Manual for discrete time continuous state quasi-dynamic programming equation solver

Tomoki FUJII *

January 15, 2008

1 Purpose of this document

This document describes a solver written in Matlab for a dynamic programming equation with non-constant discounting rate used in Fujii and Karp (2006, 2008). We only consider discrete time continuous state setting. Next section sets up the problem and review the formulae that are used in the solver. In Section 3, we provide an overview of the solver. In Section 4, we describe the specifications of the solver and the sub-programs (Matlab functions) called by the solver. While end users can skip the discussion on the sub-programs in this section, those who encounter problems or who need to improve or extend the program may find this section useful. We provide a simple numerical example in Section 5. Additional applications of this program can be found in Fujii and Karp (2006, 2008).

2 Review of Quasi Dynamic Programming Equation

2.1 Derivation of Quasi-DPE

This section is a brief summary of the results given in Fujii and Karp (2006, 2008). Suppose that a decision maker wants to maximize an objective functional by choosing a control variable $x_t \in \Omega \in \mathbf{R}$ for each period t. The payoff for each period is given by the *reward function* $f(x_t, S_t)$, where $S_t \in \mathcal{S}$ is the *state variable* for time t and $S_t \in \mathcal{S} = \left[\underline{S}, \overline{S}\right]$ is the state space. The state variable obeys the equation of motion $S_{t+1} = g(x_t, S_t)$, where g is the *transition function*. Both the reward function and transition function are bounded over $S \times \Omega$.

Now, set the current time to $t = 0$ and suppose the decision maker maximizes the following

*School of Economics, Singapore Management University. Email: tfujii@smu.edu.sg

function by choosing $\{x_t\}_{t=0}^{\infty}$.

$$\sum_{t=0}^{\infty} \theta_t f(x_t, S_t) \qquad \text{s.t.} \quad S_t = g(x_t, S_t), \quad S_0 = S, \tag{1}$$

where θ_t is the *discount factor for utility* in period t. We do not require the discount rate to be constant. Hence, letting the one-period discount factor for time $t > 0$ be σ_t, we can write $\theta_t = \prod_{\tau=1}^{t} \sigma_\tau$. We define $\theta_0 \equiv 1$ and require that the one-period discount factor becomes a positive constant after a finite time T. That is, $\sigma_t = \delta \in (0,1)$, $\forall t \geq T$, so that $\theta_t = \theta_T \delta^{t-T}$, $\forall t \geq T$. By choosing T large, this model approximates a model in which the discount rate approaches a constant only asymptotically. The standard problem corresponds to the case where $T = 0$. With quasi-hyperbolic discounting $T = 1$ and $\sigma_1 = \beta\delta$ where $0 < \beta < 1$; the current generation discounts the next generation's utility by the factor of $\beta\delta$, but each successive generation is discounted at a constant factor δ.

We search for a differentiable control rule $\chi : \mathcal{S} \to \Omega$. The control rule is an *equilibrium control rule* if no decision-maker in the infinite sequence of decision-makers wants to deviate from it.

Definition 1 *An equilibrium control rule χ satisfies the following relationship $\forall S$.*

$$\chi(S) = \arg\max_{x_0} \sum_{t=0}^{\infty} \theta_t f(x_t, S_t) \quad \text{s.t.} \quad S_0 = S, S_{t+1} = g(x_t, S_t), x_t = \chi(S_t) \quad \forall t \geq 1. \tag{2}$$

Equation (2) states that $\chi(S)$ is the current decision-maker's best response, under the belief that future decision-makers will use the rule $\chi(S)$. Letting the value function be $W_\chi(S)$, we have the following result (See Fujii and Karp (2008) for the proof):

Proposition 1 *Assume that there exists a differentiable Markov perfect control rule $\chi(S)$ and a differentiable value function $W_\chi(S)$. An equilibrium control rule $\chi(S)$ satisfies the QDPE*

$$\chi(S) = \arg\max_{x_0} f(x_0, S) + \sum_{t=1}^{T} (\theta_t - \delta\theta_{t-1}) f(\chi(S_t), S_t) + \delta W_\chi(S_1), \tag{3}$$

where the value function $W_\chi(S)$ satisfies

$$\begin{gathered} W_\chi(S) = \max_{x_0} \left[f(x_0, S) + \textstyle\sum_{t=1}^{T} (\theta_t - \delta\theta_{t-1}) f(\chi(S_t), S_t) + \delta W_\chi(S_1) \right] \\ \text{with } S_0 = S, \text{ given, and } S_{t+1} = g(\chi(S_t), S_t) \quad \text{for} \quad t \geq 1. \end{gathered} \tag{4}$$

The value function depends on (possibly non-unique) equilibrium control rule, χ. However,

for the simplicity of notation, we shall drop the subscript χ from W in what follows, unless it is necessary for clarification.

We solve Eq(4) numerically in order to find the equilibrium control rule, χ. This method assumes that χ is smooth. However, there is no guarantee it is the case. One way to check if this is the case is to look at the derivatives of χ at a steady state $(\chi(S^*), S^*)$. Using subscripts for partial derivatives and letting $*$ denote the value evaluated at a steady state[1], we have the Euler Equation (Eq(5) and a condition on χ'' as follows(see Fujii and Karp (2006, 2008) for proofs):

$$f_x^* + g_x^*(f_x^*\chi'^* + f_s^*)\sum_{t=1}^{T}(\theta_t - \delta\theta_{t-1})(g_x^*\chi'^* + g_s^*)^{t-1} + \delta\left(f_s^* g_x^* - f_x^* g_s^*\right) = 0, \tag{5}$$

