Topic 2: Cruise control example

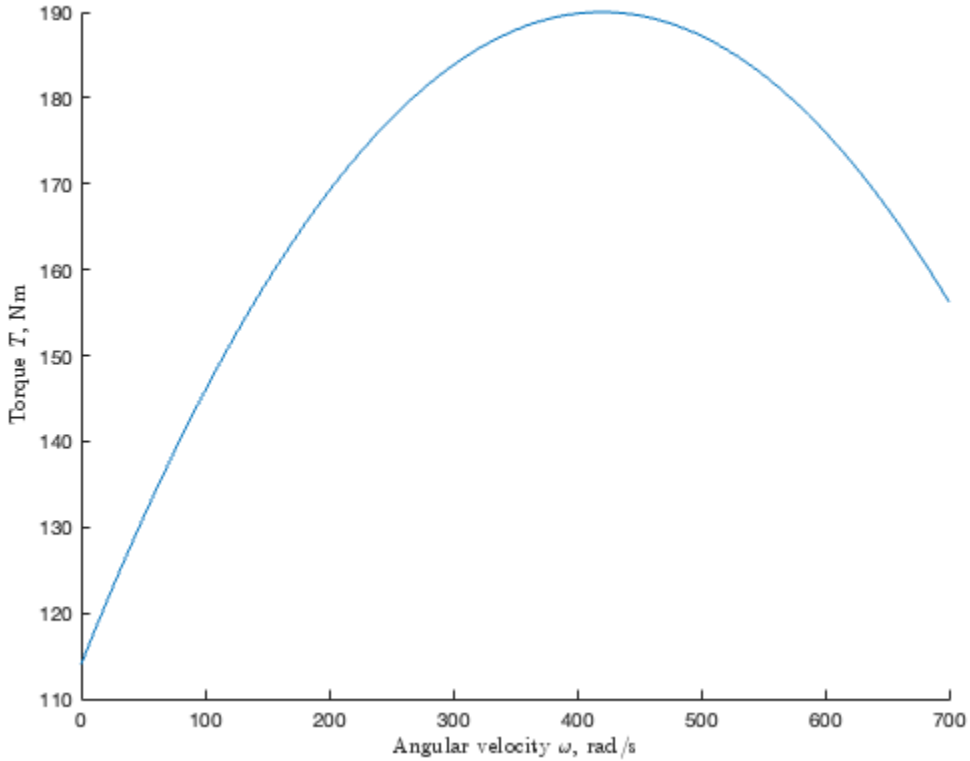
Table of Contents

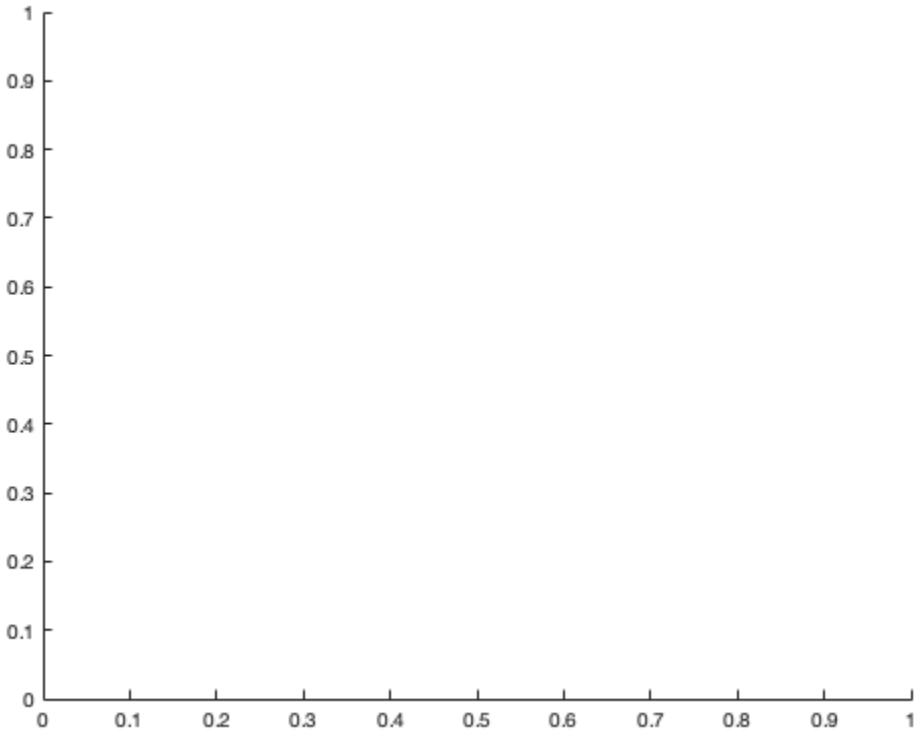
Opening

This code is an exact copy of that used to generate the supplementary results shown in Concept 2.4 (including saving the figures to PDF — a handy skill to pick up!).

Generally start worked example scripts like this with some initialisation steps. Clear the command window; close all figure windows; clear all variables.

```
clc
close all
clearvars
```


190
180
170
160
150
140
130
120
110
Torque T, Nm
0
100
200
300
400
500
600
700
Angular velocity ω, rad/s


1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

figsize.m

```
function figsize( w, h, u )
%FIGURESIZE Set a figure to a specific size
%
% When saving a figure as a PDF, it is necessary to set the
% figure size appropriately. This function sets the "paper size"
% sufficient that the figure is saved with a tight bounding box.
% It will also set the figure size on screen correspondingly.
%
% figuresize(width,height,units)
%  - sets the figure size in <units>
%  - <units> can be any of: (default 'cm')
%       'normalized','centimeters','inches','points','cm','in','pt','mm'

switch u
  case 'cm', u = 'centimeters';
  case 'in', u = 'inches';
  case 'pt', u = 'points';
  case 'mm', u = 'centimeters'; w = w/10; h = h/10;
end

set(gcf,'Units',u);
screenpos = get(gcf,'Position');
set(gcf,...
  'Position',[screenpos(1:2) w h],'PaperUnits',u,...
  'PaperPosition',[0 0 w h],'PaperSize',[w h]);

end
```

Modelling parameters

Rather than define local variables all over the place, use a structure to store parameters used throughout our modelling and simulation routines.

