SAE International

SURFACE VEHICLE INFORMATION REPORT

SAE J2841 SEP2010

Issued 2009-03
Revised 2010-09

Superseding J2841 MAR2009

(R) Utility Factor Definitions for Plug-In Hybrid Electric Vehicles Using Travel Survey Data

RATIONALE

Describing the fuel and electrical energy usage of plug-in hybrid electric vehicles (PHEVs) is very challenging for the reason that these values vary greatly depending upon the distance traveled between charging. Detailed test procedures recommended for PHEVs are found within SAE J1711, this document serves to supply appropriate UF curves for the equations in that document that weigh the charge depleting mode operation with the charge-sustaining operation depending upon the charge-depleting mode distance measured from the test. The original issue of SAE J2841 introduced the Utility Factor based upon a mileage-based "Fleet" analysis of the US DOT National Household Transportation Survey data. This issue of SAE J2841 adds more options to the Utility Factor calculations by analyzing another data set from a Georgia Tech called the "Commute Atlanta" in order to find a vehicle-weighted analysis.

1. SCOPE

This SAE Information Report establishes a set of "Utility Factor" (UF) curves and the method for generating these curves. The UF is used when combining test results from battery charge-depleting and charge-sustaining modes of a Plug-in Hybrid Electric Vehicle (PHEV). Although any transportation survey data set can be used, this document will define the included UF curves by using the 2001 United States Department of Transportation (DOT) "National Household Travel Survey" and a supplementary dataset.

1.1 Purpose

In use, the fuel and energy consumption rates of a PHEV vary depending upon the distance driven between charge events. For PHEVs, the baseline assumption regarding any UF is that operation starts fully charged and begins in charge-depleting mode. Eventually, the vehicle must change to a charge-sustaining mode. The vehicle miles traveled between charge events determines how much of the driving is performed in each of the two fundamental modes. A second assumption is that charging occurs every day after the day's driving is complete, i.e. once per day. In the absence of PHEV driver behavior data, the two additional unknown driver behavior issues of (1) how often charging occurs during the day ("opportunity charging") and (2) how often a driver will forget to charge, are assumed to be equally offsetting, thus the baseline assumes one charge per day of operation.

Given the previous assumptions, a UF describes the fraction of driving in each of the fundamental modes using a given set of recorded in-use driving data. Driving statistics from the 2001 National Household Travel Survey and a supplementary dataset are used as inputs to the UF creation to provide UF curves applicable to a vehicle's charge-depleting mode results.

TO PLACE A DOCUMENT ORDER: **Tel: 877-606-7323 (inside USA and Canada)**
Tel: +1 724-776-4970 (outside USA)
Fax: 724-776-0790
Email: CustomerService@sae.org
SAE WEB ADDRESS: http://www.sae.org

2. REFERENCES

2.1 Applicable Documents

SAE 810265, Burke, A.F. and Smith, G.E., "Impacts of Use-Pattern on the Design of Electric and Hybrid Vehicles"

SAE J1711, "Recommended Practice for Measuring the Exhaust Emissions and Fuel Economy of Hybrid-Electric Vehicles, Including Plug-in Hybrid Vehicles"

SAE J1715, "Hybrid Electric Vehicle (HEV) & Electric Vehicle (EV) Terminology"

2001 National Household Travel Survey (NHTS), U.S. Department of Transportation, website:
http://www.bts.gov/programs/national_household_travel_survey/

Commute Atlanta Study Overview, Georgia Tech. School of Civil and Environmental Engineering, website:
http://commuteatlanta.ce.gatech.edu/

Federal Register, Vol. 71 No. 248, Page 77904, December 27, 2006

3. DEFINITIONS

3.1 HYBRID ELECTRIC VEHICLE (HEV)

A road vehicle that can draw propulsion energy from both of the following sources of stored energy: (1) a consumable fuel and (2) a rechargeable energy storage system (RESS) that is recharged by an electric motor-generator system, an external electric energy source, or both.

3.2 PLUG-IN HYBRID-ELECTRIC VEHICLE (PHEV)

A classification describing an HEV with an energy storage system that is designed to be recharged from an external (off-vehicle) electric energy source, typically an AC electrical power supply system.

NOTE: Equivalent to "Off-Vehicle Charge Capable HEV," "Grid-Connected HEV," and "Externally Chargeable HEV"

3.3 VEHICLE MILES TRAVELED (VMT)

Vehicle Miles Traveled (VMT) is the total distance traveled by a vehicle in units of miles.

3.4 UTILITY FACTOR (UF)

The UF indicates the limited utility of a particular initial operating mode - for PHEVs, the CD mode. An operating mode with a very long range, for example, will have a very high utility and, thus, a UF that approaches 1.0. The UF is a function of a vehicle's charge depleting range. The UF is defined by using the assumptions that (1) the vehicle starts the day from a routinely achieved, fully charged state and (2) the vehicle is charged to said state before every day of vehicle travel. As will be discussed later in the document, many possible utility factors can be created and care must be exercised when deciding which utility factor to use for a particular analysis.

3.5 FLEET UTILITY FACTOR (FUF)

The FUF is a utility factor based on the total miles traveled for a specific fleet of vehicles. This UF is created by dividing the depleting miles by the total miles traveled. This UF is particularly useful to calculate the expected fuel and electric energy consumption of an entire fleet of vehicles. The FUF evaluated between 0 and 400 miles is shown in Appendix B.

3.6 INDIVIDUAL UTILITY FACTOR (IUF)

An IUF (IUF) considers all vehicles equally as opposed to using daily VMT weighting which is highly weighted towards long distance trips. This particular utility factor is analogous to the average individual response to a survey of fuel and electric energy consumption rates for a set of vehicles. Two possible options for an IUF are the Single Day and Multiple Day IUFs, which are differentiated by the availability of multiple survey days in the data.

3.7 SINGLE DAY INDIVIDUAL UTILITY FACTOR (SDIUF)

The SDIUF is a specific IUF created when only a single day of travel is available. For most analysis, data from a multi-day survey is more representative for generating IUFs. This is due to the variations of a driver's behavior not being adequately captured using a single day travel survey.

NOTE: The 2001 NHTS survey data only includes data from a single day per household.

3.8 MULTIPLE DAY INDIVIDUAL UTILITY FACTOR (MDIUF)

The MDIUF is a specific IUF created when multiple driving days are available from a travel survey dataset. The MDIUF better incorporates a driver's day-to-day variation into the utility calculation. This is the recommended utility factor to use when calculating an IUF to estimate an individual vehicle's expected fuel economy. The MDIUF evaluated between 0 and 400 miles is shown in Appendix B.

3.9 SPECIFIC UTILITY FACTOR (SUF)

A Specific Utility Factor is a utility factor that uses a filtered subset of data to create a utility factor that represents a certain driving style. The reasoning behind this type of UF is that certain driving styles may have different distance distributions for daily miles traveled and thus may be analyzed with different UFs. For example, if one assumes that a certain driving style leads to shorter daily distances, it might be useful to use a shorter distance subset of a travel survey to create a UF curve for that driving style. Cycle-specific utility factors can be created for either FUFs or IUFs. Note: the applicability of a particular SUF is directly related to the ability to confidently separate driving styles given the amount of information in a particular dataset.

3.10 CITY SPECIFIC FLEET UTILITY FACTOR (CSFUF)

The CSFUF is a Specific Fleet Utility Factor created for City (Urban) driving. Two examples of a CSFUF curve can be found in Appendix E.

3.11 HIGHWAY SPECIFIC FLEET UTILITY FACTOR (HSFUF)

The HSFUF is a Specific Fleet Utility Factor created for Highway driving. Two examples of a HSFUF curve can be found in Appendix E.

3.12 CHARGE-DEPLETING (CD) MODE

Operating mode of a HEV in which the vehicles run by consuming only the electric energy from the mains or along with the fuel energy simultaneously or sequentially until the CS Mode state.

3.13 CHARGE-SUSTAINING (CS) MODE

Operating mode where the HEV runs by consuming the fuel energy while sustaining the electric energy of the RESS.

3.14 CHARGE-DEPLETING RANGE (R_{CD})

Generally speaking, this is the distance a vehicle can travel in Charge-Depleting Mode. Several different range calculations are possible and are specified in SAE J1711.

4. DERIVATION OF UTILITY FACTORS

4.1 Illustrating the Utility Factor Concept

The derivation of a UF requires a data set of daily driving distances by a number of drivers. The following example will illustrate the derivation. Let there be a driving data set of five (5) vehicles that all have different daily miles traveled. Let the charge-depleting range be 40 miles in this example. For this CD range, each vehicle has a different percentage of operation in CD mode. If the miles traveled were less than or equal to 40 miles, then 100% of the driving is in CD mode. If a driver's miles traveled are higher than the CD range, then the CD range divided by the total miles defines this driver's fraction of CD mode travel. Figure 1 illustrates this example driving dataset.

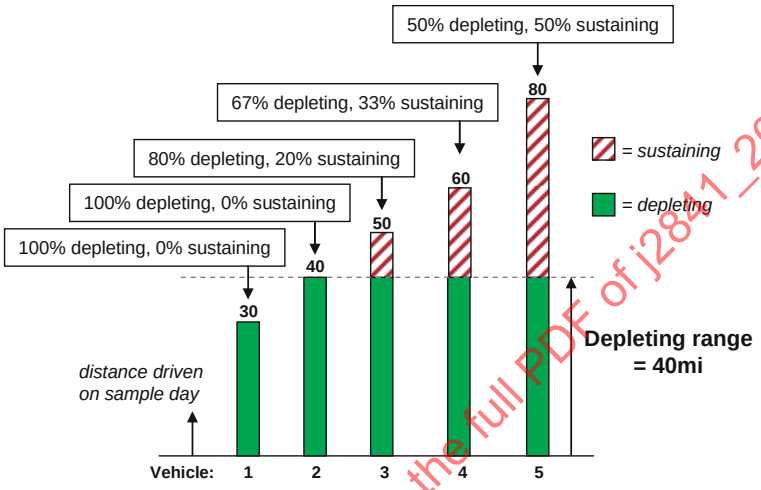

FIGURE 1 - SAMPLE ILLUSTRATION OF AN ORDERED SET OF DAILY DRIVING DISTANCES

To calculate a VMT-weighted UF for this fleet of vehicles, all of the depleting miles are divided by the total miles. In the numerator, the distance is either the driven distance or the CD range, whichever value is lowest, summed over all drivers. The denominator is the sum of the total miles driven by the fleet. For this example, the calculation is shown in Figure 2. Given the usefulness of this factor for describing the expected fuel consumption of an entire fleet of vehicles, it is referred to as the Fleet Utility Factor (FUF)

$$UF\,(40) = \frac{30+40+40+40+40}{30+40+50+60+80}$$

Depleting Miles

Total Miles

FIGURE 2 - FLEET UTILITY FACTOR CALCULATED FOR EXAMPLE ILLUSTRATION

As an alternative to the FUF, the Individual Utility Factor (IUF) considers all vehicles equally as opposed to using a VMT weighting which is highly weighted towards long distance trips. If one wanted to find an individual vehicle-weighted utility factor (IUF), as opposed to a VMT weighted utility factor, this would be found by summing the individual fractions of CD operation versus total miles driven for all drivers and then dividing by the total number of drivers. The fraction of CD operation is defined as 1.0 for drivers driving less than or equal to the CD range and the fraction of CD operation for drivers driving farther than the specified CD range is defined as the charge depleting range divided by the total distance traveled. This utility factor is described in Figure 3. Given the relevance of this utility factor for individual vehicles, this type of utility factor is commonly referred to as an Individual Utility Factor (IUF). In the case of the NHTS, where only one day of driving data is available, this type of utility factor would be termed a Single Day Individual Utility Factor (SDIUF).

