

Appendix D - UNIVERSITY HONORS PROGRAM
SENIOR PROJECT - APPROVAL

Name: William Noid

College: Arts and Sciences Department: Chemistry

Faculty Mentor: _____

PROJECT TITLE: Artificial Neural Networks:

A Modern Method for the Deconvolution of Mass Spectra

I have reviewed this completed senior honors thesis with this student and certify that it is a project commensurate with honors level undergraduate research in this field.

Signed: Theresa J. Cook, Faculty Mentor

Date: 5/8/00

Comments (Optional):

ARTIFICIAL NEURAL NETWORKS:

A Modern Technique for the Deconvolution of Mass Spectra

William Noid

directed by

Dr. Kelsey Cook

Professor of Chemistry
University of Tennessee

in collaboration with

Dr. Bobby Sumpter

Senior Scientist
Oak Ridge National Laboratory

Artificial Neural Networks: A Modern Technique for the Deconvolution of Mass Spectra

William Noid.

Dr. Kelsey Cook, Primary Investigator. Professor of Chemistry, University of Tennessee, Knoxville.

Dr. Bobby Sumpter. Senior Scientist, Oak Ridge National Laboratory.

Abstract

A primary goal of the Cook group is the development of industrially useful methods of quantitative analysis. One potential technique for on-line analysis that is under-utilized today is mass spectrometry. Although, current applications rely upon linear fitting of mass spectra to basis spectra, there exists considerable non-linearity within the spectra.

Since the re-discovery of the back-propagation training algorithm by Rummelhart, Hinton, and Williams in the mid 1980s, artificial neural networks have enjoyed great popularity as a means for non-linear prediction.¹ They have been successfully applied to a wide range of problems, ranging from the prediction of exons within DNA and the analysis of UV spectra, to stock market predictions and medical diagnoses.² However, neural networks have not been used to predict the composition of mixtures from mass spectra.

The greatest advantage of artificial neural networks stems from their ability to learn complex patterns from examples. Unlike other predictive methods, the user need not provide any direction or algorithm for the network to utilize. Instead, through a training process, the network "learns" to predict target output from input with increasing accuracy.

In this project, artificial neural networks predict the composition of a lubrisol chemical stream, consisting of methane, ethane, propane, propene, isobutene, and isobutane. This stream includes two sets of compounds differing only in the presence of a double bond, so that the mass spectra for the pure individual compounds are similar.

In the work, a number of different network paradigms are utilized, including Generalized Regression Neural Networks, as well as traditional feed-forward back-propagation neural networks. In addition to training networks with experimental spectra, networks were also trained using the difference between the spectra and linear combinations of spectra for pure compounds to predict solute concentration. Finally, to compensate for the limited experimental data, networks were trained using artificial spectra synthesized by adding empirical non-linear terms to linear combinations of spectra.

Table of Contents

Homology of artificial to biological Neural Networks	1
Feed forward back-propagation neural networks	6
Radial basis function networks	15
Bprop	21
Experimental	25
Results	29
Conclusion	43
References	46
Appendix	
Illustrations	i
Figures	vii
Tables	xvii
Plots of Rprop convergence	xxviii
Relevant codes	xlii
M/Z peaks studied	liv
Honor's statement	lv

Homology of Artificial to Biological Neural Networks:

As the name would suggest, artificial neural networks present many similarities to biological networks of neurons, particularly the human brain. The brain possesses capabilities that remain far beyond the grasp of the most well-developed traditional computer systems. A great deal of early research in the field of neural networks was directed toward preparing models of the brain that would attain these capabilities.¹ These efforts included original work towards developing artificial intelligence. It was at one time believed that a sufficiently complex neural network properly organized would actually approach the abilities of biological neural networks.¹ In the past, networks have been designed with the intent of emulating the brain's capabilities by approximating its structure and operation. However, these hopes, although much publicized, were never achieved and today many neural network researchers try to avoid such comparisons with biological systems in order to prevent unrealistic expectations. Today a great deal of research and most applications are designed without considering the biological relationships. Nevertheless, considering the biological analogies can be instructive both in comprehending the topology of neural networks, and in understanding many of the goals and early ideas in their development.

The brain is one of the largest organs in the human body and is certainly the least well-understood. Even today there exists an air of mystery as to the complex inner workings of the brain that result in capabilities including comprehension, memory, and even emotion. Weighing 1.35 kg on average, the brain consists of over 100 billion nerve cells. These nerve cells, referred to as neurons, are the functional units of the brain. These neurons are highly interconnected; each receives information from and transmits information to anywhere from 10^3 to 10^5 other neurons. Neurons receive information in the form of chemical signals (neurotransmitters) from other neurons across

regions between the cells, called synapses. Each neuron then processes these signals autonomously.

Each neuron contains three distinct segments as seen in illustration 1.² The soma is the main cell body of a neuron. It contains most of the organelles necessary for the survival of the cell including the nucleus and the machinery necessary for respiration and the production of neurotransmitters. The dendrites are thinner elongated extensions from the soma that branch out to many termini and receive signals across synapses in the form of chemical neurotransmitters from the axons of other neurons. The third distinct feature of a neuron, the axon, is a long single extension from the soma which serves to propagate information from the neuron to other neurons across synapses to the dendrites and cell bodies of other neurons.³

There exists a potential difference (voltage) of roughly -70 mV across the cell membrane of resting neurons between the cytosol and the cellular environment. When a neuron receives a signal across a synapse, ion gate channels act to change the electric potential across the membrane. Neurons differentiate between input from different synapses and a neuron's sensitivity to a given synapse can change in time. Signals that increase the potential difference across the membrane hyperpolarize the receiving cell and inhibit future action. Signals that depolarize the membrane by reducing the potential difference move the neuron closer to activation. When a signal of sufficient strength, or a sequence of signals having sufficient net effect, depolarize the cell membrane beyond a threshold voltage, typically -50 mV, the cell responds with an action potential that rapidly depolarizes the membrane and inverts the potential difference across the membrane. This potential is transferred across the neuron and along the axon, at a rate ranging from several cm/sec to 100 m/sec, where it is transmitted across the synapse to other neurons.³

Although these action potentials are not observed to travel across the length of a

neuron faster than 100 m/sec, the human brain is able to recognize objects from noisy or incomplete information at a rate unparalleled by modern technology. It is believed that the massively parallel organization allows the human brain to perform complex tasks so rapidly. Each neuron processes input entirely independently and propagates its output to thousands of other neurons at once.³

Many of these physiological characteristics of the human brain are mimicked by artificial neural networks. The computational unit of an artificial neural network, the node (or artificial neuron), mirrors many of the properties of a biological neuron. Each node receives information (data) from a number of nodes in the previous layer. These signals are weighted by a dynamic factor according to past experience. Nodes process this weighted input autonomously.

In early neural network paradigms the transfer function used was often a step function and exactly analogous to a threshold voltage for depolarization. In these networks, the output of the node was usually a binary digit, either a 1 or 0 - analogous to whether the node would propagate its signal forward or not. The network only propagated an active signal, 1, if the node's weighted input surpassed this threshold. The inputs to a node could be either inhibitory or activating, moving the network either closer to or further from the activation threshold. Step and threshold functions are rarely utilized as transfer functions today. However, the original concept is latent in the use of a sigmoidal transfer function, ($y = 1/[1+\exp(-k*x)]$; illustration 2) since as the scaling factor, k , goes to infinity the sigmoid approaches a step.⁴ Furthermore, although the output of nodes within modern artificial neural networks are rarely confined to binary digits, the biological analogy remains relevant if one considers the output to represent the frequency of firing.

The organization of the artificial neurons into a network was designed to emulate the complicated pattern of connections within a human brain. Illustration 3 shows a common neural

network topology. Each node is connected to a number of other independently processing nodes. The weights differ between nodes in much the same way that neurons distinguish between information from different synapses. Furthermore, the weighting factors connecting nodes are dynamic. They are adjusted in training to reduce prediction errors in a way analogous to the sensitivity of a neuron to information from a synapse changing in time. In humans this process of a neuron adjusting its sensitivity to information from different synapses is associated with learning.²

Current models of the brain give no indication how the brain is able to perform such remarkable feats with incredible accuracy, precision, and efficiency. Perhaps even more surprising, though, is that artificial neural networks have managed to successfully emulate some of these abilities. In spite of the incredibly crude approximation of the human brain contained in artificial neural networks, and in spite of the fact that some of the principles incorporated in earlier models that were believed to be analogous to physiological brains are now rejected, artificial neural networks have demonstrated the ability to approximate many of the capabilities of the human brain. These capabilities include the following:

- 1) The human brain allows pattern recognition at an incredibly rapid rate and with remarkable accuracy in the presence of noisy and incomplete input that would paralyze a standard computer algorithm. For example, in the presence of a large crowd people in a room, humans are able to filter out the background conversations and converse with a single individual.

- 2) Humans are able to generalize from specific experiences to act in unfamiliar environments. In this manner people are able to properly act under conditions slightly different from prior experience.

- 3) Most significantly, humans are able to learn. From experiencing the consequences of past

actions, people develop knowledge as to the correct action.

4) This learning often involves humans recognizing the appropriate action without ever having been taught explicit rules to guide these actions. This results in people having significant difficulty in developing a simple algorithm to direct the activity and decision making process necessary for implementing computer or machine algorithms for actions that are natural for humans.

General References:

1. Smith, Murray. *Neural Networks for Statistical Modeling*. Van Nostrand Reinold: New York, New York. 1993.
2. Lawrence, Jeannette. Sylvia Luedeking, ed. *Introduction to Neural Networks*. California Scientific Software: Grass Valley, California. 1991.
3. Campbell, Neil A. *Biology*. 3rd ed. Benjamin/Cummings Publishing Company, Inc.: New York, New York. 1993.
4. Wasserman, Philip D. *Neural Computing: Theory and Practice*. Van Nostrand Reinhold: New York, New York. 1989.

Feed-forward back-propagation neural networks:

The feed forward back-propagation artificial neural network is the most straightforward but also the most commonly used paradigm. The network operates in two modes. In the forward pass an input vector is applied to the network, which then performs a series of operations dependent upon the current state of the network and predicts the output. In feed forward models, information only passes in one direction. The output of each stage in the computation is always passed on to the next stage closer to the output level.

The second mode of the network is training. In backpropagation networks, training occurs in the backward pass. This process is described by illustration 4.⁵ Data is prepared for training by pairing input vectors with target output vectors. The input vector, $\mathbf{x}$, is applied to the network, which operates on the vector as in the forward pass and produces an output vector, $\mathbf{out}$. The output vector is then compared with the target vector from the pair, $\mathbf{t}$. The network then performs a series of evaluations assigning partial fractions of the error to different parts of the network and updating them to reduce this error. This assignment is referred to as propagating the error backwards and in the most simple case is a straightforward application of the chain rule for multi-variable functions. This evaluation can be performed in either a batch or an individual manner, either analyzing the error for each vector in the training set or determining the error for the entire training set at once before adjusting the elements of the network.

Forward pass.

It may be instructive to pause to consider notation before beginning the discussion. All superscripts in the following discussion refer to the number of sets of weights through which the input propagates before the operation takes place. For instance, net^1 refers to the net weighted input entering an arbitrary node in the first hidden layer, since the first hidden layer directly follows the first set of weights. Subscripts identify the variables within a given layer described by the superscript. For components of vectors, such as the input or output vector, the subscript identifies the index. Weights are described by a double subscript index. The weight w_{ij}^k connects output from the i^{th} node of the k^{th} layer to the j^{th} node of the $(k+1)^{th}$ layer.

In the forward pass an input vector $\bar{x}$ is applied to the nodes of the input layer. Each element, x_i , of the vector is then applied to each node of the first hidden layer. Nodes that are not either in the first input layer, receiving input directly from outside of the network, or the output layer, which performs the final operation on the input, are referred to as hidden layers. Each node in a hidden layer assigns a different *weight* to the input passed on to it from different nodes in the past layer. The j^{th} node of the 1st hidden layer multiplies the input element x_i by the weight w_{ij}^1 . The j^{th} node sums up the weighted contributions of all the output from the previous layer to get a value net associated with that node.

$$net_j^1 = x_1 w_{1j}^1 + x_2 w_{2j}^1 + \cdots + x_m w_{mj}^1 = \sum_i^m x_i w_{ij}^1$$

The node then applies this summed value for net to a transfer or squashing function, $f(net)$. The function can typically be any continuous monotonically increasing function with finite bounds. These

functions were initially designed to compress very large signals. Most commonly today the function used is either a sigmoid or inverse tangent function. Both have the property that small differences in input give decreasingly different output as input deviates from 0 and both converge to finite limits as the net approaches plus/minus infinity. In the code used for this program, the sigmoidal function is used.

$$y_j^1 = f(\text{net}_j^1) = \frac{1}{1 + e^{-\text{net}_j^1}} \quad (\text{sigmoidal; illustration 2})$$

The sigmoidal function is particularly popular because the unique property of its derivative saves computational time and memory.

$$\begin{aligned} f(x) &= \frac{1}{1 + e^{-\beta x}} \\ \frac{df}{dx} &= -\left(\frac{1}{(1 + e^{-\beta x})^2} \right) \frac{d}{dx} [1 + e^{-\beta x}] = -(-\beta e^{-\beta x}) \left(\frac{1}{(1 + e^{-\beta x})^2} \right) \\ &= \beta \left(\frac{1 + e^{-\beta x} - 1}{1 + e^{-\beta x}} \right) \left(\frac{1}{1 + e^{-\beta x}} \right) = \beta \left(1 - \frac{1}{1 + e^{-\beta x}} \right) \left(\frac{1}{1 + e^{-\beta x}} \right) \\ &= \beta (1 - f(x)) f(x) \end{aligned}$$

During training the derivative is used repeatedly in adjusting weights in the proper direction. This formula reduces computational time and reduces memory requirements.

In a single hidden layer network the j^{th} node of the hidden layer then propagates its output, y_j^1 , to each of the nodes of the output layer. Again the output from the j^{th} node of the 1st hidden layer is transferred to the k^{th} node of the output layer and weighted by the scalar, w_{jk}^2 . The weighted sum, net_k^2 , is computed in the output nodes and applied to the transfer function f to get the output value, out_k , for the k^{th} output node.

$$net_k^2 = y_1^1 w_{1k}^2 + y_2^1 w_{2k}^2 + \dots + y_n^1 w_{nk}^2 = \sum_i^n y_i^1 w_{ik}^2$$

$$out_k = y_k^2 = f(net_k^2) = \frac{1}{1 + e^{-net_k^2}}$$

The generalization to a larger network is straightforward. The layer of output nodes becomes a second hidden layer and the values defined above as *out* are then propagated through a third layer of weights as y_k^2 and processed through another layer of nodes to obtain *out*.

The computation can be succinctly stated using vector-matrix notation. A 1 x m row vector is applied to the network as input.

$$\text{Input vector: } \bar{\mathbf{x}} = (x_1, x_2, \dots, x_m) \in \mathbb{R}^m \quad 1 \times m \text{ row vector}$$

If the weights for the k^{th} layer of n hidden nodes are represented by a m x n matrix

$$W^k = \begin{bmatrix} w_{11}^k & w_{12}^k & \dots & w_{1n}^k \\ w_{21}^k & w_{22}^k & \dots & w_{2n}^k \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1}^k & w_{m2}^k & \dots & w_{mn}^k \end{bmatrix} \quad \text{Weight matrix for the } k^{\text{th}} \text{ layer (m x n)}$$

Then the 1 x n row vector representing the weighted input for the n nodes of the first hidden layer is computed:

$$\mathbf{net}^1 = \bar{\mathbf{x}} \mathbf{W}^1$$

The 1 x n output vector for the first layer of n hidden nodes is the vector valued function **f**, and **y** is the result of the function f applied upon $\mathbf{net}^1$.

$$\bar{\mathbf{y}}^1 = f(\mathbf{net}^1)$$

The $1 \times n$ vector $\mathbf{y}^1$ is applied to the second $n \times p$ matrix of weights to produce the **net** vector representing the weighted input for the p nodes of the second processing layer.

$$\mathbf{net}^2 = \bar{\mathbf{y}}^1 \mathbf{W}^2$$

The vector $\mathbf{net}^2$ is applied once more to the transfer function to obtain the $1 \times p$ output row vector.

$$\mathbf{out} = f(\mathbf{net}^2) = f(\bar{\mathbf{y}}^1 \mathbf{W}^2)$$

Training:

In supervised training the network is given a set of input vectors, $\bar{x}^i$ associated with a set of target output vectors $\bar{t}^i$. Each input vector containing m vector components can be described as a vector in m -space, while the neural network produces output vectors containing p elements, which can be described as vectors in p -space. The training set of input vectors with associated target output vectors can be simply represented by the set

$$T = \{ \bar{x}^i \in \mathbb{R}^m, \bar{t}^i \in \mathbb{R}^p \}$$

Each vector $\bar{x}^i$ is applied to the network which computes a resultant vector, **out**^{*i*}. The product **out**^{*i*} is then compared against the target vector $\bar{t}^i$ and some quantitative measure of error is calculated. This error function can be considered to be a function of the weights of the network,

$$\text{i.e. } \xi = \xi(\bar{w}) \text{ where}$$

$$\bar{w} = (w_{11}^1, w_{12}^1, \dots, w_{21}^1, \dots, w_{mn}^1, w_{11}^2, \dots, w_{np}^2)$$

is the vector whose elements are all the weights in the network. Training then proceeds to adjust the weights in such a direction as to minimize the error function.

In the most standard method, the error function is the sum squared difference between the output and target vectors for each training set.

$$\text{i.e. } \xi(\bar{w}_{ij}^k) = \sum_i^p (t_i - out_i)^2$$

The most standard method is to minimize this function is by using a gradient descent algorithm. From vector calculus the directional derivative of the function ξ in the direction $\bar{u}$ is defined by

$$D_{\bar{u}} \xi = \bar{\nabla} \xi \cdot \bar{u}$$

$$= \|\bar{u}\| \|\bar{\nabla} \xi\| \cos \theta$$

The minimum value for $D_{\bar{u}} \xi$ is therefore obtained when $\cos \theta = -1$, or $\bar{u} = -\bar{\nabla} \xi$. Therefore to reduce the error function at the greatest rate one needs to find the gradient of the error function, $\bar{\nabla} \xi$, and add its negative to the current weight vector $\bar{w}$. The components of the gradient are the partial derivatives of the error function with respect to the weights.

$$(\bar{\nabla} \xi) \cdot \hat{w}_{ji}^k = \frac{\partial \xi}{\partial w_{ji}^k}$$

If k is the last layer of weights (e.g. $k=2$ in the above discussion) then the derivative is simply

$$\frac{\partial \xi}{\partial w_{ji}^k} = \frac{\partial \xi}{\partial (\text{out}_i)} \frac{\partial (\text{out}_i)}{\partial (\text{net}_i^k)} \frac{\partial (\text{net}_i^k)}{\partial w_{ji}^k}$$

$$\xi(\bar{w}_{ij}^k) = \sum_i^p (t_i - \text{out}_i)^2 \quad \frac{\partial \xi}{\partial (\text{out}_i)} = -2(t_i - \text{out}_i) = \delta_{\text{out}_i}$$

$$\text{out}_i = f(\text{net}_i^k) = \frac{1}{1 + e^{-\text{net}_i^k}} \quad \frac{\partial \text{out}_i}{\partial (\text{net}_i^k)} = f'(\text{net}_i^k) = f(\text{net}_i^k)(1 - f(\text{net}_i^k))$$

$$\text{net}_i^k = \sum_j y_j w_{ji}^k \quad \frac{\partial (\text{net}_i^k)}{\partial w_{ji}^k} = y_j$$

So

$$\frac{\partial \xi}{\partial w_{ji}^k} = -2(t_i - \text{out}_i) y_j f'(\text{net}_i^k)$$

then

$$\Delta_{ji}^k = \nabla \xi \cdot \hat{w}_{ji}^k = -2(t_i - \text{out}_i) y_j f'(\text{net}_i^k)$$

$$(w_{ji}^k)_{\text{updated}} = (w_{ji}^k)_{\text{old}} - \eta \Delta_{ji}^k$$

where η is a parameter determining step size that controls the rate of convergence.

If k is not the output layer the situation is slightly more complicated but can still be solved through application of the chain rule. Essentially the same derivative is used, however, weights in layers before the last impact the error by affecting the result of all the output nodes since the i^{th} node in the k^{th} layer transmits its output to all of the nodes in the next layer.

$$\frac{\partial \xi}{\partial w_{ji}^k} = \sum_n \frac{\partial \xi}{\partial (\text{out}_n)} \frac{\partial (\text{out}_n)}{\partial (\text{net}_n^{k+1})} \frac{\partial (\text{net}_n^{k+1})}{\partial y_i} \frac{\partial y_i}{\partial (\text{net}_i^k)} \frac{\partial (\text{net}_i^k)}{\partial w_{ji}^k}$$

Again

$$\begin{aligned} \xi(\bar{w}_{ij}^k) &= \sum_i^p (t_i - \text{out}_i)^2 & \frac{\partial \xi}{\partial (\text{out}_n)} &= -2(t_n - \text{out}_n) = \delta_{\text{out}_n} \\ \text{out}_i &= f(\text{net}_i^k) = \frac{1}{1 + e^{-\text{net}_i^k}} & \frac{\partial \text{out}_i}{\partial (\text{net}_i^k)} &= f'(\text{net}_i^k) = f(\text{net}_i^k)(1 - f(\text{net}_i^k)) \end{aligned}$$

But now,

$$\begin{aligned} \text{net}_i^k &= \sum_j y_j w_{ji}^k & \frac{\partial (\text{net}_i^k)}{\partial y_j} &= w_{ji}^k \\ y_i^k &= f(\text{net}_i^k) = \frac{1}{1 + e^{-\text{net}_i^k}} & \frac{\partial y_i}{\partial (\text{net}_i)} &= f(\text{net}_i)(1 - f(\text{net}_i)) = f'(\text{net}_i) \\ \text{net}_i &= \sum_j x_j w_{ji}^1 & \frac{\partial (\text{net}_i)}{\partial w_{ji}^k} &= x_j \end{aligned}$$

Therefore,

$$\begin{aligned} \Delta_{ji}^k &= \bar{V} \xi \cdot \hat{w}_{ji}^k \\ &= \frac{\partial \xi}{\partial w_{ji}^k} = \sum_n \delta_{\text{out}_n} w_{ji}^k x_j f'(\text{net}_i) = \delta_{ji}^k x_j \end{aligned}$$

Where

$$\delta_{ji}^k = \frac{\partial \xi}{\partial (\text{net}_{ji}^k)} = \sum_n \frac{\partial \xi}{\partial (\text{out}_n)} \frac{\partial (\text{out}_n)}{\partial (\text{net}_n^{k+1})} \frac{\partial (\text{net}_n^{k+1})}{\partial y_i} \frac{\partial y_i}{\partial (\text{net}_i^k)}$$

So

$$(w_{ji}^k)_{\text{updated}} = (w_{ji}^k)_{\text{old}} - \eta \Delta_{ji}^k$$

For networks with more than 2 sets of weights the above algorithm can be generalized by simply replacing the input component x_i with y from the $k - 1$ layer.

