UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering, Mathematics & Computer Science

Department Applied Mathematics
Stochastic Operations Research

Approximate Dynamic Programming with Adaptive Multivariate Simplex Splines

---

Thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Author
Maike C. de Jongh

Supervisors
Dr.ir. A. Braaksma
Prof.dr. R.J. Boucherie

Graduation Committee
Dr.ir. A. Braaksma
Prof.dr. R.J. Boucherie
Dr. F.H.C. Bertrand
Dr.ir. M.R.K. Mes

March 23, 2021

Abstract

Large-scale Markov decision processes (MDPs) typically suffer from the curse of dimensionality, which renders exact solution methods intractable. For such problems, we rely upon approximate dynamic programming (ADP) techniques. A fundamental element of approximate dynamic programming is value function approximation. This research presents a value function approximation architecture that is based on adaptive multivariate simplex splines. Their linearity in the parameters and the fact that they are easily evaluated and adapted on a local basis renders these functions suitable candidates for this purpose. The approximation power of a multivariate simplex spline depends to a great extent on the triangulation on which it is defined. Our main contribution is a procedure that adaptively refines this triangulation in regions of the state space that require a more accurate value function approximation. This procedure is integrated into an ADP framework. The result is a method that can be applied to any MDP with a continuous (or large discrete) state space and finite discrete action space without the need of any preadjustment of the splines based on problem-specific knowledge. The method is tested on a problem of balancing an inverted pendulum. For this problem, the performance of our algorithm is not far behind that of a problem-specific method. The results indicate that multivariate simplex splines have high potential as value function approximators in an ADP context.

Preface

With this thesis, I conclude my time as a student at the University of Twente. It is meant to be the crowning achievement of 5.5 years of hard work and dedication. The pages that lie before you are the result of an adventure that lasted almost 8 months. From the solitary confinement of my student room, I made a voyage that took me along a wide variety of different aspects of both approximate dynamic programming and the theory of splines. As is inherent to any adventure, there have been moments of dazzling elation and moments of deep despair. I have felt lost in the dense woods of literature on approximate dynamic programming. I have struggled to move forward along dead-end roads, only finding myself forced to return. But all moments of hardship were made worthwile by the excitement of pursuing unexplored paths, by the joy and pride induced from reaping the first fruits of my newly planted tree and by the view over an empty plain, waiting to be seeded. For there is, I think, a lot of unexplored potential in simplex spline value function approximation.

Several people ought to be mentioned who have been of value and of guidance to me along my way. First of all, a big word of thanks goes out to Aleida Braaksma, my daily supervisor, who followed every single step of the process. Although we have never seen each other in person throughout the entire course of the project, we have spent many hours discussing online. Your critical view and detailed feedback has been of great value to me. And also, not to be forgotten, thank you for granting me access to your laptop.
Furthermore, I would like to thank Richard Boucherie for various thought-provoking discussions, which have provided me with many valuable insights, and also, of course, for drawing my attention to the splines.
In addition, my gratitude goes out to my parents and my sister, for all your support and unfaltering faith in me.
Also, I am deeply grateful to Dave, for the use of your laptop, for our refreshing conver-

sations, but most of all for your unwavering support and love.
Additionally, I owe a word of thanks to Jan-Kees van Ommeren for lending me three additional laptops.
Finally, I would like to thank Fleurianne Bertrand and Martijn Mes for taking the time and effort to read and assess my work.

Contents

List of symbols and abbreviations

MDP	-	Markov decision process.
$\mathcal{S}$	-	State space.
$\mathcal{A}$	-	Action space.
$\mathcal{F}$	-	Markovian transition model.
$\mathcal{C}$	-	Cost function.
γ	-	Discount factor.
$\mathcal{D}$	-	Initial state distribution.
T	-	Horizon.
π	-	Policy.
V^π	-	State value function corresponding to policy π.
Q^π	-	State-action value function corresponding to policy π.
V^*	-	Optimal state value function.
Q^*	-	Optimal state-action value function.
π^*	-	Optimal policy.
$\phi(\mathbf{x})$	-	Basis function or feature.
$\mathcal{M}$	-	Bellman operator.
$\mathcal{V}$	-	Space of value functions.
$\mathbf{t}$	-	Knot vector.
$B_{i,k}$	-	ith spline basis function of a B-spline of order k.
$\psi(\mathbf{x})$	-	Radial basis function.
$B_t(x\|X)$	-	Polyhedral spline of n-dimensional polyhedron $t \in \mathbb{R}^{n+s}$ and canonical projection X of vertices of t onto $\mathbb{R}^s$.
$\text{vol}_k(A)$	-	k-dimensional volume of set A.
M	-	Sample path length.
W	-	Random information.

$\mathbf{v}_i$	-	Vertex of simplex.
$\mathbf{V}_t$	-	Set of vertices of simplex t.
$\mathbf{b}(\mathbf{x})$	-	Barycentric coordinates of point $\mathbf{x}$.
$\mathbf{A_t}$	-	Normalized simplex vertex matrix of simplex t.
κ	-	Multi-index.
$\hat{d}$	-	Total number of permutations of multi-index κ of dimension n and 1-norm d.
$B^d_\kappa(\mathbf{b})$	-	Bernstein basis polynomial of degree d and multi-index κ.
$\mathbf{c}^{t_j}$	-	Vector of B-coefficients belonging to simplex t_j.
$\mathbf{B}^d_{t_j}(\mathbf{b})$	-	Vector of Bernstein basis polynomials belonging to simplex t_j.
$\mathcal{T}$	-	Triangulation.
$\mathcal{S}^r_d(\mathcal{T})$	-	Space of all spline functions s of degree d and continuity order $\mathcal{C}^r$ on a triangulation $\mathcal{T}$.
$\delta_{jk(\mathbf{x})}$	-	Simplex membership operator.
$\mathbf{D}_{t_j}(\mathbf{x})$	-	Per-simplex diagonal membership matrix.
$\mathbf{D}(\mathbf{x})$	-	Full-triangulation membership matrix.
$s^r_d(\mathbf{x}\|\mathbf{c})$	-	Simplex spline of degree d and continuity order $\mathcal{C}^r$ defined on a triangulation $\mathcal{T}$.
$\breve{\mathbf{c}}^t$	-	Sorted vector of vertex B-coefficients.
$\tilde{t}_{ij}$	-	Edge facet of simplices t_i and t_j.
$\tilde{\mathbf{v}}_{ij}$	-	Out-of-edge vertex of simplex t_i of edge facet $\tilde{t}_{ij}$.
$\mathcal{M}(j,\kappa)$	-	Tuple function that generates multi-indices.
$\rho(\mathbf{v}_i)$	-	Rank function that returns the rank of vertex $\mathbf{v}_i$ within a certain simplex t.
$\mathbf{H}$	-	Smoothness matrix.
E	-	Number of edges of triangulation.
SRLC	-	The location of the center and the radius of the circum(hyper)sphere of a simplex.
SRSC	-	The ratio between the radius of the circum(hyper)shpere and the shortest ridge of the simplex.
SMA	-	The minimum angle between two ridges of the simplex.
SDP	-	The number of data points contained by the simplex.
$\mathcal{P}_\mathcal{T}$	-	Metric to assess the quality of a triangulation $\mathcal{T}$.

PSLG	-	Planar straight-line graph.
CDT	-	Constrained Delaunay triangulation.
HCI	-	Hypercube-Convex hull-Intersection.
LOP	-	Local Optimization Procedure.
DLS	-	Discrete least squares.
$J_{\text{DLS}}(\mathbf{c})$	-	Cost function discrete least squares method.
KKT	-	Karush-Kuhn-Tucker.
GLS	-	Generalized least squares.
$\mathbf{\Sigma}$	-	Residual covariance matrix.
$J_{\text{GLS}}(\mathbf{c})$	-	Cost function generalized least squares method.
$\mathbf{r}$	-	Residual vector.
ECRLS	-	Equality constrained recursive least squares.
$J_{\text{ECRLS}}(\mathbf{c})$	-	Cost function equality constrained recursive least squares method.
$\hat{\mathbf{c}}$	-	Recursive least squares estimator of the B-coefficients.
$\mathbf{Z}$	-	Orthogonal projector onto the null-space of $\mathbf{H}$.
$\mathbf{P}$	-	Parameter covariance matrix estimate.
$\eta(a_i)$	-	Maximum value difference of region a_i.
$\zeta(a_i)$	-	Value variance of region a_i.
$\theta(a_i)$	-	Visitation frequency of region a_i.
$P^{\pi}(a_j^{t+1}\|a_i^t)$	-	Region transition probability from region a_i^t to region a_j^{t+1} under policy π.
A^{π}	-	Matrix of region transition probabilities induced by policy π.
$\varepsilon(s)$	-	Error estimate in a state $s \in \mathcal{S}$.
$\varepsilon(a_i)$	-	Local error estimate of region a_i.
$p_k^{\pi}(a_i\|a_j)$	-	k-step region transition probabilities.
$\xi^{\pi}(a_i, a_j)$	-	Influence of region a_i on region a_j under policy π.
$\mu(a_i)$	-	Error contribution of region a_i.
$S^{\max}(t)$	-	Maximum storages of water reservoirs at time t.
$\mathbf{m}$	-	Desired amounts of water in water reservoirs.
$\mathbf{C}(t)$	-	Cost coefficients for water reservoirs.
$\mathbf{q}(t)$	-	Inflows into water reservoirs at time t.

$\dot{\theta}(t)$	-	Angular velocity of the inverted pendulum.
$\ddot{\theta}(t)$	-	Angular acceleration of the inverted pendulum.
g	-	Gravitational acceleration.
l	-	Length of the pendulum.
θ	-	Angle of the pendulum with respect to vertical axis.
h	-	Discretization step size paramter.
$\dot{\theta}_{\max}$	-	Maximum angular velocity of the inverted pendulum.

Chapter 1

Introduction

Problems that involve sequential decision making under uncertainty arise in a wide variety of settings, for example inventory management, healthcare scheduling or control of heating systems. Such problems are typically modeled as a Markov decision process (MDP). Unfortunately, real-life MDPs often suffer from the curse of dimensionality, which renders exact solution methods intractable. For such complicated, large-scale problems, we rely upon approximate dynamic programming (ADP) techniques to find a sufficiently good solution without the need of huge computational efforts. This research focuses on MDPs with a continuous (or very large) discrete state space and a finite discrete action space.

One of the fundamental elements of approximate dynamic programming is value function approximation. Various approximation architectures have been developed over the past decades, which can be roughly distinguished into parametric and nonparametric models. Parametric models are easy to use and allow for the application of efficient and transparent learning methods. They are only effective, however, if they provide a good representation of the state space. Designing such a representation may be a challenging task. Nonparametric models circumvent this difficulty and can achieve a high approximation accuracy. This comes with the downside that they are typically intransparent and not easy to use.

In this thesis, we investigate the use of splines as value function approximators. Splines are functions that consist of multiple polynomial pieces. This piecewise nature suggests that these functions might achieve a higher approximation power and greater flexibility than functions that consist of only one piece. The goal of this project is reflected in the

following research question:

Are splines suitable candidates for value function approximation and how can we design an ADP framework that exploits their flexibility?

In answering this question, we let ourselves be guided by the following subquestions:

1. *Which type of splines is most suitable for value function approximation?*
2. *How can we identify "important" regions of the state space, that is, regions that require the most accurate value function approximation?*
3. *How can we adapt the splines in such a way that they provide a more accurate approximation in regions where this is required?*
4. *How can we integrate such an adaptive procedure into an ADP framework?*
5. *How well does the resulting ADP algorithm based on adaptive splines perform?*

The structure of this thesis is as follows:
in Chapter 2, we give an overview of relevant literature on value function approximation and on splines. We discuss several different types of multivariate splines and argue that the multivariate simplex spline is, in our view, the most suitable candidate for value function approximation. Furthermore, we examine earlier use of splines in the context of Markov decision theory. In addition, we briefly explain the recursive least squares approximate policy iteration (RLS API) algorithm described by Powell (2011), into which we will embed our adaptive procedure.
Chapter 3 provides an extensive overview of the theory on multivariate simplex splines. Furthermore, we discuss various methods through which multivariate simplex splines can be used as modelling tools.
The approximation power of a multivariate simplex spline depends to a great extent on the triangulation on which it is defined. In Chapter 4, we develop our main contribution, which is a procedure that adaptively refines this triangulation in regions of the state space that demand a more accurate value function approximation. We first embed this procedure into the renowned backward dynamic programming algorithm. Then, we integrate the method into the RLS API algorithm described in Chapter 2.
In Chapter 5, we test our algorithms on two case studies: a problem of controlling the

water flows in a system of water reservoirs and a problem of balancing an inverted pendulum. The simplex spline dynamic programming algorithm proves to be extremely time-consuming and not suitable for practical use. The simplex spline RLS API algorithm, on the other hand, achieves good results, which demonstrate the advantage of the triangulation refinement procedure over a fixed, balanced triangulation. Also, the performance is shown to be not far behind that of a problem-specific method. Moreover, we foresee possibilities for considerable futher performance improvement through modifications of both the algorithm itself and its implementation.

Finally, Chapter 6 presents our conclusions and some ideas on the improvement of our methods.

Chapter 2

Literature and preliminaries

In this chapter, we give an overview of the relevant literature on approximate dynamic programming, with an emphasis on value function approximation, and the theory of splines. Furthermore, we introduce the recursive least squares approximate policy iteration (RLS API) described by Powell (2011), into which we integrate our methods at a later stage.

2.1 The curse of dimensionality and approximate dynamic programming

This research studies Markov decision processes with a continuous (or large discrete) state space and a finite discrete action space. Such a Markov decision process (MDP) is commonly written as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{C}, \gamma, \mathcal{D})$, where $\mathcal{S}$ denotes the state space of the process, $\mathcal{A}$ is the set of possible actions, $\mathcal{F}$ the Markovian transition model, $\mathcal{C}$ a cost function, $\gamma \in (0, 1]$ a discount factor and $\mathcal{D}$ the initial state distribution. Here, $\mathcal{F}(s, a, s')$ is the probability density function that specifies the probability of a transition from a state s to a region $S' \subseteq \mathcal{S}$ after taking action a:

$$\int_{S'} \mathcal{F}(s, a, s')ds' = P(s_{t+1} \in S' | s_t = s \text{ and } a_t = a). \tag{2.1}$$

Furthermore, $\mathcal{C}(s, a)$ is the cost induced after taking action a in state s.
A Markov decision process is a discrete-time process: it starts at time $t = 0$ in some state $s_0 \in \mathcal{S}$ drawn from distribution $\mathcal{D}$. At each time step t, the process is in a certain state s_t and the decision maker selects an action $a_t \in \mathcal{A}$. The next state s_{t+1} is determined by

the transition model $\mathcal{F}(s_t, a_t, s')$ and the cost C_t follows from the cost function $\mathcal{C}(s_t, a_t)$. The horizon T is the number of time steps of each run of the process and is frequently infinite. A full run of the process over T timesteps is called an *episode* and is represented by a sequence of states, actions and costs: $(s_0, a_0, C_0, s_1, a_1, C_1, ..., s_T, a_T, C_T, s_{T+1})$. The goal of the decision maker is to choose actions in such a way that the so-called *expected total discounted cost* is minimized. This quantity is given by

$$E(C_0 + \gamma C_1, +\gamma^2 C_2 + \gamma^3 C_3 + ... + \gamma^T C_T | S_0 = s), \tag{2.2}$$

where s is drawn from $\mathcal{D}$.

A *policy* is a rule that dictates the decisions of the decision maker. A *deterministic policy* is a mapping $\pi : \mathcal{S} \to \mathcal{A}$ that selects a single action for each state $s \in \mathcal{S}$. A *stochastic policy* is a mapping $\pi : \mathcal{S} \to \Omega(\mathcal{A})$, where $\Omega(\mathcal{A})$ denotes the set of all probability distributions over $\mathcal{A}$. In this case $\pi(a|s)$ represents the probability of choosing action a in state s under policy π. A policy π is called *stationary* if it does not change over time, that is, $\pi_t = \pi$, $\forall t = 0, 1, ..., T$. The *optimal policy* π^* is a policy that minimizes the expected discounted costs obtained by following this policy from any state $s \in \mathcal{S}$. For every MDP, there exists at least one stationary and deterministic optimal policy. (Puterman, 1994).

The quality of a policy π can be expressed by means of a *value function*, which represents the expected costs induced by the policy for a certain initial situation. The *state value function* $V^\pi(s)$ denotes the expected costs under policy π when the process starts in state s:

$$V^\pi(s) = E_{a_t \sim \pi; S_t \sim \mathcal{F}; C_t \sim \mathcal{C}}(\sum_{t=0}^{\infty} \gamma^t C_t | S_0 = s). \tag{2.3}$$

Similarly, the *state-action value function* $Q^\pi(s, a)$ denotes the expected costs when the process starts in state s and the decision maker selects action a and then follows policy π:

$$Q^\pi(s, a) = E_{a_t \sim \pi; S_t \sim \mathcal{F}; C_t \sim \mathcal{C}}(\sum_{t=0}^{\infty} \gamma^t C_t | S_0 = s, a_0 = a). \tag{2.4}$$

The *optimal value functions* V^* and Q^* are the value functions V^{π^*} and Q^{π^*} that belong

to any optimal policy π^*. The optimal value function can be found by solving the *Bellman equations*:

$$V_t(s_t) = \max_{a_t \in \mathcal{A}_t} \left(C_t(s_t, a_t) + \gamma \int_{\mathcal{S}} \mathcal{F}(s_t, a_t, s') V_{t+1}(s') ds' \right), \ t = 0, 1, ..., T. \tag{2.5}$$

For infinite horizon MDPs, the Bellman equations reduce to

$$V(s) = \max_{a \in \mathcal{A}} \left(C(s, a) + \gamma \int_{\mathcal{S}} \mathcal{F}(s, a, s') V(s') ds' \right). \tag{2.6}$$

The operator $\mathcal{M} : \mathcal{V} \to \mathcal{V}$, where $\mathcal{V}$ denotes the space of value functions, defined by

$$\mathcal{M}V(s) = \max_{a \in \mathcal{A}} \left(C(s, a) + \gamma \int_{\mathcal{S}} \mathcal{F}(s, a, s') V(s') ds' \right) \tag{2.7}$$

is called the *Bellman operator*. For any initial value function V_0, the optimal value function can be computed by iteratively applying the Bellman operator:

$$V_k = \mathcal{M}V_{k-1}, \ k = 1, 2, ... \tag{2.8}$$

Unfortunately, real-life problems are often too complicated to find an exact solution to the optimality equations. The state and/or action spaces are in many cases simply too large, such that computing the value function requires a huge amount of computation time, data and memory space. For such problems, we rely upon *approximate dynamic programming* techniques to find a solution that is reasonably close to the optimal value function. Powell (2011) provides a detailed overview of approximate dynamic programming methods. In this research, we focus on MDPs with a continuous (or huge discrete) state space and a finite discrete action space.

2.1.1 Value function approximation

One of the fundamental elements in the field of approximate dynamic programming is value function approximation. Various methods to approximate the value function were developed by Bellman (1957). One of the most popular ones is *state aggregration* (Bellman, 1957). Here, the states are divided into several groups and one value is assigned to each of the groups.

Another way to approximate a value function is by means of a *parametric* model. The

most popular choice is a linear model, since it allows for the use of linear regression techniques to learn the parameters. In this case, the value function approximation is of the form

$$\hat{V}(s|\mathbf{w}) = \sum_{i=1}^{m} w_i \phi_i(s), \ s \in \mathcal{S}. \tag{2.9}$$

where w_i, $i = 1, ..., m$ are the parameters and $\phi_i : \mathcal{S} \to \mathbb{R}$, $i = 1, ..., m$ are the so-called *basis functions* or *features*. Parametric models are powerful approximators, since they allow for the use of efficient and transparent learning methods. However, these methods are only effective if the parametric model provides a good representation of the state space. Designing such a model may be a challenging endeavour. The features are mostly constructed by hand, using knowledge of the specific problem (for example, Gabillon et al., 2013). Various methods have been proposed that automatically construct a feature representation, without using knowledge of the problem (Barreto et al., 2018; Grover and Leskovec, 2016; Lehnert and Littman, 2019; Madjiheurem and Toni, 2019, Mahadevan, 2005; Mahadevan, 2010). These methods, however, are generally cumbersome and do not always perform well. Other examples of possible basis functions are polynomials (Bellman et al., 1963; Schweitzer and Seidmann, 1985), splines (Trick and Zin, 1997, Chen et al., 1999), the Fourier series (Konidaris and Osentoski, 2011), wavelets (Mahadevan et al., 2005) and radial basis functions (Sutton and Barto, 2018).

Nonparametric models circumvent the difficulty of constructing a suitable parametric state space representation and can achieve a great modelling accuracy. These advantages however, do come with a downside. Nonparametric models are typically intransparent and not easy to use. Each comes with its own complications. Examples of nonparametric approaches that have been used to solve MDPs are support vector machines (Tang, 2013), decision trees (Pyeatt and Howe, 1998) and neural networks (Coulom, 2002).

2.2 Splines

This research studies the use of *splines* as value function approximators. A spline can be described as a function that consists of multiple polynomial pieces which satisfies certain predefined continuity conditions between the pieces. This piecewise nature renders splines much more powerful modelling tools than functions that consist of only one piece.

Moreover, it allows for local model adaptation and efficient computational schemes. In this section, we investigate some popular types of splines, which might come of use in the context of Markov decision theory.

2.2.1 Univariate splines

The concept of a spline dates back to the middle-ages, where it was used in the design of bows and hulls of ships (de Boor, 1978). Nowadays, splines are used in a wide variety of fields, like system identification (Karagoz and Batselier, 202), medical imaging (Unser, 2002) and car design (Xue and Zhou, 2014).
The theory of one-dimensional splines, centering on the popular univariate B-spline, is advanced and considered fully developed (Cox, 1972; de Boor, 1972; de Boor, 1976). A *univariate spline* f maps a certain interval $[a, b]$ to the set of real numbers: $f : [a, b] \to \mathbb{R}$. The pieces of a spline are defined on ordered subintervals of the interval $[a, b]$. These subintervals are determined by the *knot vector* $\mathbf{t} = (t_0, ..., t_m)$, which satisfies

$$\begin{aligned} a = t_0 \leq t_1 \leq ... \leq t_{k-1} \leq t_m = b, \\ [a, b] = [t_0, t_1] \cup [t_1, t_2] \cup ... \cup [t_{m-1}, t_m]. \end{aligned} \tag{2.10}$$

On each subinterval $[t_i, t_{i+1}]$, we define a polynomial $P_i : [t_i, t_{i+1}] \to \mathbb{R}$:

$$f(t) = \begin{cases} P_0(t) \text{ if } t_0 \leq t < t_1 \\ P_1(t) \text{ if } t_1 \leq t < t_2 \\ \vdots \vdots \\ P_{m-1}(t) \text{ if } t_{k-1} \leq t \leq t_m. \end{cases} \tag{2.11}$$

Univariate splines are typically written in the form of a *B-spline* or *basis spline*. The advantage of this form is that it has minimal support with respect to a certain degree, smoothness and domain partition. It is uniquely determined by a knot vector and a corresponding sequence of coefficients. A B-spline of order k has the following form:

$$p(x) = \sum_{i=1}^{N} B_{i,k}(x) c_i, \tag{2.12}$$

where N is a positive integer, $c_1, c_2, ..., c_N$ are coefficients and $B_{i,k}$ is a polynomial of degree

$k-1$ called a *basis function*. To distinguish these functions from the basis functions used for value function approximation in the ADP context, we will refer to them as *spline basis functions*. For a given knot vector $\mathbf{t} = (t_0, ..., t_m)$, the unique basis functions of order k satisfy

$$\begin{aligned} B_{i,k}(x) &= \begin{cases} 0 \text{ if } x < t_i \text{ or } x \geq t_{i+k} \\ \text{nonzero otherwise} \end{cases} \\ \sum_i B_{i,k}(x) &= 1 \text{ for all } t_0 < x < t_m \end{aligned} \tag{2.13}$$

Spline basis functions can be constructed by means of the *Cox-de Boor recursion formula*:

$$B_{i,1}(x) = \begin{cases} 1 \text{ if } t_i \leq x < t_{i+1} \\ 0 \text{ otherwise,} \end{cases} \tag{2.14}$$

$$B_{i,k+1}(x) := \omega_{i,k}(x) B_{i,k}(x) + [1 - \omega_{i+1,k}(x)] B_{i+1,k}(x), \tag{2.15}$$

where

$$\omega_{i,k}(x) := \begin{cases} \dfrac{x - t_i}{t_{i+k} - t_i} \text{ if } t_{i+k} \neq t_i \\ 0, \text{ otherwise.} \end{cases} \tag{2.16}$$

A B-spline of order k is a polynomial function of degree $k-1$ that is continuous at each of the knots. When all knots are distinct, the first $k-2$ derivatives are continuous as well. When r knots coincide, only the first $k-r-1$ derivatives are continuous at that knot.

2.2.2 Multivariate splines

The B-spline is an elegant and powerful tool in univariate spline theory. Generalizing the B-spline to multiple dimensions, however, is not trivial and still object of research. Various types of multivariate splines have been developed, each with their own properties, but none of these truly generalize the univariate B-spline. De Boor and Ron (1990) wrote the following about this problem: "The generalization of univariate polynomial interpolation to the multivariate context is made difficult by the fact that one has to decide just which of its many nice properties to preserve, as it is impossible to preserve them all."

In this section, we discuss the most popular types of multivariate splines which possess at least some of the advantages of the univariate B-spline: the tensor product spline, the thin plate spline, the polyhedral spline and the simplex spline. Also, we discuss some earlier works on splines in the context of Markov decision theory.

2.2.2.1 Tensor product splines

The first attempt to lift the univariate B-spline to higher dimensions was made in 1959 by the mathematician Paul de Casteljau, who worked for car manufacturer Citroën (Farin, 2002). He invented a method that uses Bernstein polynomials defined on rectangular or triangular patches to fit smooth surfaces in two dimensions, which would form the basis of the multivariate simplex spline. The method of de Casteljau was kept secret by Citroën and reinvented in a slightly different form by Pierre Étienne Bézier in 1971, who based his work on what we now call Bézier curves and Bézier patches (Bézier, 1971). These rectangular patches paved the way for the widely used multivariate *tensor product splines* (TPS).

Tensor product splines are easily constructed by simply taking the tensor product of a number of univariate spline functions. The following bivariate tensor product spline for example, is a tensor product of two univariate B-splines:

$$f(x,y) = \sum_{i=1}^{r}\sum_{j=1}^{s} B_{i,v}(x)B_{j,w}(y)c_{ij}, \tag{2.17}$$

where r and s denote the number of spline basis functions of the two B-splines, $c \in \mathbb{R}^{r\times s}$ is a coefficient matrix and $B_{i,v}$ and $B_{j,w}$ are spline basis functions of orders v and w respectively. This can easily be generalized to a tensor product B-spline of dimension n:

$$\begin{aligned} f(\mathbf{x}) = \sum_{i_1=1}^{r_1}\sum_{i_2=1}^{r_2}\cdots\sum_{i_n=1}^{r_n} B_{i_1,v_1}(x_1)B_{i_2,v_2}(x_2)\cdots B_{i_n,v_n}(x_n)c_{i_1 i_2\cdots i_n} \\ = \prod_{j=1}^{n}\sum_{i_j=1}^{r_j} B_{i_j,v_j}(x_j)c_{i_1 i_2\cdots i_n}, \end{aligned} \tag{2.18}$$

where $c \in \mathbb{R}^{r_1\times r_2\times\cdots\times r_n}$ is an n-dimensional array of coefficients and B_{i_j,v_j} is a basis function of order v_j for $j = 1, 2, ..., n$.

The main advantages of tensor product splines are their computational simplicity and efficiency. They are widely used for applications in computer-aided design (Dokken et al., 2018). A major disadvantage is their dependence on rectangular domains, which renders them unsuitable to model scattered datasets (de Visser, 2011).

2.2.2.2 Thin plate splines

The desire to model real-life physical datasets gave rise to the development of the so-called *thin plate spline*. These splines are constructed by minimizing an energy integral. A thin plate spline is a linear combination of *radial basis functions* (RBFs) centered at the data points themselves. Hence, the number of basis functions of a thin plate spline equals the number of data points:

$$f(\mathbf{x}) = \sum_{j=1}^{N} \psi_j(\mathbf{x}) c_j, \tag{2.19}$$

where $\psi_j(\mathbf{x})$ are radial basis functions and $\mathbf{c} \in \mathbb{R}^N$ is a vector of coefficients.
Contrary to tensor product splines, thin plate splines are capable of modelling scattered data. However, this goes hand in hand with extreme computational inefficiency. After all, the basis functions of a thin plate spline are non-local, which means that they all contribute to each point on the spline. Hence, both the construction and the evaluation of a thin plate spline requires evaluation of each of its basis functions.

2.2.2.3 Polyhedral splines

In the 1980's, multivariate splines received a lot of attention from the academic world and great innovations took place. A breakthrough was the development of the *polyhedral spline* (de Boor, 1987; de Boor, 2000), which was regarded as the "true" generalization of the univariate B-spline. A polyhedral spline is constructed by projecting a multidimensional polyhedron in $\mathbb{R}^{n+s}$ onto a lower dimensional plane in $\mathbb{R}^s$. The value of the polyhedral spline at a certain point corresponds to the volume of the slice of the polyhedron that is projected onto this point.
Let $t \in \mathbb{R}^{n+s}$ be an n-dimensional polyhedron and let $X \subset \mathbb{R}^s$ be the canonical projection of the set of vertices of t onto $\mathbb{R}^s$. A canonical projection of a point $p = (p_1, p_2, ..., p_{s+n}) \in \mathbb{R}^{n+s}$ onto $\mathbb{R}^s$ is defined as $p|_{\mathbb{R}^s} = (p_1, p_2, ..., p_s)$. The elements of X serve as the knots of the spline in $\mathbb{R}^s$. Now, the polyhedral spline $B_t(x|X)$ is defined as follows:

$$B_t(x|X) := \frac{\mathrm{vol}_n(p \in t : p|_{\mathbb{R}^s} = x)}{\mathrm{vol}_{n+s}(t)}, \ x \in \mathbb{R}^s, \tag{2.20}$$

where $\mathrm{vol}_k(A)$ denotes the k-dimensional volume of a set A. This function has degree n. The degree of a polyhedral spline can be increased by choosing polyhedrons of higher dimensions.

