Single-agent dynamic optimization models

C.Conlon - help from M. Shum and Paul Scott

Grad IO

Rust Implementation

Rust (1987)

Likelihood function for a single bus:

$$
\begin{aligned}
& l(x_1, \cdots, x_T, i_t, \cdots, i_T | x_0, i_0; \theta) \\
= & \prod_{t=1}^{T} Prob(i_t, x_t | x_0, i_0, \cdots, x_{t-1}, i_{t-1}; \theta) \\
= & \prod_{t=1}^{T} Prob(i_t, x_t | x_{t-1}, i_{t-1}; \theta) \\
= & \prod_{t=1}^{T} Prob(i_t | x_t; \theta) \cdot \prod_{t=1}^{T} Prob(x_t | x_{t-1}, i_{t-1}; \theta_3).
\end{aligned}
$$

The third line arises from the Markovian feature of the problem, and the last equality arises due to the conditional independence assumption.

Rust (1987)

Log likelihood is additively separable in the two components:

$$\log l(\theta) = \sum_{t=1}^{T} \log Prob(i_t|x_t;\theta_1) + \sum_{t=1}^{T} \log Prob(x_t|x_{t-1}, i_{t-1};\theta_3).$$

Give the factorization of the likelihood function above, we can estimate in two steps...

Step 1: Estimate Markov TPM

- Estimate θ_3, the parameters of the Markov transition probabilities for mileage, conditional on non-replacement of engine (i.e. $i_t = 0$)
- Recall that $x_{t+1} = 0$ if $i_t = 1$

We assume a discrete distribution for $\triangle x_t \equiv x_{t+1} - x_t$, the incremental mileage between any two periods:

$$\triangle x_t = \begin{cases} [0, 5000) & \text{w/prob } p \\ [5000, 10000) & \text{w/prob } q \\ [10000, \infty) & \text{w/prob } 1 - p - q \end{cases}$$

so that $\theta \equiv \{p, q\}$, with $0 < p, q < 1$ and $p + q < 1$.

- $\hat{\theta}_3$ estimated by empirical frequencies: $\hat{p} = \text{freq}\{\triangle x_t \in [0, 5000)\}$, etc.
- Note: this does not require the behavioral model!

Step #2: Estimate Structural Parameters of Cost Function

Start by treating $(\beta, \hat{\theta}_3)$ as given:

1. Fix a guess of (RC, θ_1) the remaining parameters.
2. Iterate on the Bellman Operator for $(\beta, \theta_1, \theta_3, RC)$ using Value Function Iteration to get $V^*(x, \varepsilon)$ or $\tilde{V}^*(x, \varepsilon)$.
3. Calculate conditional choice probabilities (CCPs):

$$Pr(i_t = 1 | x_t, \varepsilon_t, \theta) = \frac{\exp[\tilde{V}_\theta(x_t, \varepsilon_t, 1)]}{\exp[\tilde{V}_\theta(x_t, \varepsilon_t, 0)] + \exp[\tilde{V}_\theta(x_t, \varepsilon_t, 1)]}$$

4. Evaluate the log-likelihood:

$$\log l(\theta) = \sum_{t=1}^{T} \log \Pr(i_t | x_t; \theta_1, RC) + \underbrace{\sum_{t=1}^{T} \log \Pr(x_t | x_{t-1}, i_{t-1}; \hat{\theta}_3)}_{\text{Can Ignore! Why?}}$$

Solve via MLE. This is the Nested Fixed Point algorithm.

Computational Details

That looked easy, except that I never really showed you how to recover $\tilde{V}_\theta(x, i)$:

- Directly iterating on Bellman's operator requires keeping track of ε's which are: (1) unobserved to you the econometrician and (2) continuous and full support (not a discrete grid).
    - AKA a big pain.
- You may (or may not) have learned some tricks for solving Bellman equations in Macro that you could apply here: VFI, Policy Iteration (PI), Howard's Policy Improvement, etc.
    - None of that really tells us how to deal with ε's.

Rust's Trick

- Rust has a nice trick that let's us work with a new function $EV_\theta(x, i)$ instead of $V_\theta(x, i, \varepsilon)$ we call this the ex ante or expected value function.

$$EV(x, i) \equiv E_{x', \varepsilon' | x, i} V(x', \varepsilon'; \theta)$$

- In words $EV_\theta(x, i)$ says at time $t - 1$ what is the expected value of $V_\theta(x_t, \varepsilon_t)$ [eq 4.14].

$$EV(x, i) = \int_y \log \left\{ \sum_{j \in C(y)} \exp[u(y, j; \theta) + \beta EV(y, j)] \right\} p(dy|x, i)$$

- Here x, i denotes the *previous* period's mileage and replacement choice, and y, j denote the *current* period's mileage and choice.