The `param.X` syntax can be used for setting fields within a **structure**, and where parameters are needed later the structure itself can be passed around. This allows parameters to be defined and controlled as a group and minimises the number of subfunction inputs needed.

```
% engine parameters:
param.wm = 420;
param.Tm = 190;
param.bb = 0.4;
param.alpha_n = [40, 25, 15, 12, 10];

% friction parameters:
param.m = 1600;
param.g = 9.81;
param.Cr = 0.01;

% drag parameters:
param.rho = 1.3;
param.Cd = 0.32;
```

```
param.A = 2.4;

% control gains:
param.kp = 0.5;
param.ki = 0.1;
```

Torques and forces

The first step in the modelling is to define the relationship between velocity, torque, and force. These relationships are plotted as a sanity check and visualisation tool.

```
function T = torque(ww,param)
  T = param.Tm*(1-param.bb*(ww/param.wm-1).^2);
end

function F = driveforce(v,u,gear,param)
  T = torque( param.alpha_n(gear)*v, param);
  F = param.alpha_n(gear)*u.*T;
end

wrange = linspace(0,700);
T = torque(wrange,param);

figure(1); clf; hold on
plot(wrange,T)
xlabel("Angular velocity $\omega$, rad/s",Interpreter="LaTeX")
ylabel("Torque $T$, Nm",Interpreter="LaTeX")

figure(2); clf; hold on
figsize(10,8,"cm")
grid on; box on
vrange = linspace(0,80);
u = 1.0;
for nn = 1:5
  F = driveforce(vrange,u,nn,param);
  plot(vrange,F/1000)
end
xlabel("Velocity $v$, m/s",Interpreter="LaTeX")
ylabel("Force $F$, kN",Interpreter="LaTeX")
ylim([0, 8])
legend("$n=1$","$n=2$","$n=3$","$n=4$","$n=5$",interpreter="LaTeX")

saveas(gcf,"pdf/car-fv.pdf")
```

Dynamics

Now the internal relationships are known, the next step is to define the car free-body-diagram dynamics. This is done in a modular fashion with functions to make them reusable later in the code, and swappable if we wanted to add more fidelity in the future.