Two major problems exist with the gradient descent algorithm. One problem is that the algorithm converges very slowly. A second is that because the algorithm only considers first derivatives of the error surface, training can often result in the network becoming trapped in local minima. The network will move in the direction that will reduce the error by the greatest amount. However, if the network reaches a local minimum, the directional derivative will be positive in all directions and the network will cease to adjust its weights even if a lower minimum exists beyond a local maximum. This state is referred to as network paralysis.

A simple and computationally cheap way of reducing the risk of paralysis is to incorporate a momentum term into the training algorithm. This term keeps track of the last adjustment to each weight and gives the system "momentum" to continue in this direction.. A common algorithm to implement this idea was proposed by Rumelhart, Hinton, and Williams¹

$$w_{ij}(n+1) = w_{ij}(n) + \left[\Delta w_{ij}(n) + \alpha (\Delta w_{ij}(n-1)) \right]$$

where

α :	momentum parameter
$w_{ij}(n)$:	is the weight before the n th evaluation
$\Delta w_{ij}(n) = -\eta \Delta_{ij}^k$	is the adjustment to the weight in the n th time step

A second problem is that convergence is very slow since by only considering first derivative information in the error surface, useful properties of the second derivative are not considered. The most common methods for this are incarnations of the conjugate gradient descent algorithm. Rather than directly following the gradient and stepping in the direction of maximum decrease, which results in many unnecessary steps, conjugate gradient descent methods move in a direction that is perpendicular to the direction of the previous step while still reducing the error. These methods require the second derivatives of the error function and the calculation of the Hessian matrix. In order to reduce the time necessary for these calculations a number of approximations are made. The error surface is considered in a Taylor expansion, often as a quadratic.

General References:

1. Smith, Murray. *Neural Networks for Statistical Modeling*. Van Nostrand Reinold: New York, New York. 1993.
5. Wasserman, Philip D. *Advanced Methods in Neural Computing*. Van Nostrand Reinhold: New York, New York. 1993. Chapter 6.
6. Fausett, Laurene. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Prentice Hall: Englewood Cliffs, New Jersey. 1994. Chapter 6.

Radial basis functions:

Radial basis function neural networks describe a broad category of neural networks including the generalized regression neural networks (GRNN) used in this study. Radial basis function networks are most easily described as non-linear mapping functions, transforming input vectors in m -space onto output vectors in n -space. In the context of neural network mapping, network input containing m values can be considered to be an m -dimensional vector. The network then maps the input onto an p -dimensional output vector.

These networks attempt to model the function space with a series of radially dependent symmetric functions, which provide essentially a basis for the space. Input vectors represent points in an n -dimensional training space. There exists some unknown (unknown to the user that is - hopefully not to the network) function that assigns a p -dimensional vector as a value to each point in the m -space. In the current context, the m -dimensional vector contains the 23 m/z peaks in the mass spectra of the chemical streams. The p -dimensional vector then describes the concentration of the six compounds in the mixture. Basis functions are used to describe this unknown function by extrapolating from the known values (concentrations) corresponding to specific points (spectra) to the whole space. Gaussian distributions are the most commonly used basis functions.

Networks within the radial basis function family offer a number of advantages over back-propagation networks. They train very rapidly - usually in orders of magnitude less time than back-propagation networks. However, once trained their predictive capabilities are less rapid. In the experiments performed in this study, the difference in time of prediction was not significant, while the difference in training time was immense. These networks also avoid other pathologies associated with training back-propagation networks such as becoming trapped in local minima of the error space.

Illustration 5 shows the structure of a simple radial basis function neural network. The structure of radial basis function networks bears superficial similarity to feed-forward back-propagation networks, although the two paradigms are fundamentally different. Nodes are responsible for the calculations performed in the mapping from input to output vectors. The calculations of each node are entirely independent of other nodes within the same layer. Nodes are often organized into two layers with the second layer feeding output nodes. The output of each node is always transmitted only to the nodes in the next layer. As in feed forward back-propagation networks, nodes are connected by weighting factors. In the most general case, these weights are adjusted according to fit the given data through a process of training in order to minimize the errors of prediction.

Furthermore, each node is fully connected with all others in the previous and successive layer. In particular, each node within the hidden layer performs a computation involving all of the elements of each input vector, and nodes in the output layer sum the weighted contributions of each node in the hidden layer. In general different nodes can perform different calculations.

In calculation, an m -dimensional input vector, $\bar{x} \in \Re^m$ is applied to the network. Each node within the hidden layer is associated with a particular point, $\bar{u}_i \in \Re^m$. The i th node applies the input vector to a function whose value depends solely on the distance from the point, i.e. $h_i = h(\|\bar{x} - \bar{u}_i\|)$.

Most commonly this function is defined as a gaussian distribution described with a standard deviation, σ_i , characteristic of that particular node. Then

$$h_i = \exp\left(-\frac{\|\bar{x} - \bar{u}_i\|^2}{2\sigma_i^2}\right)$$

The output, h_i , from each node in the hidden layer is propagated to each of n nodes in the output layer and weighted by a matrix, $\mathbf{W}$, similarly to a feed-forward network. The output of each node in the output layer then is

$$y_j = \sum_i h_i w_{ij}$$

The output from these nodes is then scaled by the sum of all of the nodal output. The result is an n -dimensional output vector, **out**, whose elements sum up to 1. Then

$$\text{out}_j = y_j / s$$

and

$$s = \sum_i^n h_i.$$

Training a radial basis function network involves adjusting 3 sets of parameters to minimize error. For each hidden node, the center of the distribution, $\bar{u}_i$, and for networks using gaussian distributions, the standard deviation, σ_i , must be determined. Additionally, the weight matrix, $\mathbf{W}$, connecting the hidden and output nodes is trained in some models, although not in GRNN.

For some networks, including the GRNN, the center for basis functions is taken to coincide with the input vectors in the training set. However, typically within a training set, a number of input vectors exist within the same region of the training space. In these cases it is highly inefficient to assign nodes with radial basis functions centered about each input vector. Instead a single node may be created so that its function is centered about a group of input vectors through clustering.

A number of different cluster methods exist for identifying centers for the functions and for grouping input vectors into these clusters. One common method is to assign the first training vector as the center of the first cluster. Each successive input vector in the training set is then considered.

All vectors within a pre-defined radius, r , of a previously identified cluster are assigned to that cluster. Any vectors that are outside of that region become centers for a new cluster.

A commonly used iterative alternative method relies upon defining initial positions for centering clusters. Each input vector is considered in turn, and its distance from current cluster centers is calculated. The nearest cluster is identified and this cluster is moved a step towards that input vector. This process is iterated until the changes in the locations of clusters converge to final locations.

In the most general case, the standard deviation, σ_i , for the radial function of each node can be different. Therefore, this parameter must be determined in training. This parameter describes the extent to which each radial basis function extends beyond the center of the distribution. A common method for defining σ_i is by finding the root mean square distance between each cluster center and the N closest centers.

In many radial basis function networks, the weights between the hidden layer and the output layer are also trained. Training these weights is exactly analogous to the training involved in feed-forward back-propagation networks and proceeds generally by a gradient descent method. However, the problem is much simpler and more rapid because only a single layer of weights need to be trained. The rule for adjusting the weight, w_{ij} , in the n th iteration is

$$w_{ij}(n+1) = w_{ij}(n) + \eta(t_j - y_j)x_i$$

where t_j : the j th element of the target vector

y_j : the output of the j th output node

x_i : i th element of the input vector.

η : training rate coefficient ($\eta \ll 1.0$)

Generalized Regression Neural Network:

A Generalized Regression Neural Network (GRNN) is a simple example of a radial basis function neural network. GRNN assigns a node in the hidden layer for each training vector. Because of this the network does not need to cluster the data. The weights from the hidden layer to the output layer are defined so that applying an input vector from the training set will result in the network calculating the correct target output nearly exactly. The weights connecting a single hidden node to all of the output nodes can be considered an n-dimensional vector, which is defined to be the target output for the input vector. When the i th input vector from the training set is applied, the output from the i th hidden node will be 1. However, if appropriate standard deviations are defined for all of the nodes, then the contribution from the other hidden nodes will be negligible. The predicted output will then be essentially the appropriate target vector. Therefore, $h_i \sim 0$, for all of the nodes except a single node, i , and the j th element of output will be

$$out_j = \left(\sum_l h_l w_{lj} \right) / \left(\sum_k h_k \right) \approx (h_i w_{ij}) / h_i = w_{ij}$$

Then the weights will be appropriately defined when the weight connecting the i th hidden node to the j th output node is equal to the j th element of the i th target vector. i.e.

$$w_{ij} = (\vec{r}^i)_j$$

$$\mathbf{W} = [\vec{r}^1 \quad \dots \quad \vec{r}^P]$$

The justification for this is derived from GRNN's origins as a nonlinear regression method.

A theorem in statistics states that the expected value for a predictor, $E[y | \mathbf{x}]$ given an input vector, $\mathbf{x}$, can be described as⁵

$$E[y|\bar{x}] = \frac{\int y \psi(\bar{x}, y)}{\int \psi(\bar{x}, y)}$$

where $\psi(\bar{x}, y)$ represents the probability density of $\mathbf{x}$ and y together and y is the predictor output.

From this theorem it can be shown that the optimal function value, y_j , for a GRNN output node can be given by⁵

$$y_j = \frac{\sum_{i=1}^n h_i w_{ij}}{\sum_{i=1}^n h_i},$$

i.e. the output function for the GRNN, where the variables are defined as previously described.

General Reference:

5. Wasserman, Philip D. *Advanced Methods in Neural Computing*. Van Nostrand Reinhold: New York, New York. 1993. Chapter 8.

Bprop;

Experiments performed simulating feed-forward back-propagation artificial neural networks were performed using the program, bprop, generously provided by Dr. Bobby Sumpter of Oak Ridge National Laboratory. The program allowed manipulation of a large number of variables within the network. Additionally the program lends itself very readily to the use of input files to run the code. Because of the format of the code input and output, a single output file could be prepared and with minor adjustments applied to most experiments.

The parameter which was studied in the greatest detail was the topology of the network. The code allowed only feed-forward networks, so that no interactions between nodes were bi-directional and nodes within the same layer had no interaction. However, the program allowed the user control over the number of layers as well as the number of nodes within each layer. Increasing the number of layers or the nodes within a layer allows the network to model systems with increasing complexity. However, networks of increasing complexity suffer from increasing tendency to over-fit or memorize data patterns, reducing robustness.

The logistic function, the most commonly used transfer function because of the useful properties of its derivative, was used in all of the experiments. The program used a sum squared error function in training. Weights were initialized prior to training through the use of a random number generator.

The program allows the user to specify whether weights are updated using a Langevin updating, scaled conjugate gradient, or Rprop training algorithm. The scaled conjugate gradient method is a modification of the gradient descent algorithm outlined in this paper. It takes advantage of second derivative information to both speed convergence and to avoid network paralysis.

The program also gives the option to re-sample the data. In the absence of re-sampling, the network propagates the training spectra through the network in the same sequence that it is found in the input file. For each spectrum, the network predicts an output vector which is compared the proper target vector from the training set. It then updates its weights using the training algorithm to reduce the error of prediction before moving on to the next line containing a spectrum. As a result, the network trains on all of the spectra in the input file without testing its predictions on any of them. However, the program also allows the option of re-sampling the data a number of times using a bootstrap algorithm. In this algorithm the program trains on different samples of the training set data several times, each time leaving out some training vectors to test its predictions. For each run, the network randomly produces n numbers between 1 and n , where n is the number of spectra in the training set. The n spectra corresponding to numbers randomly generated by the program are then put in the training set for that re-sampling. However, in picking random numbers n times, some numbers will be picked more than once, so not all of the n spectra are included in the training set. The remaining spectra are put into the testing set for that run and used to measure the error of the adjusted network's predictions. As the number of re-sampling runs increases, the predicted error of the network's output approaches a true estimate of the network's prediction capability, since an increasing fraction of the entire data set is used for testing.

The executable file, `bprop.exe`, required two independent files for training and testing the network. The input file contains the amplitudes of the spectra for input into the first layer of nodes. Spectra used for training the network must be at the beginning of the input file. Spectra for which the composition of the solution is unknown and must be predicted follow the training spectra at the end of the input file. The processing of the program requires that the data be formatted so that each

spectrum comprises a separate line containing the amplitudes of each m/z peak. Amplitudes for separate peaks are delineated by blank spaces. No mark delimits the end of the spectrum other than the line break. No header or comments should intersperse or flank the data. In experiments, each spectrum was preceded by 4 blank spaces. Each amplitude was formatted as a number with 7 decimal digits. Four blank spaces delimited each number. With these characteristics, each spectral line included 300 columns.

The output files contained the actual concentrations of the compounds in the solutions. The concentrations of the solution giving the i th spectrum in the input file were recorded on the i th line of the output file. For each spectral line in the input file used for training, the output file must contain a line of concentrations. Output files were formatted similarly to the input files. The data was similarly delimited. In each experiment the network was fed the complete average spectra for all 32 solutions as an input file. In some experiments the network predicted the concentrations of the entire solution. For the purposes of comparison, though, in others the network only predicted the concentrations of 5 of the compounds, and in many others it predicted the concentration of a single component.

In execution, the program reads both the input and output files containing the spectra and corresponding concentrations. The user specifies the transfer and error functions, as well as the training algorithm and the method by which the weights are initially assigned. The user also specifies the number of times the training set should be re-sampled and the number of epochs in each re-sampling. In each epoch, the neural network adjusts its weights to reduce the error of prediction. In each re-sampling the program partitions the training set into subsets of spectra used to adjust the weights and to monitor the progress of the network. Then in each run, the network runs through a

specified number of epochs of training. In each epoch, each of the training spectra partitioned for training is applied to the network, which predicts the concentrations of the solution. The predicted concentrations are compared with the concentrations known in the output file. The error is calculated according to the specified error function and the weights are updated according to the specified training algorithm. The epoch ends when the weights have been updated according to each of the spectra selected for training and the network then applies the spectra in the testing set to the current weights in the network. The errors in predicting the testing spectra are output as the error of the network at that stage. In each experiment performed each run lasted for 10 000 epochs, while the number of re-sampling runs ranged from 3 to 100.

Experimental:

The experimental data consisted of mass spectra from 32 solutions. The solutions contained varying concentrations of methane, ethane, propane, propene, isobutene, and isobutane, identified as compounds one through six respectively. The spectrum for each solution were recorded in triplicate, so 96 spectra were available for training. Each spectrum consisted of the amplitudes at 23 relevant m/z peaks selected from the complete spectrum.

Upon receiving the experimental data a number of analyses characterized the data. A program `dmatrix.f90` organized the data into a 96 x 29 matrix. The rank of the spectral matrix was calculated using MATLAB software. A histogram was prepared to analyze the distribution of the fractional concentrations for each compound. A program analyzed the precision among triplicates of the spectra. A program, `avgmatrix.f90`, averaged the 96 triplicate spectra to produce 32 average spectra, one representing each solution. A program, `divdata.f90`, divided the output files containing the actual fractional concentrations into separate files for each component. Additionally, `nlin.f90` analyzed the difference between the experimental spectra and purely linear combinations of the spectra for separate compounds. From the trends recognized by this program, `syndata.f90` generated 1000 synthetic training spectra of concentrations randomly prepared by the program.

Dr. Bobby Sumpter generously provided software, including executable and code files, in support of this project. These files allowed modeling with several different artificial neural network paradigms. One of these codes, `basnet.exe` simulated radial basis function neural networks. A number of experiments were performed with the code's generalized regression neural network (GRNN). GRNN was used to predict the spectra of all concentrations individually in independent runs. The program also predicted the concentrations of all compounds simultaneously in other runs.

Each trial was performed in triplicate. Experiments were performed with differing degrees of re-sampling to observe the convergence of the prediction to the true solution. Spectral data was formatted for this executable by the file dr3.f.

The majority of work with back-propagation feed-forward networks was performed with the executable file bprop.exe, also provided by Dr. Sumpter. The bprop code lent itself particularly well to direction with outside files containing pre-determined user input. This allowed many experiments to be performed highly efficiently because the program was not continuously interrupted waiting for user input. The ability of two different training algorithms, the Rprop and conjugate gradient descent algorithms, to learn the relationship between spectra and solution concentration was studied. The affect of increasing re-sampling of the spectral data was studied by running experiments which re-sampled the data 3, 5, 10, or 100 times. Independent experiments tested the ability of the network to predict the composition of the entire solution, as well as the concentration of each component individually.

Each experiment using the bprop.exe file studied the effect of certain parameters on the ability of the network to predict the composition of the solution. Preliminary experiments indicated that using all replicate spectra resulted in network paralysis and divergence problems in training. In each experimental run using observed spectra, the network received as input the thirty - two average spectra - one average spectra representing each set of triplicates. However, the network output differed between sets of experiments. In some experiments the network predicted the concentrations of all six compounds at the same time. In others, given the same spectral data, the network was trained to only predict the fractional concentration of a single compound. In a few experiments, the network predicted the concentrations of all of the compounds except for isobutane. For each set of

experiments, the network used a specified training algorithm and number of re-samplings. The network was then run using varying numbers of nodes in the hidden layer.

The convergence of prediction was tested using the Rprop algorithm and varying the number of re-samplings performed. Experiments were performed in which the data was re-sampled 3, 5, 10, and 100 times. Rprop training was compared relative to conjugate gradient descent training with 100 re-sampling.

Neural networks were used to try to recognize patterns in the deviation of the spectrum from linear combinations of pure spectra. The code `linear.f90` produced linear combinations of the spectra for the six pure compounds using the concentrations of the compounds in solution as coefficients. This linear spectral matrix was then subtracted from the experimental spectral matrix. A neural network was then trained on this difference matrix to predict the solution concentrations. The network in this study used the conjugate gradient descent algorithm and re-sampled the data 100 times.

The code `nlin.f90` performed statistical analyses of the differences between the spectral matrix and the linear matrix. The code computed the mean and standard deviation of the differences. These quantities were computed both for each spectra and for each m/z peak.

Using this information synthetic spectra were generated for training by the networks. The code `synthdata.f90` created a random 6×1000 matrix to represent possible solution concentrations. The matrix was scaled so that the sum of all six elements in each column added to 1. The matrix was then multiplied by a 23×6 matrix of the spectra where the i th column contained the spectrum for the i th pure component. The product was a 23×1000 matrix of Beer's law combinations of the pure spectra. To each element of the matrix a random non-linear term was added based upon the statistical

analysis performed by nlin.f90. The term was generated with a gaussian distribution so that all added terms were within 5 standard deviations of the mean difference from linearity for the particular m/z peak. This matrix was then used as 1000 training spectra for the network. A conjugate gradient descent algorithm was used for training; the network trained on these synthetic spectra was then used to predict the concentrations for the experimental spectra.

Results:

The data available for training neural networks consisted of the mass spectra for 32 gas mixtures, each recorded in triplicate. Figure 1a, a histogram of the data prepared using MATLAB, shows the distribution of solutions represented by experimental spectra. The training data all fall within a narrow range of concentrations with very little variance. With the exception of isobutane and isobutene, the concentrations of each compound in solution were clustered into bins between .04 and .20. Isobutane has the greatest distribution and range of concentrations within the data set. The fraction of isobutane in solution ranges from $\sim .15$ to $\sim .72$ in intervals of roughly .07. Conversely, the fraction of isobutene is represented by a single peak in fig. 1a. The fractional concentration of isobutene in solution ranges from .100 to .120, as shown by fig. 1b.

One spectrum from each set of triplicates was put into a matrix. MATLAB was then used to determine the rank of the 23×32 matrix. The rank of the experimental spectral matrix was 23. This indicated that the matrix contained no singularity; none of the spectra could be written as linear combinations of other spectra in the training set.

A set of fortran codes, entitled experr, analyzed the deviation between triplicate measurements. This code calculated the mean amplitude and standard deviation for each m/z peak between triplicate measured spectra of each solution and used this information to form a "standard deviation" vector. The "standard deviation" vector contained the standard deviation among triplicate measurements for each m/z peak for the associated average spectra. The standard deviation between triplicates at each peak for each solution is shown in figures 2a, b. Figure 2a graphs the magnitude of the standard deviation of each peak height among each set of triplicate spectra. The vast majority of the points lie on the floor of the surface, with the exception of a handful of deviations of roughly

.15. Figure 2b allows an alternate view of the data. Figure 2b graphs the standard deviation relative to the average peak height for each m/z peak in each set of triplicates. While a number of the points in fig. 2a come up off of the bottom of the graph in fig. 2b, the outliers in figure 2a are brought down. The result is that all of the triplicate measurements of each m/z peak have less than a five percent relative standard deviation. The majority of points in fig. 2a have near zero deviation among triplicate measurements because the vast majority of peaks in each spectrum have near zero amplitude. Many of the near zero deviations represent the differences in measurements among nearly zero peak heights. Conversely, the peaks with larger absolute standard deviation correspond to peaks of greater amplitude and, consequently are not especially large deviations relative to the average height among the triplicates. By far the largest standard deviations in measurement correspond to m/z peaks at 42 - 44.

Figure 3 graphs the magnitude of the deviation vector computed by the experr code. The program calculates both the norm as well as the average value among the components for each "standard deviation" vector. Figure 3a calculates the norm in terms of the absolute standard deviations. Figure 3b calculates the average value of each vector when the absolute standard deviations are scaled by the average amplitude of the m/z peak among the triplicates. Both diagrams illustrate the generally high precision among triplicates. Figure 3b shows that for 2/3rds of the solutions, the average relative standard deviation of the spectra was less than 1%.

Dr. Bobby Sumpter's basnet code simulates a Generalized Regression Neural Network (GRNN), a simple radial basis function. The GRNN code predicted the fractional composition of each compound. Each experiment was performed in triplicate. Both the mean and standard deviations for the average and maximum error of prediction are shown in the tables. The upper half of each

table describes the accuracy of the GRNN in predicting the concentrations of all compounds simultaneously, while the bottom half contains the results for predicting individual concentrations independently.

Table 1a contains data summarizing the accuracy of these predictions when the spectral data was re-sampled 10 times. The GRNN performed better predicting the concentrations of all compounds simultaneously rather than individually. The average errors for simultaneous prediction ranged from 15 % to 28 %. The GRNN predicted only compound 5, isobutene, with greater than 21 % average error. The average errors for individual predictions ranged from 13.7 % to 74.5%. Only compound 2, ethane, was predicted better on average in individual computations than in computations simultaneously predicting all concentrations. (13.7% vs. 15.0%) The difference in maximum errors of prediction between simultaneous and individual predictions was much greater than the difference in average errors of prediction for the two methods. The maximum errors for simultaneous prediction ranged from 37 to 53%. The maximum errors for individual predictions ranged from 66 % to 109%.

