UNIVERSITY OF CINCINNATI

Date: _____

I, _____,
hereby submit this work as part of the requirements for the degree of:

in:

It is entitled:

This work and its defense approved by:

Chair: _____

**Solving Inverse Problems Using Particle Swarm Optimization: An
Application to Aircraft Fuel Measurement Considering Sensor Failure**

A thesis submitted to the
Division of Research and Advanced Studies
of the University of Cincinnati
in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE (INDUSTRIAL ENGINEERING)

in the Department of Mechanical, Industrial and Nuclear Engineering
of the College of Engineering

2006

by

Kai Hu

B.E. University of Science and Technology of China, 2003

Committee Chair: Dr. Samuel Huang

An Abstract of

**Solving Inverse Problems Using Particle Swarm Optimization: An
Application to Aircraft Fuel Measurement Considering Sensor Failure**

Kai Hu

Submitted as partial fulfillment

of the requirement for the

Master of Science Degree in Industrial Engineering

The University of Cincinnati

February 2006

The study of inverse problems is to infer causes that are not directly measurable from some defined observable parameters. The last two decades have seen a great number of methods and applications of solving inverse problems. They generally involve two steps, forward modeling where the reality is represented or approximated by a mathematical model, and inverse modeling where model spaces are searched and matched to their corresponding observable parameters.

This thesis develops a robust inverse problem modeling method which uses Neural Networks to approximate the highly nonlinear physical laws and Particle Swarm Optimization (PSO), an evolutionary algorithm suitable for problems with many local minima, to explore the cause space thoroughly and efficiently. The application of PSO enables this method to be more tolerant to missing data and less susceptible to local minimum than other methods. Furthermore, unlike other methods using gradient to search the model space this proposed method does not depend on the initial value. The superiority of this

method is proved by its application to an aircraft fuel measurement case study with sensor failures. Future work will focus on applying the developed methodology in other inverse problems with missing data and data redundancy reduction applications.

Keywords: Inverse Problem, Particle Swarm Optimization, Neural Networks, Aircraft Fuel Measurement, Sensor Failure

Acknowledgment

First and foremost I would like to thank my advisor Dr. Samuel Huang who has been constantly guiding and encouraging me. His advice and guidance are of great benefit to my research work and life.

I would like to thank Dr. Sam Anand and Dr. Ernest Hall for serving on my thesis committee.

I am also thankful to the team members and friends: John, Nuo, Kanthi, Ranga, Rongsheng, Tianhui, Hemant, Yashpal, Rohit, Patrick... for all the help and support during my study at University of Cincinnati.

Special thanks go to May for her kind assistance.

Last, but not the least, I am indebted to my families who have always stood behind me.

Contents

Abstract

Acknowledgment

1. Introduction.....	1
1.1 Inverse Problem.....	1
1.2 Problem Statement	2
1.3 Thesis Structure.....	3
2. Literature Survey.....	4
3. Methodology.....	11
3.1 Forward Modeling.....	12
3.1.1 Levenberg-Marquardt Algorithm	12
3.1.2 Missing Data Treatment	16
3.1.3 Normalization.....	17
3.2 Inverse Modeling--Particle Swarm Optimization	18
4. Case Study	21
4.1 Aircraft fuel measurement	21
4.2 Modeling for Aircraft Fuel Measurement	22
4.2.1 Neural Network Training	22
4.2.2 Missing Data Treatment	23
4.2.3 Normalization.....	24
4.2.4 PSO search	24
4.3 Model Results	26
4.4 Comparison of Model Performance	27
4.4.1 Model description.....	27
4.4.2 Comparison of Model Precision.....	30
4.4.3 Comparison of Model Robustness.....	31
4.4 PSO Program Running Time	34
5. Conclusion and Future Work.....	36
References	38
Appendix A: Core Codes of PSO	42
Appendix B: Scatter Plots of PSO Model Residuals v.s Parameters.....	49
Appendix C: SAS Codes of Paired T Test	59

Appendix D: Proof of Reduce White Noise by Increasing Sample Rate...61

List of Figures

Figure 1. Many-to-one problem.	6
Figure 2. Methodology flow chart	11
Figure 3. Relation between 2 adjacent layers in a feedforward network....	13
Figure 4. Sensor placement in fuel tank.	22
Figure 5. PSO search based on neural network forward model.....	25
Figure 6. Conventional neural network model.....	27
Figure 7. Neural network with virtual layer.	28
Figure 8. Model precision comparison.....	30
Figure 9. Box-plot of residuals of NN Model and PSO Model.....	31-32
Figure A.1. Flow chart of PS.	47
Figure A.2. Screen shot of the program.	48

List of Tables

Table 1. Inverse problem theory in practical problem.	2
Table 2. Previous work on inverse problems using neural network for forward modeling.	7
Table 3. PSO Model Residuals.	26
Table 4. Descriptive Statistics of PSO Model Residuals.....	27
Table 5. Comparison of model precision.....	30
Table 6. Comparison of model robustness.	31
Table 7. Descriptive statistics of residual differences.....	33
Table 8. T test statistics and p-value.	34
Table 9. Program maximum running time.	35

1. Introduction

1.1 Inverse Problem

Inverse problems can be described as problems where the answer is known but not the question; or where the results or consequences are known, but not the cause. Such problems arise in practical situations such as heat transfer, medical imaging, geophysical exploration, remote sensing, near-surface radar-location, dielectric logging, and nondestructive evaluation where measurements made on the exterior of a body are used to determine properties of the inaccessible interior (Uhlmann 2003).

An example is in medical imaging: If the exact properties of some internal organ were known, then on doing a scan, i.e. targeting that area with radiation or ultrasound, the resultant reflection/ attenuation map would be known. That would be the forward problem. But it is nearly always the properties of the internal organ that we are trying to find, and ideally without invasive surgery. Thus we have to solve an inverse problem (<http://www.inverse-problems.com/>). In our case study, if the sensors are designed to display or monitor the remained fuel volume properly, that is a forward problem. Oppositely, if the remained fuel volume is inferred from the sensor readings, it is an inverse problem. Some examples of inverse problem theory in practical problems are summarized in Table 1.

Table 1. Inverse problem theory in practical problem

Geophysical prospecting.	Inverse scattering theory.
Physical problem of fluid flows.	Finding fluid flows from only the boundary measurements.
Medical diagnosis, scanners	Finding out the fact inside of a body from the measurements that taken outside with tomography and electrical imaging.
Structural damage detection	Vibration-based methods.
Electromagnetism and microwave	Inverse scattering theory, wave-guide structure inference in computational electromagnetics

1.2 Problem Statement

Let S represent a physical system. Assume that we can define a set of models and parameters to describe S . But the models may not be physical first-principal models and the parameters may not all be directly measurable. We can operationally define a set of observable parameters whose values depends on the models and model parameters. To solve a forward problem is to predict the values of these observable parameters, given arbitrary values of the model parameters. To solve a so-called inverse problem is to infer the values of the model parameters from given observed values of the observable parameters (Tarantola 1987).

The problem can be taken into the form

$$L\vec{x} = \vec{m} \quad (1)$$

where L is an operator, represents the models or the physical relation between model parameter $\vec{x}$ and observable parameters $\vec{m}$. The set of observable parameters $\vec{m}$ often underdetermines the model parameter $\vec{x}$ due to the intrinsic lack of data. Also there may be uncertainty in the physical relation L , so the resolution of the forward problem can be

approximations of a complex reality. When L and $\vec{m}$ are ready, the last step to solve inverse problems is to explore the value space of $\vec{x}$. Currently, if the number of parameters in $\vec{x}$ or the value space of $\vec{x}$ is not small, gradient search method is adopted, whose performance depends on the initial guess of the search and is not robust when there are many missing data in $\vec{m}$.

The goal of this work is to develop a robust modeling method using Neural Networks to approximate the physical relation L and use Particle Swarm Optimization, an evolutionary algorithm suitable for problems with many local minima, to explore the input space thoroughly and efficiently. This modeling method should tolerate noisy and missing data in $\vec{m}$.

1.3 Thesis Structure

The thesis has been structured as follows. Chapter 2 starts with the details of a literature review. Chapter 3 describes a framework of the modeling procedure, the development details and the algorithms that are used. Chapter 4 gives a case study in aircraft fuel measurement considering sensor failure, showing the precision, robustness and efficiency of the modeling methodology. Chapter 5 concludes the research performed and gives recommendations for future work.

2. Literature Survey

The procedure to solve an inverse problem can be divided into the following three steps:

- 1) Parametrizing the system: discover and decide a set of model parameters whose values can completely characterize the system.
- 2) Forward modeling: discover the physical laws or approximation of the laws that allow predicting measurements or some observable parameters based on given model parameters.
- 3) Inverse modeling: use actual results of some measurements of the observable parameters to infer the actual values of the model parameters.

