2014 International Joint Conference on Neural Networks (IJCNN)
July 6-11, 2014, Beijing, China

Particle Swarm Optimization for Convolved Gaussian Process Models

Gang Cao, Edmund M-K Lai, Fakhrul Alam
School of Engineering and Advanced Technology
Massey University, Albany
Auckland, New Zealand
Email: {gcao,e.lai,f.alam}@massey.ac.nz

***Abstract*—Convolved Gaussian process (CGP) is a type Gaussian process modelling technique applicable for multiple-input multiple-output systems. It employs convolution processes to construct a covariance function that models the correlation between outputs. Modelling using CGP involves learning the hyperparameters of the latent function and the smoothing kernel. Conventionally, learning involves the maximization of the log likelihood function of the training samples using conjugate gradient (CG) or particle swarm optimization (PSO) methods. We propose to use PSO to minimize the model error. In this way, a clearer direct indication of the quality of the current solution during the optimization process can be obtained. Simulation results on a dynamical system show that our method is able to learn appropriate CGP models and achieve better predictive performance compared with CG when the searching space is not well defined.**

I. Introduction

Gaussian Process (GP) modelling is a well established data driven technique that has been successfully applied to many areas of science and engineering [1], [2]. A major advantage of GP is that it involves far less parameters compared with parametric models such as artificial neural networks (ANN) and fuzzy models. These parameters, also called hyperparameters, are obtained through a learning process using a training input-output dataset. While GP modelling for multiple-input single-output (MISO) systems is quite well established, they are not easily extendable to multiple-input multiple-output (MIMO).

A GP model is specified by a covariance function that captures the characteristics of the training dataset. This function has to be semi-definite. For MIMO systems, it must capture the dependencies of all the observations by the auto-covariance as well as the cross-covariance. While there are many suitable choices of auto-covariance functions for MISO systems, it is not clear how such a covariance function can be determined for MIMO problems [3]. One approach is to construct the covariance function using convolution processes [4]. This idea was first implemented using Dependent Gaussian Process Models (DGPs) [5], where each output is the linear convolution of a smoothing kernel (filter) and Gaussian white noises (latent functions). Convolved Gaussian Process Models (CGPs) generalize DGPs by allowing latent functions other than Gaussian white noise [6], [7]. The hyperparameters of the smoothing kernel and latent function are learnt from the training dataset. Inferences are made in the same way as standard GP models.

The hyperparameters can be obtained by maximizing the log likelihood function. This optimization is commonly solved using conjugate gradient (CG) methods. However, CG can get stuck in local optimal instead of reaching the global optimal. Particle Swarm Optimization (PSO) has been used as an alternative to CG to overcome this problem [8], [9]. In these works, the fitness (objective) function being optimized is still the log likelihood function. In this paper, we propose an alternative way to learn the hyperparameters of a CGP model. Instead of maximizing the log likelihood function, we minimize the error of the output produced by the model. The optimization is performed through a PSO algorithm with the mean squared error of the predicted and actual outputs as its fitness function. Simulation results on a dynamical system modelling problem suggest that our approach can be used to learn CGP models effectively. The advantage of our method is that it provides a clearer direct indication of the quality of the current solution during the optimization process.

The rest of this paper is organized as follows. Section II provides a brief overview of CGPs. In Section III, we discuss our PSO algorithm for CGP hyperparameters learning. Simulation results are then presented in Section IV to demonstrate the effectiveness of our method. Finally, Section V concludes the paper.

II. Convolved Gaussian Process Models

A. Modelling

For a system with N inputs and M outputs, let $\mathbf{x} \in \mathbb{R}^N$ and $\mathbf{y}(\mathbf{x}) \in \mathbb{R}^M$ denote the vector-valued input and output respectively. In CGP, each output $y_d(\mathbf{x})$ is modelled as

$$y_d(\mathbf{x}) = f_d(\mathbf{x}) + \epsilon_d(\mathbf{x}) \tag{1}$$

for $d = 1, 2, \ldots, M$, where $\epsilon_d(\mathbf{x})$ is an independent Gaussian white noise. The function $f_d(\mathbf{x})$ is the linear convolution of a smoothing kernel $H_d(\mathbf{x})$ and a latent function $u(\mathbf{x})$:

$$f_d(\mathbf{x}) = \int H_d(\mathbf{x} - \tau) u(\tau) d\tau \tag{2}$$

The correlation between outputs are derived from the latent function $u(\mathbf{x})$ which influences on all output functions. Basically, this latent function can be any suitable random process. If it is a Gaussian white noise process, then we have a

978-1-4799-1484-5/14/$31.00 ©2014 IEEE

DGPs model. CGPs allows a wider choice of latent functions to match the modelling requirements of different physical or dynamical systems [7].