Figure 2.1 shows an example of the construction of a polyhedral spline (de Visser, 2011).

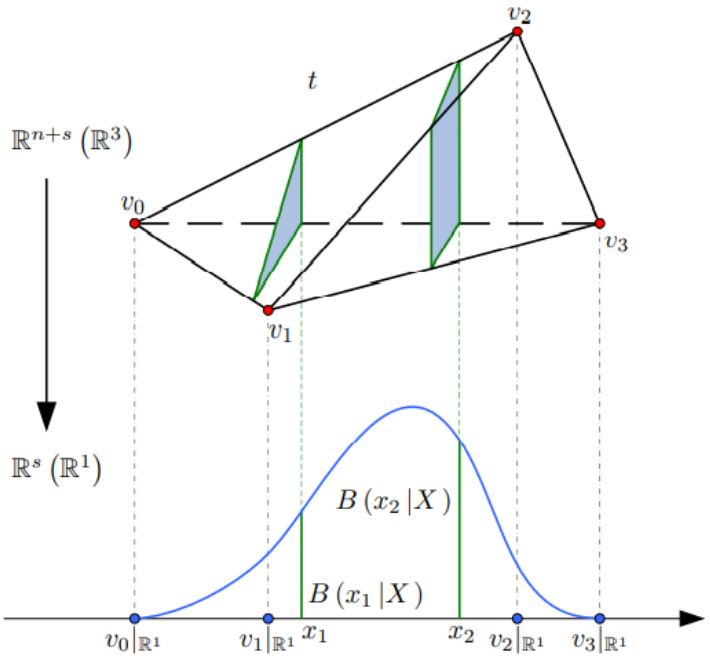


Figure 2.1: Construction of a polyhedral spline (de Visser, 2011)

Here, t is a tetrahedron in $\mathbb{R}^3$ with vertices v_0, v_1, v_2 and v_3, which is projected onto $\mathbb{R}$. This implies that the resulting polyhedral spline is univariate and has degree 2. The knots are the projections of the vertices of t onto $\mathbb{R}$, so $X = \{v_0|_{\mathbb{R}}, v_1|_{\mathbb{R}}, v_2|_{\mathbb{R}}, v_3|_{\mathbb{R}}\}$.

Despite their grand entrance, the polyhedral spline only enjoyed a short period of popularity. Although they are mathematically elegant, they have proved to be hard to construct and evaluate, very inefficient and thus almost impossible to use in practice (Goodman, 1990).

2.2.2.4 Multivariate simplex splines

While most eyes were directed to the polyhedral spline, a small number of mathematicians worked on the development of another type of multivariate spline, based on the triangular patches invented by de Casteljau (Farin, 2002). Each piece of this spline is defined on one of these patches, or *simplices*. Together, the simplices form a so-called *triangulation*. The pieces are tied together through certain predetermined continuity constraints.
The new spline became known as the *multivariate simplex spline*. Its two-dimensional version first appeared in 1986 in a paper by Farin (Farin, 1986). Its general formulation in the so-called B-form was presented one year later by de Boor (1987). The theory on simplex splines was further developed by Lai and Schumaker (Lai, 1996; Lai, 1997; Lai and Schumaker, 1997; Lai and Schumaker, 1998; Lai and Schumaker, 2001). The multivariate simplex spline was received with scepticism. The concept of the knot, which plays a fundamental role in the theory of univariate B-splines, is completely absent. Also, it was feared that the dependence on the triangulation would render simplex splines inflexible. These concerns made that the multivariate simplex spline has mainly existed in the shadow of other types of multivariate splines and has not been widely used for applications.
Despite its lack of popularity however, the multivariate simplex spline proved to be a powerful modeling tool, which can achieve a high approximation power and great computational efficiency. Against the expectations, the simplex spline actually proved to be very flexible. After all, there exists a wide variety of triangulations, each with their own properties. Moreover, a triangulation can easily be refined or simplified locally depending on the complexity of the data. Based on these properties, we consider the multivariate simplex spline the most suitable candidate for value function approximation. In Chapter 3, we discuss the multivariate simplex spline in more detail.

2.2.3 Splines in the context of Markov decision theory

Splines have sofar not been widely used in the context of Markov decision processes. Trick and Zin (1997) developed a method that uses univariate cubic splines as value function approximators. This method, however is only applicable to one-dimensional state spaces. Johnson et al. (1993) used tensor product splines to solve a stochastic 4-dimensional water supply reservoir problem. This approach, however, is computationally burdensome for higher dimensional state spaces: the number of discretization points necessary for this

approximation grows exponentially with the number of state variables ($O(4^n)$). Chen et al. (1999) addressed this issue by using the multivariate adaptive regression splines (MARS) algorithm. This procedure, introduced by Friedman (1991), consists of three parts: first of all, a forward stepwise algorithm selects a number of spline basis functions. Then, a backward stepwise algorithm deletes basis functions that cause overfitting. Finally, a smoothing method is applied to provide the approximation with a suitable degree of continuity.
Although the MARS approach paves the way for spline approximations for higher dimensional state spaces, the tensor product splines of Johnson et al. (1993) have proved to be more efficient for state spaces with dimension less than five.
Both the tensor product spline method of Johnson et al. (1993) and the MARS approach of Chen et al. (1999) were designed to be used in combination with the backward dynamic programming algorithm. They have, to our knowledge, not been integrated into an ADP framework sofar. Whereas tensor product splines can probably be used as value function approximators in an ADP algorithm, the MARS approach is a non-parametric technique, which renders it not easy, if not impossible, to use this method in an ADP context.

2.2.4 Recursive least squares approximate policy iteration

The goal of approximate dynamic programming is not only to accurately approximate the value function of a fixed policy, but also to improve this policy. Powell (2011) discusses several methods that simultaneously learn an accurate value function approximation and identify a close to optimal policy. In this thesis we use the RLS API algorithm (Powell, 2011) as a framework for our value function approximation methods. According to Powell (2011), this algorithm "represents the most natural extension of classical policy iteration for infinite horizon problems" (Powell, 2011). We expect that our methods can also be used in combination with other ADP approaches. In this section, we give a brief overview of the original RLS API algorithm as described by Powell (2011).

The RLS API algorithm makes use of a linear value function approximation of the form of expression (2.9), that is,

$$\hat{V}(s|\mathbf{w}) = \sum_{i=1}^{m} w_i \phi_i(s), \;\; s \in \mathcal{S}. \tag{2.21}$$

The algorithm starts off with an initial parameter vector $\mathbf{w}_0$ and an initial policy π_0 that is the greedy policy with respect to the value function approximation $\hat{V}(s|\mathbf{w}_0)$. Then, it alternately takes a walk through the state space to find a better value function approximation and updates the policy to the greedy policy of the thus obtained new approximation.

Suppose that after iteration $k-1$, we have a parameter vector $\mathbf{w}_k$ and a policy π_k which is greedy with respect to the value function approximation $\hat{V}(s|\mathbf{w}_k)$. We start iteration k by initializing value function parameters $\mathbf{w}_k^0$ as $\mathbf{w}_k$ and randomly choosing an initial state $s^{k,0}$. From this state, we follow a sample path
$(s^{k,0}, a^0, C^0, s^{k,1}, a^1, C^1, s^{k,2}, ..., a^{M-1}, C^{M-1}, s^{k,M})$ of length M, where the actions are chosen according to policy π_k and C^i denotes the costs induced after taking action a^i from state $s^{k,i}$, $i = 0, 1, ..., M-1$. For each step of the sample path, we compute

$$\boldsymbol{\delta}_j = \boldsymbol{\phi}(s^{k,j}) - \gamma \boldsymbol{\phi}(s^{k,j+1}). \tag{2.22}$$

The value function parameters $\mathbf{w}_k^j$ are now updated according to the recursive least squares procedure, which is specified by the following set of equations:

$$\begin{aligned}
\epsilon^{j+1} &= C^j - \boldsymbol{\delta}_j^T \mathbf{w}_k^j, \\
B^{j+1} &= B^j - \frac{B^j \boldsymbol{\phi}(s^{k,j}) \boldsymbol{\delta}_j^T B^j}{1 + \boldsymbol{\delta}_j^T B^j \boldsymbol{\phi}(s^{k,j})}, \\
\mathbf{w}_k^{j+1} &= \mathbf{w}_k^j + \frac{\epsilon^{j+1} B^j \boldsymbol{\phi}(s^{k,j})}{1 + \boldsymbol{\delta}_j^T B^j \boldsymbol{\phi}(s^{k,j})},
\end{aligned} \tag{2.23}$$

where the matrix B^0 is initialized as ϵI for a small constant ϵ.
After following the sample path, the global value function parameters are updated as $\mathbf{w}_{k+1} = \mathbf{w}_k^M$ and policy π_{k+1} is chosen to be the greedy policy with respect to the value function approximation $\hat{V}(s|\mathbf{w}_{k+1})$. An overview of the RLS API algorithm is given in Algorithm 1. For MDPs with a discrete state space, it has been shown to converge by Bradtke and Barto (1996). Ma and Powell (2009) extended these results to MDPs with continuous state spaces.
For finite horizon problems, a similar version of the RLS API algorithm applies (Powell, 2011).

Algorithm 1: Recursive least squares approximate policy iteration

Input: An infinite horizon Markov decision process $(\mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{C}, \gamma, \mathcal{D})$, a linear value function approximator $\hat{V} : \mathcal{S} \times \mathbb{R}^m \to \mathbb{R}$ as given by expression (2.21), sample path length M.

Initialize: *Initialize parameter vector* $\boldsymbol{w}_0$ *arbitrarily, for example* $\boldsymbol{w}_0 = 0$. *Let* π_0 *be the greedy policy with respect to* $\hat{V}(s|\boldsymbol{w}_0)$.

for $k = 0, 1, ...$ **do**
- Arbitrarily choose initial state $s^{k,0}$.
- Initialize $\mathbf{w}_k^0 = \mathbf{w}_k$.
- **for** $j = 0, 1, ..., M-1$ **do**
  - Sample random information W^{j+1}.
  - Choose action a^j according to policy π_k.
  - Compute next state $s^{k,j+1}$ after taking action a^j from state $s^{k,j}$ and observing W^{j+1}.
  - Compute costs C^j induced by taking action a_j from state $s^{k,j}$ and observing W^{j+1}.
  - Compute $\boldsymbol{\delta}$ as given by expression (2.22).
  - Compute $\mathbf{w}_k^{j+1}$ by the recursive least squares procedure (2.23).
- **end**
- Let $\mathbf{w}_{k+1} = \mathbf{w}_k^M$ and let π_{k+1} be the greedy policy with respect to $\hat{V}(s|\mathbf{w}_{k+1})$.

end

2.3 Conclusion literature review

In this chapter, we gave an overview of the literature on several topics that are relevant to our research. In Section 2.1.1, we discussed various approaches that have been designed to approximate the value function. These approaches can be roughly distinguished into parametric models and nonparametric models. The advantage of parametric models is that they are easy to use and allow for the use of efficient and transparent learning methods. On the other hand, it may be challenging to design the model in such a way that it provides a good representation of the state space. Nonparametric models do not suffer from this difficulty and can achieve great accuracy. They are, however, typically intransparent and not easy to use.

In Section 2.2, we discussed some popular types of splines that could potentially be used as value function approximators. Also, we gave an overview of earlier works that use splines in the context of Markov decision theory. These approaches are all designed to be used in combination with backward dynamic programming. The most promising of these methods, the MARS approach, is a non-parametric technique and therefore probably not suitable to be integrated into an ADP framework. We considered four types of multivariate splines that could potentially be used as value function approximators in an ADP algorithm: the tensor product spline, the thin plate spline, the polyhedral spline and the multivariate simplex spline. The tensor product spline is computationally efficient and easy to use. However, it is very inflexible and not capable of handling scattered datasets. Thin plate splines do have this ability, but this comes at the price of extreme computational inefficiency. The polyhedral spline also proved to be inefficient and hard to use. The multivariate simplex spline, on the other hand, is a powerful modelling tool. It can achieve a high approximation power and great computational efficiency. The fact that it is linear in its parameters, allows for the use of the efficient learning techniques that are used for linear parametric models. Moreover, the fact that it is defined on a triangulation renders the simplex spline very flexible. After all, there exists a wide variety of triangulations which are easily refined or simplified on a local basis. This is a big advantage of the multivariate simplex spline over other parametric models. Finally, like nonparametric models, simplex splines do not involve the design of a suitable state space representation for a specific problem. These characteristics render them potentially strong value function approximators.

To our knowledge, we are the first to present a value function approximation architecture that is based on adaptive multivariate simplex splines. Our main contribution is a method that exploits the flexibility of these functions. This method adaptively refines the triangulation on which the simplex splines are defined in regions of the state space that require a more accurate value function approximation. The triangulation refinement procedure is first integrated into the basic backward dynamic programming algorithm. Then, we embed it into the RLS API algorithm dicussed in Section 2.2.4.

Chapter 3

Multivariate simplex splines

This chapter provides an overview of the relevant theory on multivariate simplex splines. Furthermore, we discuss several methods that are used to construct simplex spline models.

3.1 Introduction to multivariate simplex splines

3.1.1 The simplex

At the basis of the multivariate simplex spline lies the concept of a *simplex*, a geometric entity that is established by a set of non-degenerate vertices.

Definition 3.1.1. (Non-degenerate vertices)
Let $\mathbf{V}_t = (\mathbf{v}_0, \mathbf{v}_1, ..., \mathbf{v}_n)$, $\mathbf{v}_i \in \mathbb{R}^{n+k}$, $k \geq 0$ be a tuple of $n+1$ vertices. This set of vertices is called *non-degenerate* if its convex hull has dimension n.

Definition 3.1.2. (Simplex) (Lai and Schumaker, 2007)
Let $\mathbf{V}_t = (\mathbf{v}_0, \mathbf{v}_1, ..., \mathbf{v}_n)$, $\mathbf{v}_i \in \mathbb{R}^{n+k}$, $k \geq 0$ be a tuple of $n+1$ non-degenerate vertices. The n-dimensional polytope that is the convex hull of these vertices is called a *simplex*.

Simplices of dimension higher than 3 are hard to visualize. Figure 3.1 shows the projections of the *regular simplices* of dimensions 0 to 15 onto the two-dimensional plane (de Visser, 2011). A simplex is regular if the distance between $\mathbf{v}_i$ and $\mathbf{v}_j$ is the same for any pair of vertices $\mathbf{v}_i$, $\mathbf{v}_j$.

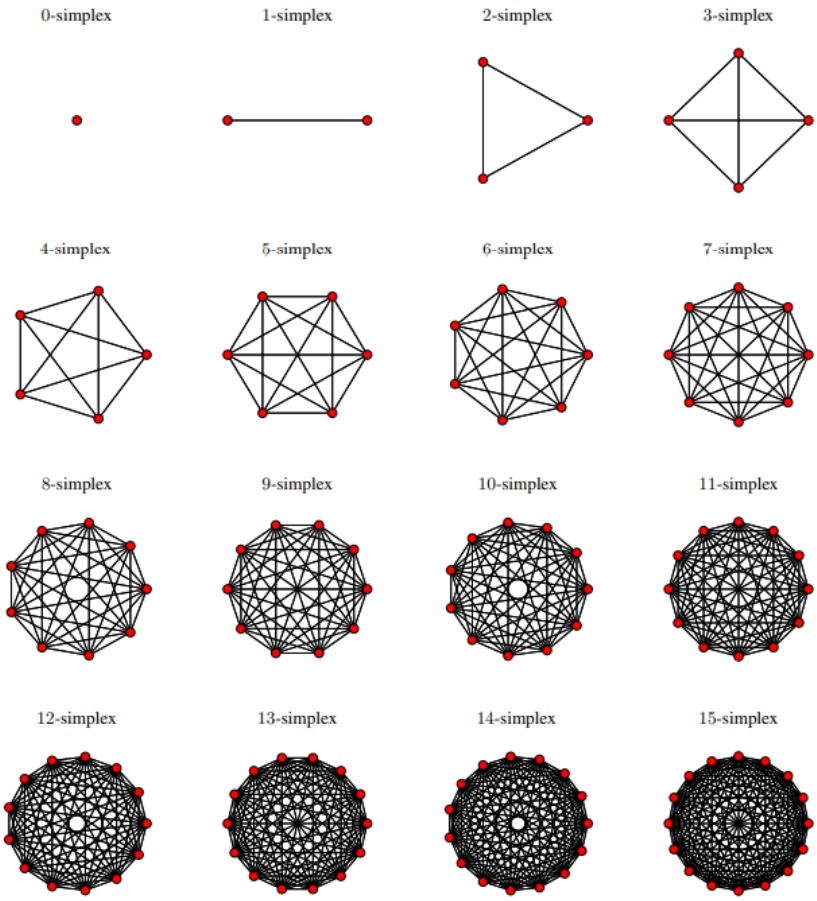


Figure 3.1: Projections of the regular simplices of dimensions 0 to 15 onto the two-dimensional plane (de Visser, 2011).

A nice property of a simplex is that its *faces*, or the convex hulls of the nonempty subsets of the vertices, are simplices themselves of lower dimensions.

3.1.2 Barycentric coordinates

Each point $\mathbf{x}$ in n-dimensional Euclidean space can be written in terms of the vertices of an n-simplex.

Definition 3.1.3. (Barycentric coordinates) (Lai and Schumaker, 2007)
Let the simplex t be defined by the vertices $\mathbf{v}_0, \mathbf{v}_1, ..., \mathbf{v}_n \in \mathbb{R}^n$. Any point $\mathbf{x} = (x_1, x_2, ..., x_n) \in$

$\mathbb{R}^n$ corresponds to a *barycentric coordinate* $b(\mathbf{x}) = (b_0, b_1, ..., b_n) \in \mathbb{R}^{n+1}$ with respect to t, through the following relation:

$$\mathbf{x} = \sum_{i=0}^{n} b_i \mathbf{v}_i, \quad \sum_{i=0}^{n} b_i = 1. \tag{3.1}$$

The barycentric coordinates of a point $\mathbf{x} \in \mathbb{R}^n$ can easily be derived using the *normalized simplex vertex matrix*.

Definition 3.1.4. (Normalized simplex vertex matrix)(de Visser, 2011)
Let the simplex t be defined by the vertices $\mathbf{v}_0, \mathbf{v}_1, ..., \mathbf{v}_n \in \mathbb{R}^n$. The *normalized simplex vertex matrix* that belongs to this simplex is given by

$$\mathbf{A}_t = [\mathbf{v}_1 - \mathbf{v}_0, \mathbf{v}_2 - \mathbf{v}_0, ..., \mathbf{v}_n - \mathbf{v}_0]. \tag{3.2}$$

The following derivation shows the relation between the normalized simplex vertex matrix of a simplex t defined by the vertices $\mathbf{v}_0, \mathbf{v}_1, ..., \mathbf{v}_n \in \mathbb{R}^n$ and the barycentric coordinates of a point $\mathbf{x} \in \mathbb{R}^n$:

$$\begin{aligned} \mathbf{x} &= \sum_{i=0}^{n} b_i \mathbf{v}_i \\ &= b_0 \mathbf{v}_0 + \sum_{i=1}^{n} b_i \mathbf{v}_i \\ &= (1 - \sum_{i=1}^{n} b_i)\mathbf{v}_0 + \sum_{i=1}^{n} b_i \mathbf{v}_i \\ &= \mathbf{v}_0 + \sum_{i=1}^{n} b_i (\mathbf{v}_i - \mathbf{v}_0). \end{aligned} \tag{3.3}$$

This implies

$$\mathbf{x} - \mathbf{v}_0 = \sum_{i=1}^{n} b_i (\mathbf{v}_i - \mathbf{v}_0) = \mathbf{A}_t \mathbf{b}^T. \tag{3.4}$$

Hence,

$$\mathbf{b}^T = A_t^{-1} (\mathbf{x} - \mathbf{v}_0). \tag{3.5}$$

3.1.3 Bernstein basis polynomials

A simplex spline is constructed from a number of so-called *Bernstein polynomials* which are each defined on their own simplex. In order to define a Bernstein polynomial, we first introduce the concept of a *multi-index*.

Definition 3.1.5. (Multi-index) (de Visser, 2011)
A multi-index κ of dimension n is a tuple $\kappa := (\kappa_0, \kappa_1, ..., \kappa_n) \in \mathbb{N}^{n+1}$. Its 1-norm satisfies

$$|\kappa| = \kappa_0 + \kappa_1 + ... + \kappa_n = d,\ d \geq 0 \tag{3.6}$$

and its factorial is given by

$$\kappa! = \kappa_0!\kappa_1! \cdots \kappa_n!. \tag{3.7}$$

The total number of permutations of κ can be shown to be (De Boor, 1987)

$$\hat{d} = \binom{d+n}{d} = \frac{(d+n)!}{n!d!}. \tag{3.8}$$

Multi-indices can be used among others to simplify the multinomial theorem:

$$(b_0 + b_1 + ... + b_n)^d = \sum_{\kappa_0+\kappa_1+...+\kappa_n=d} \frac{d!}{\kappa_0!\kappa_1! \cdots \kappa_n!} b_0^{\kappa_0} b_1^{\kappa_1} \cdots b_n^{\kappa_n} = \sum_{|\kappa|=d} \frac{d!}{\kappa!} \mathbf{b}^\kappa. \tag{3.9}$$

Using the concept of a multi-index, we define the *Berstein basis polynomials*, which are the building blocks of the Berstein polynomials.

Definition 3.1.6. (Bernstein basis polynomials)
The Bernstein basis polynomial of degree d and multi-index κ is defined as:

$$B_\kappa^d(\mathbf{b}) := \frac{d!}{\kappa!} \mathbf{b}^\kappa. \tag{3.10}$$

Combining expressions (3.9) and (3.10) yields

$$(b_0 + b_1 + ... + b_n)^d = \sum_{|\kappa|=d} B_\kappa^d(\mathbf{b}). \tag{3.11}$$

Now, suppose that $\mathbf{b}$ is a barycentric coordinate. From expression (3.1), it then follows that

$$\sum_{|\kappa|=d} B_\kappa^d(\mathbf{b}) = 1. \tag{3.12}$$

This property is called the *partition of unity* of the Bernstein basis polynomials: at every location within a simplex, the Bernstein basis polynomials sum up to 1.
A Bernstein polynomial is a linear combination of Bernstein basis polynomials.

3.1.4 The B-form

In 1987, de Boor discovered that each polynomial can be written uniquely as a linear combination of Bernstein basis polynomials, the so-called *stable local basis property*. Such an expression is called the *B-form* of a polynomial.

Theorem 1. (B-form) (de Boor, 1987)
Any polynomial $p(\mathbf{x})$ of degree d can be written in the following form:

$$p(\mathbf{x}) = \sum_{|\kappa|=d} c_\kappa B_\kappa^d(b(\mathbf{x})). \tag{3.13}$$

This is expression is called the *B-form* of $p(\mathbf{x})$. The coefficients c_κ are called the *B-coefficients*. A polynomial written in B-form is also called a *simplex polynomial*.

In order to formulate the concept of a simplex spline, it is convenient to express the B-form in its vector form (de Visser et al., 2009). This vector form consists of a vector of B-coefficients and a vector of Bernstein basis polynomials. To make sure that a B-coefficient is matched to the right basis polynomial, an ordering of the multi-index κ is necessary. Lai and Schumaker (2007) introduce a lexicographic sorting order:

$$\kappa_{d,0,0,\ldots,0} > \kappa_{d-1,1,0,\ldots,0} > \kappa_{d-1,0,1,0,\ldots,0} > \ldots > \kappa_{0,\ldots,0,1,d-1} > \kappa_{0,\ldots,0,0,d}. \tag{3.14}$$

Now, we can construct vectors of the B-coefficients and the basis polynomials belonging to a simplex t_j according to this sorting rule:

$$\mathbf{c}^{t_j} := [c_\kappa^{t_j}]_{|\kappa|=d} \in \mathbb{R}^{\hat{d}\times 1},\ \mathbf{B}_{t_j}^d(\mathbf{b}) := [B_\kappa^d(\mathbf{b})]_{|\kappa|=d} \in \mathbb{R}^{1\times\hat{d}}. \tag{3.15}$$

The B-form of equation (3.13) in vector form for simplex t_j is defined as

$$p(\mathbf{x}) = \mathbf{B}^d_{t_j}(b(\mathbf{x})) \cdot \mathbf{c}^{t_j}. \tag{3.16}$$

The Bernstein basis polynomials are the building blocks of multivariate simplex splines. A multivariate simplex spline of dimension n and degree d on a single simplex t is defined as:

$$p(\mathbf{x}) = \sum_{|\kappa|=d} c_\kappa B^d_\kappa(b(\mathbf{x})), \tag{3.17}$$

where $b(\mathbf{x}) \in \mathbb{R}^{n+1}$ is the barycentric coordinate of the point $\mathbf{x} \in \mathbb{R}^n$ with respect to t.

3.1.5 The B-coefficient net

One of the most powerful features of simplex splines in B-form lies in the fact that the B-coefficients have a spatial representation. The barycentric coordinates of a B-coefficient c_k in a certain simplex are given by

$$b(c_\kappa) = \frac{\kappa}{d}, \tag{3.18}$$

where

$$d = |\kappa|. \tag{3.19}$$

Figure 3.2 shows the spatial locations of the B-coefficients of a polynomial of degree 4 in a 2-dimensional simplex. The density of the B-net is higher for polynomials with a higher degree.

This spatial structure of B-coefficients is called the *B-coefficient net*, or *B-net*. It is convenient since it allows for local adaptation of the model by only modifying the B-coefficients in a particular region. Also, it greatly simplifies expressions of continuity between simplices.

3.1.6 Simplex splines on triangulations

Usually, simplex splines are defined on multiple different simplices. A smart combination of simplices can provide a simplex spline with great approximation power. Such a combination of simplices must satisfy a number of constraints, among others to ensure that the

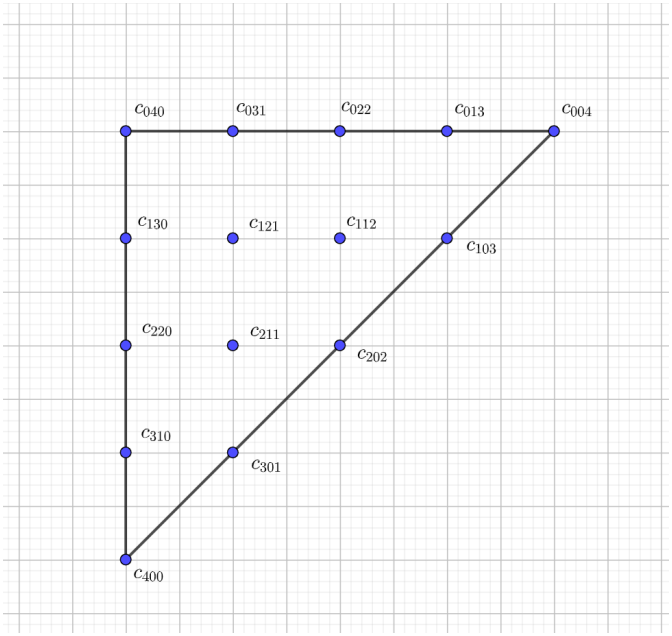


Figure 3.2: B-net of a polynomial of degree 4 in a 2-dimensional simplex

resulting simplex polynomials are continuous between neighbouring simplices. A valid set of simplices is called a *triangulation*.

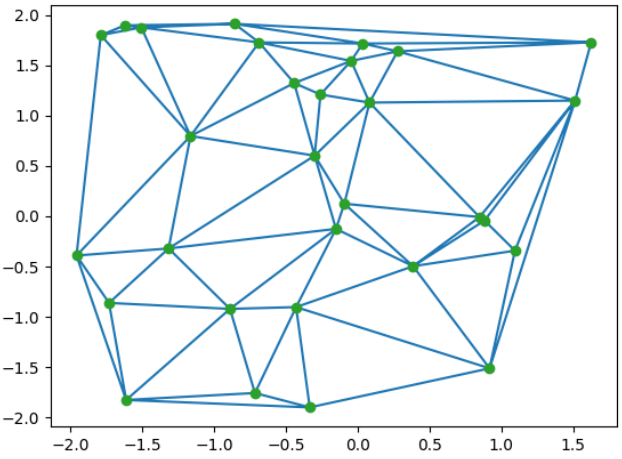
Definition 3.1.7. (Triangulation) (Lai and Schumaker, 2007)
A triangulation $\mathcal{T}$ is a partition of a domain in a set of J non-overlapping simplices:

$$\mathcal{T} \equiv \cup_{i=1}^{J} t_i,\ t_i \cap t_j \in \{\emptyset, \tilde{t}\},\ \forall t_i, t_j \in \mathcal{T},\ i \neq j, \tag{3.20}$$

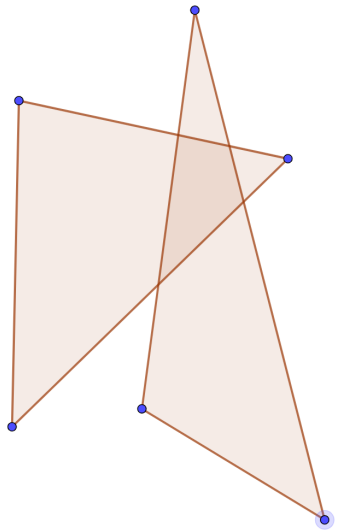
where $t_1, t_2, ..., t_J$ are simplices of dimension n and $\tilde{t}$ is a simplex of dimension less than n. If $\tilde{t}$ has dimension $n-1$, then it is called an *edge facet*.

