DPSOLVE - A MATLAB Utility for Solving Discrete-Time Bellman Equations

Mario J. Miranda
The Ohio State University

May 24, 2015

1 Introduction

`dpsolve` is a general discrete-time Bellman Equation solver included in the CompEcon2015 Toolbox. The CompEcon Toolbox is a suite of MATLAB utilities and demonstration programs originally developed to accompany the textbook "Applied Computational Economics and Finance", by Mario J. Miranda and Paul L. Fackler, MIT Press, 2002. The Toolbox has been continuously developed since its initial distribution, and the current version, CompEcon2015, distributed by Mario Miranda, contains many advancements that have been implemented over the past twelve years. The CompEcon2015 Toolbox is designed to run under MATLAB version 2013a. Its backward compatibility with previous versions of MATLAB and previous versions of the CompEcon Toolbox is neither guaranteed nor supported.

`dpsolve` may be used to solve finite- and infinite-horizon discrete-time stochastic dynamic optimization models with discrete, continuous, or mixed states and actions, provided there is at least one continuous state. The continuous states and actions, in principle, may be of arbitrary dimension, subject only to the practical constraints imposed by limitations in computer memory and execution time. A related CompEcon2015 Toolbox utility `ddpsolve` is designed to solve models with purely discrete states and actions.

This note documents the use the `dpsolve` utility distributed with the CompEcon2015 Toolbox released April 5, 2015[1]. The CompEcon2015 Tool-

[1]The CompEcon2015 Toolbox is freely available and may be obtained by emailing Mario

box contains, in addition to dpsolve, an array of supporting numerical utilities, including nonlinear equation, nonlinear complementarity, numerical integration, numerical differentiation, and function approximation tools. The Toolbox also includes higher level utilities designed for solving continuous-time stochastic dynamic optimization and financial equilibrium models and contains an extensive number of documented demonstration programs, most of which are discussed in detail in Miranda and Fackler. The dpsolve utility included in the CompEcon2015 Toolbox has greater functionality than the one documented in the Miranda-Fackler Textbook and employs different input and output protocols that render it backward incompatible with utilities by that name distributed with earlier versions of the CompEcon Toolbox.

2 Infinite Horizon Models

dpsolve is designed to solve discrete-time infinite-horizon stochastic dynamic optimization models with Bellman equations of the form

$$V_i(s) = \max_{j\in J, x\in X_{ij}(s)} \{f_{ij}(s,x) + \delta E_{\epsilon,i'|i,j} V_{i'}(g_{ij}(s,x,\epsilon))\}, \quad s\in S, i\in I. \tag{1}$$

Here,

- $V_i(s)$, the *value function*, is a real-valued function that represents the maximum attainable present value of current and expected future rewards, given the current discrete state i and continuous state s;
- s is a *continuous state* variable of dimension d_s that can assume values in a compact interval $S \subset \Re^{d_s}$ called the *continuous state space*;
- x is a *continuous action* variable of dimension d_x that can assume values in the interval $X_{ij}(s) = [a_{ij}(s), b_{ij}(s)] \subset \Re^{d_x}$, whose lower and upper bounds may depend on the current discrete state i, discrete action j, and continuous state s;
- i is the index of a *discrete state* variable that can assume values in the index set $I = \{1, 2, \ldots, n_i\}$;

- j is the index of a *discrete action* variable that can assume values in the index set $J = \{1, 2, \ldots, n_j\}$;

- $f_{ij}(s, x)$, the *reward function*, is a real-valued function of the current states and actions; f is assumed to be twice differentiable in the continuous action x, if present;

- $g_{ij}(s, x, \epsilon)$, the *state transition function*, is a d_s real-vector-valued function of the current states (i and s) and actions (j and x), and an exogenous continuous state transition shock ϵ that is realized after the current action is taken; g is assumed to be twice differentiable in the continuous action x, if present;

- $E_{\epsilon,i'|i,j}$ is the expectation with respect to the mutually independent continuous state transition shock ϵ and discrete state next period i', conditional on the current discrete state i and discrete action j;

- ϵ is a d_e-dimensional i.i.d. random variable with known distribution and i' evolves as a controlled n_i-state Markov chain with known time-invariant transition probabilities $q_{ii'j} = \Pr(i_{t+1} = i' | i_t = i, j_t = j)$; in the special case that the discrete state transitions are purely deterministic, we more succinctly write $i' = h(i, j)$ to indicate that the discrete state will be i' next period, given the current discrete state i and current discrete action j.

