time case. The resolution procedure was similar to the one used to solve discrete-time optimal control problems with infinite state spaces since a finite Markov Decision Process is used to represent the original control problem solved and that its solution is extended to the original control problem. We mentioned that the computation of the finite MDP solution from interaction with the system rather than by using the knowledge of the system dynamics and the rewards was difficult to achieve technically due to the necessity of monitoring the system trajectory continuously.*

*Some results concerning the convergence to the exact solution when the discretization step tends to zero have been presented. They do not have their equivalent in the discrete-time case.*

# Chapter 9

# Real world conditions

*This chapter has been entitled "Real world conditions" in order to stress that we will describe here several problems met when reinforcement learning algorithms are applied to control real world systems with some constraints on the return obtained during the learning, on the measurements realized on the system, on the learning time and on the control hardware capabilities.*
*We also investigate potential problems linked to the non-stationarity of the environment in which the reinforcement learning methods would have to act in practice since the operating point of real power systems changes continuously as transactions appear and disappear.*
*We start by discussing these problems in the first section of this chapter and in the aftermath some of the strategies induced from these discussions will be illustrated and assessed on time-domain simulations of a four-machine power system.*

## 9.1 Problems met when controlling a real world system

In the following subsections we describe and discuss different types of problems met when using reinforcement learning algorithms to control a real world system and propose strategies that can be adopted to overcome them. Our discussion is of course strongly oriented by the control of electric power systems but part of these considerations would also be applicable to many other real world problems.

### 9.1.1 Policy used when starting to interact with a real system

The *Action Selection* module of the reinforcement learning algorithm uses the estimate of the $Q$-function to select a control action (for example by using an $\epsilon$-Greedy policy). When the reinforcement learning is initialized from scratch, this estimate is possibly far from the true value during the early stages of learning. Its use can thus lead to the selection of largely suboptimal control actions and, hence, “bad” returns may be observed at the beginning of the learning. For some systems, and especially real world systems, these “bad” returns reflecting bad control actions may be unacceptable.

We propose two different strategies to cope with this problem.

### Ad hoc control policy to start with

In order to overcome the problem one may try for example to design by other means than the optimal control approach a policy that ensures a good (even though not optimal) return and to use it at the beginning of the learning and then, as the learning process proceeds, to shift progressively to an $\epsilon$-Greedy policy.

In the context of electric power systems there exist various linear and non-linear approaches which can help to design such a first ad hoc control policy.

### First use RL in the simulation environment

Another strategy could be achieved by making the difference between *off-line* and *on-line* learning.

As opposed to on-line learning which directly interacts with a real system, off-line learning consists in using the reinforcement learning algorithms to control a “copy” of the system for which the return observed during the learning has no importance (for example a numerical representation of the real world system we want to control). Thus, rather than starting from scratch in on-line mode, one may imagine to use first the reinforcement learning algorithms in an off-line learning mode and then, when the optimal stationary policy is sufficiently well approximated, to use them in an on-line learning mode (or even to use the control law computed in the off-line learning mode as such in the on-line mode without further learning).

It may (and it will, generally) happen that the control problem studied in the off-line learning mode is not exactly the same as the one used in the on-line learning mode (e.g. because simulation models never represent a real-system with hundred percent accuracy). If the differences are very important, one may not assure anymore that with such a strategy the return obtained at the beginning of the learning in the on-line mode will be “good”. However, if the simulation model is reasonably accurate then the initial policy so obtained should also be reasonably close to an optimal one for the real system.

### 9.1.2 Non observable rewards

In some of the power system control problems the aim is to use local controls in order to damp or stabilize system wide oscillations. For example, in the context of transient angle instability one would like to use local measurements around a particular power plant in order to detect and prevent impending loss of synchronism of this plant with respect to the remaining machines of the system.
In such a situation, even if local measurements are used to trigger controls, it might still be useful (and possibly preferable) to use global (i.e. system wide) information to define the rewards. For example, in our transient stability example it would make sense to use the rotor angles of all the machines connected to the power system to define the reward signal, even though in real-life these quantities are not available in real-time.
Since, by assumption these global reward quantities cannot be obtained in real-time, such a scheme would prevent one from using the on-line learning mode. Nevertheless, in off-line mode (e.g. using a standard time-domain simulation program) these rewards could certainly be computed at each time step and fed into the reinforcement learning algorithm to adjust its control policy.
Notice that in the problem that we will investigate later on in this chapter we could have applied such a strategy, but we did not do so since we wanted to assess the capability of reinforcement learning in on-line mode for this problem. Notice also that this problem is intimately related to the problem of partially observable states discussed in the next subsection.

### 9.1.3 Partially observable system state

Till now, it has been assumed that the state (in the sense of systems theory) of the system could be observed perfectly. But usually, the observation of the state is incomplete and noisy. The strategy we propose in this subsection to cope with the partial observability of the state can be seen in a sense as one of the most simplistic solutions that exist in the RL literature (see for example [CKL94], [Lit94], [KLC98] and [Ast65]) but will prove to be very efficient in the examples treated in the next section and has the main advantage to be able to fit perfectly into the considerations used in the previous chapters.

#### Use of past observations and controls to define pseudo-states

The main idea of the strategy is to suppose that when the observation of the state is incomplete and noisy, more information can be obtained about the current state of the system by using the history of the observations realized on the system and the actions taken by the reinforcement learning algorithms than by using only the last observation. Then we define a "pseudo-state" from the history of the observations done and the actions taken and *proceed exactly as if it were really the real state of the system.*[1]

To define this pseudo-state we proceed as follows. First, we suppose that $o$ represents the observation done on a system, $O$ the set of all possible values for $o$, $s$ a pseudo-state of the system and $S$ the set of all possible values of[2] $s$. Then the pseudo-state of the system at time $t$ is defined by the following equation :

$$s_t = (o_t, o_{t-1}, \cdots, o_{\max(0,t-Nbo+1)}, u_{t-1}, u_{t-2}, \cdots, u_{\max(0,t-Nbu)}) \quad (9.1)$$

where $Nbo$ and $Nbu$ determine respectively the number of successive observations and the number of successive actions taken by the reinforcement learning algorithms that are used in the definition of the pseudo-state. The larger the values of $Nbo$ and $Nbu$ are, the better the information about the state $x$ contained in $s$ may be. But increasing these numbers also increases the size of $S$ and may therefore penalize the learning speed. One should also note that when $t < \max(Nbo-1, Nbu)$, the number of elements that composes $s_t$ is smaller than when $t \geq \max(Nbo - 1, Nbu)$. This can lead the RL algorithm to choose bad control actions at the beginning of an episode even if when $t \geq \max(Nbo - 1, Nbu)$ the control actions are chosen suitably.

**Determination of the observation window length**

One can think about setting up methods to determine automatically, while interacting with the system, what the best choice would be for $Nbo$ and $Nbu$. This has not been investigated in this thesis and the way we will proceed to determine these two numbers will be by trial and error.

### 9.1.4 Curse of dimensionality

The methodology developed in chapter 6 in order to use reinforcement learning algorithms with infinite state space control problems can become insufficient when the number of variables composing a state (i.e. the number of state variables) is too high. Indeed, the Markov Decision Process aimed to catch the initial control

[1]Our notion of "pseudo-state" is directly related to the notion of "information state" used in the literature on optimal control of partially observable MDPs [Ber00].

[2]Although the pseudo-states are used in the reinforcement learning as if they were the real state of the system, we have chosen to denote them by another letter than $x$ in order to stress that they are just an estimation of it.

problem characteristics can then be composed of too many states to hope to obtain a good approximation of the optimal stationary policy in a reasonable learning time or even to match the computer capabilities[3].
The solution we propose to overcome this difficulty is to define in a pragmatic way from the state $x$ of the system a pseudo-state $s$ that is less “complex” than $x$ and that is able to catch the features of $x$ that intervene mainly in the control problem and then to use this pseudo-state in the reinforcement learning algorithms as if it were the real state of the system.

### 9.1.5 Non-stationarity of the system

Throughout this work we have supposed that the system dynamics and the reward function do not depend explicitly on the time $t$. Unfortunately, for many systems and especially power systems, this assumption does not hold true anymore.
From a theoretical point of view, the systematic way to handle time-varying systems in the dynamic programming framework consists in augmenting the state space by one dimension representing time. However, the appropriateness of this approach, which could be used in the context of off-line learning, strongly depends on the ability to explicitly express the way system dynamics depend on time. In the context of on-line learning, on the other hand, this approach is by itself not sufficient since the time-dimension of the system corresponds to variations from which no experience can be gained, because any particular time instant is experienced only a single time. Thus, additional assumptions are necessary to handle this aspect.
Therefore, the strategy we propose here to deal with the non-stationarity of the system assumes that the system changes “slowly” with time and consists in using the same reinforcement learning algorithms as the ones used with time-invariant systems but for which more weight is given to the last observations done on the system.
In order to give more weight to the more recent measurements, one can for example (as it has already been briefly discussed when explaining the different *Learning* modules) :

- use $0 < \beta < 1$ in the model based *Learning* modules that were using a Kalman Filter like algorithm (figures 5.4, 6.6 and 6.12)
- choose $\alpha_r$ and $\alpha_p$ constants in the model based *Learning* modules that were using a Stochastic Approximation algorithm (figures 5.5, 6.7 and 6.13)

[3]Note that an appropriate choice of $x_1^\delta, \cdots, x_m^\delta$ when using the representative states technique or of $X_1, \cdots, X_m$ when using the aggregation technique can help to reduce the curse of dimensionality effects.

- use $\alpha$ constant in the non-model based *Learning* modules (figures 5.11, 6.8 and 6.14).

Of course, if the reward function and the system dynamics are changing too "rapidly" then such an elementary strategy will be foiled.

## 9.2 A real world power system control problem

This section is dedicated to the use of reinforcement learning algorithms to control a FACTS device installed on a four-machine power system. First we will present the power system used and pose the control problem. Then, rather than solving the control problem by using the reinforcement learning algorithms in a similar way to what has been done with the OMIB power system, we will use them as if they were interacting with a real power system with all the constraints this can imply, that is like if they were acting in an *on-line mode*. These constraints will keep away the computed solution from the optimal one but as the simulation results will show it, it will still be able to reach "acceptable" performances.
The results presented in this section are partially taken from [EW02]. A similar four-machine power system is used in [GEW02] to illustrate the use of RL algorithms (in an off-line mode) to compute the control law of a dynamic brake.

### 9.2.1 The four-machine power system

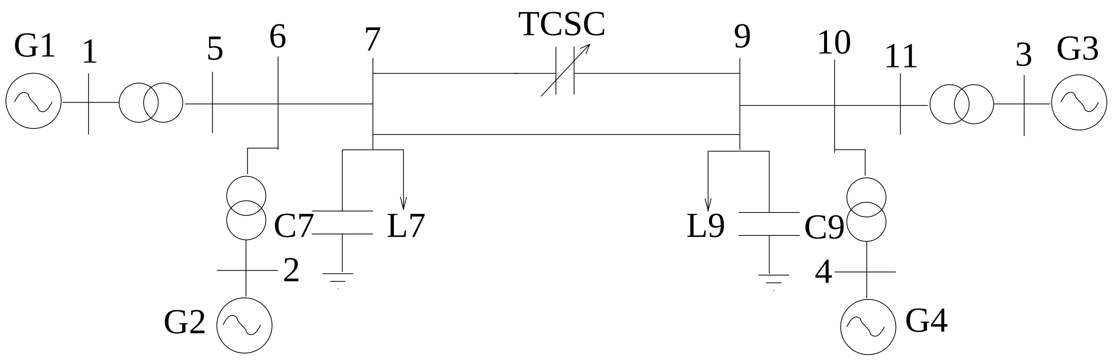


Figure 9.1: A four-machine power system

The four-machine power system used here is represented on figure 9.1 and its characteristics are largely inspired from [Kun94]. When this power system operates in steady-state conditions, the machines, identified by the symbols $G1$, $G2$, $G3$ and $G4$, produce approximately the same power : $700$ MW and the two loads $L7$ and

$L9$ consume respectively 990 and 1790 MW. This repartition of the active power production and consumption causes a power transfer of about 410 MW in the corridor connecting bus 7 to bus 9. The two lines that compose the corridor are such that the reactance of the one on which the FACTS is installed is twice as large as the other.
The loads are modeled according to the exponential model :

$$P = P_0(\frac{V}{V_0})^a \qquad Q = Q_0(\frac{V}{V_0})^b \tag{9.2}$$

where the subscript $_0$ identifies the values of the respective variables at the initial operating conditions (for example $P_0$ of the load $L7$ is equal to 990 MW).
Each machine of this power system is modeled in the same way with 15 state variables : 6 that correspond to the electrical model, 2 to the mechanical variables (the rotor angle and speed), 3 to the AVR and 4 to the turbine (including the governor).
The type of FACTS used is a TCSC which can be considered as a variable reactance placed in series with a transmission line. The reactance of the TCSC, denoted by $X_{\text{FACTS}}$, responds to the first order differential equation :

$$\frac{dX_{\text{FACTS}}}{dt} = \frac{X_{\text{ref}} - X_{\text{FACTS}}}{T_{\text{FACTS}}} \tag{9.3}$$

where $X_{\text{ref}}$ represents the FACTS reactance reference and where $T_{\text{FACTS}}$ has been chosen, in accordance with the technical specifications of such a FACTS device equal to $60\,ms$ [HG00, Gha00].
The control variable for this system is $X_{\text{ref}}$ and it supposed to belong to the interval $[-61.57, 0]\,(\Omega)$. A value of $-61.57\,\Omega$ for $X_{\text{FACTS}}$ corresponds approximately to a $30\,\%$ compensation of the line on which the FACTS is installed.
We will limit the control problem to the stability domain of this four-machine power system. The state space $X$ is then composed of all the states that belong to this stability domain plus a terminal state $x^t$ that is reached if the power system goes outside the stability region (see section 9.2.3, for further details on how we determine, on the basis of local measurements, whether the system has left its stability domain).

### 9.2.2 Control problem definition

#### Discrete-time system dynamics

The four-machine power system has continuous-time dynamics which has been briefly described previously. Discrete-time control on such a system means that we

have to set at each $t$ the evolution of the FACTS reactance reference during the time interval $[t, t+1]$. The time between $t$ and $t+1$ is chosen equal to $50\,ms$. For the sake of simplicity we will constrain $X_{\text{ref}}$ to be constant on this time interval[4].
If we denote an element of the control space by the value of $X_{\text{ref}}$ to which it corresponds, we can then state that $C = [-61.57, 0]$. Moreover we have for this control problem $U(x) = C\,, \forall x \in X \setminus \{x^t\}$.

### Controller input signal (observations)

In order to enable the proper operation of the reinforcement learning algorithm in on-line mode all the quantities used by this algorithm must be defined on the basis of real-time measurements that are used as input to the controller and to the learning agent. Since we want a local control algorithm we need to use measurements available close to the location of the FACTS device.
We choose here a minimal set of a single local measurement, namely of the active power flow through the line on which the FACTS is installed. This quantity is obtained at each time step of $50\,ms$, and is denoted by $P_{e_t}$ in the sequel. It is used to define the rewards and pseudo-states used by the reinforcement learning algorithm (including the detection of the terminal state).
Since the measurement is local we will neglect the data acquisition delay.

### Rewards

Our aim is to find a reward function $r(.)$ and a decay factor $\gamma$ such that the policy that maximizes the expected return also leads to the damping of the electric power oscillations in the line. This choice is motivated by the fact that damping improvement of the electric power oscillations in the line can also lead to an overall increase of the power system damping performances.
We have chosen $\gamma = 0.98$ and

$$r(x_t, u_t, w_t) = \begin{cases} -|P_{e_{t+1}} - \overline{P_e}| & \text{if} \quad x_{t+1} \neq x^t \\ -10000 & \text{if} \quad x_{t+1} = x^t \end{cases} \tag{9.4}$$

where $P_{e_{t+1}}$ represents the electric (active) power transmitted through the line on which the FACTS is installed at time $t+1$ and $\overline{P_e}$ represents the electric power transmitted through this line when the system is in steady-state conditions. Justification for such choices are similar to the ones we gave when dealing with the OMIB power system (see section 4.2).

[4]When dealing with the OMIB system, the use of a non-constant FACTS reactance on the interval $[t, t+1]$ did not improve significantly the control law quality (chapter 4).

Note that there exist many equilibrium points for the system, one for each value of $X_{\text{ref}}$ and therefore many values of $\overline{P_e}$. Rather than setting the value of $\overline{P_e}$ at the beginning of the learning process, we therefore estimate its value on-line (see equation (9.6)). This choice has been adopted in order to have still an appropriate reward definition when the power system production/consumption scheme or the topology changes.

### 9.2.3 Reinforcement learning algorithm used in real conditions

In this section we will describe how the reinforcement learning algorithms aimed to solve the control problem can be used in an on-line mode i.e., as if they were interacting with the real power system. Among the different difficulties that such a working mode involves, there is notably the partial observability of the system, due to the fact that we suppose that only a local measurement of the electric power $P_e$ transmitted through the line is available to the learning and control modules. Note that if we manage to find a strategy to make the reinforcement learning algorithm work well in such poor observability conditions, we can reasonably suppose that we could only do even better if the observability conditions were better.

#### The pseudo-state

To define the pseudo-state that will be used inside the reinforcement learning algorithms, we will use the strategy described in section 9.1.3.
Having already chosen the measurement, we take $o_t = P_{e_t}$ and only need to define $Nbo$ and $Nbu$. Concerning these latter, preliminary simulations have shown that a choice that leads to a satisfactory solution in a reasonable learning time is to take $Nbo = 3$ and $Nbu = 2$.
Thus the pseudo-state at time $t$ is defined by the following expression[5] :

$$s_t = (P_{e_t}, P_{e_{t-1}}, P_{e_{t-2}}, u_{t-1}, u_{t-2}). \tag{9.5}$$

Contrary to what has been said in section 9.1.3 we will consider that expression (9.5) holds still valid if $t$ equals 0 or 1 by considering that the values of $P_{e_{-i}}$ and $u_{-i}$ ($i = 1, 2$) correspond respectively to the values of the electric power transmitted through the line and the values of the TCSC reactance reference $i * 50\,ms$ before the reinforcement learning algorithms start interacting with the system.

[5]Other definitions of the pseudo-state $s_t$ could also be used, like for example $(P_{e_t}, P_{e_t} - P_{e_{t-1}}, P_{e_{t-1}} - P_{e_{t-2}}, u_{t-1}, u_{t-2})$. However these were not investigated.

### Terminal state definition

In order to compute rewards and decide when the learning episodes should be ended, we need a criterion to detect the loss of stability.
We decided to use a pragmatic criterion, based only on the local measurements available to the reinforcement learning algorithm.
More precisely, we will suppose that the loss of stability of the system manifests itself when large excursions of the electric power $P_e$ begin to appear (which is the case in the simulations we have carried out). When $|P_{e_t}|$ will be greater than $250$ MW[6], we will consider that the system has gone outside its stability domain and therefore set $s_t$ equal to $x^t$ (the terminal state of the system) rather than define it through equation (9.5). When this terminal state is reached, we consider that a blackout has occurred, and the RL algorithm stops interacting with the system. The RL algorithm starts acting again when the power system has been brought back for some duration to normal operating conditions.

### Real-time reward computation

At each $t+1$ we must be able to compute the reward $r_t$ as defined by equation (9.4) by using only the local measurements we have realized. We have just seen that we considered that the state $x^t$ was reached when the electric power transmitted through the line was larger than $250$ MW. Then if $|P_{e_{t+1}}|$ is larger than $250$ MW, the value of $r_t$ is set to $-10000$. If $|P_{e_{t+1}}|$ is smaller that $250$ MW, one has to know not only $P_{e_{t+1}}$ to compute $r_t$ but also $\overline{P_e}$. To realize an on-line estimation of $\overline{P_e}$, we suppose that $\overline{P_e}$ corresponds to the average value of $P_e$ transmitted through the line and then compute it by averaging the successive values of $P_e$ measured. In all the simulations that will be described hereafter, the $\overline{P_e}$ estimate used in the $r_t$ computation will be given by the expression :

$$\frac{1}{1200}\sum_{k=0}^{1199} P_{e_{t+1-k}} \quad . \tag{9.6}$$

The reason for using only the value of $P_e$ measured in the last $1200 * 50\,ms = 60\,s$ to estimate $\overline{P_e}$ rather than all values of $P_e$ available is to provide the algorithms with some adaptive behavior, which is preferable when the power system operating conditions change.
Note that the way $\overline{P_e}$ is defined consists of having a reward function that does not depend only on the system state and the current action taken but also on the history

[6]The electric power transferred in the line when the system is in steady-state conditions and the TCSC is acting at full range of its capacity is equal $149.81$ MW.

of the actions taken by the reinforcement learning algorithms and the states met. Strictly speaking, the optimal control problem treated is then different from the one stated in chapter 2.