Figure 3.3 shows an example of a valid and an invalid triangulation.
An extensive overview of different types of triangulations is given by Lai and Schumaker (Lai and Schumaker, 2007).

The space of all spline functions defined on a certain triangulation $\mathcal{T}$ is called a *spline space*.



(a) Valid triangulation



(b) Invalid triangulation

Figure 3.3: Examples of a valid and an invalid triangulation

Definition 3.1.8. (Spline space) (Lai and Schumaker, 2007)
The spline space $\mathcal{S}_d^r(\mathcal{T})$ is the space of all spline functions s of degree d and continuity order $\mathcal{C}^r$ on a triangulation $\mathcal{T}$:

$$\mathcal{S}_d^r(\mathcal{T}) := \{s \in \mathcal{C}^r(\mathcal{T}) : s|_t \in \mathcal{P}_d, \forall t \in \mathcal{T}\}, \tag{3.21}$$

where $\mathcal{P}_d$ denotes the space of all polynomials of degree d.

Lai and Schumaker (2007) present a metric to quantify the approximation power of a multivariate simplex spline space. Also, they show that the approximation power is inversely proportional to the length of the longest edge in the triangulation.

To define a multivariate simplex spline of a certain degree d and continuity order $\mathcal{C}^r$ on a triangulation $\mathcal{T}$ consisting of J simplices, de Visser et al. (2009) introduces a simplex membership operator:

$$\delta_{jk(\mathbf{x})} = \begin{cases} 0 \text{ if } j \neq k(\mathbf{x}) \\ 1 \text{ if } j = k(\mathbf{x}), \end{cases} \quad , \text{ for } j = 1, ..., J, \tag{3.22}$$

where $k(\mathbf{x})$ is the simplex containing point $\mathbf{x}$. This operator allows for the definition of the per-simplex $\hat{d} \times \hat{d}$ diagonal membership matrix:

$$\mathbf{D}_{t_j}(\mathbf{x}) = [(\delta_{j,k(\mathbf{x})})_{q,q}]_{q=1}^{\hat{d}} \in \mathbb{R}^{\hat{d}\times\hat{d}}. \tag{3.23}$$

The full-triangulation membership matrix $\mathbf{D}(\mathbf{x})$ is a block diagonal matrix with blocks $\mathbf{D}_{t_j}(\mathbf{x})$:

$$\mathbf{D}(\mathbf{x}) = [(\mathbf{D}_{t_j}(\mathbf{x}))_{j,j}]_{j=1}^{J} \in \mathbb{R}^{(J\cdot\hat{d})\times(J\cdot\hat{d})}. \tag{3.24}$$

This membership matrix paves the way for a formal definition of the multivariate simplex spline.

Definition 3.1.9. (Multivariate simplex splines) (de Visser, 2011)
A multivariate simplex spline of degree d and continuity order $\mathcal{C}^r$ defined on a triangulation $\mathcal{T}$ consisting of J simplices is given by

$$s_d^r(\mathbf{x}|\mathbf{c}) = \mathbf{B}^d(\mathbf{x})^T \cdot \mathbf{D}(\mathbf{x}) \cdot \mathbf{c},\ \ \mathbf{x} \in \mathbb{R}^n, \tag{3.25}$$

where

$$\mathbf{B}^d(\mathbf{x}) = [\mathbf{B}_{t_1}^d(b(\mathbf{x})), \mathbf{B}_{t_2}^d(b(\mathbf{x})), ..., \mathbf{B}_{t_J}^d(b(\mathbf{x}))]^T \in \mathbb{R}^{(J\cdot\hat{d})\times 1} \tag{3.26}$$

and

$$\mathbf{c} = [\mathbf{c}^{t_1^T}, \mathbf{c}^{t_2^T}, ..., \mathbf{c}^{t_J^T}]^T \in \mathbb{R}^{(J\cdot\hat{d})\times 1}. \tag{3.27}$$

3.1.7 The B-net orientation rule

Expression (3.18) describes the spatial structure of the B-coefficients within a simplex. But since there are $(n+1)!$ different orderings of the vertices, there are as many possible ways to orient the B-net. Hence, extending our view to an entire triangulation requires a systematic way of orienting the B-nets that belong to different simplices. To define such an *orientation rule*, we assign to each vertex a globally unique index. We call the B-coefficients that are located at the vertices *vertex B-coefficients*. Let

$$\{\breve{c}_{(\omega\cdot d)}^t\}_{|\omega|=1} \subset \{c_\kappa^t\}_{|\kappa|=d} \tag{3.28}$$

denote the set of vertex B-coefficients of a B-form polynomial of degree d on a n-simplex t. Sorting these coefficients in a lexicographic way yields the following vector:

$$\breve{\mathbf{c}}^t = [\{\breve{c}^t_{(\omega \cdot d)}\}_{|\omega|=1}] \in \mathbb{R}^{n \times 1}. \tag{3.29}$$

Now, the B-net should be oriented in such a way that the sorting order of a vertex B-coefficient corresponds to the sorting order of its vertex. This system is called *the B-net orientation rule*.

Definition 3.1.10. (The B-net orientation rule) (de Visser, 2011)
Let every n-simplex t in a triangulation $\mathcal{T}$ be described by a tuple

$$\mathbf{V}_t := (\mathbf{v}_{p_0}, \mathbf{v}_{p_1}, ..., \mathbf{v}_{p_n}) \in \mathbb{R}^{n \times k},\ k \geq 0 \tag{3.30}$$

of vertices with

$$p_0 > p_1 > ... > p_n. \tag{3.31}$$

Let the set $\mathbf{V}_{\mathcal{B}}$ be defined as follows:

$$\mathbf{V}_{\mathcal{B}} = \{(c_{d,0,0,...,0}, \mathbf{v}_{p_0}), (c_{0,d,0,...,0}, \mathbf{v}_{p_1}), ..., (c_{0,0,0,...,d}, \mathbf{v}_{p_n})\}. \tag{3.32}$$

The B-net orientation rule states that

$$\{(\breve{\mathbf{c}}^t_i, \mathbf{v}_{p_i})\}_{i=0}^n = \mathbf{V}_{\mathcal{B}},\ \forall t \in \mathcal{T}, \tag{3.33}$$

where $\breve{\mathbf{c}}^t_i$ is the vector of vertex B-coefficients from expression (3.29).

Example 1.
Figure 3.4 shows simplices t_i, t_j and t_k on which polynomials of degree 2 should be defined.

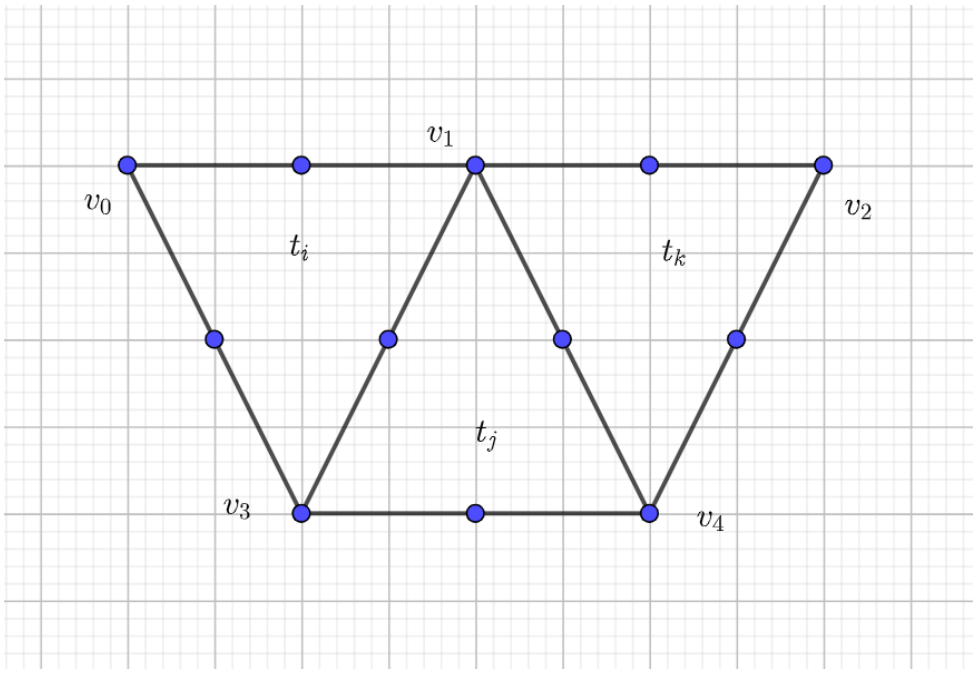


Figure 3.4: Simplices t_i, t_j and t_k.

The ordering of the vertices is as follows:

$$v_0 > v_1 > v_2 > v_3 > v_4. \tag{3.34}$$

For this ordering, the B-net orientation rule states for simplices t_i, t_j, t_k respectively:

$$\begin{aligned}
\{(\breve{\mathbf{c}}_m^{t_i}, \mathbf{v}_{p_m})\}_{m=0}^2 &= \{(c_{2,0,0}^{t_i}, v_0), (c_{0,2,0}^{t_i}, v_1), (c_{0,0,2}^{t_i}, v_3)\} \\
\{(\breve{\mathbf{c}}_m^{t_j}, \mathbf{v}_{p_m})\}_{m=0}^2 &= \{(c_{2,0,0}^{t_j}, v_1), (c_{0,2,0}^{t_j}, v_3), (c_{0,0,2}^{t_j}, v_4)\} \\
\{(\breve{\mathbf{c}}_m^{t_k}, \mathbf{v}_{p_m})\}_{m=0}^2 &= \{(c_{2,0,0}^{t_k}, v_1), (c_{0,2,0}^{t_k}, v_2), (c_{0,0,2}^{t_k}, v_4)\}.
\end{aligned} \tag{3.35}$$

The resulting B-net is shown in Figure 3.5.

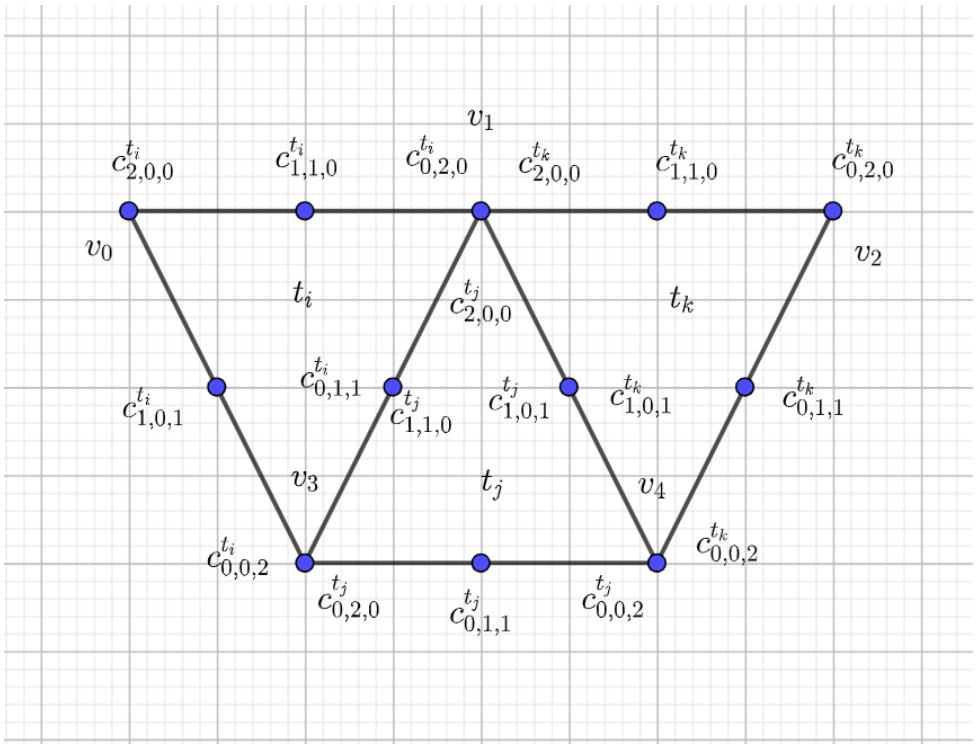


Figure 3.5: B-nets oriented according to the B-net orientation rule.

3.1.8 Smoothness constraints

Since it is a linear combination of continuous functions, the spline function $s^r_d(\mathbf{x}|\mathbf{c})$ is obviously continuous on a single simplex. To attain the desired degree of continuity between different pieces of the spline, we impose so-called *smoothness constraints* on the B-coefficients. To formulate these constraints, the B-coefficients of different polynomial pieces are related through equations called the *continuity conditions*. Recall that an $(n-1)$-dimensional simplex $\tilde{t}$ that is the intersection of two n-dimensional simplices t_i and t_j is called an *edge facet*. Two simplices are called *neighbouring simplices* if they share a (unique) edge facet. Between B-form polynomials p_{t_i} and p_{t_j} defined on two neighbouring simplices t_i and t_j there is said to be *continuity of order* C^r if each of their directional derivatives up to order C^r is equal over their entire edge facet $\tilde{t}_{ij}$, or

$$D^m_{\mathbf{u}}(p_{t_i})(b(\mathbf{x})) = D^m_{\mathbf{u}}(p_{t_j})(b(\mathbf{x})) \ \forall \mathbf{x} \in \tilde{t}_{ij}, \ m = 0, 1, ..., r \tag{3.36}$$

for each directional vector $\mathbf{u} \in \mathbb{R}^n$. The following constraints were formulated by Lai and

Schumaker (2007):

$$c^{t_i}_{(\kappa_0,\kappa_1,\ldots,\kappa_{n-1},m)} = \sum_{|\omega|=m} c^{t_j}_{(\kappa_0,\ldots,\kappa_{n-1},0)+\omega} B^m_\omega(\tilde{\mathbf{v}}_{ij}),\ 0 \le m \le r,\ t_i \ne t_j, \tag{3.37}$$

where $\omega \in \mathbb{R}^{n+1}$ is a multi-index and $\tilde{\mathbf{v}}_{ij}$ is the unique vertex of t_i which is not on the edge facet $\tilde{t}_{ij}$. Such a vertex is called an *out-of-edge* vertex.

As mentioned in Section 3.1.5, the spatial representation of the B-coefficients is very convenient for the formulation of the continuity conditions. It gives rise to a principle called the *structure of continuity*. This principle is illustrated in Figure 3.6, which shows the structures of continuity of order C^0, C^1, C^2 and C^3. For example, the conditions for continuity of order C^2 between simplices t_i and t_j for B-coefficient $c^{t_i}_{022}$ involves the B-coefficients $c^{t_j}_{220}$, $c^{t_j}_{130}$, $c^{t_j}_{040}$, $c^{t_j}_{031}$, $c^{t_j}_{022}$, $c^{t_j}_{121}$. The latter coefficients are called the *continuity body* B-coefficients and the coefficient on the left-hand side of the continuity condition is called the *continuity point* B-coefficient. Together, they form the green diamond-like shape in Figure 3.6.

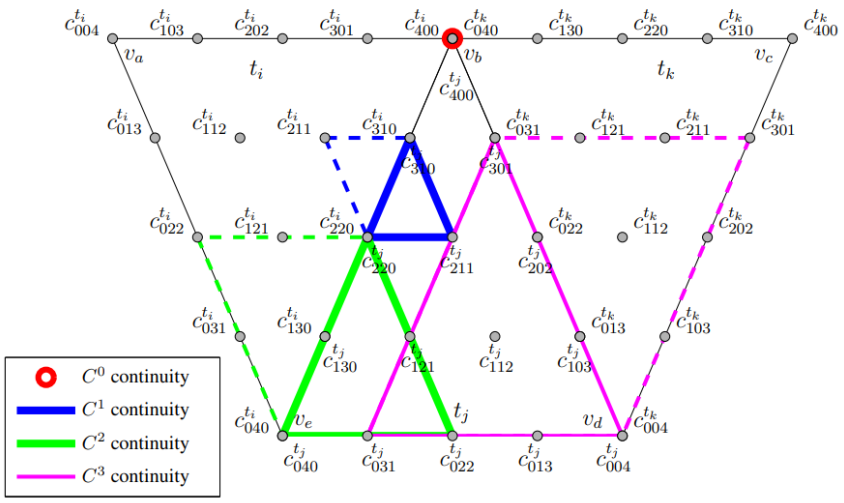


Figure 3.6: Structure of continuity of order C^0, C^1, C^2 and C^3 (de Visser, 2011).

There is one fundamental downside to the formulation of the continuity conditions in expression (3.37): it is only valid if the B-coefficient that has the lowest lexicographic sorting order in the simplex, that is $c_{0,0,\ldots,d}$, is positioned at the out-of-edge vertex. This B-

net orientation is called the *maximum degree symmetric orientation*. Such an orientation however does not exist for every possible triangulation. De Visser (2011) develops a general formulation of the continuity conditions that can be applied to B-nets oriented according to the B-net orientation rule discussed in Section 3.1.7. Figure 3.6 implies that the constant values 0 or m in the multi-indices should be located at the place of the only nonzero value in the multi-index of the B-coefficient of the out-of-edge vertex. Consider for example the condition for continuity of order C^3 at the B-coefficient $c^{t_k}_{3,0,1}$ in Figure 3.6. Since the out-of-edge vertex $\mathbf{v}_c$ has B-coefficient $c^{t_k}_{4,0,0}$, it follows that the value $m = 3$ should be placed at the first position of the multi-index. Similarly, because the B-coefficient $c^{t_j}_{0,4,0}$ at the out-of-edge vertex $\mathbf{v}_e$ has its nonzero element at the second position, the value 0 in multi-indices of the right-hand side B-coefficients should be located at the second place.
In order to formalize this system, de Visser (2011) introduces the following tuple function:

$$\mathcal{M}(j,\kappa) = \begin{bmatrix} (j,\kappa_0,\kappa_1,...,\kappa_{n-1}) \\ (\kappa_0,j,\kappa_1,...,\kappa_{n-1}) \\ \vdots \\ (\kappa_0,\kappa_1,...,j,\kappa_{n-1}) \\ (\kappa_0,\kappa_1,...,\kappa_{n-1},j) \end{bmatrix}, \ |\mathcal{M}_i(j,\kappa)| = d - m + j \tag{3.38}$$

For $j = m$, the function $\mathcal{M}(j,\kappa)$ yields all possible multi-indices for the B-coefficient of the continuity point (the left hand side of expression (3.37)) and for $j = 0$, we obtain all multi-indices for the B-coefficients of the continuity body (the right hand side of expression (3.37)) through $\mathcal{M}(j,\kappa) + \omega$.

To decide upon the right position of the constants m and 0, de Visser (2011) uses a rank function $\rho(\mathbf{v}_i)$ that returns the rank of a vertex $\mathbf{v}_i$ within a certain simplex t based on the global vertex index. This rank function is given by

$$\rho(\mathbf{v}_i) = (n+2) - \sum_{j=0}^{i} k_{\mathbf{v}_j}, \tag{3.39}$$

where

$$k_{\mathbf{v}_j} = \begin{cases} 1 \text{ if } \mathbf{v}_j \in \mathcal{V}_t \\ 0 \text{ if } \mathbf{v}_j \notin \mathcal{V}_t. \end{cases} \tag{3.40}$$

Using this rank function, the tuple function creates multi-indices in the following way:

$$\bar{\kappa} = \mathcal{M}_{\rho(\mathbf{v})}(j, \kappa), \ |\bar{\kappa}| = d - m + j. \tag{3.41}$$

The elements of $\bar{\kappa}$ are determined by the condition $|\bar{\kappa}| = d - m + j$. For two simplices t_f and t_g it outputs tuples of multi-indices for the continuity points and the continuity body:

$$(\mathcal{M}_{\rho(\tilde{\mathbf{v}}_{f,g})}(m, \kappa), \mathcal{M}_{\rho(\tilde{\mathbf{v}}_{g,f})}(0, \kappa)) = (\kappa_{\text{point}}, \kappa_{\text{body}}). \tag{3.42}$$

This is stated more formally in the following theorem:

Theorem 2. (General formulation of continuity conditions) (de Visser, 2011)
The general formulation of the continuity conditions for continuity of order C^r between two n-simplices t_f and t_g with B-nets of degree d which are oriented according to the B-net orientation rule is the following:

$$c^{t_f}_{\mathcal{M}_{\rho(\tilde{\mathbf{v}}_{f,g})}(m,\kappa)} = \sum_{|\omega|=m} c^{t_g}_{(\mathcal{M}_{\rho(\tilde{\mathbf{v}}_{g,f})}(0,\kappa)+\omega)} B^m_\omega(\tilde{\mathbf{v}}_{f,g}), \ 0 \leq m \leq r, \tag{3.43}$$

where $\mathcal{M}_{\rho(\cdot)}(\cdot)$ is the tuple function from expression (3.38), which uses the rank function from equation (3.39).

For examples of the use of the general formulation of the continuity conditions, we refer to de Visser (2011).
The total number of conditions required to achieve continuity of order C^r depends on the dimension and the degree of a spline function. This number was derived by de Visser (2011):

Theorem 3. (Number of continuity conditions) (de Visser, 2011)
Let $s \in \mathcal{S}^r_d(\mathcal{T})$ be an n-variate spline function. The number R_r of continuity conditions per edge of the triangulation required to achieve a continuity order C^r is given by

$$R_r := \sum_{m=0}^{r} \frac{(d-m+n-1)!}{(n-1)!(d-m)!}. \tag{3.44}$$

The continuity conditions in equation (3.43) can be stored into a matrix $\mathbf{H}$ called the *smoothness matrix* and thus written as

$$\mathbf{Hc} = 0. \tag{3.45}$$

Each row of $\mathbf{H}$ contains a single condition written in the following form:

$$-c^{t_f}_{\mathcal{M}_{\rho(\tilde{\mathbf{v}}_{f,g})}(m,\kappa)} + \sum_{|\gamma|=m} c^{t_g}_{(\mathcal{M}_{\rho(\tilde{\mathbf{v}}_{g,f})}(0,\kappa)+\gamma)} B^m_\gamma(\tilde{\mathbf{v}}_{f,g}) = 0,\ 0 \leq m \leq r. \tag{3.46}$$

From Theorem 3, it follows that for continuity order C^r, we have $\mathbf{H} \in \mathbb{R}^{(E\cdot R_r)\times(J\cdot\hat{d})}$, where E is the number of edges of the triangulation. The smoothness matrix generally has a very high sparseness factor and is hardly ever of full rank. This can be attributed to redundant continuity conditions. The fact that these redunancies also occur between conditions of different continuity orders renders it very difficult to construct a smoothness matrix of full rank. De Visser (2011) presents a practical method to create a full (or almost full) rank smoothness matrix. This approach is based on the *condition number* of $\mathbf{H}$. The condition number of a matrix A indicates how accurate an approximate solution to the system $Ax = b$ can be. If this number is close to 1, then the matrix is said to be *well-conditioned* and its inverse can be computed accurately. If the condition number is very large, then the matrix is called *ill-conditioned* and probably almost singular. If the condition number equals infinity, then the matrix is not invertible. To estimate the condition number, we use the estimator proposed by Hager (1984). To construct the full rank smoothness matrix, we first initialize $\mathbf{H} \in \mathbb{R}^{1\times(J\cdot\hat{d})}$ as a matrix with one row containing a single continuity condition. From this matrix, we create a new candidate smoothness matric $\mathbf{H}_c$ by appending a new condition $\mathbf{h}$. This candidate matrix has the following form:

$$\mathbf{H}_c = \begin{bmatrix} \mathbf{H} \\ \mathbf{h} \end{bmatrix} \tag{3.47}$$

Since the estimator of Hager (1984) requires a square matrix and the candidate matrix is in general not square, we multiply it by its transpose and make use of the following basic linear algebra fact to obtain a conservative estimate of the condition number of $\mathbf{H_c}$:

$$\text{rank}\mathbf{H_c} \geq \text{rank}\mathbf{H}_c\mathbf{H}_c^T. \tag{3.48}$$

Following de Visser (2011), we declare $\mathbf{H}_c$ to be singular if its estimated condition number is larger than 10^{10}. If this is the case, than the continuity condition $\mathbf{h}$ is considered redundant and will be dropped. Otherwise, we assume that $\mathbf{h}$ is not redundant and set $\mathbf{H}$ to $\mathbf{H}_c$. This process should be repeated until all continuity conditions have been considered.

A spline function with a high continuity order has a high global smoothness, which improves the numerical stability of the estimator for the B-coefficients. However, a high continuity order goes at the expense of the approximation power of the spline. After all, imposing continuity conditions on the B-coefficients reduces the degrees of freedom of the spline function.

3.2 Multivariate simplex splines as modelling tools

Choosing an appropriate simplex spline function to approximate a set of data points involves selection of a geometric model structure, that is, the triangulation on which the splines are defined, selection of the polynomial model structure, that is, the degrees and continuity orders of the splines, and estimation of suitable B-coefficients. In this section, we discuss each of these components separately.

3.2.1 Geometric model structure selection

3.2.1.1 Triangulations

The approximation power of a simplex spline space depends to a great extent on the triangulation on which it is defined. In this section, we discuss various metrics that are used to assess the quality of a triangulation. Also, we introduce some of the most popular types of triangulations: the Type I/II triangulations and the Delaunay triangulation.

Simplex metrics

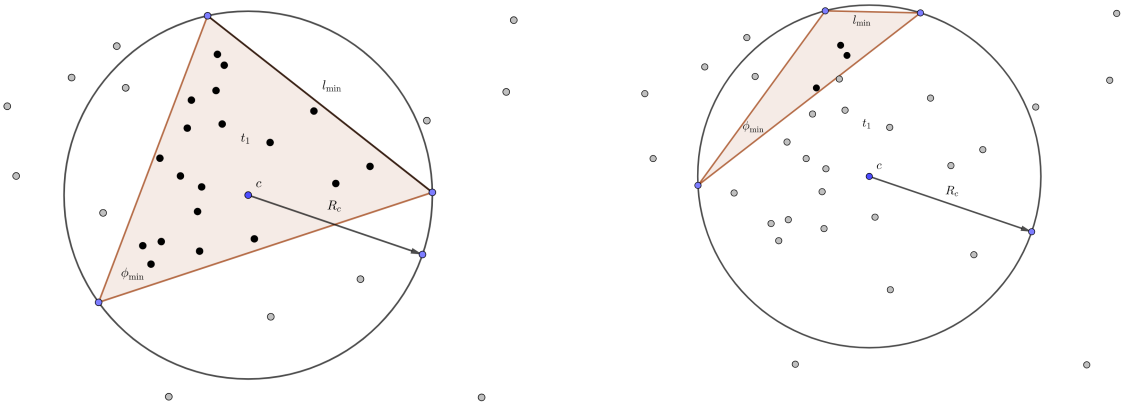
Several different metrics have been designed to assess the quality of a simplex. In this section, we discuss four of them: the location of the center and the radius of the circum-(hyper)sphere of the simplex (SRLC), the ratio between the radius of the circumsphere and the shortest ridge of the simplex (SRSC), the minimum angle between two ridges of the simplex (SMA) and the number of data points contained by the simplex (SDP).

The first three of these metrics were described by Shewchuk (2002) and the final one was introduced by de Visser (2011). Together, these metrics provide a powerful tool to determine the quality of a simplex. Table 3.1 provides some guidelines in interpreting the values of the metrics (de Visser, 2011). Here, c denotes the center of the circumsphere of a simplex t and R_c the radius of this circumsphere and $\hat{d}$ is given by expression (3.8).

Simplex metric	"Good" simplex	"Bad" simplex
SRLC	$c \in t$, R_c small	$c \notin t$, R_c large
SRSC	SRSC < 2	SRSC ≥ 2
SMA	SMA $> 14°$	SMA $\leq 14°$
SDP	SDP $\geq \hat{d}$	SDP $< \hat{d}$

Table 3.1: Guidelines for interpreting the metrics

A type of simplex that is considered "bad" according to these metrics is the so-called *sliver simplex*. Sliver simplices typically have a large circumsphere radius compared to their shortest ridge and a small minimum angle. They cause non-uniformities in the approximation power of a simplex spline, which is not desirable. Figure 3.7 shows an example of a "good" simplex and a sliver simplex.



(a) Good simplex (b) Sliver simplex

Figure 3.7: Examples of a good simplex and a sliver simplex

The SDP has proved to be the most important metric (de Visser, 2011): if only a single simplex in a triangulation has an insufficient SDP, it is in general not possible to find the

B-coefficients of a simplex spline defined on this triangulation.

