RegNeN

(**Reg**ression by **Ne**ural **N**etwork)

T. Mareš[1], L. Zrůbek[2], A. Kučerová[3]

2013

[1] Tomáš Mareš, Department of Mechanics, Faculty of Civil Engineering, Czech Technical University in Prague, email: tomas.mares.1@fsv.cvut.cz

[2] Ing. Lukáš Zrůbek, Department of Mechanics, Faculty of Civil Engineering, Czech Technical University in Prague, email: lukas.zrubek@fsv.cvut.cz

[3] Ing. Anna Kučerová, Ph.D., Department of Mechanics, Faculty of Civil Engineering, Czech Technical University in Prague, email: anicka@cml.fsv.cvut.cz

Contents

1 Introduction

RegNeN (the abbreviation of Regression by Neural Network) is a C/C++ software package for computing a regression for given data using Artificial Neural Network (ANN). Software is divided in to two separate parts `nnTrainer` which is used for training of neural network and `nnEvaluator` which is used for evaluating given experiment. Both parts need to be compiled with static library containing routines for `sqlite3`, `xml` and `cmaes`.

Usage of RegNeN should not be dependent on the used operating system, but minor problems can arise when compiling for non-traditional operating systems. Compiling for Unix and Microsoft Windows is described in section 3.

This software was created with support of the Czech Science Foundation through project No. $105/11/P370$.

2 About RegNeN

2.1 nnTrainer & nnEvaluator source files

This software is distributed as source files, so the logical first step at using RegNeN is compilation. In the table 2.1 are listed all sources files with an indication for which target is particular file needed.

Filename	Size	Target		Commentary
		nnTrainer	nnEvaluator	
BP.cpp	5 kB	○	○	
BP.h	1 kB	○	○	
cmaes.h	96 kB	○	○	header files for static library
cmaesinterface.h	5 kB	○	○	
DataManager.cpp	13 kB	○	○	
DataManager.h	6 kB	○	○	
dbDataUtils.cpp	15 kB	○	○	
dbDataUtils.h	3 kB	○	○	
evallogger.cpp	2 kB		○	
evallogger.h	1 kB		○	
evaluaterMain.cpp	4 kB		○	main function for nnEvaluator
general.cpp	5 kB	○	○	
general.h	2 kB	○	○	
makefile	2kB			makefile for Unix system
mtwister.cpp	6 kB	○	○	
mtwister.h	3 kB	○	○	
Neuronka.cpp	25 kB	○	○	
Neuronka.h	5 kB	○	○	
nnExternal		○	○	!! static library !!
NNtraining.cpp	8 kB	○	○	
NNtraining.h	2 kB	○	○	
pugiconfig.hpp	3 kB	○	○	header files for static library
pugixml.hpp	45 kB	○	○	
RecurrentNN.cpp	8 kB	○	○	
RecurrentNN.h	2 kB	○	○	
scalingann.cpp	8 kB	○	○	
scalingann.h	2 kB	○	○	
sqlite3.h	325 kB	○	○	header file for static library
Standartizor.cpp	11 kB	○	○	
Standartizor.h	5 kB	○	○	
Statistics.h	4 kB	○	○	
trainMain.cpp	4 kB	○		main function for nnTrainer

Table 2.1: Source files with indication for which target is particular file needed.

2.2 Static library nnExternal source files

For the static library are used publicly available libraries from various authors, see table 2.2.

filename	size	commentary
cmaes.cpp cmaes.h cmaes_interface.h	96 kB 5 kB 3 kB	Covariance Matrix Adaptation Evolution Strategy by Nikolaus Hansen
pugixml.cpp pugixml.hpp pugiconfig.hpp	267 kB 45 kB 3 kB	Light-weight C++ XML processing library by Arseny Kapoulkine
sqlite3.c sqlite3.h	4601 kB 325 kB	Amalgamation of many separate C source files from SQLite

Table 2.2: Files needed to compile static library

3 Compilation

3.1 Unix systems

Compilation on Unix system should be carried by `g++` compiler (with partial support of C++0x), `gcc` compiler and performed on included makefiles. To compile static library use command `make` in folder `./nnExternal` which contain prepared makefile. To compile only the `nnTrainer` use the command `make train` and for compiling only the `nnEvaluator` use the command `make eval`. To compile `nnTrainer` and `nnEvaluator` at once, use command `make`.

