SFWR ENG 3DX4 Summary

Instructor: Dr. Lawford

Course: SFWR ENG 3DX4

*Math objects made using* [*MathType*](http://www.dessci.com/en/products/mathtype/)*.*

Table of Contents

[Introduction to Systems 1](#_Toc417574629)

[Laplace 1](#_Toc417574630)

[Transfer Functions 1](#_Toc417574631)

[Electrical 1](#_Toc417574632)

[Component stuff 1](#_Toc417574633)

[Mesh Analysis 1](#_Toc417574634)

[Noodal Anal 1](#_Toc417574635)

[Cramer’s Rule 1](#_Toc417574636)

[OP-Amps 1](#_Toc417574637)

[Mechanical 1](#_Toc417574638)

[Translational Systems 1](#_Toc417574639)

[Rotational Systems 1](#_Toc417574640)

[Degrees of Freedom 1](#_Toc417574641)

[Signals 1](#_Toc417574642)

[Final Value Theorem 1](#_Toc417574643)

[Graph Stuff 1](#_Toc417574644)

[Non-/Linear Systems 1](#_Toc417574645)

[Block Diagrams 1](#_Toc417574646)

[State Space Equations 1](#_Toc417574647)

[Transfer Function -> State Space 1](#_Toc417574648)

[Stability 1](#_Toc417574649)

[Root Locus 1](#_Toc417574650)

[e.g.) 1](#_Toc417574651)

[Second Order Approximation 1](#_Toc417574652)

[Routh-Hurwitz Table 1](#_Toc417574653)

[Steady-State Error 1](#_Toc417574654)

[Canonical Form 1](#_Toc417574655)

[Phase Variable 1](#_Toc417574656)

[Control 1](#_Toc417574657)

[Observer 1](#_Toc417574658)

[MATLAB Stufff 1](#_Toc417574659)

Note: the following summaries may be useful:

* [SFWR ENG 2MX3](https://drive.google.com/open?id=0BxW61uJyyN8TTWx5d0gzQW9ZUzQ&authuser=0)
* [ENGINEER 3N03](https://docs.google.com/document/d/117z1qGbrDJJV9bx57CQ4SxEL8Ws8oL27bM7-NgFHNKU/edit)
* [TRON 3TA4](https://drive.google.com/file/d/0BxW61uJyyN8TLTR4UV9fYVdBeEU/view?usp=sharing)

I may review to clarify or correct, but mostly I will omit those things.

# Introduction to Systems

Systems can be represented by **block diagrams** to make it easier to marginalize the different parts of the systems.

**Transducer**: converts any form of energy to electrical signals

# Laplace

Useful for…

Time begins when your signal begins



Initial conditions:

1. *c*(0)

**Time domain** (*t*): variables are lower case, e.g. *f* (*t*)

**Frequency domain** (*s*): variables are upper case, e.g. *F* (*s*)

**Transfer function**:

When doing the inverse Laplace, it’s useful to break your fractions up so that you can

**Strictly Stable**: it will eventually get back to the initial position

**Marginally Stable**:

**Unstable**: it will progressively get worse



# Transfer Functions

## Electrical

### Component stuff

**Impedence**:



**Current**

****

**Voltage**



**admittance**:







### Mesh Analysis

Add the voltages, where V = IZ

### Noodal Anal

1. Identify nodes
2. Represent currents in terms of voltage

### Cramer’s Rule





### OP-Amps

## Mechanical

**Translational systems**:

**Rotational Systems**:

**Newton’s Second Law of Motion**: Σ *f = Ma*





### Translational Systems

For sure make a free-body diagram

#### e.g.











All inductances are in the opposite direction of the applied force

#### Spring

Spring is like a capacitor

**Force displacement**:

#### Viscous Damper

Using viscous fluid to slow something down

Viscous Damper is like a resistor

**Force displacement**:



#### Mass

Mass is like a inductor

**Force displacement**:

F(s) = MXs2

### Rotational Systems

**Impedence**: 



1. Each θ is on an inertia block. The impedences connected to the motion at θ include the impedences directly to the left and right of the inertia block.
2. When finding the sum of impedences between 2 θ’s only count the impedences on wires that don’t go through other θ’s, i.e. 0 if no direct connection
3. When there is a torque, but no inertial block, draw a fake inertial block





#### Motors and Gears

1. Pick an end of the system to use as a reference frame. Choose the easiest one and walls don’t move.
2. Represent T

**Meshing Gears** are represented in the following way: 

[N]: number of teeth

Let’s assume var1 = before and var2 = after.