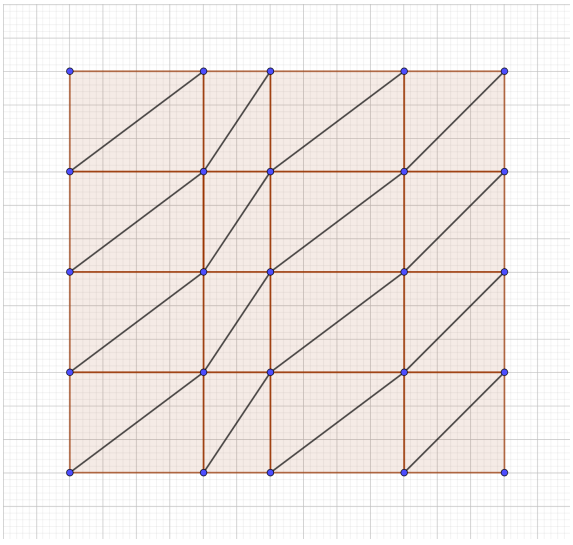
To assess the quality of the entire triangulation, de Visser (2011) introduced the following metric, based on the SRSC:

$$\mathcal{P}_{\mathcal{T}} := \frac{1}{|\mathcal{T}|} \sum_{i=1}^{|\mathcal{T}|} \frac{r_{\theta_i}}{\min_{\mathbf{v}_u, \mathbf{v}_w \in \tilde{t}_i} |\mathbf{v}_u - \mathbf{v}_w|}, \tag{3.49}$$

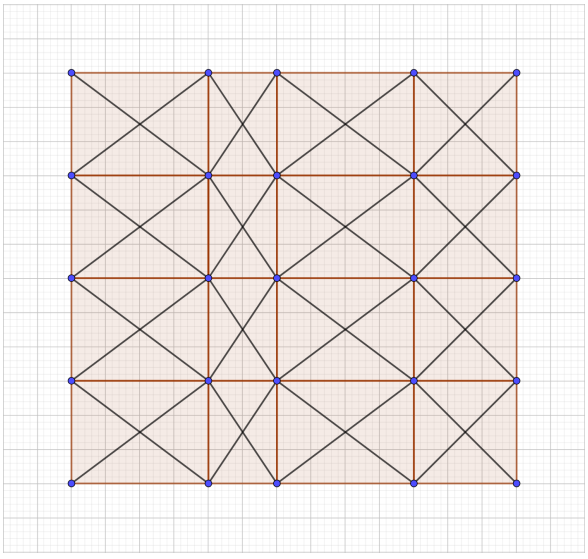
where r_{θ_i} denotes the radius of the circum-hypersphere of simplex $t_i \in \mathcal{T}$, $\tilde{t}_i$ is an edge facet of t_i and $\mathbf{v}_u$ and $\mathbf{v}_w$ are vertices of $\tilde{t}_i$.

Type I/II triangulations

Type I and Type II triangulations are simple types of triangulations with the advantage that their simplices are guaranteed to be "good" according to the metrics discussed in the previous section. These triangulations are constructed by filling each cell of a multi-dimensional grid with a certain symmetric prototype triangulation. In the two-dimensional plane for example, a Type I triangulation is created by drawing one diagonal in each cell of the grid. A Type II triangulation is created by drawing both diagonals. Examples of Type I/II triangulations in two dimensions are shown in Figure 3.8.



(a) Type I triangulation (32 triangles)



(b) Type II triangulation (64 triangles)

Figure 3.8: Type I/II triangulations

(a) Type I triangulation of 3-cube (6 tetrahedrons)

(b) Type II triangulation of 3-cube (12 tetrahedrons)

Figure 3.9: Type I/II triangulations of the 3-cube

In three dimensions, a cell of the three-dimensional grid is a 3-cube. Figure 3.9 shows Type I and Type II triangulations of a 3-cube.
Mara (1976) showed that the number of simplices in a symmetric Type I triangulation equals $n!$. Table 3.2 shows the total number of simplices in symmetric Type I and Type II triangulations of a single n-cube for dimensions 2 up to 8.

n	# of vertices n-cube	# of simplices Type I	# of simplices Type II
2	4	2	4
3	8	6	12
4	16	24	44
5	32	120	210
6	64	720	1236
7	128	5040	8870
8	256	40320	65298

Table 3.2: Number of simplices in symmetric Type I/II triangulations a single n-cube for dimensions 2 up to 8.

In principle, Type I/II triangulations are straightforward and easy to work with. However,

they cause difficulties for dimensions higher than 8. After all, it follows from Table 3.2 that the number of simplices per n-cube grows large for high dimensions. Hence, ensuring a sufficient SDP value requires a large number of data points.

Delaunay triangulations

Another popular triangulation method is the so-called *Delaunay triangulation*. It was designed in 1934 by the Russian mathematician Delaunay (Delaunay, 1934). The major advantage of the Delaunay triangulation is its flexibility. It has been used as a modelling tool in various different fields (Schröder and Robach, 1994; Li et al., 2003; de Berg et al., 2008).

Definition 3.2.1. (Delaunay triangulation)
Let $P \in \mathbb{R}^n$ be a set of points. A triangulation $\mathcal{T}$ of P is called a *Delaunay triangulation* if each point in P does not lie in the interior of the circum-hypersphere of any simplex $t_j \in \mathcal{T}$. This condition is called the *Delaunay condition*.

In general, the Delaunay triangulation of a set of points is unique. Figure 3.10 shows a valid Delaunay triangulation in the two-dimensional plane.

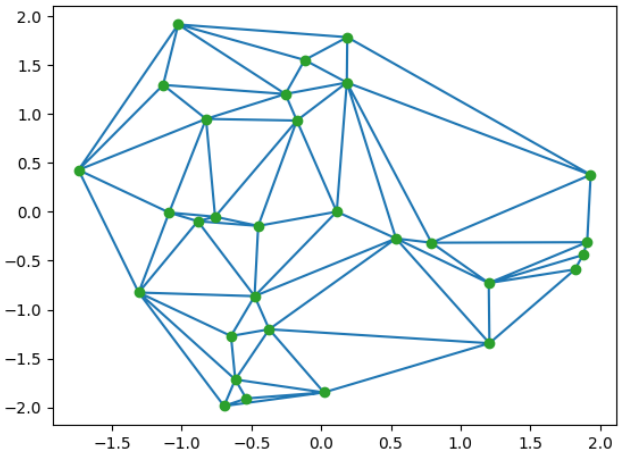


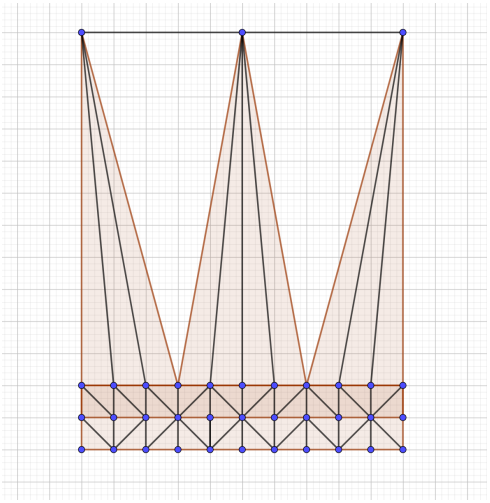
Figure 3.10: Delaunay triangulation in two-dimensional plane

Seidel (1991) showed that a Delaunay triangulation of n points in dimension d consists of $O(n^{\lceil d/2 \rceil})$ simplices.

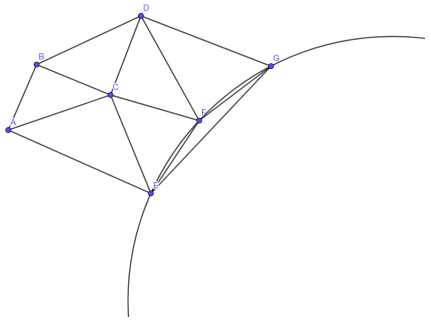
A nice property of the Delaunay triangulation is the following.

Theorem 4. (Sibson, 1978)
Let $P \in \mathbb{R}^n$ be a set of points. The minimum angle of the unique Delaunay triangulation of P is greater than the minimum angle of any other triangulation of P.

The flexibility of the Delaunay triangulation method comes with a downside: it might yield bad simplices in terms of the discussed metrics, especially at the boundaries of the domain. Two types of ugly simplices that frequently occur in Delaunay triangulations are the *simplex fan* and the *sliver simplex*. These are illustrated in Figure 3.11. A simplex fan occurs when there are great differences in the density of the vertices. These differences cause the simplices to have bad SRLC and SRSC values. Figure 3.11b shows a sliver simplex at the boundary of the triangulation domain. The radius of its circumsphere is rather out of proportion.



(a) Simplex fan



(b) Sliver simplex

Figure 3.11: Ugly simplices in Delaunay triangulations

3.2.1.2 Triangulation optimization

Finding the optimal triangulation for a data approximation problem is not a trivial task. It is a two-fold problem consisting of the placement of the vertices and finding a triangulation given a vertex set. Both are large-scale tasks: positioning N vertices in n dimensions is a continuous optimization problem involving $N \cdot n$ variables. Also, the number of possible triangulations for a given vertex set is extremely large. A lower bound on the number

of triangulations in 2 dimensions is proved to be $\Omega(2.33^N)$ (Aichholzer et al., 2004). Sharir and Welzl (2006) derived an upper bound of $O(43^N)$.

Due to the high complexity of the triangulation optimization problem, most researchers settle for a reasonably "good" triangulation instead of the optimal one. Developed methods can be distinguished into geometric optimization methods and data-dependent optimization methods. Geometric methods only focus on the triangulation, creating well-shaped simplices that are generally suitable for data approximation. Data-dependent methods on the other hand, also take into account the data that are to be approximated. In this section we give an overview of the most important advancements in triangulation optimization.

Geometric triangulation optimization methods

Constrained Delaunay triangulation methods Most geometric triangulation optimization methods are based on the so-called *constrained Delaunay triangulation*. Recall from Section 3.2.1.1 that the Delaunay triangulation has the property that the smallest angle is maximized. This property renders the Delaunay triangulation suitable for all kinds of data approximation methods. It ensures that the approximation power is quite evenly distributed over the domain.
A constrained Delaunay triangulation is defined on a planar straight-line graph (PSLG), which is an embedding of a planar graph in the plane in such a way that its edges are mapped to straight line segments. Chew (1989) presents the following definition of a 2-dimensional constrained Delaunay triangulation:

Definition 3.2.2. (Constrained Delaunay triangulation) (Chew, 1989)
A triangulation T is a constrained Delaunay triangulation (CDT) of a PSLG G if each edge of G is an edge of T and for each remaining edge $e \in T \setminus G$ there exists a circle c with the following properties:

- The endpoints of e are on the boundary of c.
- If any vertex v of G is in the interior of c, then it cannot be seen from at least one of the endpoints of e (i.e., if one draws the line segments from v to each endpoint of e then at least one of the line segments crosses an edge of G).

In the context of data approximation, the PSLG is usually taken to be the convex hull of the data set. Since a CDT defined on a PSLG is likely to contain some "ugly" triangles, it is usually improved by means of Delaunay refinement algorithms. Important contributions on this topic were made by Chew (1989), Dey et al. (1991), Mitchell and Vavasis (1992), Ruppert (1995), Si and Gärtner (2005) and Shewchuk (2001, 2008). Although Shewchuk (2008) made a start, there is no version for higher dimensions available yet.

The Hypercube-Convex hull-Intersection method The Hypercube-Convex hull-Intersection (HCI) method (de Visser, 2011) was designed specifically for simplex splines. It first intersects a Type I/II triangulation with the convex hull of the data set. Then, it removes the vertices of the triangulation that are outside the convex hull and retriangulates the area around the boundary with a Delaunay triangulation. Finally, it is possible to perform a post-processing step to ensure that each simplex contains a sufficient amount of data points. The HCI triangulation method seems to produce well-shaped simplices. De Visser (2011) demonstrates this by means of various examples. The B-coefficients are distributed quite evenly over the domain, such that the approximation power is roughly homogeneous. This, together with the fact that each simplex contains enough data points, renders the HCI method very suitable for simplex splines.

Data-dependent triangulation optimization methods

A data-dependent triangulation optimization method would be most suitable for approximation with simplex splines. In this section, we discuss the most important of those methods.

The Local Optimization Procedure (LOP), proposed by Lawson (1977), optimizes a triangulation in 2-dimensional space with respect to a certain cost function that may incorporate both information about the triangulation and about the data points. To accomplish this, it performs edge swaps until it reaches a local minimum of the cost function. The locations of the vertices do not change during the course of the algorithm. Hence, the LOP only optimizes a triangulation with respect to a fixed set of vertices.

Vertex insertion algorithms, such as the one presented by Cohen et al. (2012), also minimize a cost function that takes into account both the data points and the simplex shapes.

These algorithms consecutively insert vertices and update the triangulation in such a way that this cost function is reduced as much as possible. In general, this method will not lead to the globally optimal triangulation. After all, after inserting a vertex, the optimal locations of the other vertices might have changed.

Although it does not entirely solve the triangulation optimization problem, the method designed by de Visser et al. (2012) comes closer than any other algorithm. The authors introduce an interval analogue of simplex splines, called *intersplines*. These are evaluated and manipulated using techniques from *interval arithmetic*. An interval branch-and-bound algorithm designed for interval functions is used to find the optimal B-coefficients and vertex coordinates. This approach requires knowledge of the geometry of the Inter-spline. Since this geometry is so far only determined for univariate or bivariate linear splines with zero order continuity, the intersplines optimization method works only for these particular splines.

Although a method that takes the data points into account would be most suitable for approximation with simplex splines, the development of such methods is still in its infancy. The methods presented in this section do not lead to the globally optimal triangulation and work only for particular types of simplex splines.

Triangulation optimization for value function approximation

The application to Markov decision theory demands a different triangulation optimization approach than the previously discussed methods. After all, rather than closely approximating a set of data points, it is our goal to find an approximation architecture that allows us to learn the best possible policy. We do not aim to find the most accurate approximation of the value function, but the approximation that has the highest ability to imitate the role of the true value function in the decision-making process. Moreover, in approximating the value function, we generally do not have any information on the true value function. We can only improve a certain current approximation by using this approximation itself and information derived from observing the process. In Section 4.1 we present a new triangulation optimization procedure that is specifically tailored to value function approximation in Markov decision theory.

3.2.2 Polynomial model structure selection

As we discussed in Section 3.2.1, selecting a suitable geometric model structure is a challenging endeavour. Compared to this, polynomial model structure selection is rather straightforward. Given a triangulation, the simplex spline polynomials are completely determined by their degrees and continuity orders. Choosing the polynomial degree implies making a tradeoff between high approximation power and computational efficiency. Cubic splines seem to be the most popular option in applications.

The choice of continuity order also involves a tradeoff, in this case between high approximation power and numerical stability. Recall from Section 3.1.8 that a high continuity order provides a spline function with a high global smoothness, which results in an increased numerical stability of the estimator for the B-coefficients. However, a high continuity order also goes at the expense of the approximation power of the spline. In Chapter 5, we will investigate different choices of the degree and continuity order for the inverted pendulum problem.

3.2.3 Estimation of the B-coefficients

After selecting a suitable triangulation and polynomial model structure, the ultimate form of the simplex spline is determined by its set of B-coefficients. These coefficients should be chosen in such a way that the resulting simplex spline approximates a given set of data points most accurately. Since a simplex spline is linear in its B-coefficients, the most straightforward way to accomplish this is the *discrete least squares method*.

3.2.3.1 Discrete least squares

Suppose a data set consists of N points $(\mathbf{X}, \mathbf{y})$. These data points are to be approximated with a simplex spline

$$s_d^r(\mathbf{x}|\mathbf{c}) = \mathbf{B}^d(\mathbf{x})^T \cdot \mathbf{D}(\mathbf{x}) \cdot \mathbf{c}. \tag{3.50}$$

Let

$$\mathbf{W}(\mathbf{x}) = \mathbf{B}^d(\mathbf{x})^T \cdot \mathbf{D}(\mathbf{x}). \tag{3.51}$$

The cost function for the discrete least squares method is given by

$$J_{\text{DLS}}(\mathbf{c}) = \sum_{i=1}^{N} (\mathbf{W}(\mathbf{x}_i)\mathbf{c} - y_i)^2, \tag{3.52}$$

where $\mathbf{x}_i$ is the ith column of $\mathbf{X}$. Incorporating the continuity constraints yields the following minimization problem:

$$\min_{\mathbf{c}} J_{\text{DLS}}(\mathbf{c}), \text{ subject to } \mathbf{Hc} = 0. \tag{3.53}$$

Let the matrix $\mathbf{W}$ be defined as

$$\mathbf{W} = [\mathbf{W}(\mathbf{x}_1)^T, \mathbf{W}(\mathbf{x}_2)^T, ..., \mathbf{W}(\mathbf{x}_N)^T]^T \in \mathbb{R}^{N \times (J \cdot \hat{d})}. \tag{3.54}$$

Using the method of Lagrange multipliers, the minimization problem in (3.53) can be written as the following Karush-Kuhn-Tucker (KKT) system:

$$\begin{bmatrix} \mathbf{W}^T\mathbf{W} & \mathbf{H}^T \\ \mathbf{H} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{W}^T\mathbf{y} \\ 0 \end{bmatrix} \tag{3.55}$$

Rao (1973) showed that the solution to this system is given by

$$\begin{bmatrix} \mathbf{c} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{W}^T\mathbf{W} & \mathbf{H}^T \\ \mathbf{H} & 0 \end{bmatrix}^+ \begin{bmatrix} \mathbf{W}^T\mathbf{y} \\ 0 \end{bmatrix}, \tag{3.56}$$

where

$$\begin{bmatrix} \mathbf{W}^T\mathbf{W} & \mathbf{H}^T \\ \mathbf{H} & 0 \end{bmatrix}^+ \tag{3.57}$$

is de pseudoinverse of the matrix

$$\begin{bmatrix} \mathbf{W}^T\mathbf{W} & \mathbf{H}^T \\ \mathbf{H} & 0 \end{bmatrix}. \tag{3.58}$$

3.2.3.2 Generalized least squares

De Visser et al. (2009) present a more accurate way to estimate the B-coefficients, based on the *generalized least squares* (GLS) method. Suppose the observations $(\mathbf{X}, \mathbf{y})$ are related as follows:

$$\mathbf{y}_i = s_d^r(\mathbf{x}_i|\mathbf{c}) + \mathbf{r}_i,\ i = 1, 2, ..., N, \tag{3.59}$$

where $\mathbf{r}$ is a vector of residual terms and $\mathbf{x}_i$ is the ith column of $\mathbf{X}$. This expression can be rewritten as

$$\mathbf{Y} = \mathbf{W}\mathbf{c} + \mathbf{r} \in \mathbb{R}^{N\times 1}, \tag{3.60}$$

where $\mathbf{W}$ is the matrix defined in (3.54). The estimator of the Visser et al. (2009) relies upon the following assumptions on the residual vector:

$$\mathbb{E}(\mathbf{r}) = 0 \text{ and } \mathrm{Cov}(\mathbf{r}) = \mathbf{\Sigma}, \tag{3.61}$$

where $\mathbf{\Sigma} \in \mathbb{R}^{N\times N}$ is the so-called *residual covariance matrix*. We assume that this matrix is both nonsingular and positive definite. De Visser et al. (2009) make use of the following generalized least squares cost function:

$$J_{\mathrm{GLS}}(\mathbf{c}) = \frac{1}{2}(\mathbf{Y} - \mathbf{W}\mathbf{c})^T\mathbf{\Sigma}^{-1}(\mathbf{Y} - \mathbf{W}\mathbf{c}). \tag{3.62}$$

Incorporating the continuity conditions results in the following equality constrained GLS optimization problem:

$$\min_{\mathbf{c}}\ J_{\mathrm{GLS}}(\mathbf{c}),\ \text{subject to } \mathbf{H}\mathbf{c} = 0. \tag{3.63}$$

Like in the discrete least squares approach, we again use Lagrange multipliers to formulate the problem as a Karush-Kuhn-Tucker system:

$$\begin{bmatrix} \mathbf{W}^T\mathbf{\Sigma}^{-1}\mathbf{W} & \mathbf{H}^T \\ \mathbf{H}^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{W}^T\mathbf{\Sigma}^{-1}\mathbf{y} \\ 0 \end{bmatrix}. \tag{3.64}$$

This system has the following solution (Rao, 1973):

$$\begin{bmatrix} \mathbf{c} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{W}^T\mathbf{\Sigma}^{-1}\mathbf{W} & \mathbf{H}^T \\ \mathbf{H} & 0 \end{bmatrix}^+ \begin{bmatrix} \mathbf{W}^T\mathbf{\Sigma}^{-1}\mathbf{y} \\ 0 \end{bmatrix}. \tag{3.65}$$

De Visser et al. (2009) show that the matrix

$$\begin{bmatrix} \mathbf{W}^T\mathbf{\Sigma}^{-1}\mathbf{W} & \mathbf{H}^T \\ \mathbf{H} & 0 \end{bmatrix} \tag{3.66}$$

is invertible if each simplex contains a sufficient amount of data points and if the smoothness matrix $\mathbf{H}$ is of full rank.
Finding the residual covariance matrix $\mathbf{\Sigma}$ is not a trivial task. De Visser et al. (2009) propose the following two-stage method: first of all, assume that

$$\mathbf{\Sigma} = \sigma \mathbf{I} \tag{3.67}$$

for some positive scalar σ. In this case, the system (3.64) reduces to the discrete least squares system (3.55). Then, expression (3.60) is used to compute the residual vector $\mathbf{r}$. Using this residual vector, de Visser et al. (2009) estimate per-simplex blocks $\mathbf{\Sigma}_{\boldsymbol{t_j}} \in \mathbb{R}^{N_j \times N_j}$ of the residual covariance matrix in the following way:

$$\mathbf{\Sigma}_{t_j}(k,l) = \frac{1}{N_j} \sum_{i=1+l}^{N_j-k} r_{t_j}(i-l) r_{t_j}(i-l+k), \;\; k,l = 1,2,...,N_j \tag{3.68}$$

for $j = 1,...,J$. Together, under the assumption that each is nonsingular and positive definite, these per-simplex blocks form the full-triangulation, block diagonal, residual covariance matrix:

$$\mathbf{\Sigma} = [(\mathbf{\Sigma}_{t_j})_{j,j}]_{j=1}^{J} \in \mathbb{R}^{N \times N}. \tag{3.69}$$

Finally, new B-coefficients are estimated using the obtained matrix $\mathbf{\Sigma}$ and expression (3.65).

3.2.3.3 Recursive least squares

The linearity in the B-coefficients is a convenient property of simplex splines which allows for the use of linear regression techniques as the ones described in Sections 3.2.3.1 and 3.2.3.2. However, the subjectivity of the B-coefficients to the continuity constraints sabotages the use of the recursive least squares procedure as given by expression (2.23). To enable online adaptation of the simplex spline model, de Visser et al. (2011) developed an equality constrained recursive least squares (ECRLS) estimator for the B-coefficients. This estimator allows for efficient adaptation of the model upon the arrival of a new datapoint, circumventing the need for the time-consuming matrix inversions necessary for the previously discussed least squares methods. In order to integrate simplex spline approximations into the RLS API algorithm, we replace the recursive least squares equations in

expression (2.23) with this equality constrained recursive least squares procedure of de Visser et al. (2011).

Consider again a data set $(\mathbf{X}, \mathbf{y})$ consisting of N points. Let the matrix $\mathbf{W}$ contain the B-forms of the data points as defined in expression (3.54). Let $J_{\text{ECRLS}}(\mathbf{c})$ be the least squares cost function

$$J_{\text{ECRLS}}(\mathbf{c}) = (\mathbf{y} - \mathbf{X}\mathbf{c})^T(\mathbf{y} - \mathbf{X}\mathbf{c}) \tag{3.70}$$

of the B-coefficients $\mathbf{c} \in \mathbb{R}^{J \cdot \hat{d} \times 1}$. Including the smoothness constraints, we obtain the following equality constrained least squares problem:

$$\min_{\mathbf{c}} \; J_{\text{ECRLS}}(\mathbf{c}), \text{ subject to } \mathbf{H}\mathbf{c} = 0. \tag{3.71}$$

Instead of the Lagrange multiplier method, which was used to obtain the discrete and generalized least squares solutions to this problem, the recursive least squares estimator is obtained by solving (3.71) with the null-space method of Lawson and Hanson (1995). This yields the following estimator $\hat{\mathbf{c}}$ of the B-coefficients:

$$\hat{\mathbf{c}} = (\mathbf{W}\mathbf{Z})^+\mathbf{y}, \tag{3.72}$$

where $\mathbf{Z}$ denotes an orthogonal projector onto the null-space of $\mathbf{H}$ and is given by

$$\mathbf{Z} = \mathbf{I} - \mathbf{H}^+\mathbf{H}. \tag{3.73}$$

De Visser et al. (2011) define the *parameter covariance matrix estimate* as

$$\mathbf{P} = (\mathbf{Z}\mathbf{X}^T\mathbf{X}\mathbf{Z})^+ \in \mathbb{R}^{J \cdot \hat{d} \times J \cdot \hat{d}}. \tag{3.74}$$

Using properties of the Moore-Penrose pseudoinverse and the fact that $\mathbf{Z}$ is symmetric, expression (3.72) can be rewritten as

$$\hat{\mathbf{c}} = \mathbf{P}\mathbf{X}^T\mathbf{y}. \tag{3.75}$$

From (3.75), de Visser et al. (2011) derive the following ECRLS estimator for the B-coefficient vector:

$$\begin{aligned}
\mathbf{L}(t+1) &= \mathbf{P}(t)(\mathbf{W}(\mathbf{x}(t+1)))^T[1+\mathbf{W}(\mathbf{x}(t+1))\mathbf{P}(t)(\mathbf{W}(\mathbf{x}(t+1)))^T]^{-1}, \\
\mathbf{P}(t+1) &= \mathbf{P}(t) - \mathbf{L}(t+1)\mathbf{W}(\mathbf{x}(t+1))\mathbf{P}(t), \\
\hat{\mathbf{c}}(t+1) &= \hat{\mathbf{c}}(t) + \mathbf{L}(t+1)[\mathbf{y}(t+1) - \mathbf{W}(\mathbf{x}(t+1))\hat{\mathbf{c}}(t)],
\end{aligned} \tag{3.76}$$

where $\mathbf{x}(t+1)$ represents the new datapoint. The parameter covariance matrix and the vector of B-coefficients are initialized as

$$\mathbf{P}(0) = (\mathbf{Z}\mathbf{W}^T(0)\mathbf{W}(0)\mathbf{Z})^T \tag{3.77}$$

and

$$\hat{\mathbf{c}}(0) = \mathbf{P}(0)\mathbf{W}^T(0)\mathbf{y}, \tag{3.78}$$

where $\mathbf{W}(0)$ contains the initial datapoints.

Chapter 4

Approximate dynamic programming with adaptive multivariate simplex splines

In this chapter, we develop a method that uses simplex spline approximation methods to approximate the value function of a Markov decision process with a continuous state space and a finite, discrete action space. First of all, we introduce a procedure that refines the triangulation on which simplex splines are defined in regions of the state space that demand a more accurate value function approximation. Then, we integrate this procedure firstly into the backward dynamic programming algorithm and then into the RLS API algorithm discussed in Section 2.2.4.

4.1 The triangulation refinement procedure

The triangulation on which it is defined has a great influence on the approximation power of a spline space. Theoretically, we could construct an infinitely fine triangulation that would yield the most accurate approximation of the value function. This, however, does not solve the curse of dimensionality. In choosing the triangulation, we must find the optimal balance between approximation power and computational complexity. Certain regions of the state space might demand a higher accuracy and thus a finer triangulation, whereas in other parts, the triangulation can be coarser. In this section, we present a method that adaptively refines a triangulation during the course of an approximate dy-

namic programming algorithm.

We build our triangulation on a partition of the state space into a set of local grids. The vertices of this triangulation are chosen to be the vertices of the grid cells together with the midpoints of the grid cells. We triangulate this vertex set with a Delaunay triangulation.
To refine the triangulation, we iteratively refine the underlying grids. Each iteration, we give each of the grid cells a score based on several *refinement criteria*. Then, we select the cell with the highest score and choose a *refinement direction*, which is simply one of the n dimensions. We define a refinement along a certain direction as follows:

Definition 4.1.1. (Refinement along a direction)
Let $\mathbf{V}$ denote a set of vertices that constitutes a partition of an n-dimensional hyperrectangle $\mathcal{H}$ into a set of local grids. Let the corresponding grid cells be denoted by $a_1, a_2, ..., a_K$. Each grid cell is the convex hull of 2^n vertices in the set $\mathbf{V}$. Suppose a grid cell a_i is the convex hull of the vertices in the set $\{x_{11}, x_{12}\} \times \{x_{21}, x_{22}\} \times ... \times \{x_{n1}, x_{n2}\}$, where $x_{k1}, x_{k2} \in \mathbf{V}$ for each $k = 1, 2, ..., n$. *Refining a_i along a direction j* means adding a set of 2^{n-1} new vertices $\mathbf{V}_{\text{new}} = \{x_{11}, x_{12}\} \times \{x_{21}, x_{22}\} \times ... \times \{\frac{x_{j1} + x_{j2}}{2}\} \times ... \times \{x_{n1}, x_{n2}\}$ to $\mathbf{V}$ to obtain a new set of vertices

$$\mathbf{V}' = \mathbf{V} \cup \mathbf{V}_{\text{new}}, \tag{4.1}$$

which constitutes a new partition of the hyperrectangle $\mathcal{H}$ into local grids.

Figure 4.1 illustrates the refinement of the cell constituted by the vertex set $\{1, 2\} \times \{1, 2\}$ of the grid defined by $\{0, 1, 2\} \times \{0, 1, 2\}$ along direction 1. We thus obtain the new grids constituted by the vertices $\{0, 1\} \times \{0, 1, 2\} \cup \{1, 2\} \times \{0, 1\} \cup \{1, 1.5, 2\} \times \{1, 2\}$.
After refining the selected grid cell, we remove the midpoint of the original grid cell and add the midpoints of each of the new grid cells to the vertex set. Then, we again triangulate the result with a Delaunay triangulation. The triangulations before and after the refinement shown in Figure 4.1 are depicted in Figure 4.2.

