MATLAB / Octave Primer

for MEC524 Control Engineering

The written companion to the course videos *Octave for control engineering*. Not a graded artefact.

How to use this primer

Each section matches one video on the course page **Octave for control engineering** (Deep Dive, MEC524, in the course contents). Watch the video, then keep this beside you while you work. Every listing here is a file the videos run, and every example system is made up for the course: none is an assessment plant. Everything runs in Octave with the `control` package and in MATLAB; differences are marked *Octave note*.

#	Video	Section
1	Installing Octave	§1
2	The Octave window	§2
3	Setting up your folder	§3
4	Vectors and the dot	§4
5	Scripts, plots and saving a figure	§5
6	Loops, if and functions	§6
7	Transfer functions and step	§7
8	From Simulink to Octave	§8
9	stepinfo, root locus and Bode	§9
10	Reading Octave's errors	§10

1. Installing Octave

MATLAB is on the university lab machines. For work on your own laptop, use the free *GNU Octave*. Download it only from octave.org/download.

Windows ONE TIME

1. On `octave.org/download`, open the Windows section and download `octave-`*version*`-w64-installer.exe` (about half a gigabyte). Not the "64-bit linear algebra" variant: it brings no advantage here.
2. Run it and accept the default folder.
3. Start **GNU Octave (GUI)** from the Start menu.

macOS ONE TIME

Install Homebrew from brew.sh (paste its one-line command into Terminal, then run the two "Next steps" lines it prints). Then:

[Terminal] Type in your system Terminal

```
brew install octave
octave --gui
```

! No `brew` command?

Run the two "Next steps" lines Homebrew printed, then open a new Terminal window.

The `control` package ONE TIME

[Octave »] Type at the Octave » prompt

```
pkg list                    % is "control" in the list?
pkg install -forge control  % only if it is not (needs internet)
```

The Windows installer already bundles several packages: check with `pkg list` first.

! If `pkg install` fails

Check the internet connection and try again. A campus network can block it: try a phone hotspot.

Check that it works

[Octave »] Type at the Octave » prompt

```
pkg load control
G = tf(1, [1 1])
step(G); grid on
```

✓ Expected output

The transfer function $1/(s+1)$ printed back, and a figure window with a step response rising to 1. If both happen, your install is complete.

When MATLAB and Octave disagree

`pkg load` does not exist in MATLAB: guard it with `if exist('OCTAVE_VERSION', 'builtin'), pkg load control; end` and the same script runs in both. `stepinfo` is missing from Octave's package (§9). `exportgraphics` is MATLAB-only: use `print(..., '-dpng', '-r300')`, which works in both.

2. The Octave window

File Browser (your files), **Workspace** (your variables), **Command History** (what you typed), the **Command Window** (where you type, after >>), the **Editor** tab (scripts), and at the top the **Current Directory**: the folder Octave works in.

There are two places to type. The **Terminal** (macOS) is only for installing and starting Octave. Everything else goes into Octave's Command Window.

```
a = 3              % makes a, and prints it
b = 4;             % a semicolon hides the result
a * b              % an unnamed result goes into ans
who                % list your variables
clear; clc; close all    % empty the workspace, the screen, the figures
help sqrt          % how to call a function;  doc sqrt opens the manual
```

3. Setting up your folder

Octave runs the scripts in its current folder and saves files there; anything elsewhere it cannot see. Keep one folder for each piece of work, **named as its paper names the ZIP**, so zipping the folder gives the right name:

```
Documents/
  MEC524/
    Practice/                          <- the Octave course
    MEC524-PBL-M1-G<GroupNo>/          <- PBL Milestone 1
    MEC524-A5-<Group>-<StudentID>/     <- Individual Assignment (A5)
    MEC524-PBL-M2-G<GroupNo>/          <- PBL Milestone 2

Inside each one:
  main.m       <- the one script that runs everything
  plots/       <- every figure it saves
  stepinfo.m   <- if you use stepinfo (Octave only)
```

The papers ask for "a single `.m` entry point"; `main.m` is the name we suggest.

