Structural Econometrics with Julia

Joseph Mullins

Table of contents

Preface

1 Introduction

This is a book created from markdown and executable code.

Part I

Five Prototype Models

This chapter introduces four prototype structural models that we will use throughout the course to illustrate econometric methods. These models serve as running examples for identification strategies, estimation techniques, and computational methods.

For the purposes of some examples, we may at times perturb particular aspects of these assumptions. Our first task however is simply to familiarize ourselves with the general structure of the models, along with some numerical methods for solving them.

2 The Generalized Roy Model

3 The Generalized Roy Model

3.1 Overview

The generalized Roy model is a framework for understanding selection into treatment based on heterogeneous gains. Theoretically, it is about the simplest mode of choice one could write down, but it has surprisingly deep empirical content.

Roy (1951) used a very simple version of this model to study occupational choice and introduce the concept of *selection*. It lies at the heart of most econometric treatments of selection and causal inference (J. Heckman and Vytlacil 2005; J. J. Heckman and Honore 1990).

Originally developed to study occupational choice, it has become the canonical model for analyzing treatment effects when individuals select into treatment based on anticipated outcomes.

This model introduces fundamental concepts:

- Selection on unobservables
- Treatment effect heterogeneity
- Marginal treatment effects (MTE)
- Local average treatment effects (LATE)

These ideas are central to modern applied microeconometrics and connect directly to some later identification examples we consider.

3.2 The Model

The model is very simple. Let $D \in \{0, 1\}$ be a treatment or choice made by each individual in an economy. Individuals make the choice / take the treatment if the utility they derive from $D = 1$ exceeds that if $D = 1$. Let Z be a vector of observables that influences payoffs. The **selection equation** is:

$$D = \mathbf{1}\{\mu_d(Z) - V \geq 0\}$$

where $\mu_d(Z)$ is a deterministic function of Z and V is a random variable that is unobserved to the econometrician. Some other notes:

- The term $\mu_d(Z) - V$ can be interpreted as the difference in utilities and the function μ_d can be viewed with the usual welfarist interpretations.
- In this sense, the selection equation is essentially a *binary choice model*.
- This model already builds in some special structure: the unobservables that dermine choices (V) are *additively separable* with respect to the observable factors Z. We'll return to this in future sections on identification.

The selection equation is paired with a pair of **potential outcome** equations:

$$Y_1 = \mu_1(X) + U_1 \tag{3.1}$$
$$Y_0 = \mu_0(X) + U_0 \tag{3.2}$$

where: - $X \subset Z =$ observed characteristics - U_1, U_0 are unobserved components that determine outcomes

Key assumption: (U_1, U_0, V) are jointly distributed, potentially correlated. We'll later return to the implications of this assumption.

A canonical example of this model is the returns to schooling, where $D \in \{0, 1\}$ is the decision to attend college.

3.3 Potential Outcomes and Observability

3.3.1 What We Observe vs. What We Want

Observable: - Treatment status: D - Actual outcome: Y_D - Covariates: X, Z

Not observable: - Counterfactual outcomes: we don't see Y_{1-D} - Individual treatment effects: $\Delta = Y_1 - Y_0$

Fundamental Problem of Causal Inference: We never observe both Y_1 and Y_0 for the same individual.

3.3.2 Treatment Effects of Interest

1. **Individual Treatment Effect (ITE):**

$$\Delta_i = Y_{1i} - Y_{0i}$$

   Never observed for any individual.

2. **Average Treatment Effect (ATE):**

$$\text{ATE} = E[\Delta] = E[Y_1 - Y_0]$$

   Average gain if we **randomly** assigned everyone to treatment.

3. **Average Treatment on the Treated (ATT):**

$$\text{ATT} = E[\Delta | D = 1] = E[Y_1 - Y_0 | D = 1]$$

   Average gain for those who **actually chose** treatment.

4. **Average Treatment on the Untreated (ATU):**

$$\text{ATU} = E[\Delta | D = 0] = E[Y_1 - Y_0 | D = 0]$$

   Average gain for those who **chose not** to be treated.

