Introduction to MATLAB for Control Engineers

EE 447

Autumn 2008

Eric Klavins

Part I

Matrices and Vectors

Whitespace separates entries and a semicolon separates rows.

```
>> A=[1 2 3; 4 5 6; 7 8 9];        # A 3x3 matrix
>> x=[1;2;3];                      # A n-dim (column) vector
```

You can access parts

```
>> A(2,3)
ans =
      6
>> A(:,2)
ans =
      2
      5
      8
>> A(3,3)=-1
ans =
      1   2   3
      4   5   6
      7   8   -1
```

All the standard stuff is there

```
>> A'           # transpose
>> A*x          # multiplication
>> A*A
>> x'*x
>> det(A)       # determinant
>> inv(A)
...
```

Eigenvalues and Eigenvectors

```
>> A

A =
     1     2     3
     4     5     6
     7     8    -1
```

```
>> [P,D] = eigs(A)

P =
   -0.3054   -0.2580   -0.7530
   -0.7238   -0.3770    0.6525
   -0.6187    0.8896   -0.0847

D =
   11.8161         0         0
         0   -6.4206         0
         0         0   -0.3954

>> P*D*inv(P)
ans =

    1.0000    2.0000    3.0000
    4.0000    5.0000    6.0000
    7.0000    8.0000   -1.0000
```

Jordan Form

```
>> B=[1 3 4 1; 0 1 1 2; 0 0 1 0; 0 0 0 1]

B =
     1     3     4     1
     0     1     1     2
     0     0     1     0
     0     0     0     1

>> [M,J]=jordan(B)

M =
    3.0000         0         0         0
         0    1.0000    1.0000    2.3333
         0         0   -1.0000   -2.0000
         0         0    1.0000    1.0000
J =
     1     1     0     0
     0     1     1     0
     0     0     1     0
     0     0     0     1
```

```
>> M*J*inv(M)

ans =

    1.0000    3.0000    4.0000    1.0000
         0    1.0000    1.0000    2.0000
         0         0    1.0000         0
         0         0         0    1.0000
```

Solving Ax=b for x

```
>> A
A =

     1     3     4     1
     0     1     1     2
     0     0     1     0
     0     0     0     1
>> b=[1;0;1;0]
b =

     1
     0
     1
     0
>> x=A\b
x =
     0
    -1
     1
     0
```

Symbolic Calculations

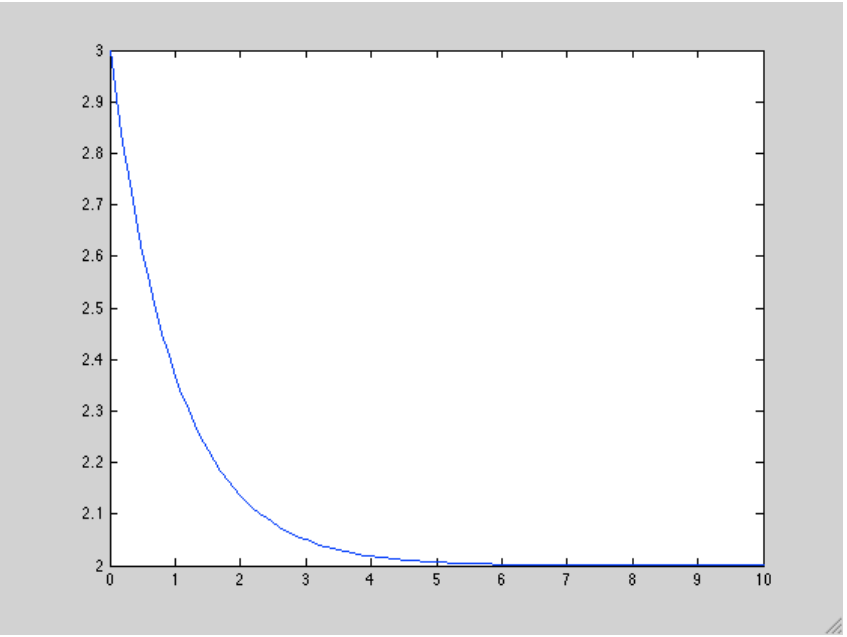
```
>> syms k        # declares k to be a symbol
>> A=[1 k; k 1]

A =
[ 1, k]
[ k, 1]


>> [M,J]=jordan(A)

M =
[  1/2,  1/2]
[ -1/2,  1/2]

J =
[ -k+1,    0]
[    0,  k+1]
```

The 2x2 identity matrix

```
>> syms lam
>> cp=det(lam * eye(2) - A)

cp =

lam^2-2*lam+1-k^2


>> lam=solve(cp)

lam =

  k+1
 -k+1
```

Solving Differential Equations Exactly

```
>> sol=dsolve('Dx = a*x + b', 'x(0) = x0')

sol =

-1/a*b+exp(a*t)*(x0+1/a*b)

>> f=subs ( sol, {'a','b','x0'}, {-1,2,3} )

f =

2+exp(-t)

>> tp=0:0.1:10;
>> plot(tp,subs(f,tp))
```