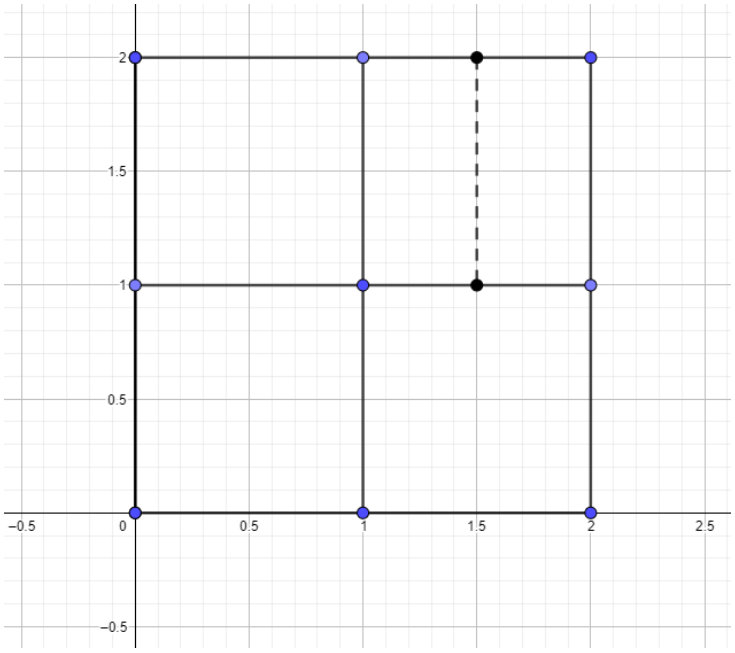
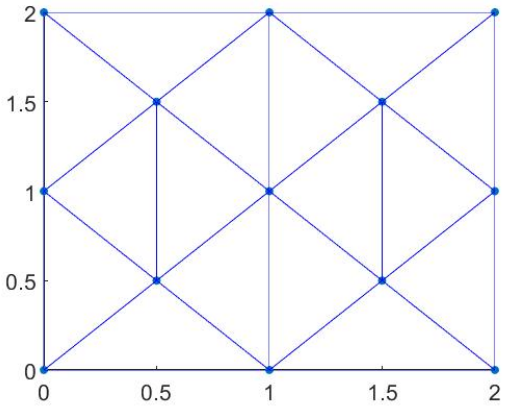
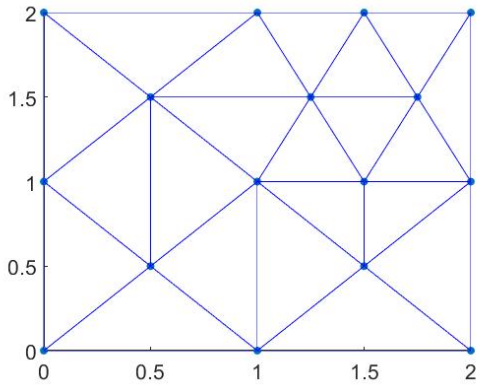


Figure 4.1: Refining the cell $\{1, 2\} \times \{1, 2\}$ along direction 1



(a) Triangulation before refinement



(b) Triangulation after refinement

Figure 4.2: Triangulations before and after refinement

4.1.1 Refinement criteria

In order to decide which cell to refine, we assess each cell according to the following *refinement criteria*:

- Maximum value difference.
- Value variance.
- Visitation frequency.
- Error contribution.

The remainder of this section explains each of these criteria in detail. For each of the criteria, we make a distinction between finite and infinite horizon problems.

4.1.1.1 Maximum value difference and value variance

Regions of the state space on which the value function highly fluctuates or has discontinuities demand for a finer triangulation. We identify such regions using the criteria *maximum value difference* and *value variance*.

Maximum value difference and value variance for finite horizon problems

Let $(\mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{C}, \gamma, \mathcal{D})$ be a finite horizon MDP. For each $t = 0, 1, ..., T$, let $\{a_1^t, a_2^t, ..., a_{N_t}^t\}$ denote a partition of the state space, where a_i^t is a Lebesgue-measurable set for each $i = 1, ..., N_t$. Let $\hat{V}_t$, $t = 0, 1, ..., T$ denote an approximation of the value function.

Definition 4.1.2. (Maximum value difference and value variance for finite horizon problems)
The maximum value difference $\eta(a_i^t)$ and the value variance $\zeta(a_i^t)$ of region a_i^t, $i = 1, 2, ..., N_t$ for value function approximation $\hat{V}_t$ are defined as follows:

$$\eta(a_i^t) = \max_{s_j^t, s_k^t \in a_i^t} |\mathcal{M}\hat{V}(s_j^t) - \mathcal{M}\hat{V}(s_k^t)| \tag{4.2}$$

and

$$\zeta(a_i^t) = \mathrm{Var}(\{\mathcal{M}(s_j^t) | s_j^t \in a_i^t)\}), \tag{4.3}$$

where $\mathcal{M}$ is the Bellman operator of equation (2.7).

To estimate the maximum value difference and value variance of a region a_i^t, we randomly pick M points $s_1^t, s_2^t, ..., s_M^t \in a_i^t$. Let $\hat{V}_t$, $t = 0, 1, ..., T$ again be an approximation of the value function. The estimates are given by

$$\hat{\eta}(a_i^t) = \max_{i,j \in 1,...,M} |\mathcal{M}\hat{V}(s_i^t) - \mathcal{M}\hat{V}(s_j^t)| \tag{4.4}$$

and

$$\hat{\zeta}(a_i^t) = \mathrm{Var}(\{\mathcal{M}\hat{V}_t(s_1^t), \mathcal{M}\hat{V}_t(s_2^t), ..., \mathcal{M}\hat{V}_t(s_M^t)\}). \tag{4.5}$$

Maximum value difference and value variance for infinite horizon problems

For infinite horizon problems, we use the same regions for each time $t = 0, 1, 2, ...$ Hence, we omit the superscripts t in the definitions of maximum value difference and value variance. Apart from that, the definitions remain the same.

4.1.1.2 Visitation frequency

If a part of the state space is hardly ever reached during the course of the process, refining the triangulation in this area would be a waste of computational effort. On the other hand, if a region is frequently visited, errors in the value function approximation will have great effect on the approximations in other regions. Hence, it will probably be worthwile to refine this region and thus achieve a more accurate approximation. We let the *visitation frequency* of a region be a measure of the amount of time the process spends in this particular region.

Visitation frequency for finite horizon problems

Let $(\mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{C}, \gamma, \mathcal{D})$ be a finite horizon Markov decision process. For each $t = 0, 1, ..., T$, let $\{a_1^t, a_2^t, ..., a_{N_t}^t\}$ denote a partition of the state space, where a_i^t is a Lebesgue-measurable set for each $i = 1, ..., N_t$. Let $\hat{V}_t$, $t = 0, 1, ..., T$ denote an approximation of the value function. For finite horizon processes, we define the visitation frequency of a region as the expected number of visits to that region. If we follow the greedy policy with respect to this value function approximation, note that a region a_i^t is either visited or not during

the course of the process. Thus, the visitation frequency $\theta(a_i^t)$, or the expected number of visits to region a_i^t, for this approximation is given by

$$\theta(a_i^t) = 0 \cdot P(a_i^t \text{ is not visited}) + 1 \cdot P(a_i^t \text{ is visited}) = P(a_i^t \text{ is visited}). \tag{4.6}$$

The probability that a region is visited is easily calculated. First of all, we define the *region transition probabilities*.

Definition 4.1.3. (Region transition probabilities)
The *region transition probability* $P^\pi(a_j^{t+1}|a_i^t)$ is defined as the probability that the process makes a transition from region a_i^t to region a_j^{t+1} under a certain policy π.

Definition 4.1.4. (Visitation frequency for finite horizon problems)
The visitation frequency $\theta(a_i^t)$ of a region a_i^t, $t = 0, 1, ..., T$ corresponding to value function approximation $\hat{V}_t$ is defined recursively as

$$\theta(a_i^0) = P(s_0 \in a_i^0), \tag{4.7}$$

or the probability that the initial state lies in a_i^0, and

$$\theta(a_i^t) = \sum_{l=1}^{N_{t-1}} P^\pi(a_i^t|a_l^{t-1})\theta(a_l^{t-1}) \text{ for } t = 1, ..., T, \tag{4.8}$$

where π is the greedy policy with respect to $\hat{V}_t$.

Visitation frequency for infinite horizon problems

Let $(\mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{C}, \gamma, \mathcal{D})$ be an infinite horizon MDP. For infinite horizon MDPs, we define the visitation frequency of a region as the expected proportion of time the process spends in this region. Let $\{a_1, a_2, ..., a_N\}$ denote a partition of the state space, where a_i is a Lebesgue-measurable set for each $i = 1, ..., N$. The region transition probabilities are defined as in Definition 4.1.3 with the superscripts t omitted. Hence, the probability that the process makes a transition from region a_i to region a_j under a policy π is given by $P^\pi(a_j|a_i)$. Observe that the regions and the region transition probabilities do not constitute a Markov chain. To approximate the expected number of visits to each of the regions, we define a Markov chain $\{A_i, i \geq 1\}$ with states $a_1, a_2, ..., a_N$ and transition

probabilities $P^\pi(a_j|a_i)$. The expected proportion of time this Markov chain spends in each of its states is given by its steady state distribution. Let A^π denote the matrix of region transition probabilities induced by policy π, that is

$$A^\pi = \begin{bmatrix} P^\pi(a_1|a_1) & P^\pi(a_2|a_1) & \dots & P^\pi(a_N|a_1) \\ P^\pi(a_1|a_2) & P^\pi(a_2|a_2) & \dots & P^\pi(a_N|a_2) \\ \vdots & \vdots & \ddots & \dots \\ P^\pi(a_1|a_N) & P^\pi(a_2|a_N) & \dots & P^\pi(a_N|a_N) \end{bmatrix} \tag{4.9}$$

If the Markov chain we defined is ergodic, there exists a unique steady state distribution $\boldsymbol{\theta}$, which is the probabilistic eigenvector of A^π. In many real-life situations, however, this Markov chain will not be ergodic. If this is the case, we slightly alter the chain by assuming that it randomly jumps to some other state with a small probability δ, a trick that is also used in the well-known PageRank algorithm (Brin and Page, 1998). The transition probability matrix of this altered Markov chain is given by

$$A'^\pi = (1-\delta)A^\pi + \delta C, \tag{4.10}$$

where

$$C = \frac{1}{N}\begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{bmatrix}. \tag{4.11}$$

It is easily shown that the altered Markov chain with transition probability matrix A'^π is in fact ergodic.

We approximate the expected proportion of time the MDP spends in a region a_i under policy π with the expected proportion of time the Markov chain $\{A_i, i \geq n\}$ spends in state a_i.

Definition 4.1.5. (Visitation frequency for infinite horizon problems)
Let π be a policy. Let A^π be the matrix of region transition probabilties as given by expression (4.9). Define a Markov chain $\{A_i, i \geq 1\}$ with states $a_1, a_2, ..., a_N$ and transition probability matrix A^π. If this Markov chain is ergodic, then the visitation frequency $\theta(a_i)$ of a region a_i is the ith element of the unique steady state distribution $\boldsymbol{\theta}$ of the Markov chain, which is the unique probabilistic vector that satisfies

$$A^{\pi}\boldsymbol{\theta} = \boldsymbol{\theta}. \tag{4.12}$$

If the Markov chain is not ergodic, we define the visitation frequency vector as the unique steady state distribution $\boldsymbol{\theta}$ of the altered Markov chain with transition probability matrix A'^{π} given by expression (4.10).

4.1.1.3 Error contribution

If the value function approximation in a certain region deviates a lot from the true value function, this can be ascribed to two reasons. First of all, the simplex splines on this region could fail to yield an accurate model due to an unsuitable triangulation. In this case, refining the triangulation in this area will probably reduce the approximation error. Secondly, the poorness of the approximation in this region might be a consequence of errors that propagated from other parts of the state space. The *error contribution* criterion estimates to what extent a certain region of the state space contributes to overall approximation errors. In defining this criterion, we again make a distinction between problems with a finite and problems with an infinite horizon.

Error contribution for finite horizon problems

Let $(\mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{C}, \gamma, \mathcal{D})$ be a finite horizon Markov decision process. For each $t = 0, 1, ..., T$, let $\{a_1^t, a_2^t, ..., a_{N_t}^t\}$ denote a partition of the state space, where a_i^t is a Lebesgue-measurable set for each $i = 1, ..., N_t$.
Recall that the optimal value function $V_t^*, t = 0, 1, ..., T$ satisfies

$$V_{t+1}^* = \mathcal{M}(V_t^*), \tag{4.13}$$

where $\mathcal{M}$ denotes the Bellman operator. For a certain approximation $\hat{V}_t$ of the value function, we define the *error estimate* $\varepsilon(s)$ in a state $s \in \mathcal{S}$ as

$$\varepsilon(s) = |\mathcal{M}(\hat{V}_t(s)) - \hat{V}_{t+1}(s)|. \tag{4.14}$$

Let the *local error estimate* of a region a_i^t be defined as the mean error estimate of this region, that is

$$\varepsilon(a_i^t) = \frac{1}{\lambda(a_i^t)} \int_{a_i^t} \varepsilon(s) ds, \tag{4.15}$$

where $\lambda(a_i^t)$ denotes the Lebesgue measure of the region a_i^t. To reduce computational complexity, we estimate the local error estimate by randomly choosing a number of states $s_1, s_2, ..., s_M$ in region a_i^t and taking the average of the error estimates in these states:

$$\hat{\varepsilon}(a_i^t) = \frac{1}{M} \sum_{i=1}^{M} \varepsilon(s_i). \tag{4.16}$$

To measure the extent to which errors in the value function approximation at a certain region of the state space affect the approximation at another part of the state space, we introduce the notion of *influence*. In order to formally define the influence, we present the *k-step region transition probabilities*.

Definition 4.1.6. (k-step region transition probabilities)
Let the probability that the system makes a transition from a_i^t to a_j^{t+1} under policy π be denoted by $P^\pi(a_j^{t+1}|a_i^t)$. The k-step region transition probabilities $p_k^\pi(a_j^{t+k}|a_i^t)$ are defined recursively as follows:

$$p_0^\pi(a_j^t|a_i^t) = \begin{cases} 1 \text{ if } a_i^t = a_j^t \\ 0 \text{ else} \end{cases} \quad \text{for } t = 0, 1, ..., T, \tag{4.17}$$

$$p_1^\pi(a_j^{t+1}|a_i^t) = P^\pi(a_j^{t+1}|a_i^t) \text{ for } t = 0, 1, ..., T \tag{4.18}$$

and

$$p_k^\pi(a_j^{t+k}|a_i^t) = \sum_{l=1}^{N_{t+k-1}} p_1^\pi(a_j^{t+k}|a_l^{t+k-1}) p_{k-1}^\pi(a_l^{t+k-1}|a_i^t) \text{ for } t = 0, 1, ..., T-k. \tag{4.19}$$

Now, we define the influence of a region a_i^{t+k} on a region a_j^t as the expected number of visits to a_i^{t+k} from a_j^t using discounted k-step region transition probabilities $\gamma^k p_k^\pi(a_i^{t+k}|a_j^t)$, where $0 < \gamma < 1$ is the discount factor.

Definition 4.1.7. (Influence for finite horizon problems)
Let π be a policy. The *influence* $\xi^\pi(a_i^{t+k}, a_j^t)$ of region a_i^{t+k} on region a_j^t under policy π is defined as

$$\xi^{\pi}(a_i^{t+k}, a_j^t) = \gamma^k p_k^{\pi}(a_i^{t+k}|a_j^t). \tag{4.20}$$

The influences can be easily computed in a recursive way.

Theorem 5. (Recursive definition of influence for finite horizon problems)
For any regions a_i^{t+k} and a_j^t, $t = 0, 1, ..., T-k$, $k > 0$ and any policy π, we have

$$\xi^{\pi}(a_i^{t+k}, a_j^t) = \sum_{n=1}^{N_{t+k-1}} \gamma p_1^{\pi}(a_i^{t+k}|a_n^{t+k-1}) \cdot \xi^{\pi}(a_n^{t+k-1}, a_j^t) \text{ for } t = 0, 1, ..., T-k. \tag{4.21}$$

and for any a_i^t and a_j^t, $t = 0, 1, ..., T$ we have

$$\xi^{\pi}(a_i^t, a_j^t) = \begin{cases} 1 \text{ if } a_i^t = a_j^t \\ 0 \text{ else.} \end{cases} \tag{4.22}$$

Proof.
From definitions 4.1.6 and 4.1.7, we obtain

$$\xi^{\pi}(a_i^t, a_j^t) = p_0^{\pi}(a_i^t, a_j^t) = \begin{cases} 1 \text{ if } a_i^t = a_j^t \\ 0 \text{ else} \end{cases} \tag{4.23}$$

for $t = 0, ...T$ and

$$\begin{aligned} \xi^{\pi}(a_i^{t+k}, a_j^t) &= \gamma^k p_k^{\pi}(a_i^{t+k}|a_j^t) \\ &= \sum_{n=1}^{N_{t+k-1}} \gamma p_1^{\pi}(a_i^{t+k}|a_n^{t+k-1}) \gamma^{k-1} p_{k-1}^{\pi}(a_n^{t+k-1}|a_j^t) \\ &= \sum_{n=1}^{N_{t+k-1}} \gamma p_1^{\pi}(a_i^{t+k}|a_n^{t+k-1}) \xi^{\pi}(a_n^{t+k-1}, a_j^t) \end{aligned} \tag{4.24}$$

for $t = 0, 1, ..., T-k$, $k > 0$. □

Now, the error contribution of a region should be high if this region has a great influence on parts of the state space with a high local error estimate.

Definition 4.1.8. (Error contribution for finite horizon problems)
Let $\hat{V}_t$ be an approximation of the value function and let π be the greedy policy with respect to this approximation. The *error contribution* $\mu(a_i^t)$ of a region a_i^t for this value function approximation is defined as

$$\mu(a_i^t) = \sum_{\tau=0}^{t} \frac{1}{N_\tau} \sum_{j=1}^{N_\tau} \xi^\pi(a_i^t, a_j^\tau)\varepsilon(a_j^\tau). \tag{4.25}$$

Error contribution for infinite horizon problems

Let $(\mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{C}, \gamma, \mathcal{D})$ be an infinite horizon MDP. Let $\{a_1, a_2, ..., a_N\}$ denote a partition of the state space, where a_i is a Lebesgue-measurable set for each $i = 1, ..., N$. The *local error estimate* of a region a_i for a value function approximation $\hat{V}$ is defined in a similar way as for the finite horizon case. Recall that the optimal value function V^* of the infinite horizon MDP is a fixed point of the Bellman operator

$$V^* = \mathcal{M}(V^*). \tag{4.26}$$

For a certain approximation $\hat{V}$, the *error estimate* $\varepsilon(s)$ in a state $s \in \mathcal{S}$ is defined as

$$\varepsilon(s) = |\mathcal{M}(\hat{V}(s)) - \hat{V}(s)|. \tag{4.27}$$

As in the finite horizon case, the *local error estimate* of a region a_i is defined as the mean error estimate of this region, that is

$$\varepsilon(a_i) = \frac{1}{\lambda(a_i)} \int_{a_i} \varepsilon(s)ds, \tag{4.28}$$

where $\lambda(a_i)$ denotes the Lebesgue measure of the region a_i.

As for finite horizon problems, the *error contribution* of a region is a measure of the influence of this region on the overall approximation error. Defining the concept of *influence* is a bit more involved for infinite horizon MDPs. First of all, we present the infinite horizon equivalents of the *k-step region transition probabilities*.

Definition 4.1.9. (k-step region transition probabilities for infinite horizon problems)
Let the probability that the system makes a transition from a_i to a_j under policy π

be denoted as $P^\pi(a_j|a_i)$. The k-step region transition probabilities $p_k^\pi(a_j|a_i)$ are defined recursively as follows:

$$p_0^\pi(a_j|a_i) = \begin{cases} 1 \text{ if } a_i = a_j \\ 0 \text{ else} \end{cases}, \tag{4.29}$$

$$p_1^\pi(a_j|a_i) = P^\pi(a_j|a_i) \tag{4.30}$$

and

$$p_k^\pi(a_j|a_i) = \sum_{n=1}^{N_t} p_1^\pi(a_j|a_n) p_{k-1}^\pi(a_n|a_i) \tag{4.31}$$

Again, we define the influence of a region a_i on a region a_j as the expected number of visits to a_i from a_j using discounted k-step region transition probabilities $\gamma^k p_k^\pi(a_i|a_j)$, where $0 < \gamma < 1$ is the discount factor.

Definition 4.1.10. (Influence for infinite horizon problems)
The *influence* $\xi^\pi(a_i, a_j)$ of region a_i on region a_j under policy π is defined as

$$\xi^\pi(a_i, a_j) = \sum_{k=0}^{\infty} \gamma^k p_k^\pi(a_i|a_j). \tag{4.32}$$

Computing the influence for infinite horizon problems is more complicated than for the finite horizon case. The following lemma presents a relation similar to (4.21) for this situation.

Lemma 1.
For any regions a_i and a_j and any policy π, we have

$$\xi^\pi(a_i, a_j) = \begin{cases} \sum_{n=1}^{N} \gamma p_1^\pi(a_i|a_n) \cdot \xi^\pi(a_n, a_j) + 1 \text{ if } a_i = a_j \\ \sum_{n=1}^{N} \gamma p_1^\pi(a_i|a_n) \cdot \xi^\pi(a_n, a_j) \text{ else.} \end{cases} \tag{4.33}$$

Proof.
Using the definition of the influence (4.1.10) and the recursive definition of the k-step region transition probabilities (4.29), (4.30) and (4.31), we obtain

$$
\begin{aligned}
\xi^{\pi}(a_i, a_j) &= \sum_{k=0}^{\infty} \gamma^k p_k^{\pi}(a_i|a_j) \\
&= \sum_{k=1}^{\infty} \gamma^k p_k^{\pi}(a_i|a_j) + p_0^{\pi}(a_i|a_j) \\
&= \sum_{k=0}^{\infty} \gamma^{k+1} p_{k+1}^{\pi}(a_i|a_j) + p_0^{\pi}(a_i|a_j) \\
&= \sum_{k=0}^{\infty} \sum_{n=1}^{N} \gamma p_1^{\pi}(a_i, a_n) \cdot \gamma^k p_k^{\pi}(a_n, a_j) + p_0^{\pi}(a_i, a_j) \\
&= \sum_{n=1}^{N} \gamma p_1^{\pi}(a_i, a_n) \sum_{k=0}^{\infty} \gamma^k p_k^{\pi}(a_n, a_j) + p_0^{\pi}(a_i, a_j) \\
&= \begin{cases} \sum_{n=1}^{N} \gamma p_1^{\pi}(a_i|a_n) \cdot \xi^{\pi}(a_n, a_j) + 1 \text{ if } a_i = a_j \\ \sum_{n=1}^{N} \gamma p_1^{\pi}(a_i|a_n) \cdot \xi^{\pi}(a_n, a_j) \text{ else.} \end{cases}
\end{aligned} \tag{4.34}
$$

□

This lemma paves the way for an iterative definition of the influence for infinite horizon problems.

Theorem 6. (Iterative definition of the influence for infinite horizon problems)
Let π be a policy. Recursively define

$$
\xi_0^{\pi}(a_i, a_j) = \begin{cases} 1 \text{ if } a_i = a_j \\ 0 \text{ else} \end{cases} \tag{4.35}
$$

and

$$
\xi_{n+1}^{\pi}(a_i, a_j) = \begin{cases} \sum_{l=1}^{N} \gamma p_1^{\pi}(a_i|a_l) \cdot \xi_n^{\pi}(a_l, a_j) + 1 \text{ if } a_i = a_j \\ \sum_{l=1}^{N} \gamma p_1^{\pi}(a_i|a_l) \cdot \xi_n^{\pi}(a_l, a_j) \text{ else.} \end{cases} \tag{4.36}
$$

This iterative process converges to the influence $\xi^{\pi}(a_i, a_j)$, that is

$$
\lim_{n \to \infty} \xi_n^{\pi}(a_i, a_j) = \xi^{\pi}(a_i, a_j). \tag{4.37}
$$

Proof.
Let U denote the space of functions that map the set $\{a_1, a_2, ..., a_N\}$ to $\mathbb{R}$. Let $d : U \times U \to \mathbb{R}$ be the distance function defined by

$$d(f, g) = ||f - g||_1 \tag{4.38}$$

for $f, g \in U$. Let A be an operator on the metric space (U, d) defined as

$$Af(\cdot) = \sum_{l=1}^{N} \gamma p_1^{\pi}(\cdot|a_l) f(a_l) \tag{4.39}$$

for a function $f \in U$ and a policy π. This operator is a contraction operator if there exists a constant K, $0 < K < 1$ such that

$$d(Af, Ag) \leq K d(f, g) \ \forall f, g \in U. \tag{4.40}$$

We show that A satisfies this condition. For $f, g \in U$ we have

$$\begin{aligned} d(Af, Ag) &= ||Af - Ag||_1 \\ &= \sum_{i=1}^{N} |(Af - Ag)(a_i)| \\ &= \sum_{i=1}^{N} \left| \sum_{l=1}^{N} \gamma p_1^{\pi}(a_i|a_l)(f(a_l) - g(a_l)) \right| \\ &\leq \sum_{i=1}^{N} \sum_{l=1}^{N} \gamma p_1^{\pi}(a_i|a_l) \, |f(a_l) - g(a_l)| \\ &= \sum_{l=1}^{N} \gamma \, |f(a_l) - g(a_l)| \sum_{i=1}^{N} p_1^{\pi}(a_i|a_l). \end{aligned} \tag{4.41}$$

Since

$$\sum_{i=1}^{N} p_1^{\pi}(a_i|a_l) = 1 \text{ for } l = 1, ..., N, \tag{4.42}$$

we obtain

$$\sum_{l=1}^{N} \gamma \left| f(a_l) - g(a_l) \right| \sum_{i=1}^{N} p_1^{\pi}(a_i|a_l) = \sum_{l=1}^{N} \gamma \left| f(a_l) - g(a_l) \right| = \gamma ||f - g||_1. \tag{4.43}$$

Since $0 < \gamma < 1$, it follows that A is a contraction operator. Using this and Lemma 1, we obtain

$$\begin{aligned} ||\xi_{n+1}^{\pi}(\cdot, a_j) - \xi^{\pi}(\cdot, a_j)||_1 &= \sum_{i=1}^{N} \left| \xi_{n+1}^{\pi}(a_i, a_j) - \xi^{\pi}(a_i, a_j) \right| \\ &= \sum_{i=1}^{N} |A\xi_n^{\pi}(\cdot, a_j)(a_i) - A\xi^{\pi}(\cdot, a_j)(a_i)| \\ &= ||A\xi_n^{\pi}(\cdot, a_j) - A\xi^{\pi}(\cdot, a_j)||_1 \\ &\leq \gamma ||\xi_n^{\pi}(\cdot, a_j) - \xi^{\pi}(\cdot, a_j)||_1. \end{aligned}$$

Thus,

$$\lim_{n \to \infty} \xi_n^{\pi}(a_i, a_j) = \xi^{\pi}(a_i, a_j). \tag{4.44}$$

□

Finally, the error contribution is defined in a similar way as for finite horizon problems.

Definition 4.1.11. (Error contribution for infinite horizon problems)
Let $\hat{V}$ be a value function approximation and let π be its corresponding greedy policy. The *error contribution* $\mu(a_i)$ of a region a_i for this approximation is given by

$$\mu(a_i) = \frac{1}{N} \sum_{j=1}^{N} \xi^{\pi}(a_i, a_j) \varepsilon(a_j). \tag{4.45}$$

Defining the *influence matrix* under policy π as

$$\Xi^{\pi} = \begin{bmatrix} \xi^{\pi}(a_1, a_1) & \xi^{\pi}(a_1, a_2) & \dots & \xi^{\pi}(a_1, a_N) \\ \xi^{\pi}(a_2, a_1) & \xi^{\pi}(a_2, a_2) & \dots & \xi^{\pi}(a_2, a_N) \\ \vdots & \vdots & \ddots & \vdots \\ \xi^{\pi}(a_N, a_1) & \xi^{\pi}(a_N, a_2) & \dots & \xi^{\pi}(a_N, a_N) \end{bmatrix}, \tag{4.46}$$

expression (4.45) can be rewritten as

$$\boldsymbol{\mu} = \frac{1}{N}\Xi^{\pi}\boldsymbol{\varepsilon}, \tag{4.47}$$

where

$$\boldsymbol{\mu} = [\mu(a_1), \mu(a_2), ..., \mu(a_N)] \tag{4.48}$$

and

$$\boldsymbol{\varepsilon} = [\varepsilon(a_1), \varepsilon(a_2), ..., \varepsilon(a_N)]. \tag{4.49}$$

Scoring the cells

After evaluating each of the refinement criteria, we give each cell of the gridded state space a score using the scoring function

$$\text{score}(a_i) = \alpha_1\eta(a_i) + \alpha_2\zeta(a_i) + \alpha_3\theta(a_i) + \alpha_4\mu(a_i), \tag{4.50}$$

for parameters $\alpha_1, \alpha_2, \alpha_3, \alpha_4$. The optimal values of these parameter values are problem dependent. The cell with the highest score will be selected for refinement.

