Optimization of a hybrid electric vehicle energy management control parameters by metaheuristic method

Team E:

Russ Campbell

Carl Edwards

Samira Shamsir

Submitted: May 3, 2018

1. Introduction

Hybrid electric vehicles (HEV) offer benefits of low fuel consumption and reduced emissions when compared to conventional vehicles. With these benefits comes the burden of complexity in blending the operation of electric and conventional powertrain components to maximize efficiency while still meeting vehicle performance requirements. In this project an objective function will be utilized to attempt to minimize fuel consumption, and subsequently exhaust emissions, through application of metaheuristic optimization methods.

Optimization algorithms like particle swarm and genetic algorithms can be used to search for solutions to multi-objective problems. Genetic algorithms work based on treating problem parameters like genes in natural selection, while particle swarm moves a number of "particles" around the problem's solution space by assigning velocity based on discovered best values. These are both metaheuristic approaches which allow them to search large solution spaces given few assumptions. Simulation results will relate the effectiveness of the control strategy optimization and the practicality of the approach.

2. Problem Description

The fuel economy, emissions, and dynamic performance of a HEV depend significantly on the control strategy of the powertrain components, which is a complex system integrating the operation of mechanical, electrical, chemical, and thermodynamic devices. Thus, optimization of the operation of this system through refinement of control strategy parameters is required to obtain an efficient HEV design [1]. Thus, the selection of an optimization algorithm may have a significant impact on the performance and efficiency of the vehicle.

The implemented control strategy for the parallel HEV architecture, depicted in Figure 1, operates in such a way that the internal combustion engine (ICE) works as the main source of power and supplies the majority of the driver demanded power, while the electric motor is used to supply dynamic or peak power when requested by the driver. Charge sustaining operation whereby the state-of-charge (SOC) of the battery is maintained during all operating conditions functions based on the following rules:

(1) The electric motor is used for power assist, if the driver demanded power is greater than the

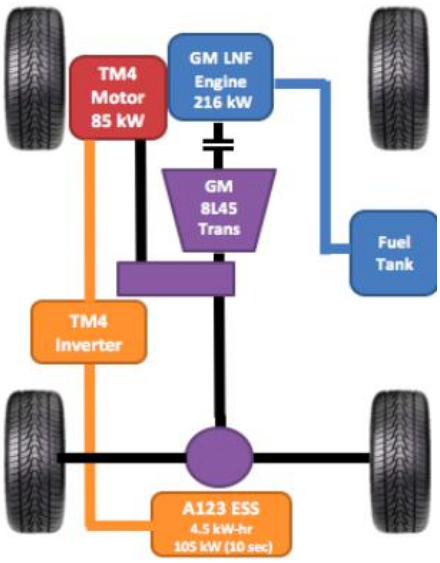


Figure 1: Parallel HEV Architecture

maximum ICE power, at a given ICE operating speed;

(2) The electric motor charges the battery during regenerative braking events;

(3) When battery SOC is lower than the set minimum value, the ICE produces extra power, to sustain battery SOC.

There are several current research works that employ traditional approaches to optimize the parameters of HEV control strategies. However, the requirement of numerous assumptions in the objective function including continuity, differentiability, and satisfaction of the Lipschitz condition makes these methods unsuitable [3]. In addition, usually there are many local minima in the multi-modal response function of the parallel HEV. Most of the local optimizers involve gradient based algorithms such as sequential quadratic programming (SQP) use the derivative information to find the local minima and they do not search the entire design space to find the global minimum [4]. Hence, derivative-free algorithms, such as genetic algorithm (GA) and particle swarm optimization (PSO) are more suitable to address the issues of a noisy and discontinuous objective function like the HEV drivetrain [5]. These derivative-free methods usually sample a large portion of the design space to find a global solution. Population based stochastic optimization methods such as the GA and PSO are initialized with a population of random solutions and the optimum is determined by updating generations. PSO has some advantages over GA such as it has no evolution operators like crossover and mutation [6]. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles.

In this project, optimization of the HEV control system parameters to improve fuel economy will be evaluated by implementing particle swarm optimization (PSO) and genetic algorithm (GA) optimization. These algorithms will consider drivetrain components and control strategy parameters as design objectives, and vehicle performance parameters as constraints.

3. Methodology

The HEV will be modeled utilizing a software in the loop (SIL) model created using Matlab. The SIL environment uses soft electronic control units (soft-ECUs) to simulate the controls logic which interacts with plant models representing the major vehicle powertrain systems, driver, and environment. The focus of this project will be on optimizing energy management control parameters which are responsible for conveying the maximum available power from the battery, maintaining SOC, and ensuring safe vehicle operation.

As outlined in the problem description maintaining charge sustaining operation is accomplished through regenerative braking or by utilizing power from the engine through cruise charging. The functionality of this strategy relies on the effective selection of several parameters including the target SOC, maximum electrical accessory power, power required to maintain SOC (Psoc) offset, and maximum cruise charge (CC) power. These values must be derived from modeling and real world testing to enable charge sustaining operation under all conditions. This compromise creates an opportunity to optimize the parameters based on the current vehicle operating conditions to reduce the amount of fuel used and improve vehicle performance.

Within the optimization algorithm simulation iterations the control input values will be evaluated using metaheuristic techniques to maximize fuel efficiency. Contributing operating parameters are listed below in Table 1, and the function used to evaluate fuel efficiency, Fc, is defined as Equation 1.

Where r = the reference input, u = the control input, and y represents the model outputs.

$$Fc(\,r,u\,) = \frac{\int_0^T |y1(t)|\ dt}{\int_0^T |y2(t)|\ dt} \qquad \text{Equation 1}$$

Table 1: Reference Input, control input, and output of the system

Variable	Description
r1	Target vehicle speed (km/h)
u1	Target SOC (%)
u2	AccELec maximum power (W)
u3	Psoc offset (%)
u4	Cruise Charge maximum power (W)
y1	Actual vehicle speed (km/h)
y2	Instantaneous fuel consumption (L)

The particle swarm optimization algorithm is governed by the following equations and variables. There are N particles each with A parameters. The position of these particles, p, is an $A\ x\ N$ matrix. We represent the velocity, v, of all the particles with an $A\ x\ N$ matrix as well. There are four empirically determined weights, $w_1 = 0.3$, $w_2 = 0.5$, $w_3 = 0.4$, and $w_4 = 1.5$. The first weight applies to inertia, the second to the global best difference, the third to the personal best difference, and the fourth to a randomized value. The algorithm also uses the global best solution, $gbest$, and personal best, $pbest$, parameters of each particle. $pbest$ is an $A\ x\ N$ matrix of all the particles best solutions, and $gbest$ is an $A\ x\ 1$ matrix as it only

represents the best solution for one particle. The algorithm runs through t_{end} iterations where t is the current iteration. It also utilizes an $A\ x\ 2$ matrix, $ranges$, which represent the potential range of each parameter. Utilizing a random number R uniformly chosen in $[0,1]$, we calculate the starting population randomly.

$$p_{i,j} = R * (ranges_{j,2} - ranges_{j,1}) + ranges_{j,1} \quad \text{Equation 1}$$

A detailed description of the optimization algorithm follows.
Begin iteration one ($t = 1$):
For each iteration, evaluate the model using given parameters for each particle j (column in p). If the solution for the particle is better than its personal best (which we remembered for efficiency), we will store that column in the appropriate column j of $pbest$. We will calculate velocity using the following formula:

$$gbestm = [gbest \mid gbest \mid \ldots \mid gbest] \quad \text{Equation 3}$$
($gbest$ augmented with itself N times)

Let Rm be a matrix $A\ x\ N$ which is comprised of random numbers [0,1].

$$v = w_1 * v + w_2 * (pbest - p) + w_3 * (gbestm - p) + w_4 * (Rm * 2 - 1) \quad \text{Equation 4}$$

Next, apply the velocity to p:

$$p = p + v; \quad \text{Equation 5}$$

Apply boundaries:

$$pij = \begin{cases} ranges_{j,1} & if\ p_{i,j} < ranges_{j,1} \\ ranges_{j,2} & if\ p_{i,j} > ranges_{j,2} \\ p_{i,j} & otherwise \end{cases} \quad \text{Equation 6}$$

Begin the next iteration until t_{end} iterations have occurred. Increment t and repeat the same process if $t \leq t_{end}$.

The genetic algorithm utilizes the mechanisms of mutation and crossover, using methodology from Ref. [7] with significant modification to fit the problem. It iterates for G generations with a empirical mutation rate $P_m = 0.033$ and a crossover rate $P_c = 0.6$. The population is created randomly from within given ranges, just like the PSO. Each individual in the algorithm has L genes, each of which is a floating point number.

The algorithm begins by computing the fitness, or value of the objective function, for all N individuals in the population. N is constrained $N\ mod\ 2 = 0$. Fitness is recorded in a vector of

length N. Evaluating fitness can be parallelized to improve algorithm speed. Following the fitness evaluation, we then check the population for a new best fitness, and record population statistics.

Next, the subsequent population of the algorithm is created. We start by creating a new vector for fitness by normalizing it with respect to its sum:

$$Normalized\ Fitness\ = \frac{Fitness}{\sum Fitness} \quad \text{Equation 7}$$

$$\text{Thus,} \quad \sum Normalized\ Fitness = 1 \quad \text{Equation 8}$$

Next, we create a running average from these values. This is a vector of size N as well. The value of element i in this vector is equivalent to the sum of the previous elements plus the i^{th} element of $Normalized\ Fitness$. Thus, the last value of the running average is 1.

$$Running\ Average_i = \sum_{n=1}^{i} Normalized\ Fitness_n \quad \text{Equation 9}$$

Next, two random numbers, i_1 and i_2 are selected to enable selection of parents for the algorithm based on the running average. We select the parents based on the running average. The parent 1 or 2 selected is the highest index n where $Running\ Average_n < i_{1\ or\ 2}$. Higher fitness parents are more likely to be selected. If the second parent is identical to the first, we use a heuristic of selecting the next individual in case there is only one high fitness member of the population.

To increase the effectiveness of this procedure, we shift the fitness value down by 23 (determined empirically) to cause higher fitness parents to have an increased chance of procreation. This is because the approach used for selecting parents involves normalizing the values. All fitness values remain greater than zero.

We create two offspring o_1 and o_2 as copies of each parent individual. Next, we create a random number within $[0,1]$. If this random number is less than the probability of crossover, P_c, then we uniformly select a random integer $a = [1, L]$. We use this as a single inflection point and switch all genes (parameter values) in o_1 and o_2 after (and including) this point a.

Subsequently, the algorithm loops through all genes individually in both children. We mutate each gene to a random number in the correct range with a probability P_m. We add the two

children to our population and repeat the process until the new population is size N. We repeat this procedure for G iterations.

In addition, we also implemented elitism into a version of the genetic algorithm. This implementation involves copying a small proportion of the fittest candidates, unchanged, into the next generation. This was done with two elites, so then only $\frac{N}{2} - 2$ children are created. The two elites join the new population without any changes. This will hopefully prevent the population from regressing to lower fitness levels while still retaining benefits of the algorithm.