**State space and control space discretization**

The reinforcement learning algorithms have to interact with a system for which the sets $C$ and $S$ contain an infinite number of elements. To deal with such infinite sets, one has at one's disposal two kinds of techniques, the representative states technique and the aggregation technique. Nevertheless we will use in the simulations only the latter one[7].
The $C^{\delta}$ finite set definition that this technique requires will be chosen in the RL algorithms as being either equal to $\{-61.57, -46.18, -30.78, -15.39, 0.\}$ (referred to as $\#C^{\delta} = 5$ in the example) or to $\{-61.57, -30.78, 0.\}$ (referred to as $C^{\delta} = 3$ in the examples). With such a choice for the $C^{\delta}$ sets it can be said that the control variable $u$ has been discretized using a constant discretization step of $\Delta u = -15.39$ ($\#C^{\delta} = 5$) or of $\Delta u = -30.78$ ($\#C^{\delta} = 3$).
Concerning $S$, it must be noted first that due to the fact that we are going to use an *Action Selection* module that will always select an action belonging to $C^{\delta}$, $s_t = (P_{e_t}, P_{e_{t-1}}, P_{e_{t-2}}, u_{t-1}, u_{t-2})$ will necessarily belong to $[-250, 250] \times [-250, 250] \times [-250, 250] \times C^{\delta} \times C^{\delta}$. The aggregation technique requires to define a finite number of disjoint subsets $S_i$ (the equivalent of the $X_i$ subsets) such that $s_t$ always belongs to one of these subsets (to each of these subsets is going to correspond an element of $S^{\delta}$ (the equivalent of the set $X^{\delta}$)).
Each of these subsets we are going to use can be defined by such an expression :

$$\begin{aligned}[-250 + i\Delta P_e, -250 + (i+1)\Delta P_e] \times [-250 + j\Delta P_e, -250 + (j+1)\Delta P_e] \\ \times[-250 + k\Delta P_e, -250 + (k+1)\Delta P_e] \times \{u_A\} \times \{u_B\}\end{aligned}$$

where $\Delta P_e$ can be seen as the discretization step according to $P_e$[8], $u_A, u_B \in C^{\delta}$ and $i, j, k \in \{1, 2, \cdots, max\_step\}$ with $max\_step$ equal to the smallest integer such as the condition $max\_step \geq \frac{500}{\Delta P_e} - 1$ is verified. In the simulations we will study two variants for the value of $\Delta P_e$. In one case it will be chosen equal to 5

[7]The section objective is not to illustrate which technique (aggregation technique or representative states technique) is the best but rather to show that RL algorithms can be applied on a more complex power system than the OMIB system already studied. We preferred the aggregation technique to the representative states technique because it is easier to use.

[8]The same discretization step has been used whatever the value of $P_{e_{t-i}}$ ($i \in \{0, 1, 2\}$) we refer to. Another strategy would have been to take a discretization step varying with $i$.

MW (referred to as $\Delta P_e = 5$ in the examples) and in the other to $2.5$ MW (referred to as $\Delta P_e = 2.5$ in the examples).

### Learning modules

Concerning the *Learning* module used, it will be either composed of a model based technique that uses a Kalman Filter like algorithm as *Estimation of the MDP$^\delta$ structure* module and a Prioritized sweeping algorithm as *Resolution of the MDP$^\delta$* module (denoted by MB in the examples) or a non-model based technique that uses a $Q$-learning algorithm with $\alpha = 0.1$ (denoted by NMB in the examples). Whatever the *Learning* module, the $Q^\delta$-function will always be initialized to 0 everywhere.

### Action selection module

The action selection module used will be an $\epsilon$-Greedy policy with $\epsilon = 0.01$. Note that the value of $\epsilon$ has been chosen sufficiently small to benefit almost completely from the estimate of the optimal control law already computed. When the estimate of the optimal control law has converged, one could set this value equal to 0 because no exploration would be needed anymore. However, because we consider that the power system operating conditions can change (and therefore the optimal control policy) we always keep $\epsilon \neq 0$.

### Assessment of the quality of the obtained policies

Given the huge size of the pseudo state space and the practical impossibility to initialize the system in a controlled way at an arbitrary point of this state space, the score measure defined in chapter 4 to assess the control policies can not be transposed to the present simulations.
Therefore, we define an alternative score measure based on the obtained policy's *return at time t* computed as follows :

$$R_t = \sum_{i=t}^{+\infty} \gamma^{i-t} r_{i+1}. \tag{9.7}$$

At a given time instant this quantity is actually obtained with some delay by observing the rewards obtained at a certain number of subsequent time-steps (i.e. until $\gamma^{i-t}$ becomes negligible).
As we will observe, by computing this value at each $t$ while interacting with the system, we will still be able to get an idea of how good the policy learned by the RL algorithm is.

In the next four subsections we will describe how the RL algorithms we just described behave while interacting with the power system. To simulate the power system we have used the Simulink models [Mat].

### 9.2.4 Case I : control of the stable equilibrium point

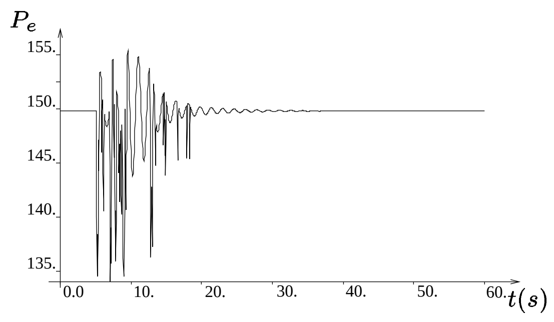


Figure 9.2: The system operates in steady-state conditions. The RL algorithm takes control of the TCSC at $t = 5\,s$.

We suppose that the TCSC is working like a fixed capacitor at full range of its capacity ($X_{\text{FACTS}} = -61.57$) and that the power system is in steady-state conditions. In such conditions the electric power transmitted through the TCSC is constant and equal to 149.81 MW. Then, at one instant, the reinforcement learning algorithm (MB, $\Delta P_e = 5$ MW, $\#C^\delta = 5$) enters into action to control the TCSC.
Notice that the best control strategy one could adopt would be to choose $u$ always equal to $-61.57$ because by proceeding this way the electric power transmitted through the line would stay constant and the reward obtained would be maximal (indeed $P_{e_{t+1}}$ and $\overline{P_e}$ would both be equal to 149.81 MW and the reward $r_t$ would be equal to 0).
But the reinforcement learning algorithm adopts another strategy. Indeed, the first pseudo-state used inside the RL algorithm is equal to

$$s_0 = (149.81, 149.81, 149.81, -61.57, -61.57).$$

Due to the fact that $Q(s_0, u) = 0$, $\forall u \in U^\delta(s_0) = C^\delta$, and that an $\epsilon$-Greedy policy is being used, the value of $u$ imposed by the *Action Selection* module is chosen at random among $C^\delta$. Thus the RL algorithm will certainly in a first time drive the system away from its stable equilibrium point. The question is how far and how long it will drive the system away from this equilibrium point.

To answer this question, we have depicted on figure 9.2 the temporal evolution of the electrical power in the line during the first 60 seconds. We see that, at time $t(s) = 5\,s$, when the RL algorithm begins to act, its interaction with the system indeed creates power oscillations. Nevertheless, quite quickly the algorithm is able to find out how to control the stable equilibrium point. Indeed after $20\,s$ these electric power oscillations have almost disappeared. The RL algorithm thus managed to find quite quickly a control strategy that allows the system to reach again the stable equilibrium point and stay in steady-state conditions.
Notice that, after $1\,h$ of interaction, the number of pseudo-states of $S^{\delta}$ that have actually been visited is equal to $213$, which is only a very small percentage of the total number of such states, which is here equal to about $25 * 10^6$.

### 9.2.5 Case II : damping of self-sustained power oscillations

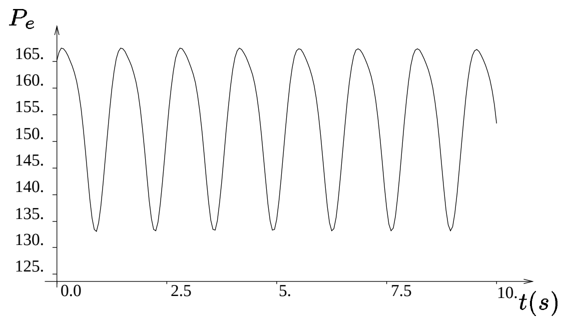
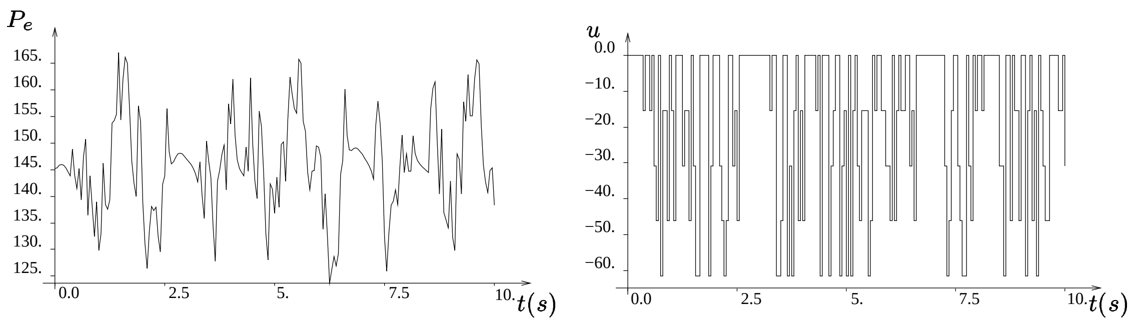


Figure 9.3: Representation of the self-sustained electric power oscillations

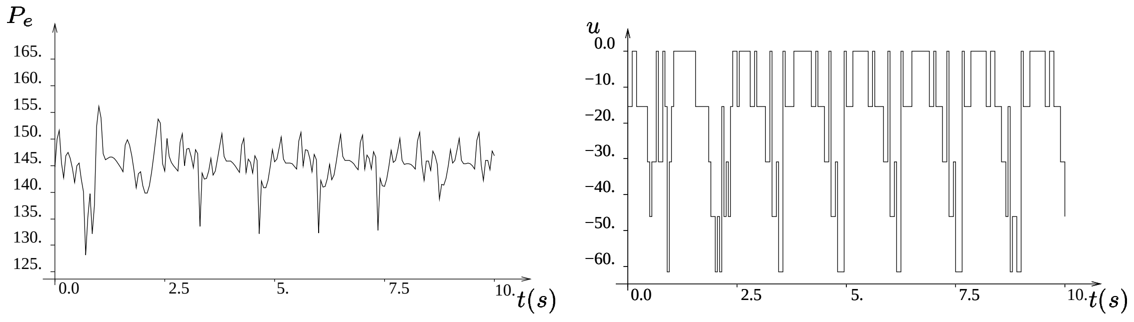
We first consider the case where the TCSC is working like a fixed capacitor at full range of its capacity ($X_{\text{FACTS}} = -61.57$) but where the system dynamics has been (slightly) modified so that the initial stable equilibrium point becomes an unstable equilibrium point. This has been achieved by changing the parameters of the AVR of the machines $G1$, $G2$. Due to the unstable aspect of the equilibrium point, the system will be driven away from it and electric power oscillations will begin to grow. Saturation on the machines' field voltage will however limit the magnitude of these oscillations. Hence, after a certain time a stable limit cycle appears. The evolution of $P_e$ over a period of $10\,s$ when the limit cycle has been reached is illustrated on figure 9.3.
Starting with this behavior, at a certain point in time, the reinforcement learning algorithm (MB, $\Delta P_e = 5$ MW, $\#C^{\delta} = 5$) enters into action to control the TCSC,



(a) $Pe - t$ (b) $u - t$

Figure 9.4: After ten minutes of control

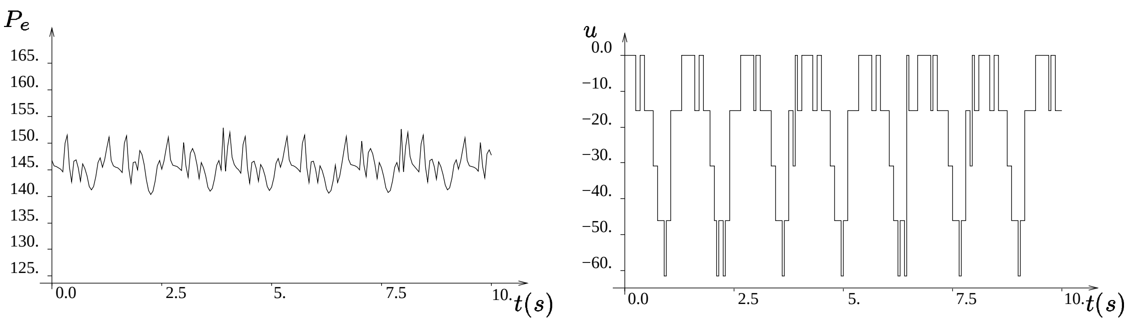
in order to try to progressively reduce the amplitude of the limit cycle. For example, figures 9.4a and 9.4b show the evolution of the electric power transmitted through the line ($P_e$) and the control action taken (i.e. the value of $u$) over a period of $10\,s$, after $10\,min$ of learning control (the reinforcement learning algorithm has imposed each $50\,ms$ the value of $u$ for already $10\,min$). We observe that the magnitude of the $P_e$ oscillations is still very large and that the evolution of the action $u$ seems to be driven by an almost random process. The control algorithm has not yet learned sufficient knowledge about the control problem to act efficiently.



(a) $Pe - t$ (b) $u - t$

Figure 9.5: After one hour of control

After $1\,h$ of control (see figure 9.5a), however, the electric power transferred in the line starts being well damped. At the same time, a more organized structure appears in the sequence of control actions taken (see figure 9.5b).
After $5\,h$ of control (see figures 9.6a and 9.6b), the results are further improved.



(a) $Pe - t$ (b) $u - t$

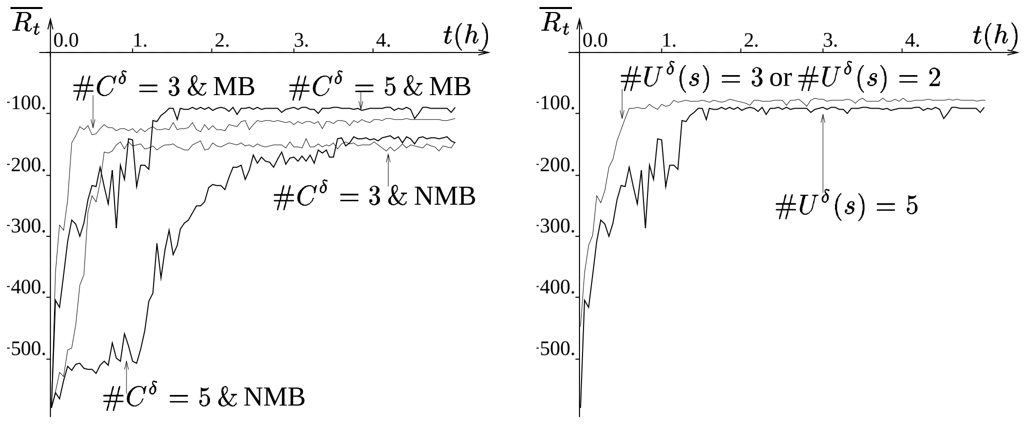
Figure 9.6: After five hours of control

Now, the magnitude of the electric power oscillations has significantly decreased. The variation of the control variable $u$ has a periodic behavior of approximately the same frequency ($0.8\,Hz$) as the electric power oscillations observed when no control occurs. The harsh aspect of the electric power observed comes from the discontinuous variation of $X_{\text{ref}}$. Such behavior could possibly be circumvented by increasing the time delay of the FACTS (image of the time needed by the TCSC to meet the reactance reference $X_{\text{ref}}$) or by using control variables $u$ that do not correspond necessarily to a constant value of $X_{\text{ref}}$ over the time interval $]t, t+1]$.

**Policy quality evaluation and variants**

To determine the quality of the policy obtained at time $t$, we will compute the *return at time t* as defined by expression (9.7) and represent the curves $R_t$ vs $t$ on a figure. But in order to avoid difficulties to visualize the results due to the high variance of $R_t$, we will slightly modify this value by taking its average over the next minute (denoted by $\overline{R_t}$).
The curve sketched on figure 9.7a and labeled "$\#C^{\prime\delta} = 5$ & MB" represents the $\overline{R_t}$ evolution that corresponds to the learning process we have used. As we can observe, this curve converges around a value of $-100$ rather than 0 due to the fact that there still exist small electric power oscillations at the end of the learning period. On this figure three other curves are also drawn. They correspond to simulations for which the size of the control set $C^{\prime\delta}$ has been modified and/or the type of *Learning* module used. This allows us to draw some observations. First we can note that for a same $C^{\prime\delta}$, model based techniques always offer better results. But we can also see that for a same type of *Learning* module, the larger the $C^{\prime\delta}$ size is, the slower

the learning is but the better the solution obtained will be. Note that an increase in $\#C^\delta$ also increases the number of states that compose $S^\delta$ (remind that $s_t$ is equal to $(P_{e_t}, P_{e_{t-1}}, P_{e_{t-2}}, u_t, u_{t-1})$ and therefore that the number of elements of $S^\delta$ increases quadratically with $\#C^\delta$).



(a) $\#U^\delta(s) = \#C^\delta, \forall s \in S \setminus \{x^t\}$ (b) $\#U^\delta(s) \neq \#C^\delta, \forall s \in S \setminus \{x^t\}$

Figure 9.7: $\overline{R_t}$ as a function of the learning time

The simulations reported on figure 9.7b consider another variant of the control space, where the control selected is such that

$$u_t \in \{u_{t-1}, \min(0, u_{t-1} + \Delta u), \max(-61.57, u_{t-1} - \Delta u)\}$$

with $\Delta u = 15.39$ which means that the variation of the control variable from one instant to the other is constrained to be smaller than $\Delta u$. This smaller set of candidate control actions has been suggested by the behavior shown on figure 9.6b, which indeed clearly highlights that the control varies, between two successive time-steps, at most by a value of $\pm\Delta u = \pm 15.39$. Figure 9.7b compares the two cases using the model based RL algorithm. We clearly see that the learning speed is much higher when the reduced control space is implemented (curve labeled "$\#U^\delta(s) = 3$ or $\#U^\delta(s) = 2$" against the curve labeled $\#U^\delta(s) = 5$).
An apparently strange by-product of this enhancement is that also the solution obtained at the end of the learning period is better when taking $U^\delta(s) \neq C^\delta$. Our explanation is that this is actually linked to the $\epsilon$-greediness of the control policy : indeed when an action is taken at random, the choice $U^\delta(s) = C^\delta$ is "more random" than the choice $u_t \in \{u_{t-1}, \min(0, u_{t-1} + \Delta u), \max(-61.57, u_{t-1} - \Delta u)\}$, and hence less favorable to yield large returns.

### 9.2.6 Case III : adaptation to time-varying conditions

The power system configuration is the same as in *Case II* but this time the load is not constant anymore. The load variation is cyclic with a $5\,h$ period[9] and has been modeled according to the equation :

$$z(t) = z(0) - 0.3z(0)\sin(2\pi\frac{1}{5*3600}t), \tag{9.8}$$

where $z$ stands for the active or reactive parts of the load (the $P_0$ and $Q_0$ terms of equation (9.2) ) and where $z(0)$ corresponds to the $P_0$ and $Q_0$ terms used in *Case II*. Moreover, in order to follow the load, the electric power production reference on each machine has also been modeled by equation (9.8).