1. **Make the folders** in File Explorer or Finder, inside *Documents*. On Windows, if the address bar says *OneDrive*, that is where your Documents are. Turn on **File name extensions** (View → Show on Windows 11, the View tab on Windows 10), or `main.m` can hide as `main.m.txt`. Never work in *Downloads*: move downloaded files out of it.
2. **Point Octave at the folder**: the folder button beside *Current Directory*, or `cd`:

[Octave »] Type at the Octave » prompt

```
cd ~/Documents/MEC524/Practice                % macOS
cd C:/Users/you/Documents/MEC524/Practice     % Windows (forward slashes work)
cd '~/OneDrive - UiTM/Documents/MEC524'       % a path with spaces needs quotes
pwd                                           % where is Octave now?
ls                                            % what can it see?
```

3. **Make the files there**: `mkdir plots`, and `edit main.m` at the prompt, which creates `main.m` in the current folder. Check with `ls`, and `which stepinfo` shows the file Octave will use.
4. **Before you hand in**: close Octave, open it again, make the folder current, type `main`. Every plot must appear in `plots/`. Then zip the folder itself.

`main.m`

```
% main.m -- the one script that runs everything, top to bottom
clear; clc; close all

cooling          % each part of the work in its own script, in this
   folder
```

4. Vectors and the dot

```
t = 0:0.5:2             % start:step:stop  -> 0 0.5 1 1.5 2
x = linspace(0, 1, 5)   % start, stop, number of points
t(1), t(end)            % counting starts at 1; end is the last
A = [1 2; 3 4];         % rows separated by ;
A(2, 1), A'             % one element; the transpose
A * A                   % matrix product
```

```
A .* A                  % element by element
t.^2, 1 ./ (t + 1)      % element by element: the dot
[m, i] = max(v); find(v > 5); sum(v); size(A)
```

The dot rule. `*`, `/` and `^` are matrix operations. When you mean "each element", put a dot before them: `.*`, `./`, `.^`. `t^2` on a row vector is an error.

5. Scripts, plots and saving a figure

Script names: letters, digits and `_`, starting with a letter; no spaces, no dashes; never the name of a function you use (`step.m`, `plot.m`, `tf.m`). Run a script by typing its name, or with Run (F5) in the editor, with its folder current.

`cooling.m`

```
% cooling.m -- two hot parts cooling in still air
clear; clc; close all

t   = linspace(0, 12, 601);       % time, min
dT1 = 40 * exp(-t / 1.5);         % temperature rise of the thin part, K
dT2 = 40 * exp(-t / 4);           % temperature rise of the thick part,
    K

figure
plot(t, dT1, t, dT2, '--')
grid on
xlabel('Time (min)')
ylabel('Temperature rise (K)')
legend('thin part', 'thick part')
title('Two parts cooling')

if ~exist('plots', 'dir'), mkdir('plots'); end
print('plots/cooling.png', '-dpng', '-r300')
disp('saved plots/cooling.png')
```

Label both axes **with units**; `legend` names lines in the order plotted; `hold on` keeps the last line; `subplot(2,1,1)` stacks two plots. Save every figure from the script at 300 dpi, never as a screenshot.

6. Loops, if and functions

`spring_sweep.m`

```
% spring_sweep.m -- one spring, three masses
clear; clc

k  = 360;                  % spring stiffness, N/m
m  = [0.4 0.9 1.6];        % masses, kg
wn = zeros(size(m));       % room for one answer per mass

for i = 1:numel(m)
  wn(i) = natural_freq(k, m(i));
  fprintf('m = %.1f kg    wn = %.1f rad/s\n', m(i), wn(i));
end
```

```
if all(wn > 12)
  disp('every mass stays above 12 rad/s')
else
  disp('at least one mass is below 12 rad/s')
end
```

`natural_freq.m`: one function per file, named after it

```
function wn = natural_freq(k, m)
% NATURAL_FREQ  undamped natural frequency of a mass on a spring, rad/s
%   wn = natural_freq(k, m)     k in N/m, m in kg
  wn = sqrt(k / m);
end
```

`wn = zeros(size(m))` makes room before the loop. In `fprintf`, `%.1f` prints one decimal place and `\n` ends the line. `==` compares, `=` assigns. Often no loop is needed: `wn = sqrt(k ./ m)` does every mass at once. Use a loop when each step needs the one before (§8).