Depleting Fractions

$$UF\ (40) = \frac{1}{5}(1+1+0.8+0.67+0.5)$$

Number of Vehicles

FIGURE 3 - INDIVIDUAL UTILITY FACTOR CALCULATED FOR EXAMPLE ILLUSTRATION

Which factor to use for analysis of a particular PHEV depends upon the purpose of the analysis. If the intent is to predict the combined fuel usage of a fleet of vehicles, then the mileage-weighted Fleet UF concept (FUF) should be used. If the intent is to find a prediction of an individual vehicle's expected fuel economy, then a vehicle-weighted Individual Utility Factor (IUF) result should be used.

While the expected fuel consumption of a collection of drivers on a single day can be calculated from the NHTS data, it is also useful to observe daily driving distance over a period of time. This is because drivers will typically have a varying driving distance profile over the course of the year due to business trips, vacations, etc. Since this data is not directly available from the NHTS data, an additional data set was used to incorporate longer term driving behavior. The supplementary dataset used for this analysis was from a Commute Atlanta (CA) survey of roughly 530 vehicles. Although this dataset is much smaller, it contains multiple days of driving data and therefore provides insight into longer term driving patterns that the NHTS data alone is unable to address. With this supplementary dataset, a new Multi-day Individual Utility Factor (MDIUF) may be calculated in a similar fashion to the SDIUF with the inclusion of the multiple driving days. When using an individual utility factor, it is recommended that a MDIUF be used.

4.2 Defining Utility Factor Equations

Let *S* be a reference set of driving data, which is a finite set of driving distances, and let R_{CD} be a given CD range. Let d_i be a daily distance driven by a particular vehicle. Let N_s be the total number of vehicles multiplied by the total number of driving days considered in the set *S*. Equation 1 illustrates the membership in this set.

$$S = \{d_1, d_2, ..., d_{N_S}\} \quad \text{(Eq. 1)}$$

When S is ordered such that ($d_i \le d_{i+1}$), the sorted daily driving distances can be plotted, as shown in Figure 4.

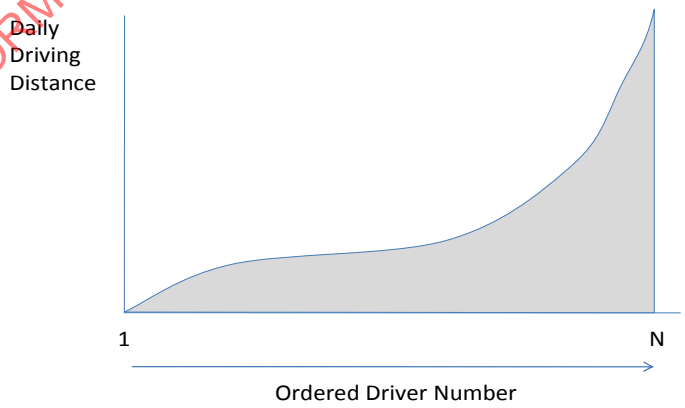

FIGURE 4 - ORDERED REFERENCE SET OF N DAILY DRIVING DISTANCES

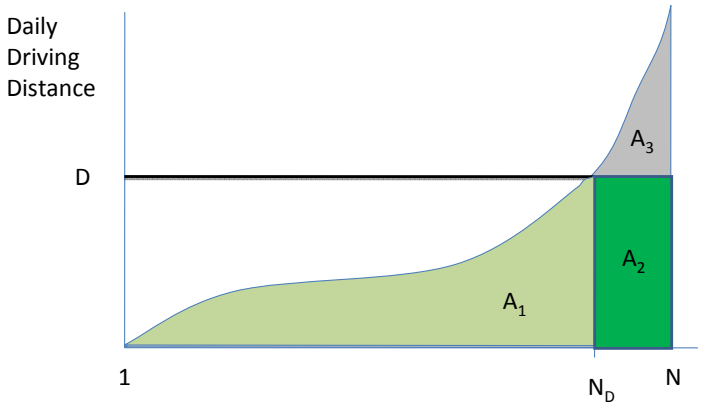

FIGURE 5 - DATA PARTITIONS FOR UTILITY FACTOR (UF) DERIVATION

Consider the partitions illustrated in Figure 5 for the following parameters. The fraction of drivers traveling less than *D* miles per day is described in Equation 2.

$$F(D) = \frac{N_D}{N} \quad \text{(Eq. 2)}$$

The fraction of VMT driven in the initial mode up to *D* miles (charge-depleting or EV range, *D*) is depicted in Equation 3.

$$UF(D) = \frac{A_1 + A_2}{A_1 + A_2 + A_3} \quad \text{(Eq. 3)}$$

Thus, the UF calculation, given the initial data set S, is the sum of the minimum of either R_{CD} or the driving distance of d_k data, divided by the sum of all the distances. This calculation is shown in Equation 4.

$$UF\ (R_{CD}) = \frac{\sum_{d_k \in S} \min(d_k, R_{CD})}{\sum_{d_k \in S} d_k} \quad \text{(Eq. 4)}$$

As discussed previously, an IUF may be desirable when conveying information for an average vehicle. The basic buildup of this utility factor is similar to the previous discussion, but rather than a percentage of total miles traveled, this utility factor represents the expected depleting fraction enabled by a particular depleting range relative to the set of vehicles. Equation 5 illustrates this calculation.

$$IUF\ (R_{CD}) = \frac{\sum_{i=1}^{nVehicles} \frac{\sum_{j=1}^{nDays} \min(d_{i,j}, R_{CD})}{\sum_{j=1}^{nDays} d_{i,j}}}{nVehicles} \quad \text{(Eq. 5)}$$

5. NHTS AND COMMUTE ATLANTA SURVEY BACKGROUND

"The National Household Travel Survey (NHTS) is a U.S. Department of Transportation (DOT) effort sponsored by the Bureau of Transportation Statistics (BTS) and the Federal Highway Administration (FHWA) to collect data on both long-distance and local travel by the American public. The joint survey gathers trip-related data such as mode of transportation, duration, distance and purpose of trip. It also gathers demographic, geographic, and economic data for analysis purposes." - Obtained February 2010 from *http://www.bts.gov/programs/national_household_travel_survey/*

The 2001 NHTS was conducted during the period March 2001 through May 2002. Each sampled household was assigned a travel day to report all trips made on that day. Travel days were assigned for all seven days of the week, including holidays. The travel day began at 4 AM and ended at 3:59 AM of the following day. On a typical day, 4 AM represents the time when the fewest number of people are in the middle of a trip.

"In 2004, Commute Atlanta monitored one full year of baseline travel activity from more than 275 participating households. These volunteer households allowed the research team to professionally install a GT Trip Data Collector in their vehicles. Approximately 465 vehicles in these households were equipped with instrumentation to monitor second-by-second vehicle speed and position for every trip. Researchers remotely monitor the travel patterns of these vehicles, uploading vehicle and engine operating data via a cellular data connection. General travel data, such as number of trips per household per day and selected travel routes, are used to evaluate transportation demand models currently used in Atlanta's transportation planning process. Vehicle position and speed data are used to identify locations of recurrent traffic congestion…"

"More than 100 households participated in Phase II value-pricing research designed to assess household travel response to simulated payment of transportation costs on a per-mile basis. A similar number of households will experience real-time congestion pricing later this year. Analytical efforts for this project are expected to continue throughout 2007." - Obtained February 2010 from http://commuteatlanta.ce.gatech.edu/

By filtering the Commute Atlanta dataset, the multi-day driving patterns of the survey drivers may be calculated and used as a multi-day supplement to the NHTS data.

The CA dataset was found to have FUF and SDIUF distributions similar to the NHTS dataset and was therefore decided to be a reasonable supplement to the NHTS dataset. The following Figures 6 and 7 compare the Commute Atlanta and NHTS data sets relative to the FUF and SDIUF curves. The plots on the right side of Figures 6 and 7 are an expanded view of the curve between 0 and 100 miles.

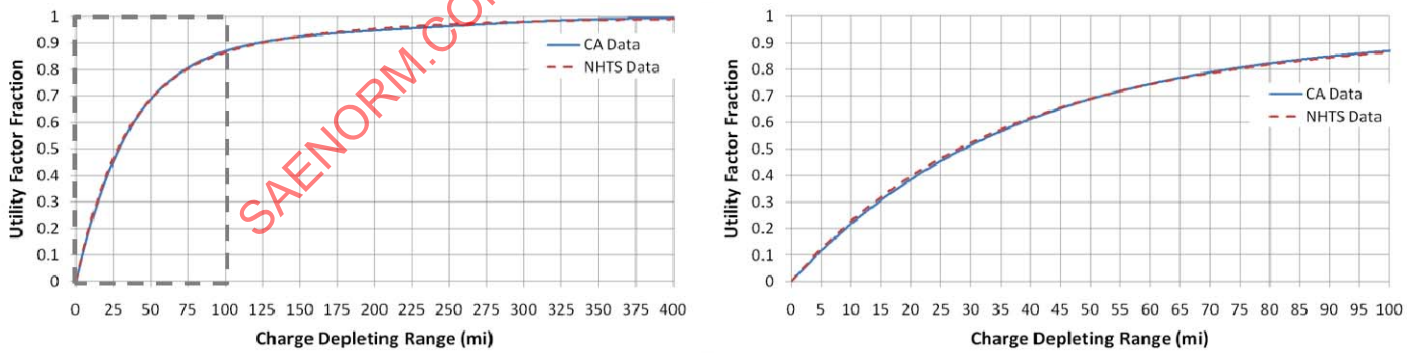

FIGURE 6: FLEET UTILITY FACTOR COMPARISON: COMMUTE ATLANTA (CA) VERSUS NHTS (FULL AND <100 MI)

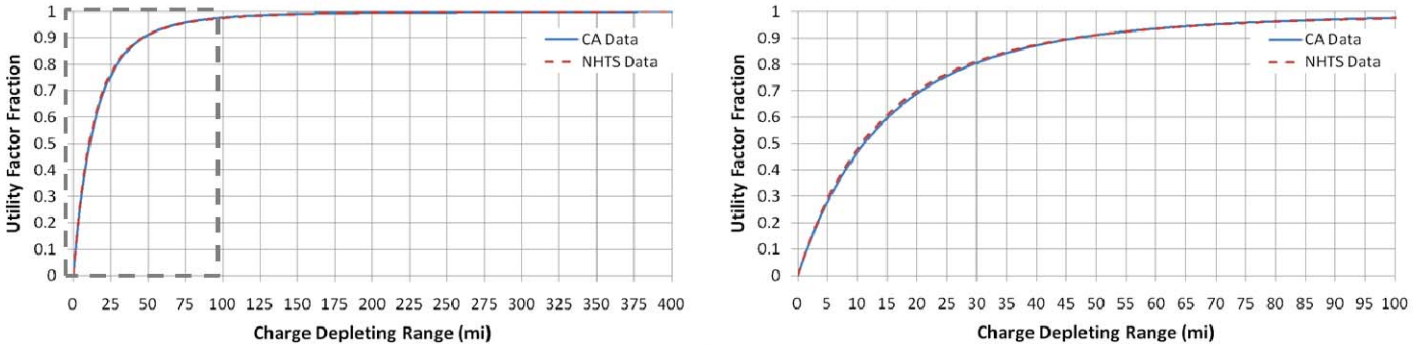

FIGURE 7 - SINGLE-DAY INDIVIDUAL UTILITY FACTOR COMPARISON: COMMUTE ATLANTA (CA) VERSUS NHTS (FULL AND <100 MI)

6. UTILITY FACTOR PROCESSING AND VALUES

6.1 Data Processing

The 2001 NHTS data in the "Day Trip File" named DAYPUB comes in SAS, database, and ASCII format. It consists of separate individual trips from the entire survey. The total number of records in the file is 642 292. The data were reduced by using the filters and selected parameters listed in Table 1.