4. Results and Discussion

For this project a modified SIL model was developed which allows the optimization function to update parameters, shown in Table 1, programmatically. Real world driving data, ~1600 miles, was utilized to validate the software in the loop (SIL) model against the real world vehicle performance. At each iteration the fuel economy and electrical energy consumption of the vehicle is evaluated.

Table 2: Control Input parameters for optimizing Vehicle Fuel Economy			Min	Max
b1	Target SOC (%)	Target_SOC	40	80
b2	Accessory Electrical maximum power (W)	AccElec	300	900
b3	Psoc offset	Psoc_Offset	1	20
b4	Cruise Charge Max Power (W)	CC_Upper	5000	15000

To evaluate performance of the vehicle and optimization method simulation has been conducted over a standard set of drive cycles, four EPA US06 or Supplemental Federal Test Procedure drive cycles as shown in Figure 2, for the baseline and optimized parameter values resulting from the optimization methods. This set of cycles represents 40 minutes and 32 miles of driving with an average speed of just over 48 miles per hour (MPH).

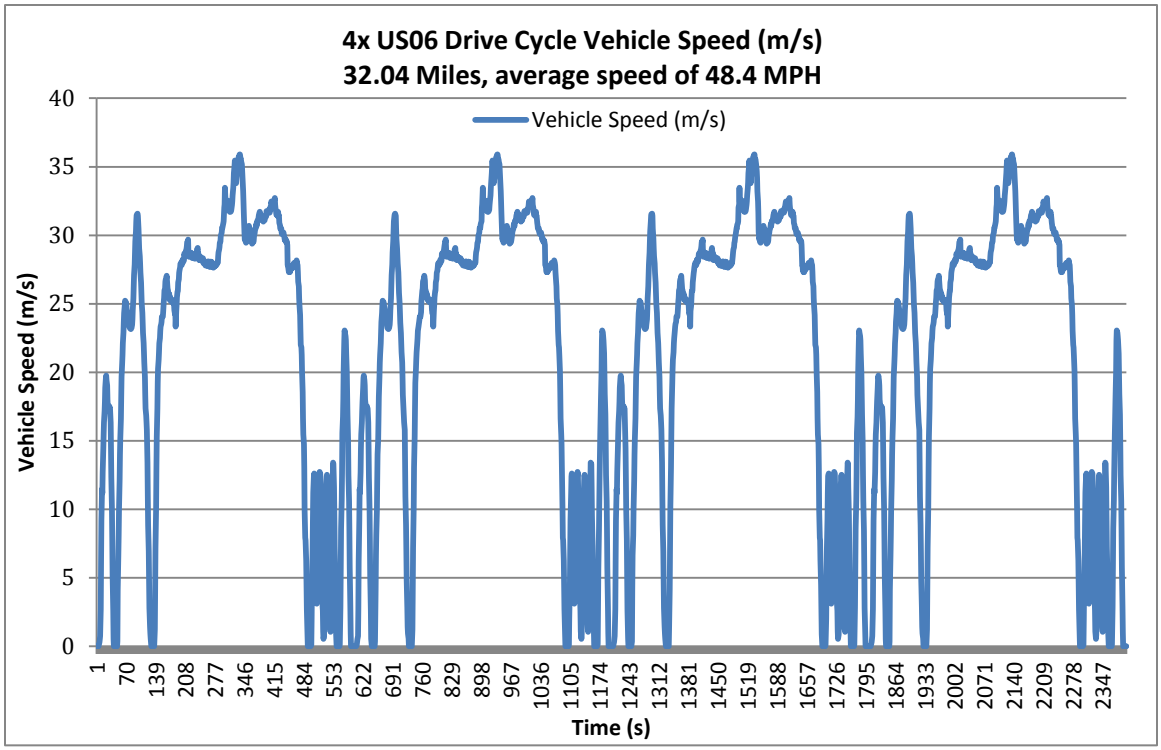


Figure 2: Four US06 Drive Cycles, Vehicle Speed vs. Time

Baseline parameter values were identified through trial and error controls prototyping, and adjusted based on real-world driving. The baseline model will be run over 10 sets of cycles representing 320 miles and over 6 hours of driving, resulting in a baseline fuel economy of 32.58 miles per gallon (MPG). The results from the 10 simulations will then be used to compare against the "optimal" solution generated by the algorithm.

4.1 Particle Swarm Optimization Results

To investigate the efficacy of the PSO algorithm simulations were run with varying numbers of particles, from ten to one hundred in increments of ten. The results of all PSO simulations are summarized in Table 3, and Figure 2 depicts the 20 particle, P20, simulation results. Figures depicting the results of all the remaining PSO simulations are included in the appendix for your reference.

The data clearly illustrates that a larger number of particles does not necessarily indicate a better result. The parameter values are also similar over the range of particle numbers, with the exception of the cruise charge value for the 20 particle simulation.

Table 3: PSO Simulation Results

	Iteration	Max MPG	AccElec (W)	CruiseCharge(W)	PsocOffset	TargetSOC
P10	29	34.92	711	10732	5.19	40%
P20	49	34.93	682	7001	5.68	40%
P30	188	34.92	741	11556	2.74	40%
P40	377	34.91	531	10915	3.87	40%
P50	417	34.91	598	11189	2.27	40%
P60	349	34.9	737	12451	6.77	40%
P70	326	34.92	557	10955	5.55	40%
P80	448	34.91	773	10321	6.71	40%
P90	431	34.92	623	8907	2.75	40%
P100	908	34.91	485	11364	4.62	40%

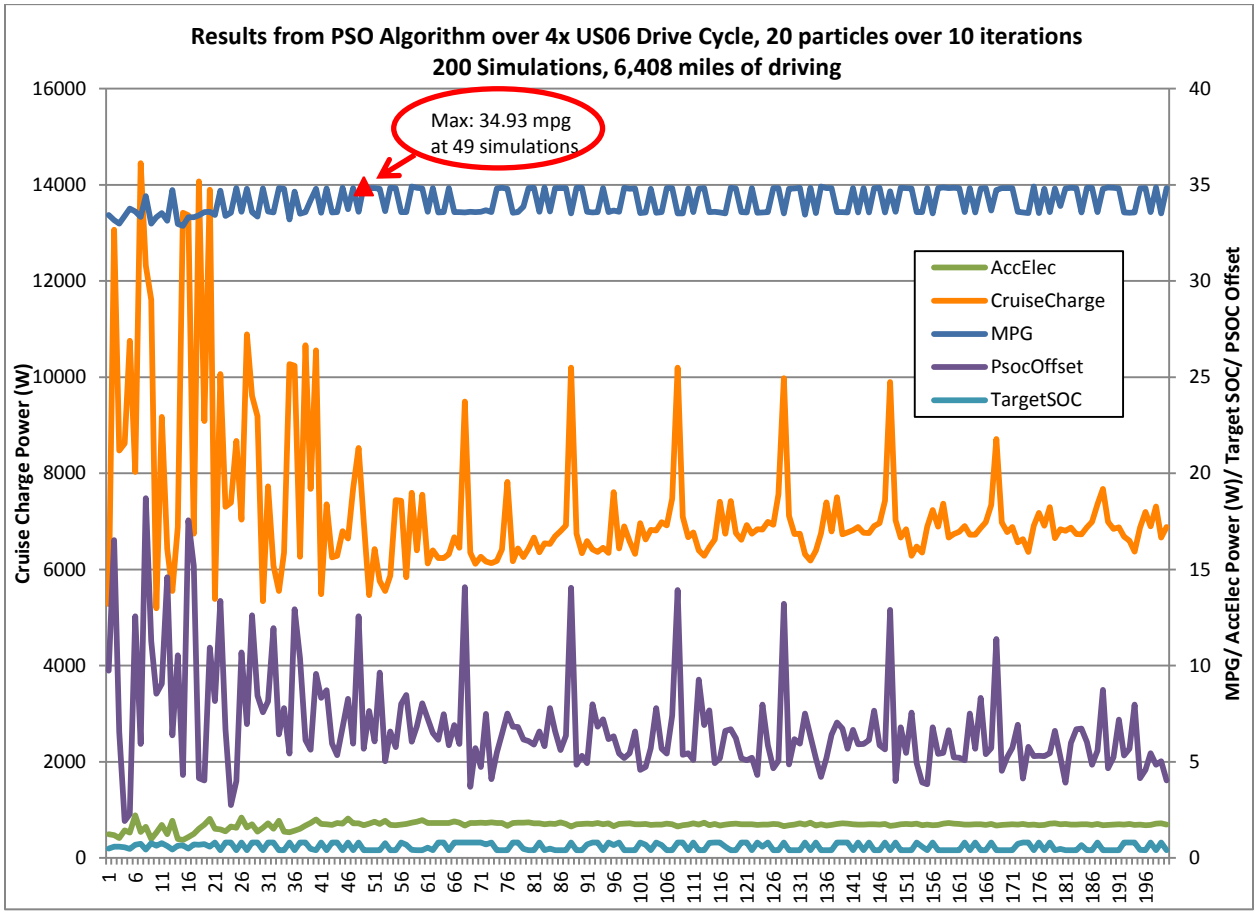


Figure 3: PSO Simulation results, 20 particles over 10 iterations

The results displayed in Figure 2 allow us to propose several conclusions. First the oscillatory nature of the simulated fuel economy suggests that the solution space contains multiple local maximums. Due to the shape of the solution space multiple similar local optimums occur which are close to the global maximum, which allows for a smaller number of particles to converge to

a local maximum. These local optimums, and similar points, exist for several different configurations of parameters which offer multiple solutions for optimizing fuel economy and reducing emissions. In addition the Cruise Charge and Psoc Offset parameters appear to converge to optimal values as the simulation proceeds confirming visually the functionality of the algorithm.

The particle swarm optimized fuel economy reached a maximum value of 34.93 MPG after 49 iterations. This is a 6.7% improvement over the baseline simulated fuel economy value of 32.58 MPG.

4.2 Genetic Algorithm Optimization Results

In a manner similar to the approach taken with the PSO algorithm the GA optimization simulations were run over a range of generation values as shown in Table 4. The performance of the GA approaches the maximum fuel economy result from the PSO with similar parameter values.

