**The Pennsylvania State University**

**The Graduate School**

**College of Engineering**

# COMPUTATIONALLY EFFICIENT DETERMINISTIC DYNAMIC PROGRAMMING FOR OPTIMAL SUPERVISORY POWER MANAGEMENT OF POWER-SPLIT PHEVS

A Thesis in

Mechanical Engineering

by

Timothy R. Montgomery



Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science

May 2013

The thesis of Timothy R. Montgomery was reviewed and approved* by the following:

Hosam Fathy
Assistant Professor of Mechanical Engineering
Thesis Advisor

Christopher Rahn
Professor of Mechanical Engineering

Karen A. Thole
Professor of Mechanical Engineering
Department Head of Mechanical and Nuclear Engineering

*Signatures are on file in the Graduate School.

# Abstract

This thesis addresses some of the computational challenges of applying Deterministic Dynamic Programming (DDP) to plug-in hybrid electric vehicle (PHEV) power management. The goal of this thesis is to develop an approach for reducing the computational and memory needs of DDP-based optimal PHEV power management. The underlying motivation for this work is to create a dynamic program that accommodates dense state variable meshes and can be expanded to include additional state variables and optimization objectives.

DDP is a trajectory-based optimization method that can be used as a tool for benchmarking and studying optimal power management strategies. It can be used to optimize powertrain performance for multiple objectives, but is limited by the number of states and the density of mesh discretization that can be handled, due to the numerical complexity of the control policy search algorithm. These computational difficulties must be overcome before a comprehensive optimization objective can be studied.

To reduce the numerical complexity of applying DDP to PHEV power management, this thesis first develops a powertrain control framework with the engine as the sole independent control input device. This thesis then uses mesh space vectorization to improve the efficiency of the exhaustive optimal input policy searching process. Finally, this thesis employs the novel use of mesh space partitioning to maximize the use of the parallel processing units within a single computer without exceeding the computer's physical memory limits. Using these numerical acceleration methods, the thesis delivers a DDP algorithm that is capable of optimizing in near-real-time, is well-suited to handling dense state and input meshes, and is amenable to the addition of new state variables and optimization objectives.

# Table of Contents

# List of Figures

# List of Tables

# Acknowledgments

I would like to thank, first and foremost, my research advisor, Dr. Hosam Fathy. To say that Hosam has played a critical role in the completion of this thesis would be an extreme understatement. When I began my research, I would have said that I am most thankful for his patience and understanding, but now I realize that I am most thankful for his ability to determine exactly how and when to motivate true progress. Since I have met him, Hosam has molded me as my professor, guided me as my advisor, and inspired me as a person.

I would like to thank every member of the Control Optimization Lab for their feedback, support, and camaraderie. I would like to specifically thank Mike Rothenberger, who continually went out of his way to help anyone, with anything, at any time. Saeid Bashash has been a model of excellent research and productivity, and his work ethic spreads to those around him. Mike Beeney and Sergio Mendoza I thank for not only providing relief from my work, but also reminding me to get back to it, making our office conditions *optimal*....

I am extremely thankful to my entire family for their love, encouragement, and support. I believe my parents are directly responsible for this thesis, as they have taught me from a young age to strive for excellence, realize my God given potential, and of course to bleed Blue and White.

Last, and certainly not least, I would like to thank all of my friends back home who incessantly called upon me to "FINISH YOUR THESIS!" It certainly never hurts to be reminded of one more thing to work for.

# Chapter 1

# Introduction

This thesis improves the efficiency of using Deterministic Dynamic Programming (DDP) to examine the problem of optimal power management of plug-in hybrid electric vehicle (PHEV) powertrains. DDP is a trajectory based optimization algorithm that guarantees a globally optimal solution. It is therefore a useful tool that can be viewed as a benchmark for comparison with other control strategies [1, 2]. It can also be used as a means of studying optimal powertrain performance in order to extract near-optimal control strategies that can be implemented in-vehicle [16–18]. The benefits of these studies are directly related to the accuracy of the powertrain model and the ability of the algorithm to freely select from an appropriate set of control input decisions. An increase in the accuracy or scope of the model is unavoidably linked to an increase in numerical complexity and convergence time. The overarching goal of this thesis is to construct a deterministic dynamic programming algorithm that minimizes computation requirements while maximizing the fidelity of the PHEV model and the control strategies selected.

## 1.1 Background

Over the last two decades, electrification of automobile drive trains has been an increasing trend in the consumer market, as well as the research community [7]. Increasing fuel prices have caused consumers to turn toward more fuel efficient vehicles, such as hybrid electric vehicles (HEVs). On the research and production side, legislation pushing for stricter emissions standards has forced automotive companies to make progress in the development of cleaner cars [9].

Hybrid electric vehicles improve upon conventional vehicle design by adding to the powertrain an energy storage device (such as a battery) and a means of converting that energy into propulsive power (an electric motor). The addition of supplemental sources of energy and power creates opportunities for optimizing the use of the combustion engine while still delivering the total power demanded by the driver. Specifically, HEVs have three main advantages over conventional vehicles. First, the additional power source allows for the engine to be downsized, which will result in more efficient energy conversion during combustion and reduce the weight of the engine block. Second, by isolating the engine from the final drive, the operating point of the engine

can be shifted into efficient regions more consistently. Third, electrochemical energy can be regenerated during braking and periods of low power demand.

Plug-in hybrid electric vehicles are HEVs with battery packs which can be charged in-vehicle during operation, or by plugging directly into the electric grid between trips. The battery packs used in PHEVs typically have higher storage capacity than HEVs. With a substantial source of electrochemical energy, PHEVs can not only use the electric machines as a compliment to the combustion engine, but as an *alternative* to the engine in many instances. If the electric machines are sized to handle typical power demands, a PHEV can operate as though it were a pure electric vehicle (EV) for as long as the battery holds sufficient charge. The distance that can be travelled without consuming fossil fuels is known as all-electric range, or AER, and is an important defining characteristic of a PHEV because of its implication on fuel economy. The fuel economy of a PHEV is that of an EV when the distance travelled is less than AER, but once this range is exceeded, the fuel economy is determined by the amount of stored electrochemical energy, trip length, and control strategy [8, 9, 11, 17]. In contrast, the fuel economy of a HEV is relatively independent of trip length. Another important characteristic of a PHEV (and some HEVs) is its ability to shut down the engine mid drive cycle, or when the vehicle comes to rest, such as at a traffic light. This added level of control, known as start/stop functionality, allows for further improvement in fuel economy and significant reduction of harmful emissions [16–18].

## 1.2 Power Split PHEV Architecture

This thesis focuses on the modeling and control of a PHEV with a single-mode power split architecture, which can be seen in Figure 1.1. This architecture, or similar configurations, can be found in various HEVs and PHEVs, most notably the Toyota Prius [3]. A power split PHEV is capable of delivering propulsive power to the wheels via an internal combustion engine, electric machines, or both simultaneously. It is because of these multiple paths that the power split architecture is sometimes referred to as a "series/parallel" configuration. Along the series path, mechanical power from the combustion engine is transformed to electrochemical form using a generator, stored in the battery pack, and then transformed back into mechanical power by a motor. The parallel paths consist of an electrical path from the battery pack, through the electric machines, to the wheels, and a mechanical path from the engine to the wheels. This flexibility

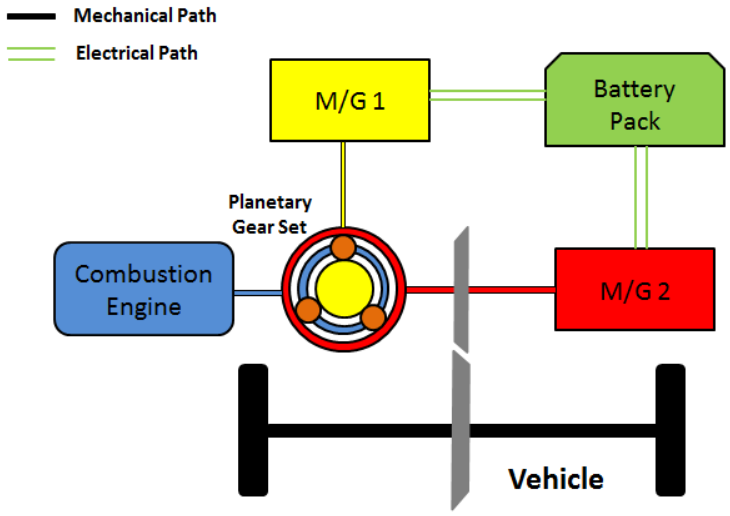


**Figure 1.1.** Single mode power-split hybrid powertrain configuration

in energy routing and power delivery provides the potential for optimal powertrain control while still delivering the performance demanded by the driver.

## 1.3 Optimization Objectives

The goal of PHEV supervisory power management is to optimize the powertrain's performance for some metric, or metrics, while meeting overall driver power demand. The selection of the metrics to be used in the optimization objective is a crucial decision for the control engineer. There are at least four objectives to be considered in the PHEV power management problem:

1. To minimize the consumption of fossil fuel.
2. To minimize the amount of energy drawn from the grid.
3. To minimize the on-road emissions.
4. To minimize battery degradation.

The most obvious optimization objective in any HEV power management problem is to minimize the vehicle's consumption of fossil fuels while driving. This objective is generally considered the primary objective because fuel consumption is the most expensive and easily observable cost associated with driving an automobile (with the exception of the initial purchase price).

The use of energy stored in the battery becomes a significant cost for this problem because PHEVs, by definition, are capable of charging the battery pack by plugging into the electric grid. The cost associated with using that energy while driving becomes non-trivial. The cost associated with fossil fuels and grid energy can both be expressed in dollar amounts, and several authors have used this information to create optimization objectives that combine fuel and electrochemical energy lumped together to represent the total cost of operation [10, 11, 15, 19, 26].

PHEV power management can also focus on reducing the harmful tailpipe emissions that are produced while driving. Tailpipe emissions are closely related to fuel consumption in that consuming more fuel leads to more combustion which necessarily leads to more emissions. However, the combustion process is not the only factor that affects tailpipe emissions, because a significant portion of engine-out emissions are removed by a three-way catalytic converter before they leave the vehicle as exhaust. This conversion process is only effective when the catalytic converter is operating at a high temperature. When the engine first starts up, a significant portion of the engine-out emissions makes it to the tailpipe. While the engine is running, the engine exhaust gases create and maintain a high temperature environment. If the engine is shut down for a sufficient amount of time, the catalyst may cool below its light-off temperature, and another cold start will occur. A PHEV control strategy that optimizes emissions must carefully consider the implications of the Engine ON/OFF strategy [16–18].

Designing a control algorithm that minimizes battery degradation in a PHEV is very important, and also very challenging. One way to avoid damaging the battery pack is to place strict limits on minimum and maximum state of charge or voltage, and by limiting the amount of current that can be drawn from or sunk into the pack. This strategy is easy to implement, but is suboptimal because the determination of these limits involves a non-trivial trade-off between battery health and the amount of usable energy. A very conservative operating range will largely avoid degradation, but will also reduce the amount of usable energy in the battery pack, which means the pack will need to contain a greater number of cells to meet the minimum energy and power requirements. A control policy that can optimize this trade-off is desirable, but difficult to realize, because a high-fidelity battery model would be required [31–33].

Each of these four optimization objectives appear in control strategies found in the literature, but the development of an optimal control policy that considers all four simultaneously remains an extremely difficult task. Deterministic Dynamic Programming is one tool that is excellently

suited for studying the effects of considering individual or combined optimization objectives when developing an optimal control policy. However, the addition of an optimization objective usually requires additional states that must be modeled, which exponentially increases the numerical complexity of the algorithm.

This thesis limits the choice of optimization objective to fuel consumption minimization. The reason for this decision is to simplify the problem formulation so that the focus of the thesis remains on improving the computational efficiency of applying DDP to the PHEV supervisory power management problem. The complications of adding new optimization objectives are discussed again in Chapter 3.

## 1.4 Review of Control Strategies in the Literature

The previous section briefly summarized the numerous potential benefits of HEVs and PHEVs. In order to *realize* these benefits, a suitable control strategy, or power management strategy, must be developed and implemented. The power management literature contains a variety of approaches to optimally controlling electrified powertrains. Despite differing objectives and approaches, the common goal of supervisory power management in hybrid vehicles is to optimize some key metric, such as fuel economy, while delivering the power demanded by the driver at any point in time. While the focus of this thesis is on optimal power management of the drivetrain, it is important to note that PHEV optimization extends beyond powertrain control. The literature also contains studies involving optimal configuration design and component sizing [13, 24, 26, 30], as well as PHEV charging patterns and interaction with the grid [10, 27].

The literature devoted to optimal supervisory power management of hybrid vehicles can be grouped into two categories: rule-based strategies and trajectory-based strategies. Rule-based strategies are implemented such that control decisions are made based on the vehicle's current states and the power demanded by the driver at a particular instant in time. The decision comes from a map, table, or otherwise deterministic rule base that relates state information, such as vehicle speed and battery state of charge, to input commands that are sent to the engine and electric machines. It is possible for these rule bases to be developed heuristically, using only engineering intuition, but more recent efforts have involved rigorous methods. For example, Charge Depletion/Charge Sustenance (CD/CS) strategies can be developed using insight or optimization [22].

Another well-known method, Equivalent Consumption Minimization Strategy (ECMS), can be derived analytically based on Pontryagin's minimum principle [19, 20]. Fuzzy logic has also been used with the goal of using a pre-defined set of control strategies to develop a single rule base that is tailored to the average behavior of a single driver [23]. Stochastic Dynamic Programming (SDP) approaches are well suited to PHEV powertrain management because the decision map can be rigorously optimized over a probabilistic distribution of drive cycles. [4, 13, 24–26].

Regardless of the mathematical rigor with which a power split rule base is developed, a decision map that is a function only of the vehicle's current states and inputs will always be suboptimal in a global sense. Global optimality can, however, be guaranteed when using a trajectory based approach, such as Deterministic Dynamic Programming, or DDP [1, 2, 10, 11, 13, 14, 16–18]. These approaches rely on *a priori* knowledge of a drive cycle in order to find the optimal state trajectory and the sequence of control decisions that will cause the PHEV to follow that path. Because they rely on knowledge of future events, trajectory based control strategies cannot be implemented online. A few techniques, however, have been presented that merge trajectory based optimization with real time implementation using stochastic projections of future states. These methods include Model Predictive Control and Adapative-ECMS [14, 15, 28]. The literature also contains methods for the extraction of rule based strategies from studies of powertrain control using trajectory based methods [16–18].

While the literature is rich in studies devoted to optimal power management of standard HEVs, PHEV power management is a younger, but rapidly growing research area. Early research in the area shows that PHEV-specific powertrain control strategies cannot simply be extensions of the approaches used previously for standard HEVs [8, 9]. PHEVs have different capabilities and objectives than HEVs, and therefore require different control strategies than HEVs. Notably, the ability to operate in all-electric mode for extended durations makes PHEV power management a significantly more complex problem. With a large capacity that can be supplied by the electric grid, the battery pack becomes a source of energy that is both substantial and costly. Furthermore, the charge of the battery is intended to be depleted in a PHEV, unlike an HEV in which SOC is sustained. This implies that the PHEV controller must make power split decisions based not only on the current states of the vehicle, but also the trip length that remains in the future. This extends the decision making horizon, which in turn increases the difficulty of achieving near-optimality with a rule-based control strategy. A similar argument can

be applied when the optimization objective explicitly considers reduction of harmful emissions. The short term decision to shut down the engine to cease emissions could later result in the production of large quantities of emissions if the engine is off for a sufficient amount of time, due to the poor performance of the catalytic converter during cold starts [16–18].

## 1.5 Original Contributions

This thesis presents a systematic method for applying deterministic dynamic programming to the optimal supervisory power management problem for a power split PHEV. The optimizations are performed on a Toshiba Satellite P750 laptop computer with a 2.20 GHz CPU and 8.00 GB of RAM. The techniques developed in the thesis are intended to enable the application of DDP using personal computers, eliminating the need for supercomputers or computing clusters, but are generally extendable to other machines, regardless of computing power.

A control oriented model is developed using power conservation laws and the equations of motion that describe the powertrain dynamics. Engine start/stop control is included as an independent control input, and the set of equations is solved analytically for the cases of all-electric operation, power split operation, and start/stop events. It is shown that by explicitly guaranteeing adherence to a given drive cycle, which is assumed to be known *a priori*, the set of independent dynamic states is reduced to two, and the set of independent control inputs is likewise reduced to two (including engine start/stop commands). Similar applications of DDP to HEV and PHEV power management found in the literature have included two states (engine speed and battery SOC) and three independent control inputs (engine start/stop control, engine torque, and one electric machine torque) [4, 12, 13]. This thesis uses the same state variables, but the only independent control inputs needed are engine torque and start/stop control. To the author's knowledge, this is the first control-oriented model of a power split hybrid electric vehicle powertrain that eliminates the need for establishing either of the electric machines as an independent input device. This novel formulation reduces the dimension-space of the DDP search algorithm by one, decreasing the total number of computations exponentially.

This thesis uses several numerical acceleration techniques to further reduce the DDP algorithm time-to-convergence. First, the mesh space is vectorized so that the model dynamics can be solved more quickly by utilizing Matlab's built-in parallel computing tools. Second, the vectorized mesh

space is partitioned (when necessary) in order to avoid memory storage requirements that exceed the physical memory limits of the computer. To the author's knowledge, the use of mesh space partitioning is novel in the application of DDP to hybrid vehicle power management.

This thesis also demonstrates how the improvements in computational efficiency allow the optimization algorithm to produce more accurate results by increasing the density of the state mesh space. Specifically, the results of studies on the effects of mesh densities will prove the need to carefully mesh SOC in order to accurately capture the dynamics of the powertrain's electrical path.

The final deliverable of this thesis is a DDP algorithm that

1. Performs optimization within the same order of magnitude as real-time.
2. Improves accuracy by allowing for denser state and input discretizations.
3. Is amenable to additional indepent state variables.

The rest of this thesis is structured as follows: The next Chapter presents the model of the power split vehicle and each subsystem. Chapter 3 presents the mathematical foundation for DDP, applies DDP to the PHEV power management problem, and discusses the difficulties associated with the optimization tool. Chapter 4 discusses the methods used to improve the computational efficiency of the DDP algorithm and presents the results of these techniques. Chapter 5 will draw conclusions from these results and briefly discuss the potential outlook of the work accomplished in this thesis.

# Chapter 2

# Power Split PHEV Model

This thesis models a mid-sized sedan PHEV that is similar in shape, size, and specifications to a Toyota Prius. The vehicle's powertrain has a single-mode power split architecture, depicted in Figure 1.1. This architecture has two energy storage devices: a Li-ion battery pack that stores energy as electric charge, and a fuel reservoir that stores energy as liquid fossil fuel. The powertrain also has three means of delivering propulsive power: one internal combustion engine and two electric machines. The engine consumes fossil fuel through combustion to produce mechanical power. The two electric machines are both capable of operating as a motor or a generator. When operating as a motor, a machine converts electrochemical power from the battery into mechanical energy to propel or impede the vehicle. When operating as a generator, a machine absorbs mechanical energy, from either the vehicle or the engine, and converts it into electrochemical energy to recharge the battery pack. The three power sources are all mechanically inter-connected by a planetary gear set, or PGS. The key components of the power-split architecture are modeled in the next four subsections, and Section 2.5 will develop the complete powertrain model.

## 2.1 Engine Submodel

**Table 2.1.** Engine Parameters

| Parameter | Value |
|---|---|
| Size | 1.5 L |
| Power | 57 kW |
| Max Torque | 110 N·m |

This model uses a high-efficiency 57 kW internal combustion engine comparable to engines seen in modern sedan-class PHEVs with similar architectures. In hybrid vehicles with power-split configurations, the engine is sized to meet the typical power demands required for cruising at medium-high speeds. The electric machines assist heavily in acceleration, especially at low speeds, which means the selected engine can be significantly smaller than in a conventional vehicle of the same class.

For a supervisory control problem in which power, efficiency, and fuel consumption are the

primary focus, the combustion process and high-frequency engine dynamics can be ignored. The model for this engine is a look-up table relating fuel consumption to the speed at which the engine is operating and the torque demanded by the driver. The engine specifications are listed in Table 2.1.

## 2.2 Motor/Generator Submodels

**Table 2.2.** Motor/Generator Parameters

| Component | Parameter | Value |
|---|---|---|
| M/G 1 | Max Power | ±25 kW |
| | Max Torque | ±55 N·m |
| M/G 2 | Max Power | ±40kW |
| | Max Torque | ±305 N·m |

There are two electric machines in the power-split architecture, each of which is capable of operating as either a motor or a generator. The specifications for these machines are listed in Table 2.2. Both machines can provide propulsive power to the vehicle or impede the vehicles motion, but they typically serve different roles. M/G2, sometimes referred to just as the motor, is attached to the same axle as the ring gear and is connected to the final drive through a torque-amplifying gear. Its operation is therefore used to provide drive force directly to the wheels, and also to convert the vehicles kinetic energy into electrochemical energy to be stored in the battery during regenerative braking. Conversely, M/G1 is typically used to shift the operating speed of the engine or to use excess engine power to recharge the battery, and is therefore sometimes referred to as the generator in a power-split configuration. Similar to the engine model, the key focus of the motor/generator models is to relate the mechanical power of each machine to the electric power it consumes or produces. The mechanical power of an electric machine is given by the product of its torque, $T_{mg}$, and speed, $\omega_{mg}$.

$$P_{mg} = T_{mg}\omega_{mg} \tag{2.1}$$

The electric power is then

$$PE_{mg} = P_{mg}\eta^{-k} \tag{2.2}$$

where $\eta$ is an efficiency factor and $k$ is the sign of the mechanical power. If the machine exerts a torque in the same direction as the shaft is rotating, the power is positive and it is acting as a motor. It depletes the battery, and the power draw on the battery is greater than the mechanical power delivered, due to conversion losses. When the machine exerts a torque that opposes the shaft's rotation, the power is negative and it is functioning as a generator. The mechanical power, minus losses, is converted to electric power and used to add charge to the battery. The efficiency of each machine comes from look-up tables derived from experimental data from the ADVISOR database.

## 2.3 Battery Pack Submodel

**Table 2.3.** Battery Parameters

| Parameter | Value |
|---|---|
| Capacity | 7.035 A·hr (4.4 kW·hr) |
| Nominal Voltage | 633 V |
| Max Discharge Power | 40 kW |
| Max Charge Power | 35 kW |

This model uses a 4.4 kW·hr Li-ion battery pack with specifications listed in Table 2.3. The energy capacity of this pack is similar to recent versions of the Toyota Prius [5, 26], but is relatively low compared to long range PHEVs or extended range electric vehicles (EREVs) such as the Chevy Volt. The reason for selecting a smaller battery pack size will be discussed further in Chapter 3. An internal resistance equivalent circuit model is used to model the SOC dynamics of the battery pack. This model is sufficient to capture SOC dynamics with reasonable accuracy, yet simple enough to be used in a control-oriented model in an optimization framework. The battery pack open circuit voltage, $V_{oc}$, is a function of the pack state of charge. The internal resistance, $R$, is a function of the packs state of charge and the direction of current flow. The battery power is then given by the product of the open circuit voltage and the current flowing through the battery minus ohmic losses.

$$P_{batt} = VI - I^2R \tag{2.3}$$

The energetic state of the battery, or $SOC$, is defined as the ratio of electrochemical charge stored in the pack to the maximum storage capacity of the pack, or $Q_{pack}$. This gives us the

dynamic state equation

$$S\dot{O}C = \frac{-I}{Q_{pack}} \tag{2.4}$$

Using Equations 2.3 and 2.4 we can determine the relationship between battery state of charge and power.

$$S\dot{O}C = \frac{-V_{oc} - \sqrt{V_{oc}^2 - 4P_{batt}R_{batt}}}{2R_{batt}Q_{batt}} \tag{2.5}$$

The power into or out of the battery pack is the sum of the electric power drawn from or added to the pack by the two electric machines.

$$P_{batt} = PE_{mg1} + PE_{mg2} = T_{mg1}\omega_{mg1}\eta^{-k} + T_{mg2}\omega_{mg2}\eta^{-k} \tag{2.6}$$

Positive power is definied as adding positive kinetic energy to the vehicle, therefore when $P_{batt}$ is positive, the battery is being discharged and $S\dot{O}C$ is negative.