$$\begin{aligned}
&(f_{xx}^*\chi'^* + f_{xs}^*) + g_x^*(g_x^*\chi'^* + g_s^*)\cdot\Bigg[\sum_{t=1}^{T}(\theta_t - \delta\theta_{t-1}) \\
\cdot\quad &\Big\{(f_x^*\chi'^* + f_s^*)(g_{xx}^*(\chi'^*)^2 + 2g_{xs}^*\chi'^* + g_x^*\chi''^* + g_{ss}^*)\frac{(g_x^*\chi'^* + g_s^*)^{2t-3} - (g_x^*\chi'^* + g_s^*)^{t-2}}{g_x^*\chi'^* + g_s^* - 1} \\
+\quad &(f_{xx}^*(\chi'^*)^2 + 2f_{xs}^*\chi'^* + f_x^*\chi''^* + f_{ss}^*)(g_x^*\chi'^* + g_s^*)^{2t-2}\Big\} \\
+\quad &\delta\Big\{(f_{xs}^*\chi'^* + f_{ss}^*) + \frac{f_x^* g_s^*(g_{xx}^*\chi'^* + g_{xs}^*)}{(g_x^*)^2} - \frac{(f_{xx}^*\chi'^* + f_{xs}^*)g_s^* + f_x^*(g_{xs}^*\chi'^* + g_{ss}^*)}{g_x^*}\Big\}\Bigg] \\
-\quad &\frac{f_x^*}{g_x^*}(g_{xx}^*\chi'^* + g_{xs}^*) = 0.
\end{aligned} \tag{6}$$

3 Overview of the solver

We apply the collocation method to the QDPE. The solver we developed uses a number of Matlab functions provided in the CompEcon Toolbox provided by Miranda and Fackler (2002, hereafter M&F). We assume that the users are reasonably familiar with the CompEcon Toolbox and the materials given in M&F book. In this section, we provide a brief review of the collocation method, and then describe the structure of the solver `dtcssolve`. We also mention important differences of the QDPE from the standard DPE.

Those who need more detailed explanation of the collocation method should refer to M&F book. In particular, Chapters 6, 8 and 9 are useful. Users of this solver should know at least some of the functions provided in the CompEcon Toolbox. In particular, they should at least know how to define a function family structure using `fundefn` and define the collocation nodes using `funnode`. It is also useful to know other related commands such as `fundef`, `funbas`, `funfitxy` and `funeval`. In addition, `chebdop` and `splidop` are used in `funfitxy2` and `dtcsdiff`, which are a part of our solver.

[1] For example, $f_x^* = \left.\frac{\partial f(x,S)}{\partial x}\right|_{(x,s)=(\chi(S^*),S^*)}$.

In the collocation method, an unknown function $F(x)$ of interest is approximated by a linear combination of N known *basis functions* $\phi_n(x)$ so that the approximant has a form of $\hat{F}(x) \equiv \sum_{n=1}^{N} c_n \phi_n(x)$. The N coefficients $c_1, c_2, \ldots, c_N$ are fixed by requiring the approximant to approximately or exactly satisfy the functional equation of interest at N prescribed points $x_1, x_2, \cdots, x_N$ called the *collocation nodes* in a fixed domain $[\underline{x}, \overline{x}]$. The basis functions must be linearly independent at the collocation nodes.

The collocation nodes could be evenly spaced over this domain. It is known, however, that polynomial approximants are known to work better with Chebyshev nodes. Depending on the nature of the function to approximate and the type of approximants used, the user should define the appropriate collocation nodes appropriately.[2] The facilities to generate Chebyshev nodes are provided in the CompEcon Toolbox.

Suppose for now that the function $F(x)$ can be evaluated at the collocation nodes. Then, finding its approximant corresponds to finding the coefficients c_n. To uniquely determine c_n, we need to have at least N evaluation nodes. When the number of collocation nodes K is equal to N, then $\hat{F}(x_n) = F(x_n)$ for all $1 \leq n \leq N$. If $K \leq N$, we can determine c_n by minimizing residuals at the collocation nodes.

In our application for QDPE, we cannot evaluate $\chi(\cdot)$ or $W(\cdot)$ at the collocation nodes simply because we do not know what they are. Hence, we start instead from the guess of the control rule to numerically solve QDPE. Then, we can approximate the value function using the definition of the value function, and solve back for the control rule using the approximated value function. By iterating this procedure until the control rule and value function converge, we can find the approximants of the control rule and value function.

Note that it is difficult to start from the guess of the value function because the second term of the right hand side of the QDPE Eq(4) involves the evaluation of the reward function and transition function for the first T periods, which in turn requires the value of control rules. This problem does not exist in the standard DPE problem.

Let us now explain how the solver `dtcssolve` works. First, the user specifies K collocation nodes, and let s^k be the k-th collocation node with $s^k < s^{k+1}$ for all $k(< K-1)$. Let $\chi^{(r)}$ be the approximant in the r-th iteration, where $\chi^{(0)}$ is the initial guess supplied by the user. Using Eq(4),

[2] Right now, only the Chebyshev polynomial is supported. In the future, we may extend the solver to linear and/or cubic splines.

the sub-program `dtcsevalw.m` calculates the value function at the collocation nodes as follows:

$$\begin{aligned} W^{(r)}(s^k) &\leftarrow \sum_{t=0}^{T_w} \theta_t f(\chi^{(r)}(s_t^k), s_t^k) \quad \text{s.t.} \quad s_0^k = s^k, s_{t+1}^k = g(\chi^{(r)}(s_t^k), s_t^k) \\ &\approx \sum_{t=0}^{\infty} \theta_t f(\chi^{(r)}(s_t^k), s_t^k) \quad \text{s.t.} \quad s_0^k = s^k, s_{t+1}^k = g(\chi^{(r)}(s_t^k), s_t^k) \\ &\equiv W_{\chi^{(r)}}(s^k), \end{aligned} \tag{7}$$

where s_t^k is the state variable at time t when the initial state is s^k. Since we cannot compute Eq(7) over the infinite time horizon in practice, we calculate until a finite time T_w. T_w must be large enough to make the approximation good enough. The user can specify the tolerance condition `evalwtol` to set the acceptable level of approximation. `dtcsevalw.m` takes summation until such t that $|\theta_t f(\chi^{(r)}(s_t^k), s_t^k)|$ <`evalwtol` is satisfied. Alternatively, the user can also specify `evalwsumt`, in which case the program continues to take the sum at least until t =`evalwsumt`. If both `evalwtol` and `evalwsumt` are specified, `dtcsevalw.m` takes the sum until both conditions are satisfied.