Solving Differential Equations Approximately

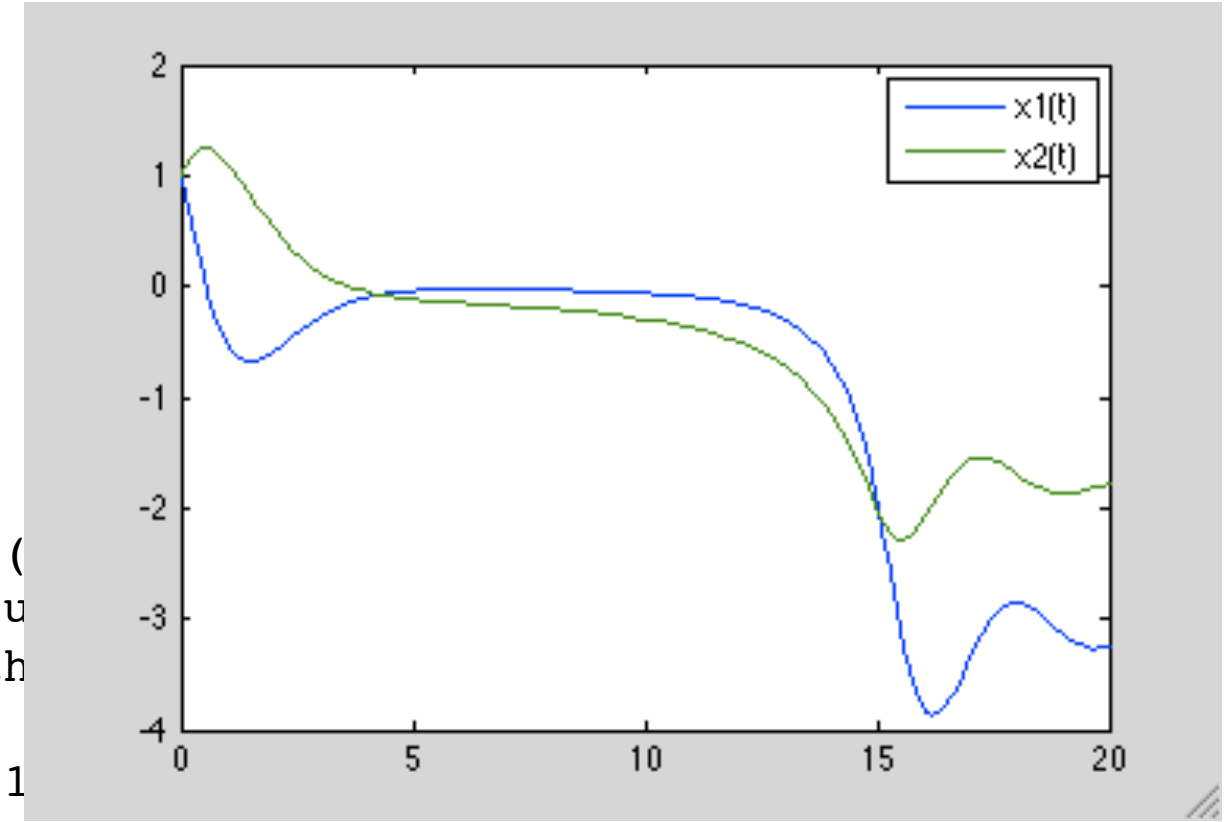
```
# this defines a function of t and a column vector x
>> f = @(t,x) [ -x(1) - x(2)^2; sin(x(1)) ]

>> f(1,[2,3])          # for example, this is f
                       # applied to some arguments

ans =

  -11.0000
    0.9093

>> [t,x] = ode45 ( f, [0,20], [1,1] );
    # the ode45 function numerically
    # simulates the ode represented by f

>> plot ( t, x(:,1), t, x(:,2) )
>> legend('x1(t)', 'x2(t)')
```

time interval

initial condition

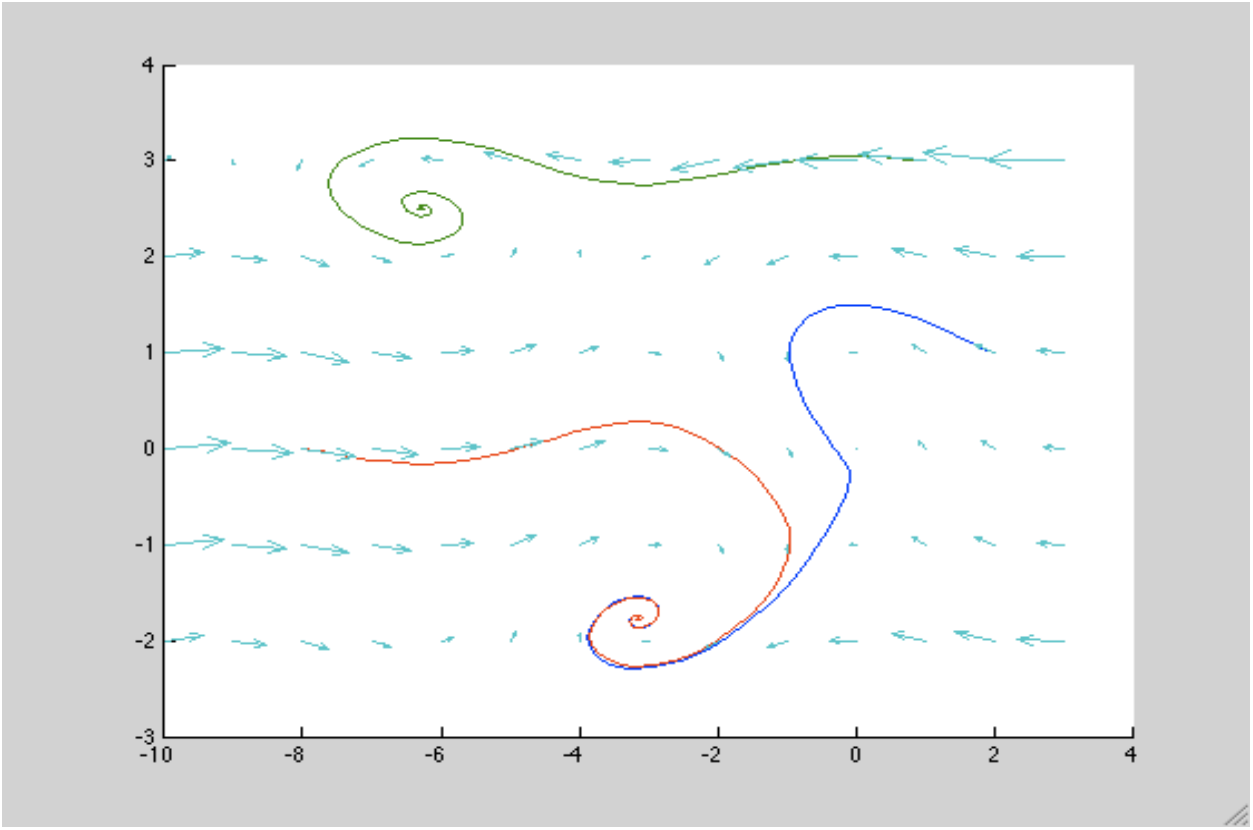
Solving Differential Equations Approximately

```
# this defines a function of t and a column vector x
>> f = @(t,x) [ -x(1) - x(2)^2; sin(x(1)) ]

>> f(1,[2,3])


ans =

  -11.0000
    0.9093

>> [t,x] = ode45 (
    # the ode45 fu
    # simulates th

>> plot ( t, x(:,1
>> legend('x1(t)', 'x2(t)')
```