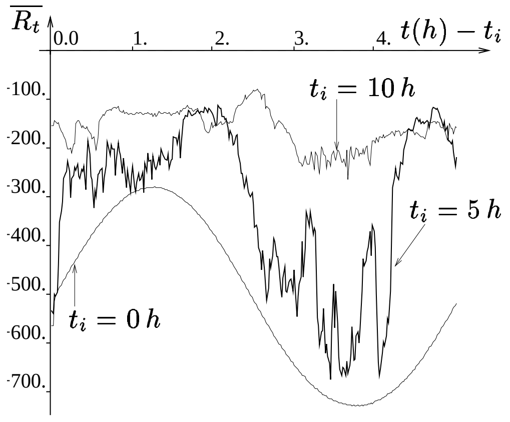


Figure 9.8: $\overline{R_t}$ as a function of the learning time

We handle this case by introducing a forgetting factor in the Kalman Filter like model based learning algorithm ($\beta = 0.95 < 1$) and using a discretization of respectively $\Delta P_e = 5$ and $\#C^{\delta} = 5$. Figure 9.8 represents the resulting evolution of $\overline{R_t}$ over a period of $15\,h$, by showing on the same graph three curves corresponding respectively to :

- the first cycle of 5 hours (curve labeled $t_i = 0h$) : during this cycle the reinforcement learning algorithm is kept inactive and thus the return directly

[9]We have chosen a $5\,h$ period for the load curve rather than a $24\,h$ period in order to lighten the computational burden needed to simulate these 61 state variables power system during several periods. With a larger period, the RL algorithm would have more time to adapt itself to changing operating conditions and the results would probably be even better.

reflects the amplitude and periodicity of the limit cycle which is higher if the load level is higher;

- the second cycle of 5 hours (curve labeled $t_i = 5h$) : here learning is active and slowly improves the behavior;
- the third cycle of 5 hours (curve labeled $t_i = 10h$) : learning is still active and still continues to improve the behavior, by almost completely removing the periodic component.

An important feature that can be drawn from this figure is that even after 1 cycle of control, the RL algorithm continues improving its ability to damp efficiently electric power oscillations. The RL algorithm behaves well for this non-autonomous system due to its capability to adapt itself fast to a new environment.

### 9.2.7 Case IV : damping of oscillations caused by short-circuits

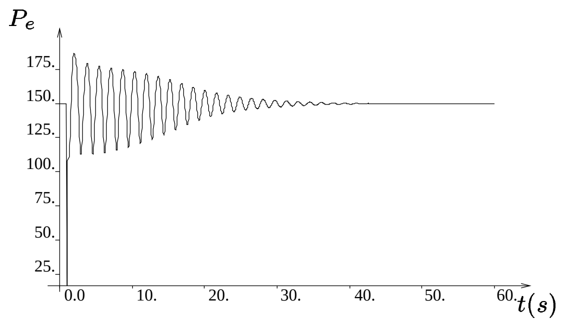
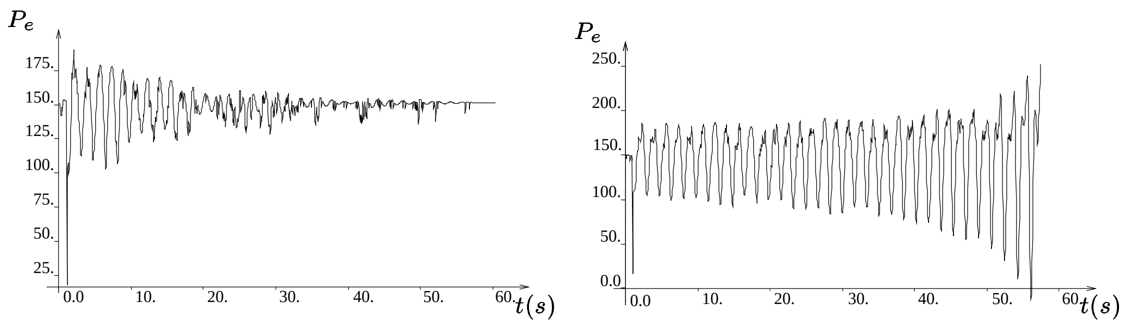


Figure 9.9: Representation of the electric power oscillations caused by a $60\,ms$ three-phase short-circuit at bus $\#10$

Suppose that the system is in the same configuration as in *Case I* and that the TCSC is acting like a fixed capacitor. The evolution of the electric power transmitted through the line is constant and equal to 149.81 MW. Suddenly, a three-phase short-circuit[10] occurs at bus $\#10$ i.e., close to generator $G4$. This short-circuit is self-cleared after $60\,ms$. It causes electric power oscillations in the line on which the TCSC is installed; they are represented on figure 9.9 when the short-circuit occurs at $t = 1\,s$. Due to the natural damping of the power system these oscillations disappear after a certain time.

[10]These short-circuits cause the system to lose its time-invariance property.



(a) First time the short-circuit is met (b) Fifth time the short-circuit is met

Figure 9.10: Representation of $P_e - t$ at different stages of the learning process

Let us see whether the TCSC controlled by means of RL algorithm would be able to speed up this damping or whether on the contrary it would worsen it. To this end, we simulate several scenarios during which several short-circuits are applied successively. We suppose that a learning episode is such that the first fault[11] happens 2 minutes after the RL algorithm ( $\#C^{\delta} = 5$, $\Delta P_e = 5$ MW, MB) started controlling the TCSC and that a new fault occurs after each ten minutes. When the system reaches its terminal state (i.e. if $|P_e| > 250$ MW) a new learning episode is launched.

Under such learning conditions, the evolution of the electric power obtained the first time the fault is met is represented on figure 9.10a. Compared to figure 9.9, one can see that the damping obtained is not better. This is not surprising since the learning of a control law able to damp these oscillations will certainly require to meet several times the fault.

It is surprising that the fifth time the fault is met, the system is driven to instability. This is shown on figure 9.10b where the loss of stability is declared $56.45\,s$ after the fault inception where $P_e$ becomes greater than 250 MW. Rather than damping the electric power oscillations, the TCSC controlled by means of reinforcement learning algorithms has done the opposite : it has increased their magnitude to finally cause a loss of stability. Such behavior is linked to the fact that the algorithms do not have yet enough information about the system dynamics and the reward function to control it efficiently.

[11]A fault will always consist in a $60\,ms$ three-phase short-circuit at bus $\#10$.

## Reducing the probability of loss of synchronism

We attempt to avoid this destabilizing behavior of the reinforcement learning algorithm by making another investigation.

First let us try to understand what happens. Consider first the curve labeled "$U^\delta$ used in the DP equation" on figure 9.11, which represents the (cumulative) number of times the system loses stability as a function of the number of times the fault has been met. The steady growth of this curve shows that indeed the RL algorithm drives the system many times to instability rather than producing some damping for the electric power oscillations, even after a large number of faults have been seen. The explanation of this behavior is linked to the way the $Q^\delta$-function has been initialized. Indeed, suppose that the state $s^\delta$ has already been met but that not all state-action pairs $(s^\delta, u), u \in U^\delta(s^\delta)$ have already been visited. In that case the $V^\delta(s^\delta)$ value that intervenes in the resolution of the MDP$^\delta$ problem will be equal to $0$ which is the best $V^\delta$ value a state can have due to the fact that the reward (as we have defined it in these simulations) is always negative. Thus, each time that a state is encountered for which all possible actions have not yet been taken a problem may arise. More precisely, if in some already visited state $s^\delta$, some actions have not yet been tried out, the resolution by DP of the MDP$^\delta$ structure will actually lead to a policy that tries to make the system reach such states due to the high expected $V^\delta$ value they are believed to have. This causes the RL algorithms to somehow do the opposite of what is desired, because it drives the system to visit many times areas that are close to the stability boundary, which implies of course a high risk of loss of stability.

One way to avoid such phenomena could be to initialize the $Q^\delta$-function to a pessimistic value rather than zero but this strategy will provide the algorithm with less exploratory behavior at the beginning of the learning (remind that the choice of the action is done according to the $Q$-function). Of course this could be circumvented by changing the policy used. A more elegant way to deal with this problem is to solve the MDP$^\delta$ structure differently by considering in the solution process only the state-action pairs that have already been visited. Notice that this idea has already been discussed in section 5.3.4. If we proceed this way in the present simulations, we obtain the curve labeled "$U^{\delta'}$ used in the DP equation" of figure 9.11. We observe that, under the same conditions this strategy leads to a much smaller number of stability losses (only four times, the loss of stability occurring the $75$th time the fault is met), which is much more acceptable.

In the subsequent simulations we will always use this modified version of the DP algorithm.

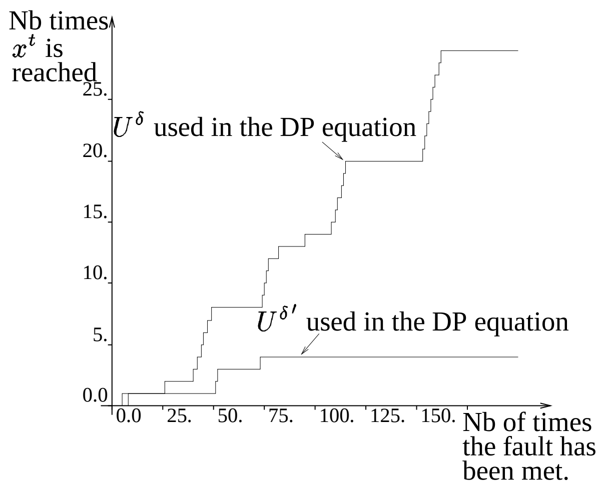


Figure 9.11: Number of times the loss of stability has been observed

## Policy quality evaluation

The value of $R_t$ observed after convergence of the RL algorithm $1\,s$ before the fault inception and $59\,s$ afterward is represented on figure 9.12 by the curve labeled "RL algorithm". At the beginning of the curve, the value of $R_t$ is very negative because large electric power oscillations occur (see figures 9.13a and 9.13b for the corresponding electric power oscillations representation). As the magnitude of these oscillations decreases, $R_t$ increases to finally equal 0 when they disappear.
On figure 9.12 we have also drawn the value of $R_t$ we would have observed by a policy that would consist in using $u$ constant and equal to $-61.57$, rather than the $\epsilon$-Greedy policy. We observe that the results obtained when the TCSC is controlled by the RL algorithm are significantly better.
The evolution of $u$ that corresponds to the $\epsilon$-Greedy policy is sketched on figures 9.14a and 9.14b. To large oscillations of $P_e$ correspond large variations of the control variable $u$. When the magnitude of the oscillations decreases, the $u$ variations tend also to decrease. The value of the control variable imposed by the RL algorithm when the oscillations have disappeared becomes constant (except when a non-greedy action is taken) and equal to $-61.57$.

## Finer discretization of pseudo-states

At this point one may perhaps be disappointed by the damping produced by the RL algorithm. But one should keep in mind that many factors can foil the results

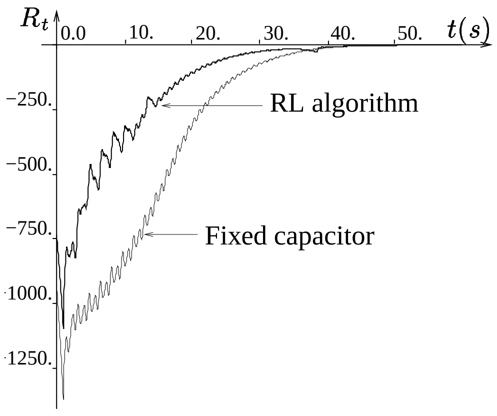


Figure 9.12: $R_t$ obtained after convergence of the RL algorithm compared to $R_t$ obtained while using a fixed capacitor. $\Delta P_e = 5$ MW

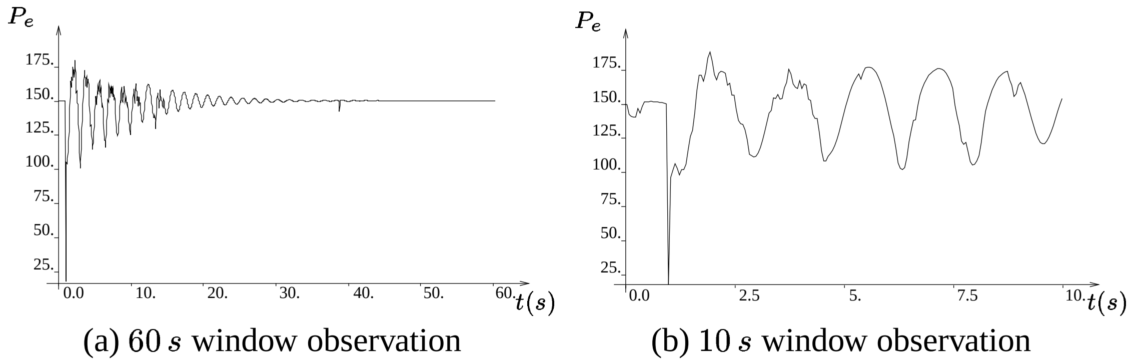
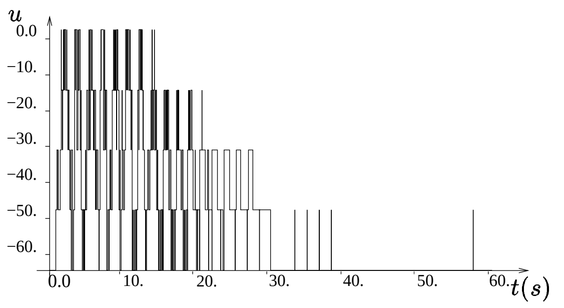
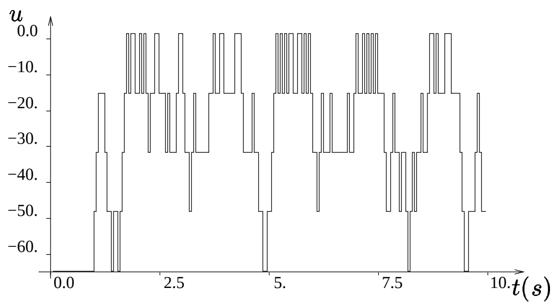


Figure 9.13: Representation of $P_e - t$ after convergence of the algorithm. $\Delta P_e = 5$ MW



(a) 60 $s$ window observation



(b) 10 $s$ window observation

Figure 9.14: Representation of $u - t$ after convergence of the algorithm. $\Delta P_e =$ 5 MW

notably concerning the way the state has been defined or even the way the infinite set $S$ has been discretized. For example, if a finer discretization step on $P_e$ is used ($\Delta P_e = 2$ MW rather than $\Delta P_e = 5$ MW), the results obtained and represented on figure 9.15, 9.16 and 9.17 are slightly better. But the price to pay for these improvements has been an increase in the learning time because the number of $s^\delta$ states that have been visited by the RL algorithm is now higher. Similarly, it can be shown that if more observations and more actions are used to define a state then the damping quality increases as well as, unfortunately, the learning time.

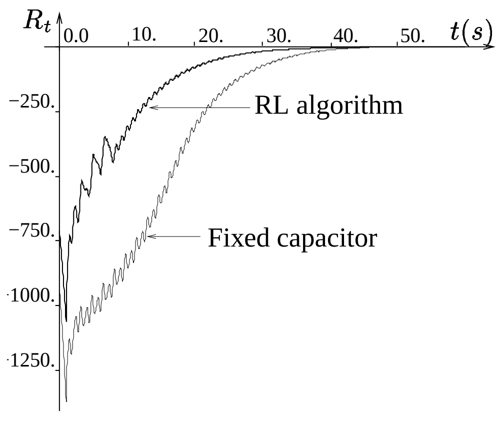
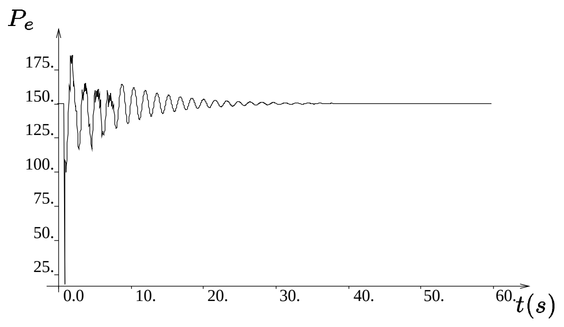
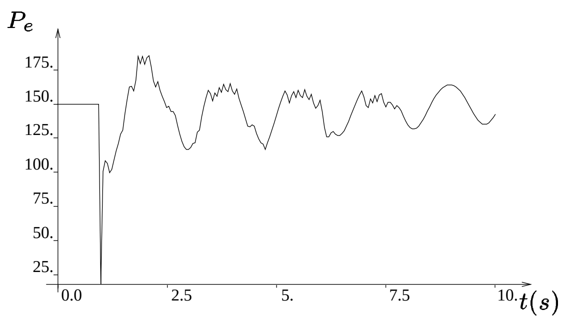


Figure 9.15: $R_t$ obtained after convergence of the RL algorithm compared to $R_t$ obtained while using a fixed capacitor. $\Delta P_e = 2$ MW

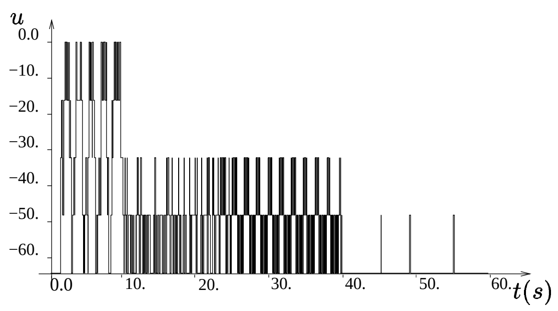


(a) 60 $s$ window observation

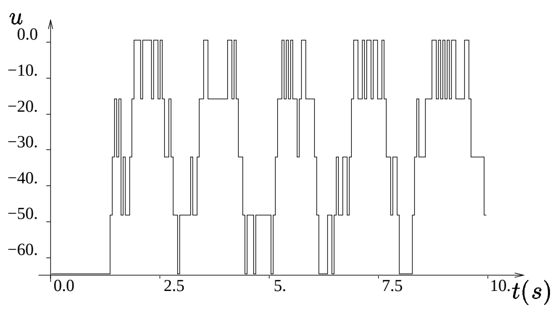


(b) 10 $s$ window observation

Figure 9.16: Representation of $P_e - t$ after convergence of the RL algorithm. $\Delta P_e = 2$ MW



(a) 60 $s$ window observation



(b) 10 $s$ window observation

Figure 9.17: Representation of $u-t$ after convergence of the RL algorithm. $\Delta P_e = 2$ MW

**Rewards obtained during the faulted configuration**

Another factor that corrupts the learned control law is linked to the fact that the RL algorithm does not distinguish the no-fault from the fault configuration, two configurations in which the system dynamics differ a lot. By way of example suppose that the system is in steady-state conditions and that the TCSC works at full range of its capacity. The pseudo-state is equal to $(149.81, 149.81, 149.81, -61.57, -61.57)$. Suppose that while being in this pseudo-state, the three-phase short-circuit occurs. This short-circuit will drive the system in state space areas where the rewards obtained are bad (large electric power oscillations) and the RL algorithm will interpret these bad rewards as a consequence of being in the pseudo-state $(149.81, 149.81, 149.81, -61.57, -61.57)$ rather than due to an outside perturbation. This false interpretation foils the estimate of the $Q$-function. One way to avoid such a problem is to never use the four-uples $(s_t, u_t, r_t, s_{t+1})$ as input of the *Learning* modules each time $s_t$ or $s_{t+1}$ are composed of electric power values that correspond to the fault configuration.

## 9.3 Summary

*In this chapter we have discussed and illustrated a certain number of problems that have to be addressed when reinforcement learning algorithms are applied to real world control problems.*
*Many of our considerations being driven by applications in the context of electric power system control, they have also been illustrated by a case study on a realistic electric power system FACTS control problem. During this case study we could also identify and explain some additional problems which would have been difficult to anticipate beforehand, such as the the destabilizing effect of the initialization of the $Q$-function as well as the necessity to provide adaptive capabilities to the learning algorithm and the effectiveness of the latter in realistic conditions.*
*While, obviously, there is still a lot of work to do before this methodology reaches complete maturity, we believe that these simulations on the one hand demonstrate the feasibility of the reinforcement learning approach in real world power system problems, and on the other hand underline the flexibility of the overall framework.*
*Last but not least, we believe that the discussions of this chapter highlight the fact that in order to successfully apply these techniques in real-world conditions it is necessary to combine a deep understanding of the theory and algorithms behind the approach with physical understanding of the practical problem addressed. In this latter respect, the large scale non-linear and multifaceted power system control problems offer a particularly interesting and challenging ground of experimenta-*

*tion. In the next chapter we will discuss why these applications also offer very strong practical motivations for new control methodologies such as those developed in this thesis.*

Chapter 10

# Power system control and reinforcement learning

*This chapter is intended for power system engineers who want to grasp the possibilities that reinforcement learning methods offer to control power systems.*
*In this chapter, we:*

- *expose our viewpoint on why power system control becomes more and more important;*
- *review the different elements that make up a control scheme, the new technical possibilities we have to design present-day power system control schemes and the difficulties associated therewith;*
- *describe a general procedure to design such control schemes;*
- *present reinforcement learning methods as a way to design agents and explain how they can meet power system control needs;*
- *review different power system applications that have already been tackled by using reinforcement learning methods.*

## 10.1 Power system control : a high priority

*Power system stability* is the property of a power system which enables it to remain in a state of equilibrium under normal operating conditions and to regain an acceptable state of equilibrium after being subjected to a disturbance.