When gears are lined up

**Applied Armature Voltage** [ea]: a.k.a. input voltage

**Armature Resistance** [Ra]:

**Motor Torque Constant** [Kt]:

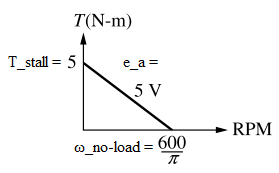
**Back EMF Constant** [Kb]:

**No load speed** [ωno-load]: when the voltage line touches the x-axis



**Stall torque**[Tstall]: when angular velocity reaches 0, i.e. y-intercept if equation is given.





[Ja]: any J on the same line, including a motor

[JL]: load J

[Jm]:

[Dm]: coefficient of viscous dampening







**Hints**:

* If you have a spring and / or a damper in series, the wire between them rotates independently

### Degrees of Freedom

How to calculate

1. count the number of masses/moments of inertia blocks
2. find any hidden inertia blocks

## Signals

**Transducer**: anything that converts energy to electrical energy

**Transmitter**: long distances

Unstable systems have ∞ steady state error

**Steady-state error** [*e∞*]:



### Final Value Theorem

**Final value theorem**: finds steady state error



So and you’re given F(s), so just multiply by s and find the limit.

There are limitations as to where you can use this theorem. It is dependent on the location of the poles.

#### 1) Right half plane



System is unstable: e+ 🡪 ∞

#### 2) Imaginary Axis − Origin



Unstable: ei 🡪 Oscillatory system, so limit will be average, i.e. midpoint

#### 3) Left Half Plane



Stable: e− converges to 0, but makes transfer function 0 for every single pole

#### 4) Origin



Stable: integrator, i.e. 1/s, so 

Don’t use this theorem if any poles are 1 or 2.

### Graph Stuff

**Rise time** [*Tr*]: time between 10% and 90% of final value [*c*final]

**Peak time** [*Tp*]: time it takes to get to highest peak [*c*max]

**Settling time** [*Ts*]: how long it takes to get to the steady state within ±2%

****

**Damping Ratio** [ζ]: 

**Percent overshoot** [%OS]: how much further is the peak from the final



**Time Constant** [τ]: the time it takes the system’s step response to reach 1 – 1/e = 63.2% of cfinal

**Second-order**:







For each pole,



**Forced response**: when a = 0

**Natural response**: when a > 0

**Nonminium-phase system**: Initially the system starts in the wrong direction, then stabilizes at the right place

# Non-/Linear Systems

1. Op Amps are linear
2. If you don’t have enough voltage, your motor magnets won’t have enough power to switch poles, so they require a minimum voltage

You can’t model non-linear systems, until you linearize it. To do this, we find the slope and approximate the equation of the line, using y=mx+b

# Block Diagrams

A way of representing a system

**Summing junction**: could be an X or +, but usually an X in this course

**Cascade**: subsystems in series are multiplied

**Parallel**: parallel subsystems have a *summing junction* at the end, so you just add everything together

**Feedback**: positive feedback is bad

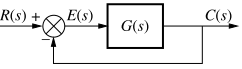


Positive:

Negative: 

Simplification:

**Unity Feedback**: when the feedback path has multiplicative value of 1



# State Space Equations

Yeah, you think you know them from 2MX3, but you don’t really know them. Apparently the ABCD variables actually have names.

* **System Matrix [A]**:
* **Input Matrix [B]**:
* **Output Matrix [C]**:
* **Feedforward Matrix [D]**:



# Transfer Function -> State Space

**Phase Variable Approach**:

The *n* state variables will consist of:

* *y*
* the derivatives of *y*

A-1 = 

adjoint(A) = (cofactor matrix(A))T



cofactor(A) = 



# Stability

**Root Mean Square (RMS)**: the effective DC value of an AC current, by finding a special average



**Gain** [K]:

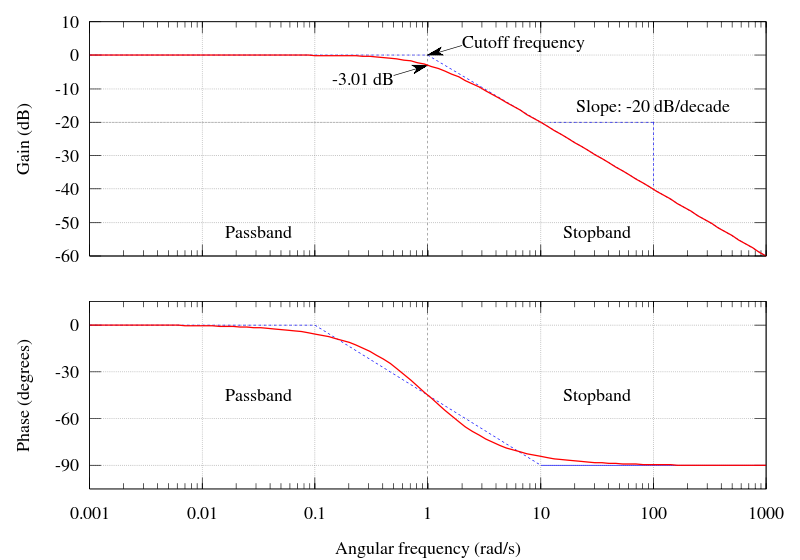
**Bode plot**: graph of frequency response of a system, using a *phase graph* and *gain graph*

1. Find all poles, zeroes, and K value
2. Represent each pole and zero in terms of a fraction added to a 1, i.e.

**Cutoff Frequency**: (a.k.a. *breakaway point*) low pass filter is said to pass frequencies lower than ωc and reject those that are higher than ωc. In other words, the pass(ing) band is ω < ωc.

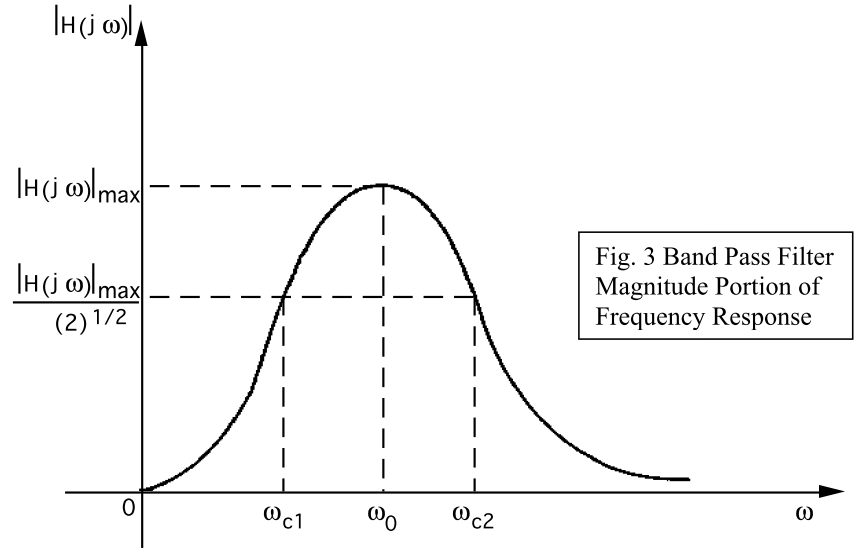
How to find from chart:

* magnitude = −3Db
* phase = −45°
* ωc = ω((½)½ × amplitudemax) = ω(0.707 × Amax)



Types:

1. Constant(K): M = 20log(K), Ø = 0
2. Integration(1/s): M = –20log(|jω|)
3. Derivative(s):
4. 1st order lag: focus on poles
5. 1st order lead: focus on zeroes
6. 2nd order lag: focus on poles
7. 2nd order lead: focus on zeroes



## Root Locus

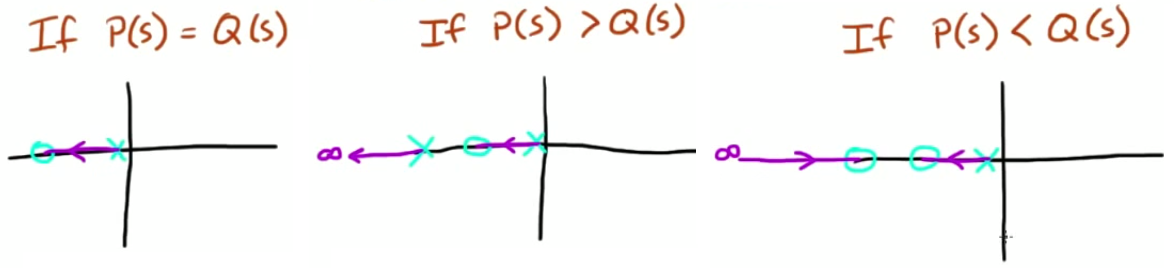
A plot that helps you find the k value that gives your system your desired level of stability.