You can also look at the x-y plane

```
f = @(t,x) [ -x(1) - x(2)^2; sin(x(1)) ];

hold all

[t,x] = ode45 ( f, [0,20], [2,1] ); plot ( x(:,1), x(:,2) );
[t,x] = ode45 ( f, [0,20], [1,3] ); plot ( x(:,1), x(:,2) );
[t,x] = ode45 ( f, [0,20], [-8,0] ); plot ( x(:,1), x(:,2) );

[X,Y] = meshgrid(-10:1:3,-2:1:3);
DX=-X-Y.^2;
DY=sin(X);
quiver(X,Y,DX,DY);
```



How does numerical simulation work?

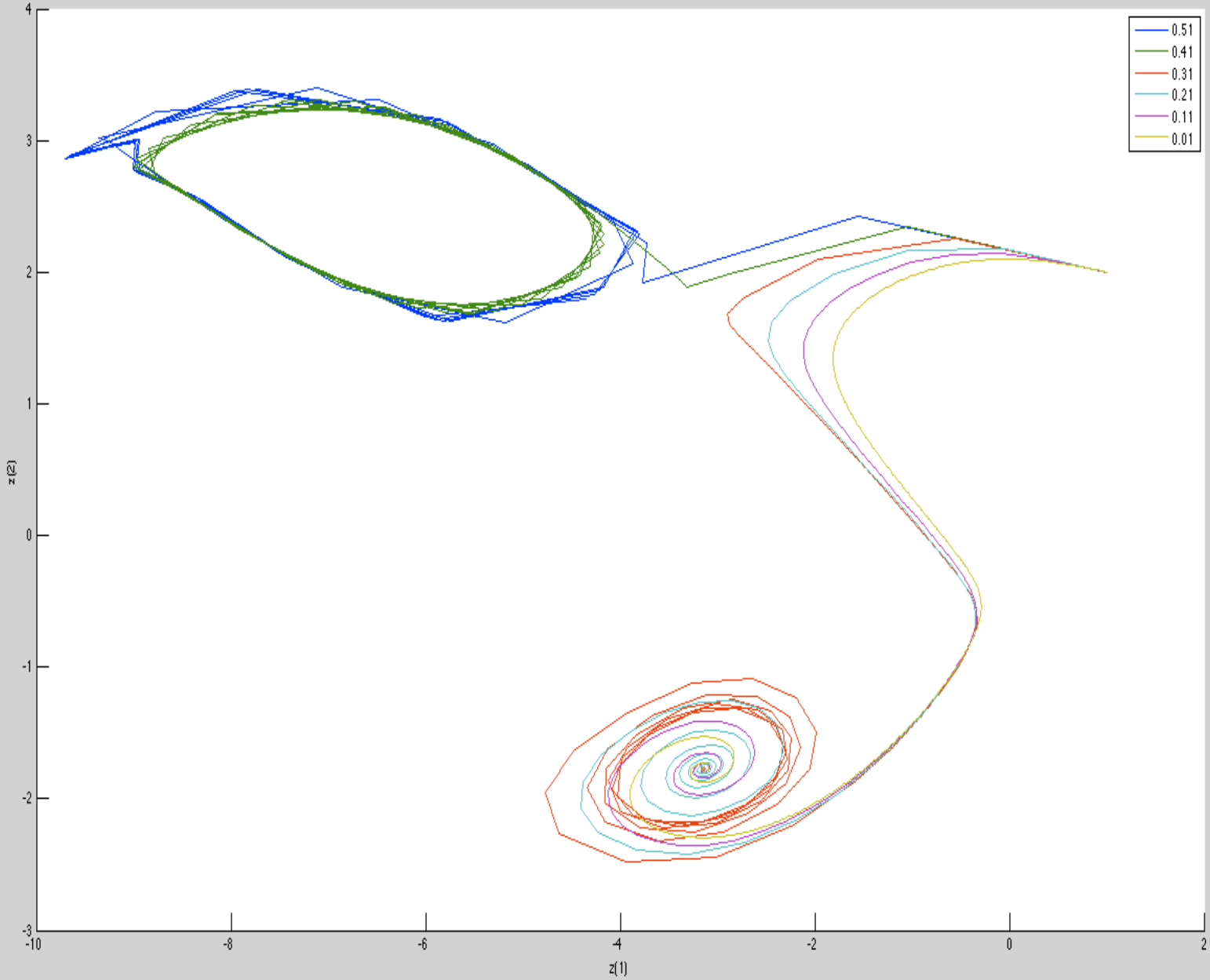
$$\dot{x} = f(x,t)$$

$$\dot{x} = \lim_{h \to 0} \frac{x(t+h) - x(t)}{h}$$

$$\frac{x(t+h) - x(t)}{h} \approx f(x,t)$$

$$x(t+h) \approx x(t) + hf(x,t)$$

```
f = @(t,x) [ -x(1) - x(2)^2; sin(x(1)) ];
tf=30;
zinit=[1;2];

figure
hold all

for h=0.51:-0.1:0.01

    n=int16(tf/h);
    t=zeros(1,n);
    z=zeros(2,n);
    z(:,1) = zinit;
    t(1) = 0;

    for i=2:n
       z(:,i) = z(:,i-1) + h * f ( i*h, z(:,i-1) );
       t(i) = i*h;
    end

    plot ( z(1,:), z(2,:) );

end
```