7. Transfer functions and step

```
pkg load control                % Octave, once per session
s = tf('s');
G = 9/(s^2 + 3*s + 9)           % as on paper
G = tf(9, [1 3 9]);             % coefficients in DESCENDING powers of
   s
pole(G), zero(G), dcgain(G), damp(G)
step(G); grid on                % draw it
[y, t] = step(G);               % or take the numbers
max(y), y(end)
G1 * G2                         % blocks in series
G1 + G2                         % blocks side by side into a sum
zpk([-3], [-1 -2 -10], 1)       % from zeros, poles and gain
ss(A, B, C, D);  tf(ss(A, B, C, D))   % state space, and back
```

Trap: `tf(9, [9 3 1])` is $9/(9s^2 + 3s + 1)$, a different system, and Octave will not complain.

8. From Simulink to Octave

Simulink block	Octave
Transfer Fcn	`G = 3/(2*s + 1)`
Gain	`Kp*G`
Sum and the feedback wire	`T = feedback(Kp*G, 1)` (negative feedback by default)
Step	`step(T)`, or `r = 1.0*(t >= 0.5)`
Any input you draw	`y = lsim(T, r, t)`
Scope	`plot(t, y)`
To Workspace	`[y, t] = step(T)`
Saturation, switches, anything nonlinear	a `for` loop, as in `fan_limit.m`

`fan_loop.m`

```
% fan_loop.m -- the Simulink model of a fan speed loop, as a script
clear; clc; close all
if exist('OCTAVE_VERSION', 'builtin'), pkg load control; end

s  = tf('s');
```

```
G  = 3 / (2*s + 1);          % Transfer Fcn: fan speed (krpm) per volt
Kp = 4;                      % Gain: volts per krpm of error

T = feedback(Kp*G, 1)        % Sum, Gain, Transfer Fcn and the feedback
    wire

t = 0:0.002:3;               % time, s
r = 1.0 * (t >= 0.5);        % Step: 1 krpm from t = 0.5 s
y = lsim(T, r, t)';          % run it (row vector, like t)
u = Kp * (r - y);            % the Gain block's output, V

figure
subplot(2, 1, 1); plot(t, r, '--', t, y); grid on
ylabel('Speed (krpm)'); legend('setpoint', 'speed', 'location', '
   southeast')
subplot(2, 1, 2); plot(t, u); grid on
xlabel('Time (s)'); ylabel('Drive voltage (V)')

if ~exist('plots', 'dir'), mkdir('plots'); end
print('plots/fan_linear.png', '-dpng', '-r300')
```

`step` and `lsim` solve the *linear* model: whatever voltage the controller asks for, it gets. To respect an actuator limit, step the model forward yourself and clip the signal on each step:

`fan_limit.m`

```
% fan_limit.m -- the same loop, with a Saturation block: the drive
   gives at most 1.5 V
% Run fan_loop.m first: it makes G, Kp, t, r and the linear result y.
umax = 1.5;                          % drive limit, V
dt   = t(2) - t(1);
[A, B, C] = ssdata(c2d(ss(G), dt)); % the plant as one step-by-step
   rule

x  = zeros(size(A, 1), 1);           % plant state, starts at rest
ys = zeros(size(t));  us = ys;
for k = 1:numel(t)
  ys(k) = C * x;                              % what the sensor reads now
  us(k) = Kp * (r(k) - ys(k));                % Gain
  us(k) = min(max(us(k), -umax), umax);       % Saturation
  x = A * x + B * us(k);                      % plant moves on by dt
end

figure
subplot(2, 1, 1); plot(t, y, '--', t, ys); grid on
ylabel('Speed (krpm)'); legend('no limit', 'limited to 1.5 V', '
   location', 'southeast')
subplot(2, 1, 2); plot(t, u, '--', t, us); grid on
xlabel('Time (s)'); ylabel('Drive voltage (V)')
print('plots/fan_limited.png', '-dpng', '-r300')
```

Whenever a controller asks for a large signal, check it against the actuator's limit this way.

