18th IEEE International Conference on Control Applications
Part of 2009 IEEE Multi-conference on Systems and Control
Saint Petersburg, Russia, July 8-10, 2009

A Generic Dynamic Programming Matlab Function

Olle Sundström and Lino Guzzella

Abstract— **This paper introduces a generic dynamic programming function for Matlab. This function solves discrete-time optimal-control problems using Bellman's dynamic programming algorithm. The function is implemented such that the user only needs to provide the objective function and the model equations. The function includes several options for solving optimal-control problems. The model equations can include several state variables and input variables. Furthermore, the model equations can be time-variant and include time-variant state and input constraints. The syntax of the function is explained using two examples. The first is the well-known Lotka-Volterra fishery problem and the second is a parallel hybrid-electric vehicle optimization problem.**

I. INTRODUCTION

When developing causal suboptimal controllers it is an advantage if the optimal controller is known, even if this controller is not causal. In many cases, such optimal controllers can be found using the deterministic dynamic programming (DP) algorithm introduced in [1]. Of course, this optimal controller can be found only if all future disturbance and reference inputs are known. In this sense, this solution is not causal. Nevertheless, this optimal solution is very useful, because it can be used as a benchmark to which all other causal controllers can be compared to.

Many excellent text books have been published on the subject of DP theory, among them [2] and [3]. An overview of the history and development of dynamic programming is shown in [4]. Interested readers are referred to these references for a detailed discussion of the basic ideas of DP. When implementing the deterministic DP algorithm on a computer there are many numerical issues that arise that have, so far, not yet received sufficient attention. Also, since the computational complexity of every DP algorithm is exponential in the number of states and inputs, special attention must be given to minimizing the overall computational cost. The implementation of suitable numerical algorithms that efficiently solve a given DP problem is, therefore, a nontrivial part of a design process.

This paper presents a Matlab function that efficiently solves deterministic DP problems. The focus lies on the optimal control of non-linear, time-variant, constrained, discrete-time approximations of continuous-time dynamic models. One area where such DP tools have been used successfully is the energy management problem in hybrid-electric vehicles [5], [6].

O. Sundström, Department of Mechanical and Process Engineering ETH Zurich, 8092 Zurich, Switzerland and Empa, Swiss Federal Laboratories for Materials Testing and Research, 8600 Dübendorf, Switzerland, sundstroem@imrt.mavt.ethz.ch

L. Guzzella, Department of Mechanical and Process Engineering ETH Zurich, 8092 Zurich, Switzerland, guzzella@imrt.mavt.ethz.ch

The class of optimal control problems that can be solved using the proposed Matlab function can be written as:

$$\min_{u(t)} J(u(t)) \tag{1}$$

$$\text{s.t.}$$

$$\dot{x}(t) = F(x(t), u(t), t) \tag{2}$$

$$x(0) = x_0 \tag{3}$$

$$x(t_f) \in [x_{f,min},\ x_{f,max}] \tag{4}$$

$$x(t) \in \mathcal{X}(t) \subset \Re^n \tag{5}$$

$$u(t) \in \mathcal{U}(t) \subset \Re^m \tag{6}$$

where

$$J(u(t)) = G(x(t_f)) + \int_0^{t_f} H(x(t), u(t), t)dt \tag{7}$$

is the cost functional. The important characteristics of the considered optimal-control problems are the time-variant constraints on the input and the state, the constrained final state, and the time-variant model equation. These problems are in general difficult to solve. However, these problems can be solved using Bellman's DP [1], provided that the conditions stated in the next section are satisfied.

II. DYNAMIC PROGRAMMING ALGORITHM

This section gives a brief overview of the deterministic DP algorithm as it is implemented in the `dpm` function. Since DP is used here to solve a continuous-time control problem, the continuous-time model (2) must be discretized in time first. Let the discrete-time model be given by

$$x_{k+1} = F_k(x_k, u_k), \qquad k = 0, 1, \dots, N-1 \tag{8}$$

with the state variable $x_k \in \mathcal{X}_k$ and the control signal $u_k \in \mathcal{U}_k$.