The first two steps are mainly inductive and the third step is mainly deductive. Parametrizing the system is unique to the specific problems. And there are mainly 3 approaches for forward modeling:

- 1) First-principle physical theory models

A first-principle physical theory is always the first choice if it is available. The theory models can be used to predict the results of observations accurately. Trlifaj (1989) gave a good example of how to transfer transformations and infer equations and solve inverse problems.

- 2) Numerical methods

In some cases, it is possible to obtain a simple analytical expression for the physical laws. But when the physical first-principle models are not available to describe the relationship between parameter/cause and measurement/result, the inverse problem may be

analytically intractable. Numerical methods, especially finite element methods are popularly used to approximate the physical laws. Zhang and Xu (1996) successfully built electromagnetic field distribution model in planar dielectric waveguide via finite element method, and used it as forward model to infer the unknown structure of planar dielectric waveguide with discontinuities. Wolters, Grasedyck and Hackbusch (2004) used finite element methods to realistically model tissue conductivity inhomogeneities and anisotropies, which is crucial for an accurate reconstruction of the current distribution while reconstructing the underlying current distribution in the human brain using potential differences and/or magnetic fluxes that are measured non-invasively directly. Bonnet and Constantinescu (2005) reviewed some inverse problems arising in the context of linear elasticity, namely the identification of distributions of elastic moduli, model parameters or buried objects such as cracks, in which finite element method is a general method used in forward modeling.

3) Neural networks

Neural networks have been successfully applied to nonlinear relationship approximation and inverse parameter retrieval problems in the past since it is well known that multilayer feedforward networks with one or more hidden layers are able to approximate any static continuous nonlinear function (Leshon *et al.* 1993) and also because of their abilities to deal with noisy data. When there is not a clear relationship or the relationship is nonlinear, neural networks can be used to approximate the relationship. Physical models or experimental data can be used to train the neural networks. In previous works, explicit neural network inverse (Chang and Tsang 1992, Tsang *et al.* 1995, Tsang *et al.* 1992, Priddy *et al.* 1993), recurrent neural network (Duckett *et al.* 2001), neural network iterative inversion

(Davis *et al.* 1993), and Bayesian neural network iterative inversion (Davis and Hwang 1997) were used. However, these methods have certain constraints:

- Explicit inverse processes cannot deal with the many-to-one mapping as shown in Figure 1; namely, more than one parameter $\vec{x}$ could account for the observed measurement $\vec{m}$.

$$\vec{m} = f(\vec{x}_1) = f(\vec{x}_2) = \dots \quad (2)$$

Figure 1. Many-to-one problem.

- Recurrent neural network can be applied only when the outputs contain just a few fixed patterns;
- Neural network iterative inversion and Bayesian neural network iterative inversion use a gradient descent search algorithm to obtain the desired $\hat{\vec{x}}$. A guess or initial estimate of $\vec{x}$ is set by combining other information; otherwise, the gradient search can be trapped in local minima if the search process begins too far from the true parameter $\vec{x}$, when the object function surface is not convex or concave. Also, for a trained neural network, if there are too many missing data in the measurement $\vec{m}$, it is difficult to reach the solution during the iterative inversion process. A list of papers related is summarized

in Table 2.

Table 2. Previous work on inverse problems using neural network for forward modeling

Paper	Applications	Search Methods	Missing Data
Studer and Peters 2003	damage identification	gradient search	1 of 20 sensors' data are missing
Mujica <i>et al.</i> 2003	damage identification	table search	did not consider
Duckett <i>et al.</i> 2001	robot outdoor sensing	recurrent neural network	did not consider
Wang <i>et al.</i> 1998	satellite remote sensing	gradient search	did not consider
Davis and Hwang 1997	satellite remote sensing	gradient search	did not consider
Davis <i>et al.</i> 1993	satellite remote sensing	gradient search	did not consider
Priddy <i>et al.</i> 1993	pump fault diagnosis	gradient search	did not consider
Tsang <i>et al.</i> 1992a	satellite remote sensing	gradient search	did not consider
Tsang <i>et al.</i> 1992b	satellite remote sensing	gradient search	did not consider

Most general method for solving nonlinear inverse problems needs a complete exploration of the model space. For problems other than academic, the method is in general too computer intensive to be useful. This is why usual methods limit their scope to obtaining some “best” model (Tarantola 1987). There are 3 mainly approaches for the inverse modeling:

1) Systematic exploration methods

If the number of the model parameters is small, the parameter value space is discrete, and the computation of each value for an arbitrary $\vec{m}$ is inexpensive, we can systematic explore the value space using methods like grid computation.

2) Gradient methods

If the forward problem is not excessively nonlinear and has a single extremum, or a

good initial guess can be set with the help of other knowledge, the gradient search method can be used. Gradient methods use the local properties of the function at a current point x_n to decide on a direction of search for the updated model x_{n+1} . Let $\mathbf{g}_n$ represent the direction of steepest ascent at x_n . A crude steepest descent method defines x_{n+1} by

$$x_{n+1} = x_n - \mathbf{m}_n \mathbf{g}_n \quad (3)$$

where $\mathbf{m}_n$ is an arbitrary positive real number small enough to ensure that function value at x_{n+1} will be smaller than that at x_n .

3) Monte Carlo methods

For highly nonlinear problems, there is a considerable risk that gradient methods will converge to local extrema and lead to secondary solutions. Any method which uses a random or pseudo-random generator at any stage is named Monte Carlo. They can solve problems of relatively large size in a fully nonlinear form. Some interesting references to the use of Monte Carlo methods in inverse problems are Press (1968) and Rothman (1986). The proposed Particle Swarm Optimization is also a kind of Monte Carlo method.

If there is only one parameter $\bar{x}$ that can yield the given measurement $\bar{m}$, the explicit inverse process works well (Chang and Tsang 1992). Duckett *et al.* (2001) located the source of an odor with a mobile robot. They used recurrent neural network to learn the required mapping from sensor readings to the direction of the odor source. However, recurrent neural network can be applied only when the outputs contain only a few fixed patterns.

Priddy *et al.* (1993) used neural networks to identify pump faults. Mujica *et al.* (2003) described a soft computing approach to solve the damage identification problem using

case-based reasoning to obtain an initial diagnostic. They created an incremental database to future diagnosis by analogy. However, the authors did not consider sensor failure and the associated missing data problem. Studer and Peters (2003) combined sensing and neural networks for damage identification. To test the robustness of the method, one of the 20 fibers of the sensing network was removed alternatively each time. The missing data was replaced by the average value of the two neighboring sensors or by one neighboring sensor. Nonetheless, the authors did not show the robustness of their model when multiple sensors fail.

Neural network iterative inversion generates the parameters $\hat{x}$ based on measurements $\bar{m}$ from a trained forward neural network mapping model $\hat{f}(x)$, which gives a network output as close as possible to a desired measurement vector. By taking advantage of the duality between the weights and the input activation values in minimizing the mean squared error or other cost function, the iterative gradient descent algorithm can be applied to obtain the desired input vector by searching the input space. The idea is similar to the back-propagation algorithm, where the error signals are propagated back through the network to adjust the input $\bar{x}$ in an attempt to decrease the output error. Davis and Hwang (1997) systemically studied how to solve inverse problems using Bayesian neural network iterative inversion with ground truth incorporation. Let $f(x|m)$ be the conditional probability for the parameter x given the set of measurement m . To find the x that maximizes the posterior probability, the following are needed: probability distributions of the x neighborhood, the prior distribution $f(x)$ and the sensor noise and model mismatch distribution. However, such information is not available always. Besides, the authors used gradient search, which is easy

to be trapped in local minima. Furthermore, for a trained neural network, if there is too much missing data in the measurement m , it is hard to reach the solution during the iterative inversion process.

3. Methodology

As summarized in the previous chapters, after parametrizing the system, there are two steps to solve the inverse problems: Forward Modeling and Inverse Modeling. First, a forward model $\vec{m} = \hat{f}(\vec{x})$ is created to approximate the physical process, where $\vec{x}$ is the set of input parameters, and $\vec{m}$ is the resulting set of measurements. For a general purpose, neural networks are used and physical models or experimental data can be used to train the neural networks. In Inverse Modeling procedure, the input space is explored to find $\vec{x}$ that produces the measurements that match $\vec{m}$ using PSO. The methodology is shown in Figure 2.