4.1.2 Refinement direction

After selecting a cell to refine, it remains to decide upon a suitable refinement direction. In choosing this direction, we aim to limit the number of refinements necessary to find a sufficiently accurate approximation of the value function.
Let a_{i1}^j and a_{i2}^j denote the two new cells obtained after refining a cell a_i along a certain direction j, $j = 1, 2, ..., n$. If cell a_{i1}^j exhibits great fluctuations of the value function, has a high visitation frequency and a high error contribution compared to cell a_{i2}^j, then we consider direction j a good refinement direction. After all, through refining along this direction, we managed to isolate the part of the cell a_i that was causing trouble. In the following iterations, we can focus our attention to cell a_{i1}^j, whereas a_{i2}^j might require no or few further refinements.
Based on this intuitive idea, we construct the following scoring function for refinement directions for a cell a_i:

$$\begin{aligned}\text{score}(j) = \beta_1 \left|\eta(a_{i1}^j) - \eta(a_{i2}^j)\right| + \beta_2|\zeta(a_{i1}^j) - \zeta(a_{i2}^j)| + \\ \beta_3|\theta(a_{i1}^j) - \theta(a_{i2}^j| + \beta_4|\mu(a_{i1}^j) - \mu(a_{i2}^j)|.\end{aligned} \tag{4.51}$$

for parameters $\beta_1, \beta_2, \beta_3, \beta_4$. We refine the selected cell along the direction with the highest score.

4.2 Dynamic programming with simplex splines

In this section, we integrate the triangulation refinement procedure into the basic backward dynamic programming algorithm to find a simplex spline approximation of the value function for finite horizon Markov decision processes.

Let $(\mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{C}, \gamma, \mathcal{D})$ be a finite horizon MDP. Algorithm 2 shows the original backward dynamic programming algorithm for discrete state spaces.

Algorithm 2: Backward dynamic programming

Input: A finite horizon Markov decision process $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mathcal{D})$ with horizon T.

Initialize: *Initialize $V_{T+1}(s_{T+1})$ as the final costs for all states $s_{T+1} \in \mathcal{S}$. Set $t = T$.*

while $t \geq 0$ **do**

  Calculate $V_t(s_t) = \max_{a_t \in \mathcal{A}} \left(C_t(s_t, a_t) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s_t) V_{t+1}(s') \right)$ for all $s_t \in \mathcal{S}$.

  Decrement t by 1.

end

Instead of the true value function, the simplex spline dynamic programming algorithm calculates a value function approximation $\hat{V}_t$ for each time $t = 0, 1, ..., T$ that is based on simplex splines defined on a triangulation of the state space. We initialize the triangulation for each time t as a Delaunay triangulation constructed on the smallest hyperrectangle that covers the state space. Each iteration k of the algorithm roughly consists of two steps: first, we compute simplex spline approximations $\hat{V}_t^k$ of the value function for each time t defined on the current triangulations. Then, we refine the triangulations for each time t by

the procedure described in Section 4.1. We proceed to explain each of these steps in detail.

Suppose after iteration k, we obtained grid cells $a_1^{k,t}, a_2^{k,t}, ..., a_{N_{k,t}}^{k,t}$ corresponding to a set of vertices $\mathbf{V}_t^k$ for each $t = 0, 1, ..., T$. For each time t, the corresponding set of local grids is triangulated with a Delaunay triangulation $\mathcal{T}_t^k$. We search for a simplex spline of degree d and continuity order $\mathcal{C}^r$ defined on this triangulation which gives rise to the new value function approximation

$$\hat{V}_t^{k+1}(s|\mathbf{c}_t^{k+1}) = s_d^r(s|\mathbf{c}_t^{k+1}),\ s \in \mathcal{S}. \tag{4.52}$$

We find the B-coefficients $\mathbf{c}_t^{k+1}$ of these simplex splines by means of the generalized least squares method from Section 3.2.3.2. To this end, we need a set of "data points" in the state space. Recall from Section 3.2.1.1 that each simplex needs to contain at least $\hat{d}$ data points for us to be able to calculate the B-coefficients. Hence, for each $t = 0, 1, ..., T$, we randomly choose $\hat{d}$ states $s_{j,1}^{k,t}, s_{j,2}^{k,t}, ..., s_{j,\hat{d}}^{k,t}$ in each simplex $t_j^{k,t} \in \mathcal{T}_t^k$, which will serve as these "data points". We refer to these states as *control states*. To each of these control states, we assign a value $v(s_{j,i}^{k,t})$ using the current value function approximation $\hat{V}_t^k(s|\mathbf{c}_t^k)$, which is defined on the current triangulations $\mathcal{T}_t^k$:

$$v(s_{j,i}^{k,t}) = \max_{a_t \in \mathcal{A}} \left(C_t(s_{j,i}^{k,t}, a_t) + \gamma \int_{\mathcal{S}} \mathcal{F}(s_{j,i}^{k,t}, a_t, s') \hat{V}_{t+1}^{k-1}(s'|\mathbf{c}_{t+1}^{k-1}) ds' \right) \tag{4.53}$$

for $j = 1, ..., |\mathcal{T}_t^k|,\ i = 1, ..., \hat{d}$ and $t = 0, 1, ..., T$.
Using the control states and their values, we apply the generalized least squares algorithm to obtain the B-coefficients of the simplex splines in expression (4.52), securing the new value function approximations $\hat{V}_t^{k+1}(s|\mathbf{c}_t^{k+1})$, $t = 0, 1, ..., T$, $s \in \mathcal{S}$.

In the second part of iteration k, we refine the triangulations $\mathcal{T}_t^k$ using the procedure presented in Section 4.1. For each time t, we compute the score of each grid cell $a_i^{k,t}$, $i = 1, 2, ..., N_{k,t}$ by expression (4.50). After selecting the cell with the highest score for each time t, we compute the scores of the refinement directions corresponding to these cells using expression (4.51). Then, for each time t, we refine the cell with the highest score along the direction with the highest score. Finally, we triangulate the thus obtained new grids with Delaunay triangulations $\mathcal{T}_t^{k+1}$, $t = 0, 1, ..., T$.
Once the value function approximation for time T satisfies the condition

$$||\hat{V}_T^{k+1}(\cdot|\mathbf{c}^{k+1}) - \hat{V}_T^k(\cdot|\mathbf{c}^k)||_p < \varepsilon \tag{4.54}$$

for some $p \geq 1$ and $\varepsilon > 0$, we stop refining the triangulation and keep the approximation $\hat{V}_T(s|\mathbf{c}^k)$ fixed during the next iterations. Then, we continue until condition (4.54) is satisfied for time $T-1$ and we stop refining the triangulation for time $T-1$. Thus we proceed until the triangulations for all times t are fixed. Algorithm 3 provides an overview of the simplex spline dynamic programming algorithm.

4.2.1 Analysis simplex spline dynamic programming

In this section, we show that the simplex spline dynamic programming algorithm converges and present an error bound on the result.

Theorem 7. (Convergence simplex spline dynamic programming)
Let $\hat{V}_t^k(s|\mathbf{c}^k) = s_d^r(s, \mathbf{c}^k)$ denote the value function approximation after the kth iteration of the simplex spline dynamic programming algorithm. As $k \to \infty$, the approximation $\hat{V}_t^k(s|\mathbf{c}^k)$ converges to the optimal value function $V_t^*(s)$ for each $t = 0, 1, ..., T$.

Proof.
We prove this theorem by induction over $n = T - t$.
For $n = 0$, we have $t = T$. The function $\hat{V}_T^k(s|\mathbf{c}^k)$ approximates $\mathcal{M}V_{T+1}$, where V_{T+1} is a known function containing the final costs. As $k \to \infty$, the triangulation $\mathcal{T}_T^k$ grows infinitely dense. On an infinitely dense triangulation, the simplex spline approximator $\hat{V}_T^k(s|\mathbf{c}^k)$ coincides with $\mathcal{M}V_{T+1}$. Thus, as $k \to \infty$, the approximation $\hat{V}_T^k(s|\mathbf{c}^k)$ converges to $\mathcal{M}V_{T+1} = V_T^*$.
Suppose that the theorem holds for $n = m$, where $0 \leq m \leq T-1$. We show that it must hold for $n = m+1$ as well. For $n = m+1$, we have $t = T-m-1$. The function $\hat{V}_t^k(s|\mathbf{c}^k)$ approximates $\mathcal{M}\hat{V}_{t+1}^k(s|\mathbf{c}^k)$. As $k \to \infty$, the triangulation $\mathcal{T}_t^k$ grows infinitely dense. On an infinitely dense triangulation, the simplex spline approximator $\hat{V}_t^k(s|\mathbf{c}^k)$ coincides with $\mathcal{M}\hat{V}_{t+1}^k$. By the induction hypothesis, the function $\hat{V}_{t+1}^k$ converges to V_{t+1}^* as $k \to \infty$. This implies that $\mathcal{M}\hat{V}_{t+1}^k$ converges to $\mathcal{M}V_{t+1}^* = V_t^*$ as $k \to \infty$. Hence, the approximation $\hat{V}_t^k(s|\mathbf{c}^k)$ converges to V_t^* as $k \to \infty$.
Thus, $\hat{V}_t^k(s|\mathbf{c}^k)$ converges to the optimal value function $V_t^*(s)$ as $k \to \infty$ for each $t = 0, 1, ..., T$.

□

Algorithm 3: Simplex spline dynamic programming

Input: A Markov decision process $(\mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{C}, \gamma, \mathcal{D})$ with finite horizon T. Simplex spline value function approximators $\hat{V}_t(s|\mathbf{c}) = s_d^r(s, \mathbf{c}), s \in \mathcal{S}$ for each $t = 0, 1, ..., T$.

Initialize: *Initialize* $\boldsymbol{V}_t^0$ *as the set of vertices of the smallest hyperrectangle* $a_1^{0,t}$ *that covers the state space and the midpoint of this hyperrectangle for each* $t = 0, 1, ..., T$.

Let $\mathcal{T}_t^0$, $t = 0, 1, ..., T$ *denote the Delaunay triangulations of these vertex sets.*

Initialize vectors of B-coefficients $\boldsymbol{c}_t^0 \in \mathbb{R}^{(|\mathcal{T}_t^0| \cdot \hat{d}) \times 1}$ *arbitrarily, for example* $\boldsymbol{c}_t^0 = \boldsymbol{0}$ *for each* $t = 0, 1, ..., T$. *Initialize* $k = 0$.

Set $\tau = T$.
while $\tau \geq 0$ **do**
  for $t = 0, 1, ..., \tau$ **do**
    for $j = 1, 2, ..., |\mathcal{T}_t^k|$ **do**
      Choose $\hat{d}$ control states $s_{j,1}^{k,t}, s_{j,2}^{k,t}, ..., s_{j,\hat{d}}^{k,t}$ in simplex $t_j^{k,t} \in \mathcal{T}_t^k$.
      Assign a value $v(s_{j,i}^{k,t})$ to each control state $s_{j,i}^{k,t}$ using expression (4.53).
    end
    Use the generalized least squares algorithm to obtain B-coefficients $\mathbf{c}_t^{k+1} \in \mathbb{R}^{(|\mathcal{T}_t^k| \cdot \hat{d}) \times 1}$, constituting $\hat{V}_t^{k+1}(s|\mathbf{c}_t^{k+1})$, $t = 0, 1, ..., T$.
  end
  for $t = 0, 1, ..., \tau$ **do**
    Compute score of each grid cell $a_i^{k,t}$, $i = 1, 2, ..., N_{k,t}$ by expression (4.50).
    Choose the cell with the highest score.
    Compute score of each refinement direction $j = 1, ..., n$.
    Refine the selected cell along the selected direction, yielding grid cells $a_1^{k+1,t}, a_2^{k+1,t}, ..., a_{N_{k+1},t}^{k+1,t}$, constituted by the new vertex set $\mathbf{V}_t^{k+1}$.
    Triangulate the new set of local grids with a Delaunay triangulation, yielding $\mathcal{T}_t^{k+1}$.
  end
  if $||\hat{V}_\tau^{k+1}(\cdot|\boldsymbol{c}^{k+1}) - \hat{V}_\tau^k(\cdot|\boldsymbol{c}^k)||_p < \varepsilon$ **then**
    Let $\hat{V}_\tau(s) = \hat{V}_\tau^{k+1}(s|\mathbf{c}^{k+1})$.
    Set $\tau = \tau - 1$.
  end
  Set $k = k + 1$.
end

In order to provide an error bound on the result of the simplex spline dynamic programming algorithm, we make the following assumption:

Assumption 1.
Let $0 \leq \tau \leq T$. We assume that

$$||\hat{V}_\tau^{k+1}(s|\mathbf{c}^{k+1}) - \hat{V}_\tau^k(s|\mathbf{c}^k)||_p \leq ||\hat{V}_\tau^k(s|\mathbf{c}^k) - \hat{V}_\tau^{k-1}(s|\mathbf{c}^{k-1})||_p \tag{4.55}$$

for all $k = 1, 2, ...$

From the fact that it converges to $\mathcal{M}\hat{V}_{\tau+1}(s)$, it follows that the sequence $\{\hat{V}_\tau^k(s|\mathbf{c}^k), k \geq 0\}$ is Cauchy. Hence, for each $\varepsilon > 0$, there exists a positive integer N_ε such that for all $k > N_\varepsilon$, we have

$$||\hat{V}_\tau^k(s|\mathbf{c}^k) - \hat{V}_\tau^{k-1}(s|\mathbf{c}^{k-1})||_p < \varepsilon. \tag{4.56}$$

If for some iteration k, the value function approximation $\hat{V}_\tau^k(s|\mathbf{c}^k)$ satisfies condition (4.54), Assumption 1 implies that $k \geq N_\varepsilon$. It follows that $||\hat{V}_\tau^k - \mathcal{M}\hat{V}_{\tau+1}||_p < \varepsilon$.
Throughout the course of the algorithm, the impact that the refinements have on the triangulations is likely to decrease. Therefore, the value function approximations will probably change less and less, which renders Assumption 1 a reasonable assumption.

Theorem 8. (Error bound simplex spline dynamic programming)
Suppose Assumption 1 holds. Let $\hat{V}_t(s)$, $s \in \mathcal{S}$ and $t = 0, 1, ..., T$, denote the value function approximation that results from the simplex spline dynamic programming algorithm. Let $V_t^*(s)$, $s \in \mathcal{S}$ and $t = 0, 1, ..., T$, denote the optimal value function. The value function approximation $\hat{V}_t^k(s)$ satisfies the following error bound:

$$||\hat{V}_t^k - V_t^*||_p < \sum_{i=0}^{T-t} \gamma^i \varepsilon, \ t = 0, 1, ..., T. \tag{4.57}$$

Proof.
We prove expression (4.57) by induction over $n = T - t$.
For $n = 0$, we have $t = T$. Since T is the end of the horizon, the sequence $\hat{V}_T^k$ converges to the actual optimal value function V_T^*. Upon termination of the algorithm, we have

$$||\hat{V}_T^k - \hat{V}_T^{k-1}||_p < \varepsilon. \tag{4.58}$$

Since the sequence $\{\hat{V}_T^k, k \geq 0\}$ converges to V_T^* and satisfies Assumption 1, this implies

$$||\hat{V}_T^k - V_T^*||_p < \varepsilon. \tag{4.59}$$

Now, suppose (4.57) holds for $n = m$, where $0 \leq m \leq T-1$. We show that it must hold for $n = m+1$ as well.
For $n = m+1$, we have $t = T-m-1$. Upon termination of the algorithm, we have

$$||\hat{V}_t^k - \hat{V}_t^{k-1}||_p < \varepsilon. \tag{4.60}$$

Since the sequence $\{\hat{V}_t^k, k \geq 0\}$ converges to $\mathcal{M}\hat{V}_{t+1}$ and satisfies Assumption 1, this implies

$$||\hat{V}_t^k - \mathcal{M}\hat{V}_{t+1}||_p < \varepsilon. \tag{4.61}$$

It follows that

$$\begin{aligned} ||\hat{V}_t^k - V_t^*||_p &= ||\hat{V}_t^k - \mathcal{M}\hat{V}_{t+1} + \mathcal{M}\hat{V}_{t+1} - V_t^*||_p \\ &\leq ||\hat{V}_t^k - \mathcal{M}\hat{V}_{t+1}||_p + ||\mathcal{M}\hat{V}_{t+1} - V_t^*||_p \\ &< \varepsilon + ||\mathcal{M}\hat{V}_{t+1} - \mathcal{M}V_{t+1}^*||_p. \end{aligned} \tag{4.62}$$

By the contraction property of the Bellman operator, we obtain

$$\begin{aligned} ||\hat{V}_t^k - V_t^*||_p &< \varepsilon + ||\mathcal{M}\hat{V}_{t+1} - \mathcal{M}V_{t+1}^*||_p \\ &\leq \varepsilon + \gamma||\hat{V}_{t+1} - V_{t+1}^*||_p. \end{aligned} \tag{4.63}$$

The induction hypothesis now yields

$$
\begin{aligned}
||\hat{V}_t^k - V_t^*||_p &< \varepsilon + \gamma||\hat{V}_{t+1} - V_{t+1}^*||_p \\
&< \varepsilon + \gamma \sum_{i=0}^{T-t-1} \gamma^i \varepsilon \\
&= \sum_{i=0}^{T-t} \gamma^i \varepsilon.
\end{aligned} \tag{4.64}
$$

Thus, expression (4.57) holds for all $t = 0, 1, ..., T$. □

4.3 Recursive least squares approximate policy iteration with simplex splines

In this section, we integrate the simplex spline approximation methods and the triangulation refinement procedure into the RLS API algorithm described in Section 2.2.4. We present the method for infinite horizon problems. To finite horizon problems, a similar version of the algorithm applies (Powell, 2011). To update the B-coefficient vector, we replace the recursive least squares procedure given in expression (2.23) by the equality constrained recursive least squares estimator discussed in Section 3.2.3.3. Learning the B-coefficients and refining the triangulation happens simultaneously. As in the simplex spline dynamic programming algorithm, we initialize the triangulation as a Delaunay triangulation of the smallest hyperrectangle that contains the state space. Each iteration k of the algorithm, we refine the triangulation by the procedure described in Section 4.1. Then, we choose new B-coefficients in such a way that the simplex spline on the new triangulation resembles the simplex spline we had before the refinement as close as possible. Finally, we update the B-coefficients in accordance with the RLS API algorithm.

Suppose that after iteration $k-1$, we have a Delaunay triangulation $\mathcal{T}^k$ of the state space corresponding to a set of vertices $\mathbf{V}^k$ and a simplex spline value function approximation $\hat{V}_k(s|\mathbf{c}_k)$, where $\mathbf{c}_k \in \mathbb{R}^{|\mathcal{T}^k|\cdot\hat{d}}$. First of all, we refine the triangulation by the procedure of Section 4.1 to obtain the new triangulation $\mathcal{T}^{k+1}$ with vertices $\mathbf{V}^{k+1}$. Recall that the triangulation refinement procedure removes one of the vertices of the triangulation and adds at most $2^{n-1}+2$ new vertices, where n is the number of dimensions of the state space. This process might not only affect the configuration of the simplices in the selected grid cell, but also in surrounding regions of the state space. Hence, it is possible that the B-coefficients in the vector $\mathbf{c}_k$ violate the smoothness constraints of the new triangulation. To ensure a smooth transition between the learning phases of iterations k and $k+1$, we compute a new B-coefficient vector $\mathbf{c}'_k$ in such a way that the new simplex spline closely resembles the previous simplex spline that was defined on triangulation $\mathcal{T}^k$ with coefficients $\mathbf{c}_k$. To this end, we first sample N random states $s_{i,1}, s_{i,2}, ..., s_{i,N}$ in each simplex t_i of $\mathcal{T}^{k+1}$. Then, we compute the values of these states according to the current value function approximation $\hat{V}(s|\mathbf{c}_k)$ en store these values in a vector $\mathbf{y} \in \mathbb{R}^{|\mathcal{T}^{k+1}|\cdot N}$. Finally, we apply the generalized least squares procedure from Section 3.2.3.2 to the sampled states and the vector $\mathbf{y}$ to find the B-coefficient vector $\mathbf{c}'_k$.

After refining the triangulation, we update the vector $\mathbf{c}'_k$ in accordance with the RLS API

algorithm: first of all, we sample a random state from each simplex. For each of these states, we take a walk of length M through the state space and update the B-coefficients according to the equality constrained least squares procedure. Thus we obtain the vector $\mathbf{c}_{k+1} \in \mathbb{R}^{|\mathcal{T}^{k+1}|\cdot\hat{d}|}$.
An overview of the simplex spline RLS API algorithm is given in Algorithm 4.

Algorithm 4: Simplex spline recursive least squares approximate policy iteration

Input: An infinite horizon Markov decision process $(\mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{C}, \gamma, \mathcal{D})$. A simplex spline value function approximator $\hat{V}(s|\mathbf{c}) = s_d^r(s, \mathbf{c}), s \in \mathcal{S}$ and sample path length M.

Initialize: *Initialize grid cell a_1^0 as the smallest hyperrectangle that contains the state space. Initialize $\boldsymbol{V}^0$ as the set of vertices of a_1^0 and its midpoint.*
Let $\mathcal{T}^0$ denote the Delaunay triangulation of this grid.
Initialize a vector of B-coefficients $\boldsymbol{c}_0 \in \mathbb{R}^{|\mathcal{T}^0|\cdot\hat{d}}$ arbitrarily, for example $\boldsymbol{c}_0 = \boldsymbol{0}$.

for $k = 0, 1, ...$ **do**
- Refine triangulation $\mathcal{T}^k$ according to Section 4.1, yielding $\mathcal{T}^{k+1}$ with vertex set $\mathbf{V}^{k+1}$.
- Use the generalized least squares procedure to obtain $\mathbf{c}'_k$.
- Initialize $\mathbf{c}_k^0 = \mathbf{c}'_k$.
- Let π_k^0 be the greedy policy with respect to $\hat{V}(s|\mathbf{c}_k^0)$.
- **for** $i = 1, 2, ..., |\mathcal{T}^{k+1}|$ **do**
  - Arbitrarily choose initial state $s^{k+1,i,0}$ in simplex $t_i^{k+1} \in \mathcal{T}^{k+1}$.
  - Initialize $\mathbf{c}_k^{i,0} = \mathbf{c}_k^i$.
  - **for** $j = 0, 1, ..., M-1$ **do**
    - Sample random information W^{j+1}.
    - Choose action a^j according to policy π_k^i.
    - Compute next state $s^{k+1,i,j+1}$ after taking action a^j from state $s^{k+1,i,j}$ and observing W^{j+1}.
    - Compute costs C^j induced by taking action a_j from state $s^{k+1,i,j}$ and observing W^{j+1}.
    - Compute $\mathbf{c}_k^{i,j+1}$ by the equality constrained recursive least squares procedure (3.76).
  - **end**
  - Let $\mathbf{c}_k^{i+1} = \mathbf{c}_k^{i,M}$.
- **end**
- Let $\mathbf{c}_{k+1} = \mathbf{c}_k^{|\mathcal{T}^{k+1}|}$.

end

4.3.1 Alternative approach for linear simplex splines of continuity order 0

For simplex splines of degree 1 and continuity order $\mathcal{C}^0$, using the equality constrained recursive least squares estimator is not necessary. In this section, we present an alternative formulation for this type of splines. For a linear spline of continuity order $\mathcal{C}^0$ that is written in this form, finding the B-coefficients is an unconstrained optimization problem, which can simply be solved by means of the original recursive least squares procedure.

Written in the original B-form, a linear simplex spline of continuity order $\mathcal{C}^0$ defined on a triangulation $\mathcal{T} \equiv \cup_{i=1}^{J} t_i$ is given by

$$s_1^0(\mathbf{x}) = \mathbf{B}^1(\mathbf{x})^T \cdot \mathbf{D}(\mathbf{x}) \cdot \mathbf{c}, \ \mathbf{x} \in \mathbb{R}^n, \tag{4.65}$$

where

$$\mathbf{B}^1(\mathbf{x}) = [\mathbf{B}^1(b(\mathbf{x}))_{t_1}, \mathbf{B}^1(b(\mathbf{x}))_{t_2}, ..., \mathbf{B}^1(b(\mathbf{x}))_{t_J}]^T \in \mathbb{R}^{(J \cdot \hat{d}) \times 1}, \tag{4.66}$$

$$\mathbf{c} = [\mathbf{c}^{t_1^T}, \mathbf{c}^{t_2^T}, ..., \mathbf{c}^{t_J^T}]^T \in \mathbb{R}^{(J \cdot \hat{d}) \times 1}. \tag{4.67}$$

and $\mathbf{D}(\mathbf{x})$ is the membership matrix from expression (3.24).
Recall that each B-coefficient $c_\kappa^{t_j}$ belonging to a simplex $t_j \in \mathcal{T}$ has a spatial representation. From Section 3.1.8, it follows that the constraints for continuity of order $\mathcal{C}^0$ only involve the B-coefficients that are located at the vertices of the triangulation. Let $\mathbf{V}_{t_j}$ denote the set of vertices of a simplex $t_j \in \mathcal{T}$ and let $\mathbf{V}$ denote the set of all vertices, that is $\mathbf{V} = \cup_{j=1}^{J} \mathbf{V}_{t_j}$. Finally, let $\mathcal{T}_{v_i}$ denote the triangulation formed by the simplices in $\mathcal{T}$ of which v_i is a vertex. For linear simplex splines, all B-coefficients of a simplex $t_j \in \mathcal{T}$ are located at the vertices. Hence, they can be written as $c_{v_i}^{t_j}$ for $v_i \in \mathbf{V}_{t_j}$. The constraints for continuity of order $\mathcal{C}^0$ state that the B-coefficients at the vertices should be equal to a common value c_{v_i}, or

$$c_{v_i}^{t_j} = c_{v_i} \ \forall v_i \in \mathbf{V}, \ t_j \in \mathcal{T}_{v_i}. \tag{4.68}$$

Note that the multiindices κ for linear splines have only one nonzero element. Hence, Bernstein basis polynomials $B_\kappa^1(b(\mathbf{x}))$ of a simplex t_j return only the barycentric coordi-

nate of $\mathbf{x}$ that belongs to the vertex v_i that corresponds to the nonzero element of κ. Let this coordinate be denoted by $b_{v_i}^{t_j}(\mathbf{x})$. Using these coordinates and the new notation of the B-coefficients, the simplex spline of expression (4.65) can be written as

$$s_1^0(\mathbf{x}) = \sum_{t\in\mathcal{T}} \sum_{v\in\mathbf{V}_t} b_v^t(\mathbf{x})\delta_{tk(\mathbf{x})}c_v^t, \tag{4.69}$$

where $\delta_{tk(\mathbf{x})}$ is the simplex membership operator from expression (3.22). Interchanging the summations and using expression (4.68) yields

$$\sum_{v\in\mathbf{V}} \sum_{t\in\mathcal{T}_v} b_v^t(\mathbf{x})\delta_{tk(\mathbf{x})}c_v^t = \sum_{v\in\mathbf{V}} c_v \sum_{t\in\mathcal{T}_v} b_v^t(\mathbf{x})\delta_{tk(\mathbf{x})}. \tag{4.70}$$

Defining

$$b_v(\mathbf{x}) = \sum_{t\in\mathcal{T}_v} b_v^t(\mathbf{x})\delta_{tk(\mathbf{x})} \tag{4.71}$$

results in the following formulation of the simplex spline from expression (4.65):

$$s_1^0(\mathbf{x}) = \sum_{v\in\mathbf{V}} c_v b_v(\mathbf{x}) = \hat{\mathbf{b}}(\mathbf{x})^T\hat{\mathbf{c}}, \tag{4.72}$$

where

$$\hat{\mathbf{b}}(\mathbf{x}) = [b_{v_1}(\mathbf{x}), b_{v_2}(\mathbf{x}), ..., b_{v_{|\mathbf{V}|}}(\mathbf{x})]^T \in \mathbb{R}^{|\mathbf{V}|\times 1} \tag{4.73}$$

and

$$\hat{\mathbf{c}} = [c_{v_1}, c_{v_2}, ..., c_{v_{|\mathbf{V}|}}]^T \in \mathbb{R}^{|\mathbf{V}|\times 1}. \tag{4.74}$$

Apart from the fact that expression (4.72) directly incorporates the continuity constraints, it also significantly reduces the number of B-coefficients. For a data set $(\mathbf{X}, \mathbf{y})$ consisting of N points, the problem of finding the B-coefficients is now reduced to the unconstrained least squares problem

$$\hat{\mathbf{B}}\hat{\mathbf{c}} = \mathbf{y}, \tag{4.75}$$

where

$$\hat{\mathbf{B}} = [\hat{\mathbf{b}}(\mathbf{x}_1), \hat{\mathbf{b}}(\mathbf{x}_2), ..., \hat{\mathbf{b}}(\mathbf{x}_N)]^T \in \mathbb{R}^{N \times |\mathbf{V}|}. \tag{4.76}$$

4.3.2 Analysis recursive least squares approximate policy iteration with simplex splines

In this section, we prove the convergence of the simplex spline recursive least squares approximate policy iteration algorithm.

Theorem 9. (Convergence simplex spline RLS API)
Let $\hat{V}^k(s|\mathbf{c}^k) = s_d^r(s, \mathbf{c}^k)$ denote the value function approximation after the kth iteration of the simplex spline RLS API algorithm. As $k \to \infty$, $\hat{V}^k(s, \mathbf{c}^k)$ converges to the optimal value function $V^*(s)$.

Proof.
Bradtke and Barto (1996) showed that if the true optimal value function $V^*(s)$ of an MDP with a discrete state space can be expressed as a linear combination of the selected features, then the RLS API algorithm converges to $V^*(s)$. Ma and Powell (2009) extended this result to problems with a continuous state space. The function $\hat{V}^k(s|\mathbf{c}^k) = s_d^r(s, \mathbf{c})$ is defined on the triangulation $\mathcal{T}^k$. As $k \to \infty$, the triangulation $\mathcal{T}^k$ grows infinitely dense. Since each function can be expressed as a simplex spline of degree d and continuity order $\mathcal{C}^r$ defined on an infinitely dense triangulation, the simplex spline RLS API algorithm converges to the optimal value function. □

Chapter 5

Case studies

To gain insight into the performance of our methods, we first test the simplex spline dynamic programming algorithm from Section 4.2 on a problem concerning the control of water flows in a water supply reservoir system. Then, we test simplex spline RLS API algorithm on a problem of balancing an inverted pendulum.

