Lecture 1: Introduction and Linear Equations

Yuya Takahashi

January 31, 2018

1 Introduction

1.1 Course Information

- The course covers a set of numerical methods that are used to compute and estimate economic models. We mainly study dynamic models and their applications in IO and labor economics, including dynamic discrete choice, dynamic games, two-step methods (CCP based), and general equilibrium models. We also cover several technical tools, such as methods for solving linear/nonlinear equations, numerical integration, approximation, and optimization.

There is no required textbook for this course. During the lectures, I will mainly use expositions and examples from the following three textbooks:

1. Judd, K. (1998). *Numerical Methods in Economics*. The MIT Press.

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

3. Heer, B. and Maussner, A. (2009). *Dynamic General Equilibrium Modeling*. Springer, 2nd edition.

You do not need to buy any of these textbooks. I will distribute class slides every week. Additional readings for each topic are listed below.

- **Material.** Each week, class notes are posted by Tuesday noon on the course website at https://sites.google.com/site/yuyasweb/teaching/numerical. Homework assignments and notifications are also available there.

2 Why are Numerical Methods Important?

- Computation significantly broadens a class of models that we can use to analyze economic phenomena and answer policy questions.

- Analytical procedure:

    1. Impose a set of assumptions: environment, objective functions, equilibrium concept, etc

    2. Derive optimality conditions, equilibrium conditions, etc

    3. Obtain a solution, preferably a closed form solution as a function of primitives, exogenous variables, etc

    4. Comparative statics: $\frac{\partial Y}{\partial \alpha}$ where Y is an equilibrium outcome and α is some parameter

- We will know what would happen to Y if α increases

- While elegant and often involves no ambiguity, a class of models where we can perform steps 3 and 4 is limited. To do so, economists have to impose a lot of simplifying assumptions.

- Numerical methods enable us to solve much richer models than we can analytically solve.

- "Theoretical analysis does not necessarily involve theorems" (Judd 1997, JDEC).

- Many elaborate policy simulations become possible.

- To perform counter-factual analyses, we need to estimate structural parameters.

  - A theoretical model is a mapping from model's primitive to outcomes
  - We need to get an inverse mapping, which has no closed form and complicated most of the time
  - Need to numerically solve

- Why do we need this class?
  - In many cases, solutions are at most "approximate"
  - We want to know how to best (better) approximate solutions
  - Even if we can obtain a very precise solution, it may take time
  - There are cases where you need to solve the model millions of times (e.g., NFP algorithm)
  - We want to know efficient algorithms

3 Example: Static Entry Models

3.1 Homogeneous firm model

- Suppose you have a cross-section of markets

- Bresnahan and Reiss (1990,1991)

- Market Size ('Demand'): S

- The number of firms active: N

- Profits: $\Pi(N, S)$:

  - Declining in N
  - Increasing in S
  - Note: Firms are symmetric

- Component of profits: ϵ_t (Interpretation?) drawn from distribution Φ.

- ϵ_t observed by potential entrants, not observed by econometrician

- Static complete information game

- Observe (S, N) for many independent markets

- Inference on $\Pi(N, S)$

- Parameterization:

$$\Pi(N, S) = S \cdot (\alpha - \gamma N) - F$$

- Pure Strategy Nash equilibrium: N firms enter if

$$\Pi(N, S) + \epsilon_t > 0 : and : \Pi(N + 1, S) + \epsilon_t \leq 0$$

- Likelihood: Probability that one observes N firms in a market with Demand S :

$$\ell(N, S) = \Phi(\Pi(N, S)) - \Phi(\Pi(N + 1, S))$$

- Using (S_m, N_m) for $m = 1, ..., M$, we can compute $\Pi_{m=1}^{M} \ell(N, S)$ for any set of (α, γ, F).

- We find (α, γ, F) that maximizes the likelihood.

- Even for this simple model, we need numerical methods to
  - compute profits and CDFs
  - search for the maximizer of the likelihood over the parameter space (nonlinear optimization)

- Once we recover α, γ, and F, we can calculate 'demand entry threshold' S_N^* such that

$$\Pi(N, S_N^*) = S_N^* \cdot (\alpha - \gamma N) - F = 0$$

$\Leftrightarrow$

$$S_N^* = \frac{F}{\alpha - \gamma N}$$

for all N.

- Calculate $s_N^* = S_N^*/N$.

- What does s^*_{N+1}/s^*_N tell us about the nature of competition?