Table 4: Genetic Algorithm Simulation Results

Genetic Algorithm Optimization on 4 x US06 Drive Cycle						
	Iteration	Max MPG	AccElec (W)	CruiseCharge(W)	PsocOffset	TargetSOC
G20	249	34.84	432	12749	5	40%
G40	125	34.86	525	13086	4	40%
G80	787	34.79	846	13547	9	40%
G100	144	34.78	725	8500	7.3	40%

Similar to the PSO, the simulations with a smaller number of generations seem to perform better due to a large number of local maximums within the solution space. In contrast to the PSO results shown in Figure 2, the GA results do not depict convergence of the parameters towards an optimal value, as shown in Figure 3. The maximum fuel economy result from the GA

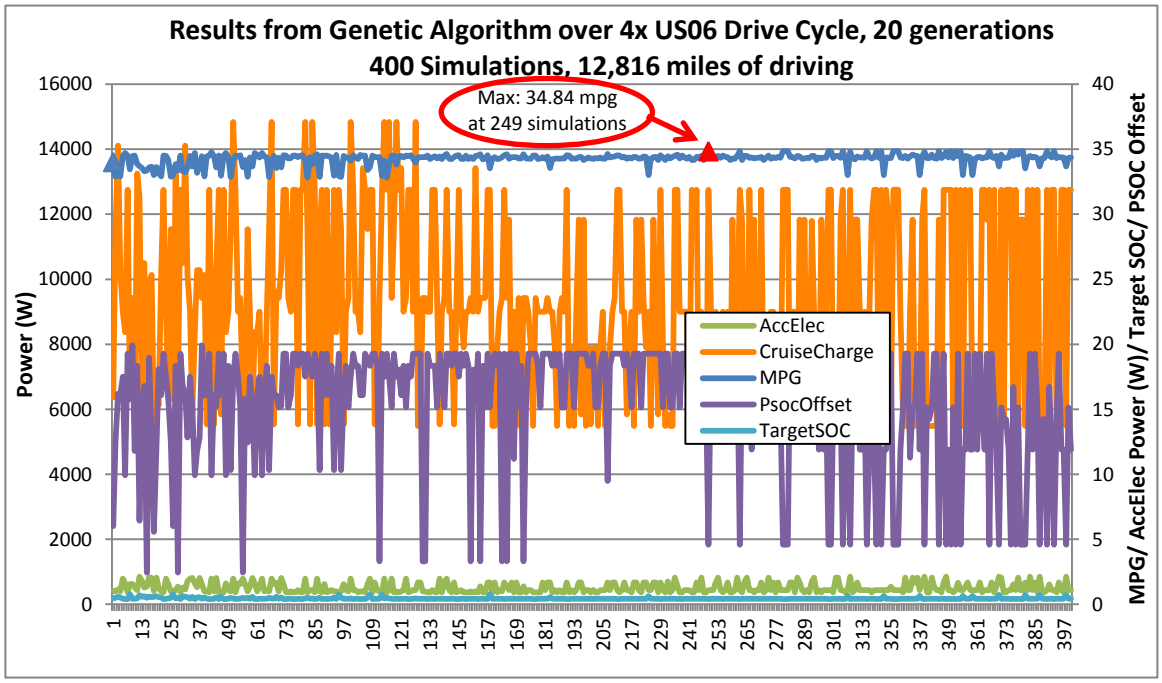


Figure 4: Genetic Algorithm Results for 20 generations

represents a 6.5% improvement over the baseline of 32.58 MPG.

As the GA did not reach or exceed the results of the PSO algorithm a modified GA was implemented utilizing an elitism selection method which copies a small proportion of the fittest candidates unchanged into the next generation. This Elitism GA, (EGA), was then applied to the 20 generation case to further explore the solution space for maximal values of fuel economy that meet or exceed the results of the PSO. Figure 4 represents EGA results which exceed the maximum fuel economy found by the PSO and GA. This maximum of 35.17 MPG represents a 7.4% increase over the baseline fuel economy of 32.58 MPG.

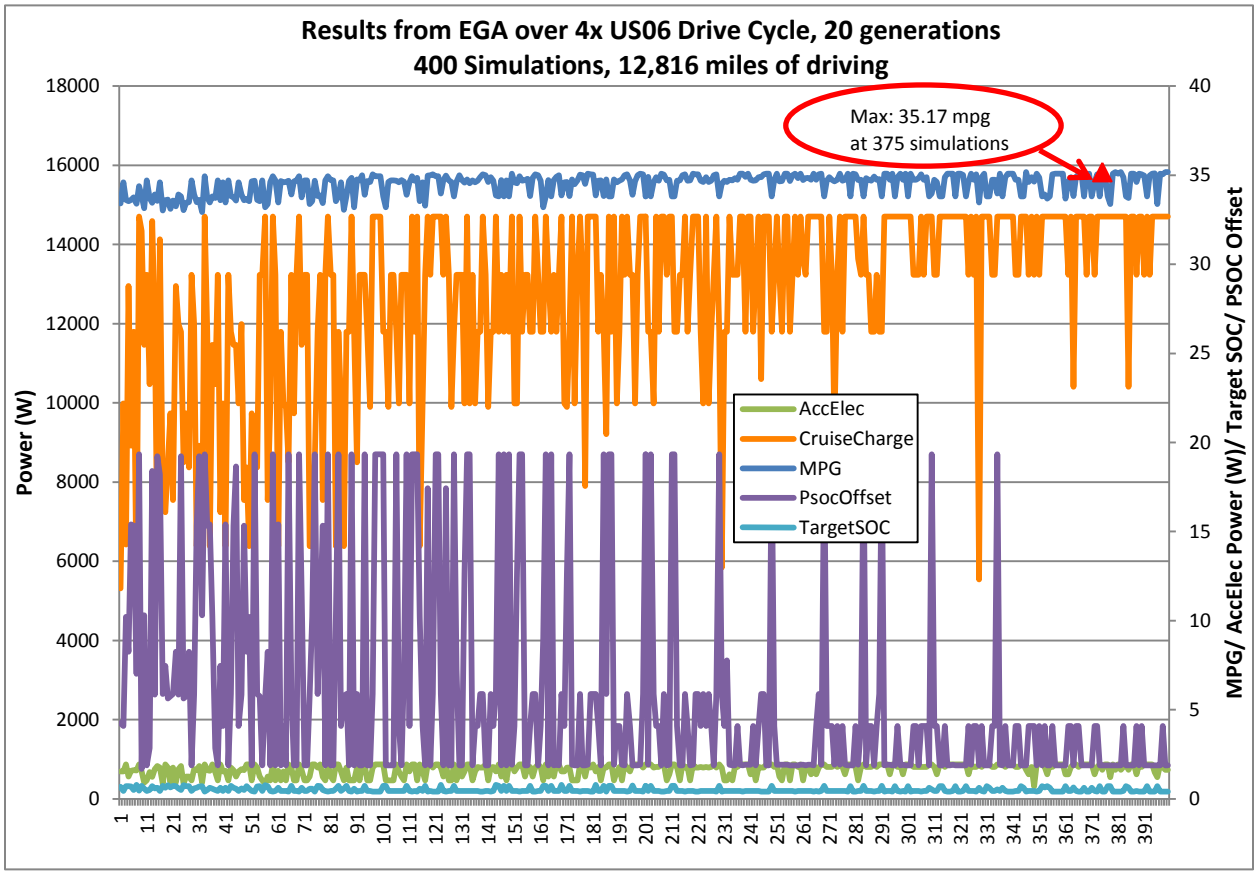


Figure 5: Elitism Genetic Algorithm Simulation Results

Additionally the PsocOffset and Cruise Charge parameters appear to converge to an optimal value as the simulation progresses, in a similar manner to the PSO results. Therefore the EGA method has proven to be optimal for exploring this solution space as it reached what is assumed to be the global maximum, exceeding the maximum reached by the PSO, and provides some semblance of convergence for the parameters.

5. Conclusion

In this project, population based stochastic optimization techniques such as particle swarm optimization (PSO) and genetic algorithm (GA) have been implemented to derive optimized parameters for the battery management system of a HEV. The intent of the algorithms is to optimize the HEV energy management parameters on the operating conditions of the vehicle to achieve minimal fuel consumption.

Results from simulations totaling over 340,000 miles show that both of these optimization techniques can provide significant improvement in vehicle control system optimization over conventional controls prototyping methods. Based on the results presented, the GA and PSO simulations with a smaller number of generations or particles respectively seem to perform better due to a large number of local maximums within the solution space that are close to the global maximum. Therefore the EGA method has proven to be optimal for exploring this solution space as it reached what is assumed to be the global maximum, exceeding the maximums reached by the GA and PSO, as well as providing some semblance of convergence for the parameters.

In addition, the developed methodology can provide a firm selection platform for components and parameters to efficiently handle the complex task of HEV system design which contains numerous local minima, discontinuous objective functions and nonlinear constraints.

In the future, this work will be formulated into an abstract for submission to the Society of Automotive Engineers for presentation at the upcoming International Powertrains, Fuels and Lubricants Meeting under the general powertrain development section which features a session on control system design and calibration. More immediately this work will be combined with tangential work to improve the SIL plant model fidelity and presented at the EcoCAR 3 competition May 18-19 as part of the MathWorks Modeling Award event.

References:

[1] X. Li and S. S. Williamson, “Comparative investigation of series and parallel hybrid electric vehicle (HEV) efficiencies based on comprehensive parametric analysis,” in Proc. *IEEE Vehicle Power and Propulsion Conf.*, Arlington, TX, Sept. 2007.

[2] J. Kennedy and R. C. Eberhart, “Particle swarm optimization,” in Proc. *IEEE International Conf. on Neural Networks*, 1995, vol. 4, pp. 1942-1948.

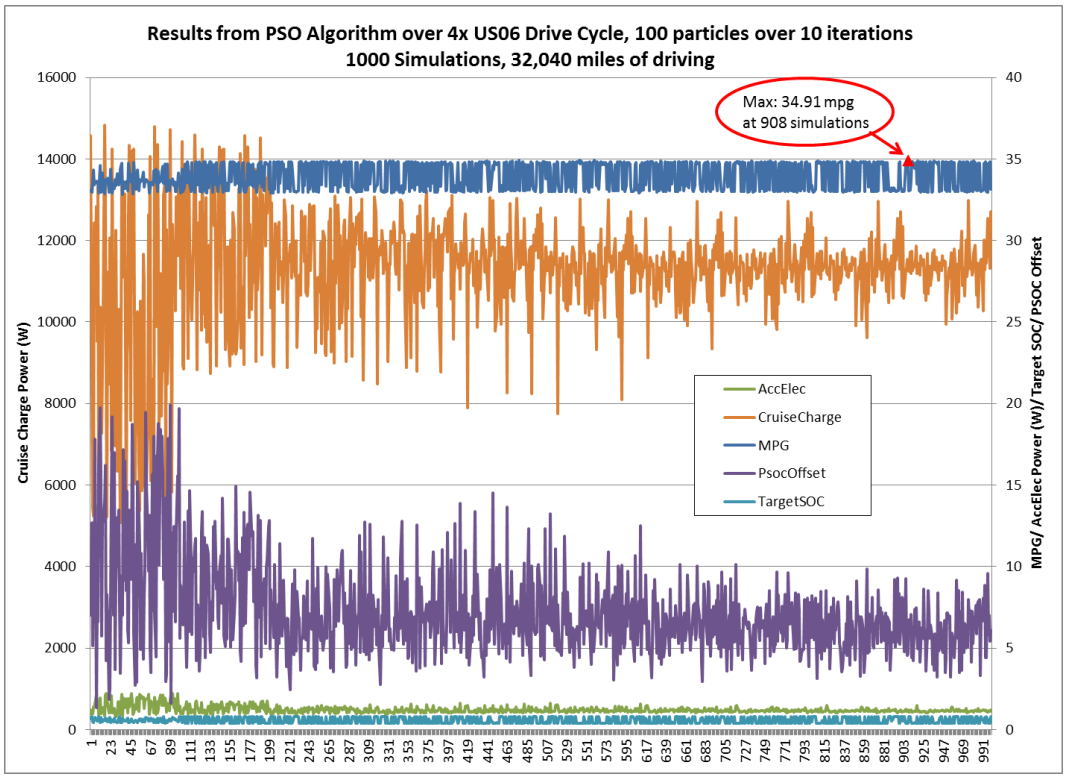
[3] K. Wipke, T. Markel, and D. Nelson, “Optimizing energy management strategy and degree of hybridization for a hydrogen fuel cell SUV,” in Proc. *18th International Electric Vehicle Symposium*, Berlin, Germany, 2001.

