Optimization of Axle NVH Performance Using Particle Swarm Optimization

Glenn Meinhardt
Department of Industrial and Systems Engineering
Oakland University
Rochester, Michigan 48309
Email: gameinha@oakland.edu

Sankar Sengupta
Department of Industrial and Systems Engineering
Oakland University
Rochester, Michigan 48309
Email: ssengupta@oakland.edu

Abstract

An approach to optimization of automobile axles for noise, vibration and harshness (NVH) performance based on end-of-line testing is presented. The method used, the particle swarm optimization method (PSO), iteratively solves an objective function based on positioning values of the independent variables of the objective function within their corresponding constraints. A Matlab program written by the authors is presented and discussed. The algorithm used within the method is presented along with solutions under different population sizes.

Introduction

Noise, Vibration and Harshness (NVH) performance is a critical quality characteristic for automobile manufacturers (original equipment manufacturers, or OEMs) and driveline component manufacturers alike. A major component of the driveline is the axle. The axle transfers torque from the engine and driveshaft to the wheels. For axle manufacturers, one of the primary NVH metrics is gear whine [1]. To ensure satisfactory gear whine performance when the automobile leaves the factory, many OEMs now require axle assemblies to be tested for gear whine performance at the end of the assembly line using an end-of-line NVH test (EOLT) prior to shipment to their assembly plants. It is in the best interest of both the OEMs and axle manufacturers to ensure that the vibration levels of axles not only meet the requirement at the EOLT, but that the levels are as low as possible [2]. One way to control the levels at the EOLT is to understand the correlation of the upstream performance variables to the EOLT result. A previous work by the authors examined one such correlation [3, 4] involving the assembly parameters of the axle and the resulting coast-side vibration. This work illustrates the use of the cross entropy method to minimize the EOLT result with the regression equation presented in [4] used as the objective function. The solution of the same problem is presented in other works by the authors using the Cross Entropy Method [5] and a Genetic Algorithm [6].
The Optimization Problem

The desire is to minimize the cost-side vibration given by the regression equation from [4]:

\[
Y(X) = 486.32 - 2.8049 a1 - 2.7890 a5 - 0.19745 c6 + 0.36987 c7 - 0.29785 d3 - 0.26230 d6
\]  \hspace{1cm} (1)

Therefore, the optimization problem is written

\[
\gamma^* = \min Y(X) = Y(X^*) \hspace{1cm} (2)
\]

where \(\gamma^*\) is the optimum value of \(Y\), \(X^* = [a1 \ a5 \ c6 \ d3 \ d6]\), and \(X^*\) are the values of \(X\) associated with \(\gamma^*\). The regression equation was derived from 21 samples of data collected from the assembly line. Clearly the regression equation is only valid for the range of data from which it was derived. Therefore, the boundary conditions (constraints) for the optimization problem are the range of each variable from which the objective function was derived. The constraints are taken from range of data in [4] and summarized in Table 1. Equations (1) and (2) along with Table 1 completely define the optimization problem. Now, this problem can be solved very easily deterministically and that solution is given in Table 2. The purpose of this work is to illustrate how the cross entropy method can be used to solve optimization problems. The simple problem presented above and the deterministic solution can be used as a basis for such an illustration, the results of the optimization compared to the deterministic solution.

Table 1 – The Constraints for the Parameters, \(X\), of the Regression Equation (Various Units, dB)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>66.8485</td>
<td>69.5424</td>
</tr>
<tr>
<td>a5</td>
<td>68.8496</td>
<td>70.9555</td>
</tr>
<tr>
<td>c6</td>
<td>40.0000</td>
<td>71.5957</td>
</tr>
<tr>
<td>c7</td>
<td>46.0206</td>
<td>78.6900</td>
</tr>
<tr>
<td>d3</td>
<td>46.0206</td>
<td>65.1055</td>
</tr>
<tr>
<td>d6</td>
<td>56.9020</td>
<td>71.3640</td>
</tr>
</tbody>
</table>

Particle Swarm Optimization

Particle Swarm Optimization method (PSO) is an optimization method that was developed by Eberhart and Kennedy in 1995 [7]. The method iteratively converges to the solution of an
optimization problem by manipulating the variables in such a way that mimics a swarm or flock of birds converging on a food source or to a particular perch or other location. This work will apply PSO following the method as it was used by Robinson and Rahmat-Samii to solve an electromagnetics problem [8]. Robinson and Rahmat-Samii apply the fundamental PSO method outlined by Kennedy and Eberhart, but they also present an interesting approach to handling constraints. PSO follows a very simple algorithm:

1. Select the number of particles – usually known solutions, or random solutions.
2. Calculate the fitness of each solution.
3. Compare all of the fitness values – the solution that has the fitness value closest to optimum is the “Global Best”.
4. Calculate the “velocity” of each particle – changes in each variable that will move the particle closer to the optimum.
5. Adjust the position based on the velocity and the constraints – only feasible positions are allowed.
6. Compare the fitness value of each new position of each particle to its previous position – store the position that has the fitness value closest to optimum as the new “Personal Best”.
7. Compare all of the Personal Bests to the Global Best – whichever solution is closest to the optimum becomes the new Global Best.

Table 2 – The Solution to the Deterministic Form of the Optimization Problem

<table>
<thead>
<tr>
<th>Optimization Method</th>
<th>a1</th>
<th>a5</th>
<th>c6</th>
<th>c7</th>
<th>d3</th>
<th>d6</th>
<th>Optimum Solution Y(X*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deterministic</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>46.0206</td>
<td>65.1055</td>
<td>71.3640</td>
<td>58.1402</td>
</tr>
</tbody>
</table>
8. Repeat Steps 4 through 7 until desired convergence criteria are met (e.g., small change in Global Best, maximum iterations).

The Global Best solution when the method converges is the optimum solution identified by the method.