All around the world power system stability margins can be observed decreasing. The reasons are multiple. We only point out three main reasons.

- *The inhibition of further transmission or generation constructions by economical and environmental restrictions*. In many power systems, system capacity has not kept pace with population growth or increased per-capita use of electricity (load increase). As a consequence, transmission and generation must be operated closer to their limits, with smaller security margins [Wil97, Kun00, oAGoSC97].

- *The restructuring of the electric power industry*. Restructuring processes are performed in some power systems in order to implement competition at the generation level, by unbundling vertical utilities into independent generation, transmission and distribution companies. The unbundling processes decrease the stability margins due to the fact that power systems are not operated in a cooperative way anymore. It notably implies that generation companies can operate in a way that threatens system security [DeM98].

- *The multiplication of pathological characteristics when power system complexity increases*. Complex power systems exhibit many complex physical behaviors that can lead to dramatic consequences. By way of examples we can cite the risk of large scale oscillations originating from nonlinear phenomena, frequency differences between weakly tied power system areas, interactions with saturated devices, interaction among power system controls, ... [DeM98, oAGoSC00].

Beyond a certain level, the decrease in power system stability margins can lead to unacceptable operating conditions and/or to frequent power system collapses. One way to avoid these phenomena i.e., to increase power system stability margins, is to control power systems more effectively.

## 10.2 Power system control : a generic description

All power system control schemes are characterized by three basic elements :

- the device(s) that is (are) utilized to influence the power system dynamics. It can be a breaker, the excitation system of a generator, a FACTS, ...

- the agent(s) that controls (control) the device(s). It can be the logical rule that switches on/off a breaker, a computer that determines new generation patterns from the analysis of system security margins with respect to credible contingencies, a PID controller, . . .

- the observations realized on the power system and sent to the agent(s). These carry information about the system topology, voltage at some buses, frequency of the system, . . .

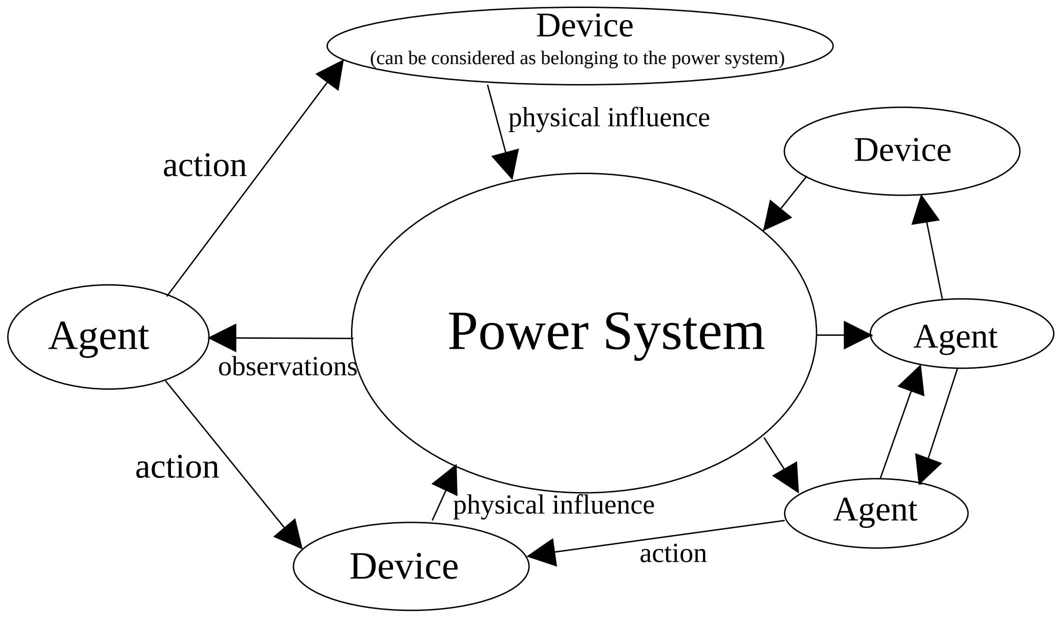


Figure 10.1: Power system and control : observations, agent, action, device and physical influence

On figure 10.1 we have sketched some power system control schemes. Observations are realized on the power system and transmitted to agents that process them in order to control appropriately the devices they are responsible for.
The main features of power system control schemes are as follows.

- *Many agents can control the same device.* An Automatic Voltage Regulator and a Power System Stabilizer can be mentioned as example. Both these agents control the generator excitation system. The AVR is aimed to modify the excitation for the generator output voltage (or the voltage at another location of the power system) to reach a specified value while the PSS function has to control the generator excitation system to provide positive damping torque to power swing modes.

- *A same agent can control many devices*. This is notably the case for an agent responsible for system separation upon detection of an impending instability since system separation can imply to open many lines, possibly combined with load shedding.

- *All agents do not observe the same information*. The reason why all the observations realized on a power system are not transmitted to a single agent that would control all the devices is double. Firstly, such a centralized control scheme would be particularly vulnerable to communication problems. Secondly, some power system phenomena need the devices to react in a time inferior to the one needed to transmit data over long distances.

- *Agents can communicate with each other.* This is notably true in relay coordination. When a relay (the relay plays the agent role and breakers are the devices it controls) identifies a "target", it transmits "transfer trip" signals to other relays. Relays that receive these transfer trip signals can use them to either initiate or block actions [PT88].

Today there are many new possibilities to implement control schemes at our disposal, a few of which are highlighted below.

- *New control devices* that can influence the power systems dynamics. These new devices are mainly linked to the power electronics (SVC, TCSC, . . . ).

- *Better communication capabilities and data acquisition techniques* so that agents can get a better observability of the environment and send control actions to remote control devices (possibility of centralized controllers). Among the most striking improvements in data acquisition techniques there are the Phasor Measurements Units (PMU). These PMUs provide extremely accurate stamping of the power system information by using the Global Positioning Satellite (GPS) time signal [Pha93]. Therefore phase measurements coming from remote places of the power system can be used. These are particularly welcome for control schemes aimed to damp power system oscillations between two areas of a power system.

- *Better computational capabilities*. The increase in computer capabilities allows the realization of more complex studies to design control schemes. It also makes feasible the building of more intelligent agents that use vast amounts of computation to process the observations. To set an example, one can imagine that after a disturbance inception, an agent would simulate the system faster than the real time phenomena, in order to determine whether the system is going to be unstable or not. If the agent finds out that the system is driven to instability, it could compute the appropriate corrective actions that would save the power system stability. However for fast instability phenomena (like transient stability phenomena) for which the instability can occur a few hundred milliseconds after the fault inception, such procedures are still out of reach of today's computers capabilities.

All these new technological advances give the opportunity to realize more efficient control schemes. But they also increase the burdens associated to the control schemes design task because it implies the choice of the best one of the many possibilities that are offered. In this respect the power system swing modes damping problem can be mentioned. Solutions commonly used consisted in adding PSS to the generators. These PSS were using local measurements and were usually designed by studying the power system behavior around some equilibrium points (linear analysis of the power system). Nowadays there exist many possibilities to solve this problem. One can for example use global measurements as PSS input, install FACTS devices at appropriate locations, design the PSS by using more sophisticated methods than the ones linked to the linear analysis, . . . .

## 10.3 Design of control schemes : a systematic approach

The design or improvement of a control scheme is generally triggered by the fact that some power system weaknesses have been identified (either by simulation or by observing the real power system). When this happens, an appropriate family of candidate control schemes possibly able to counteract them has first to be identified. Then, among these candidate schemes a particular one has to be identified as the solution to be implemented in the field. Let us formalize this procedure in a systematic way.
Basically, to design a control scheme, we start with at our disposal a set of observations $O$ and a set of devices $D$ (these devices already exist or can potentially be installed at particular locations of the system under consideration).
Let $\mathrm{CS}^n = \{(o_1, a_1, d_1), \cdots, (o_n, a_n, d_n)\}$ represent an $n$-agent control scheme, where $o_i$ and $d_i$ are subsets of respectively $O$ and $D$, and where each $a_i$ denotes one agent that processes the informations contained in $o_i$ to control the devices comprised in $d_i$.
To decide whether one such a multi-agent control scheme is better than another one, several factors must be taken into account : their ability to counteract the power system instability, their cost of implementation, their technical feasibility, their reliability, . . .

To identify the best control scheme, one should (in principle) iterate on all the possible combinations $\{(o_1, d_1), \cdots, (o_n, d_n)\}$, identify for each one the best combination of controlling agents $a_1, \cdots, a_n$, evaluate each scheme and sort out the best one.
Clearly, in any realistic application the number of possible control schemes is virtually infinite. But it is also true that, in most cases, some engineering knowledge, cost considerations and technical constraints can help to strongly reduce the number of candidate combinations $\{(o_1, d_1), \cdots, (o_n, d_n)\}$ on which one would actually have to iterate.
Note however that the design of the agents $a_i$ can reveal itself to be much more difficult. There are indeed several reasons for this.

- Power systems becoming more and more complex (increase in size, multiplication of the number of components linked to the power electronics, ... ), some of the simplifying assumptions and intuitive reasonings usually done to design agents (based on some strong assumptions on the power system dynamics) do/will not hold true anymore. This problem could be overcome by relying more heavily on power system simulations.

- One must ensure that the control schemes do not decrease the performances of other control schemes or even create stability problems that did not exist before.

- Control schemes are usually designed from a power system model. Even if good power system models exist, they never represent truly the real power system. Strong assumptions are done in the way they are modeled [PS02]. The agents must therefore be robust in order to be able to act correctly even if there exist some discrepancies between the model and the real power system.

- If the control schemes work well for some topologies or some operating conditions of the power system, nothing guarantees that they will behave adequately for other power system configurations. To overcome this problem, one can design the agents such that their control strategy is robust. Another solution consists in designing the agents such that they adapt themselves to the changing environment. Agents installed on a power system that have a changing behaviour do usually rely on outside interventions to change their input-output relations. When dealing with new operating conditions, new computations are done at a higher level (like computations done by a higher level agent) and new control laws are sent to agents.

- The disturbances (short-circuit, sudden loss of a transmission element, . . . ) that occur on a power system are unpredictable to a large extent. Therefore the design of the control scheme must be realized by using a set of plausible disturbances. Because these disturbances rarely match the real ones, the need for robust agents is further enhanced.

Actually, the methodology based on reinforcement learning developed in this thesis provides a systematic approach to design such controlling agents. This is discussed in the next section.

## 10.4 Reinforcement learning as a tool to design agents

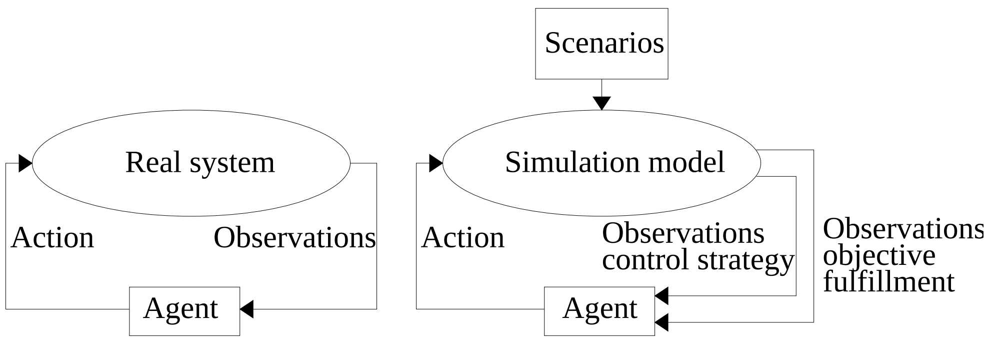
Reinforcement learning methods can be seen as a panel of methods that allow agents to learn a goal oriented control law from interaction with the system. The reinforcement learning driven agents make observations on the system, take actions and observe the effects of these actions. By processing the experience they accumulate from interaction with the system they learn an appropriate behavior (or control law) i.e., how to associate suitably actions to the observations realized in order to fulfill the specified objective. The more experience they accumulate, the better the quality of the control law they learn. The learning of the control law from interaction with the system, the goal oriented aspect of the control law and the ability to handle complex control problems are three distinguishing characteristics of reinforcement learning methods.

### 10.4.1 Two modes of application

We have sketched on figure 10.2 the two modes of application for reinforcement learning introduced in the preceding chapter. They are further discussed below.

#### On-line mode

The on-line mode of application consists in using the reinforcement learning driven agent directly on the real system. This mode of application is sketched on figure 10.2a. Its main drawback originates from the fact that at the beginning of the interaction no experience is available to the RL driven agent to control adequately the system. This absence of knowledge can lead the agent to control badly the system (i.e. to be unable to fulfill well its objective) and therefore jeopardize its stability. One solution to this problem is to use first the agent in a simulation environment. Another solution is to use the RL driven agent in combination with a “traditional



(a) On-line mode (b) Off-line mode

Figure 10.2: Two modes of application

control law" that can already ensure a "good" power system behavior and to switch to the learned control law only when its quality becomes sufficient.

### Off-line mode

In an off-line mode, the RL driven agent interacts with a simulation model of the system (figure 10.2b). Once the agent behavior is sufficiently good,

- one may extract the control law it has learned in order to implement it on the real system. If no further learning occurs on the real system (the control law learned in the simulation environment is used at such), the observations the agent using the learned control law needs can be lesser than the ones required by the RL driven agent in the simulation environment. Indeed it can get rid of the observations needed to quantify how good it fulfills its objective. These observations are identified as "Observations objective fulfillment" in figure 10.2b and the ones used for the control strategy (i.e. the observations needed to identify a situation to which control actions are associated) as "Observations control strategy". As way of example, suppose that a PSS is aimed to damp inter-area oscillations. The observations needed to quantify the objective fulfillment could be the frequency in each area while the observations used to map situations to action could be limited to local measurements;

- one may implement the RL driven agent on the real system where it will benefit from the experience it has acquired in the simulation environment and still be able to improve its behavior from interaction with the real system.

### 10.4.2 Usefulness of RL methods to power system control

Reinforcement learning methods can reveal themselves to be an interesting tool for power system agents design for several reasons enumerated below.

- These methods work without making any strong assumptions on the power system dynamics. Their only requirement is to be able to interact with the system. These methods can therefore be applied to design all practical types of control schemes.

- The control law these methods learn are closed-loop control laws known to be robust. This closed-loop aspect is important notably when the real power system is facing random perturbations that were not accounted for in the simulation model. While a closed-loop control (or feedback) law will tend to correct automatically for these, an open-loop (or precomputed) control law takes no account of them.

- If the learning occurs in an on-line mode, there is no need for a power system model. This is particularly interesting when it is difficult to model the power system or when some phenomena are difficult to reproduce in a simulation environment[1].

- Even if some learning occurs first in an off-line mode, the additional learning that takes place on the real system can improve the learned control law by adapting it to the real system.

- RL methods open avenues to adaptive control since the RL driven agents learn continuously and can therefore adapt to changing operating conditions or change of power system topology.

- RL methods can be used in combination with traditional control laws to improve their performances. As way of example, they could be used to determine parameters of control laws obtained by realizing a linear analysis of the power system in which case the RL driven agent does not control directly the device but well some parameters of another agent (representing the linear control law) responsible for the device control.

- They open avenues to the design of a vast panel of intelligent agents that could help operators to run efficiently a power system.

[1]Notice nevertheless that it would be difficult to convince a utility to install and use a new class of control device in situations in which it is hard to reproduce the phenomena in simulation and hence demonstrate the efficacy and safety of the control in simulation.

### 10.4.3 Multi-agent systems

When many RL driven agents interact at the same time with a system, each of them will try to act in order to fulfill at best its own objective as reflected by its own reward signal. If the agents have divergent objectives they may enter into competition. However, if each power system agent has roughly the same objective i.e., to enhance power system stability, these competitions between agents can be naturally limited. The key aspect for making all the RL agents work well together is to be able to specify their goals such that satisfaction of one agent goal does not prevent some others to reach theirs[2].

## 10.5 Some problems already tackled by RL methods

The power system control problems that have already been tackled by using reinforcement learning methods can be classified into two categories. The first one concerns the controllers that act continuously, like an Automatic Voltage Regulator (AVR), a Power System Stabilizer (PSS), a turbine speed regulator, a FACTS device controller, an Automatic Generation Control (AGC) ... The second one concerns controllers that act only to avoid power system breakdown after unforeseen events, such as a very rapid load increase or a severe fault that occurs during stressed operational conditions. These controllers are referred to as System Protection Schemes (SPS)[3] [38.01] and the actions they take to counteract power system instability phenomena are for example generation shedding, turbine fast-valving, gas turbine start-up, load shedding, braking resistors, tap changers blocking ...
When dealing with continuously acting controllers, the amount of experience the agents can gather from interaction with the real power system can be sufficient to use the RL driven agents in an on-line mode. This is not the case anymore with the System Protection Schemes since they rarely enter into action and therefore produce little "real experience".
Hereafter we review some of the applications that reinforcement learning methods have already met in each of these categories[4].

[2]The conflicts of interests that may occur between RL driven agents having different objectives have been used to model the different actors of a power system in a simulator known as SEPIA [HBWS00]. In this simulator, the agents representing the generator companies have a reinforcement learning module to help them to make price decisions when responding to requests for quotes. Generator companies make their price decision so as to maximize their individual payoff without worrying about the payoffs obtained by other companies.

[3]Also known as Special Protection Schemes

[4]RL applications to power system control are still in their infancy. Many of the works reported hereafter have been realized at the University of Liège in the service of Stochastic Methods.

### 10.5.1 Continuously acting controllers

#### Control of a TCSC

Results obtained by controlling a TCSC by using RL methods are reported in this work as well as in two papers : [Ern01] and [EW02].
In all these works, the RL driven agent controls the TCSC in order to damp electrical power oscillations in the transmission line on which it is installed.
The main conclusions that can be drawn from these works are :

- observations that the RL driven agents need to control successfully the TCSC can be limited to local measurements
- they are able to handle non-linear phenomena
- they provide robust control strategies
- when used in an on-line mode the RL driven agents first need to acquire some experience in an off-line mode to avoid the endangering of the power system stability.

#### The automatic generation control problem (AGC)

Another application of RL methods to continuously acting controllers is done in [ARS02]. The authors reformulate the automatic generation control (AGC) problem as a stochastic multistage decision process and solve it in an on-line mode by using RL algorithms.

### 10.5.2 System Protection Schemes

Usually these system protection schemes are designed heuristically by using trial and error methods and result in open-loop control rules. These open-loop control rules act in a case-based way (e.g. if one transmission line is lost then two generators are tripped) and do not take into consideration the real state of the system that is reached after the fault.
Due to the fact that RL driven agents compute closed-loop control laws (they learn how to map situations to actions) they can take into consideration the real system state that is reached in the post-fault configuration. Moreover, the closed-loop nature of the control law will make it much more robust.
Among the applications of RL algorithms used to System Protection Schemes design one can mention two works reported in [DEW00] and [GEW02]. In the first

article RL algorithms are used to design a generation tripping scheme for the Brazilian power system and in the other one they are used to demonstrate their ability to control a dynamic brake. The content of these two papers is briefly discussed hereafter.

### Generation tripping

The power system considered is the Brazilian power system and the generation tripping scheme is implemented at the Itaipu power plant. It is an hydraulic power plant with 8 machines of 700 MW. The power plant is connected to the South part and the South East part of the power system through long transmission lines. When a severe contingency occurs, like a short-circuit followed by the loss of a transmission line, all the 8 generators reach within a few hundred milliseconds their speed limit and are disconnected due to overspeed protection. The sudden loss of 5600 MW of generation power can endanger the whole system integrity. The aim of the closed-loop control law computed by using RL algorithms is to associate to the angle and speed (angle and speed of the machines are the observations used by the RL driven agents) of the hydraulic power plant generators control actions that consist in shedding some machines before they reach their speed limit. Rather than losing all the generators, only a few ones are shed, which diminishes the contingency effects.
The control law computed by means of RL algorithm has the main advantage to shed a number of generators proportional to the actual contingency severity and not to the severest contingency unlike in the case with the SPS presently installed.