We could plot families of curves (say) based on these definitions, but they are all reasonably straightforward so let's assume they are fine.

```
function Fg = fricforce(v, param)
  Fg = param.m * param.g * param.Cr * sign(v);
end

function Fg = gravforce(theta, param)
  Fg = param.m * param.g * sin(theta);
end

function Fa = dragforce(v, param)
  Fa = 0.5 * param.rho * param.A * param.Cd * v .* abs(v);
end

function dv = car_dynamics(v,u,gear,sl,param)
  m = param.m; % shorthand to save space in the following lines
  dv = + 1/m * driveforce(v,u,gear,param)...
       - 1/m * fricforce(v,param)...
       - 1/m * gravforce(sl,param)...
       - 1/m * dragforce(v,param);
end
```

ODE

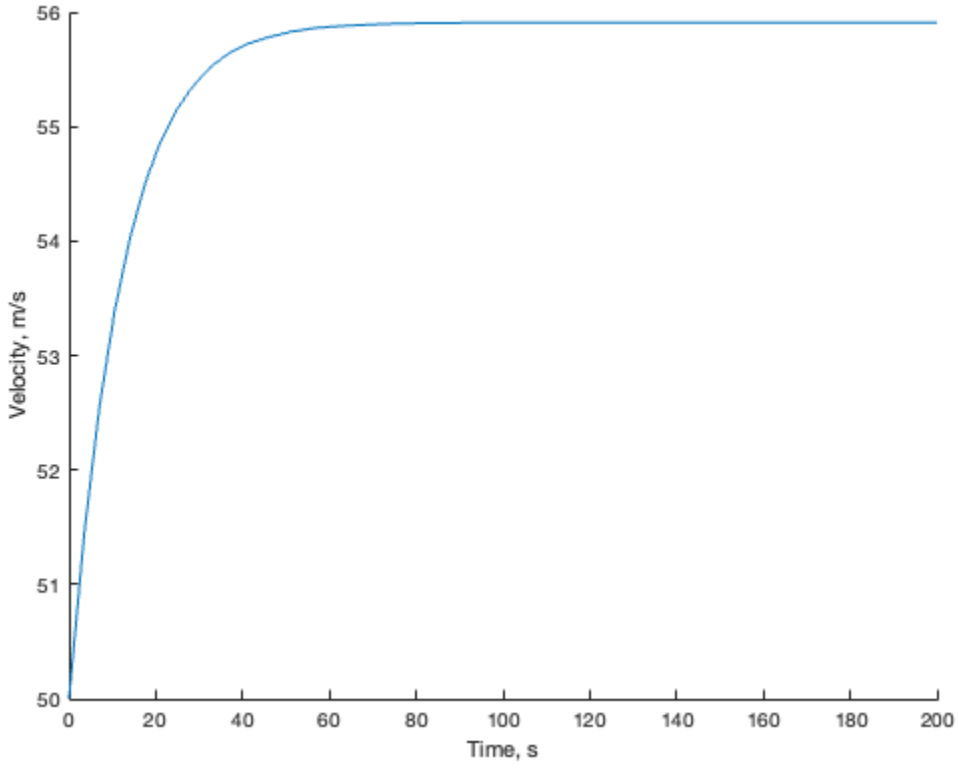
Going from dynamics to an ODE is straightforward once we have functions defined, the only "trick" is to have already made sure to represent the maths as cleanly as possible in the code.

You may be wondering why "gear" and "slope" are kept as function inputs rather than parameters. The answer to this is that we expect these might change dynamically with more complex simulations, whereas parameters are expected to remain constant.

