ASSIGNMENT 1

AER1415: Computational Optimization

Atilla Saadat Dehghan
atilla.saadat@mail.utoronto.ca
February 25th 2021

1. Particle Swarm Optimization (PSO) Algorithm Implementation and Testing

1.1 Algorithm Description

The proposed PSO algorithm is designed using Python 3 (available on Github), utilizing the object-oriented nature of python to efficiently design the particle class definitions. The PSO algorithm is divided into two object classes, one intended to invoke a the general PSO algorithm, and the Particle class which defined the N number of particles created for the swarm. The PSO algorithm utilizes an iterative approach to minimizing an objective function, using the notion (and varying) a particles Position, x, and Velocity, v. Specifically, x_i^k and v_i^k define a certain particle's positions and velocity of the ith particle at the kth time step, respectively. The position of the ith particle is updated at iteration $k+1$ as follows:

$$\boldsymbol{x}_i^{k+1} = \boldsymbol{x}_i^k + \boldsymbol{v}_i^{k+1}\Delta t$$

$$\boldsymbol{v}_i^{k+1} = \omega \boldsymbol{v}_i^k + c_1 r_1 \frac{\boldsymbol{p}_i^k - \boldsymbol{x}_i^k}{\Delta t} + c_2 r_2 \frac{\boldsymbol{p}_g^k - \boldsymbol{x}_i^k}{\Delta t}$$

Where:

$r_1, r_2 \sim UNIFORM[0,1]$

$p_i^k -$ best position of particle i

$p_g^k -$ swarm best particle position

$c_1 -$ cognitive parameter

$c_2 -$ social parameter

$\omega -$ inertial weight

The various particle objects are defined inside the general PSO class. For each particle generated, the new position and cost function values are calculated. From the various particles, the global best is updated within the PSO class, based on the value of the new particle value, to be passed onto later particle calculations. This is iterated many times until the max iteration number is reached.

The PSO algorithm has 3 different bounding methods integrated for testing, however one was ultimately utilized for ease of parameter tuning combinations and as a result of the conclusions in [1]. The input bounds for a given problem is given by: $x_{min} \leq x \leq x_{max}$

Specifically, in [1] it was found that the reflect bounding method "appears to be the overall best-performing candidate". The other methods explored in the development of this algorithm includes the Hyperbolic method and the traditional Nearest Method, defined in Table 1.

Table 1: Bounding Methods integreated in the PSO Algorithm

Reflect:	If $x_{t+1} > x_{max}$ then $x'_{i,t+1} = x_{max} - (x_{t+1} - x_{max})$ If $x_{t+1} < x_{min}$ then $x'_{i,t+1} = x_{min} + (x_{min} - x_{t+1})$
Hyperbolic:	If $v_{i,t+1} > 0$: $v_{i,t+1} = \frac{v_{i,t+1}}{1 + \left\lvert \frac{v_{i,t+1}}{x_{i,max} - x_{i,t}} \right\rvert}$ else: $v_{i,t+1} = \frac{v_{i,t+1}}{1 + \left\lvert \frac{v_{i,t+1}}{x_{i,t} - x_{i,min}} \right\rvert}$
Nearest:	If $x_{i,t+1} > x_{i,max}$ then $x'_{i,t+1} = x_{i,max}$ If $x_{i,t+1} < x_{i,min}$ then $x'_{i,t+1} = x_{i,min}$

Using a selected bounding method, the PSO can be called with any given objective function outside of the PSO class object. This allows the PSO algorithm to be highly modular with respect to the input cost function. The various cost functions were defined with an imbedded quadratic penalty function. This allows the cost function to accept and consider constraints and ensures that they are maintained throughout the calculations. This results in the PSO algorithm solving for a new objective function:

$$\pi(x, \alpha) = f(x) + \rho_k \phi(x)$$

where ρ_k is a positive penalty function parameters and α is the penalty function constant coefficient. The quadratic penalty function is given by:

$$\phi(x) = \sum_{i=1}^{m} \max[0, g_i(x)]^2 + \sum_{i=1}^{q} h_i(x)^2$$

where $g_i(x)$ and $h_i(x)$ are the inequality and equality constraints, respectively, that the function is subject to, which have the forms:

$$g_i(x) \leq 0\ , i = 1,2, \dots m$$

$$h_i(x) = 0\ , i = 1,2, \dots q$$

The penalty function parameter, ρ_k, needs to grow to ∞ as the max iteration number is reached within the PSO algorithm. As stated in [2, p. 498], by driving ρ_k to ∞, we penalize the constraint violations with increasing severity. Therefore, it was chosen that the penalty function will have the form

$$\rho_k = \alpha^k$$

so that ρ_k exhibits an exponentially increate behaviour towards infinity as a function of iteration number. This also entails that α is another parameter that required rough tuning.

Therefore, the PSO algorithm's attempts to minimize:

$$\pi(x, \alpha), x \in \mathbb{R}^n$$

The PSO algorithm is also executed with each parameter for an M number of independent runs, where the mean and standard deviation of the all the results is returned. The best solution (the positions and the cost function value) are thus the mean of the M runs, giving a better indication of the performance of the PSO algorithm with the given parameters. For all subsequent analysis, M will be set to 10. Similarly, the PSO algorithm will run for 100 iterations and define the number of particle numbers with

$$Particle_{num} = \max[200, 10i]$$

where i is the number of positions or estimated state variables.

1.2 Parameter tuning studies

To select appropriate PSO parameters for use in the case studies listed in Section 1.3, a preliminary parameter tuning study was conducted on the 2D P4 Bump Test function. As stated in Section 1.1, the PSO algorithm has several tuning parameters that can be varied in many ways. The chosen tuning parameter sets are listed as follows:

Table 2: PSO Tuning Parameter Sets

Set #	1	2	3	4	5	6	7	8	9	10	11	12
ω	0.3	0.3	0.3	0.5	0.5	0.5	0.7	0.7	0.7	0.9	0.9	0.9
c_1	0.5	1.5	3.5	0.5	1.5	3.5	0.5	1.5	3.5	0.5	1.5	3.5
c_2	3.5	1.5	0.5	3.5	1.5	0.5	3.5	1.5	0.5	3.5	1.5	0.5
r_{tol}	1e-9	1e-9	1e-9	1e-9	1e-9	1e-9	1e-9	1e-9	1e-9	1e-9	1e-9	1e-9
α	5	5	5	5	5	5	5	5	5	5	5	5
b_{method}	Reflect	Reflect	Reflect	Reflect	Reflect	Reflect	Reflect	Reflect	Reflect	Reflect	Reflect	Reflect

This set allows for a distributed range (or selected grid search) of the important tuning parameters: ω, c_1, and c_2. Through initial testing, the penalty function coefficient, α, yielded minimal impact into the results, given that α is sufficiently large so that p_k approached near ∞ (relative to the iteration range). The bounding function method, b_{method}, chosen to use the Reflect method exclusively as it prevents division by zero errors for certain cost functions (especially when the bounds include a zero value) and its superior performance noted in [1]. Therefore, α and b_{method} were omitted from the grid search. This yields a manageable combination of tuning parameters that can be analyzed and considered. P4 Bump Test with n=2 yields the following results for these parameter sets, shown in Figure 2, which shown the mean and convergence trends for the M independent runs executed on a given parameter set.

The relative tolerance, r_{tol}, set to $1E-9$ defined the threshold for which to evaluate when the PSO algorithm has converged to a certain value. This is done by comparing the discrete derivative of the values and the set relative tolerance, defined as

$$\frac{d(\pi(x,\rho))}{dk} \leq r_{tol}$$

It was determined that this method was not sufficient to determine the best parameter set as the stochastic nature of the PSO algorithm could yield small differentials without yet converging to its minimum value with continued iterations.

An analysis on the nature of the penalty function was also conducted to determine which method provides better optimization performance. Specifically, the comparison between a static and quadratic penalty function (defined in Section 1.1) was performed, shown in Figure 1. Although it initially seems that the static penalty function outperforms the quadratic penalty function, the lower cost function value from the static does not properly comply with the constraints of P4, while the quadratic method does. Therefore, for the remaining PSO algorithm calculations, the quadratic penalty function will be utilized.