- $\delta \in (0, 1)$ is a discount factor.

The Bellman equation may be written equivalently as

$$V_i(s) = \max_{j \in J} V_{ij}(s) \tag{2}$$

where

$$V_{ij}(s) = \max_{x \in X_{ij}(s)} \{f_{ij}(s, x) + \delta E_{\epsilon,i'|i,j} V_{i'}(g_{ij}(s, x, \epsilon))\}, \tag{3}$$

is the *discrete-action-contingent value function*, which gives the maximum attainable present value of current and expected future rewards, given the current state, *contingent* on taking the discrete action $j \in J$.

3 Solution Method

dpsolve employs the method of collocation to compute an approximation for the value function that solves Bellman equation (1), then generates approximate discrete-action-contingent value functions using (3). Specifically, the value function is approximated using a linear combination of a user-specified n-dimensional basis of real-valued functions defined on the continuous state space S, $\phi : S \mapsto \Re^n$:

$$V_i(s) \approx c_i \phi(s). \tag{4}$$

The $1 \times n$ coefficient vectors c_i, $i \in I$, are then fixed by requiring the value function approximant to satisfy the Bellman equation, not at all possible values of the continuous state s, but rather at n *continuous state collocation nodes* $s_1, s_2, \ldots, s_n \in S$ judiciously selected by the user. The collocation strategy replaces the fundamentally difficult Bellman functional equation (1) with a finite-dimensional nonlinear equation, known as the *collocation equation*, whose unknowns are the basis function coefficient vectors c_i:

$$c_i \phi(s_k) = \max_{j \in J, x \in X_{ij}(s_k)} \{f_{ij}(s_k, x) + \delta E_{\epsilon, i'|i,j} c_{i'} \phi(g_{ij}(s_k, x, \epsilon))\}, \tag{5}$$

$i \in I$ and $k = 1, 2, \ldots, n$.

The accuracy of the approximation afforded by the collocation method is assessed by computing the Bellman equation residual function, the difference between the left- and right-hand sides of the Bellman equation (1) over the domain of continuous states s, when the value function or discrete-action-contingent value functions are replaced by their approximants in the Bellman equation. An exact solution to the Bellman equation will have a residual that is zero everywhere. An approximate solution to the Bellman equation will be zero only at the continuous state collocation nodes, by construction. The accuracy of an approximate solution is thus measured by how much the residual deviates from zero at non-collocation continuous state nodes.

dpsolve allows the user to choose between two classes of basis functions, Chebychev orthogonal polynomials or polynomial splines (typically cubic splines). dpsolve also allows the user to solve the nonlinear collocation equation using either function iteration or Newton's method (the default). For further discussion on the collocation method, Chebychev polynomial and polynomial spline function approximation, and nonlinear equation solution methods, see Miranda and Fackler.

`dpsolve`, by default, will attempt to maximize the optimand embedded in the Bellman equation with respect to the continuous action variable x by solving the associated Karush-Kuhn-Tucker conditions as a nonlinear complementarity problem using a derivative-based adapted Newton method. Alternatively, the user may specify a finite set of possible values of x to be considered, in which case `dpsolve` will search for the best value of x within that set by completely enumerating all possible values of the optimand; this derivative-free method produces less accurate results and will require longer execution time, but generally will be more numerically stable than the derivative-based method, and thus is useful if the derivative-based method fails.

4 Usage

4.1 Calling Protocol

The calling protocol for `dpsolve` is

```
[c,sr,vr,xr,resid] = dpsolve(model,basis,v,x)
```

Input:

`model`	structured array containing model specifications, further discussed below basis n-dimensional
`basis`	basis for real-valued functions defined on the continuous state space S
`v`	$n \times n_i \times n_j$ array of initial guesses for the discrete-action-contingent value function values at the n continuous state collocation nodes, per discrete state and discrete action (optional, default is array of zeros)
`x`	$n \times d_x \times n_i \times n_j$ array of initial guesses for the optimal continuous actions at the n continuous state collocation nodes, per discrete state and discrete action (optional, default is array of zeros)

Output:

`c`	$n \times n_i$ vector of value function approximant basis function coefficients, per discrete state
`sr`	$n_s \times d_s$ refined array of continuous state nodes
`vr`	$n_s \times n_i \times n_j$ array of discrete-action-contingent values on the refined array of continuous state nodes, per discrete state and discrete action
`xr`	$n_s \times d_x \times n_i \times n_j$ array of discrete-action-contingent optimal continuous actions on the refined array of continuous state nodes, per discrete state and discrete action
`resid`	$n_s \times n_i$ array of Bellman equation residuals on the refined array of continuous state nodes, per discrete state