## 2.4 Planetary Gear Set

A planetary gear set is a power splitting device consisting of three concentric rotating axles: a sun gear, a ring gear, and a planet carrier. The three gears, or nodes, are physically connected by several small pinions, called planets. The planets, while not connected to any input or output shafts, are the means of transmitting force from each node to the other two nodes. Because the axis of each planet is fixed to the carrier and the gear teeth of each planet are meshed with the gear teeth of the sun and ring gears, the following kinematic relationship holds true at all times:

$$\omega_r R + \omega_s S = \omega_c (R + S) \tag{2.7}$$

R and S are the number of gear teeth on the ring and sun gears, respectively.

Under normal operation a PGS is an input-splitting device, where the carrier gear is referred to as the input node, and the ring and sun gears are the two output nodes. A driving torque applied to the carrier axle will be transmitted to the output nodes via the planet pinions, causing all three nodes to rotate at the same speed in the positive direction. The relative velocity between

the three nodes will be non-zero if there are external torques acting on multiple axles. Note that while all three nodes are free to spin at different speeds, the kinematic relationship in Equation 2.7 implies that the PGS actually only has two degrees of freedom. Since the ring and sun nodes are defined as outputs, a positive external torque applied to either axle will oppose positive rotation, causing those gears to decelerate or spin in reverse. The complete set of dynamic equations for a PGS in input-split mode is given by

$$I_c\omega_c = T_c - FR - FS \tag{2.8}$$

$$I_s\omega_s = FS - T_s \tag{2.9}$$

$$I_r\omega_r = FR - T_r \tag{2.10}$$

where $F$ is the internal reaction force between the planet pinions and each node.

The PGS can also act as a single-input single-output ratio gear by locking one of the nodes to the ground. If the carrier node is locked to the ground, $\omega_c$ is zero and Equation 2.7 can be solved for $\omega_r$ to show that

$$\omega_r = -\frac{S}{R}\omega_s \tag{2.11}$$

which leads to

$$T_s = -\frac{S}{R}T_r \tag{2.12}$$

due to the conservation of power. The set of dynamic equations during carrier-lock mode are then given by

$$I_c\omega_c = 0 \tag{2.13}$$

$$I_s\omega_s = T_s - FS \tag{2.14}$$

$$I_r\omega_r = FR - T_r \tag{2.15}$$

## 2.5 Full Powertrain Model

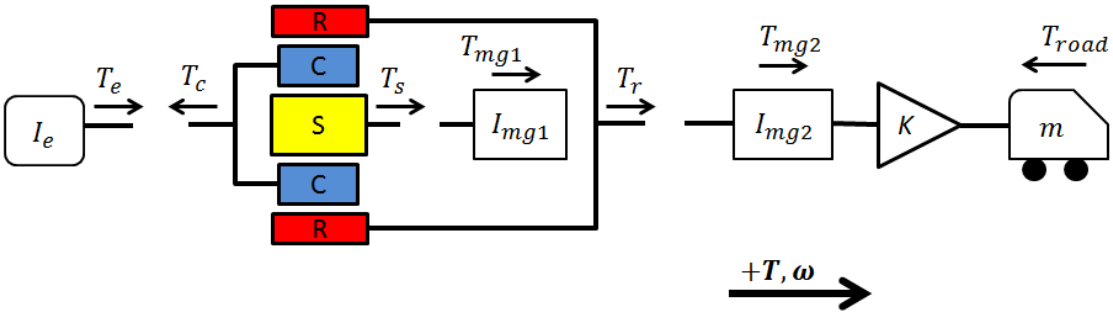


**Figure 2.1.** Free body diagram of the single mode power-split hybrid powertrain

To complete the model of the power-split powertrain, we must define the relationships between the engine, the electric machines, and the vehicle dynamics, as depicted in Figure 2.1. The vehicle dynamics are governed by

$$m\dot{v} = F_{drive} - F_{road} \tag{2.16}$$

where $m$ is the mass of the vehicle, $\dot{v}$ is the longitudinal accelerations, and $F_{drive}$ is the driving force delivered by the tires to the ground. $F_{road}$ is the sum of external forces opposing the vehicles motion:

$$F_{road} = F_{roll} + F_{damp} + F_{drag} \tag{2.17}$$

$F_{roll}$ is the rolling resistance, given by

$$F_{roll} = \mu mg \tag{2.18}$$

where $\mu$ is the coefficient of friction between the tires and the road, $m$ is the mass of the vehicle, and $g$ is acceleration due to gravity. $F_{damp}$ is the viscous bearing friction force, given by

$$F_{damp} = \frac{b_w}{R_{tire}} v \tag{2.19}$$

where $b_w$ is the viscous damping constant and $R_{tire}$ is the radius of the vehicles tires. $F_{drag}$

is aerodynamic drag given by

$$F_{drag} = \frac{1}{2}\rho A_f C_d v^2 \tag{2.20}$$

where $\rho$ is the air density, $A_f$ is the frontal surface area, and $C_d$ is the drag coefficient. Finally, the resistive torque from the road acting on the final drive is

$$T_{road} = R_{tire} F_{road} \tag{2.21}$$

**Table 2.4.** Vehicle Parameters and Constants

| Vehicle Parameter | Parameter Value |
|---|---|
| $m$ | 1420 kg |
| $R_{tire}$ | 0.287 m |
| $K$ | 3.93 |
| $\mu$ | 0.009 |
| $b_w$ | 0.03 N·s |
| $A_{fr}$ | 2.23 $m^2$ |
| $C_d$ | 0.26 |
| $\rho$ | 1.2 kg/$m^3$ |
| $g$ | 9.81 m/$s^2$ |

A complete list of vehicle specifications and constants used in the model can be found in Table 2.4. Referring back to Figure 1.1, the carrier, sun, and rung gears of the PGS are attached to the engine, M/G1, and M/G2 respectively. We can complete the powertrain model by describing the dynamics of each axle. The rotational speed of the axle connected to the carrier and the engine will be called $\omega_{ce}$, and is given by

$$I_e \dot{\omega}_{ce} = T_e - T_c \tag{2.22}$$

where $I_e$ is the mass moment of inertia of the engine and flywheel, $T_e$ is the engine torque, and $T_c$ is the reaction torque at the carrier node. The rotational velocity of the axle connected to the ring gear and M/G1, $\omega_{s1}$, is given by

$$I_{mg1} \dot{\omega}_{s1} = T_{mg1} + T_s \tag{2.23}$$

where $I_{mg1}$ is the mass moment of inertia of M/G1, $T_{mg1}$ is the torque exerted by M/G1, and $T_s$ is the reaction torque at the sun gear. The axle connecting the ring gear to M/G2 is also

connected to the final drive through a gear ratio. The rotational velocity, $\omega_{r2}$, is given by

$$I'_{mg2}\dot{\omega}_{r2} = T^*_{mg2} + T_r - \frac{1}{K}T_{road} \tag{2.24}$$

where $K$ is the final drive ratio, and $I'_{mg2}$ is the equivalent mass moment inertia of M/G2 and the vehicle, definied as

$$I'_{mg2} = \frac{R^2_{tire}}{K^2}m + I_{mg2} \tag{2.25}$$

$T^*_{mg2}$ represents the combined torque output from M/G2 and mechanical friction brakes applied at the wheels when necessary. During mild braking, M/G2 operates as a generator to slow the vehicle and regenerate charge that is stored in the battery pack. The friction brakes are only applied if the stopping power of M/G2 is exceeded.

$$T^*_{mg2} = T_{mg2} + \frac{1}{K}T_{fb} \tag{2.26}$$

If we make the reasonable assumption that the moment of inertia of each gear in the planetary gear set is negligible, we can combine Equations 2.22-2.24 with Equations 2.8-2.15 to express each reaction torque using the internal reaction force, F. Recall that Equations 2.8-2.10 only apply when all three nodes are free to rotate, and Equations 2.13-2.15 only apply when the carrier gear is locked to ground. To complete the model, we take the time derivative of Equation 2.7 to describe the relationship that relates Equations 2.22-2.24.

$$\dot{\omega}_{r2}R + \dot{\omega}_{s1} = \dot{\omega}_{ce}(R + S) \tag{2.27}$$

This section has presented a set of equations that model the mechanical dynamics of a single-mode power split powertrain. The method for solving this set of equations depends on the selection of known inputs. This selection is critical when the model is used in an optimal power management context, and will be discussed in the next chapter.

# Chapter 3

# Solution Method

This thesis focuses on the use of Deterministic Dynamic Programming as a tool in developing optimal power management strategies for plug-in hybrid electric vehicles. DDP is a trajectory-based optimization technique used to find the set of control decisions that result in a state trajectory in a discrete domain which minimizes some additive cost function. The result is guaranteed to be a globally optimal solution, within the tolerances afforded by the state and input discretization [1,2]. While DDP has been used extensively to solve the power management problem for HEVs [10, 11, 13, 14, 16–18], extending its use to the PHEV power management problem presents a new set of challenges that has been previously unaddressed in the literature.

## 3.1 Mathematical Principle

This thesis will only briefly present the mathematical principle behind the DDP algorithm and the proof of its optimality. Interested readers are referred to [1,2] for more information regarding the algorithm and its application to similar problems.

The purpose of DDP is tofind the sequence of control decisions at every time instant $k$ that minimizes the aggregated cost function

$$J = G_N(x(N)) + \sum_{k=1}^{n} L_k(x(k), u(k), w(k)) \tag{3.1}$$

for which

$$x(k+1) = f(x(k), u(k), w(k)) \tag{3.2}$$

subject to

$$xk \in X(k) \qquad u(k,x) \in U(k) \tag{3.3}$$

In this formulation, $J$ is the aggregated cost, $L$ is the instantaneous transition cost, and $G_N$ is the terminal cost at $k = N$. $x(k)$ is the state vector within the state space $X(k)$, $u(k)$ is the control input vector within the input space $U(x(k), k)$, $w(k)$ is known disturbance, and $f$

represents the dynamics of the system. The problem statement can also be formulated with constraints on the state and input variables.

$$g_i(x(k)) \leq 0 \qquad i = 1, 2, \cdots q \tag{3.4}$$

$$h_i(u(k)) \leq 0 \qquad i = 1, 2, \cdots p \tag{3.5}$$

The solution to this problem relies on Bellman's Principle of Optimality, which states "An optimal policy has the property that, whatever the initial state and optimal first decision may be, the remaining decisions constitute an optimal policy with regard to the state resulting from the first decision" [1]. Conceptually, this "optimal policy" can be thought of as a map with a set of paths that are known to be optimal. If a system's initial conditions place it at the beginning of one of these paths at time step $k = 0$, the total cost of following the pre-mapped path to its destination is guaranteed to be lower than the total cost of following any other path to the same destination, because the path was known to be optimal. The problem, then, is to construct this map. More precisely, the problem is to develop an optimal policy, which is a map containing the optimal control decision at every step in time for every feasible state.

To determine this optimal policy, we solve a series of sub problems beginning at the second-to-last step in time. The cost of applying an input $u$ to the system in state $u$ at time step $N-1$ is referred to as the cost-to-go, $J$. The cost-to-go for each input is the sum of the transition to the next state and the terminal cost of that state. The optimal input at each state is the input for which the cost-to-go is minimal.

$$J^*(x(N-1)) = \min_{u(N-1)} \{G_N(x(N)) + L_{N-1}(x(N-1), u(N-1), w(N-1))\} \tag{3.6}$$

Propogating this method backwards in time, the cost-to-go resulting from applying input $u$ to the system in state $x$ at time step $k$ is then given by the sum of the cost of transitioning to $x(k+1)$ and the optimal cost-to-go from $x(k+1)$ onward. The optimal cost-to-go then is

$$J^*(x(k)) = \min_{u(k)} \{L_k(x(k), u(k), w(k)) + J^*(x(k+1))\} \tag{3.7}$$

and the optimal input that to this transition, $u^*(x, k)$ is stored. Once the optimal input is discovered for every state at every step, the control policy is complete. The optimal state and

input trajectories are then found by beginning at the initial conditions and moving forward in time step by step until the final state is reached.

## 3.2 Complications and Limitations of DDP

Before we begin to apply DDP to the PHEV power management problem, it is important to discuss two of the relevant complications and limitations of the algorithm. The first issue is the impact of the number of independent states and inputs that must be discretized to solve the problem. The computation time and memory used to solve the problem increase exponentially with the number of discretized variables. Problems with few discretized variables, therefore, are exponentially easier to solve with DDP. This well-known problem is referred to as "The Curse of Dimensionality." The second issue is that the accuracy of a control policy determined using DDP is limited by the extent to which the variables are discretized. A coarse mesh has large gaps between mesh points in the state and input grids and will lead to inaccurate dynamics. Furthermore, the next state that is calculated as a result of a particular input is not guaranteed to be coincident with a point on the state mesh. This needs to be rectified since the cost-to-go information is only stored at discrete points on the state mesh. One remedy for this situation is to use linear interpolation to calculate cost-to-go at an intermediate state based on the cost-to-go information stored for multiple nearby points on the mesh. Linear interpolation becomes problematic when boundary constraints are imposed on the state variables, because the cost imposed for violating these constraints becomes mixed with the actual transition cost. This mixing, known as constraint leakage, can propagate constraint violation costs through the discrete state space, making feasible paths appear infeasible causing them to be eliminated as options. In order to avoid constraint leakage, this thesis uses a "nearest neighbor" approach. When the next state resulting from a particular input is calculated, it is then shifted to the closest discrete point on the state mesh. As a result, the accuracy of the selection of the optimal control policy becomes directly related to discretization density. Using a finer mesh density will lead to more accurate dynamics, calculation of cost-to-go, and selection of control policies, but also increases the length of computation. The significance of mesh density is one of the key reasons DDP for PHEV power management is a more difficult problem than DDP for HEV power management. PHEVs generally have battery packs with much higher capacity than packs that would be found

in HEVs. For the same amount of power, a battery with a larger capacity will exhibit smaller changes in $SOC$. In order for smaller changes to be observed in a discretized state space, the mesh density must be increased. Therefore, the $SOC$ mesh density must increase as the amount of electrochemical energy storage of a vehicle increases. The effects of unobserved changes in battery pack $SOC$ are examined in Section 4.2.3. It is important to keep these issues in mind when formulating a DDP problem for a practical application, and we will revisit their effects in Chapter 4.

## 3.3 Applying DDP to Power Split PHEV Power Management

To apply DDP to optimal supervisory power management of power-split PHEVs, we begin with the decision of which states and control inputs to discretize. As discussed previously, it is desirable to reduce the number of discretized variables as much as possible. The complete set of states in the powertrain model presented in Chapter 2 include $\omega_{ce}$, $\omega_{s1}$, $\omega_{r2}$, and $SOC$. The inputs to the system are $T_e$, $T_{mg1}$, $T_{mg2}$, and the decision to start/stop the engine.

The dynamics of battery SOC are separate from the mechanical states and therefore cannot be related to them. Furthermore, the battery dynamics are directly related to the overarching control decision of how to split the power demanded at any instant between the combustion engine and the battery pack. Therefore SOC must be discretized. Similarly, engine speed must be discretized, and it must include a state that represents that the engine is off. Memory of engine ON/OF control decisions must be stored, and the only way to do is to use engine speed as a state variable. Engine ON/OFF *could* be added as a new state variable, but this would double the size of our state mesh, increasing the computation time. Ultimately it is unnecessary because the status of the engine can be stored in the engine speed state mesh.

The choice of control inputs to discretize that is made in this thesis is novel. In similar studies, it is typical for authors to discretize engine torque and M/G1 torque [4, 12, 13]. This is a logical choice because it determines how much power is drawn from the engine as well as how the battery power is split between M/G1 and M/G2. However, it is actually unnecessary to discretize more than one torque. The decision that a power-splitting control policy tries to make is determining the portion of power demanded that will be supplied by the engine and portion that will be supplied by the battery through the electric machines. If engine speed and

torque are both meshed, the power split decision has already been made and neither of the M/G torques need to be meshed. The sufficiency of this decision will be proven once the model has been fully solved. We begin this solution by re-emphasizing that DDP relies on a prior knowledge of all disturbances. The goal of the problem is to optimize performance while following a pre-established drive cycle, which is known *a priori*. By assuming that there is no tire slip at the road, we can relate the speed and acceleration of the ring gear to the drive cycle.

$$\begin{aligned} \omega_{r2} &= \frac{K}{R_{tire}} v \\ \dot{\omega}_{r2} &= \frac{K}{R_{tire}} \dot{v} \end{aligned} \tag{3.8}$$

With $\omega_{r2}$ known and $\omega_{ce}$ meshed, $\omega_{s1}$ can be calculated using Equation 2.7. Furthermore, since $v$ and $\dot{v}$ are known, the power demand at every time step can be given by

$$P_{dem} = F_{drive} v \tag{3.9}$$

and we can apply the conservation of power to determine a relationship between the power demand and the total power output of the powertrain:

$$P_{dem} = T_e \omega_{ce} + T_{mg1} \omega_{s1} + T_{mg2} \omega_{r2} \tag{3.10}$$

The entire set of equations can now be solved for every unknown variable. In order to do this, however, Engine ON operation and Engine OFF operation must be considered separately.

## 3.4 Engine OFF Operation

When the engine is shut down, the carrier node of the planetary gear set is locked to the ground so that the engine shaft is not rotating and wasting energy due to friction in the engine cylinders. Additionally, if the shaft were free to rotate, the shaft speed would be very difficult to control because the motors exert large torques and the inertia of the gear and shaft are very small. With the carrier locked, Equations 2.13–2.15 can be applied to Equations 2.22–2.24 to obtain the full set of equations that describe the powertrain mechanical dynamics during engine OFF operation.

$$I_e\dot{\omega}_{ce} = 0 \tag{3.11}$$

$$I_{mg1}\dot{\omega}_{s1} = T_{mg1} - FS \tag{3.12}$$

$$I'_{mg2}\dot{\omega}_{r2} = T_{mg2} + FR - \frac{1}{K}T_{road} \tag{3.13}$$

$$\dot{\omega}_{s1}S + \dot{\omega}_{r2}R = 0 \tag{3.14}$$

Since $\omega_{r2}$ and $\dot{\omega}_{r2}$ are known from the drive cycle, $\omega_{s1}$ and $\dot{\omega}_{s1}$ become known using the kinematic constraint of the planetary gear set. Combining these dynamics and Equation 3.10 for power demand, we can solve the entire system analytically. Solving Equation 3.12 and 3.13 for M/G torque, we obtain

$$T_{mg1} = I_{mg1}\dot{\omega}_{s1} + FS \tag{3.15}$$

$$T^*_{mg2} = I'_{mg2}\dot{\omega}_{r2} - FR + \frac{1}{K}T_{road} \tag{3.16}$$

If we substitute Equation 3.15 and 3.16 into Equation 3.10, we can solve to obtain an analytic expression for $F$ in terms of known variables.

$$F = \frac{1}{\omega_{s1}S - \omega_{r2}R}[I_{mg1}\dot{\omega}_{s1}\omega_{s1} + I'_{mg2}\dot{\omega}_{r2}\omega_{r2} + \frac{1}{K}T_{road}\omega_{r2} - P_{dem}] \tag{3.17}$$

Note that if the vehicle is not moving, the denominator in this equation evaluates to zero. In this case, $F$ is zero because there is no force between the gear teeth when none of the gears are rotating. With $F$ known, we can solve for the two M/G2 torques using Equation 3.15 and Equation 3.16, and there are no longer any unknowns in the system.

## 3.5 Engine ON Operation

When the engine is on, the carrier gear is free to rotate and the planetary gear set is used as an input-split devie. Combining Equations 2.8–2.10 with Equations 2.22–2.24 we obtain the set of equations governing the powertrain dynamics while the engine is on.

$$I_e\dot{\omega}_{ce} = T_e - F(R + S) \tag{3.18}$$

$$I_{mg1}\dot{\omega}_{s1} = T_{mg1} + FS \tag{3.19}$$

$$I'_{mg2}\dot{\omega}_{r2} = T^*_{mg2} + FR - \frac{1}{K}T_{road} \tag{3.20}$$

$$\dot{\omega}_{s1}S + \dot{\omega}_{r2}R = \dot{\omega}_{ce}(R+S) \tag{3.21}$$

Combining these equations with Equation 3.10 for power demand, we can again solve the entire systems of equations analytically. We begin by rearranging Equations 3.18 and 3.19 to solve for $\dot{\omega}_{ce}$ and $\dot{\omega}_{s1}$, respectively.

$$\dot{\omega}_{ce} = \frac{1}{I_e}\left[T_e - F(R+S)\right] \tag{3.22}$$

$$\dot{\omega}_{s1} = \frac{1}{I_{mg1}}[T_{mg1} + FS] \tag{3.23}$$

Substituting Equations 3.22 and 3.23 into Equation 3.21 and solving for $T_{mg1}$, we obtain

$$T_{mg1} = \frac{R+S}{S}\frac{I_{mg1}}{I_e}T_e - \frac{R}{S}I_{mg1}\dot{\omega}_{r2} - I'_f F \tag{3.24}$$

where the equivalent inertia, $I'_f$, is

$$I'_f = \left(\frac{(R+S)^2}{S}\frac{I_mg1}{I_e} + S\right)F \tag{3.25}$$

Similarly, Equation 3.20 can be arranged for $T^*_{mg2}$.

$$T^*_{mg2} = I'_{mg2}\dot{\omega}_{r2} - FR + \frac{1}{K}T_{road} \tag{3.26}$$

Substituting Equations 3.24 and 3.26 into Equation 3.10 for power demand, we rearrange to find an analytic expression for $F$ in terms of known quantities.

$$F = \frac{T_e\omega_{ce} + \frac{R+S}{S}\frac{I_{mg1}}{I_e}T_e\omega_{s1} - \frac{R}{S}I_{mg1}\omega_{s1}\dot{\omega}_{r2} + I'_{mg1}\omega_{r2}\dot{\omega}_{r2} + \frac{1}{K}T_{road}\omega_{r2} - P_{dem}}{I'_f\omega_{s1} + R\omega_{r2}} \tag{3.27}$$

With $F$ known, $T_{mg1}$ and $T_{mg2}$ can be calculated using Equations 3.24 and 3.26, respectively. Finally, $\dot{\omega}_{ce}$ and $\dot{\omega}_{s1}$ are given by Equations 3.18 and 3.19, and there are no longer any unknowns in the system.

## 3.6 Engine Start-up and Shut-down

To complete the power-split powertrain model, the transitions between Engine ON and Engine OFF operation must be accounted for. The transient dynamics of the engine do not need to be considered, but the input control policy should be calculated to determine feasibility. Also, a fuel penalty needs to be applied in order to make the model realistic, and in order to develop an optimal control policy that discourages chatter.