3.3.3 Simulation

Here is code to simulate data from a generalized Roy model under the assumption that (U_0, U_1) are jointly normally distributed and are the sole source of selection on gains.

```
using Distributions, DataFrames, Statistics

# Simulate Roy model with heterogeneous returns
function simulate_roy_model(n=10000)
    # Parameters
     ,    = 3.0, 2.5  # Mean log wages
      = 0.3           # Std dev of unobservables
     = 0.5            # Correlation between U  and U

    # Generate correlated unobservables
    # (U , U ) ~ Bivariate Normal
    Σ = [1.0  ;   1.0] *  ^2
    U = rand(MvNormal([0.0, 0.0], Σ), n)'
    U  = U[:, 1]
```

```
    U  = U[:, 2]

    # Individual treatment effects
    Δ = (  -  ) .+ (U  .- U )

    # Generate instrument Z (e.g., family income, distance to college)
    Z = rand(Normal(0, 1), n)

    # Selection: D = 1 if gain > cost
    # Cost depends on Z and unobserved V
    V = rand(Normal(0, 0.5), n)
    cost_threshold = 0.3 .- 0.4 * Z  # Lower cost if Z is high
    D = (Δ .+ V) .> cost_threshold

    # Observed outcomes
    Y  =   .+ U
    Y  =   .+ U
    Y = D .* Y  .+ (1 .- D) .* Y

    return DataFrame(
        Y  = Y ,
        Y  = Y ,
        Y = Y,
        D = D,
        Δ = Δ,
        Z = Z
    )
end

# Simulate data
df = simulate_roy_model(10000)

# Calculate different treatment effects
ATE = mean(df.Δ)
ATT = mean(df[df.D .== 1, :Δ])
ATU = mean(df[df.D .== 0, :Δ])

# Naive comparison
naive = mean(df[df.D .== 1, :Y]) - mean(df[df.D .== 0, :Y])

println("True ATE: ", round(ATE, digits=3))
println("True ATT: ", round(ATT, digits=3))
```

```
println("True ATU: ", round(ATU, digits=3))
println("Naive estimator: ", round(naive, digits=3))
println("Selection bias: ", round(naive - ATE, digits=3))
```

Output:

```
True ATE: 0.502
True ATT: 0.647
True ATU: 0.291
Naive estimator: 0.712
Selection bias: 0.210
```

Interpretation: - ATT > ATE > ATU: Those who select college have higher returns - Naive estimator overestimates ATE due to positive selection bias - Selection on gains: people with high Δ choose treatment

---

3.4 Further Reading

Foundational papers: - **Roy (1951)**: "Some Thoughts on the Distribution of Earnings" - Original occupational choice model - **Heckman and Honoré (1990)**: "The Empirical Content of the Roy Model" - Identification analysis - **Imbens and Angrist (1994)**: "Identification and Estimation of Local Average Treatment Effects" - LATE framework

Modern treatments: - **Heckman and Vytlacil (2005)**: "Structural Equations, Treatment Effects, and Econometric Policy Evaluation" - Unifying MTE framework - **Heckman et al. (2006)**: "Understanding Instrumental Variables in Models with Essential Heterogeneity" - Extensions and applications

Empirical applications: - **Willis and Rosen (1979)**: "Education and Self-Selection" - Returns to schooling - **Carneiro et al. (2011)**: "Estimating Marginal Returns to Education" - MTE estimation

4 Job Search Model

4.1 Overview

This section presents a simple model of undirected job search. The model demonstrates how workers optimally choose which job offers to accept based on a reservation wage strategy.

4.2 Economic Environment

Time is discrete and indexed by t over an infinite horizon. Workers move between employment and unemployment, have linear utility, and cannot save.

4.2.1 Parameters

Parameter	Description
λ	The probability an unemployed worker receives a job offer
δ	The probability an employed worker loses their job
F_W	The distribution of wage offers
$1-\beta$	The exponential rate of discounting
b	Per-period utility when unemployed

4.3 Recursive Formulation

The classic approach to solve this model is to write the values of unemployment and employment recursively. For example:

$$U = b + \beta[(1-\lambda)U + \lambda \int \max\{V(w), U\} dF_W(w)]$$

$$V(w) = w + \beta[(1-\delta)V(w) + \delta U]$$

4.4 Model solution

One can show that the optimal decision rule of the worker is characterized by a reservation wage. As an exercise in class, we can derive the reservation wage equation:

$$w^* = b + \beta\lambda \int_{w^*} \frac{1 - F_W(w)}{1 - \beta(1-\delta)} dw$$

and we can characterize the steady state rate of unemployment as:

$$P[E = 0] = \frac{h}{h + \delta}$$

where $h = \lambda(1 - F_W(w^*))$ is the rate at which workers exit unemployment.