A. Basic Algorithm

Let $\pi = \{\mu_0, \mu_1, \dots \mu_{N-1}\}$ be a control policy. Further let the discretized cost of (7) using π with the initial state $x(0) = x_0$ be

$$\begin{aligned} J_\pi(x_0) = & g_N(x_N) + \phi_N(x_N) \dots \\ & + \sum_{k=0}^{N-1} h_k(x_k, \mu_k(x_k)) + \phi_k(x_k), \end{aligned} \tag{9}$$

where $g_N(x_N) + \phi_N(x_N)$ is the final cost. The first term $g_N(x_N)$ represents the final cost in (7). The second term is an additional penalty function $\phi_N(x_N)$ that can be used to enforce a constraint on the final state (4). The function $h_k(x_k, \mu_k(x_k))$ is the cost of applying the control $\mu_k(x_k)$ at x_k, according to $H(x(t), u(t), t)$ in (7). The state constraints (5) are enforced by the penalty function $\phi_k(x_k)$

for $k = 0, 1, \ldots, N-1$. The optimal control policy π^o is the policy that minimizes J_π

$$J^o(x_0) = \min_{\pi \in \Pi} J_\pi(x_0), \tag{10}$$

where Π is the set of all admissible policies.

Based on the principle of optimality [1], the DP algorithm evaluates the optimal cost-to-go[1] function $\mathcal{J}_k(x^i)$ at every node in the discretized state-time space[2] by proceeding *backward* in time:

1) End cost calculation step

$$\mathcal{J}_N(x^i) = g_N(x^i) + \phi_N(x^i) \tag{11}$$

2) Intermediate calculation step for $k = N-1$ to 0

$$\mathcal{J}_k(x^i) = \min_{u_k \in \mathcal{U}_k} \{ h_k(x^i, u_k) + \phi_k(x^i) \ldots \\ + \mathcal{J}_{k+1}(F_k(x^i, u_k)) \} \tag{12}$$

The optimal control is given by the argument that minimizes the right-hand side of equation (12) for each x^i at time index k of the discretized state-time space.

The cost-to-go function $\mathcal{J}_{k+1}(x)$ used in (12) is evaluated only on discretized points in the state space. Furthermore, the output of the model function $F_k(x^i, u_k)$ is a continuous variable in the state space which can be between the nodes of the state grid. Consequently, the last term in (12), namely $\mathcal{J}_{k+1}(F_k(x^i, u_k))$ must be evaluated appropriately. There exist several methods of finding the appropriate cost-to-go $\mathcal{J}_{k+1}(F_k(x^i, u_k))$ such as using a nearest-neighbor approximation or using more advanced interpolation schemes. In the `dpm` function introduced in this paper, linear interpolation of the cost-to-go $\mathcal{J}_{k+1}$ is used. Since the state and the input grids are equally spaced, the computational cost of this interpolation is low compared to the cost induced by the model evaluations.

The output of the algorithm (11)–(12) is an optimal control signal map. This map is used to find the optimal control signal during a forward simulation of the model (8), starting from a given initial state x_0, to generate the optimal state trajectory. In the optimal control signal map the control signal is only given for the discrete points in the state space grid. The control signal, therefore, must be interpolated when the actual state does not coincide with the points in the state grid. In general, the complexity of the DP algorithm is exponential in the number of state and input variables.

III. DPM-FUNCTION

The `dpm` function solves the discretized version of the optimal control problem (1)–(7) using the dynamic programming algorithm introduced in Section II-A. This section shows the syntax and commands for solving such problems. In particular the syntax is shown for a simple optimal control problem. Since the problem is simple the entire code is shown and explained. The `dpm` function can be downloaded at [7].