TABLE 1 - FIELDS USED TO EXTRACT SET OF DAILY DRIVING DISTANCES FROM NHTS DATA

Field	Filter	Remarks
DRVR_FLG	=1	1 = Subject was driver on this trip
SMPLSRCE	=1	1 = National Sample
TRPMILES	>0	Trip distance in miles
TRVL_MIN	>0	Time to complete entire trip in minutes
VEHTYPE	= 1, 2, 3, or 4	Type of vehicle, 01=Car, 02=Van, 03=SUV, 04=Pickup truck, 05=Other truck, 06=RV, 07=Motorcycle, 91=Other

One last step was taken before the UF equations could be applied. For each household vehicle (unique "HOUSEID" and "VEHUSED" label in data set) that was used on the travel day, trip distance and trip time were summed to arrive at daily vehicle usage. The resulting data provides daily driving distances for roughly 32 000 vehicles and is used alongside the supplementary dataset as the input for the utility factor analysis in this document.

6.2 Utility Factor Exponential Equation Fits

The various UF results can be described by entering the depleting range into the general form shown in Equation 6, where x is the input depleting range. The coefficients for the FUF and MDIUF fit calculations are shown in Table 2. The UF results are valid from 0 to the normalized distance where the Utility Factor converges to 1.0. The equation structure for fitting a UF curve was arrived at experimentally and was found to provide a good fit with relatively few coefficients or significant digits. The fit coefficients were found using a modified least squares approach, which ensures that the fits are monotonic. The fit coefficients for the FUF are identical to those found in the previous version of SAE J2841. For the MDIUF, the fitting algorithm was further adjusted to achieve a mix of low relative error and a minimum number of coefficients/significant digits. The maximum fit-error was less than 0.004 and 0.007 for the FUF and MDIUF respectively. An example fitting script is shown in Appendix A and may be adjusted to find the desired coefficient number and resolution to be used in Equation 6.

$$UF = 1 - \exp\{ -[C1*(x/norm\_dist) + C2*(x/norm\_dist)^2 + \ldots + C10*(x/ norm\_dist)^{10}]\} \quad \text{(Eq. 6)}$$

TABLE 2 - UTILITY FACTOR EQUATION COEFFICIENTS

Value	FUF Fit	MDIUF Fit
norm_dist	399.9	400
C1	10.52	1.31E+01
C2	-7.282	-1.87E+01
C3	-26.37	5.22E+00
C4	79.08	8.15E+00
C5	-77.36	3.53E+00
C6	26.07	-1.34E+00
C7	-	-4.01E+00
C8	-	-3.90E+00
C9	-	-1.15E+00
C10	-	3.88E+00
Max Error	0.00391	0.00658

Due to some unique processing requirements, the Multi-day Individual utility factor has a limited set of points available for data fitting. More specifically, the MDIUF data was fit using 20 points distributed across the entire 400 mile range. In order to provide a more accurate fit in the regions of typical interest, the majority of the fit points were allocated between 0 and 50 miles. To evaluate that this reduced set of points was sufficient to match UF shaped data, a fit was created using the full FUF curve and the 20 point reduced set. Figure 8 shows that the reduced point fit is close to the full-data fit and suggests that the 20 points used for fitting the MDIUF should be suitable.

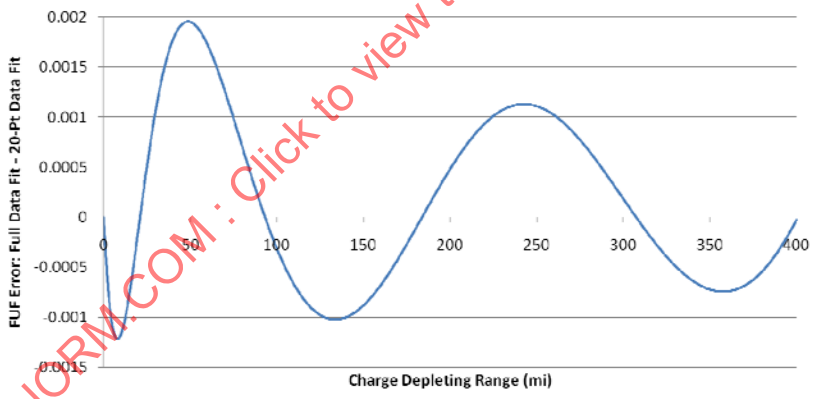

FIGURE 8 - FULL AND 20 POINT FUF CURVE ERROR PLOT

Utilizing Equation 6 and the coefficients in Table 2, the Fleet Utility Factor and Multi-day Individual Utility Factor curves are shown in Figure 9. The plot on the right side of Figure 9 shows an expanded view of the Utility Curves in the 0 to 100 mile range. Appendix B contains a table of the FUF and MDIUF equations evaluated at 1 mile increments and rounded to the nearest 0.001. For non-integer distances, it is recommended that the proper UF fit equation be evaluated to calculate the UF.

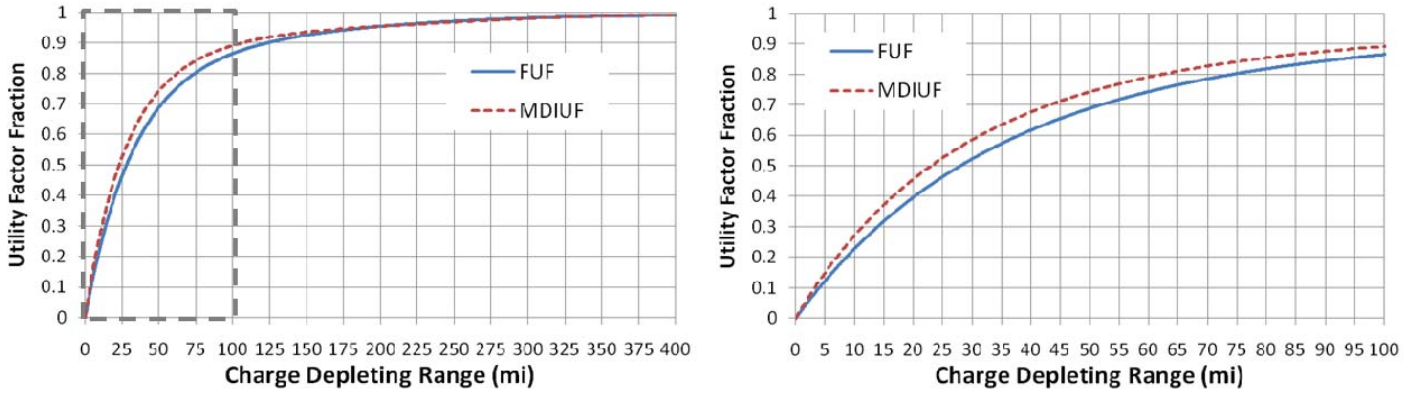

FIGURE 9 - FUF AND MDIUF WITH RESPECT TO CHARGE-DEPLETING RANGE

7. CITY AND HIGHWAY SPECIFIC UTILITY FACTORS

7.1 Calculating City and Highway Specific Utility Factors

In the FUF and MDIUF, all driving data from a particular survey is included together to create the UF curve. However, different types of driving may differ in the distribution of distance traveled. For example, some high-speed drivers will likely travel farther than the majority of stop-and-go drivers. Because energy consumption for a PHEV is a function of distance driven between charges, a UF which is specific to either a particular style of driving may be useful. A UF constructed for use with a particular type of driving is referred to as a Specific Utility Factor (SUF) and can be created as either an FUF or IUF.

The City and Highway Specific UFs can be built from a data set that contains the distance driven between charges for one or more vehicles over one or more days. For simplicity, one charge per day is assumed. The equations are normalized to work with data sets of one or more days of data per vehicle. Each data point in the set is assigned a weight with a value between zero and one. Using this data, the cycle-specific FUF for a fleet is shown in Equation 7. More information regarding the properties of Specific UFs can be found in Appendix D.

$$SUF_{Fleet}(R_{cd}) = \frac{\sum_{k=1}^{K} \frac{1}{N_k} \cdot \sum_{n=1}^{N_k} w_{k,n} \cdot \min\left(R_{cd}, d_{k,n}\right)}{\sum_{k=1}^{K} \frac{1}{N_k} \cdot \sum_{n=1}^{N_k} w_{k,n} \cdot d_{k,n}} \quad \text{(Eq. 7)}$$

Similarly, the cycle-specific IUF for an individual is shown in Equation 8.

$$SUF_{Individual}(R_{CD}) = \frac{\sum_{k=1}^{K}\left(\left(\frac{\sum_{n=1}^{N_k} w_{k,n} \cdot d_{k,n}}{\sum_{n=1}^{N_k} d_{k,n}}\right) \cdot \left(\frac{\sum_{n=1}^{N_k} w_{k,n} \cdot min\left(R_{CD}, d_{k,n}\right)}{\sum_{n=1}^{N_k} w_{k,n} \cdot d_{k,n}}\right)\right)}{\sum_{k=1}^{K}\left(\frac{\sum_{n=1}^{N_k} w_{k,n} \cdot d_{k,n}}{\sum_{n=1}^{N_k} d_{k,n}}\right)} \quad \text{(Eq. 8)}$$

TABLE 3 - CYCLE-SPECIFIC UF EQUATION VARIABLES

Variable	Definition
$d_{k,n}$	Distance driven on each day.
K	Number of vehicles in data set
N	Number of days in data set
R_{cd}	Charge decreasing range.
$w_{k,n}$	Weighting applied to each day, contains a value between zero and one.

These equations are a special case of the composite UF calculation. In the case where all weights are equal to one, these equations are identical to the composite UF calculations. These equations assume that the data set results in the sum of weights and distances having a non-zero value; the total distance traveled is greater than zero. For an SUF, if a vehicle has no weighted distance, that vehicle should be removed from the data set for that cycle-specific calculation. For these SUFs, if the weighted distance traveled is zero, the SUF is undefined.

One particular option for creating SUF curves may be computed from travel surveys, such as the NHTS, using only the trip distances and trip travel times. Since only trip information is available, each trip is wholly assigned to either city or highway driving. In the NHTS, only average trip time and trip distance is available and thus an allocation must be based on calculated vehicle speed alone. For example, consider the common assumption of 55% of vehicle miles traveled (VMT) in urban settings and 45% in highway settings for partitioning trips into bins for city and highway driving. The slowest 55% of VMT is assigned to city driving. The trips that are associated with this travel are considered to be city driving samples. The remaining trips are assigned to highway driving. Figure 10 shows the cumulative VMT from the NHTS versus the average trip speed. This chart is generated by taking each trip, calculating the average speed for the trip, then sorting the list and plotting the cumulative distance versus the average trip speed. The speed below which 55% of VMT occurs is 42 mph. For illustration purposes, an additional set of SUF curves have also been included in the final coefficients table with a 43% city versus 57% highway split. More discussion regarding the two different split proportions may be found in the Federal Register, Vol. 71 No.248.

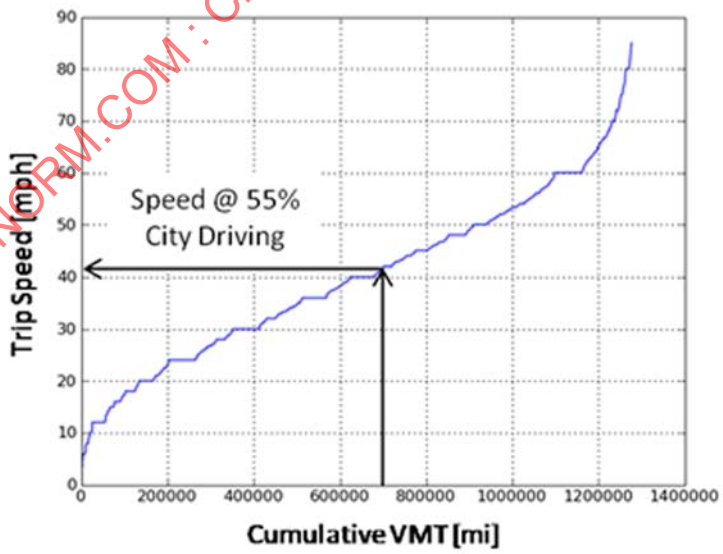

FIGURE 10 - CUMULATIVE VMT VERSUS AVERAGE TRIP SPEED

To calculate the urban weight for a day's travel, $d_{k,n}$,use

$$w_{urban,k,n} = \frac{\sum_{m=1}^{M_{k,n}} \{ t_{k,n,m} \text{ if } s_{k,n,m} \le s_{threshold}, \text{else } 0 \}}{\sum_{m=1}^{M_{k,n}} t_{k,n,m}}$$

(Eq. 9)

To calculate the highway weight for a day's travel, $d_{k,n}$,use

$$w_{highway,k,n} = \frac{\sum_{m=1}^{M_{k,n}} \{ t_{k,n,m} \text{ if } s_{k,n,m} > s_{threshold}, \text{else } 0 \}}{\sum_{m=1}^{M_{k,n}} t_{k,n,m}}$$

(Eq. 10)

In either equation, if no travel occurs (e.g. $\sum_{n=1}^{N} t_{k,n,m}$ equals zero), the weight is assigned a value of zero. In both of these equations, the variables in Tables 3 and 4 are used.