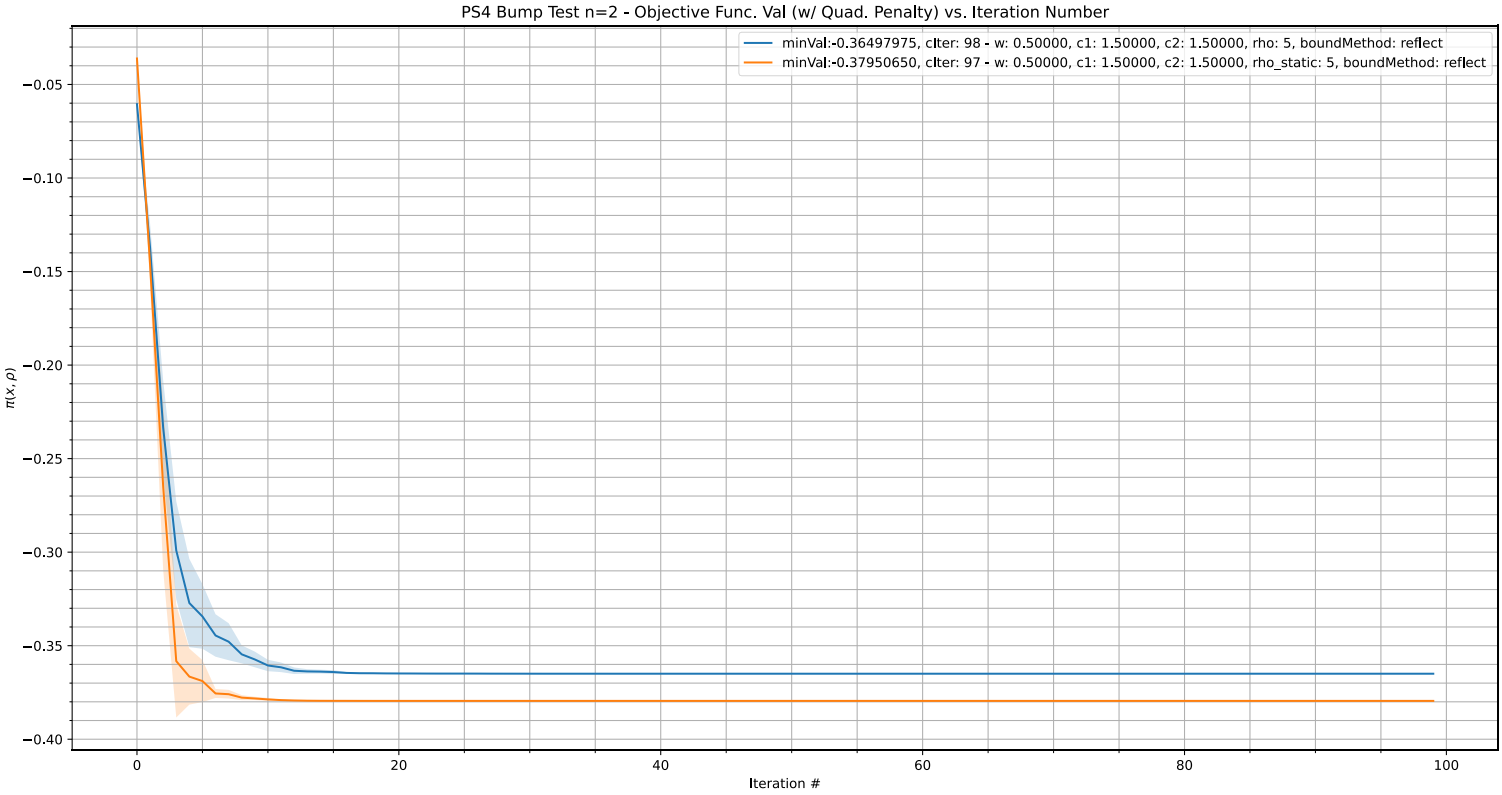

Figure 1: Comparison between Static and Quadratic Penalty function performance

To choose the best resulting parameter set, a visual inspection of the Figure 2 was conducted so that the choice can be optimized based on both the final minimum value and the iterations taken to reach the minimum value. This is done by choosing the mean line corresponding to the line that reached near the overall minimum value that all the sets reach, at the lowest iteration number. This process allowed Set #5, defined in in Table 2, to be chosen at the best parameter set for the rest of the case study functions. A 3D visualization of this Set #5 is also shown in Figure 3. The graph shows that the PSO algorithm settled inside the largest minimum lobe but does not reach the base cost function's global minimum. This is intended, as the inequality

constraints set on P4 would not allow the PSO algorithm to reach the global minimum (which would violate the set constraints).