When solving discrete-time optimal control problems the `dpm` function is normally called using

```
[res dyn] = dpm(fun,par,grd,prb,options);
```

where `fun` is the model function handle, `par` is any user defined parameter structure that is forwarded to the model, `grd` is the grid structure, `prb` is the problem structure, and `options` is the option structure. The output of the `dpm` function are normally two structures representing the DP-output and the signals from forward simulation of the model using the optimal control input map.

Since the DP algorithm is often time consuming, the `dpm` function can also be used only for forward simulation when the DP output structure `dyn` is precalculated. This can be very useful when changing the initial condition or when increasing the starting time `N0` of the problem. To call the `dpm` function when the DP output structure is already calculated use

```
res = dpm(dyn,fun,par,grd,prb,options);
```

All the structures in the code above are further explained in the remainder of this section.

A. Problem

In the problem structure all necessary parameters that define the problem are given. The important parameters are the time step `Ts` of the model description and the problem length `N`. Moreover, in the problem structure an optional cell array can be defined, which contains time-variant information relevant for the problem description. For example, if the model explicitly depends on the time the cell array `W{1}` would contain a time vector with `N` elements. The corresponding elements in these time-variant vectors are forwarded to the model function throughout the problem. The problem structure can also contain a starting time index where the forward simulation starts. This can be helpful when searching for a time optimal solution. An overview of the problem structure is shown in Table I.

TABLE I
PROBLEM-STRUCTURE (PRB)

`Ts`	time step (is passed to the model function)
N	number of time steps in problem (integer that defines the problem length)
`N0`	(optional) start time index (only used in forward simulation)
`W{.}`	(optional) vectors with length `N` containing time-variant data for the model

B. State/Input Grids and Constraints

The grid structure `grd` contains all the information about the state and input grids and constraints. An overview of the

[1]The terms *cost-to-go* and *optimal cost-to-go* are used equivalently throughout this paper referring to *optimal cost-to-go*. It is important to note that the term optimal is used in the sense of optimality achievable under the numeric errors.

[2]The following notation is used: x_k^i denotes the state variable x in the discretized state-time space at the node with time-index k and state-index i. x_k denotes a (state-)continuous state-variable at time k.

grd structure is shown in Table II. The grd structure is composed by cell arrays, where there is a cell for each state variable and each input variable. For example, for a problem with two state variables the grd structure contains grd.X{1}, grd.X{2}, grd.Xn{1}.lo, grd.Xn{2}.lo, and so on. The input grid is used in a similar way depending on the number of input variables of the problem.

TABLE II
GRID-STRUCTURE (GRD)

Nx{.}	number of grid points in state grid
Xn{.}.lo	lower limits for each state (vector for time-variant or scalar for fixed)
Xn{.}.hi	upper limits for each state (vector for time-variant or scalar for fixed)
XN{.}.lo	final state lower constraints
XN{.}.hi	final state upper constraints
X0{.}	inital value (only used in forward sim)
Nu{.}	number of grid points in input grid
Un{.}.lo	(optional) upper limits for each input (vector for time-variant or scalar for fixed)
Un{.}.hi	(optional) upper limits for each input (vector for time-variant or scalar for fixed)

C. Options

The DP approach can be used for many different problem settings and the options structure defines how to use the algorithm. An overview of the options that can be specified in the options structure is shown in Table III. The HideWaitbar options decides if waitbars are shown or not when running the DP algorithm. The SaveMap option determines if the optimal cost-to-go is saved and returned. Note that the memory requirements increase when SaveMap=1.

An important option is the UseLine option, which decides if the boundary line method, introduced in [8], is used or not. The boundary line method is very useful for increasing the accuracy of problems with final state constraints. For more information about the boundary line method readers are referred to [8]. Note in the actual version of the dpm function it can only be used when there is only one state variable. If the boundary line method is used, i.e., if UseLine=1, there are three additional options Iter, Tol, and FixedGrid that must be defined. The options Iter and Tol determines the stopping criteria when numerically inverting the model function. The option FixedGrid decides whether to adjust the grid to the boundary lines or fix the grid to the definition in grd.

