Computational Tools for Macroeconomics using MATLAB

Week 4 – Matrix Algebra for Economics

Cristiano Cantore

Sapienza University of Rome

Learning Outcomes

By the end of this week, you will be able to:

1. Create and manipulate matrices in MATLAB.
2. Understand the difference between element-wise and matrix operations.
3. Solve systems of linear equations relevant to economics.
4. Compute matrix inverses, determinants, and eigenvalues.
5. Apply matrix algebra to a simple macroeconomic model (IS–LM).

Creating Vectors and Matrices - Recap

```
% Row and column vectors
r = [1 2 3 4];          % 1x4 row vector
c = [1; 2; 3; 4];       % 4x1 column vector

% Matrices (rows separated by semicolons)
A = [1 2; 3 4];         % 2x2 matrix
B = [5 6; 7 8];         % 2x2 matrix

% Common constructors
z = zeros(3,2);         % 3x2 of zeros
o = ones(2,3);          % 2x3 of ones
I = eye(3);             % 3x3 identity
D = diag([10 20 30]); % diagonal matrix
```

Colon Operator and `linspace` - Recap

```
% Colon operator: start:step:stop
x = 1:5;             % [1 2 3 4 5]
y = 0:0.5:2;         % [0 0.5 1.0 1.5 2.0]

% linspace(start, stop, nPoints)
g = linspace(0, 1, 6);   % 6 points from 0 to 1

% Quick grids (for later plotting/surfaces)
% [X,Y] = meshgrid(x, y); % not needed now, just a teaser
```

- Use `a:b:c` for integer-like sequences; `linspace` for exact endpoint control.

Indexing and Slicing - Recap

```
A = [10 20 30; 40 50 60; 70 80 90]; % 3x3

A(2,3)        % element at row 2, col 3 -> 60
A(1,:)        % entire row 1 -> [10 20 30]
A(:,2)        % entire column 2 -> [20; 50; 80]
A(1:2,2:3)    % submatrix -> [20 30; 50 60]

% Logical indexing (very useful)
mask = A > 50;
A(mask) = 999; % replaces all elements > 50 with 999
```

Concatenation and Reshape

```
% Horizontal and vertical concatenation
C = [1 2 3];      D = [4 5 6];
H = [C D];        % [1 2 3 4 5 6]  (1x6)
V = [C; D];       % 2x3 matrix (rows stacked)

% Reshape (fill by column, preserve #elements)
E = reshape(1:12, 3, 4); % 3x4 matrix: columns filled 1..12
E(:) = 0;                % set all entries to zero (linear index
```

- `reshape` changes shape, not order (fills by columns).
- Use `[]` to concatenate; dimensions must match.

Replication: `repmat` and Implicit Expansion

```
v = (1:3)';               % 3x1
M = repmat(v, 1, 4);   % 3x4 by repeating columns of v

% Modern MATLAB (R2016b+): implicit expansion (broadcasting)
w = 10:10:40;             % 1x4
S = v + w;                % 3x4: each row adds w automatically
```

- Prefer implicit expansion when available; it is cleaner and faster.

Basic Arithmetic

```
A = [1 2; 3 4];
B = [5 6; 7 8];

A + B           % element-wise addition
A - B           % element-wise subtraction
2*A             % scalar multiply
A/2             % scalar divide (same as A * 0.5)
A.^2            % element-wise power (square each entry)
```

- Scalars act on all entries.
- `.^` is element-wise power (vs matrix power `^`).

Element-wise vs Matrix Operations (Important!)

Element-wise

```
A .* B    % multiply each entry
A ./ B    % divide each entry
A .^ 2    % square each entry
```

Matrix (linear algebra)

```
A * B     % matrix product
A \ b     % solve Ax = b
A ^ 2     % A * A (square)
```

Common pitfall: Dimension mismatch.

```
% If sizes don't align for A * B, MATLAB errors:
% Error using  *
% Inner matrix dimensions must agree.
```

Useful Helpers

```
A = rand(3,4);          % random 3x4
size(A)                 % [3 4]
length(A)               % max(dim) -> 4
numel(A)                % # of elements -> 12
sum(A,1)                % column sums (1 keeps rows)
sum(A,2)                % row sums (2 keeps cols)
mean(A,1)               % column means
max(A,[],2)             % row-wise max
```

- Remember the dimension flag: 1=columns, 2=rows.