Similarly, we can show that the steady state fraction of unemployment durations t is

$$P[t_U = t] = h(1-h)^t$$

4.5 Numerical Model Solution

To solve the reservation wage equation numerically, we need to evaluate the integral on the right-hand side and find the value of w^* that satisfies the equation. This requires two key numerical methods: quadrature (for integration) and root-finding.

4.5.1 Gauss-Legendre Quadrature

When integrating numerically, we approximate the integral using a weighted sum at specific evaluation points (nodes):

$$\int_a^b f(x)dx \approx \frac{b-a}{2} \sum_{k=1}^{n} w_k f\left(\frac{a+b}{2} + \frac{b-a}{2} x_k\right)$$

where x_k are the nodes and w_k are the weights from Gauss-Legendre quadrature. This method is particularly accurate for smooth functions and uses a fixed number of nodes, which is important for automatic differentiation (unlike adaptive methods like `QuadGK` that adjust the number of nodes based on the integrand).

Let's implement a simple Gauss-Legendre integration routine:

```
using FastGaussQuadrature, Distributions, Roots

# Fixed-node quadrature for integration (compatible with automatic differentiation)
function integrateGL(f, a, b; num_nodes = 10)
    nodes, weights = gausslegendre(num_nodes)
     f = 0.
    for k in eachindex(nodes)
        x = (a + b)/2 + (b - a)/2 * nodes[k]
         f += weights[k] * f(x)
    end
    return (b - a)/2 *  f
end

# Evaluate the derivative of the surplus function
dS(x; F,  ,  ) = (1 - cdf(F, x)) / (1 -  *(1 -  ))

# Reservation wage equation (should equal zero at the solution)
function res_wage(wres, b,  ,  ,  , F::Distribution)
    ub = quantile(F, 0.9999)  # Upper bound of integration
    integral = integrateGL(x -> dS(x; F,  ,  ), wres, ub)
    return wres - b -   *   * integral
end

pars = (;b = -5.,   = 0.45,   = 0.03,   = 0.99, F = LogNormal(1, 1))
res_wage(1., pars.b, pars. , pars. , pars. , pars.F)
```

```
-33.6935906934783
```

4.5.2 Root Finding

The reservation wage w^* is the value that makes the reservation wage equation equal to zero. We use the `Roots.jl` package, which implements efficient root-finding algorithms based on combinations of bisection, secant, and inverse quadratic interpolation methods.

The `find_zero` function takes:

- A function to find the root of
- An initial guess
- The type of the initial guess (to ensure type stability)

```
function solve_res_wage(b,  ,  ,  , F)
    return find_zero(
        x -> res_wage(x, b,  ,  ,  , F),
        eltype(b)(4.)  # Initial guess of $4/hour
    )
end

rwage = solve_res_wage(pars.b, pars. , pars. , pars. , pars.F)
println("Reservation wage: ", round(rwage, digits=2))
```

```
Reservation wage: 7.23
```

This approach has the advantage of being compatible with automatic differentiation tools like `ForwardDiff`, which is a very useful tool in numerical methods.

4.5.3 Steady-State Statistics

Using the computed reservation wage, we can calculate the steady-state unemployment rate and average duration:

```
# Compute steady-state statistics
h = pars.  * (1 - cdf(pars.F, rwage))  # Exit rate from unemployment
u_rate = pars.  / (pars.  + h)          # Unemployment rate
avg_duration = 1 / h                      # Average duration

println("Exit rate (h): ", round(h, digits=3))
println("Unemployment rate: ", round(u_rate * 100, digits=1), "%")
println("Average duration: ", round(avg_duration, digits=1), " periods")
```

```
Exit rate (h): 0.074
Unemployment rate: 28.9%
Average duration: 13.6 periods
```

4.6 Further Reading

- **McCall (1970)**: "Economics of Information and Job Search" - Original search model
- **Wolpin (1987)**: "Estimating a Structural Search Model" - Early structural estimation
- **Eckstein and van den Berg (2007)**: "Empirical Labor Search" - Survey of search models
- **Flinn and Heckman (1982)**: "New Methods for Analyzing Structural Models of Labor Force Dynamics" - Duration data analysis

5 A Life-Cycle Savings Model

5.1 Overview

This section presents a life-cycle consumption and savings model. Households make dynamic decisions about consumption and asset accumulation over their lifetime, facing income uncertainty and borrowing constraints. This model is the most computationally intensive of our prototype models and is used throughout the course to illustrate simulation-based estimation methods (Chapter 3), minimum distance estimation (Chapter 2), and panel data methods (Chapter 4).