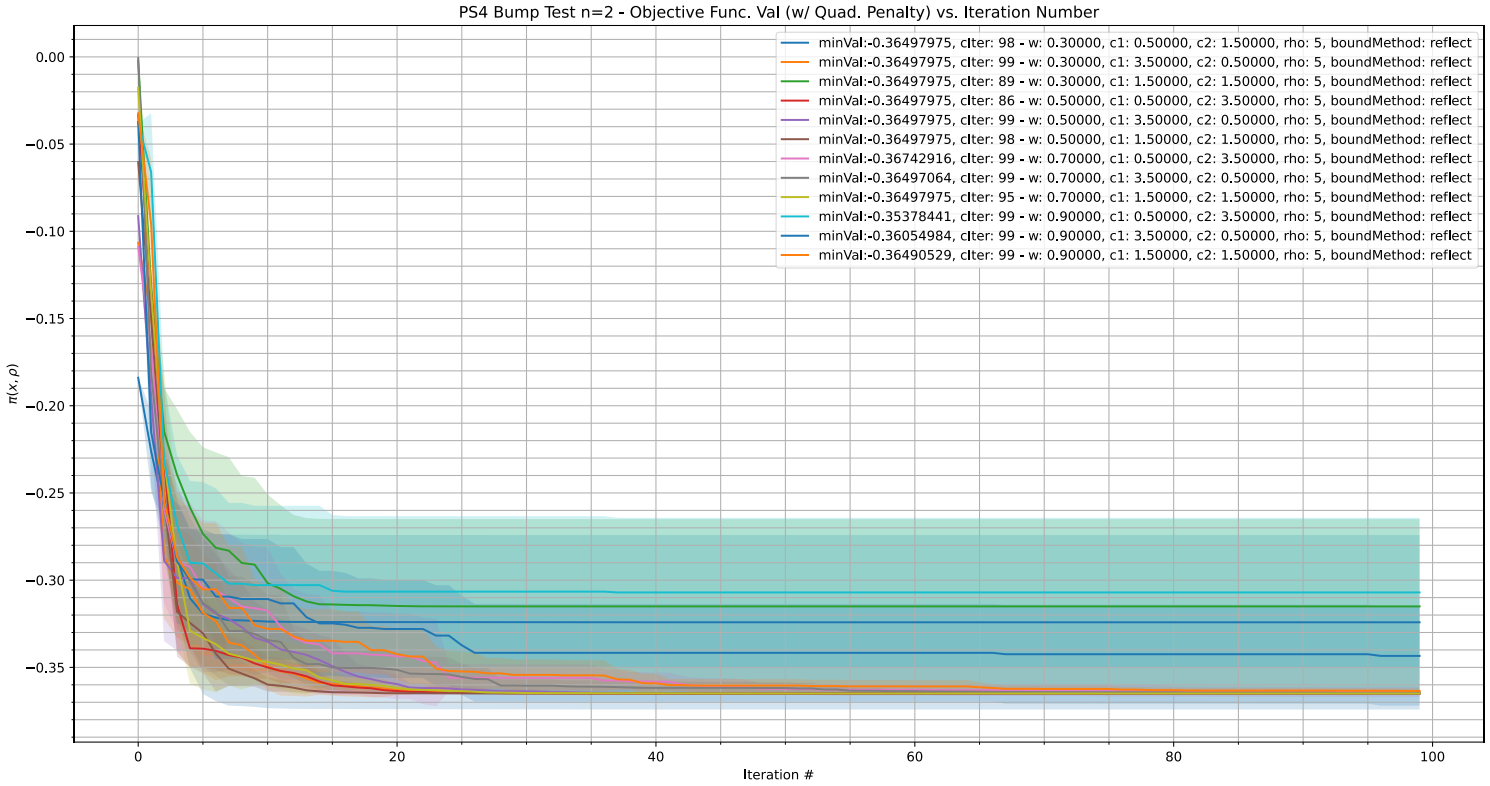

Figure 2: P4 Bump Test n=2 Cost Function results

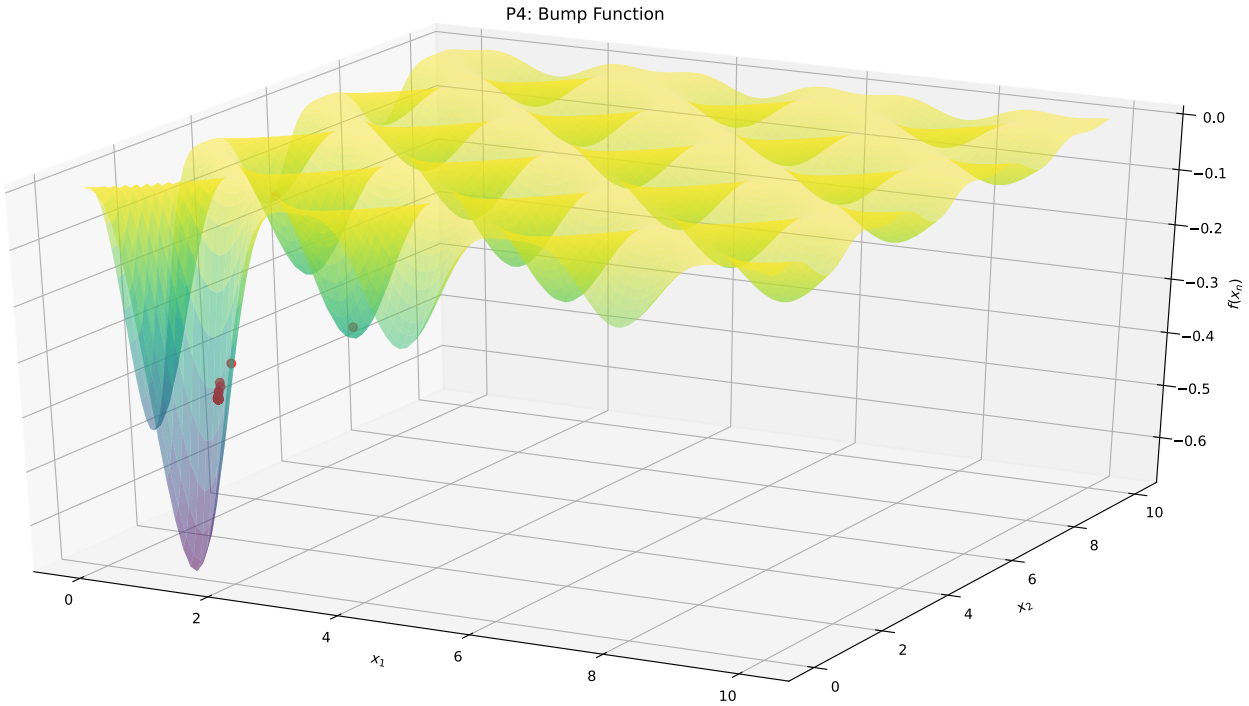

Figure 3: 3D Visualization - P4 Bump Test n=2

1.3 Case Study Results

The following tables list the resulting values from the PSO algorithm for P1-5 list in [3], using Parameter Set #5 as stated in Section 1.2:

Table 3: PS4 Bump Function best mean solution - n=2

x^*	1.60086041	0.46849806
$f(x^*)$	-0.364980	

Table 4: PS4 Bump Function Solution – n=10

x^*	3.17223546	3.01939868	2.98765697	0.89166832	0.64345997
	0.5351772	0.62587748	0.50429997	0.55324293	0.4887965
$f(x^*)$	-0.717130533				

