

```
capture program drop do_simulation
program define do_simulation
/*
```

This program generates Table 2 regressions, one at a time.

The adj type option specifies which of these types of adjustment costs to use:

```
zero = Zero adjustment costs
sym  = Symmetric quadratic adjustment costs
asym = Asymmetric linear adjustment costs
fixed = Fixed (lump-sum) adjustment costs
```

The pii option specifies the probability of facing the same shock in the next period. Since we simulate 10 different shock levels, we use:

```
0.1 = IID shocks, i.e. equal probability of facing the same shock
0.5 = Persistent shocks, i.e. 50% probability of facing the same shock
```

*/

version 9.2

```
syntax , adjtype(string) pii(real)
```

```
* Define alpha', which is  $\alpha \cdot (1 + 1/\mu)$ , where alpha is the production function
* parameter in  $Q = \phi \cdot \text{labor}^\alpha$  and mu is the elasticity of demand (assuming an
* isoelastic demand curve). We choose alpha and mu based on our estimates of
* labor share of value added and elasticity of demand in Lafontaine and
* Sivadasan -- Within-firm Labor Productivity across Countries: A Case Study.
```

```
scalar alpha = 0.36
scalar mu = -2.5
scalar alphap = alpha * (1 + 1/mu)
```

```
* Set the discount rate and discount factor (beta)
```

```
scalar disrate = 0.08
scalar beta = (1 / (1 + disrate))^(1/52)
```

```
* The wage constant is found by solving for w such that  $L_{max} = 10$  (see page 30).
```

```
scalar wage = exp(ln(alphap) - ln(10) * (1 - alphap))
```

```
* We allow 50 labor levels ranging from .2 to 10.
```

```

local nlabor = 50
scalar laborinc = 0.2

```

* We allow 10 demand/productivity shock levels ranging from .1 to 1.

```

local nphi = 10
scalar phinc = 0.1

```

* Calculate the transition probabilities.

```

scalar tp_ij = (1 - `pii')/(`nphi' - 1) // to be used if phi != phi'
scalar tp_ii = tp_ij + (`nphi'*`pii' - 1)/(`nphi' - 1) // to be used if phi == phi'

```

* Define the part of the equation that deals with adjustment costs.

```

local adjcosts "" // zero case
if "`adjtype'" == "sym" {
    local adjcosts "- c_sym*(labor - Lprime)^2"
}
if "`adjtype'" == "asym" {
    local adjcosts "- c_asym*(labor - Lprime)*(Lprime<labor)"
}
if "`adjtype'" == "fixed" {
    local adjcosts "- c_fix*(Lprime != labor)"
}

```

* Simulate 45 adjustment cost regimes, ranging from 0 to 44.

```

local lastregime = 44
forvalues i reg = 0/`lastregime' {

    // Calculate the adjustment cost parameters
    scalar c_sym = `i reg' / `lastregime' * wage
    scalar c_fix = `i reg' / `lastregime' * wage * 4
    scalar c_asym = c_fix

    // Start with nphi observations. Create phi_n so that we always
    // know which instance of phi this is
    drop _all
    quietly set obs `nphi'
    gen double phi = _n*phinc
    gen phi_n = _n

```

```

// Set the initial values of the function at the static optimum.

gen double V1 = (alphap*phi/wage)^(1/(1 - alphap)) // labor value
quietly replace V1 = (phi *V1^alphap - wage*V1)/(1 - beta)

// Expand the data to allow for nlabor choices.

quietly expand `nlabor'
sort phi
by phi: gen double labor = _n*labornc

/*
Find the optimal policy functions by solving numerically the Bellman
equation (A4). We start and end with nphi *nlabor observations
representing all the possible combinations of phi and labor.

We iterate until such time as

V2(phi,labor) = max( ... + beta*E[V1(phi',labor')|phi] )

where V1 contains the values of V2 at the previous iteration.

At each iteration, the dataset is expanded by nlabor and V2 is
evaluated for all values of labor'. Only the record with the largest
V2 is kept for each phi labor group.

Convergence is achieved when the sum(V2-V1)^2 is close to zero and
labor' stops changing.
*/

scalar i block = `nlabor' * `nlabor'
scalar j block = `nlabor'
gen double Lsave = 0
local nzero = 0
local iter = 0
scalar di ffsq = 999

while (`nzero' < 5) | (di ffsq > 1) {
    local iter = `iter' + 1

    // expand the dataset such that for each phi labor, we have
    // nlabor records representing labor' values

    quietly expand `nlabor'
    sort phi labor

    // To calculate the expected values, we loop over values of phi'.
    /***** Page 3 *****/