If the residual is requested, `dpsolve` will return the values, optimal continuous actions, and residuals on a refined array of `ns` equally-spaced continuous state nodes. The degree of refinement is governed by n_r, an optional parameter with default value of 10 that may be set by the user using `optset` (see below). If $n_r > 0$, the refined continuous state array is created by forming the Cartesian product of equally-spaced coordinates along each dimension, with n_r times the number of coordinates possessed by the continuous state collocation array along each dimension. If $n_r = 0$, `dpsolve` will return the values, optimal continuous actions, and residuals on the original array of continuous state collocation nodes. On output, `dpsolve` eliminates the singleton dimensions of `c`, `vr`, `xr`, and `resid` to facilitate analysis in the calling program.

4.2 Model Structure

The user specifies the model to be solved via a structured array `model` and a user-coded *function file* further discussed below. The structured array `model` contains different fields that specify essential features of the model to be solved. These fields, with default values for optional fields in parentheses, are

`horizon`	time horizon (infinite)
`func`	name of function file (required)
`params`	model parameters required by function file (empty)
`discount`	discount factor (required)
`ds`	dimension d_s of the continuous state s (1)
`dx`	dimension d_x of the continuous action x (1)
`ni`	number n_i of discrete states i (no discrete states)
`nj`	number n_j of discrete actions j (no discrete actions)
`e`	$n_e \times d_e$ array of discretized continuous state transition shocks (0)
`w`	$n_e \times 1$ vector of discretized continuous state transition shock probabilities (1)
`q`	$n_i \times n_i \times n_j$ array of stochastic discrete state transition probabilities (empty)
`h`	$n_j \times n_i$ array of deterministic discrete state transitions (empty)
`X`	$n_x \times d_x$ array of discretized continuous actions (empty)

Usage notes:

- It is assumed that the continuous state transition shock ϵ, if itself continuous, is replaced prior to creating `model` by a discrete random variable of dimension d_e that assumes n_e distinct values using some appropriate quadrature scheme.
- If `X` is nonempty, then `dpsolve` will maximize the optimand embedded in Bellman's equation by searching exclusively among the values contained in the $n_x \times d_x$ array `X`; otherwise, it will maximize the optimand by solving the Karush-Kuhn-Tucker conditions as a nonlinear complementarity problem using a derivative-based adapted Newton method.
- The user should specify either the stochastic discrete state transition probabilities `q` or the deterministic state transitions `h`, but not both. If both are specified, the latter is ignored. If neither is specified, `dpsolve` uses the default $h(j, i) = j$.

4.3 Function File

The user-coded *function file*, which we shall generically refer to as `func`, evaluates the reward function and the continuous state transition function

at n_s continuous state nodes, and, if needed, the bounds on the continuous action and the first and second derivatives of the reward and continuous state transition functions with respect to the continuous action. The function file is not designed to be called directly by the user. It is intended primarily for use by `dpsolve` and thus must be coded to precise input and output specifications.

The function file takes as input a `flag` that specifies the operations desired by `dpsolve`, an $n_s \times d_s$ array of continuous state nodes `s`, an $n_s \times d_x$ array of continuous actions `x`, an $n_s \times 1$ or 1×1 array of discrete state indices `i`, an $n_s \times 1$ or 1×1 array of discrete actions `j`, an $n_s \times d_e$ array of continuous state transition shocks `e`, and a list of optional function parameters. The output generated depends on the model function being evaluated, as indicated by `flag`.

If `X` is empty (the default) and $d_x > 0$, then `dpsolve` will attempt to solve the continuous action maximization problem embedded in Bellman equation by solving the associated Karush-Kuhn-Tucker conditions as a nonlinear complementarity problem. This will require repeated evaluation of the optimand and its first and second derivatives with respect to continuous action x. In this case, the function file takes the form (user-supplied input in brackets):

```
function [out1,out2,out3] = func(flag,s,x,i,j,e,<parameters>)
switch flag
case 'b'
  out1 = <lower bounds on continuous action x>;
  out2 = <upper bounds on continuous action x>;
case 'f'
  out1 = <reward function f values>;
  out2 = <first derivative of f with respect to x>;
  out3 = <second derivative of f with respect to x>;
case 'g'
  out1 = <continuous state transition function g values>;
  out2 = <first derivative of g with respect to x>;
  out3 = <second derivative of g with respect to x>;
end
```