- Bresnahan and Reiss (1991) estimate a similar model using isolated local markets of doctors, dentists, druggists, plumbers, tire dealers, and etc

- Use town population for S

- Result: per-firm entry threshold ratios

	s_2/s_1	s_3/s_2	s_4/s_3	s_5/s_4
Doctors	1.98	1.10	1.00	0.95
Dentists	1.78	0.79	0.97	0.94
Druggists	1.99	1.58	1.14	0.98
Plumbers	1.06	1.00	1.02	0.96
Tire dealers	1.81	1.28	1.04	1.03

3.2 Heterogeneous firm model

- Suppose you have a cross-section of markets

- Consider the same profit as above; $\Pi(N, S) = S \cdot (\alpha - \gamma N) - F$

- Assume that payoff shocks are firm specific ϵ_{it} and identically and independently drawn from distribution Φ

- $\{\epsilon_{it}\}_i^N$ observed by all potential entrants, not observed by econometrician

- Firm i's profit is

$$\begin{cases} \Pi(N, S) - \epsilon_{it} & \text{if enter} \\ 0 & \text{if not enter} \end{cases}$$

- N firms simultaneously decide whether to enter the market or not

- Example $N = 2$. Let $y_i = 1$ if firm i enters and $y_i = 0$ otherwise

- Pure Strategy Nash equilibrium: $(y_1, y_2) = (1, 1)$ is Nash if

$$S \cdot (\alpha - 2\gamma) - F - \epsilon_{1t} > 0 : and : S \cdot (\alpha - 2\gamma) - F - \epsilon_{2t} > 0$$

$(y_1, y_2) = (0, 0)$ is Nash if

$$S \cdot (\alpha - \gamma) - F - \epsilon_{1t} < 0 : and : S \cdot (\alpha - \gamma) - F - \epsilon_{2t} < 0$$

$(y_1, y_2) = (1, 0)$ is Nash if

$$S \cdot (\alpha - \gamma) - F - \epsilon_{1t} > 0 : and : S \cdot (\alpha - 2\gamma) - F - \epsilon_{2t} < 0$$

$(y_1, y_2) = (0, 1)$ is Nash if

$$S \cdot (\alpha - 2\gamma) - F - \epsilon_{1t} < 0 : and : S \cdot (\alpha - \gamma) - F - \epsilon_{2t} > 0$$

- Thus, if $(\epsilon_{1t}, \epsilon_{2t})$ are such that

$$S \cdot (\alpha - 2\gamma) - F < \epsilon_{1t} < S \cdot (\alpha - \gamma) - F$$
$$S \cdot (\alpha - 2\gamma) - F < \epsilon_{2t} < S \cdot (\alpha - \gamma) - F,$$

both $(y_1, y_2) = (1, 0)$ and $(y_1, y_2) = (0, 1)$ are Nash (multiple equilibria)

- What is the likelihood that we observe the outcome $(y_1, y_2) = (1, 0)$?

- We cannot write the likelihood

- However, in terms of the number of entrants, the outcome is unique for any realization of $(\epsilon_{1t}, \epsilon_{2t})$

- Likelihood: Probability that one observes N firms in a market with Demand S :

$$\ell(N = 0, S) = (1 - \Phi(\Pi(1, S)))^2$$
$$\ell(N = 2, S) = \Phi(\Pi(2, S))^2$$
$$\ell(N = 1, S) = 1 - \ell(N = 0, S) - \ell(N = 2, S)$$

- There are several variants:

  - Berry (1992): heterogeneity in profitability between firms

  - Mazzeo (2002): different modes of entry

4 Computer Arithmetic

- Several things that we have to know

- Most of real numbers in computer are not exact.
  - Rational numbers are stored in the form $\pm m2^{\pm n}$ where m and n are integers.
  - For example, -3210.48 is stored as $-7059920181484585 \times 2^{-41}$

- Several useful concepts.
  - Machine epsilon: the smallest ε such that the machine knows that $1 + \varepsilon > 1 > 1 - \varepsilon$
  - Single precision: 6-8 decimal digits of accuracy
  - Double precision: 12-16 decimal digits of accuracy

- In general, addition is faster than multiplication, which is faster than division. Exponentiation is a lot slower than multiplication

- We want to exploit this knowledge and reduce computation costs
  - For example, consider evaluating $a_0 + a_1x + a_2x^2 + a_3x^3$
  - Direct method: three additions, three multiplications, and two exponentiations
  - Horner's method: $a_0 + a_1x + a_2x^2 + a_3x^3 = a_0 + x(a_1 + x\,(a_2 + xa_3)$
  - Only three additions and three multiplications

- To speed up your code, identify which part of your code is most time consuming

- Computer mathematics is usually approximate, so errors almost always exist

- Sources of error:

    1. rounding error: a computer must round off the results. Rounding replaces a number with its nearest machine representable number

    2. mathematical truncation: $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$ we have to stop at some point

- Many cases involve both kinds of errors.