5.1 Three-dimensional water reservoir problem

First of all, we test the performance of the simplex spline dynamic programming algorithm on a three-dimensional water supply reservoir problem described by Johnson et al. (1993). In this problem, three reservoirs are linked together to form a water supply system. Each period of time, a certain amount of water flows into reservoirs 1 and 2. The decision maker decides upon the amount of water to release from each of the reservoirs. The water released from reservoirs 1 and 2 flows into reservoir 3 and the water released from reservoir 3 flows out of the system. Each reservoir has a maximum capacity and a desired amount of water at the end of the horizon.

The water reservoir problem can be easily modelled as a Markov decision process with a continuous, three-dimensional state space $\mathcal{S}$ consisting of vectors (s_1, s_2, s_3), where s_i, $i = 1, 2, 3$ denotes the amount of water in reservoir i. Each time period t, an action $a(t) = (a_1(t), a_2(t), a_3(t))$ is chosen, where $a_i(t)$ denotes the amount of water that is released from reservoir i. We assume that only integer amounts of liters can be released from the reservoirs, so the action space $\mathcal{A}$ is finite and discrete. Let $S_i^{\max}(t)$ denote the

maximum storage of reservoir i at time t and let the inflows to reservoirs 1 and 2 at time t be denoted by $q_1(t)$ and $q_2(t)$ respectively. These inflows are independent, uniformly distributed random variables $q_1(t)$ and $q_2(t)$, respectively.
The state transition equations are as follows:

$$\begin{aligned} s_1(t+1) &= s_1(t) - a_1(t) + q_1(t) \\ s_2(t+1) &= s_2(t) - a_2(t) + q_2(t) \\ s_3(t+1) &= s_3(t) - a_3(t) + a_1(t) + a_2(t) \end{aligned} \tag{5.1}$$

where

$$\begin{aligned} 0 \leq s_i(t) \leq S_i^{\max}(t),\ i = 1,2,3;\ t = 1,...,T \\ A_i^{\min}(t) \leq a_i(t) \leq A_i^{\max},\ i = 1,2,3;\ t = 1,...T. \end{aligned} \tag{5.2}$$

The bounds on the decision variables are fully determined by the bounds on the storage of the reservoirs. The cost function for each period is given by

$$C[s(t), a(t)] = \sum_{i=1,2,3} C_i(t)[a_i(t) - 1]^2, \tag{5.3}$$

where $C_i(t)$ are given cost coefficients for reservoir i at time t. Let m_i, $i = 1, 2, 3$ denote the desired amount of water in reservoir i at the end of the horizon T. The final cost incurred at the end of the horizon is given by

$$C[s(t+1)] = \sum_{i=1,2,3} [s_i(T+1) - m_i]. \tag{5.4}$$

The goal in this problem is to minimize the total costs. Following Johnson et al. (1993), we specify the variables of the problem according to Table 5.1.

5.1.1 Performance on a fixed balanced triangulation

First of all, we attempt to gain some general insight into the performance of simplex spline value function approximations. To this end, we study the results of the simplex spline dynamic programming algorithm on a fixed, balanced triangulation. To construct this triangulation, we cover the state space hyperrectangle by a single grid with cell width h in each dimension. Figure 5.1 shows the resulting triangulation of the state space for $h = 5$.

Parameter	Value
$S^{\max}(t)$	$(12, 12, 12)$ for all t
$\mathbf{m}$	$(5, 5, 7)$
$\mathbf{C}(t)$	$(1.1, 1.2, 1.3)$ for all t
$\mathbf{q}(t)$	Independent uniformly distributed random variables on intervals (0,4) and (0,8) respectively.
T	3

Table 5.1: Specification of the parameters of the water reservoir problem

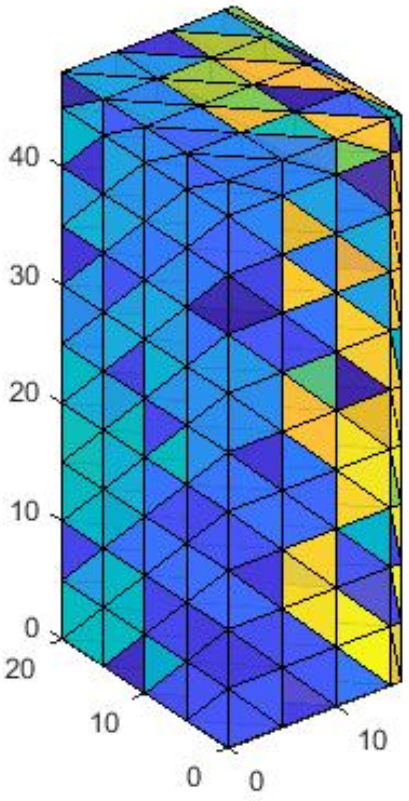


Figure 5.1: Fixed, balanced triangulation of the state space for $h = 5$. (The colours in this figure are just meant to clarify the shape of the triangulation and do not have any deeper meaning).

We compare the results of the simplex spline approximations to the results of the multilinear approximation and the approximation with tensor product splines, which were used by Johnson et al. (1993). For the multilinear approximation, we use the same triangulation as the one we use for the simplex splines. The value at a certain state is simply a linear combination of the values of the vertices of the triangulation, weighted with the barycentric coordinates of this state with respect to these vertices. For the tensor product spline approximation, we use the grid that underlies the triangulation on which the simplex splines are defined. The concept of a tensor product spline is explained in Section 2.2.2.1.

As a performance measure of a policy, we use the average total costs induced during the entire process after following this policy. To compute these average total costs, we construct a set of 50 arbitrarily chosen initial states. For each of these initial states, we generate a sequence of random information, that is, the value of the vector $\mathbf{q}$ for each time t. Then, we evaluate the total costs for each of these pairs of initial states and sequences of random information and take the average. For each experiment, we use the same initial states and the same vectors $\mathbf{q}$. The results are shown in Figure 5.2. The horizontal axis shows the cell width h and the vertical axis shows the average total costs induced after following the resulting policy. The corresponding computation times are shown in Figure 5.3.

Figure 5.2 indicates that the performance of simplex splines of degree 1 and continuity order $\mathcal{C}^0$ is similar to the performance of a multilinear approximation and an approximation with tensor product splines of degree 1 and continuity order $\mathcal{C}^0$. Simplex splines of degree 2 and continuity order $\mathcal{C}^1$ achieve the best results and outperform tensor product splines of the same degree and continuity order. Figure 5.3, however, shows that the computation times required by the simplex spline approximations is considerably higher than for the other approximation architectures and grows substantially as the cell width decreases. These high computation times are due to the fact that a lot of control states are necessary to apply the generalized least squares procedure, which implies a huge amount of evaluations of the simplex spline. Because of these high computation times, we seized the experimentation at this point and consider simplex spline dynamic programming not suitable for practical use. The results, however, do suggest that simplex splines can perform well as value function approximators, if we can limit the amount of evaluations of the splines.

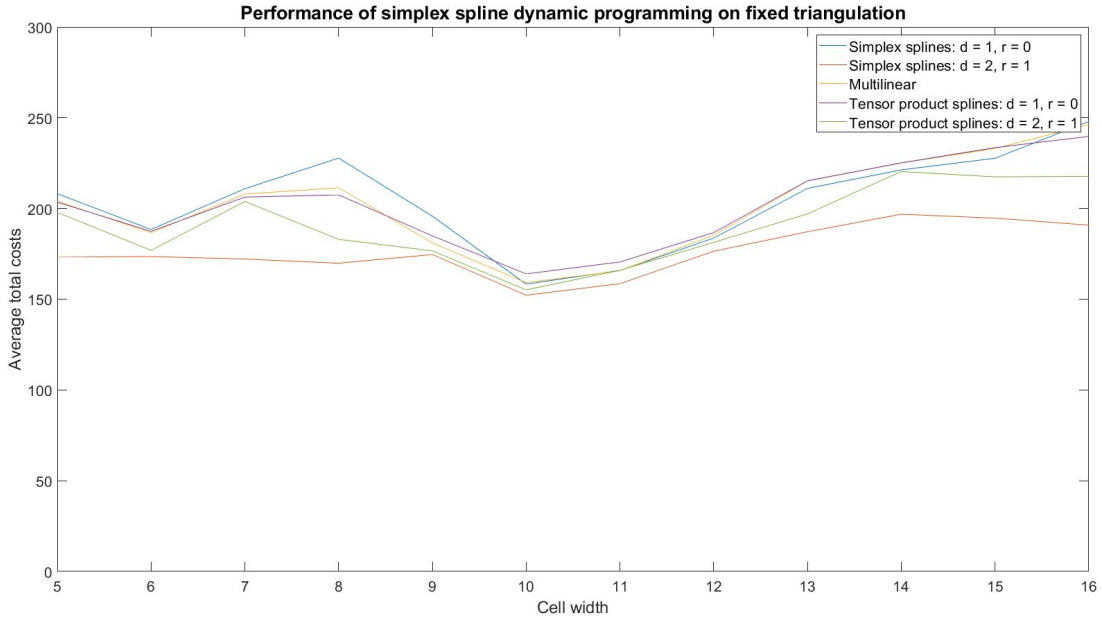


Figure 5.2: Results of the simplex spline dynamic programming algorithm on a fixed, balanced triangulation compared to a multilinear approximation and a tensor product spline approximation for different cell widths h.

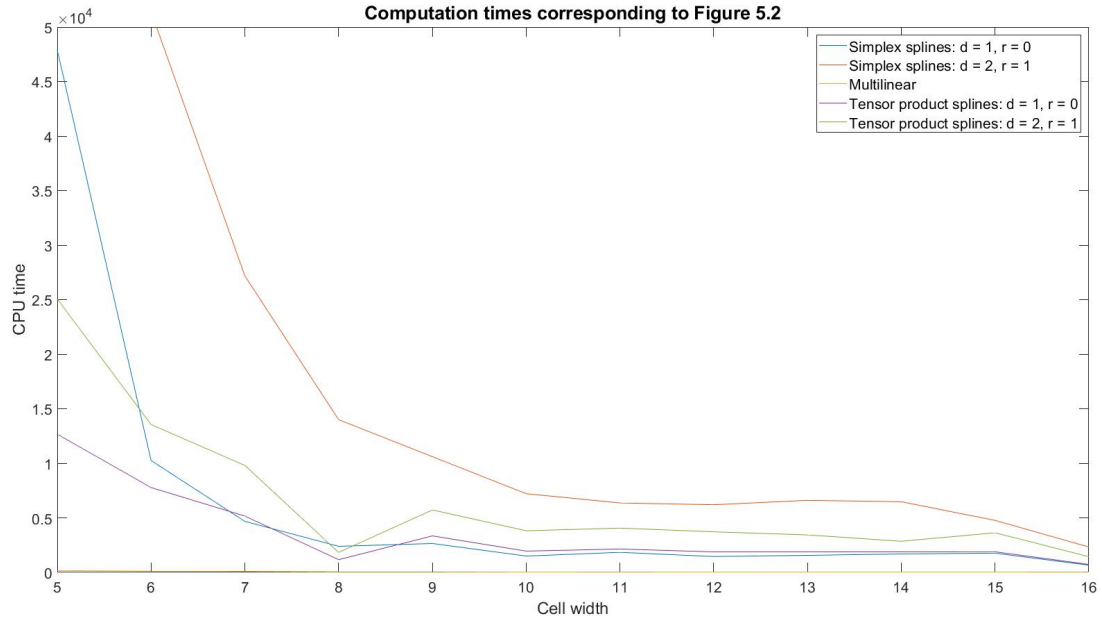


Figure 5.3: Computation times corresponding to Figure 5.2.

5.2 Inverted pendulum

To test the performance of the simplex spline RLS API algorithm, we study the problem of balancing an inverted pendulum that is attached to a cart. The weight of this pendulum is located above the cart, which renders the system unstable. The pendulum ought to be balanced by applying a torque at the cart. Figure 5.4 shows a schematic picture of the inverted pendulum problem.

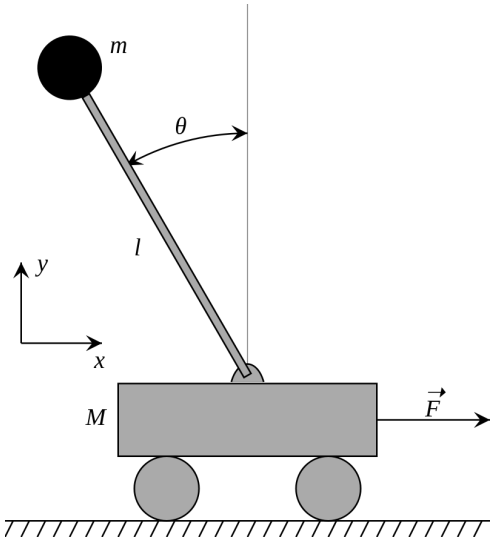


Figure 5.4: Schematic picture of the inverted pendulum problem

The system is governed by the following dynamics:

$$\ddot{\theta}(t) = \frac{g\sin(\theta) - \alpha ml(\dot{\theta})^2\sin(2\theta)/2 - \alpha\cos(\theta)(F+u)}{4l/3 - \alpha ml\cos^2(\theta)}, \tag{5.5}$$

where $\ddot{\theta}$ denotes the angular acceleration of the system, m the mass of the weight, g the gravitational acceleration, l the length of the pendulum, θ the angle of the pendulum with respect to the vertical axis, F the torque applied at the pivot point, and u a random disturbance of this torque. The parameter α is given by $\alpha = \dfrac{1}{m+M}$, where M denotes the mass of the cart. These dynamics are approximated using the discrete time Euler's approximation:

$$
\begin{aligned}
s_1^{k+1} &= s_1^k + h s_2^k \\
s_2^{k+1} &= s_2^k + h\left(\frac{g\sin(s_1^k) - \alpha ml (s_2^k)^2 \sin(2s_1^k)/2 - \alpha\cos(s_1^k)(F+u)}{4l/3 - \alpha ml\cos^2(s_1^k)}\right).
\end{aligned} \tag{5.6}
$$

Here, the pair (s_1, s_2) approximates $(\theta, \dot{\theta})$, where $|\theta| \leq \pi/2$ and $|\dot{\theta}| \leq \dot{\theta}_{\max}$, and $h > 0$ is a discretization step size parameter. Hence, the Markov decision process that models the inverted pendulum problem has state space $\mathcal{S} = \{(s_1, s_2) || s_1| \leq \pi/2,\ |s_2| \leq \dot{\theta}_{\max}\}$. Furthermore, we assume that only torques of -50, 0 and 50 Newton can be applied to the cart, which yields the action space $\mathcal{A} = \{-50, 0, 50\}$. The state transition equations are given by (5.6). The disturbance u of the torque is a uniformly distributed random variable on the interval $(-10, 10)$. Finally, the process has the following reward function

$$
r(s, F) = \begin{cases} 1 - \max\left\{\left|\dfrac{s_1}{K_1}\right|, \left|\dfrac{s_2}{K_2}\right|\right\}, & \text{if } |s_1| \leq K_1 \text{ and } |s_2| \leq K_2, \\ 0 \text{ otherwise,} \end{cases} \tag{5.7}
$$

where $K_1 \leq \pi/2$ and $K_2 \leq \dot{\theta}_{\max}$. The aim of the MDP is to maximize the discounted rewards over an infinite horizon, with discount factor γ. The parameters are chosen in accordance with Table 5.2.

Parameter	Value
m	2
g	9.81
l	0.5
M	8
h	0.1
$\dot{\theta}_{\max}$	7
K_1	$\pi/2 - 0.5$
K_2	1.5
γ	0.95

Table 5.2: Specification of the parameters of the inverted pendulum problem

5.2.1 Performance on a fixed, balanced triangulation

As for the simplex spline dynamic programming algorithm, we first test the simplex spline RLS API algorithm for a fixed, balanced triangulation. To construct this triangulation, we cover the state space hyperrectangle by a grid of cell width $\pi/6$ and height 2. Then, we triangulate the vertex set constituted by the vertices of the gridcells and the midpoints of the gridcells with a Delaunay triangulation. The result is shown in Figure 5.5

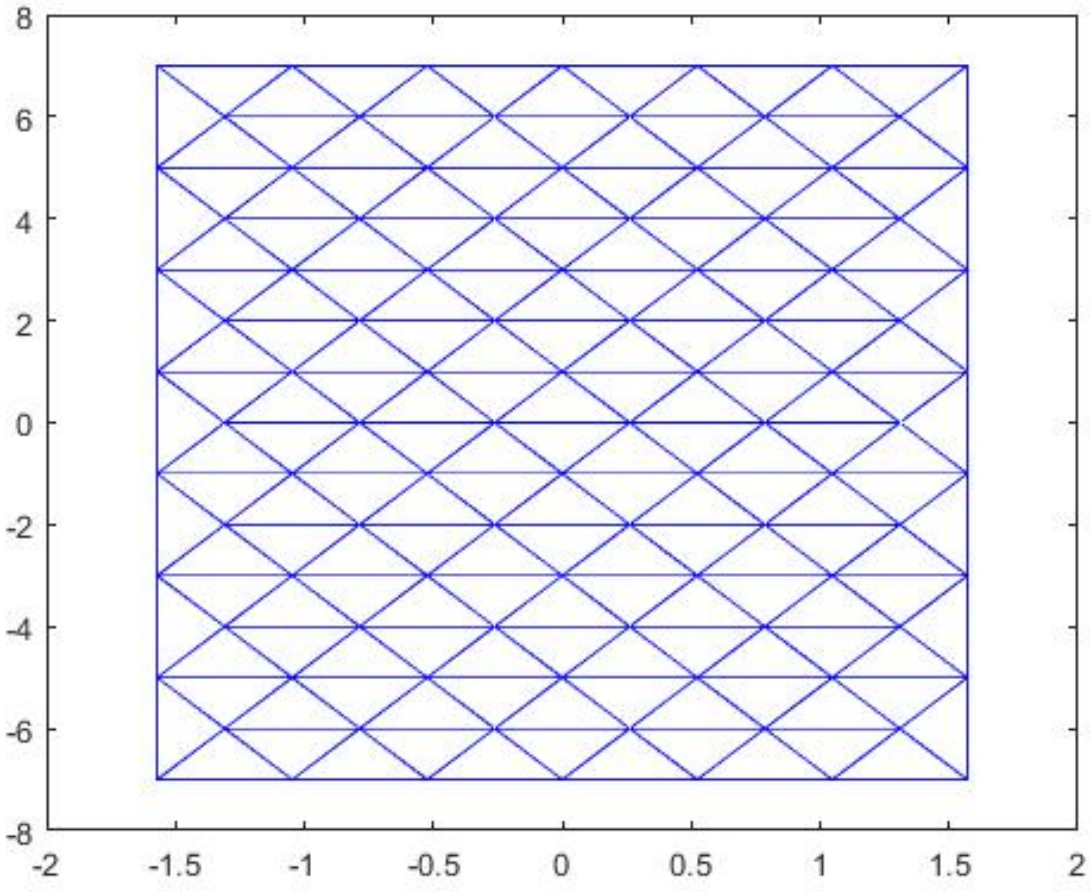


Figure 5.5: Fixed, balanced triangulation

We compare the results of the RLS API algorithm with simplex spline value function approximation on the triangulation of Figure 5.5 to the results obtained by Lagoudakis and Parr (2003). The latter use the so-called LSPI algorithm to learn the state-action value function $Q(s, a)$, which is quite similar to the RLS API algorithm. The main

difference is that the LSPI algorithm, unlike the RLS API algorithm, is a model-free method, that is, it does not make use of the transition probabilities. Instead of simplex splines, Lagoudakis and Parr (2003) use a set of 10 basis functions for each of the actions to approximate the value function, amounting to a total of 30 basis functions. Each set consists of 1 constant and 9 radial basis functions, which are given by

$$e^{\frac{||s-\mu_i||^2}{2\sigma^2}}, \ i = 1, 2, ..., 9, \tag{5.8}$$

where the μ_i's were chosen to be the 9 vertices of the grid $\{\pi/4, 0, \pi/4\} \times \{-1, 0, 1\}$ and $\sigma^2 = 1$.

We conduct all experiments in this section in the same manner: each experiment is carried out 100 times. For each of these runs, we evaluate the policy 10 times after each iteration. As a performance measure of a policy, we use the number of time steps that passes before the pendulum falls down, when it starts in a balanced position ($s = (0, 0)$). If the pendulum stays upright for 3.000 timesteps, which is equivalent to a period of 5 minutes, we abort the episode and consider it successful. An experiment is considered successful if it manages to find a policy that leads to a successful episode within 30 iterations and the performance does not deteriorate over the next 10 iterations.
To assess the quality of an algorithm, we use two performance measures: the proportion of the experiments that were successful and the average CPU time before a successful policy was found, which was maintained over the next 10 iterations. If the number of successful experiments is greater than 40, we also give 95%-confidence intervals for the computation times based on the results of these experiments. The experiments are run on a AMD Ryzen 5 PRO 4650U with 8 GB ram.

First, we set out to find a suitable sample size for the LSPI algorithm. To this end, we examine the performance of the algorithm for several different sample sizes. The results are shown in Table 5.3.
Table 5.3 shows that a sample size of 12.000 is sufficient to find the desired policy. As we seem to achieve the best performance with 14.000 samples, we select a sample size of 14.000.

We proceed to select a suitable sample path length for the simplex spline RLS API

Sample size	Success proportion	Mean success time	95% CI for mean success time
500	0	-	-
1000	0	-	-
5000	0.25	3.1678	[-1.7689, 8.1045]
10.000	0.95	1.0304	[0.8734, 1.1873]
12.000	1.00	0.9940	[0.7255, 1.2625]
14.000	1.00	0.7681	[0.6636, 0.8726]
16.000	1.00	0.9686	[0.8259, 1.1113]
18.000	1.00	1.0131	[0.7075, 1.3187]
20.000	1.00	1.5740	[1.2684 1.8796]

Table 5.3: Results of the LSPI algorithm for different sample sizes

algorithm. To this end, we run the algorithm for several different sample path lengths. We use simplex splines of degree 1 and continuity order $\mathcal{C}^0$ defined on the fixed balanced triangulation of Figure 5.5. The results are shown in Table 5.4.

Sample path length	Success proportion	Mean success time	95% CI for mean success time
1	1.00	56.6017	[53.7539, 59.4496]
2	1.00	62.3884	[58.8571, 65.9198]
3	1.00	65.9482	[64.4077, 67.4887]
4	1.00	63.4551	[62.6832, 64.2270]
5	1.00	54.4207	[49.6266, 59.2149]
6	1.00	85.6815	[84.7159, 86.6471]
10	1.00	78.1120	[73.7872, 82.4368]

Table 5.4: Results of the simplex spline RLS API algorithm on a fixed, balanced triangulation for splines of degree 1 and continuity order $\mathcal{C}^0$ for different sample path lengths.

The table shows that the algorithm is able to find a successful policy for each of the sample path lengths. A sample path length of 5 achieves the best performance. Hence, for the remaining experiments in this section, we use a sample path length of 5.

Now, we study the results of the RLS API algorithm for splines of several different degrees and continuity orders, again using the fixed triangulation of Figure 5.5. The results are

shown in Table 5.5.

Spline type	Success proportion	Mean success time	95% CI for mean success time
$d = 1$, $r = 0$	1.00	56.6017	[53.7539, 59.4496]
$d = 2$, $r = 0$	1.00	368.9745	[355.2413, 382.7077]
$d = 2$, $r = 1$	1.00	224.3646	[211.9566, 236.7726]
$d = 3$, $r = 0$	0.93	610.7972	[564.0021, 657.5923]
$d = 3$, $r = 1$	1.00	679.9640	[635.9592, 723.9688]
$d = 3$, $r = 2$	1.00	585.8786	[529.0829, 642.6743]

Table 5.5: Results of the simplex spline RLS API algorithm on a fixed, balanced triangulation for splines of different degrees and continuity orders

Except for splines of degree 3 and continuity order $\mathcal{C}^0$, all spline types achieve a success proportion of 100 %. For splines of degree 3 and continuity order $\mathcal{C}^0$, some of the experiments failed. This is probably due to the fact that splines of higher degree are more prone to overfitting and the continuity order is too low to provide enough numerical stability. Splines of type 1 and continuity order $\mathcal{C}^0$ decidedly achieve the best results. We continue to use this type of splines in the experiments to come.

5.2.2 Performance of the triangulation refinement procedure

We proceed to study the performance of the simplex spline RLS API algorithm with the triangulation refinement procedure. In scoring the directions, we omit the final two terms of expression (4.51), because they take a long time to compute. Hence, we only use the criteria that belong to β_1 and β_2. First of all, we set the parameters to $\alpha_i = 1$, $i = 1, 2, 3, 4$ and $\beta_i = 1$, $i = 1, 2$. We compare the results of the triangulation refinement procedure with the performance of two other refinement methods: the gradual refinement method and the random refinement method. In the gradual refinement method, we refine a randomly chosen cell from the set of largest cells. Hence, we gradually work towards the fixed, balanced triangulation of Figure 5.5. In the random refinement method, we simply refine a random cell along a random direction each iteration. The results for each of the refinement methods are shown in Table 5.6.

Refinement method	Succes proportion	Mean success time	95% CI for mean succes time
Triangulation refinement procedure	0.73	48.4889	[29.6171, 67.3607]
Gradual refinement method	0.91	83.1480	[64.2724, 102.0236]
Random refinement method	0.52	196.4601	[146.0773, 246.8429]
Fixed, balanced triangulation	1.00	56.6017	[53.7539, 59.4496]
LSPI	1.00	0.7681	[0.6636, 0.8726]

Table 5.6: Comparison of triangulation refinement procedure with gradual and random refinement methods.

Table 5.6 indicates that the triangulation refinement procedure achieves good results. When it comes to the mean success time, it outperforms both the other refinement methods and the algorithm on a fixed, balanced triangulation. Note, though, that the confidence interval for the success time is much narrower for the fixed triangulation. This implies that the triangulation refinement procedure might be slower in some cases.
The second column shows that the triangulation refinement procedure is not guaranteed to find the required policy. Only 73% of the experiments were successful. As to the gradual refinement method, 91% of the experiments resulted into the desired policy, although the computation times were slightly higher than for the triangulation refinement procedure. The random refinement method appears to perform poorly: only 52% of the experiments were successful and the computation times were on average considerably higher than for the other methods. The LSPI algorithm of Lagoudakis and Parr (2003) outperforms all versions of the simplex spline RLS API algorithm. Note, though, that the comparison between these two algorithms is not entirely fair. Firstly, the radial basis functions that were used by Lagoudakis and Parr (2003) were hand-crafted and specifically chosen for this particular problem. The simplex splines, on the other hand, were not in any way predetermined and are suitable for any problem, which is a big advantage of the simplex spline RLS API algorithm over LSPI.

Another downside of the LSPI algorithm is the fact that it learns a state-action value function instead of a state value function, and thus requires a distinct set of basis functions for each action. For the inverted pendulum problem, which has a very small action space, this does not cause any trouble. For problems with a larger action space, on the other hand, we expect a significant increase in computation time.

