# A MW Framework for Solving Recursive Economic Models


Kenneth L. Judd
Hoover Institution

Che-Lin Su
Kellogg, Northwestern Univ.

Greg Thain
UW-Madison

Stephen J. Wright
UW-Madison


Condor Week 2006

April 25, 2006

# Wealth Accumulation Example

Given initial capital stock $x_0$, find $V(x_0)$

$$\boxed{V(x_0) = \begin{cases} \max\limits_{(c_t,l_t)} & \sum_{t=0}^{\infty} \beta^t u(c_t, l_t) \\ s.t. & x_{t+1} = x_t + f(x_t, l_t) - c_t \end{cases}}$$

- $c_t$ and $l_t$ are consumption and labor supply at time $t$
- capital evolves according to $x_{t+1} = x_t + f(x_t, l_t) - c_t$
- $\beta$ is the discount factor and $u(c_t, l_t)$ is the utility given consumption $c_t$ and labor supply $l_t$
- $V(x)$ is the *value function* for $x_0 = x$

# Dynamic Programming

As written, this is an optimization problem with infinitely many variables: $c_t$, $l_t$, $x_t$, $t = 0, 1, 2, \ldots$, so as written it is impossible to solve.

But we can make use of the *dynamic programming principle*, based on the observation that the optimal objective $V(x_0)$ depends only on $x_0$. Also note by the form of the objective and constraints, we have at the optimal values of $x_t$, $c_t$, $l_t$ that

$$\begin{aligned} V(x_0) &= u(c_0, l_0) + \beta \sum_{t=0}^{\infty} \beta^t u(c_{t+1}, l_{t+1}) \\ &= u(c_0, l_0) + \beta V(x_1) \\ &= u(c_0, l_0) + \beta V(x_0 + f(x_0, l_0) - c_0). \end{aligned}$$

We can use this formula involving $V$ to solve for the entire function $V$, not just find its value at a given $x_0$.

# Bellman Equation for $V(x)$

$$\boxed{V(x) = \max_{(c,l)} u(c,l) + \beta V(x + f(x,l) - c)} \qquad (1)$$

- The function $V$ is unknown
- Parametric dynamic programming: Approximate $V(x)$ by $\hat{V}(x; a)$, and solve for the parameters $a$ using the information in the Bellman equation.
  - e.g., $\hat{V}(x; a) = \sum_{j=0}^{p} a_j x^j$
  - find $a \in R^p$ such that $\hat{V}(x; a)$ "approximately" satisfies the Bellman equation (1), on a finite grid of $x$ values: $x_1, x_2, \ldots, x_n$.

# Value Function Iteration

Step 0. *Initialization.* Choose functional form for $\hat{V}(x; a)$ and approximation grid $X = \{x_1, \ldots, x_n\}$.
Make initial guess $\hat{V}(x; a^0)$ and choose $\epsilon > 0$.

Step 1. *Maximization step.* Fix $a^k = (a_j^k)_{j=1}^p$.
For $i = 1, \ldots, n$, compute
$$v_i = T\hat{V}^k(x_i, a^k) = \max_{(c_i, l_i)} u(c_i, l_i) + \beta \hat{V}(x_i^+, a^k)$$
where $x_i^+ = x_i + f(x_i, l_i) - c_i$

Step 2. *Fitting step.* Fix $c, l$. Find $a^{k+1}$ s.t.
$$a^{k+1} = \mathbf{argmin}_a \|\hat{V}(x, a) - v\|^2$$

Step 3. *Convergence.* If $\|\hat{V}(x, a^{k+1}) - \hat{V}(x, a^k)\|_\infty > \epsilon$, go to Step 1; otherwise stop and report solution.

# Value Function Iteration in MW

Objective: Solve the Bellman equation (1)

MASTER: *Initialization.* Choose functional form for $\hat{V}(x; a)$ and approximation grid $X = \{x_1, \ldots, x_n\}$.
Make initial guess $\hat{V}(x; a^0)$ and choose $\epsilon > 0$.

WORKER: *Maximization step.* Fix $a^k = (a_j^k)_{j=1}^p$.
For $i = 1, \ldots, n$, compute (in parallel)
$$v_i = T\hat{V}^k(x_i, a^k) = \max_{(c_i, l_i)} u(c_i, l_i) + \beta \hat{V}(x_i^+, a^k)$$

MASTER: *Fitting step.* Fix $c, l$. Find $a^{k+1}$ s.t.
$$a^{k+1} = \mathbf{argmin}_a \|\hat{V}(x, a) - v\|^2$$

MASTER: *Convergence.* If $\|\hat{V}(x, a^{k+1}) - \hat{V}(x, a^k)\|_\infty > \epsilon$, go to Step 1; otherwise stop and report solution.

# MW Implementation Notes

- Each task finds the optimal $(c_i, l_i)$ for a batch of $x_i$'s.
  - Calls a FORTRAN code (to demonstrate that we can!) to do minimizations.
  - Hot starting - the optimal $(c_i, l_i)$ is usually a great starting point for $(c_{i+1}, l_{i+1})$.
  - The task "wrapper" and the FORTRAN code communicate via files.

# MW Implementation Notes

- `act_on_complete_task()` on the Master stores the $v_i$'s as they arrive from the workers. When all workers have reported, it solves the least-squares problem (fitting step) to find $a^{k+1}$.
  - Could still take a fitting step without waiting for all tasks to report, to avoid hangups if some workers go down.
  - Could adapt size of task (number of $x_i$'s in the batch) to accommodate workers of different speeds.

# How Big Can These Get?

Judd: These models can get very Big!!!

- Investment Portfolio
  - $d$ assets in the portfolio
  - $X_j = \{x_{j1}, \ldots, x_{jn}\}$ represents $j$-th asset's position
  - state space: $X = X_1 \times X_2 \times \cdots \times X_d$
  - transaction cost occurs when adjusting asset positions
- Dynamic Principal-Agent Problem
  - the CEO's performance is evaluated by multiple measures, e.g. stock price, annual profits, etc.
  - the company decides the CEO's compensation package
- Many other economic applications

# Current and Future Work

- Adapting a Fortran90 code with a more complex representation for $\hat{V}(x, a)$, multidimensional $x$, "legacy" minimization routine.
- Dynamic Programming is
  - ubiquitous
  - compute-intensive,
  - algorithmically well suited to the master-worker paradigm supported by MW.
- We are investigating a suitable abstracted implementation of dynamic programming in MW: MW-DP.