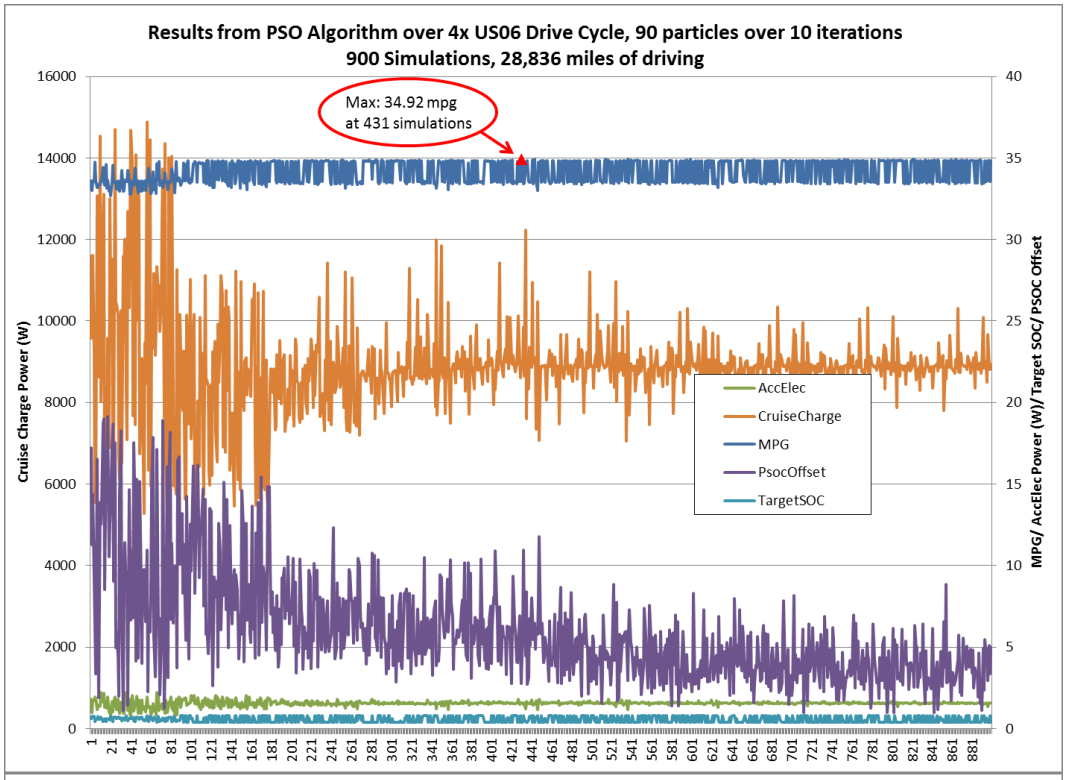
[4] R. Fellini, N. Michelena, P. Papalambros, and M. Sasena, “Optimal design of automotive hybrid powertrain systems,” in Proc. *1st International Symposium on Environmentally Conscious Design and Inverse Manufacturing*, Tokyo, Japan, Feb. 1999, pp.400-405.

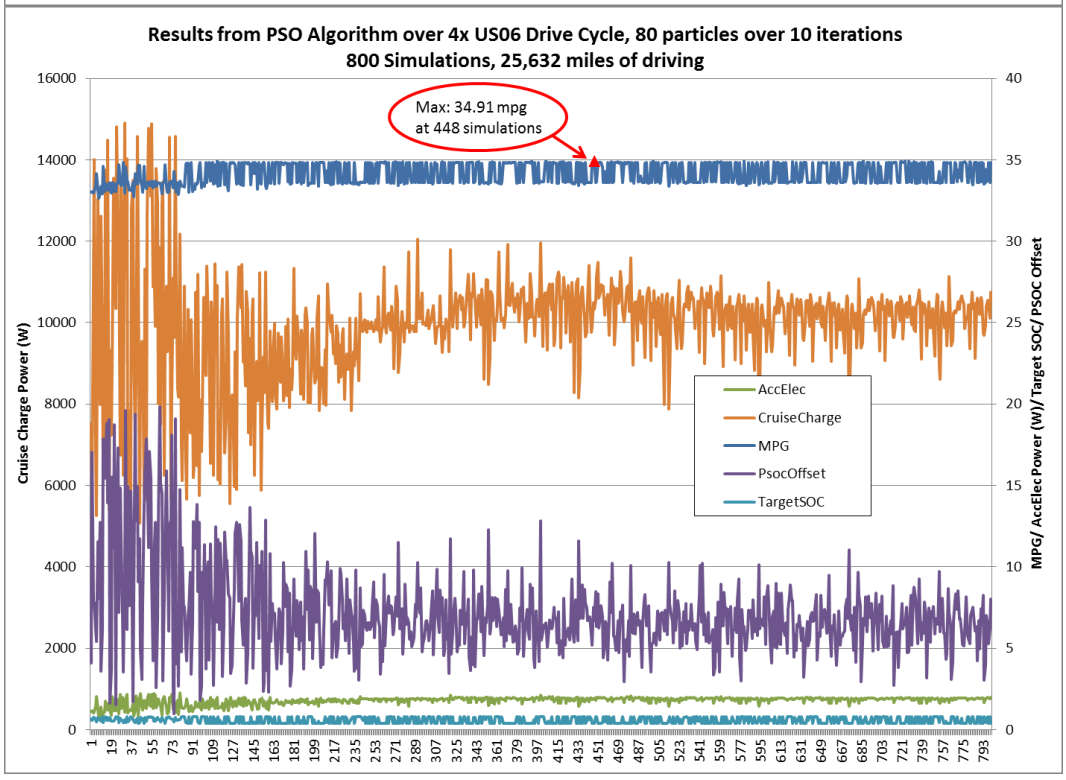
[5] T. Markel and K. Wipke, “Optimization techniques for hybrid electric vehicle analysis using ADVISOR,” in Proc. *ASME International Mechanical Engineering Congress and Expo.*, New York, NY, Nov. 2001.

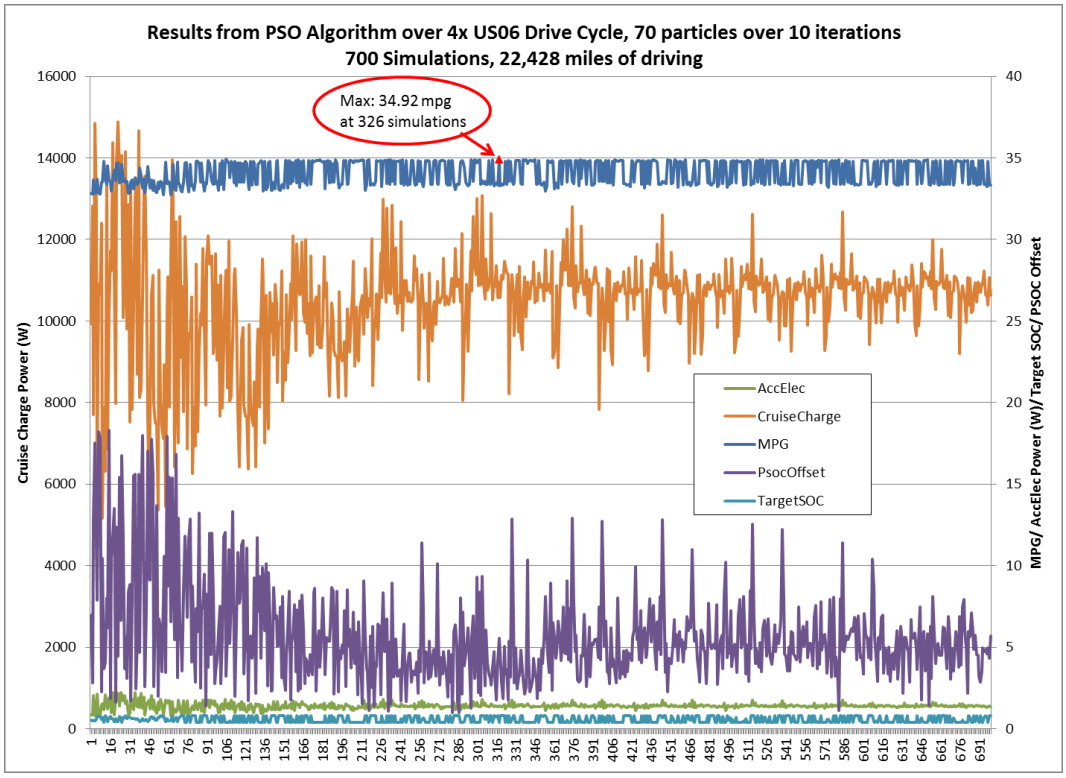
[6] S. N. Sivanandam and S. N. Deepa, *Introduction to Genetic Algorithms,* Springer, 2008.
[7] G. W. Flake, *The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation,* Cambridge, MA, MIT Press, 1998, pp. 343-350.

