Appendices: MATLAB® Examples

These appendices list, with suitable explanations, the MATLAB® programs or 'script files' that are used in Chapter 7, where electric vehicle modelling is considered. In most cases the simulations can be performed nearly as easily with programs such as EXCEL®. However, in some cases MATLAB® is noticeably better. It is also certainly much easier to explain what has been done, and to relate it to the underlying mathematics with MATLAB®.

All the example programs here will work with the student edition of MATLAB®, which can be obtained at very reasonable cost.

Appendix 1: Performance Simulation of the GM EV1

In Section 7.3.3 we gave the MATLAB® script file for simulating the acceleration from a standing start of an electric scooter. In Section 7.3.4 we turned our attention to the groundbreaking GM EV1, and developed some equations for its performance. The script file for modelling the acceleration of that vehicle is quite similar to that for the scooter, and is given below.

```
% GMEV1  **********************
% Simulates the WOT test of the GM EV1 electric car.
t=linspace(0,15,151);  % 0 to 15 seconds, in 0.1 sec. steps
v=zeros(1,151);        % 151 readings of velocity
dT=0.1;            % 0.1 second time step
% In this case there are three phases to the acceleration, as
% explained in the text.
for n=1:150
   if v(n)< 19.8
      % Equation 7.21
      v(n+1) = v(n) + dT*(3.11 + (0.000137*(v(n)^2)));
   elseif v(n) > 35.8
      % Controller stops any more speed increase
      v(n+1) = v(n);
   else
```

Electric Vehicle Technology Explained James Larminie and John Lowry
 ISBN: 0-470-85163-5

```
      % Equation 7.22
      v(n+1) = v(n) + dT * ((62.1/v(n)) - 0.046 - (0.000137*(v(n)^2)));
   end;
end;
v=v.*3.6;  %Multiply all v values by 3.6 to
           %convert to kph
plot(t,v);
xlabel('Time/seconds');
ylabel('velocity/kph');
title('Full power (WOT) acceleration of GM EV1 electric car');
```

The output of this script file can be seen in Figure 7.6.

Appendix 2: Importing and Creating Driving Cycles

An important part of range modelling is the use of standard driving cycles, as explained in Section 7.4.1. When using MATLAB® these driving cycles must be set up as one-dimensional arrays which contain successive values of velocity. There are many ways of doing this, including methods involving directly reading files. However, the easiest way to do the task is to simply set up a matrix by cutting and pasting in the data as text into a file, as follows.

Let us imagine a very simple driving cycle involving increasing the speed at $1\,\mathrm{ms}^{-1}$ each second, up to $3\,\mathrm{ms}^{-1}$, and then slowing down again. This would be set up as a MATLAB® script file like this:

```
V = [ 0
1
2
3
2
1
0];
```

This would be saved under a name such as SIMPLE.m and would be called from the simulation program.

For real cycles the values would be obtained from a file from the WWW, and then cut and pasted, making a file like the one above, except that it would be several hundred lines long. Each line would just have one number. Such files should be saved under suitable names, and will be called early in the simulation program as shown in the examples that follow. Some of the most common, including those discussed in Chapter 7, have been made available as M-files on the website associated with this book.[1]

Because each cycle will have a different number of values, the simulation program will need to find how many numbers there are, so that it cycles through the correct number of points. This is done using the MATLAB® function `length()`.

Once the velocity values are loaded into an array, care must be taken with the units, as all simulation work must be carried out in ms^{-1}. Thus, in some of the programs that follow all the values in the matrix `V` are divided by 3.6, to convert from $\mathrm{km.h}^{-1}$ to ms^{-1}.

[1] www.wileyeurope.com/electricvehicles.

The first three lines of a range simulation will often contain the lines:

```
sfuds; % Get the velocity values.
N=length(V); % Find out how many readings
V=V./3.6;
```

As we saw in Section 7.4.1, not all driving cycles are a simple string of values, but are calculated from a series of values of accelerations, followed by so many seconds at such a speed, and so on. This sort of cycle can reasonably easily be created, and an example is given below. It creates an ECE-47 driving cycles for the scooter shown in Figure 7.3. The velocity/time graph produced by this program (or script file) was shown in Figure 7.12.

```
%  ECE_47    *****************
% This file is for constructing the velocity
% profile of the ECE-47 cycle for an
% electric scooter.

% The cycles last 110 seconds, so we set up a
% one dimensional array for 111 readings.
V=zeros(1,111);
% The first phase is a 50 second acceleration phase.
% We use exactly the same program as in Section 7.3.2,
% except that the graph drawing elements are removed,
% and the conversion to kph.
Scoota;
% "Scoota" finds values of velocity every 0.1 seconds,
% so we need to decimate these readings.
for n=1:51
   V(n)=vel(((n-1)*10)+1);
end;
%The velocity is then reduced to 5.56 m/sec over the
%next 15 seconds.
decel=(V(51)-5.56)/15;
for n=52:65
   V(n)=V(n-1) - decel;
end
%This velocity is then maintained for 35 seconds
for n=66:101
   V(n)=5.56;
end;
%The scooter is then stopped over 8 seconds
decel=5.56/8;
for n=102:108
   V(n)=V(n-1)-decel;
end;
V(109)=0;
% The zero speed is then held for a further 2 seconds.
V(110)=0;
V(111)=0;
% *****************
```

