ON NUMERICAL ALGORITHM AND INTERACTIVE VISUALIZATION FOR OPTIMAL CONTROL PROBLEMS *

Lars Grüne
Institut für Mathematik
Universität Augsburg
Universitätsstr. 14
86135 Augsburg, Germany
`lars.gruene@math.uni-augsburg.de`

Martin Metscher
Institut für Angewandte Mathematik
Universität Bonn
Wegelerstr. 6
53115 Bonn, Germany
`metscher@iam.uni-bonn.de`

Mario Ohlberger
Institut für Angewandte Mathematik
Universität Freiburg
Hermann-Herder-Str. 10
79104 Freiburg, Germany
`mario@mathematik.uni-freiburg.de`

Abstract: We present methods for the visualization of the numerical solution of optimal control problems. The solution is based on dynamic programming techniques where the corresponding optimal value function is approximated on an adaptively refined grid. This approximation is then used in order to compute approximately optimal solution trajectories. We discuss requirements for the efficient visualization of both the optimal value functions and the optimal trajectories and develop graphic routines that in particular support adaptive, hierarchical grid structures, interactivity and animation. Several implementational aspects using the Graphics Programming Environment 'GRAPE' are discussed.

Keywords: visualization, optimal control problems, dynamic programming, hierarchical grid structure, interactivity

AMS Classification: 68U05, 49L20, 65N50

1 Introduction

The numerical solution of optimal control problems using dynamic programming techniques or Hamilton-Jacobi-Bellman PDEs has been an active field of research for the last few decades, cf. e.g. [3] for an overview. In contrast to trajectorywise approaches like

*This paper has been written while the first author was a member of the Graduiertenkolleg "Nonlinear Problems in Analysis, Geometry und Physics" (GRK 283) financed by the DFG and the State of Bavaria. Research partially supported by the DFG Schwerpunkt "Ergodentheorie, Analysis und effiziente Simulaion dynamischer Systeme"

Pontryagin's maximum principle this method allows a global solution to optimal control problems by calculating their optimal value functions.

Recently, many achievements of modern numerical analysis and scientific computing like adaptive state space discretization [10], high-order schemes [6] and efficient iterative schemes [9], [18] have been adapted to this problem and made it possible to follow this approach also for problems in more than two space dimensions. Furthermore, the concept of discrete or sampled feedback control has turned out to form a suitable class of control functions realizing approximately optimal solution trajectories and being compatible with the numerical discretization, see [8].

Until now, however, not much effort has been put into the development of efficient visualization tools for the numerical solution of these problems in three or even higher dimensional state spaces. For one and two-dimensional problems existing standard visualization programs could be used, although even in these cases features like interactivity and animation are in general missing. The situation gets worse if three-dimensional problems are to be handled for which efficient numerical methods as cited above have been used for the solution.

First, adaptive grids for the discretization of the state space (and corresponding hierarchical data structures) are rarely supported by standard software, hence projections of the solution data onto equidistant grids or conversion of the data structure are necessary, resulting in slow performance or poor output. Second, visualization routines for three-dimensional solution trajectories producing a 3d-output that is clear and easy to interpret are difficult to find. If in addition one demands both interactivity (i.e. a direct coupling with the numerical routine) and animation no standard solutions are available.

It is the aim of the present paper to provide concepts and implementations to close this gap. The implementation is based on the GRAphics Programming Environment 'GRAPE' [25], which already provides basic visualization features in three-dimensional space. The recent development of visualization routines making efficient use of adaptive grids on hierarchical data structures [13], [15] is connected with a procedural interface in order to couple these routines with the numerical data structures. For the visualization of optimally controlled as well as uncontrolled trajectories existing concepts for tracing on time dependent vectorfields [16] have been extended and combined with the numerical optimal control routines. All in all this results in a comprehensive visualization package for this kind of numerical problems.

Furthermore, in Section 3 of this paper we give some new ideas for the generation of anisotropic adaptive grids for the approximation of optimal value functions including the description of a suitable hierarchical data structure, thus extending the results from [10].

This paper is organized as follows: In Section 2 we formulate the problem to be solved, describe the basic numerical schemes we have used and formulate functional and implementational requirements for an efficient visualization. In Section 3 we highlight several implementational details in order to specify the requirements for the visualization in greater detail. Here we focus in particular on the generation of the adaptive grids and the data structure used. Section 4 is concerned with the visualization of optimal value functions on adaptive grids via a procedural interface and hierarchical algorithms. In Section 5 we present the concepts for the generation of trajectories along with their interactive handling and finally, in Section 6, we illustrate our routines by two examples.

2 Problem setup and numerical methods

The problem we consider is an infinite horizon discounted optimal control problem given by a nonlinear control system on an n-dimensional Riemannian manifold M

$$\dot{x}(t) = f(x(t), u(t)), \;\; x(0) = x_0 \in M \tag{2.1}$$

where

$$u(\cdot) \in \mathcal{U} := \{u : \mathbb{R} \to U \mid u(\cdot) \text{ measurable}\}$$

and $U \subset \mathbb{R}^m$ is compact, together with a cost function

$$g : M \times U \to \mathbb{R}$$

Here both f and g are supposed to by Lipschitz and bounded. The trajectories of (2.1) which we assume to exist for all times are denoted by $\varphi(t, x_0, u(\cdot))$.