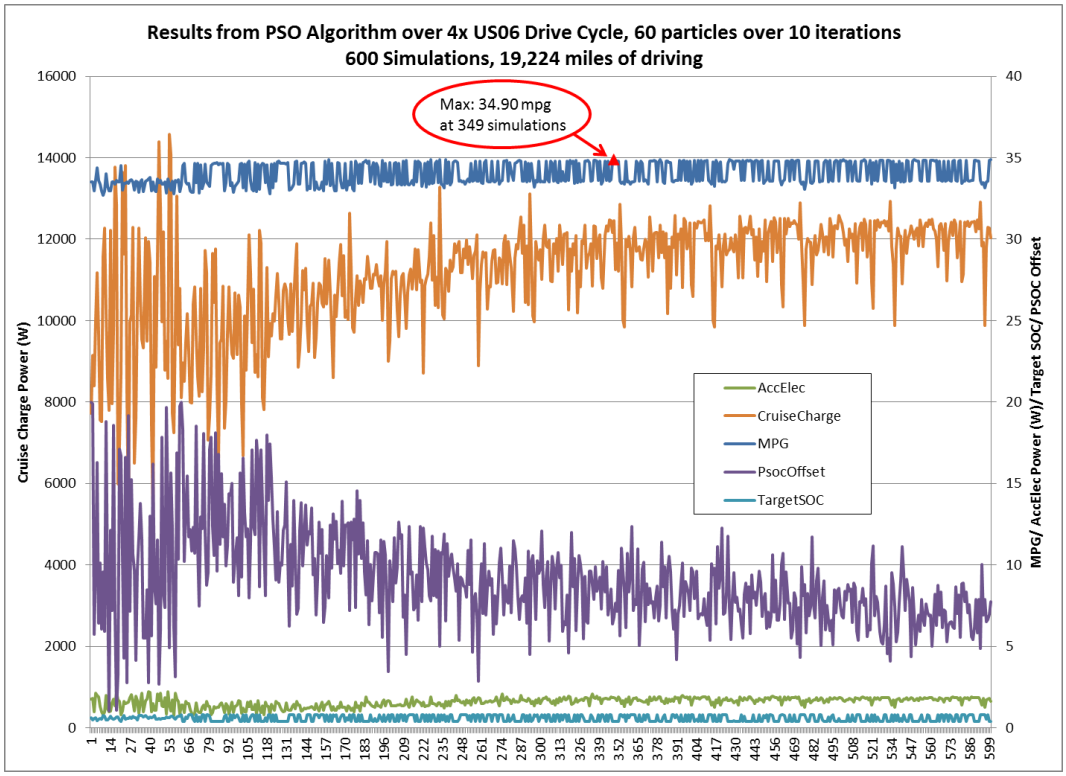
Appendix of Figures

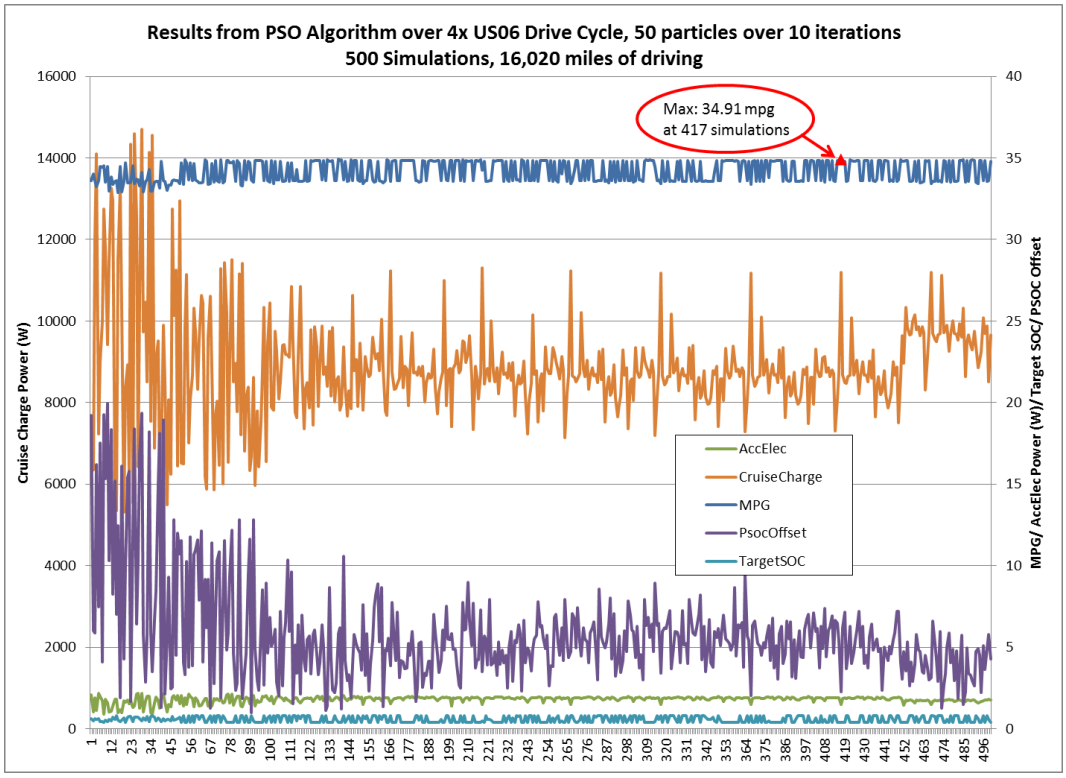


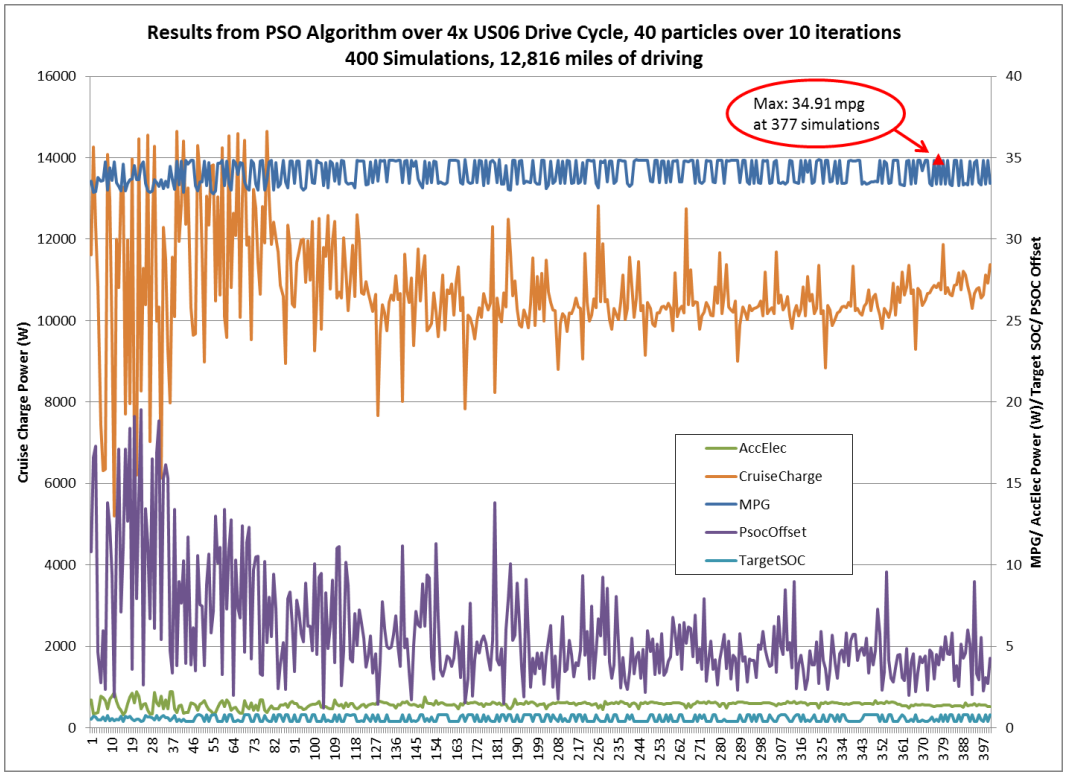

Results from PSO Algorithm over 4x US06 Drive Cycle, 90 particles over 10 iterations
900 Simulations, 28,836 miles of driving
Max: 34.92 mpg
at 431 simulations
Cruise Charge Power (W)
MPG/ AccElec Power (W)/ Target SOC/ PSOC Offset
AccElec
CruiseCharge
MPG
PsocOffset
TargetSOC

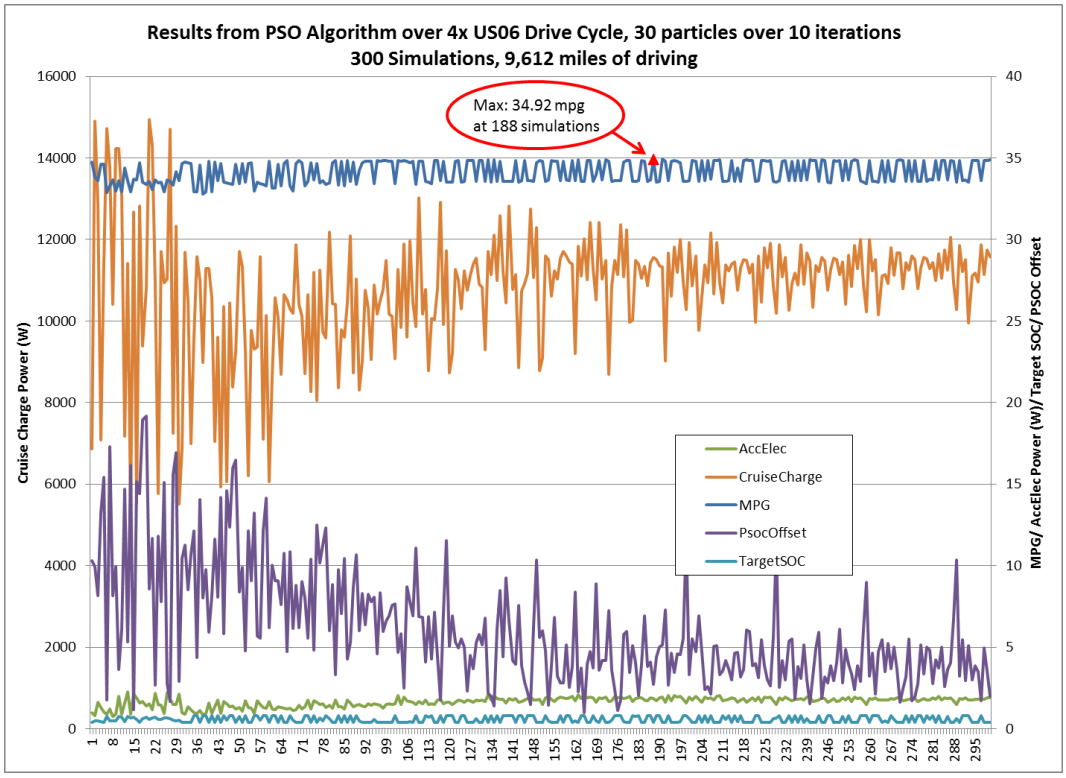

Results from PSO Algorithm over 4x US06 Drive Cycle, 80 particles over 10 iterations
800 Simulations, 25,632 miles of driving
Max: 34.91 mpg
at 448 simulations
Cruise Charge Power (W)
MPG/ AccElec Power (W)/ Target SOC/ PSOC Offset
AccElec
CruiseCharge
MPG
PsocOffset
TargetSOC