Derivation of Rust's Trick

This ex ante value function can be derived from Bellman's equation:

$$
\begin{aligned}
V(y,\varepsilon;\theta) &= \max_{j\in 0,1}[u(y,j;\theta)+\varepsilon_j+\beta EV(y,j)] \\
\Longrightarrow & E_{y,\varepsilon}[V(y,\varepsilon;\theta)|x,i] \equiv EV(x,i;\theta) \\
&= E_{y,\varepsilon|x,i}\left\{\max_{j\in 0,1}[u(y,j;\theta)+\varepsilon_j+\beta EV(y,j)]\right\} \\
&= E_{y|x,i}E_{\varepsilon|x,i}\left\{\max_{j\in 0,1}[u(y,j;\theta)+\varepsilon_j+\beta EV(y,j)]\right\} \\
&= E_{y|x,i}\log\left\{\sum_{j=0,1}\exp[u(y,j;\theta)+\beta EV(y,j)]\right\} \\
&= \int_y \log\left\{\sum_{j=0,1}\exp[u(y,j;\theta)+\beta EV(y,j)]\right\} p(dy|x,i).
\end{aligned}
$$

Value Function Iteration

1. Start with an initial guess at $\tau = 0$ for $EV_\theta^\tau(x,i)$. A common guess is $EV_\theta^\tau(x,i) = 0$ for all (x,i)
2. Iterate Bellman Operator

   $$T_\theta\left(EV_\theta^\tau\right) = \int_y \log\left\{\sum_{j=0,1} \exp[u(y,j;\theta) + \beta EV^\tau(y,j)]\right\} p(dy|x,i).$$

   with $p(dy|x,i;\hat{\theta}_3)$ estimated in Step 1.
   $T_\theta\left(EV_\theta^\tau(x,i)\right) \equiv EV_\theta^{\tau+1}(x,i)$.
3. Compare $\epsilon(\tau) \equiv \sup_{(x,i)} |EV_\theta^{\tau+1}(x,i) - EV_\theta^\tau(x,i)|$ to ϵ^{tol}. If $\epsilon(\tau) \leq \epsilon^{tol}$ then stop.

See my notes on Numerical Dynamic Programming for more details.

Solving the fixed point

Obvious approach is contraction iterations (VFI):

$$EV_\theta^\tau = T_\theta\left(EV_\theta^{\tau-1}\right) = T_\theta^\tau\left(EV_0\right)$$

Rust actually switches to Newton-Kantorovich Iteration:

$$EV_\theta^{\tau+1} = EV_\theta^\tau - \left[I - T_\theta'\right]^{-1}\left(I - T_\theta\right)\left(EV_\theta^\tau\right)$$

The first is slow, but globally convergent. The second is fast but locally convergent. To get the gradient of the log-likelihood we must also calculate the Jacobian using the IFT:

$$\partial EV_\theta / \partial\theta = \left[I - T_\theta'\right]^{-1} \partial T_\theta\left(EV_\theta\right) / \partial\theta$$

See `https://editorialexpress.com/jrust/nfxp.pdf` for more details.

Value Function Iteration: Bounds

- Suppose we set $V_0 = 0$ then the value function iteration approach is just like solving the finite horizon problem by backward induction.
- The CMT guarantees consistency at a geometric rate or linear convergence with modulus β
- We can derive an expression for the number of steps we need to get an ϵ-approximation.

$$T(\epsilon, \beta) = \frac{1}{|\log(\beta)|} \log\left(\frac{1}{(1-\beta)\epsilon}\right)$$

- This tells us that when $\beta \to 1$ that VFI gets very very slow.