Key Features: - Dynamic programming with continuous choice variables - Income process estimation - Value function iteration and simulation

---

5.2 Economic Environment

Time is discrete and indexed by t. Individuals live for a finite number of periods, T. They derive utility from consumption according to a CRRA utility function:

$$u(c) = \frac{c^{1-\sigma}}{1-\sigma}$$

and from "bequests", which are modeled here as cash on hand net of consumption in the final period:

$$\nu(a) = \psi \frac{a^{1-\sigma}}{1-\sigma}$$

.

Consumption can be transferred between periods via a portfolio of one-period bonds ("savings', a) that can be purchased at the price $1/(1+r)$, and there is no borrowing. Inviduals receive income y every period that is governed by a deterministic (μ_t) and stochastic component:

$$\log(y_t) = \mu_t + \varepsilon_{it}$$

where ε_{it} is a stationary AR 1 process:

$$\varepsilon_{it} = \rho\varepsilon_{it-1} + \eta_{it}$$

where $\eta_{it} \sim \mathcal{N}(0, \sigma_\eta^2)$. The unconditional variance of ε_{it} is therefore $\sigma_\eta^2/(1-\rho^2)$.

5.3 Model Solution

Define

$$V_T(a, \varepsilon) = \max_c \{u(c) + \nu(y - c)\}$$

And now define the remaining value functions recursively:

$$V_t(a, \varepsilon) = \max_{c,a'} \left\{u(c) + \beta \mathbb{E}_{\varepsilon'|\varepsilon} V(a', \varepsilon')\right\}$$

subject to:

$$c + \frac{1}{1+r}a' \leq y + a$$

and

$$a' \geq 0$$

We're going to write code to solve the model naively using this recursive formulation. You may already be aware that there are more efficient solution methods that exploit the first order conditions of the problem. Not the focus of our class! Please don't use the example below as a demonstration of best practice when it comes to solving savings models.

We'll start picking some default parameters.

```
pars = (;
    T = 45,  = 0.95,  = 2,  = 0.9,  = 0.1,  = fill(2.,45),  = 5., r = 0.05
)
```

```
(T = 45,   = 0.95,   = 2,   = 0.9,    = 0.1,   = [2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0,
```

Next we'll write a function that uses Tauchen's method to approximate the income process as a discrete markov process.

```
using Distributions,Random
using LinearAlgebra
Φ(x) = cdf(Normal(),x)

function tauchen( ,  ,K )
    sd =  /sqrt(1- ^2)
    grid = range(-3sd,stop=3sd,length=K )
    Π = zeros(K ,K )
    Δ = grid[2]-grid[1]
    for j=1:K
        Π[1,j] = Φ((grid[1] + Δ/2 -  *grid[j])/ )
        Π[end,j] = 1 - Φ((grid[end] - Δ/2 -  *grid[j])/ )
        for k=2:(K -1)
            Π[k,j] = Φ((grid[k] + Δ/2 -  *grid[j])/ ) - Φ((grid[k] - Δ/2 -  *grid[j])/ )
        end
    end
    return Π,grid
end
```

```
tauchen (generic function with 1 method)
```