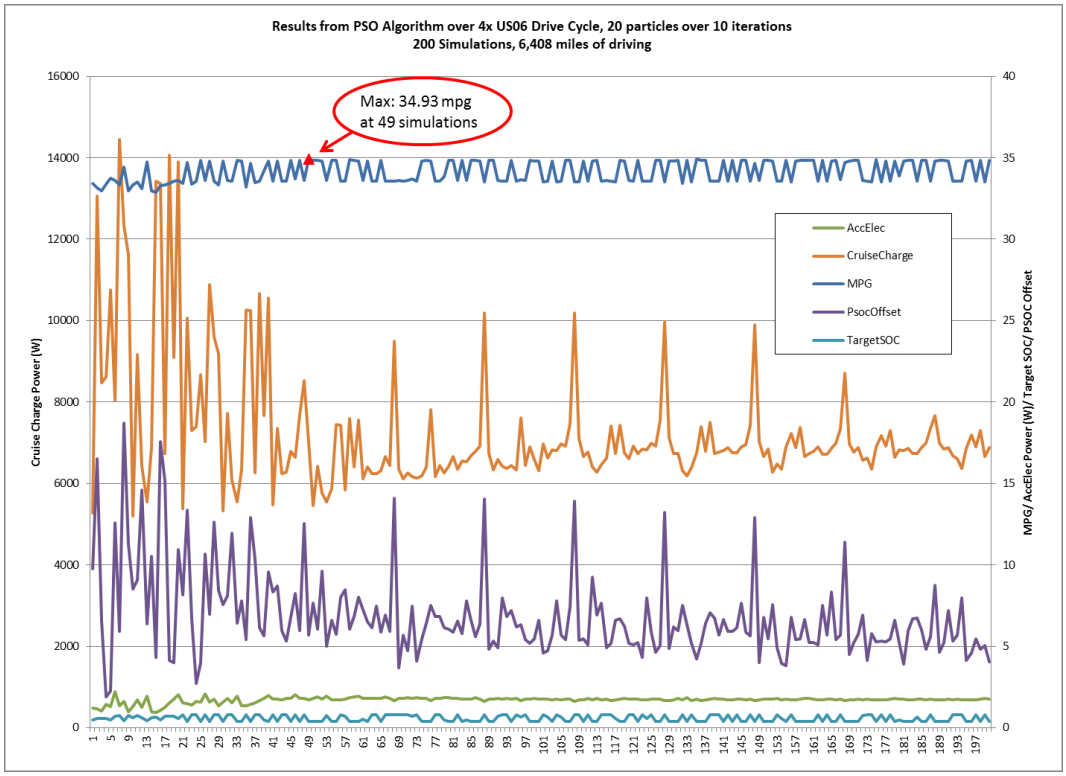

Results from PSO Algorithm over 4x US06 Drive Cycle, 70 particles over 10 iterations
700 Simulations, 22,428 miles of driving
Max: 34.92 mpg
at 326 simulations
Cruise Charge Power (W)
MPG/ AccElec Power (W)/ Target SOC/ PSOC Offset
AccElec
CruiseCharge
MPG
PsocOffset
TargetSOC

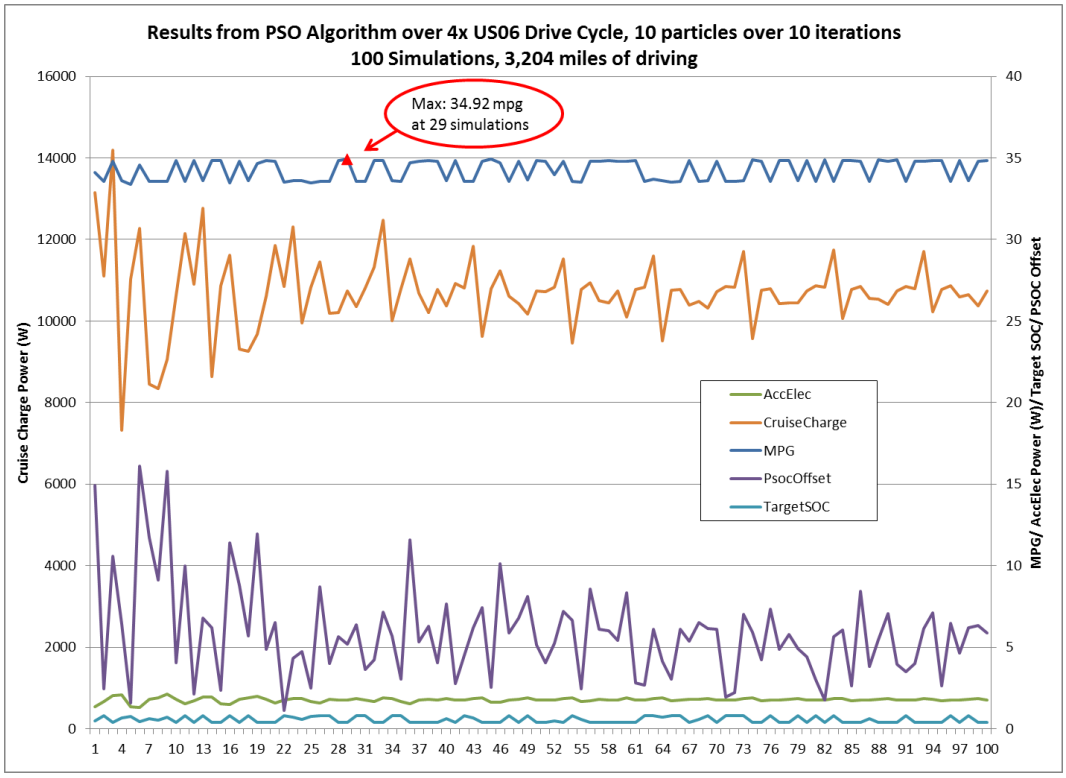

Results from PSO Algorithm over 4x US06 Drive Cycle, 60 particles over 10 iterations
600 Simulations, 19,224 miles of driving
Max: 34.90 mpg
at 349 simulations
Cruise Charge Power (W)
MPG/ AccElec Power (W)/ Target SOC/ PSOC Offset
AccElec
CruiseCharge
MPG
PsocOffset
TargetSOC

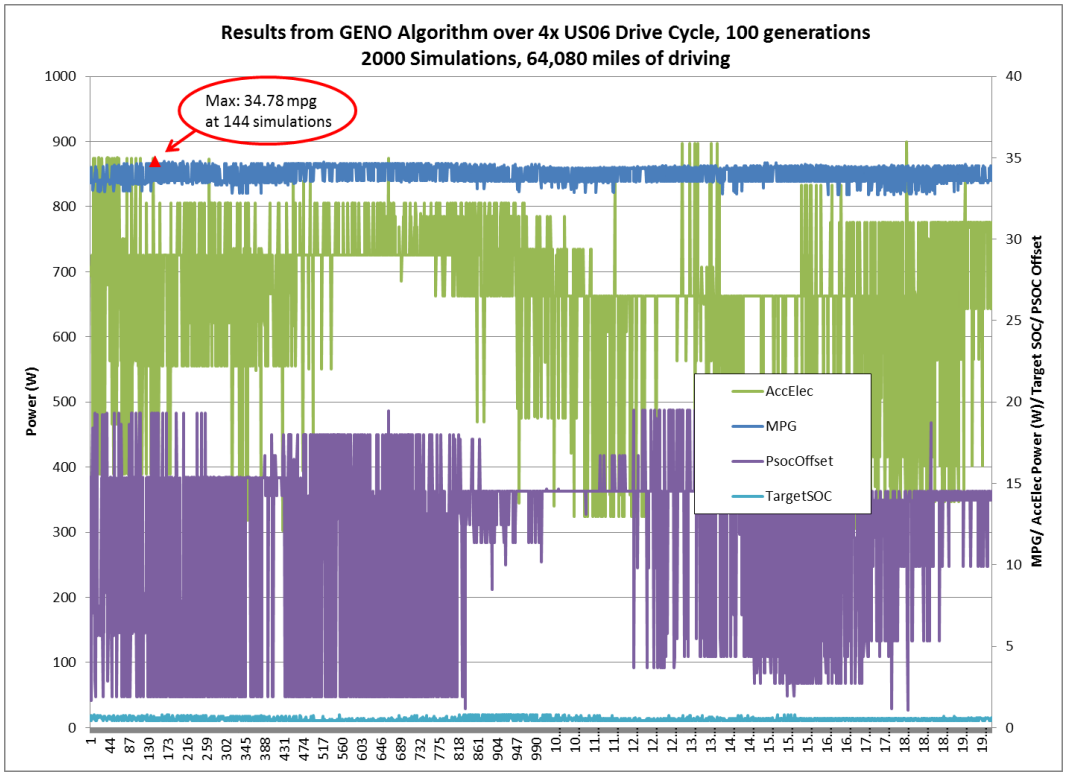

Results from PSO Algorithm over 4x US06 Drive Cycle, 50 particles over 10 iterations
500 Simulations, 16,020 miles of driving
Max: 34.91 mpg
at 417 simulations
Cruise Charge Power (W)
MPG/ AccElec Power (W)/ Target SOC/ PSOC Offset
AccElec
CruiseCharge
MPG
PsocOffset
TargetSOC

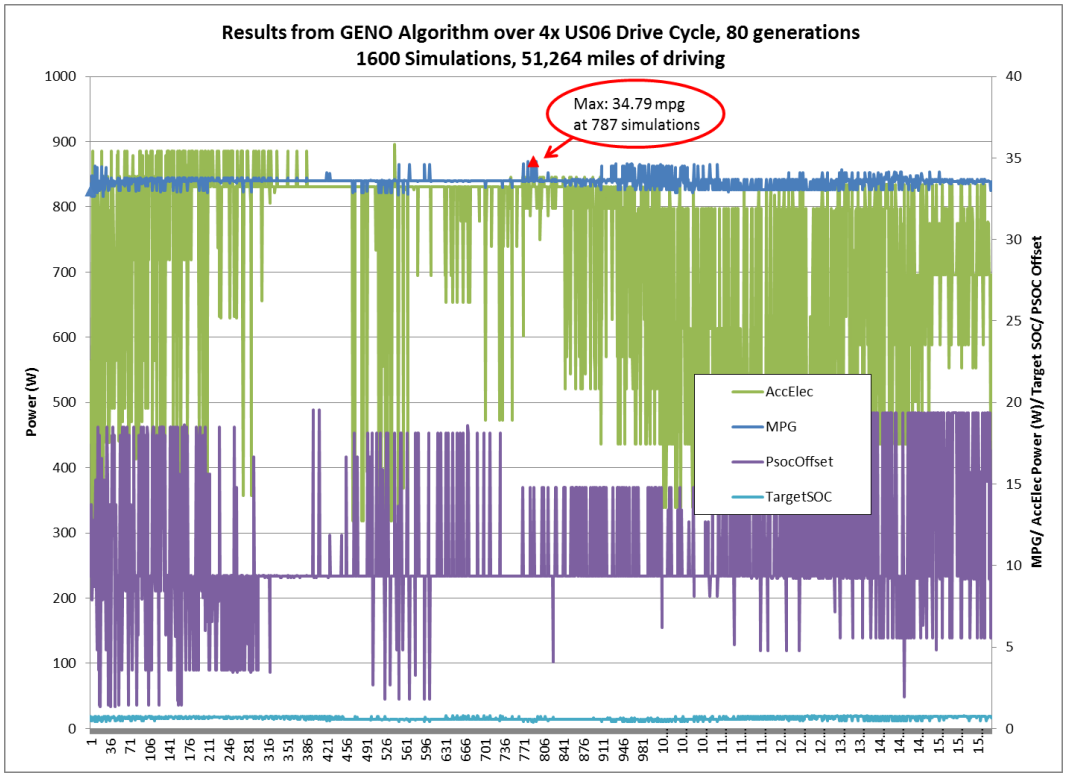

Results from PSO Algorithm over 4x US06 Drive Cycle, 40 particles over 10 iterations
400 Simulations, 12,816 miles of driving
Max: 34.91 mpg
at 377 simulations
Cruise Charge Power (W)
MPG/ AccElec Power (W)/ Target SOC/ PSOC Offset
AccElec
CruiseCharge
MPG
PsocOffset
TargetSOC