The problem is now to minimize the functional

$$J_\delta(x_0, u(\cdot)) := \int_0^\infty e^{-\delta t} g(\varphi(t, x_0, u(\cdot)), u(t)) dt$$

for a fixed positive *discount rate* $\delta > 0$, i.e. to determine the optimal value function

$$v_\delta : M \to \mathbb{R}, \;\; v_\delta(x) = \inf_{u(\cdot)\in\mathcal{U}} J_\delta(x, u(\cdot))$$

and the optimal control functions and trajectories.

Optimal control problems of this kind originate in economics (see e.g. [19]) and have recently turned out to be suitable for the approximation of average time optimal control problems measuring asymptotic properties of the given control system (cf. [26], [9] and [12]) and the stabilization of nonlinear control systems (cf. [8] and [11]).

We will now briefly describe the discretization scheme and algorithms used in the numerical approach. For a thorough analysis of the discretization errors we refer to [4], [6], [10] and [8], and also to the survey article [3] as well as to the corresponding chapter in [5]. Note that the minimum time problem [1] and also pursuit-evasion games [2] can be treated in a similar way.

The main key to the numerical approach is Bellman's optimality principle (or — if one prefers this point of view — its infinitesimal version, the Hamilton-Jacobi-Bellman equation, on which we will not focus here). The optimality principle for v_δ is

$$v_\delta(x) = \inf_{u(\cdot)\in\mathcal{U}} \left\{ \int_0^\tau e^{-\delta t} g(\varphi(t, x, u(\cdot)), u(t)) dt + e^{-\delta\tau} v_\delta(\varphi(\tau, x, u(\cdot))) \right\} \tag{2.2}$$

which holds for every $\tau > 0$. Note that v_δ is uniquely characterized by this equation. Basically it states that end pieces of optimal trajectories are optimal trajectories itselves.

In order to discretize (2.2) we apply two discretization steps. For the first one we fix a time step $h > 0$ and choose a numerical scheme $\tilde{\varphi}$ for the solution of (2.1) for constant control values, i.e.

$$\tilde{\varphi}(h, x, u) \approx \varphi(h, x, u) \text{ for all } u \in U$$

(any standard ODE-solver will do, in the numerical examples in this paper we simply used the Euler method). Using this method we replace (2.2) by

$$v_h(x) = \inf_{u \in U} \left\{ hg(x, u) + e^{-\delta h} v_h(\tilde{\varphi}(h, x, u)) \right\} \tag{2.3}$$

Equation (2.3) — which is referred to as the *discrete Hamilton-Jacobi-Bellman equation* — also has a unique solution v_h which gives an approximation for v_δ. This discretization implicitely includes an approximation of $\mathcal{U}$ by piecewise constant control functions, cf. [8].

Note that all quantities in (2.3) are now numerically computable, however we still have a infinite dimensional problem. In order to reduce this to a finite dimensional problem we first assume that the (discrete time) system can be transformed from M to some bounded set $\Omega \subset \mathbb{R}^n$ by a suitable parameterization. On Ω we consider a cuboid grid Ξ with cuboids Q_j and nodes x_i and the space of multilinear functions

$$W_\Xi := \{ w : \Omega \to \mathbb{R} \,|\, w(x + \alpha e_k) \text{ is linear in } \alpha \text{ on each } Q_j \text{ for each } k \}$$

where the e_k, $k = 1, \ldots, n$ denote the standard basis vectors of the $\mathbb{R}^n$.

A unique approximation of v_h in W_Ξ is then characterized by

$$v_h^\Xi(x_i) = \min_{u \in U} \left\{ hg(x_i, u) + e^{-\delta h} v_h^\Xi(\tilde{\varphi}(h, x_i, u)) \right\} \tag{2.4}$$

for all nodes x_i of Ξ. Several iterative schemes have been proposed for the solution of this equation, see the references above and also [7] and [18]. Here usually U is approximated by a finite set in order to simplify the calculation of the minimum.

For the generation of a suitable grid Ξ we use reliable and efficient a-posteriori error estimates as discussed in [10]. These are based on the observation that v_h^Ξ on some grid Ξ satisfies (2.3) in the nodes of Ξ. Using a collection of test points y_l in each Q_j we calculate the residual

$$\eta(y_l) := \left| v_h^\Xi(y_l) - \inf_{u \in U} \left\{ hg(y_l, u) + e^{-\delta h} v_h^\Xi(\tilde{\varphi}(h, y_l, u)) \right\} \right| \tag{2.5}$$

and refine those elements Q_j for which this quantity is large. (For the details of the refining of an element Q_j see Section 3.1.) In addition we use the interpolation error as a criterion for the coarsening of previously refined elements, cf. [10]. Starting with a coarse grid Ξ_0 we can iteratively construct adaptive grids by calculating solutions of (2.4) on Ξ_i and constructing a new grid Ξ_{i+1} using the error estimates (2.5). Thus we end up with a good approximation for v_h and in turn — for $h > 0$ sufficiently small — also for v_δ.

We will now show how optimal control functions and trajectories can be approximated numerically. Again the exploitation of the optimality principle, here in its discrete version (2.3), leads to the solution.