0.51
0.41
0.31
0.21
0.11
0.01
z(2)
z(1)
4
3
2
1
0
-1
-2
-3
-10
-8
-6
-4
-2
0
2

Functions Can be Defined in Files Instead (must be in same directory)

```
function dx = lorenz_func ( t, x )

sigma = 10;
rho = 28;
beta = 8/3;

dx = [
    sigma * ( x(2) - x(1) );
    x(1)*(rho-x(3))-x(2);
    x(1) * x(2) - beta*x(3)
    ];
```

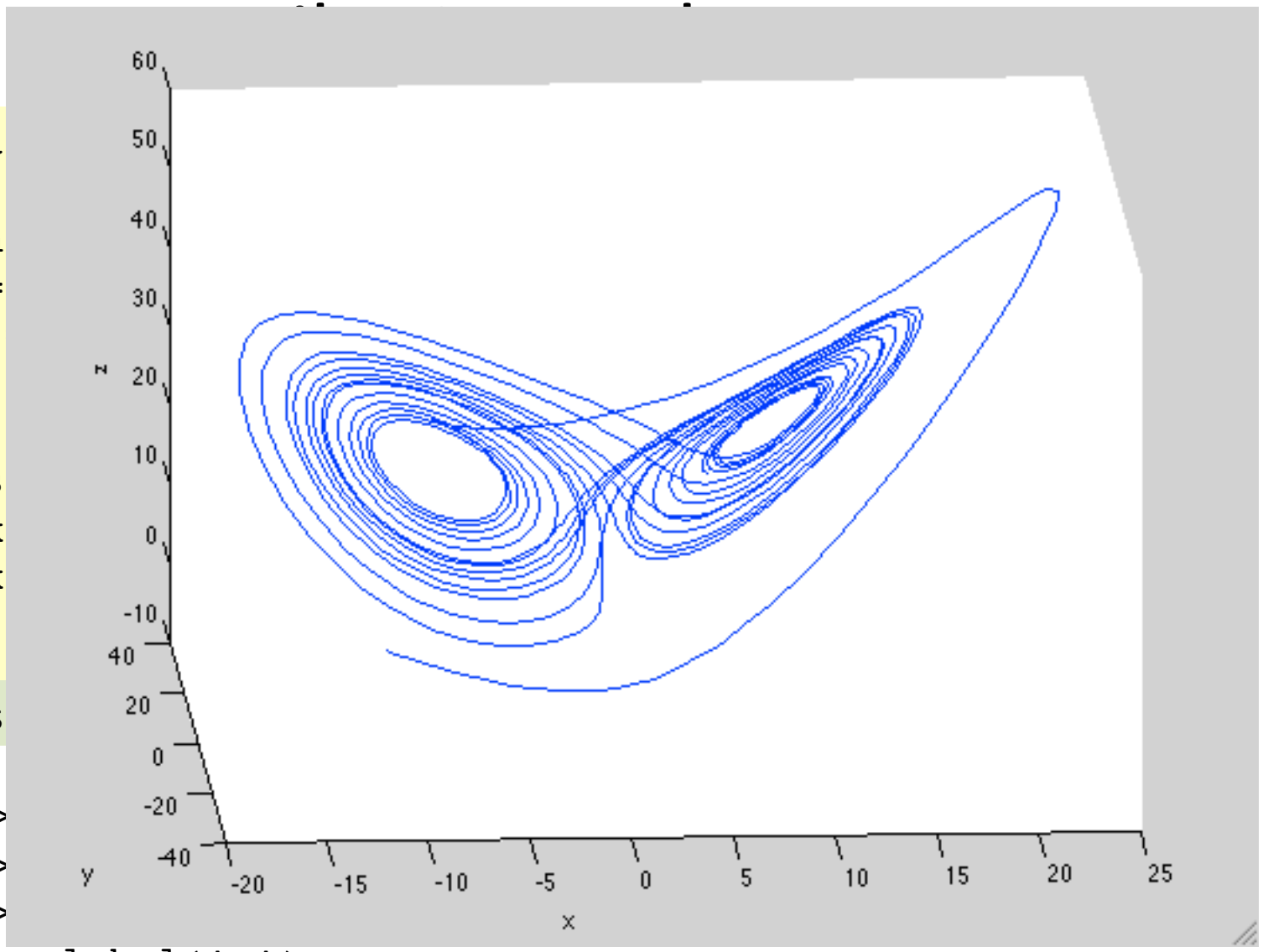
Contents of lorenz_func.m

```
>> [t,x] = ode45 ( 'lorenz_func', [0,20], [-10,20,-5] );
>> plot3 ( x(:,1), x(:,2), x(:,3) )
>> xlabel('x')
>> ylabel('y')
>> zlabel('z')
```

Functions Can be Defined in

```
funct

sigma
rho =
beta

dx =
     s
     x
     x
     ]
```