$W^{(r)}(s^k)$ provides the values of the value function at the collocation nodes. Hence, by finding the coefficients on the basis functions of the approximant of the value function, we have the approximant $W^{(r)}$ for the value function for the entire state space for the r-th iteration.

Now, we can evaluate the control rule at each of the approximation nodes by maximizing the right hand side of the QDPE Eq(4). The solver carries out this step by calling `dtcsevalfn.m`.

$$\begin{aligned} \chi^{(r+1)}(s^k) &\leftarrow \arg\max_x f(x, s^k) + \sum_{t=1}^{T} (\theta_t - \delta\theta_{t-1}) f(\chi^{(r)}(s_t^k), t) + \delta W^{(r)}(s_1^k) \\ &\quad \text{s.t.} \quad s_1^k = g(x, s^k), s_{t+1}^k = g(\chi(s_t^k), s_t^k) \quad \text{for} \quad t \geq 1 \end{aligned} \tag{8}$$

Thus, by finding the coefficients of the approximant of the control rule, we now of the approximant $\chi^{(r+1)}$, we have the control rule for the entire state space for the $(r+1)$-th iteration. After the initial iteration, the user can choose to evaluate $W^{(r)}$ by the maximizing QDPE in this step by setting `evalwbyqdpe`= 1. In this case, the following equation is used instead of Eq(7).

$$\begin{aligned} W^{(r+1)}(s^k) &\leftarrow \max_x f(x, s^k) + \sum_{t=1}^{T} (\theta_t - \delta\theta_{t-1}) f(\chi^{(r)}(s_t^k), t) + \delta W^{(r)}(s_1^k) \\ &\quad \text{s.t.} \quad s_1^k = g(x, s^k), s_{t+1}^k = g(\chi(s_t^k), s_t^k) \quad \text{for} \quad t \geq 1 \end{aligned} \tag{9}$$

The iteration continues until termination conditions are attained. The user can specify `xtol` to set the acceptable maximum error for the control rule at the collocation nodes. That is, when

$\max_k |\chi^{(r+1)}(s^k) - \chi^{(r)}(s^k)| <$`xtol` holds for all k, the convergence of control rule is attained. The user can also specify `wtol` to set the acceptable maximum error for the value function at the collocation nodes. The convergence is attained when $\max_k |W^{(r+1)}(s^k) - W^{(r)}(s^k)| <$`wtol` holds for all k. The user can also specify `maxit`, the maximum number of iterations. This is helpful for prevent the solver from getting into an endless loop. The solver exits from the iteration when `maxit` is reached, or both `xtol` and `wtol` are satisfied.

When convergence is attained, the solver finds the steady states supported by the equilibrium control rule. The solver first calls a sub-program `dtcsdels.m`. It returns $\Delta S = g(\chi(S), S) - S$, or the change in the state variable between the current and next periods. The solver evaluates ΔS at each collocation node and searches for a segment $[s_k, s_{k+1}]$ such that $\Delta s_k \Delta s_{k+1} < 0$. Notice that, if $\Delta s_k > 0$ and $\Delta s_{k+1} < 0$, it is likely that a stable steady state exists in this segment. When steady states are found, the solver checks if the necessary condition for the asymptotic stability ($|g_x^* \chi'^* + g_s^*| < 1$) and the second order condition Eq(11) given in the appendix are satisfied. It also checks whether Eq(5) and Eq(6) are satisfied.

The user may be interested in finding a control rule that supports a particular state S^* as a steady state. When S^* is specified, the solver first finds χ^*, the value of the control rule at the steady state, which satisfies $g(\chi^*, S^*) = S^*$. Additionally, the user can impose the Euler condition at the steady state. In theory, this is a redundant condition. However, it may not be satisfied well if the derivative of the approximant of the control function does not approximate the derivative of the control function well . Hence, imposing the Euler condition may help approximate better the control rule around the steady state. When the Euler equation (Eq(5) is required, χ'^*, the derivative of the control rule at the steady state, is obtained by solving the Euler equation Eq(5). The solver calls `dtcsss.m` to compute χ^* and χ'^*. Once we find them, we can find the coefficients of the approximant of the control rule by a constrained least squares regression.

4 Specifications of the solver and sub-programs

4.1 `dtcssolve.m`

The main component of the solver is `dtcssolve` as discussed in the previous section. User should call the function in the following format. Users may be interested in looking at an example given in Section 5.

```
[cx, cw, ss, convinfo]=dtcssolve(model,fspace,s,xinit[,dtcsopt])
```

Input The required arguments are `model`, `fspace`, `s` and `xinit`. `fspace` is a struct that contains information on the basis function. It should be defined by `fundefn` in CompeEcon toolbox. `s`

contains a K-vector of collocation nodes and `x` is a K-vector of initial (guess of) control rule at the collocation nodes.

`model` specifies the problem the user wants to solve. It is a struct that is compatible with the one used for `dpsolve` in CompEcon Toolbox. It must contain the following fields:

`model.func` model function.

`model.discount` a $T-$vector of one-period discount factors σ.