3.2 Microsoft Windows

The compilation of RegNeN on Microsoft Windows systems, could be done by many methods and software solutions, but we are only describing compilation using Microsoft Visual Studio software. Remember that the used compiler need to have partial support of C++0x.

MS Visual Studio:
At first, you need to compile a static library. We recommend to create a *Win32 Project* and follow *Win32 Application Wizard* to create a project for static library (*.lib). Then include all necessary files (all files listed in table 2.2) and compile them into a static library with name of your choice.

For `nnTrainer` and `nnEvaluator` you can create multi-target project or separate project for each part. Type of both targets should be *Console Application*. Then include all necessary files (all files listed in table 2.1) for each target respectively. Do not forget to include already compiled static library.

4 SQLite

Inputs and outputs of RegNeN software are performed using free to use embedded relational SQL database engine called SQLite. But because everything about SQLite has already been written elsewhere, we will provide only brief information in the following paragraphs. For further information we would like to refer you to the SQLite homepage `http://www.sqlite.org/`.

4.1 Installation

Because of precompiled binaries for various operating systems, the installation should be fairly easy. The more so if you are using Unix OS. Then you just have to check if you already have SQLite installed on your machine, because today almost all the flavors of Unix OS are being shipped with SQLite. On the other hand if you have not SQLite installed or you are using Microsoft Windows you need to download and install SQLite. Just go to `http://www.sqlite.org/download.html` and download respective files. For Unix OS we recommend *sqlite-autoconf-*.tar.gz* and for Microsoft Windows *sqlite-shell-win32-*.zip* and *sqlite-dll-win32-*.zip*. On Unix follow the following steps:

```
$tar xvfz sqlite-autoconf-3071502.tar.gz
$cd sqlite-autoconf-3071502
$./configure --prefix=/usr/local
$make
$make install
$
```

On Microsoft Windows create a folder (for example `C:\sqlite`) and unzip two downloaded files in this folder which will give you `sqlite3.def`, `sqlite3.dll` and `sqlite3.exe` files. Then you can add `C:\sqlite` in your PATH environment variable and finally go to the command prompt and issue `sqlite3` command, which should display a result something as below.

```
C:\>sqlite3
SQLite version 3.7.15.2 2013-01-09 11:53:05
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
```

Note: You do not need to add `C:\sqlite` in your PATH environment variable, but then you have to use the `sqlite3` command only in directory where you have the unzipped files `sqlite3.def`, `sqlite3.dll` and `sqlite3.exe`.

4.2 Usage

Using SQLite is same on both systems Unix an Microsoft Windows. First you need to create and fill your first database. In order to do that, we recommend to use any kind on database manager, or use SQLite commands in command prompt. For further information please see following tutorials: `http://www.tutorialspoint.com/sqlite/index.htm` or `http://zetcode.com/db/sqlite/`.

5 Usage of RegNeN

5.1 Preprocessing & input files

As mentioned above the inputs and outputs of data to or from RegNeN are performed using SQL database engine called SQLite. But before SQLite can do the work for you, you need to tell SQLite which database to use, which table and data from table to use and optionally how to scale them. For that purpose the input XML file in specified format is used, see the example below.

```
<?xml version="1.0"?>
<DBPATTERNS>
    <DBPATTERN>
        <INPUTS>
            <SCALE type="LIN" min="-1" max="1">
                <COLUMN>name_of_column</COLUMN>
            </SCALE>
            <SCALE type="LOG" min="-1" max="1">
                <COLUMN>name_of_column</COLUMN>
            </SCALE>
            <SCALE type="EXP" min="-1" max="1">
                <COLUMN>name_of_column</COLUMN>
            </SCALE>
        </INPUTS>
        <OUTPUTS>
            <SCALE type="LIN" min="-1" max="1">
                <COLUMN>name_of_column</COLUMN>
            </SCALE>
            <SCALE type="LOG">
                <SCALE type="LIN" min="-1" max="1">
                    <COLUMN>name_of_column</COLUMN>
                </SCALE>
            </SCALE>
        </OUTPUTS>
        <SET>WHERE p_id BETWEEN 1 and 10</SET>
        <SET>WHERE p_id BETWEEN 11 AND 20</SET>
        <SET>WHERE p_id BETWEEN 21 AND 30</SET>
        <DB>path_to_database</DB>
        <TAB>name_of_table</TAB>
    </DBPATTERN>
</DBPATTERNS>
```

Every used tag have to begin with opening tag `<TAG>` and end with closing tag `</TAG>`. The XML file start with this first line `<?xml version="1.0"?>`. The next tags `<DBPATTERNS>` and nested `<DBPATTERN>` are mandatory and every other tags should be nested inside of them.