TABLE 4 - CYCLE-SPECIFIC UF EQUATION VARIABLES

Variable	Definition
$M_{k,n}$	Number trips taken by vehicle k on day n
$s_{k,n,m}$	7.2 Average trip speed for trip m by vehicle k on day n
$s_{theshold}$	7.2.1.1.1.1 Trip speed threshold between urban and highway
$t_{k,n,m}$	7.2.1.2 Distance driven on trip m by vehicle k on day n

Using this process with the 2001 NHTS data and city/highway splits of 55/45 and 43/57 percent city to percent highway driving, the City and Highway Specific FUF curves are shown in Figure 11. As a supplement to Equations 7-10, Appendix C provides an example Python script file which can be used to parse the NHTS data and create the City and Highway Specific FUF curves discussed in this document.

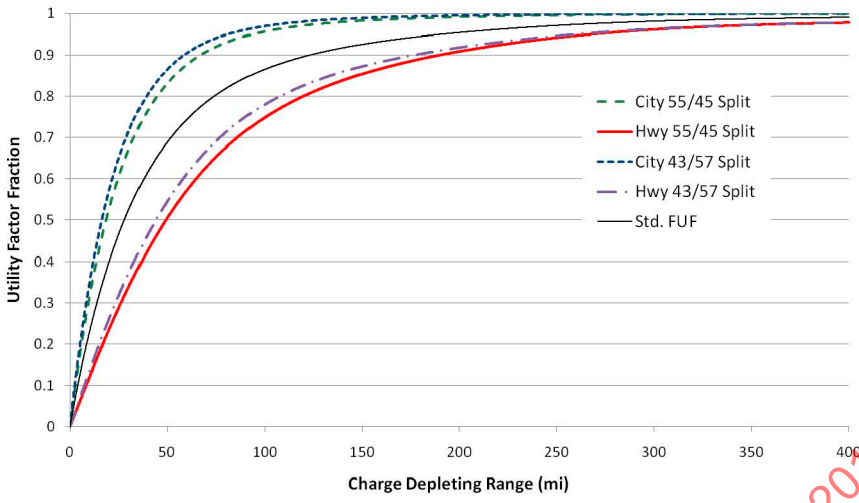

FIGURE 11 - CITY AND HIGHWAY SPECIFIC FUF VALUES

The corresponding fit coefficient table for these curves is shown in Table 5. Appendix E contains tables of the CSFUF and HSFUF equations evaluated at 1 mile increments and rounded to the nearest 0.001. For non-integer distances, it is recommended that the appropriate UF fit equation be evaluated to calculate the UF.

TABLE 5 - FIT COEFFICIENTS FOR EXAMPLE CITY/HWY SPECIFIC FUF CURVES

Value	55/45 Split - City	55/45 Split - Hwy	43/57 Split - City	43/57 Split - Hwy
norm_dist	399	399	399	399
C1	1.486E+01	4.8E+00	1.69E+01	5.43E+00
C2	2.965E+00	1.3E+01	1.84E+00	1.49E+01
C3	-8.405E+01	-6.5E+01	-9.63E+01	-8.00E+01
C4	1.537E+02	1.2E+02	1.86E+02	1.50E+02
C5	-4.359E+01	-1.0E+02	-5.60E+01	-1.26E+02
C6	-9.694E+01	3.1E+01	-1.23E+02	3.95E+01
C7	1.447E+01	-	1.99E+01	-
C8	9.170E+01	-	1.21E+02	-
C9	-4.636E+01	-	-6.30E+01	-
Max Error	0.00558	0.00387	0.00683	0.00487

7.3 Discussion Regarding the Use and Creation of Specific Utility Factors

There are numerous ways to generate the weighting for each data point, d_{kn}, and therefore many possible methods and SUF curves. Moreover, the applicability of the allocation between types of driving is only as good as the weighting method used to parse the cycles. If continuous data for driving is available, features of each driving trace can be used to determine the weighting applied to a particular data point. For example, to build a City Specific UF, the weighting for each data point could be determined from features such as number of stops, top speed, average speed, and average moving speed. Although the weighting criteria may change, the process in Equations 7 to 10 should remain the same with the exception of the threshold in Equations 9 and 10. Given the many options for creating SUF curve types, care must be taken to ensure that sufficient data exists to create a representative SUF curve and that the SUF used is appropriate for the given analysis situation.

To illustrate the breadth of options regarding methods to allocate City and Highway driving, several alternative options are presented here for discussion. The first option is similar to the previous cutoff speed method, but in this case two distinct speeds are used to differentiate between City and Highway driving style. In addition to the use of two cutoff speeds, this method assigns a City weighting between 0 and 1 according to average trip speed whereas the previous method fully allocated a trip to either City or Highway driving. For trips with an average speed above 60 mph, the trip is assumed to 100 percent Highway driving and thus receives a value of 1 for the Highway weighting and a value of 0 for the City weighting. Similarly, trips with an average speed under 25 mph are wholly assigned to City driving. Between these two cutoff speeds, trips receive a weighting that is scaled linearly between 0 and 1. In order to retain a City/Highway driving mix similar to the previous method, the City cutoff speed (25 mph) for this example was chosen to create a 55 percent City to 45 percent Highway mix relative to total miles driven. The Highway cutoff speed (60 mph) was simply chosen as a reasonable estimate. Figure 12 illustrates the weighting values as a function of average trip speed.

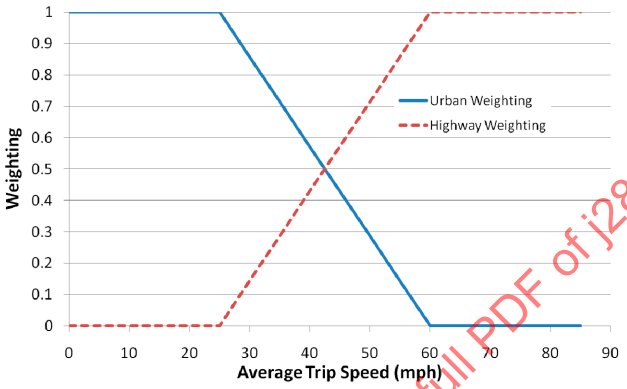

FIGURE 12 - CITY AND HIGHWAY WEIGHTS FOR ALTERNATIVE SPEED-BASED ALLOCATION METHOD

Another related method for allocating trips based on average speed is to simply create a cutoff speed for exclusively City driving and exclusively Highway. Trips between these two cutoff speeds are assumed to be equally likely to be City of Highway driving and thus are included in both the City and Highway datasets used for calculating the SUFs. This method creates two sets of overlapping data to create the City and Highway SUFs while removing the trips that are unlikely to be driven in the style of the SUF being calculated. Figure 13 illustrates the three distinct sections; the cutoff speeds shown are used for example only and would need to be adjusted depending on the desired amount of overlap between City and Highway driving. In this example, the actual mix of City and Highway relative to VMT is unknown due to the overlapping region.

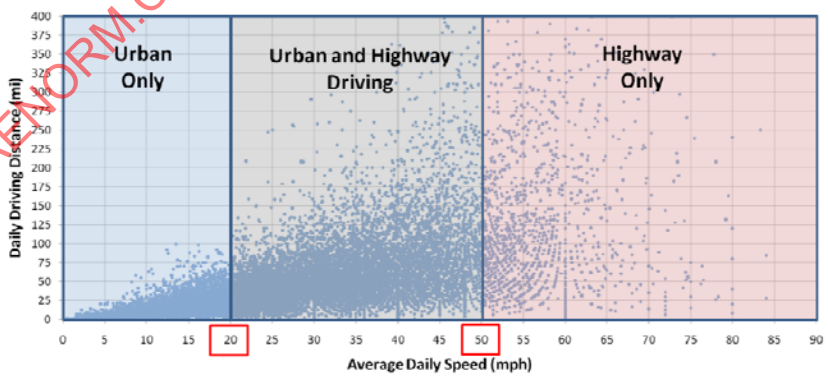

FIGURE 13 - DRIVING STYLE SECTIONS FOR OVERLAPPING DATASET ALLOCATION METHOD

The last alternative method proposed for discussion attempts to incorporate additional information into the determination of driving style. This method seeks to categorize an entire day of driving data using the average daily speed and the daily amount of trips. It is assumed that in addition to reduced speed, City driving will contain more trips compared to Highway driving. This method allocates the entire day to City or Highway driving due to the nature of the NHTS data, but a similar method could be done on a trip-by-trip basis using stops as opposed to trips. Figure 14 illustrates the allocation of City and Highway trips using a speed/number-of-trips dividing line. The dividing line allows for higher speed driving to still be categorized as City driving if there are a sufficient number of trips during the day. The dividing line was set so that the driving mix was again 55 percent City to 45 percent Highway.

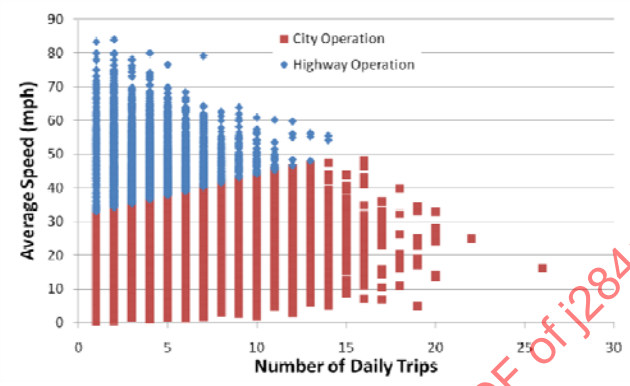

FIGURE 14 - EXAMPLE ALLOCATION OF TRIPS USING AVERAGE SPEED AND NUMBER OF TRIPS

Additionally, not all datasets should be parsed using a prescribed percentage mix of City and Highway driving. For example, if a particular dataset is known to be biased towards a particular driving style in terms of total vehicles sampled, using a percentage of miles driven weighting criterion may not be the most representative methodology. Moreover, if a method to allocate driving style is known to work well, this allocation technique should take precedence over the expected percentage of VMT methodology discussed previously. In the case of the 2001 NHTS dataset, a national sampling, it is assumed that the mix of driving should be reasonably similar to the existing City versus Highway trends. As with the discussion in the previous paragraphs, more data would be needed to more conclusively allocate between City and Highway driving for the included NHTS sample or any other dataset used to create a Specific Utility Factor.

8. NOTES

8.1 Marginal Indicia

A change bar (l) located in the left margin is for the convenience of the user in locating areas where technical revisions, not editorial changes, have been made to the previous issue of this document. An (R) symbol to the left of the document title indicates a complete revision of the document, including technical revisions. Change bars and (R) are not used in original publications, nor in documents that contain editorial changes only.