`model.params` additional parameters to be passed to `func`.

`func` field gives the name of the Matlab function called by `dtcssolve`. This matlab function file `func.m` has the following calling syntax

```
[out1, out2, out3]=func(flag,s,x,e,params)
```

`s` and `x` are an l-vector containing the state and the control rule, where l is an arbitrary number.[3] The purpose of `e` is to maintain the compatibility with `dpsolve` in CompEcon Toolbox. `e` is not used in `dtcssolve` and is ignored in the solver. `params` is the parameters that is used in this function.

As with `dpsolve`, `flag` can take `'f'`, `'g'` or `'b'`. In addition, it can take `'fs'` and `'gs'`. When `'f'` is specified, `out1` is l-vector of the value of the reward function evaluated at s and x. `out2` and `out3` are the first and second derivatives of the reward function with respect to the control rule. When `flag` is `'g'`, `out1` is l-vector of the value of the transition function evaluated at s and x, and `out2` and `out3` are its first and second order derivatives with respect to the control rule. If `flag` is `'b'`, `out1` and `out2` return the lower and upper bounds on the actions associated with the state `s`. `out3` is not used in this case.

`'fs'` and `'gs'` are the flags used in `dtcssolve` but not in `dpsolve`. If `flag` is `'fs'`, `out1` and `out2` return the first and second derivatives of the reward function with respect to the state. `out3` returns the cross partial derivative of the reward function with respect to the state and control rule. Likewise, `'gs'` is specified for `flag`, `out1` and `out2` are the first and second derivatives of the transition function with respect to the state, and `out3` is the cross partial derivative.

In addition to `model`, `fspace`, `s` and `xinit`, user can specify a number of options by adding an argument `dtcsopt`. It may contain a number of fields listed below. The default value is used when the corresponding field is omitted.

`dtcsopt.uss` User specified steady state. If this field takes a non-missing value, a linear constraint is imposed to find the coefficients for the approximant for the control rule. `uss` must be a scalar. (Default: `[]`).

[3] Unlike `dpsolve`, `dtcssolve` does not support multidimensional states or controls.

`dtcsopt.euler` If this field takes `1` and `uss` takes a non-missing value, a linear constraint corresponding to Eq(5) is imposed in addition to the steady-state condition. The value of χ' is obtained using `dtcseuler`. (Default: `0`)

`dtcsopt.xtol` Tolerance for the control rule. If a negative value is specified, this condition will be ignored. (Default: `1.0e-8`)

`dtcsopt.wtol` Tolerance for the value function. If a negative value is specified, this condition will be ignored. (Default: `1.0e-8`)

`dtcsopt.maxit` Maximum number of iteration. If a negative value is specified, the solver does not stop until both `wtol` and `xtol` are satisfied. (Default: `1.0e+3`)

`dtcsopt.evalwtol` Tolerance for the evaluation of value function used in `evalw.m`. If a negative value is specified, this condition is ignored. (Default: `1.0e-2*wtol`)

`dtcsopt.evalwsumt` Time horizon used in `evalw.m`. When `evalwtol` is also specified, `evalw.m` continues the summation until the conditions specified by `evalwtol` and `evalwsumt` are both satisfied. If a negative value is specified, this condition is ignored. At least one of `evalwtol` and `evalwsumt` must be positive. (Default:`-1`)

`dtcsopt.evalwbyqdpe` If `evalwbyqdpe` takes 1, the value function after the initial iteration will be evaluated by Eq(9). (Default: `0`)

`dtcsopt.fmsevalfn` Options used in `fminsearch` for policy function iteration. (Default: `[]`).

`dtcsopt.fzsss` Options used in `fzero` for finding the linear restriction consistent with user specified steady state. (Default: `[]`)

`dtcsopt.dtcsdels` Options used in `fzero` for finding steady state. (Default: `[]`)

`dtcsopt.fzeuler` Options used in `fzero` for finding χ' in Euler equation. (Default: `[]`)

`dtcsopt.smethod` Search method. 1: `fminsearch`, Otherwise:`fminbnd` (Default: `fminsearch`)

`dtcsopt.itinfo` Display the iteration information. 1: Yes, Otherwise: No (Default: `1`)

`dtcsopt.ssinfo` Display the steady-state information. 1: Yes, Otherwise: No (Default: `1`)

Output `dtcssolve` returns `cx`, `cw`, `ss` and `convinfo`. `cx` and `cw` are the coefficient vectors for the approximant of the control rule. The dimension of these vectors are determined by the input argument `fspace`. `ss` is a struct that constraints information on the steady states and `convinfo` contains information on the convergence of the program.

`ss` and `convinfo` have the following fields. All of the fields of `ss` have a missing value `[]` when no steady state is found.

`ss.nss` Number of distinct steady states found.

`ss.state` `nss`-vector of steady states S^*, where `nss` is the number of steady states `dtcssolve` has found.

`ss.control` `nss`-vector of control rule at steady states $\chi(S^*)$.

`ss.value` `nss`-vector of the value function evaluated at steady states $W(S^*)$.

`ss.reward` `nss`-vector of the value of reward function at steady states $f(S^*)$.

`ss.stable` `nss`-vector of the stability of the steady state based on the necessary condition $|g_x^* \chi'^* + g_s^*| < 1$. `+1` means stable and `-1` means unstable. If `steady` is `zero`, $|g_x \chi' + g_s| = 1$ holds at the steady state. There may be a continuum of steady states in this case.

`ss.dels` `nss`-vector of ΔS^*. Users should check if this is sufficiently close to zero.

`ss.xp` χ' at ΔS^*.

`ss.xpp` χ'' at ΔS^*.

`ss.soc` `nss`-vector of second order condition at S^*. `0`: SOC Satisfied. `1`: SOC not satisfied. Calculated using Eq(11)

`ss.euler` `nss`-vector of the left-hand-side of the Euler equation Eq(5).