In general, more than one latent function can be used. Let there be Q groups of latent functions and the q^{th} group has R_q smoothing kernels. Then the d^{th} output function would become

$$f_d(\mathbf{x}) = \sum_{q=1}^{Q}\sum_{k=1}^{R_q}\int H_{d,q}^{k}(\mathbf{x}-\tau)u_q^k(\tau)d\tau \tag{3}$$

The covariance between output $y_d(\mathbf{x})$ and $y_{d'}(\mathbf{x}')$ can be expressed as:

$$\begin{aligned}\mathbf{cov}\left[y_d(\mathbf{x}), y_{d'}(\mathbf{x}')\right] &= \mathbf{cov}\left[f_d(\mathbf{x}), f_{d'}(\mathbf{x}')\right] + \\ &\quad \mathbf{cov}\left[\epsilon_d(\mathbf{x}), \epsilon_{d'}(\mathbf{x}')\right]\delta_{d,d'}\end{aligned} \tag{4}$$

where $\delta_{d,d'}$ is the Kronecker delta function. The cross-covariance between $f_d(\mathbf{x})$ and $f_{d'}(\mathbf{x}')$ is given by (5). Thus, data modelling using CGPs basically involves obtaining the appropriate smoothing kernels and latent functions that reflect the covariance of the outputs.

The output function in (3) is a linear combination of independent random functions. Hence if these functions are Gaussian processes, then $f_d(\mathbf{x})$ will also be a Gaussian process. In this case, the smoothing kernels are given by

$$H_{d,q}^{k}(\gamma) = \frac{\nu_{d,q}^{k}\left|\mathbf{P}_{d,q}^{k}\right|^{1/2}}{(2\pi)^{M/2}}\mathbf{exp}\left[-\frac{1}{2}(\gamma)^T\mathbf{P}_{d,q}^{k}(\gamma)\right] \tag{6}$$

where $\nu_{d,q}^{k}$ is a length-scale coefficient, $\mathbf{P}_{d,q}^{k}$ is an $N\times N$ precision matrix for the smoothing kernel. To simplify the model further, the covariance between latent functions in each group, denoted $k_q(\eta)$ is assumed the same and is Gaussian:

$$k_q(\eta) = \frac{\upsilon_q\left|\mathbf{P}_q\right|^{1/2}}{(2\pi)^{M/2}}\mathbf{exp}\left[-\frac{1}{2}(\eta)^T\mathbf{P}_q(\eta)\right] \tag{7}$$

where υ_q is the length-scale coefficient, $\mathbf{P}_{d,q}^{k}$ and $\mathbf{P}_q$ are $N\times N$ precision matrices.

In this paper, we shall assume that $R_q = 1$ for all Q groups of latent functions. Moreover, the precision matrices of the smoothing kernels are assumed to be the same for each group of latent functions. Under this assumption, given the smoothing kernel (6) and latent function covariance (7), the covariance of the outputs can be expressed as:

$$\begin{aligned}\mathbf{K}_{\mathbf{f_d},\mathbf{f_{d'}}}(\mathbf{x},\mathbf{x}') &= \mathbf{cov}\left[f_d(\mathbf{x}), f_{d'}(\mathbf{x}')\right] \\ &= \sum_{q=1}^{Q}\frac{\nu_{d,q}\nu_{d',q}\upsilon_q}{(2\pi)^{M/2}\left|P\right|^{1/2}}\mathbf{exp}\left[-\frac{1}{2}(\mathbf{x}-\mathbf{x}')^T\mathbf{P}^{-1}(\mathbf{x}-\mathbf{x}')\right]\end{aligned} \tag{8}$$

where $\mathbf{P} = \mathbf{P_d^{-1}} + \mathbf{P_{d'}^{-1}} + \mathbf{P_q^{-1}}$. Note that this multiple-output covariance function maintains a Gaussian form, i.e. $\mathbf{K}_{\mathbf{f_d},\mathbf{f_{d'}}}(\mathbf{x},\mathbf{x}')\sim\mathcal{N}(\mathbf{x}-\mathbf{x}'|\mathbf{0},\mathbf{P})$.