Contents



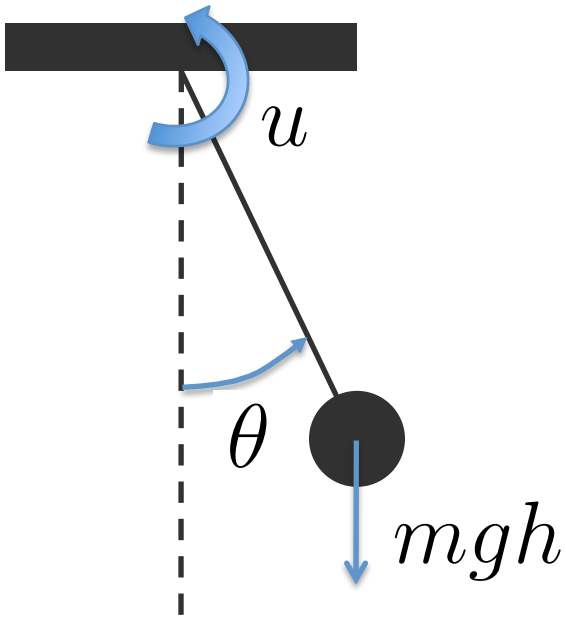

```
>>                                                    );
>>
>>
>> ylabel('y')
>> zlabel('z')
```

How to use MATLAB in This Class

- Unless an exercises says to use MATLAB, you should do the exercise by hand.
- You can use MATLAB to check your work.
- You must show your work in mathematical problems and calculations.
  - E.g. when finding jordan form, what equations did you set up and solve?

Part II: An Example

The Pendulum



To do:

1) Understand the natural behavior (when u=0)
2) Build and analyze a feedforward controller
3) Build and analyze a proportional controller
4) Build a PD controller
5) Build and analyze a PID controller

$$\begin{pmatrix} \dot{\theta} \\ \dot{\omega} \end{pmatrix} = \begin{pmatrix} \omega \\ -a \sin\theta - b\omega + u \end{pmatrix}$$

Torque due to gravity

Torque due to friction

Torque from motor

Natural Behavior (u=0)

Equilibrium points

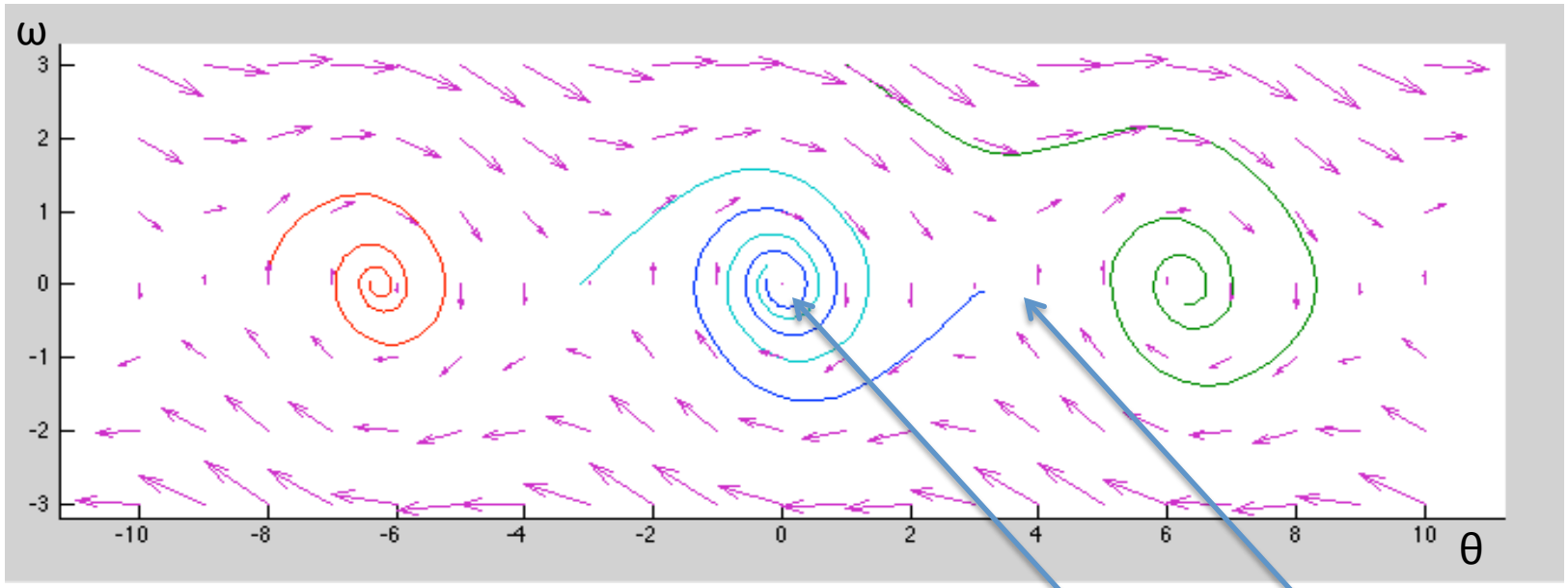
$$\begin{pmatrix} \omega \\ -a\sin\theta - b\omega \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\omega = 0$$

$$\theta = 0,\ \pm\pi,\ \pm 2\pi, \ldots$$

We expect that 2πn are stable and (2n+1)π are unstable – but we don't have the analytical tools yet.

So let's plot the vector field.




```
a=1;
b=0.25;
f = @(t,x) [ x(2); -a*sin(x(1)) - b*x(2) ];

hold all

[t,x] = ode45 ( f, [0,20], [pi-0.01,-0.1] );
plot ( x(:,1), x(:,2) );
...

[TH,OM] = meshgrid(-10:1:10,-3:1:3);
DTH=OM;
DOM=-a*sin(TH)-b*OM;
quiver(TH,OM,DTH,DOM);
```

Feedforward

Say we want θ to go to a desired angel r.

