

AER1202 Tutorial 2

Preparation for Assignment 2

Acknowledgement: These slides were developed with reference to course materials by Professor Hugh H. T. Liu.

1. Tutorial Roadmap

1. Determinants and characteristic equations — groundwork for Assignment 2 Question 1.
2. Roll control, with the MATLAB commands Assignment 2 Question 2 needs.
3. Simulink practice for the kinematic blocks of Assignment 2 Question 3.

Cofactor Expansion: Pick the Easy Row

Any determinant can be expanded along *any* row or column,

$$\det \mathbf{M} = \sum_j (-1)^{i+j} m_{ij} M_{ij},$$

where M_{ij} is the minor: the determinant of what is left after deleting row i and column j . The signs alternate in a checkerboard starting from $+$ at the top left.

The whole art is **choosing the row or column with the most zeros** — each zero kills a term before you compute its minor. For

$$\mathbf{M} = \begin{bmatrix} 1 & 2 & -1 \\ 0 & 3 & 1 \\ 2 & -1 & 4 \end{bmatrix},$$

expanding along the first *column* needs two minors instead of three:

$$\det \mathbf{M} = 1 \begin{vmatrix} 3 & 1 \\ -1 & 4 \end{vmatrix} - 0 + 2 \begin{vmatrix} 2 & -1 \\ 3 & 1 \end{vmatrix} = 13 + 2(5) = 23.$$

When the Leading Matrix is Singular

For $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$ the modes solve $\det(s\mathbf{I} - \mathbf{A}) = 0$, a polynomial of degree n with n roots.

A *quasi-steady* assumption replaces one differential equation by an algebraic one, giving

$$\mathbf{\Pi}\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}, \quad \mathbf{\Pi} = \text{diag}(0, 1, \dots, 1),$$

and the modes now solve $\det(s\mathbf{\Pi} - \mathbf{A}) = 0$.

Each zero on the diagonal of $\mathbf{\Pi}$ removes one power of s . That row contributes no s to any term of the expansion, so the polynomial degree — and the number of finite roots — drops. The missing roots have gone to infinity: they are the modes the constraint has removed. This is why a fourth-order system can reduce to a *quadratic*.

Worked Example, and the MATLAB Check

With the same $\mathbf{A}$ and $\mathbf{\Pi} = \text{diag}(0, 1, 1)$,

$$s\mathbf{\Pi} - \mathbf{A} = \begin{bmatrix} -1 & -2 & 1 \\ 0 & s-3 & -1 \\ -2 & 1 & s-4 \end{bmatrix}, \quad \det = -s^2 + 9s - 23.$$

Degree 2, not 3 — one power of s lost to the zero in $\mathbf{\Pi}$. Roots $s = 4.5 \pm 1.658i$.

Expand along the *first column* again: the single zero there is what makes it quick.

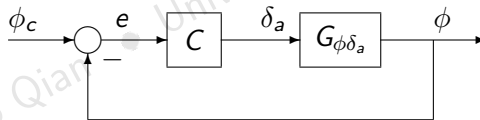
MATLAB will confirm it. $\det(s\mathbf{\Pi} - \mathbf{A}) = 0$ is a generalized eigenvalue problem $\mathbf{A}\mathbf{v} = s\mathbf{\Pi}\mathbf{v}$:

```
eig(A,Pi)           % -> Inf, 4.5000 + 1.6583i, 4.5000 - 1.6583i
```

The Inf is the mode the constraint removed. Use this to *check* a hand expansion — it does not replace the derivation.

3. Roll Control and MATLAB

When we move the attention to the lateral control, we shall discuss the bank and yaw control of the aircraft through the aileron and rudder. In terms of roll control, it is most effectively handled by deflecting the aileron δ_a .



In the [B747-100] case, we have

$$G_{\phi\delta_a} = \frac{-0.1433s^2 - 0.02733s - 0.1104}{s^4 + 0.6365s^3 + 0.9393s^2 + 0.5122s + 0.003678}$$

where the open-loop zeros and poles are:

$$z_{1,2} = -0.0953 \pm 0.8719i,$$

$p_{1,2} = -0.0329 \pm 0.9460i, p_3 = -0.5614, p_4 = -0.0073$. Again, it suggests the opposite impact of deflecting the aileron to the bank angle, i.e., the positive δ_a (left up right down) will cause bank to the right motion. The classical approach can be applied for control design, where the Bode plot and root locus are shown here. (Please note, we use $-G_{\phi\delta_a}$ to normalize the transfer function).