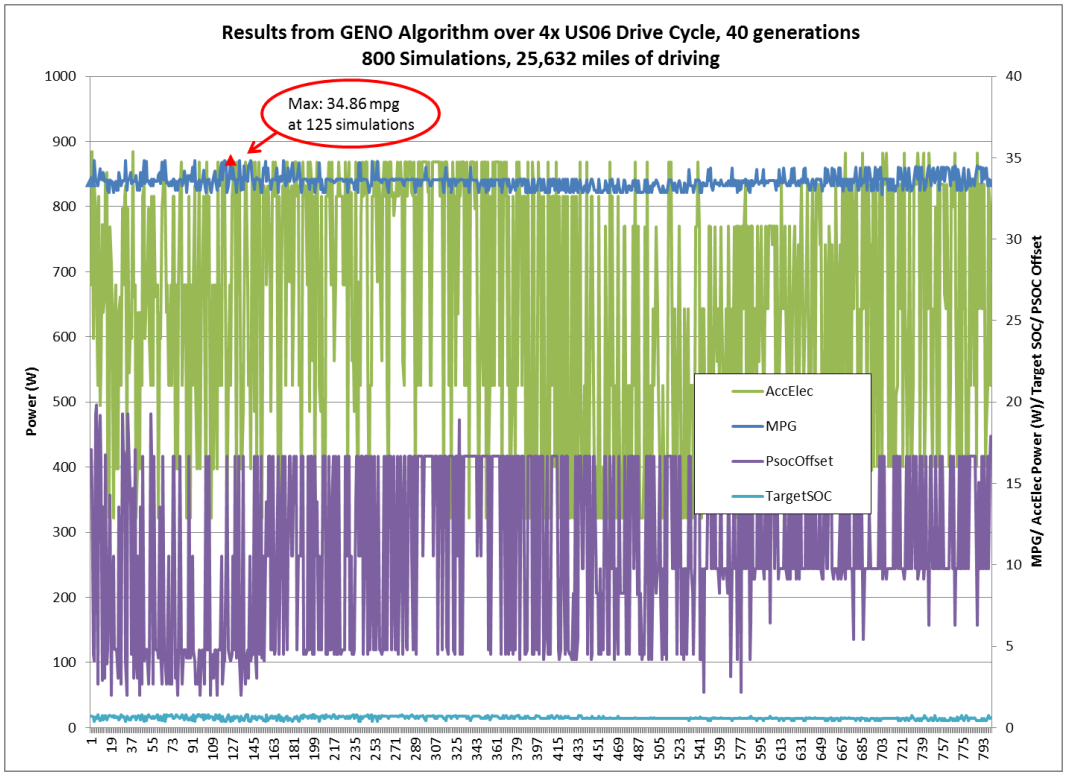

Results from PSO Algorithm over 4x US06 Drive Cycle, 30 particles over 10 iterations
300 Simulations, 9,612 miles of driving
Max: 34.92 mpg
at 188 simulations
Cruise Charge Power (W)
MPG/ AccElec Power (W)/ Target SOC/ PSOC Offset
AccElec
CruiseCharge
MPG
PsocOffset
TargetSOC

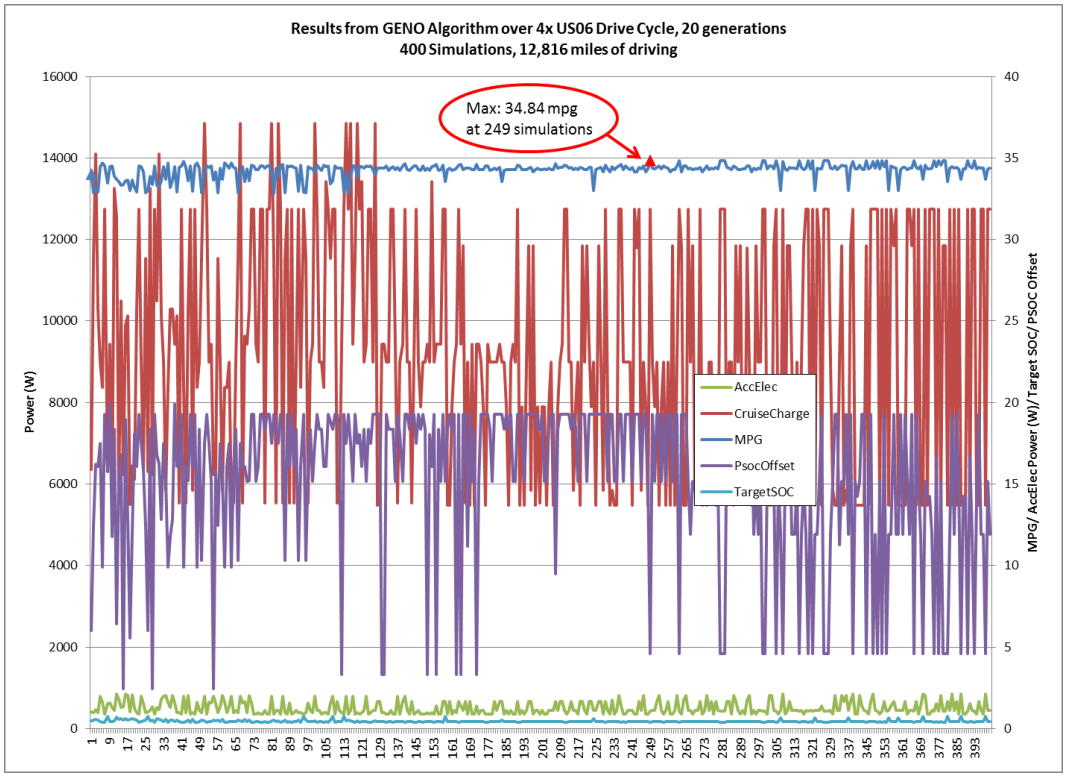

Results from PSO Algorithm over 4x US06 Drive Cycle, 20 particles over 10 iterations
200 Simulations, 6,408 miles of driving
Max: 34.93 mpg
at 49 simulations
Cruise Charge Power (W)
MPG/ AccElec Power (W)/ Target SOC/ PSOC Offset
AccElec
CruiseCharge
MPG
PsocOffset
TargetSOC

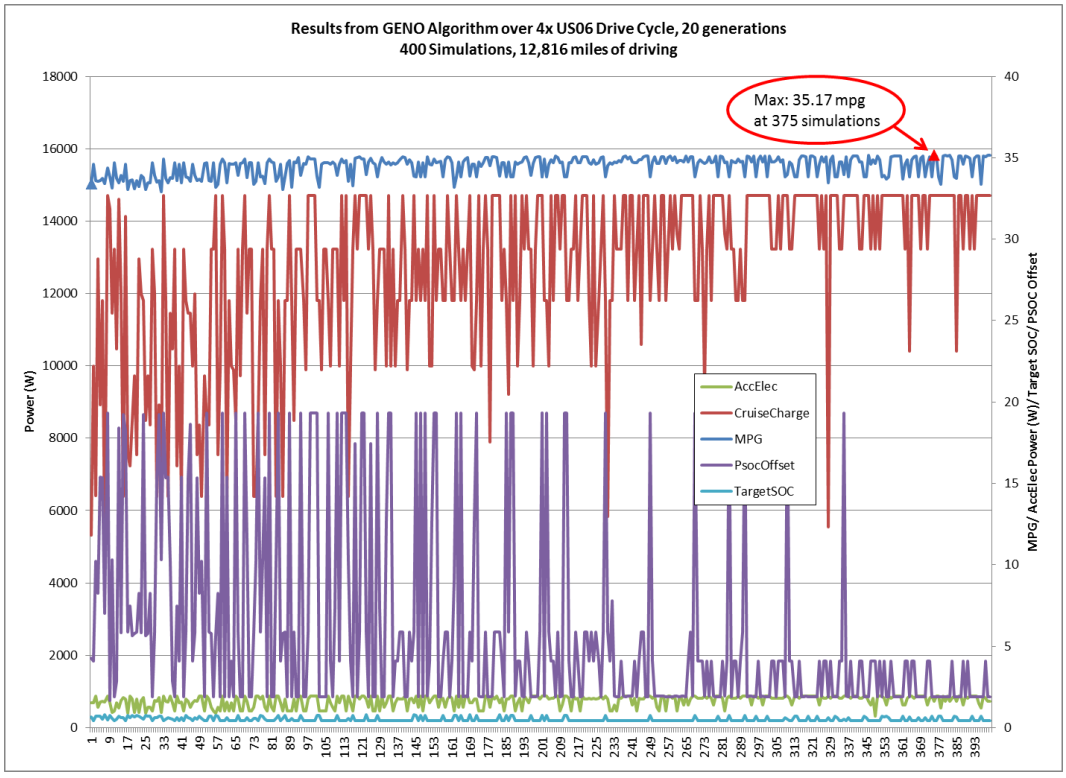

Results from PSO Algorithm over 4x US06 Drive Cycle, 10 particles over 10 iterations
100 Simulations, 3,204 miles of driving
Max: 34.92 mpg
at 29 simulations
Cruise Charge Power (W)
MPG/ AccElec Power (W)/ Target SOC/ PSOC Offset
AccElec
CruiseCharge
MPG
PsocOffset
TargetSOC


Results from GENO Algorithm over 4x US06 Drive Cycle, 100 generations
2000 Simulations, 64,080 miles of driving
Max: 34.78 mpg
at 144 simulations
Power (W)
MPG/ AccElec Power (W)/ Target SOC/ PSOC Offset
AccElec
MPG
PsocOffset
TargetSOC


Results from GENO Algorithm over 4x US06 Drive Cycle, 80 generations
1600 Simulations, 51,264 miles of driving
Max: 34.79 mpg
at 787 simulations
Power (W)
MPG/ AccElec Power (W)/ Target SOC/ PSOC Offset
AccElec
MPG
PsocOffset
TargetSOC


Results from GENO Algorithm over 4x US06 Drive Cycle, 40 generations
800 Simulations, 25,632 miles of driving
Max: 34.86 mpg
at 125 simulations
Power (W)
MPG/ AccElec Power (W)/ Target SOC/ PSOC Offset
AccElec
MPG
PsocOffset
TargetSOC


Results from GENO Algorithm over 4x US06 Drive Cycle, 20 generations
400 Simulations, 12,816 miles of driving
Max: 34.84 mpg
at 249 simulations
Power (W)
MPG/ AccElec Power (W)/ Target SOC/ PSOC Offset
AccElec
CruiseCharge
MPG
PsocOffset
TargetSOC


Results from GENO Algorithm over 4x US06 Drive Cycle, 20 generations
400 Simulations, 12,816 miles of driving
Max: 35.17 mpg
at 375 simulations
Power (W)
MPG/ AccElec Power (W)/ Target SOC/ PSOC Offset
AccElec
CruiseCharge
MPG
PsocOffset
TargetSOC