APPENDIX A - UTILITY FACTOR CURVE FIT ALGORITHM

The algorithm used to fit the parameters for the various utility factors is a modification of the usual regression methods. When a curve is fit by using linear regression, the fit is unconstrained, and the resulting fit may have numeric properties that do not reflect the known behavior of a noise-free data set. For the utility factor, we know that the curve is monotonic and equal to zero at the origin. Furthermore, instead of minimizing the square of the absolute errors, we would like to minimize the square of the relative errors. Taking these criteria into account, a constrained optimization problem is formed and used to find the fit parameters for each curve. The function used in the fit was found by experimentation. Future revisions of the underlying utility factors may be fit best by different functions.

Example Matlab™ code to find the parameter fits is provided below in Figure A1.

```
clear variables

fit_order = 6;                % maximum order is 9
force_monotonicity = true;    % adds constraints to force monotonic shape
use_weighted = true;          % adds weighting so smaller UF values are fit tighter
force_UF_to_1 = false;         % forces the utility factor to equal 1 at 400 miles

urban = 2;
hwy = 3;
total = 4;
ind_urban = 5;
ind_hwy = 6;
ind_total = 7;


fit_target = ind_hwy;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% if fit_target == hwy || fit_target==total
%     force_UF_to_1 = false;
% end

load Full_UF_Data.mat

dd = Mi;
if fit_target == urban
    UF = UrbanUF;
elseif fit_target == hwy
    UF = HWYUF;
elseif fit_target == total
    %UF = UF; % UF is already the correct variable
elseif fit_target == ind_urban
    UF = IndUrbanUF;
elseif fit_target == ind_hwy
    UF = IndHWYUF;
elseif fit_target == ind_total
    UF = IndUF;
end
UF_ref = UF;
%UF   = UF_Data(1:end,fit_target);
%dd   = UF_Data(1:end,1);



UF = […];   % utility factor values at each distance
dd = […];   % distance to use for each utility factor value


% clamp data to first entry with a value of 1
```

```
% e.g. delete all redundant entries with a value of 1
idx = find(UF>=1);
if length(idx)>1
    UF = -log(1-UF(1:idx(1)-1));
    UF_ref =UF_ref((1:idx(1)-1));
    dd = (dd(1:idx(1)-1));
else
    UF = -log(1-UF);
    %UF_ref =UF_ref((1:idx(1)-1));
    %dd = (dd(1:idx(1)-1));
end


% build matrix for the optimization
A = [];
b = [];
Aeq = [];
beq =[];

norm_dist = max(dd);

c1 = dd/norm_dist;
c2 = (dd/norm_dist).^2;
c3 = (dd/norm_dist).^3;
c4 = (dd/norm_dist).^4;
c5 = (dd/norm_dist).^5;
c6 = (dd/norm_dist).^6;
c7 = (dd/norm_dist).^7;
c8 = (dd/norm_dist).^8;
c9 = (dd/norm_dist).^9;
c10 = (dd/norm_dist).^10;
c11 = (dd/norm_dist).^11;
c12 = (dd/norm_dist).^12;

dc1 = ones(size(c1));
dc2 = 2*(dd/norm_dist);
dc3 = 3*(dd/norm_dist).^2;
dc4 = 4*(dd/norm_dist).^3;
dc5 = 5*(dd/norm_dist).^4;
dc6 = 6*(dd/norm_dist).^5;
dc7 = 7*(dd/norm_dist).^6;
dc8 = 8*(dd/norm_dist).^7;
dc9 = 9*(dd/norm_dist).^8;
dc10 = 10*(dd/norm_dist).^9;
dc11 = 11*(dd/norm_dist).^10;
dc12 = 12*(dd/norm_dist).^11;



C  = [ c1  c2  c3  c4  c5  c6  c7 c8  c9 c10 c11 c12];    % fit terms
dC = [dc1 dc2 dc3 dc4 dc5 dc6 dc7 dc8 dc9 dc10 dc11 dc12];   % constraint terms
W  = (1./(eps+UF_ref));                          % weighting for fit

C= C(:,1:fit_order);
dC = dC(:,1:fit_order);

d = UF;

if force_UF_to_1
   Aeq = [C(end,:) ]; % force UF=1 at 400 miles
   beq = [1; ];
else
   Aeq = [C(end,:) ]; % force UF=1 at 400 miles
   beq = [UF(end); ];
end
```

```
if force_monotonicity
    % set up constraints to force slope to be positive for UF
    % this prevent non-monotonicity
    A = -dC;
    b = zeros(size(dC,1),1);
end

if use_weighted
    C0 = repmat(W,1,fit_order).*C;
    d0 = W.*d;
else
    C0 = C;
    d0 = d;
end

options=optimset('Diagnostics','on','Display','iter','TolCon',1e-15,'TolX',1e-12,'TolF',1e-
12,'MaxIter',5000);
[p,resnorm,residual,exitflag,output,lambda] = lsqlin(C0,d0,A,b,Aeq,beq,[],[],[],options);

p2 = zeros(12,1);
p2(1:length(p))=p;
% p= p2;


fit_curve = C*p;
fit_curve = 1-exp(-C*p)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(1)
subplot(2,1,1)
plot(dd,UF_ref,'.r');
hold on
plot(dd,fit_curve,'k');
title('UF data vs Constrained Fit');
legend('Raw Data','Fit Curve');
grid on
hold off

subplot(2,1,2)
semilogy(dd,fit_curve);
title('Slope of fit');
grid on
hold off

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(2)
subplot(2,1,1)
plot(dd,fit_curve-UF_ref,'.')
title('fit error');
hold off
grid on

subplot(2,1,2)
hist(fit_curve-UF_ref,100)
title('Histogram of fit error');
hold off
grid on

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(3)
subplot(2,1,1)
plot(dd,(fit_curve-UF_ref)./(UF_ref+eps),'.')
title('relative fit error');
```

```
hold off
grid on

subplot(2,1,2)
hist((fit_curve-UF_ref)./(UF_ref+eps),100)
title('Histogram of relative fit error');
hold off
grid on

%fprintf('UF = \n%25.17f*(D/400) + \n%25.17f*(D/400)^2 + \n%25.17f*(D/400)^3 + \n%25.17f*(D/400)^4
+ \n%25.17f*(D/400)^5',p2(1),p2(2),p2(3),p2(4),p2(5));
%fprintf('\n%25.17f*(D/400)^6 + \n%25.17f*(D/400)^7 + \n%25.17f*(D/400)^8 + \n%25.17f*(D/400)^9
+',p2(6),p2(7),p2(8),p2(9));
%fprintf('\n%25.17f*(D/400)^10 + \n%25.17f*(D/400)^11 + \n%25.17f*(D/400)^12 +
\n%25.17f*(D/400)^13\n',p2(10),p2(11),p2(12));

fprintf('UF = 1-exp(-(');
fprintf('\n%25.17f*(D/norm_dist) + \n%25.17f*(D/norm_dist)^2 + \n%25.17f*(D/norm_dist)^3 +
\n%25.17f*(D/norm_dist)^4 + \n%25.17f*(D/norm_dist)^5',p2(1),p2(2),p2(3),p2(4),p2(5));
fprintf('\n%25.17f*(D/norm_dist)^6 + \n%25.17f*(D/norm_dist)^7 + \n%25.17f*(D/norm_dist)^8 +
\n%25.17f*(D/norm_dist)^9 +',p2(6),p2(7),p2(8),p2(9));
fprintf('\n%25.17f*(D/norm_dist)^10 + \n%25.17f*(D/norm_dist)^11 + \n%25.17f*(D/norm_dist)^12 +
\n%25.17f*(D/norm_dist)^13\n',p2(10),p2(11),p2(12));
fprintf('))\n');

fprintf('\nnorm_dist = %10.3f\n',norm_dist);
```

FIGURE A1 - EXAMPLE MATLAB™ CODE TO CALCULATE FIT PARAMETERS

APPENDIX B - FLEET AND MULTI-DAY INDIVIDUAL UTILITY FACTOR TABLES

Fleet Utility Factor (FUF)

Distance (mi)	UF	Distance (mi)	UF	Distance (mi)	UF	Distance (mi)	UF
0	0.000	41	0.625	82	0.824	330	0.985
1	0.026	42	0.633	83	0.827	340	0.986
2	0.051	43	0.641	84	0.829	350	0.987
3	0.075	44	0.648	85	0.832	360	0.988
4	0.099	45	0.655	86	0.835	370	0.989
5	0.122	46	0.662	87	0.837	380	0.989
6	0.145	47	0.669	88	0.840	390	0.990
7	0.166	48	0.676	89	0.842	400+	1.000
8	0.187	49	0.682	90	0.844		
9	0.208	50	0.689	91	0.847		
10	0.228	51	0.695	92	0.849		
11	0.247	52	0.701	93	0.851		
12	0.265	53	0.707	94	0.853		
13	0.284	54	0.712	95	0.855		
14	0.301	55	0.718	96	0.857		
15	0.318	56	0.723	97	0.859		
16	0.335	57	0.728	98	0.861		
17	0.351	58	0.734	99	0.863		
18	0.367	59	0.738	100	0.865		
19	0.382	60	0.743	110	0.882		
20	0.397	61	0.748	120	0.896		
21	0.411	62	0.753	130	0.907		
22	0.425	63	0.757	140	0.917		
23	0.438	64	0.761	150	0.925		
24	0.451	65	0.766	160	0.932		
25	0.464	66	0.770	170	0.939		
26	0.477	67	0.774	180	0.944		
27	0.489	68	0.778	190	0.949		
28	0.500	69	0.782	200	0.954		
29	0.512	70	0.785	210	0.958		
30	0.523	71	0.789	220	0.962		
31	0.533	72	0.793	230	0.965		
32	0.544	73	0.796	240	0.968		
33	0.554	74	0.800	250	0.971		
34	0.564	75	0.803	260	0.973		
35	0.573	76	0.806	270	0.976		
36	0.583	77	0.809	280	0.978		
37	0.592	78	0.812	290	0.980		
38	0.600	79	0.815	300	0.981		
39	0.609	80	0.818	310	0.983		
40	0.617	81	0.821	320	0.984		

Multi Day Individual Utility Factor (MDIUF)

Distance (mi)	UF	Distance (mi)	UF	Distance (mi)	UF	Distance (mi)	UF
0	0.000	41	0.684	82	0.859	330	0.985
1	0.032	42	0.692	83	0.861	340	0.986
2	0.063	43	0.699	84	0.864	350	0.988
3	0.093	44	0.706	85	0.866	360	0.989
4	0.121	45	0.712	86	0.868	370	0.990
5	0.149	46	0.719	87	0.870	380	0.990
6	0.175	47	0.725	88	0.872	390	0.991
7	0.200	48	0.731	89	0.874	400	0.992
8	0.225	49	0.737	90	0.875	>400	1.000
9	0.248	50	0.743	91	0.877		
10	0.271	51	0.748	92	0.879		
11	0.293	52	0.754	93	0.881		
12	0.314	53	0.759	94	0.882		
13	0.334	54	0.764	95	0.884		
14	0.353	55	0.769	96	0.886		
15	0.372	56	0.774	97	0.887		
16	0.390	57	0.778	98	0.889		
17	0.407	58	0.783	99	0.890		
18	0.424	59	0.787	100	0.892		
19	0.440	60	0.791	110	0.904		
20	0.456	61	0.795	120	0.915		
21	0.471	62	0.799	130	0.923		
22	0.486	63	0.803	140	0.930		
23	0.500	64	0.807	150	0.935		
24	0.513	65	0.811	160	0.940		
25	0.526	66	0.814	170	0.944		
26	0.539	67	0.817	180	0.948		
27	0.551	68	0.821	190	0.951		
28	0.563	69	0.824	200	0.954		
29	0.574	70	0.827	210	0.957		
30	0.585	71	0.830	220	0.960		
31	0.596	72	0.833	230	0.963		
32	0.606	73	0.836	240	0.965		
33	0.616	74	0.839	250	0.968		
34	0.625	75	0.842	260	0.970		
35	0.635	76	0.845	270	0.973		
36	0.644	77	0.847	280	0.975		
37	0.652	78	0.850	290	0.977		
38	0.661	79	0.852	300	0.980		
39	0.669	80	0.855	310	0.982		
40	0.677	81	0.857	320	0.983		