The GRNN predicted the concentration of compound 2, ethane, with greatest accuracy. In simultaneous prediction the maximum error of predicting the concentration of compound 1, methane, was less than the maximum error for ethane (37% vs. 42%). However, in every other prediction, ethane was the most accurately predicted. In particular the maximum error in the individual computation of ethane was nearly 20% better than the maximum error for independent prediction of any other concentration.

Compound 5, isobutene, was predicted with the least accuracy in every experiment. The difference in error for predicting the concentration of isobutene relative to the other compounds was

particularly great for experiments calculating concentrations for individual compounds independently. The average error in individual predictions of isobutene concentration was more than double that of any other compound. The maximum error of 110% was 17% greater than the maximum error for predicting the concentration of any other compound by any method.

The experiment was repeated using the same procedure with the exception that the spectra were re-sampled 100 times. The data for these experiments is summarized in table 1b. The results are qualitatively similar to those for 10 times re-sampling. With the exception of the maximum errors for simultaneously predicting compounds 1,2,and 6 (methane, ethane, and isobutane), the results agree within the observed standard deviations. The mean and standard deviation of the average error increased for all of the compounds in simultaneous prediction. The average error for individual predictions remained unchanged within experimental spread. The result is that the average error for simultaneous and individual predictions of compounds converged to within experimental spread for all of the compounds except for compound 5, isobutene. Isobutene was the sole compound for which the average error in individual prediction was experimentally different from the average error in simultaneous prediction of all six concentrations. The standard deviations of the average error for individual prediction were significantly reduced on average, as would be expected from the increased sampling of data.

Surprisingly, the maximum errors in the simultaneous predictions were reduced from the 10 re-sampling experiment, although for 4 compounds this difference was accounted for by large variation within prediction errors. The maximum error for individual predictions remained nearly the same as for the previous GRNN experiment. The only change in maximum error by more than 5% was for compound 2 (ethane), for which the maximum error increased from 66 % to 72 %. This

increase is within the experimental spread for the ten and 100 re-sampling tests. In fact for all of the compounds, the standard deviation for the maximum error in individual predictions decreased significantly from the GRNN experiment with 10 re-sampling.

The training process repeats for each time that the data is re-sampled. When the data is re-sampled, the network randomly selects a number of vectors from the training set to test against and then trains with the remaining vectors. The network trains starting with randomly initialized weights for a specified maximum number of epochs or until the maximum error of prediction is less than the pre-defined threshold error. During each epoch the network predicts the output vector for each associated input vector remaining in the training set. Each output vector is compared with the associated target vector from the training set and the weights are updated. At the end of each epoch, the network predicts the concentrations for each spectrum selected for the testing set for that re-sampling. Therefore, as the number of re-samplings increases in the training process, the network will have been tested on increasingly greater percentages of the entire data set. As a result, the error of prediction should go up as the network is asked to predict the output from more difficult regions of the data set.

Furthermore, as the network attempts to predict more difficult spectra, one expects that more complex topologies would become increasingly favorable. Smaller topologies would be expected to be more favorable for predicting smaller subsets of spectra, but should have more difficulty generalizing to larger sets. Conversely, larger networks would be expected to have less predictive capabilities for smaller data sets due to problems of over-parameterization, but as the network attempts to predict greater portions of the data set, more parameters allow greater flexibility.

An experiment analogous to the GRNN experiment was performed to observe the effect

increasing the data re-sampling had on training the network with the Rprop algorithm. The network was first trained when the data was re-sampled only three times. The network configurations resulting in the best predictions for each compound were identified. Networks with these topologies were then trained on the experimental spectra using the Rprop training algorithm and with increasing re-sampling of data.

Table 2 contains the results of these experiments studying re-sampling with the Rprop training algorithm. Graphical representations of this data, plotting the errors in predicting each compound as a function of re-sampling for each topology are included in appendix D using a logarithmic scaling for re-sampling.

To a certain extent these expectations were confirmed by experimental results. The minimum among average errors of different networks for calculating concentrations increased with re-sampling for each compound except compound 3 (propane). The average error in predicting propane decreased from 3.69% at 3 re-samplings to 2.85% when the data was re-sampled 100 times. The concentrations of compounds 1, 2, 4, and 6 (methane, ethane, propene, and isobutene) were generally predicted with the smallest average errors. For these compounds the expectation of increasing network complexity (in terms of hidden nodes) improving predictions with increasing re-sampling was supported by data. For methane, ethane, and propene the most complex network (i.e. the network with the most hidden nodes) predicted the concentrations with the least accuracy at the smallest degree of re-sampling, but proved to be the most accurate predictor at large re-sampling. Conversely for these compounds the least complex network predicted the smallest sample size best, but least well at the largest degree of re-sampling.

Compound 5 also demonstrated the expected trend. The largest network performed least

well of all the topologies in the experiments with small re-sampling. As the re-sampling increased this topology was the only one which improved at predicting the output. Compound 6 (isobutane) proved to be the exception: the largest network had the best predictive capabilities for the smallest sample, but was indistinguishable from the others in the 100 re-sampling experiment. Compound 3 (propane) demonstrated results most contrary to expectation. For propane, the network with the fewest hidden nodes and the network with the most hidden nodes were the least accurate at small re-sampling and also indistinguishable at larger degrees of re-sampling.

From considering the table and graphs in the appendix, the most obvious and general trend is convergence of average error in predictions between topologies with increasing re-sampling. For smaller amounts of re-sampling, significant differences in predictive capability exist between topologies. However, as the number of re-sampling increased, the difference in accuracy between topologies actually decreased, contrary to expectations. Another trend was that the minimum average error of prediction increased with re-sampling for each network except for compound 3, propane. This result is easily explained by the increasing number of spectra that are predicted.

Analysis of the effect of re-sampling upon the maximum error results in similar conclusions. The maximum error of prediction increases with re-sampling for all of the compounds except for compound 3 (propane). The largest network complexity has the smallest maximum error of prediction for compounds 1, 2, and 4 (methane, ethane, and propene) at 100 re-sampling. For compounds 1 and 2 the error of the smallest network increases with re-sampling while the error for the largest network either decreases or increases less rapidly than for other topologies. However, for compound 3 the results of the smallest and largest networks are indistinguishable for both the most and the least re-sampling. One major difference between trends for the average and maximum error

is that, while the average error of different topologies converge with re-sampling, the maximum error among topologies remains significantly different with changes in re-sampling.

Table 3 shows the results for training feed forward back-propagation neural networks using the Rprop algorithm while re-sampling the data 100 times. The first two sets of columns describe predictions for networks with a single hidden layer. This data describes the ability of the network to predict the concentrations of each compound individually and as an element of the entire group for different topologies. For each group of columns, the first column defines the number of nodes in each hidden layer, while the second and third columns describe the average and maximum error for that topology. Networks with two hidden layers are described by two numbers, describing the number of nodes in the first and second hidden layer, respectively. The first group of columns describes the results for computing concentrations individually. The second group of columns describes the results for predicting all of the concentrations simultaneously. The third group describes predictions of networks using two hidden layers.

The network predicted the concentrations of individual compounds independently much more accurately than it predicted the concentration of all compounds simultaneously. In particular, the network predicting all of the compounds together performed very poorly when using 5 nodes in the hidden layer. For this network the smallest errors of predicting any component were 15%/46% for compound 6 (isobutane). (The split denotes average and maximum error [avg.err. % /max. err.%]) All of the other predictions were off by at least 25% / 60%. However, even discarding the results of this experiment using 5 nodes in a single hidden layer to predict all six output, neural networks clearly performs better predicting single concentrations. The network predicts all of the output simultaneously most accurately using only three hidden nodes. The maximum error of prediction for

the network with 3 hidden and 6 output nodes is slightly less than the maximum error for the corresponding single output network for compounds 2 and 3 (ethane and propane): for compound 6 (isobutane), the improvement with 6 output nodes over 1 output is substantial. The average error for this network in predicting isobutane is slightly lower than the average error for the corresponding network with a single output. Networks predicting six output are less accurate than single predictors in every other case. The network does particularly poorly in predicting the concentration of compound 5 (isobutene). No other network configuration predicting all six concentrations performs with accuracy comparable to the single predicting network with the same number of hidden nodes..

The data shows that there exist topologies for single prediction that accurately calculate the concentration for every compound except compound 5. Compounds 1, 2, and 4 (methane, ethane, and propene) were all predicted with average errors of less than 3 % and maximum errors of 8.3%. The concentration of compound 6 (isobutane) was predicted with 3% average error and 12% maximum error. The concentration of compounds 3 and 5 (propane and isobutene) proved most difficult to predict. The network configuration predicting the concentration of compound 3 individually resulted in an average error under 3% but a maximum error of 16%. The concentration of compound 5 proved even more difficult to predict accurately. The lowest average error for compound 5 was more than 7.5% and the maximum error for this configuration was over 21%.

Because of the difficulty in predicting all of the compounds at once using networks with a single hidden layer, a number of topologies with two hidden layers were tested. The results of several networks with two hidden layers are shown in the third group of columns of table 3. The network using 5 nodes each in two hidden layers predicted the concentrations of all compounds with excellent accuracy with the exception of compound 5, which it predicted with significantly less

accuracy than other network topologies. The network using 3 nodes each in two hidden layers also performed well. It did not predict the concentrations of compounds 2 and 4 (ethane and propene) with as great accuracy, but gave the best performance of any network for compound 5 (isobutene) and also predicted the concentrations of compounds 3 and 6 with high accuracy.

Because of the universal problems in predicting the concentration of compound 5, an experiment was performed to see the effect of removing compound 5 from the network output. The network was trained on the same data under the same conditions, but did not predict the concentration of compound 5. The network used two hidden layers of five nodes each. Surprisingly this network provided results far superior to any other network. The average errors for predicting compounds 1 and 2 were less than 1%. The average errors for compounds 3 and 6 were both less than 1.5% and compound 4 was predicted within 3% on average. The maximum errors for this experiment were also very low. The maximum errors for predicting compounds 1-3 and 6 were all less than 4%, and the greatest error in predicting compound 4 was under 9%. (data not shown)

In order to provide a comparison and in an effort to determine the best predictive capabilities of neural networks for de-convoluting mass spectra, networks were also trained using a scaled conjugate gradient descent algorithm. All of the experiments re-sampled the data 100 times and used the same transfer and error functions as previous experiments.

Table 4 provides a summary of results using single hidden layer networks trained with a scaled conjugate gradient descent algorithm. Similarly to experiments using the Rprop training algorithm, the network was able to predict the concentrations of single compounds independently with much greater accuracy than the concentrations of all the compounds simultaneously. Networks predicting the concentrations of all six compounds had maximum errors at least 10% greater than the

corresponding single output network in every instance except two (compounds 2 and 3). Often the difference in maximum error between corresponding predictions was 20% greater for networks predicting six concentrations. The average error of prediction did not generally differ by such a large margin, but was usually at least 5 % greater for networks with 6 output. Compound 5 (isobutene) presented a particularly great difference in predictability between networks predicting six and networks predicting one output. For four of five networks with six output the average error for predicting the concentration of compound 5 was more than 20% greater than single predictor networks with the same number of hidden nodes. Moreover, the difference in maximum error between corresponding networks predicting the concentration of compound 5 ranged from 35 % to nearly 75 % .

As seen in table 5, networks trained using the scaled conjugate gradient descent algorithm were significantly less accurate than networks trained using the Rprop algorithm. In particular, neural networks trained with the conjugate gradient descent algorithm predicted the concentrations of compounds 1-4 with less accuracy (methane, ethane, propane, propene). Surprisingly, however, the network predicted the concentrations of compounds 5 and 6 six with better accuracy. In earlier experiments compounds 5 and 6 (isobutane, isobutene) were the most difficult to predict. A comparison between the results of training using the two training algorithms is given in table 6.

One of the primary motivations of the project was to compare the capabilities of a non-linear predictor with traditional linear methods of deconvolution. It was therefore interesting to observe whether the difference of experimental spectra from purely linear combinations of spectra for pure compounds is characteristic of the solution, since the spectra for pure compounds are characteristic. If the spectra do prove to have characteristic deviation from linearity, one would expect that this

deviation would provide information describing the composition of the solution.

To study the significance of this non-linearity, neural networks were trained to predict the composition of solution based upon the difference of the experimental spectra and linear combinations. For this purpose, a computer program subtracted linear combinations of the pure spectra multiplied by the percent concentration of the compound from the experimental spectra. The network was then trained with these vectors representing differences between experimental and purely linear spectra using the fractional composition of the solutions as target vectors. The preliminary results of this experiment were very poor relative to other methods and the experiment was discontinued after only a few trials. The preliminary results are shown in table 6. Although the network predicted the concentrations of compounds 1 and 4 with reasonable accuracy, it performed very poorly for the other compounds. Compounds 2, 3, and 6, which generally were predicted with high accuracy, were predicted with 2 or 3 times greater maximum errors than corresponding networks trained on the experimental spectra.

Another program was written to calculate statistics for the deviation from linearity among spectral averages of the triplicates. For each experimental average spectrum, the program *nlin* multiplies the concentration of each solute by the its spectrum. These weighted spectra are summed and scaled relative to the maximum peak height to generate normalized spectra. This linear combination of spectra was subtracted from the observed normalized spectra and divided by the relative intensity of the peak in the observed normalized spectra to arrive at relative differences from linearity for each peak in each spectrum. The result of this calculation formed a vector of relative differences from linearity, where each vector component represents the relative difference between the actual m/z peak and the height expected for purely linear combinations. This difference vector

was defined $\Delta = S - S_0$. The program calculated the average non-linear term as well as the standard deviation for each m/z peak among the 32 average experimental spectra. (Equivalently, each difference vector component was averaged over the 32 difference vectors.)

For each of the 32 average spectra, the average relative difference between theoretical and experimental peak heights was calculated. The standard deviation among these non-linear terms as well as the magnitude of the difference vector were determined. Similar calculations were performed among the 32 non-linear terms for each of the 23 m/z peaks. The output from this program is included in table 7. Figure 4 plots the average percent non-linearity of each m/z peak.

Analysis of the output from this code showed several interesting trends. The first of these was a systematic trend towards increasingly non-linear spectra with increasing m/z ratio. A line of best fit ($R^2 = .67$) has a slope of .0046 (i.e. $\frac{1}{2}$ % increase in nonlinearity with each m/z unit). In this context a negative deviation means that the height of the actual observed spectra at a given m/z peak is greater than the height calculated for the appropriate linear combination of basis spectra. With the exception of a single point ($m/z = 28$), the deviation from linearity is negative for all m/z peaks where $m/z \leq 42$. Conversely, the deviation is positive for all peaks after 42. The average differences from linearity for each m/z range from -11% to +15%. The standard deviation for each element of the deviation vector among the 32 vectors range from 1.0% to 4.6%.

The magnitude of each difference vector was also calculated, as well as the average component and deviation about the value for the mean component for each vector. As could be expected from the predominance of negative deviations calculated as averages for the component deviations, the average element of most difference vectors is negative. The value for the average

component of each difference vector ranges from -2.3% to +2.0%. The standard deviations calculated for the elements of the difference spectra from the average for non-linearity for that vector range from 4.9% to 7.0%. This standard deviation among the 23 components of each difference vector is much greater than the standard deviation for calculating the average non-linearity for each m/z . As a result, the non-linearity is more a characteristic of m/z ratio than solution composition. Figure 5a displays these results in relation to the relative concentration of each compound. Figure 5b shows these results as a function of isobutane concentration, as isobutane was the only compound for which there was significant variation in concentration.

This output was then used to produce artificial spectra using the program `syndata.f90`, which is also in the appendix. These synthetic spectra were linear combinations of spectra for pure components, with an additional empirical term introduced. This empirical term was a randomly determined factor added to each m/z peak described by a Gaussian centered around the observed mean difference from linearity observed in the experimental spectra for that peak with a spread equal to the standard deviation empirically calculated for that m/z peak. An example of a synthetic spectra generated this way is included in the appendix graphed beside the spectra that would have been produced by purely linear combinations of the basis spectra. A neural network was trained on 1000 randomly generated synthetic spectra using a scaled conjugate gradient descent algorithm. The network was then tested on the experimental spectra. The accuracy of these predictions is included in the table. The concentrations of compounds 1 and 4 (methane, propene) were predicted with moderate accuracy. However, the network predicted the other concentrations, compound 5 (isobutene), in particular, much less accurately than previous networks.

Conclusion:

As expected, the experimental mass spectra demonstrate significant deviation from linear combinations of pure spectra. This deviation cannot be accounted for by experimental noise or imprecision among measurements. Figure 3 shows that for the vast majority of the points in the spectra, the relative standard deviation among measurements was essentially zero. A handful of outliers exist with standard deviations all less than 5 % of the measurement..

The precision among replicate spectral measurements justifies the existence of significant characteristic non-linear terms within the spectra. Furthermore, this non-linearity is characteristic of each peak over 32 spectra rather than spectra with 23 peaks. Figure 4, showing the non-linearity as a function of m/z , illustrates the relative non-linear terms ranging from -15 to 15 % of the experimental average amplitude for the peak. The standard deviation in non-linearity for a given m/z peak among the 32 recorded spectra ranges from 2.2 % to 4.6 %. In comparison, the average of non-linear terms for a given spectra are significantly smaller, falling from -2.5 to 2.5 %. However, the standard deviation (5 to 7%) among these terms for a given spectrum is always much larger than the average measured deviation from linearity.

Artificial neural networks learned to predict the concentrations of the spectra with moderate success. Generalized Regression Neural Networks predicted the solution composition most accurately when predicting the concentration of all compounds simultaneously. The average error for prediction ranged from 21% for methane to 31 and 33% for propene and isobutene, respectively. Surprisingly the maximum errors for prediction were not significantly greater than the average errors, ranging from 22% to 35 and 38% for the same 3 compounds.

Feed-forward back-propagation neural networks predicted the spectra with greater accuracy

than GRNN. Unlike the GRNN, the feed-forward back-propagation networks predicted the fractional concentration of individual compounds with greater accuracy in independent networks than in a single network. Additionally, isobutene was particularly difficult for feed-forward networks to predict, while GRNN did not have special difficulty with it relative to other compounds.

The Rprop algorithm trained the networks to the greatest accuracy. Networks existed with maximum error of prediction 12% or less for every compound except propene (16%) and isobutene (21%). The average error for these networks ranged from 1 to 8%. Networks trained with the scaled conjugate gradient descent algorithm were generally less accurate. Although, there existed networks predicting concentrations for each compound with a maximum error of less than 20 %, the network was unable to train to within less than 15 % maximum error for 3 compounds.

The training set consisted of only 32 spectra and this probably presented a significant problem for the networks. Predictions using 2 or more hidden layers of nodes should be viewed with skepticism, as at this size the network is likely over-fitting the training set by using too many parameters for so little data. Also the range of solution composition over the spectra is very limited. In particular, the fraction of isobutene in solution varied from 10 to 12%. It is then somewhat surprising that Rprop networks predicted the concentration of isobutene with much greater errors than any other compound. The limited data set affects the robustness of the predictor. It is obviously dangerous to generalize the results of this study to solutions of composition outside the range used.

Further work using the networks should be directed toward studying network predictions for isobutene (compound 5). Inclusion of the other concentrations as network input may facilitate prediction. Systematic trends in network prediction could be studied if the code were adjusted to subtract the predicted concentrations of the other compounds to calculate isobutene. Modification

of the source code should allow the network to predict isobutene concentration by backsubtracting the concentration of the other compounds.

Finally, studies performed to this point have not considered instrumental sensitivity of the mass spectroscopy data. Differential sensitivity of mass spec to m/z peaks should be incorporated into calculations of linear combinations of spectra. This may have significant impact upon the calculation of non-linearity within the spectra and may reveal different trends. These trends may prove more successful in generating synthetic spectra for training.

References:

1. Smith, Murray. *Neural Networks for Statistical Modeling*. Van Nostrand Reinhold: New York, New York. 1993.
2. Lawrence, Jeannette. Sylvia Luedeking, ed. *Introduction to Neural Networks*. California Scientific Software: Grass Valley, California. 1991.
3. Campbell, Neil A. *Biology*. 3rd ed. Benjamin/Cummings Publishing Company, Inc.: New York, New York. 1993.
4. Wasserman, Philip D. *Neural Computing: Theory and Practice*. Van Nostrand Reinhold: New York, New York. 1989.
5. Wasserman, Philip D. *Advanced Methods in Neural Computing*. Van Nostrand Reinhold: New York, New York. 1993.
6. Fausett, Laurene. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Prentice Hall: Englewood Cliffs, New Jersey. 1994.

Appendix

Illustrations	i
Figures	vii
Tables	xvii
Plots of Rprop convergence	xxviii
Relevant codes	xlii
M/Z peaks studied	liv
Honor's statement	lv

Illustrations:

1) physiological neuron		ii
2) sigmoidal function		iii
3) feed forward organization		iv
4) back-propagation scheme for training neural networks		v
5) generalized regression neural network schematic		vi

Illustration 1.

A typical physiological neuron. The soma, or cell body, performs the metabolic activities necessary for cell survival. Dendrites receive signals from other neurons across synapses. Activated neurons then propagate action potentials along axons that terminate at synapses with other nerve cells.

Campbell. *Biology*, p.984.