Figure 2 Methodology flow chart

Step 1. Parametrizing the system: discover and decide a set of model parameters whose values can completely characterize the system

Step 2. Forward modeling: discover the physical laws or approximation of the laws that allow predicting measurements on some observable parameters based on given model

parameters (Neural Networks)

Step 3. Inverse modeling: use actual results of some measurements of the observable parameters to infer the actual values of the model parameters (PSO)

3.1 Forward Modeling

Considering the highly nonlinear relationship, the Levenberg-Marquardt algorithm is used to train a neural network model.

3.1.1 Levenberg-Marquardt Algorithm

The Levenberg-Marquardt algorithm is a nonlinear least squares algorithm to batch training. Consider a multilayer feedforward network shown in Figure 3, the net input to unit i in layer $k + 1$ is:

$$n^{k+1}(i) = \sum_{j=1}^{S_k} w^{k+1}(i, j) a^k(j) + b^{k+1}(i) \quad (4)$$

The output of unit i will be

$$a^{k+1}(i) = f^{k+1}(n^{k+1}(i)) \quad (5)$$

Figure 3. Relation between 2 adjacent layers in a feedforward network.

For an M-layer network the system equations in matrix form are given by

$$\vec{a}^0 = \vec{x} \quad (6)$$

$$\vec{a}^{k+1} = \vec{f}^{k+1}(W^{k+1}\vec{a}^k + \vec{b}^{k+1}), \quad k = 0, 1, \dots, M-1 \quad (7)$$

The approximate steepest gradient descent algorithm is

$$\Delta w^k(i, j) = -\mathbf{a} \frac{\partial \hat{V}}{\partial w^k(i, j)} \quad (8)$$

$$\Delta b^k(i) = -\mathbf{a} \frac{\partial \hat{V}}{\partial b^k(i)} \quad (9)$$

where $\mathbf{a}$ is the learning rate.

Define the sensitivity of the cost function to changes in the input of unit i in layer k as:

$$\mathbf{d}^k(i) \equiv \frac{\partial \hat{V}}{\partial n^k(i)} \quad (10)$$

Now it can be shown that

$$\frac{\partial \hat{V}}{\partial w^k(i, j)} = \frac{\partial \hat{V}}{\partial n^k(i)} \frac{\partial n^k(i)}{\partial w^k(i, j)} = \mathbf{d}^k(i) a^{k-1}(j) \quad (11)$$

$$\frac{\partial \hat{V}}{\partial b^k(i, j)} = \frac{\partial \hat{V}}{\partial n^k(i)} \frac{\partial n^k(i)}{\partial b^k(i, j)} = \mathbf{d}^k(i) \quad (12)$$

and the sensitivities satisfy the following recurrence relation

$$\mathbf{d}^k = \dot{F}^k(n^k) W^{k+1T} \mathbf{d}^{k+1} \quad (13)$$

where

$$\dot{F}^k(n^k) = \begin{bmatrix} \dot{f}^k(n^k(1)) & 0 & \dots & 0 \\ 0 & \dot{f}^k(n^k(2)) & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \dot{f}^k(n^k(Sk)) \end{bmatrix} \quad (14)$$

and

$$\dot{f}^k(n) = \frac{df^k(n)}{dn} \quad (15)$$

This relation is initialized at the final layer

$$\bar{\mathbf{d}}^M = -\dot{F}^M(n^M)(\bar{m}_q - \bar{a}_q) \quad (16)$$

Now suppose we have a function $V(\vec{x})$ which we want to minimize with respect to the parameter vector $\vec{x}$, then Newton's method would be

$$\Delta x = -[\nabla^2 V(\vec{x})]^{-1} \nabla V(\vec{x}) \quad (17)$$

where $\nabla V(\vec{x})$ is the gradient and $\nabla^2 V(\vec{x})$ is the Hessian matrix. Now the cost function $V(\vec{x})$ is a sum of squared errors:

$$V(\vec{x}) = \sum_{i=1}^N e_i^2(\vec{x}) \quad (18)$$

Then

$$\nabla V(\vec{x}) = J^T(\vec{x}) \vec{e}(\vec{x}) \quad (19)$$

$$\nabla^2 V(\vec{x}) = J^T(\vec{x}) J(\vec{x}) + S(\vec{x}) \quad (20)$$

where $J(\vec{x})$ is the Jacobian matrix

$$J(\vec{x}) = \begin{bmatrix} \frac{\partial e_1(\vec{x})}{\partial x_1} & \frac{\partial e_1(\vec{x})}{\partial x_2} & \dots & \frac{\partial e_1(\vec{x})}{\partial x_n} \\ \frac{\partial e_2(\vec{x})}{\partial x_1} & \frac{\partial e_2(\vec{x})}{\partial x_2} & \dots & \frac{\partial e_2(\vec{x})}{\partial x_n} \\ \dots & \dots & \dots & \dots \\ \frac{\partial e_N(\vec{x})}{\partial x_1} & \frac{\partial e_N(\vec{x})}{\partial x_2} & \dots & \frac{\partial e_N(\vec{x})}{\partial x_n} \end{bmatrix} \quad (21)$$

and

$$S(\vec{x}) = \sum_{i=1}^N e_i(\vec{x}) \nabla^2 e_i(\vec{x}) \quad (22)$$

In the Gauss-Newton method it is assumed that $S(\vec{x}) \approx 0$, and the update becomes

$$\Delta \vec{x} = [J^T(\vec{x})J(\vec{x})]^{-1} J^T(\vec{x}) \vec{e}(\vec{x}) \quad (23)$$

The Levenberg-Marquardt modification is

$$\Delta \vec{x} = [J^T(\vec{x})J(\vec{x}) + \mathbf{m}I]^{-1} J^T(\vec{x}) \vec{e}(\vec{x}) \quad (24)$$

The parameter $\mathbf{m}$ is multiplied by a factor $\mathbf{b}$ when a step would result in an increased $V(\vec{x})$ and $\mathbf{m}$ is divided by $\mathbf{b}$ when a step reduces $V(\vec{x})$. When $\mathbf{m}$ is large the algorithm becomes steepest descent with step $1/\mathbf{m}$, while for small $\mathbf{m}$ it becomes Gauss-Newton.

For the network mapping problem the terms in the Jacobian matrix can be computed by a modification to the backpropagation algorithm. We list all the parameters of network in one vector:

$$\vec{x} = [w^1(1,1), w^1(1,2), \dots, w^1(S1, R), b^1(1), \dots, b^1(S1), w^2(1,1), \dots, b^M(SM)]^T \quad (25)$$

$$N = Q \times SM \quad (26)$$

Standard backpropagation calculates terms like

$$\frac{\partial \hat{V}}{\partial w^k(i, j)} = \frac{\partial \sum_{l=1}^{SM} e_q^2(l)}{\partial w^k(i, j)} \quad (27)$$

For the elements of the Jacobian matrix we need to calculate terms like

$$\frac{\partial e_q(l)}{\partial w^k(i, j)} \quad (28)$$

These terms can be calculated using the standard backpropagation algorithm with one modification at the final layer

$$\Delta^M = -\dot{F}^M(n^M) \quad (29)$$

The Levenberg-Marguardt algorithm is summarized as follows (Hagan and Menhaj 1994):

1. Compute the network outputs using (6) and (7) and sum of squared errors over all input.
2. Compute the Jacobian matrix using (29), (13), (11), (12) and (21).
3. Solve (24) to obtain $\Delta\vec{x}$.
4. Re-compute the sum of squared errors using $\vec{x} + \Delta\vec{x}$. If new sum is smaller than that in step 1, then reduce $\mathbf{m}$ by $\mathbf{b}$, let $\vec{x} = \vec{x} + \Delta\vec{x}$ and go back to step 1; otherwise increase $\mathbf{m}$ by $\mathbf{b}$ and go back to step 3.
5. The algorithm stops when the norm of the gradient (19) is less than some predetermined value or when the sum of squares has been reduced to some goal.

3.1.2 Missing Data Treatment

The cost function (18) can be expanded as:

$$V = \frac{1}{2} \sum_{q=1}^Q (\vec{m}_q - \vec{a}_q^M)^T (\vec{m}_q - \vec{a}_q^M) = \frac{1}{2} \sum_{q=1}^Q \vec{e}_q^T \vec{e}_q \quad (30)$$

where $\vec{a}_q^M = (a_1, a_2 \dots a_{n_c})$ is the output of the network under the q th input $\vec{x}_q$;

$\vec{m}_q = (m_1, m_2 \dots m_{n_c})$ is the expected output; and $\vec{e}_q = \vec{m}_q - \vec{a}_q^M$ is the error for the q th input.