For any point $x \in \Omega$ we choose a value $u \in U$ such that

$$\left\{ hg(x, u) + e^{-\delta h} v_h^\Xi(\tilde{\varphi}(h, x, u)) \right\} \tag{2.6}$$

becomes minimal and define a function $F : \Omega \to U$ by $F(x) := u$. This function F can now be applied to (2.1) as a discrete feedback law via

$$\dot{x}(t) = f(x(t), F(x(ih))), \quad \text{for } t \in [ih, (i+1)h) \tag{2.7}$$

Then the corresponding trajectories $\varphi_F(t, x_0)$ are approximately optimal with respect to v_δ, cf. [8].

In particular this kind of control law does not require a precalculation of an open loop control $u(\cdot)$ for a long time interval but can be evaluated along the controlled trajectory. The approximately optimal open loop control $u(\cdot)$ can easily be constructed via

$$u(t) = F(\varphi_F(ih, x_0)), \quad t \in [ih, (i+1)h)$$

and hence is piecewise constant which corresponds to the discretization in time from (2.2) to (2.3).

What we have obtained by the numerical procedures are

(i) a numerical scheme to calculate v_h^Ξ, the approximation of the optimal value function v_δ, on an adaptively generated grid.

(ii) a numerical way to calculate discrete feedback laws F which generate approximately optimal trajectories and control functions.

Remark 2.1 A class of optimal control problems of particular interest are those formulated for stabilization of bilinear control systems, see e.g. [8] and [9]. Here the optimal control problem (2.1) is given on $M = \mathbb{P}^{d-1}$, the real projective space, whereas the system to be stabilized has $\mathbb{R}^d$ as its state space. Clearly, in this case the visualization routines for the trajectories should be capable of visualizing both the non projected and the projected trajectories.

We will now formulate the properties an efficient interactive visualization should satisfy based on the structure of the optimal control problem and on its numerical approximation.

Remark 2.2 (Requirements for an efficient visualization) The visualization routines should provide the following features:

(i) Visualization of the approximated value function v_δ^Ξ and the corresponding grid Ξ

(ii) Interactive generation and animated visualization of optimal as well as uncontrolled trajectories

(iii) Support of the data structure used in the numerical calculation in order to ensure efficient and fast performance

Note that (i) and (ii) can be regarded as requirements on the functionality of the visualization for which the motivation is immediate: From the numerical point of view the grid generation and the quality of the approximation should become "visible". From the optimal control point of view interactive exploration of optimal trajectories and of the structure of the underlying optimal control problem are worthwhile. Since the trajectories evolve in time the advantages of an animated visualization are obvious. We will address (i) in Section 4 and (ii) in Section 5.

In contrast to that requirement (iii) concerns implementational details of the numerical method which we will specify now.

3 Numerical data structure and implementational details

In this section we will highlight three topics which are essential for the specification of Remark 2.2 (iii) in Section 2. These are

- The construction of the adaptive grids
- The data structure used for the grids
- The calculation of the optimal trajectories

3.1 Construction of the adaptive grid

As already pointed out in Section 2 we have criteria for the refinement and the coarsening of elements Q_j of Ξ, which have been exposed in detail in [10]. However, in [10] the implementation of simplicid grids has been discussed, whereas here we are going to use cuboid grids. This is motivated by two reasons:

First, by looking at the equations (2.4) and (2.6) it can be easily seen that the values of $v_h^{\Xi}(x)$ at points x which are not necessarily nodes of the grid should be easy to obtain. For this purpose in particular the fast determination of the element Q_j containing x is necessary. Using cuboid grids this can be done by a simple coordinatewise check. Second, for small discount rates $\delta > 0$ the adaptivity of the grid mainly refines steep regions of the value function which follow hyperplanes, as the examples in [9] and [10] show. Therefore anisotropic refinement is more suitable for that problem which is of course much easier to implement using cuboids. Since the domain Ω in our problems usually has a very simple shape we do not encounter additional difficulties in the gridding procedure.

We will hence briefly explain the anisotropic refinement and some structural requirements for cuboid grids.

Figure 3.1 shows the possible anisotropic refinements of a cuboid in 3d.

Criteria for anisotropic refinements have been developed for the finite element discretization for different kinds of PDEs, see e.g. [20] and [24]. Here we use a criterion for the direction of refinement of a cuboid by the selection of an appropriate set of test points y_l for the evaluation of (2.5). We choose this set in such a way that it contains the mid points of each edge, i.e. the points depicted in Figure 3.2 for cuboids in 2d.

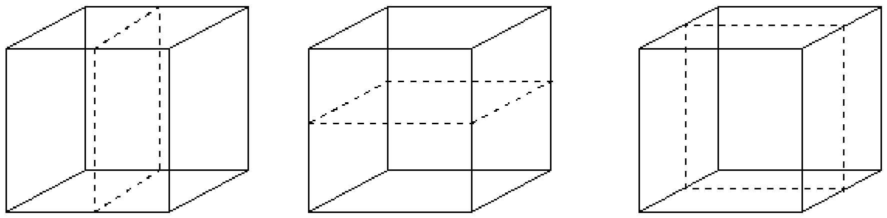
Figure 3.1: Possible refinements in x, y and z-direction in 3d

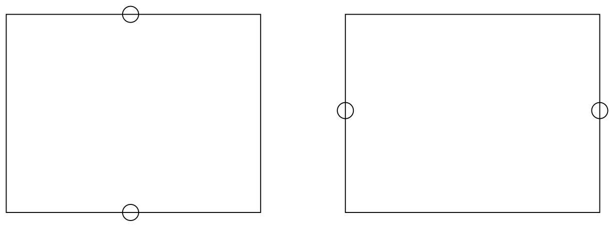
Figure 3.2: Test points for anisotropic refinement in 2d

Depending on the error estimates in these test points we will perform a refinement either in x direction, in y direction, or in both directions.