Note, sin θ ≈ 0 so at equilibrium we get

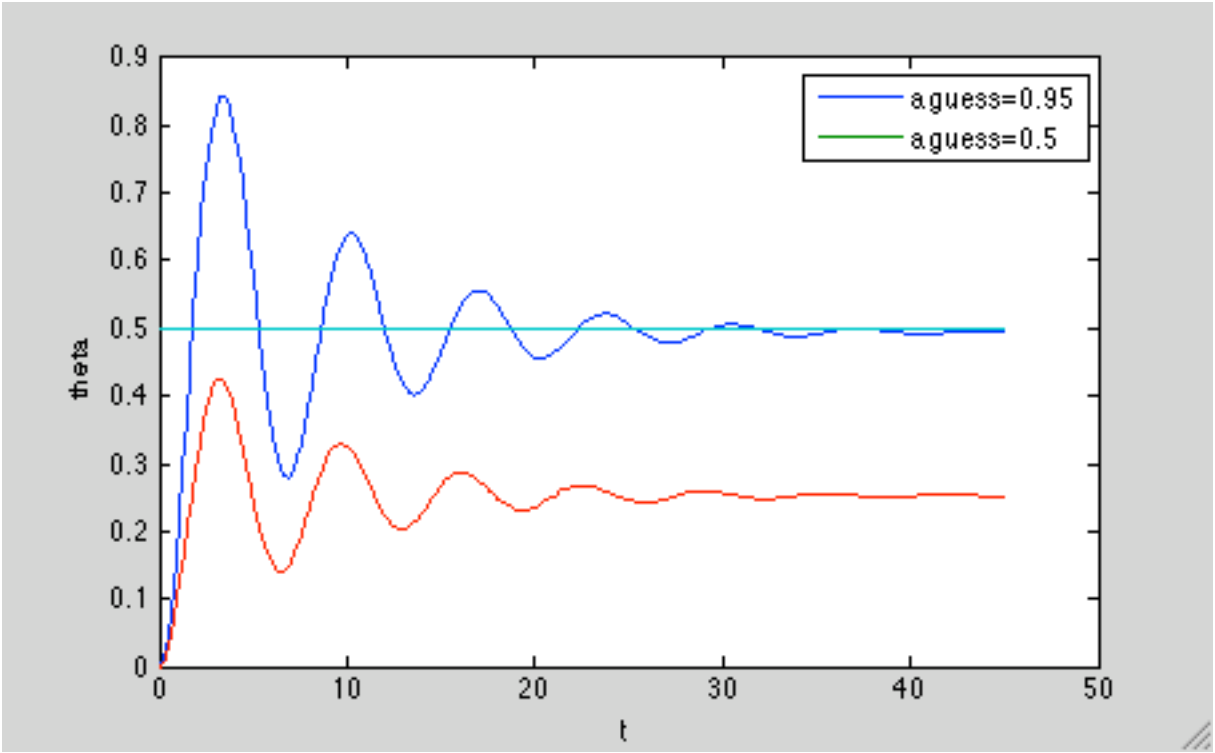
$$-a\sin\theta - b\omega - u \approx -a\theta - b\omega - u$$

$$\Rightarrow u = a\theta + b\omega$$

Since we want θ = r and ω = 0 at equilibrium we get

$$u = ar$$

Feedforward

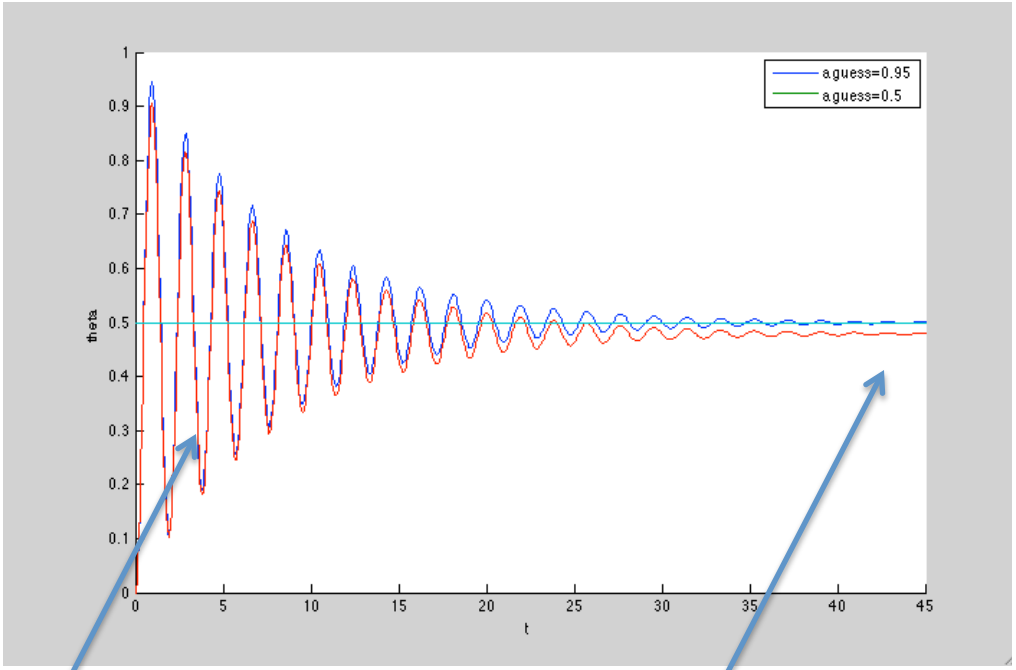



```
r=0.5;
aguess = 0.95;
uff = aguess * r;
f = @(t,x) [ x(2); -a*sin(x(1)) - b*x(2)
+ uff ];
[t,x] = ode45 ( f, [0,tf], [0,0] );
plot ( t, x(:,1), [0 tf], [r r] );

aguess=0.5;
uff = aguess * r;
f = @(t,x) [ x(2); -a*sin(x(1)) - b*x(2)
+ uff ];
[t,x] = ode45 ( f, [0,tf], [0,0] );
plot ( t, x(:,1), [0 tf], [r r] );
```