B. Inference

Similar to standard GP models, given the set of observations $\{\mathbf{x}_j,\mathbf{y}_j\}_{j=1}^{J}$, a Gaussian process prior can be defined for the output functions, i.e.

$$\mathbf{f}(\mathbf{x}) \sim \mathcal{GP}(\mu(\mathbf{x}), \mathbf{K}_{\mathbf{f},\mathbf{f}}(\mathbf{x},\mathbf{x}')) \tag{9}$$

where $\mathbf{f}(\mathbf{x}) = [\mathbf{f_1}(\mathbf{x}), \mathbf{f_2}(\mathbf{x}), ..., \mathbf{f_M}(\mathbf{x})]^{\mathbf{T}}$ and has entries $\mathbf{f_d}(\mathbf{x}) = [\mathbf{f_d}(\mathbf{x_1}), \mathbf{f_d}(\mathbf{x_2}), ..., \mathbf{f_d}(\mathbf{x_J})]^{\mathbf{T}}$. Without loss of generality, we assume that these are zero mean processes. The covariance matrix $\mathbf{K}_{\mathbf{f},\mathbf{f}}$ is a block partitioned matrix with block entries given by (8). Computational complexity for the covariance matrix is high because $\mathbf{K}_{\mathbf{f},\mathbf{f}} \in \mathbb{R}^{NJ\times NJ}$ is high dimensional. Sparse approximations have been proposed to reduce the complexities of CGPs [6].

The key to using (8) to model the data is to correctly learn the hyperparameters Θ. Approaches for learning these hyperparameters is the main topic of this paper and will be discussed in subsequent sections. Given the hyperparameters, a prior distribution (the marginal likelihood) is given by

$$p(\mathbf{y}|\mathbf{X},\Theta) \sim \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K}_{\mathbf{f},\mathbf{f}} + \mathbf{\Sigma}) \tag{10}$$

where $\mathbf{\Sigma}$ is a diagonal matrix of noise variances $\left\{\boldsymbol{\sigma}_d^2\right\}_{d=1}^{M}$. The observed outputs $\mathbf{y}$ and the predicted outputs $\mathbf{y}^*$ for an input $\mathbf{x}^*$ under the Gaussian prior have the following joint distribution:

$$\begin{bmatrix}\mathbf{y}\\ \mathbf{y}^*\end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix}\mathbf{K}_{\mathbf{f},\mathbf{f}} + \boldsymbol{\Sigma} & \mathbf{K}_{\mathbf{f},\mathbf{f}^*}\\ \mathbf{K}_{\mathbf{f}^*,\mathbf{f}} & \mathbf{K}_{\mathbf{f}^*,\mathbf{f}^*}\end{bmatrix}\right) \tag{11}$$

The predictive distribution is Gaussian with mean $\hat{\boldsymbol{\mu}}(\mathbf{x}^*)$ and variance $\hat{\mathbf{var}}(\mathbf{x}^*)$. Thus,

$$\mathbf{y}^*|\mathbf{X},\mathbf{y},\mathbf{x}^* \sim \mathcal{N}(\hat{\boldsymbol{\mu}}(\mathbf{x}^*), \hat{\mathbf{var}}(\mathbf{x}^*)) \tag{12}$$

$$\hat{\boldsymbol{\mu}}(\mathbf{x}^*) = \mathbf{K}_{\mathbf{f}^*,\mathbf{f}}\left[\mathbf{K}_{\mathbf{f},\mathbf{f}} + \mathbf{\Sigma}\right]^{-1}\mathbf{y} \tag{13}$$

$$\hat{\mathbf{var}}(\mathbf{x}^*) = \mathbf{K}_{\mathbf{f}^*,\mathbf{f}^*} - \mathbf{K}_{\mathbf{f}^*,\mathbf{f}}\left[\mathbf{K}_{\mathbf{f},\mathbf{f}} + \mathbf{\Sigma}\right]^{-1}\mathbf{K}_{\mathbf{f},\mathbf{f}^*} \tag{14}$$

where $\mathbf{y}^T = [(y_{11}...y_{1J_1})\,(y_{21}...y_{2J_2})\,...\,(y_{M1}...y_{MJ_M})]$.

$$\begin{aligned}\mathbf{cov}\left[f_d(\mathbf{x}), f_{d'}(\mathbf{x}')\right] &= E\left[\sum_{q=1}^{Q}\sum_{k=1}^{R_q}\int H_{d,q}^{k}(\mathbf{x}-\tau)u_q^k(\tau)d\tau\sum_{q=1}^{Q}\sum_{k=1}^{R_q}\int H_{d',q}^{k}(\mathbf{x}'-\tau')u_q^k(\tau')d\tau'\right]\\ &= \sum_{q=1}^{Q}\sum_{k=1}^{R_q}k_q(\tau,\tau')\int H_{d,q}^{k}(\mathbf{x}-\tau)H_{d',q}^{k}(\mathbf{x}'-\tau)d\tau\end{aligned} \tag{5}$$