Initializing the Swarm

Initialization of PSO begins with the definition of the sample space for each variable. Here, the sample space is defined as the range of values that were used to define the optimization equation derived by Meinhardt and Sengupta [4]. The sample space are all of the values that satisfy the constraints on X presented in Table 1. Next, a set of samples from the sample space of X are defined. Here, for the first trial, the original set of 21 samples, shown in Table 3, is used to initialize the swarm. The initial fitness value is the actual NVH result from the assembly line. This is also shown in Table 3. Another acceptable method for initialization is to select a desired number of particles, and assign the positions randomly using the initial set of 21 samples to define the sampling distributions, calculating the fitness of each particle using Equation (1). This method is also used in additional trials.

Moving the Swarm

In the PSO analog, each sample is treated as a particle. The particle analogy is appropriate because each sample has no ‘mass’, but is assigned ‘position’ and ‘velocity’. The position of each particle corresponds to the current value of each variable, X, and is modified at each iteration according to Equation (3)

$$X = XP + V$$  \hspace{1cm} (3)

with XP defined as the position of each particle at the end of the previous iteration. At initialization, XP = X. The velocity of the particle is defined by Equation (4).

$$V = wVP + c_1 \text{ rand}() (PBEST - X) + c_2 \text{ rand}() (GBEST - X)$$  \hspace{1cm} (4)

Equation (4) has three terms, each with a very specific purpose:

1. An inertia term

   This term, $wVP$, applies a weight ($w$) to the current velocity (that is, the velocity at the end of the previous iteration, VP)
Table 3 – The Initial Data From the Assembly Line, Various Units, dB

<table>
<thead>
<tr>
<th>Sample</th>
<th>a1</th>
<th>a5</th>
<th>c6</th>
<th>c7</th>
<th>d3</th>
<th>d6</th>
<th>NVH</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>67.9588</td>
<td>69.5134</td>
<td>71.5957</td>
<td>60.0000</td>
<td>46.0206</td>
<td>61.5836</td>
<td>81.03</td>
</tr>
<tr>
<td>2</td>
<td>66.8485</td>
<td>70.9555</td>
<td>53.9794</td>
<td>58.0618</td>
<td>58.0618</td>
<td>60.8279</td>
<td>78.18</td>
</tr>
<tr>
<td>3</td>
<td>67.6042</td>
<td>70.2377</td>
<td>53.9794</td>
<td>66.0206</td>
<td>49.5424</td>
<td>63.5218</td>
<td>86.26</td>
</tr>
<tr>
<td>4</td>
<td>68.2995</td>
<td>70.6296</td>
<td>55.5630</td>
<td>62.9226</td>
<td>49.5424</td>
<td>63.5218</td>
<td>76.49</td>
</tr>
<tr>
<td>5</td>
<td>67.2346</td>
<td>70.0212</td>
<td>66.4444</td>
<td>60.8279</td>
<td>52.0412</td>
<td>60.0000</td>
<td>82.29</td>
</tr>
<tr>
<td>6</td>
<td>69.5424</td>
<td>70.4228</td>
<td>56.9020</td>
<td>60.8279</td>
<td>46.0206</td>
<td>59.0849</td>
<td>75.86</td>
</tr>
<tr>
<td>7</td>
<td>68.6273</td>
<td>69.9662</td>
<td>46.0206</td>
<td>62.2789</td>
<td>62.9226</td>
<td>62.9226</td>
<td>80.95</td>
</tr>
<tr>
<td>8</td>
<td>68.6273</td>
<td>69.1273</td>
<td>52.0412</td>
<td>59.0849</td>
<td>53.9794</td>
<td>60.0000</td>
<td>80.22</td>
</tr>
<tr>
<td>9</td>
<td>68.2995</td>
<td>70.6040</td>
<td>40.0000</td>
<td>70.3703</td>
<td>49.5424</td>
<td>66.8485</td>
<td>83.64</td>
</tr>
<tr>
<td>10</td>
<td>68.6273</td>
<td>70.3703</td>
<td>58.0618</td>
<td>77.5012</td>
<td>49.5424</td>
<td>63.5218</td>
<td>83.44</td>
</tr>
<tr>
<td>11</td>
<td>68.9432</td>
<td>70.9309</td>
<td>60.0000</td>
<td>78.6900</td>
<td>46.0206</td>
<td>70.1030</td>
<td>82.09</td>
</tr>
<tr>
<td>12</td>
<td>67.6042</td>
<td>70.5526</td>
<td>46.0206</td>
<td>76.1236</td>
<td>49.5424</td>
<td>71.3640</td>
<td>85.15</td>
</tr>
<tr>
<td>13</td>
<td>68.2995</td>
<td>70.1030</td>
<td>53.9794</td>
<td>68.2995</td>
<td>52.0412</td>
<td>64.6090</td>
<td>80.36</td>
</tr>
<tr>
<td>14</td>
<td>67.9588</td>
<td>70.3703</td>
<td>56.9020</td>
<td>71.8213</td>
<td>65.1055</td>
<td>61.5836</td>
<td>77.54</td>
</tr>
<tr>
<td>15</td>
<td>68.9432</td>
<td>69.2180</td>
<td>58.0618</td>
<td>72.0412</td>
<td>63.5218</td>
<td>60.8279</td>
<td>82.33</td>
</tr>
<tr>
<td>16</td>
<td>68.2995</td>
<td>68.8496</td>
<td>55.5630</td>
<td>67.6042</td>
<td>64.0824</td>
<td>60.0000</td>
<td>80.61</td>
</tr>
<tr>
<td>17</td>
<td>68.6273</td>
<td>70.0212</td>
<td>59.0849</td>
<td>69.8272</td>
<td>64.0824</td>
<td>60.8279</td>
<td>75.29</td>
</tr>
<tr>
<td>18</td>
<td>69.2480</td>
<td>69.1273</td>
<td>60.0000</td>
<td>53.9794</td>
<td>64.6090</td>
<td>66.8485</td>
<td>70.63</td>
</tr>
<tr>
<td>19</td>
<td>67.6042</td>
<td>69.2480</td>
<td>61.5836</td>
<td>61.5836</td>
<td>46.0206</td>
<td>65.1055</td>
<td>82.28</td>
</tr>
<tr>
<td>20</td>
<td>67.6042</td>
<td>69.9109</td>
<td>60.0000</td>
<td>58.0618</td>
<td>46.0206</td>
<td>62.9226</td>
<td>80.53</td>
</tr>
<tr>
<td>21</td>
<td>68.2995</td>
<td>69.6575</td>
<td>62.9226</td>
<td>46.0206</td>
<td>59.0849</td>
<td>56.9020</td>
<td>72.87</td>
</tr>
<tr>
<td>Avg</td>
<td>68.2429</td>
<td>69.9922</td>
<td>56.6050</td>
<td>64.7995</td>
<td>54.1594</td>
<td>62.7365</td>
<td>79.9067</td>
</tr>
<tr>
<td>St Dev</td>
<td>0.6723</td>
<td>0.6226</td>
<td>6.9816</td>
<td>8.2071</td>
<td>7.3371</td>
<td>3.7297</td>
<td>3.9578</td>
</tr>
<tr>
<td>Max</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>78.6900</td>
<td>65.1055</td>
<td>71.3640</td>
<td>86.26</td>
</tr>
<tr>
<td>Min</td>
<td>66.8485</td>
<td>68.8496</td>
<td>40.0000</td>
<td>46.0206</td>
<td>46.0206</td>
<td>56.9020</td>
<td>70.63</td>
</tr>
</tbody>
</table>
2. A term containing knowledge of its personal ‘best’ position so far