Finally, the InfCost is the cost of infeasible states and inputs of the model. When not using the boundary line method InfCost is also used to enforce the final state constraints in (9), with $\phi_N(x_N)$ =InfCost when $x_N \notin [x_{f,min},\ x_{f,max}]$.

D. Output

The outputs of the dpm function are two structures, namely res and dyn. The res structure contains the results from

TABLE III
OPTIONS-STRUCTURE (OPTIONS)

HideWaitbar	hide waitbars (0/1)
Warnings	show warnings (0/1)
SaveMap	save cost-to-go map (0/1)
UseLine	use boundary line method (0/1)
FixedGrid	(used if UseLine=1) using the original grid as specified in grd or adjust the grid to the boundary lines (0/1)
Iter	(used if UseLine=1) maximum number of iterations when inverting model
Tol	(used if UseLine=1) minimum tolerance when inverting model
InfCost	a large cost for infeasible states (I=1)
Minimize	(optional) minimizing (or maximizing) cost function (0/1) default is minimizing
InputType	(optional) string with the same number of characters as number of inputs. Contains the character 'c' if input is continuous or 'd' if discrete (default is all continuous).
gN{1}	(optional) Cost matrix at the final time (must be of size(options.gN{1}) = [grd.Nx{1} grd.Nx{2} ... grd.Nx{.}])

the forward simulation of the model when applying the optimal control input map. The dyn structure is associated with the dynamic programming algorithm, the optimal cost-to-go, and the optimal control input map. When the boundary line method is used the dyn structure also contains the boundary lines (with the states, inputs, and costs). An overview of the two structures are shown in Tables IV and V.

TABLE IV
DP OUTPUT-STRUCTURE (DYN)

B.hi	Xo,Uo{.},Jo contains the cost, input, and state for the upper boundary line
B.lo	Xo,Uo{.},Jo contains the cost, input, and state for the lower boundary line
Jo{.,.}	optimal cost-to-go (indexed by input number and time index)
Uo{.,.}	optimal control input (indexed by input number and time index)

TABLE V
RESULTS-STRUCTURE (RES)

X{.}	state trajectories
C{.}	cost trajectory
I	infeasible vector (problem is not solved if nonzero elements)
signals	structure containing all the signals that were saved in the model function

E. Model

The equations describing the model must be implemented in a correct format in order to be used with the dpm function. To generate a sample function use the command

```
dpm('sample_model',Nx,Nu);
```

This command will save an m-function as sample_model.m with a random model of Nx state variables and Nu input variables, suitable for usage with the dpm function, which can be used as a template when developing a new problem description.

In general the model function should have the format:

```
function [X, C, I, signals] = mymodel(inp,par)
```

where the model input structure inp is generated by the dpm-function and contains the elements in Table VI. The structure par can contain any user defined parameters necessary in the model function. It is important that the model function preserves the size of the inputs to the outputs. Consequently, the elements inp.X{.}, inp.U{.} and the outputs X{.}, C{.}, and I must have the same size. The structure signals can contain any user defined internal signals in model. These signals are stored during the forward simulation and returned in the res structure when calling the dpm-function, Table V.

TABLE VI
INPUT-STRUCTURE (INP)

X{.}	current states (n+m dimensional matrix form depending on the number of inputs and state variables)
U{.}	current inputs (n+m dimensional matrix form depending on the number of inputs and state variables)
W{.}	current time-variant data (scalar)
Ts	time step

TABLE VII
MODEL OUTPUTS

X{.}	resulting states after applying inp.U{.} at inp.X{.} (same size as inp.X{.})
C{.}	resulting cost of applying inp.U{.} at inp.X{.} (same size as inp.X{.})
I	set with infeasible combinations (feasible=0, infeasible=1) (same size as inp.X{.})
signals	structure with user defined signals (same size as inp.X{.})