In order to produce diagrams such as Figure 7.12 plot commands are added at the end of the file. However, when doing range and other simulations, as outlined in Section 7.4, these should not be used.

In all this work, and the examples that follow, it is important to note that with MATLAB® variables are normally 'global'. This means that a variable or array created in one file can be used by another.

Appendix 3: Simulating One Cycle

The simulation of the range of a vehicle involves the continuous running of driving cycles or schedules until there is no more energy left. The script file below is for the simulation of just one cycle. It is saved under the name `one_cycle.m`. It is called by the range simulation programs that follow. Broadly, it follows the method outlined in Section 7.4.2, and the flowchart of Figure 7.14.

This file requires the following:

- An array of velocity values V must have been created, corresponding to the driving cycle, as outlined in the previous section.
- The value of N must have been found, as also explained in the previous section.
- Two MATLAB® functions, `open_circuit_voltage_LA` and `open_circuit_voltage_NC` must have been created. These functions have been outlined and explained in Chapter 2.
- All the variables such as `mass`, `area`, `Cd`, etc., must have been created by the MATLAB® file that uses this file. Rather than listing them again here, refer to either of the programs in the two following sections.

```
% ******************************
%  ONE CYCLE
% This script file performs one cycle, of any
% drive cycle of N points with any vehicle and
% for lead acid or NiCad batteries.
% All the appropriate variables must be set
% by the calling program.
% ******************************

for C=2:N
   accel=V(C) - V(C-1);
   Fad = 0.5 * 1.25 * area * Cd * V(C)^2; % Equ. 7.2
   Fhc = 0;             % Eq. 7.3, assume flat
   Fla = 1.05 * mass * accel;
      % The mass is increased modestly to compensate for
   % the fact that we have excluded the moment of inertia
   Pte = (Frr + Fad + Fhc + Fla)*V(C); %Equ 7.9 & 7.23
   omega = Gratio * V(C);
   if omega == 0 % Stationary
      Pte=0;
      Pmot_in=0; % No power into motor
      Torque=0;
      eff_mot=0.5; % Dummy value, to make sure not zero.
   elseif omega > 0 % Moving
```

```
    if Pte < 0
       Pte = Regen_ratio * Pte;  % Reduce the power if
    end;     % braking, as not all will be by the motor.
    % We now calculate the output power of the motor,
    % Which is different from that at the wheels, because
    % of transmission losses.
    if Pte>=0
       Pmot_out=Pte/G_eff; % Motor power> shaft power
    elseif Pte<0
       Pmot_out=Pte * G_eff; % Motor power diminished
    end;         % if engine braking.

    Torque=Pmot_out/omega; % Basic equation, P=T * omega
     if Torque>0 % Now use equation 7.23
        eff_mot=(Torque*omega)/((Torque*omega)+((Torque^2)*kc)+
        (omega*ki)+((omega^3)*kw)+ConL);
     elseif Torque<0
      eff_mot=(-Torque*omega)/((-Torque*omega) +
      ((Torque^2)*kc)+(omega*ki)+((omega^3)*kw)+ConL);
    end;
    if Pmot_out > = 0
       Pmot_in = Pmot_out/eff_mot;  % Equ 7.23
    elseif Pmot_out < 0
       Pmot_in = Pmot_out * eff_mot;
    end;
end;

 Pbat = Pmot_in + Pac;  % Equation 7.26

 if bat_type=='NC'
    E=open_circuit_voltage_NC(DoD(C-1),NoCells);
 elseif bat_type=='LA'
    E=open_circuit_voltage_LA(DoD(C-1),NoCells);
 else
    error('Invalid battery type');
end;

if Pbat > 0 % Use Equ. 2.20
   I = (E - ((E*E) - (4*Rin*Pbat))^0.5)/(2*Rin);
   CR(C) = CR(C-1) +((I^k)/3600);   %Equation 2.18
 elseif Pbat==0
      I=0;
 elseif Pbat <0
   % Regenerative braking. Use Equ. 2.22, and
   % double the internal resistance.
   Pbat = - 1 * Pbat;
   I = (-E + (E*E + (4*2*Rin*Pbat))^0.5)/(2*2*Rin);
   CR(C) = CR(C-1) - (I/3600);   %Equation 2.23
end;

DoD(C) = CR(C)/PeuCap;   %Equation 2.19
if DoD(C)>1
   DoD(C) =1;
end
```

```
   % Since we are taking one second time intervals,
   % the distance traveled in metres is the same
   % as the velocity. Divide by 1000 for km.
   D(C) = D(C-1) + (V(C)/1000);
   XDATA(C)=C;       % See Section 7.4.4 for the use
   YDATA(C)=eff_mot; % of these two arrays.
end;
% Now return to calling program.
```