In a refined grid the "hanging nodes", i.e. the nodes at the interfaces between finer and coarser cubes have to be treated separately in order to ensure continuity of the solution v_h^Ξ on Ξ. Here we interpolate the values in these nodes and thus obtain a continuous solution. In order to obtain a well defined value for this interpolation we require for each two neighbouring cuboids Q_j and Q_k that if Q_j is finer than Q_k in one direction it must not be coarser in any other direction (except the one not affecting the hyperplane on which the cuboids touch).

Furthermore, by requiring that the difference in the refinement between two neighbouring cubes is restricted to at most one level we guarantee that the number of hanging nodes does not become too large.

3.2 Data structure for the grid

One of the main reasons for the usage of adaptive grids is to reduce the amount of memory needed for the storage of the numerical approximation v_h^Ξ. However, the grid handling algorithms should also be fast in order to minimize the time necessary for refining, coarsening etc. These considerations along with the structure of the refinement strategy as described in Section 3.1 almost naturally lead to a hierarchical data structure for the grids as shown in Figure 3.3.

Starting with an equidistant grid Ξ_0 any refinement will be stored hierarchically using this tree structure. In order to illustrate the efficiency of this data structure we briefly indicate how the element Q_j containing a given point $x \in \Omega$ can be found:

Step 1: Find the cuboid in Ξ_0 containing x by modulo operations for each coordinate

Step 2: If this cuboid is a leaf of the refinement tree STOP

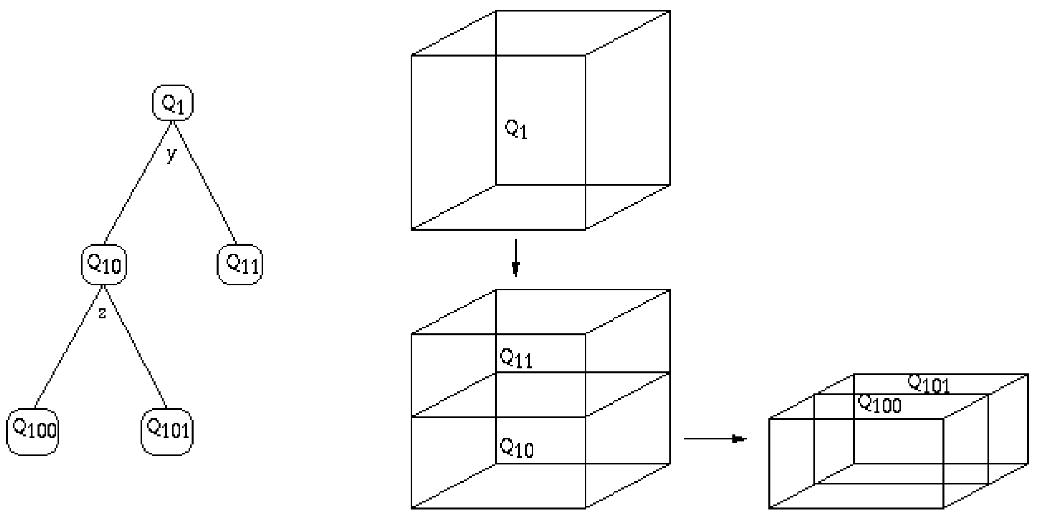

Figure 3.3: Refinement tree

Step 3: Go down the refinement tree one step and determine the cuboid containing x by checking the coordinate refined. CONTINUE WITH STEP 2

In a similar manner neighbouring elements and hanging nodes can be determined by recursive procedures acting on the tree. By a consequent usage of recursive routines the coordinates of an element can be reconstructed while passing through the tree structure and hence no explicit storage of the coordinates is necessary.

Summarizing these arguments we obtain that the main advantage of the hierarchical data structure is to combine memory reduction due to adaptivity with fast performance due to matched recursive algorithms. For the visualization tool to be efficient it is therefore necessary to support this data structure since otherwise this advantage would be immediately lost.

Therefore Remark 2.2 (iii) can now be specified to the fact that hierarchical data structures should be supported in the visualization routines. In addition the ability to visualize the computed function v_h^{Ξ} on a lower level of refinement is useful in order to get a "quick glance" at the data without using the full information (for which the adaptive grid in a 3d problem easily exceeds 200000 cuboids).

Both features are implemented via the procedural interface as described in Section 4.

3.3 Calculation of the trajectories

The calculation of (optimally controlled) trajectories requires the solution of (2.7). Since F can easily be obtained minimizing (2.6) (which again is done using a finite approximation of U) this numerical procedure is rather fast and can therefore be implemented in the following way:

Given an initial value x_0 and an end time T we compute the solutions to (2.7) by evaluating F for each time step h and then solving (2.1) for this constant control value for the next time step. In order to obtain accurate solutions here we use an extrapolation scheme as described in [22, Section 7.2.14] (note that this scheme does not coincide with the scheme $\tilde{\varphi}$, since its purpose is to model the "real" trajectories, cf. [8])

The routine then returns a point list consisting of $[T/h]+1$ entries representing the controlled trajectory, which can then be visualized as described in 5. Here the initial value x_0 and the end time T are to be chosen interactively.