C. Learning

From the Bayesian viewpoint, learning a convolved Gaussian process model involves computing the predictive distribution of a new target value $\mathbf{y}^*$,

$$\mathbf{P}\left(\mathbf{y}^*|\mathbf{X},\mathbf{y},\mathbf{x}^*\right)=\int \mathbf{P}\left(\mathbf{y}^*|\Theta,\mathbf{X},\mathbf{y},\mathbf{x}^*\right)\mathbf{P}\left(\Theta|\mathbf{X},\mathbf{y}\right)d\Theta \tag{15}$$

given a set of observations $(\boldsymbol{x},\boldsymbol{y})$ and a new test value $\mathbf{x}^*$. In most situations, this integral is analytically intractable. A possible solution is to use numerical integration approaches such as Monte-Carlo methods. However, the computational burden is very high.

An alternative approach is to perform maximum likelihood optimization where the log of the marginal likelihood, given by

$$\mathbf{L}\left(\mathbf{\Theta}\right)=-\frac{1}{2}\mathbf{y^T}\mathbf{K}_{\mathbf{y},\mathbf{y}}^{-\mathbf{1}}\mathbf{y}-\frac{1}{2}\mathbf{log}\left|\mathbf{K}_{\mathbf{y},\mathbf{y}}\right|-\frac{\mathbf{MJ}}{2}\mathbf{log2}\pi \tag{16}$$

is the cost function for optimization. Here, $\mathbf{K}_{\mathbf{y},\mathbf{y}}=\mathbf{K}_{\mathbf{f},\mathbf{f}}+\mathbf{\Sigma}$. The set of hyperparameters $\mathbf{\Theta}$ can then be obtained by using conjugate gradient (CG) optimization. However, it is well known that CG techniques produce results that are very dependent on the initial values for cost functions that have many local optima. Thus the optimization often needs to be repeated with different initial values. Evolutionary optimization techniques can be used to overcome this shortcoming.

III. PSO for CGPs Learning

Particle Swarm Optimization (PSO) is an evolutionary computation technique inspired by the social behaviour of organisms [10]. Many particles are initialized simultaneously where each individual one represents a solution to the problem. A particle then moves around in the search space and optimizes itself according to its own experience as well as that of the whole population. A fitness function is used to evaluate each particle and only 'qualified' particles survive in the competition. After several iterations, with the accumulation of "experience", the optimal or near optimal solution can be obtained.

PSO has previously been used to replace CG as a more robust optimization method for maximizing the log-likelihood function given by (16) [8], [9]. The experimental results in both of these papers indicate that PSO is able to avoid getting stuck in local optima and finds the near-optimal values.

In this paper, we use PSO to obtain the hyperparameters in a more direct way. Instead of maximizing the log-likelihood function, we seek to minimize the model error. This model error can be measured by the mean squared error (MSE) between the actual output and that produced from the model. MSE is a common and reasonable measure of assessing predictive performance when the number of observations is large enough. Given a current set of hyperparameters, the covariance matrix $\mathbf{K}_{\mathbf{y},\mathbf{y}}$ of the outputs can be obtained. This covariance matrix is then used to compute the predicted outputs using (13). For a system with multiple outputs, the MSE values of each individual output are calculated independently. This learning approach is similar to that for training an ANN. The advantage is that it is able to provide a clearer direct indication of the quality of the current solution during the optimization process.

Algorithm 1 CGPs Learning using PSO

Require:
Particles Θ, PSO parameters;
Ensure:
while $t<\mathbf{T_{max}}$ and $err(t-1)>\mathbf{Err}$ **do**
  for $i=1\rightarrow N_p$ **do**
    Evaluation particles fitness:$f(\mathbf{x}_i^t)$;
    if $f(\mathbf{x}_i^t)>f(\mathbf{pbest}_i^{t-1})$ **then**
      $\mathbf{pbest}_i^t=\mathbf{pbest}_i^{t-1}$
    else
      $\mathbf{pbest}_i^t=\mathbf{x}_i^t$
    end if
    if $f(\mathbf{pbest}_i^t)>f(\mathbf{gbest}^{t-1})$ **then**
      $\mathbf{gbest}^t=\mathbf{pbest}_i^t$
    end if
    Update Particles:$\mathbf{x}_i^{t+1}$ and $\mathbf{v}_i^{t+1}$
  end for
  iteration = $t+1$;
  Compute $err(t)$;
end while
Output: the optimized particle $\mathbf{\Theta}_{opt}$.