`ss.cpp` `nss`-vector of the right-hand-side of Eq(6), which χ'' has to satisfy at S^*.

`convinfo.stat` Convergence status. When the conditions specified by `xtol` and `vtol` are met, it takes `1`. Otherwise it takes `0`.

`convinfo.solvestat` At least one steady state was found if `solvestat` takes `1`.

`convinfo.mu` The Lagrange multiplier μ for the constrained least square regression. This is relevant only when the steady state is specified by the user. It is a scalar when `euler`$= 0$, and a 2-vector when `euler`$= 1$.

`convinfo.it` Number of iterations until convergence is achieved. If convergence is not achieved, it returns `-1`.

`convinfo.mdifw` maximum change in value function at collocation nodes between the last iteration and the one before the last. That is, `mdifw`$= |\max_k W^{(\mathtt{it})}(s^k) - W^{(\mathtt{it}-1)}(s^k)|$

`convinfo.mdifx` maximum change in control rule at collocation nodes between the last iteration and the one before the last. That is, `mdifx`= $|\max_k \chi^{(\mathtt{it})}(s^k) - \chi^{(\mathtt{it}-1)}(s^k)|$

`convinfo.message` Text message about the convergence status.

4.2 `dtcsevalw.m`

`dtcsevalw` evaluates the value function by taking the sum of the present discounted value of reward function using Eq(7). This function calls `dtcstheta`, and has the syntax below. Note that this does not involve the approximant of W *per se*, because it is calculated directly from the definition of the objective function.

```
w=dtcsevalw(fspace,cx,s,model[,dtcsopt])
```

Input

`fspace` Function space.

`cx` (Approximant of) control rule expressed in terms of the coefficients for basis functions. This is used to simulate the dynamic evolution of the state and control variables over time.

`s` A vector of evaluation nodes. This may or may not be the same as the collocation nodes.

`model` Model specification.

`dtcsopt` Options for the computation of W. The relevant options are `evalwtol` and `evalwsumt`.

Output

`w` The value of the value function evaluated at `s`.

4.3 `dtcsevalfn.m`

`dtcsevalfn` evaluates the maximand in the right hand side of Eq(4) given the current control rule is `x` and the current state variable is `s`. `dtcsevalfn` is used with `fminsearch` in order to find a new control rule at each collocation node. `dtcsevalfn` returns the value of the maximand multiplied by `-1` so that the problem can be treated as a minimization problem.

```
w0=dtcsevalfn(x0,s,model,fspace,cx,cw[,dtcsopt])
```

Input

`x0` Control for the current period

`s` State variable (this is a scalar)

`model` Model specification

`fspace` Functional space

`cx` (Approximant of) control rule expressed in terms of the coefficients for basis functions.

`cw` (Approximant of) value function expressed in terms of the coefficients for basis functions.

Output

`w0` The value of the maximand in the right-hand-side of Eq(4) when the current period is `x0`.

4.4 `dtcsdels.m`

`dtcsdels` gives the change in state given the control rule at evaluation nodes. `dtcsdels` has the following syntax:

```
dels=dtcsdels(s,fspace,cx,model)
```

Input

`s` Evaluation nodes

`fspace` Function space

`cx` (Approximant of) control rule expressed in terms of the coefficients for basis functions.

`model` Model specification

Output

`dels` Change in the state variable between this and next periods ($\Delta S = g(\chi(S), S) - S$).

4.5 `dtcstheta.m`

`dtcstheta` returns a vector of discount factors. It is called in the following form:

```
theta=dtcstheta(time,sigma)
```

Input

`time` Time period $t \leq 0$ (scalar).

`sigma` $T-$Vector of one-period discount factors. The last element is the terminal one-period discount factor δ.

Output

`theta` Discount factor θ_t.

4.6 `dtcsss.m`

`dtcsss` returns the change in the state variable at `ss` given the current control `x0`. `dtcsss.m` is similar to `dtcsdels`. However, `dtcsss` has `x0` in the first argument instead of the approximant of the control `cx`. `dtcsss.m` is used to find the value of control rule to support `ss` as a steady state.

```
dels=dtcsss(x0,ss,model)
```

Input

`x0` Control for the current period

`ss` State. Typically, user-specified steady state is used.

`model` Model specification

Output

`dels` Change in state variable between this and next periods

4.7 `funfitxy2.m`

`funfitxy2` is similar to `funfitxy` provided by CompEcon Toolbox. However, `funfitxy2` uses constrained least squares when user specifies the steady state. If Euler condition is also imposed, `funfitxy2.m` also takes it into account. It is called in the following format:

```
[c, B, mu]=funfitxy2(fspace,bx,y[,fixp])
```

Input

`fspace` Function space

`bx` Input values or basis structure

`y` Value of the function to approximant at collocation nodes

`fixp` Information on the constraint, which has the following fields.

`fixp.x` `x` at which function is fixed.

`fixp.y` the value of the function at `fix.x`

`fixp.yprim` the value of the derivative of the function at `fix.x`. This is optional.

Output

`c` Fitted function expressed as the coefficients for basis functions

`B` Basis structure

`mu` Lagrange multiplier for constraints

4.8 `dtcseuler.m`

`dtcseuler` evaluates the Euler equation Eq(5). It is used in conjunction with `fzero` to find χ'^* that satisfies the Euler equation. It has the following calling sequence:

```
euler=dtcseuler(xprimss,uss,xss,model)
```

Input

`xprimss` χ' at `uss`.

`uss` User specified steady state.

`xss` χ at `uss`.

`model` Model specification.

Output

`euler` The value of the left-hand-side of the Euler Equation Eq(5).