Since we do also want to cover the case mentioned in Remark 2.1 the visualization routine should be able to handle projections of $\mathbb{R}^d$ trajectories to projective space. In this case the numerical routine will return the unprojected trajectory which also provides the full information for the projection.

4 Procedural interface and hierarchical visualization

4.1 The concept of a procedural data access

As we just mentioned in Remark 2.2 and Subsection 3.2 it is very important for the visualization to support the numerical data structure in order to profit from the implemented grid handling. From the visualization point of view there are mainly two possibilities of data access. Most of the frequently used visualization software works on prescribed data formats. A user has to convert his own data structures into such a format, whereas in case of a procedural data access the numerical data structures are addressed by functions. Then visualization tools directly work on the data structures the user is accustomed to from his numerical methods. He only has to provide some access procedures. In what follows we will describe such a procedural data access, which was implemented in the GRAphics Programming Environment GRAPE [25].

In [17] a visualization interface for arbitrary meshes with general data functions on them has been proposed. This interface tries to avoid restrictions on the element types. A mesh is defined as a procedurally linked list of non intersecting elements. The access to data is done by user supplied procedures addressing the user data structures and returning the required data temporarily in a prescribed element structure *ELEMENT*. This *ELEMENT* structure especially contains a polygonal boundary representation, the coordinate vectors for the nodes and function data on them.

Combining this information with local coordinates for the elements, a large class of element types can be handled. Figure 4.1 shows some typical examples of elements that can be addresed in this context. (Here we restrict ourselves to the basic concept. The true data structures is slightly more general, especially concerning the interface for function data (cf. [17]).) As the rendering procedures only take into account these representations of an element, the approach is not restricted to structured meshes and applies to a wide range of commonly used grids. In [15] and [14] various applications on different mesh types in two and three dimensions are shown.

There are mainly six element access procedures in a hierarchical structure (cf. [13] and [15] for a detailed description). A call *first_macro()* returns any first element and *next_macro()* supports a successive traversal of all other macro elements in list order. Thereby only one

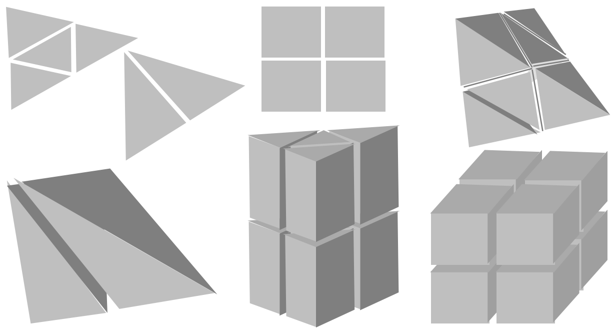

Figure 4.1: Basic element types in two and three dimensions with possible refinements.

ELEMENT structure is to be stored, as each call of *next_macro()* overwrites the preceding element data.

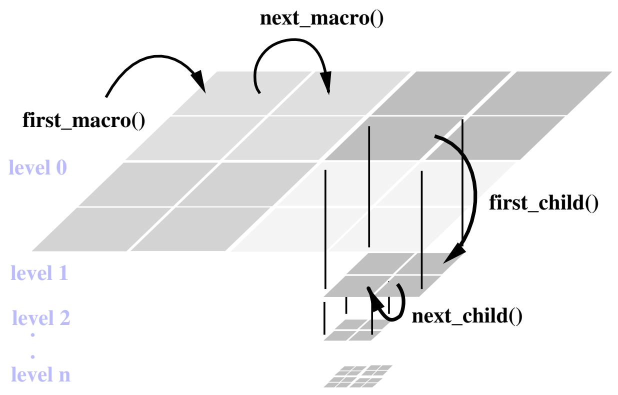

Figure 4.2: A schematic sketch of the procedural access to hierarchical grids by the four routines *first_macro(), next_macro(), first_child(), next_child().*

A call of *first_child()* generates and fills an additional *ELEMENT* structure with some first child data. Finally successive calls of *next_child()* traverse the other child elements of the same parent element and replaces previous child data (cf. Fig. 4.2). Thereby during a recursive traversal of the grid hierarchy a list of at most n temporarily filled ELEMENT structures are present in memory at the same time, where n is the depth of the hierarchy. Finally *copy_element()* provides a copy of element data if information on more than one element is needed simultaneously. In order to address adjacent elements a call of *neighbour()* provides all necessary operations, again overwriting previous element data. This especially implies that also in the user data structures the element information has not to be stored completely on all grid levels but it may be generated when needed based on complete parent information and economically stored offset data. Therefore this procedural interface between visualization and numerics has the necessary provision to support our data structure, as described in Section 3.

The access procedures shown above supply a visualization method with all necessary information to locally evaluate and graphically represent grid geometry and data. This is sufficient to run merely all visualization algorithm, e. g. isosurface rendering or slicing combined with a color shading on the generated slices. Especially the visualization of the approximate value v_δ^Ξ and the corresponding grid Ξ is supported (cf. Remark 2.2).

4.2 The interface to the numerical data structure

According to the general idea of a hierarchical and procedural visualization interface, which we described in the previous subsection we now want to explain some more details of the implementation of this concept.