This function is modified to handle missing data. During the training procedure, if there are missing data, i.e., $m_i = 0$ or the value of a_i is out of the possible range A , the difference is not considered in the squared errors. This procedure is shown in the following equations:

$$\vec{m}_q = (m_1, m_2 \dots m_{n_c}) \quad (31)$$

$$\vec{a}_q^M = (a_1, a_2 \dots a_{n_c}) \quad (32)$$

$$\text{if } m_i = 0 \text{ or } a_i \notin A, \text{ then } a_i = m_i \quad (33)$$

$$\vec{e}'_q = \vec{m}'_q - \vec{a}'_q^M \quad (34)$$

For the standard backpropagation algorithm, the cost function is approximated by

$$\hat{V} = \frac{1}{2} \vec{e}'_q{}^T \vec{e}'_q \quad (35)$$

where the total sum of squares is replaced by the squared errors for a single input/output pair.

3.1.3 Normalization

Least squares are so popular for solving inverse problems because they lead to the easiest computations. Their only drawback is their lack of robustness for their strong sensitivity to the large errors in data sets. If there are different variables with various scales in the data sets, it is better to normalize them so that their effects are equivalent. Linear normalization is the most common pre-processing of the input data and it is often useful if different variables have typical values which differ significantly (Bishop 1996).

3.2 Inverse Modeling--Particle Swarm Optimization

When forward model and measurement data are ready, PSO can be used to search the input space. PSO is based on a social simulation of the movement of flocks of birds and schools of fish. It was first presented in a 1995 paper (Kennedy and Eberhart 1995) that presented it as a new way to solve function optimization problems. A swarm is a population of particles, each of which exists in N-dimensional space. There are as many dimensions as there are parameters to be optimized, so each point in that space represents a possible solution to the function.

Each particle (also called agent) has 3 vectors and 2 fitness values (Ratnaweera *et al.* 2004):

X-vector records the current position of the particle in the search space

P-vector records the location of the best solution found so far by the particle

V-vector contains a gradient (direction) for which particle will travel to if undisturbed

X-fitness records the fitness of the x-vector

P-fitness records the fitness of the p-vector

After each unit of time, the particles' velocity is stochastically adjusted by the direction to the global and personal best positions. The velocity is constrained in each dimension such that it cannot be higher or lower than a set maximum and minimum respectively. The update equations for velocity and position are shown in (36) and (37), respectively:

$$v_{id(t)} = wv_{id(t-1)} + c_1 \times rand() \times (p_{id} - x_{id}) + c_2 \times rand() \times (p_{gd} - x_{id}) \quad (36)$$

$$x_{id(t)} = x_{id(t-1)} + v_{id(t)} \quad (37)$$

where

x_{id} : current value of the dimension “ d ” of the individual “ i ”

v_{id} : current velocity of the dimension “ d ” of the individual “ i ”

p_{id} : optimal value of the dimension “ d ” of the individual “ i ” so far

p_{gd} : current optimal value of the dimension “ d ” of the swarm

c_1, c_2 : acceleration coefficients

w : inertia weight factor

The algorithm can be summarized in the following steps:

1. Initialize particles in the search space at random.
2. Assign random initial velocities for each particle.
3. Evaluate the fitness of each particle according to a user defined objective function.
4. Calculate the new velocities for each particle.
5. Move the particles.
6. Repeat steps 3 to 5 until a predefined stopping criterion is satisfied.

Clow and White (2004) explained the algorithm. The particles “fly” through the N-dimensional space, exploring the solution space when they travel. Once a better global best position is found, the entire swarm will migrate towards it, exploring the more profitable solution space around the global best position. The personal best provides the drive to explore the local solution space of the particle as the swarm moves through space, particles will further explore those regions they’ve individually found to be profitable. The cognitive parameter c_1 controls how much emphasis is placed on information learned by the particle and the social parameter c_2 controls how much emphasis is placed on information learned

from other particles. The inertia weight w controls the global and local exploration ability of the swarm by controlling the emphasis on the previous velocity. A low inertia means that the particles move more directly towards the global/local bests. A high inertia means that the particles will move indirectly towards the best positions, exploring a greater area of the solution space during the movement. At the beginning, it is desirable to have a high inertia to be sure that particles explore the global solution space thoroughly. Once a good solution region is found it is important to explore it in detail, so the inertia weight is gradually decreased down to 0 at the end of the training. As the run progresses, the inertia is decreasing and particles tend to explore the local solution spaces and fine-tune the solutions discovered. The choice of maximum velocity is also critical. If the maximum velocity is too high, particles will explore the solution space in large discrete jumps, which may bypass good solution spaces. If it is too low, the particles cannot obtain sufficient velocity to move to a new solution space before being attracted back towards the global or personal best positions. Decreasing the maximum velocity as the training progresses increases the performance in a similar way to decreasing the inertia weight.

4. Case Study

4.1 Aircraft fuel measurement

A fuel measurement system in an aircraft gauges the fuel volume in the tanks located in the aircraft wings, fuselage, and horizontal stabilizer. It monitors the distribution of fuel while commanding pumps and valves of the aircraft to control the fuel transfer and refuel, and maintain the center of gravity. Accurate measurement of residual fuel is directly correlated to the flight maneuverability and the fuel consumption and endurance. The current measurement method is to install a set of sensors in the fuel tank (Figure 4), and use the sensor outputs $c_{i1}, c_{i2}, \dots, c_{in_c}$ together with the flight status-pitch and roll $\mathbf{q}, \mathbf{g}_i$ to obtain the fuel volume V_i through computing or experiment ($i = 1, 2, \dots, n$ is the index of computing or experiment, n_c is the number of sensors). A table is then built to save the data (sensor readings by fuel volume and flight status). During the flight, the fuel volume can be computed by interpolation based on the flight status $\mathbf{q}, \mathbf{g}$, sensor outputs $c_1, c_2, \dots, c_{n_c}$ and the table (Jiang et al. 2000). Fuel measurement is an inverse problem: fuel Volume (V) and flight status ($\mathbf{q}, \mathbf{g}$) determines sensor readings ($c_1, c_2, \dots, c_{n_c}$), but V is determined based on the measurement of sensor outputs ($c_1, c_2, \dots, c_{n_c}$). Note that the sensor readings may contain noise and sometimes sensors do fail, which may yield incorrect readings or no readings at all.

Figure 4. Sensor placement in fuel tank (Courtesy of Parker Hannifin).

In aircraft fuel measurement, there is a highly nonlinear relationship between the measurements and the parameters. The shape of the tank is too complicated to use an analytical method to determine the remaining fuel under different flight status. Some sensors may fail, which will cause missing data in measurements. So it is a typical case for our study.

4.2 Modeling for Aircraft Fuel Measurement

4.2.1 Neural Network Training

In the aircraft fuel measurement problem, the training data sets are generated from experiments – 1750 sensor readings under 35 combinations of pitch and roll ($\mathbf{q}, \mathbf{g}$) and 20 different volume values in each combination (courtesy of Parker Hannifin).

Different neural network models were built, and the number of hidden neurons is determined by Akaike Information Criteria (AIC).

$$AIC = \ln(\hat{V}) + 2K \quad (38)$$

where K is the number of adjustable weights of the neural network

The network is trained in Matlab because of its high speed. The activation functions of hidden layer and output layer are “tansig” and “purelin”. But it is not easy to adapt the functions in Matlab to our problems; therefore, a neural network with same structure and parameters is built in VC++.

4.2.2 Missing Data Treatment

The cost function (18) is rewritten:

$$V = \frac{1}{2} \sum_{q=1}^Q (\vec{m}_q - \vec{a}_q^M)^T (\vec{m}_q - \vec{a}_q^M) = \frac{1}{2} \sum_{q=1}^Q \vec{e}_q^T \vec{e}_q \quad (39)$$

This function is modified in the fuel measurement application to handle missing data. In fuel measurement, the sensor readings indicate the length of sensors under the fuel surface. If a sensor is out of fuel, the “length” under the fuel surface is negative, but the reading is 0. So during the training procedure, if the desired sensor reading is 0 and the output of the network is negative, we do not account for the difference the squared errors. Similarly, if a sensor fails, the desired sensor reading is 0, and the difference is not considered in the squared errors either. This procedure is shown in the following equations:

$$\vec{m}_q = (m_1, m_2 \dots m_{n_c}) \quad (40)$$

$$\vec{a}_q^M = (a_1, a_2 \dots a_{n_c}) \quad (41)$$

$$\text{if } m_i = 0 \text{ and } a_i < 0, \text{ then } a_i = m_i \quad (42)$$

$$\vec{e}_q' = \vec{m}_q' - \vec{a}_q'^M \quad (43)$$

4.2.3 Normalization

Usually, the data are normalized into a range with mean of 0 and standard deviation of 1. But we found the network with this kind of normalization method to be extremely sensitive at the 0 data points. For example, the fuel volume value is from 0 to 24370, and after normalization 0 represents a volume of 12000. If some noise is present, the error of output is much larger than average. We checked the network, and found that some neurons' weights and biases were extremely large whereas their inputs were small, which means these neurons are more sensitive to noise. Therefore, we normalize the data into a range [0.1, 0.9] to avoid 0.