After that follows the tag `<INPUTS>` to specify input data. Each column of input table have to be specified by its name between tags `<COLUMN>` and `</COLUMN>`. The name of column need to correspond with name of column in database. The input data can also be scaled. To do that just surround the particular column specification by tag `<SCALE type="LIN" min="-1" max="1">`. Parameter `type` set type of scaling which can be linear `"LIN"`, logarithmic `"LOG"` and exponential `"EXP"`. Optional parameters `min` and `max` set minimal and maximal values for scaling. Scaling can be also nested in another scaling as shown in example.

The tag `<OUTPUTS>` stands on the same level as tag `<INPUTS>`. All tags to specify the outputs, scaling of outputs and name of the columns which stores the results are same as in the input section. These outputs are needed for the ANN training phase as a reference results.

Following tags after outputs specify the cross-validating sets. As you can see in the example, the tag `<SET>` is used three times. The number of repetition of this tag determines number of sets for cross-validation. Each set is given by this specific format:

```
<SET>WHERE p_id BETWEEN x and y</SET>
```

Where `p_id` is name of column storing the primary keys. In other words your table have to contain column with numbers from 1 to n, where n is number of rows of the table. The x stands for starting row number of set and y for last row of set. The cross-validating sets don't need to bee equal size, but need to include all rows of table. There have to be at least two sets.

Before the closing tags `</DBPATTERN>` and `</DBPATTERNS>` need to be tags `<DB> </DB>` which surrounds the path to database and `<TAB> </TAB>` tags which surrounds name of table in database.

5.2 Training of Neural Network

After creating the input database and the configuration file you have to run the training part of RegNeN. To do that just execute `nnTrainer` with parameters `-d` which specifies the name[4] of the configuration file (`*.xml` type) and the parameter `-o` specifying the name[4] of the output file which will contain the trained ANN (also `*.xml` type). Additional `*.csv` files containing information about adaptation process are also created, but for these there is no configuration. If anything goes wrong during the process, the errors will be saved in the file named `ERRORS.txt`. Example of the executing command can be seen below.

[4]Absolute or relative paths can be used instead of names.

```
nnTrainer -d configuration_file.xml -o trained_ANN.xml
```

5.3 Evaluation of experiment using Neural Network

To use the trained neural network and evaluate some data the configuration file will be needed. If you want to evaluate the same data that was used for training, do not change anything in the configuration file. The outputs and cross-validation sets will be ignored[5] (output data will not be overwritten). To evaluate new data the name of table or path to database have to be changed. Do not change anything else because only data with the same format as the data used for training can be evaluated.

To run the process the `nnEvaluator` need to be executed with three mandatory and one optional parameter. The first parameter is `-d` and is followed by name[4] of the configuration file. Second parameter is `-n` with the name[4] of the output file from the training phase. The third parameter is `-o` with name[4] of the file to store results (`*.csv` type). And the fourth parameter `-e` is followed by the name[4] of file to store calculated differences between given values and values predicted by ANN (also `*.csv` type). This last parameter is optional and if it is not given the differences will not be calculated. And same as in the training phase all errors will be saved in file named `ERRORS.txt` and example of the executing command can be seen below.

```
nnEvaluator -d configuration_file.xml -n trained_ANN.xml -o output_file.csv
-e difference.csv
```

[5]Depending on the input parameters the outputs will be ignored or used to calculate difference between given output value and the prediction given by ANN. In any case the output values need to have some value.

6 Theory

6.1 Introduction

Development in numerical modeling provides the possibility to describe a lot of complex phenomena in material or structural behavior. The resulting models are, however, often highly nonlinear and defined by many parameters, which have to be estimated so as to properly describe the investigated system and its behavior. The aim of the model calibration is thus to rediscover unknown parameters knowing the experimentally obtained response of a system to the given excitations. The principal difficulty of model calibration is related to the fact that while the numerical model of an experiment represents a well-defined mapping from input (model, material, structural, or other) parameters to output (structural response), there is no guarantee that the inverse relation even exists.