    - To compute $e^{\frac{1}{3}}$, we have to replace $\frac{1}{3}$ with the nearest machine representable number and then we must use an algorithm to approximate the exponentiation.

5 Linear Equations

- We often encounter a system of linear equations in economics.

5.1 Examples

5.1.1 Linear demand model

- Assume that there are 6 products and each of them is produced by a single firm. Marginal costs are constant. Demand is given by

$$
\begin{aligned}
q_1 &= a_1 + b_{11}p_1 + b_{12}p_2 + b_{13}p_3 + b_{14}p_4 + b_{15}p_5 + b_{16}p_6 \\
q_2 &= a_2 + b_{21}p_1 + b_{22}p_2 + b_{23}p_3 + b_{24}p_4 + b_{25}p_5 + b_{26}p_6 \\
q_3 &= a_3 + b_{31}p_1 + b_{32}p_2 + b_{33}p_3 + b_{34}p_4 + b_{35}p_5 + b_{36}p_6 \\
q_4 &= a_4 + b_{41}p_1 + b_{42}p_2 + b_{43}p_3 + b_{44}p_4 + b_{45}p_5 + b_{46}p_6 \\
q_5 &= a_5 + b_{51}p_1 + b_{52}p_2 + b_{53}p_3 + b_{54}p_4 + b_{55}p_5 + b_{56}p_6 \\
q_6 &= a_6 + b_{61}p_1 + b_{62}p_2 + b_{63}p_3 + b_{64}p_4 + b_{65}p_5 + b_{66}p_6
\end{aligned}
$$

where we assume $b_{jk} < 0$ when $j = k$ and $b_{jk} > 0$ when $j \neq k$. That is, own-price coefficients are negative and cross-price coefficients are all positive. Assuming Bertrand competition, the set of FOCs is

$$q_j + \left(p_j - c_j\right) \frac{\partial q_j}{\partial p_j} = 0$$

$\Leftrightarrow$

$$q_j + \left(p_j - c_j\right) b_{jj} = 0.$$

Now we can write the system of equations in matrix form

$$q + (\Delta \cdot B)(p - c) = 0 \tag{1}$$

where Δ is the identity matrix and " $\cdot$ " represents element-by-element multiplication.

Plugging q into the system and solving it for p give

$$p = \left(\Delta \cdot B + B'\right)^{-1} (\Delta \cdot Bc - a). \tag{2}$$

5.1.2 Dynamic optimization

- An infinite horizon, discrete time, finite state, Markov decision problem.

- State $s \in \mathbf{S}$ and per-period profit $\pi(s)$.

- The value function

$$V(s) = \pi(s) + \beta \sum_{s' \in \mathbf{S}} f\left(s, s'\right) V\left(s'\right),$$

  where f is the transition matrix. (To be able to write the problem this way, several details should be specified. Let's forget about the details for now)

- In matrix notation

$$V = \Pi + \beta f V$$

  $\Leftrightarrow$

$$V = (I - \beta f)^{-1} \Pi.$$

5.2 L-U Factorization

- Consider a system of linear equations

$$Ax = b$$

  where b is an n-dimensional vector and A is an n-by-n matrix.

- Non-singularity of A is assumed.

- Then, the analytical solution is $x = A^{-1}b$.

  - In MATLAB, $x = inv(A)$*b.

  - In many cases, this is a bad idea.

  - Calculating an inverse is costly. In addition, storing A^{-1} consumes a large amount of memory.

- Let's consider several alternatives.

- First assume A is lower triangular:

$$A = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

c.f. Upper triangular matrices have all nonzero entries on or above the diagonal.

- In this case, it is very easy to find x :

$$x_1 = \frac{b_1}{a_{11}}$$

$$x_k = \frac{b_k - \sum_{j=1}^{k-1} a_{kj} x_j}{a_{kk}}, \quad k = 2, 3, ..., n.$$

- This is called *back-substitution*, which is well-defined for nonsingular, lower triangular matrices.