Proportional Feedback

$$e = r - \theta$$

$$u = u_{ff} + u_{fb} = ar + K_p e$$



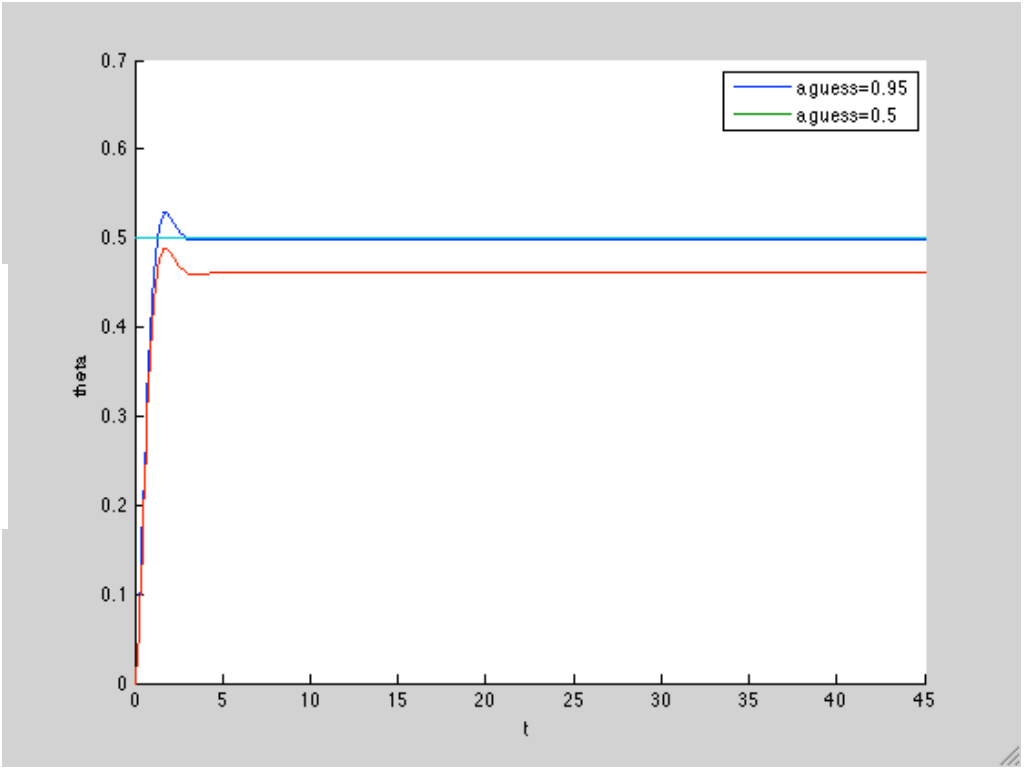
This “ring” is annoying.

Can make the steady state error small by turning up Kp.