Since the carrier gear remains free to rotate during transitions between Engine ON and OFF modes, the same dynamic equations for Engine ON operation can be applied, but the solution to the set of equations is much simpler. The transition is controlled by using M/G1 to reduce the engine speed to zero during shut-down, or rev the engine up to an operable speed, $\omega_{rev}$. The desired change in engine speed can then be given by

$$\dot{\omega}_{ce} = \frac{\omega_{rev}}{dt} \tag{3.28}$$

for engine start-up and

$$\dot{\omega}_{ce} = -\frac{\omega_{ce}}{dt} \tag{3.29}$$

for engine shut-down. Using Equation 2.27 for the kinematic constraint on the planetary gear set,

$$\dot{\omega}_{s1} = \frac{1}{S}(\dot{\omega}_{ce}(R+S) - \dot{\omega}_{r2}R) \tag{3.30}$$

The engine will not produce power during either process, so we can use Equation 3.18 to calculate the internal reaction force, $F$, needed to spin the engine to the desired speed.

$$F = -\frac{1}{R+S} I_e \dot{\omega}_{ce} \tag{3.31}$$

Finally, $T_{mg1}$ and $T_{mg2}$ can be solved using Equations 3.24 and 3.26, respectively. Although the power demand equation was not used in this derivation, it still holds true. The total power demanded of the drive train during start-up or shut-down is actually the sum of the power required by the drive cycle and the power required to alter the energetic state of each gear in the

power train. If this additional term is included in the calculation of power demand, the solution method for Engine ON operation will be consistent with the special cases derived in this section.

## 3.7 Complete DDP Formulation

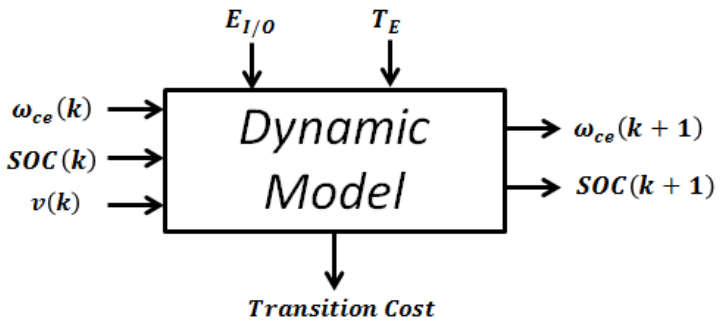


**Figure 3.1.** Block diagram representation of the system's inputs and outputs

The DDP problem can now be formulated in its entirety using the meshed variables listed in Table 3.1. As discussed in Section 1.3, the optimization objective is to minimize fuel consumption.

$$\min_{k=1}^{N} \{m_{fuel}(T_e, \omega_{ce})\} \tag{3.32}$$

The dynamics are constrained by

$$x(k+1) = f(x(k), u(k), w(k)) \tag{3.33}$$

as described in the previous sections. In this problem, the disturbance trajectory, $w(k)$ is the velocity profile for the given drive cycle.

**Table 3.1.** Meshed Variables

| | |
|---|---|
| Meshed State Variables | $\omega_{ce}$, $SOC$ |
| Meshed Input Variables | $E_{IO}$, $T_e$ |
| Disturbance Variables | $v$, $\dot{v}$ |

The states and inputs are also subject to a set of constraints that represent the physical limitations of the battery pack, the engine, and the electric machines:

$$
\begin{aligned}
&\omega_{ce,min} \leq \omega_{ce} \leq \omega_{ce,max} \\
&\omega_{s1,min} \leq \omega_{s1} \leq \omega_{s1,max} \\
&\omega_{r2,min} \leq \omega_{r2} \leq \omega_{r2,max} \\
&SOC_{min} \leq SOC \leq SOC_{max} \\
&T_{e,min} \leq T_e \leq T_{e,max} \\
&T_{mg1,min} \leq T_{mg1} \leq T_{mg1,max} \\
&T_{mg2,min} \leq T_{mg2} \leq T_{mg2,max} \\
&P_{e,min} \leq P_e \leq T_{e,max} \\
&P_{mg1,min} \leq P_{mg1} \leq P_{mg1,max} \\
&P_{mg2,min} \leq P_e \leq P_{mg2,max} \\
&P_{batt,min} \leq P_{batt} \leq P_{batt,max}
\end{aligned}
\tag{3.34}
$$

If any of these constraints are violated, an extremely large penalty cost ($1x10^6 galfuel$) is applied. The algorithm will rule such a case infeasible and will not select the associated inputs as an optimal policy.

This chapter has presented deterministic dynamic programming and its application to power split PHEV power management. The next chapter discusses the techniques that this thesis uses to improve the computational efficiency of the DDP algorithm and the improvements that result from these methods.

# Chapter 4

# Results and Discussions

This chapter discusses the effectiveness of methods used to resolve some of the computational challenges of applying DDP to PHEV power management. The first major section of the chapter describes how vectorizing the mesh space can reduce the time of computation and how the novel use of mesh space partitioning allows for dense mesh discretization without exceeding the computers physical memory limits. The second major section presents three case studies that portray the effectiveness of the novel methods used in this thesis. The first case study demonstrates that vectorizing and partitioning the mesh space maximizes the use of the computer's parallel processors and avoids exceeding the computer's physical memory limits. The second case study shows that the elimination of M/G torque as an independent control input reduces the computation time by a factor of twenty-six for a typical mesh size found in the literature [13]. The third case study determines that discretization of the state of charge state variable must be particularly dense for the optimization and simulation results to be accurate.

## 4.1 Methods for Improving Computational Efficiency

This section discusses the factors that affect the run-time and memory requirements of performing DDP optimization, and outlines how vectorizing and partitioning the mesh space can improve the computational efficiency. This thesis uses Matlab to apply the algorithm, so the effectiveness of these techniques may be specific to the Matlab environment. To motivate and illustrate these methods, it is helpful to quantify the size of a specific mesh space. The discussions in this section will refer to the parameters listed in Table 4.1, which reflect mesh densities and discretized variables similar to those found in the literature [13].

**Table 4.1.** DDP Parameters

| Physical Variable | DDP Variable | Min | Max | n |
|---|---|---|---|---|
| Engine ON/OFF | $E_{IO}$ | 0 | 1 | 2 |
| $T_e$ | $u1$ | 0 N·m | 110 N·m | 50 |
| $T_{mg1}$ | $u2$ | -55 N·m | 55 N·m | 50 |
| $\omega_{ce}$ | $x1$ | 0, 100 rad/s | 500 rad/s | 50 |
| $SOC$ | $x2$ | 0.3 | 0.8 | 200 |

Two overarching issues dictate the amount of time required to solve a DDP algorithm:

1. The total number of operations that must be performed
2. The amount of memory that must be stored at any given time

The total number of operations, $O_t$, that must be performed throughout the algorithm is given by

$$O_t = O_m * S_{MS} * N \tag{4.1}$$

where $O_m$ is the number of operations required to solve the model for a single combination of states and inputs, $N$ is the number of time steps, $S_{MS}$ is the size of (number of points on) the mesh space:

$$S_{MS} = n_{x1} * n_{x2} * n_{u1} * n_{u2} * n_{u3} \tag{4.2}$$

For a given mesh space, the only way to reduce $O_t$ is to reduce $O_m$ by simplifying the model and its intermediate steps as far as possible. Once the model is fully reduced and the number of intermediate calculations is minimized, $O_t$ cannot be reduced any further.

The next two subsections will address these issues and present approaches to resolving them.

### 4.1.1 Vectorizing the Mesh Space

Vectorizing the mesh space reduces the time required to perform DDP optimization by simulating the underlying powertrain model for multiple sets of state and input combinations in parallel rather than in series. A non-vectorized DDP algorithm will use the model to fully calculate the system dynamics for a single combination of states and inputs, then proceed to the next combination of states and inputs, etc. Using this iterative method, the model must be solved $S_{MS}$ times for every time step of optimization. Alternatively, a vectorized DDP algorithm sends a matrix containing every possible combination of states and inputs for one time step to the model to be solved at once. This mesh space matrix has one column for every discretized variable, and every row represents a discrete point in the mesh space. At each time step, the dynamic model, system constraints, and cost functions must be solved for each of the $S_{MS}$ row vectors. Matlab

is designed to perform these vector operations efficiently by parallelizing the computations and distributing them throughout the CPU cores without any additional coding or hardware.

Mesh space vectorization is a tactic that is commonly employed with DDP optimization, and the effectiveness of vectorized calculations is known in the vehicular power management literature. For example, Liu and Peng show that by vectorizing two of four discretized variables (and looping iteratively through the other two mesh vectors), model simulation time can be reduced by a factor of 300 [12]. Vectorizing all discretized states and inputs amplifies this improvement.

### 4.1.2 Partitioning the Mesh Space

Vectorizing the mesh space when using DDP optimization can present a significant memory storage issue if the maximum amount of memory that needs to be stored at any point in time exceeds the physical memory limits, or RAM, of the computer. This problem can be mitigated by vectorizing and partitioning the mesh space, or breaking up the mesh space into parts that are small enough for the computer's capabilities. Consider performing DDP optimization using the mesh space spanned by the discretized variables in Table 4.1 on a machine with 4 GB of RAM. The number of possible combinations of states and inputs, and therefore the number of rows in the input matrix is

$$S_{MS} = 2 * 50 * 50 * 50 * 200 = 50x10^6 \tag{4.3}$$

Using the default settings in Matlab, every element in a vector or matrix takes up 8 bytes of physical memory storage, or RAM. Each vector with $50x10^6$ rows therefore takes up 0.37 GB of physical memory. Each vectors sent to the model of the system dynamics, as well as the vector for every intermediate variable that is created in the process of solving the model, will also have $50x10^6$ rows and take up 0.37 GB of memory. If at one point in time, ten of these variables exist, 3.7 GB of the total 4 GB of RAM are being used. No new variables can be created until space can be created using virtual memory or by clearing existing physical memory. This is a lengthy process, during which no operations are performed. The duration of each memory clear halt is measured in minutes, as opposed to the microseconds each operation takes to perform. As a result, vectorized DDP with mesh sizes that exceed the physical memory limits of the computing device will take longer to converge than non-vectorized DDP with the same mesh size on the same

computer. We can avoid exceeding the computer's memory limits by first vectorizing the mesh space, and then partitioning it into sub-spaces to be handled by the underlying system model. Similar techniques are employed in the computer science fields, but to the author's knowledge, this thesis presents the first application of this method to the power management and control literature. We select the number of partitions, $P$, in order to maximize the benefits of vector operations without exceeding the available physical memory.

$$P = \frac{MEM_{avail}}{MEM_{max}} \tag{4.4}$$

Where $MEM_{avail}$ is the total physical memory available before the iterative solution process begins, and $MEM_{max}$ is the maximum amount of memory that needs to be stored during the solution progress. By breaking the vectorized mesh space into parts that are as large as possible without exceeding the physical memory limits of the computer, we can combine the fast, efficient use of parallel operations with the lower temporary memory storage requirements of performing computations in series. In Section 4.2.1, Case Study 1 will prove the effectiveness the vectorizing and partitioning strategy.

## 4.2 Case Studies

This section presents three case studies that analyze the effectiveness of the novel methods developed in this thesis. All three case studies are performed using a Toshiba Satellite P750 laptop computer with a 2.20 GHz CPU and 8.00 GB of RAM.

### 4.2.1 Case Study 1: Effectiveness of the Partitioning Strategy

This case study demonstrates that the strategy of vectorizing and partitioning the mesh space improves the computational efficiency of the DDP algorithm by maximizing the use of the computer's parallel processing units without exceeding its physical memory limits. First, we show that without partitioning, the optimization time is dominated by long memory clearing pauses when the physical memory limit is reached. Then we demonstrate that the partitioning method avoids these long pauses by breaking the vectorized mesh space into sizable parts that can be handled within the computer's memory capabilities. The results of this case study are summarized in Table 4.2, where $\gamma$ is the total time required to find the optimal control policy for every

state on the mesh at one time step in the drive cycle.

**Table 4.2.** Effect of Partitioning on Simulation Time

| DDP Strategy | $\gamma$ |
|---|---|
| Vectorized Only | 502.1 s |
| Vectorized and Partitioned | 19.5 s |

To demonstrate the efficacy of the partitioning strategy, we optimize the performance of a power split PHEV's powertrain over the U.S. FTP-72 cycle, also known as the Urban Dynamometer Driving Schedule (UDDS). The DDP algorithm used in this case study has two discrete state variables ($SOC$ and $\omega_{ce}$), and two discrete control input variables ($T_e$ and $T_{mg1}$). Engine start/stop commands are not included in this case study. The number of discrete points on each state and input vector are shown in Table 4.3.

**Table 4.3.** Case Study 1: DDP Parameters

| Discrete Variable | Min | Max | n |
|---|---|---|---|
| $T_e$ | 0 N·m | 110 N·m | 50 |
| $T_{mg1}$ | -55 N·m | 55 N·m | 50 |
| $\omega_{ce}$ | 100 rad/s | 500 rad/s | 50 |
| $SOC$ | 0.3 | 0.8 | 201 |

The optimization is first performed using DDP with a fully vectorized mesh space that is *not* partitioned. Figure 4.1 shows the computer's CPU Usage and Memory Storage history as the algorithm begins to develop the optimal control policy for a single time step. The entire vectorized mesh space is input to the powertrain model to generate the next state and transition cost, but before the model is completely solved, the computer's physical memory limits are reached. When the physical memory limit is neared, the CPU is unable to perform any more computations until room is made to store the next variable that will be created. In order to clear space and store new variables, old variables must be deleted or transferred into virtual memory. As the figure shows, memory is stored very quickly and then removed very slowly, during which time the processing units are not doing any substantial work.

Figure 4.2 captures the entire optimization process for a single time step and illustrates this process further. The Physical Memory Usage history shows that most of the optimization time is spent slowly clearing memory, and the CPU Usage history shows that the processing units are

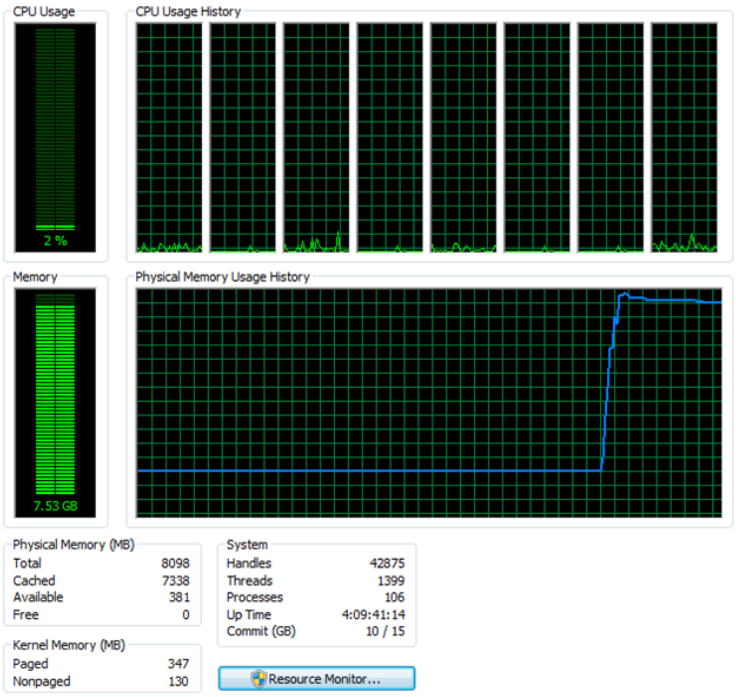


**Figure 4.1.** Screenshot of Memory and CPU usage showing memory clearing during vectorized DDP optimization

only being used in the very brief periods in which operations are performed and new memory is generated. Each memory clearing process takes minutes, in comparison to the milliseconds that most mathematical operations take, and thus memory clearing becomes the dominant factor in determining the time required to perform optimization. As a result, the average optimization time per simulation time step, $\gamma$, for this case is 502.1 seconds, and optimization for the entire UDDS cycle would take approximately eight days.

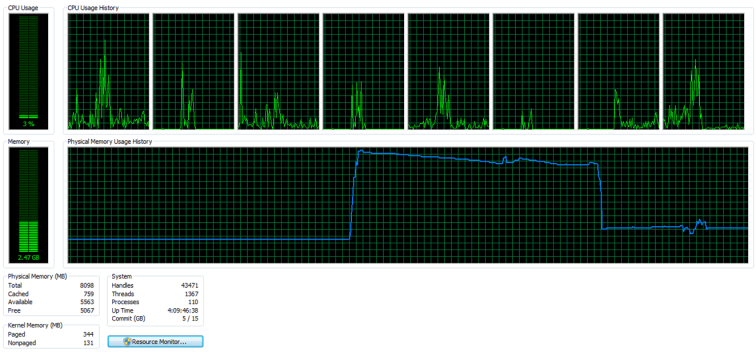

**Figure 4.2.** Screenshot of Memory and CPU usage histories after optimization of a single time step using vectorized DDP

To eliminate the time spent clearing memory during optimization, we employ the partitioning strategy to avoid reaching the computer's physical memory limits at any point during optimization. To demonstrate the efficacy of this strategy, we again optimize PHEV performance over the UDDS drive cycle using the same DDP algorithm, this time employing both vectorization and partitioning. For this particular mesh space size, it is sufficient to break the mesh space matrix into two equal parts, each containing half of all possible combinations of states and inputs. Each partition is separately input to the powertrain model, and the next state and transition cost are calculated and stored. Between each simulation of the model, all temporary memory that was stored is deleted, reducing the maximum amount of memory that needs to be stored at any one point in time during optimization.

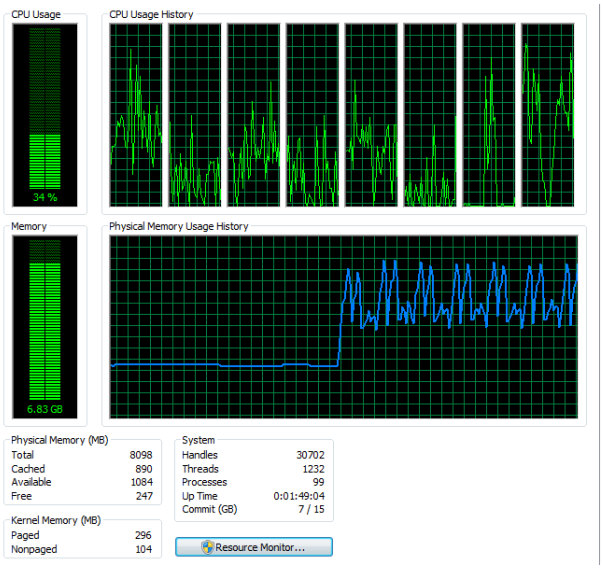


**Figure 4.3.** Screenshot of Memory and CPU usage during vectorized and partitioned DDP optimization

Figure 4.3 shows the CPU Usage and Memory Storage histories during optimization for this vectorized and partitioned case. The screen capture is taken during the seventh time stop that is optimized. The Memory Storage history shows that during each optimization step, memory is quickly created as the underlying model is simulated and then quickly removed to make room for the next simulation. The CPU Usage history shows that all eight processing units are consistently being utilized to their full potential as the vectorized operations are being performed in parallel. The physical memory limit is never reached, so there are no slow memory clearing process and total optimization time is only determined by actual computation time. As a result of the

vectorization and partitioning strategy, the average optimization time per simulation time step is reduced to 19.5 seconds, and optimization for the entire UDDS cycle can be performed in 7.4 hours. Table 4.2 summarize the results of this case study.

In this particular case study, it is sufficient to partition the vectorized mesh space into two parts. This strategy is extendable to denser discretization and additional state variables because larger mesh spaces can be accommodated simply by increasing the number of parts.

### 4.2.2 Case Study 2: Effect of Reduced Mesh Space Dimension on Optimization Time

This case study demonstrates the reduction of optimization time afforded by eliminating M/G torque as an independent control input, as described in Chapter 2. To do this, we optimize a power split PHEV's powertrain performance over the UDDS cycle using two different DDP algorithms, one with M/G torque as a meshed variable, and one without. Both algorithms use mesh space vectorization, and partitioning is not necessary for this level of discretization. The objective of each optimization is to minimize fuel consumption over the drive cycle. For the sake of direct comparison, engine start/stop control is removed as a control input and the engine remains on the entire time. The results of this case study are summarized in Table 4.4.

The first optimization is posed as a problem with four discretized variables. There are two dynamic states ($SOC$ and $\omega_{ce}$) and two control inputs ($T_e$ and $T_{mg1}$). Using the mesh sizes listed in Table 4.4, there are over 16 million possible combinations of states and inputs. The mesh space is vectorized, making use of Matlab's ability to perform efficient matrix calculations to solve the dynamics equations in parallel, but due to the large number of operations, optimization takes more than twelve times real-time to generate the optimal control policy, as shown in Table 4.4.

To reduce the optimization time, we can reduce the total number of operations that must be performed by eliminating one of the control inputs. Chapter 2 showed that using complete *a priori* knowledge of the vehicle's velocity, it is unnecessary to discretize either M/G torque as

**Table 4.4.** Case Study 2: DDP Parameters and Results

| | $\#\omega_{ce}\,pts$ | $\#SOC\,pts$ | $\#T_e\,pts$ | $\#T_{mg1}\,pts$ | $\gamma$ |
|---|---|---|---|---|---|
| 4 Discretized Variables | 50 | 201 | 40 | 41 | 12.7474 s |
| 3 Discretized Variables | 50 | 201 | 40 | X | 0.4779 s |

a control input. If engine torque is used as an independent control input, both M/G torques are dependent inputs that can be calculated using the vehicle velocity, acceleration, and power demand in conjunction with the powertrain dynamic equations. With this formulation of the DDP algorithm (and neglecting engine start/stop), there are three discretized variables: $SOC$, $\omega_{ce}$, and $T_e$. To demonstrate how the run-time improvement using this new formulation, we perform a second optimization for the same vehicle traversing the same drive cycle. Using the parameters listed in Table 4.4, there are now only 402,000 combinations of states and inputs that need to be solved at each time step. Due to the reduced number of computations, the optimization algorithm can now find the optimal control policy faster than real time, while achieving the same level of discretization accuracy.

### 4.2.3 Case Study 3: Examining the Significance of SOC Mesh Density

The goal of this case study is to examine the effect of SOC mesh density on the accuracy of the system dynamics and optimal control policy determined using the DDP algorithm. To perform this study, we optimize PHEV powertrain performance for fuel economy over the UDDS drive cycle by applying the DDP algorithm multiple times, varying the number of points on the SOC mesh vector and holding all other mesh vectors constant. The results show that for this PHEV model and the DDP algorithm used, over one thousand discrete SOC mesh points must be used to accurately predict the fuel economy and battery pack depletion over the drive cycle.

The previous two sections in this chapter demonstrate the substantial improvements in computation time and memory use that can be achieved using the methods presented in this paper. As a result of these improvements, the model is amenable to finer discretization of the state and input variables. It is expected that finer discretization results in more accurate predictions of the PHEV's performance for a given drive cycle for two reasons. First, the underlying model is solved using discrete time forward Euler integration. Second, cost-to-go and control policy information can only be stored for points on the mesh space. If this information is stored too sparsely, the algorithm may predict optimal fuel and battery usage that cannot be physically achieved.

The density of the battery state of charge mesh is of particular interest in the application of DDP to optimal PHEV powertrain control because control decisions are dependent on $SOC$, and the change in $SOC$ across each time step is determined by the discrete model. To examine the impact of $SOC$ mesh density on the accuracy of the control policy, we compare the fuel economy

and battery depletion predicted by the DDP algorithm using a range of $SOC$ mesh densities. In this case study we use the DDP algorithm outlined in Section 3.7 with $\omega_{ce}$ and $SOC$ as discrete state variables and $T_e$ and *Engine On/Off* as independent control inputs. The drive cycle used is the UDDS standard drive cycle, and the parameters for this study are listed in Table 4.5. The number of points used for the $SOC$ mesh vector vary from 221 points (coarse) to 1621 points (fine), and the number of points used for each other discrete variable is constant.