Estimates

TABLE IX

STRUCTURAL ESTIMATES FOR COST FUNCTION $c(x, \theta_1) = .001\theta_{11}x$

FIXED POINT DIMENSION = 90

(Standard errors in parentheses)

Parameter		Data Sample			Heterogeneity Test	
Discount Factor	Estimates/ Log-Likelihood	Groups 1, 2, 3 3864 Observations	Group 4 4292 Observations	Groups 1, 2, 3, 4 8156 Observations	LR Statistic ($df = 4$)	Marginal Significance Level
$\beta = .9999$	RC	11.7270 (2.602)	10.0750 (1.582)	9.7558 (1.227)	85.46	1.2E − 17
	θ_{11}	4.8259 (1.792)	2.2930 (0.639)	2.6275 (0.618)		
	θ_{30}	.3010 (.0074)	.3919 (.0075)	.3489 (.0052)		
	θ_{31}	.6884 (.0075)	.5953 (.0075)	.6394 (.0053)		
	LL	−2708.366	−3304.155	−6055.250		
$\beta = 0$	RC	8.2985 (1.0417)	7.6358 (0.7197)	7.3055 (0.5067)	89.73	1.5E − 18
	θ_{11}	109.9031 (26.163)	71.5133 (13.778)	70.2769 (10.750)		
	θ_{30}	.3010 (.0074)	.3919 (.0075)	.3488 (.0052)		
	θ_{31}	.6884 (.0075)	.5953 (.0075)	.6394 (.0053)		
	LL	−2710.746	−3306.028	−6061.641		
Myopia test:	LR Statistic ($df = 1$)	4.760	3.746	12.782		
$\beta = 0$ vs. $\beta = .9999$	Marginal Significance Level	0.0292	0.0529	0.0035		

Discount factor

- While Rust finds a better fit for $\beta = .9999$ than $\beta = 0$, he finds that high levels of β basically lead to the same level of the likelihood function.
- Furthermore, the discount factor is non-parametrically non-identified. Note: He loses ability to reject $\beta = 0$ for more flexible cost function specifications.

Discount factor

TABLE VIII

SUMMARY OF SPECIFICATION SEARCH[a]

	Bus Group		
Cost Function	1, 2, 3	4	1, 2, 3, 4
Cubic	Model 1	Model 9	Model 17
$c(x, \theta_1) = \theta_{11}x + \theta_{12}x^2 + \theta_{13}x^3$	−131.063	−162.885	−296.515
	−131.177	−162.988	−296.411
quadratic	Model 2	Model 10	Model 18
$c(x, \theta_1) = \theta_{11}x + \theta_{12}x^2$	−131.326	−163.402	−297.939
	−131.534	−163.771	−299.328
linear	Model 3	Model 11	Model 19
$c(x, \theta_1) = \theta_{11}x$	−132.389	−163.584	−300.250
	−134.747	−165.458	−306.641
square root	Model 4	Model 12	Model 20
$c(x, \theta_1) = \theta_{11}\sqrt{x}$	−132.104	−163.395	−299.314
	−133.472	−164.143	−302.703
power	Model 5[b]	Model 13[b]	Model 21[b]
$c(x, \theta_1) = \theta_{11}x^{\theta_{12}}$	N.C.	N.C.	N.C.
	N.C.	N.C.	N.C.
hyperbolic	Model 6	Model 14	Model 22
$c(x, \theta_1) = \theta_{11}/(91 - x)$	−133.408	−165.423	−305.605
	−138.894	−174.023	−325.700
mixed	Model 7	Model 15	Model 23
$c(x, \theta_1) = \theta_{11}/(91 - x) + \theta_{12}\sqrt{x}$	−131.418	−163.375	−298.866
	−131.612	−164.048	−301.064
nonparametric	Model 8	Model 16	Model 24
$c(x, \theta_1)$ any function	−110.832	−138.556	−261.641
	−110.832	−138.556	−261.641

Application

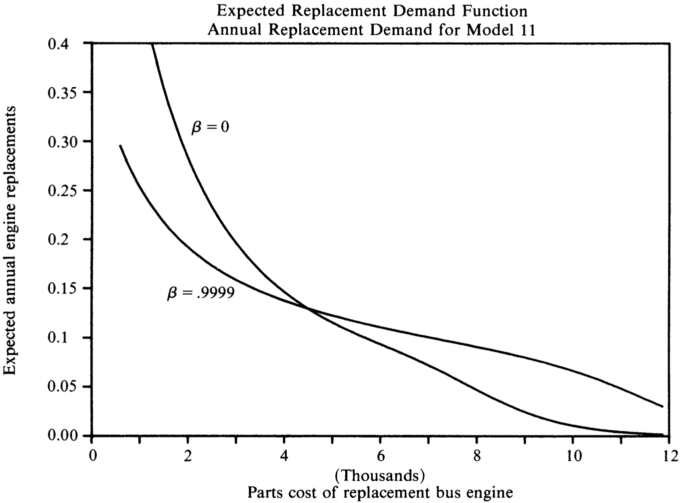

FIGURE 7