IV. EXAMPLES

To illustrate the usefulness of the dpm function, two examples are discussed below. First, the well-known Lotka-Volterra fishery problem [9] is explained and solved using the dpm function. Of course, there exist an analytic solution to the continuous-time Lotka-Volterra fishery problem, and it is therefore not necessary to use a DP algorithm to solve it. However, since this problem is simple and is similar to the problems normally solved with DP, it is used as an example to illustrate the syntax of the dpm function. Second, an example of an optimal energy management problem for a parallel hybrid-electric vehicle is solved using the dpm function. This problem is well suited for the DP algorithm. Not surprisingly, DP has been used extensively proposed in the literature to solve such energy management problems, both for comparison to causal controllers and for evaluation of different system configurations. Some examples are [10], [11], [12], and [13].

In the first example of the Lotka-Volterra fishery problem the entire code necessary to use the dpm function is shown. In the hybrid-electric vehicle example, however, the model function contains far too many lines to be included in this paper. Interested readers can download the complete model equations at [7].

A. Lotka-Volterra Fishery

In order to evaluate the optimal solution by means of DP a discrete-time approximation of the continuous-time Lotka-Volterra model is used. Using an Euler forward approximation with a time step $T_s = 0.2$ days, the discrete-time model is

$$x_{k+1} = f(x_k, u_k) + x_k, \qquad k = 0, 1, \ldots, N-1 \tag{13}$$

where

$$f(x_k, u_k) = T_s \cdot \left(\frac{2}{100} \cdot \left(x_k - \frac{x_k^2}{1000} \right) - u_k \right). \tag{14}$$

The state $x_k \in [0,\ 1000]$ is the amount of fishes in a lake, the control signal $u_k \in [0,\ 10]$ is the constant fishing rate during one time step. The discrete-time optimal control problem of maximizing the amount of fishes caught over a fixed period of time can be formulated as follows:

$$\min_{u_k \in [0,\ 10]} \sum_{k=0}^{N-1} -u_k \cdot T_s \tag{15}$$

s.t.

$$x_{k+1} = f(x_k, u_k) + x_k \tag{16}$$

$$x_0 = 250 \tag{17}$$

$$x_N \geq 750 \quad (= x_{f,min}) \tag{18}$$

$$x_k \in [0,\ 1000] \tag{19}$$

$$N = \frac{200}{T_s} + 1. \tag{20}$$

To solve this optimal control problem the model function (13) is implemented in Matlab as:

```
function [X, C, I, signals] = fishery(inp,par)

% state update
func = (0.02.*(inp.X{1}-inp.X{1}.^2/1000)-inp.U{1});
X{1} = inp.Ts.*func + inp.X{1};

% cost
C{1} = -inp.Ts.*inp.U{1};

% infeasibility
I = 0;

signals.U{1} = inp.U{1};
```

Since the state and input spaces have to be discretized, the dpm function includes a simple way to define such grids. Let the state variable be limited between 0 and 1000 and let it be discretized using a step of 5 such that $x_k \in \{0, 5, 10, ..., 995, 1000\}$. Also, let the control input variable be limited between 0 and 10 and let it be discretized with a step of 0.5 such that $u_k \in \{0, 0.5, 1, ..., 9.5, 10\}$. The

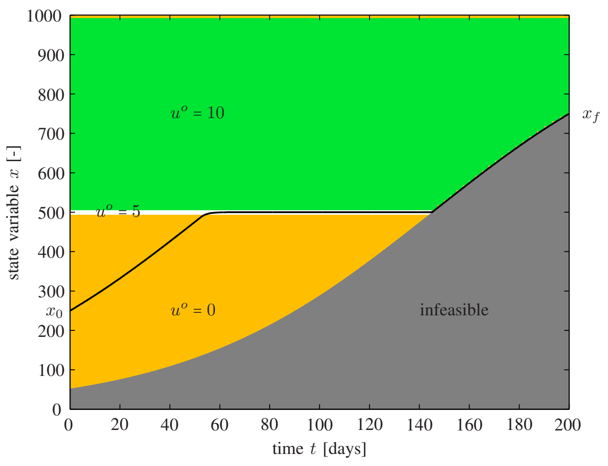

Fig. 1. The optimal control signal map, determined using dynamic programming, for the discrete-time Lotka-Volterra system. The optimal state trajectory for $x_0 = 250$ when using the map is shown as the solid black line.