What we would like to discuss, using this roll control example, is that we often engage a so-called multi-loop design. In this case, we acknowledge that when the bank angle is regulated through the aileron, the bank rate p is affected as well, and the rate change is even more sensitive, faster in reaction than the angle (the angle is the integral term of the rate, takes a while to kick in). Therefore, a well received approach is to set up an inner-loop and outer-loop structure to reflect this fact. The standard control structure can be described by Fig. 1.

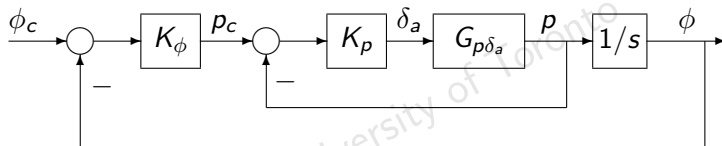


Figure 1: Roll Control Block Diagram: Inner-outer Loop Structure

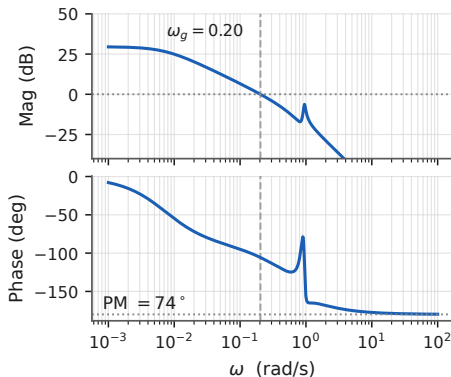


Figure 2: Bank-angle channel $-G_{\phi\delta_a}$: gain crossover $\omega_g = 0.20$ rad/s, phase margin 74° .

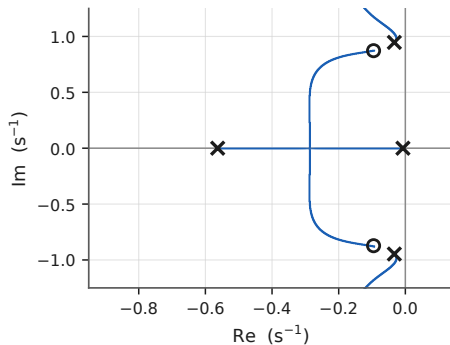


Figure 3: Root locus of $-G_{\phi\delta_a}$: poles -0.563 , $-0.033 \pm 0.947i$, -0.007 ; zeros $-0.095 \pm 0.872i$.

We can apply the classical approach to design the inner loop first, using the frequency response and root locus plot of $G_{p\delta_a}$, as follows in [B747-100] case:

$$G_{p\delta_a} = \frac{-0.1433s^3 - 0.02733s^2 - 0.1104s}{s^4 + 0.6365s^3 + 0.9393s^2 + 0.5122s + 0.003678}$$

Please note the additional zero $z = 0$ represents the fact that $p = \dot{\phi}$, or $p(s) = s\phi(s)$. Once the inner loop is designed, it is often supplemented by an outer-loop, where a simple proportional gain (or PID) control will generate the new command for p_c based on the feedback of ϕ . A simulation result of such a design is shown in Fig. 6.

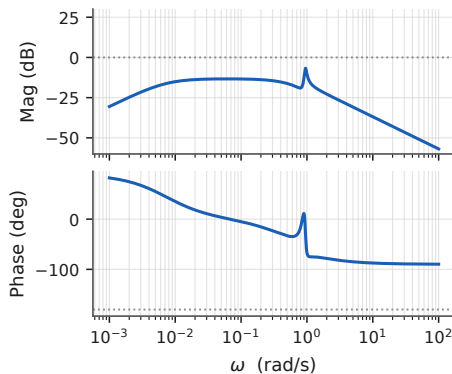


Figure 4: Roll-rate channel $-G_{p\delta_a}$: the magnitude peaks at -6.7 dB, so no crossover exists until $K \gtrsim 2.2$ — the inner loop needs gain before a margin can be read at all.