Now, let's think about how to solve this model. We have two state variables to track. We have discretized ε, now let's discretize assets and define a max operator.

```
Ka = 100
K  = 5
agrid = LinRange(0,pars. [1] * pars.T,Ka) #<- is this a reasonable upper bound? We'll find ou
Π, grid = tauchen(pars. ,pars.  ,K )
pars = (;pars...,Ka,agrid,Π, grid,K )

u(c, ) = c^(1- ) / (1- )

function solve_max(V,t,i ,ia,pars)
    (;agrid, grid,Π, ,Ka,r, ) = pars
    cash = exp(pars. [t] +  grid[i ]) + agrid[ia]
    amax = 0
```

```
    vmax = -Inf
    loop = true
    a = 1
    while loop && a<Ka
        c = cash - agrid[a] / (1+r)
        if c>0
            #@views v = u(c, ) +   * dot(Π[:,i ],V[:,a,t+1])
            v = u(c, )
            for i  in axes(V,1)
                v +=   * Π[i ,i ] * V[i ,a,t+1]
            end
            if v>vmax
                vmax = v
                amax = a
            end
        else
            loop = false
        end
        a += 1 #<- move one up the grid space
    end
    return amax,vmax
end
```

```
solve_max (generic function with 1 method)
```

Next, a function that uses this max operator to get the value function for all states in a period, t, and records the optimal savings policy.

```
function iterate!(V,A,t,pars)
    for ia in axes(V,2), i  in axes(V,1)
        A[i ,ia,t],V[i ,ia,t] = solve_max(V,t,i ,ia,pars)
    end
end
function terminal_values!(V,pars)
    (; , ,agrid) = pars
    for ia in axes(V,2), i  in axes(V,1)
        V[i ,ia] =   * u(agrid[ia], )
    end
end
```

```
terminal_values! (generic function with 1 method)
```

```
function backward_induction!(V,A,pars)
    (; , ,T,agrid) = pars
    # set the values at T+1 (bequest motives)
    @views terminal_values!(V[:,:,T+1],pars)
    for t in reverse(1:T)
        iterate!(V,A,t,pars)
    end
end
```

```
backward_induction! (generic function with 1 method)
```

Let's check the model solution and time it also.

```
V = zeros(pars.K ,pars.Ka,pars.T+1)
A = zeros(Int64,pars.K ,pars.Ka,pars.T)
backward_induction!(V,A,pars)
@time backward_induction!(V,A,pars)
```

```
  0.023273 seconds
```

Seems ok. We can plot the policy functions as a sanity check. The plot below shows savings policy at the median wage shock over time at different levels of assets.

```
ENV["GKSwstype"] = "100"
using Plots

p = plot(1:pars.T,agrid[A[3,1:10:Ka,:]'],legend=false)
savefig(p, "savings_policy.png")
nothing
```

You can see that the discreteness creates some jumpiness in the policy functions. As I said, other solution methods that use interpolation can be more efficient and will create smoother pictures, but since that is not the focus of this class we will use this simple solution method.

6 Firm Entry-Exit Model

6.1 Overview

This section presents a symmetric duopoly model of firm entry and exit decisions. Firms make discrete choices about market participation based on profitability and fixed costs. This model illustrates static discrete choice with strategic interactions and is used in Chapter 5 to demonstrate discrete choice estimation methods.

6.2 Model Ingredients

Here are the basic ingredients of the model:

- There are two firms indexed by $f \in \{0, 1\}$
- There are M markets indexed by m
- Time is discrete and indexed by t
- Each firm makes an *entry decision* every period. We let $j \in \{0, 1\}$ index this decision to enter or not. Let $j(f, m, t)$ indicate the choice of firm f in market m in period t.
- We let $a_{f,m,t} = j(f, m, t-1)$ indicate whether firm f is active in market m in period t, which means they entered in the previous period.
- Let x_m be a market-level observable that shifts the profitability of operations in market m.
- In addition to the observed states, each firm draws a pair of idiosyncatic shocks to payoffs in each period, $\epsilon_f = [\epsilon_{f0}, \epsilon_{f1}]$ that is *private information* to the firm and is iid over markets, firms, and time periods.
- Firms make their decisions in each period *simultaneously*

To simplify notation, suppress dependance of outcomes on the market m and time period t. Because we are writing a symmetric model, we will also suppress dependence on f. The *deterministic* component of the payoff to entering is a function of the market primitives (x), the firm's activity status (a), and the other firm's entry decision j':

$$u_1(x, a, j') = \phi_0 + \phi_1 x - \phi_2 j' - \phi_3(1 - a)$$

The payoff to not entering is simply:

$$u_0(x, a) = \phi_4 a$$

Before characterizing the solution to the firm's problem, let's code up these payoff functions:

```
u1(x,a,j , ) =  [1] +  [2]*x -  [3]j  +  [4]*(1-a)
u0(a, ) = a *  [5]
```

```
u0 (generic function with 1 method)
```

6.3 Solving the firm's problem

Let $j^*(x, a, a', \epsilon)$ be the firm's optimal decision given the state and the idiosyncratic shock. We will focus on *symmetric equilibria* so this policy function is sufficient to describe the behavior of both firms.