The PSO algorithm used here is shown in Algorithm 1. Each particle Θ is given by

$$\Theta=\{\Theta_{K1},...,\Theta_{KM},\Theta_{L1},...,\Theta_{LQ},\} \tag{17}$$

where $\Theta_{Kd}=\{\nu_{d1},\nu_{d2},...,\nu_{dQ},\mathbf{P_d}\}$ for $d=1,\ldots,M$ are the hyperparameters of the smoothing kernels in (6), and $\Theta_{Lq}=\{v_q,\mathbf{P_q}\}$ for $q=1,\ldots,Q$ are the hyperparameters of the latent function covariances in (7). In each iteration, the fitness value of all N_p particles are evaluated. The position $\mathbf{x}_i^t$ and velocity $\mathbf{v}_i^t$ of each particle is then updated according to local and global best 'experience' given by $\mathbf{pbest}_i^t$ and $\mathbf{gbest}^t$ respectively.

At the beginning of the process, we generally want better global search ability and therefore a larger inertia factor is preferred. Towards the end of optimization, a smaller inertia factor will provide better local search ability. This is achieved by using a nonlinear time-varying inertia factor:

$$\omega^t=\omega_{end}+(\omega_{start}-\omega_{end})\mathbf{exp}(-k\times(\frac{t}{\mathbf{T}_{max}})) \tag{18}$$

where ω_{start} and ω_{end} are pre-determined start and final values of the inertia factor respectively. k is a factor controlling the shape of the equation and $\mathbf{T_{max}}$ is the maximum number of iterations. The algorithm terminates when the maximum number of iterations $\mathbf{T_{max}}$ is reached or the model error $err(\cdot)$ falls below a preset threshold $\mathbf{Err}$. $\mathbf{\Theta}_{opt}$ denotes the final solution.

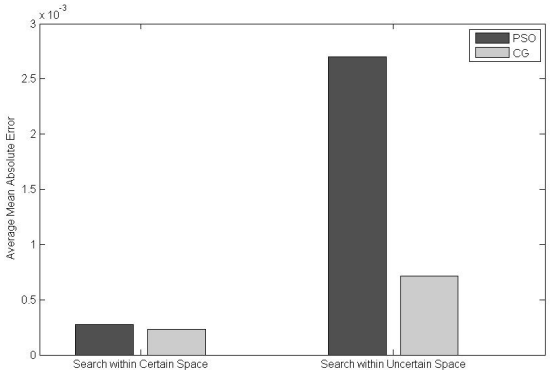


Fig. 1: Average MAE for single-output dynamical system modelling.

TABLE I: Comparison of two PSO approaches with different population sizes

N_p	MAE		Var	
	PSO/1	PSO/2	PSO/1	PSO/2
10	0.2297	0.2355	1.52e-02	2.44e-02
25	0.0054	0.0047	9.05e-03	3.64e-03
50	0.0022	0.0021	8.66e-03	4.37e-03
100	0.0011	0.0012	9.14e-03	9.74e-04

IV. Simulation Results

A. Single-Output Modelling

The modelling of a non-linear dynamical system is used to evaluate the effectiveness of our approach to learning CGPs. This system is described by the following difference equation:

$$\begin{aligned} y(k) &= 0.893y(k-1) + 0.037y^2(k-1) \\ &\quad -0.05y(k-2) + 0.157u(k-1) \\ &\quad -0.05u(k-1)y(k-1) \end{aligned} \tag{19}$$

where $u(k)$ is the input and $y(k)$ is the output. Although this dynamical system has only one input and one output, the inputs to the CGPs will be $u(k-1)$, $y(k-1)$ and $y(k-2)$, making it a three-input model. 1000 observations are generated using random inputs $u \sim \mathbf{U}(-2, 4)$. For the noise term in (1), zero-mean Gaussian white noise with unit variance is used.

In the first experiment, we aim to verify that our approach in using PSO to optimize the model error is effective. Here, PSO/1 denotes our approach and PSO/2 denotes using PSO to maximize the log-likelihood function. Only a single output is used here for modelling to simplify the comparison. Ten simulations are performed for each population size with the same 200 training and 50 test data from the 1000 observations randomly generated. Table I shows the results in terms of mean absolute error (MAE) and average variance (Var). Overall, both approaches result in models that produce similar MAE and variances in the predicted outputs. Moreover the results suggest that a population of 25 to 50 is a suitable choice for this problem.

