Appendices: MATLAB® Examples

These appendices list, with suitable explanations, the MATLAB programs or 'script files' that are used in Chapter 8, where electric vehicle design 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 8.3.2 we gave the MATLAB script file for simulating the acceleration from a standing start of an electric scooter. In Section 8.3.3 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 second 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 (8.21)
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
```

```
% Equation (8.22)
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 8.6.

Appendix 2: Importing and Creating Driving Cycles

An important part of range modelling is the use of standard driving cycles, as explained in Section 8.4.1. 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 simply to 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 $1\,\mathrm{m\,s^{-1}}$ each second, up to $3\,\mathrm{m\,s^{-1}}$, 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 World Wide Web, 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 8, have been made available as M-files on the web site 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()`.

[1] www.wileyeurope.com/electricvehicles.

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 metres per second. Thus, in some of the programs that follow all the values in the matrix `V` are divided by 3.6, to convert from kilometres per hour to metres per second.

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 8.4.1, 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 be created reasonably easily, and an example is given below. It creates an ECE-47 driving cycle for the scooter shown in Figure 8.3. The velocity/time graph produced by this program (or script file) was shown in Figure 8.12.

```
% 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 8.3.2,
% 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/s 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 8.12 plot commands are added at the end of the file. However, when doing range and other simulations, as outlined in Section 8.4, 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 8.4.2 and the flowchart of Figure 8.14.

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 3.
- All the variables such as `mass`, `area`, `Cd`, and so on, 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; % Equation (8.2)
```

```
Fhc = 0; % Equation (8.3), 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); %Equations (8.9) & (8.23)**
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 (8.23)**
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; % Equation (8.23)**
elseif Pmot out < 0
Pmot in = Pmot out * eff mot;
end;
end;
Pbat = Pmot in + Pac; % Equation (8.26)**
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 Equation (3.26)**
I = (E - ((E*E) - (4*Rin*Pbat))^0.5)/(2*Rin);
CR(C) = CR(C-1) +((I^k)/3600); %Equation (3.24)**
```

```
elseif Pbat==0
I=0;
elseif Pbat <0
% Regenerative braking. Use Equation (3.28)**, 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 (3.29)**
end;
DoD(C) = CR(C)/PeuCap; %Equation (2.19)**
if DoD(C)>1
DoD(C) =1;
end
% Since we are taking 1 second time intervals,
% the distance travelled 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 8.4.4 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 8.4.2.3 the simulation of this important vehicle was discussed. Figure 8.15 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/s
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, (8.25)
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 (8.1)
Rin= (0.022/Capacity)*NoCells; % Int. res, Equation (3.2)
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 travelled. 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 Chapter 3.
CR=zeros(1,N); % Charge removed from battery, Peukert
% corrected, as in Chapter 3.
D=zeros(1,N); % Record of distance travelled 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;
plot(D end,DoD end,'k+');
ylabel('Depth of discharge');
xlabel('Distance traveled**/km');
```

The plot lines at the end of the program produce a graph such as in Figure 8.15. This graph has two sets of values. This is achieved by running the program above a second time, using the MATLAB `hold on` command. The second running was with much a higher value (800) for the average accessory power P_{ac}, and a slightly higher value (1.16) for the Peukert coefficient.

Appendix 5: Electric Scooter Range Modelling

By way of another example, the MATLAB script file below is for the range modelling of an electric scooter. The program is very similar, except that almost all the variables are different, and a different driving cycle is used. This shows how easy it is to change the system variables to simulate a different vehicle.

```
% Simulation of the electric scooter running
% the ECE-47 driving cycle. This simulation is
% for range measurement. The run continues until
% the battery depth of discharge > 90%.
ECE 47; % Get the velocity values, they are in
% an array V, and in m/s.
N=length(V); % Find out how many readings
% First we set up the vehicle data.
mass = 185 ; % Scooter + one 70 kg passenger.
area = 0.6; % Frontal area in square metres
Cd = 0.75; % Drag coefficient
Gratio = 2/0.21; % Gearing ratio, = G/r
G eff = 0.97; % Transmission efficiency
Regen ratio = 0.5; %This sets the proportion of the
% braking that is done regeneratively
% using the motor.
bat type='NC'; % NiCAD battery.
```

```
NoCells=15; % 3 of 5 cell (6 Volt) batteries.
Capacity=100; % 100 Ah batteries. This is
% assumed to be the 5 hour rate capacity
k=1.05; % Peukert coefficient, typical for NiCad.
Pac=50; % Average power of accessories.
kc=1.5; % For copper losses
ki=0.1; % For iron losses
kw=0.00001; % For windage losses
ConL=20; % For constant motor losses
% Some constants which are calculated.
Frr=0.007 * mass * 9.8; % Equation (8.1)
Rin = (0.06/Capacity)*NoCells; % Int. resistance, Equation (3.2)
Rin = Rin + 0.004; %Increase int. resistance to allow for
% the connecting cables.
PeuCap = ((Capacity/5)^k)*5 % See Equation (3.18)
% Set up arrays for storing data for battery,
% and distance travelled. 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 Chapter 3.
CR=zeros(1,N); % Charge removed from battery, Peukert
% corrected, as in Chapter 3.
D=zeros(1,N); % Record of distance travelled in km.
XDATA=zeros(1,N);
YDATA=zeros(1,N);
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.************
one cycle;
% **********
% 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;
plot(XDATA,YDATA,'k+');
```