```

```

// By grouping records by phi labor, we use the current observation
// number (_n) to indicate which labor' we are doing. We can then
// use explicit subscripting to obtain the value of V1(phi', labor').
// We have i block records per value of phi and j block records per
// value of labor.

gen double V2 = 0
forvalues i = 1/`nphi' {
    by phi labor: gen ij_n = (`i'-1)*iblock + _n*jblock
    quietly replace V2 = V2 + beta*V1[ij_n]* ///
        (tp_ij*(phi_n != `i') + tp_ii*(phi_n == `i'))
    drop ij_n
}

// calculate the rest of the function with the appropriate adj. costs

by phi labor: gen double Lprime = _n*Laborinc
quietly replace V2 = V2 + phi*Lprime^al phap - wage*Lprime `adj costs'

// For each phi labor, select the record with the largest V2. The
// variable Lprime contains the optimal labor choice candidate.

sort phi labor V2
quietly by phi labor: keep if _n == _N

// Note how we are doing.

gen x = (V2-V1)^2
quietly sum x
scalar dffsq = r(sum)
drop x

quietly count if Lprime != Lsave
local ndiff = r(N)
if `ndiff' == 0 {
    local nzero = `nzero' + 1
}
else {
    local nzero = 0
}

drop V1 Lsave
rename V2 V1
rename Lprime Lsave
} // end of iterations

keep Lsave
rename Lsave Lprime

```

```

display as text "Converged [" as result "`adjtype', pii=" `pii' ///
as text "]" regime = " as result `ireg' ///
as text " iter = " as result `iter' ///
as text " nzero = " as result `nzero' ///
as text " di ffsq = " as result di ffsq

/*
Generate simulated data. We use explicit subscripting to obtain the
labor choice stored in Lprime. Because of this, we keep the sample at
nphi*nlabor even though we only keep 75 outlet records per period at
the end.

Generate data for 153 periods. Skip the first 48 periods then start
saving. We save a total of 105 periods, the first of which will be
dropped later when lagging. This leaves 104 periods that simulate 2
* 52 weeks of data.
*/

gen int outlet = _n
gen week = 0
gen phi0 = int(uni form()*`nphi' + 1) * phi inc
gen labor = (al phap*phi0/wage)^(1/(1-al phap))

// Stata's random number generator returns uniformly distributed
// pseudorandom numbers on the interval [0,1). Use a cutoff value
// to select, in the case of persistent shocks, a phi = phi0 with
// probability pii.

scalar pii_cutoff = (`pii' - (1/`nphi')) / (1 - (1/`nphi'))

local ikeep = 0
forval wk = 1/153 {
    gen rand_pii = uni form()
    gen rand_phi = uni form()

    quietly gen phi = phi0 if rand_pii <= pii_cutoff
    quietly replace phi = int(rand_phi*`nphi' + 1) * phi inc if phi == .

    // Dataset has nphi*nlabor obs.
    gen ij_n = round(phi/phi inc-1)*`nlabor' + round(labor/labor inc)
    quietly replace labor = Lprime[ij_n]

    drop rand_pii rand_phi ij_n phi0

    if `wk' >= 49 {
        quietly replace week = `ikeep'
        tempfile f`ikeep'
        /****** Page 5 *****/
    }
}

```

```

                quietly save `f`ikeep'
                local ikeep = `ikeep' + 1
            }
            rename phi phi0
        }

        // combine all periods

        rename phi0 phi
        forvalues wk = 0/103 {
            append using `f`wk'
        }

        // We only need data for 75 outlets.

        quietly drop if outlet > 75
        drop Lprime
        gen regime = `ireg'

tempfile regdata`ireg'
quietly save `regdata`ireg'
} // forvalues ireg = 0/`lastregime'

```

* Combine the simulated data for all adjustment cost regimes run do the regression

```

use `regdata0', clear
forvalues ireg = 1/`lastregime' {
    append using `regdata`ireg'
}

```

* Order the data by regime, outlet, and week. Then lag labor and drop week 0

```

sort regime outlet week
gen log_labor = log(labor*wage)
quietly by regime outlet: gen lag_labor = log_labor[_n-1]
quietly drop if week == 0

```

* Generate output given the underlying production function parameters (equation A1)

```

scalar theta = 1
gen log_Quantity = log(theta*labor^alpha)

```

* In equation A2, we assume an isoelastic demand $P = \lambda * Q^{(1/\mu)}$

* When $\theta == 1$, $\phi = \lambda * \theta^{(1+1/\mu)}$ and so $\phi == \lambda$.

```

gen log_price = log(phi) + 1/mu*log_Quantity
gen log_revenue = log_Quantity + log_price
gen regime_lag = (regime/`lastregime' - 0.5)*lag_labor // Adjustment cost X Log(Lagged Labor cost)
gen regime_rev = (regime/`lastregime' - 0.5)*log_revenue // Adjustment cost X Log(Revenue)

```

- * Create the categorical variable that will represent season dummies.
- * Records are already ordered by regime outlet and week. Use a running
- * sum to create a new dummy each 13 weeks.

```

gen reg_outlet_season = sum(mod(week, 13) == 1)

```

```

areg log_labor lag_labor log_revenue regime_lag regime_rev, ///
      absorb(reg_outlet_season) cluster(regime)

```

```

save_results

```

```

end

```

```

clear

```

```

set seed 123456789 // for replicability purposes

```

```

quietly include save_results

```

```

foreach i in zero sym asym fixed {
  foreach j in 0.1 0.5 {
    do_simulation , adjtype(`i') pi(`j')
  }
}

```

```

save_results, list

```