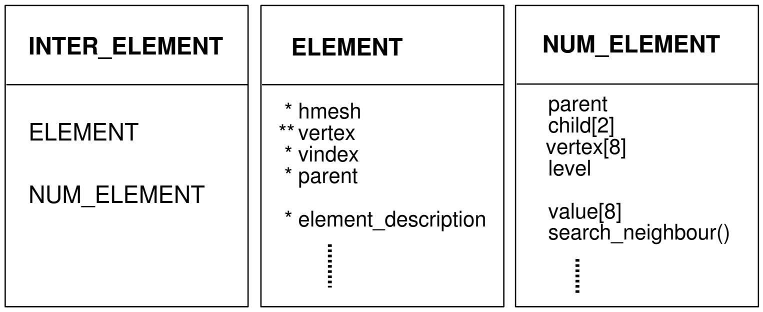

Figure 4.3: Blow up of the *ELEMENT* to an *INTER_ELEMENT* structure

For the implementation of the interface we introduce an *INTER_ELEMENT* structure (cf. Figure 4.3) that consists of an *ELEMENT* on the one hand side and of a *NUM_ELEMENT* structure on the other hand. Here *ELEMENT* represents an element of the mesh which is used and exploited by the visualization routines and *NUM_ELEMENT* consists of pointers to the numerical data structure and some access procedures, provided by the numerical code. Based on this *INTER_ELEMENT* structure a call of an access procedure like *first_macro()*, *next_macro()* or *first_child()*, *next_child()* fills the *ELEMENT* structure using the *NUM_ELEMENT*. Thereby it is mainly necessary to set some pointers. No large data arrays have to be copied. Taking into account that each call of an access procedure *first_macro()*, *next_macro()* or *next_child()* overwrites the previous *INTER_ELEMENT* we have to store only one such element per level of the hierarchical tree at the same time. Additionally to the recursive traversal of the hierarchical mesh by the calls of *first_macro()*,*next_macro()*, *first_child()* and *next_child()* the implemented adjacent search algorithm in the numerical code (cf. Section 3) can be used to fill the procedure *neighbour()* for the visualization. Finally the access to the approximated function v_δ^Ξ is ensured straightforward by the implementation of a data access function f(). This function returns the requested value by exploiting the numerical data structure *NUM_ELEMENT*.

Based on this interface all visualization tools of the existing rendering code GRAPE [25] can be used to visualize the adaptive grid and the approximated function v_δ^Ξ on it.

4.3 Complements to hierarchical visualization

Visualization methods especially on 3D data sets such as isosurface extraction and color shading on slices can benefit from the nested structure of the underlying hierarchical grid. In the following we will focus on the isosurface case. Similar considerations hold for other visualization methods as well.

The cost to extract an isosurface from a given volume data set can be reduced enormously taking into account hierarchical information. Instead of traversing all elements, like a standard marching cube strategy does, we can recursively test for intersections on coarser level elements E to decide whether the children $\mathbb{C}(E)$ have to be visited or not. For example let us consider a uniform mesh with N_{macro} elements on the coarsest grid, where each element is n times refined into N_{child} child elements. Then a standart marching cube algorithm is of complexity $O(N)$, where $N := N_{macro} \cdot {N_{child}}^n$ is the number of elements on the finest grid level. In contrast to this the hierarchical algorithm has a complexity of $O(N_{macro} + N_{child} \cdot n) = O(\log N)$, if only one element has to be extracted. If the considered function on the d-dimensional grid is smooth this leads to a cost reduction of one order of magnitude up to a logarithmic factor, as the isosurface is $d-1$ dimensional.

The intersection test on an element requires the calculation of robust data bounds. Simply taking into account only the function values on the element vertices $x^l \in \mathbb{N}(E)$ will not be sufficient because we might overlook information apparent on finer grid levels only, e. g. strongly curved segments of an isosurface (cf. Fig. 4.4). Let us suppose that we have at

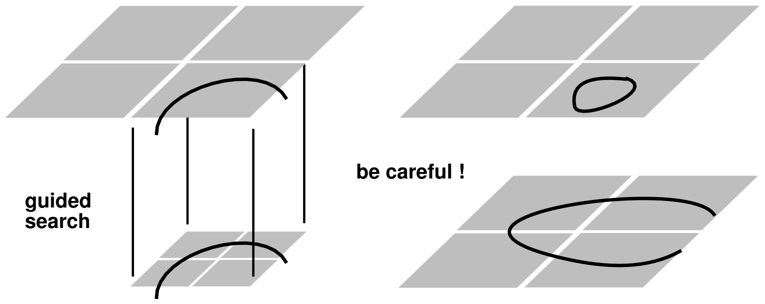

Figure 4.4: Isoline segments in 2D will be missed if just vertex information is taken into account to test for intersections. The same holds in 3D for isosurfaces.

hand an estimate for the function's second derivatives or something comparable on each element of the hierarchy. Then a straightforward calculation of bounds is possible by means of Taylor expansion. For details we refer to [15].

5 Interactive selection and rendering of trajectories

After having solved the optimal control problem numerically it is of natural interest to integrate the approximative optimal trajectories. They should be displayed either in the projected phase space of the control problem or in the unprojected one. In addition it should be useful to compare them with the uncontrolled ones. For that reason some interactive methods have been implemented in GRAPE which allow selection and visualization of these solution curves. (For their computation cf. section 3.3.)