### Dynamic brake

The power system used is an academic four-machine power system whose main characteristics are reported in [Kun94].
The agent that controls the dynamic brake has a threefold objective : to damp large electromechanical oscillations, to avoid the loss of synchronism between the generators when a severe incident occurs and to limit the time the dynamic brake is switched on.
The learning is realized by assuming that the system can be decomposed into two areas such that the machines swing coherently inside each area and that one area swings against the other. The RL driven agent uses as observations an image of the frequency difference between the two areas. The control law obtained fulfills the objectives and is even able to control the system successfully when some incidents unseen during the learning phase are encountered.

## 10.6 Summary

*In this chapter we have first considered the problem of power system control from a very general point of view, discussing the needs for more effective control systems and the interest of systematic methodologies to help designing them.*
*The second part of the chapter is a "plaidoyer" for the use, in this context, of the reinforcement learning approach investigated in this thesis.*
*We would like here to apologize for those works which we have forgotten to cite in our short, and certainly not exhaustive, review of some publications, and reinsist that we are quite concious of the fact that reinforcement learning is only one of the many "hammers" that can be used to handle our "nail".*

# Chapter 11

# Conclusions and future prospects

## 11.1 Closure

A main objective of the research reported in this thesis has been to explore how reinforcement learning methods can be used to solve power system control problems and to show the technical feasibility of these methods.

We have seen that these methods have two classes of applications to power system control. The first one consists in using these reinforcement learning methods to design intelligent agents that interact directly with the real power system and learn from this interaction how to control it. These agents learn continuously and are therefore able to adapt their behavior to changing operating conditions and modifications of the power system structure. Moreover, as the learning is performed on the real system, there is no need to worry about the accuracy of a power system model. This first class of applications has met one main difficulty, namely, the reinforcement learning driven agents could drive the system to unacceptable operating conditions at the beginning of learning due to the lack of information they had about the power system. Nevertheless, some solutions have been proposed to overcome this difficulty. The second class of applications was related to the use of reinforcement learning algorithms in a simulation environment and to the implementation of the control law they have learned on the real power system. We highlighted the many advantages of these learned control laws. Among them we mention their ability to handle non-linear phenomena and their natural robustness.

The reinforcement learning methods have been presented as methods, which allow one to learn the solution of optimal control problems from interaction with the system. The solution of the optimal control problems taken into consideration could be characterized by a $Q$-function defined on the state-action pairs set. The learning consisted in determining this $Q$-function from interaction with the system. We explained the difficulties to learn this function when the state space and/or the control space are infinite. The strategy adopted to overcome these difficulties was to use an approximation architecture to represent the $Q$-function and to learn the parameters of this approximation architecture. Two types of approximation architecture have been studied as well as many algorithms that allowed to learn the parameters. These algorithms can be classified into two families. The first reconstructs the structure of a finite MDP and uses its solution to compute the value of the approximation architecture parameters while the other one learns the parameters directly.

After having applied the different RL algorithms to control a TCSC installed on the OMIB power system, we have used them to design intelligent agents aimed at controlling a TCSC installed on a much larger power system. Although many technical constraints have been taken into account, notably the ones linked to the state partial observability and the learning time, the results obtained were very encouraging.

## 11.2 Problems, solutions and future research prospects

In this section we review some of the problems addressed in this thesis as well as the proposed solutions. We also discuss some future research prospects.

• *Resolution of a* **finite** *state space optimal control problem*[1] *with* **known** *system dynamics and reward function (labeled as Problem I in the Introduction).*

This problem has been treated in chapter 2 by using classical DP algorithms.

• *Resolution of an* **infinite** *state space control problem with* **known** *system dynamics and reward function (labeled as Problem II in the Introduction).*

In chapter 3 two techniques that we named the aggregation technique and the representative states technique have been introduced to solve this type of control problems. Both techniques consisted in defining from the infinite state space control problem knowledge a finite MDP. The solution of the MDP was used to approximate the solution of the infinite state space control problem.

It would be interesting to attempt to define the finite MDP structure so as to approximate at best the initial optimal control problem. For example, in the context of the representative states technique this would require selecting the most representative states and defining the most appropriate notion of distance between the states.

Other techniques, like the ones linked to the Bellman error method have not been investigated but still face up roughly the same challenges since their performance

[1]We remind that the expression "**finite** state space control problem" has been used throughout this thesis to designate control problems having a number of states suitable for computers capabilities.

depends on the states selected for the minimization process and on the approximation architecture used to represent the $Q$-function.

• *Resolution of an optimal control problem with* **finite** *state space and* **unknown** *system dynamics and reward function, the absence of knowledge of these elements being compensated by the possibility of interacting with the system (labeled as Problem III in the Introduction).*

Two families of reinforcement learning algorithms to solve this problem have been thoroughly investigated. The first one, known as *model based*, reconstructs the structure of the optimal control problem and solves it while the other one, known as *non-model based*, computes directly the optimal control problem solution. Convergence proofs of these two families of algorithms are well established in the literature.

However, a better investigation of the performances of model based versus non-model based algorithms could be interesting. In the examples treated, model based methods were always performing better.

• *Resolution of an optimal control problem with* **infinite** *state space and* **unknown** *system dynamics and reward function, the absence of knowledge of these elements being compensated by the possibility to interact with the system (labeled as Problem IV in the Introduction).*

The algorithms we have used to solve these problems can be classified into two categories. The first is related to algorithms that learn the structure of a finite MDP, solve it and extrapolate its solution to the initial control problem (model based algorithm) whereas for the second category the learning consists in adapting directly the parameters of an approximation architecture used to represent the solution. The best results have been obtained with the model based algorithms. Those used when the states of the finite MDP correspond to states of the initial control problem (model based algorithms used with the representative states technique) were specific to this thesis and provided very promising results both in terms of learning speed and quality of the solution obtained.

Admittedly, the ideal algorithm should be able to combine high learning speed and quality of solution. In other words, it should combine ability to adapt automatically the approximation architecture to the learning experience (the more learning experience is gathered in a region, the finer the approximation architecture can be in this region) and/or to the control problem solution itself (a fine approximation architecture is required in state space regions where the solution characteristics vary a lot).

• *Partial observability of the system.*

The use of RL algorithms in partially observable environments has only been briefly discussed. Basically the procedure we used consisted in defining a pseudo-state for the RL algorithms based on the history of the actions taken and observations done. The history length used to define this pseudo-state was chosen heuristically. Methods able to select a history length automatically from interaction with the system should be developed.

• *Curse of dimensionality*

The curse of dimensionality is the main limitation for a successful application of RL methods to large control problems. A preprocessing of the system state before using it as input of the RL algorithms as well as a proper choice of the approximation architecture can help circumvent, at least partially, this problem. In this respect the use of neural networks or regression trees as approximation architectures would be worth to investigate.

• *Influence of the policy on the solution obtained*

Suppose that the return obtained during the learning has no importance (i.e. the RL algorithms are used in an off-line mode). What kind of policy should be used during the learning ? When dealing with finite state space control problems one should certainly prefer a policy that favors the exploration since it will speed up the convergence of the RL algorithms to the optimal solution.

When dealing with infinite state space control problems, an approximation architecture is used to represent the learned solution. Generally speaking, if the RL algorithms converge, they will converge to something else than the optimal solution. One can wonder about the influence of the policy used during the learning on the solution to which they converge. Should we still in this case use a policy that favors the exploration to speed up the convergence or should we prefer a policy that uses almost at best the information already obtained about the control problem solution (i.e. a $\epsilon$-greedy policy with $\epsilon$ small) ? It can reasonably be supposed that with a policy that favors the exploitation the solution obtained would be better since the observations would be concentrated along the optimal trajectory of the system. Nevertheless, a firm conclusion would require a deeper study.

• *Multi-agent systems*

In this work we did not investigate systems in which many RL driven agents learn at the same time.

Among the many difficulties that arise with such systems, one is linked to the non-stationary environment in which each agent has to learn while the other agents are free to change their behavior as they learn and adapt themselves. Another difficulty arises from the “conflicts” appearing among the agents because each one of them tries to maximize its return at the expense of the others.

• *Off-line learning and system uncertainties*

Suppose the RL algorithms are used in an off-line mode and that the control law they compute is going to be implemented on a real system later on.

If the system dynamics and the reward function depend on a parameter vector $a$ whose value is not known, how should we realize the learning in order to have a control law that is “suitable” for all the possible values of $a$ ? Should we do the learning for only one specified value of $a$ and invoke the control law robustness to justify its validity for control problems having other values of $a$ ? Or should we use during the learning several types of episodes, each one of them corresponding to a different value of $a$ ? Answers to these questions are important especially if we want to use RL algorithms to control power systems whose operating conditions vary continuously.

# Appendix A

# Linear approximation functions

*This appendix is mainly dedicated to :*

- *the description of an iterative method to solve the Weighted Least Square Estimation (WLSE) problem*[1]
- *the description of a stochastic approximation algorithm used to minimize iteratively an integral of a continuum of cost functions*
- *derivation of some proofs of propositions used in chapters 3, 5 and 6.*

*For a deeper insight into this chapter content the reader may refer to [DH01], [BT96] and [May79].*

## A.1 Weighted Least Square Estimation (WLSE)

### A.1.1 Problem statement

Let $y_1, y_2, \cdots, y_A \in \mathbb{R}^n$, $b \in \mathbb{R}^A$ and $n, A \in \mathbb{N}$. Usually $A \gg n$. We want to determine the value of $a \in \mathbb{R}^n$ that minimizes the expression

$$\sum_{i=1}^{A} \beta^{A-i}[a^T y_i - b(i)]^2 \tag{A.1}$$

where $0 < \beta \leq 1$. If $\beta = 1$ the problem is known as the LSE problem.

[1]This iterative method is inspired from the Kalman filtering algorithm developed in the context of dynamic state estimation of stochastic systems with linear dynamics [Kal60].

If $Y_\beta$ is the $A \times n$ matrix whose $i$th line is equal to $(\beta^{\frac{A-i}{2}} y_i)^T$ and $b_\beta \in \mathbb{R}^A$ with $b_\beta(i) = \beta^{\frac{A-i}{2}} b(i)$ then expression (A.1) can be rewritten as follows[2] :

$$||Y_\beta a - b_\beta||^2 \quad . \tag{A.2}$$

It can be shown that the value of $a$ that minimizes (A.1) always satisfies the equation :

$$Y_\beta^T Y_\beta a = Y_\beta^T b_\beta \quad . \tag{A.3}$$

If $Y_\beta^T Y_\beta$ is invertible, the value of $a$ that minimizes expression (A.1) is unique.
Let us treat a particular WLSE problem. Suppose that $n = 1$, $y_i(1) = 1 \, \forall i \in \{1, \cdots, A\}$ and $\beta = 1$. By using equation (A.3) we can observe that the value of $a$ that minimizes (A.1) satisfies :

$$a(1) = \frac{1}{A} \sum_{i=1}^{A} b(i). \tag{A.4}$$

### A.1.2 Resolution of the WLSE problem : pseudo-inverse

If $Y_\beta^T Y_\beta$ is invertible, then the value of $a \in \mathbb{R}^n$ that minimizes (A.1) satisfies the equation :

$$a = (Y_\beta^T Y_\beta)^{-1} Y_\beta^T b_\beta \quad . \tag{A.5}$$

$(Y_\beta^T Y_\beta)^{-1} Y_\beta^T$ is called the pseudo-inverse of $Y_\beta$.

### A.1.3 Kalman filter like algorithm

We describe hereafter an iterative algorithm aimed to compute the value of $a$ that minimizes (A.1) in $A$ iterations. This algorithm being inspired from the Kalman filter, it is referred to in this work as the “Kalman filter like algorithm”.
This algorithm proceeds as follows. It initializes an $n \times n$ matrix $H$ to $\delta I$ ($I$ is the $n \times n$ identity matrix and $\delta > 0$) and $a$ arbitrarily.
At stage $k$ ($0 < k \leq A$) of the iterative process, the algorithm updates successively $H$ and $a$ as follows :

$$\begin{aligned} H &\leftarrow \beta H + y_k y_k^T && \text{(A.6)} \\ a &\leftarrow a + H^{-1}(b(k) - y_k^T a) y_k \quad . && \text{(A.7)} \end{aligned}$$

[2] $||Y_\beta a - b_\beta||^2 = (Y_\beta a - b_\beta)^T (Y_\beta a - b_\beta)$

The matrix $H$ being initialized to a positive definite matrix ($\delta I$ with $\delta > 0$), it can be shown that it stays positive definite during the iterative process. So its invertibility is ensured.
If $\delta \to 0$ then the value of $a$ obtained after the $k$th iteration of the recursive algorithm converges to a value of $a$ that minimizes the expression :

$$\sum_{i=1}^{k} \beta^{k-i}[a^T y_i - b(i)]^2 \quad . \tag{A.8}$$

Usually rather than initializing $H$ to $\delta I$, one determines a number $l$ such that the matrix

$$\sum_{i=1}^{l} y_i y_i^T \tag{A.9}$$

is positive definite, computes

$$\arg\min_{a\in\mathbb{R}^n} \sum_{i=1}^{l} \beta^{l-i}[a^T y_i - b(i)]^2 \tag{A.10}$$

by using the pseudo-inverse method and uses the algorithm from stage $l+1$ by considering that the value of $H$ and $a$ obtained after stage $l$ are respectively equal to (A.9) and (A.10). By proceeding so we can guarantee that the value of $a$ obtained after $k$ stages ($l < k \leq A$) minimizes (A.8).
Let us treat a particular case of the algorithm. Suppose that $n = 1$ and that $y_i(1) = 1\,\forall i \in \{1, \cdots, A\}$. In such a context, the value of $H$ obtained after the $k$th stage of the algorithm is equal to $\sum_{i=1}^{k} \beta^{k-i}$. Hence the $a$ update at stage $k$ can be rewritten as follows :

$$a \leftarrow a + \frac{1}{\sum_{i=1}^{k} \beta^{k-i}}(b(k) - a) \quad . \tag{A.11}$$

### A.1.4 Infinite set of values $(\mathbf{A} \to \infty)$

Let $\theta$ be a random variable with a probability distribution $P_\theta(.)$. Let $y_\theta$ be a function of the random variable $\theta$ that takes its values on $\mathbb{R}^n$ and $z_\theta$ a function of the random variable $\theta$ that takes its values on $\mathbb{R}$. Suppose that $y_\theta$ and $z_\theta$ are square integrable, i.e. that $\underset{\theta}{E}[(y_\theta(i))^2] < \infty, \forall i = 1, \ldots, n$ and $\underset{\theta}{E}[(z_\theta)^2] < \infty$.
Then, $\forall a \in \mathbb{R}^n$ the quantity

$$R(a) = \underset{\theta}{E}[(a^T y_\theta - z_\theta)^2] \quad , \tag{A.12}$$

is well defined and bounded. We will call this quantity the *true risk* of vector $a$. Suppose now that $\theta_1,\ \theta_2,\ \cdots,\ \theta_A$ are $A$ values of the random variable $\theta$ that are drawn independently according to $P_\theta(.)$. Let $y_{\theta=\theta_i}$ and $z_{\theta=\theta_i}$ be respectively the value of $y_\theta$ and $z_\theta$ when $\theta = \theta_i$. Denoting by $y_i = y_{\theta=\theta_i}$ and $b(i) = z_{\theta=\theta_i}$, let us define by

$$R\text{emp}(a) = \sum_{i=1}^{A}[a^T y_i - b(i)]^2 \tag{A.13}$$

the so-called *empirical* risk. Let $a_A$ denote a (not necessarily unique) vector which realizes the minimum in the right hand side of this equation, i.e.

$$a_A = \arg\min_a R\text{emp}(a). \tag{A.14}$$

Then, one can show that the following two sequences converge (in probability) to the same limit as $A \to \infty$ :

$$R(a_A) \overset{P}{\to} \min_{a\in\mathbb{R}^n} R(a), \tag{A.15}$$

$$R\text{emp}(a_A) \overset{P}{\to} \min_{a\in\mathbb{R}^n} R(a). \tag{A.16}$$

We paraphrase this statement by saying that *when $A \to \infty$ the value of $a$ that minimizes the empirical risk (equation (A.1) with $\beta = 1$) also minimizes the true risk (equation (A.12)).*
The preceding result is a particular case of the consistency property of the so-called empirical risk minimization principle studied in statistical learning theory. The latter theory generalizes these ideas to more complex (e.g. non-linear) model spaces and studies also error-bounds in the finite sample case.

## A.2 Minimization of an integral of a continuum of cost functions

### A.2.1 Problem statement

Under the assumptions (notably of square integrability) introduced in the preceding section, we want to compute a vector $a \in \mathbb{R}^n$ that minimizes :

$$\underset{\theta}{E}[(a^T y_\theta - z_\theta)^2]. \tag{A.17}$$

We notice that the values of $a$ that minimize expression (A.17) also satisfy :

$$\underset{\theta}{E}[y_\theta(i)(a^T y_\theta - z_\theta)] = 0 \quad \forall i \in \{1, \cdots, n\} \quad . \tag{A.18}$$

We define an $n \times n$ matrix $F$ as follows :

$$F_{ij} = \underset{\theta}{E}[y_\theta(i) y_\theta(j)] \quad \forall i, j \in \{1, \cdots, n\} \quad . \tag{A.19}$$

If $F$ is invertible then the value of $a$ that solves equation (A.18) is unique.
Let us treat a particular case of this minimization problem. Suppose that $n = 1$ and $y_\theta(1) = 1$. By using equation (A.18) we can see that the value of $a$ that minimizes expression (A.17) satisfies :

$$a(1) = \underset{\theta}{E}[z_\theta] \quad . \tag{A.20}$$

### A.2.2 Iterative algorithm

Hereafter we describe an iterative algorithm to solve the minimization problem stated in section A.2.1. This algorithm belongs to the class of stochastic approximation algorithms.
It computes iteratively the value of $a$ that minimizes (A.17). First it initializes $a$ arbitrarily. At stage $k$ ($0 < k < \infty$) of the iterative process it draws a value $\theta_k$ of the random variable $\theta$ independently according to the probability distribution $P_\theta(.)$ and updates $a$ as follows :

$$a \leftarrow a + \alpha_k (z_{\theta=\theta_k} - a^T y_{\theta=\theta_k}) y_{\theta=\theta_k} \tag{A.21}$$

where $\alpha_k > 0$ and where $z_{\theta=\theta_k}$ and $y_{\theta=\theta_k}$ denote respectively the value of $z_\theta$ and $y_\theta$ when $\theta = \theta_k$.
If $\alpha_k$ satisfies :

$$\sum_{k=1}^{\infty} \alpha_k \to \infty \tag{A.22}$$

$$\sum_{k=1}^{\infty} (\alpha_k)^2 < \infty \tag{A.23}$$

then it can be shown that the iterative process converges with a probability 1 to a value of $a$ that minimizes expression (A.17) (see e.g. [BT96]).

### A.2.3 LSE and stochastic approximation algorithm

Let $y_1, y_2, \cdots, y_A \in \mathbb{R}^n$, $b \in \mathbb{R}^A$. Suppose that the random variable $\theta$ takes its values in the set $\{1, 2, \cdots, A\}$ such that $P_\theta(i) = \frac{1}{A} \forall i \in \{1, 2, \cdots, A\}$. Suppose that $y_{\theta=i} = y_i$ and $z_{\theta=i} = b(i)$. In such conditions, the value of $a$ that minimizes (A.17) also minimizes equation (A.1) in which $\beta = 1$.
Indeed, expression (A.17) can be rewritten as follows :

$$\frac{1}{A}\sum_{i=1}^{A}[a^T y_i - b(i)]^2 \tag{A.24}$$

which proves the proposition.

## A.3 Proofs and additional explanations

### A.3.1 Estimation of the transition probabilities of the MDP$^\delta$

The proposition hereafter is used in sections 3.3.4 and 6.6 to characterize the transition probabilities of the MDP$^\delta$ structure computed.

#### Proposition

Let $\beta \in ]0, 1]$ and $\beta_A \in \mathbb{R}^A$ be such that $\beta_A(i) = \beta^{\frac{A-i}{2}}$ $\forall i \in \{1, \cdots, A\}$ and $u \in \mathbb{R}^n$ such that $u(i) = 1$ $\forall i \in \{1, \cdots, n\}$.
If

- $Y$ is an $A \times n$ real-valued matrix ($n \leq A$) such that $Y^T Y$ is invertible, and such that $\sum_{j=1}^{n} Y_{ij} = \beta^{\frac{A-i}{2}}, \forall i \in \{1, \cdots, A\}$,
- $b_i \in \mathbb{R}^A, \forall i = 1, \ldots, n$ denote $n$-vectors such that $\sum_{i=1}^{n} b_i = \beta_A$,
- $a_i \in \mathbb{R}^n, \forall i = 1, \ldots, n$ such that $a_i$ is the value of $a \in \mathbb{R}^n$ that minimizes $\|Ya - b_i\|$,

then $\sum_{i=1}^{n} a_i = u$.