Micro-exercise

Given

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \quad B = \begin{bmatrix} 2 & 0 & -1 \\ 1 & 1 & 1 \end{bmatrix}.$$

1. Compute `A + B`, `A .* B`.
2. Extract the second column of `A` and call it `c2`.
3. Form the 2x2 matrix `C` by stacking `c2` and a vector of ones.
4. Try `A * B'` and explain why transpose is needed.

Solving Linear Systems

- Most of the time in Economics we solve (linear) systems of equations.
- Example: Represent IS–LM as a linear system: $Ax = b$.
- Use MATLAB tools:
  - `A\b` (preferred method).
  - `inv(A)*b` (less efficient, can be numerically unstable).
- Work through a 2×2 system demo.

IS–LM: Equations, Calibration, and Matrix Form

Step 1: Model equations

$$\text{IS:} \quad Y = \bar{A} - \alpha\, i \qquad (\alpha > 0)$$
$$\text{LM:} \quad i = \beta\, Y - \frac{M}{P} \qquad (\beta > 0)$$

Economic meaning: IS has negative slope ($\partial Y/\partial i < 0$); LM has positive slope ($\partial i/\partial Y > 0$).

IS–LM: Equations, Calibration, and Matrix Form

Step 1: Model equations

$$\begin{aligned} \text{IS:} \quad & Y = \bar{A} - \alpha\, i \qquad && (\alpha > 0) \\ \text{LM:} \quad & i = \beta\, Y - \frac{M}{P} \qquad && (\beta > 0) \end{aligned}$$

Economic meaning: IS has negative slope ($\partial Y/\partial i < 0$); LM has positive slope ($\partial i/\partial Y > 0$).

Step 2: Calibrate parameters (example)

$$\bar{A} = 100, \quad \alpha = 0.5, \quad \beta = 0.2, \quad \frac{M}{P} = 50.$$

Economic Motivation: IS–LM as a System

Step 3: Move terms to the left-hand side

$$\underbrace{Y + \alpha i}_{\text{IS in LHS}} = \bar{A} \;\Rightarrow\; [1 \;\; \alpha]\begin{bmatrix} Y \\ i \end{bmatrix} = \bar{A}$$

$$\underbrace{-\beta Y + i}_{\text{LM in LHS}} = -\frac{M}{P} \;\Rightarrow\; [-\beta \;\; 1]\begin{bmatrix} Y \\ i \end{bmatrix} = -\frac{M}{P}$$

Economic Motivation: IS–LM as a System

Step 3: Move terms to the left-hand side

$$\underbrace{Y + \alpha i}_{\text{IS in LHS}} = \bar{A} \;\Rightarrow\; [1 \;\; \alpha]\begin{bmatrix} Y \\ i \end{bmatrix} = \bar{A}$$

$$\underbrace{-\beta Y + i}_{\text{LM in LHS}} = -\frac{M}{P} \;\Rightarrow\; [-\beta \;\; 1]\begin{bmatrix} Y \\ i \end{bmatrix} = -\frac{M}{P}$$

Step 4: Stack the two equations: $Ax = b$

$$\begin{bmatrix} 1 & \alpha \\ -\beta & 1 \end{bmatrix}\begin{bmatrix} Y \\ i \end{bmatrix} = \begin{bmatrix} \bar{A} \\ -\frac{M}{P} \end{bmatrix} \;\Rightarrow\; \begin{bmatrix} 1 & 0.5 \\ -0.2 & 1 \end{bmatrix}\begin{bmatrix} Y \\ i \end{bmatrix} = \begin{bmatrix} 100 \\ -50 \end{bmatrix}.$$

MATLAB Tools for Linear Systems

- Preferred: `A\b` (backslash operator).
  - Efficient, stable.
  - Adapts method to the system (Gaussian elimination, LU, etc.).
- Alternative: `inv(A)*b`.
  - Works but slower, less accurate.
  - Avoid in serious computations.

Demo: 2x2 System in MATLAB

```
% Define matrix and vector
A = [1 0.5; -0.2 1];
b = [100; -50];

% Preferred solution
x = A\b;      % returns [Y; i]

% Alternative (less efficient)
x_inv = inv(A)*b;

disp(x)       % show solution
disp(x_inv)

% Compare the difference
diff_norm = norm(x - x_inv);
fprintf('Difference between methods: %.3e\n', diff_norm);

% Check residuals
r = norm(A*x - b);
r_inv = norm(A*x_inv - b);
```

Micro-exercise

Solve the following system by hand and then check in MATLAB:

$$\begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 8 \\ 13 \end{bmatrix}$$

1. Write the system as $Ax = b$.
2. Use MATLAB's backslash operator to solve.
3. Verify with `inv(A)*b`.
4. Try with a 3x3 system of your choice.