Here, the lower and upper bounds on the continuous action x are $n_s \times d_x$ arrays; the rewards, their first derivatives, and their second derivatives, are $n_s \times 1$, $n_s \times d_x$, and $n_s \times d_x \times d_x$ arrays, respectively; and the continuous state transitions, their first derivatives, and their second derivatives, are

$n_s \times d_s$, $n_s \times d_s \times d_x$, and $n_s \times d_s \times d_x \times d_x$, respectively. The parameter list in the function definition line must be identical to the list in the field `model.parameters` of the structured model array.

If `X` is a nonempty or $d_x = 0$, then `dpsolve` uses derivative-free methods to approximately maximize the optimand embedded in Bellman's equation. Specifically, it maximizes the optimand searching sequentially among the values found in `X` for the highest value. In this case, the function file takes the simpler form (user-supplied input in brackets):

```
function out = func(flag,s,x,i,j,e,<parameters>)
switch flag
case 'f'
  out = <reward function f values>;
case 'g'
  out = <continuous state transition function g values>;
end
```

Here, the rewards and continuous state transitions are $n_s \times 1$ and $n_s \times d_s$ arrays, respectively. It is important to note that the reward function must be set to negative infinity for infeasible values of the continuous action.

4.4 User Options

The utility `dpsolve` allows the user to set the following options using `optset` (defaults in parentheses):

`algorithm`	collocation equation solution algorithm, either Newton's method ('newton') or function iteration 'funcit'; Newton is default
`ncpmethod`	nonlinear complementarity solution algorithm for continuous action maximization problem embedded in Bellman equation, either semi-smooth formulation ('ssmooth') or min-max formulation 'minmax'
`maxit`	maximum number of iterations, collocation equation solution algorithm (500)
`maxitncp`	maximum number of iterations, nonlinear complementarity solver (50)
`tol`	convergence tolerance (square root of machine epsilon)
`nr`	continuous state array refinement factor (10)

To override an option default, `optset` must be called from the main program prior to execution of `dpsolve` as follows

```
optset('dpsolve','<option name>',<new option value>)
```

For example, to solve the collocation equation using function iteration instead of Newton's method, write

```
optset('dpsolve','algorithm','funcit')
```

and to increase the number of permissible of iterations to 1000, write

```
optset('dpsolve','maxit',1000)
```

5 Demonstration Programs

The CompEcon2015 Toolbox contains a variety of demonstration program that illustrate the use of `dpsolve`. The demonstration programs can be found in the directory CEdemos that accompanies the distribution of this file. The names of the relevant demonstration programs take the form "demdpxx" where "xx" is the number of the program, as indicated in the table below. The table also specifies the dimension of the continuous state variable d_s, the dimension of the continuous action variable d_x, the number of discrete states n_i, and the number of discrete actions n_j.

Table 1: DPSOLVE Demonstration Programs

Number	Title	d_s	d_x	n_i	n_j	Horizon
00	Timber Harvesting Model (Simple)	1	0	0	2	Infinite
01	Timber Harvesting Model	1	0	0	2	Infinite
02	Asset Replacement Model	1	0	6	2	Infinite
03	Industry Entry-Exit Model	1	0	2	2	Infinite
04	Job Search Model	1	0	2	2	Infinite
05	American Option Pricing Model	1	0	2	2	Finite
06	Deterministic Economic Growth Model	1	1	0	0	Infinite
07	Stochastic Economic Growth Model	1	1	0	0	Infinite
08	Public Renewable Resource Model	1	1	0	0	Infinite
09	Private Non-Renewable Resource Model	1	1	0	0	Infinite
10	Water Resource Management Model	1	1	0	0	Infinite
11	Monetary Policy Model	2	1	0	0	Infinite
12	Production Management Model	2	1	0	0	Infinite
13	Inventory Management Model	2	2	0	0	Infinite
14	Livestock Feeding Model	1	1	0	0	Finite
15	Savings Transactions Costs Model	1	1	2	2	Infinite
16	Linear-Quadratic Model	3	2	0	0	Infinite
19	Credit with Strategic Default Model	1	1	4	2	Infinite
20	Lifecycle Consumption-Savings Model	1	1	0	0	Finite
21	Fertility-Savings Model	1	1	> 1	2	Finite