Proportional-Derivative Control

$$u = u_{ff} + u_{fb} = ar + K_p e + K_d \dot{e}$$

$$\dot{e} = -\omega$$

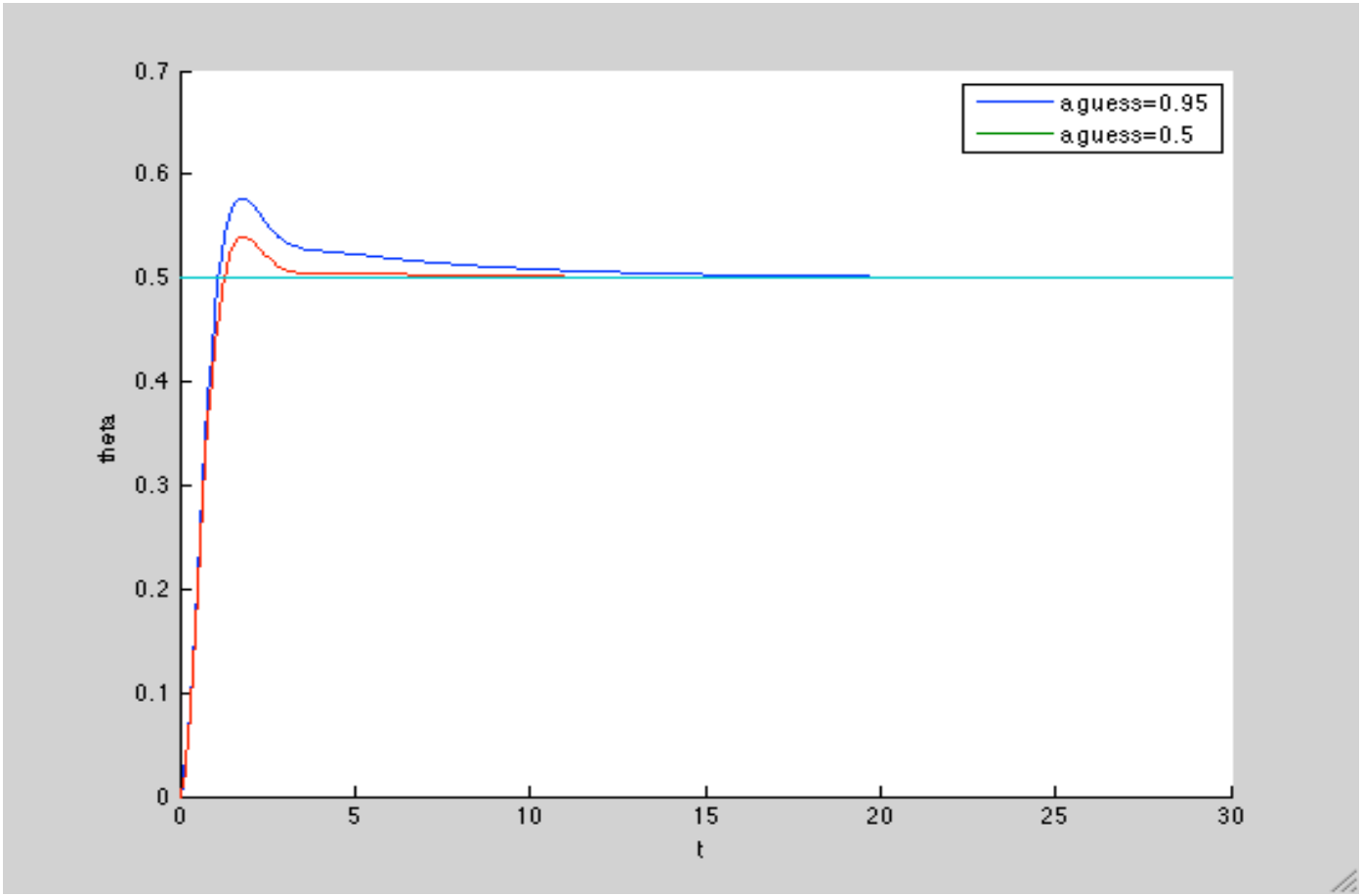


Integral Feedback

Define a new state z (maintained by the controller) and put

$$\dot{z} = e$$

$$u = u_{ff} + u_{fb} = ar + \underbrace{K_p e}_{\text{Proportional}} + \underbrace{K_d \dot{e}}_{\text{Derivative}} + \underbrace{K_I z}_{\text{Integral}}$$




```
aguess = 0.95;

uff = aguess * r;

f = @(t,x) [ x(2); -a*sin(x(1))-b*x(2)+uff+Kp*(r-x(1))-Kd*x(2)+Ki*x(3); r-x(1) ];

[t,x] = ode45 ( f, [0,tf], [0,0,0] );

plot ( t, x(:,1), [0 tf], [r r] );
```

feedforward

proportional

derivative

intergral

Integrator dynamics

Analysis

$$\begin{pmatrix}\dot{\theta}\\ \dot{\omega}\end{pmatrix} = \begin{pmatrix}\omega\\ -a\sin\theta - b\omega + u\end{pmatrix}$$

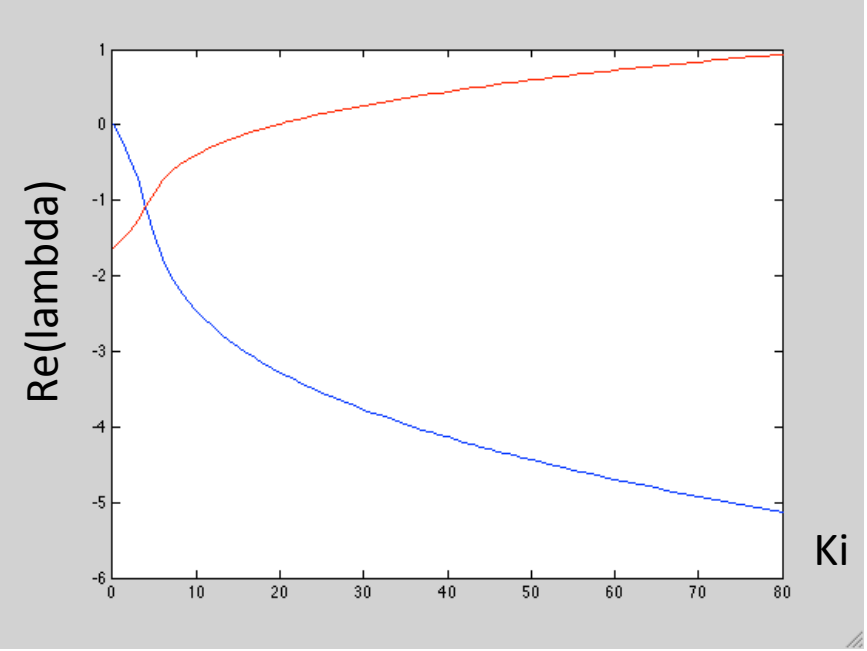
$$\sin\theta \approx \theta$$

$$u = u_{ff} + u_{fb} = ar + K_p e + K_d \dot{e} + K_I z$$

$$\begin{pmatrix}\dot{\theta}\\ \dot{\omega}\\ \dot{z}\end{pmatrix} = \begin{pmatrix}0 & 1 & 0\\ -a - K_p & -b - K_d & K_i\\ -1 & 0 & 0\end{pmatrix} \begin{pmatrix}\theta\\ \omega\\ z\end{pmatrix} + \begin{pmatrix}0\\ K_p\\ 1\end{pmatrix} r$$

```
syms a b Kp Kd Ki th om z r

A = [ 0 1 0 ; -a-Kp, -b-Kd, Ki; -1 0 0 ];
B = [ 0; Kp; 1 ];

% steady state analysis
ss=A\(-B*r)

% transient analysis
lam=eig(A);
subs (lam, {a,b,Ki,Kd,Kp}, {1,1/4,1,1,1} )
lb=simplify(subs (lam, {a,b,Kp,Kd}, {1,1/4,5,3} ));
plot ( 0:80, subs(lb(1,1),{Ki},{0:80} ), 0:80, subs(lb(2,1),{Ki},{0:80} ),
0:80, subs(lb(3,1),{Ki},{0:80} )   )
```

```
ss =

         r
         0
 1/Ki*r*a



ans =

  -0.6214
  -0.3143 + 1.2291i
  -0.3143 - 1.2291i
```



Simulink Demo