The most broadly used approach to parameter identification is usually done by means of an error minimisation technique, where the distance between parameterised model predictions and observed data is minimised [21]. Since the inverse relation (mapping of model outputs to its inputs) is often ill-posed, the error minimisation technique leads to a difficult optimisation problem, which is highly nonlinear and multi-modal. Therefore, the choice of an appropriate identification strategy is not trivial.

Another approach intensively developed during the last decade is based on Bayesian updating of uncertainty in parameters' description [15, 14]. The uncertainty in observations is expressed by corresponding probability distribution and employed for estimation of the so-called posterior probabilistic description of identified parameters together with the prior expert knowledge about the parameter values [10, 22]. The unknown parameters are thus modelled as random variables originally endowed with prior expert-based probability density functions which are then updated using the observations to the posterior density functions. While the error minimisation techniques lead to a single point estimate of parameters' value, the result of Bayesian inference is a probability distribution that summarizes all available information about the parameters. Another very important advantage of Bayesian inference consists in treating the inverse problem as a well-posed problem in an expanded stochastic space.

Despite the progress in uncertainty quantification methods [16, 19], more information provided by Bayesian inference is generally related to more time-consuming computations. In many situations, the single point estimate approach remains the only feasible one and development of efficient tools suitable for this strategy is still an actual topic. Within the several

last decades, a lot of attention was paid to the so-called intelligent methods of information processing and among them especially to soft computing methods such as artificial neural networks (ANNs), evolutionary strategies or fuzzy systems [8]. A review of soft computing methods for parameter identification can be found e.g. in [13]. In this paper, we focus on applications of ANNs in the single point approach to parameter identification.

6.2 Artificial Neural Network

Artificial neural networks (ANNs) [3, 4] are powerful computational systems consisting of many simple processing elements - so-called neurons - connected together to perform tasks analogously to biological brains. Their main feature is the ability to change their behaviour based on an external information that flows through the ANN during the learning (training) phase.

A particular type of ANN is the so-called feedforward neural network, which consists of neurons organized into layers where outputs from one layer are used as inputs into the following layer, see figure 1. There are no cycles or loops in the network, no feed-back connections. Most frequently used example is the multi-layer perceptron (MLP) with a sigmoid transfer function and a gradient descent method of training called the back-propagation learning algorithm. In practical usage, the MLPs are known for their ability to approximate non-linear relations and therefore, when speaking about an ANN, the MLP is considered in the following text.

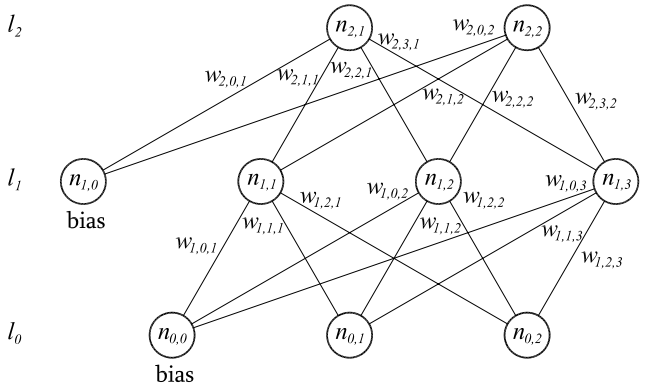

Figure 1: Architecture of multi-layer perceptron

The input layer represents a vector of input parameters which are directly the outputs of the input layer. The outputs of one layer are multiplied by a vector of constants, the so-called synaptic weights, summarized and used as inputs into the following layer. Elements in the hidden and output layers - neurons - are defined by an activation function, which is applied on

the input and produces the output value of the neuron. The synaptic weights are parameters of an ANN to be determined during the training process. The type of the activation function is usually chosen in accordance with the type of a function to be approximated. In the case of continuous problems, the sigmoid activation function is the most common choice.

One bias neuron is also added into the input and hidden layers. It does not contain an activation function, but only a constant value. Its role is to enable to shift the value of a sum over the outputs of his neighbouring neurons before this sum enters as the input into the neurons in the following layer. The value of biases is determined by training process together with the synaptic weights.

Despite of ANN's popularity there are only few recommendations for the choice of ANN's architecture. The authors, e.g. in [7, 6]], show that the ANN with any of a wide variety of continuous nonlinear hidden-layer activation functions and one hidden layer with an arbitrarily large number of units suffices for the "universal approximation" property. Therefore, we limit our numerical experiments to such case. The number of units in the input and the output layer is usually given by the studied problem itself, but there is no theory yet specifying the number of units in the hidden layer.