This term, \( c_1 \text{rand} ( \text{PBEST} - X) \), applies a weight \( (c_1) \) to the distance of the variable from its best position in all previous iterations \( (\text{PBEST}) \). This term is analogous to the personal ‘memory’ of the particle with respect to its best position in the direction of the optimum position.

3. A term containing knowledge of the global ‘best’ position

This term, \( c_2 \text{rand} ( \text{GBEST} - X) \), applies a weight \( (c_2) \) to the distance of the particle from the global best position in all previous iterations \( (\text{GBEST}) \). This term is analogous to the ‘knowledge of the swarm’ with respect to the position of the particle relative to the best position found by the swarm.

The random number function, \( \text{rand}() \), is the uniform probability density function in Matlab and returns a value between 0 and 1. The \( \text{rand}() \) value is different for each term and is intended to represent the unpredictability in swarm behavior.

Increasing the weights of any of the weighting factors \( w, c_1 \) or \( c_2 \) will encourage the particle to explore the sample space in the direction represented by the associated term. Conversely, decreasing the weights of any of the weighting factors \( w, c_1 \) or \( c_2 \) will discourage the particle from exploring the sample space in the direction represented by the associated term. The inertia weight, \( w \), is generally assigned a value between 0 and 1. According to Robinson and Rahmat-Samii, a large inertia weight, \( w \), tends to encourage global exploration even as the swarm approaches the optimum solution and for this reason they adopt the strategy of linearly decreasing \( w \) as the optimization progresses, so that global exploration reduces as the solution converges to optimum. Robinson and Rahmat-Samii decrease \( w \) linearly from 0.9 to 0.4 over the course of the optimization. This strategy is also adopted in this work. The values of the weighting factors, \( c_1 \) and \( c_2 \), chosen by Robinson and Rahmat-Samii are based on analytical analysis of PSO by Clerc and Kennedy [9] which resulted in a value of 1.49 for each. Those values for the weighting factors will also be used here.

Handling Constraints

As discussed above, the constraints are the ranges of the data from which the regression equation (the fitness function) was derived. Particles that violate these constraints are considered to be “infeasible”. Inevitably through the course of PSO the calculated velocities, if not handled accordingly, will carry some or all particles beyond the constraints into the infeasible region. Robinson and Rahmat-Samii discuss three ways of handling constraint violations. The first is to ignore the constraints with respect to the position of the particle and to not evaluate the fitness for infeasible values of \( X \) (the “Invisible Wall”) effectively eliminating the influence of the particle on the swarm. The second is to reverse the velocity of any particle that might otherwise violate a constraint (the “Reflecting Wall”), and finally, to set the velocity equal to zero once the
particle achieves the constraint value (the “Absorbing Wall”). In this work, it is proposed that another new method (the “Arresting Wall”) be considered where the velocity is set to zero if the particle would otherwise violate a constraint. This new philosophy may allow particles to explore regions closer to the constraint boundary. These methods of handling constraint violations are illustrated in Figure 1.

One of the problems with PSO can be the tendency of the particles to fly out of the solution space never to return. The control on the constraints is one way to prevent this. Another method for minimizing constraint violation is to limit the velocity within each iteration. Eberhart and Shi [10] propose limiting the velocity by introducing a maximum velocity, $v_{\text{max}}$, to be applied at each iteration where $v_{\text{max}}$ is equal to $10\% - 20\%$ of the range of feasible values for each variable. Here, using a limiting factor of $10\%$ of the range, this strategy is applied directly to the constraints in Table 1 to give the values for $v_{\text{max}}$ for the optimization problem. These values are presented in Table 4.

Convergence to the Optimum Solution

As with any optimization method, PSO requires that the conditions for convergence be defined. Typically in an iterative optimization program convergence is defined to be the state when the optimum value has not changed, or has changed very little over the course of a predetermined number of iterations or CPU time. In PSO, however, there is a tendency for the optimization to stagnate temporarily, sometimes for 100 iterations or more depending on the choice of parameters and the character of the fitness function, constraints and the sample space. For this reason, the researcher must take great care when defining the conditions for convergence and the parameters within PSO. In this work the solution must remain unchanged for 1,000 iterations to consider PSO converged to a solution. The next section presents the details of the solution using PSO.

Numerical Solution of the Optimization Problem by the Particle Swarm Optimization Method

The optimization problem is solved with Matlab. A detailed algorithm is shown in the Appendix. Although the number of iterations required for convergence is greater than needed for CE, as discussed earlier, there is a tendency of PSO to stagnate occasionally and there is very little penalty in allowing the routine to continue for this number of iterations. The program requires the following input:

1. Whether to use the 21 samples from the assembly line as the number of particles or to generate any desired number of particles using the values from the 21 samples to define the sampling distributions.