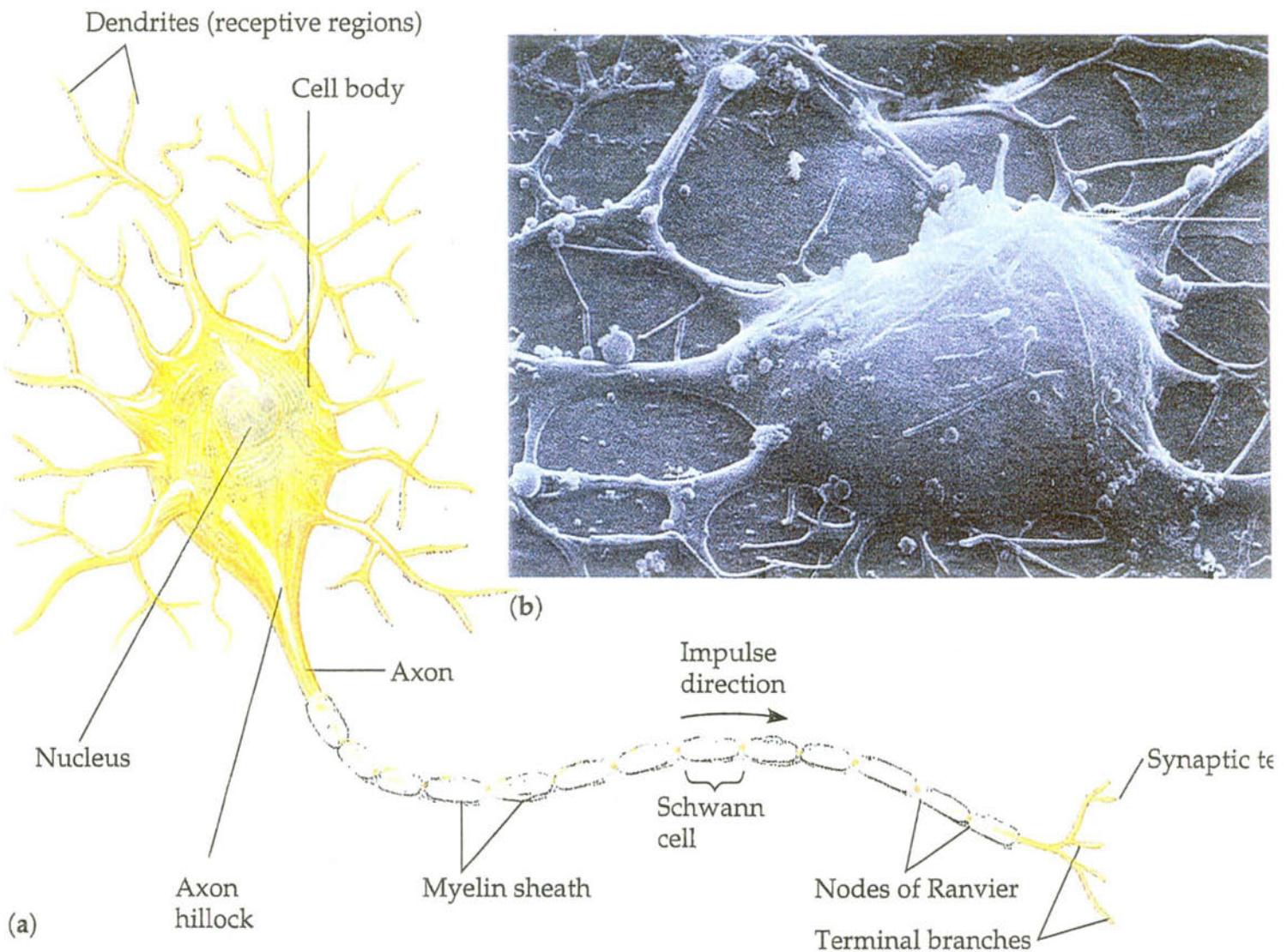
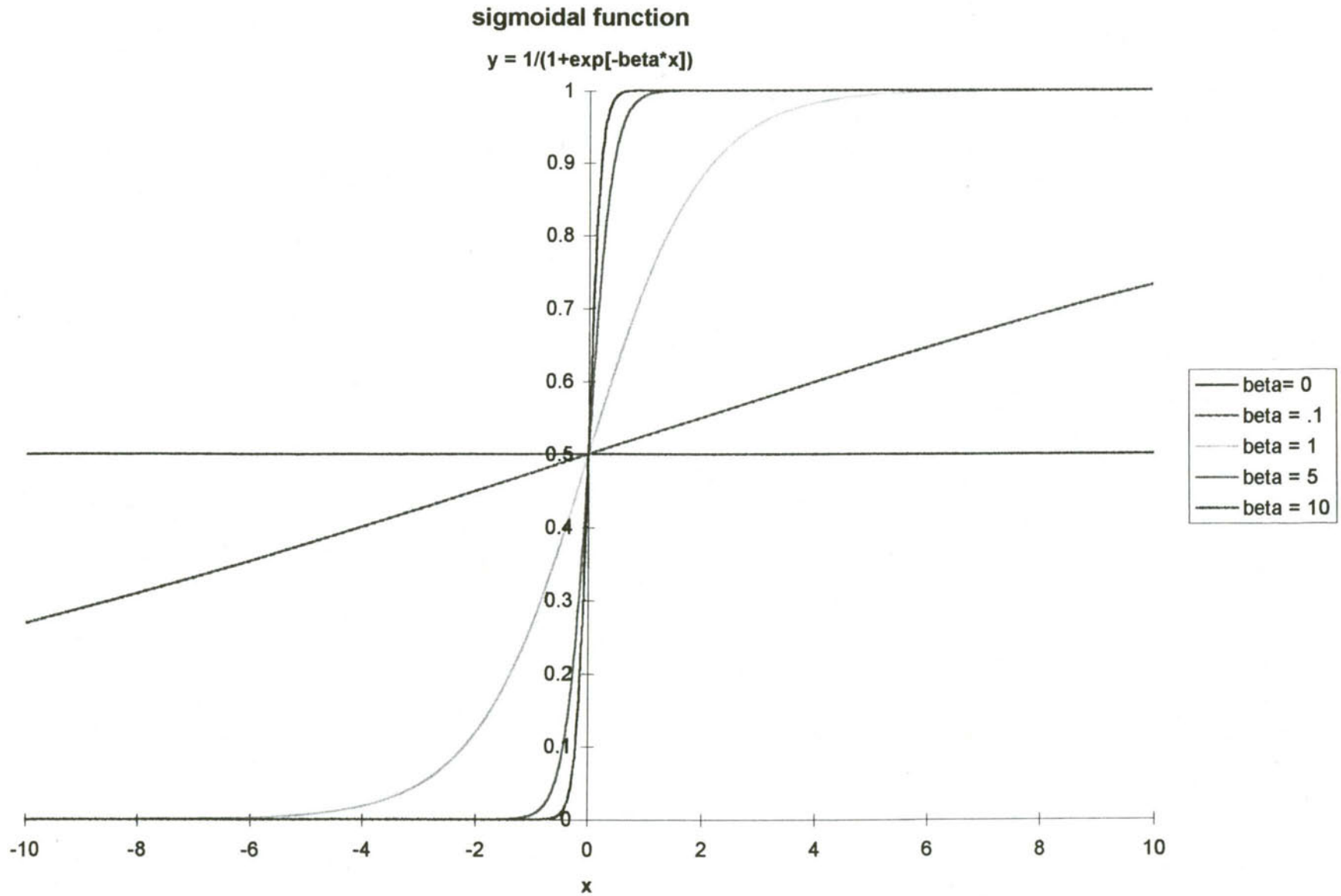


Figure 44.3

The structure of a typical vertebrate neuron.

(a) The cell body of the neuron contains the nucleus and other organelles and gives rise to various branched projections called processes, which are of two types: dendrites and axons. The dendrites typically receive inputs and conduct sig-

nals toward the cell body, while the axon conducts signals away from the cell body. (This direction of the usual flow of information is indicated by the arrow.) At the end of the axon are numerous synaptic terminals, where the neuron makes connections with other neurons or other target cells.



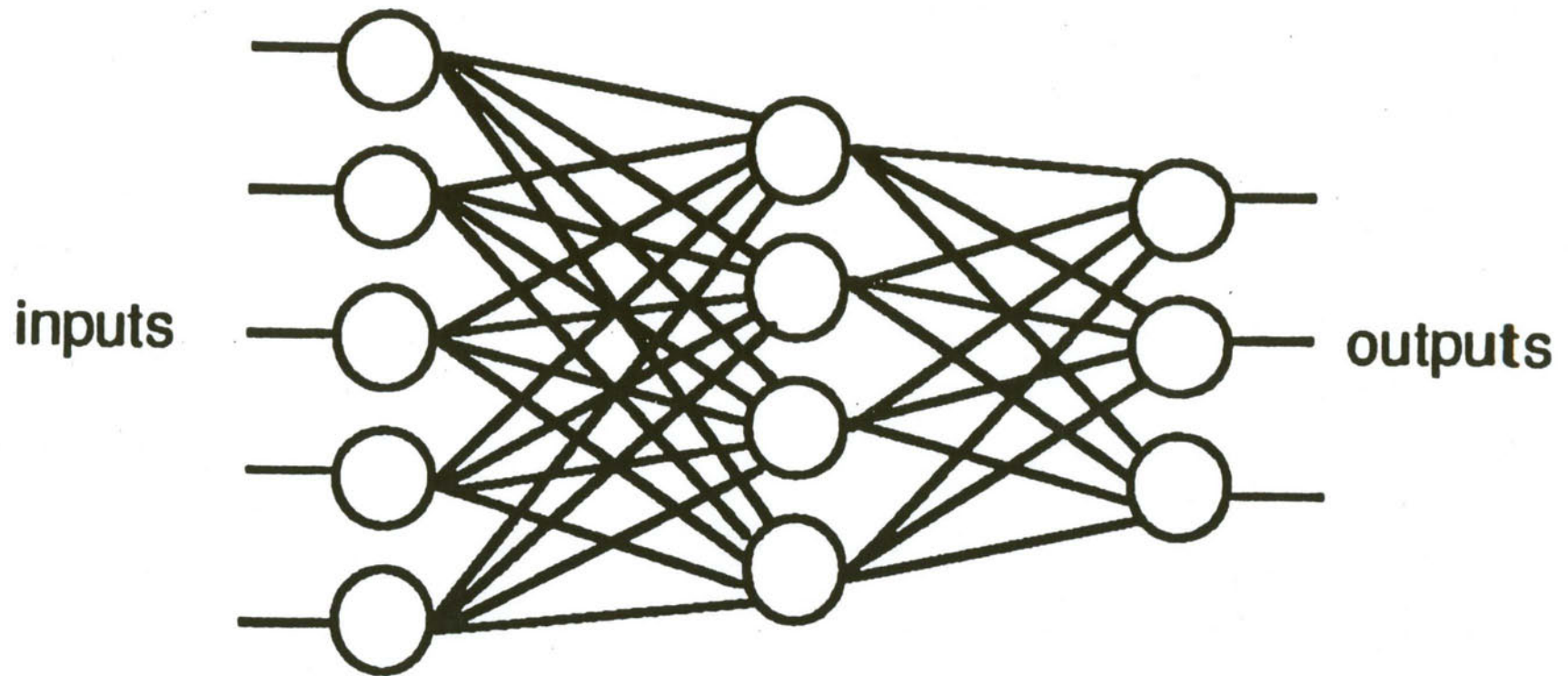


diagram 5.6: A feed forward network

Illustration 3

A typical scheme for a feed forward neural net. Nodes are organized into layers. Nodes within the same layer do not interact, but receive input from each node in the previous layer and transmit output to each node in the next layer. Input nodes receive input from outside of the program. Output nodes perform the final calculations and relay network output to the user. Nodes in layers that do not directly interact with the user are referred to as hidden layers. These nodes perform the bulk of computation.

Lawrence. *Introduction to Neural Networks*, p.74.

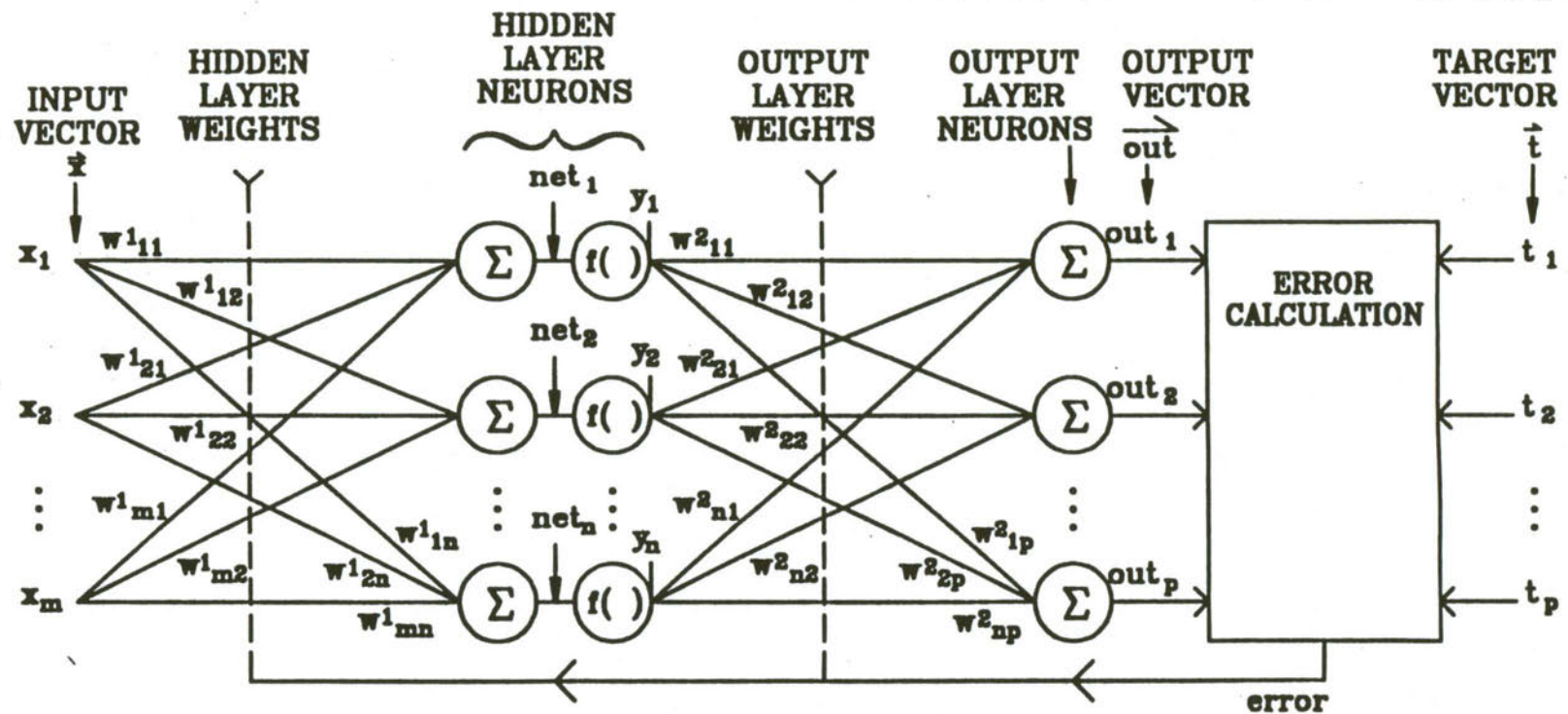


Figure 6-1. Feed-forward neural network.

Illustration 4

A schematic for training in feed forward back-propagation networks. An m -dimensional input vector $\vec{x}$ from the training set is applied to the network. The network computes a p -dimensional output vector, $\vec{out}$, which is compared to the associated target output vector, $\vec{t}$. The error is calculated and propagated back through the network, which adjusts the weights to minimize the error.

Wasserman. *Advanced Methods in Neural Computing*, p.99.

Illustration 5

A typical scheme for a radial basis function network. An m -dimensional input vector $\mathbf{x}$ produces an n -dimensional output vector, $\mathbf{out}$. Each node is associated with a radially valued function, h , centered at a point in m -dimensional space, $\mathbf{u}_i$ and described by a parameter describing its spread, σ_i .

Wasserman, *Advanced Methods in Neural Computing*, p.152.

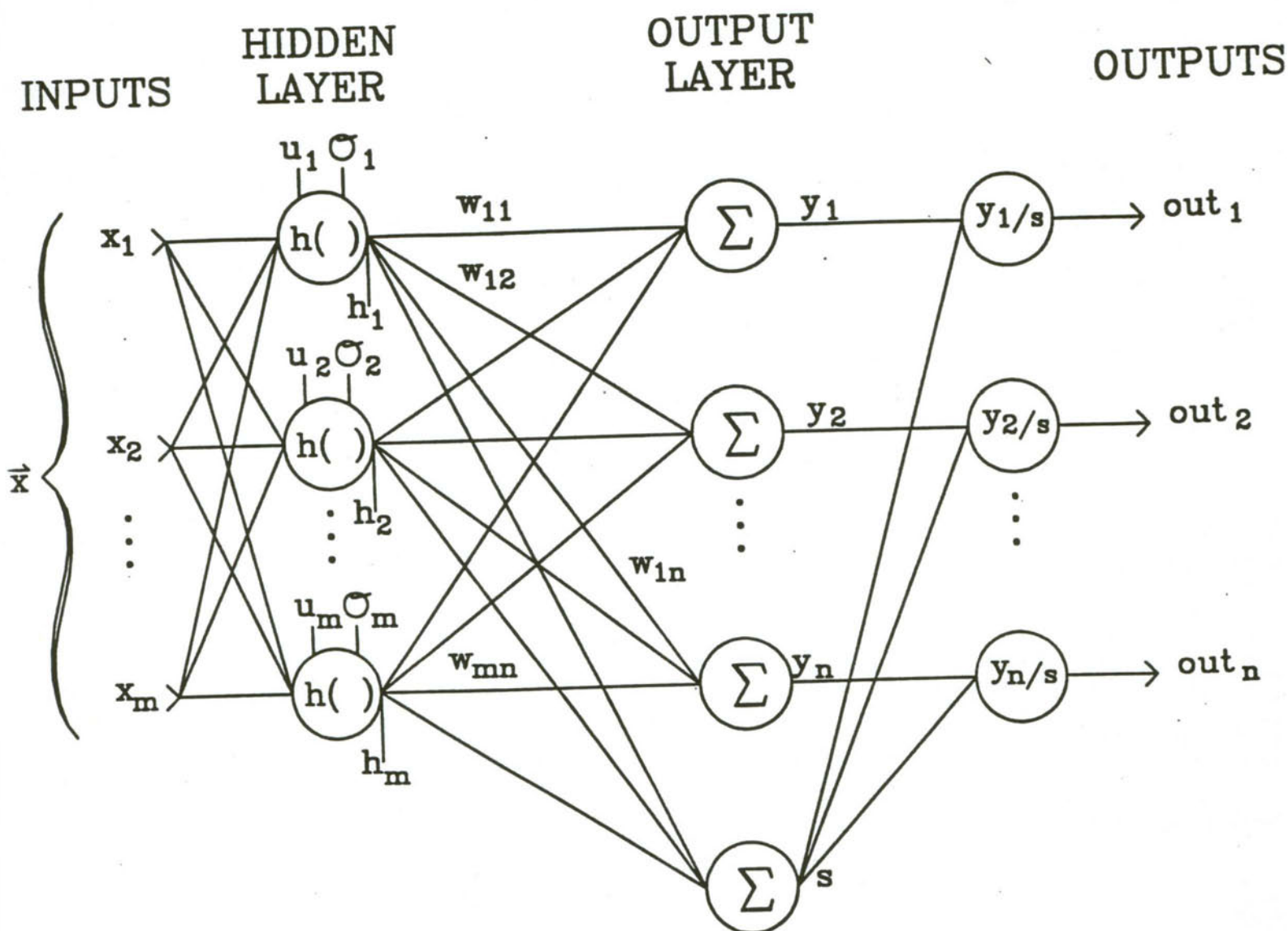
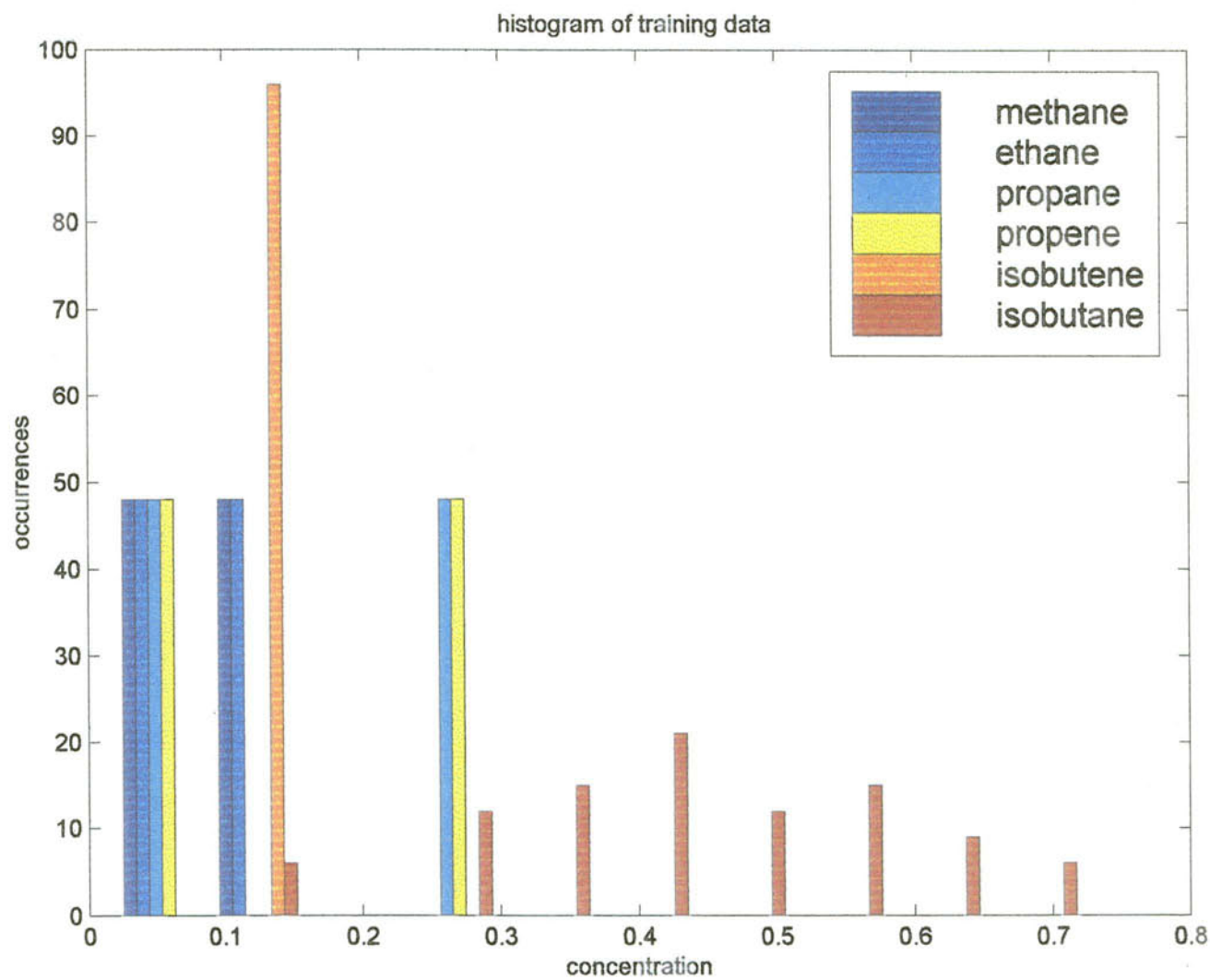
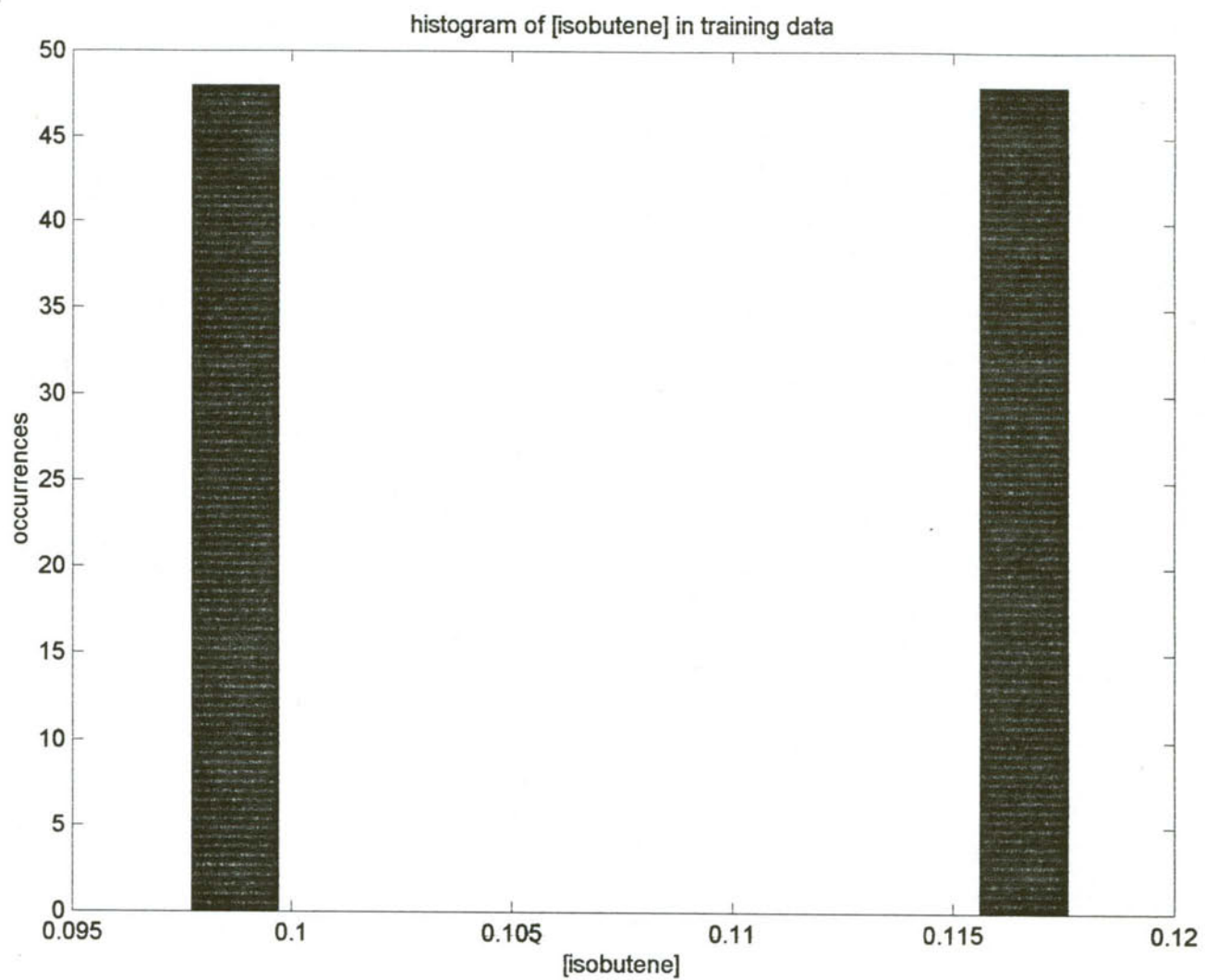


Figure 8-4. Multiple output basis-function network with output normalization.