4.9 dtcsdiff.m

`dtcsdiff` is used to take the numerical derivative of the control rule. It has the following syntax:

```
[prime prime2]=dtcsdiff(c,fspace,s)
```

Input

`c` Function (approximant) to differentiate expressed as the coefficients of the basis functions.

`fspace` Function space.

`s` Evaluation nodes.

Output

`prime` The derivative of the function evaluated at `s`.

`prime` The second derivative of the function evaluated at `s`.

4.10 dtcschipp.m

`dtcschipp` evaluates the right hand side of Eq(6), which χ'' must satisfy. This may be used to check the condition on χ'' or to find the appropriate value of χ'' at a steady state. It has the following syntax:

```
chipeq=dtcschipdq(xpp,xp,uss,xss,model)
```

Input

`xpp` χ'' at `uss`.

`xp` χ' at `uss`.

`uss` (User-specified) steady state.

`xss` Control at `uss`.

`model` Model specification.

Output

`chippeq` The right-hand-side of Eq(6).

4.11 `dtcsgraph.m`

`dtcsgraph` draws several graphs based on the information on the approximants. This is not used by `dtcssolv`, but users may find this convenient. It has the following syntax:

```
[evals,xx,ww,rr,dd]=dtcsgraph(cx,cw,fspace,model[,plotdense])
```

Input

`cx` (Approximant of) control rule expressed in terms of the coefficients for basis functions.

`cw` (Approximant of) value function expressed in terms of the coefficients for basis functions.

`fspace` Function space.

`model` Model specification.

`plotdense` Density of plot as a factor of model.n (Default: 20).

Output

`evals` Evaluation nodes (for drawing graphs).

`xx` Control rule at evaluation nodes.

`ww` Value function at evaluatio nodes.

`rr` Reward function at evaluation nodes.

`dd` ΔS at evaluation nodes.

5 Karp (2005) Global Warming Example

5.1 Setup

We first consider the linear quadratic model considered by Karp (2005). Additional examples are given in Section B. We specialize in the following function.

$$\begin{aligned} f(x,S) &= -\frac{1}{2}\left(G((S-\bar{S})^2 + B(x-\bar{x})^2\right) \\ g(x,S) &= \bar{S} + \eta(S-\bar{S}) + x \\ T &= 1 \\ \theta_t &= \beta^{\mathrm{Ind}(t\geq 1)}\delta^t \end{aligned}$$

In this case, we have the following first and second derivatives:

$$\begin{array}{lcllcl} f_x(x,S) & = & -B(x-\bar{x}) & g_x(x,S) & = & 1 \\ f_{xx}(x,S) & = & -B & g_{xx}(x,S) & = & 0 \\ f_s(x,S) & = & -G(S-\bar{S}) & g_s(x,S) & = & \eta \\ f_{ss}(x,S) & = & -G & g_{ss}(x,S) & = & 0 \\ f_{xs}(x,S) & = & 0 & g_{xs}(x,S) & = & 0 \end{array}$$

Therefore, the model function file should look like this:

```
%linear_quad_func.m
%Linear quadratic programming problem.
function [out1,out2,out3]=linear_quad_func(flag,s,x,e,sbar,xbar,eta,B,G,smin,smax);
switch flag
case 'f';
out1=-0.5*(G*(sbar-s).^2+B*(x-xbar).^2); %f(x,s)
out2=-B*(x-xbar);                        %f_x(x,s)
out3=-B*ones(size(x));                   %f_xx(x,s)

case 'b';
out1=smin-sbar-eta*(s-sbar);             %minimum x
out2=smax-sbar-eta*(s-sbar);             %maximum x

case 'g';
out1=sbar+eta*(s-sbar)+x;                %g(x,s)
out2=ones(size(x));                      %g_x(x,s)
out3=zeros(size(x));                     %g_xx(x,s)

case 'fs';
out1=-G*(s-sbar);                        %f_s(x,s)
out2=-G*ones(size(x));                   %f_ss(x,s)
out3=zeros(size(x));                     %f_xs(x,s)

case 'gs';
out1=eta*ones(size(x));                  %g_s(x,s)
out2=zeros(size(x));                     %g_ss(x,s)
out3=zeros(size(x));                     %g_xs(x,s)
end;
```

Note that the bounds on the action is chosen so that the state variable never go outside the range specified by `smin` and `smax`. Once the model function file is written, it is easy to run the solver. The user just needs to set up the parameters accordingly.

```
%linear_quad_short_sample.m
%Defining parameters;
sbar=590;
xbar=116.7;
eta=0.9204;
b=1.9212;
g=0.0223;
delta=exp(-0.3);
beta=exp(-0.2);
smin=500;
```

```
smax=2500;
model.params={sbar xbar eta b g smin smax};
model.discount=[beta*delta delta]';
model.func='linear_quad_func';
dtcsopt.evalwbyqdpe=1;
n=20;

%Path should go through dtcs/ directory;
addpath '../dtcs';

%Define function space;
%'cheb' and 'spli' are supported;
fspace=fundefn('cheb', n, smin, smax);

%Define collocation nodes;
s=funnode(fspace);

%Initial guess;
xinit=repmat(xbar,n,1); %initial guess of x;

%Running the solver;
[cx cw ss convinfo]=dtcssolve(model,fspace,s,xinit);

%Drawing the graph;
dtcsgraph(cx,cw,fspace,model);
```