2. Which constraint philosophy to use throughout the run (Invisible Wall, Reflecting Wall, Arresting Wall or Absorbing Wall)
Figure 1 – Handling Constraint Violations – Four Methods (adapted from Robinson and Rahmat-Samii [8], © 2004 IEEE)
Table 4 – Values of vmax

<table>
<thead>
<tr>
<th>Variable</th>
<th>vmax</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>0.2694</td>
</tr>
<tr>
<td>a5</td>
<td>0.2106</td>
</tr>
<tr>
<td>c6</td>
<td>3.1596</td>
</tr>
<tr>
<td>c7</td>
<td>3.2669</td>
</tr>
<tr>
<td>d3</td>
<td>1.9085</td>
</tr>
<tr>
<td>d6</td>
<td>1.4462</td>
</tr>
</tbody>
</table>

3. The number of iterations required for convergence.

4. The maximum number of iterations before concluding that PSO will not converge.

5. The choice to review the detailed calculations of each iteration as they happen (otherwise only the iteration number, the number of iterations toward convergence and the current solver time used are displayed).

The program provides the following output:

1. All of the parameters for PSO including the user-input.

2. The iteration at which the optimum value was identified.

3. The optimum solution.

4. A file of GBEST for each iteration (X and Y) (how the solution progressed).

5. The average PBEST for each iteration (X and Y) (although not of much interest).

6. The average value of X for each iteration (the general position of the swarm).

7. The solver time used for the iteration.

Table 5 summarizes the setup parameters that are used for the optimization. The detailed Matlab code is included as an Appendix. The optimization is conducted in two trials with five runs each.
Table 5 – Setup Parameters for PSO

<table>
<thead>
<tr>
<th>Description</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Velocity Inertia Weight</td>
<td>$w$</td>
<td>0.9 - 0.4</td>
</tr>
<tr>
<td>Personal Best Weight</td>
<td>$c_1$</td>
<td>1.49</td>
</tr>
<tr>
<td>Global Best Weight</td>
<td>$c_2$</td>
<td>1.49</td>
</tr>
<tr>
<td>Stall Iterations Required</td>
<td>iConverge</td>
<td>1000</td>
</tr>
<tr>
<td>Maximum Iterations</td>
<td>iMax</td>
<td>20000</td>
</tr>
<tr>
<td>Number of Particles - Trial 1</td>
<td>N</td>
<td>21</td>
</tr>
<tr>
<td>Number of Particles - Trial 2</td>
<td>N</td>
<td>100</td>
</tr>
</tbody>
</table>

Due to the rand() function, it can be expected that the routine will not produce identical results with any subsequent run, at least in terms of the number of iterations and solver time. For this reason each run of the program is repeated four times - five runs in total. Table 6 is a summary of the runs and solutions used for solving the optimization problem with PSO using 21 particles with the data from the assembly line as the initial position. Table 7 is the same summary using 100 particles with the initial position drawn randomly from PDFs defined by the 21 samples data from the assembly line (the data from Table 2) as the initial position.

It is clear that this optimization problem did not benefit from the Reflecting Wall and Arresting Wall constraint strategies. The Absorbing Wall strategy did provide an improvement in run-time efficient with significantly fewer iterations required. The method saw no significant improvement in performance using a larger number of particles based on the original 21 samples. Figure 2 is another representation of the optimum solutions presented in Tables 6 and 7 that perhaps further clarify these points. Figure 3 is an illustration of the number of iterations required to complete the optimization for each method. Figure 4 is the solver time required for the method to converge at each run. It can be concluded that the Absorbing Wall constraint philosophy using the initial samples to define the particles identified the optimum value more efficiently than the other constraint philosophies, even with a larger population of particles.

Summary of the Solution by the Particle Swarm Optimization Method

This section presented the solution of the optimization problem using the Particle Swarm Optimization method. This section now concludes with a comparison of the best performance of the Particle Swarm Optimization Method, the Cross-Entropy Method and the deterministic solution. Table 8 shows the comparison. PSO using the Absorbing Wall constraint philosophy agrees exactly with the deterministic solution to four decimal places. However, PSO required twice the CPU time and more than 20 times more iterations. It remains, for future work, to confirm that axles built to the optimum conditions indeed produce improved vibration.
performance. Other papers by the authors illustrate solving the same optimization problem using the Cross Entropy Method [5] and Particle Swarm Optimization [6].
Table 6– Solutions of the Optimization Problem by the Particle Swarm Optimization Method n = 21