Notice that the last line plots data collected during one cycle, as explained in Section 8.4.4. Graphs such as Figures 8.16 and 8.17 were produced in this way. If we wish to find the range to exactly 80% discharged, then the `while DD < 0.9`; line is changed to `while DD < 0.8`;. The following line is added to the end of the program in place of the plot command.

```
Range = D(N)*0.8/DoD(N)
```

The lack of a semicolon at the end of the line means that the result of the calculation will be printed, without the need for any further command. Results such as those in Table 8.3 were obtained this way.

Appendix 6: Fuel Cell Range Simulation

In Section 8.4.5 the question of the simulation of vehicle range simulation was discussed in relation to fuel cells. In the case of systems with fuel reformers it was pointed out that such simulations are highly complex. However, if the hydrogen fuel is stored as hydrogen, and we assume (not unreasonably) that the fuel cell has more or less constant efficiency, then the simulation is reasonably simple. An example, which is explained in Section 8.4.5, is given below:

```
% Simulation of a GM EV1 modified to incorporate
% a fuel cell instead of the batteries, as outlined
% in Section 8.4.5.
% All references to batteries can be removed!
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/s
V=V./3.6;
% First we set up the vehicle data.
mass = 1206 ; % Vehicle mass + two 70 kg passengers.
% See Section 8.4.5
area = 1.8; % Frontal area in square metres
Cd = 0.19; % Drag coefficient
Gratio = 37; % Gearing ratio, = G/r
```

```
Pac=2000; % Average power of accessories. Much larger,
% as the fuel cell needs a fairly complex controller.
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 (8.1)
% Set up arrays for storing data.
% Rather simpler, as hydrogen mass left
% and distance travelled is all that is needed.
% This first array is for storing the values at
% the end of each cycle.
% We need many more cycles now, as the range
% will be longer. We will allow for 800.
D end=zeros(1,800);
H2mass end = zeros(1,800);
H2mass end(1) =8.5; % See text; 8.5 kg at start.
% We now need a similar array for use within each cycle.
D=zeros(1,N);
H2mass=zeros(1,N); % Depth of discharge, as in Chapter 3
H2mass(1)=8.5;
CY=1;
% CY defines the outer loop, and counts the number
% of cycles completed. We want to keep cycling till the
% the mass of hydrogen falls to 1.7 kg, as explained in
% Section 8.4.5.
% We use the variable MH to monitor the discharge,
% and to stop the loop going too far.
MH=8.5; % Initially full, 8.5 kg
while MH > 1.7
%Beginning of a cycle.************
for C=2:N
accel=V(C) - V(C-1);
Fad = 0.5 * 1.25 * area * Cd * V(C)^2; % Equation (8.2)
Fhc = 0; % Equation (8.3), assume flat
Fla = 1.01 * 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); %Equations (8.9) & (7.23)
omega = Gratio * V(C);
if omega == 0
Pte=0;
Pmot=0;
Torque=0;
elseif omega > 0
Torque=Pte/omega; % Basic equation, P = T * ω
if Torque>=0
```

```
% Now Equation (8.23)
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 Pte > = 0
Pmot = Pte/(0.9 * eff mot); % Equation (8.23)
elseif Pte < 0
% No regenerative braking
Pmot = 0;
end;
end;
Pfc = Pmot + Pac;
H2used = 2.1E-8 * Pfc; % Equation (8.29)
H2mass(C) = H2mass(C-1) - H2used; %Equation (7.29) gives
% the rate of use of hydrogen in kg per second,
% so it is the same as the H2 used in 1 second.
% Since we are taking 1 second time intervals,
% the distance travelled in metres is the same
% as the velocity. Divide by 1000 for km.
D(C) = D(C-1) + (V(C)/1000);
end;
% Now update the end of cycle values.
H2mass end(CY) = H2mass(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.
H2mass(1)=H2mass(N); D(1)=D(N);
MH=H2mass end(CY) % Update state of discharge
%END OF ONE CYCLE ***************
CY = CY +1;
end;
% Print the range.
Range = D(N)
```