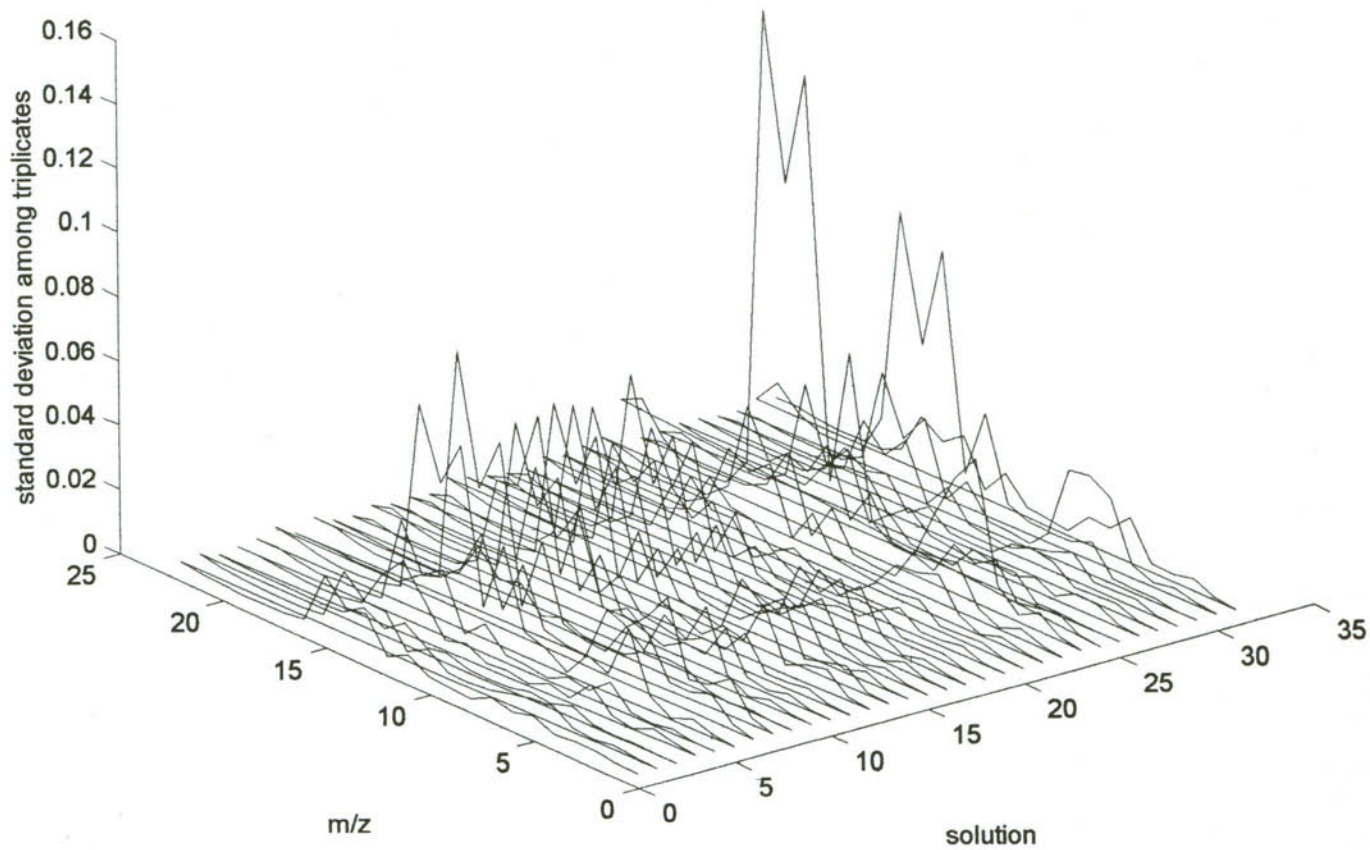
Figures

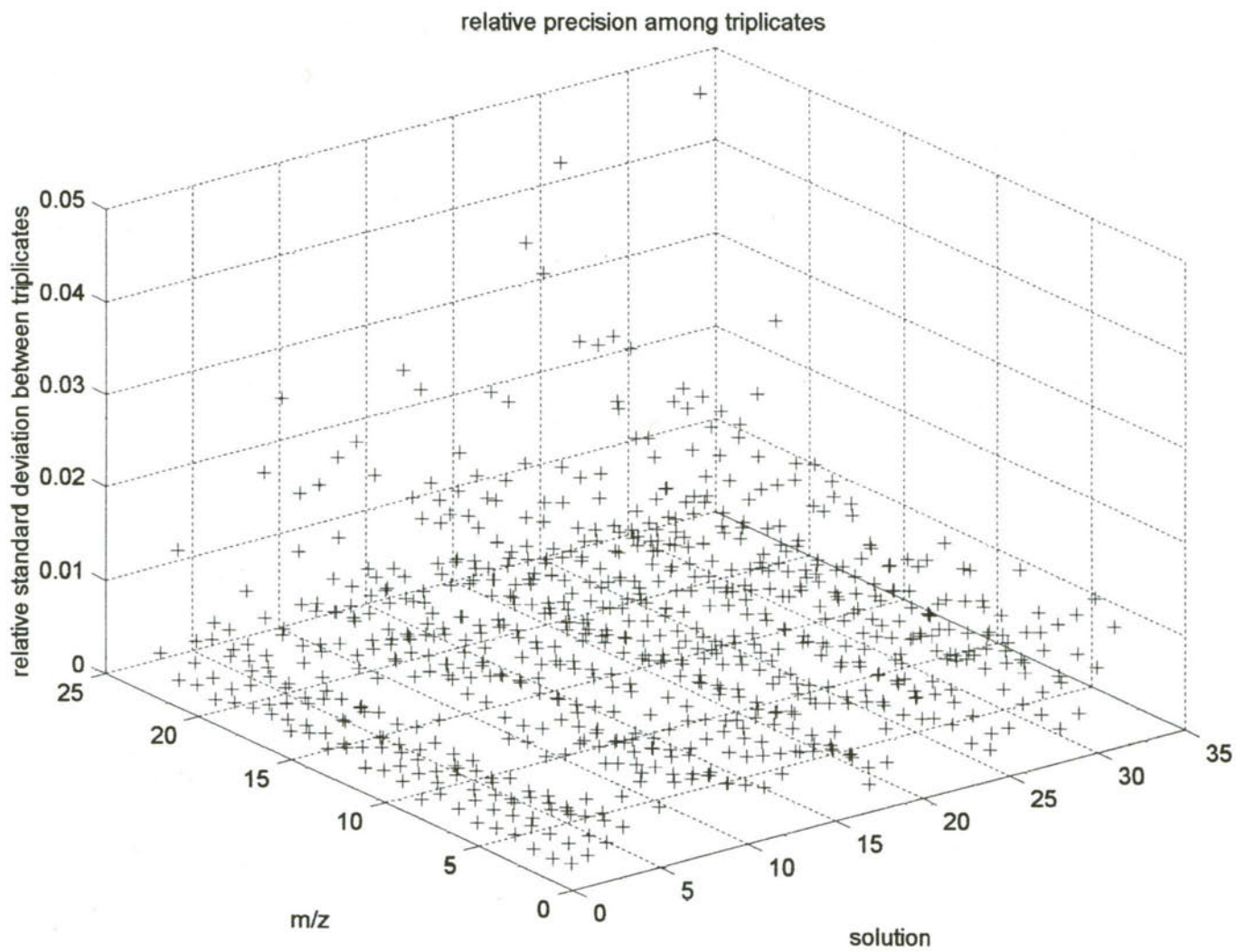
1a) histogram of compound concentration	viii
b) histogram of isobutene concentration	ix
2a) precision among triplicates	x
b) relative precision among triplicates	xi
3a) analysis of experimental deviation within triplicates	xii
b) average standard deviation among triplicates	xiii
4) component analysis	xiv
5a) non-linearity as a function of solution concentration	xv
b) non-linearity as a function of isobutane concentration	xvi