<table>
<thead>
<tr>
<th>Run</th>
<th>Number of Particles</th>
<th>Constraint Philosophy</th>
<th>Iterations Required to Converge</th>
<th>Solver Time</th>
<th>a1</th>
<th>a5</th>
<th>c6</th>
<th>c7</th>
<th>d3</th>
<th>d6</th>
<th>Optimum Solution Y((X^*))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21</td>
<td>Invisible</td>
<td>5728</td>
<td>170</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>46.0482</td>
<td>65.1055</td>
<td>71.3640</td>
<td>58.1504</td>
</tr>
<tr>
<td>2</td>
<td>21</td>
<td>Invisible</td>
<td>7327</td>
<td>218</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>46.0206</td>
<td>65.1054</td>
<td>71.3640</td>
<td>58.1402</td>
</tr>
<tr>
<td>3</td>
<td>21</td>
<td>Invisible</td>
<td>5191</td>
<td>154</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>46.0206</td>
<td>65.1054</td>
<td>66.6384</td>
<td>59.3797</td>
</tr>
<tr>
<td>4</td>
<td>21</td>
<td>Invisible</td>
<td>4601</td>
<td>137</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>46.0206</td>
<td>65.1054</td>
<td>71.3640</td>
<td>58.1521</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>Invisible</td>
<td>4740</td>
<td>140</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>46.0206</td>
<td>65.1054</td>
<td>71.3640</td>
<td>58.1402</td>
</tr>
<tr>
<td>1</td>
<td>21</td>
<td>Reflecting</td>
<td>513</td>
<td>70</td>
<td>69.4846</td>
<td>70.3764</td>
<td>61.1302</td>
<td>46.1155</td>
<td>63.5278</td>
<td>66.5952</td>
<td>63.7399</td>
</tr>
<tr>
<td>2</td>
<td>21</td>
<td>Reflecting</td>
<td>222</td>
<td>30</td>
<td>69.2577</td>
<td>69.5087</td>
<td>60.8789</td>
<td>46.4431</td>
<td>65.0064</td>
<td>62.4810</td>
<td>67.6056</td>
</tr>
<tr>
<td>3</td>
<td>21</td>
<td>Reflecting</td>
<td>825</td>
<td>112</td>
<td>69.2236</td>
<td>70.9517</td>
<td>59.9873</td>
<td>48.2129</td>
<td>64.7831</td>
<td>67.1225</td>
<td>63.3564</td>
</tr>
<tr>
<td>4</td>
<td>21</td>
<td>Reflecting</td>
<td>444</td>
<td>60</td>
<td>69.4951</td>
<td>69.8167</td>
<td>59.8196</td>
<td>46.1429</td>
<td>64.2124</td>
<td>68.8239</td>
<td>64.7774</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>Reflecting</td>
<td>868</td>
<td>119</td>
<td>69.5391</td>
<td>70.2467</td>
<td>60.2865</td>
<td>46.1620</td>
<td>63.8339</td>
<td>68.7466</td>
<td>63.4770</td>
</tr>
<tr>
<td>1</td>
<td>21</td>
<td>Arresting</td>
<td>9538</td>
<td>284</td>
<td>69.5424</td>
<td>70.9534</td>
<td>60.5298</td>
<td>46.0378</td>
<td>64.7733</td>
<td>71.2373</td>
<td>60.4696</td>
</tr>
<tr>
<td>2</td>
<td>21</td>
<td>Arresting</td>
<td>18985</td>
<td>564</td>
<td>69.5424</td>
<td>70.9003</td>
<td>71.5957</td>
<td>46.5591</td>
<td>65.1054</td>
<td>68.2686</td>
<td>59.3053</td>
</tr>
<tr>
<td>3</td>
<td>21</td>
<td>Arresting</td>
<td>8841</td>
<td>263</td>
<td>69.5319</td>
<td>69.9192</td>
<td>71.5957</td>
<td>46.0206</td>
<td>64.3967</td>
<td>66.8889</td>
<td>62.4449</td>
</tr>
<tr>
<td>4</td>
<td>21</td>
<td>Arresting</td>
<td>8859</td>
<td>264</td>
<td>69.4087</td>
<td>70.9547</td>
<td>71.5957</td>
<td>46.0206</td>
<td>64.2124</td>
<td>66.7476</td>
<td>59.9944</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>Arresting</td>
<td>7656</td>
<td>228</td>
<td>69.4626</td>
<td>70.8970</td>
<td>61.2003</td>
<td>46.0206</td>
<td>65.1054</td>
<td>68.0458</td>
<td>61.4502</td>
</tr>
<tr>
<td>1</td>
<td>21</td>
<td>Absorbing</td>
<td>1402</td>
<td>42</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>46.0206</td>
<td>65.1055</td>
<td>71.3640</td>
<td>58.1402</td>
</tr>
<tr>
<td>2</td>
<td>21</td>
<td>Absorbing</td>
<td>1603</td>
<td>47</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>46.0206</td>
<td>65.1055</td>
<td>71.3640</td>
<td>58.1402</td>
</tr>
<tr>
<td>3</td>
<td>21</td>
<td>Absorbing</td>
<td>2124</td>
<td>64</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>46.0206</td>
<td>65.1055</td>
<td>71.3640</td>
<td>58.1402</td>
</tr>
<tr>
<td>4</td>
<td>21</td>
<td>Absorbing</td>
<td>2137</td>
<td>63</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>46.0206</td>
<td>65.1055</td>
<td>71.3640</td>
<td>58.1402</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>Absorbing</td>
<td>1041</td>
<td>31</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>46.0206</td>
<td>65.1055</td>
<td>71.3640</td>
<td>58.1402</td>
</tr>
</tbody>
</table>
Table 7 – Solutions of the Optimization Problem by the Particle Swarm Optimization Method n = 100