To overcome this problem, we use two sets of data for ANN's preparation: training data are used for calibration of the synaptic weights of the ANN with a chosen number of hidden units and the resulting ANN is then evaluated on independent validation data. Then, one hidden neuron is added to the existing ANN, which is again trained, evaluated on validation data and the ratio between the obtained error to the error obtained for the previous ANN is computed. We count the situations, where the ratio is higher than 0.99. When these situations occur three times, the addition of hidden neurons is stopped. Then the ANN with the smallest error on validation data is employed for model calibration.

6.3 Strategies for Model Calibration

In overall, there are two main philosophies for application of ANN in identification problems. In a forward mode/direction, the ANN is applied to approximate the model response. The error minimisation technique then becomes a minimisation of distance between the ANN's predictions and experimental data. The efficiency of this strategy relies on the evaluation of the trained ANN to be significantly much faster than the full model simulation. The advantage of this strategy is that the ANN is used to approximate a known mapping which certainly exists and is well-posed. Computational costs of this strategy are separated in two parts of a similar size: (i) the ANN training - optimisation of synaptic weights and (ii) the

minimisation of error in ANN prediction for experimental data - optimisation of ANN inputs (i.e. determination of investigated model parameters). The latter part concerns optimisation of an error function which is often multi-modal, non-differentiable and some robust optimisation method has to be applied to solve this problem. An important shortcoming of this method is that this ill-posed optimisation problem needs to be solved repeatedly for any new experimental measurement. This way of ANN application to the parameter identification was presented e.g. in [1], where an ANN is used for predicting load-deflection curves and the conjugate directions algorithm is then applied for optimisation of ductile damage and fracture parameters. Authors in [18] train an ANN to approximate the results of FE simulations of jet-grouted columns and optimise the column radius and a cement content of the columns by a genetic algorithm. Principally same methods are used for identification of elasto-plastic parameters in [2].

The second philosophy, an inverse mode, assumes the existence of an inverse relationship between the outputs and the inputs of the calibrated model. If such a relationship exists at least on a specified domain of parameters' value, it can be approximated by an ANN. Then the retrieval of desired inputs is a matter of seconds and could be easily executed repeatedly for any new experiment and no other optimisation process is necessary. Here the ANN training represents the whole computational costs and a solution of the ill-posed problem. This way of ANN application to parameter identification was presented e.g. in [17] for identification of mechanical material parameters, in [23] for estimation of elastic modulus of the interface tissue on dental implants surfaces, in [24] for identification of interfacial heat transfer coefficient or in [12] for determination of geometrical parameters of circular arches.

In computational mechanics, there is often a disproportion between the number of inputs and outputs of a numerical model. While the model has usually only several parameters (inputs), their response is mostly described by a load-deflection curve, stress or strain fields. In other words, the response is usually a quantity defined in discredized spatial, time and/or pseudo-time domain. Generally, these domains can be easily parameterised. In the forward mode of identification, the problem of many outputs can be handled e.g. by including the domain parameters among the ANN inputs and thus reducing the number of outputs. An approximation of the complete model response is then obtained by repeated evaluation of the ANN with varying values of domain parameters. In the inverse mode, the situation is more difficult, because usage of high number of inputs leads to excessive complexity of the ANN architecture and the training process. Fortunately, particular components of a model response are usually highly correlated and thus the principal component analysis (PCA) [11]

can be easily applied to transform them into a smaller number of uncorrelated quantities ordered according to their variance (i.e. significance). Then only a selected number of most important principal components can be used as ANN's inputs.

Since the ANN training needs a preparation of a set of training data, it is also worthy to use these data for a sampling-based sensitivity analysis [5, 20] and obtain some information about importance of particular observations or significance of each parameter for a system behaviour. To achieve some reliable information from sensitivity analysis as well as a good approximation by ANN, one has to choose the training data carefully according to a suitable design of experiments, see e.g. [9] for a competitive comparison of several experimental designs.

7 Future plans for RegNeN

- Unified compilation for different operating systems using CMake (`http://www.cmake.org/`)
- Create a project on GitHub for RegNeN (`https://github.com/`)

References

[1] M. Abendroth and M. Kuna. Identification of ductile damage and fracture parameters from the small punch test using neural networks. *Engineering Fracture Mechanics*, 73:710–725, 2006.