9. stepinfo, root locus and Bode

Octave note: the `control` package lists `stepinfo` but has not written it. Download `stepinfo.m` from the Octave course page and save it in your work folder (§3). MATLAB has its own: do not

copy it there.

```
S = stepinfo(G)                    % RiseTime (10-90 %), SettlingTime (2 %
    band),
                                   % Overshoot (%), Peak, PeakTime ...
L = 1/((s+1)*(s+3)*(s+6));         % an open loop
rlocus(L)                          % closed-loop poles as the gain rises
bode(60*L); grid on                % magnitude (dB) and phase (deg)
[gm, pm, wpc, wgc] = margin(60*L);    % margins of the OPEN loop
fprintf('GM %.1f, PM %.1f deg\n', gm, pm)
T = feedback(60*L, 1);  all(real(pole(T)) < 0)    % closed loop stable?
bandwidth(T)
print('plots/bode.png', '-dpng', '-r300')
```

Do the algebra first (Routh, the root-locus rules, asymptotes), then let Octave agree. If it does not, look for the mistake in the algebra, then in the code. Automatic tuners are not a design.

10. Reading Octave's errors

Read the line that starts with `error:`: it names what went wrong. In a script it gives the line number.

Octave says	Usually
`'tf' undefined ...` installed, but not loaded	`pkg load control`
`'main' undefined`	the script is not in the current folder: check `pwd`
`'speed' undefined`	a typo, or a `clear` after you made it
`nonconformant arguments (op1 is 1x3, op2 is 1x2)`	sizes do not fit: `size(x)`; a missing dot
`for x^y, only square matrix arguments`	use `.^`
`x(5): out of bound 3`	past the end of the vector
`subscripts must be either integers 1 to ...`	index 0: counting starts at 1
`syntax error near line N`	a bracket or quote left open, on line N or above
`Invalid call to cd`	a path with spaces: put it in quotes
`invalid call to script .../step.m`	your file is named after a function: `which step`, rename it

Asking for help? Send the script, the line that broke, and the exact message, copied; and say Octave or MATLAB, and the version (`version`).

11. Common pitfalls in control work

1. **Coefficient order.** `tf(num, den)` wants descending powers of s.
2. **The dot.** `t.^2`, not `t^2`.
3. **The integrator.** A position plant built from a velocity model needs an extra $1/s$: `tf(1, [1 1])` is velocity, `tf(1, [1 1 0])` is position.
4. **Saturation hidden by `step`.** `step` ignores actuator limits: check the control signal separately (§8).
5. **Feedback sign.** `feedback(G, H)` is negative feedback; `feedback(G, H, +1)` is positive.
6. **`margin` on the closed loop.** It wants the open loop L, never T.

7. **Two integrators.** A position plant with a PI or PID controller is type 2: the Bode magnitude starts at $-40\,\mathrm{dB/dec}$ and the phase at $-180°$. A warning from `margin` is information, not an error.

12. Cheat sheet

Task	Code
Start every script	`clear; clc; close all` then `pkg load control` (Octave)
Where am I?	`pwd`, `ls`, `cd 'path'`
$G(s) = \dfrac{K}{s(s+a)(s+b)}$	`s = tf('s'); G = K/(s*(s+a)*(s+b));`
Poles, damping	`pole(G)`, `damp(G)`
Step response, numbers	`[y, t] = step(G); stepinfo(G)`
Closed loop	`T = feedback(C*G, 1)`
Any input	`y = lsim(T, r, t)`
Root locus	`rlocus(L)`
Bode, margins	`bode(L); [gm, pm] = margin(L)`
Stability of a loop	`all(real(pole(T)) < 0)`
Save a figure	`print('plots/name.png', '-dpng', '-r300')`

13. Where this fits in the semester

Assessment	Week	Sections to have open
PBL Milestone 1 (group)	10	§3–§8
Individual MATLAB Assignment (A5)	13	§3–§9
PBL Milestone 2 (group)	14	§8–§9

Weeks are the due weeks in the course plan; the papers, due dates and how to submit come from your lecturer.

MEC524, 202642 semester. Questions to your lecturer: send the script, the line that broke, and the exact message.