Table 5: P1 Rosenbrock test function solution - n=2

x^*	1.0	1.0
$f(x^*)$	0.0000	

Table 6: P1 Rosenbrock test function - n=5

x^*	1.0	1.0	1.0	1.0	1.0
$f(x^*)$	0.0000				

Table 7: P2 Rastrigin test function - n=2

x^*	-2.65276426e-09	3.20007356e-10
$f(x^*)$	0.0000	

Table 8: P2 Rastrigin test function - n=5

x^*	1.55669952e-09	-9.94958638e-01	-9.94958637e-01	-9.94958639e-01	3.05506992e-09
$f(x^*)$	2.984877				

Table 9: P3 Constrained optimization problem - n=2

x^*	-0.80645161	0.20967741
$f(x^*)$	-0.040323	

In order to linearize the dynamic response problem in P5, certain formulations must be made to modify it to fit the PSO algorithm. The measured response is modeled by an ordinary differential equation, which is analytically modeled with the equations:

$$u(t) = (A\cos(\omega_d t) + B\sin(\omega_d t))\exp(-\frac{ct}{2m}) + \frac{F_0}{C}\cos(\omega t - \alpha),$$

where

$$A = -\frac{F_0}{C}\cos(\alpha), B = -\frac{F_0}{C\omega_d}\left(\omega\sin(\alpha) + \frac{c}{2m}\cos(\alpha)\right),$$

$$C = \sqrt{(k - m\omega^2)^2 + (c\omega)^2}, \omega_d = \sqrt{\frac{k}{m} - \left(\frac{c}{2m}\right)^2}, \alpha = tan^{-1}\left(\frac{c\omega}{k - m\omega^2}\right)$$

$$m = 1.0, F_0 = 1, \omega = 0.1$$

Since the time and measured displacement data is given from the input dataset, problem transforms into a minimization problem between the calculated displacement from the set estimated parameters, c and k, and the measured displacement (u_m). To properly numerate this difference, the root mean squared error between the estimated and measured response will be used.

Therefore, the cost function to be minimized becomes:

$$f(x) = \sqrt{\frac{\sum_{i=1}^{N}\left(u_i(x) - u_{m,i}\right)^2}{N}}, \quad x \in \mathbb{R}^N$$

Optimizing this cost function yields the estimated response from the calculated c and k values given in Table 10.