This proof of concept shows that we can take the car dynamical equation and solve it numerically as an initial value problem and plot the solution of velocity versus time.

```
tmax = 200;
throttle = 1;
gear = 3;
slope = 0;
v0 = 50;
[t,x] = ode45(@(t,z) car_dynamics(z,throttle,gear,slope,param),[0,tmax],v0);

figure(10); clf; hold on
plot(t,x)
xlabel("Time, s")
ylabel("Velocity, m/s")
```

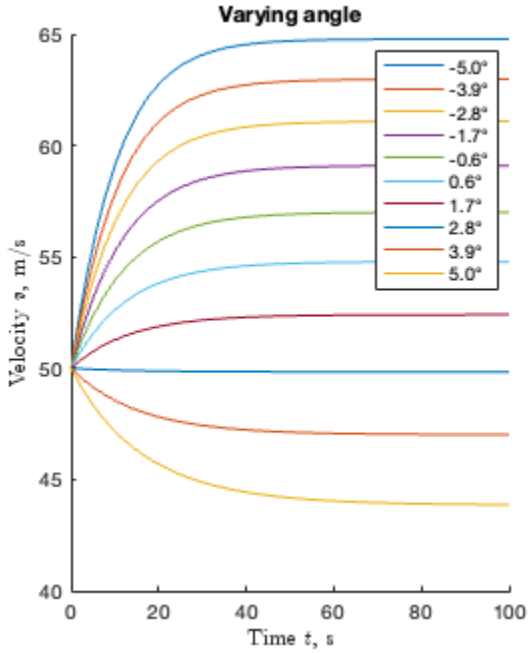


Plot families of curves to demonstrate the behaviour of the "open loop" car dynamics

```
figure(11); clf; hold on
figsize(10,12,"cm")
tmax = 100;
vrange = 45:2:65;
throttle = 1;
gear = 3;
slope = 0;
for v0 = vrange
  [t,x] = ode45(@(t,z) car_dynamics(z,throttle,gear,slope,param),[0,tmax],v0);
  h = plot(t,x);
  plot(0,v0,'.',Color=h.Color,MarkerSize=15)
end
xlabel("Time $t$, s",Interpreter="LaTeX")
ylabel("Velocity $v$, m/s",Interpreter="LaTeX")
title("Varying initial velocity")
saveas(gcf,"pdf/cruise-v0.pdf")

figure(12); clf; hold on
figsize(10,12,"cm")
tmax = 100;
trange = linspace(deg2rad(-5),deg2rad(5),10);
```

```
v0 = 50;
for t0 = trange
  [t,x] = ode45(@(t,z) car_dynamics(z,throttle,gear,t0,param),[0,tmax],v0);
  h = plot(t,x);
end
legend(arrayfun(@(t) sprintf("%1.1f°",rad2deg(t)),trange))
xlabel("Time $t$, s",Interpreter="LaTeX")
ylabel("Velocity $v$, m/s",Interpreter="LaTeX")
title("Varying angle")
saveas(gcf,"pdf/cruise-t0.pdf")
```