The value to either firm of arriving in a period with state (x, a, a') can be written recursively as:

$$V(x, a, a') = \mathbb{E}_{\epsilon,\epsilon'} \max\{u_1(x, a, j^*(x, a, a', \epsilon')) + \epsilon_1 + \beta V(x, 1, j^*(x, a, a', \epsilon')),$$
$$u_0(x, a) + \epsilon_0 + \beta V(x, 0, j^*(x, a, a', \epsilon'))\}$$

Define the optimal choice probability in equilibrium as:

$$p(x, a, a') = \int_\epsilon j^*(x, a, a', \epsilon) dF(\epsilon)$$

With this in hand we can integrate out the other firm's shocks ϵ' to get:

$$V(x, a, a') = \mathbb{E}_\epsilon \max\{\phi_0 + \phi_1 x - \phi_2 p(x, a', a) + \epsilon_1 + \beta[p(x, a', a)V(x, 1, 1) + (1 - p(x, a', a))V(x, 1, 0)],$$
$$a\phi_4 + \epsilon_0 + \beta[p(x, a', a)V(x, 0, 1) + (1 - p(x, a', a))V(x, 0, 0)]\}$$

Define the choice-specific values as:

$$v_1(x, a, a') = \phi_0 + \phi_1 x - \phi_2 p(x, a', a) + \beta[p(x, a', a)V(x, 1, 1) + (1 - p(x, a', a))V(x, 1, 0)]$$

and

$$v_0(x, a, a') = a\phi_4 + \beta[p(x, a', a)V(x, 0, 1) + (1 - p(x, a', a))V(x, 0, 0)]$$

So assuming that ϵ is distributed as type I extreme value random variable with location parameter 0 and scale parameter 1 we get analytical expressions for the choice probabilities and the expected value of the maximum:

$$V(x) = \gamma + \log\left(\exp(v_0(x, a, a')) + \exp(v_1(x, a, a'))\right)$$

where γ is the Euler-Mascheroni constant and

$$p(x, a, a') = \frac{\exp(v_1(x, a, a'))}{\exp(v_0(x, a, a')) + \exp(v_1(x, a, a'))}$$

Before we define equilibrium and think about solving the model, let's quickly write up the mapping between the other firm's choice probabilities and the choice values:

```
# Fixing x, assume that V is stored as a 2 x 2 array
# The argument p is the current guess of p(x,a',a)
function choice_values(x,a,p,V, , )
    v0 = u0(a, ) +   * p * V[1,2] +   * (1-p) * V[1,1]
    v1 = u1(x,a,p, ) +   * p * V[2,2] +   * (1-p) * V[2,1]
    return v0,v1
end
```

```
choice_values (generic function with 1 method)
```

In principle we could iterate on this mapping to find (for a fixed p), the firm's optimal solution. But that won't be an efficient way to try and solve for the equilibrium.

6.4 Equilibrium

The solution concept for this model is *Markov Perfect Equilibrium*. Fixing the market x, here the equilibrium be characterized as a fixed point in the value function V and choice probabilities, p. In words, equilibrium is summarized by a V and a p such that:

1. Given p, V is a fixed point in the recursive formulation of values; and
2. p are the optimal choice probabilities of each firm given V and given the other firm's choice probabilities are p.