4.2.4 PSO search

After training a neural network as a forward model, fuel volume V can be determined given pitch ($\mathbf{q}$), roll ($\mathbf{g}$) and sensor readings $\vec{m} (c_1, c_2, \dots, c_{n_c})$ as shown in Figure 5. Given $\mathbf{q}$ and $\mathbf{g}$, any value V will yield an output $\hat{\vec{m}} (\hat{c}_1, \hat{c}_2, \dots, \hat{c}_{n_c})$ through the forward model. PSO search is used to find the value V that minimizes the following objective function:

$$E_2 = \frac{1}{2} (\vec{m} - \hat{\vec{m}})^2 \quad (44)$$

Note that when sensor i fails, sensor reading c_i will have no value. In this case, the objective function is calculated as follows:

$$E_2 = \frac{1}{2} \sum_{i=1}^{n_c} I_i (c_i - \hat{c}_i)^2 \quad (45)$$

where $I_i = \begin{cases} 1, & \text{if } c_i \text{ has value} \\ 0, & \text{if } c_i \text{ has no value} \end{cases}$

Figure 5. PSO search based on neural network forward model.

When using PSO search, there is a tradeoff between converging time and the accuracy.

After experiments, the PSO parameters are chosen as follows:

Swarm size $S = 20$

Max number of informed particles by a given one $K = 3$

Terminating accuracy = 0.0000001

Max number of evaluations=10000

Max number of evaluations determines the longest running time of the program and 10000 epochs will cost almost 1 second (experiment results will be shown). Error decreases when terminating accuracy requirement increases. And the error level is stable when Terminating accuracy is less than 0.0000001.

Twenty (20) particles are initialized in the input space, and then each particle gets a random velocity. The fitness of each particle calculated from objective function (45) is evaluated, and the results are used to update the particle velocities using (36) and particle

positions using (37). Then the process repeated until the algorithm converges or maximum epochs are reached. The maximum velocity and inertia weight are high at the beginning, and decrease gradually when a good solution region is found that should be explored in detail. They are decreased down to 0 at the end. Appendix A gives the core codes of this procedure and a screen shot of the program that is developed in Visual C++.

4.3 Model Results

A noise-free data set and a noisy data set (provided by Parker Hannifin) are used to test model precision, measured in terms of error (residual) calculated as follows:

$$E = \frac{|V - \hat{V}|}{V_{\max}} \times 100\% \quad (46)$$

where V is the true volume, $\hat{V}$ is the model output, and $V_{\max}$ is the capacity of the fuel tank. In addition to the noise, 2 and 6 sensor readings (out of the total of 13 sensors) are erased randomly, simulating the effect of sensor failure. The results are listed in Table 3.

Table 3. PSO Model Residuals

Model	Error	No Sensor Failure		2 Sensor Failures		6 Sensor Failures	
		Noise-free	Noisy	Noise-free	Noisy	Noise-free	Noisy
PSO	Avg.	0.3962	0.4951	0.4942	0.5977	0.5690	0.7365
	Max.	1.8153	1.8522	1.9158	2.4538	1.7045	2.4433

It shows that the model precision only dropped slightly when dealing with noisy data. The drop of precision when dealing with sensor failures is slightly higher; but the precision is still within practical requirement. The number of sensor failures does not seem to have much effect on the model precision.

A residual analysis helps to better understand the model performance. The descriptive statistics of PSO residuals are listed in Table 4. The mean, median, standard

deviation and variance stay in a tight range through these 6 groups of residuals, which indicate the stableness of model performance. The interquartile ranges of residuals without noise are tighter than the ones with noise.

Table 4. Descriptive Statistics of PSO Model Residuals

	No-missing Noise free	No-missing Noisy	2-missing Noise free	2-missing Noisy	6-missing Noise free	6-missing Noisy
Mean	0.396191	0.495065	0.494189	0.597673	0.569047	0.745678
Median	0.21571	0.273977	0.313081	0.5244	0.498139	0.621833
Std.dev	0.46658	0.51428	0.54147	0.55356	0.45886	0.58701
Variance	0.21769	0.26448	0.29319	0.30642	0.21056	0.34458
Range	1.79539	1.84689	1.89436	2.41027	1.68879	2.43226
Interquartile range	0.30598	0.64668	0.3814	0.55838	0.48874	0.64269

Appendix B gives the scatter plot of Model Residuals v.s. all parameters (actual volume value, pitch, roll, and 13 sensor readings). There is no obvious pattern across the plots, which means the goodness of the modeling methods.

4.4 Comparison of Model Performance

4.4.1 Model description

Model 1: Conventional neural network model A conventional neural network with inputs $\vec{m} (c_1, c_2, \dots, c_{n_c})$, $\mathbf{q}$, $\mathbf{g}$ and output V is trained as shown in Figure 6.

Figure 6. Conventional neural network model.

Model 2: Neural network iterative inversion model A neural network forward model similar to the one shown in Figure 7 is built. Gradient search is used instead of PSO search to find V . To facilitate computation, a virtual layer is added with three inputs fixed at 1. Each input connects to only one neuron of the original input layer and the weights x_1, x_2, x_3 represents the original values of $\mathbf{q}$, $\mathbf{g}$, and V , respectively. In this way, the conventional Levenberg-Marquardt algorithm can be used directly in the iterative inversion process. In each iteration step, only the weight x_3 (representing the volume V) is updated in an attempt to minimize the objective function (45).

Figure 7. Neural network with virtual layer.

$$E_k = \frac{1}{2} (o_k - d_k)^2 \quad (47)$$

$$E = \sum_{k=1}^K E_k \quad K: \text{ number of outputs} \quad (48)$$

$$d_{ok} = f_o' (o_k - d_k) \quad (49)$$

$$d_{yj} = f_h' \sum_{k=1}^K d_{ok} \times w_{kj} \quad (50)$$

$$\mathbf{d}_{zi} = f_i' \sum_{j=1}^J \mathbf{d}_{yj} \times v_{ji} \quad \text{J: number of hidden neurons} \quad (51)$$

The algorithm can be summarized as following steps:

For each epoch(Loop)

Set learningRate $\mathbf{h} = 0.001$;

For each sample (Loop)

1. Calculate $\mathbf{d}$;
2. Calculate error, denote it as e_1 ;
3. Change input

$$x_i = x_i + \mathbf{h} \times \mathbf{d}_{zi} \times 1 \quad (52)$$

4. Calculate error, denote it as e_2 ;
5. IF ($e_1 > e_2$)

Add e_2 to errorsum;

Next sample;

Else

Undo the change;

Add e_1 to errorsum;

Next sample

$\mathbf{h} = \mathbf{h} / 1.5$; // If $\mathbf{h} < 0.00001$, break;

Next epoch

Model 3: PSO model. As described in Section 4.2.

4.4.2 Comparison of Model Precision

The average error and maximum error are compared. The result is shown in Table 5 and Figure 8. It is shown that the conventional neural network model has the best precision with the noise-free data set. Neural network iterative inversion model highly depends on the initial guess. Since it is impractical to have the true volume as an initial guess, neural network iterative inversion model can not be reliably used in aircraft fuel measurement, and hence not considered further.

Table 5. Comparison of model precision.

Model	Error (%)	
	Average	Maximum
Conventional neural network model	Average	0.0794
	Maximum	0.2421
Neural network iterative inversion model with initial guess close to true volume	Average	0.2016
	Maximum	1.7764
Neural network iterative inversion model with initial guess as 0	Average	> 50
	Maximum	> 90
PSO model	Average	0.3962
	Maximum	1.8153

Figure 8. Model precision comparison

4.4.3 Comparison of Model Robustness

Model robustness refers to the model's ability to maintain precision while dealing with measurement noise and sensor failure. Again, the residuals are calculated in the same way as (46). The result is shown in Table 6 and Figure 9.

Table 6. Comparison of model robustness.