Appendix 7: Motor Efficiency Plots

In Chapter 8 the question of efficiency plots of electric motors was addressed. An example was given in Figure 8.7. It is very useful to be able to print this sort of diagram, to see the operating range of electric motors, and where they operate most efficiently. Furthermore, this can be done very effectively and quickly with MATLAB. The script file is given below:

```
% A program for plotting efficiency contours for
% electric motors.
% The x axis corresponds to motor speed (w),
% and the y axis to torque T.
% First, set up arrays for range.
x=linspace(1,180);% speed, N.B. rad/s NOT rpm
y=linspace(1,40); % 0 to 40 N.m
% Allocate motor loss constants.
kc=1.5; % For copper losses
ki=0.1; % For iron losses
kw=0.00001; % For windage losses
ConL=20; % For constant motor losses
% Now make mesh
[X,Y]=meshgrid(x,y);
Output power=(X.*Y); % Torque x speed = power
B=(Y.^2)*kc; % Copper losses
C=X*ki; % Iron losses
D=(X.^3)*kw; % Windage losses
Input power = Output power + B + C + D + ConL;
Z = Output power./Input power;
% We now set the efficiencies for which a contour
% will be plotted.
V=[0.5,0.6,0.7,0.75,0.8,0.85,0.88];
box off
grid off
contour(X,Y,Z,V);
xlabel('Speed/rad.s^-^1'), ylabel('Torque/N.m');
hold on
% Now plot a contour of the power output
% The array Output Power has
% already been calculated. We draw contours at
% 3 and 5 kW.
V=[3000,5000];
contour(X,Y,Output power,V);
```

This program was used to give the graph shown as Figure 7.7. In Figure A.1 we show the result obtained for a higher power, higher speed motor, without brushes. All that has happened is that the motor loss constants have been changed, and the ranges of values for torque and angular speed have been increased as follows:

```
x=linspace(1,1000); % N.B. rad/s, not rpm
y=linspace(1,250);
% Allocate motor loss constants.
kc=0.2; % For copper losses
ki=0.008; % For iron losses
```

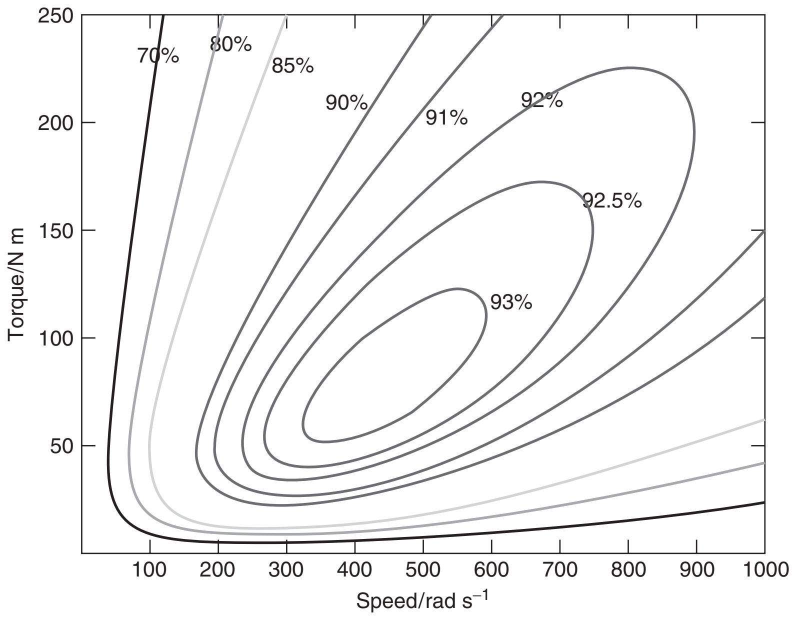

Figure A.1 A plot showing the efficiency of a motor at different torque/speed operating points. It shows the circular contours characteristic of the brushless DC motor

```
kw=0.00001; % For windage losses
ConL=400; % For constant motor losses
```

Also, the values of efficiency were changed, and the last lines that plot constant power contours were removed.