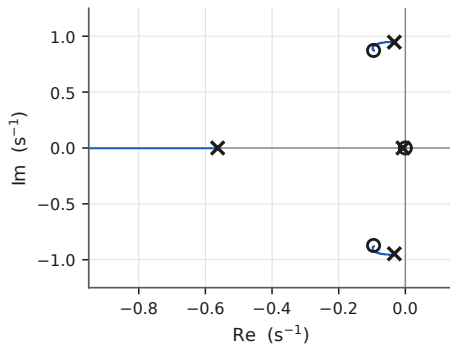


Figure 5: Root locus of $-G_{p\delta_a}$: same poles, with the extra zero at the origin that expresses $p = \dot{\phi}$.

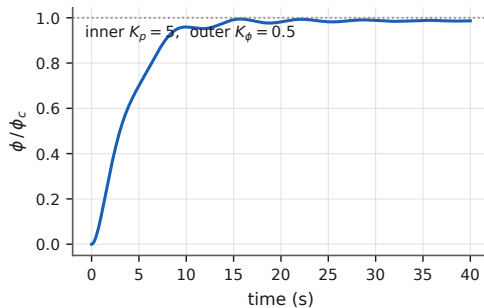


Figure 6: Two-loop closed-loop response: inner $K_p = 5$, outer $K_\phi = 0.5$.

The MATLAB for a Two-Loop Design

The roll example above and Assignment 2 Question 2 share one architecture: an inner loop closed first, then an outer loop designed around it. The commands are the same in both cases.

```
G = ss(A,B,eye(4),zeros(4,2)); % pick the channel
Gpa = tf(G(2,1)); % p <- delta_a (G(output
dcgain(Gpa) % CHECK THE SIGN before closi

margin(Kp*Gpa) % w_g and PM of the inner loo
rlocus(Gpa) % or the root-locus view

Tin = feedback(Kp*Gpa, 1); % INNER loop closed
Tout= feedback(Kphi*Tin/s, 1); % OUTER loop around it

stepinfo(Tout), pole(Tout), dcgain(Tout)
```

Close and verify the **inner** loop on its own before writing the outer one: if the inner loop is wrong, nothing outside it means anything. And build interconnections in `ss` rather than `tf`, so the model order never inflates.

Build Small, Verify, Then Connect

Assignment 2 Question 3 asks for three kinematic blocks. The blocks themselves are straightforward; what costs people time is debugging them once they are wired together. The discipline that avoids this:

- ▶ Build **one** block. Give it constant inputs. Compare the output against a value you already know.
- ▶ Only then make it a subsystem (select the blocks, right-click → *Create Subsystem from Selection*) and give it a name and labelled ports.
- ▶ Connect the verified subsystems last.

Every block in Q3 comes with a table of check points for exactly this purpose. A block that reproduces five independent test values is almost certainly right; one that is only tested after assembly tells you nothing about *which* block is wrong.

Two ways to realise any of them: primitive blocks (Trigonometric Function, Product, Sum, Gain) or a single MATLAB Function block containing the expression. Either is acceptable; the second is usually quicker to get right and easier to read.

Practice 1: A Static Nonlinear Block

Before the three-dimensional case, build the two-dimensional analogue — Cartesian to polar:

$$r = \sqrt{x^2 + y^2}, \quad \vartheta = \text{atan2}(y, x).$$

It exercises the same pieces as a body-velocity block: a norm, a quadrant-correct inverse tangent, and two outputs from one vector input.

(x, y)	r	ϑ [deg]
(3, 4)	5.0000	53.13
(-1, 1)	1.4142	135.00
(0, -2)	2.0000	-90.00

Use `atan2`, never `atan`: the latter cannot tell $(-1, 1)$ from $(1, -1)$. The same trap waits in the angle-of-attack block.

Practice 2: Driving a Model from Data, and Integrating

Question 3(c) needs a recorded input history and an integrator. Practise the mechanics on something whose answer you can check by hand — integrate a constant rate:

```
t = (0:0.02:10)';           % column vector
u = 2*ones(size(t));        % constant input
simin = [t u];              % From Workspace wants [time, data]
```

Feed `simin` to a **From Workspace** block, pass it through an **Integrator**, and confirm the output is $2t$: exactly 20 at $t = 10$ s.

Three things to get right before trusting any integration:

- ▶ the **initial condition** on the Integrator (Q3(c) starts from zero);
- ▶ the **solver step** — a fixed step matching your data, or a variable-step solver with a small enough maximum;
- ▶ **units** — the model works in radians throughout; convert to degrees only for plotting.

Summary

- 1 Tutorial Roadmap
- 2 Determinants
- 3 Roll Control and MATLAB
- 4 Simulink Practice