**Table 4.5.** Case Study 3: DDP Parameters

| Discrete Variable | Min | Max | n |
|---|---|---|---|
| $E_{IO}$ | 0 | 1 | 2 |
| $T_e$ | 0 N·m | 110 N·m | 50 |
| $\omega_{ce}$ | 100 rad/s | 500 rad/s | 50 |
| $SOC$ | 0.3 | 0.8 | 221:200:1621 |

To begin this study, we optimize PHEV powertrain performance for fuel economy over the UDDS drive cycle using DDP with a very coarse $SOC$ mesh vector of 221 points. The predicted success of the control strategy in using the battery to minimize fuel consumption is summarized by the amount of battery energy that is consumed in order to avoid using the engine, and the fuel economy of the PHEV once the battery pack has been depleted. We measure this predicted success using two metrics. First, we simulate the drive cycle with the PHEV's battery pack initially at full charge (80%) and record the net $SOC$ depletion at the end of the drive cycle. Second, we simulate the drive cycle with the PHEV's battery pack initially at minimum acceptable state of charge (30%) and record the fuel economy attained over the drive cycle in miles per gallon. These two metrics are theoretical predictions, and are only as accurate as the underlying discrete model and the level of discretization, or mesh density, used in their calculation. To assess the accuracy of these predictions (without access to an actual PHEV for validation), we increase the SOC mesh density and look for convergence in the predicted $SOC$ depletion and fuel economy. To this end, we increase the number of points on the $SOC$ mesh vector by 200, and repeat until convergence is achieved. The results of this process are depicted in Figure 4.4.

As the results in Figure 4.4 show, the predicted PHEV performance decreases sharply as the number of $SOC$ mesh points is increased from 221 to 421. We infer from this sharp contrast that the fuel economy and minimal battery depletion predicted using 221 mesh points are not phys-

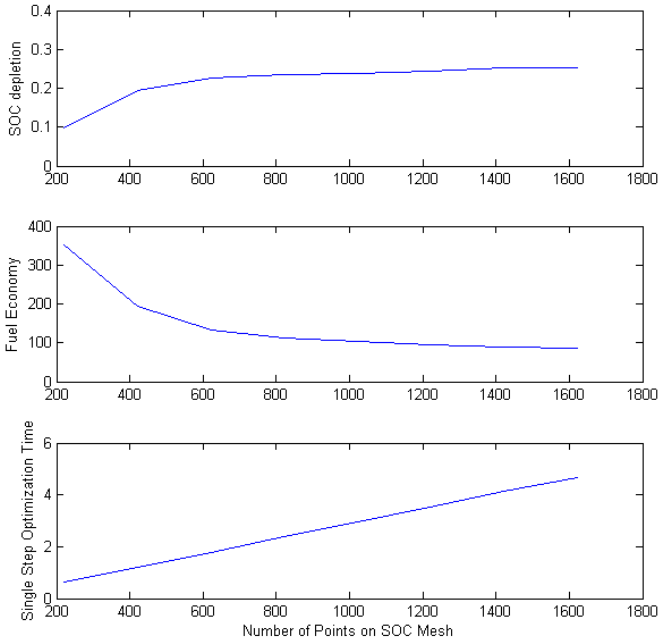


**Figure 4.4.** Comparison of PHEV performance over a single UDDS cycle, as predicted using different SOC mesh densities

ically achievable or meaningful. As the number of mesh points increases further, the predicted fuel economy decreases and the predicted battery pack depletion increases. These predictions are less optimal – since the objective is to minimize fuel consomuption – but are more accurate because the underlying model is being solved with improved tolerance due to the finer discretization. If we continue to increase the number of $SOC$ mesh points, we find the $SOC$ depletion from full charge predicted using 1221 points is within 5% of the depletion predicted using 1421 points. Similarly, the fuel economy predicted when the battery pack is initially at its lower limit with 1421 points is within 5% of the fuel economy predicted using 1621 points. The true value of optimal fuel economy and battery pack depletion are still unknown, but without a means of physically validating the predicted PHEV performance, improving the simulation accuracy any further is moot.

We can gain more insight into the effect of $SOC$ mesh density on the accuracy of the DDP algorithm and model simulation by examining state trajectories as the optimal control policy is applied to the PHEV traversing the UDDS drive cycle. Figure 4.5 shows trajectories for the vehicle's velocity, battery pack power, SOC, and fuel consumed by the engine during the

drive cycle when the battery pack is initially fully charged, generated using DDP with 221 $SOC$ mesh points. Figure 4.6 shows the corresponding trajectories using DDP with 1621 $SOC$ mesh points. The second subplot of each figure shows that the battery is continually being used as power is flowing into or out of the pack, depending on the use of the two electric machines. As expected, the $SOC$ subplot of Figure 4.6 shows that the battery is continually being discharged or replenished to correspond with the power flowing through the battery. In contrast, the $SOC$ trajectory shown in Figure 4.5 contains regions where $SOC$ remains constant despite substantial power flow through the battery. This occurs because the change in $SOC$ that results from the battery power is not large enough to be observed in the discrete state space. As a result, the $SOC$ depletion predicted is underestimated. Furthermore, the control policy is developed in a way that allows for the electric machines to be used to drive the vehicle without any observed loss of energy stored in the battery, despite the clear violation of the law of conservation of energy.

We can make similar observations by studying the relevant PHEV state and input trajectories for the case of the vehicle traversing the UDDS drive cycle with its battery pack already fully discharged at the beginning of the cycle. The results for this scenario generated using 221 and 1621 $SOC$ mesh points are shown in Figure 4.7 and Figure 4.8, respectively. In Figure 4.8, the fuel consumption subplot shows that the engine is being used substantially for the first half of the drive cycle as the battery is being recharged to store electrochemical energy for later use. At its peak, the battery pack reaches a 37% state of charge. After that point, the engine is shut down and the vehicle is powered using the available charge throughout the remainder of the cycle. As a result, the predicted fuel economy during this trip is 86.6 miles per gallon. In contrast, Figure 4.7 shows that the battery pack reaches a maximum charge of only 32%, after which point the engine is turned off and the PHEV is able to complete the rest of the drive cycle in all-electric mode because the power flowing out of the battery does not result in an observed depletion of $SOC$. The result of this simulation is a predicted fuel economy of 352 miles per gallon, which we know by comparison to be very innacurate. This predicted performance is only possible because the DDP algorithm using a coarse $SOC$ state mesh results in an invalid optimal control policy that continually draws "free" power from the battery in order to avoid using the engine.

This case study shows that the viability of a control strategy developed using a DDP algorithm is strongly dependent on the density of the $SOC$ state vector. With a coarse mesh (221 points), the SOC dynamics are not captured accurately, and as a result a control policy is developed that

relies on the use of the eletric machines without expecting substantial battery state of charge depletion. Such a control strategy promises overly optimistic PHEV performance that can not actually be achieved. In order to achieve accurate performance predictions and develop a viable control strategy, a fine $SOC$ state mesh must be used. By using 1421 mesh points, we can achieve 95% confidence in the fuel economy and battery depletion predicted using the DDP algorithm developed in this thesis.

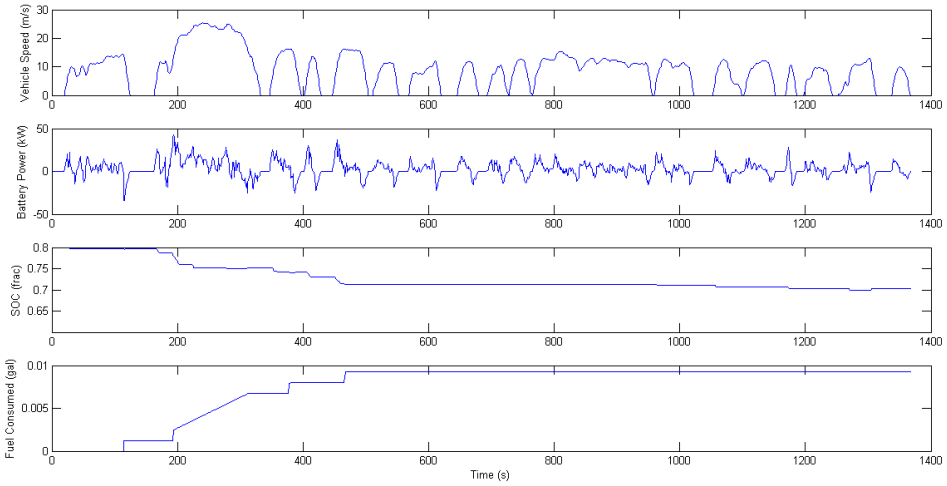


**Figure 4.5.** PHEV performance over the UDDS cycle with $SOC_0 = SOC_{max}$, predicted using 221 mesh points

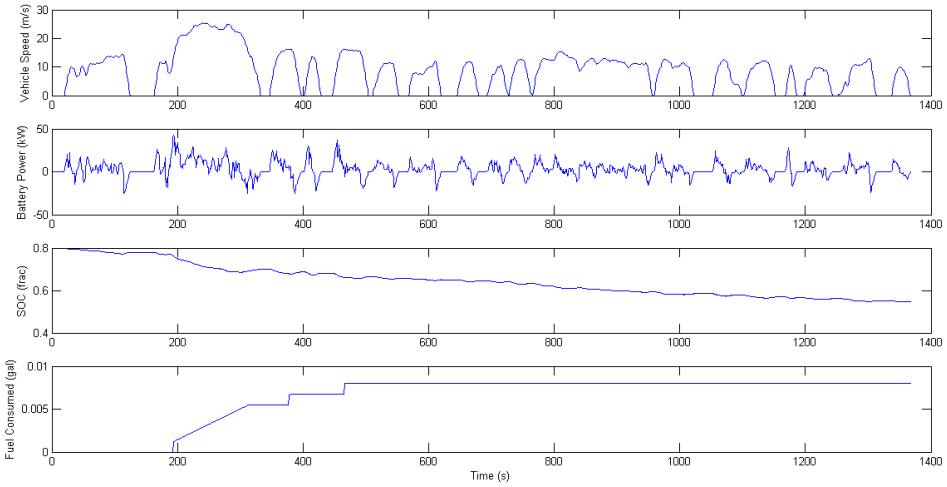


**Figure 4.6.** PHEV performance over the UDDS cycle with $SOC_0 = SOC_{max}$, predicted using 1621 mesh points

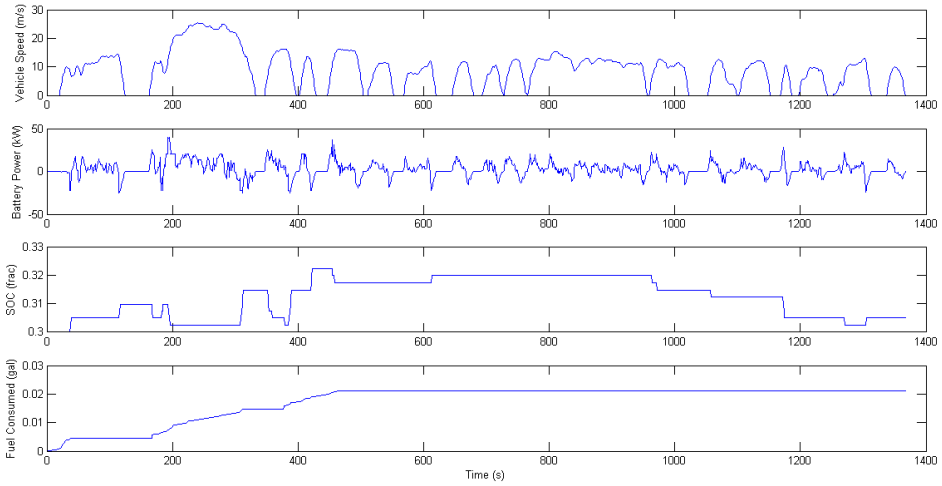


**Figure 4.7.** PHEV performance over the UDDS cycle with $SOC_0 = SOC_{min}$, predicted using 221 mesh points

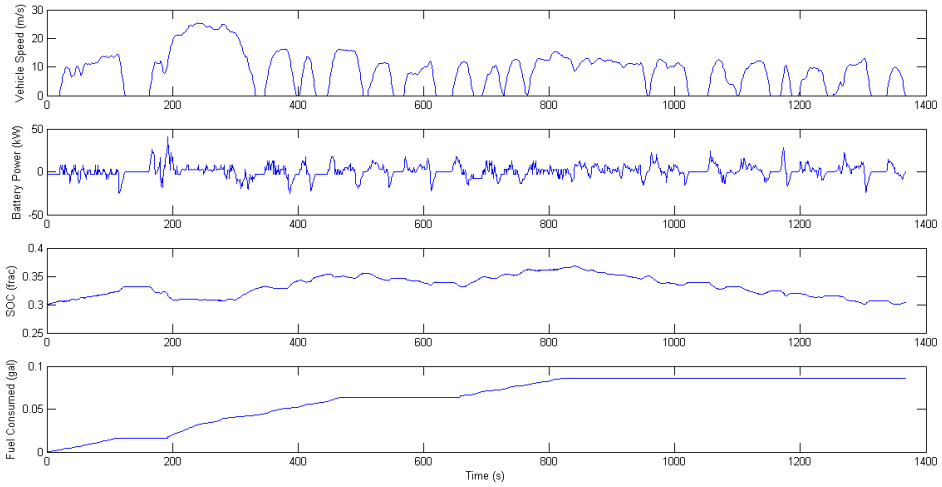


**Figure 4.8.** PHEV performance over the UDDS cycle with $SOC_0 = SOC_{min}$, predicted using 1621 mesh points

# Chapter 5

# Conclusions

The two main goals that this thesis aims to accomplish are

1. To develop an efficient and accessible formulation of the DDP algorithm applied to the power split PHEV power management problem
2. To demonstrate the need to use very dense $SOC$ mesh sizes to capture the system dynamics accurately

Both of these objectives are critically important in the area of optimal PHEV supervisory power management. DDP is a powerful tool that can develop a drive cycle specific control policy that guarantees a globally optimal solution to the objective function. This optimal control policy can be used as a benchmark to evaluate the success of an implementable control strategy. DDP can also be used as a tool in the optimal design process, because the PHEV design parameters are easily tunable and the optimal control policy will determine the potential of a specific PHEV.

DDP is also an excellent tool for multi-objective optimization problems, because the additive cost function can include and trade-off multiple cost functions. As a result of their promising potential (improved fuel economy, reduced emissions) and the degrees of freedom in the powertrain (multiple energy storage devices, multiple power conversion devices), PHEV power management is by very nature a multi-objective optimization problem.

Using DDP to optimize powertrain performance for multiple objectives has been a difficult challenge because applications of DDP to this problem have already pushed the boundaries of computing power and memory storage that is available in modern personal computers. Control objectives such as reduction of emissions or minimization of battery degradation are functions of dynamic states that are not typically included in a powertrain model, and each new state added to the model increases the numerical complexity of the problem exponentially.

To address this challenge directly, this thesis develops a DDP algorithm that is tailored to the power-split PHEV optimal power management. To improve the efficiency of this of the algorithm, this thesis has

1. Reduced the dimension of the input mesh space to exponentially reduce the numerical

complexity

2. Employed mesh space *vectorization* to maximize the speed of computations
3. And employed mesh space *partitioning* to maximize the use of physical memory without exceeding the computer's RAM limits

As a result of these numerical acceleration methods, the DDP algorithm developed in this thesis

1. Is capable of running in near-real-time
2. Is well-suited to handling dense state and input meshes
3. Is amenable to the addition of state variables and optimization objectives.

The end deliverable of this thesis is a powerful tool that opens the door to many opportunities for exploration, experimentation, and optimization in the realm of PHEV supervisory power management.

# Appendix A

# MATLAB Code - Primary DDP Scripts

## A.1 DDP Main Script

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DDP Main Code - PHEV Optimal Supervisory Power Management
% Written by: Tim Montgomery        2012.07.03
%
% States: Engine speed, battery state of charge
% Inputs: Engine I/O, engine torque
%
% Revised by: Tim Montgomery        2012.11.30
%    Full, analytical models of Engine Start/Stop included
%
% Objective: Minimize Fuel Consumption
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Initialize
clear all
close all
clc

%% Check Memory Status At Initiliazation
[UserView SystemView] = memory;
MemMatlab_0 = UserView.MemUsedMATLAB/1024^3; %(GB) Memory currently being used by Matlab
RAMavail_0 = SystemView.PhysicalMemory.Available/1024^3; %(GB) RAM currently available

%% Pick Drive Cycle DDP Parameters
CYC_name = 'CYC_UDDS';
model_rev = 'IO_1621';
mesh_style = 'm50162150';
% numparts = 1;
% MaxNumVars = 32;

% Segment to be tested
t_start = 1; % ENTER 1 to start at beginning
t_end = 0; % ENTER 0 to test to end of cycle

%% Begin!
disp('DETERMINISTIC DYNAMIC PROGRAMMING FOR PHEV POWER MANAGEMENT')
disp('Please do not disturb!')
disp('---------------------------------------')
disp(datestr(clock,0))
tic;

% Load Parameters
param_veh;
param_batt;
param_ICE;
param_MG;
param_trans;

% Load Drive Cycle
dt = 1;     % 1 second time step, suitable for supervisory power mgmnt
load(CYC_name);
```

```
if t_end == 0
    v_mph = cyc_mph(t_start:end,2);
else
    v_mph = cyc_mph(t_start:t_end,2);
end
v = v_mph * mph2mps;
v_shift = zeros(size(v));
v_shift(1:end-1) = v(2:end);
v_dot = v_shift - v;
v_dot(end) = v_dot(end-1);
N = length(v);

%% Mesh the State and Input Variables
param_DDP;
gen_mesh_file = strcat('gen_mesh_IO_',mesh_style);
run(gen_mesh_file);
make_mesh_space_IO;

%% Create State Configuration Matrix

% x1 = omega_ce (engine/planet carrier gear speed, rad/s)
% x2 = SOC      (battery pack state of charge, fraction)

x1 = repmat(omega_ce_vec',SOC_pts,1);
x1 = x1(:);

x2 = repmat(SOC_vec,omega_ce_pts,1);

Xconfig = [x1 x2];

Nconfigs = size(Xconfig,1);

%% Create Input Policy Matrix

% u1 = EIO      (Engine ON/OFF control decision)
% u2 = trq_e    (Engine torque, N*m)

u1 = EIO_vec;
u2 = trq_e_vec;

% The matrix of FEASIBLE input policies is not a full factorial
% trq_e can only be non-zero when the engine is on

Upol = [0 0; ones(trq_e_pts,1) trq_e_vec];

Npols = size(Upol,1);

%% Initialize matrices for the Optimal Cost-2-Go, C2G, and the corresponding Optimal Control Policy

C2G_opt = 1e6 * ones(length(Xconfig),N);
C2G_opt(:,N) = 0;
Upol_opt = zeros(length(Xconfig),N-1);

%% Check Memory Status After Overhead (OH)
% [UserView SystemView] = memory;
% MemMatlab_OH = UserView.MemUsedMATLAB/1024^3; %(GB) Memory currently being used by Matlab
% RAMavail_OH = SystemView.PhysicalMemory.Available/1024^3; %(GB) RAM currently available


%% Actual DDP!
% Stepping backwards from time step N, for every step in time
    % Use drive cycle to complete vehicles present state
    % Simulate appropriate model for every FEASIBLE point in the 4dim mesh space
    % Compute Xnext and trans_cost for each
```

```
    % Calculate C2G(k) = trans_cost + C2G(k+1,Xnext)
    % For each point on the STATE mesh, find the point on the INPUT mesh
    % for which C2G(k) is minimum
        % C2G_opt(k,Xconfig) = min(C2G(k))
        % Save the corresponding input policy
% Step back in time and repeat


TimeStep = zeros(N-1,1);

%% Start with N-1 Step and loop backwards to N = 1

for step = N-1:-1:1

    ticStep = tic;

    disp(['Current Step: ' num2str(step)])

    % Get info from drive cycle
    V = v(step);
    V_dot = v_dot(step);


    %% Split mesh_space into the appropriate cases and run the models seperately
    % Instead of creating the 4D mesh space blindly and needing to identify
    % the proper case for each mesh point... create the mesh space
    % strategically and use appropriate indexing.

    % Case: OFF2OFF
    [omega_ce_next_OO SOC_next_OO trans_cost_OO] = model_DDP_OFF2OFF(mesh_OO(:,1),mesh_OO(:,2),mesh_OO(:
    X_next_NN_ind_OO = nearest_neighbor_IO(omega_ce_vec,SOC_vec,omega_ce_next_OO,SOC_next_OO);
    C2G_OO = trans_cost_OO + C2G_opt(X_next_NN_ind_OO,step+1);

    % Case: OFF2ON
    [omega_ce_next_OI SOC_next_OI trans_cost_OI] = model_DDP_OFF2ON(mesh_OI(:,1),mesh_OI(:,2),mesh_OI(:,
    X_next_NN_ind_OI = nearest_neighbor_IO(omega_ce_vec,SOC_vec,omega_ce_next_OI,SOC_next_OI);
    C2G_OI = trans_cost_OI + C2G_opt(X_next_NN_ind_OI,step+1);

    % Case: ON2OFF
    [omega_ce_next_IO SOC_next_IO trans_cost_IO] = model_DDP_ON2OFF(mesh_IO(:,1),mesh_IO(:,2),mesh_IO(:,
    X_next_NN_ind_IO = nearest_neighbor_IO(omega_ce_vec,SOC_vec,omega_ce_next_IO,SOC_next_IO);
    C2G_IO = trans_cost_IO + C2G_opt(X_next_NN_ind_IO,step+1);

    % Case: ON2ON
    [omega_ce_next_II SOC_next_II trans_cost_II] = model_DDP_ON2ON(mesh_II(:,1),mesh_II(:,2),mesh_II(:,3
    X_next_NN_ind_II = nearest_neighbor_IO(omega_ce_vec,SOC_vec,omega_ce_next_II,SOC_next_II);
    C2G_II = trans_cost_II + C2G_opt(X_next_NN_ind_II,step+1);

    %% Create the complete C2G matrix for this time step
    % Every column is a State Configuration (Xconfig)
    % Every row is a feasible Input Policy (Upol)

    C2G = 1e6*ones(Npols,Nconfigs);

    C2G(1,1:SOC_pts) = C2G_OO';
    C2G(2,1:SOC_pts) = C2G_OI';
    C2G(1,SOC_pts+1:end) = C2G_IO';
    C2G(2:end,SOC_pts+1:end) = reshape(C2G_II,trq_e_pts,(omega_ce_pts-1)*SOC_pts);


    % Select the optimal control input for each State Configuration
    [min_C2G ind_Upol] = min(C2G);
    C2G_opt(:,step) = min_C2G';
    Upol_opt(:,step) = ind_Upol';
```

```
    TimeStep(N-step,1) = toc(ticStep);

    if(step == N-1)
        ProjectedTotalTime = TimeStep(1) * (N-1)/60
    end
end

disp(' ')
disp('Dynamic Programming Complete!')