[2] H. Aguir, H. BelHadjSalah, and R. Hambli. Parameter identification of an elasto-plastic behaviour using artificial neural networks–genetic algorithm method. *Materials and Design*, 32:48–53, 2011.

[3] K. N. Gurney. *An introduction to neural networks*. UCL Press, London, 2002.

[4] S. S. Haykin. *Neural networks and learning machines*. Prentice Hall/Pearson, New York, 3rd edition, 2009.

[5] J. Helton, J. Johnson, C. Sallaberry, and C. Storlie. Survey of sampling-based methods for uncertainty and sensitivity analysis. *Reliability Engineering & System Safety*, 91:1175–1209, 2006.

[6] K. Hornik. Some new results on neural network approximation. *Neural Networks*, 6:1069–1072, 1993.

[7] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[8] Jyh-Shing Roger Jang and Chuen-Tsai Sun. *Neuro-fuzzy and soft computing: a computational approach to learning and machine intelligence*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.

[9] Eliška Janouchová and Anna Kučerová. Competitive comparison of optimal designs of experiments for sampling-based sensitivity analysis. *Computers & Structures*, 2011.

[10] E. T. Jaynes. *Probability Theory, The Logic of Science*. Cambridge University Press, Cambridge, 2003.

[11] I. T. Jolliffe. *Principal Component Analysis*. Springer–Verlag, 2^{nd} edition, 2002.

[12] M. Klos and Z. Waszczyszyn. Modal analysis and modified cascade neural networks in identification of geometrical parameters of circular arches. *Computers and Structures*, 89:581–589, 2011.

[13] A. Kučerová. *Identification of nonlinear mechanical model parameters based on soft-computing methods*. PhD thesis, Ecole Normale Supérieure de Cachan, Laboratoire de Mécanique et Technologie, 2007.

[14] A. Kučerová, J. Sýkora, B. Rosić, and H. G. Matthies. Acceleration of uncertainty updating in the description of transport processes in heterogeneous materials. *Journal of Computational and Applied Mathematics*, 2012. In press, e-print: arXiv:1201.0942.

[15] Y.M. Marzouk, H.N. Najm, and L. Rahn. Stochastic spectral methods for efficient Bayesian solution of inverse problems. *Journal of Computational Physics*, 224(2):560–586, 2007.

[16] H. G. Matthies. *Encyclopedia of Computational Mechanics*, chapter Uncertainty Quantification with Stochastic Finite Elements. John Wiley & Sons, Ltd., 2007.

[17] D. Novák and D. Lehký. ANN inverse analysis based on stochastic small-sample training set simulation. *Engineering Applications of Artificial Intelligence*, 19:731–740, 2006.

[18] B. Pichler, R. Lackner, and H. A. Mang. Back analysis of model parameters in geotechnical engineering by means of soft computing. *International journal for numerical methods in engineering*, 57:1943–1978, 2003.

[19] B. Rosić, A. Litvinenko, O. Pajonk, and H. G. Matthies. Direct bayesian update of polynomial chaos representations. *Journal of Computational Physics*, 2011. submitted for publication.

[20] A. Saltelli, K. Chan, and E. M. Scott. *Sensitivity analysis*. NY:Wiley, New York, 2000.

[21] G.E. Stavroulakis, G. Bolzon, Z. Waszczyszyn, and L. Ziemianski. 3.13 - inverse analysis. In Editors in Chief: I. Milne, R. O. Ritchie, , and B. Karihaloo, editors, *Comprehensive Structural Integrity*, pages 685 – 718. Pergamon, Oxford, 2003.

[22] Albert Tarantola. *Inverse Problem Theory and Methods for Model Parameter Estimation*. Society for Industrial and Applied Mathematics, 2005.

[23] K. Zaw, G. R. Liu, B. Deng, and K. B. C. Tan. Rapid identification of elastic modulus of the interface tissue on dental implants surfaces using reduced-basis method and a neural network. *Journal of Biomechanics*, 42:634–641, 2009.

[24] L. Zhang, L. Li, H. Ju, and B. Zhu. Inverse identification of interfacial heat transfer coefficient between the casting and metal mold using neural network. *Energy Conversion and Management*, 51:1898–1904, 2010.