Table 10: P5 Dynamic Response problem - n=2

x^*	0.799734	10.203439
$f(x^*)$	0.000867	

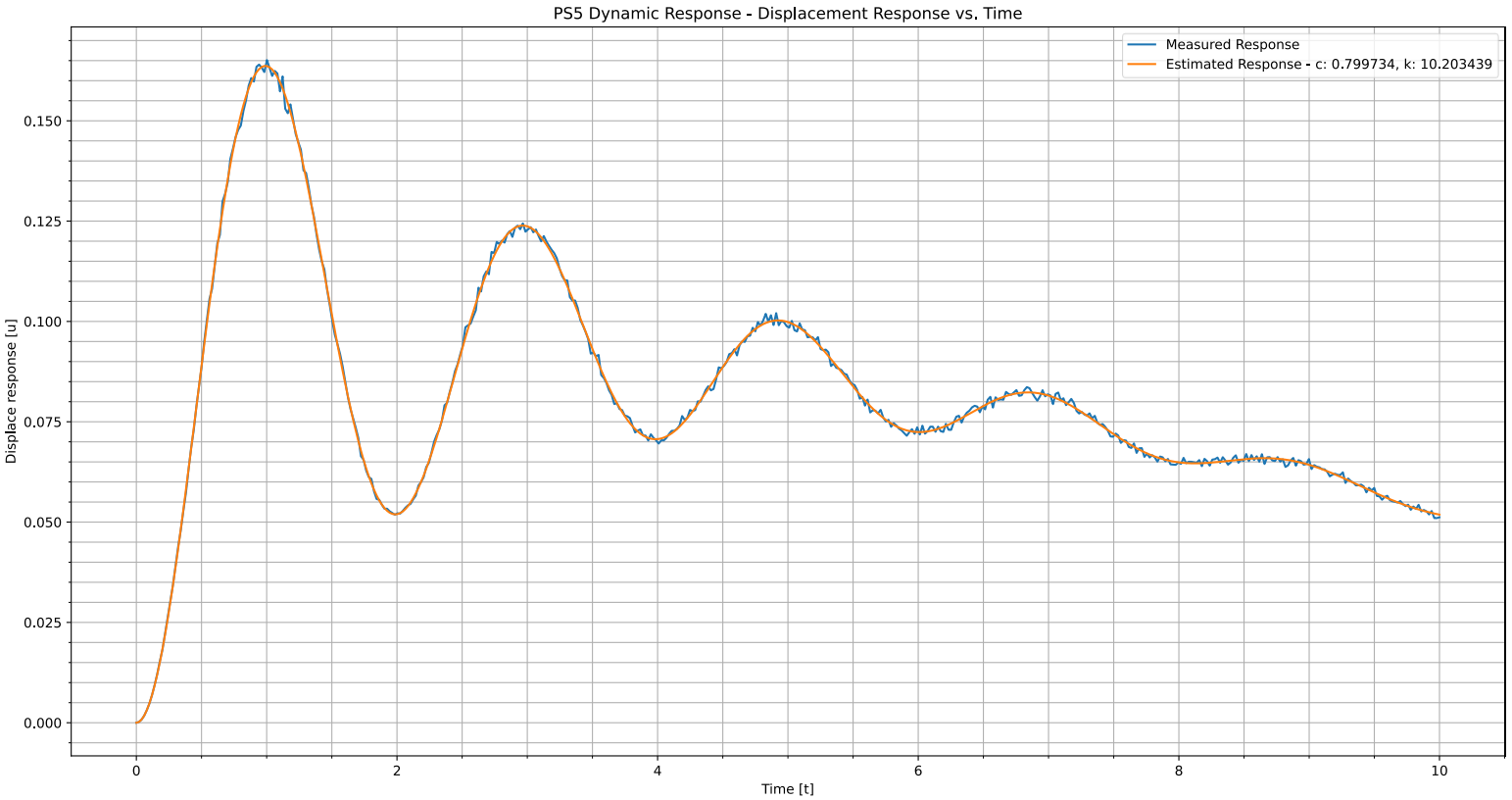

Figure 4: Measured and Estimated response with PSO-based c and k values

1.4 Conclusions

From the case study results shown in Section 1.3, it is evident that the developed PSO algorithm is working properly and effectively to produce the global minimum of the given test function within the set constraints and bounds. Using the selected parameter set, Set #5, deemed an effective set to use for the all of the cost functions analyzed. However, ideally parameter tuning would be done on a problem-by-problem basis, as demonstrated by the results in Table 8. The Rastrigin test function had a global minimum of $f(x^*) = 0$ at $x^* = (0, \dots, 0)$, while the results show that positions do not exactly results in near-zero values. This could be a by-product of many things, such a as the selected parameter set, the number of iterations set, etc. Through experimentation, it was deemed that the problem is solvable with the PSO algorithm using a custom parameter set. Therefore, the developed PSO algorithm is deemed perform well with a wide range of cost functions and is proven to be robust and highly adaptable to hyperparameter tuning analysis.

2.0 Question 2

Considering the generalized linear model

$$\hat{f}(x, w) = w_0 + \sum_{i=1}^{D} w_i \phi_i(x)$$

where $x \in \mathbb{R}^D$, $w_i, i = 0,1,2, \dots, D$ are the undetermined weight of the model, and $\phi_i: \mathbb{R}^D \rightarrow \mathbb{R}$ are the known basis functions.