Appendix 4: Range Simulation of the GM EV1 Electric Car

In Section 7.4.2.3 the simulation of this important vehicle was discussed. Figure 7.15 gives an example output from a range simulation program. The MATLAB® script file for this is shown below. Notice that it calls several of the MATLAB® files we have already described. However, it should be noted how this program sets up, and often gives values to, the variables used by the program `one_cycle` described in the preceding section.

```
% Simulation of the GM EV1 running the SFUDS
% driving cycle. This simulation is for range
% measurement. The run continues until the
% battery depth of discharge > 90%

sfuds; % Get the velocity values, they are in
       % an array V.
N=length(V); % Find out how many readings
%Divide all velocities by 3.6, to convert to m/sec
V=V./3.6;

% First we set up the vehicle data.
mass = 1540 ; % Vehicle mass+ two 70 kg passengers.
area = 1.8;   % Frontal area in square metres
Cd = 0.19;    % Drag coefficient
Gratio = 37;  % Gearing ratio, = G/r
G_eff = 0.95;   % Transmission efficiency
Regen_ratio = 0.5; % This sets the proportion of the
            % braking that is done regeneratively
            % using the motor.
bat_type='LA'; % Lead acid battery
NoCells=156;   % 26 of 6 cell (12 Volt) batteries.
Capacity=60;   % 60 Ah batteries. This is
           % assumed to be the 10 hour rate capacity
k=1.12;   % Peukert coefficient, typical for good lead acid
Pac=250;  % Average power of accessories.

% These are the constants for the motor efficiency
% equation, 7.23
kc=0.3;   % For copper losses
ki=0.01;  % For iron losses
kw=0.000005;  % For windage losses
ConL=600;     % For constant electronics losses
```

```
% Some constants which are calculated.
Frr=0.0048 * mass * 9.8;  % Equation 7.1
Rin= (0.022/Capacity)*NoCells;   % Int. res, Equ. 2.2
Rin = Rin + 0.05; % Add a little to make allowance for
                  % connecting leads.
PeuCap= ((Capacity/10)^k)*10; % See equation 2.12
% Set up arrays for storing data for battery,
% and distance traveled. All set to zero at start.
% These first arrays are for storing the values at
% the end of each cycle.
% We shall assume that no more than 100 of any cycle is
% completed. (If there are, an error message will be
% displayed, and we can adjust this number.)
DoD_end = zeros(1,100);
CR_end = zeros(1,100);
D_end = zeros(1,100);

% We now need similar arrays for use within each cycle.
DoD=zeros(1,N);  % Depth of discharge, as in Chap. 2
CR=zeros(1,N);   % Charge removed from battery, Peukert
                 % corrected, as in Chap 2.
D=zeros(1,N);    % Record of distance traveled in km.

CY=1;
% CY controls the outer loop, and counts the number
% of cycles completed. We want to keep cycling till the
% battery is flat. This we define as being more than
% 90% discharged. That is, DoD_end > 0.9
% We also use the variable XX to monitor the discharge,
% and to stop the loop going too far.
DD=0;  % Initially zero.

while DD < 0.9
%Beginning of a cycle.************
% Call the script file that performs one
% complete cycle.

one_cycle;

% One complete cycle done.
% Now update the end of cycle values.
DoD_end(CY) = DoD(N);
CR_end(CY) = CR(N);
D_end(CY) = D(N);

% Now reset the values of these "inner" arrays
% ready for the next cycle. They should start
% where they left off.

DoD(1)=DoD(N); CR(1)=CR(N);D(1)=D(N);
DD=DoD_end(CY)  % Update state of discharge
%END OF ONE CYCLE ***************
CY = CY +1;
end;
```