APPENDIX C - EXAMPLE CYCLE-SPECIFIC UTILITY CALCULATION CODE

```
'''
Usage Instructions:
    1) Install python & scipy.
        The website for python is www.python.org
        The website for scipy is www.scipy.org
        A portable version of python & scipy able to run this script is available at:
            www.portablepython.com
        An installer for python and scipy able to run this script can be found at:
            www.pythonxy.com

    2) Install this file and daypub.csv in the same directory.

Key assumptions:
    1) If the 'TDAYDATE' data is not supplied in the form YYYYMMDD, then all
        travel for a given value of 'TRAVDAY' is assumed to take place on the
        SAME calendar date (e.g., as opposed to different Mondays that month).
    2) The first trip of a given calendar date starts with a full battery, and
        no other external charging occurs that day.

'''

import sys
import os
# import os.path
import datetime
# import numpy as np
import pylab as plt
import csv
from numpy import arange as arange
# import unittest


class Table(object):
    def __init__(self,filename,**kwargs):
        """
        To build the table from a CSV file pass the filename and optional parameters
        maxLines: maximum number of rows to keep from the file
        omitList: list of columns to omit from data set
        keepList: list of columns to keep
        """
        recsReader = csv.reader(open(filename))
        self._data = list()
        data = object.__getattribute__(self,'_data')
        for i,row in enumerate(recsReader):
            # stop after a maximum number of lines if specified
            if 'maxLines' in kwargs.keys():
                # stop after
                if i>kwargs['maxLines']:
                    break
            # add header or data rows
            if i==0:
                self._header = row
                header = row
                for i,item in enumerate(header):
                    header[i]=item.strip()
            else:
                # pad the row if it is shorter than the header
```

```
                if len(header)>len(row):
                    for ii in range(0,len(header)-len(row)):
                        row.append('')
                # eliminate data columns specified
                omitList = kwargs.get('omitList',[])
                for key in omitList:
                    row[header.index(key)]=None
                # eliminate data columns not requested
                keepList = kwargs.get('keepList',[])
                if len(keepList)>0:
                    for idx,key in enumerate(header):
                        if not key in keepList:
                            row[idx]=None
                # add row to stored data list
                data.append(row)

    def __getattribute__(self,name):
        if name=='row':
            pass
        elif name=='col':
            pass
        elif name=='data':
            return object.__getattribute__(self,'_data')
        elif name=='header':
            return object.__getattribute__(self,'_header')
        else:
            #print 'returning default object'
            return object.__getattribute__(self,name)

    def __getitem__(self,key):
        data = object.__getattribute__(self,'_data')
        header = object.__getattribute__(self,'_header')
        if type(key)==str:
            # return the column with this name
            idx = header.index(key)
            col = []
            for row in data:
                col.append(row[idx])
            return col
        elif type(key)==int:
            thisData = data[key]
            thisHeader = self._header
            #return dict(zip(thisHeader,thisData))
            return thisData
        else:
            #print 'returning default object'
            return object.__getattribute__(self,key)

    def simplify(self):
        """
        Eliminates all columns in the table which were not kept when read in
        """
        data = object.__getattribute__(self,'_data')
        header = object.__getattribute__(self,'_header')
        #print header
        deleteList = []
        for i,value in enumerate(data[0]):
            if value==None:
                deleteList.append(header[i])
        newHeader = []
```

```
        newData = []
        for row in data:
            newData.append([])
        for i,column in enumerate(header):
            if header[i] not in deleteList:
                newHeader.append(header[i])
                for row,newRow in zip(data,newData):
                    newRow.append(row[i])
        self._data   = newData
        self._header = newHeader

    def filter(self,filterList):
        data = object.__getattribute__(self,'_data')
        header = object.__getattribute__(self,'_header')
        newHeader = []
        newData = []
        for i,row in enumerate(data):
            testPassed=True
            for test in filterList:
                testFunction = test[1]
                testColumnName = test[0]
                testColumn = header.index(testColumnName)
                if not testFunction(row[testColumn]):
                    testPassed=False
                    break
            if testPassed:
                newData.append(row)
        self._data   = newData

def loadDataFile(filename,maxRows=1e8):
    keepList =
['HOUSEID','VEHUSED','TRAVDAY','TDAYDATE','DRVR_FLG','SMPLSRCE','TRPMILES','TRVL_MIN','VE
HTYPE']
    t = Table(filename,maxLines=maxRows,keepList=keepList)
    t.simplify()
    return t

def preprocessTable(table, verbose):

    '''
    This routine converts the table data into the lists needed by functions
    determineSplitSpeed and buildDayTravelDistList.

    vehUniqIDList = list of unique vehicle IDs
    tripSpeedList = list of speeds for every trip
    tripDistList = list of distances for every trip
    tripDayList = list of how many days each vehicle was driven
    '''

    ########################################
    # Filter the data to remove trips not applicable to UFs
    #   - trips are recorded by a driver
    #   - the sample source is the NHTS national sample
    #   - the trip has positive time and distance
    if verbose: print 'Number of rows before filtering = %f' % len(table.data)
    table.filter(
            [('VEHTYPE',lambda x:float(x)>=1 and float(x)<=4),
             ('SMPLSRCE',lambda x:float(x)==1),
             ('TRPMILES',lambda x:float(x)>0),
             ('TRVL_MIN',lambda x:float(x)>0),
```

```
                ('DRVR_FLG',lambda x:float(x)==1)]
              )
    if verbose: print 'Number of rows after filtering = %f' % len(table.data)

    ########################################
    # Extract columns of data from the table as lists.
    # Convert text from the table into numbers where needed
    header = table.header
    data = table.data

    vehUniqIDList = [hid+vid for hid,vid in zip(table['HOUSEID'],table['VEHUSED'])]
    tripDistList  = [float(x) for x in table['TRPMILES']]
    tripTimeList  = [float(x) for x in table['TRVL_MIN']]
    tripSpeedList = [min(70,float(d)/(float(t)/60.0)) for d,t in
zip(tripDistList,tripTimeList)]
    tripDayList   = [day for day in table['TRAVDAY']]
    tripDateList  = [date for date in table['TDAYDATE']]

    preppedTable = {
        'vehUniqIDList':vehUniqIDList,
        'tripDistList':tripDistList,
        'tripSpeedList':tripSpeedList,
        'tripDayList':tripDayList,
        'tripDateList':tripDateList
        }

    return preppedTable


def determineSplitSpeed(VMT_Fract,tripSpeedList,tripDistList,
        displayPlot=False,savePlot=False, verbose=False):

    # calculate cummulative distance versus sorted speed
    tripList = zip(tripSpeedList,tripDistList)
    tripList.sort()
    totalDist = 0.0
    cumDist = []
    for trip in tripList:
        totalDist += trip[1]
        trip = list(trip)
        cumDist.append(totalDist)

    if displayPlot or savePlot:
        x = cumDist
        y = [trip[0] for trip in tripList]

        ############################
        # plot trip speeds vs cumulative distance
        plt.figure(1)
        plt.clf()
        plt.plot(x,y)
        plt.grid(True)
        plt.title('Trip Speed vs Cum VMT')
        plt.xlabel('Cummulative Distance [miles]')
        plt.ylabel('Trip Speed [mph]')
        if savePlot: plt.savefig('TripSpeedVsVMT.png')

    ############################
    # calculate speed below which VMT_Fract of travel occurs
    maxCumDist = cumDist[-1]
```

```
    transCumDist = VMT_Fract*maxCumDist
    lowSpeed = [speed_dist[0] for idx,speed_dist in enumerate(tripList) if
cumDist[idx]<=transCumDist]

    try:
        transSpeed = lowSpeed[-1]
    except IndexError:
        # assume no trips occur before the transCumDist so lowSpeed==[]
        #    therefore, just grab the slowest trip's speed
        transSpeed = min(tripSpeedList)
    if verbose:
        print 'calculated transition speed = %f' % transSpeed
    return transSpeed

def buildDayTravelDistList(
        vehUniqIDList,tripDistList,tripSpeedList,tripDayList,tripDateList,
        transSpeed, verbose = False, debug = False):

    '''
    This routine converts lists of trips into lists of day travel.
    It assumes all travel on a given day (a unique TRAVDAY / TDAYDATE pair)
    starts with a fully battery and has no other external charging.  It cannot
    distinguish between different TRAVDAY for a given TDAYDATE unless the
    TDAYDATE data is of the format YYYYMMDD.

    vehUniqIDList   - a list that uniquely identifies the vehicle associated with each
trip
    tripDistList    - a list of the distance traveled in each trip
    tripSpeedList   - a list of the average speed of each trip
    tripDayList     - a list of the day index (1-7) for each trip
    transSpeed      - the speed above which a trip is assigned to hwy driving,
                      below which a trip is assigned to city driving.
    verbose         - a flag which indicates if status messages should be displayed
    debug           - a flag which indicates if debugging messages should be displayed

    This function returns a dictionary with the daily driving information. The
        dictionary contains the following information:

    dayVehIDList          - a list of vehID for each unique (day,vehID) combination
    dayTravelDistList     - a list of sum(dist) for each unique (day,vehID) combination
    dayHwyTravelDistList  - a list of sum(dist|v>splitSpeed) for each unique (day,vehID)
combination
    dayHwyWtList          - a list of hwy weighting for each unique (day,vehID)
combination
    '''

    # initialize some variables
    vehUniqIDSet = set(vehUniqIDList)
    dayVehIDList          = []
    dayTravelDistList     = []
    dayHwyTravelDistList  = []
    dayHwyWtList          = []
    if verbose:
        count = 0
        ctIncrement = int(max(1,len(vehUniqIDSet)/10))

    # this builds a quick index to use to find all trips
    #   associated with a unique vehicle
    thisIdxDict = {}
    for idx,veh in enumerate(vehUniqIDList):
```

```
        try:
            thisIdxDict[veh].append(idx)
        except KeyError:
            thisIdxDict[veh]=[idx]

    for veh in list(vehUniqIDSet):
        if verbose:
            # update about every 10% of completion
            if count.__divmod__(ctIncrement)[1]==0:
                print 'pct vehicles done = %f' % ((100.0*count)/len(vehUniqIDSet))
            count+=1
        idxList = thisIdxDict[veh]
        thisVehTripDistList  = [tripDistList[idx] for idx in idxList]
        thisVehTripSpeedList = [tripSpeedList[idx] for idx in idxList]
        thisVehTripDayList   = [tripDayList[idx] for idx in idxList]
        thisVehTripDateList  = [tripDateList[idx] for idx in idxList]
        thisVehTripDayAndDateList  = [tripDateList[idx]+tripDayList[idx] for idx in
idxList]

        # reduce each day to a single distance.
        # build new lists for these single day distances.
        # By concatenating day and date columns, we'll assume there is a unique
        # calendar day for each day/date combination, and sum trips over it.
        for tripDate in list(set(thisVehTripDayAndDateList)):
            idxList = [idx for idx,dummy in enumerate(thisVehTripDayAndDateList)