Matrix Properties & Stability

- Compute key properties: determinant, inverse, rank, eigenvalues/eigenvectors.
- Link properties to economics (existence/uniqueness, stability).
- Recognize numerical issues (singularity, ill-conditioning).
- Apply eigenvalue analysis to a simple dynamic system.

Determinant, Invertibility, Rank

- **Determinant** ($\det(A)$): measures volume scaling; $\det(A) = 0 \Rightarrow$ singular.
- **Inverse** (A^{-1}): exists iff $\det(A) \neq 0$; unique solution $Ax = b$ iff A invertible.
- **Rank** ($\mathrm{rank}(A)$): # of linearly independent columns/rows.

Key facts (square $A \in \mathbb{R}^{n\times n}$):

$$\det(A) \neq 0 \iff \mathrm{rank}(A) = n \iff A^{-1} \text{ exists.}$$

Computing Properties in MATLAB

```
A = [1 2; 3 4];

detA = det(A)        % determinant
rA   = rank(A)       % rank
Ainv = inv(A)        % inverse (compute only if needed)
```

- **Remember:** Prefer `x = A\b` to solve $Ax = b$; avoid forming `inv(A)`.

Eigenvalues & Eigenvectors (Why we care)

- $Av = \lambda v$ with $v \neq 0$.
- **Static models:** $\det(A) = \prod_i \lambda_i$, $\mathrm{tr}(A) = \sum_i \lambda_i$.
- **Dynamics:** For state transition $x_{t+1} = Tx_t$, stability if all $|\lambda_i(T)| < 1$.
- **Symmetric** A: real eigenvalues; orthogonal eigenvectors (spectral theorem).

Eigenvalues in MATLAB

```
A = [0.8 0.1; 0.2 0.9];
lambda = eig(A)          % eigenvalues
[V, D] = eig(A)
% columns of V = eigenvectors, D diagonal of eigenvalues

% % Check spectral decomposition: A \approx V*D/V
% A_recon = V * D / V;    % should match A up to numerical error
%
```

Positive (Semi)Definiteness (PSD/PD)

- A symmetric PD $\Rightarrow x'Ax > 0$ for $x \neq 0$; all eigenvalues > 0.
- Common in quadratic costs, covariance matrices, least squares normal equations.

Checks: eigenvalues > 0 (PD) / ≥ 0 (PSD).

Numerical Notes: Singular vs Ill-Conditioned

```
% Bad practice:
x1 = inv(A)*b;       % forms inverse explicitly

% Good practice:
x2 = A\b;            % uses stable factorization internally

% Quick diagnostics
s = svd(A);          % singular values
kappa = cond(A);
% condition number; watch out if kappa is large (e.g., 1e10)
```

- **Singular** ($\det(A) = 0$): no unique solution.
- **Ill-conditioned**: solution exists but is numerically fragile.

Example: Stability Analysis of a 2D System

- Consider $x_{t+1} = Tx_t$ with
$$T = \begin{bmatrix} 0.85 & 0.10 \\ 0.15 & 0.90 \end{bmatrix}.$$
- Stable if all eigenvalues satisfy $|\lambda| < 1$.

```
T = [0.85 0.10; 0.15 0.90];
eigT = eig(T)       % check magnitudes
```

Micro-exercise

Let

$$A = \begin{bmatrix} 4 & 2 \\ 1 & 3 \end{bmatrix}, \quad T = \begin{bmatrix} 0.7 & 0.5 \\ 0.1 & 0.8 \end{bmatrix}.$$

1. Compute $\det(A)$, $\mathrm{rank}(A)$; is A invertible?
2. Find eigenvalues of A; verify $\det(A) = \prod \lambda_i$, $\mathrm{tr}(A) = \sum \lambda_i$.
3. Compute eigenvalues of T; is the dynamic $x_{t+1} = Tx_t$ stable?

Cholesky Factorization: Motivation

- Many macro models involve correlated shocks with a covariance matrix Σ.
- To simulate or estimate such systems, we need to generate random vectors ε satisfying:

$$\mathrm{Cov}(\varepsilon) = \Sigma.$$

- The **Cholesky factorization** provides a simple way to do this.

Definition and Properties

Cholesky decomposition:

$$\Sigma = LL', \quad L \text{ lower triangular.}$$

- Exists if and only if Σ is **symmetric positive definite**.
- Unique when diagonal entries of L are positive.
- Useful for:
  - Simulating correlated random shocks.
  - Solving systems $Ax = b$ efficiently when A is SPD.