In the second experiment, we want to determine the effect of the size of hyperparameters search space. Two different cases are considered here. In the first case, we assume that we have prior knowledge of the range of values for the parameters in (17). More specifically, they are

$$\begin{aligned} \nu_{d,i}, \upsilon_q &\in \{1, 2\} \\ \alpha_i, \beta_j &\in \{0, 1\} \end{aligned} \tag{20}$$

where α_i and β_j are the elements of the diagonal precision matrices $\mathbf{P_d}$ and $\mathbf{P_q}$ respectively. In the second case, we do not assume any prior knowledge of the range of values. The average MAE obtained using our PSO algorithm and those using CG are shown in Figure 1 over 10 independent simulations. The results show that when the search space is constrained, models obtained using PSO and CG both perform equally well. Figure 2a confirms that the predicted results are very close to the actual values. However, when the search space is unconstrained, PSO outperforms CG by a wide margin. Figure 2b shows that while the output predicted by the PSO model is still very close to the actual values, there are some clear deviations with the CG model.

TABLE II: Results of Linear Relationship

	MAE		SE	
	PSO	CG	PSO	CG
$\mathbf{y}_1$	1.41E-04	4.46E-04	0084	0.0327
$\mathbf{y}_2$	1.11E-04	2.18E-04	0.147	0.0446

TABLE III: Results of Nonlinear Relationship

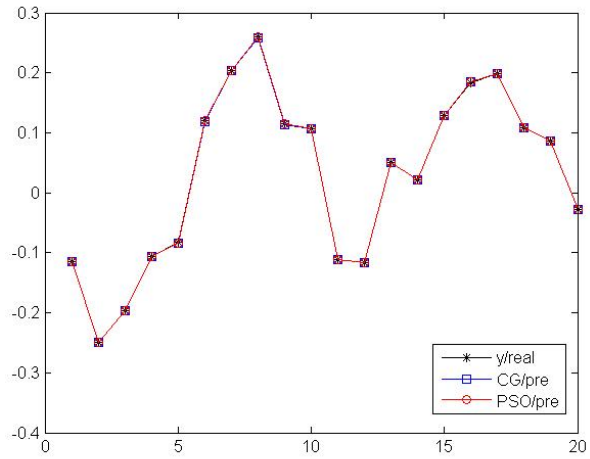
	MAE		SE	
	PSO	CG	PSO	CG
$\mathbf{y}_1$	4.41E-04	5.36E-04	0.0065	0.043
$\mathbf{y}_2$	3.57E-04	8.11E-04	0.0064	0.0356

B. Two-output Modelling

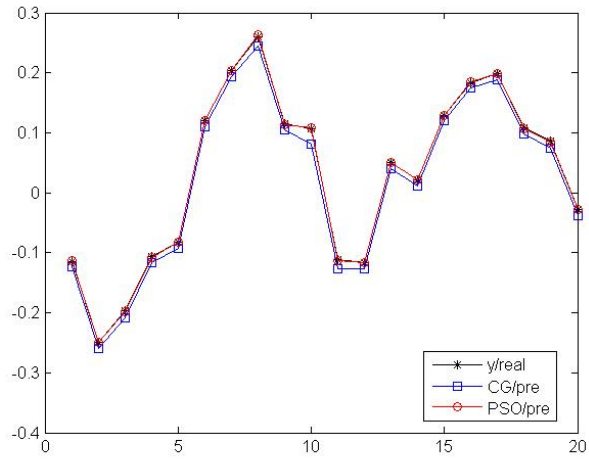
Systems with multiple-outputs can be modelled in two different ways. One is to use multiple single-output models and the other is to provide a single model for all outputs at the same time. While the first approach is often simpler, the latter approach is able to capture correlation between outputs. For example, a robot arm system with multiple degrees of freedom has multiple outputs that are strongly correlated. Another example is the prediction of steel mechanical properties in [11], where the yield and tensile strength are predicted from the chemical compositions and grain size. These two "outputs" are highly correlated.

We shall continue to use the dynamical system in (19). Since it has only one output y (denoted y_1 here), a second output y_2 will be created which is a function of y_1. Two such functions are considered, one linear and the other nonlinear, given by $\mathbf{y}_2 = -\mathbf{y}_1$ and $\mathbf{y}_2 = \mathbf{exp}(\mathbf{y}_1)$ respectively. Two different sets of data, each has 200 samples, are selected from the 1000 randomly generated observations for training. The test data consists of 50 samples which are different from the training samples.