TotalTimeMins = toc/60


[MinCost MinInd] = min(C2G_opt(:,1));

if MinCost > 1000
    Result = 'No Feasible Path :('
else
    Result = 'There Is A Solution!'
    disp(strcat('Minimum Cost is:  ',num2str(MinCost),'  gal'))
end

if any(isnan(C2G_opt(:,1)))
    disp('There are NaNs in the C2G matrix!')
end


if step==1
save(strcat('SOL_',model_rev,'_',CYC_name,'_',num2str(t_start),'to',num2str(t_end),'s_',mesh_style),'C2G
disp('DDP Results Saved')
disp(strcat('SOL_',model_rev,'_',CYC_name,'_',num2str(t_start),'to',num2str(t_end),'s_',mesh_style));
end
```

## A.2 DDP Parameters

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generate State Space and Input Mesh Ranges, and Physical Constraints
% Written by: Tim Montgomery        2012.07.08
% Revised (heavily) 2012.10.08
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Battery Power Constraints

P_batt_ch_lim = -35e3;
P_batt_dis_lim = 40e3;

%% Battery SOC
% SOC_map =[0 10 20 40 60 80 100]/100;

SOC_min = 0.3;
SOC_max = 0.8;

% Mesh limits may be refined when mesh is generated
SOC_mesh_min = 0.29;
SOC_mesh_max = 0.81;

%% Engine Speed
% fuel_map_spd_rpm = [1000 1250 1500 1750 2000 2250 2500 2750 3000 3250 3500 4000];
% fuel_map_spd = fuel_map_spd_rpm*2*pi/60;
```

```
omega_e_min = 1000 *2*pi/60;    % 1000 rpm = 104.7198 rad/s
omega_e_max = 4000 *2*pi/60;    % 4000 rpm = 418.8790 rad/s

% Mesh limits may be refined when mesh is generated
omega_ce_mesh_min = 750 *2*pi/60;   % 750 rpm = 78.5394 rad/s
omega_ce_mesh_max = 4250 *2*pi/60;  % 4250 rpm = 445.0590 rad/s

% %% M/G1 Speed
% spd_g = [-5500 -4000 -3500 -3000 -2500 -2000 -1500 -1000 -500 0 500 ...
%     1000 1500 2000 2500 3000 3500 4000 5500 6000]*(2*pi)/60;
%
% spd_g_min = min(spd_g);
% spd_g_max = max(spd_g);
%
% omega_s1_mesh_min = -5750*(2*pi)/60;
% omega_s1_mesh_max = 6250*(2*pi)/60;

%% Engine Torque - physical constraints defined by param_ICE
lbft2Nm = 1.356; %conversion from lbft to Nm
% fuel_map_trq_lbft = [3.15 6.3 12.5 18.8 25.1 31.3 37.6 43.9 50.1 56.4 62.7 68.9 75.2];
% Trq (lb-ft)
% fuel_map_trq = fuel_map_trq_lbft*lbft2Nm;

trq_e_min = 0 *lbft2Nm;
trq_e_max =  75.2 *lbft2Nm;

trq_e_mesh_min = 0 *lbft2Nm;    % 2 lb-ft = 2.7120 N*m
trq_e_mesh_max = 78 *lbft2Nm;   % 78 lb-ft = 105.7680 N*m

%% M/G1 Torque
% trq_g = [-55 -45 -35 -25 -15 -5 0 5 15 25 35 45 55];

trq_mg1_mesh_min = -65; % N*m
trq_mg1_mesh_max = 65;  % N*m
```

## A.3 Generate Mesh Vectors

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generate State Space Mesh, Input Mesh, and Constraint Parameters
% Written by: Tim Montgomery        2012.07.08
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Define Mesh Sizes
omega_ce_pts = 50;
SOC_pts = 1621;
EIO_pts = 2;
trq_e_pts = 50;

%% Select State Mesh mins and maxs
% To make sure we mesh outside the feasible limits, but include the
% feasible limits on the mesh

buffer_omega_ce = 3;
n_main_omega_ce = omega_ce_pts - 2*buffer_omega_ce -1;
d_omega_ce = (omega_e_max - omega_e_min)/(n_main_omega_ce-1);
omega_ce_mesh_min = omega_e_min - buffer_omega_ce*d_omega_ce;
omega_ce_mesh_max = omega_e_max + buffer_omega_ce*d_omega_ce;

buffer_SOC = 10;
```

```
n_main_SOC = SOC_pts - 2*buffer_SOC;
d_SOC = (SOC_max - SOC_min)/(n_main_SOC-1);
SOC_mesh_min = SOC_min - buffer_SOC*d_SOC;
SOC_mesh_max = SOC_max + buffer_SOC*d_SOC;

%% Generate Grid Vectors
omega_ce_vec = [0 linspace(omega_ce_mesh_min,omega_ce_mesh_max,omega_ce_pts-1)]';
SOC_vec = linspace(SOC_mesh_min,SOC_mesh_max,SOC_pts)';
EIO_vec = [0 1]';
trq_e_vec = linspace(trq_e_mesh_min,trq_e_mesh_max,trq_e_pts)';
```

## A.4 Generate Full Mesh Space

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generate mesh matrix that includes all FEASIBLE State/Input Combinations*
% Written by: Tim Montgomery 2012.10.16
%
% * When engine I/O is considered, there are 2 states and 2 inputs, but the
% full 4 dimensional mesh space does not need to be tested,
% because trq_e (u2) does not always need to be meshed, because it can only
% be non-zero of the engine is on at step k AND step k+1.
%
% The full FEASIBLE mesh space looks like this:
%
% Case OFF2OFF: omega_ce = 0       SOC = meshed  EIO = 0  trq_e = 0
%
% Case OFF2ON:  omega_ce = 0       SOC = meshed  EIO = 1  trq_e = 0
%
% Case ON2OFF:  omega_ce = meshed  SOC = meshed  EIO = 0  trq_e = 0
%                          (w/out 0)
% Case ON2ON:   omega_ce = meshed  SOC = meshed  EIO = 1  trq_e = meshed
%                          (w/out 0)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x1 = omega_ce_vec;
x2 = SOC_vec;
u1 = EIO_vec;
u2 = trq_e_vec;

nx1 = numel(x1);
nx2 = numel(x2);
nu1 = numel(u1);
nu2 = numel(u2);

mesh_OO = zeros(nx2,4);
mesh_OO(:,2) = SOC_vec;

mesh_OI = mesh_OO;
mesh_OI(:,3) = 1;

mesh_IO = zeros((nx1-1)*nx2,4);
mesh_IO(:,1) = reshape(repmat(x1(2:end)',nx2,1),(nx1-1)*nx2,1);
mesh_IO(:,2) = repmat(x2,nx1-1,1);

mesh_II = ones((nx1-1)*nx2*nu2,4);
mesh_II(:,1) = reshape(repmat(x1(2:end)',nx2*nu2,1),(nx1-1)*nx2*nu2,1);
mesh_II(:,2) = repmat(reshape(repmat(x2',nu2,1),nx2*nu2,1),nx1-1,1);
mesh_II(:,4) = repmat(u2,(nx1-1)*nx2,1);

mesh_space = [mesh_OO; mesh_OI; mesh_IO; mesh_II];
```

```
space_length = length(mesh_space);
```

## A.5 Solve Engine OFF Model

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PHEV Discrete Time Dynamic Model
% Case: OFF2OFF
% Written by: Tim Montgomery        2012.11.29
%
% In this case, the engine is initial OFF, and our control decision is to
% keep the engine OFF
%
% Inputs: SOC, omega_ce, trq_e (% of Pdem met by engine), EIO, v, v_dot
% Outputs: SOC_next, omega_ce_next, trans_cost
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% function [SOC_next omegas dots omegas_next trqs powers trans_cost] = model_OFF2OFF(SOC,omega_ce,EIO,tr
function [omega_ce_next SOC_next trans_cost] = model_DDP_OFF2OFF(omega_ce,SOC,EIO,trq_e,v,v_dot)

% Make sure we are in the right Case... we can get rid of this later
if ¬(omega_ce == 0 & trq_e == 0 & EIO == 0)
    disp('Evaluating Wrong Case!')
end

long = length(omega_ce);  % to be used to size paired vectors
% part = evalin('base','part');
% numparts = evalin('base','numparts');

% Load Parameters
param_veh;
param_trans;
param_MG;
param_ICE;
param_DDP;

% Time Step
dt = 1;     % 1 second time step, suitable for supervisory power mgmt

% Initialize Transition Cost
trans_cost = zeros(long,1);

%% Universal Calculations

% Calculate road loads
F_roll = mu_roll*m*g .* (v≠0);      % Rolling Resistance (N)
F_drag = 0.5*rho*A_fr*C_d*v^2;      % Aerodynamic Drag (N)
F_damp = (Cw/R_tire)*v;             % Wheel/Bearing Damping (N)
F_road = F_roll + F_drag + F_damp;  % Total Road Load (N)
trq_road = F_road*R_tire;           % Torque from road loads at FINAL DRIVE

% Force/Torqe for Driveability
% m*v_dot = F_drive - F_road
F_drive = F_road + m*v_dot; % (N) Force required at tire/ground to overcome
                            % road loads and achieve acceleration
trq_drive = F_drive*R_tire; % (N*m) Torque required at FINAL DRIVE

% Power Demand for Driveability
P_dem = F_drive*v;  % (Watts) Total power required for Driveability

% ** If the vehicle is accelerating from rest, this equation is invalid!
```

```
% ** The energy required can be calculated later and taken from the battery

% Backtrack vehicle velocity and acceleration through drive train
omega_r2 = K*v/R_tire;
omega_r2_dot = K*v_dot/R_tire;

% Use PGS Kinematics for Velocity
omega_s1 = 1/S*(omega_ce*(R+S) - omega_r2*R);


%% OFF2OFF Specific Calculations

omega_ce_dot = 0;
trq_e = 0;

% Use PGS kinematics for acceleration
omega_s1_dot = -R/S*omega_r2_dot;


% P_rev = I_mg1*omega_s1*omega_s1_dot + I_e*omega_ce*omega_ce_dot + I_mg2*omega_r2*omega_r2_dot;
% P_dem = P_dem + P_rev;
P_rev = 0;

% Solve for the PGS internal reaction force, F, using Newton's Law at Sun
% and Ring nodes, and P_dem equation
% ** With the Carrier node locked, Fr = -Fs because trq_r = -R/S*trq_s **
% ** This follows from the gear ratio omega_s = -R/S*omega_r **
I_prime_mg2 = (R_tire/K)^2*m + I_mg2;

if omega_r2 == 0
    Fs = 0;
else
    Fs = 1./(-R*omega_r2 + S*omega_s1).*(I_mg1.*omega_s1_dot.*omega_s1 + I_prime_mg2.*omega_r2_dot.*omeg
P_dem);
end

% Calculate trq_mg1 using Newton's Law at Sun node
trq_mg1 = I_mg1*omega_s1_dot - Fs*S;

% Calculate trq_mg2 using Newton's Law at Ring node
I_prime_mg2 = (R_tire/K)^2*m + I_mg2;
trq_mg2_star = I_prime_mg2*omega_r2_dot + 1/K*trq_road + Fs*R;

% trq_mg2_star is the ideal value of trq_mg2, but is limited in practice
% If trq_mg2 star is feasible, trq_mg2 = trq_mg2_star
trq_mg2 = trq_mg2_star;

% If we need strong, negative torque, trq_mg2 is saturated and a friction
% brake, trq_fb(applied at wheel), takes care of the rest of the braking

trq_fb = zeros(size(trq_mg2_star)); % Do we ever use this? Delete?
ind = find(trq_mg2_star < trq_m_min);
if (any(ind))
    trq_mg2(ind) = trq_m_min;
    trq_fb(ind) = K*(trq_m_min - trq_mg2_star(ind));
end

% trq_m_min can violate P_mg2_min if speed is high
ind = find(trq_mg2.*omega_r2 < P_mg2_min);
if any(ind)
    trq_mg2(ind) = P_mg2_min./omega_r2;
    trq_fb(ind) = K*(trq_mg2(ind) - trq_mg2_star(ind));
end
```

```
% If we need strong, positive torque, trq_mg2 is saturated at trq_m_max
% This will need to be constrained!

% Euler Forward Integration
omega_ce_next = omega_ce + omega_ce_dot*dt;
omega_s1_next = omega_s1 + omega_s1_dot*dt;
omega_r2_next = omega_r2 + omega_r2_dot*dt;

% Powers
P_e = trq_e.*omega_ce;
P_mg1 = trq_mg1.*omega_s1;
P_mg2 = trq_mg2.*omega_r2;

%% Apply Constraints to Inputs

% ** Case: OFF2OFF - make sure the engine IS off
% Engine Torque
ind = find(trq_e ~= 0);
if any(ind)
    trans_cost(ind) = 1e6;
end

% % Engine Hot Max Open Throttle Torque as a function of engine speed
% % p_max_trq = evalin('base','p_max_trq');
% trq_e_max = polyval(p_max_trq,omega_ce);
% ind = find(trq_e > trq_e_max);
% if any(ind)
%     trans_cost(ind) = 1e6;
% end

% Engine Power
% For THIS engine, max power will never be violated

% M/G1 Torque
ind = find(trq_mg1 < trq_g_min | trq_mg1 > trq_g_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% M/G1 Power
ind = find(P_mg1 < P_mg1_min | P_mg1 > P_mg1_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% M/G2 Torque
ind = find(trq_mg2 < trq_m_min | trq_mg2 > trq_m_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% M/G2 Power
ind = find(P_mg2 < P_mg2_min | P_mg2 > P_mg2_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

%% Apply Constraints to Present States

% % Engine Speed (only needs to be constrained if engine is on)
% under = find(omega_ce < omega_e_min | omega_ce > omega_e_max );
% if any(under)
%     trans_cost(under) = 1e6;
% end
```

```
% M/G1 Speed
ind = find(omega_s1 < spd_g_min | omega_s1 > spd_g_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% % M/G2 Speed
% % Will only be violated if v > 102 mph
% ind = find(omega_r2 < spd_m_min | omega_r2 > spd_m_max);
% if any(ind)
%     trans_cost(ind) = 1e6;
% end

%% Apply Constraints to Next States

% Engine Speed at Step k+1
% ** THIS IS CASE SPECIFIC **
% OFF2OFF: omega_ce = omega_ce_next = 0;

% M/G1 Speed at Step k+1
ind = find(omega_s1_next < spd_g_min | omega_s1_next > spd_g_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% % M/G2 Speed at Step k+1
% % Will only be violated if v > 102 mph
% ind = find(omega_r2_next < spd_m_min | omega_r2_next > spd_m_max);
% if any(ind)
%     trans_cost(ind) = 1e6;
% end

%% Battery Model

% Load Parameters
param_batt

% Determine electrical power for M/Gs
% If P_mg is +, PE > P, PE = P*eff^-1, k = -1
% If P_mg is -, PE < P, PE = P*eff^1, k = 1     * k = - sign(P)

% Look up efficiency from map(torque,speed)
eff_mg1 = interp2(trq_g,spd_g,eff_g,trq_mg1,omega_s1,'*nearest',.1');
eff_mg2 = interp2(trq_m,spd_m,eff_m,trq_mg2,omega_r2,'*nearest',.1)';
% Extrapolating outside of the efficiency table results in NaN efficiencies
% So we replace with poor efficiencies. The Upol will be ruled infeasible anyway

PE_mg1 = P_mg1.*eff_mg1.^(-sign(P_mg1));
PE_mg2 = P_mg2.*eff_mg2.^(-sign(P_mg2));

% Total Power in (-) or out (+) of the battery
P_batt = PE_mg1 + PE_mg2;

% ** If the vehicle is accelerating from rest, P_dem was calculated to be 0
% ** Calculate power required to accelerate using change in Kinetic Energy
% ** Withdraw this energy from the battery (with a poor efficiency factor)
if v == 0
    P_batt = 2*(1/dt*(.5*I_e*omega_ce_next.^2 + .5*I_mg1*omega_s1_next.^2 + .5*I_prime_mg2*omega_r2_next
end

% Apply Constraints To Battery Power
ind = find(P_batt < P_batt_ch_lim | P_batt > P_batt_dis_lim);
if any(ind)
```

```
    trans_cost(ind) = 1e6;
end

% Apply Constraints To Current SOC
ind = find(SOC < SOC_min | SOC > SOC_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% Simulate pack model to calculate SOC_next
[SOC_next] = model_DDP_batt(SOC,P_batt);

% Apply Constraints To Next SOC
ind = find(SOC_next < SOC_min | SOC_next > SOC_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

%% Calculate Fuel Use

% OFF2OFF
gal_dot_fuel = 0;

%% Report Total Transition Cost
trans_cost = trans_cost + gal_dot_fuel;
```

## A.6 Solve Engine Start-up Model

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PHEV Discrete Time Dynamic Model
% Case: OFF2ON
% Written by: Tim Montgomery        2012.11.29
%
% In this case, the engine is initial OFF, and our control decision is to
% turn the engine ON
%
% ** THIS VERSION DOES NOT ACCOUNT FOR TRANSIENT DYNAMICS **
% This model meets the drive cycle speed, revs up the engine to minimum
% operable speed, and satisifying PGS kinematics
%
% The required M/G torques are not calculated, but it is generally accurate
% to assume the transition is feasible
%
% Both a fuel cost and SOC depletion are applied based on KE change
%
% Inputs: SOC, omega_ce, trq_e (% of Pdem met by engine), EIO, v, v_dot
% Outputs: SOC_next, omega_ce_next, trans_cost
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% function [SOC_next omegas dots omegas_next trqs powers trans_cost] = model_OFF2ON(SOC,omega_ce,EIO,trq
function [omega_ce_next SOC_next trans_cost] = model_DDP_OFF2ON(omega_ce,SOC,EIO,trq_e,v,v_dot)

% Make sure we are in the right Case... we can get rid of this later
if ¬(omega_ce == 0 & trq_e == 0 & EIO == 1)
    disp('Evaluating Wrong Case!')
end

long = length(omega_ce);  % to be used to size paired vectors
% part = evalin('base','part');
% numparts = evalin('base','numparts');
```

```
% Load Parameters
param_veh;
param_trans;
param_MG;
param_ICE;
param_DDP;

% Time Step
dt = 1;     % 1 second time step, suitable for supervisory power mgmt

% Initialize Transition Cost
trans_cost = zeros(long,1);

%% Universal Calculations

% Calculate road loads
F_roll = mu_roll*m*g * (v≠0);       % Rolling Resistance (N)
F_drag = 0.5*rho*A_fr*C_d*v^2;      % Aerodynamic Drag (N)
F_damp = (Cw/R_tire)*v;             % Wheel/Bearing Damping (N)
F_road = F_roll + F_drag + F_damp;  % Total Road Load (N)
trq_road = F_road*R_tire;           % Torque from road loads at FINAL DRIVE

% Force/Torqe for Driveability
% m*v_dot = F_drive - F_road
F_drive = F_road + m*v_dot; % (N) Force required at tire/ground to overcome
                            % road loads and achieve acceleration
trq_drive = F_drive*R_tire; % (N*m) Torque required at FINAL DRIVE

% Power Demand for Driveability
P_dem = F_drive*v;  % (Watts) Total power required for Driveability

% ** If the vehicle is accelerating from rest, this equation is invalid!
% ** The energy required can be calculated later and taken from the battery

% Backtrack vehicle velocity and acceleration through drive train
omega_r2 = K*v/R_tire;
omega_r2_dot = K*v_dot/R_tire;

% Use PGS Kinematics for Velocity
omega_s1 = 1/S*(omega_ce*(R+S) - omega_r2*R);


%% OFF2ON Specific Calculations

% The carrier lock is released, but the engine will not exert power until
% it is brought up to speed by the M/Gs
trq_e = 0;

% The engine will be brought up to speed in the next time step by the M/Gs
omega_ce_dot = 200/dt;
omega_ce_next = omega_ce + omega_ce_dot*dt;

% Use PGS Kinematics for Accleration
omega_s1_dot = 1/S*(omega_ce_dot*(R+S) - omega_r2_dot*R);

% Calculate the true power demand, which is the sum of the power demanded
% for drivability and the power to rev the engine up to speed
P_rev = I_mg1*omega_s1*omega_s1_dot + I_e*omega_ce*omega_ce_dot + I_mg2*omega_r2*omega_r2_dot;
P_dem = P_dem + P_rev;

% ** If the vehicle is accelerating from rest, this equation is invalid!
% ** The energy required can be calculated later and taken from the battery
```

```
% Calculate the PGS internal reaction force, F, using Newton's Law at Carrier node
F = -1/(R+S)*I_e*omega_ce_dot;

% Calculate trq_mg1 required to rev up engine using Newton's Law at Sun node
trq_mg1 = omega_s1_dot*I_mg1 - F*S;

% Calculate trq_mg2 using Newton's Law at Sun node
I_prime_mg2 = (R_tire/K)^2*m + I_mg2;
trq_mg2_star = I_prime_mg2*omega_r2_dot + 1/K*trq_road - F*R;

% trq_mg2_star is the ideal value of trq_mg2, but is limited in practice
% If trq_mg2 star is feasible, trq_mg2 = trq_mg2_star
trq_mg2 = trq_mg2_star;

% If we need strong, negative torque, trq_mg2 is saturated and a friction
% brake, trq_fb(applied at wheel), takes care of the rest of the braking

trq_fb = zeros(size(trq_mg2_star)); % Do we ever use this? Delete?
ind = find(trq_mg2_star < trq_m_min);
if (any(ind))
    trq_mg2(ind) = trq_m_min;
    trq_fb(ind) = K*(trq_m_min - trq_mg2_star(ind));
end

% trq_m_min can violate P_mg2_min if speed is high
ind = find(trq_mg2.*omega_r2 < P_mg2_min);
if any(ind)
    trq_mg2(ind) = P_mg2_min./omega_r2;
    trq_fb(ind) = K*(trq_mg2(ind) - trq_mg2_star(ind));
end

% If we need strong, positive torque, trq_mg2 is saturated at trq_m_max
% This will need to be constrained!

% Euler Forward Integration
omega_ce_next = omega_ce + omega_ce_dot*dt;
omega_s1_next = omega_s1 + omega_s1_dot*dt;
omega_r2_next = omega_r2 + omega_r2_dot*dt;

% Powers
P_e = trq_e.*omega_ce;
P_mg1 = trq_mg1.*omega_s1;
P_mg2 = trq_mg2.*omega_r2;


%% Apply Constraints to Inputs

% ** Case: OFF2ON - make sure engine IS off
% Engine Torque
ind = find(trq_e ~= 0);
if any(ind)
    trans_cost(ind) = 1e6;
end

% % Engine Hot Max Open Throttle Torque as a function of engine speed
% % p_max_trq = evalin('base','p_max_trq');
% trq_e_max = polyval(p_max_trq,omega_ce);
% ind = find(trq_e > trq_e_max);
% if any(ind)
%     trans_cost(ind) = 1e6;
% end

% Engine Power
% For THIS engine, max power will never be violated
```

```
% M/G1 Torque
ind = find(trq_mg1 < trq_g_min | trq_mg1 > trq_g_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% M/G1 Power
ind = find(P_mg1 < P_mg1_min | P_mg1 > P_mg1_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% M/G2 Torque
ind = find(trq_mg2 < trq_m_min | trq_mg2 > trq_m_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% M/G2 Power
ind = find(P_mg2 < P_mg2_min | P_mg2 > P_mg2_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

%% Apply Constraints to Present States

% % Engine Speed (only needs to be constrained if engine is on)
% under = find(omega_ce < omega_e_min | omega_ce > omega_e_max );
% if any(under)
%     trans_cost(under) = 1e6;
% end

% M/G1 Speed
ind = find(omega_s1 < spd_g_min | omega_s1 > spd_g_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% % M/G2 Speed
% % Will only be violated if v > 102 mph
% ind = find(omega_r2 < spd_m_min | omega_r2 > spd_m_max);
% if any(ind)
%     trans_cost(ind) = 1e6;
% end

%% Apply Constraints to Next States

% Engine Speed at Step k+1
% ** THIS IS CASE SPECIFIC **
% OFF2ON: omega_ce_dot = omega_ce_next = omega_e_min;

% M/G1 Speed at Step k+1
ind = find(omega_s1_next < spd_g_min | omega_s1_next > spd_g_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% % M/G2 Speed at Step k+1
% % Will only be violated if v > 102 mph
% ind = find(omega_r2_next < spd_m_min | omega_r2_next > spd_m_max);
% if any(ind)
%     trans_cost(ind) = 1e6;
% end
```

```
%% Battery Model

% Load Parameters
param_batt

% Calculate Mechanical Power
P_mg1 = omega_s1.*trq_mg1;
P_mg2 = omega_r2.*trq_mg2;

% Determine electrical power for M/Gs
% If P_mg is +, PE > P, PE = P*eff^-1, k = -1
% If P_mg is -, PE < P, PE = P*eff^1, k = 1     * k = - sign(P)

% Look up efficiency from map(torque,speed)
eff_mg1 = interp2(trq_g,spd_g,eff_g,trq_mg1,omega_s1,'*nearest',.1');
eff_mg2 = interp2(trq_m,spd_m,eff_m,trq_mg2,omega_r2,'*nearest',.1)';
% Extrapolating outside of the efficiency table results in NaN efficiencies
% So we replace with poor efficiencies. The Upol will be ruled infeasible anyway

PE_mg1 = P_mg1.*eff_mg1.^(-sign(P_mg1));
PE_mg2 = P_mg2.*eff_mg2.^(-sign(P_mg2));

% Total Power in (-) or out (+) of the battery
P_batt = PE_mg1 + PE_mg2;

% ** If the vehicle is accelerating from rest, P_dem was calculated to be 0
% ** Calculate power required to accelerate using change in Kinetic Energy
% ** Withdraw this energy from the battery (with a poor efficiency factor)

if v == 0
    P_batt = 2*(1/dt*(.5*I_e*omega_ce_next.^2 + .5*I_mg1*omega_s1_next.^2 + .5*I_prime_mg2*omega_r2_next
end


% ** We will NOT constrain Battery Power during Engine Start Events **
% ** Large spikes are undesirable but acceptable when the engine must start
% % Apply Constraints To Battery Power
% ind = find(P_batt < P_batt_ch_lim | P_batt > P_batt_dis_lim);
% if any(ind)
%     trans_cost(ind) = 1e6;
% end

% Apply Constraints To Current SOC
ind = find(SOC < SOC_min | SOC > SOC_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% Simulate pack model to calculate SOC_next
[SOC_next] = model_DDP_batt(SOC,P_batt);

% Regardless of P_dem, an engine start-up  event should always be associated
% with an observable drop in SOC
d_SOC = evalin('base','d_SOC');

SOC_next = min(SOC_next,SOC-d_SOC);



% Apply Constraints To Next SOC
ind = find(SOC_next < SOC_min | SOC_next > SOC_max);
if any(ind)
    trans_cost(ind) = 1e6;
```

```
end

%% Calculate Fuel Use

% OFF2OFF

% To turn the engine, we apply a fuel penalty by assuming that 3
% complete cycles occur before we receive power from the engine, so
% m_dot_fuel =

% gallons = grams * 1/1000 *1/rho * cum2gal
gal_dot_fuel = m_dot_fuel_start *(1/1000) * (1/rho_fuel) * cum2gal;
%% Report Total Transition Cost
trans_cost = trans_cost + gal_dot_fuel;
```