That the triangulation refinement procedure does not always succeed, we ascribe to two possible reasons. The first problem is that refining a grid cell does not only affect the triangulation in this particular cell, but also in other parts of the state space. In many cases, this effect is limited to only the close neighborhood of the refined cell, but it may occur that the refinement has a more substantial impact on the triangulation. In areas different from the selected region, the new triangulation might not be as suitable as the previous one, which causes the performance to deteriorate. Also, it may be hard to find a new simplex spline that resembles the one defined on the previous triangulation, which prevents a smooth continuation of the learning process.
The second problem is that the triangulation refinement procedure tends to construct simplices that are not "good" according to the metrics discussed in Section 3.2.1.1. As mentioned in that section, the high flexibility of the Delaunay triangulation comes at the price of the quality of the simplices. "Ugly" simplices, like simplex fans and sliver simplices, reduce the approximation power of the simplex spline and slow down the process. Figure 5.6 and Figure 5.7 show two triangulations that resulted from 30 iterations of the RLS API algorithm with the triangulation refinement procedure. The experiment corresponding to the triangulation in Figure 5.6 was successful, whereas the triangulation in Figure 5.7 did not yield the desired policy.
In Figure 5.6, the triangulation is most dense at the regions where the pendulum tends to fall down, namely for angles roughly between $-\pi/2$ and -1 and 1 and $\pi/2$ and for angular velocities between -4 and -2 and 2 and 4. This makes sense, as these are the regions in which the decisions of the decision maker have a crucial impact on the future of the process.
Compared to the triangulation in Figure 5.6, the triangulation in Figure 5.7 exhibits quite some "ugly" simplices. Also, the triangulation is very fine in a small region at the right side of the state space, whereas other possibly important parts are completely neglected. We suspect that the triangulation yields a value function approximation that is off to such an extent that the refinement procedure is unable to provide the grid cells with sensible

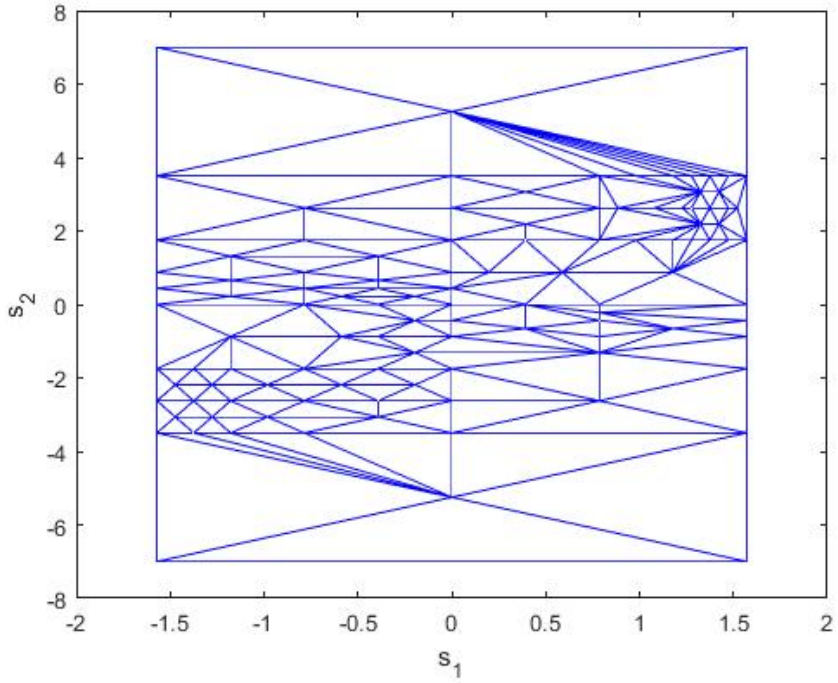


Figure 5.6: Triangulation corresponding to successful experiment

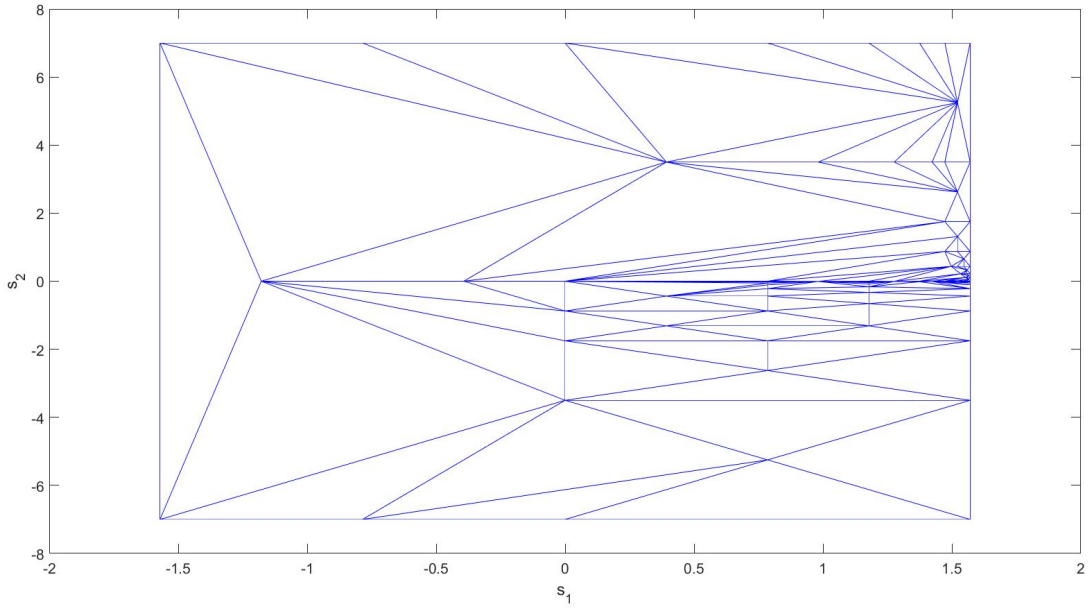


Figure 5.7: Triangulation corresponding to unsuccessful experiment

scores. This might cause the algorithm to follow a wrong track.

We proceed to take a closer look at the refinement criteria. We again run the simplex spline RLS API algorithm with the triangulation refinement procedure, but this time using only one or two of the refinement criteria. Hence, we set one or two of the α_i's to one and the remaining ones to zero. The results are displayed in Table 5.7.

$\boldsymbol{\alpha}$	Success proportion	Mean success time	95% CI for mean success time
(1, 0, 0, 0)	0.72	25.6123	[21.2919, 29.9327]
(0, 1, 0, 0)	0.77	51.8112	[41.4898, 62.1327]
(0, 0, 1, 0)	0.41	67.8759	[40.6670, 95.0848]
(0, 0, 0, 1)	0.46	43.8625	[27.9841, 59.7409]
(1, 1, 0, 0)	0.76	49.2496	[36.6144, 61.8847]
(1, 0, 1, 0)	0.83	53.9809	[44.4930, 63.4688]
(1, 0, 0, 1)	0.56	55.4102	[40.5228, 70.2977]
(0, 1, 1, 0)	0.78	59.1378	[44.7831, 73.4924]
(0, 1, 0, 1)	0.72	63.8601	[47.9000, 79.8203]
(0, 0, 1, 1)	0.54	63.6690	[48.4460, 78.8920]
$(1, 1, 1, 1)$	0.73	48.4889	[29.6171, 67.3607]

Table 5.7: Results for combinations of one or two parameters.

It follows from Table 5.7 that the maximum value difference criterion ($(\boldsymbol{\alpha} = (1, 0, 0, 0)$) performs quite well on its own. It achieves almost the same success proportion as all refinement criteria together and requires considerably less computation time. This is probably due to the fact that this criterion does not involve the time-consuming computation of the region transition probabilities and the influences. The value variance ($\boldsymbol{\alpha} = (0, 1, 0, 0)$) achieves a higher success proportion than all criteria together. The mean success time is slightly higher than the one that corresponds to $\boldsymbol{\alpha} = (1, 1, 1, 1)$, but the confidence interval is narrower. This suggests that the algorithm using only the value variance possibly performs better than the algorithm using all refinement criteria in many cases. A combination of the maximum value difference and the value variance, ($\boldsymbol{\alpha} = (1, 1, 0, 0)$) achieves a similar performance. The visitation frequency ($\boldsymbol{\alpha} = (0, 0, 1, 0)$) and error contribution criteria ($\boldsymbol{\alpha} = (0, 0, 0, 1)$) perform poorly on their own. Using these criteria, the

algorithm succeeded in only 41% and 46% of the cases respectively. Combining the visitation frequency with the maximum value difference or the value variance ($\boldsymbol{\alpha} = (1, 0, 1, 0)$ or $\boldsymbol{\alpha} = (0, 1, 1, 0)$, respectively), however, proves to result in a high success proportion. The best performance, in terms of probability of success, is achieved by a combination of the maximum value difference and the visitation frequency ($\boldsymbol{\alpha} = (1, 0, 1, 0)$), which yields a success proportion of 83%. The mean success time of this combination is comparable to the mean success time that belongs to $\boldsymbol{\alpha} = (1, 1, 1, 1)$. The corresponding confidence interval, however, is again narrower.
Table 5.7 suggests that the error contribution criterion is not of any benefit to the results of the triangulation refinement procedure. To determine whether this criterion is truly useless and to gain more insight in the influence of the remaining criteria, more extensive experimentation is necessary. Furthermore, as the performance of the criteria is likely to be problem dependent, the simplex spline RLS API algorithm should also be tested on a number of different problems.

5.3 Remarks on the implementation

While implementing the simplex spline dynamic programming and simplex spline RLS API algorithms, we ran into some technical difficulties. In this section, we discuss these issues and explain how we attempted to solve or circumvent them.

5.3.1 Computing the transition probabilities

Suppose we have a value function approximation $\hat{V}^k$ and a corresponding greedy policy π_k. To find the action we should take according to this policy from a certain state s, we need to compute the expression

$$C(s, a) + \gamma \int_{\mathcal{S}} \mathcal{F}(s, a, s') \hat{V}^k(s') ds' \tag{5.9}$$

for each action $a \in \mathcal{A}$. It is generally time-consuming to find the exact (or nearly exact) value of the integral. Therefore, we approximate this term in the following way: first of all, we construct a partition $\{A_1, ..., A_N\}$ of the state space. Then, we compute the probabilities

$$P(S_{t+1} \in A_i | S_t = s, a_t = a) = \int_{A_i} \mathcal{F}(s, a, s')ds'. \tag{5.10}$$

The partition is chosen in such a way that the integral in expression (5.10) is easy to evaluate. We proceed to pick a random state $s_i \in A_i$ for each $i = 1, ..., N$. Finally, we approximate expression (5.9) by

$$C(s, a) + \gamma \sum_{i=1}^{N} P(S_{t+1} \in A_i | S_t = s, a_t = a)\hat{V}^k(s_i). \tag{5.11}$$

5.3.2 Computing the region transition probabilities

The region transition probability $P^\pi(a_j^{t+1}|a_i^t)$, or the probability that the process makes a transition from a region a_i^t to region a_j^{t+1} under a policy π, is given by

$$P^\pi(a_j^{t+1}|a_i^t) = \frac{P^\pi(s_{t+1} \in a_j^{t+1} \text{ and } s_t \in a_i^t)}{P^\pi(s_t \in a_i^t)}. \tag{5.12}$$

As this expression is extremely hard to compute, we estimate the region transition probabilities in the following way: we pick a large number of M randomly chosen states in region a_i^t. From each of these states, we make a transition according to policy π. Let x be the number of times we end up in region a_j^{t+1} after such a transition. The region transition probability $P^\pi(a_j^{t+1}|a_i^t)$ is estimated as follows:

$$\hat{P}^\pi(a_j^{t+1}|a_i^t) = \frac{x}{M}. \tag{5.13}$$

Chapter 6

Conclusions and recommendations

6.1 Conclusions

The aim of this thesis was to study the use of splines in the context of approximate dynamic programming. This aim is captured by the following main research question, which was stated in Chapter 1:

Are splines suitable candidates for value function approximation in ADP and how can we design an ADP framework that exploits their flexibility?

In this section, we present the main conclusions of our work. In answering the question above, we use the five subquestions formulated in Chapter 1 as a guidance.

1. Which type of splines is most suitable for value function approximation?

In Section 2.2, we examined several popular types of multivariate splines that might come of use in the context of approximate dynamic programming. From our findings, we concluded that the multivariate simplex spline is most suitable for this purpose. The fact that it is linear in its parameters, provides the multivariate simplex spline with the advantages of linear parametric models. Furthermore, it can achieve a high approximation power and its stable local basis property allows for the use of efficient computational schemes. Also, the fact that it is defined on a triangulation renders the simplex spline very flexible. After all, there exists a wide variety of possible triangulations which are

easily refined or simplified. These characteristics are big advantages of multivariate simplex splines over other parametric models. Moreover, as nonparametric models, simplex splines do not involve any preprocessing using problem-specific knowledge. Based on these findings, we concluded that the multivariate simplex spline is indeed a suitable candidate for value function approximation.

2. How can we identify "important" regions of the state space, that is, regions that require the most accurate value function approximation?
3. How can we adapt the splines in such a way that they provide a more accurate value function approximation in regions where this is required?

In Chapter 4, we designed a method that adaptively refines the triangulation on which simplex splines are defined in regions of the state space that require a more accurate approximation of the value function. In this triangulation refinement procedure, a triangulation is built on a partition of the state space into local grids. To refine this triangulation, we iteratively refine these underlying grids. Each iteration, we give each of the grid cells a score that is meant to quantify the "importance" of this cell. This score is based on four refinement criteria: the maximum value difference, the value variance, the visitation frequency and the error contribution. After selecting the cell with the highest score, we choose a suitable refinement direction and refine the selected grid cell along this direction. Finally, we build a new triangulation on the obtained new set of local grids.

4. How can we integrate such an adaptive procedure into an ADP framework?

We integrated the triangulation refinement procedure first of all in the basic backward dynamic programming algorithm, giving rise to simplex spline dynamic programming. Then, we embedded the procedure into the RLS API algorithm described by Powell (2011). To do so, we replaced the original least squares procedure in this algorithm by an equality constrained least squares estimator designed by de Visser et al. (2011).

5. How well does the resulting ADP algorithm based on adaptive splines perform?

In Chapter 5, we tested the simplex spline dynamic programming algorithm on a three-dimensional water reservoir problem and the simplex spline RLS API algorithm on a

problem of balancing an inverted pendulum. The simplex spline dynamic programming algorithm proved to be very time-consuming: it requires a high amount of control states for each time t, which implies a high amount of evaluations of the simplex spline. We therefore do not consider this algorithm suitable for practical use and only tested it for a fixed triangulation.

The simplex spline RLS API algorithm, on the other hand, achieved good results, which demonstrate the advantage of the triangulation refinement procedure over a fixed, balanced triangulation. The RLS API algorithm with the triangulation refinement procedure proved to require on average less time to find the desired policy than the algorithm on a fixed triangulation. Furthermore, the triangulation refinement procedure was shown to outperform a gradual and a random refinement method.

Although the triangulation refinement procedure proved to reduce the computation time, the results in Chapter 5 also indicate that it is not guaranteed to succeed. We ascribe this possiblity of failure to two problems: first of all, refinement of a grid cell might have a substantial impact on the triangulation in parts of the state space that are not in the neighborhood of this grid cell. Although the approximation power in the selected area may have improved, the triangulation in other regions may not be as suitable as the previous one, which causes the overall performance to deteriorate. Also, it may be difficult to find a simplex spline on the new triangulation that resembles the previous simplex spline, which complicates a smooth continuation of the learning process.

The second problem is that the triangulation refinement procedure tends to construct simplices that are not "good" in terms of the metrics discussed in Section 3.2.1.1. "Ugly" simplices, like simplex fans and sliver simplices, frequently occur in Delaunay triangulations. These simplices reduce the approximation power of the simplex spline and slow down the algorithm.

We compared the simplex spline RLS API algorithm to the LSPI algorithm of Lagoudakis and Parr (2003), which is similar to RLS API, but uses radial basis functions as approximation architecture. Although, the LSPI algorithm outperformed all versions of the RLS API algorithm, the performances of these algorithms do not differ too much. Also note that a downside of LSPI compared to our algorithm is the fact that the radial basis functions that are used as value function approximators were specifically tailored to the inverted pendulum problem in advance, whereas the simplex splines do not require any predetermination based on knowledge of the problem. Another disadvantage of the LSPI algorithm is the fact that it learns a state-action value function instead of a state value

function. This implies that it requires a distinct set of basis functions for each action, which might cause inefficiency for problems with a larger action space. The RLS API algorithm, on the other hand, uses the same amount of simplex splines for each possible size of the action space.

All in all, our findings prove that multivariate simplex splines have great potential as value function approximators. In answer to our main research question, we developed a method that shows promise in the context of value function approximation in ADP. It can be applied to any MDP with a continuous state space and finite, discrete action space, without any preadjustments of the splines using problem-specific knowledge. For the inverted pendulum problem, the performance is not far behind that of an algorithm that was in fact tuned to this problem in advance. Moreover, we see various possibilities for further improvement of our method. We elaborate on these possibilites in Section 6.2.

6.2 Recommendations

Due to the many time-consuming experiments, we have not been able to test the simplex spline RLS API as thoroughly as we wanted to. To gain true insight into the triangulation refinement procedure and the impact of each of the refinement criteria, more experimentation is required. Also, the algorithm should be tested on various other problems, besides the inverted pendulum problem, and ought to be compared to a wide range of other algorithms, including methods that make use of nonparametric models. Furthermore, we are convinced that the algorithm has a high potential to achieve better results than the ones presented in this thesis. In the final weeks of this project, some small modifications to the algorithm as well as to the implementation brought about considerable improvements of the computation times. In the remainder of this section, we sketch some ideas that could lead to an even better performance.

One of the aspects of the simplex spline RLS API algorithm from which there is still, in our view, a lot to gain, is the transition between different triangulations. As we mentioned before, the refinement of a grid cell may have a considerable effect on the triangulation in regions where this is not desirable, which complicates a smooth transition to the next iteration. In an attempt to smoothen these transitions, constrained Delaunay triangulation methods might come of use, which we briefly discussed in Section 3.2.1.2. We did

some quick experiments with these methods, but sofar without success: the ability to fix a part of triangulation went at the expense of the quality of the simplices. The triangulations we obtained throughout the course of the algorithm were increasingly ugly in terms of the metrics discussed in Section 3.2.1.1 and did therefore not yield good results. We do, however, think that this is a worthwile direction of further research. After all, fixing a part of the triangulation during the transition is not only beneficial for the stability of the algorithm, but also easens the computation of the coefficients of the new simplex spline: it might enable us to keep a set of coefficients fixed and only adapt the ones that are located in the surroundings of the refined grid cell. As the generalized least squares procedure is a time-consuming part of the algorithm, this might save a lot of computation time.

Another aspect of the simplex spline RLS API algorithm that has a negative impact on its performance is the frequent occurrence of bad simplices, like simplex fans or sliver simplices. Such simplices decrease the approximation power of the spline and slacken the speed of the process. Preventing the occurrence of ugly simplices is not an easy task and is a general problem in the field of triangulation optimization. Perhaps an additional refinement criterion could be introduced in the refinement procedure that concerns the quality of the triangulation. Introduction of another criterion, however, also implies an increase in computation time. Also, the potential of other triangulations than the Delaunay triangulation could be investigated.

Another aim of future research could be the design of a triangulation optimization method that is not based on grids. The reason we based our triangulations on a set of local grids was the fact that it allowed for an easy definition of "refinement" of a part of the state space and that grid cells are easy to work with. Also, we hoped to reduce the number of ugly simplices by positioning the vertices in a somewhat evenly distributed manner. Other ways of finding suitable locations for new vertices in the state space might yield a better performance.
In addition, a triangulation optimization method does not need to be limited to addition of vertices. More sophisticated procedures could be designed that allow for the removal and/or shifting of vertices.

All in all, triangulation optimization is a challenging endeavour. It has been extensively studied for the purpose of data approximation and is still an active area of research. In

the context of value function approximation in Markov decision theory, the problem is even more difficult, as the triangulation can only be adapted based on the current approximation and observation of the process. To our knowledge, this is the first work that deals with triangulation optimization in the context of value function approximation with simplex splines. The fact that we obtained quite good results, despite some weaknesses in our method, is a promise of the great potential of multivariate simplex splines in the context of approximate dynamic programming.

Bibliography

[1] O. Aichholzer, F. Hurtado, and M. Noy. A lower bound on the number of triangulations of planar point sets. *Computational Geometry*, 29(2):135–145, 2004.

[2] A. Barreto, D. Borsa, J. Quan, T. Schaul, D. Silver, M. Hessel, D. Mankowitz, A. Zidek, and R. Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 501–510, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[3] R. Bellman. *Dynamic Programming*. Dover Publications, 1957.

[4] R. Bellman, R. Kalaba, and B. Kotkin. Polynomial approximation–a new computational technique in dynamic programming: Allocation processes. *Mathematics of Computation*, 17(82):155–161, 1963.

[5] P.E. Bézier. Example of an existing system in the motor industry: the unisurf system. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 321:207 – 218, 1971.

[6] S. Bradtke and A. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22:33–57, 03 1996.

[7] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1):107–117, 1998. Proceedings of the Seventh International World Wide Web Conference.

[8] C. Cervellera and D. Macciò. A novel approach for sampling in approximate dynamic programming based on f -discrepancy. *IEEE Transactions on Cybernetics*, 47(10):3355–3366, 2017.

[9] V. Chen, D. Ruppert, and C. Shoemaker. Applying experimental design and regression splines to high-dimensional continuous-state stochastic dynamic programming. *Operations Research*, 47:38–53, 02 1999.

[10] V. C. P. Chen, D. Ruppert, and C.A. Shoemaker. Applying experimental design and regression splines to high-dimensional continuous-state stochastic dynamic programming. *Operations Research*, 47(1):38–53, 1999.

[11] L.P. Chew. Constrained delaunay triangulations. In *Proceedings of the Third Annual Symposium on Computational Geometry*, SCG '87, page 215–222, New York, NY, USA, 1987. Association for Computing Machinery.

[12] A. Cohen, N. Dyn, F. Hecht, and J. Mirebeau. Adaptive multiresolution analysis based on anisotropic triangulations. *Mathematics of Computation - Math. Comput.*, 81, 01 2011.

[13] R. Coulom. *Reinforcement Learning Using Neural Networks, with Applications to Motor Control*. PhD thesis, Institut National Polytechnique de Grenoble, 2002.

[14] M. Cox. The numerical evaluation of b-splines. *Ima Journal of Applied Mathematics*, 10:134–149, 1972.

[15] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd edition, 2008.

[16] C. de boor. On calculating with b-splines. *Journal of Approximation Theory*, 6(1):50–62, 1972.

[17] C. de Boor. Splines as linear combinations of b-splines. *Approximation Theory II*, 01 1976.

[18] C. de Boor. *A Practical Guide to Splines*, volume 27. 01 1978.

[19] C. de Boor. B-form basics. in: G. farin, editor, geometric modeling: Algorithms and new trends. *SIAM*, pages 131–148, 01 1987.

[20] C. de Boor and A. Ron. On multivariate polynomial interpolation. *Constructive Approximation*, 6:287–302, 09 1990.

[21] C. de Visser. *Global Nonlinear Model Identification with Multivariate Splines*. PhD thesis, TU Delft, 07 2011.

[22] C. de Visser, E. Van Kampen, Q. Chu, and J.A. Mulder. Intersplines: A new approach to globally optimal multivariate splines using interval analysis. *Reliable Computing*, 17:153–191, 12 2012.

[23] C.C. de Visser, Q.P. Chu, and J.A. Mulder. A new approach to linear regression with multivariate splines. *Automatica*, 45(12):2903–2909, 2009.

[24] B. Delaunay. Sur la sphère vide. a la mémoire de georges voronoi. *Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et na*, (6):793–800, 1934.

[25] T. Dey, C. Bajaj, and K. Sugihara. On good triangulations in three dimensions. *International Journal of Computational Geometry and Applications*, 2:431–441, 01 1991.

[26] T. Dokken, V. Skytt, and O. Barrowclough. Trivariate spline representations for computer aided design and additive manufacturing. *Computers and Mathematics with Applications*, 78, 03 2018.

[27] G. Farin. Triangular bernstein-bézier patches. *Computer Aided Geometric Design*, 3(2):83–127, 1986.

[28] G. Farin. Chapter 1 - a history of curves and surfaces in cagd. In F. Gerald, H. Josef, and K. Myung-Soo, editors, *Handbook of Computer Aided Geometric Design*, pages 1–21. North-Holland, Amsterdam, 2002.

[29] J.H. Friedman. Multivariate Adaptive Regression Splines. *The Annals of Statistics*, 19(1):1 – 67, 1991.

[30] V. Gabillon, M. Ghavamzadeh, and B. Scherrer. Approximate dynamic programming finally performs well in the game of tetris. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 1754–1762, Red Hook, NY, USA, 2013. Curran Associates Inc.

[31] T. N. T. Goodman. *Polyhedral Splines*, pages 347–382. Springer Netherlands, Dordrecht, 1990.

[32] A. Grover and J. Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 855–864, New York, NY, USA, 2016. Association for Computing Machinery.

[33] W.W. Hager. Condition estimates. *SIAM Journal on Scientific and Statistical Computing*, 5(2):311–316, 1984.

[34] P.J.H. Hulshof, M.R.K. Mes, R.J. Boucherie, and E.W. Hans. Patient admission planning using approximate dynamic programming. *Flexible services and manufacturing journal*, 28(1):30–61, 04 2016.

[35] S.A. Johnson, J.R. Stedinger, C.A. Shoemaker, Y. Li, and J.A. Tejada-Guibert. Numerical solution of continuous-state dynamic programs using linear and spline interpolation. *Operations Research*, 41(3):484–500, 1993.

[36] R. Karagoz and K. Batselier. Nonlinear system identification with regularized tensor network b-splines. *Automatica*, 122:109300, 12 2020.

[37] G. Konidaris, S. Osentoski, and P. Thomas. Value function approximation in reinforcement learning using the fourier basis. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, AAAI'11, page 380–385. AAAI Press, 2011.

[38] M.G. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, December 2003.

[39] M. Lai. Scattered data interpolation and approximation using bivariate c1 piecewise cubic polynomials. *Computer Aided Geometric Design*, 13(1):81–88, 1996.

[40] M. Lai and L. Schumaker. On the approximation power of bivariate splines. *Advances in Computational Mathematics*, 9:251–279, 1998.

[41] M. Lai and L. Schumaker. On the approximation power of splines on triangulated quadrangulations. *SIAM Journal on Numerical Analysis*, 36, 02 2001.

[42] M. Lai and L. L. Schumaker. Scattered data interpolation using c2 supersplines of degree six. *SIAM Journal on Numerical Analysis*, 34(3):905–921, June 1997.

[43] M. Lai and L.L. Schumaker. *Spline Functions on Triangulations*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2007.

[44] Lai, M. Geometric interpretation of smoothness conditions of triangular polynomial patches. *Computer Aided Geometric Design*, 14(2):191–199, 1997.

[45] C.L. Lawson. Software for c1 surface interpolation. In R.R. John, editor, *Mathematical Software*, pages 161–194. Academic Press, 1977.

[46] C.L. Lawson and R.J. Hanson. *Solving Least Squares Problems*. Society for Industrial and Applied Mathematics, 1995.

[47] A. Lazaric, M. Ghavamzadeh, and R. Munos. Analysis of a classification-based policy iteration algorithm. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 607–614, Madison, WI, USA, 2010. Omnipress.

[48] L. Lehnert and M.L. Littman. Successor features support model-based and model-free reinforcement learning. *ArXiv*, abs/1901.11437, 2019.

[49] X Li, G. Calinescu, P. Wan, and Y. Wang. Localized delaunay triangulation with application in ad hoc wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 14(10):1035–1047, 2003.

[50] J. Ma and W.B. Powell. A convergent recursive least squares approximate policy iteration algorithm for multi-dimensional markov decision process with continuous state and action spaces. In *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages 66–73, 2009.

[51] S. Madjiheurem and L. Toni. Representation learning on graphs: A reinforcement learning application. In K. Chaudhuri and M. Sugiyama, editors, *Proceedings of Machine Learning Research*, volume 89 of *Proceedings of Machine Learning Research*, pages 3391–3399. PMLR, 16–18 Apr 2019.

[52] S. Mahadevan. Proto-value functions: Developmental reinforcement learning. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, page 553–560, New York, NY, USA, 2005. Association for Computing Machinery.

[53] S. Mahadevan. Representation discovery in sequential decision making. In *Proceedings of the National Conference on Artificial Intelligence*, volume 3, 01 2010.

[54] S. Mahadevan and M. Maggioni. Value function approximation with diffusion wavelets and laplacian eigenfunctions. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems*, volume 18. MIT Press, 2006.

[55] P.S. Mara. Triangulations for the cube. *Journal of Combinatorial Theory, Series A*, 20(2):170–177, 1976.

[56] S. Mitchell and S. Vavasis. Quality mesh generation in three dimensions. *Proceedings of the ACM Computational Geometry Conference*, 12 1999.

[57] W.B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality (Wiley Series in Probability and Statistics)*. Wiley-Interscience, USA, 2011.

[58] M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., USA, 1st edition, 1994.

[59] L.D. Pyeatt and A.E. Howe. Decision tree function approximation in reinforcement learning. Technical report, In Proceedings of the Third International Symposium on Adaptive Systems: Evolutionary Computation and Probabilistic Graphical Models, 1998.

[60] C.R. Rao. *Linear Statistical Inference and its Applications*. John Wiley & Sons, Inc., 2nd edition, 04 1973.

[61] J. Ruppert. A delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18(3):548–585, 1995.

[62] B. Scherrer, M. Ghavamzadeh, V. Gabillon, B. Lesner, and M. Geist. Approximate modified policy iteration and its application to the game of tetris. *Journal of Machine Learning Research*, 16(49):1629–1676, 2015.

[63] F. Schröder and P. Roßbach. Managing the complexity of digital terrain models. *Computers & Graphics*, 18(6):775–783, 1994.

[64] P.J. Schweitzer and A. Seidmann. Generalized polynomial approximations in markovian decision processes. *Journal of Mathematical Analysis and Applications*, 110(2):568–582, 1985.

[65] H. Seidel. Symmetric recursive algorithms for surfaces: B-patches and the de boor algorithm for polynomials over triangles. *Constructive Approximation*, 7:257–279, 1991.

[66] M. Sharir and E. Welzl. Random triangulations of planar point sets. In *Proceedings of the Twenty-Second Annual Symposium on Computational Geometry*, SCG '06, page 273–281, New York, NY, USA, 2006. Association for Computing Machinery.

[67] J. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry*, 22:21–74, 05 2001.

[68] J. Shewchuk. General-dimensional constrained delaunay and constrained regular triangulations, i: Combinatorial properties. *Discrete & Computational Geometry*, 39:580–637, 03 2008.

[69] J.R. Shewchuk. What is a good linear element? - interpolation, conditioning, and quality measures. In *In 11th International Meshing Roundtable*, pages 115–126, 2002.

[70] H. Si and K. Gärtner. Meshing piecewise linear complexes by constrained delaunay tetrahedralizations. In B.W. Hanks, editor, *Proceedings of the 14th International Meshing Roundtable*, pages 147–163, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[71] R. Sibson. Locally equiangular triangulations. *Computer Journal*, 21:243–245, 1978.

[72] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

[73] Y. Tang. Deep learning using linear support vector machines. *arXiv: Learning*, 2013.

[74] M. Trick and S. Zin. Spline approximations to value functions. *Macroeconomic Dynamics*, 1:255–277, 1997.

[75] M.A. Unser. Splines: a perfect fit for medical imaging. In M. Sonka and J.M. Fitzpatrick, editors, *Medical Imaging 2002: Image Processing*, volume 4684, pages 225 – 236. International Society for Optics and Photonics, SPIE, 2002.

[76] X. Xue and L. Zhou. Compact car-body surface design with t-spline surface. *Transactions of Nanjing University of Aeronautics and Astronautics*, 31:615–621, 12 2014.