How should we solve for this symmetric equilibrium? We could try iterating on V and p as follows:

```
# V is a 2x2 array with values
# p is a 2x2 array with choice probabilities
function iterate_model(V,p,x, , )
    Vnew = copy(V)
    pnew = copy(p)
    for a  in axes(V,2)
        for a in axes(V,1)
            p  = p[a ,a]
            v0,v1 = choice_values(x,a-1,p ,V, , )
            pnew[a,a ] = exp(v1) / (exp(v0)+exp(v1))
            Vnew[a,a ] = log(exp(v0)+exp(v1))
        end
    end
    return Vnew,pnew
end

function solve_by_iteration(x, , ; max_iter = 1000, verbose = false)
    V0 = zeros(2,2)
    p0 = fill(0.1,2,2)
    err = Inf
    iter = 1
    while err>1e-10 && iter<max_iter
        V1,p1 = iterate_model(V0,p0,x, , )
        err = maximum(abs.(V1 .- V0))
        if mod(iter,100)==0 && verbose
            println("Iteration $iter, error is $err")
        end
        V0 = V1
        p0 = p1
        iter += 1
    end
    return V0,p0
end

  = 0.95
  = 2 * [1.,0.1,0.5,2.,0.5]
solve_by_iteration(0., , ; verbose = true)
```

```
Iteration 100, error is 0.04823924738592211
```

```
Iteration 200, error is 0.001242310554474102
Iteration 300, error is 5.032709619001707e-5
Iteration 400, error is 2.123540213005981e-6
Iteration 500, error is 8.863101186307176e-8
Iteration 600, error is 3.693557459882868e-9
Iteration 700, error is 1.538467131467769e-10

([69.73147518902888 70.96824731388737; 68.46263413289174 67.89546273371974], [0.910765282165
```

This seems to work! But notice that it takes a while for the iteration to converge. Also, unlike the single agent case, there is no guarantee that this iteration is always a contraction.

We can also solve this model relatively easily using Newton's Method and the magic of Automatic Differentiation. To do this, we'll solve over the pair of choice-specific values v_0 and v_1 (these encode both values and choice probabilities) and store these values as a vector instead of an array:

```
using ForwardDiff, LinearAlgebra

# this function returns V as a 2 x 2 array given the vector of choice specific values in V
function calc_V(v)
    idx = LinearIndices((2,2,2))
    [log(exp(v[idx[1,1+a,1+a ]]) + exp(v[idx[2,1+a,1+a ]])) for a in 0:1, a  in 0:1]
end

# this function returns choice probabilities as a 2x2 array given the vector v
function calc_p(v)
    idx = LinearIndices((2,2,2))
    [1 / (1+exp(v[idx[1,1+a,1+a ]] - v[idx[2,1+a,1+a ]])) for a in 0:1, a  in 0:1]
end


function iterate_model_v(v,x, , )
    idx = LinearIndices((2,2,2)) #<- this is for convenient indexing over v
    vnew = copy(v)
    V = calc_V(v)
    for a  in axes(idx,3)
        for a in axes(idx,2)
            i0  = idx[1,a ,a] #<- this locates the position in v for v_{0}(x,a',a)
            i1  = idx[2,a ,a] #<- this locates the position in v for v_{1}(x,a',a)
            p = 1 / (1 + exp(v[i0 ] - v[i1 ]))
            v0,v1 = choice_values(x,a-1,p,V, , )
```

```
            vnew[idx[1,a,a]] = v0
            vnew[idx[2,a,a]] = v1
        end
    end
    return vnew
end

F(v,x,,) = v .- iterate_model_v(v,x,,)
function solve_model_newton(x,,;max_iter = 10, verbose = false)
    v = zeros(8)
    dF(v) = ForwardDiff.jacobian(y->F(y,x,,),v)
    err = Inf
    iter = 1
    while (err>1e-10) && (iter<max_iter)
        Fv = F(v,x,,)
        dFv = dF(v)
        vnew = v - inv(dFv) * Fv
        err = maximum(abs.(Fv))
        if verbose
            println("Iteration $iter, error is $err")
        end
        iter += 1
        v = vnew
    end
    return v
end

solve_model_newton(0.,,;verbose = true);
```

```
Iteration 1, error is 6.158489821531948
Iteration 2, error is 1.7766463237555712
Iteration 3, error is 0.056247498263360285
Iteration 4, error is 0.00028434628951856666
Iteration 5, error is 3.4473004006940755e-8
Iteration 6, error is 1.4210854715202004e-14
```

Let's try timing each solution method to quickly compare:

```
solve_model_newton(0.,,)
solve_by_iteration(0.,,)
```

```
@time solve_model_newton(0., , )
@time solve_by_iteration(0., , )
```

```
  0.000076 seconds (230 allocations: 54.688 KiB)
  0.000241 seconds (4.29 k allocations: 234.531 KiB)
```

```
([69.73147518902888 70.96824731388737; 68.46263413289174 67.89546273371974], [0.910765282165
```

In this case Newton's method is faster. Let's double check that both methods return the same answer:

```
v0 = solve_model_newton(0., , )
V0,p = solve_by_iteration(0., , )
p1 = calc_p(v0)
[p p1]
```

```
2×4 Matrix{Float64}:
 0.910765   0.99055   0.910765   0.99055
 0.0524759  0.277297  0.0524759  0.277297
```

Looks good! We can re-use this code when we get to thinking about estimation later on. To do this we will have to solve the model for different values of x_m, but that can be done by using this code and iterating (potentially in parallel) over different values of x.