Introducing a dummy feature $x_0 = 1$, x can be rewritten as $x = \{x_0, x_1, \dots, x_D\}^T \in \mathbb{R}^{D+1}$. Therefore, the linear model can be expressed as

$$\hat{f}(x, w) = w^T x$$

Defining matrix $X \in \mathbb{R}^{N \times (D+1)}$ whose ith row contains $x^i \in \mathbb{R}^{D+1}$ or $X_{ij} = c_i x_j^i$. $\therefore \hat{y} = Xw \in \mathbb{R}^N$

Let $\alpha = diag(\lambda_1, \dots, \lambda_N)$

Considering the ℓ_2 regularized weighted least-squares error function:

$$\begin{aligned} \mathcal{L}(w) &= \sum_{i=1}^{N} c_i \left(\hat{f}\left(x^i, w\right) - y^i\right)^2 + \lambda_i \sum_{i=1}^{M} w_i^2 \\ &= (Xw - y)^T (Xw - y) + \alpha w^T w \end{aligned}$$

To minimize and solve for the linear algebraic equations for the weights, we differentiate $\mathcal{L}(w)$ w.r.t w and set it to zero

$$\begin{aligned} \frac{\partial \mathcal{L}(w)}{\partial w} &= \frac{\partial}{\partial w}(w^T X^T X w - 2y^T X w + y^T y + \alpha w^T w) \\ 0 &= 2X^T X w - 2X^T y + 0 + aI \end{aligned}$$

$$\therefore w = (X^T X + \alpha I)^{-1} X^T y$$

3.0 Question 3

Consider the following Larangian function:

$$L(\mu) = \min\{c^T x + \mu^T (g(x) - \delta) \mid x \in X\}$$

where μ is the Lagrange multiplier, used to relax the explicit linear constraints by driving them to the objective function.

Since $Ax = b$ for all feasible solutions of x, for any μ: $x^* = \min\{c^T x \mid g(x) - \delta, x \in X\} = \min\{c^T x + \mu^T (g(x) - \delta) | g(x) \leq \delta, x \in X\}$. If the $g(x) \leq \delta$ constraint is removed, the second

formula would not lead to an increase in the on the objective function, $\therefore x^* \geq \min\{c^T x + \mu^T(g(x) - \delta) | \, x \in X\} = L(\mu)$

To obtain the finest lower bound possible, the following optimization problem must be solved:

$$L^* = \max\big(L(\mu)\big)$$

This optimal solution, L^*, is a lower bound of x^*, $\therefore L^* \leq x^*$ as $L(\mu) \leq L^* \leq c^T x$

From this, x might not be an optimal solution of the primal problem even if x is feasible for the primal and if x achieves the optimal of the $L^* = L(\mu)$ for some $\mu \geq 0$.

Therefore, if $L(\mu)$ is achieved by vector x, such that 1) x is feasible and 2) x satisfies $\mu^T(g(x) - \delta) = 0$, then $L(\mu) = L^*$ and $x = x^*$.

4.0 Bonus

Running P4 on n=50 yields (saved in ps4_n50_solution.txt) with params:

PS4 N=50 Soln:

x* = [3.12911892 3.27760746 3.27073039 2.40032953 0.71314415 3.15712693

2.54692409 3.16640317 0.95935472 3.18189302 0.40568397 2.67471495

0.62553678 0.96486652 2.9963939 0.34258747 1.20888377 2.97827777

3.0565148 1.05344658 2.92343469 0.87513606 1.45964505 3.02881275

0.40495849 2.32604255 0.414675 0.50826805 2.59109798 0.58787612

0.34688281 0.65225734 0.42637078 1.73494188 0.46406096 0.44716499

0.61610953 1.12894 0.31457955 0.64056126 0.27538871 0.45197841

0.47213751 0.6534026 1.29564226 0.45902522 0.74571736 0.31461258

0.43555552 0.55503499]

f(x*) = -0.600257

References

[1] S. Helwig, J. Branke, and S. Mostaghim, "Experimental Analysis of Bound Handling Techniques in Particle Swarm Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 2, pp. 259–271, 2013.

[2] J. Nocedal and S. J. Wright, *Numerical optimization*. New York, NY: Springer, 2006.

[3] P. B. Nair, AER1415 Computational Optimization: List of Case Studies, *University of Toronto,* 2021.