The interactive concept includes control of integration parameters like integartion interval and optimization value. It enables the user to pick on clipping planes in the phase space and thereby to choose the start point of the trajectories. A method to project the calculated solutions into the projected phase space is offered. Once these curves are calculated display routines and animation features (see [16]) for general curves are offered. Examples of visualized trajectories are shown in 6.2.

6 Examples

In this section we will illustrate the visualization routines by two examples. We emphasize that two of the main features of our routines are interactivity and animation which can of course not be illustrated here. This section mainly aims to give an impression how the output of the visualization looks like.

Both examples we present are discounted optimal control problems formulated for the stability analysis and stabilization of semilinear control systems. Here the cost function g is chosen in such a way that the optimal control problem is to minimize the exponential growth rates of the corresponding trajectories for each initial value, as explained in [8], [9] and [11]. A comprehensive description of these kind of problems will appear in [5].

6.1 A chemical reactor

The first problem is the model of a two dimensional nonlinear chemical reactor as introduced in [23]

$$\begin{aligned} \dot{y}_1 &= -y_1 + B\alpha(1-y_2)e^{y_1} - u(y_1 - x_c) \\ \dot{y}_2 &= -y_2 + \alpha(1-y_2)e^{y_1}, \end{aligned}$$

together with its linearization along the trajectories

$$\dot{x} = \begin{pmatrix} -1 + B\alpha(1-y_2)e^{y_1} - u & -B\alpha e^{y_1} \\ \alpha(1-y_2)e^{y_1} & -1-\alpha e^{y_1} \end{pmatrix} x.$$

Projecting the x-component via $s = x/\|x\|$ onto the unit sphere $\mathbb{S}^1$ which can be parameterized by its angular coordinate we obtain a three-dimensional system. The cost function g to be optimized is given by $g(y,s,u) = s^T A(y,u)s$ where A denotes the matrix from the linearization above.

With this example we are going to illustrate the visualization of the value function and the corresponding grid. Recall that the values of the value function in this example represent minimal exponential growth rates for the linearized trajectories depending on the (projected) initial value.

Optimizing on the domain $\Omega = [2.64, 2.97] \times [0.32, 0.58] \times [0.34, 0.41]$ using $\delta = 0.01$ and $h = 0.01$ we calculated the value function $v_h^{\Xi_i}$ on an number of adaptively refined grids Ξ_i. Figure 6.1 shows the value functions on three different grids visualized by three level sets $v_h^{\Xi_i} = 0$, $v_h^{\Xi_i} = 25$ and $v_h^{\Xi_i} = 50$ (from the inside to the outside) and cuts through the corresponding grids. Here the outer level sets ($v_h^{\Xi_i} = 50$ and $v_h^{\Xi_i} = 25$) have been cut off at their upper and lowed edges in order to make the inner ones visible. For the coarsest grid Ξ_1 in the left picture the inequality $v_h^{\Xi_1} > 50$ holds, hence no level sets appear.

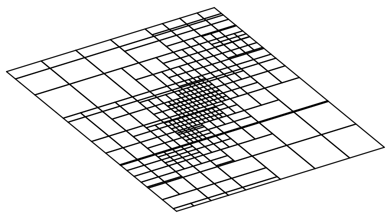

Figure 6.1: Level sets for $v_h^{\Xi_i}$ on different grids Ξ_1, Ξ_2, Ξ_3. For a colored version see Figure 6.4.

6.2 Two coupled oscillators

The second example to be visualized is given by two coupled oscillators as presented in [21]

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -1-u & 0.2 & -u & 0 \\ 0 & 0 & 0 & 1 \\ -u/\sqrt{2} & 0 & -2-u/\sqrt{2} & 0.2\sqrt{2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}.$$

The discounted optimal control problem — as discussed in [8] — is again given by the projection of x to $\mathbb{S}^3$ via $s = x/\|x\|$. Here the parametrization of $\mathbb{S}^3$ with the stereographic projection is used. The cost function reads $g(s, u) = s^t A(u) s$ where $A(u)$ is the system matrix from above.

Here we are going to depict trajectories of the system. Note that this optimal control problem is formulated in such a way that a minimization of the discounted functional for the projected system yields exponential stability of the trajectory of the unprojected system, cf. [8]. Hence (cf. Remark 2.1) it is interesting to see both unprojected and projected trajectories. Using $\Omega = [-1, 1]^3$ (covering the whole projective space in our parametrization) we calculated v_h^{Ξ} with $\delta = 0.01$ and $h = 0.01$.

Figure 6.2: Unprojected trajectories (Controlled and Uncontrolled)

Figure 6.2 shows a controlled (light) as well as an uncontrolled (dark) trajectory of the unprojected system from various angles. As intended in the formulation of the optimal control problem, the controlled trajectory (slowly) converges to the origin whereas the uncontrolled trajectory diverges. In the visualization the x_4-components of the four-dimensional trajectories are not depicted.

The left picture in Figure 6.4 shows the projection of the controlled trajectory to the projective space $\mathbb{P}^3$ which is mapped to $\Omega = [-1, 1]^3$. Due to the fact that the trajectory circles around the corresponding sphere $\mathbb{S}^3$ the trajectory is divided into several pieces in this parameterization.