If you play around with parameters, you will see how convergence times may change and that solution methods are not always stable, especially when choice probabilities in equilibrium are very close to one or zero.

7 A Simple Labor Supply Model

7.1 Model Setup and Solution

Consider a dynamic labor supply model (with no uncertainty) where each agent n chooses a sequence of consumption and hours, $\{c_t, h_t\}_{t=1}^{\infty}$, to solve:

$$\max \sum_{t=0}^{\infty} \beta^t \left(\frac{c_t^{1-\sigma}}{1-\sigma} - \frac{\alpha_n^{-1}}{1+1/\psi} h_t^{1+1/\psi} \right)$$

subject to the intertemporal budget constraint:

$$\sum_t q_t c_t \leq A_{n,0} + \sum_t q_t W_{n,t} h_t, \qquad q_t = (1+r)^{-t}.$$

Let $H_{n,t}$ and $C_{n,t}$ be the realizations of labor supply for agent n at time t. Labor supply in this model obeys:

$$H_{n,t}^{1/\psi} = (\alpha_n W_{n,t}) C_{n,t}^{-\sigma}.$$

To simplify below, assume that $\beta = (1+r)^{-1}$, so that the optimal solution features perfectly smoothed consumption, C_n^*. Making appropriate substitutions gives C_n^* as the solution to:

$$\left(\sum_t q_t \right) C_n^* = \sum_t \left(q_t W_{n,t}^{1+\psi} \right) \alpha_n^{\psi} (C_n^*)^{-\psi\sigma} + A_{n,0}.$$

7.2 Code to solve the model

There is only one object to solve here which is consumption given a sequence of net wages. If one were to assume also constant wages the function below solves optimal consumption.

```
using Optim
function solve_consumption(r, ,W,A, , )
    Q = 1/ (1 - 1/(1+r))
    f(c) = (Q * c - Q * W^(1 +  ) *  ^  * c^(- * ) - A)^2
    r = Optim.optimize(f,0.,A+W)
    return r.minimizer
end
```

```
solve_consumption (generic function with 1 method)
```

7.3 Code to simulate a cross-section

Here we'll assume that wages, tastes for work, and assets co-vary systematically. For simplicity we'll use a multivariate log-normal distribution.

Below is code to simulate a cross-section of 1,000 observations.

```
using Distributions
function simulate_data( , ,r,N)
    ch = [0.3 0. 0.; 0.5 0.5 0.; 0.4 0.8 1.8]
    Σ = ch * ch'
    X = rand(MvNormal(Σ),N)
      = exp.(X[1,:])
    W = exp.(X[2,:])
    A = exp.(X[3,:])
    C = [solve_consumption(r, [i],W[i],A[i], , ) for i in eachindex(A)]
    @views H = exp.( X[1,:] .+   .* X[2,:] .-   *   .* log.(C) )
    return (; ,W,A,C,H)
end

# assume risk-aversion of 2 and frisch of 0.5
  = 2.
  = 0.5
r = 0.05

dat = simulate_data( , ,r,1_000)
```

```
( = [1.1895559672807827, 1.0057190238756233, 0.8774062392621625, 1.5610598524240673, 1.48085
```

8 Summary

In summary, this book has no content whatsoever.

References

Heckman, James J, and Bo E Honore. 1990. "The Empirical Content of the Roy Model." *Econometrica: Journal of the Econometric Society*, 1121–49.

Heckman, James, and Edward Vytlacil. 2005. "Structural equations, treatment effects, and econometric policy evaluation." *Econometrica* 73 (3): 669–738.

Roy, A. D. 1951. "Some Thoughts on the Distribution of Earnings." *Oxford Economic Papers* 3 (2): 135–46. http://www.jstor.org/stable/2662082.