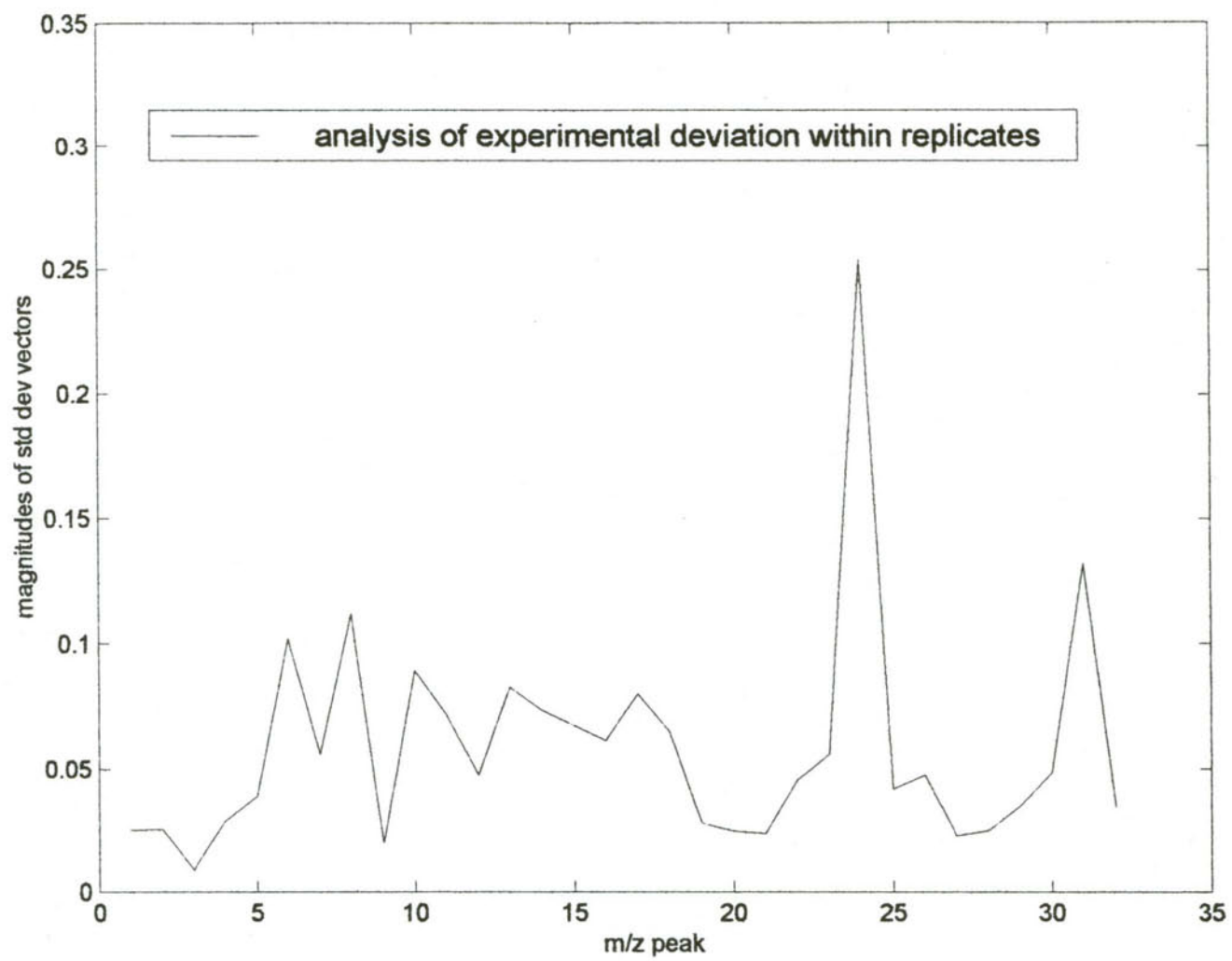


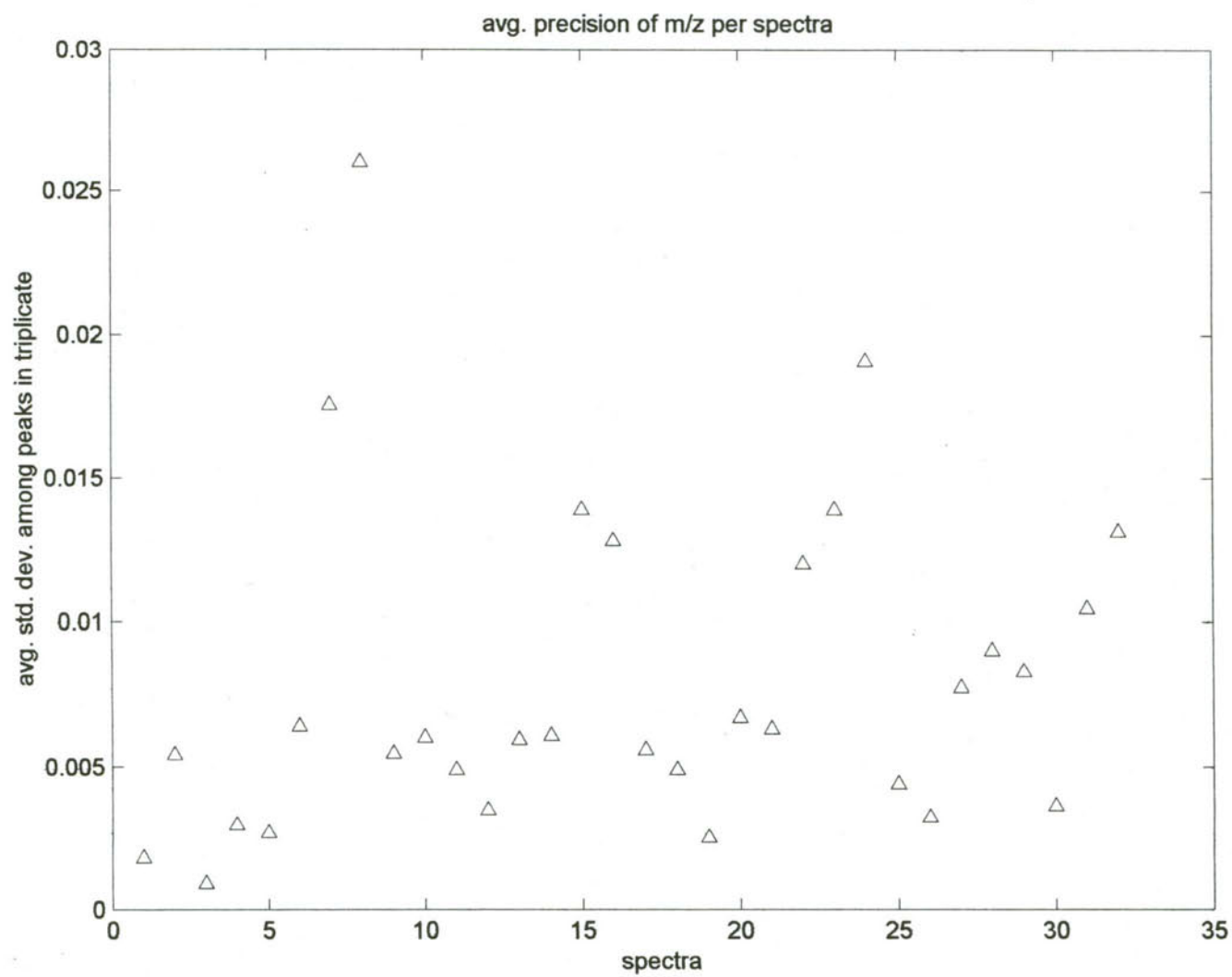


precision among triplicates

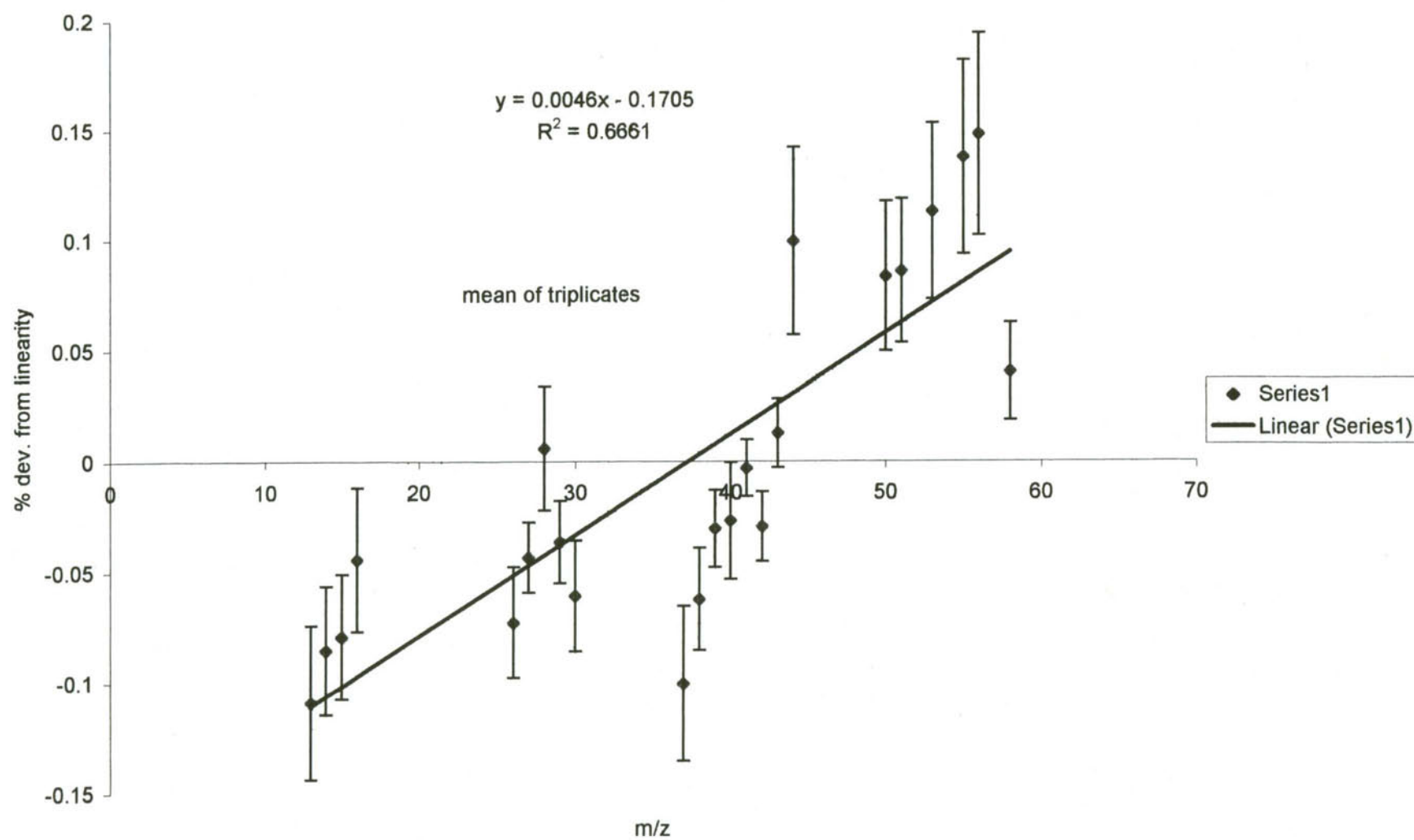




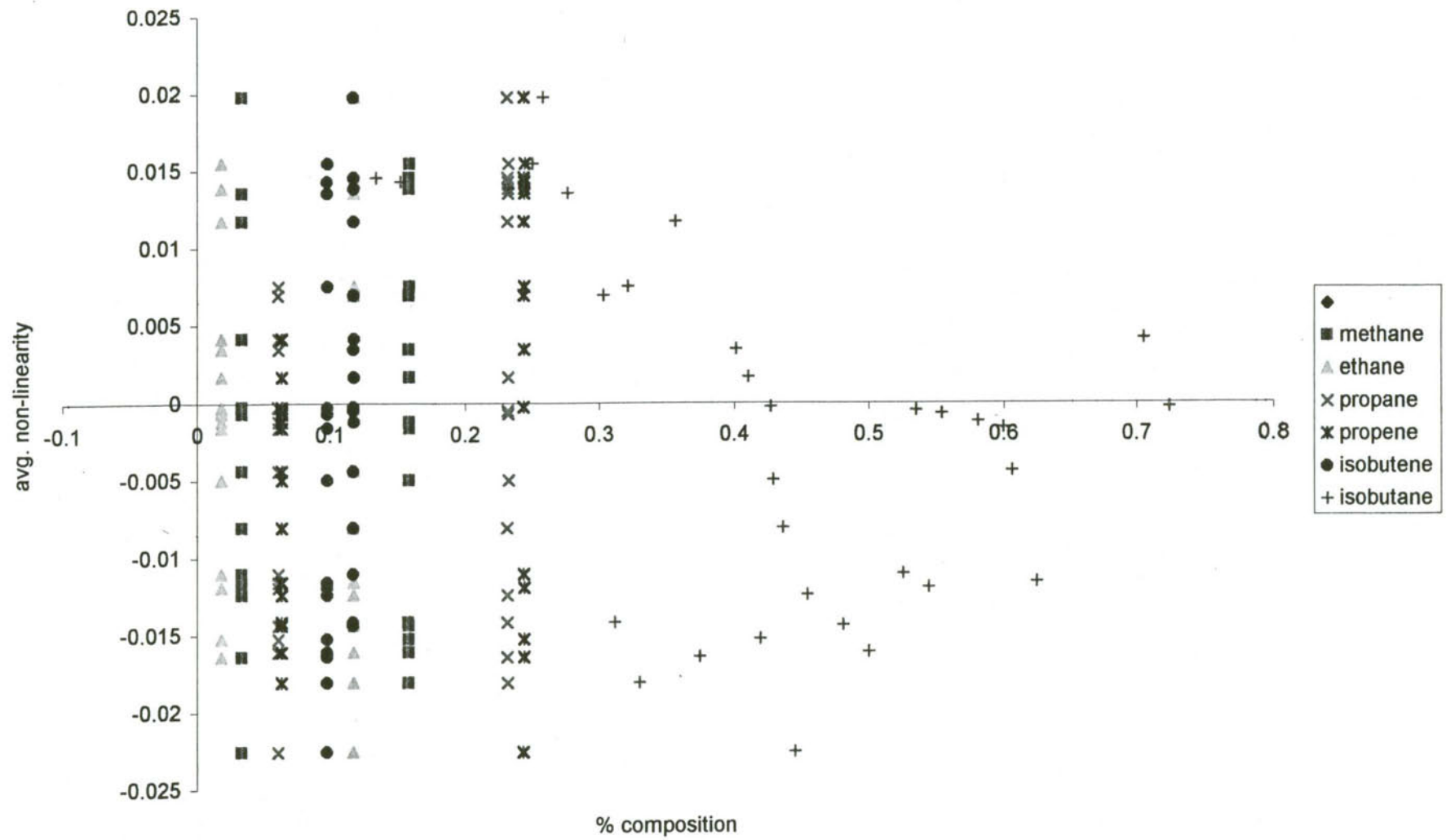




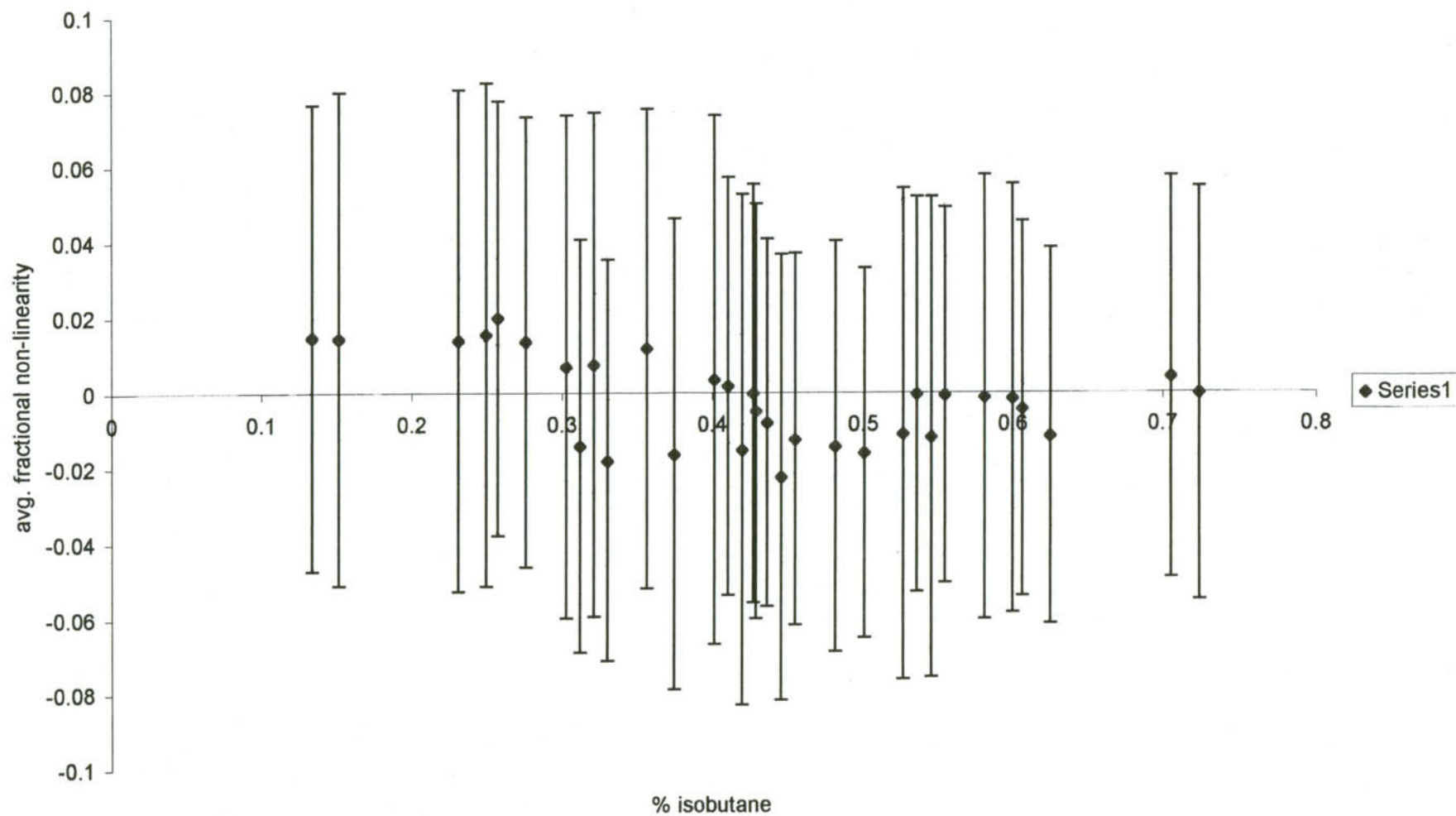
component analysis
% deviation from linearity vs. m/z



average non-linearity
vs. comp.



isobutane
avg. fractional non-linearity of spectra components
as a function of % isobutane



Tables

1) Summary of GRNN training	
a) GRNN training with 10 re-samplings	 xviii
b) GRNN training with 100 re-samplings	 xix
2) Summary of error convergence with re-sampling in networks trained with Rprop algorithm	 xx
3) Summary of Rprop training	 xxi
4) Summary of error in networks trained with scaled conjugate gradient descent algorithm	 xxii
5) Comparison of networks trained with Rprop and scaled conjugate gradient descent algorithm	 xxiii
6) Summary of error in networks using dS matrix as training data	 xxiv
7) Statistical analysis of experimental spectra	 xxv
8) Summary of error in networks using synthetic spectra as training data	 xxvii

table 1a

Summary of error for GRNN with 10 re-samplings

	avg. errors		max. errors	
	mean	std	mean	std
all				
- out1	16.56266	2.387481	37.80107	7.843845
- out2	15.04833	0.241002	42.05457	9.654558
- out3	18.82541	3.425133	48.72954	8.962862
- out4	19.94127	4.156942	44.73306	6.144901
- out5	27.60717	2.990756	53.15907	8.248805
- out6	20.75584	5.461684	52.02293	6.748798
out1	25.98337	7.706221	86.53269	15.21375
out2	13.66259	4.207658	65.97676	10.87094
out3	19.23096	2.699475	83.81093	11.60181
out4	31.21177	0.783743	92.43827	7.610597
out5	74.46403	9.289097	109.4189	3.352017
out6	32.47452	0.879099	89.98789	6.780278

table 1b

Error Summary for GRNN with 100 re-sampling

	avg. error		max % err.	
	mean	std.	mean	std
all				
- out1	20.70647	3.427918	22.36601	3.106585
- out2	23.22434	7.316148	26.30457	7.610231
- out3	25.23753	6.516983	27.24182	5.827828
- out4	31.91518	8.332401	35.33356	9.998353
- out5	33.85049	12.18589	38.18668	13.84708
- out6	22.34491	4.549942	26.33683	3.056703

	avg. error		max % err.	
	mean	std	mean	std
out1	25.59845	1.83841	87.51071	2.154372
out2	18.72378	3.022949	72.40205	4.392138
out3	22.57595	0.95409	85.02489	1.152462
out4	36.70417	2.214681	95.66114	1.206327
out5	76.46859	1.402957	110.1835	0.976411
out6	34.10085	1.686427	93.13903	1.709759

table 2

Convergence of prediction errors of networks trained with Rprop algorithm

	(resampling: 3)		(resampling: 5)		(resampling: 10)		(resampling: 100)	
	avg % err.	max err.	avg % err.	max err.	avg % err.	max err.	avg.% err.	max. err.
component 1								
23->3->1	0.900038	3.227698	1.389269	3.671652	1.526219	4.726506	1.875768	8.371139
23->5->1	0.979042	4.015583	0.77908	2.954765	0.911689	3.037317	1.582541	7.463859
23->7->1	2.898803	11.4653	2.058076	7.625224	1.580723	5.844707	1.491769	6.146308
component 2								
23->3->1	0.69595	3.079355	0.577029	2.511954	0.764916	4.355514	1.820672	9.936958
23->5->1	0.719026	3.956917	0.930194	5.444761	1.657814	8.053644	1.716655	9.052915
23->7->1	0.973021	3.521112	1.018134	4.263822	0.781997	3.350493	1.583868	7.784387
component 3								
23->3->1	5.657658	35.38598	4.173508	25.40824	2.98028	15.56264	3.383516	19.21859
23->5->1	5.632734	42.62712					3.358766	20.95414
23->7->1	4.127717	24.76233					3.321621	18.87217
23->9->1	3.286931	24.13552	3.678431	25.04255	2.670484	17.55561	3.570097	21.32842
23->11->1	3.690385	32.22926	2.68242	20.58423	4.113739	26.69946	2.845648	15.71374
23->13->1	5.590722	36.22888	5.32148	26.31691	4.153757	21.31105	3.31663	19.19737
component 4								
23->3->1	2.315092	5.099878					3.205614	10.54722
23->5->1	4.239189	11.86606					3.109023	9.736809
23->7->1	1.858755	6.753367	1.815879	6.394004	2.471255	7.466185	3.106925	9.245844
23->9->1	2.233451	4.868468	2.604417	6.797573	2.384755	6.805819	3.061737	9.545032
23->11->1	2.006427	6.665687	1.982923	6.790865	2.249135	8.072443	2.68949	8.31011
23->13->1	2.836911	10.43573					2.773543	8.383466
component 5								
23->3->1	4.081864	13.60674	9.210715	26.51507			7.787429	22.60976
23->5->1	7.897307	19.35731	7.004729	18.19607			8.405558	24.45915
23->7->1	5.434866	21.56318	6.189896	22.12471	5.823522	17.59742	9.172284	26.2025
23->9->1	3.847737	11.23313	4.786747	17.19227	4.738179	16.03931	8.234231	23.92159
23->11->1	5.534502	13.40319	4.392021	9.902669	7.013566	18.21838	7.603944	21.35958
23->13->1	13.16721	35.06879	9.438505	25.12813	8.286002	21.87045	7.945209	22.86464
component 6								
23->3->1	4.27421	14.90023					4.28085	19.5635
23->5->1	3.753625	12.93189	4.273102	21.05307	3.796639	18.58344	3.565317	17.21727
23->7->1	3.051094	11.63011	2.781902	9.230928	2.961454	10.37184	3.645163	14.54494
23->9->1	2.800074	9.482873	3.802316	16.69867	3.894172	15.56181	3.604075	15.24357
23->11->1	3.048866	17.75561					3.032735	12.06363
23->13->1							3.536607	14.65735

table 3

Summary of errors for networks trained with the Rprop algorithm									
<u>individual components</u>			<u>entire solution</u>			<u>entire solution</u>			
	avg.% err.	max. err.		avg.% err.	max. err.		avg.% err.	max err.	
out1									
3	1.875768	8.371139	3	4.535059	13.23845	3,3	4.428165	10.90852	
5	1.582541	7.463859	5	25.45427	67.5412	3,5	5.100531	13.41026	
7	1.491769	6.146308	7	4.935121	20.05268	3,7	6.956166	19.40834	
9	1.787423	8.158305	9	3.705296	13.63822	5,3	5.083733	12.13704	
11	1.191575	4.597734	11	4.649037	18.66643	5,5	2.284373	5.946995	
out2									
3	1.820672	9.936958	3	4.735859	9.846632	3,3	4.601196	8.146855	
5	1.716655	9.052915	5	23.23063	63.88605	3,5	7.013138	23.27133	
7	1.583868	7.784387	7	3.789111	14.58678	3,7	8.328069	16.2628	
9	1.992659	10.33645	9	2.898979	10.78802	5,3	4.418253	10.0845	
11	1.147512	5.517378	11	3.397823	14.14547	5,5	2.732453	8.32434	
out3									
3	3.383516	19.21859	3	8.347183	16.1321	3,3	5.641262	14.92258	
5	3.358766	20.95414	5	27.19308	69.95159	3,5	6.320565	11.3054	
7	3.321621	18.87217	7	6.984454	32.00767	3,7	7.114686	21.8813	
9	3.570097	21.32842	9	5.35359	21.07021	5,3	5.998969	19.59931	
11	2.845648	15.71374	11	5.437289	23.30536	5,5	3.01447	6.822129	
13	3.31663	19.19737							
15	3.555551	22.0283							
out4									
3	3.205614	10.54722	3	8.424801	15.00775	3,3	4.369133	13.06863	
5	3.109023	9.736809	5	25.28372	61.09335	3,5	5.920683	12.8522	
7	3.106925	9.245844	7	6.778574	19.43932	3,7	8.427239	23.42449	
9	3.061737	9.545032	9	5.500587	15.67155	5,3	9.421378	25.44405	
11	2.68949	8.31011	11	5.348993	16.09695	5,5	3.18677	7.294748	
13	2.773543	8.383466							
15	2.593431	8.246513							
17	3.052221	9.108615							
out5									
3	7.787429	22.60976	3	10.50165	47.07878	3,3	10.56717	15.37237	
5	8.405558	24.45915	5	41.46879	88.52113	3,5	8.488215	19.03257	
7	9.172284	26.2025	7	13.32947	39.24092	3,7	13.97126	97.15998	
9	8.234231	23.92159	9	12.04862	34.39338	5,3	17.47784	80.84656	
11	7.603944	21.35958	11	11.56976	35.58741	5,5	10.59584	30.79658	
13	7.945209	22.86464							
out6									
3	4.28085	19.5635	3	3.067277	13.12718	3,3	2.662561	5.930258	
5	3.565317	17.21727	5	14.95773	45.57058	3,5	3.631436	16.90378	
7	3.645163	14.54494	7	6.548359	28.20182	3,7	7.35833	27.98236	
9	3.604075	15.24357	9	4.56636	17.12147	5,3	5.594043	18.90233	
11	3.032735	12.06363	11	4.065644	15.43206	5,5	2.350181	4.646427	
13	3.536607	14.65735							

table 4

Summary of errors for networks trained with scaled conjugate gradient descent algorithm

	<u>single prediction</u>		<u>group prediction</u>	
	avg. err.	max. err.	avg. err.	max. err.
output 1				
3	1.827376	7.76541	3	23.8474 60.18851
5	1.931972	7.384161	5	12.38448 31.41208
7	1.737606	6.327071	7	10.70581 29.95492
9	1.86924	7.029675	9	10.28015 28.15853
11	3.228335	8.230346	11	8.895982 24.08147
output 2				
3	2.481163	12.32247	3	18.92465 50.57401
5	2.377371	11.86749	5	7.66168 24.71836
7	2.708594	12.11763	7	7.378339 23.62265
9	2.628159	13.45129	9	6.633426 20.4603
11	2.418532	8.423113	11	6.612305 22.86073
output 3				
3	5.047917	30.07136	3	26.69486 67.41367
5	4.421536	25.24833	5	13.61532 35.77672
7	4.446441	22.77991	7	11.41378 33.12102
9	4.098636	21.99147	9	10.53517 27.97829
11	4.240508	17.50396	11	10.1996 30.65437
output 4				
3	3.53475	10.8753	3	25.32015 57.73321
5	3.32931	9.399576	5	14.14611 32.06069
7	3.464803	9.228732	7	12.68896 29.56506
9	3.480385	9.487152	9	10.96997 25.39016
11	4.104034	10.47939	11	10.36654 25.47553
output 5				
3	5.484759	15.80598	3	46.82405 90.60326
5	5.025775	14.56219	5	33.29113 68.07477
7	5.421071	15.63494	7	25.76926 57.77999
9	5.24611	14.98171	9	25.25124 55.51535
11	6.025807	15.71551	11	22.9834 51.60285
output 6				
3	4.589919	15.16048	3	15.34499 43.67104
5	3.688397	12.34563	5	12.10535 35.10246
7	3.958551	12.54765	7	11.34991 33.60364
9	3.348481	10.60771	9	9.248043 27.28092
11	3.792792	10.63748	11	8.17838 23.58764

table 5

Comparison of networks trained with Rprop and conjugate gradient descent algorithm

	<u>Rprop</u>			<u>conjugate gradient descent</u>		
		avg. % err.	max. err.		avg. err.	max. err.
out1	3	1.875768	8.371139	3	1.827376	7.76541
	5	1.582541	7.463859	5	1.931972	7.384161
	7	1.491769	6.146308	7	1.737606	6.327071
	9	1.787423	8.158305	9	1.86924	7.029675
	11	1.191575	4.597734	11	3.228335	8.230346
out2	3	1.820672	9.936958	3	2.481163	12.32247
	5	1.716655	9.052915	5	2.377371	11.86749
	7	1.583868	7.784387	7	2.708594	12.11763
	9	1.992659	10.33645	9	2.628159	13.45129
	11	1.147512	5.517378	11	2.418532	8.423113
out3	3	3.383516	19.21859	3	5.047917	30.07136
	5	3.358766	20.95414	5	4.421536	25.24833
	7	3.321621	18.87217	7	4.446441	22.77991
	9	3.570097	21.32842	9	4.098636	21.99147
	11	2.845648	15.71374	11	4.240508	17.50396
	13	3.31663	19.19737			
out4	15	3.555551	22.0283			
	3	3.205614	10.54722	3	3.53475	10.8753
	5	3.109023	9.736809	5	3.32931	9.399576
	7	3.106925	9.245844	7	3.464803	9.228732
	9	3.061737	9.545032	9	3.480385	9.487152
	11	2.68949	8.31011	11	4.104034	10.47939
	13	2.773543	8.383466			
	15	2.593431	8.246513			
out5	17	3.052221	9.108615			
	3	7.787429	22.60976	3	5.484759	15.80598
	5	8.405558	24.45915	5	5.025775	14.56219
	7	9.172284	26.2025	7	5.421071	15.63494
	9	8.234231	23.92159	9	5.24611	14.98171
	11	7.603944	21.35958	11	6.025807	15.71551
out6	13	7.945209	22.86464			
	3	4.28085	19.5635	3	4.589919	15.16048
	5	3.565317	17.21727	5	3.688397	12.34563
	7	3.645163	14.54494	7	3.958551	12.54765
	9	3.604075	15.24357	9	3.348481	10.60771
	11	3.032735	12.06363	11	3.792792	10.63748
	13	3.536607	14.65735			

table 6

Summary of errors for networks trained with difference vectors

resampling: 100

training algorithm: Rprop

all output:

23->3->3->6	26.55919	72.85822
	31.41016	77.43961
	32.25219	83.48596
	19.45945	54.51325
	46.3377	106.644
	20.87569	65.84743

23->5->5->6	12.33039	41.9322
	15.52345	50.92508
	29.60899	89.65151
	7.580453	27.0317
	52.93933	134.1372
	19.58014	63.48158

all output \ 5

23->5->5->5	7.298111	28.62576
	14.36009	50.63424
	25.03945	76.16302
	4.2413	15.85852
	18.91915	64.88762

output 1:

23->5->1	5.086556	14.96406
23->7->1	4.699104	13.94952

output 2:

23->5->1	10.46415	33.42968
23->7->1	10.84995	33.35932

output 3:

23->9->1	20.09951	62.73922
23->11->1	19.91627	62.33355
23->13->1	21.67834	67.1376

output 4:

23->15->1	3.083209	8.384586
-----------	----------	----------

output 5:

23->11->1	34.71608	96.56348
23->13->1	35.66671	95.15925
23->5->9->1	41.48022	105.4514

output 6:

23->11->1	12.47899	42.6462
23->13->1	13.68984	46.13524

table 7

! output from c:\wgn\NNf99\linear\f\bak\nout2.out

vector analysis

vector:	1	norm:	.2672424	avg:	-.0015996	std:	.0569528
vector:	2	norm:	.2405667	avg:	-.0115639	std:	.0499075
vector:	3	norm:	.2336923	avg:	-.0006990	std:	.0498182
vector:	4	norm:	.3049849	avg:	-.0119145	std:	.0638716
vector:	5	norm:	.2510009	avg:	.0041056	std:	.0533487
vector:	6	norm:	.2576922	avg:	-.0002722	std:	.0549394
vector:	7	norm:	.2990065	avg:	.0145679	std:	.0619838
vector:	8	norm:	.2426898	avg:	-.0160745	std:	.0490618
vector:	9	norm:	.2393032	avg:	-.0123980	std:	.0494197
vector:	10	norm:	.3040436	avg:	-.0163910	std:	.0626183
vector:	11	norm:	.3106937	avg:	-.0110289	std:	.0652732
vector:	12	norm:	.2767892	avg:	-.0012107	std:	.0589987
vector:	13	norm:	.3268804	avg:	-.0152517	std:	.0679240
vector:	14	norm:	.2597046	avg:	-.0049609	std:	.0551364
vector:	15	norm:	.2344567	avg:	-.0043787	std:	.0497854
vector:	16	norm:	.2982871	avg:	-.0225437	std:	.0592706
vector:	17	norm:	.2461826	avg:	-.0004873	std:	.0524839
vector:	18	norm:	.3041804	avg:	.0117094	std:	.0637367
vector:	19	norm:	.3302276	avg:	.0034339	std:	.0703172
vector:	20	norm:	.2649367	avg:	-.0143541	std:	.0545446
vector:	21	norm:	.2646689	avg:	-.0180955	std:	.0533080
vector:	22	norm:	.2881970	avg:	.0135545	std:	.0598604
vector:	23	norm:	.2321197	avg:	-.0080573	std:	.0487975
vector:	24	norm:	.2608927	avg:	-.0002438	std:	.0556219
vector:	25	norm:	.3226763	avg:	.0154683	std:	.0669521
vector:	26	norm:	.2605044	avg:	.0016620	std:	.0555137
vector:	27	norm:	.3163292	avg:	.0075223	std:	.0670016
vector:	28	norm:	.2657065	avg:	-.0141566	std:	.0547683
vector:	29	norm:	.3150785	avg:	.0069372	std:	.0667994
vector:	30	norm:	.3196948	avg:	.0138639	std:	.0666688
vector:	31	norm:	.2872608	avg:	.0197816	std:	.0578079
vector:	32	norm:	.3151918	avg:	.0142833	std:	.0655929