The right picture in addition shows the level sets of the corresponding value function for $v_h^{\Xi} = -1.5$, $v_h^{\Xi} = -1.75$ and $v_h^{\Xi} = -2$ (from the outside to the inside of the picture). In particular this picture shows that the (approximately) optimal trajectory most of the time stays inside a region where the corresponding (numerically approximated) value function is small, which in fact gives a useful information for the analysis of discounted and average time functionals, see [9, Theorem 2.1].

Figure 6.3: Projected controlled trajectory and level sets

References

[1] M. Bardi and M. Falcone, *An approximation scheme for the minimum time function*, SIAM J. Control Optim., 28 (1990), pp. 950–965.

Figure 6.4: Enlarged colored level sets from Figure 6.1

[2] M. Bardi, M. Falcone, and P. Soravia, *Fully discrete schemes for the value function of pursuit-evasion games*, in Advances in Dynamic Games and Applications, T. Basar and A. Haurie, eds., 1994, pp. 89–105.

[3] I. Capuzzo Dolcetta and M. Falcone, *Discrete dynamic programming and viscosity solutions of the Bellman equation*, Ann. Inst. Henri Poincaré, Anal. Non Linéaire, 6 (supplement) (1989), pp. 161–184.

[4] I. Capuzzo Dolcetta and H. Ishii, *Approximate solutions of the Bellman equation of deterministic control theory*, Appl. Math. Optim., 11 (1984), pp. 161–181.

[5] F. Colonius and W. Kliemann, *The Dynamics of Control*, Birkhäuser, to appear.

[6] M. Falcone and R. Ferretti, *Discrete time high-order schemes for viscosity solutions of Hamilton-Jacobi-Bellman equations*, Numer. Math., 67 (1994), pp. 315–344.

[7] R. L. V. Gonzáles and C. A. Sagastizábal, *Un algorithme pour la résolution rapide d'équations discrètes de Hamilton-Jacobi-Bellman*, C. R. Acad. Sci., Paris, Sér. I, 311 (1990), pp. 45–50.

[8] L. Grüne, *Discrete feedback stabilization of semilinear control systems*, ESAIM: Control, Optimisation and Calculus of Variations, 1 (1996), pp. 207–224.

[9] ——, *Numerical stabilization of bilinear control systems*, SIAM J. Control Optim., 34 (1996), pp. 2024–2050.

[10] ——, *An adaptive grid scheme for the discrete Hamilton-Jacobi-Bellman equation*, Numer. Math., 75 (1997), pp. 319–337.

[11] ——, *Asymptotic controllability and exponential stabilization of nonlinear control systems at singular points*, SIAM J. Control Optim., 36 (1998), pp. 1585–1603.

[12] ——, *On the relation of discounted and average optimal value functions*, J. Differ. Equ., 148 (1998), pp. 65–99.

[13] R. Neubauer, M. Ohlberger, M. Rumpf, and R. Schwörer, *Efficient visualization of large-scale data on hierarchical meshes*, preprint, IAM, Universität Freiburg, 1997.

[14] M. Ohlberger and M. Rumpf, *Adaptive Projektion Methods in Multiresolutional Scientific Visualization*, Report 20, SFB 256, Universität Bonn, 1997.

[15] ——, *Hierarchical and Adaptive Visualization on Nested Grids*, Computing, 59 (1997), pp. 365–385.

[16] M. Rumpf, M. Geiben, and T. Gessner, *Moving and tracing in time-dependent vector fields on adaptive meshes*, preprint, SFB 256, Universität Bonn, 1997.

[17] M. Rumpf, A. Schmidt, and K. Siebert, *Functions defining meshes, a flexible interface between numerical data and visualization routines*, Computer Graphics Forum, 15 (1996), pp. 129–141.

[18] A. Seeck, *Iterative Lösungen der Hamilton-Jacobi-Bellman-Gleichung bei unendlichem Zeithorizont.* Diplomarbeit, Universität Kiel, 1997.

[19] A. Seierstad and K. Sydsæter, *Optimal Control Theory with Economic Applications*, North-Holland, Amsterdam, 1987.

[20] K. Siebert, *An a posteriori error estimator for anisotropic refinement*, Numer. Math., 73 (1996), pp. 373–398.

[21] N. Sri Namachchivaya and H. Van Roessel, *Maximal Lyapunov exponents and rotation numbers for two coupled oscillators driven by real noise*, J. Stat. Phys., 71 (1993), pp. 549–567.

[22] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*, Springer Verlag, New York, 1980.

[23] A. Uppal, W. Ray, and A. Poore, *On the dynamic behaviour of continuous stired tank reactors*, Chem. Eng. Science, 29 (1974), pp. 967–985.

[24] R. Vilsmeier and D. Haenel, *Adaptive methods on unstructured grids for Euler and Navier-Stokes equations*, Comput. Fluids, 4–5 (1993), pp. 485–499.

[25] M. Wierse and M. Rumpf, *GRAPE, Eine interaktive Umgebung für Visualisierung und Numerik*, Informatik, Forschung und Entwicklung, Springer-Verlag, 7 (1992), pp. 145–151.

[26] F. Wirth, *Convergence of the value functions of discounted infinite horizon optimal control problems with low discount rates*, Math. Oper. Res., 18 (1993), pp. 1006–1019.