#### Proof

We can write :

$$Y^T Y a_p = Y^T b_p \quad \forall p \in \{1, \cdots, n\} \quad . \tag{A.25}$$

This implies :

$$\sum_{p=1}^{n} Y^T Y a_p = \sum_{p=1}^{n} Y^T b_p \quad . \tag{A.26}$$

Knowing that $\sum_{p=1}^{n} b_p = \beta_A$, we have :

$$\sum_{p=1}^{n} Y^T Y a_p = Y^T \beta_A \quad . \tag{A.27}$$

We can write :

$$(Y^T \beta_A)(i) = \sum_{k=1}^{A} Y_{ki} \beta^{\frac{A-k}{2}} \tag{A.28}$$

and :

$$\begin{aligned}(Y^T Y u)(i) &= \sum_{j=1}^{n} \sum_{k=1}^{A} Y_{ki} Y_{kj} \\ &= \sum_{k=1}^{A} Y_{ki} \sum_{j=1}^{n} Y_{kj} \\ &= \sum_{k=1}^{A} Y_{ki} \beta^{\frac{A-k}{2}} \quad .\end{aligned} \tag{A.29}$$

From equations (A.28) and (A.29) we have :

$$Y^T Y u = Y^T \beta_A \quad . \tag{A.30}$$

From equations (A.27) and (A.30) and since $Y^T Y$ is invertible we have $\sum_{p=1}^{n} a_p = u$. **QED**

### A.3.2 MDP structure estimation by solving WLSE problems

In this subsection we provide a deeper insight into the way the MDP structure is estimated by the algorithm described on figure 5.4 and entitled "*Estimation of the MDP structure* : Kalman Filter like algorithm".

**Estimation of $r(x,u)$**

Let $N(x,u)$ be the number of times the particular state-action pair $(x,u)$ has been visited during the learning. Let $w(i)$ be the value of the random variable $w$ the $i$th time the state-action pair $(x,u)$ has been visited. Each $w(i)$ is drawn independently according to $P_w(.|x,u)$.
The value of $r(x,u)$ estimated by the algorithm described on figure 5.4 minimizes :

$$\sum_{i=1}^{N(x,u)} \beta^{N(x,u)-i}(r(x,u) - r(x,u,w(i)))^2 \quad . \tag{A.31}$$

The minimization is done iteratively by using the algorithm described by equations (A.6) and (A.7).
In order to highlight that, let us rewrite the minimization problem by using the notations of section A.1.
We set :

- $n = 1$
- $A = N(x,u)$
- $y_i(1) = 1 \; \forall i \in \{1, 2, \cdots, A\}$
- $a(1) = r(x,u)$
- $b(i) = r(x,u,w(i)) \; \forall i \in \{1, 2, \cdots, A\}$.

The minimization problem becomes equivalent to find the value of $a$ that minimizes expression (A.1). We can observe that the minimization problem is indeed solved in figure 5.4 by using the Kalman filter like algorithm described by equations (A.6) and (A.7).
If we set :

- $\theta = w$
- $P_\theta(\theta) = P_w(w|x,u)$
- $y_\theta(1) = 1$
- $z_\theta = r(x,u,w)$

and use the results of section A.1.4, we see that if $\beta = 1$ and $A \to \infty$ then $a(1) \to \underset{\theta}{E}[z_\theta]$ or equivalently $r(x,u) \to \underset{w}{E}[r(x,u,w)]$.

**Estimation of** $p(x'|x,u)$

Let $N(x,u)$ be the number of times the particular state-action pair $(x,u)$ has been visited. Let $w(i)$ be the value of the random variable $w$ the $i$th time the state-action pair $(x,u)$ has been visited. Each $w(i)$ is drawn independently according to $P_w(.|x,u)$.
The value of $p(x'|x,u)$ estimated by the algorithm described on figure 5.4 minimizes :

$$\sum_{i=1}^{N(x,u)} \beta^{N(x,u)-i}(p(x'|x,u) - I_{\{x'\}}(f(x,u,w(i))))^2 \quad . \tag{A.32}$$

The minimization is done iteratively by using the algorithm described by equations (A.6) and (A.7).
In order to highlight that, let us rewrite the minimization problem by using the notations of section A.1.
We set :

- $n = 1$
- $A = N(x,u)$
- $y_i(1) = 1 \; \forall i \in \{1, 2, \cdots, A\}$
- $a(1) = p(x'|x,u)$
- $b(i) = I_{\{x'\}}(f(x,u,w(i))) \; \forall i \in \{1, 2, \cdots, A\}$.

The minimization problem becomes equivalent to find the value of $a$ that minimizes expression (A.1). We can observe that the minimization problem is indeed solved in figure 5.4 by using the Kalman filter like algorithm described by equations (A.6) and (A.7).
If we set :

- $\theta = w$
- $P_\theta(\theta) = P_w(w|x,u)$
- $y_\theta(1) = 1$
- $z_\theta = I_{\{x'\}}(f(x,u,w))$

and use the results of section A.1.4, we see that if $\beta = 1$ and $A \to \infty$ then $a(1) \to \underset{\theta}{E}[z_\theta]$ or equivalently $p(x'|x,u) \to \underset{w}{E}[I_{\{x'\}}(f(x,u,w))]$.

### A.3.3 MDP structure estimation by minimizing an integral of a continuum of cost functions

In this subsection we provide a deeper insight on the way the MDP structure is estimated by the algorithm described on figure 5.5 and entitled "*Estimation of the MDP structure* : Stochastic Approximation algorithm".

#### Estimation of $r(x, u)$

The algorithm described on figure 5.5 aims to find the value of $r(x, u)$ that minimizes

$$\underset{w}{E}[(r(x,u) - r(x,u,w))^2] \quad . \tag{A.33}$$

The minimization is done iteratively by using the algorithm described in section A.2.
In order to highlight that, let us rewrite the minimization problem by using the notations of section A.2.
We set :

- $n = 1$
- $a(1) = r(x, u)$
- $\theta = w$
- $P_\theta(\theta) = P_w(w|x, u)$
- $y_\theta(1) = 1$
- $z_\theta = r(x, u, w)$

The minimization problem becomes equivalent to find the value of $a$ that minimizes expression (A.17). We can observe that the minimization problem is indeed solved in figure 5.5 by using the iterative algorithm described by equation (A.21). It implies $a(1) \to \underset{\theta}{E}[z_\theta]$ or equivalently $r(x, u) \to \underset{w}{E}[r(x, u, w)]$.

#### Estimation of $p(x'|x, u)$

The algorithm described on figure 5.5 aims to find the value of $p(x'|x, u)$ that minimizes

$$\underset{w}{E}[(p(x'|x,u) - I_{\{x'\}}(f(x,u,w)))^2] \quad . \tag{A.34}$$

The minimization is done iteratively by using the algorithm described in section A.2.
In order to highlight that, let us rewrite the minimization problem by using the notations of section A.2.
We set :

- $n = 1$
- $a(1) = p(x'|x, u)$
- $\theta = w$
- $P_\theta(\theta) = P_w(w|x, u)$
- $y_\theta(1) = 1$
- $z_\theta = I_{\{x'\}}(f(x, u, w))$

The minimization problem becomes equivalent to find the value of $a$ that minimizes expression (A.17). We can observe that the minimization problem is indeed solved in figure 5.5 by using the iterative algorithm described by equation (A.21). It implies $a(1) \rightarrow \underset{\theta}{E}[z_\theta]$ or equivalently $p(x'|x, u) \rightarrow \underset{w}{E}[I_{\{x'\}}(f(x, u, w))]$.

# Appendix B

# Triangulation of an $n$-cube

*This appendix is dedicated to the triangulation of an $n$-cube (or equivalently of an $n$-rectangle by noting that a linear change of coordinates can transform the $n$-rectangle into an $n$-cube). We will*

- *describe an easily implementable method that triangulates the $n$-cube into $n!$ simplices*
- *discuss the triangulation of the $n$-cube into a minimum number of simplices*
- *illustrate the influence the triangulation quality can have on the approximate solution of an infinite state-space optimal control problem computed by using the representative states technique (see section 3.3).*

## B.1 Definitions

Let $n$ be a positive integer. An *$n$-simplex* is the convex hull of $n+1$ points in an $n$-dimensional Euclidean space ($\mathbb{R}^n$). For example, a 1-simplex is a line segment, a 2-simplex is a triangle (with its interior), and a 3-simplex is a tetrahedron (with its interior).
An $n$-cube is *triangulated* if it is partitioned into a finite number of $n$-simplices with disjoint interiors, subject to the constraint that the vertices of any $n$-simplex are also the vertices of the $n$-cube.
The *simplexity* $s(n)$ of the $n$-cube is the minimum number of $n$-simplices required to triangulate it.
One basic measure of the *triangulation quality* is usually the number of $n$-simplexes that are needed to partition an $n$-cube. The higher this number, the lower the quality of the triangulation [Sal82].

There exists a wide variety of methods that allow the triangulation of an $n$-cube [Sal84, Tod84]. Usually, the better the triangulation quality, the higher the computational burden.

## B.2 Triangulation of the $n$-cube into $n!$ simplices

The triangulation method used in this thesis consists in partitioning the $n$-cube into $n!$ simplices.
This method proceeds as follows. Suppose that the $n$-cube is the set of points $\{x \in \mathbb{R}^n | 0 \leq x(i) \leq 1 \, \forall i \in \{1, \cdots, n\}\}$. Each simplex that partitions the $n$-cube corresponds to a permutation $(k_1, \cdots, k_n)$ of $(1, \cdots, n)$ and is defined as follows ([Moo92]) :

$$\{x \in \mathbb{R}^n | 0 \leq x(k_n) \leq x(k_{n-1}) \leq \cdots \leq x(k_1) \leq 1\} \quad . \tag{B.1}$$

The $n!$ possible permutations of $(1, \cdots, n)$ define the $n!$ simplices that triangulate the $n$-cube.
When this method is used to triangulate a 3-cube, it yields the $3! = 6$ simplices represented on figure B.1a.
With such a triangulation method, it is easy to identify the $n+1$ vertices $v_1$, $v_2$, $\cdots$, $v_{n+1}$ of the simplex to which a point $x \in n$-cube belongs. Indeed if the permutation $(k_1, \cdots, k_n)$ of $(1, \cdots, n)$ is such that $0 \leq x(k_n) \leq x(k_{n-1}) \leq \cdots \leq x(k_1) \leq 1$, then the terms $v_i(k_j)$ can be defined as follows :

$$v_i(k_j) = \begin{cases} 1 & \text{if } i + j \leq n + 1 \\ 0 & \text{otherwise} \end{cases} \quad \forall i \in \{1, \cdots, n+1\} \text{ and } j \in \{1, \cdots, n\} \tag{B.2}$$

## B.3 Triangulation of an $n$-cube into $s(n)$ simplices

It is obvious that $s(1) = 1$ and $s(2) = 2$. Unfortunately, the determination of $s(n)$ and the triangulation of the $n$-cube are problems that require a computing time which is exponential in $n$. So, it is already more difficult to prove that $s(3) = 5$ and to find the corresponding triangulation which is represented on figure B.1b.
Only an enormous amount of computation can lead to the determination of the values of $s(n)$ given in table B.1 ([Smi88]) and to our knowledge, if $n > 7$ then no value of $s(n)$ has ever been computed.

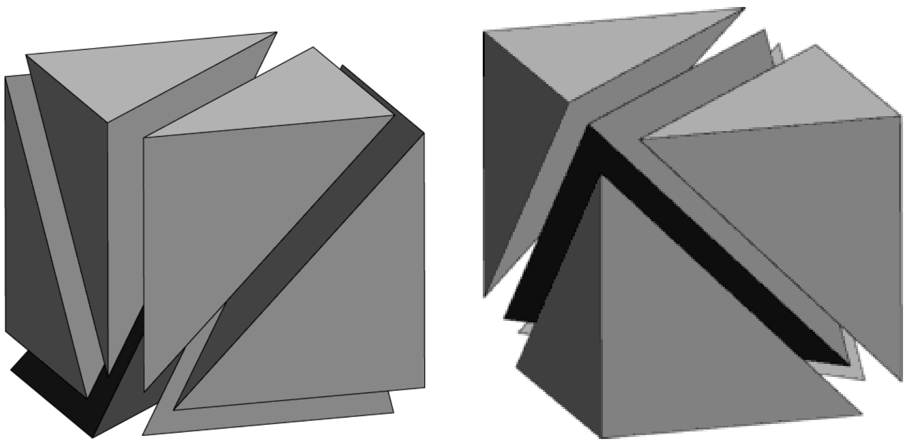

(a) $n!$ simplices (b) Minimum number of simplices

Figure B.1: Cube triangulation. Taken from [Epp].

| $n$ | $s(n)$ | $n!$ |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 5 | 6 |
| 4 | 16 | 24 |
| 5 | 67 | 120 |
| 6 | 308 | 720 |
| 7 | 1493 | 5040 |

Table B.1: Comparison between the minimum number of $n$-simplices required to triangulate the $n$-cube and $n!$

## B.4 Triangulation quality and control law quality

In section 4.6 we have treated a control problem whose resolution required to triangulate 3-rectangles into tetrahedra. Each 3-rectangle was partitioned into $3! = 6$ tetrahedra by using the method described in section B.2. If we partition the 3-rectangle into $s(3) = 5$ tetrahedra by using the triangulation scheme described on figure B.1b, we obtain the results gathered in table B.2. The second column of this table corresponds to the third column of table 4.5.

| score | 6 tetrahedra | 5 tetrahedra |
|---|---|---|
| $X \setminus \{x^t\}$ | -38.6561 | -38.3735 |
| $X'$ | -6.4278 | -6.2277 |
| $X''$ | -2.3210 | -2.1843 |

Table B.2: Score obtained for different triangulations. The OMIB power system with AVR. $\Delta E = 0.2$

We remark that the scores obtained by the control law are better when the triangulation quality is higher. One plausible physical interpretation of this phenomenon is that when the cube is partitioned into 5 simplices the distance of a point of the cube to a vertice of the simplex it belongs to is at maximum $\sqrt{2}$ while it can be equal to $\sqrt{(1+\sqrt{2})}$ when the cube is partitioned into 6 simplices (see figure B.1a and B.1b). Therefore, when the cube is partitioned into 5 simplices, a state inherits from the characteristics of states that are closer and thus more likely to have similar properties.
This example illustrates the influence the triangulation quality can have on the policy obtained. Nevertheless, to be able to draw some firm conclusions from such observations, one should study this influence on a larger number of control problems.

# Appendix C

# Algorithms built under contraction assumptions

*In the first section we introduce the notion of contraction mapping and describe its properties. The second section is dedicated to the description of algorithmic models that compute the fixed point of a contraction mapping. In the other sections, we use the properties of these algorithmic models to prove :*

- *the convergence of dynamic programming algorithms (value iteration, asynchronous value iteration, $\cdots$ )*
- *the convergence of the $Q$-learning algorithm*
- *the convergence of the $Q$-learning algorithm used with the aggregation technique under particular learning conditions or under conditions of equivalence with the initial control problem*
- *the convergence of the $Q$-learning algorithm used with the representative states technique under particular learning conditions.*

## C.1 Contraction mapping

Let $B(E)$ be the set of all bounded real-valued functions defined on an arbitrary set $E$. With every function $R : E \to \mathbb{R}$ that belongs to $B(E)$, we associate the scalar :

$$\|R\|_\infty = \sup_{e \in E} |R(e)|. \tag{C.1}$$

A mapping $G : B(E) \to B(E)$ is said to be a *contraction mapping* if there exists a scalar $\rho < 1$ such that :

$$\|GR - GR'\|_\infty \leq \rho \|R - R'\|_\infty \quad \forall R, R' \in B(E). \tag{C.2}$$

$R^* \in B(E)$ is said to be a *fixed point* of a mapping $G : B(E) \to B(E)$ if :

$$GR^* = R^*. \tag{C.3}$$

If $G : B(E) \to B(E)$ is a *contraction mapping* then there exists a unique fixed point of $G$. Furthermore if $R \in B(E)$, then

$$\lim_{k \to \infty} \|G^k R - R^*\|_\infty = 0. \tag{C.4}$$

For a proof, see references [LS61] and [Lue69].

## C.2 Description of algorithmic models

In this section we suppose that :

- $E$ is finite and composed of $n$ elements
- $G : B(E) \to B(E)$ is a contraction mapping whose fixed point is denoted by $R^*$
- $R \in B(E)$.

### C.2.1 All elements of $R$ are refreshed

Suppose we have the algorithm that updates at stage $k$ ($k \geq 0$) $R$ as follows :

$$R \leftarrow GR. \tag{C.5}$$

The value of $R$ computed by this algorithm converges to the fixed point $R^*$ of $G$. This is an immediate consequence of equation (C.4).

### C.2.2 One element of $R$ is refreshed

Suppose we have the algorithm that selects at each stage $k$ ($k \geq 0$) an element $e \in E$ and updates $R(e)$ as follows :

$$R(e) \leftarrow (GR)(e) \tag{C.6}$$

leaving the other components of $R$ unchanged. If each element $e$ of $E$ is selected an infinite number of times then the value of $R$ computed by this algorithm converges to the fixed point $R^*$.
We now provide the convergence proof of this algorithm (taken from [BT96]). We denote the value of $R$ at stage $k$ by $R_k$.
Without loss of generality, let us assume that $R^* = 0$ (this is no loss of generality). Suppose that we start with $R_0$ and that $|R_0(e)| \leq c \, \forall e \in E$ for some constant $c$ (we can always take $c = \|R_0\|_\infty$). In other words, $R_0$ lies within a "cube" of size $2c$ along each dimension, centered at the origin. Suppose that $R_k$ lies within that cube and that at stage $k$ of the algorithm, the element $e$ is selected in the algorithm which conducts to the following $R_{k+1} : R_{k+1}(e) = (GR_k)(e)$ and $R_{k+1}(e') = R_k(e')$ $\forall e' \in E \setminus \{e\}$. Due to the contraction condition, we have $|R_{k+1}(e)| \leq \rho c < c$, which shows that given the initial cube, any subsequent update of any element keeps us within that cube. Furthermore, any component of $R$ that corresponds to an element of $\{e_0, e_1, \cdots, e_k\}$ is at the very most $\rho c$ in magnitude. It means that when all the components have been updated at least once, we find ourselves within a smaller cube, of $2\rho c$ along each direction. By continuing similarly, and assuming that each component is updated an infinite number of times, we get progressively into smaller cubes; convergence to 0 follows.

### C.2.3 One element of $R$ is refreshed and noise introduction

Let $\eta \in \mathbb{R}$ be a noise factor and $\alpha \in \mathbb{R}$. Suppose we have the algorithm that selects at stage $k$ ($k \geq 0$) an element $e \in E$ and updates $R(e)$ according to :

$$R(e) \leftarrow (1-\alpha)R(e) + \alpha((GR)(e) + \eta) \tag{C.7}$$

leaving the other components of $R$ unchanged.
We denote by $e_k$ the element of $E$ selected at stage $k$, by $\eta_k$ the noise value at stage $k$ and by $R_k$ the value of $R$ at stage $k$ and by $\alpha_k$ the value of $\alpha$ at stage $k$. In order to ease further notation we set $\alpha_k(e) = \alpha_k$ if $e = e_k$ and $\alpha_k(e) = 0$ otherwise.
With this notation equation (C.7) can be rewritten equivalently as follows :

$$R_{k+1}(e_k) = (1-\alpha_k)R_k(e_k) + \alpha_k((GR_k)(e_k) + \eta_k). \tag{C.8}$$

We define the history $\mathcal{F}_k$ of the algorithm at stage $k$ as being :

$$\mathcal{F}_k = \{R_0, \cdots, R_k, e_0, \cdots, e_k, \alpha_0, \cdots, \alpha_k, \eta_0, \cdots, \eta_{k-1}\}. \tag{C.9}$$

We assume moreover that the following conditions are satisfied :

1. For every $k$, we have
$$E[\eta_k|\mathcal{F}_k] = 0. \tag{C.10}$$
2. There exist two constants $A$ and $B$ such that $\forall k$
$$E[\eta_k^2|\mathcal{F}_k] \leq A + B||R_k||_\infty^2. \tag{C.11}$$
3. The $\alpha_k(e)$ are nonnegative and satisfy
$$\sum_{k=0}^{\infty} \alpha_k(e) = \infty, \quad \sum_{k=0}^{\infty} \alpha_k^2(e) < \infty. \tag{C.12}$$

Then the algorithm converges with probability 1 to $R^*$.
A rigorous proof can be found in [Tsi94] while an informal one is given in [JJS94]. It follows roughly the same lines as the one given in previous subsection except that in presence of a small stepsize $\alpha$, it will take several updates before we get into the next smaller cube; in addition, we need to verify that the presence of the noise term $\eta$ cannot have an adverse effect on the convergence.