<table>
<thead>
<tr>
<th>Run</th>
<th>Number of Particles</th>
<th>Constraint Philosophy</th>
<th>Iterations Required to Converge</th>
<th>Solver Time</th>
<th>a1</th>
<th>a5</th>
<th>c6</th>
<th>c7</th>
<th>d3</th>
<th>d6</th>
<th>Optimum Solution Y(X*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>Invisible</td>
<td>5444</td>
<td>401</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>46.0206</td>
<td>65.1054</td>
<td>71.3640</td>
<td>58.1402</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>Invisible</td>
<td>7532</td>
<td>554</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>46.0206</td>
<td>65.1054</td>
<td>71.3640</td>
<td>58.1402</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>Invisible</td>
<td>5247</td>
<td>385</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>46.0206</td>
<td>65.1054</td>
<td>71.3640</td>
<td>58.1402</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>Invisible</td>
<td>5165</td>
<td>379</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>46.0206</td>
<td>65.1054</td>
<td>71.3640</td>
<td>58.1403</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>Invisible</td>
<td>5777</td>
<td>425</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>46.0206</td>
<td>65.1054</td>
<td>71.3640</td>
<td>58.1402</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>Reflecting</td>
<td>1795</td>
<td>132</td>
<td>69.5357</td>
<td>70.9494</td>
<td>70.8944</td>
<td>47.0921</td>
<td>64.9142</td>
<td>70.2936</td>
<td>61.2060</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>Reflecting</td>
<td>830</td>
<td>61</td>
<td>69.3352</td>
<td>70.3283</td>
<td>70.3594</td>
<td>46.0279</td>
<td>64.4072</td>
<td>70.2936</td>
<td>61.2060</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>Reflecting</td>
<td>126</td>
<td>9</td>
<td>68.9376</td>
<td>70.5637</td>
<td>64.0866</td>
<td>46.2584</td>
<td>60.3654</td>
<td>68.8608</td>
<td>64.5685</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>Reflecting</td>
<td>201</td>
<td>15</td>
<td>69.4146</td>
<td>70.2294</td>
<td>58.3638</td>
<td>49.4231</td>
<td>64.8848</td>
<td>64.1467</td>
<td>66.3538</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>Reflecting</td>
<td>118</td>
<td>9</td>
<td>69.3322</td>
<td>70.8433</td>
<td>61.6919</td>
<td>55.2147</td>
<td>60.6359</td>
<td>68.7659</td>
<td>66.4116</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>Arresting</td>
<td>7579</td>
<td>557</td>
<td>69.1517</td>
<td>70.9369</td>
<td>71.5957</td>
<td>46.0206</td>
<td>60.9479</td>
<td>63.8873</td>
<td>63.8873</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>Arresting</td>
<td>12844</td>
<td>943</td>
<td>69.5424</td>
<td>70.9532</td>
<td>71.5631</td>
<td>46.0206</td>
<td>58.0015</td>
<td>66.1168</td>
<td>61.6453</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>Arresting</td>
<td>1979</td>
<td>145</td>
<td>69.5424</td>
<td>70.9489</td>
<td>53.1735</td>
<td>46.0749</td>
<td>58.5073</td>
<td>61.7258</td>
<td>66.3095</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>Arresting</td>
<td>6552</td>
<td>482</td>
<td>69.8201</td>
<td>68.5383</td>
<td>68.5383</td>
<td>46.0206</td>
<td>65.1028</td>
<td>63.5838</td>
<td>64.5359</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>Arresting</td>
<td>1830</td>
<td>135</td>
<td>67.9886</td>
<td>70.9242</td>
<td>71.5957</td>
<td>50.2843</td>
<td>64.8387</td>
<td>63.6681</td>
<td>66.2609</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>Absorbing</td>
<td>1715</td>
<td>127</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>46.0206</td>
<td>65.1055</td>
<td>71.3640</td>
<td>58.1402</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>Absorbing</td>
<td>820</td>
<td>60</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>46.0206</td>
<td>65.1055</td>
<td>71.3640</td>
<td>58.1402</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>Absorbing</td>
<td>1119</td>
<td>83</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>46.0206</td>
<td>65.1055</td>
<td>71.3640</td>
<td>58.1402</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>Absorbing</td>
<td>1220</td>
<td>89</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>46.0206</td>
<td>65.1055</td>
<td>71.3640</td>
<td>58.1402</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>Absorbing</td>
<td>986</td>
<td>73</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>46.0206</td>
<td>65.1055</td>
<td>71.3640</td>
<td>58.1402</td>
</tr>
</tbody>
</table>
Figure 2 – Results of Particle Swarm Optimization by Run Number and Constraint Philosophy
Figure 3 – Iterations Required for Convergence of Particle Swarm Optimization (by Run Number and Constraint Strategy)
Figure 4 – Solver Time Required for Convergence of Particle Swarm Optimization (by Run Number and Constraint Strategy)
Appendix

A Detailed Algorithm for Particle Swarm Optimization

The implementation follows the algorithm shown below.

1. Initialize the particles (X and Y) from Table 3.
2. Initialize the constraints and vmax from Table 2 and Table 4.
3. Set the parameters w, c1 and c2.
4. Calculate the new velocities for each particle.
   a. Allow w to decrease from 0.9 to 0.4 in increments of 0.005 (100 iterations)
5. Calculate the new positions based on the new velocities.
6. Check feasibility.
7. Calculate the fitness value of each feasible solution using Equation (1).
8. Check each solution against the particle’s Personal Best (PBEST) and update PBEST if the new position yielded better results.
9. Check each solution against the Global Best (GBEST) and update if any of the solutions yielded better results.
10. Repeat from Step 4 until 1,000 iterations yield no change in GBEST.

The Matlab Code for Particle Swarm Optimization for Solving the Optimization Problem

```
% Revised 03/22/2013
```

---

Table 8 – A Comparison of the Deterministic Solution of the Optimization Problem to the Best Performance of CE

<table>
<thead>
<tr>
<th>Optimization Method</th>
<th>N</th>
<th>Iterations to Solve</th>
<th>Solver Time (sec)</th>
<th>a1</th>
<th>a5</th>
<th>c6</th>
<th>c7</th>
<th>d3</th>
<th>d6</th>
<th>Optimum Solution Y(X*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deterministic</td>
<td>0</td>
<td>0</td>
<td>&lt; 10</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>46.0206</td>
<td>65.1055</td>
<td>71.3640</td>
<td>58.1402</td>
</tr>
<tr>
<td>PSO - Absorbing Wall</td>
<td>100</td>
<td>820</td>
<td>60</td>
<td>69.5424</td>
<td>70.9555</td>
<td>71.5957</td>
<td>46.0206</td>
<td>65.1055</td>
<td>71.3640</td>
<td>58.1402</td>
</tr>
</tbody>
</table>
warning('off','all');
warning;

% Open the data files
O = load('nvhdata.mat', '-ASCII');
constraint = load('const.mat', '-ASCII');

% Get the range of the data and establish vmax
[ic,jc] = size(constraint);
j = 1;

while j < jc +1
    vmax(j) = 0.1 * (constraint(1,j) - constraint(2,j));
    j = j+1;
end

% Establish the number of variables (c) and the number of samples (r)
[r,c] = size(O);

% Make room for the iteration number in the arrays

c = c + 1;

% Initialize the parameters

usex = 0;
detail = 0;
n = 0;
const = 1;
usex = input('Enter ''1'' to use the original data as the particles. ');
if usex > 0
    n = 21;
end
if n == 0
    n = input('How many random samples shall we generate? ');
end
const = input('How shall constraints be handled?\n( 1 = invisible, 2 = reflecting, 3 = arresting, 4 = absorbing) ');
gbmax = input('How many iterations are required to establish convergence? ');
imax = input('What is the maximum number of iterations allowed? ');
detail = input('Enter ''1'' if you would like to see the detailed calculations. ');

