MATLAB Fundamentals for Macroeconomic Modeling

1 Introduction

This document introduces the essential MATLAB concepts needed to implement dynamic programming solutions for macroeconomic models, with a focus on practical skills required for value function iteration, policy function computation, and simulation of economic models.

The matlab file "matlab_intro.m" contains all the code in this document. You should open this file and run each piece of code as you read through this document. This will help you lean the syntax and get you familiar with MATLAB.

2 Variables and Basic Operations

2.1 Variable Assignment

MATLAB uses simple assignment with the `=` operator. Unlike some languages, you don't need to declare variable types.

```matlab
clear;
% Parameters
beta = 0.96;         % Discount factor
sigma = 2;           % Risk aversion
N = 200;             % Number of grid points
```

Key points:

- `clear` removes all variables stored in workspace
- Use `%` for comments
- Variables are created on first assignment
- Semicolons suppress output; omit them to see results
- Variable names are case-sensitive

3 Arrays and Vectors

3.1 Creating Vectors

Vectors are the fundamental data structure in MATLAB.

```matlab
% Column vector of zeros
V = zeros(100, 1);

% Row vector (note the commas)
prices = [1.5, 2.3, 4.7];

% Column vector (note the semicolons)
quantities = [10; 20; 30];

% Transpose operator
row_to_col = prices';  % Now a column vector
```

3.2 Creating Grids

The `linspace` function creates evenly spaced points, crucial for discretizing state spaces.

```matlab
% Create a grid from 0 to 10 with 100 points
k_grid = linspace(0, 10, 100);

% Alternatively, using colon operator
x = 0:0.1:10;  % From 0 to 10 in steps of 0.1
```

3.3 Array Indexing

```matlab
% Create a vector
v = [10, 20, 30, 40, 50];

% Access single element (MATLAB uses 1-based indexing)
first = v(1);       % Returns 10
third = v(3);       % Returns 30

% Access multiple elements
subset = v(2:4);    % Returns [20, 30, 40]

% Access from beginning to specific index
start = v(1:3);     % Returns [10, 20, 30]

% Access from specific index to end
finish = v(3:end); % Returns [30, 40, 50]

% Last element
last = v(end);      % Returns 50
```

4 Element-wise vs Matrix Operations

This is one of the most important concepts in MATLAB.

4.1 Matrix Operations (Linear Algebra)

Standard operators perform linear algebra operations:

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

C = A * B;     % Matrix multiplication
D = A^2;       % Matrix power (A * A)
```

4.2 Element-wise Operations

For element-by-element operations, use the dot operator:

```
% Element-wise multiplication
x = [1, 2, 3];
y = [4, 5, 6];
z = x .* y;     % Returns [4, 10, 18]

% Element-wise division
w = x ./ y;     % Returns [0.25, 0.4, 0.5]

% Element-wise power
p = x .^ 2;     % Returns [1, 4, 9]
```

4.3 When to Use Each

```
% Production function: y = k^alpha
k_grid = linspace(1, 10, 100)';
alpha = 0.33;

% CORRECT: Element-wise power
y = k_grid .^ alpha;  % Applies to each element

% WRONG: Matrix power
% y = k_grid ^ alpha;  % This will cause an error.

% CRRA utility: u(c) = c^(1-sigma) / (1-sigma)
c = linspace(0.1, 5, 100)';
sigma = 2;

% CORRECT:
u = (c .^ (1-sigma)) / (1-sigma);
```

Rule of thumb: When working with vectors representing economic variables (grids, consumption paths, etc.), almost always use element-wise operations.

5 The max() Function

The max() function is central to value function iteration. There are two ways to use the max() function.

5.1 Finding the maximum of a vector

When using the max() function to find the maximum in a vector the function can return two values.

```
values = [2.3, 5.1, 3.7, 6.2, 1.8];

% Get only the maximum value
max_val = max(values);  % Returns 6.2

% Get both value and location
[max_val, max_idx] = max(values);  % max_val = 6.2, max_idx = 4
```

5.2 Finding the maximum between each element and a number

```
values = [-1, 3, 5, -2, 4];

% Ensure non-negative values
non_negatives = max(values, 0);  % Returns [0, 3, 5, 0, 4];
```

6 Essential Built-in Functions

6.1 Mathematical Functions

```
% Common functions
x = -2.5;
abs_x = abs(x);          % Absolute value: 2.5
sqrt_x = sqrt(4);        % Square root: 2
log_x = log(10);         % Natural log: 2.3026
exp_x = exp(1);          % Exponential: 2.7183
```

6.2 Statistical Functions

```
data = [1.2, 3.4, 2.1, 5.6, 4.3];

mean_val = mean(data);       % Average
std_val = std(data);         % Standard deviation
max_val = max(data);         % Maximum
min_val = min(data);         % Minimum

% Correlation
x = [1, 2, 3, 4, 5];
y = [2, 4, 5, 4, 5];
correlation = corr(x', y');  % Must be column vectors.
```

6.3 The `find()` Function

Locates indices where a condition is true:

```
x = [1, 2, 2, 2, 3, 4, 5];