## A.7 Solve Engine Shut-down Model

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PHEV Discrete Time Dynamic Model
% Case: ON2OFF
% Written by: Tim Montgomery        2012.11.29
%
% In this case, the engine is initial ON, and our control decision is to
% turn the engine OFF
%
% Inputs: SOC, omega_ce, trq_e (% of Pdem met by engine), EIO, v, v_dot
% Outputs: SOC_next, omega_ce_next, trans_cost
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% function [SOC_next omegas dots omegas_next trqs powers trans_cost] = model_ON2OFF(SOC,omega_ce,EIO,trq
function [omega_ce_next SOC_next trans_cost] = model_DDP_ON2OFF(omega_ce,SOC,EIO,trq_e,v,v_dot)

% Make sure we are in the right Case... we can get rid of this later
if ¬(omega_ce ≠ 0 & trq_e == 0 & EIO == 0)
    disp('Evaluating Wrong Case!')
end

long = length(omega_ce);  % to be used to size paired vectors
% part = evalin('base','part');
% numparts = evalin('base','numparts');

% Load Parameters
param_veh;
param_trans;
param_MG;
param_ICE;
param_DDP;

% Time Step
dt = 1;     % 1 second time step, suitable for supervisory power mgmt

% Initialize Transition Cost
trans_cost = zeros(long,1);

%% Universal Calculations

% Calculate road loads
F_roll = mu_roll*m*g * (v≠0);      % Rolling Resistance (N)
F_drag = 0.5*rho*A_fr*C_d*v^2;      % Aerodynamic Drag (N)
F_damp = (Cw/R_tire)*v;             % Wheel/Bearing Damping (N)
```

```
F_road = F_roll + F_drag + F_damp;  % Total Road Load (N)
trq_road = F_road*R_tire;           % Torque from road loads at FINAL DRIVE

% Force/Torqe for Driveability
% m*v_dot = F_drive - F_road
F_drive = F_road + m*v_dot; % (N) Force required at tire/ground to overcome
                            % road loads and achieve acceleration
trq_drive = F_drive*R_tire; % (N*m) Torque required at FINAL DRIVE

% Power Demand for Driveability
P_dem = F_drive*v;  % (Watts) Total power required for Driveability

% Backtrack vehicle velocity and acceleration through drive train
omega_r2 = K*v/R_tire;
omega_r2_dot = K*v_dot/R_tire;

% Use PGS Kinematics for Velocity
omega_s1 = 1/S*(omega_ce*(R+S) - omega_r2*R);


%% OFF2ON Specific Calculations

% The engine is turning off, so we will not draw power from it
trq_e = 0;

% The goal is to bring the engine down to a low speed before the lock
% engages, sending it to 0 should be easy in 1 second
omega_ce_dot = -omega_ce/dt;

% Use PGS Kinematisc for Accleration
omega_s1_dot = 1/S*(omega_ce_dot*(R+S) - omega_r2_dot*R);



P_rev = I_mg1.*omega_s1.*omega_s1_dot + I_e.*omega_ce.*omega_ce_dot + I_mg2.*omega_r2.*omega_r2_dot;
P_dem = P_dem + P_rev;


% Calculate the internal reaction force, F, using Newton's Law at Carrier node
F = -1/(R+S)*I_e*omega_ce_dot;

% Calculate trq_mg1 using Newton's Law at Sun node
trq_mg1 = I_mg1*omega_s1_dot - F*S;

% Calculate trq_mg2 using Newton's Law at Ring node
I_prime_mg2 = (R_tire/K)^2*m + I_mg2;
trq_mg2_star_Newt = I_prime_mg2*omega_r2_dot + 1/K*trq_road - F*R;

% Calculate trq_mg2 using P_dem equation
trq_mg2_star_Pdem = 1/omega_r2*(P_dem - trq_mg1.*omega_s1);

trq_mg2 = trq_mg2_star_Newt;
% % trq_mg2_star is the ideal value of trq_mg2, but is limited in practice
% % If trq_mg2 star is feasible, trq_mg2 = trq_mg2_star
% trq_mg2 = trq_mg2_star;
%
% % If we need strong, negative torque, trq_mg2 is saturated and a friction
% % brake, trq_fb(applied at wheel), takes care of the rest of the braking
%
% trq_fb = zeros(size(trq_mg2_star)); % Do we ever use this? Delete?
% ind = find(trq_mg2_star < trq_m_min);
% if (any(ind))
%     trq_mg2(ind) = trq_m_min;
%     trq_fb(ind) = K*(trq_m_min - trq_mg2_star(ind));
```

```
% end
%
% % trq_m_min can violate P_mg2_min if speed is high
% ind = find(trq_mg2.*omega_r2 < P_mg2_min);
% if any(ind)
%     trq_mg2(ind) = P_mg2_min./omega_r2;
%     trq_fb(ind) = K*(trq_mg2(ind) - trq_mg2_star(ind));
% end
%
% % If we need strong, positive torque, trq_mg2 is saturated at trq_m_max
% % This will need to be constrained!

% Euler Forward Integration
omega_ce_next = omega_ce + omega_ce_dot*dt;
omega_s1_next = omega_s1 + omega_s1_dot*dt;
omega_r2_next = omega_r2 + omega_r2_dot*dt;

% Powers
P_e = trq_e.*omega_ce;
P_mg1 = trq_mg1.*omega_s1;
P_mg2 = trq_mg2.*omega_r2;


%% Apply Constraints to Inputs

% **Case: ON2OFF - make sure engine shuts off
% Engine Torque
ind = find(trq_e ≠ 0);
if any(ind)
    trans_cost(ind) = 1e6;
end

% % Engine Hot Max Open Throttle Torque as a function of engine speed
% % p_max_trq = evalin('base','p_max_trq');
% trq_e_max = polyval(p_max_trq,omega_ce);
% ind = find(trq_e > trq_e_max);
% if any(ind)
%     trans_cost(ind) = 1e6;
% end

% Engine Power
% For THIS engine, max power will never be violated

% M/G1 Torque
ind = find(trq_mg1 < trq_g_min | trq_mg1 > trq_g_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% M/G1 Power
ind = find(P_mg1 < P_mg1_min | P_mg1 > P_mg1_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% M/G2 Torque
ind = find(trq_mg2 < trq_m_min | trq_mg2 > trq_m_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% M/G2 Power
ind = find(P_mg2 < P_mg2_min | P_mg2 > P_mg2_max);
if any(ind)
```

```
    trans_cost(ind) = 1e6;
end

%% Apply Constraints to Present States

% Engine Speed (only needs to be constrained if engine is on)
under = find(omega_ce < omega_e_min | omega_ce > omega_e_max );
if any(under)
    trans_cost(under) = 1e6;
end

% M/G1 Speed
ind = find(omega_s1 < spd_g_min | omega_s1 > spd_g_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% % M/G2 Speed
% % Will only be violated if v > 102 mph
% ind = find(omega_r2 < spd_m_min | omega_r2 > spd_m_max);
% if any(ind)
%     trans_cost(ind) = 1e6;
% end

%% Apply Constraints to Next States

% Engine Speed at Step k+1
% ** THIS IS CASE SPECIFIC **
% ON2OFF: omega_ce_next = 0;

% M/G1 Speed at Step k+1
ind = find(omega_s1_next < spd_g_min | omega_s1_next > spd_g_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% % M/G2 Speed at Step k+1
% % Will only be violated if v > 102 mph
% ind = find(omega_r2_next < spd_m_min | omega_r2_next > spd_m_max);
% if any(ind)
%     trans_cost(ind) = 1e6;
% end

%% Battery Model

% Load Parameters
param_batt

% Calculate Mechanical Power
P_mg1 = omega_s1.*trq_mg1;
P_mg2 = omega_r2.*trq_mg2;

% Determine electrical power for M/Gs
% If P_mg is +, PE > P, PE = P*eff^-1, k = -1
% If P_mg is -, PE < P, PE = P*eff^1, k = 1     * k = - sign(P)

% Look up efficiency from map(torque,speed)
eff_mg1 = interp2(trq_g,spd_g,eff_g,trq_mg1,omega_s1,'*nearest',.1');
eff_mg2 = interp2(trq_m,spd_m,eff_m,trq_mg2,omega_r2,'*nearest',.1)';
% Extrapolating outside of the efficiency table results in NaN efficiencies
% So we replace with poor efficiencies. The Upol will be ruled infeasible anyway

PE_mg1 = P_mg1.*eff_mg1.^(-sign(P_mg1));
PE_mg2 = P_mg2.*eff_mg2.^(-sign(P_mg2));
```

```
% Total Power in (-) or out (+) of the battery
P_batt = PE_mg1 + PE_mg2;

% Apply Constraints To Battery Power
ind = find(P_batt < P_batt_ch_lim | P_batt > P_batt_dis_lim);
if any(ind)
    trans_cost(ind) = 1e6;
end

% Apply Constraints To Current SOC
ind = find(SOC < SOC_min | SOC > SOC_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% Simulate pack model to calculate SOC_next
[SOC_next] = model_DDP_batt(SOC,P_batt);

% Apply Constraints To Next SOC
ind = find(SOC_next < SOC_min | SOC_next > SOC_max);
if any(ind)
    trans_cost(ind) = 1e6;
end


%% Calculate Fuel Use

% OFF2OFF
% gallons = grams * 1/1000 *1/rho * cum2gal
gal_dot_fuel = m_dot_fuel_off *(1/1000) * (1/rho_fuel) * cum2gal;

%% Report Total Transition Cost
trans_cost = trans_cost + gal_dot_fuel;
```

## A.8 Solve Engine ON Model

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PHEV Discrete Time Dynamic Model
% Case: ON2ON
% Written by: Tim Montgomery        2012.11.29
%
% In this case, the engine is initial ON, and our control decision is to
% keep the engine ON
%
% Inputs: SOC, omega_ce, trq_e (% of Pdem met by engine), EIO, v, v_dot
% Outputs: SOC_next, omega_ce_next, trans_cost
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% function [SOC_next omegas dots omegas_next trqs powers trans_cost] = model_ON2ON(SOC,omega_ce,EIO,trq_
function [omega_ce_next SOC_next trans_cost] = model_DDP_ON2ON(omega_ce,SOC,EIO,trq_e,v,v_dot)

% Make sure we are in the right Case... we can get rid of this later
if ¬(omega_ce ≠ 0 & EIO == 1)
    disp('Evaluating Wrong Case!')
end

long = length(omega_ce);  % to be used to size paired vectors
% part = evalin('base','part');
% numparts = evalin('base','numparts');
```

```
% Load Parameters
param_veh;
param_trans;
param_MG;
param_ICE;
param_DDP;

% Time Step
dt = 1;     % 1 second time step, suitable for supervisory power mgmt

% Initialize Transition Cost
trans_cost = zeros(long,1);

%% Universal Calculations

% Calculate road loads
F_roll = mu_roll*m*g * (v≠0);      % Rolling Resistance (N)
F_drag = 0.5*rho*A_fr*C_d*v^2;      % Aerodynamic Drag (N)
F_damp = (Cw/R_tire)*v;             % Wheel/Bearing Damping (N)
F_road = F_roll + F_drag + F_damp;  % Total Road Load (N)
trq_road = F_road*R_tire;           % Torque from road loads at FINAL DRIVE

% Force/Torqe for Driveability
% m*v_dot = F_drive - F_road
F_drive = F_road + m*v_dot; % (N) Force required at tire/ground to overcome
                            % road loads and achieve acceleration
trq_drive = F_drive*R_tire; % (N*m) Torque required at FINAL DRIVE

% Power Demand for Driveability
P_dem = F_drive*v;  % (Watts) Total power required for Driveability

% Backtrack vehicle velocity and acceleration through drive train
omega_r2 = K*v/R_tire;
omega_r2_dot = K*v_dot/R_tire;

% Use PGS Kinematics for Velocity
omega_s1 = 1/S*(omega_ce*(R+S) - omega_r2*R);



% P_rev = I_mg1*omega_s1*omega_s1_dot + I_e*omega_ce*omega_ce_dot + I_mg2*omega_r2*omega_r2_dot;
% P_dem = P_dem + P_rev;
P_rev = 0;

% Solve for the PGS internal reaction force, F, using Newton's Law
    % at Sun and Carrier, PGS acceleration kinematics, and P_dem

I_prime_mg2 = (R_tire/K)^2*m + I_mg2;
I_prime_f = ((R+S)^2/S)*(I_mg1/I_e) + S;

F = ( trq_e.*omega_ce + (R+S)/S*I_mg1/I_e.*trq_e.*omega_s1 - R/S*I_mg1.*omega_s1.*omega_r2_dot + I_prime
P_dem )./(I_prime_f*omega_s1 + R*omega_r2);

% Calculate trq_mg1 using Newton's Law at Sun and Carrier, and PGS acc kin
trq_mg1 = (R+S)/S*I_mg1/I_e*trq_e - R/S*I_mg1*omega_r2_dot - I_prime_f*F;

% Calculate trq_mg2 using Newton's Law at Ring node
trq_mg2_star = I_prime_mg2*omega_r2_dot + 1/K*trq_road - F*R;

% trq_mg2_star is the ideal value of trq_mg2, but is limited in practice
% If trq_mg2 star is feasible, trq_mg2 = trq_mg2_star
trq_mg2 = trq_mg2_star;
```

```
% If we need strong, negative torque, trq_mg2 is saturated and a friction
% brake, trq_fb(applied at wheel), takes care of the rest of the braking

trq_fb = zeros(size(trq_mg2_star)); % Do we ever use this? Delete?
ind = find(trq_mg2_star < trq_m_min);
if (any(ind))
    trq_mg2(ind) = trq_m_min;
    trq_fb(ind) = K*(trq_m_min - trq_mg2_star(ind));
end

% trq_m_min can violate P_mg2_min if speed is high
ind = find(trq_mg2.*omega_r2 < P_mg2_min);
if any(ind)
    trq_mg2(ind) = P_mg2_min./omega_r2;
    trq_fb(ind) = K*(trq_mg2(ind) - trq_mg2_star(ind));
end

% If we need strong, positive torque, trq_mg2 is saturated at trq_m_max
ind = find(trq_mg2_star > trq_m_max);
if (any(ind))
    trans_cost_now(ind) = 1e6;
end

% Calculate angular accelerations using Newton's Law at each node
omega_ce_dot = 1/I_e*(trq_e - F*(R+S));
omega_s1_dot = 1/I_mg1*(trq_mg1 + F*S);


% Euler Forward Integration
omega_ce_next = omega_ce + omega_ce_dot*dt;
omega_s1_next = omega_s1 + omega_s1_dot*dt;
omega_r2_next = omega_r2 + omega_r2_dot*dt;

% Powers
P_e = trq_e.*omega_ce;
P_mg1 = trq_mg1.*omega_s1;
P_mg2 = trq_mg2.*omega_r2;

%% Apply Constraints to Inputs

% Engine Torque
ind = find(trq_e < trq_e_min | trq_e > trq_e_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% Engine Hot Max Open Throttle Torque as a function of engine speed
% p_max_trq = evalin('base','p_max_trq');
trq_e_max = polyval(p_max_trq,omega_ce);
ind = find(trq_e > trq_e_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% Engine Power
% For THIS engine, max power will never be violated

% M/G1 Torque
ind = find(trq_mg1 < trq_g_min | trq_mg1 > trq_g_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% M/G1 Power
```

```
ind = find(P_mg1 < P_mg1_min | P_mg1 > P_mg1_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% M/G2 Torque
ind = find(trq_mg2 < trq_m_min | trq_mg2 > trq_m_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% M/G2 Power
ind = find(P_mg2 < P_mg2_min | P_mg2 > P_mg2_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

%% Apply Constraints to Present States

% Engine Speed (only needs to be constrained if engine is on)
under = find(omega_ce < omega_e_min | omega_ce > omega_e_max );
if any(under)
    trans_cost(under) = 1e6;
end

% M/G1 Speed
ind = find(omega_s1 < spd_g_min | omega_s1 > spd_g_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% % M/G2 Speed
% % Will only be violated if v > 102 mph
% ind = find(omega_r2 < spd_m_min | omega_r2 > spd_m_max);
% if any(ind)
%     trans_cost(ind) = 1e6;
% end

%% Apply Constraints to Next States

% Engine Speed at Step k+1
% ** THIS IS CASE SPECIFIC **
% ON2ON:
under = find(omega_ce_next < omega_e_min | omega_ce_next > omega_e_max );
if any(under)
    trans_cost(under) = 1e6;
end

% M/G1 Speed at Step k+1
ind = find(omega_s1_next < spd_g_min | omega_s1_next > spd_g_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% % M/G2 Speed at Step k+1
% % Will only be violated if v > 102 mph
% ind = find(omega_r2_next < spd_m_min | omega_r2_next > spd_m_max);
% if any(ind)
%     trans_cost(ind) = 1e6;
% end

%% Battery Model

% Load Parameters
```

```
param_batt

% Calculate Mechanical Power
P_mg1 = omega_s1.*trq_mg1;
P_mg2 = omega_r2.*trq_mg2;

% Determine electrical power for M/Gs
% If P_mg is +, PE > P, PE = P*eff^-1, k = -1
% If P_mg is -, PE < P, PE = P*eff^1, k = 1     * k = - sign(P)

% Look up efficiency from map(torque,speed)
eff_mg1 = interp2(trq_g,spd_g,eff_g,trq_mg1,omega_s1,'*nearest',.1');
eff_mg2 = interp2(trq_m,spd_m,eff_m,trq_mg2,omega_r2,'*nearest',.1)';
% Extrapolating outside of the efficiency table results in NaN efficiencies
% So we replace with poor efficiencies. The Upol will be ruled infeasible anyway

PE_mg1 = P_mg1.*eff_mg1.^(-sign(P_mg1));
PE_mg2 = P_mg2.*eff_mg2.^(-sign(P_mg2));

% Total Power in (-) or out (+) of the battery
P_batt = PE_mg1 + PE_mg2;

% Apply Constraints To Battery Power
ind = find(P_batt < P_batt_ch_lim | P_batt > P_batt_dis_lim);
if any(ind)
    trans_cost(ind) = 1e6;
end

% Apply Constraints To Current SOC
ind = find(SOC < SOC_min | SOC > SOC_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

% Simulate pack model to calculate SOC_next
[SOC_next] = model_DDP_batt(SOC,P_batt);

% Apply Constraints To Next SOC
ind = find(SOC_next < SOC_min | SOC_next > SOC_max);
if any(ind)
    trans_cost(ind) = 1e6;
end

%% Calculate Transition Cost
% fc_fuel_coef = evalin('base','fc_fuel_coef');
% m_dot_fuel = fc_fuel_coef(1) + fc_fuel_coef(2).*omega_ce + fc_fuel_coef(3).*trq_e + fc_fuel_coef(4).*o

m_dot_fuel = interp2(fc_map_trq,fc_map_spd,fc_fuel_map,trq_e,omega_ce,'*nearest',1e6);

% Apply idling fuel cost to penalize the engine for being on
% ind = find(trq_e < .01);
% if any(ind)
%     m_dot_fuel(ind) = m_dot_fuel_idle;
% end

m_dot_fuel = max(m_dot_fuel,m_dot_fuel_idle);

% gallons = grams * 1/1000 *1/rho * cum2gal
gal_dot_fuel = m_dot_fuel * (1/1000) * (1/rho_fuel) * cum2gal;

%% Report Total Transition Cost
trans_cost = trans_cost + gal_dot_fuel;
```

## A.9 Nearest Neighbor Function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Nearest Neighbor Function
% Written by: Tim Montgomery        2012.07.03
% Updated for DDP_IO                2012.12.02
%
% Shifts X_next (calculated by the DDP model) to a point on the state
% mesh
%
% Accepts info in the form:  nearest_neighbor(x1_vec,x2_vec,x1,x2)
%
%   x1 = omega_ce
%   x2 = SOC
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [X_NN_ind] = nearest_neighbor_IO(omega_ce_vec,SOC_vec,omega_ce,SOC)

omega_ce_ind = interp1(omega_ce_vec,1:length(omega_ce_vec),omega_ce,'nearest');

under = find(omega_ce < min(omega_ce_vec));
if any(under)
    omega_ce_ind(under) = 1;
end
over = find(omega_ce > max(omega_ce_vec));
if any(over)
    omega_ce_ind(over) = length(omega_ce_vec);
end