When this program (`linear_quad_short_sample.m`) is run, the user will see the output below. `DelW` is the maximum change in the value function in the final iteration. `DelX` is the maximum change in the control rule in the final iteration. `MaxR` is the maximum residual in the fitting. This is very close to zero when user-specified steady state is not imposed and the number of collocation nodes and the dimension (order) of the basis function match. The numbers in the bracket is the state at which the maximum change occurred.

```
Iter    1 DelW: 5.695e+003 (2.497e+003), DelX: 3.937e+001 (2.497e+003), MaxR:5.684e-014 (2.497e+003)
Iter    2 DelW: 4.336e+001 (2.497e+003), DelX: 1.630e+000 (2.497e+003), MaxR:4.263e-014 (2.472e+003)
Iter    3 DelW: 5.206e-001 (2.497e+003), DelX: 1.645e-002 (2.497e+003), MaxR:5.684e-014 (5.761e+002)
Iter    4 DelW: 5.181e-003 (2.149e+003), DelX: 1.986e-004 (2.022e+003), MaxR:2.842e-014 (5.276e+002)
Iter    5 DelW: 0.000e+000 (5.031e+002), DelX: 0.000e+000 (5.031e+002), MaxR:2.842e-014 (5.276e+002)


  1 steady state(s) were found
******************************
Steady State Stock     : 1.73889976e+003
Steady State Control   : 9.14524207e+001
Steady State Value     : -5.12048721e+004
Steady State Reward    : -1.53299979e+004
Steady State Stability : 1
Steady State Delta S   : 0.00000000e+000
Steady State Chi'      : -1.65002540e-002
Steady State Chi''     : 6.63495684e-010
Steady State SOC       : 1
Steady State Euler     : -2.99720824e-006
Steady State CPP       : -6.56261193e-003
******************************
```

Once convergence is achieved, `dtcssolve` reports several numbers. Stock, control, value, reward

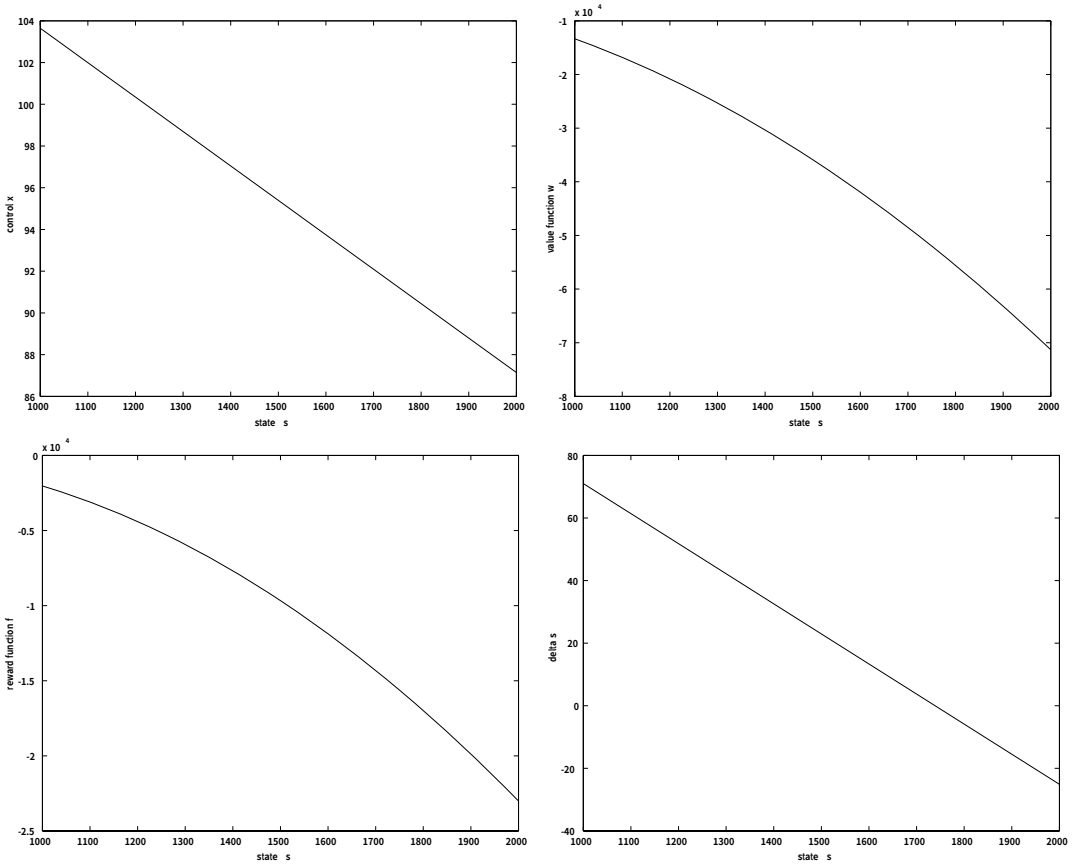

Figure 1: Graph of control $\chi(S)$ (top, left), value function $W(S)$ (top, right), reward function $f(\chi(S), S)$ (bottom, left) and change in state ΔS (bottom, right) against the state variable S.

respectively refar to S^*, χ^*, $W(S^*)$ and $f(\chi^*, S^*)$ respectively. Stability is the local asymptotic stability at the steady state. Delta S is the change in state $\Delta(S^*)$. This should be very close to zero. Chi' and Chi" denote χ'^* and χ''^* respectively. Euler is the right hand side of the Euler equation Eq(5), and CPP is the right hand side of Eq(6), which χ'' is supposed to satisfy. Once the solution is obtained, users can use `dtcsgraph` to draw the graphs. Figure 1 shows the graphs for this example.