% Find first occurrence
idx1 = find(x == 0, 2, "first"); % Returns idx = 2

% Find all occurrence
idx2 = find(x == 0, 2); % Returns idx = [2, 3, 4]

% Find all elements greater than 3
large_idx = find(x > 3); % Returns large_idx = [6, 7]
```

7 Anonymous Functions

Anonymous functions create function handles without separate files.

7.1 Basic Syntax

```
% Syntax: function_name = @(inputs) expression
f = @(x) x^2;               % Function that squares input
g = @(x, y) x + 2*y;        % Function with two inputs
```

7.2 Economic Applications

```
% CRRA utility function
u = @(c) (c.^(1-sigma)) / (1-sigma);

% Cobb-Douglas production
f = @(k) k.^alpha;

% Complete production function
F = @(k, h) k.^alpha .* h.^(1-alpha);

% Using the functions
consumption = 2.5;
utility_value = u(consumption);

capital = linspace(1, 10, 50)';
output = f(capital);  % Vectorized
```

Key point: Use element-wise operators inside anonymous functions when you plan to pass vectors.

8 Vectorization and Broadcasting

8.1 Broadcasting

MATLAB automatically expands scalars and vectors in arithmetic operations:

```matlab
% Scalar + vector
a = 5;
v = [1, 2, 3];
result = a + v;  % Returns [6, 7, 8]
```

8.2 Vectorized Operations

```matlab
% Instead of loop:
for i = 1:length(k_grid)
    y(i) = k_grid(i)^alpha;
end

% Use vectorization:
y = k_grid .^ alpha;  % Much faster

% Multiple operations
k = linspace(1, 10, 100);
alpha = 0.33;
delta = 0.1;

% All vectorized
output = k .^ alpha;
depreciation = delta * k;
net_output = output - depreciation;
```

9 Control Flow

9.1 For Loops

```matlab
% Basic syntax
for i = 1:10
    disp(i);  % Display the value
end

% Loop over array indices: Returns New = [2, 3, ..., 11]
N = length(k_grid);
New = zeros(1,N);
for i = 1:N
    New(i) = i+1;
end
```

9.2 While Loops

Used for iterating until convergence:

```
% Value function iteration pattern
iter = 0;
tol = 1e-6;
diff = 7;

while diff > tol
    iter = iter + 1 % without a semicolon, the value of iter displays for
       every iteration

    % update difference
    diff = diff - sqrt(iter);
end
```

9.3 If-Else Statements

```
% Basic if statement
x = 2;
if x > 0
    disp("Positive");
end

% If-else
y = 3;
if x >= y
    positive = x - y;
else
    positive = y - x;
end
```

10 Random Number Generation

10.1 Setting the Seed

For reproducible results:

```
rng(123);  % Set seed to 123
% Now random numbers will be the same each time
```

10.2 Generating Random Numbers

```
% Uniform random number in [0,1]
u = rand();

% Vector of random numbers
u_vec = rand(100, 1);
```

11 Plotting

11.1 Basic Plotting

```
% Simple line plot
x = linspace(0, 10, 100);
y = x.^2;
plot(x, y);
xlabel("x");
ylabel("y");
title("Quadratic Function");

% Multiple series
y2 = x.^2 + 2;
plot(x, y1, x, y2);
legend("Series 1", "Series 2");

% Formatting
plot(x, y, "LineWidth", 2);  % Thicker line
plot(x, y, "r--");           % Red dashed line

% Overlaying plots
plot(x, y1);
hold on;
plot(x, y2);
hold off;
```

12 Interpolation

We will have a function defined on a grid, that is, x values and corresponding y values, stored in vectors. Sometimes we will want to evaluate the function at an x-value that is not part of the x-grid (i.e. the values in the x vector). To do this we need to interpolate the function. This function fits a linear function between each pair of (x, y) values to evaluate off gird points.

```
% A function defined on grid
x_grid = [1, 2, 3, 4, 5, 6, 7];
y = x_grid.^2;

% x value not on grid
x_value = 3.4;

% Interpolate
y_value = interp1(x_grid, y, x_value);
```

In this example, we know the functional for, i.e. $y = x^2$, and so we could find `y_value` = `x_value`^{2}. However, in our application we will not know the function form, we will only have the x-grid vector and the corresponding y values, so we will need to interpolate.

13 Common Mistakes and Debugging

13.1 Common Errors

1. **Dimension mismatch**

```
% ERROR: Row vector + column vector
x = linspace(0, 10, 100);       % Row vector
y = zeros(100, 1);              % Column vector
z = x + y;  % Error

% FIX: Make dimensions match
x = linspace(0, 10, 100)';      % Now column
z = x + y;  % Works
```

2. **Forgetting element-wise operators**

```
% ERROR
k = linspace(1, 10, 100)';
y = k^0.33;  % Matrix power doesn't make sense here

% FIX
y = k.^0.33;  % Element-wise power
```

3. **Using 0-based indexing**

```
% ERROR (MATLAB is 1-indexed)
first = array(0);  % Error

% CORRECT
first = array(1);  % First element
```