For the complete set of vectors the mean values for the following:

vector norm:	.2794026
average componential deviation:	-.0018372
relative standard deviation:	.0583777

analysis by component:

component:	1	mean:	-.1092615	std:	.0348484
component:	2	mean:	-.0855511	std:	.0290786
component:	3	mean:	-.0793431	std:	.0281924
component:	4	mean:	-.0444126	std:	.0325903
component:	5	mean:	-.0727997	std:	.0250101
component:	6	mean:	-.0434744	std:	.0159976
component:	7	mean:	.0058910	std:	.0280817
component:	8	mean:	-.0363801	std:	.0186599
component:	9	mean:	-.0608292	std:	.0250172
component:	10	mean:	-.1006297	std:	.0352064
component:	11	mean:	-.0623799	std:	.0231447
component:	12	mean:	-.0304496	std:	.0174729
component:	13	mean:	-.0269460	std:	.0264871
component:	14	mean:	-.0031157	std:	.0128074
component:	15	mean:	-.0295606	std:	.0158006
component:	16	mean:	.0126709	std:	.0156406

component:	17	mean:	.0994453	std:	.0425579
component:	18	mean:	.0834735	std:	.0340066
component:	19	mean:	.0859200	std:	.0326680
component:	20	mean:	.1130319	std:	.0399435
component:	21	mean:	.1375702	std:	.0442276
component:	22	mean:	.1479361	std:	.0460206
component:	23	mean:	.0404023	std:	.0220045

all 6 components

3	23.6514	59.8258						
	19.69248	54.50677		output 1				
	33.96904	77.25164				3	7.684353	14.28387
	51.31199	113.1693				5	8.260033	18.0164
	18.1166	40.02551				7	8.266618	18.40476
	28.78729	40.02926				9	7.214067	12.79004
						11	6.630808	14.66956
5	14.43781	41.95458				13	7.801867	17.54347
	16.66015	51.66843			3,3		7.272126	12.69216
	13.12094	34.48113						
	28.45485	59.82518		output 2				
	7.995706	17.11751				3	6.800414	20.88541
	7.458151	9.262978				5	8.084137	35.76761
						7	10.49095	30.8679
7	14.81865	36.06386						
	19.20083	63.90417		output 3				
	14.67254	45.24207				3	4.460556	11.48128
	26.37489	61.43571				5	6.156462	13.63081
	8.677609	22.30017				7	7.207002	20.43836
	7.794196	11.54837				9	5.846857	11.83307
						11	5.925042	12.60651
9	16.62778	36.94844				13	6.98146	20.88604
	18.63313	59.16512						
	11.81829	28.51824		output 4				
	25.99765	53.89827				3	26.6979	58.26362
	7.893457	7.114601				5	23.59265	48.20838
	6.879716	10.32961				7	23.19394	46.24497
						9	23.51921	47.5852
11	13.67973	37.06952				11	24.12766	49.912
	15.47285	49.85826				13	23.48177	49.70149
	11.89529	28.88481				15	129.5572	213.1021
	26.63023	59.32285						
	9.461969	18.53481		output 5				
	4.282503	8.667038				3	11.46692	18.0195
						5	13.70178	18.78899
13	13.66619	34.25002				7	13.27984	16.1475
	19.76397	55.47872				9	13.19703	15.97831
	13.69702	43.55702				11	13.35648	17.38862
	26.55803	56.67937				13	14.41209	17.84034
	11.07388	23.10814						
	7.37508	14.0695		output 6				
						3	8.00165	14.38361
						5	4.840116	9.647195
						7	5.017542	9.345578
						9	5.178855	8.848963
						11	4.275376	8.754382
						13	5.140564	10.85612

Rprop convergence

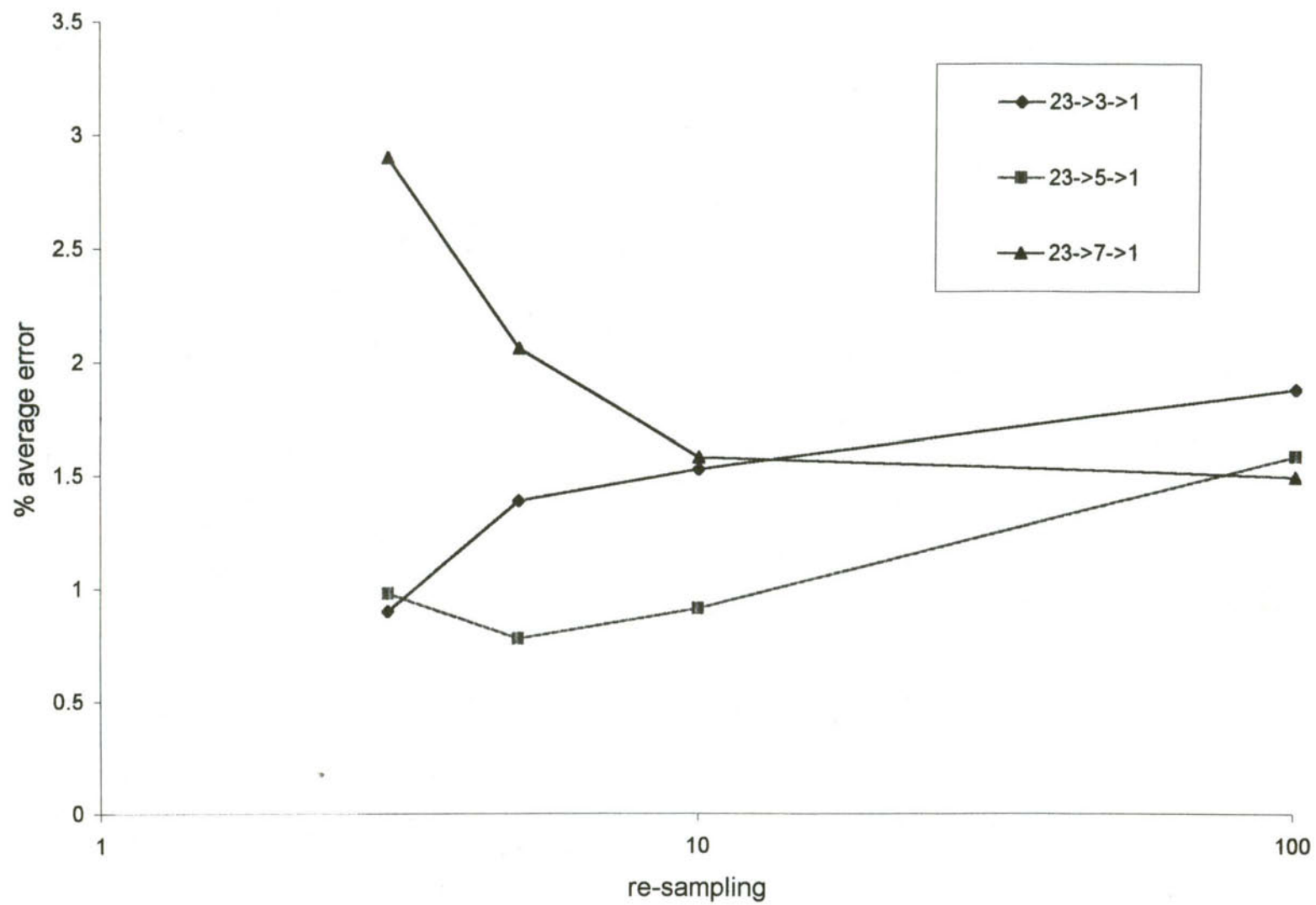
1) table 2	 xxix
2) logarithmic plots of average and maximum error against re-sampling number	 xxx

Sheet1

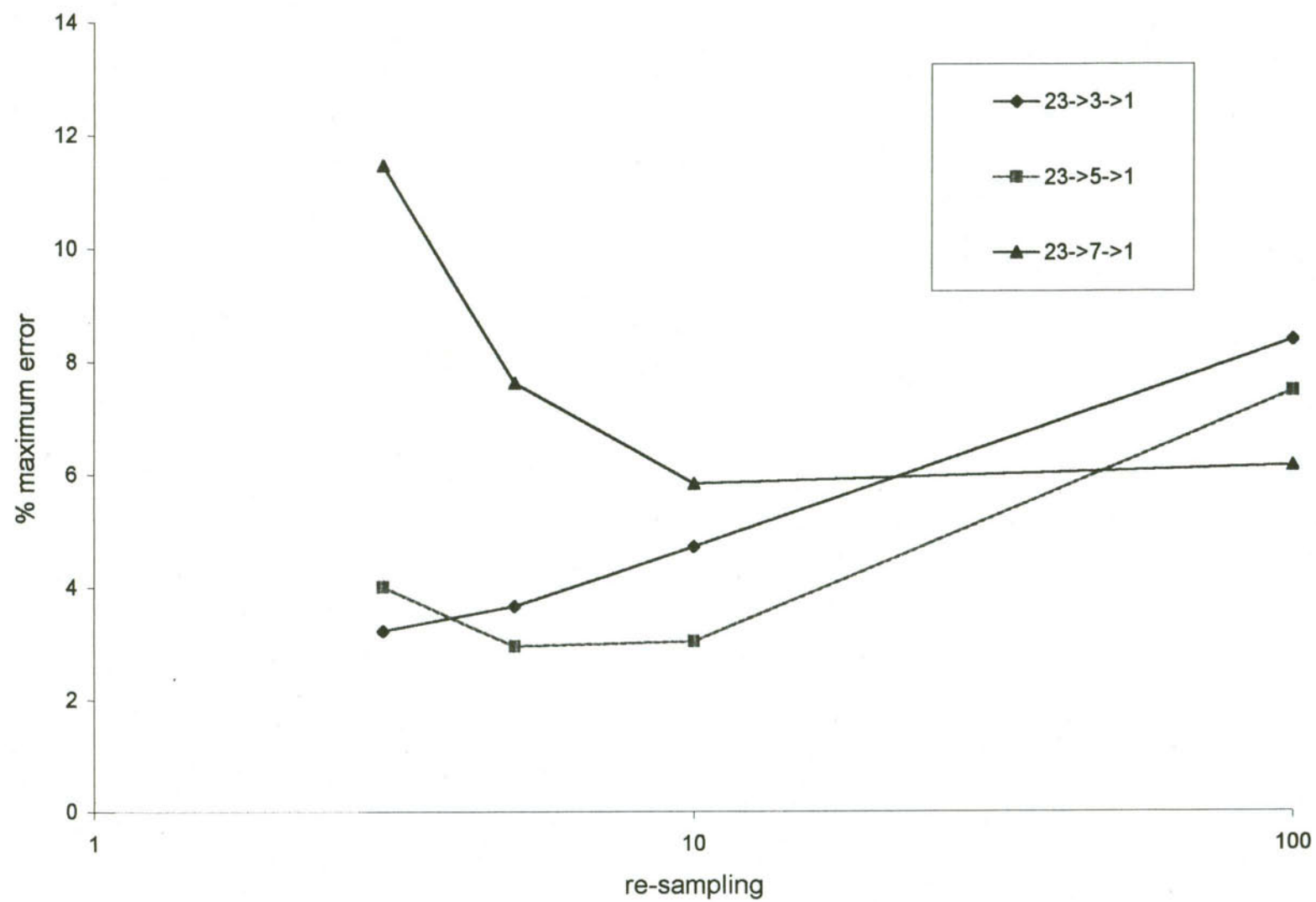
Error Summary (Rprop training)

	(resampling: 3)		(resampling: 5)		(resampling: 10)		(resampling: 100)	
	avg % err.	max err.	avg % err.	max err.	avg % err.	max err.	avg.% err.	max. err.
component 1								
23->3->1	0.900038	3.227698	1.389269	3.671652	1.526219	4.726506	1.875768	8.371139
23->5->1	0.979042	4.015583	0.77908	2.954765	0.911689	3.037317	1.582541	7.463859
23->7->1	2.898803	11.4653	2.058076	7.625224	1.580723	5.844707	1.491769	6.146308
component 2								
23->3->1	0.69595	3.079355	0.577029	2.511954	0.764916	4.355514	1.820672	9.936958
23->5->1	0.719026	3.956917	0.930194	5.444761	1.657814	8.053644	1.716655	9.052915
23->7->1	0.973021	3.521112	1.018134	4.263822	0.781997	3.350493	1.583868	7.784387
component 3								
23->3->1	5.657658	35.38598	4.173508	25.40824	2.98028	15.56264	3.383516	19.21859
23->5->1	5.632734	42.62712					3.358766	20.95414
23->7->1	4.127717	24.76233					3.321621	18.87217
23->9->1	3.286931	24.13552	3.678431	25.04255	2.670484	17.55561	3.570097	21.32842
23->11->1	3.690385	32.22926	2.68242	20.58423	4.113739	26.69946	2.845648	15.71374
23->13->1	5.590722	36.22888	5.32148	26.31691	4.153757	21.31105	3.31663	19.19737
component 4								
23->3->1	2.315092	5.099878					3.205614	10.54722
23->5->1	4.239189	11.86606					3.109023	9.736809
23->7->1	1.858755	6.753367	1.815879	6.394004	2.471255	7.466185	3.106925	9.245844
23->9->1	2.233451	4.868468	2.604417	6.797573	2.384755	6.805819	3.061737	9.545032
23->11->1	2.006427	6.665687	1.982923	6.790865	2.249135	8.072443	2.68949	8.31011
23->13->1	2.836911	10.43573					2.773543	8.383466
component 5								
23->3->1	4.081864	13.60674	9.210715	26.51507			7.787429	22.60976
23->5->1	7.897307	19.35731	7.004729	18.19607			8.405558	24.45915
23->7->1	5.434866	21.56318	6.189896	22.12471	5.823522	17.59742	9.172284	26.2025
23->9->1	3.847737	11.23313	4.786747	17.19227	4.738179	16.03931	8.234231	23.92159
23->11->1	5.534502	13.40319	4.392021	9.902669	7.013566	18.21838	7.603944	21.35958
23->13->1	13.16721	35.06879	9.438505	25.12813	8.286002	21.87045	7.945209	22.86464
component 6								
23->3->1	4.27421	14.90023					4.28085	19.5635
23->5->1	3.753625	12.93189	4.273102	21.05307	3.796639	18.58344	3.565317	17.21727
23->7->1	3.051094	11.63011	2.781902	9.230928	2.961454	10.37184	3.645163	14.54494
23->9->1	2.800074	9.482873	3.802316	16.69867	3.894172	15.56181	3.604075	15.24357
23->11->1	3.048866	17.75561					3.032735	12.06363
23->13->1							3.536607	14.65735