Model	Error	No Sensor Failure		2 Sensor Failures		6 Sensor Failures	
		Noise-free	Noisy	Noise-free	Noisy	Noise-free	Noisy
Neural Network	Avg.	0.0794	0.3904	4.1253	10.9916	Omitted since the errors are too high	
	Max.	0.242094	0.753364	21.24683	30.89776		
PSO	Avg.	0.3962	0.4951	0.4942	0.5977	0.5690	0.7365
	Max.	1.8153	1.8522	1.9158	2.4538	1.7045	2.4433

(a) Noise-free data without sensor failure

(b) Noisy data without sensor failure

(c) Noise-free data with 2 sensor failures

(d) Noisy data with 2 sensor failures

Figure 9. Box-plot of residuals of NN Model and PSO Model

It seems like that the conventional neural network model has a higher precision than that of the PSO model when there is no sensor failure. Yet the level of precision has dropped significantly. When there are 2 sensor failures, its model precision deteriorates to an unacceptable level (Parker Hannifin requires an average error of 0.8%). When there are 6 sensor failures, the conventional neural network model fails miserably as most of the output value is not even within the range of the fuel tank capacity. On the other hand, the PSO model is highly robust.

To validate our conclusions, a T test is conducted to compare residual means in addition the box-plots. If a model is preferred to the other, this model should generate significantly smaller residuals than the other one. Since we are comparing the residuals of 2 models on the same input data set. Apparently, a paired T test suffices to solve this problem (Milton and Arnold 1995). The SAS codes can be found in Appendix C.

The residuals from 2 models are matched to each other by design; therefore the assumption that these 2 groups of residuals are independent doesn't hold any more. 2 new variables are created for the purpose of paired T test.

$$D_{ij} = \text{Residual}_{NN_{ij}} - \text{Residual}_{PSO_{ij}} \quad (53)$$

where $i=1$, in the case of no-missing, noise-free data;

=2, in the case of no-missing, noisy data;

=3, in the case of 2-missing, noise-free data;

=4, in the case of 2-missing, noisy data;

j= 1, 2... 31, the jth corresponding residuals from both models.

$$\bar{D}_i = \sum_j D_{ij} / n, \text{ where } n \text{ is the number of residuals.}$$

The descriptive statistics of D_i is listed in Table 7:

Table 7 Descriptive statistics of residual differences

	Case 1	Case 2	Case 3	Case 4
Mean	-0.31675	-0.1046	3.631204	10.39392
Median	-0.12515	0.02298	0.947449	6.44167
Std.Dev	0.46291	0.5556	5.98067	10.21278
Variance	0.21428	0.30869	35.7684	104.301
Range	1.95234	2.27281	20.29605	31.21326
Interquartile				
Range	0.25502	0.7921	4.06562	19.95277

The means of residuals difference without any missing observations are much closer to zero than the ones with 2 missing observations. Standard deviation, variance, range and interquartile range bear much larger values when missing observations exist. These 4 statistics suggest a significant difference in model performance when missing observations exist, and similarity in model performance without any missing observation.

The t statistic is defined as $t = \frac{\bar{D} - 0}{S_d / \sqrt{n}}$ with $n-1$ degree of freedom. In case 1 and

case 2, we want to test:

$$H_0 : \mathbf{m}_{NN} = \mathbf{m}_{PSO}$$

$$H_a : \mathbf{m}_{NN} < \mathbf{m}_{PSO}$$

where $\mathbf{m}_{NN}$ denotes the residual mean from NN model, and $\mathbf{m}_{PSO}$ denotes the

residual mean from PSO model. This is equivalent to testing: $H_0 : \mathbf{m}_p = 0$
 $H_a : \mathbf{m}_p < 0$

And the t statistic follows a T_{30} distribution. For case 3 and case 4, we want to test:

$$H_0 : \mathbf{m}_{NN} = \mathbf{m}_{PSO}$$

$$H_a : \mathbf{m}_{NN} > \mathbf{m}_{PSO}$$

The t statistics we got are listed in Table 8:

Table 8 T test statistics and p-value

	No-missing		2-Missing	
	t-statistic	p-value	t-statistic	p-value
Noise-free data	-3.809781573	0.9996788698	3.3805061787	0.0010123652
Noisy data	-1.048257316	0.8485538898	5.6665132479	1.7764067E-6

From the results above 2 characteristics are prominent:

- 1) In the case 1 and case 2 that there is no missing observation in input data set, the p-value of either noise-free or noisy data is close to 1, which means there is no significant evidence to reject the hypothesis. And it is concluded that the PSO model is doing as well as NN model.
- 2) In the case 3 and case 4 that there are 2 missing observations in input data set, the p-value of either noise-free or noisy data is close to zero, which means if there are missing values in the input data set, there is significant evidence to reject the hypothesis and it is concluded that PSO model is doing much better than NN model, especially when noise exist.

4.4 PSO Program Running Time

If the data can make program converge, program running time depends on the

accuracy requirement. Otherwise, the running time will depend on the max number of evaluations.

At a computer with Intel® Pentium® III 1133 MHz Mobil CPU and 256M RAM, if the data can not converge (some sensor readings are set far away from valid values), the running time is listed in Table 9.

Table 9 Program maximum running time

Max number of evaluations	Running Time (second)
9400	1.231
10000	1.192
94000	11.570
100000	12.428

So the running time one epoch can be estimated around 1.2×10^{-4} second. The two data sets are used to test the running time. Under the accuracy parameter 0.0000001, running time is in the range [0.090, 0.231] second, which means the program is quick enough for a real time application. And it also shows that the test data will converge in 700~2000 epochs. The accuracy parameter 0.0001 is also tried, and the running time is in [0.064, 0.182] second.

5. Conclusion and Future Work

This thesis novelly uses Particle Swarm Optimization to explore the model parameter space in solving inverse problem. Unlike the conventional gradient search methods that need a good initial point, otherwise will likely trap in local minimum, PSO does not need initial values and it can explore the parameter space entirely and reach the global minimum. In practical inverse problem, most relationships of parameters and measurements is highly nonlinear, hence the new exploration method is more efficient.

More importantly, a robust methodology to handle inverse problem with noisy and missing measurement data is developed. This can be a general framework for applications where measurement may easily contain missing data such as medical imaging, geophysical exploration, remote sensing, and nondestructive evaluation. Incomplete data will also be valuable for the model rather than being discarded as usual.

The methodology can also be used to reduce the data redundancy. If the model works well after erasing one dimension of the data, then this dimension is redundant and can be deleted from the parameter space. In our aircraft fuel measurement case study, one future work is to find out how many sensors can be cut down and which sensors are necessary.

Compare with the conventional neural network model, the running time of our methodology is much longer since it will run 20 times forward model in each epoch (if there are 20 particles). But our case study presents that its normal running time is less than 0.25 second, which is fast enough even for a real time application.

To reduce the affection of random white noise in aircraft fuel measurement, we can try to increase the sampling rate. That is, for one volume, multiple readings are got at

almost one moment. The idea is summarized in Appendix D.

References

- Bishop, C., 1996, *Neural Networks for Pattern Recognition*, New York, Oxford University Press.
- Bonnet, M. and A. Constantinescu, 2005, "Inverse problems in elasticity", *Inverse Problems*, Vol. 21, No. 2, pp. R1-R50.
- Chang, A. T. C. and L. Tsang, 1992, "A neural network approach to inversion of snow water equivalent for passive microwave measurements," *Nordic Hydrol.*, Vol. 23, pp.173-182.
- Clow, B. and T. White, 2004, "An evolutionary race: A comparison of genetic algorithms and particle swarm optimization for training neural networks," *Proceedings of the International Conference on Artificial Intelligence*, pp.582-588, Las Vegas, Nevada, USA, June 21-24.
- Davis, D. T. and J.-N. Hwang, 1997, "Solving inverse problems by bayesian neural network iterative inversion with ground truth incorporation," *IEEE Transactions on Signal Processing*, Vol. 45, No.11, pp. 2749-2757.
- Davis, D. T., Z. Chen, L. Tsang, J. H. Hwang, and A. T. C.Chang, 1993, "Retrieval of snow parameters by iterative inversion of a neural network," *IEEE Trans. Geosci. Remote Sensing*, Vol. 31, pp.842-852.
- Duckett, T., M. Axelsson, and A. Saffiotti, 2001, "Learning to locate an odour source with a mobile robot," *Proceedings of the 2001 IEEE International Conference on Robotics & Automation*, pp. 4017-4022, Seoul, Korea, May 21-26.
- Hagan, M. T. and M. B. Menhaj, 1994, "Training feedforword networks with the Marquardt

- algorithm,” *IEEE Transactions on Neural Network*, Vol. 5, No. 6, pp. 989-993.
- Jiang, D. F, Q. Chen and M. Chen, 2000, “Building fuel measurement model for aircraft fuel tank with neural network,” *Measurement & Control Technology*, Vol. 19, No. 9, pp. 48-50.
- Kennedy, J., and Eberhart, R. C., 1995, “Particle swarm optimization,” *Proc. IEEE International Conference on Neural Networks*, Perth, Australia, IEEE Service Center, Piscataway, NJ, IV: 1942-1948.
- Leshno, M., V. Y. Lin, A. Pinkus, and S. Schocken, 1993, "Multilayer Feedforward Networks with a Nonpolynomial Activation Function Can Approximate Any Function," *Neural Networks*, Vol. 6, No. 6, pp. 861-867.
- Milton J. S. and J. C. Arnold, 1995, *Introduction to Probability and Statistics: Principles and Applications for Engineering and the Computing Sciences*, 3rd ed., New York, NY, McGraw-Hill Inc.
- Mujica, L. E., J. Vehi, J. Rodellar, and P. Koakowski, 2003, “Damage identification by using soft-computing techniques,” *AMAS Workshop on Smart Materials and Structures SMART’03*, pp.261–269, Jadwisin, September 2-5.
- Press, F., 1968, “Earth models obtained by Monte Carlo inversion”, *J. Geo-phys. Res.*, Vol. 73, No. 16, pp. 5223-5234
- Priddy, K. L., M. D. Lothers, and R. E. Saeks, 1993, “Integration of neural network technology in automated fault diagnosis equipment,” *AUTOTESTCON '93. IEEE Systems Readiness Technology Conference*, pp. 397-401, San Antonio, TX, September 20-23.