- An analogy applies for an upper triangular matrix.

- For A nonsingular, it is known what we get

$$A = LU$$

where L is lower-triangular and U is upper-triangular matrices.

- This way, the original problem is equivalent to finding y in

$$Ly = b \tag{3}$$

and then finding x in

$$Ux = y. \tag{4}$$

- Note that both for (3) and (4), we can apply back-substitution!

- Then, the question is, of course, how do we find such L and U?

- Let's work with an example: $Ax = b$ where

$$A = \begin{bmatrix} 2 & 0 & -1 & 2 \\ 4 & 2 & -1 & 4 \\ 2 & -2 & -2 & 3 \\ -2 & 2 & 7 & -3 \end{bmatrix}.$$

- Consider the initial state

$$L^0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } U^0 = \begin{bmatrix} 2 & 0 & -1 & 2 \\ 4 & 2 & -1 & 4 \\ 2 & -2 & -2 & 3 \\ -2 & 2 & 7 & -3 \end{bmatrix}$$

and the following sequence:

$$L^1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } U^1 = \begin{bmatrix} 2 & 0 & -1 & 2 \\ 0 & 2 & 1 & 0 \\ 2 & -2 & -2 & 3 \\ -2 & 2 & 7 & -3 \end{bmatrix}$$

$$L^2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } U^2 = \begin{bmatrix} 2 & 0 & -1 & 2 \\ 0 & 2 & 1 & 0 \\ 0 & -2 & -1 & 1 \\ -2 & 2 & 7 & -3 \end{bmatrix}$$

$$L^3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix} \text{ and } U^3 = \begin{bmatrix} 2 & 0 & -1 & 2 \\ 0 & 2 & 1 & 0 \\ 0 & -2 & -1 & 1 \\ 0 & 2 & 6 & -1 \end{bmatrix}$$

$$L^4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 1 & -1 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix} \text{ and } U^4 = \begin{bmatrix} 2 & 0 & -1 & 2 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 2 & 6 & -1 \end{bmatrix}$$

$$L^5 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 1 & -1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{bmatrix} \text{ and } U^5 = \begin{bmatrix} 2 & 0 & -1 & 2 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 5 & -1 \end{bmatrix}$$

$$L^6 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 1 & -1 & 0 & 1 \\ -1 & 1 & 1 & 0 \end{bmatrix} \text{ and } U^6 = \begin{bmatrix} 2 & 0 & -1 & 2 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 5 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Now we can apply back-substitution in the form of (3) and (4). This is called *Gaussian elimination* (decomposition).

- In MATLAB, the solution to $Ax = b$ is given by the statement $x = A\backslash b$.

- The L-U factorization is efficient.

- For large n, L-U factorization requires approximately $n^3/3 + n^2$ operations (multiplications and divisions). A naive calculation of $x = A^{-1}b$ requires approximately $n^3 + n^2$ operations.

- Exercise. Find x in the following system:

$$Ax = b$$

where

$$A = \begin{bmatrix} 2 & 3 & -1 & 2 \\ 4 & 3 & -2 & 4 \\ -2 & -2 & 5 & 4 \\ 1 & 2 & 6 & 7 \end{bmatrix}, \; b = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

5.2.1 Cholesky Factorization

- If A is a symmetric positive definite matrix, we have

$$A = LL^\top.$$

- This is a special case of standard the L-U factorization.

  We know that $l_{11} = \sqrt{a_{11}}$,

$$l_{i1} = \frac{a_{i1}}{l_{11}},\ i = 2, ..., n.$$

  And $l_{22} = \sqrt{a_{22} - l_{21}^2}$ and

$$l_{i2} = \frac{a_{i2} - l_{i1} l_{21}}{l_{22}},\ i = 3, ..., n.$$

  We repeat this process. That $\sqrt{a_{11}}$ and $\sqrt{a_{22} - l_{21}^2}$ are real numbers is guaranteed by the positive definiteness of A.

- The cost of Cholesky decomposition is about half the cost of the Gaussian decomposition.

5.2.2 Sparse Matrix

- Often we encounter cases in which A is sparse.

- When A is sparse, the Gaussian elimination algorithm involves a lot of multiplications and additions with zero, which is meaningless but costly.

- We can possibly reduce the computation costs significantly.