Numerical Example in MATLAB

Given

$$\Sigma = \begin{bmatrix} 1.5 & -0.9 \\ -0.9 & 2 \end{bmatrix},$$

compute its Cholesky factor L:

```
Sigma = [1.5 -0.9; -0.9 2];
L = chol(Sigma, 'lower');
L * L'
```

Numerical Example in MATLAB

Given

$$\Sigma = \begin{bmatrix} 1.5 & -0.9 \\ -0.9 & 2 \end{bmatrix},$$

compute its Cholesky factor L:

```
Sigma = [1.5 -0.9; -0.9 2];
L = chol(Sigma, 'lower');
L * L'
```

$$L = \begin{bmatrix} 1.2247 & 0 \\ -0.7348 & 1.2083 \end{bmatrix}, \quad LL' \approx \Sigma.$$

Interpretation

- Suppose $u \sim N(0, I)$ (independent standard normals).
- Then $\varepsilon = Lu$ has:

$$\mathrm{Cov}(\varepsilon) = L\,\mathrm{Cov}(u)\,L' = LL' = \Sigma.$$

- In macro models:
  - u are independent structural shocks.
  - L introduces realistic correlations.

Homework Overview

Goal:

- Extend the IS–LM model using matrix algebra.
- Apply MATLAB tools to solve and analyze equilibrium.
- Introduce uncertainty using Cholesky factorization.

Part 1 – Solving the Extended IS–LM System

- Model equations:

$$
\begin{aligned}
Y &= \bar{A} + b_G G - \alpha i, \\
-\frac{M}{P} &= \beta Y - \gamma i - \theta G.
\end{aligned}
$$

- Write in matrix form: $Ax = b$ with $x = [Y, i]'$.
- Solve using:
  - `x = A_` (preferred)
  - `x_inv = inv(A)*b` (for comparison)
- Perform comparative statics varying G.

MATLAB Setup for Part 1: Calibration

- $\bar{A}$=120;
- b_G=0.8;
- α=0.5;
- β=0.25;
- γ=1.2;
- θ=0.1;
- G=100;
- MP=80;
- Compute equilibrium (Y, i).
- Loop over a grid of G to plot $Y(G)$ and $i(G)$.

Part 2 – Cholesky Decomposition of Covariance Matrix

$$\Sigma = \begin{bmatrix} 4 & 1.2 \\ 1.2 & 3 \end{bmatrix}.$$

- Check that all eigenvalues of Σ are positive.
- Compute $L = \texttt{chol(Sigma,'lower')}$.
- Interpret what L means for correlated IS–LM shocks.

Part 3 – IS–LM with Correlated Shocks (Extension)

Now suppose the economy experiences stochastic shocks to the IS and LM equations:

$$Ax = b + \varepsilon,$$

where

$$\varepsilon = Lu, \qquad u \sim N(0, I),$$

and L is the Cholesky factor of the covariance matrix Σ.

Then:

$$x = A^{-1}(b + \varepsilon) = A^{-1}b + A^{-1}Lu.$$

So $A^{-1}L$ maps structural shocks to equilibrium fluctuations in Y and i.

Interpretation of the Stochastic IS–LM System

Economic interpretation:

- $A^{-1}b$: deterministic equilibrium (the one you solved in part 1).
- $A^{-1}Lu$: random deviations from equilibrium, shaped by both:
  - the **economic structure** (A), and
  - the **correlation of shocks** (Σ) introduced via the Cholesky factor L.
- The matrix $A^{-1}L$ acts as a **transmission mechanism** from structural shocks to observable fluctuations in (Y, i).

Part 3 – IS–LM with Correlated Shocks (Extension)

- Introduce stochastic shocks:

$$Ax = b + \varepsilon, \qquad \varepsilon = Lu, \;\; u \sim N(0, I).$$

- Compute simulated equilibria (simulating 10000 shocks):

$$x_t = A^{-1}(b + Lu_t).$$

- Analyze the empirical variance and covariance of (Y, i).
- Visualize (Y, i) scatter plot to show correlation.
- **Important:** To ensure reproducibility, set the random seed to 123.

Deliverables

Submit one MATLAB file named `week4_homework_solution.m` containing:

1. Parameter definitions and matrix setup.
2. Equilibrium solution and comparative statics for G.
3. Cholesky decomposition of Σ.
4. Simulation of correlated shocks and their effects.

Deadline: Next lab session.