Appendix of Code

Genetic Algorithm Code

```
function finalp = GeneticAlgorithm_1_eliticism()

%Test function
f= @(x) cos(x(1))+sin(x(2))+cos(x(3))+sin(x(4))+cos(x(5)) + ...
    sin(x(1)+x(2)); %the best so far is [37.7399   89.4826  546.6246    1.6254   56.7295]

L=4; %number of genes
N=20; %population size, make this even
G=20;%number of generations 15:100
pm=.033;%probability of mutation
pc=.6;%probability of crossover

eliteNum=2;

%Parameter ranges (smallest value then largest, this is a Lx2 matrix.
ranges=[300,900;
        1,20;
        .40,.80;
        5000,15000;
        ];

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

pop=zeros(N,L); %population
pop=init(pop,ranges); %initialize population


gbest=zeros(L,1); %global best parameters
gbestVal=0; %global best val


 fprintf('     MPG            AccElec         PsocOffset     TargetSOC      CruiseCharge\n');       %[labels
for values]

%Remember fitness
fitness=zeros(1,N);
fitness_norm=zeros(1,N);

%Stats:
totalFitness_s=zeros(G,1);
averageFitness_s=zeros(G,1);
bestFitness_s=zeros(G,1);



%Generation loop
for g=1:G

    %Calculate fitness. -23 increases algorithm performance.
    parfor j=1:N
        fitness(j)=F(pop(j,:))-23;
    end

    %Go through fitnesses and test for new best value. This is done in
    %a seperate for loop to allow parallelization of the algorithm.
    for j=1:N
        if (fitness(j)>gbestVal)
            gbest=pop(j,:);
            gbestVal=fitness(j);
        end
    end

    %Normalize the fitnesses so that the sum is 1
    totalFitness=sum(fitness);
    fitness_norm=fitness/totalFitness;

    %fprintf("G: %d, F=%f\n",g,totalFitness);

    %Record stats at iteration g
    totalFitness_s(g)=totalFitness;
    averageFitness_s(g)=mean(fitness);
    bestFitness_s(g)=max(fitness);

    %Generate running total for parent selection later.
    %This allows us to use a rand() in order to select parents randomly,
    %but weighted towards better fitnesses. Randomness is important for
    %the function of the algorithm. The following for loop creates the
    %running total using the already normalized fitness values.
    running_total=zeros(1,N);
    running_total(1)=fitness_norm(1);
```

```
for j=2:N
    running_total(j)=fitness_norm(j)+running_total(j-1);
end


%Change population
newPop=zeros(N,L);
%This is done N/2 times because you get two children each time.
%Thus, you get a population of size N again. Note, constrain N with
% N mod 2 = 0, N > eliteNum

%This requires eliteNum==2
[tmp,tmp2]=sort(fitness);
elite1=pop(tmp2(end),:);
elite2=pop(tmp2(end-1),:);

for j=1:N/2 - eliteNum/2
    %Select two parents p1,p2 using two random values i1,i2
    i1=rand();
    p1=-1;
    i2=rand();
    p2=-1;

    %Use our random value to select a parent. This is weighted based
    %on the porportion of fitness values.
    for k=1:N
        if (p1==-1 && i1<running_total(k))
            p1=k;
        end
    end

    %Select parent 2 the same way, but ignore duplicates.
    while (p2==-1 || p2==p1) %we haven't run or they are the same
        for k=1:N %I use a second loop so this can repeat
            if (i2<running_total(k))
                p2=k;
                if (k==p1 && p1~=N) %Quick way of ignoring duplicate
                    p2=k+1;
                else
                    p2=k;
                end
                break;
            end
        end
        if (p2==p1)%if statement is not needed, but is more clear
            i2=rand(); %We need to try a new random parent
        end
    end

    %Create offspring - initially
    o1=pop(p1,:);
    o2=pop(p2,:);

    %Cross them over - I pick a random index and flip all the genes
    %in the two children after that happens
    if (rand()<pc)%Check if it occurs
        point=randi(L);
        tmp=o1(point:L);
        o1(point:L)=o2(point:L);
        o2(point:L)=tmp;
    end

    %Mutate offspring - each gene in each offspring has a pm chance of
    %being mutated to something uniformly random in its desired range.
    for k=1:L %offspring 1
        if (rand()<pm)
            o1(k)=rand()*(ranges(k,2)-ranges(k,1))+ranges(k,1);
        end
    end
    for k=1:L %offspring 2
        if (rand()<pm)
            o2(k)=rand()*(ranges(k,2)-ranges(k,1))+ranges(k,1);
        end
    end
    %Put the children in the new population
    newPop(2*j-1,:)=o1;
    newPop(2*j,:)=o2;
end
%assume eliteNum == 2
newPop(end,:)=elite2;
newPop(end-1,:)=elite1;
pop=newPop;
```

```
    %Bounderies - this shouldn't be an issue on a GA, but we'll check
    %anyway
    pop=bound(pop, ranges);

end

finalp=gbest;
fprintf('%f\n', gbestVal);


figure(1);
plot(1:G,totalFitness_s);
title('Total Fitness');
figure(2);
plot(1:G,averageFitness_s);
title('Average Fitness');
figure(3);
plot(1:G,bestFitness_s);
title('Best Fitness');
figure(4);
plot(1:N,sort(fitness));
title('Population Fitness');

end


%Put points outside the boundery back into the boundery
function p = bound(p, ranges)

[pnum]=size(p);
pnum=pnum(1);
[par] = size(ranges);
par=par(1);

for i=1:pnum
    for j=1:par %ranges
        if (p(i,j)<ranges(j,1)); p(i,j)=ranges(j,1); end
        if (p(i,j)>ranges(j,2)); p(i,j)=ranges(j,2); end
    end

end

end

%Initialize uniformly randomly from desired ranges
function p = init(p, ranges)

[pnum]=size(p);
pnum=pnum(1);
[par] = size(ranges);
par=par(1);

for i=1:pnum
    for j=1:par %ranges
        p(i,j)=rand()*(ranges(j,2)-ranges(j,1))+ranges(j,1);
    end
end

end
```

Fuel Economy Function Code

```
function fuel_mileage_mpg = F(x)

%Get fuel consumption and distance
%drive_cycle_distance_m = evalin('base', 'sch_metadata.distance.value');
evalin('base', 'load(''wtf.mat'')');

evalin('base', sprintf('accelec.plant.init.pwr=%d;',x(1)));
evalin('base', sprintf('vpc.prop.init.emcp_psoc_table_offset=%d;',x(2)));
assignin('base', 'Target_SOC',x(3));
assignin('base', 'CC_Upper',x(4));

%assignin('base', 'BPP_Regen_Max',x(1));
%assignin('base', 'OffThrottleRegen_Max',x(2));
%assignin('base', 'Psoc_Upper',x(5));
%assignin('base', 'Psoc_Lower',x(6));

%For structs use this format


warning off;
```

```
    simOut = sim('Camaro_SIL','SimulationMode','Accelerator');
    %set_param('Camaro_SIL','SimulationCommand','update');

FC = max(simOut.get('eng_plant_fuel_cum_simu'));
%Dist = evalin('base','sch_metadata.distance.value');
V = trapz(simOut.get('ess_plant_volt_out_simu'));
I = trapz(simOut.get('ess_plant_curr_out_simu'));
D = trapz(simOut.get('chas_plant_lin_spd_out_simu'));
Dx = D*0.0000621371;
    fuel_mileage_mpg = Dx/FC;
    PElec = V*I;

    fprintf(' %e     %e        %4.2f        %4.2f        %4.2f        %4.2f        %4.2f
%4.2f\n',fuel_mileage_mpg,PElec,x(1),x(2),x(3),x(4),x(5),x(6));

    %disp(max(evalin('base','eng_plant_fuel_cum_simu')));
    %disp(max(evalin('base','BPP_Regen_Max')));

end
```

Particle Swarm Code

```
function finalp = ParticleSwarm()
 ms = MultiStart('UseParallel',true);
 pctRunOnAll('addpath C:\Users\campbellru\Documents\MATLAB\PSO_04112018')
%Function
%f= @(x) ((cos(x(1)./4) + sin(x(2))./((x(1).^2+x(2).^2).^.25)).*sin(x(1).*x(2)/10) + ...
%    ((cos(x(3)./4) + sin(x(4))./((x(3).^2+x(4).^2).^.25)).*sin(x(3).*x(4)/10) + ...
%    sin(x(5));

f= @(x) cos(x(1))+sin(x(2))+cos(x(3))+sin(x(4))+cos(x(5)) + ...
    sin(x(1)+x(2));

par=4;
pNumber=60;

tend=10;


%weights
%inertia, gbest, pbest, rand
w=[.3,.5,.4, 1.5];

%Weights on individual velocities
%v_weights=[1,1,100,1,1];

ranges=[300,900;
        1,20;
        .40,.80;
        5000,15000;
        ];

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

p=zeros(pNumber,par); %particles (position)
p=init(p,ranges);
solu=zeros(pNumber,1); %Remember solution
v=zeros(pNumber,par); %velocity

best=zeros(pNumber,1); %Remember value of personal best
bestp=p; %Remember coordinates of personal best
gbest=0; %global best index
gbestVal=0; %global best val


%Initialize
    fprintf('     MPG            PElec      Psoc_Upper     Psoc_Lower     CruiseCharge     PsocOffset
TargetSOC\n');        %[labels for values]

for i=1:tend

    parfor j=1:pNumber
        rv = F(p(j,:)); %This is where the optimized function is called
        solu(j)=rv;
        if rv > best(j)
            best(j) = rv;
            bestp(j,:)=p(j,:);
        end
    end
```

```
    [gbestVal, gbest]=max(best);

    %Find velocity
    %w1*intertia + w2 * globalBestDifference + w3 * personal Best distance
    v= w(1)*v + w(2)*(bestp-p) + w(3)*(repmat(bestp(gbest,:),pNumber,1)-p) +w(4)*(rand(pNumber,par).*2-1);
    %note that rand is [0,1] so it is changed to [-1,1]=[0,1]*2-1


    %Scale to 1
    %v=sqrt(particle(:,6).^2+particle(:,7).^2);
    %v=v+.01; %avoid divide by 0 hack
    %particle(:,6)=2.*particle(:,6)./v;
    %particle(:,7)=2.*particle(:,7)./v;

    %v=v.*v_weights;

    disp(i);
    %disp(v);

    %Add velocity
    p=p+v;

    %Bounderies
    p=bound(p, ranges);

    %quantize

    %bounds

end

finalp=bestp(gbest,:);
disp(finalp);
%disp(best);
fprintf('%i  %f\n',gbest, gbestVal);


end


function p = bound(p, ranges)

[pnum]=size(p);
pnum=pnum(1);
[par] = size(ranges);
par=par(1);

for i=1:pnum
    for j=1:par %ranges
        if (p(i,j)<ranges(j,1)); p(i,j)=ranges(j,1); end
        if (p(i,j)>ranges(j,2)); p(i,j)=ranges(j,2); end
    end

end

end

function p = init(p, ranges)

[pnum]=size(p);
pnum=pnum(1);
[par] = size(ranges);
par=par(1);

for i=1:pnum
    for j=1:par %ranges
        p(i,j)=rand()*(ranges(j,2)-ranges(j,1))+ranges(j,1);
    end
end

end
```