SOC_ind = interp1(SOC_vec,1:length(SOC_vec),SOC,'*nearest');

under = find(SOC < min(SOC_vec));
if any(under)
    SOC_ind(under) = 1;
end
over = find(SOC > max(SOC_vec));
if any(over)
    SOC_ind(over) = length(SOC_vec);
end

nans = find(isnan(SOC));
if any(nans)
    SOC_ind(nans) = 1;
end

X_NN_ind = (omega_ce_ind - 1).*length(SOC_vec) + SOC_ind;

% X_NN_ind = (SOC_ind - 1).*length(omega_ce_vec) + omega_ce_ind;
```

# Appendix B

# MATLAB Code - Postprocessing Scripts

## B.1 Postprocessing Script

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Post-DDP Processing Tool for analyzing the optimal trajectory
% Written by: Tim Montgomery        2012.09.01
%
% The model is for a power-split PHEV with Engine I/O capability
%
% Starts at the beginning of a drive cycle, with inital State conditions
% input by user. Beginning at time step = 1, applies the optimal control
% policy determined in the DDP algorithm to generate the optimal State
% and Control Input Trajectories
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Initialize
clear all
close all
clc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% BEGIN USER INPUT

% Pick Drive Cycle and DDP Parameters
% CYC_name = input('Drive Cycle File Name?  ','s');
% mesh_style = input('Mesh Density?  ','s');
CYC_name = 'CYC_UDDS';
model_rev = 'IO_1621';
t_start = '1';
t_end = '0';
mesh_style = 'm50162150';
gen_mesh_file = strcat('gen_mesh_IO_',mesh_style);
param_DDP;

% Initial Battery Pack SOC (fraction)
SOC_0 = 0.3;

% Initial Engine Speed (rad/s)
% To find OPTIMAL initial engine speed
% enter omega_ce_0 = -1;
omega_ce_0 = -1;


%% END USER INPUT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

run(gen_mesh_file);
load(strcat('SOL_',model_rev,'_',CYC_name,'_',t_start,'to',t_end,'s_',mesh_style));
param_veh;
param_ICE;

% Load Drive Cycle
dt = 1;     % 1 second time step, suitable for supervisory power mgmnt
load(CYC_name);
```

```
% Determine if the entire cycle was tested, or just a portion
if ¬exist('t_end')
    t_start = 1;
    time = cyc_mph(:,1);
    v_mph = cyc_mph(:,2);
elseif t_end == 0
    time = cyc_mph(t_start:end,1);
    v_mph = cyc_mph(t_start:end,2);
else
    time = cyc_mph(t_start:t_end,1);
    v_mph = cyc_mph(t_start:t_end,2);
end

v = v_mph * mph2mps;
v_shift = zeros(size(v));
v_shift(1:end-1) = v(2:end);
v_dot = v_shift - v;
v_dot(end) = v_dot(end-1);
N = length(v);


% Find Optimal Starting Engine Speed
SOC_0_ind = interp1(SOC_vec,1:length(SOC_vec),SOC_0,'*nearest');
SOC_0 = SOC_vec(SOC_0_ind);
keepers_ind = find(Xconfig(:,2) == SOC_0);
keepers_config = Xconfig(keepers_ind,:);
keepers_cost = C2G_opt(keepers_ind,1);
[min_cost min_ind] = min(keepers_cost);
opt_IC_ind = keepers_ind(min_ind);

% Initialize State and Input Trajectories
Xconfig_traj = zeros(size(time));
Upol_traj = zeros(size(time)-1);

% IF starting engine speed isn't specified, use the optimal IC
if omega_ce_0 == -1
    Xconfig_traj(1) = opt_IC_ind;
else
    Xconfig_traj(1) = nearest_neighbor_IO(omega_ce_vec,SOC_vec,omega_ce_0,SOC_0);
end


% Check For Feasibilitiy
if min_cost > 1000
    disp('NO FEASIBLE PATH')
else


%% Start at the beginning and step forward in time

for step = 1:N-1

    % Current State Configuration Index
    X_ind = Xconfig_traj(step);

    % Optimal Input to Apply (based on State AND Step)
    Upol_app = Upol_opt(X_ind,step);

    % Record Input Policy Trajectory
    Upol_traj(step,1) = Upol_app;

    % Drive Cycle Info
    V = v(step);
    V_dot = v_dot(step);
```

```
    % Physical States
    omega_ce = Xconfig(X_ind,1);
    SOC = Xconfig(X_ind,2);
    omega_wheel = V/R_tire;
    omega_r2 = omega_wheel*K;

    % Physical Inputs
    EIO = Upol(Upol_app,1);
    trq_e = Upol(Upol_app,2);


    %% Generate the Next State

    % Determine which EIO case to evaluate
    if omega_ce == 0
        if EIO == 0
            % Case: OFF2OFF
            [omegas omega_dots omegas_next SOC SOC_dot SOC_next EIO trqs powers trans_cost] = model_PP_O
        else
            % Case: OFF2ON
            [omegas omega_dots omegas_next SOC SOC_dot SOC_next EIO trqs powers trans_cost] = model_PP_O
        end
    else
        if EIO == 0
            % Case: ON2OFF
            [omegas omega_dots omegas_next SOC SOC_dot SOC_next EIO trqs powers trans_cost] = model_PP_O
        else
            % Case: ON2ON
            [omegas omega_dots omegas_next SOC SOC_dot SOC_next EIO trqs powers trans_cost] = model_PP_O
        end
    end

    % Store information in time trajectories
    omegas_traj(step,:) = omegas;
    omega_dots_traj(step,:) = omega_dots;
    omegas_next_traj(step,:) = omegas_next;

    SOC_traj(step,:) = SOC;
    SOC_dot_traj(step,:) = SOC_dot;
    SOC_next_traj(step,:) = SOC_next;

    EIO_traj(step,:) = EIO;
    trqs_traj(step,:) = trqs;
    powers_traj(step,:) = powers;

    trans_cost_traj(step,:) = trans_cost;

    %% Nearest Neighbor - shift X_next to a point on the state mesh
    X_next_NN_ind = nearest_neighbor_IO(omega_ce_vec,SOC_vec,omegas_next(1),SOC_next);

    % Move forward to the next State at the next Step
    Xconfig_traj(step+1) = X_next_NN_ind;


end

omega_ce = omegas_traj(:,1);
omega_s1 = omegas_traj(:,2);
omega_r2 = omegas_traj(:,3);

omega_ce_dot = omega_dots_traj(:,1);
omega_s1_dot = omega_dots_traj(:,2);
omega_r2_dot = omega_dots_traj(:,3);
```

```
omega_ce_next = omegas_next_traj(:,1);
omega_s1_next = omegas_next_traj(:,2);
omega_r2_next = omegas_next_traj(:,3);

SOC = SOC_traj;
SOC_dot = SOC_dot_traj;
SOC_next = SOC_next_traj;

EIO = EIO_traj;

trq_e = trqs_traj(:,1);
trq_mg1 = trqs_traj(:,2);
trq_mg2 = trqs_traj(:,3);

P_e = powers_traj(:,1);
P_mg1 = powers_traj(:,2);
P_mg2 = powers_traj(:,3);
PE_mg1 = powers_traj(:,4);
PE_mg2 = powers_traj(:,5);
P_batt = powers_traj(:,6);
P_dem = powers_traj(:,7);
P_add = P_e + P_mg1 + P_mg2;

gal_dot_fuel = trans_cost_traj;
fuel_gal = cumsum(gal_dot_fuel);

if fuel_gal(end) > 1000
    disp('THIS path is not feasible!')
end

% %*********************************************************
% param_trans_TRM
% P_loss_e = I_e.*omega_ce.*omega_ce_dot;
% P_loss_mg1 = I_mg1.*omega_s1.*omega_s1_dot;
% P_loss_mg2 = I_mg2.*omega_r2.*omega_r2_dot;
%
% P_add_star = P_add - P_loss_e - P_loss_mg1 - P_loss_mg2;
% %*********************************************************

% Are the SOC Dynamics Being Fully Captured?
SOC_sensitivity = 0.1 *(SOC_vec(end)-SOC_vec(end-1));
SOC_dot_seen = SOC(2:end) - SOC(1:end-1);
free = sum(SOC_dot_seen == 0 & SOC_dot(1:end-1) < -SOC_sensitivity);
wasted = sum(SOC_dot_seen == 0 & SOC_dot(1:end-1) > SOC_sensitivity);

if free - 2*wasted > 0
    disp('There is a LOT of free electrochemical energy!')
elseif free - wasted > 0
    disp('There is some free electrochemical energy')
end


t = time(1:end-1);
t_next = time(2:end-1);

% Total Distance Driven
distance_meters = sum(v)*dt;
distance_miles = distance_meters*m2mi;

% Velocity Statistics
v_max_mph = max(v)/mph2mps;
v_avg_mph = mean(v)/mph2mps;
```

```
% Fuel Efficiency
FuelEconomy = distance_miles/fuel_gal(end);

%% Plot Results

figure

subplot(321)
plotyy(t,v(1:end-1),t,P_dem)
% legend('Velocity','Power Demand')
title(strcat('Drive Cycle: ',CYC_name))
xlabel('Time (s)')
ylabel('Vehicle Speed (m/s)')

subplot(323)
plot(t,omega_ce,'r',t,omega_s1,'g',t,omega_r2,'b')
hold on
plot(t_next,omega_ce_next(1:end-1),'r--',t_next,omega_s1_next(1:end-1),'g--',t_next,omega_r2_next(1:end-
legend('omega_c_e','omega_s_1','omega_r_2')
title('State Trajectories')
xlabel('Time (s)')
ylabel('Rotational Speed (rad/s)')
line([time(1) time(end)],[omega_e_min omega_e_min],'color','r','LineStyle','-.')
line([time(1) time(end)],[omega_e_max omega_e_max],'color','r','LineStyle','-.')

subplot(325)
plot(t,trq_e,'r',t,trq_mg1,'g',t,trq_mg2,'b')
legend('trq_e','trq_m_g_1','trq_m_g_2')
title('Control Input Trajectory')
xlabel('Time (s)')
ylabel('Torque (N*m)')

subplot(322)
plot(t,P_e,'r',t,P_mg1,'g',t,PE_mg1,'g--',t,P_mg2,'b',t,PE_mg2,'b--',t,P_batt,'c--')
legend('P_e','P_m_g_1','PE_m_g_1','P_m_g_2','PE_m_g_2','PE_b_a_t_t')
title('Component Power')
xlabel('Time(s)')
ylabel('Power (Watts)')

subplot(324)
plot(t,SOC,'c')
hold on
plot(t_next,SOC_next(1:end-1),'c--')
title('SOC Trajectory')
xlabel('Time (s)')
ylabel('SOC (frac)')

subplot(326)
plot(t,fuel_gal)
title('Fuel Consumption')
xlabel('Time (s)')
ylabel('Fuel Consumed (gal)')

% figure
% plot(t,P_dem,'r',t,P_add,'b--',t,P_add_star,'g:')
% legend('P_d_e_m','P_a_d_d','P_s_t_a_r')
% title('Power Demand')
% xlabel('Time (s)')
% ylabel('Power (Watts)')

disp(' ')
TotalFuelUsed = fuel_gal(end);
SOC_initial = SOC(1);
SOC_final = SOC(end);
```

```
Δ_SOC = SOC_final - SOC_initial;

a = 'Total Distance Travelled: ';
disp([a num2str(distance_miles) ' miles'])
a = 'Total Fuel Used: ';
disp([a num2str(TotalFuelUsed) ' gal'])
a = 'Fuel Efficiency: ';
disp([a num2str(FuelEconomy) ' mpg'])
a = ' ';
disp(a)
a = 'Initial SOC: ';
disp([a num2str(SOC_initial)])
a = 'Final SOC: ';
disp([a num2str(SOC_final)])
a = 'SOC Depletion: ';
disp([a num2str(Δ_SOC)])


%% Plots Used In Thesis
figure

subplot(411)
plot(t,v(1:end-1))
ylabel('Vehicle Speed (m/s)')

subplot(412)
plot(t,P_batt/1000)
ylabel('Battery Power (kW)')

subplot(413)
plot(t,SOC)
ylabel('SOC (frac)')

subplot(414)
plot(t,fuel_gal)
xlabel('Time (s)')
ylabel('Fuel Consumed (gal)')

%% Plots used in Masters Defense
figure

subplot(311)
plot(t,v(1:end-1))
title(strcat('UDDS Drive Cycle'))
xlabel('Time (s)')
ylabel('Vehicle Speed (m/s)')

subplot(312)
plotyy(t,SOC,t,P_batt/1000)
% hold on
% plot(t_next,SOC_next(1:end-1),'c--')
% title('SOC Trajectory')
xlabel('Time (s)')
ylabel('SOC (frac)')

subplot(313)
plot(t,fuel_gal)
% title('Fuel Consumption')
xlabel('Time (s)')
ylabel('Fuel Consumed (gal)')

end
```

## B.2 PHEV Predicted Performance Comparison

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Script to compare the accuracy of the DDP algorithm
% for different SOC mesh densities
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
close all
clc

meshes = [221 421 621 821 1021 1221 1421 1621];
SOC_f = [0.7025 0.6050 0.5733 0.5656 0.5615 0.55708 0.54857 0.54875];
SOC_dep = 0.8 - SOC_f;
fuel_econ = [352.115 194.815 134.297 113.64 103.02 96.15 89.29 86.67];
gamma = [0.6192 1.1943 1.7887 2.3805 2.9581 3.5435 4.1233 4.6585];

figure

subplot(311)
plot(meshes,SOC_dep)
ylabel('SOC depletion')

subplot(312)
plot(meshes,fuel_econ)
ylabel('Fuel Economy')

subplot(313)
plot(meshes,gamma)
ylabel('Single Step Optimization Time')
xlabel('Number of Points on SOC Mesh')
```

# Appendix C

# MATLAB Code - Submodel Parameter Scripts

## C.1 PHEV Parameters

```
%% Vehicle Dynamics Parameters
%   Adapted: July 2012 by Tim Montgomery
%   Created: January 4, 2008 by Scott Moura

%% Conversion Factors

mph2mps = 0.44704;      % MPH to m/s
m2mi = 0.00062137;      % meters to miles
lbs2kg = 1/2.205;       % lbs to kg

%% Vehicle: Modified Toyota Prius PHEV

R_tire = 0.287;             % Wheel radius (m)
m_tire = 181*lbs2kg;        % Wheel mass (converted to kg)
Iw = m_tire*R_tire^2/2;     % Wheel Inertia (kg*m^2)
Cw = 0.3;                   % Estimated Wheel damping (N*s)
C_roll = 0.00475;           % Rolling resistance coefficient

m_curb = 1420;          % Curb weight (kg) of vehicle, including 80kg Li-ion battery pack
m_passengers = 4*160*lbs2kg;    % Mass (kg) of four 160 lb passengers
m = m_curb + m_passengers;  % Total mass of vehicle including passengers
A_fr = 2.23;            % Frontal area (m^2)
C_d = 0.26;             % Drag coefficient
mu_roll = 0.009;        % Constant rolling coefficient

K = 3.93;               % Final drive ratio

rho = 1.2;              % Density of air (kg/m^3)
g = 9.81;               % Acceleration due to gravity (m/s^2)

% WwMin = 5*mph2mps/Rw;   % Minimum Wheel Speed (rad/s) -> 5 mph
% R_tire = 0.3107;        % Rolling radius of driven wheels (m)
```

## C.2 Battery Pack Parameters

```
%% Battery Cell/Pack Parameters
% Created: May 2012 by Tim Montgomery
%
% Based from ADVISOR data file:  ESS_LI7_temp.m
% 6 Ah Saft Lithium Ion battery
%


%% Individual Cell Parameters
V_cell_nom = 3.3; % Nominal voltage per cell (V)
V_cell_min = 2;
V_cell_max = 3.9;
Q_cell_Ah = 7.035;      % Capacity in Ah
Q_cell = Q_cell_Ah*3600; % A*s or Coulombs
```

```
%***
% Q_cell = Q_cell/5;
%****

%% Module Parameters (Module = 3 cells in series???????)
Nmod = 3;
V_mod_nom = V_cell_nom*Nmod;
V_mod_min = V_cell_min*Nmod;
V_mod_max = V_cell_max*Nmod;

%% Battery Pack Configuration
Ns = 64;   % Number of MODULES in series, to achieve Vpack¬633.6V
Np = 1;    % Number of strings in parallel, to achieve EnPack¬4.4kWh
               % based on Q_cell = 7.035 Ah

% %*****
% Ns = 20;



%% Pack Parameters
V_pack_nom = V_mod_nom*Ns;  % 633.6 V
V_pack_min = V_mod_min*Ns;  % 384.0 V
V_pack_max = V_mod_max*Ns;  % 748.8 V
Q_pack = Q_cell*Np;   % Capacity in A*s or Coulombs
Q_pack_kWh = Q_pack/3600*V_pack_nom;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SOC RANGE over which data is defined
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
SOC_map =[0 10 20 40 60 80 100]/100;  % (--)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Temperature range over which data is defined
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T_batt_map =[0 25 41]+273;  % (C)+273 = K

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LOSS AND EFFICIENCY parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Parameters vary by SOC horizontally, and temperature vertically
batt_max_ah_cap = [5.943 7.035 7.405];
    % (A*h), max. capacity at C/3 rate, indexed by T_batt_range

% average coulombic (a.k.a. amp-hour) efficiency below, indexed by T_batt_range
batt_coulombic_eff = [0.968 0.99 0.992];  % (--)

% module's resistance to being discharged, indexed by SOC_range and T_batt_range
% R_batt_dis_range =[0.0419 0.0288  0.0221  0.014   0.0145  0.0145   0.0162;
%                   0.072  0.01515 0.00839 0.00493 0.00505 0.005524 0.005722;
%                   0.0535 0.0133  0.0082  0.0059  0.0059  0.006    0.0063]; % (ohm)

%
R_batt_dis_map =[0.072  0.01515 0.00839 0.00493 0.00505 0.005524 0.005722]*Nmod; % (ohm)

% module's resistance to being charged, indexed by SOC_range and T_batt_range
% R_batt_ch_range = [0.021  0.018  0.0177   0.0157  0.0138  0.0138  0.015;
%                   0.0124 0.0068 0.005426 0.00442 0.00463 0.00583 0.00583;
%                   0.0104 0.0079 0.0072   0.0064  0.0059  0.0058  0.006]; % (ohm)

R_batt_ch_map = [0.0124 0.0068 0.005426 0.00442 0.00463 0.00583 0.00583]*Nmod; % (ohm)

% module's open-circuit (a.k.a. no-load) voltage, indexed by SOC_range and T_batt_range
```

```
% Voc_range = [3.44  3.473 3.496 3.568 3.637 3.757 3.896;
%              3.124 3.349 3.433 3.518 3.616 3.752 3.898;
%              3.128 3.36  3.44  3.528 3.623 3.761 3.899]; % (V)

Voc_map = [3.124 3.349 3.433 3.518 3.616 3.752 3.898]*Nmod;    % (V)


%% Battery Thermal Parameters
batt_th_calc = 1;                                % --     0=no ess thermal calculations, 1=do calc's
batt_mod_cp = 795;                               % J/kgK  ave heat capacity of module
batt_set_tmp = 35;                               % C      thermostat temp of module when cooling fan comes
batt_mod_sarea = .032;                            % m^2    total module surface area exposed to cooling ai
batt_mod_airflow = .07/12;                      % kg/s   cooling air mass flow rate across module (140 cfm=
batt_mod_flow_area = .0011;                      % m^2    cross-sec flow area for cooling air per module
batt_mod_case_thk = .001;                      % m      thickness of module case
batt_mod_case_th_cond = 15;                    % W/mK   thermal conductivity of module case material
batt_air_vel = batt_mod_airflow/(1.16*batt_mod_flow_area); % m/s  ave velocity of cooling air
batt_air_htcoef = 30*(batt_air_vel/5)^0.8;      % W/m^2K cooling air heat transfer coef.
batt_th_res_on = ((1/batt_air_htcoef)+(batt_mod_case_thk/batt_mod_case_th_cond))/batt_mod_sarea; % K/W
tot thermal res key on
batt_th_res_off = ((1/4)+(batt_mod_case_thk/batt_mod_case_th_cond))/batt_mod_sarea; % K/W
tot thermal res key off (cold soak)
% set bounds on flow rate and thermal resistance
batt_mod_airflow = max(batt_mod_airflow,0.001);
batt_th_res_on = min(batt_th_res_on,batt_th_res_off);
clear ess2_mod_sarea ess2_mod_flow_area ess2_mod_case_thk ess2_mod_case_th_cond ess2_air_vel ess2_air_ht

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% OTHER DATA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                % Not sure if this *3 is original or added by Scott, but...
batt_module_mass = .37824*Nmod;  % (kg), mass of single module
mass_pack = batt_module_mass*Ns*Np;
        % ... it seems to belong...
        % mass_pack = 72.6221 kg, which is close to the
        % reported 80kg for the 4.4kW pack!

% batt_cap_scale=1; % scale factor for module max ah capacity
    % Don't know what this means, but it seems to be the same as Np??