- When A is diagonal matrix, the solution is trivial. However, there are many different types of sparseness. The efficient coding depends on a particular application.

  - MATLAB has command $S = sparse(A)$ that creates a version S of the matrix A stored in a sparse matrix format.

5.3 Iterative Methods

5.3.1 Gauss Jacobi

- Methods based on Gaussian elimination are exact, but costly especially for solving large linear equations.

- Iterative methods are powerful in such a case, although they do not yield exact solutions.

- Gauss-Jacob algorithm works as follows. The first equation is

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_x = b_1$$

Rearranging gives

$$x_1 = a_{11}^{-1}\left(b_1 - a_{12}x_2 - \cdots - a_{1n}x_x\right)$$

for $a_{11} \neq 0$. In general

$$x_i = \frac{1}{a_{ii}} \left\{ b_i - \sum_{j \neq i} a_{ij} x_j \right\}.$$

- The algorithm iterate x following

$$x_i^{k+1} = \frac{1}{a_{ii}} \left\{ b_i - \sum_{j \neq i} a_{ij} x_j^k \right\}, \quad i = 1, ..., n. \tag{5}$$

5.3.2 Gauss Seidel

- In Gauss-Jacob algorithm, in each round of iterations, we update x_i for all i, keeping x_i from the previous round for all i fixed.

- The Gauss-Seidel algorithm proposes to use any new information whenever it becomes available.

- The update rule (5) is modified to

$$x_i^{k+1} = \frac{1}{a_{ii}} \left\{ b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right\}, \quad i = 1, ..., n.$$

- We use x_i^{k+1} to calculate x_j^{k+1} for $j > i$.

- The order of updating components matters.

5.3.3 More General Formulation

- Consider an easily invertible matrix Q. Add both sides of $Ax = b$.

$$Qx + Ax = Qx + b$$

rearranging gives

$$x = Q^{-1}b + \left(I - Q^{-1}A\right) x.$$

Then consider the following iteration:

$$x^{k+1} = Q^{-1}b + \left(I - Q^{-1}A\right)x^k.$$

- The Gauss-Jacobi sets Q equal to the diagonal matrix formed from the diagonal entries of A.

- The Gauss-Seidel sets Q equal to the upper triangular matrix formed from the upper triangular elements of A.

- The iteration is a contraction mapping if

$$||I - Q^{-1}A|| < 1$$

for any matrix norm.

5.4 Computing Ergodic Distributions

- Let $\mathbf{P}$ be an n-state Markov transition matrix.

- State space $\mathbf{S}$ and typical state $s \in \mathbf{S}$.

- A typical element $p(s, s')$ of $\mathbf{P}$ equals the probability that state s' is reached when the current state is given by s.

- State s' is reachable from s if there exists an integer T and a sequence of states $(s^1, ..., s^T)$ so that the chain $\mathbf{P}$ will be at state s' after T periods. If s' is reachable from s, and s is reachable from s', then the states s and s' are said to communicate.

- Suppose all states $s, s' \in \mathbf{S}$ of the chain $\mathbf{P}$ communicate with themselves. Then the unique steady state distribution $\mathbf{Q} = [Q(s)]_{s \in \mathbf{S}}$ exists ($\sum_{s \in \mathbf{S}} Q(s) = 1$). It satisfies $Q(s) > 0$ for all $s \in \mathbf{S}$ and

$$\mathbf{Q} = \mathbf{QP} \tag{6}$$

- Therefore, the iterative method is

$$\mathbf{Q} = \lim_{k \to \infty} \mathbf{Q}^0 \mathbf{P}^k$$

for any $\mathbf{Q}^0$ such that $\sum_{s \in \mathbf{S}} Q^0(s) = 1$.

- Alternatively, the direct method is to find the solution to the following system:

$$\mathbf{Q} \begin{pmatrix} p_{11} - 1 & p_{12} & \cdots & p_{1,n-1} & 1 \\ p_{21} & p_{22} - 1 & \cdots & p_{2,n-1} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ p_{n-1,1} & p_{n-1,2} & \cdots & p_{n-1,n-1} - 1 & 1 \\ p_{n,1} & p_{n,2} & \cdots & p_{n,n-1} & 1 \end{pmatrix} = (0, ..., 0, 1)$$

the altered last column forces $\sum_{s \in \mathbf{S}} Q(s) = 1$, while the other columns satisfy the condition (6).

- The direct method requires $n^3/2$ multiplications, while the iterative approach takes kn^2 operations.