                           if thisVehTripDayAndDateList[idx]==tripDate]
            dayVehIDList.append(veh)
            dayTravelDistList.append(sum([thisVehTripDistList[idx] for idx in idxList]))
            dayHwyTravelDistList.append(sum([thisVehTripDistList[idx] for idx in idxList
                    if thisVehTripSpeedList[idx]>transSpeed]))
            dayHwyWtList.append(float(dayHwyTravelDistList[-1])/dayTravelDistList[-1])

    if verbose:
        print 'number samples in summary list = %i' % len(dayVehIDList)
    if debug:
        print 'dayVehIDList = ', dayVehIDList
        print 'dayTravelDistList = ', dayTravelDistList
        print 'dayHwyWtList = ', dayHwyWtList

    d = {
        'dayVehIDList':dayVehIDList,
        'dayTravelDistList':dayTravelDistList,
        'dayHwyWtList':dayHwyWtList,
        }

    return d

def generateUFs(Rcd,dayVehIDList,dayTravelDistList,dayHwyWtList,
                verbose=False, debug=False,savePlots=False,displayPlots=False):
    '''
    This routine converts lists of distances per day into Utility Factors.

    Rcd               - a list of charge decreasing ranges over which to compute UF
values
    dayVehIDList      - a list of vehID for each unique (day,vehID) combination
    dayTravelDistList - a list of sum(dist) for each unique (day,vehID) combination
    dayHwyWtList      - a list of hwy weighting for each unique (day,vehID) combination
    verbose           - a flag which indicates if status messages should be displayed
    debug             - a flag which indicates if debugging messages should be displayed
    savePlots         - a flag to indicate whether or not to save the UF plots to file
```

```
    displayPlots      - a flag to indicate whether or not to display the UF plots

    This function returns a dictionary with the UF calculations. The
        dictionary contains the following information:

    Rcd      - a list of charge decreasing ranges over which to compute UF values
    UF       - a list of Fleet Utility Factors
    IUF      - a list of Individual Utility Factors
    UF_city  - a list of city-specific Fleet Utility Factors
    UF_hwy   - a list of highway-specific Fleet Utility Factors
    IUF_city - a list of city-specific Individual Utility Factors
    IUF_hwy  - a list of highway-specific Individual Utility Factors
    '''
    if verbose:
        print 'Generating UFs'
        count    = 0
        ctIncrement = int(max(1,len(Rcd)/10))

    # initialize some variables
    vehList  = list(set(dayVehIDList))
    vehNumDays      = []
    vehSumD         = []
    vehSumHwyWtD    = []
    vehSumCityWtD   = []
    vehHwyFraction  = []
    vehCityFraction = []
    vehOnesVector = [1 for x in vehList]
    IUF     = []
    cityIUF = []
    hwyIUF  = []
    FUF     = []
    cityFUF = []
    hwyFUF  = []

    # this builds a quick index to use to find all days
    #   associated with a unique vehicle
    idxDict={}
    for idx,veh in enumerate(dayVehIDList):
        try:
            idxDict[veh].append(idx)
        except KeyError:
            idxDict[veh]=[idx]

    # for each vehicle, compute sum(wt*d) over all days
    for veh in vehList:
        idxList = idxDict[veh]
        vehNumDays.append(len(idxList))
        vehSumD.append(sum([dayTravelDistList[idx] for idx in idxList]))
        vehSumHwyWtD.append(sum([float(dayHwyWtList[idx])*dayTravelDistList[idx] for idx
in idxList]))
        vehSumCityWtD.append(sum([float((1.0-dayHwyWtList[idx]))*dayTravelDistList[idx]
for idx in idxList]))
        vehHwyFraction.append(float(vehSumHwyWtD[-1])/float(vehSumD[-1]))
        vehCityFraction.append(float(vehSumCityWtD[-1])/float(vehSumD[-1]))
    if debug:
        print 'vehNumDays = ', vehNumDays
        print 'vehSumD = ', vehSumD
        print 'vehSumHwyWtD = ', vehSumHwyWtD
        print 'vehSumCityWtD = ', vehSumCityWtD
```

```
    for r in Rcd:

        if verbose:
            # update about every 10% of completion
            if count.__divmod__(ctIncrement)[1]==0:
                print 'calculating UF for Rcd = %5.2f. (%3.1f pct complete) @ %s' %
(r,(100.0*count)/len(Rcd),str(datetime.datetime.now()))
            count+=1

        vehSumWtDcd     = []
        vehSumHwyWtDcd  = []
        vehSumCityWtDcd = []

        # for each vehicle, compute sum(wt*dcd) over all days for a given Rcd
        for veh in vehList:
            idxList = idxDict[veh]
            vehSumWtDcd.append(    sum([min(r,dayTravelDistList[idx]) for idx in
idxList]))
            vehSumHwyWtDcd.append( sum([min(r,dayTravelDistList[idx])*(dayHwyWtList[idx])
for idx in idxList]))
            vehSumCityWtDcd.append(sum([min(r,dayTravelDistList[idx])*(1.0-
dayHwyWtList[idx]) for idx in idxList]))

        # if debug:
        #     print 'vehSumWtDcd = ', vehSumWtDcd
        #     print 'vehSumHwyWtDcd = ', vehSumHwyWtDcd
        #     print 'vehSumCityWtDcd = ', vehSumCityWtDcd

        # finish calculations for cycle-specific UF (new equation)
        IUF.append(    calcIUF(vehSumD,       vehSumWtDcd,     vehOnesVector))
        hwyIUF.append( calcIUF(vehSumHwyWtD,  vehSumHwyWtDcd,  vehHwyFraction))
        cityIUF.append(calcIUF(vehSumCityWtD, vehSumCityWtDcd, vehCityFraction))
        # # finish calculations for cycle-specific UF (original equation)
        # IUF.append(    calcIUF(vehSumD,     vehSumWtDcd))
        # hwyIUF.append( calcIUF(vehSumHwyWtD,  vehSumHwyWtDcd))
        # cityIUF.append(calcIUF(vehSumCityWtD, vehSumCityWtDcd))

        # finish calculations for fleet UF
        FUF.append(    calcFUF(vehSumD,       vehSumWtDcd,     vehNumDays))
        hwyFUF.append( calcFUF(vehSumHwyWtD,  vehSumHwyWtDcd,  vehNumDays))
        cityFUF.append(calcFUF(vehSumCityWtD, vehSumCityWtDcd, vehNumDays))


    UFDict = {'Rcd':Rcd, 'UF':FUF,'IUF':IUF,
              'UF_city':cityFUF, 'UF_hwy':hwyFUF,
              'IUF_city':cityIUF, 'IUF_hwy':hwyIUF}

    # generate plots, if requested
    if savePlots or displayPlots:
        plt.figure(2)
        plt.clf()
        plt.plot(Rcd,zip(IUF,cityIUF,hwyIUF),lw=2)
        plt.legend(['IUF','IUF city','IUF hwy'],loc='lower right')
        plt.grid(True)
        if savePlots: plt.savefig('IUF.png')

        plt.figure(3)
        plt.clf()
        plt.plot(Rcd,zip(FUF,cityFUF,hwyFUF),lw=2)
        plt.legend(['UF','UF city','UF hwy'],loc='lower right')
```

```
        plt.grid(True)
        if savePlots: plt.savefig('FUF.png')

    return UFDict

def calcIUFOrig(vehSumWtD,vehSumWtDcd):
    '''
    This function calculates the Individual Utility Factor, IUF, for a list of
       vehicles. The IUF determines the average fraction of miles
       traveled electrically by a sample of vehicles.

    vehSumWtD   - a list of sum(wt*d) over all days (1 element per vehicle)
    vehSumWtDcd - a list of sum(wt*dcd) over all days (1 element per vehicle)
    '''
    mySum = 0.0
    numVeh = 0.0
    for wD,wDcd in zip(vehSumWtD,vehSumWtDcd):
        try:
            delSum = float(wDcd)/float(wD)
            numVeh += 1
        except ZeroDivisionError:
            # if wD=0, then this vehicle is not included in the summation
            delSum = 0.0
        mySum += delSum
    return mySum/numVeh

def calcIUF(vehSumWtD,vehSumWtDcd,vehCycleFraction):
    '''
    This function calculates the Individual Utility Factor, IUF, for a list of
       vehicles. The IUF determines the average fraction of miles
       traveled electrically by a sample of vehicles.

    vehSumWtD   - a list of sum(wt*d) over all days (1 element per vehicle)
    vehSumWtDcd - a list of sum(wt*dcd) over all days (1 element per vehicle)
    vehCycleFraction - a list of sum(wt*d)/sum(d) over all days (1 element per vehicle)
    '''
    myNumSum = 0.0
    myDenSum = 0.0
    for wD,wDcd,cf in zip(vehSumWtD,vehSumWtDcd,vehCycleFraction):
        try:
            numSum = float(cf)*float(wDcd)/float(wD)
            denSum = float(cf)
        except ZeroDivisionError:
            # if wD=0, then this vehicle is not included in the summation
            numSum = 0.0
            denSum = 0.0
        myNumSum += numSum
        myDenSum += denSum
    return myNumSum/myDenSum

def calcFUF(vehSumWtD,vehSumWtDcd,vehNumDays):
    '''
    This function calculates the Fleet Utility Factor, FUF, for a list of
       vehicles. The FUF determines the fraction of miles
        traveled electrically by a sample of vehicles.

    vehSumWtD   - a list of sum(wt*d) over all days (1 element per vehicle)
    vehSumWtDcd - a list of sum(wt*dcd) over all days (1 element per vehicle)
    vehNumDays  - a list of the number of days each vehicle is driven
    '''
```

```
    myNumSum = 0.0
    myDenSum = 0.0
    for wD,wDcd,N in zip(vehSumWtD,vehSumWtDcd,vehNumDays):
        try:
            numSum = float(wDcd)/float(N)
            denSum = float(wD)/float(N)
        except ZeroDivisionError:
            # if N=0, then this vehicle is not included in the summation
            numSum = 0.0
            denSum = 0.0
        myNumSum += numSum
        myDenSum += denSum
    return myNumSum/myDenSum

###############################################################################

def reportEqResult(testID,phaseID,this,expected,passMsg=None,failMsg=None,epsilon=0):
    if abs(this-expected)<=epsilon:
        #print (('Test %s: Phase %s: Pass: %s') % (testID,phaseID,passMsg))
        #print  '                            Expected result = %f.' % expected
        return 0
    else:
        print (('Test %s: Phase %s: Fail: %s') % (testID,phaseID,failMsg))

        print  '                           Result should be %f, %f was computed' %
(expected,this)
        return 1


def doValidationTests():


    print '----------- start of validation tests --------------------'
    print ''
    print 'Performing validation tests:'

    notice = '''

    The tests that follow are not complete or exhaustive. They are designed
    to catch significant errors introduced by incorrect datasets or changes
    in the scripts.

    '''
    print notice

    ###########################################################################
    # test:
    #   If daypub.csv is in current working directory, load the file and test
    #   for number of lines, number of households, number of vehicles.
    #   If these numbers are different then it indicates that the
    #   the data set is different than the 2001 NHTS data this code was tested
    #   on.
    print 'Test 01: If using 2001 NHTS data, these tests check data characteristics.'
    print '         Failures on these tests indicates either a problem'
    print '         with script changes, corrupted data sets, or'
    print '         the use of data other than the 2001 NHTS'
    print '         See http://nhts.ornl.gov/'
    print ''
```

```
    failCount = 0

    # phase 01 - Test for existence of file
    if os.path.isfile('daypub.csv'):

        # phase 02 - Test number of rows read from the file
        print 'Test 01: Phase 01: Pass: daypub.csv in working directory'
        table = loadDataFile(dataFileName)
        numDataRows = len(table.data)
        if numDataRows==642292:
            print 'Test 01: Phase 02: Pass: correct number of data rows'
        else:
            print 'Test 01: Phase 02: Fail: incorrect number of data rows'
            print '                       : - check for truncated file'
            print '                       : - table class in scrfipt may have changed'
            failCount += 1

        # phase 03 - Test number of columns from the file
        if len(table.header)==8:
            print 'Test 01: Phase 03: Pass: correct number of data columns retained from
file'
        else:
            print 'Test 01: Phase 03: Fail: incorrect number of data columns'
            print '                       : - check for edited file'
            print '                       : - table class in script may have changed'
            failCount += 1

        # phase 04 - Test presence of minimally necessary columns
        neededColumns = ['HOUSEID','VEHUSED','TRAVDAY','DRVR_FLG',
                         'SMPLSRCE','TRPMILES','TRVL_MIN','VEHTYPE']
        missingColumns = []