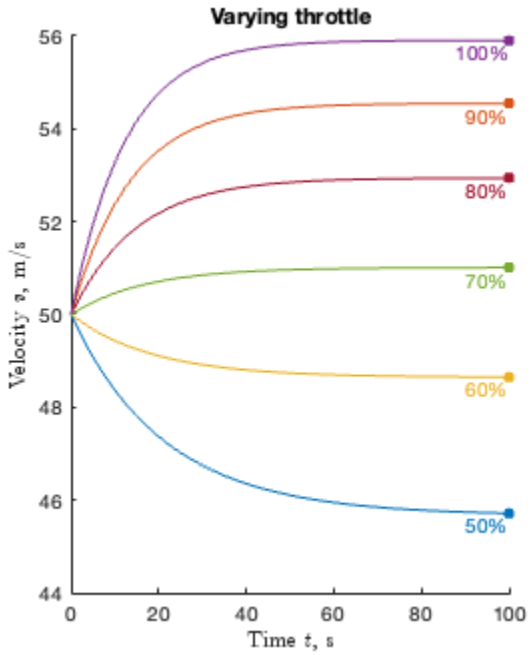


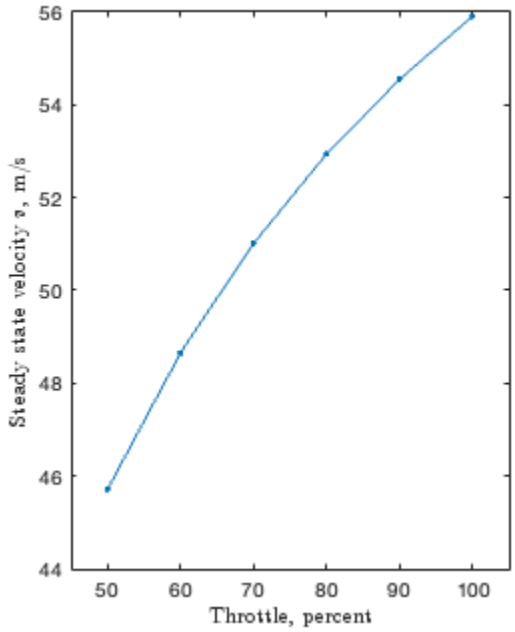
Similarly, varying throttle

```
figure(13); clf; hold on
figsize(10,12,"cm")
tmax = 100;
urange = linspace(0.5,1,6);
v0 = 50;
gear = 3;
slope = 0;
vf = nan(size(urange));
for ii = 1:numel(urange)
  u = urange(ii);
  [t,x] = ode45(@(t,z) car_dynamics(z,u,gear,slope,param),[0,tmax],v0);
  h = plot(t,x);
  vf(ii) = x(end);
  plot(t(end),x(end),'.',Color=h.Color,MarkerSize=15)
  text(t(end),x(end),sprintf("%2.0f%
%",100*u),HorizontalAlignment="Right",VerticalAlignment="Top",Color=h.Color)
end
xlabel("Time $t$, s",Interpreter="LaTeX")
```

```
ylabel("Velocity $v$, m/s",Interpreter="LaTeX")
title("Varying throttle")
saveas(gcf,"pdf/cruise-u.pdf")

figure(14);
figsize(10,12,"cm")
plot(100*urange,vf,'.-')
xlim([45 105])
xlabel("Throttle, percent",Interpreter="LaTeX")
ylabel("Steady state velocity $v$, m/s",Interpreter="LaTeX")
saveas(gcf,"pdf/cruise-uv.pdf")
```



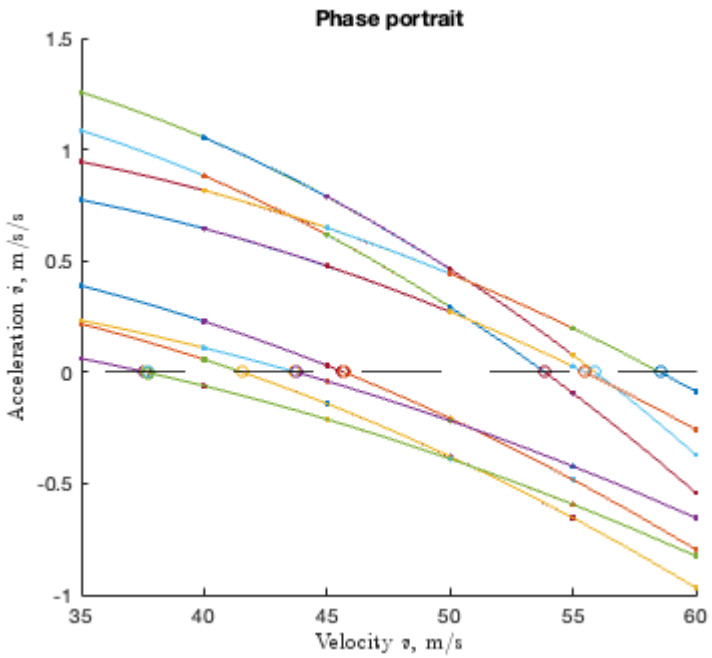


Now attempting to plot a phase portrait

Because this is a 1DOF system, the dynamics require `v` and `dv` to be coupled -- they cannot take independent parameters. This means we can't plot a traditional phase portrait with arbitrary initial conditions on a plane.