## C.3 $T$ is a contraction mapping : consequences

We recall the $T$ mapping introduced in section 2.2. $T : B(X) \rightarrow B(X)$ :
$$(TJ)(x) = \max_{u \in U(x)} \underset{w}{E}[r(x,u,w) + \gamma J(f(x,u,w))] \quad \forall x \in X \tag{C.13}$$
with $J \in B(X)$ and suppose that $X$ and $C$ are finite.
The $T$ mapping is a contraction mapping. Indeed we have for any functions $J, \overline{J} \in B(X)$ :
$$\begin{aligned}
||TJ - T\overline{J}||_\infty &= \max_{x \in X}| \max_{u \in U(x)} \underset{w}{E}[r(x,u,w) + \gamma J(f(x,u,w))] \\
&\quad - \max_{u \in U(x)} \underset{w}{E}[r(x,u,w) + \gamma \overline{J}(f(x,u,w))]| \\
&\leq \gamma \max_{x \in X}| \max_{u \in U(x)} \underset{w}{E}[|J(f(x,u,w)) - \overline{J}(f(x,u,w))|]| \\
&\leq \gamma \max_{x \in X}| \underset{w}{E}[\max_{x' \in X}|J(x') - \overline{J}(x')|]| \\
&= \gamma \max_{x \in X}|J(x) - \overline{J}(x)| \\
&= \gamma||J - \overline{J}||_\infty.
\end{aligned}$$

The $T$ mapping being a contraction mapping, it has a unique fixed point. This unique fixed point is the optimal control problem value function.
$T$ being a contraction mapping,

- the convergence of the algorithm described in section C.2.1 implies the convergence of the value iteration algorithm (figure 2.2)
- the convergence of the algorithm described in section C.2.2 implies the convergence of the Gauss-Seidel version of the value iteration algorithm.

## C.4 Introduction of the $H$ mapping and consequences

We define the $H$ mapping. $H : B(X \times U) \to B(X \times U)$ such that

$$(HK)(x,u) = \underset{w}{E}[r(x,u,w) + \gamma \max_{u' \in U(f(x,u,w))} K(f(x,u,w),u')] \; \forall (x,u) \in X \times U \quad \text{(C.14)}$$

with $K \in B(X \times U)$ and suppose that the set $X \times U$ is finite.
This $H$ mapping is a contraction mapping. Indeed, we have for any functions $K, \overline{K} \in B(X \times U)$ :

$$\begin{aligned}
\|HK - H\overline{K}\|_\infty &= \gamma \max_{(x,u) \in X \times U} |\underset{w}{E}[\max_{u' \in U(f(x,u,w))} K(f(x,u,w),u') - \\
&\quad \max_{u' \in U(f(x,u,w))} \overline{K}(f(x,u,w),u')]| \\
&\leq \gamma \max_{(x,u) \in X \times U} |\underset{w}{E}[\max_{u' \in U(f(x,u,w))} |K(f(x,u,w),u') - \\
&\quad \overline{K}(f(x,u,w),u')|]| \\
&\leq \gamma \max_{x \in X} \max_{u \in U(x)} |K(x,u) - \overline{K}(x,u)| \\
&= \gamma \|K - \overline{K}\|_\infty
\end{aligned}$$

The $H$ mapping being a contraction mapping, it has a unique fixed point. This unique fixed point is the optimal control problem $Q$-function.
$H$ being a contraction mapping,

- the convergence of the algorithm described in section C.2.1 implies the convergence of algorithm sketched on figure 2.3 and entitled *The Value Iteration algorithm : $Q$-function computation*
- the convergence of the algorithm described in section C.2.2 implies the convergence of the algorithm sketched on figure 2.4 and entitled *The Gauss-Seidel Value Iteration algorithm : $Q$-function computation.*

## C.5 Q-learning convergence proof

The $Q$-learning algorithm (figure 5.11) updates $Q$ at stage $k$ in the following way[1] :

$$\begin{aligned} Q_{k+1}(x_k,u_k) &= (1-\alpha_k)Q_k(x_k,u_k) + \alpha_k(r(x_k,u_k,w_k) \\ &\quad +\gamma \max_{u\in U(f(x_k,u_k,w_k))} Q_k(f(x_k,u_k,w_k),u_k)), \end{aligned} \tag{C.15}$$

$Q_k$ representing the estimate of the $Q$-function at stage $k$. $w_k$ is drawn independently according to $P_w(.|x_k,u_k)$.
By using the $H$ mapping definition (equation (C.14)), equation (C.15) can be rewritten as follows :

$$Q_{k+1}(x_k,u_k) = (1-\alpha_k)Q_k(x_k,u_k) + \alpha_k((HQ_k)(x_k,u_k)+\eta_k) \tag{C.16}$$

with

$$\begin{aligned} \eta_k &= r(x_k,u_k,w_k) + \gamma \max_{u\in U(f(x_k,u_k,w_k))} Q_k(f(x_k,u_k,w_k),u) - (HQ_k)(x_k,u_k) \\ &= r(x_k,u_k,w_k) + \gamma \max_{u\in U(f(x_k,u_k,w_k))} Q_k(f(x_k,u_k,w_k),u) - \\ &\quad \underset{w}{E}[r(x_k,u_k,w) + \gamma \max_{u\in U(f(x_k,u_k,w))} Q_k(f(x_k,u_k,w),u)] \end{aligned}$$

which has exactly the same form as equation (C.8)[2].
We know that $H$ is a contraction mapping. If the $\alpha_k(x_k,u_k)$ terms satisfy expression (C.12), we still have to verify that $\eta_k$ satisfies expressions (C.10) and (C.11), where

$$\mathcal{F}_k = \{Q_0,\cdots,Q_k,(x_0,u_0),\cdots,(x_k,u_k),\alpha_0,\cdots,\alpha_k,\eta_0,\cdots,\eta_{k-1}\}, \tag{C.17}$$

in order to ensure the convergence of the $Q$-learning algorithm.
We have :

$$\begin{aligned} E[\eta_k|\mathcal{F}_k] &= \underset{w_k}{E}[r(x_k,u_k,w_k) + \gamma \max_{u\in U(f(x_k,u_k,w_k))} Q_k(f(x_k,u_k,w_k),u) - \\ &\quad \underset{w}{E}[r(x_k,u_k,w) + \gamma \max_{u\in U(f(x_k,u_k,w))} Q_k(f(x_k,u_k,w),u)]|\mathcal{F}_k] \\ &= 0 \end{aligned}$$

[1]The four-tuple $(x_t,u_t,r_t,x_{t+1})$ used as input of the $Q$-learning algorithm described in figure 5.11 is "replaced" here by $(x_k,u_k,r(x_k,u_k,w_k),f(x_k,u_k,w_k))$.

[2]$Q_k$ corresponding to $R_k$, $H$ to $G$, $(x_k,u_k)$ to $e_k$ and $X\times U$ to $E$.

and expression (C.10) is indeed satisfied.
In order to prove that expression (C.11) is satisfied, one can first note that :

$$|\eta_k| \leq 2B_r + 2\gamma \max_{(x,u)\in X\times U} Q_k(x,u) \tag{C.18}$$

where $B_r$ is the bound on the rewards (section 2.1). Therefore we have :

$$\eta_k^2 \leq 4B_r^2 + 4\gamma^2(\max_{(x,u)\in X\times U} Q_k(x,u))^2 + 8B_r\gamma \max_{(x,u)\in X\times U} Q_k(x,u) \tag{C.19}$$

By noting that

$$8B_r\gamma \max_{(x,u)\in X\times U} Q_k(x,u) < 8B_r\gamma + 8B_r\gamma(\max_{(x,u)\in X\times U} Q_k(x,u))^2 \tag{C.20}$$

and by choosing $A = 8B_r\gamma + 4B_r^2$ and $B = 8B_r\gamma + 4\gamma^2$ we can write

$$\eta_k^2 \leq A + B\|Q_k\|_\infty^2 \tag{C.21}$$

and expression (C.11) is satisfied. **QED**

## C.6 Aggregation technique : convergence to the MDP$^\delta$ solution

### C.6.1 Convergence conditions

In next the subsection we prove that the algorithm described in figure 6.8 and entitled *Q-learning algorithm used with the aggregation technique* converges with probability 1 to the $Q^\delta$-function corresponding to the MDP$^\delta$ defined by equations (3.1) and (3.3) if :

- the $\alpha_k(x^\delta, u)$ are non negative and satisfy $\sum_{k=0}^{\infty} \alpha_k(x^\delta, u) = \infty$ and $\sum_{k=0}^{\infty} \alpha_k^2(x^\delta, u) < \infty$, $\forall (x^\delta, u) \in X^\delta \times U^\delta$
- at least one of these two conditions is met :
  - equations (3.9) and (3.10) are satisfied (conditions of equivalence), that is :

$$r^\delta(x_i^\delta, u) = \underset{w}{E}[r(x,u,w)] \tag{C.22}$$
$$p^\delta(x_j^\delta|x_i^\delta, u) = \underset{w}{E}[I_{X_j}(f(x,u,w))] \tag{C.23}$$

$\forall i, j \in \{1, \cdots, m\}, \forall u \in U^\delta(x_i^\delta)$, $\forall x \in X_i$.

– one-step episodes are used, with $P_0(x)$ given by :

$$P_0(x) = \begin{cases} c_1 p_1(x) & \text{if } x \in X_1 \\ c_2 p_2(x) & \text{if } x \in X_2 \\ \vdots & \\ c_m p_m(x) & \text{if } x \in X_m \end{cases} \tag{C.24}$$

with $c_i > 0$ and $\sum_{i=1}^{m} c_i = 1$, and the probability to select an action $u \in U^\delta(x_i^\delta)$ while being in any state of $X_i$ is the same (and non zero).

## C.6.2 Proof

We define the $H^\delta$ mapping. $H^\delta : B(X^\delta \times U^\delta) \to B(X^\delta \times U^\delta)$ such that

$$\begin{aligned}(H^\delta K)(x_i^\delta, u) &= r^\delta(x_i^\delta, u) + \\ &\gamma \sum_{j=1}^{m} p^\delta(x_j^\delta | x_i^\delta, u) \max_{u' \in U^\delta(x_j^\delta)} K(x_j^\delta, u') \quad \forall (x_i^\delta, u) \in X^\delta \times U^\delta\end{aligned} \tag{C.25}$$

with $K \in B(X^\delta \times U^\delta)$, $r^\delta(x_i^\delta, u)$ and $p^\delta(x_j^\delta | x_i^\delta, u)$ being defined respectively by equations (3.1) and (3.3).
By using the same procedure as in section C.4, it can be shown that $H^\delta$ is a *contraction mapping* and that the $Q^\delta$-function solution of the MDP$^\delta$ defined by equations (3.1) and (3.3) is the unique fixed point of this contraction mapping.
Hereafter we consider that the action taken while being in a state $x$ belongs to $U^\delta(x)$.
The algorithm described in figure 6.8 realizes at stage $k$ an update of the $Q^\delta$-function of this type[3] :

$$\begin{aligned}Q_{k+1}^\delta(x_{i_k}^\delta, u_k) &= (1-\alpha_k) Q_k^\delta(x_{i_k}^\delta, u_k) + \alpha_k (r(x_k, u_k, w_k) \\ &+\gamma \sum_{j=1}^{m} I_{X_j}(f(x_k, w_k, u_k)) \max_{u \in U^\delta(x_j^\delta)} Q_k^\delta(x_j^\delta, u))\end{aligned} \tag{C.26}$$

with $x_k \in X_{i_k}$. $Q_k^\delta$ represents the estimate of the $Q^\delta$-function at stage $k$ and $w_k$ is drawn independently according to $P_w(.|x_k, u_k)$.
By using the $H^\delta$ mapping definition (equation (C.25)), equation (C.26) can be rewritten as follows :

$$Q_{k+1}^\delta(x_{i_k}^\delta, u_k) = (1-\alpha_k) Q_k^\delta(x_{i_k}^\delta, u_k) + \alpha_k ((H^\delta Q_k^\delta)(x_{i_k}^\delta, u_k) + \eta_k) \tag{C.27}$$

[3]The four-tuple $(x_t, u_t, r_t, x_{t+1})$ used as input of the algorithm described in figure 6.8 is “replaced” here by $(x_k, u_k, r(x_k, u_k, w_k), f(x_k, u_k, w_k))$.

with

$$\begin{aligned}
\eta_k &= r(x_k,u_k,w_k)+\gamma\sum_{j=1}^{m} I_{X_j}(f(x_k,w_k,u_k))\max_{u\in U^\delta(x_j^\delta)} Q_k^\delta(x_j^\delta,u)\\
&\quad -(HQ_k^\delta)(x_{i_k}^\delta,u_k)\\
&= r(x_k,u_k,w_k)+\gamma\sum_{j=1}^{m} I_{X_j}(f(x_k,w_k,u_k))\max_{u\in U^\delta(x_j^\delta)} Q_k^\delta(x_j^\delta,u)\\
&\quad -r^\delta(x_{i_k}^\delta,u_k)-\gamma\sum_{j=1}^{m} p^\delta(x_j^\delta|x_{i_k}^\delta,u_k)\max_{u\in U^\delta(x_j^\delta)} Q_k^\delta(x_j^\delta,u)
\end{aligned}$$

and $x_k \in X_{i_k}$. Equation (C.26) has exactly the same form as equation (C.8)[4].
$H^\delta$ being a contraction mapping and $\alpha_k$ being nonnegative and satisfying expression (C.12), we still have to verify that $\eta_k$ satisfies expressions (C.10) and (C.11), where

$$\mathcal{F}_k = \{Q_0^\delta,\cdots,Q_k^\delta,(x_{i_0}^\delta,u_0),\cdots,(x_{i_k}^\delta,u_k),\alpha_0,\cdots,\alpha_k,\eta_0,\cdots,\eta_{k-1}\}, \quad \text{(C.28)}$$

in order to ensure the convergence of the algorithm to the fixed point of $H^\delta$.
In order to prove that expression (C.11) is satisfied, one can first note that :

$$|\eta_k| \le 2B_r + 2\gamma \max_{(x^\delta,u)\in X^\delta\times U^\delta} Q_k^\delta(x^\delta,u) \quad \text{(C.29)}$$

where $B_r$ is the bound on the rewards (section 2.1). Expression (C.29) implies that there exist two constants $A$ and $B$ such that

$$\eta_k^2 \le A + B\|Q_k^\delta\|_\infty^2 \quad \text{(C.30)}$$

and expression (C.11) is satisfied.
It is obvious that if equations (C.23) and (C.22) are satisfied then the condition (C.10) on the noise, i.e. $\underset{(x_k,w_k)}{E}[\eta_k|\mathcal{F}_k]=0$, is satisfied too.
By using equations (3.3) and (3.1) we can write :

$$\begin{aligned}
r^\delta(x_{i_k}^\delta,u) &= \underset{(x,w)}{E}[r(x,u,w)|x\in X_{i_k},p_{i_k}(x)] && \text{(C.31)}\\
p^\delta(x_j^\delta|x_{i_k}^\delta,u) &= \underset{(x,w)}{E}[I_{X_j}(f(x,u,w))|x\in X_{i_k},p_{i_k}(x)]. && \text{(C.32)}
\end{aligned}$$

[4] $Q_k^\delta$ corresponding to $R_k$, $H^\delta$ to $G$, $(x_{i_k}^\delta,u_k)$ to $e_k$ and $X^\delta\times U^\delta$ to $E$.

Equations (C.31) and (C.32) allow us to rewrite $\eta_k$ as follows :

$$\begin{aligned}\eta_k &= r(x_k,u_k,w_k)+\gamma\sum_{j=1}^{m} I_{X_j}(f(x_k,w_k,u_k))\max_{u\in U^\delta(x_j^\delta)} Q_k^\delta(x_j^\delta,u) \\ &\quad -\underset{(x,w)}{E}[r(x,u_k,w)|x\in X_{i_k},p_{i_k}(x)] \\ &\quad -\gamma\sum_{j=1}^{m}\underset{(x,w)}{E}[I_{X_j}(f(x,u_k,w))|x\in X_{i_k},p_{i_k}(x)]\max_{u\in U^\delta(x_j^\delta)} Q_k^\delta(x_j^\delta,u).\end{aligned} \tag{C.33}$$

We can verify that if one-step episodes are used with a probability distribution on the initial states given by (C.24) and if the probability to select an action $u\in U(x_{i_k}^\delta)$ while being in any state of $X_{i_k}$ is the same then $E[\eta_k|\mathcal{F}_k]=0$. Indeed with such conditions we have :

$$E[\eta_k|\mathcal{F}_k]=\underset{(x_k,w_k)}{E}[\eta_k|Q_k^\delta,\alpha_k,u_k,x_k\in X_{i_k},p_{i_k}(x_k)] \tag{C.34}$$

and equation (C.34) used with (C.33) allows us to verify that condition (C.10) is satisfied. **QED**

## C.7 Aggregation technique : convergence

### C.7.1 Convergence conditions

In the next subsection we prove that the algorithm described in figure 6.8 and entitled *Q-learning algorithm used with the aggregation technique* converges with probability 1 to the $Q^\delta$-function corresponding to the MDP$^\delta$ defined by equations :

$$r^\delta(x_i^\delta,u) = \underset{(x,w)}{E}[r(x,u,w)|x\in X_i,P_0(x,u)] \tag{C.35}$$

$$p^\delta(x_j^\delta|x_i^\delta,u) = \underset{(x,w)}{E}[I_{X_j}(f(x,u,w))|x\in X_i,P_0(x,u)] \tag{C.36}$$

if

- the $\alpha_k(x^\delta,u)$ are non negative and satisfy $\sum_{k=0}^{\infty}\alpha_k(x^\delta,u)=\infty$ and $\sum_{k=0}^{\infty}\alpha_k^2(x^\delta,u)<\infty$, $\forall(x^\delta,u)\in X^\delta\times U^\delta$
- one-step episodes are used and if each state-action pair $(x_0,u_0)$ is drawn independently according to the probability distribution $P_0(x,u)$.

### C.7.2 Proof

The proof follows the same lines as the one carried out in the previous section. We define a $H^\delta$ mapping by using equation (C.25) where $r^\delta(x_i^\delta, u)$ and $p^\delta(x_j^\delta|x_i^\delta, u)$ are this time defined by equations (C.35) and (C.36) and we prove that the *Q-learning algorithm used with the aggregation technique* converges with probability 1 to the fixed point of $H^\delta$ when the learning conditions specified in the previous subsection are met.
By following exactly the same procedure as the one carried out in the previous section, we obtain a noise factor defined this time by equation :

$$\begin{aligned}\eta_k &= r(x_k, u_k, w_k) + \gamma \sum_{j=1}^{m} I_{X_j}(f(x_k, w_k, u_k)) \max_{u \in U^\delta(x_j^\delta)} Q_k^\delta(x_j^\delta, u) \\ &\quad - \underset{(x,w)}{E}[r(x, u_k, w)|x \in X_{i_k}, P_0(x, u_k)] \\ &\quad -\gamma \sum_{j=1}^{m} \underset{(x,w)}{E}[I_{X_j}(f(x, u_k, w))|x \in X_{i_k}, P_0(x, u_k)] \max_{u \in U^\delta(x_j^\delta)} Q_k^\delta(x_j^\delta, u)\end{aligned} \tag{C.37}$$

that has to satisfy conditions (C.10) and (C.11).
One can immediately see that condition (C.11) on the noise is satisfied. It can be seen that condition (C.10) is satisfied by noting that :

$$E[\eta_k|\mathcal{F}_k] = \underset{(x_k,w_k)}{E}[\eta_k|Q_k^\delta, \alpha_k, u_k, x_k \in X_{i_k}, P_0(x_k, u_k)]. \tag{C.38}$$

Expression (C.38) combined with equation (C.37) allows us to verify that condition (C.10) is indeed satisfied.