Tables II and III show the performances of using PSO and CG, averaged over 10 simulations. The same results are also shown in Figure.3 with an indication of the deviations. For

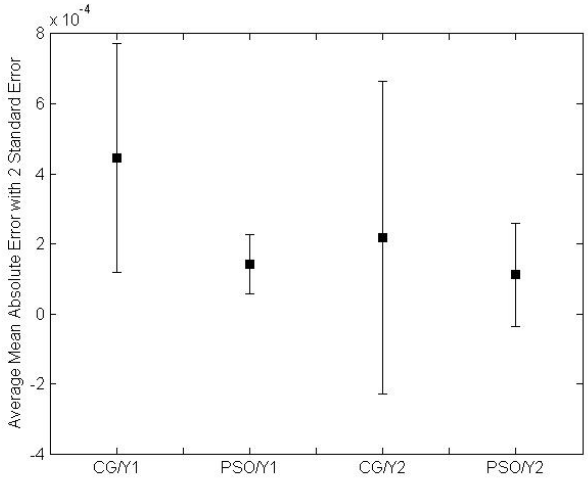


(a) Models with Certain Search Space

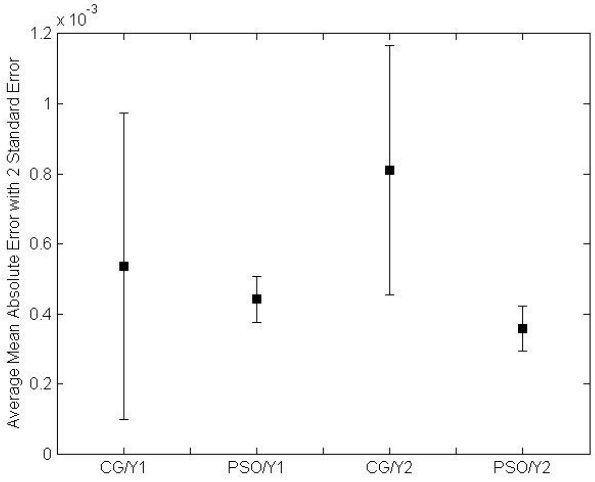


(b) Models with Uncertain Search Space

Fig. 2: Predicted outputs of the single-output simulations. "yreal", "CGpre" and "PSOpre" refer to the actual output and the predicted outputs of CGPs produced using CG and PSO respectively.



(a) $\mathbf{y}_2 = -\mathbf{y}_1$



(b) $\mathbf{y}_2 = \mathbf{exp}(\mathbf{y_1})$

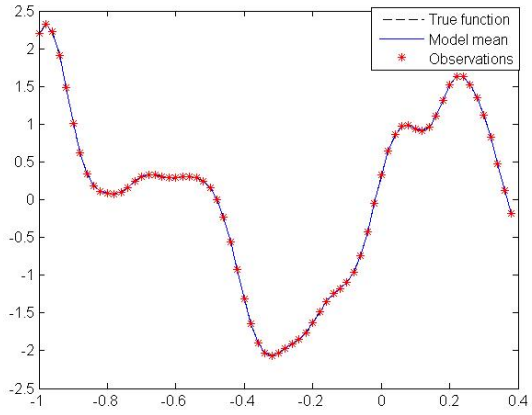
Fig. 3: Multiple-Output dynamical system modelling results: average mean absolute error and 2 standard errors (divided by 0.01).

both outputs in both systems, the models produced by using PSO exhibit smaller average MAE and standard error (SE) values.

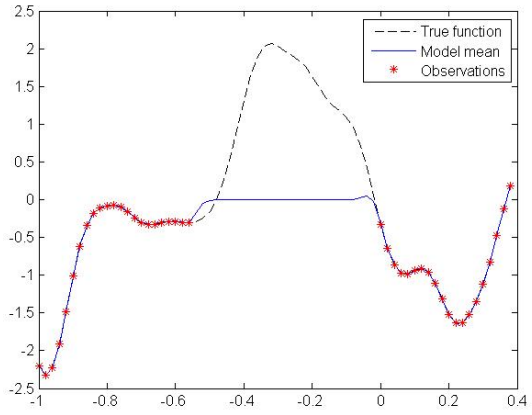
The benefit of using multiple-output models is illustrated when there are missing data in one of the two outputs. When we have full knowledge for both outputs, there is almost no difference in using independent single-output GP models (IGP) and CGP models, as shown in Figures 4a and 4c. However, if a section of 27 samples is taken out of y_2 while 70 samples of y_1 in the same interval, then IGP fails to capture the output in the missing interval. On the other hand, CGP is able to learn the correlation between the outputs and make use of that knowledge to fill in the gap for y_2.