- Ratnaweera, A., S. K. Halgamuge, and H. C. Watson, 2004, "Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients," *IEEE Transactions on Evolutionary Computation*, Vol. 8, No. 3, pp. 240-255
- Rothman, D. H., 1986, "Automatic estimation of large residual statics corrections", *Geophysics*, Vol. 51, pp. 332-346.
- Studer, M. and K. Peters, 2003, "Combined multi-scale sensing and neural network data fusion for damage identification," *Proceedings of IMECE'03 2003 ASME International Mechanical Engineering Congress* paper # 42867, Washington, D.C., November 15-21.
- Tarantola, A., 1987, *Inverse Problem Theory: Methods for Data Fitting and Model Parameter Estimation*, New York, NY, Elsevier Science Publishing Company Inc.
- Trlifaj, L., 1989, "The Darboux and Abraham-Moses transformations of the one-dimensional periodic Schrodinger equation and inverse problems", *Inverse Problems*, Vol. 5, No. 6, pp. 1145-1155.
- Tsang, L., D. Davis, R. West, Z. Chen., J.-N. Hwang, and D. Winebrenner, 1995, "Passive microwave remote sensing of snow: Scattering in snow based on dense media theory and parametric inversion of snow parameters with an artificial neural networks," in *Passive Microwave Remote Sensing Land-Atmosphere Interaction*, ed. by B.J. Choudhury, Y.H. Kerr, E.G. Njoku, and P. Pampaloni, Utrecht, The Netherlands, VSP Press, pp. 295-314.
- Tsang, L., J. N. Hwang, D. T. Davis, and Z. Chen, 1992, "Multi-parametric inversion in microwave remote sensing with an artificial neural network trained with multiple

scattering theory,” in *Proc. IEEE Antennas Propagat. Soc. Int. Symp.*, vol. 3, pp. 1399, Chicago, IL, July.

Tsang, L., Z. Chen, S. Oh, R. J. Marks, II, and A. T. C. Chang, 1992b, “Inversion of snow parameters from passive microwave remote sensing measurements by a neural network trained with a multiple scattering model,” *IEEE Trans. Geosci. Remote Sensing*, Vol. 30, pp. 1015–1024.

Wang Y. K., J. N. Hwang, C. T. Chen, and L. T. Tsang, 1998, “Neural network inversion of snow parameters by fusion of snow hydrology prediction and SSM/I microwave satellite measurements,” *Proc. ICASSP98*, pp. 1217-1220, Seattle, WA, May 12-15.

Wolters, C. H., L. Grasedyck and W. Hackbusch, 2004, “Efficient computation of lead field bases and influence matrix for the FEM-based EEG and MEG inverse problem”, *Inverse Problems*, Vol. 20, No. 4, pp. 1099-1116.

Uhlmann G. 2003, *Inside Out: Inverse Problems and Applications*, Cambridge, UK: Cambridge University Press.

Zhang, L. J. and S. J. Xu, 1996, “Finite element analysis of planar dielectric waveguide discontinuities”, *International Journal of Infrared and Millimeter Waves*, Vol. 17, No. 10, pp. 1735-1747.

Appendix A : Core Codes of PSO

```
double BPNET::psorun(double minx, double maxx, double *psoinput, double *psooutput, bool
*DontCare)
{

double  c1,cmax; // Confidence coefficient
int  d;          // Current dimension
double  eps;      // Wanted accuracy
double  eps_moyen; // Average accuracy
double  erreur;   // Total error for a given position
double  erreur_prec; // Total error after previous iteration
int  eval_max;    // Max number of evaluations

double  eval_moyen; // Mean number of evaluations
int  fonction[F_max];
int  g;           // Rank of the best informer
int  i;
int  init_liens;  // Flag to (re)init or not the information links
int  k;
int  K;           // Max number of informed particles by a given one
int  m;

double min; // Best result along several runs
double  phi;   // Control exploration/exploitation
int  n_f;      // (multiobjective) number of functions to optimize
int  n_echec;   // Number of failuers
int  n_exec,n_exec_max; // Nbs of executions
int  option_meilleure; // Option for the best informer choice
                        // 0 => really the best (classical)
                        // 1 => using pseudo-gradient
int  s;         // Rank of the current particle

seed_rand_kiss(1); // Initialize KISS
E=exp(1);
pi=acos(-1);

// Control parameters
phi=4.14; //4.14 is usually a good compromise exploration/ exploitation
// Confidence coefficients are computed according to
// a theoretical constriction coefficient
c1=2/(phi-2+sqrt(phi*phi-4*phi));
cmax=c1*phi/2;
S=20; // Swarm size. Typically 20, but may be far smaller or easy problems
```

```

//S=5;

K=3;    // Max number of informed particles by a given one
        // Either 3 or S (thumb rule)
// Information links topology is partly random
// Proximity distributions are classical D-rectangles

//-----
n_f=1;    // Nb of functions to optimize simultaneously
fonction[0]=16;
//by Kai ****fonction[0]=15; //Function code (see perf())
// by Kai ****fonction[1]=12; // Test multiobjective for n_f=2 and fonction[0]=11

// D=3;    // Search space dimension

D=1; // Kai's problem

/* // D-cube data
for (d=0;d<D;d++) // Intervals and granularities
{ xmin[d]=-100; xmax[d]=100; granul[d]=0;}
*/

// Data for pft

xmin[0]=minx; xmax[0]=maxx; granul[0]=0;

// Save if multiobjective
if (n_f>1)
    f_multi=fopen("f_multi.txt","w");

eps=0.0000001;    // Accuracy
fmin.taille=n_f;
for (i=0;i<n_f;i++)
    fmin.fi[i]=0.0; // Objective(s) to reach
//n_exec_max=5;    // Numbers of runs
n_exec_max=2;
eval_max= 10000; // Max number of evaluations
option_meilleure=0; // 1 or 0. Using pseudo-gradient or not

// Initialisation of information variables
n_exec=0; eval_moyen=0; eps_moyen=0; n_echec=0;

init: // Initialisations of positions and velocities
n_exec=n_exec+1;

```

```

for (s=0;s<S;s++) // Loop on particles
{
    P[s].taille=D;
    for (d=0;d<D;d++) P[s].x[d]=alea(xmin[d],xmax[d]);
    V[s].taille=D;
    for (d=0;d<D;d++)
        V[s].v[d]=alea((xmin[d]-xmax[d])/2,(xmax[d]-xmin[d])/2);
    confin_granul(s);
}

// First evaluations
nb_eval=0;

for (s=0;s<S;s++)
{
    P[s].f.taille=n_f;
    for (i=0;i<n_f;i++)
        P[s].f.fi[i]=fabs(perf(P[s],fonction[i],psoinput,psooutput,DontCare)-fmin.fi[i]);
    P_m[s]=P[s]; // Best position = current one
}

// Save the best of the bests
meilleure=P_m[0];
for (s=0;s<S;s++) if (meilleur(P_m[s].f,meilleure.f)==1) meilleure=P_m[s];

// Current min error
erreur_prec=erreur_totale(meilleure.f);
init_liens=1; // So that information links will be initialized

boucle:
if(init_liens==1)
{
    // Who informs who, at random
    for (s=0;s<S;s++) //
    {
        for (m=0;m<S;m=m++) LIENS[m][s]=0;
        LIENS[s][s]=1; // However, each particle informs itself
    }

    for (m=0;m<S;m=m++) // Other links
    {
        for (k=0;k<K;k++){s=alea_entier(0,S-1);LIENS[m][s]=1;}
    }
}