## C.8 Representative states technique : convergence

### C.8.1 Convergence conditions

In the next subsection we prove that the algorithm described in figure 6.14 and entitled *Q-learning algorithm used with the representative states technique* converges with probability 1 to the $Q^\delta$-function corresponding to the MDP$^\delta$ defined by equations (3.14) and (3.15) if :

- one-step episodes starting from an element of $X^\delta$ are used
- condition of equivalence (3.21) is satisfied, that is if $\forall x \in X$

$$\max_{u \in U^\delta(x)} \sum_{x^\delta \in X^\delta} W(x, x^\delta) Q^\delta(x^\delta, u) = \sum_{x^\delta \in X^\delta} W(x, x^\delta) \max_{u \in U^\delta(x)} Q^\delta(x^\delta, u) \tag{C.39}$$

- the $\alpha_k(x^\delta, u)$ are non negative and satisfy $\sum_{k=0}^{\infty} \alpha_k(x^\delta, u) = \infty$ and $\sum_{k=0}^{\infty} \alpha_k^2(x^\delta, u) < \infty$, $\forall (x^\delta, u) \in X^\delta \times U^\delta$

where $\alpha_k(x^\delta, u) = \alpha_k$ if at the $k$th stage of the algorithm $(x_t, u_t)$ (figure 6.14) coincides with $(x^\delta, u)$ and 0 otherwise.

### C.8.2 Proof

We define the $L^\delta$ mapping. $L^\delta : B(X^\delta \times U^\delta) \to B(X^\delta \times U^\delta)$ such that

$$\begin{aligned}(L^\delta K)(x_i^\delta, u) &= r^\delta(x_i^\delta, u) + \\ &\gamma \max_{u' \in U^\delta(Reach(x_i^\delta, u))} \sum_{x_j^\delta \in X^\delta} p^\delta(x_j^\delta | x_i^\delta, u) K(x_j^\delta, u') \ \forall (x_i^\delta, u) \in X^\delta \times U^\delta\end{aligned} \tag{C.40}$$

with $K \in B(X^\delta \times U^\delta)$, $r^\delta(x_i^\delta, u)$ and $p^\delta(x_j^\delta | x_i^\delta, u)$ being defined respectively by equations (3.14) and (3.15) and $Reach(x_i^\delta, u)$ denotes an element of $X$ that can be reached when taking action $u$ in state $x_i^\delta$.
It can be shown that $L^\delta$ is a *contraction mapping*.
We consider hereafter that :

- one-step episodes starting from an element of $X^\delta$ are used
- the action taken while being in a state $x$ belongs to $U^\delta(x)$.

In such conditions the algorithm described in figure 6.14 updates at stage $k$ $Q^\delta$ as follows [5] :

$$\begin{aligned}Q_{k+1}^\delta(x_k, u_k) &= (1 - \alpha_k) Q_k^\delta(x_k, u_k) + \alpha_k (r(x_k, u_k, w_k) \\ &+\gamma \max_{u \in U^\delta(f(x_k, u_k, w_k))} \sum_{x^{\delta'} \in X^\delta} W(f(x_k, u_k, w_k), x^{\delta'}) Q_k^\delta(x^{\delta'}, u)),\end{aligned} \tag{C.41}$$

$Q_k^\delta$ representing the estimate of the $Q^\delta$-function at stage $k$ and $w_k$ being drawn independently according to $P_w(.|x_k, u_k)$.
By using the $L^\delta$ mapping definition (equation (C.40)), equation (C.41) can be rewritten as follows :

$$Q_{k+1}^\delta(x_k, u_k) = (1 - \alpha_k) Q_k^\delta(x_k, u_k) + \alpha_k ((L^\delta Q_k^\delta)(x_k, u_k) + \eta_k) \tag{C.42}$$

[5] The four-tuple $(x_t, u_t, r_t, x_{t+1})$ used as input of the algorithm described in figure 6.14 is "replaced" here by $(x_k, u_k, r(x_k, u_k, w_k), f(x_k, u_k, w_k))$.

with $\eta_k$ given by :

$$\begin{aligned}\eta_k &= r(x_k,u_k,w_k)\\ &+\gamma \max_{u\in U^\delta(f(x_k,u_k,w_k))} \sum_{x^{\delta\prime}\in X^\delta} W(f(x_k,u_k,w_k),x^{\delta\prime})Q_k^\delta(x^{\delta\prime},u)\\ &-[r^\delta(x_k,u_k)+\gamma \max_{u\in U^\delta(f(x_k,u_k,w_k))} \sum_{x^{\delta\prime}\in X^\delta} p^\delta(x^{\delta\prime}|x_k,u_k)Q_k^\delta(x^{\delta\prime},u)]\end{aligned}$$

which has exactly the same form as equation (C.8)[6].
$L^\delta$ being a contraction mapping and $\alpha_k$ being nonnegative and satisfying expression (C.12), we still have to verify that $\eta_k$ satisfies expressions (C.10) and (C.11), where

$$\mathcal{F}_k = \{Q_0^\delta,\cdots,Q_k^\delta,(x_0,u_0),\cdots,(x_k,u_k),\alpha_0,\cdots,\alpha_k,\eta_0,\cdots,\eta_{k-1}\}, \quad (C.43)$$

in order to ensure the convergence of the algorithm to the fixed point of $L^\delta$.
By using equations (3.14) and (3.15) we can write :

$$\begin{aligned}r^\delta(x_k,u_k) &= \underset{w}{E}[r(x_k,u_k,w)] && (C.44)\\ p^\delta(x^{\delta\prime}|x_k,u_k) &= \underset{w}{E}[W(f(x_k,u_k,w),x^{\delta\prime})]. && (C.45)\end{aligned}$$

Equations (C.44) and (C.45) allow us to rewrite $\eta_k$ as follows :

$$\begin{aligned}\eta_k &= r(x_k,u_k,w_k)\\ &+\gamma \max_{u\in U^\delta(f(x_k,u_k,w_k))} \sum_{x^{\delta\prime}\in X^\delta} W(f(x_k,u_k,w_k),x^{\delta\prime})Q_k^\delta(x^{\delta\prime},u)\\ &-\underset{w}{E}[r(x_k,u_k,w)]-\gamma \max_{u\in U^\delta(f(x_k,u_k,w_k))} \sum_{x^{\delta\prime}\in X^\delta} \underset{w}{E}[W(f(x_k,u_k,w),x^{\delta\prime})]Q_k^\delta(x^{\delta\prime},u)].\end{aligned}$$

Written under this form, it can easily be verified that $\underset{w_k}{E}[\eta_k|\mathcal{F}_k]=0$ ($\eta_k$ satisfies expression (C.10)).
In order to prove that expression (C.11) is satisfied, one can first note that :

$$|\eta_k| \leq 2B_r + 2\gamma \max_{(x^\delta,u)\in X^\delta\times U^\delta} Q_k^\delta(x^\delta,u) \quad (C.46)$$

[6] $Q_k^\delta$ corresponding to $R_k$, $L^\delta$ to $G$, $(x_k,u_k)$ to $e_k$ and $X^\delta\times U^\delta$ to $E$.

where $B_r$ is the bound on the rewards (section 2.1). Expression (C.46) implies that there exist two constants $A$ and $B$ such that

$$\eta_k^2 \leq A + B\|Q_k^\delta\|_\infty^2 \tag{C.47}$$

and expression (C.11) is satisfied.
Therefore convergence to the fixed point of the $L^\delta$ mapping is ensured. It can be shown that if equation (C.39) is satisfied then the fixed point of the $L^\delta$ mapping coincides with the $Q^\delta$-function solution of the MDP$^\delta$ which structure is defined by equations (3.14) and (3.15). **QED**

# Bibliography

[38.01] Task Force 38.02.19. System Protection Schemes in Power Networks. Technical report, CIGRE, June 2001.

[ACBF02] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47:235–256, 2002.

[ARS02] T.P.I. Ahamed, P.S.N. Rao, and P.S. Sastry. A reinforcement learning approach to automatic generation control. *Electric Power Systems Research*, 63:9–26, August 2002.

[AS97] C.G. Atkeson and J.C. Santamaria. A Comparison of Direct and Model-Based Reinforcement Learning. *International Conference on Robotics and Automation*, 1997.

[Ast65] K.J. Astrom. Optimal control of Markov decision processes with incomplete state estimation. *J. Math. Anl. Appl.*, 10, 1965.

[Bai95] L.C. Baird. Residual Algorithms : Reinforcement Learning with Function Approximation. In *Machine Learning : Proceedings of the Twelfth International Conference*, San Francisco, CA, 1995. Morgan Kaufmann.

[BD94] A.G. Barto and M. Duff. Monte-Carlo matrix inversion and reinforcement learning. In J.D. Cohen, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems : Proceedings of the 1993 Conference*, pages 687–694, San Francisco, 1994. Morgan Kaufmann.

[Bel57] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[Ber75] D.P. Bertsekas. Convergence of Discretization Procedures in Dynamic Programming. *IEEE Trans. on Automatic Control*, AC-20:415–419, 1975.

[Ber81] D.P. Bertsekas. Distributed Dynamic Programming. *IEEE Trans. on Automatic Control*, 27:610–616, 1981.

[Ber95] D.P. Bertsekas. *Dynamic Programming and Optimal Control*, volume II. Athena Scientific, Belmont, MA, 1995.

[Ber00] D.P. Bertsekas. *Dynamic Programming and Optimal Control*, volume I. Athena Scientific, Belmont, MA, 2nd edition, 2000.

[BEW03] O. Bilenne, D. Ernst, and L. Wehenkel. Adaptive Discretization for Model Based Reinforcement Learning. In preparation, to be submitted to ICML'03. 2003.

[BP90] G. Barles and B. Perthame. Comparison principle for dirichlet-type hamilton-jacobi equations and singular perturbations of degenerated elliptic equations. *Applied Mathematics and Optimization*, 21:21–44, 1990.

[BT89] D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and Distributed Computation : Numerical Methods*. Prentice Hall, Englewood Cliffs, N.J., 1989.

[BT96] D.P. Bertsekas and J.N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[CCPBS98] W.S. Cook, W.H. Cunningham, W.R. Pulley Blank, and A. Schrijver. *Combinatorial Optimization*. Wiley-Interscience, 1998.

[CKL94] A.R. Cassandra, L.P. Kaelbling, and M.L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1994.

[DeM98] C.L. DeMarco. The Threat of Predatory Generation Control : Can ISO Police Fast Time Scale Misbehavior ? In *Proceedings of Bulk Power System Dynamics and Control IV-Restructuring (IREP 1998)*, pages 281–289, Santorini, Greece, August 1998.

[DEW00] C. Druet, D. Ernst, and L. Wehenkel. Application of reinforcement learning to electrical power system closed-loop emergency control. In *Proceedings of PKDD 2000*, pages 86–95, September 2000.

[DH01] R.O. Duda and P.E. Hart. *Pattern Classification*. John Wiley & Sons, Inc., second edition, 2001.

[Die00] T. G. Dietterich. Ensemble Methods in Machine Learning. In *Proceedings of the first International Workshop on Multiple Classifier Systems*, 2000.

[EBZ$^{+}$98] D. Ernst, A.L. Bettiol, Y. Zhang, L. Wehenkel, and M. Pavella. Real-Time Transient Stability Emergency Control of the South-Southeast Brazilian System. In *Proceedings of the VI Symposium of Specialists in Electric Operational and Expansion Planning (SEPOPE 1998)*, Salvador, Brazil, May 1998.

[EGW03] D. Ernst, M. Glavic, and L. Wehenkel. Power System Stability Control : Reinforcement Learning Framework. In preparation, to be submitted to IEEE Transactions on Power Systems. 2003.

[Epp] D. Eppstein. Cube Triangulation and How Many Tetrahedra ? http://www.ics.uci.edu/~eppstein/junkyard/triangulation.html.

[Ern01] D. Ernst. Reinforcement Learning Applied to Power System Oscillations Damping. In *Proceedings of 40th IEEE Conference on Decision and Control*, Orlando, USA, December 2001.

[EW02] D. Ernst and L. Wehenkel. FACTS devices controlled by means of reinforcement learning algorithms. In *Proceedings of PSCC 2002*, Sevilla, Spain, June 2002.

[FS02] E.A. Feinberg and A. Shwartz, editors. *Handbook of Markov Decision Processes. Methods and Applications*. International series in Operations Research & Management Science. Kluwer Academic Publisher, 2002.

[GAPE01] M. Ghandhari, G. Andersson, M. Pavella, and D. Ernst. A Control Strategy for Controllable Series Capacitor. *Automatica*, 37(2):1575–1583, 2001.

[Geu02] P. Geurts. *Contributions to decision tree induction: bias/variance tradeoff and time series classification*. PhD thesis, University of Liège, Dept. of Electrical Engineering & Computer Science, Belgium, May 2002.

[GEW02] M. Glavic, D. Ernst, and L. Wehenkel. A Reinforcement Learning Based Discrete Supplementary Control for Power System Transient Stability Enhancement. Submitted to ISAP 2003. 2002.

[Gha00] M. Ghandhari. *Control Lyapunov Functions: A Control Strategy for Damping of Power Oscillations in Large Power Systems*. PhD thesis, Royal Instituate of Technology, Dept. of Electric Power Engineering, Electric Power Systems, 2000.

[HBWS00] S.A. Harp, S. Brignone, B.F. Wollenberg, and T. Samad. SEPIA. A Simulator for Electric Power Industry Agents. *IEEE Control Systems Magazine*, 20(4):53–59, August 2000.

[HG00] N.G. Hingorani and L. Gyugyi. *Understanding FACTS*. IEEE press, 2000.

[HL86] A. Haurie and P. L'Ecuyer. Approximation and Bounds in Discrete Event Dynamic Programming. *IEEE Trans. on Automatic Control*, AC-31:227–235, 1986.

[HLL96] O. Hernández-Lerma and B. Lasserre. *Discrete-Time Markov Control Processes*. Springer, New-York, 1996.

[HLL99] O. Hernández-Lerma and B. Lasserre. *Further Topics on Discrete-Time Markov Control Processes*. Springer, New-York, 1999.

[JJS94] T. Jaakkola, M.I. Jordan, and S.P. Singh. On the convergence of the stochastic iterative dynamic programming algorithms. *Neural Computation*, 6:1185–1201, 1994.

[Kal60] R.E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Trans. ASME Ser. D. J. Basic Engrg.*, 82:35–45, 1960.

[KD92] H.J. Kushner and Dupuis. *Numerical Methods for Stochastic Control Problems in Continuous Time*. Applications of Mathematics. Springer-Verlag, 1992.

[KLC98] L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101, 1998.

[Kun94] P. Kundur. *Power System Stability and Control*. McGraw-Hill, 1994.

[Kun00] P. Kundur. Future Directions in Power Systems. In *Proceedings of the VII Symposium of Specialists in Electric Operational and Expansion Planning (SEPOPE 2000)*, Curitiba, Brazil, May 2000.

[Kus90] H.J. Kushner. Numerical Methods for Stochastic Control Problems in Continuous Time. *SIAM J. Control and Optimization*, 28:999–1048, 1990.

[Lit94] M.L. Littman. The Witness Algorithm : Solving Partially Observable Markov Decision Processes. Technical report, Brown University, December 1994.

[LS61] L. Liusternik and V. Sobolev. *Elements of Functional Analysis*. Ungar, N.Y., 1961.

[Lue69] D.G. Luenberger. *Optimization by Vector Space Methods*. Wiley, N.Y., 1969.

[MA93] A.W. Moore and C.G. Atkeson. Prioritized Sweeping: Reinforcement Learning with Less Data and Less Real Time. *Machine Learning*, 13:103–130, 1993.

[Mat] The MathWorks, Inc. *Simulink, version 6.2*.

[May79] P.S. Maybeck. *Stochastic models, estimation, and control*, volume 141 of *Mathematics in Science and Engineering*. Academic Press, Inc., 1979.

[Meu96] N. Meuleau. *Le dilemme Exploration/Exploitation dans les systèmes d'apprentissage par renforcement*. PhD thesis, University of Caen, 1996.

[Meu99] N. Meuleau. Exploration of Multi-State Environments: Local Measures and Back-Propagation of Uncertainty. *Machine Learning*, 35:117–154, 1999.

[Moo92] D.W. Moore. *Simplical Mesh Generation with Applications*. PhD thesis, Cornell University, 1992.

[Mun97] R. Munos. *Apprentissage par renforcement, étude du cas continu*. PhD thesis, Ecole des Hautes Etudes en Sciences Sociales, 1997.

[Mun00] R. Munos. A study of reinforcement learning in the continuous case by the means of viscosity solutions. *Machine Learning*, 40:265–299, 2000.

[oAGoSC97] Task Force 13 of Advisory Group 02 of Study Committee 38. New Trends and Requirements for Dynamic Security Assessment. Technical report, CIGRE, December 1997.

[oAGoSC00] Task Force 16 of Advisory Group 02 of Study Committee 38. Impact of the Interaction Among Power System Controls. Technical report, CIGRE, 2000.

[Oga95] K. Ogata. *Discrete-Time Control Systems*. Prentice Hall, second edition, 1995.

[Pai89] M.A. Pai. *Energy Function Analysis for Power System Stability*. Power Electronics and Power Systems. Kluwer Academic Publishers, 1989.

[PBGM62] L. Pontryagin, V. Boltyanskii, R. Gamkriledze, and E. Mischenko. *Mathematical Theory of Optimal Processes*. Interscience, New-York, 1962.

[PERV00] M. Pavella, D. Ernst, and D. Ruiz-Vega. *Transient Stability of Power System. A Unified Approach to Assessment and Control*. Power Electronics and Power Systems. Kluwer Academic Publishers, 2000.

[Pha93] A.G. Phadke. Synchronized Phasor Measurements in Power. *IEEE Computer Applications in Power*, 6(2):10–15, April 1993.

[PS78] M.L. Puterman and M.C. Shin. Modified Policy Iteration Algorithm for Discounted Problems. *Management Science*, 24:1127–1137, 1978.

[PS82] M.L. Puterman and M.C. Shin. Action Elimination Procedures for Modified Policy Iteration Algorithms. *Operations Research*, 30:301–318, 1982.

[PS02] J. Persson and L. Söder. Validity of a Linear Model of Thyristor-Controlled Series Capacitor for Dynamic Simulations. In *Proceedings of PSCC 2002*, Sevilla, Spain, June 2002.

[PT88] A.G. Phadke and J.S. Thorp. *Computer Relaying for Power Systems*. Research Studies Press LTD., Taunton, Somerset, England, 1988.

[Rub81] R.Y. Rubinstein. *Simulation and the Monte-Carlo Method*. Wiley, New-York, 1981.

[Sal82] J.F. Sallee. A triangulation of the $n$-cube. *Discrete Math.*, 40:81–86, 1982.

[Sal84] J.F. Sallee. Middle-Cut Triangulations of the $n$-cube. *SIAM J. Algebraic and Discrete Methods*, 5:407–418, 1984.

[SB98] R.S. Sutton and A.G. Barto. *Reinforcement Learning, an Introduction*. MIT Press, 1998.

[SK00] W.D. Smart and L.P. Kaelbling. Practical Reinforcement Learning in Continuous Spaces. In *Proceedings of the Sixteenth International Conference on Machine Learning*, 2000.

[Smi88] W.D. Smith. *Studies in computational geometry motivated by mesh generation*. PhD thesis, Princeton University, 1988.

[SS85] P.J. Schweitzer and A. Seidman. Generalized Polynomial Approximations in Markovian Decision Processes. *Journal of Mathematical Analysis and Applications*, 110:568–582, 1985.

[SS96] S.P. Singh and R.S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158, 1996.

[Tod84] M.J. Todd. $J'$ : A New Triangulation of $R^n$. *SIAM J. Algebraic and Discrete Methods*, 5:244–254, 1984.

[Tsi94] J.N. Tsitsiklis. Asynchronous Stochastic Approximation and $Q$-learning. *Machine Learning*, 16:185–202, 1994.

[Wat89] C.J.C.H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, England, 1989.

[WD92] C.J.C.H. Watkins and P. Dayan. $Q$-learning. *Machine learning*, 8:279–292, 1992.

[Whi78] W. Whitt. Approximations of Dynamic Programs I. *Math. Operations Research*, 3:231–243, 1978.

[Whi79] W. Whitt. Approximations of Dynamic Programs II. *Math. Operations Research*, 4:179–185, 1979.

[Wil97] A.M. Wildberger. Complex Adaptive Systems : Concepts and Power Industry Applications. *IEEE Control Systems*, 17(6):77–88, 1997.