V. Conclusion

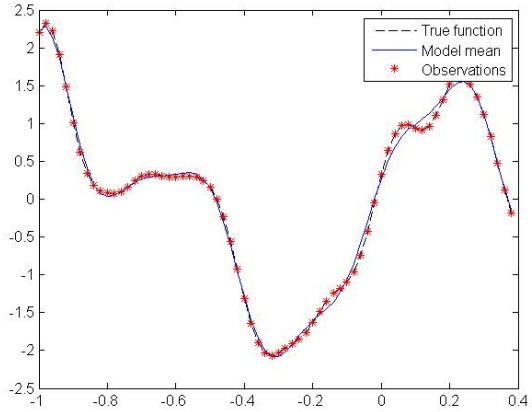
We proposed a new approach to using PSO for learning CGP models based on minimizing the model error. This is different from previous works which apply PSO to maximize the log-likelihood function. The advantage of our method is that it provides a clearer direct indication of the quality of the current solution during the optimization process. Through simulations, we have shown that our method is able to produce the same quality of results as previous PSO methods. The results also



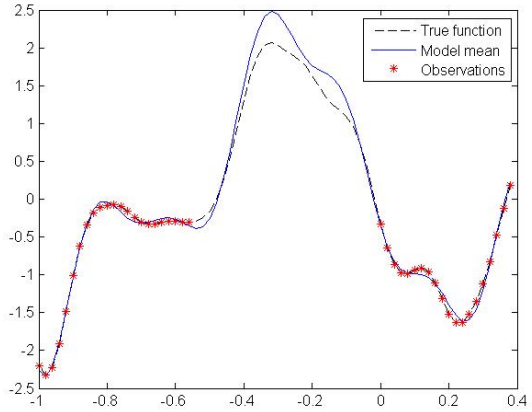
(a) Output y_1 for IGP model



(b) Output y_2 for IGP model



(c) Output y_1 of CGP model



(d) Output y_2 of CGP model

Fig. 4: Effects of missing data for Independent single-output GP models and CGP model.

demonstrate that CGP with our PSO produces better models compared with CG when the parameter search space is large or uncertain. Furthermore, we demonstrated that for MIMO system modelling, CGP is able to learn the output correlations and therefore should be used when there are missing data in one of the outputs.

REFERENCES

[1] K. Ažman and J. Kocijan, "Application of Gaussian processes for black-box modelling of biosystems," *ISA Transactions*, vol. 46, no. 4, pp. 443–457, 2007.

[2] G. Gregorčič and G. Lightbody, "Gaussian process approach for modelling of nonlinear systems," *Engineering Applications of Artificial Intelligence*, vol. 22, no. 4, pp. 522–533, 2009.

[3] P. Boyle, "Gaussian processes for regression and optimisation," Ph.D. dissertation, Victoria University of Wellington, 2007.

[4] D. Higdon, "Space and space-time modeling using process convolutions," in *Quantitative methods for current environmental issues*. Springer, 2002, pp. 37–56.

[5] P. Boyle and M. Frean, "Multi-task Gaussian process prediction," in *Advances in Neural Information Processing Systems 17*, L. K. Saul, Y. Weiss, and L. Bottou, Eds. Cambridge, MA: MIT Press, 2005, pp. 217–224.

[6] M. Alvarez and N. D. Lawrence, "Sparse convolved Gaussian processes for multi-output regression," in *Advances in Neural Information Processing Systems 21*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds., 2009, pp. 57–64.

[7] M. A. Alvarez, "Convolved Gaussian process priors for multivariate regression with applications to dynamical systems," Ph.D. dissertation, University of Manchester, 2011.

[8] F. Zhu, C. Xu, and G. Dui, "Particle swarm hybridize with Gaussian process regression for displacement prediction," in *IEEE Proceedings of International Conference on Bio-Inspired Computing: Theories and Applications*. IEEE, 2010, pp. 522–525.

[9] D. Petelin and J. Kocijan, "Control system with evolving Gaussian process models," in *IEEE Workshop on Evolving and Adaptive Intelligent Systems (EAIS)*. IEEE, 2011, pp. 178–184.

[10] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *IEEE Proceedings of International Conference on Neural Networks*, Perth, WA, Australia, 1995, pp. 1942–1948.

[11] S. Gaffour, M. Mahfouf, and Y. Y. Yang, "'Symbiotic' data-driven modelling for the accurate prediction of mechanical properties of alloy steels," in *IEEE Proceedings of International Conference of Intelligent Systems*. IEEE, 2010, pp. 31–36.