optimal control problem (15)–(20) is then solved with the dpm function using:

```
% create grid
grd.Nx{1}     = 201;
grd.Xn{1}.lo = 0;
grd.Xn{1}.hi = 1000;
grd.Nu{1}     = 21;
grd.Un{1}.lo = 0;
grd.Un{1}.hi = 10;

% set initial state
grd.X0{1}     = 250;

% set final state constraints
grd.XN{1}.hi = 1000;
grd.XN{1}.lo = 750;

% define problem
prb.Ts = 1/5;
prb.N  = 200*1/prb.Ts + 1;

% set options
options = dpm();
options.UseLine    = 1;
options.SaveMap    = 1;
options.InfCost    = 1200;
options.FixedGrid = 1;

[res dyn] = dpm(@fishery,[],grd,prb,options);
```

The output of the DP algorithm is an optimal control signal map, specifying the optimal control signal at each time step k and at each state $x_k \in \mathcal{X}_k$. The optimal control signal map for the Lotka-Volterra system is shown in Fig. 1. It shows that the optimal control is "not fishing" $u = 0$ if the fish population is small $x < 500$, "moderate fishing" $u = 5$ if the population is $x = 500$ and "full fishing" $u = 10$ if the population is large $x > 500$. Toward the end of the problem, one must stop fishing as late as possible, such that the population reaches the specified minimum final size of $x_{f,min} = 750$. The resulting optimal state trajectory, i.e., the fish population for an initial state of $x_0 = 250$, is shown in Fig. 1 by the black solid line.

B. Hybrid-Electric Vehicle Example

The energy consumption of hybrid-electric vehicles can be described well using a quasi-static discrete-time model. The modeling follows the ideas described in [14], [15]. Essentially, the model contains the battery state-of-charge as the only state variable. In a nutshell, the combustion engine is modeled using an affine Willans approximation, the electric motor is modeled using an electric-power map (derived from detailed simulations), and the battery is modeled as a voltage source together with a resistance in series. The vehicle model includes air drag, rolling friction, and inertial forces. The gearbox is modeled using a constant efficiency of 95%. The hybrid vehicle considered in this study has a 20% hybridization as defined in [16].

The model equations can be summarized and described as

$$x_{k+1} = f(x_k, u_k, v_k, a_k, i_k) + x_k, \tag{21}$$

where x_k is the battery state-of-charge, u_k is the torque split factor, v_k is the vehicle speed, a_k is the vehicle acceleration, and i_k is the gear number. The model assumes isothermal conditions of the components, no extra fuel consumption during the startup of the combustion engine, and no energy losses during gear shifting. A constant auxiliary electric power demand of 350 W is used in the model.

Since the drive cycle is assumed to be known in advance the particular driving speed v_k, acceleration a_k and gear number i_k at instance k can be included in the model function to form the time-variant model:

$$x_{k+1} = f_k(x_k, u_k) + x_k, \qquad k = 0, 1, \ldots, N-1. \tag{22}$$

The optimization problem of minimizing the total fuel mass consumed

$$J = \sum_{k=0}^{N-1} \Delta m_f(u_k, k) \cdot T_s \tag{23}$$

for the hybrid vehicle over a given drive cycle, here the Japanese 10-15 driving cycle (J1015), can be stated as the discrete-time optimal control problem:

$$\min_{u_k \in \mathcal{U}_k} \sum_{k=0}^{N-1} \Delta m_f(u_k, k) \tag{24}$$

s.t.