```

```

// The swarm MOVES
for(s=0;s<S;s++) // For each particle ...
{
    g=meilleure_info(s,option_meilleure); // .. find the best informer
    // ... compute new velocity
    for (d=0;d<D;d++)
    {
        V[s].v[d]=c1*V[s].v[d]+alea(0,cmax)*(P_m[s].x[d]-P[s].x[d]);
        V[s].v[d]=V[s].v[d]+alea(0,cmax)*(P_m[g].x[d]-P[s].x[d]);
    }
    // ... and move
    for (d=0;d<D;d++) P[s].x[d]=P[s].x[d]+V[s].v[d];

    // ... interval confinement
    confin_interv(s);

    // ... granularity confinement
    confin_granul(s);

    // ... evaluate the new position
    for(i=0;i<n_f;i++) P[s].f.fi[i]=fabs(perf(P[s],fonction[i],psoinput,psoutil,DontCare)-fmin.fi[i]);

    // ... update the best position
    if (meilleur(P[s].f,P_m[s].f)==1) P_m[s]=P[s];

    // ... update the best of the bests
    if (meilleur(P_m[s].f,meilleure.f)) meilleure=P_m[s];
}

// Check if finished
erreur=erreur_totale(meilleure.f);
// If no improvement, information links will be reinitialized
if(erreur>=erreur_prec) init_liens=1; else init_liens=0;
erreur_prec=erreur;

if (erreur>eps && nb_eval<eval_max) goto boucle;
if (erreur>eps) n_echec=n_echec+1;

// Result display
printf("\n\nExec %i Eval= %i. Value(s) ",n_exec,nb_eval);
for (i=0;i<n_f;i++) printf(" %f",meilleure.f.fi[i]);
printf("\n Position :\n");

```

```

for (d=0;d<D;d++) printf(" %f",meilleure.x[d]);

if (n_exec==1) min=erreur; else if(erreur<min) min=erreur;

// Multiobjective result save
if (n_f>1)
{
    for (i=0;i<n_f;i++)
        fprintf(f_multi,"%f ",meilleure.f.fi[i]);
    for (d=0;d<D;d++) fprintf(f_multi," %f",meilleure.x[d]);
    fprintf(f_multi,"\n");
}

// Compute and display some statistical information
eval_moyen=eval_moyen+nb_eval;
eps_moyen=eps_moyen+erreur;

if (n_exec<n_exec_max) goto init;

eval_moyen=eval_moyen/(double)n_exec;
eps_moyen=eps_moyen/(double)n_exec;
printf("\n\n Eval. (mean)= %f",eval_moyen);
printf("\n Error (mean) = %f",eps_moyen);
printf("\n Success rate = %.2f%%",100*(1- n_echec/(double)n_exec));
if (n_exec>1) printf("\n Best min value = %f",min);

return meilleure.x[0];
}

```

```

double BPNET::perf(struct position x,int fonction, double *psoinput, double *psooutput, bool *DontCare)
{
    // Evaluate the function value at position x
    double c;
    int    D, d;
    int     i,j,k;
    double f,p,xd,x1,x2,x3,x4;
    double sum1,sum2;
    double t0, tt, t1;

    case 16:

```

```

        double *perfinput= new double [NumberOfInput_I];
        int perfi;
        for (perfi=0; perfi<NumberOfInput_I-1;perfi++)

```



```

        perfinput[perfi]=psoinput[perfi];
        perfinput[NumberOfInput_I-1] = x.x[0];
    //
    //  perfinput[NumberOfInput_I-1] = 0.251294246;
    f= Error(perfinput,psoutput,DontCare);

    //f= Error(perfinput,psoutput);
    f=f;
break;

}

return f;
}

```

Figure A.1. Flow chart of PSO

Figure A.2. Screen shot of the program

Appendix B: Scatter Plots of PSO Model Residuals v.s Parameters

PSONF V.S Sensor6

PSONF V.S Sensor7

PSONF V.S Sensor8

PSONF V.S Sensor9

PSONF V.S Sensor10

PSONF V.S Sensor11

PSONF V.S Sensor12

PSONF V.S Sensor13

PSON V.S Sensor6

PSON V.S Sensor7

PSON V.S Sensor8

PSON V.S Sensor9

PSON V.S Sensor10

PSON V.S Sensor11

PSON V.S Sensor12

PSON V.S Sensor13

PSOMNF V.S Actual Value

PSOMNF V.S Pitch

PSOMNF V.S Roll

PSOMNF V.S Sensor1

PSOMNF V.S Sensor2

PSOMNF V.S Sensor3

PSOMNF V.S Sensor4

PSOMNF V.S Sensor5

PSOMNF V.S Sensor6

PSOMNF V.S Sensor7

PSOMNF V.S Sensor8

PSOMNF V.S Sensor9

PSOMNF V.S Sensor10

PSOMNF V.S Sensor11

PSOMNF V.S Sensor12

PSOMNF V.S Sensor13

PSOMN V.S Actual Value

PSOMN V.S Pitch

PSOMN V.S Roll

PSOMN V.S Sensor1

PSOMN V.S Sensor2

PSOMN V.S Sensor3

PSOMN V.S Sensor4

PSOMN V.S Sensor5

PSOMN V.S Sensor6

PSOMN V.S Sensor7

PSOMN V.S Sensor8

PSOMN V.S Sensor9

Legend:

PSOMNF: PSO Model Noise-free data without missing

PSOMN: PSO Model Noisy data without missing

PSOMNF: PSO Model Noise-free data with 2 sensor failures

PSOMN: PSO Model Noisy data with 2 sensor failures

Appendix C: SAS Codes of Paired T Test

```
PROC IMPORT OUT= WORK.residualdata
    DATAFILE= "C:\Documents and Settings\T29632\Desktop\kai\resi
dualdata.xls"
    DBMS=EXCEL REPLACE;
    SHEET="data$";
    GETNAMES=YES;
    MIXED=NO;
    SCANTEXT=YES;
    USEDATE=YES;
    SCANTIME=YES;
RUN;

data residualdata;
set residualdata;
DIFFNF_NM=NNNF-PSONF;
DIFFN_NM=NNN-PSON;
DIFFNF_2M=NNMNF-PSOMNF;
DIFFN_2M=NNMN-PSOMN;
run;

proc means data=residualdata mean std ;
var diffnf_nm diffn_nm diffnf_2m diffn_2m;
output out=residualout mean=meannfnm meannnm meannf2m meann2m std=stdnfnm
stdnnm stdnf2m stdn2m;
run;

data tstatistic;
set residualout;
tnfnm=meannfnm/(stdnfnm/sqrt(31));
tnnm=meannnm/(stdnnm/sqrt(31));
tnf2m=meannf2m/(stdnf2m/sqrt(31));
tn2m=meann2m/(stdn2m/sqrt(31));
pnfnm=1-probt(tnfnm, 30);
pnnm=1-probt(tnnm, 30);
pnf2m=1-probt(tnf2m, 30);
pn2m=1-probt(tn2m, 30);
Label tnfnm="t statistic for noise free no missing" tnnm="t statistic for noisy
no missing"
    tnf2m="t statistic for noise free 2 missing" tn2m="statistic for noisy
2 missing"
    pnfnm="p-value for noise free no missing" pnnm="p-value for noisy
no missing"
```

```
pnf2m="p-value for noise free 2 missing" pn2m="p-value for noisy 2
missing";
drop meannfnm meannnm meannf2m meann2m stdnfnm stdnnm stdnf2m stdn2m _type_
_freq_;
run;
```

Appendix D: Proof of Reduce White Noise by Increasing Sample Rate

Consider a noisy reading $g(x, t)$ which is formed by the addition of noise $h(x, t)$ to an original reading $f(x, t)$;

$$g(x, t) = f(x, t) + h(x, t)$$

Now we add the set of noisy readings $\{g_i(x, t)\}$ and we get:

$$\bar{g}(x, t) = \frac{1}{M} \sum_{i=1}^M g_i(x, t)$$

Then it follows that

$$E\{\bar{g}(x, t)\} = f(x, t)$$

And

$$s_{g(x, t)}^2 = \frac{1}{M} s_{h(x, t)}^2$$

Where $E\{\bar{g}(x, t)\}$ is the expected value of $\bar{g}(x, t)$, $s_{g(x, t)}^2$ and $s_{h(x, t)}^2$ are the variances of $\bar{g}(x, t)$ and $h(x, t)$.

The equations above indicate that as M increases, the variability of the probe readings decreases and the mean $\bar{g}(x, t)$ will approach $f(x, t)$.