5.2 The Uniqueness of the Equilibrium

This section proves the claim made in footnote 4 of Fujii and Karp (2008). From Eq(12) in Fujii and Karp (2008), we have:

$$f_x(0) + (\beta - 1)\delta f_x(1)\chi'(1) + \beta\delta f_s(1) - \delta\eta f_x(1) = 0$$

Now, let $S_0 = S$. Then $S_1 = g(\chi(S), S) = \bar{S} + \eta(S - \bar{S}) + \chi(S)$. Hence, the Euler condition is:

$$
\begin{aligned}
0 \quad = \quad & -B(\chi(S)-\bar{x}) - \\
& \delta B(\chi(\bar{S}+\eta(S-\bar{S})+\chi(S))-\bar{x})\left[(\beta-1)\chi'(\bar{S}+\eta(S-\bar{S})+\chi(S))-\eta\right] - \\
& \beta\delta G(\bar{S}+\eta(S-\bar{S})+\chi(S)-\bar{S})
\end{aligned}
$$

Note that this equation must be satisfied for $\forall S$. Now, suppose that $\chi(S)$ is a polynomial of order n with respect to S. Then, $\chi(\bar{S}+\eta(S-\bar{S})+\chi(S))$ is a polynomial of order n^2 and $\chi'(\bar{S}+\eta(S-\bar{S})+\chi(S))$ is a polynomial of order $n(n-1)$. Therefore, the right hand side of the equation is a polynomial of S of order $n^3(n-1)$. Therefore, n cannot exceed one, because we will be left with a $n^3(n-1)$ degree term otherwise. Therefore, we can write in the form of $\chi(S)=AS+B$, where A and B are a constant. Plugging these values to the Euler equation, we can establish the uniqueness of the solution within the class of polynomials.

Now, suppose that $\chi(S)$ is defined only over (S_l, S_u). Then, we can write $S \equiv S_l + (\frac{1}{\pi}\arctan T + \frac{1}{2})(S_u - S_l)$ for $T \in (-\infty, \infty)$.

5.3 Quadratic approximation of χ around steady state

In a steady state, we have $g(\chi^*, S^*)$. Thus, $\chi^* = (1-\eta)(S-\bar{S})$. To obtain χ', we can plug in the Euler equation to arrive at:

$$
\chi'^* = \frac{\beta\delta G(S^*-\bar{S}) + (1-\eta\delta)B(\chi^*-\bar{x})}{\delta B(\chi^*-\bar{x})(1-\beta)}
$$

The second order condition is

$$
\chi''^* = \frac{B\chi'^* + [\delta B((\beta-1)\chi'^*-\eta)\chi'^* + \beta\delta G](\eta+\chi'^*)}{\delta B(\eta+\chi'^*)(\chi^*-\bar{x})(1-\beta)}
$$

From these, we can construct a quadratic approximation of χ around the user-specified steady state. This would serve as the initial guess.

References

Fujii, T., and L. Karp (2006) 'Numerical analysis of non-constant discounting with an application to renewable resource management.' CUDARE Working Paper 1019, Department of Agricultural and Resource Economics, University of California, Berkeley

_ (2008) 'Numerical analysis of non-constant pure rate of time preference: a model of climate policy.' *Journal of Environmental Economics and Management*

Karp, L. (2005) 'Global warming and hyperbolic discounting.' *Journal of Public Economics* 89(2–3), 261–282

Miranda, M. J., and P. L. Fackler (2002) *Applied Computational Economics and Finance* (MIT Press)

A Second order condition

With a little abuse of notation, we write $f(t) = f(\chi(S_t), S_t)$. We use similar shorthand notation for g and the derivatives. Taking the derivative of the maximand of Eq(4), we have the following first order condition:

$$f_x(0) + \left[\sum_{t=1}^{T}(\theta_t - \delta\theta_{t-1})(f_x(t)\chi'(t)+f_s(t))\frac{dS_t}{dS_1} + \delta W'(S_1)\right] g_x(0) = 0. \tag{10}$$

Now, let us define

$$Z \equiv \sum_{t=1}^{T}(\theta_t - \delta\theta_{t-1})(f_x(t)\chi'(t) + f_s(t))\frac{dS_t}{dS_1} + \delta W'(S_1)$$

Then, dropping the argument (0) for notational simplicity, Eq(10) can be written as:

$$f_x + Zg_x = 0 \Rightarrow Z = -\frac{f_x}{g_x}$$

Then, the second order condition is:

$$\begin{aligned}
SOC &= \frac{\partial(f_x + Zg_x)}{\partial x} \\
&= f_{xx} + \frac{dZ}{dS_1}\cdot\frac{\partial S_1}{\partial x}\cdot g_x + Zg_{xx} \\
&= f_{xx} + \left(\frac{dZ}{dS}\cdot\frac{dS}{dS_1}\right)\cdot g_x \cdot g_x - \frac{f_x g_{xx}}{g_x} \\
&= f_{xx} + \frac{f_x(g_{xx}\chi' + g_{xs}) - g_x(f_{xx}\chi' + f_{xs})}{g_x^2}\cdot\frac{1}{g_x\chi' + g_s}\cdot g_x^2 - \frac{f_x g_{xx}}{g_x} \\
&= \frac{f_{xx}g_x(g_x\chi' + g_s) + f_x g_x(g_{xx}\chi' + g_{xs}) - g_x^2(f_{xx}\chi' + f_{xs}) - f_x g_{xx}(g_x\chi' + g_s)}{g_x(g_x\chi' + g_s)} \\
&= \frac{f_x(g_x g_{xs} - g_s g_{xx}) + g_x(f_{xx}g_s - g_x f_{xs})}{g_x(g_x\chi' + g_s)} (< 0)
\end{aligned} \tag{11}$$

B Additional examples

B.1 Application in Fujii and Karp (2006)

Users can run `renewable_resource.m` to get the results reported in Fujii and Karp (2006). Further details of this application are given in Fujii and Karp (2006).

B.2 Application in Fujii and Karp (2008)

Users can run `stern_model_all.m` and `stern_model_all_new.m` in this order get the results reported in Fujii and Karp (2008). The latter program is used to refine the output given in the former. Further details of this application are given in Fujii and Karp (2008).