$$x_{k+1} = f_k(x_k, u_k) + x_k \tag{25}$$

$$x_0 = 0.55 \tag{26}$$

$$x_N = 0.55 \tag{27}$$

$$x_k \in [0.4,\ 0.7] \tag{28}$$

$$N = \frac{660}{T_s} + 1 \tag{29}$$

where $\Delta m_f \cdot T_s$ is the fuel mass consumption at each time step. The time step in this example is $T_s = 1$ s. The optimal control problem (24)–(29) is solved using DP. Figure 2 shows the resulting optimal control map `dyn.Uo{1,:}` and state trajectory `res.X{1}` when using the `dpm`-function as described below.

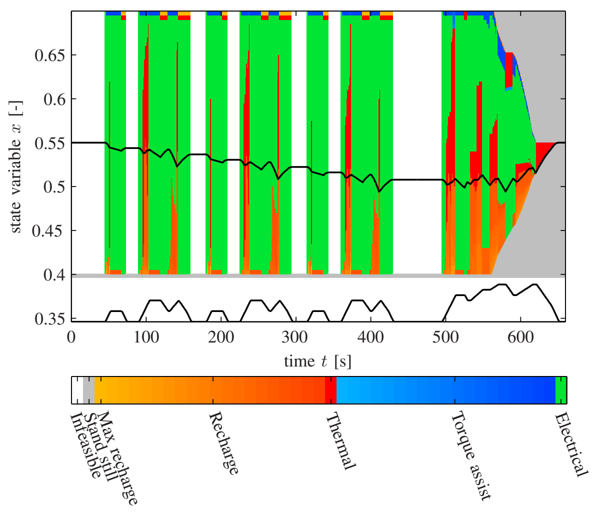

Fig. 2. Optimal input map obtained using the DP algorithm for a full parallel hybrid-electric vehicle driving the Japanese 10-15 drive cycle. The black curve shows the optimal state-of-charge trajectory when the battery is 55% charged at the start.

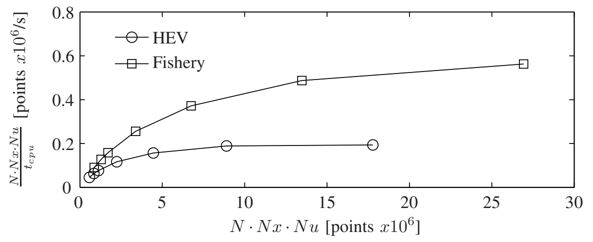

Fig. 3. The computational cost for the two examples. The values are given in calculated grid points per second as a function of the total number of grid points.

```
% create grid
grd.Nx{1}=61; grd.Xn{1}.hi=0.7; grd.Xn{1}.lo=0.4;
grd.Nu{1}=21; grd.Un{1}.hi=1; grd.Un{1}.lo=-1;
% set initial state
grd.X0{1}    = 0.55;
% final state constraints
grd.XN{1}.hi = 0.55;
grd.XN{1}.lo = 0.55;
% define problem
prb.W{1} = speed_vector;        % (661 elements)
prb.W{2} = acceleration_vector; % (661 elements)
prb.W{3} = gearnumber_vector;   % (661 elements)
prb.Ts   = 1;
prb.N    = 660*1/prb.Ts + 1;
% set options
options            = dpm();
options.UseLine    = 1;
options.SaveMap    = 1;
options.InfCost    = 1000;
options.Iter       = 5;
options.InputType = 'c';
options.FixedGrid = 0;

[res dyn] = dpm(@hev,par,grd,prb,options);
```

V. CONCLUSIONS AND FUTURE WORKS

In this paper a Matlab function is introduced that efficiently solves deterministic DP problems. The syntax and the main features of the function are highlighted using two examples. This dpm function together with the model functions introduced in this paper can be downloaded at [7]. The computational time[3] required for backward calculation for the two examples, without using the boundary line, is shown in Fig. 3. It shows that the function evaluates 600000 points/s for the fishery problem and 200000 points/s for the HEV problem. This is due to the more complex model in the HEV problem. Future work includes the attempt of a possible extension of the boundary line method to more general problems, and the support of simple discrete-time Simulink models. The main task for the near future will be to optimize the memory requirements of the function.