% Default PSO velocity parameters
a1 = 1.49;
a2 = 1.49;
w = 0.9;

% Enable these input statements to allow changes to the PSO velocity parameters
% a1 = input('What weight shall be applied to the PBEST value? ');
% a2 = input('What weight shall be applied to the GBEST value? ');
% w = input('What weight shall be applied to the velocity? ');
% stop = input('What minimum change in Y you are looking for before stopping the program? ');
% stopn = input('Over how many iterations? ');

% Initialize the arrays and variables
winit = w;
rr1(n,c)=0;
rr2(n,c)=0;
X(n,c)=0;
XP(n,c)=0;
V(n,c)=0;
VP(n,c)=0;
PBEST(n,c)=0;
GBEST(c)=0;
GBESTS(n,c)=0;
AVERAGES(n,c)=0;
PAVERAGES(n,c)=0;
avfile(1,c)=0;
pavfile(1,c)=0;
gfile(1,c)=0;
gflag = 0;
pflag = 0;
count=1;
iopt=0;
opt=0;
write=0;
gbcount=1;
dcount=imax;
d4=('Agrees with Classical Solution? ');
agree={'No'};

% Initialize the counters for the datafile stats
err = 0;
x = 0;
px = 0;
i = 1;
j = 1;

% Calculate the averages

AVERAGES(1,1)=0;
while j < c
while i < r + 1
    x = x + O(i,j);
    i = i + 1;
end
Average(j) = x / r;
AVERAGE(j) = Average(j);
x = 0;
i = 1;
j = j + 1;
end
Average

% Calculate the Standard Deviations

i = 1;
j = 1;
while j < c
while i < r + 1
    err = err + (O(i,j) - Average(j))^2;
    i = i + 1;
end
StDev(j) = sqrt(err / (r-1));
err = 0;
i = 1;
j = j + 1;
end
StDev

% Initialize the optimization values PBEST and GBEST

Ymin = 10,000;
i = 1;
while i < n + 1
    X(i,1) = i;
    XP(i,1)=i;
    PBEST(i,1)=i;
    GBEST(1)=i;
    i = i + 1;
end
i = 1;
j = 1;
if r < n
    ri = r;
else


ri = n;
end

while i < ri + 1
while j < c

% Initialize PBEST

X(i,j + 1) = O(i,j);
XP(i,j + 1) = O(i,j);
PBEST(i,j + 1) = O(i,j);
j = j + 1;
end

% Initialize GBEST

if X(i,8) == 0 || X(i,8) < Ymin
    Ymin = X(i,8);
    jj = 1;
    while jj < c + 1
        GBEST(jj) = X(i,jj);
        jj = jj + 1;
    end
end
j = 1;
i = i + 1;
end

% Write GBESTS to an array

gfile(1,1)=0;
j = 2;
while j < c + 1
    GBESTS(count,j)=GBEST(j);
j = j + 1;
end

% Write the initial run

if detail == 1
    iteration = count - 1
    X
    XP
    V
    VP
    PBEST
    GBEST
end

%
% Begin Particle Swarm Optimization
%

count = count + 1;

while count < imax + 2
    GBESTS(count,1)=count - 1;
    AVERAGES(count,1)=count - 1;
    PAVERAGES(count,1)=count -1;
    if detail == 1
        iteration = count - 1
    end
    i = 1;
    confail = 0;

    while i < n + 1
        j = 2;
        while j < 8
            % Generate the velocity and position for each particle
            % Calculate the position for each particle
            % If random samples will be the particles, generate the initial
            % 'n' sample positions
            if count == 2 && usex < 1
                X(i,j) = random('norm', Average(j-1), StDev(j-1));
            else
                % Calculate the new velocity for each particle
                r1(i,j) = rand(1);
                r2(i,j) = rand(1);

                if w > 0.4
                    w = w - 0.005;
                end
                %a1 = 2;
                %a2 = 2;
                %w = 0.9;

                V(i,j) = w * VP(i,j) + a1 * r1(i,j) * (PBEST(i,j) - XP(i,j)) + a2 * r2(i,j) * (GBEST(j) - XP(i,j));

            end
        end
    end

end
% Don't let the velocity exceed vmax

if abs(V(i,j)) > vmax(j-1)
    V(i,j) = V(i,j) / abs(V(i,j)) * vmax(j-1);
end

% Calculate the new position for each particle
X(i,j) = XP(i,j) + V(i,j);

% Check the feasibility of the new position at the upper boundary

if X(i,j) + V(i,j) > constraint(1,j-1)

    % The "invisible wall" philosophy
    if const == 1
        di = {'Invisible'};
        %display(di)
    end

    % The "reflecting wall" philosophy
    if const == 2
        X(i,j) = XP(i,j) - V(i,j)/i;
        di = {'Reflecting'};
        %display(di)
    end

    % The "arresting wall" philosophy
    if const == 3
        X(i,j) = XP(i,j);
        di = {'Arresting'};
        %display(di)
    end

    % The "absorbing wall" philosophy
    if const == 4
        X(i,j) = constraint(1,j-1);
        di = {'Absorbing'};
        %display(di)
    end

end
% Check the feasibility of the new position at the lower boundary
if X(i,j) + V(i,j) < constraint(2,j-1)

% The "invisible wall" philosophy
if const == 1
    di = {'Invisible'};
    %display(di)
end

% The "reflecting wall" philosophy
if const == 2
    X(i,j) = XP(i,j) - V(i,j)/i;
end

% The "arresting wall" philosophy
if const == 3
    X(i,j) = XP(i,j);
end

% The "absorbing wall" philosophy
if const == 4
    X(i,j) = constraint(2,j-1);
end

end

if X(i,j) > constraint(1,j-1) || X(i,j) < constraint(2,j-1)
    confail = 1;
end

end

j = j +1;
end

% Calculate the fitness value
if confail == 0
    X(i,8) = -2.8049 * X(i,2) - 2.789 * X(i,3) - 0.19745 * X(i,4) + 0.36987 * X(i,5) - 0.29785 * X(i,6) - 0.2623 * X(i,7) + 486.32;
else
    X(i,8) = 1,000;
end
% Check if this is the Personal Best

    if PBEST(i,8) == 0 || X(i,8) < PBEST(i,8)
        if X(i,8) > 0
            pflag = 1;
            pi = i;
        end
    end