Instead we plot some families of curves to illustrate the sensitivity of the system to the context.

```
figure(15); clf; hold on
figsize(14,12,"cm")
tmax = 200;
vrange = 35:5:60;
for gear = [3 4]
  for sl = [0 deg2rad(1)]
    for th = [0.5 1]
      for v0 = vrange
        [~,x] = ode45(@(t,z) car_dynamics(z,th,gear,sl,param),[0,tmax],v0);
        dv = car_dynamics(x,th,gear,sl,param);
        h = plot(x,dv);
        plot(x(1),dv(1),".",Color=h.Color)
        plot(x(end),dv(end),"o",Color=h.Color)
      end
    end
  end
end
yline(0,"k--")
xlabel("Velocity $v$, m/s",Interpreter="LaTeX")
ylabel("Acceleration $\dot v$, m/s/s",Interpreter="LaTeX")
title("Phase portrait")
saveas(gcf,"pdf/cruise-phase.pdf")
```



Adding a PI cruise control system

When we have more complex ODEs, the structure of the code generally works best when considered in terms of a generalised state vector (in this case `q`). I like to break apart `q` to calculate the individual derivative terms, and then reconstruct `dq` from the components. This takes a few more lines of code but the structured nature of it helps avoid common mistakes.

Once again, the more closely the code can mirror the physics and maths the better.

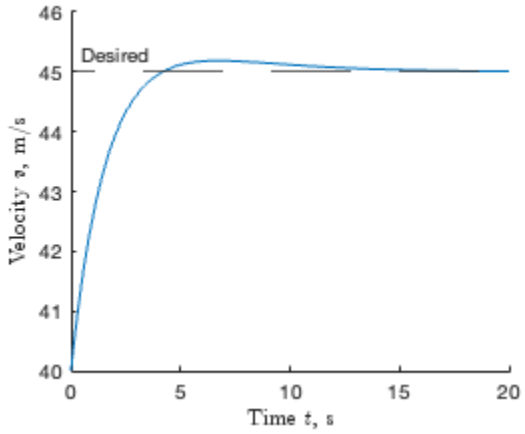
```
function dq = cruise_ode(q,vr,gear,slope,param)
  v = q(1); z = q(2);
  u = param.kp * (vr-v) + param.ki * z;
  dv = car_dynamics(v,u,gear,slope,param);
  dz = vr - v;
  dq = [dv; dz];
end

tmax = 20;
tspan = linspace(0,tmax,1e8);
options = odeset(RelTol=1e-6);
gear = 3;
slope = 0;
v0 = 40;
vr = 45;
[t,z] = ode45(@(t,z) cruise_ode(z,vr,gear,slope,param),[0,tmax],
[v0,0],options);

figure(10); clf; hold on
figsize(10,8,"cm")

yline(vr,"--","Desired","LabelHorizontalAlignment","left")
```

```
plot(t,z(:,1))
xlabel("Time $t$, s",Interpreter="LaTeX")
ylabel("Velocity $v$, m/s",Interpreter="LaTeX")
saveas(gcf,"pdf/cruise-step.pdf")
```



Cruise control phase portrait

Now we have a 2DOF system the phase portrait is slightly more interesting. We expect to see convergence on the reference set point; in the example show two set points to see this is the case.

```
figure(15); clf; hold on
figsize(9,9,"cm")
tmax = 200;
vrange = 40:2.5:55;
kp = 0.5; ki = 0.1;
for vr = [45 50]
  for gear = 3
    for sl = deg2rad(0)
      for v0 = vrange
        tspan = linspace(0,tmax,1e8);
        options = odeset(RelTol=1e-6);
        [t,x] = ode45(@(t,z) cruise_ode(z,vr,gear,sl,param),[0,tmax],
[v0,0],options);
        v = x(:,1); z = x(:,2);
        u = kp*(vr-v)+ki*z;
        dv = car_dynamics(x(:,1),u,gear,sl,param);
        h = plot(v,dv);
        plot(v(1),dv(1),".",Color=h.Color)
        plot(v(end),dv(end),"o",Color=h.Color)
      end
    end
  end
end
xline(45,"k--","$v_r=45$ m/s",Interpreter="LaTeX")
xline(50,"k--","$v_r=50$ m/s",Interpreter="LaTeX")
yline(0,"k--","Steady state",Interpreter="LaTeX")
xlabel("Velocity $v$, m/s",Interpreter="LaTeX")
ylabel("Acceleration $\dot v$, m/s/s",Interpreter="LaTeX")
```

```
title("Phase portrait")
saveas(gcf,"pdf/cruise-portrait.pdf")
```

Published with MATLAB® R2024b