        for header in neededColumns:
            if not header in table.header:
                missingColumns.append(header)
        if len(missingColumns)==0:
            print 'Test 01: Phase 04: Pass: required columns present'
        else:
            s = ''
            for count,header in enumerate(missingColumns):
                if count==0:
                    s = header
                else:
                    s += header
            print 'Test 01: Phase 04: Fail: following columns are missing: %s' % s
            print '                       : - check for edited file'
            print '                       : - table class in script may have changed'
            failCount += 1


        # phase 05 - Test number of rows eliminated by filtering
        initialRows = len(table.data)
        table.filter(
                [('VEHTYPE',lambda x:float(x)>=1 and float(x)<=4),
                 ('SMPLSRCE',lambda x:float(x)==1),
                 ('TRPMILES',lambda x:float(x)>0),
                 ('TRVL_MIN',lambda x:float(x)>0),
                 ('DRVR_FLG',lambda x:float(x)==1)]
                )
        finalRows = len(table.data)
        if initialRows-finalRows==642292-141538:
```

```
            print 'Test 01: Phase 05: Pass: Filtered number of rows is correct'
        else:
            if finalRows>141538:
                print 'Test 01: Phase 05: Fail: too many data rows passed filtering: %i'
% finalRows
            else:
                print 'Test 01: Phase 05: Fail: too few data rows passed filtering: %i' %
finalRows
            print '                          : - check for corrupted data'
            print '                          : - table class in script may have changed'
            failCount += 1

        # phase 06 - Check name of file to validate
        if dataFileName == 'daypub.csv':
            print 'Test 01: Phase 06: Pass: dataFileName is ''daypub.csv'''
        else:
            print 'Test 01: Phase 06: Fail: dataFileName is not ''daypub.csv'''
            failCount += 1
    else:
        print ('Test 01: Undefined: 2001 NHTS data file not present in ' +
                'current working directory or has unexpected name.')
    # recover memory used by the table
    # del table

    ###############################################################################
    # test 02:
    #   Fleet Utility Factors for a single vehicle with a single day data is created.
    print 'Test 02: Validating Fleet Utility Factor calculations'
    test = '02'
    Rcds = [0,10,20,30,40,50,100,200,400]
    # note: the UF calculation is not expected to be robust to 0 distance
    drivingRanges = [1,10,20,30,40,50,100,200,400]
    count = 1
    for Rcd in Rcds:
        for dist in drivingRanges:
            phase = '%02i' % count
            passMsg = 'Properly calculated Fleet UF for single vehicle driving %i miles
with %i miles of Rcd' % (dist,Rcd)
            failMsg = 'Improperly calculated Fleet UF for single vehicle driving %i miles
with %i miles of Rcd' % (dist,Rcd)

            fueledDist = max(0,dist-Rcd)
            thisUF = calcFUF([dist],[fueledDist],[1])
            expectedUF = 1-max(0,dist-Rcd)/float(dist)
            failCount += reportEqResult(test,phase,thisUF,expectedUF,
                        passMsg = passMsg, failMsg = failMsg, epsilon=1e-6)
            count += 1

    ###############################################################################
    # test 03:
    #   Fleet Utility Factors for a multiple vehicles with same daily distance.
    print 'Test 03: Validating Fleet Utility Factor calculations'
    test = '03'
    Rcds = [0,10,20,30,40,50,100,200,400]
    # note: the UF calculation is not expected to be robust to 0 distance
    drivingRanges = [1,10,20,30,40,50,100,200,400]
    count = 1
    for Rcd in Rcds:
        for dist in drivingRanges:
            phase = '%02i' % count
```

```
            passMsg = 'Properly calculated Fleet UF for multiple vehicles driving %i
miles with %i miles of Rcd' % (dist,Rcd)
            failMsg = 'Improperly calculated Fleet UF for multiple vehicles driving %i
miles with %i miles of Rcd' % (dist,Rcd)

            fueledDist = max(0,dist-Rcd)
            fueledDistList = [fueledDist for i in range(0,10)]
            wtList = [ 1 for x in fueledDistList]
            distList = [dist for i in range(0,10)]
            thisUF = calcFUF(distList,fueledDistList,wtList)
            expectedUF = 1-max(0,dist-Rcd)/float(dist)
            failCount += reportEqResult(test,phase,thisUF,expectedUF,
                        passMsg = passMsg, failMsg = failMsg, epsilon=1e-6)
            count += 1

    ##########################################################################
    # test 04:
    #   Fleet Utility Factors for a multiple vehicles with different daily distances
    print 'Test 04: Validating Fleet Utility Factor calculations'
    test = '04'
    Rcds = [0,10,20,30,40,50,100,200,400]
    # note: the UF calculation is not expected to be robust to 0 distance
    drivingRanges = [1,10,20,30,40,50,100,200,400]
    count = 1
    for Rcd in Rcds:
        for i in range(1,len(drivingRanges)):
            phase = '%02i' % count

            fueledDistList = [max(0,drivingRanges[j]-Rcd) for j in range(0,i)]
            distList = [drivingRanges[j] for j in range(0,i)]
            wtList = [ 1 for x in fueledDistList]
            thisUF = calcFUF(distList,fueledDistList,wtList)

            expectedUF = 1-sum(fueledDistList)/float(sum(distList))

            distStr = '['+''.join(['%f,' % d for d in distList])+']'
            passMsg = 'Properly calculated Fleet UF for multiple vehicles driving %s
miles with %i miles of Rcd' % (distStr,Rcd)
            failMsg = 'Improperly calculated Fleet UF for multiple vehicles driving %s
miles with %i miles of Rcd' % (distStr,Rcd)
            failCount += reportEqResult(test,phase,thisUF,expectedUF,
                        passMsg = passMsg, failMsg = failMsg, epsilon=1e-6)
            count += 1

    ##########################################################################
    # test 05:
    #   Individual Utility Factors for a single vehicle with a single day data is
created.
    print 'Test 05: Validating Individual Utility Factor calculations'
    test = '05'
    Rcds = [0,10,20,30,40,50,100,200,400]
    # note: the UF calculation is not expected to be robust to 0 distance
    drivingRanges = [1,10,20,30,40,50,100,200,400]
    count = 1
    for Rcd in Rcds:
        for dist in drivingRanges:
            phase = '%02i' % count
            passMsg = 'Properly calculated Individual UF for single vehicle driving %i
miles with %i miles of Rcd' % (dist,Rcd)
            failMsg = 'Improperly calculated Individual UF for single vehicle driving %i
```

```
miles with %i miles of Rcd' % (dist,Rcd)

            fueledDist = max(0,dist-Rcd)
            thisUF = calcIUF([dist],[fueledDist],[1])
            expectedUF = 1-max(0,dist-Rcd)/float(dist)
            failCount += reportEqResult(test,phase,thisUF,expectedUF,
                        passMsg = passMsg, failMsg = failMsg, epsilon=1e-6)
            count += 1

    ##########################################################################
    # test 06:
    #   Individual Utility Factors for a multiple vehicles with same daily distance.
    print 'Test 06: Validating Fleet Utility Factor calculations'
    test = '06'
    Rcds = [0,10,20,30,40,50,100,200,400]
    # note: the UF calculation is not expected to be robust to 0 distance
    drivingRanges = [1,10,20,30,40,50,100,200,400]
    count = 1
    for Rcd in Rcds:
        for dist in drivingRanges:
            phase = '%02i' % count
            passMsg = 'Properly calculated Fleet UF for multiple vehicles driving %i
miles with %i miles of Rcd' % (dist,Rcd)
            failMsg = 'Improperly calculated Fleet UF for multiple vehicles driving %i
miles with %i miles of Rcd' % (dist,Rcd)

            fueledDist = max(0,dist-Rcd)
            fueledDistList = [fueledDist for i in range(0,10)]
            distList = [dist for i in range(0,10)]
            wtList = [ 1 for x in fueledDistList]
            thisUF = calcIUF(distList,fueledDistList,wtList)
            expectedUF = 1-max(0,dist-Rcd)/float(dist)
            failCount += reportEqResult(test,phase,thisUF,expectedUF,
                        passMsg = passMsg, failMsg = failMsg, epsilon=1e-6)
            count += 1

    ##########################################################################
    # test 07:
    #   Individual Utility Factors for a multiple vehicles with different daily distances
    print 'Test 07: Validating Fleet Utility Factor calculations'
    test = '07'
    Rcds = [0,10,20,30,40,50,100,200,400]
    # note: the UF calculation is not expected to be robust to 0 distance
    drivingRanges = [1,10,20,30,40,50,100,200,400]
    count = 1
    for Rcd in Rcds:
        for i in range(1,len(drivingRanges)):
            phase = '%02i' % count

            fueledDistList = [max(0,drivingRanges[j]-Rcd) for j in range(0,i)]
            distList = [drivingRanges[j] for j in range(0,i)]
            wtList = [ 1 for x in fueledDistList]
            thisUF = calcIUF(distList,fueledDistList,wtList)

            expectedUF = (1.0/len(fueledDistList))*(sum([1-f/float(d) for f,d in
zip(fueledDistList,distList)]))

            distStr = '['+''.join(['%f,' % d for d in distList])+']'
            passMsg = 'Properly calculated Fleet UF for multiple vehicles driving %s
miles with %i miles of Rcd' % (distStr,Rcd)
```

```
            failMsg = 'Improperly calculated Fleet UF for multiple vehicles driving %s
miles with %i miles of Rcd' % (distStr,Rcd)
            failCount += reportEqResult(test,phase,thisUF,expectedUF,
                        passMsg = passMsg, failMsg = failMsg, epsilon=1e-6)
            count += 1

    ##########################################################################
    # test 08:
    #   Validate the speed split function
    print 'Test 08: Validating logic for selecting the city/hwy split speed'
    test = '08'
    tripDists  = [1 for idx in range(0,100)]
    tripSpeeds = [s for s in range(1,101)]
    VMT_Fracts = arange(0,1,0.1)
    count = 1
    for fract in VMT_Fracts:
            phase = '%02i' % count
            thisSplit= determineSplitSpeed(fract,tripSpeeds,tripDists)
            if fract==0:
                # handle case where the are no trips below VMT fraction
                expectedSplit=1
            else:
                expectedSplit =fract*100
            passMsg = 'Properly calculated split speed for 100 equal trips with speeds
between 1 and 100 for %f city VMT fraction' % fract
            failMsg = 'Improperly calculated split speed for 100 equal trips with speeds
between 1 and 100 for %f city VMT fraction' % fract
            failCount += reportEqResult(test,phase,thisSplit,expectedSplit,
                        passMsg = passMsg, failMsg = failMsg, epsilon=1e-6)
            count += 1

    ##########################################################################

    ##########################################################################
    # test 09:
    #   Validate the preprocessTable function
    print 'Test 09: Validating table preprocessing function'
    test = '09'

    data = loadDataFile('DummyData.csv')
    vehID = [('100000001', ' 1'), ('100000001', ' 1'), ('100000001', ' 2')]
    dist = [10.0, 10.0, 15.0]
    time = [10.0, 20.0, 20.0]
    v = [float(d)/(float(t)/60) for d,t in zip(dist,time)]
    t = preprocessTable(data,True)

    for idx in arange(0,len(vehID)):
        phase = '%02i' % count
        passMsg = 'Properly calculated vehUniqIDList in preprocessTable function'
        failMsg = 'Improperly calculated vehUniqIDList in preprocessTable function'
        #failCount += reportEqResult(test,phase,t['vehUniqIDList'][idx],vehID[idx],
        #        passMsg = passMsg, failMsg = failMsg, epsilon=1e-6)
        count += 1

    ##########################################################################
    # test 10:
    #   Validate the buildDayTravelDistList function
    print 'Test 10: Validating buildDayTravelDistList function'
    test = '10'
```