%% From Kelsey's Research

% Open Circuit Voltage as a Function of SOC for A123 18650 cell

% Q_batt = 1.1;          % [A-h]
% R_cell = 0.0522;       % [ohms]
%
% SOC = 1:1:101;
%
% V = [2.000; 2.760; 2.885; 2.963; 3.020; 3.066; 3.103; 3.136; 3.163; 3.188;
%      3.209; 3.225; 3.233; 3.237; 3.241; 3.245; 3.250; 3.256; 3.262; 3.268;
%      3.273; 3.279; 3.284; 3.288; 3.292; 3.295; 3.299; 3.303; 3.306; 3.310;
%      3.313; 3.315; 3.318; 3.320; 3.321; 3.323; 3.323; 3.324; 3.324; 3.324;
%      3.325; 3.325; 3.325; 3.326; 3.326; 3.327; 3.327; 3.328; 3.329; 3.329;
%      3.330; 3.330; 3.331; 3.332; 3.333; 3.334; 3.335; 3.336; 3.337; 3.338;
%      3.339; 3.340; 3.341; 3.343; 3.344; 3.345; 3.347; 3.348; 3.350; 3.351;
%      3.353; 3.355; 3.356; 3.358; 3.359; 3.361; 3.362; 3.363; 3.364; 3.364;
%      3.365; 3.365; 3.366; 3.366; 3.367; 3.367; 3.368; 3.369; 3.370; 3.371;
%      3.373; 3.375; 3.378; 3.381; 3.386; 3.394; 3.405; 3.423; 3.453; 3.505;
%      3.600];
```

## C.3 Combustion Engine Parameters

```
%% Internal Combustion Engine Parameters
%   Adapted: June 2012 by Tim Montgomery
%   Created: January 4, 2008 by Scott Moura

%   1.497L 57kW MY04 US Prius gasoline engine

%% Fuel Parameters
LHV = 44.4;                                 % LHV of Gasoline
rho_fuel = 737.22;                          % Density of gas (kg/m^3)
gal2cum = 3.785412e-3;                      % Gallons per cu. meter
cum2gal = 264.172;                          % Gallons per cu. meter

% gallons = grams * 1/1000 *1/rho * cum2gal

% Speed range of the engine
fuel_map_spd_rpm = [1000 1250 1500 1750 2000 2250 2500 2750 3000 3250 3500 4000];
fuel_map_spd = fuel_map_spd_rpm*2*pi/60;

lbft2Nm = 1.356; %conversion from lbft to Nm

% Torque range of the engine
fuel_map_trq_lbft = [3.15 6.3 12.5 18.8 25.1 31.3 37.6 43.9 50.1 56.4 62.7 68.9 75.2];
% Trq (lb-ft)
fuel_map_trq = fuel_map_trq_lbft*lbft2Nm;   % Trq (N*m)

% Fuel use map indexed vertically by fc_map_spd and horizontally by fc_map_trq
% (g/s)
% fuel use from Feng An's model calibrated with actual data for Prius_jpn (Atkinson cycle) engine
fuel_map = [
 0.0756    0.1513    0.1984    0.2455    0.2925    0.3396    0.3867    0.4338    0.4808
0.5279    0.5279    0.5279    0.5279
 0.0917    0.1834    0.2423    0.3011    0.3599    0.4188    0.4776    0.5365    0.5953
0.6541    0.6689    0.6689    0.6689
 0.1073    0.2145    0.2851    0.3557    0.4263    0.4969    0.5675    0.6381    0.7087
0.7793    0.8146    0.8146    0.8146
 0.1226    0.2451    0.3274    0.4098    0.4922    0.5746    0.6570    0.7393    0.8217
0.9041    0.9659    0.9659    0.9659
 0.1379    0.2759    0.3700    0.4642    0.5583    0.6525    0.7466    0.8408    0.9349
1.0291    1.1232    1.1232    1.1232
 0.1538    0.3076    0.4135    0.5194    0.6253    0.7312    0.8371    0.9430    1.0490
1.1549    1.2608    1.2873    1.2873
 0.1704    0.3407    0.4584    0.5761    0.6937    0.8114    0.9291    1.0468    1.1645
1.2822    1.3998    1.4587    1.4587
 0.1887    0.3773    0.5068    0.6362    0.7657    0.8951    1.0246    1.1540    1.2835
1.4129    1.5424    1.6395    1.6395
 0.2100    0.4200    0.5612    0.7024    0.8436    0.9849    1.1261    1.2673    1.4085
1.5497    1.6910    1.8322    1.8322
 0.2351    0.4701    0.6231    0.7761    0.9290    1.0820    1.2350    1.3880    1.5410
1.6940    1.8470    1.9999    2.0382
 0.2645    0.5290    0.6938    0.8585    1.0233    1.1880    1.3528    1.5175    1.6823
1.8470    2.0118    2.1766    2.2589
 0.3394    0.6789    0.8672    1.0555    1.2438    1.4321    1.6204    1.8087    1.9970
2.1852    2.3735    2.5618    2.7501];

% Multilinear Regression Coefficients (for this (Scott's) fuel map)
fc_fuel_coef = [1.5463e-002, 8.5156e-004, -1.7334e-003, 5.8427e-005];
% Such that: m_dot_fuel = fc_fuel_coef(1) + fc_fuel_coef(2)*omega_e + fc_fuel_coef(3)*T_e + fc_fuel_coef


%% hot max wide open throttle curves

max_hot_map_spd_rpm = [1000:125:4500]; % Engine speed (rpm)
max_hot_map_spd = max_hot_map_spd_rpm * 2*pi/60;
```

```
% (Torque in N*m)
max_hot_map_trq = [ ...
    77 82 85 88 91 93 95 97 100 101 102 102 103 104 105 105 106 107 107 108 108 ...
    109 110 110 110 111 111 112 112]*57/52; % hot wide open throttle torque, 57/52 is to correct the pow

% Polynomial fit:  trq_max(w) = p(1)*w^3 + p(2)*w^2 + p(3)*w + p(4)
%                  T_e_max = polyval(p_max_trq, omega_e)
p_max_trq = [1.4072e-6, -0.0015, 0.5924, 38.6986];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Consumption tables from PSAT (both BSFC and fuel rate, denser mesh than Scott's)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Speed range (converted to rad/s)
fc_map_spd = (1000:125:4500) *2*pi/60;

% Torque range (N*m)
fc_map_trq = 0:5:110;

% Rows represent speed (rad/s).  Columns represent torque (N-m).  Table is
% fuel rate (g/kW-hr)
 fc_bsfc_map = [...
%0          10        20        30        40        50        60          70        80
90          100        110
 800  700  600  500   400  350  325  310  295  280  270  260  255  250   247  247  247
247   247    247   247  248   250;...%1000
 800  700  580  500   390  345  325  300  290  275  267  260  253  248   243  242  242
242   242    243   245  246   247;...
 800  700  580  500   390  345  325  300  287  275  267  260  253  247   242  240  240
240   240    240   242  243   246;...%1250
 800  700  570  500   385  342  320  300  287  274  266  258  252  246   242  239  238
238   238    238   240  242   244;...
 800  680  580  490   385  345  325  300  287  273  264  257  250  245   240  238  238
237   237    238   238  240   242;...%1500
 800  680  580  480   390  345  325  300  287  270  262  256  248  244   239  237  236
236   236    236   237  238   240;...
 800  690  590  490   395  348  325  300  285  270  263  255  248  243   238  237  235
234   233    233   235  236   237;...%1750
 800  690  590  500   395  350  325  300  284  270  263  255  248  243   238  235  234
233   232    232   233  234   236;...
 800  690  590  500   395  350  325  300  283  270  263  255  248  243   238  235  233
232   230    230   231  232   235;...%2000
 800  700  600  500   395  350  325  300  282  270  262  255  248  243   238  234  233
231   230    229   229  230   232;...
 800  700  600  500   395  345  325  300  280  270  262  254  247  242   238  234  232
230   229    228   228  229   230;...%2250
 800  700  600  500   395  345  325  300  280  269  262  253  247  242   238  234  232
229   228    228   227  228   229;...
 800  700  600  500   390  345  325  300  281  270  263  252  248  242   238  234  232
229   228    227   227  227   228;...%2500
 800  700  600  500   395  350  325  300  282  270  263  253  248  243   238  235  232
230   228    227   226  226   227;...
 800  700  600  500   395  350  325  305  282  271  263  255  247  243   238  235  232
230   228    227   225  225   226;...%2750
 800  700  600  500   400  350  330  308  283  272  263  256  248  243   238  235  233
230   228    227   225  224   225;...
 800  700  600  500   400  350  330  308  286  273  265  258  250  244   239  236  233
231   229    227   226  225   224;...%3000
 800  700  600  500   400  355  330  310  288  274  266  258  250  244   239  236  234
232   230    228   227  225   224;...
 800  700  600  500   400  355  330  312  290  275  266  258  251  245   240  237  234
232   230    229   227  226   225;...%3250
 800  700  605  500   400  355  332  314  291  276  267  259  252  246   241  238  235
232   230    229   227  227   225;...
```

```
 800  710  605  500   400  360  333  315  292  277  267  259  253  247   242  238  235
233   232    230   228  227   226;...%3500
 800  710  610  505   400  360  334  315  293  278  268  259  253  247   243  239  235
234   232    230   229  228   226;...
 800  700  610  505   400  360  335  315  294  279  269  260  254  248   244  239  236
234   233    231   230  229   227;...%3750
 800  700  610  505   405  360  336  315  295  280  270  260  254  248   244  240  237
235   233    231   231  229   227;...
 800  700  610  500   410  360  337  317  295  280  270  261  255  249   245  241  237
236   233    232   232  230   227;...%4000
 800  700  615  505   415  365  338  317  296  281  271  262  256  250   245  242  238
236   234    233   232  231   228;...
 800  700  615  510   420  365  339  318  296  282  272  263  257  250   245  242  238
237   234    232   232  231   228;...%4250
 800  700  615  515   425  370  340  320  298  283  273  264  257  251   245  242  239
237   235    233   232  232   229;...
 800  700  620  520   430  370  340  325  300  285  275  265  258  251   246  242  240
237   235    233   233  232   230];%4500

% Fuel consumption table (g/s)
% Rows represent speed (rad/s).  Columns represent torque (N-m)
fc_fuel_map = (fc_map_spd' * fc_map_trq) .* fc_bsfc_map / 3600 / 1000;


%% Fuel Penalties for DDP - included by Tim Montgomery

% There is always a small amount of fuel consumed while the engine is on,
% even idling. It is generally very small compared to when the engine is
% outputting power, but in the context of DDP, it must be penalized so that
% there is incentive to turn the engine off!
% ** This is not a citable or experimentally recorded number. It is
% marginally smaller than the lowest non-zero value in the fuel map **
% min(fc_fuel_map) = 0.1018 g/s
m_dot_fuel_idle = 0.1; % (g/s)

% Turning the engine on will always have a fuel cost, often estimated as
% the amount of fuel required to run the engine through 3 full cycles
% without power output. This cost is substantial, and must be penalized in
% the context of DDP to minimize engine I/O control chatter
% ** This is not a citable or experimentally recorded number. It is
% the largest value in the fuel map **
m_dot_fuel_start = max(max(fc_fuel_map)); % 3.3118 g/s

% This paramater is very interesting! Including a fuel penalty for shutting
% the engine down helps reduce EIO control chatter, but the control
% strategy for idling becomes USER defined, not necessarily optimal!
% eg. If fuel_off = 10*fuel_idle, the engine will stay ON opposed to idling
% for 10 seconds or less, but shut OF if idling for longer than 10s
m_dot_fuel_off = 2 * m_dot_fuel_idle;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% To remove costs applied to IO
% m_dot_fuel_idle = 0;
% m_dot_fuel_start = 0;
% m_dot_fuel_off = 0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

## C.4 Motor/Generator Parameters

```
%% Motor/Generator Parameters
%   Adapted: June 2012 by Tim Montgomery
```

```
%   Created: January 4, 2008 by Scott Moura

% ** Remember, efficiency always reduces OUTPUT **
%     Motor:  P_mech = eff*P_elec
% Generator:  P_mech = 1/eff * P_elec


% Motor Efficiency Map from ADVISOR
% *** Motor data are used as parameters for M/G1 in both modes of operation ***
trq_m = [-305 -275 -245 -215 -185 -155 -125 -95 -65 -35 -5 0 5 35 65 95 ...
    125 155 185 215 245 275 305];
spd_m = [0 500 1000 1500 2000 2500 3000 3500 4000 4500 6000]*(2*pi)/60;

% Multiply by 0.95 for power electronics efficiency
eff_m = 0.95*[...
 .905    .905    .905    .905    .905    .905    .905    .905    .905    .905    .905
.905     .905    .905    .905    .905    .905    .905    .905    .905    .905    .905
.905
0.56    0.59    0.62    0.65    0.68    0.72    0.76    0.8     0.85    0.9     0.87    .905    0.87
0.85    0.8     0.76    0.72    0.68    0.65    0.62    0.59    0.56
0.72    0.74    0.76    0.78    0.81    0.83    0.86    0.89    0.91    0.94    0.85    .905    0.85
0.72    0.74    0.76    0.78    0.86    0.88    0.9     0.92    0.93    0.94    0.83    .905    0.83
0.88    0.86    0.78    0.76    0.74    0.72
0.72    0.74    0.76    0.78    0.86    0.88    0.92    0.93    0.95    0.95    0.82    .905    0.82
0.72    0.74    0.76    0.78    0.86    0.88    0.92    0.94    0.95    0.95    0.81    .905    0.81
0.72    0.74    0.76    0.78    0.86    0.88    0.92    0.95    0.96    0.95    0.81    .905    0.81
0.72    0.74    0.76    0.78    0.86    0.88    0.92    0.95    0.96    0.95    0.8
.905    0.8     0.95    0.96    0.95    0.92    0.88    0.86    0.78    0.76    0.74    0.72
0.72    0.74    0.76    0.78    0.86    0.88    0.92    0.95    0.95    0.95    0.8     .905    0.8
0.95    0.95    0.95    0.92    0.88    0.86    0.78    0.76    0.74    0.72
0.72    0.74    0.76    0.78    0.86    0.88    0.92    0.95    0.95    0.95    0.79    .905    0.79
0.72    0.74    0.76    0.78    0.86    0.88    0.92    0.95    0.95    0.95    0.79    .905    0.79

trq_m_min = min(trq_m);
trq_m_max = max(trq_m);
spd_m_min = min(spd_m);
spd_m_max = max(spd_m);

% Generator Efficiency Map from ADVISOR
% *** Generator data are used as parameters for M/G1 in both modes of op. ***
trq_g = [-55 -45 -35 -25 -15 -5 0 5 15 25 35 45 55];
spd_g = [-5500 -4000 -3500 -3000 -2500 -2000 -1500 -1000 -500 0 500 ...
    1000 1500 2000 2500 3000 3500 4000 5500 6000]*(2*pi)/60;

% Multiply by 0.95 for power electronics efficiency
eff_g = 0.95*[...
0.88    0.89    0.9     0.91    0.9     0.79    0.79    0.79    0.9     0.91    0.9
0.89    0.88
0.88    0.89    0.9     0.91    0.9     0.79    0.79    0.79    0.9     0.91    0.9
0.89    0.88
0.87    0.88    0.9     0.9     0.9     0.8     0.8     0.8     0.9     0.9     0.9
0.88    0.87
0.85    0.87    0.89    0.9     0.9     0.81    0.81    0.81    0.9     0.9     0.89    0.87    0.85
0.83    0.85    0.87    0.89    0.89    0.82    0.82    0.82    0.89    0.89    0.87    0.85    0.83
0.8     0.83    0.85    0.87    0.89    0.82    0.82    0.82    0.89    0.87    0.85    0.83    0.8
0.76    0.79    0.82    0.85    0.87    0.82    0.82    0.82    0.87    0.85    0.82    0.79    0.76
0.68    0.72    0.76    0.8     0.84    0.81    0.8     0.81    0.84    0.8     0.76    0.72    0.68
0.52    0.57    0.63    0.69    0.77    0.8     0.8     0.8     0.77    0.69    0.63    0.57    0.52
0.52    0.57    0.63    0.69    0.77    0.8     0.8     0.8     0.77    0.69    0.63    0.57    0.52
0.52    0.57    0.63    0.69    0.77    0.8     0.8     0.8     0.77    0.69    0.63    0.57    0.52
0.68    0.72    0.76    0.8     0.84    0.81    0.8     0.81    0.84    0.8     0.76    0.72    0.68
0.76    0.79    0.82    0.85    0.87    0.82    0.82    0.82    0.87    0.85    0.82    0.79    0.76
0.8     0.83    0.85    0.87    0.89    0.82    0.82    0.82    0.89    0.87    0.85    0.83    0.8
0.83    0.85    0.87    0.89    0.89    0.82    0.82    0.82    0.89    0.89    0.87    0.85    0.83
```

```
0.85    0.87    0.89    0.9     0.9     0.81    0.81    0.81    0.9     0.9     0.89    0.87    0.85
0.87    0.88    0.9     0.9     0.9     0.8     0.8     0.8     0.9     0.9     0.9
0.88    0.87
0.88    0.89    0.9     0.91    0.9     0.79    0.79    0.79    0.9     0.91    0.9
0.89    0.88
0.88    0.89    0.9     0.91    0.9     0.79    0.79    0.79    0.9     0.91    0.9
0.89    0.88
0.88    0.89    0.9     0.91    0.9     0.79    0.79    0.79    0.9     0.91    0.9
0.89    0.88];

trq_g_min = min(trq_g);
trq_g_max = max(trq_g);
spd_g_min = min(spd_g);
spd_g_max = max(spd_g);

% M/G Limits
P_mg1_min = -25e3;
P_mg1_max = 25e3;

P_mg2_min = -40e3;
P_mg2_max = 40e3;
```

## C.5 Transmission Parameters

```
%% Transmission Parameters
%   Adapted: May 2012 by Tim Montgomery
%   Created: January 4, 2008 by Scott Moura

% Transmission
R = 78;             % Number of teeth in ring gear
S = 30;             % Number of teeth in sun gear
K = 3.93;           % Final drive ratio

% Inertias
I_e = 0.18;         % Inertia of engine (kg*m^2)
I_mg1 = 0.0226;     % Inertia of M/G1 (kg*m^2)
I_mg2 = 0.0226;     % Inertia of M/G2 (kg*m^2)
I_s = 0;            % Inertia of sun gear (kg*m^2) IGNORE
I_c = 0;            % Inertia of carrier gear (kg*m^2) IGNORE
I_r = 0;            % Inertia of ring gear (kg*m^2) IGNORE

I_eq_veh = 7.573;   % Equivalent inertia of the vehicle seen at M/G2
```

# Bibliography


[1] R. E. Bellman "Dynamic Programming," Princeton University Press, NJ, 1957.

[2] R. E. Bellman and S. E. Dreyfus, "Applied Dynamic Programming," USAF Project Rand, Santa Monica, CA, May 1962.

[3] S. Sasaki, "Toyota's Newly Developed Hybrid Powertrain," *Proceedings of 1988 International Symposium on Power Semiconductor Devices & ICs*, pp. 17–22, Kyoto, Japan, 1998.

[4] J. Liu *et al.*, "Modeling and Analysis of the Toyota Hybrid System," *Proceedings of the 2005 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pp. 134–139, Monterey, CA, July 2005.

[5] M. Zolot *et al.*, "Thermal Evaluation of Toyota Prius Battery Pack," *FutureCar Conference*, 2002.

[6] F. U. Syed *et al.*, "Derivation and Experimental Validation of a Power-Split Hybrid Electric Vehicle model," *IEEE Transactions on Vehicular Technology*, vol. 55, no. 6, pp. 1731–1747, Nov. 2006.

[7] A. Sciarretta and L. Guzzella, "Control of Hybrid Electric Vehicles: Optimal Energy-Management Strategies," *IEEE Control Systems Magazine*, pp. 60–70, Apr. 2007.

[8] P. Tulpule *et al.*, "Effects of Different PHEV Control Strategies on Vehicle Performance," *American Control Conference, 2009*, pp. 3950–3955, June 2009.

[9] S. G. Wirasingha and A. Emadi, "Classification and Review of Control Strategies for Plug-In Hybrid Electric Vehicles," *IEEE Transactions on Vehicular Technology*, vol. 60, no. 1, pp. 111–122 Jan. 2011.

[10] R. Patil *et al.*, "A Framework for the Integrated Optimization of Charging and Power Management in Plug-in Hybrid Electric Vehicles," *American Control Conference, 2012*, pp. 1327–1334, June 2012.

[11] R. Patil *et al.*, "Comparison of Optimal Supervisory Control Strategies for a Series Plug-in Hybrid Electric Vehicle Powertrain," *ASME Dynamic Systems and Control Conference*, Oct. 2011.

[12] J. Liue and H. Peng, "Modeling and Control of a Power-Split Hybrid Vehicle," *IEEE Transactionf on Control Systems Technology*, vol. 16, no. 6, pp. 1242–1251, Nov. 2008.

[13] J. Liu, "Modeling, Configuration and Control Optimization of Power-split Hybrid Vehicles," Ph.D. dissertation, Dept. of Mechanical Engineering, University of Michigan, Ann Arbor, 2007.

[14] Q. Gong *et al.*, "Optimal Power Management of Plug-in HEV with Intelligent Transportation System," *IEEE*, 2007.

[15] Y. He *et al.*, "An energy optimization strategy for power-split drivetrain plug-in hybrid electric vehicles," *Transportation Research Part C*, pp. 29–41, 2012.

[16] D. Kum *et al.*, "Supervisory Control of Parallel Hybrid Electric Vehicles for Fuel and Emissions Reduction," *ASME Journal of Dynamic Systems, Measurement and Control*, Apr. 2010.

[17] D. Kum *et al.*, "Optimal Energy and Catalyst Temperatuer Management of Plug-in Hybrid Electric Vehicles for Minimum Fuel Consumption and Tail-Pipe Emissions," *IEE Transactions on Control Systems Technology*, 2011.

[18] D. Kum, "Modeling and Optimal Control of Parallel HEVs and Plug-in HEVs for Multiple Objectives," Ph.D. dissertaion, Dept. of Mechanical Engineering, University of Michigan, Ann Arbor, 2010.

[19] L. Serrao *et al.*, "ECMS as a realization of Pontryagin's minimum principle for HEV control," *American Control Conference, 2009*, pp. 3964–3969, June 2009.

[20] J. Lescot *et al.*, "On the integration of optimal energy management and thermal management of hybrid electric vehicles," *IEEE Vehicle Power and Propulsion Conference*, Sept. 2010.

[21] D. Ambuhl *et al.*, "Explicit optimal control policy and its practical application for hybrid electric powertrains," *Control Engineering Practice*, pp. 1429–1439, 2010.

[22] A. Rousseau *et al.*, "Plug-in Hybrid Electric Vehicle Control Strategy Optimization," *Journal of Asian Electric Vehicles*, vol. 6, no. 2, pp. 1125–1133, 2008.

[23] N. Schouten *et al.*, "Fuzzy Logic Control for Parallel Hybrid Vehicles," *IEE Transactions on Control Systems Technology*, vol. 10, no. 3, pp. 460–468, May 2002.

[24] S. J. Moura *et al.*, "Tradeoffs between battery energy capacity and stochastic optimal power management in plug-in hybrid electric vehicles," *Journal of Power Sources*, pp. 2979–2988, 2010.

[25] S. J. Moura *et al.*, "A Stochastic Optimal Control Approach for Power Management in Plug-in Hybrid Electric Vehicles," *IEEE Transactions on Control Systems Technology*, 2010.

[26] S. J. Moura, "Plug-in Hybrid Electric Vehicle Power Management: Optimal Control and Battery Sizing," M.S. thesis, Dept. of Mechanical Engineering, University of Michigan, Ann Arbor, 2008.

[27] S. Bashash *et al.*, "Plug-in hybrid electric vehicle charge pattern optimization for energy cost and battery longevity," *Journal of Power Sources*, pp. 541–549, 2011.

[28] M. Koot *et al.*, "Energy Management Strategies for Vehicular Electric Power Systems," *IEEE Transactions on Vehicular Technology*, vol. 54, no. 4, pp. 772–782, May 2005.

[29] B. Adornato *et al.*, "Characterizing Naturalistic Driving Patterns for Plug-in Hybrid Electric Vehicle Analysis," *IEEE Vehicle Power and Propulsion Conference*, pp. 655–660, 2009.

[30] C. Desai and S. S. Williamson, "Optimal Design of Parallel Hybrid Electric Vehicle using Multi-Objective Genetic Algorithm," *IEEE Vehicle Power and Propulsion Conference*, pp. 871–876, 2009.

[31] K. Hatzell, A. Sharma, and H. K. Fathy, "A Survey of Long-Term Health Modeling, Estimation, and Control Challenges and Opportunities for Lithium-Ion Batteries," *American Control Conference, 2012*, pp. 584–591, June 2012.

[32] Y. Hu. *et al.*, "Electro-thermal battery model identification for automotive applications," *Journal of Power Sources*, vol. 196, pp. 449–457, 2011.

[33] K. Smith and C. Y. Wang, "Power and thermal characterization of a lithium-ion battery pack for hybrid-electric vehicles," *Journal of Power Sources*, vol. 160, pp. 662–673, 2006.

[34] J. R. Selman *et al.*, "Cooperative research on safety fundamentals of lithium batteries," *Journal of Power Sources*, vol. 97–98, pp. 726–732, 2001.

[35] S. H. .Chan, "A practical approach for rapid catalyst light-off by means of strategic engine control," in *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 215 Part D, pp. 545–555, 2001.

[36] J. T. B. A. Kessels *et al.*, "Towards Integrated Powertrain Control: Thermal Management of NG Heated Catalyst System," *Les Rencontres Scientifiques de l'IFP - Advances in Hybrid Powertrains*, Nov. 2008.