% Check if this is the Global Best

    if X(i,8) < Ymin
        if X(i,8) > 0
            Ymin = X(i,8);
            if iopt < 1
                if X(i,8) > 58.14 && X(i,8) < 58.1405
                    iopt = count -1;
                    agree = {'Yes'};
                end
            end
            gflag = 1;
            gi = i;
        end
    end

    i = i + 1;
end

% Write the Personal Best values for each iteration of the optimization

if pflag == 1
    j = 2;
    while j < c + 1
        PBEST(pi,j) = X(pi,j);
        j = j + 1;
    end

pflag = 0;
end

% Write the Global Best Value for each iteration of the optimization

if gflag == 1
    j = 1;
    while j < c + 1
        GBEST(j) = X(gi,j);
        j = j + 1;
    end

gflag = 0;
% Update GBESTS to include this iteration's Global Best

j = 2;
while j < c + 1
    GBESTS(count,j)=GBEST(j);
    j = j + 1;
end

% Calculate the averages for X and PBEST

i = 1;
j = 1;

while j < c
    while i < r + 1
        x = x + X(i,j + 1);
        px = px + PBEST(i,j+1);
        i = i + 1;
    end
    AVERAGES(count, j + 1) = x / r;
    PAVERAGES(count, j + 1) = px / r;
    x = 0;
    px = 0;
    i = 1;
j = j + 1;
end

% Record the X and V values as the previous values for the next iteration

i = 1;
j = 1;

while i < n + 1
    while j < c + 1
        XP(i,j) = X(i,j);
        VP(i,j) = V(i,j);
        j = j + 1;
    end
    j = 1;
i = i + 1;
end

% Display the results for this iteration (if desired)
if detail == 1
    rr1
    rr2
    X
    V
    XP
    VP
    PBEST
    GBESTS
end

if GBESTS(count,8) == GBESTS(count-1,8)
    gbcoun= gbcoun+1;
else
    gbcoun= 1;
end

if gbcoun >= gbmax

    display('Optimization has converged.')
dcount = count;
opt = dcount - gbmax;
count = imax + 2;
else

if GBESTS(count,8) < 0.0000

    display('The solution is negative.')
dcount = count;
count = imax + 2;
else

if count == imax +2

    display('The maximum allowable iterations has been reached.')
end
end
end
end

count = count + 1;
clc
Iteration = count - 2
gbcount
Optimum = GBEST(8)
end
%
%
%   End of Particle Swarm Optimization
%
%
%   Write the results

i = 1;
j = 1;

while i < dcount + 1
    while j < c + 1
        avfile(i,j) = AVERAGES(i,j);
pavfile(i,j) = PAVERAGES(i,j);
gfile(i,j) = GBESTS(i,j);
j = j + 1;
end
j = 1;
i = i + 1;
end
%
%   Display the results

%AVERAGES;
%GBESTS;
%avfile
%pavfile
gfile
display('Agrees with classical solution?')
display(agree);
Iteration = dcount - gbmax
Optimum = GBESTS(opt + 1,8)
%gbcount
%imax
wall = di
%
%   Write to a spreadsheet?

write = input('Enter ''1'' to write this data to a spreadsheet. ');

if write == 1
    dl = {'usex', 'n', 'iopt', 'gbmax', 'imax',
          'const', 'vmax1', 'vmax2', 'vmax3', 'vmax4', 'vmax5', 'vmax6', 'w-init', 'w-end', 'a1', 'a2'};
d2 = ['i123456Y'];
d3 = {'Optimum','Iteration','Agrees with Classical?'};
xlsxwrite('PSO_Solution.xls',d1,'Parameters','A1');
xlsxwrite('PSO_Solution.xls',usex,'Parameters','A2');
xlsxwrite('PSO_Solution.xls',n,'Parameters','B2');
xlsxwrite('PSO_Solution.xls',iopt,'Parameters','C2');
xlsxwrite('PSO_Solution.xls',gbmax,'Parameters','D2');
xlsxwrite('PSO_Solution.xls',imax,'Parameters','E2');
xlsxwrite('PSO_Solution.xls',const,'Parameters','F2');
xlsxwrite('PSO_Solution.xls',vmax,'Parameters','G2');
xlsxwrite('PSO_Solution.xls',winit,'Parameters','M2');
xlsxwrite('PSO_Solution.xls',w,'Parameters','N2');
xlsxwrite('PSO_Solution.xls',a1,'Parameters','O2');
xlsxwrite('PSO_Solution.xls',a2,'Parameters','P2');
xlsxwrite('PSO_Solution.xls',d3,'Parameters','A3');
xlsxwrite('PSO_Solution.xls',Optimum,'Parameters','A4');
xlsxwrite('PSO_Solution.xls',Iteration,'Parameters','B4');
xlsxwrite('PSO_Solution.xls',agree,'Parameters','C4');
xlsxwrite('PSO_Solution.xls',di,'Parameters','F3');
xlsxwrite('PSO_Solution.xls',d2,'GBESTS','A1');
xlsxwrite('PSO_Solution.xls',d2,'PBESTS','A1');
xlsxwrite('PSO_Solution.xls',d2,'X-Averages','A1');
xlsxwrite('PSO_Solution.xls',gfile,'GBESTS','A2');
xlsxwrite('PSO_Solution.xls',pavfile,'PBESTS','A2');
xlsxwrite('PSO_Solution.xls',avfile,'X-Averages','A2');

end

% End of test. Ask to clear the memory.

reply = input ('Do you want to clear everything? (y/n)[n]', 's');
if reply == 'y'
    clear
clc
elseif isempty(reply)
    reply = 'n';
end

Bibliography