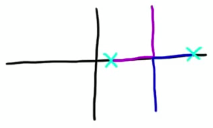
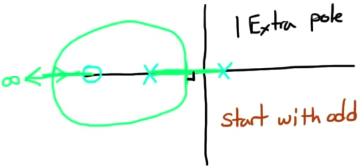


**Branch**: starts at a pole, i.e. open-loop-zero

branches can be endless, going to infinite

1. Number of poles/zeros (whichever is greater) = number of branches
2. As K moves from 0→∞, roots move from poles of G(s) to zeros of G(s). In other words, lines go from poles to zeroes.



1. Roots that are complex, i.e. not on the real axis, always come in pairs of positive and negative, i.e. above and below at the same r location. In fact, the path is completely mirrored.
2. The path the roots take will never cross itself, unless 2 roots meet, in which case the lines break out 
3. Right-to left priority.
4. Lines only break out at 90°.
5. Poles with no zeros on the left will go to infinity. Zeros with no poles on the right will have lines coming from infinity. 
6. To find the position of the asymptote (8.27): , where you’re summing the positions of the poles and the angles of the asymptotes are 
7. Hint: if there are 2/+ lines going to infinite, Σroots = constant

You can have multiple σ’s: ||n−m|| = # of σ’s

**Break out points**:

Solve for σ, where:

(denominator, i.e. poles)

**Break in points**:

Solve for σ, where

(numerator, i.e. zeros)

Sometimes you’ll need to put extra poles/zeros to get the proper stability because K isn’t enough.

Cancelling zeroes / poles

### e.g.)

If you have k in a weird random spot in the denominator, you need a way to access it.



1. Group all *k* terms
2. Divide by non-*k* terms



### Second Order Approximation







For dumb people who don’t use radians, use 180 degrees instead of π. I hate degrees though, so I’m not even going to write it out.

**Compensator Zero** [zc]: eventually cancelled out,

θzc is the angle that is the result of the zc.

#### Graph method

noodles

## Routh-Hurwitz Table

[MG]: Gain Magnitude



**Phase-variable representation**:

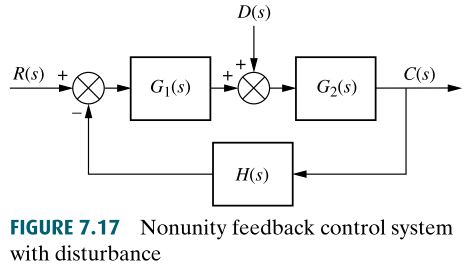
**Phase margin** []: 

## Steady-State Error

Difference between the final value of the step response and the predicted final value

There are 3 static error constants (Kp, Kv, Ka)

* Step:
  + 
  + 
  + 
  + 
  + Kp (position constant)
* Ramp:
  + 
  + 
  + 
  + 
  + Kv (velocity constant)
* Parabolic:
  + 
  + 
  + 
  + 
  + KA (acceleration constant)





**Disturbance [D]**:

**Proportional Integral (PI)**:

PI: Kp & Ki enter system

**Proportional Derivative (PD)**: a.k.a. proportional differential

**Proportional-Integral-Derivative (PID)**: Kp & Ki & Kd enter the system

Note: If your gears are vibrating, your PID is probably too high

**Compensator**: a way of altering your system to your design constraints, mathematically

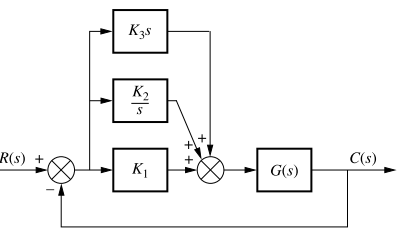
Some types of compensators:

**Compensator equation**: a.k.a. controller equation

[Kd]: the derivative of the error of the system (K3)

[Kp]: a proportion that is multiplied by the error of the system (K1)

[Ki]: an integral of the error of the system (K2)



# Canonical Form

## Phase Variable

Phase variable: 



rotate matrices=>

Controller Canonical: 



## Control

, where n is the rank of B



**Rank**: length of the smallest size when it’s a triangular matrix

## Observer

lala

# MATLAB Stufff

* angle:
* atan2:
* evalfr:
* feedback: for all your transfer function needs
* pole(compensator): input
* roots(polynomial): column vector of polynomial roots
* rlocus: graphs the root locus
* sisotool(compensator):
* step(compensator): step input to compensator
* s=tf('s'): transfer function
* [num,den] = ss2tf(A, B, C, D, 1): State Space to transfer function, outputting
* zpk([zero1 zero2], [pole1 pole2], gain): zero pole gain