VI. ACKNOWLEDGMENTS

Thanks to Daniel Ambühl and Moritz Oetiker for their constructive inputs during the development of the code and for testing and evaluating the dpm function.

REFERENCES

[1] R. E. Bellman, *Dynamic programming*. Princeton - N.J.: Princeton University Press, 1957.

[2] D. Bertsekas, *Dynamic programming and optimal control*, 3rd ed. Belmont, Massachusetts: Athena Scientific, 2005.

[3] R. Luus, *Iterative dynamic programming*, ser. Monographs and surveys in pure and applied mathematics. Boca Raton: Chapman & Hall/CRC, 2000, vol. 110.

[4] R. E. Bellman and E. S. Lee, "History and development of dynamic programming," *IEEE Control Systems Magazine*, vol. 4, no. 4, pp. 24–28, 1984.

[5] M. Back, S. Terwen, and V. Krebs, "Predictive powertrain control for hybrid electrical vehicles," in *IFAC Symposium on Advances in Automotive Control*, Salerno, Italy, April 2004, pp. 451–457.

[6] J. Pu and C. Yin, "Optimal control of fuel economy in parallel hybrid electric vehicles," *Journal of Automobile Engineering*, vol. 221, pp. 1097–1106, 2007.

[7] O. Sundström and L. Guzzella, "DPM-function," *Institute for Dynamic Systems and Control, Department of Mechanical and Process Engineering, ETH Zurich*, 2009, http://www.idsc.ethz.ch/research/downloads.

[8] O. Sundström, D. Ambühl, and L. Guzzella, "On implementation of dynamic programming for optimal control problems with final state constraints," *Oil & Gas Science and Technology - Revue de l'IFP*, 2009, Accepted for publication.

[9] M. Schaefer, "Some aspects of the dynamics of populations important to the management of the commercial marine fisheries," *Bulletin of Mathematical Biology*, vol. 53, pp. 253–279, 1991, Reprinted from the Bulletin of the Inter-American Tropical Tuna Commission, 1(2):27–56, 1954.

[10] H. Mosbech, "Optimal control of hybrid vehicle," in *International Symposium on Automotive Technology & Automation*, vol. 2. Turin, Italy: Automotive Automation Ltd, 1980, pp. 303–310.

[11] C.-C. Lin, H. Peng, J. W. Grizzle, and J.-M. Kang, "Power management strategy for a parallel hybrid electric truck," *IEEE Transactions on Control Systems Technology*, vol. 11, no. 6, pp. 839–849, 2003.

[12] A. Sciarretta, M. Back, and L. Guzzella, "Optimal control of parallel hybrid electric vehicles," *IEEE Transactions on Control Systems Technology*, vol. 12, no. 3, pp. 352–363, 2004.

[13] A. Sciarretta and L. Guzzella, "Control of hybrid electric vehicles," *IEEE Control Systems Magazine*, vol. 27, no. 2, pp. 60–70, 2007.

[14] L. Guzzella and A. Sciarretta, *Vehicle propulsion systems introduction to modeling and optimization*, 2nd ed. Berlin: Springer, 2007.

[15] L. Guzzella and C. H. Onder, *Modelling and control of internal combustion engine systems*. Berlin: Springer, 2004.

[16] O. Sundström, L. Guzzella, and P. Soltic, "Optimal hybridization in two parallel hybrid electric vehicles using dynamic programming," in *17th IFAC World Congress*, ser. Proc. of the 17th IFAC World Congress, Seoul, Korea, 2008.

[3]Calculated on a 32-bit Intel Pentium D 2.8GHz with 2.0 GB RAM.