average error for compound 1



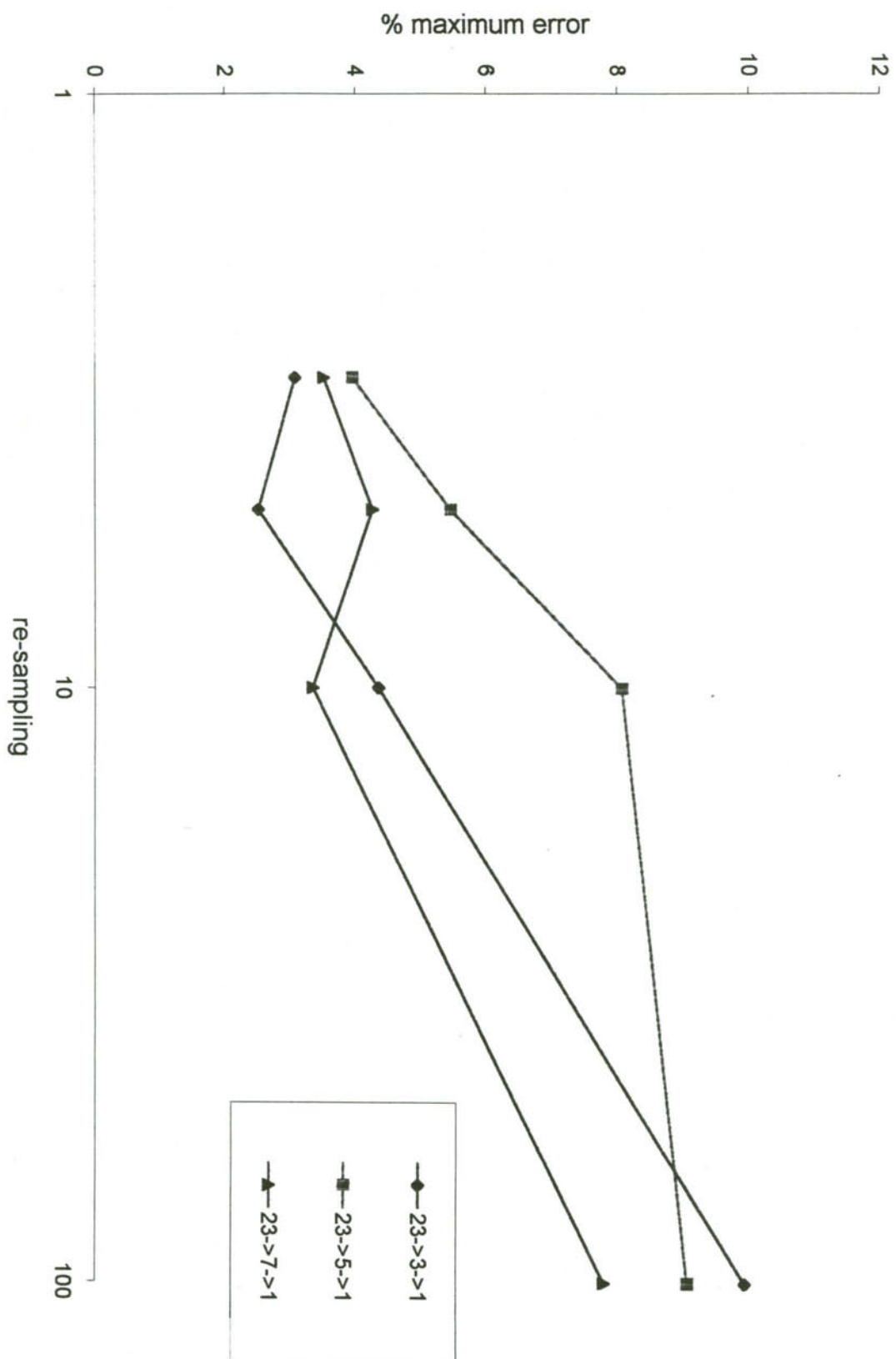
maximum error for compound 1



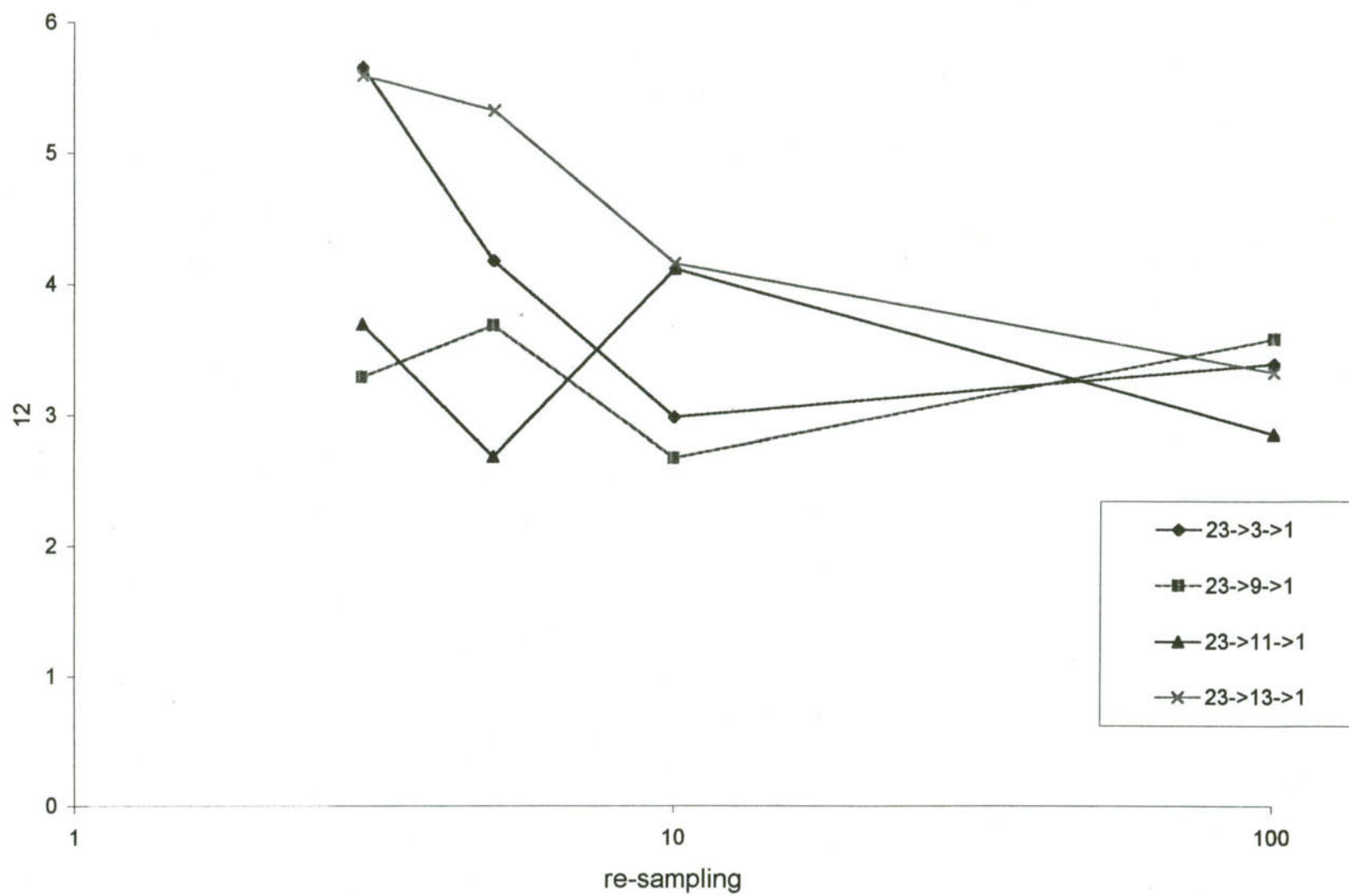
average error for compound 2



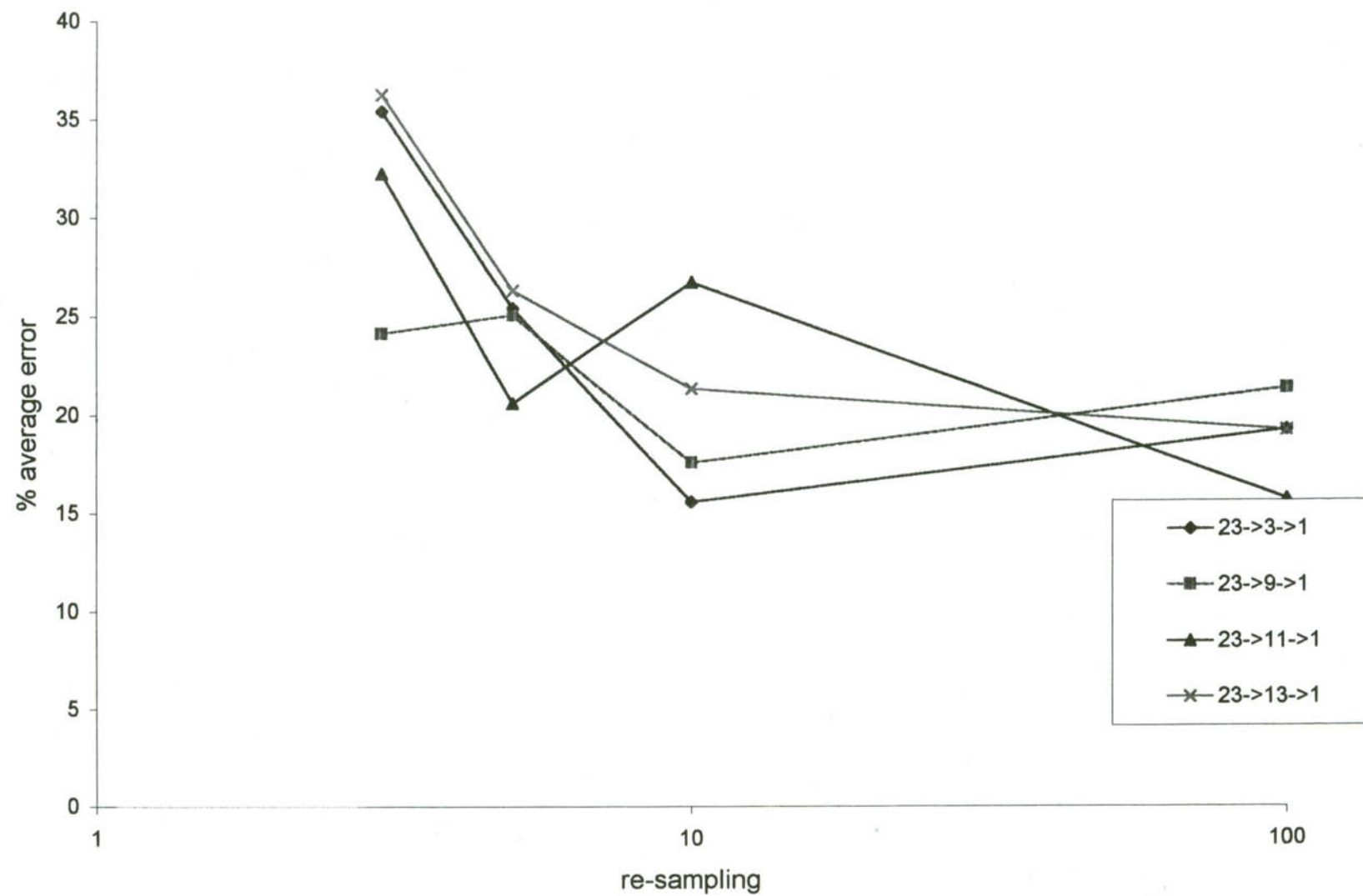
maximum error for compound 2



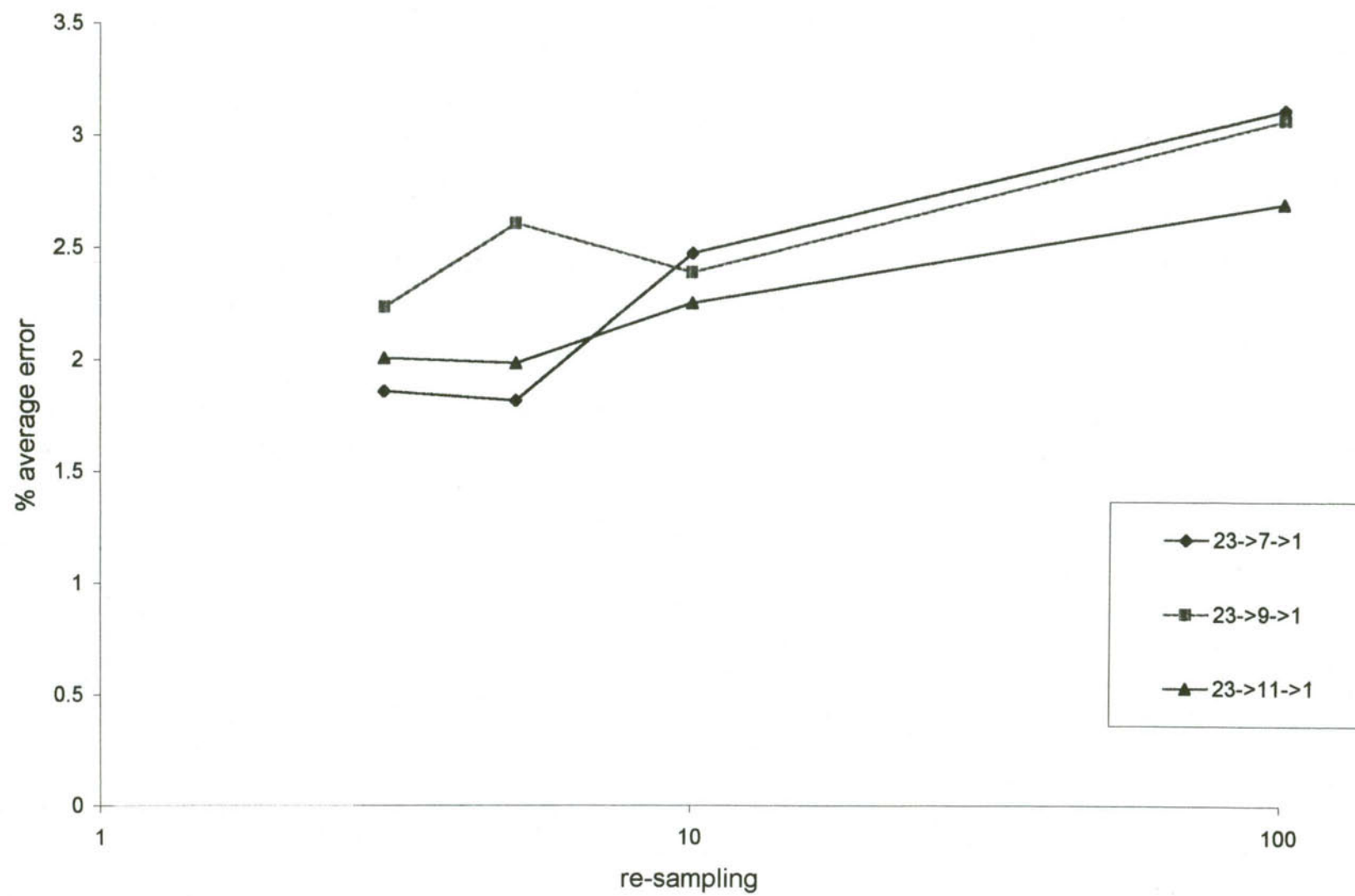
average error for compound 3



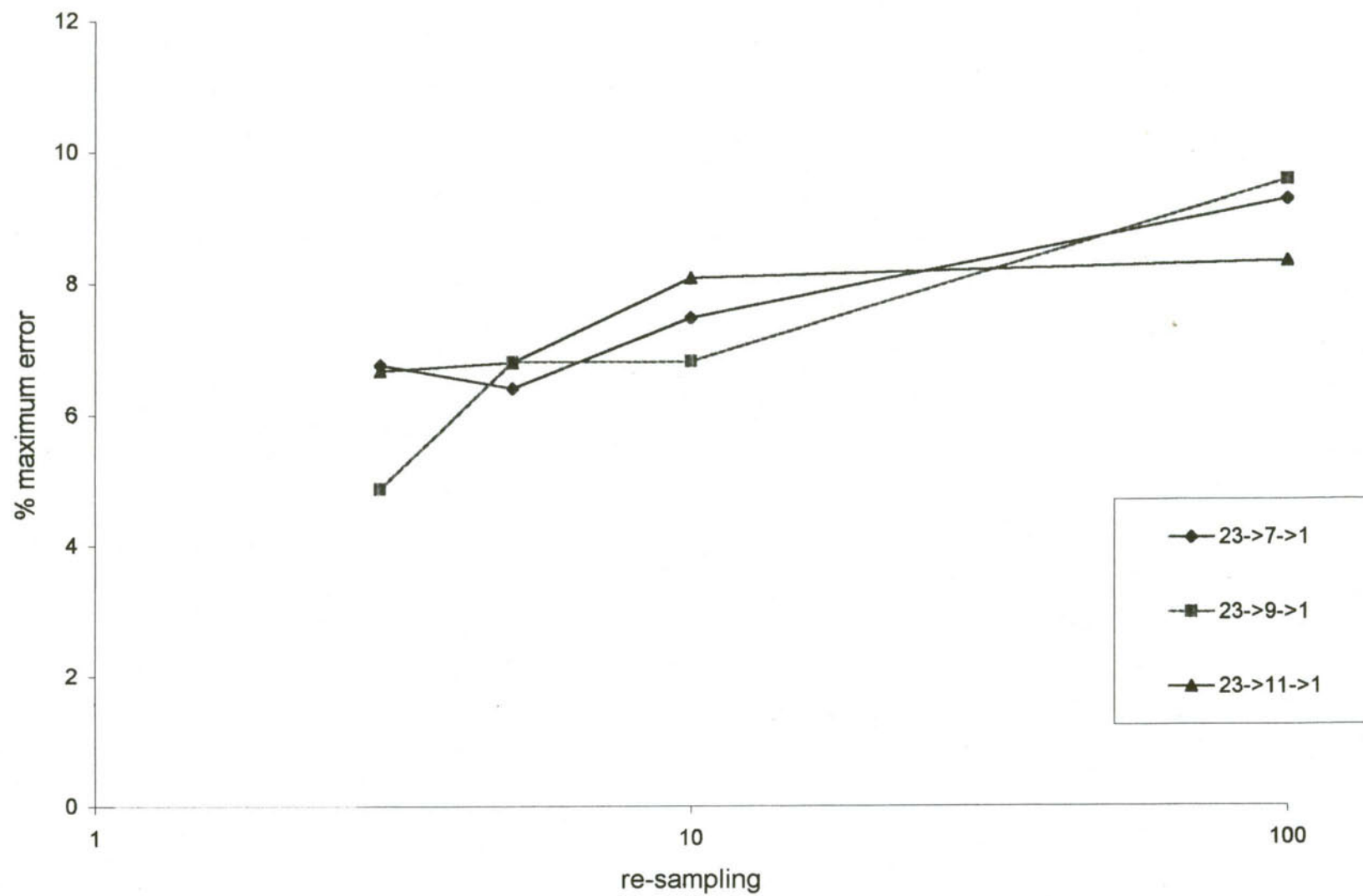
maximum error for compound 3



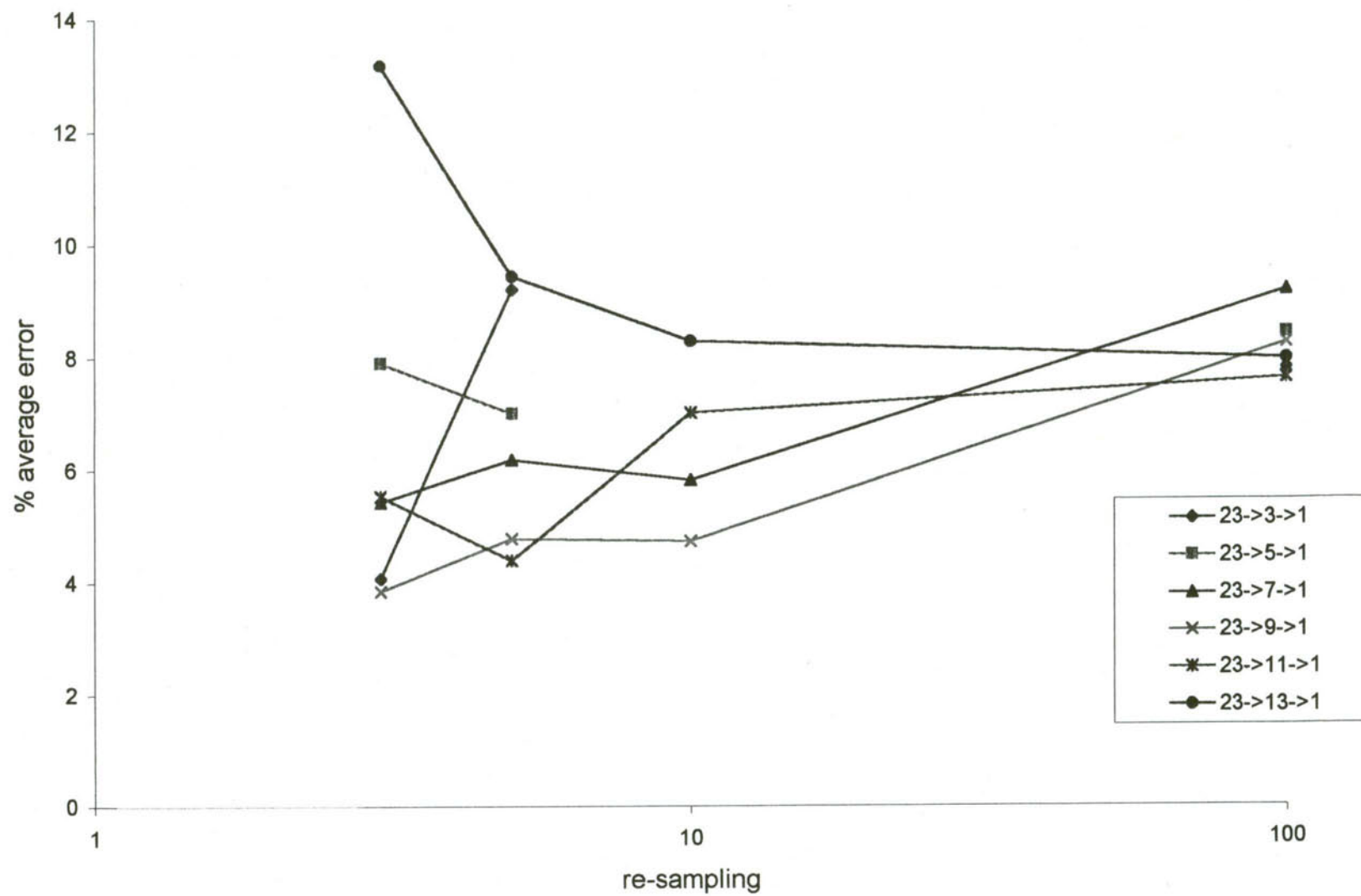
average error for compound 4



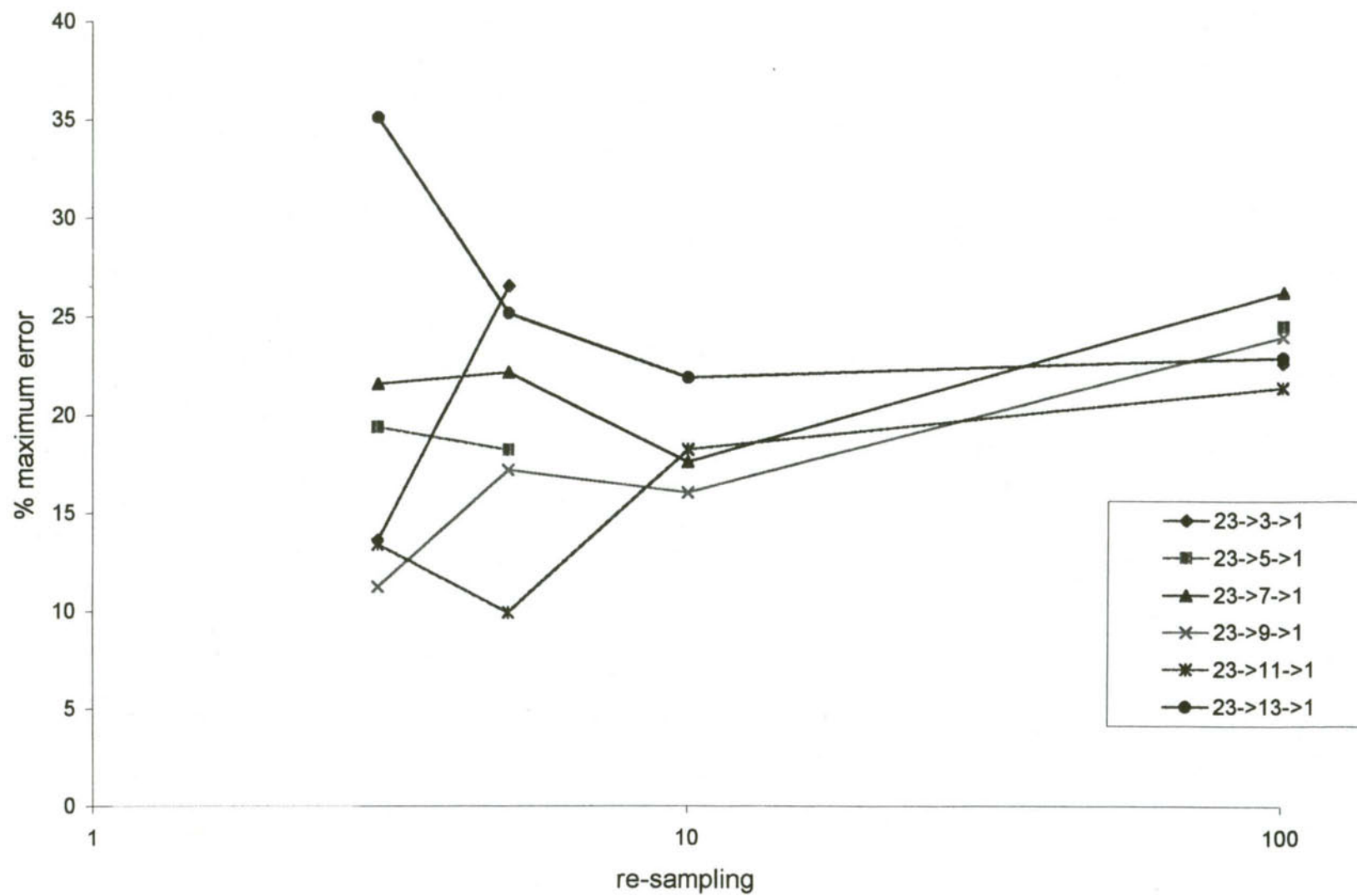
maximum error for compound 4



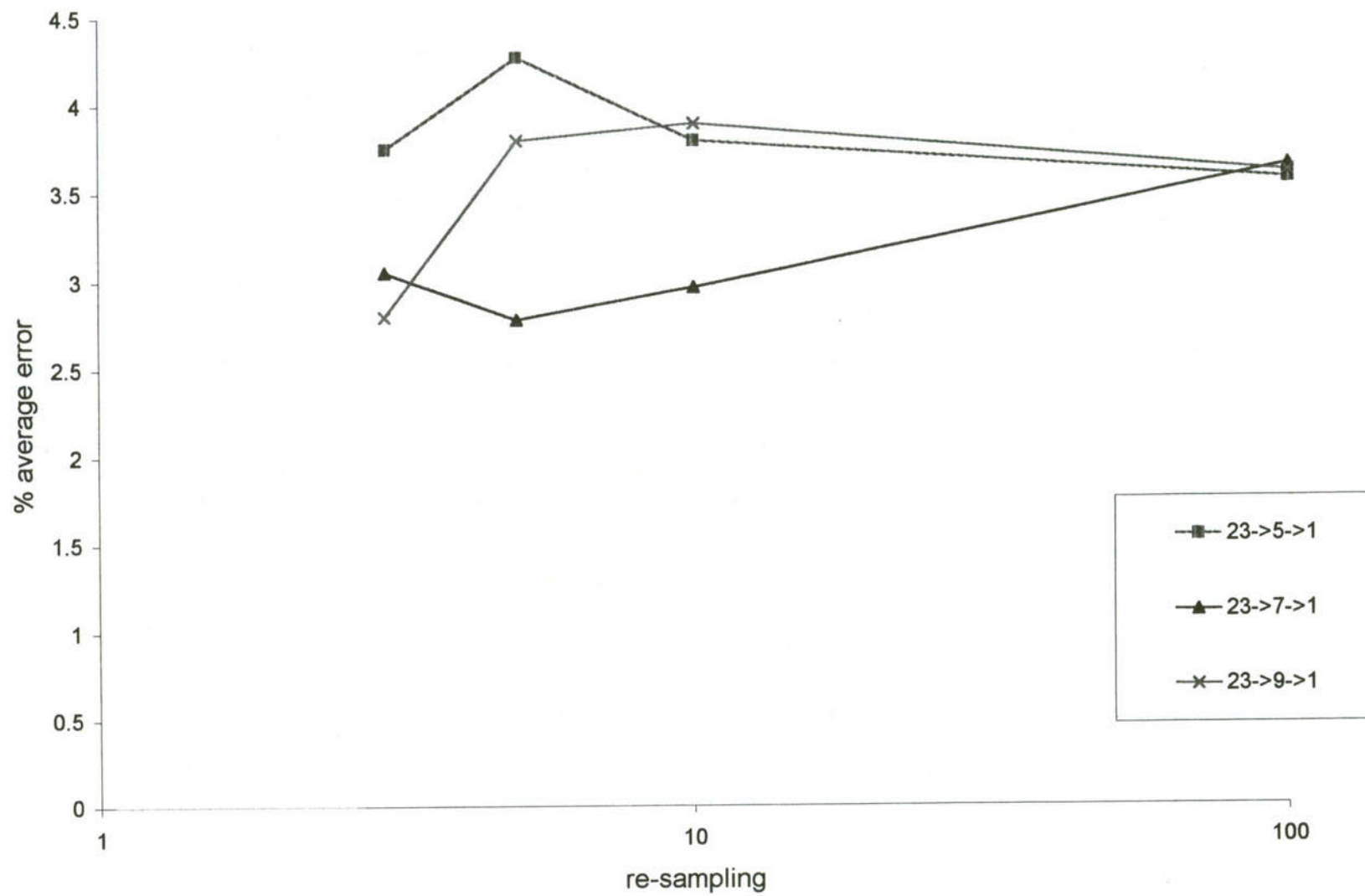
average error for compound 5



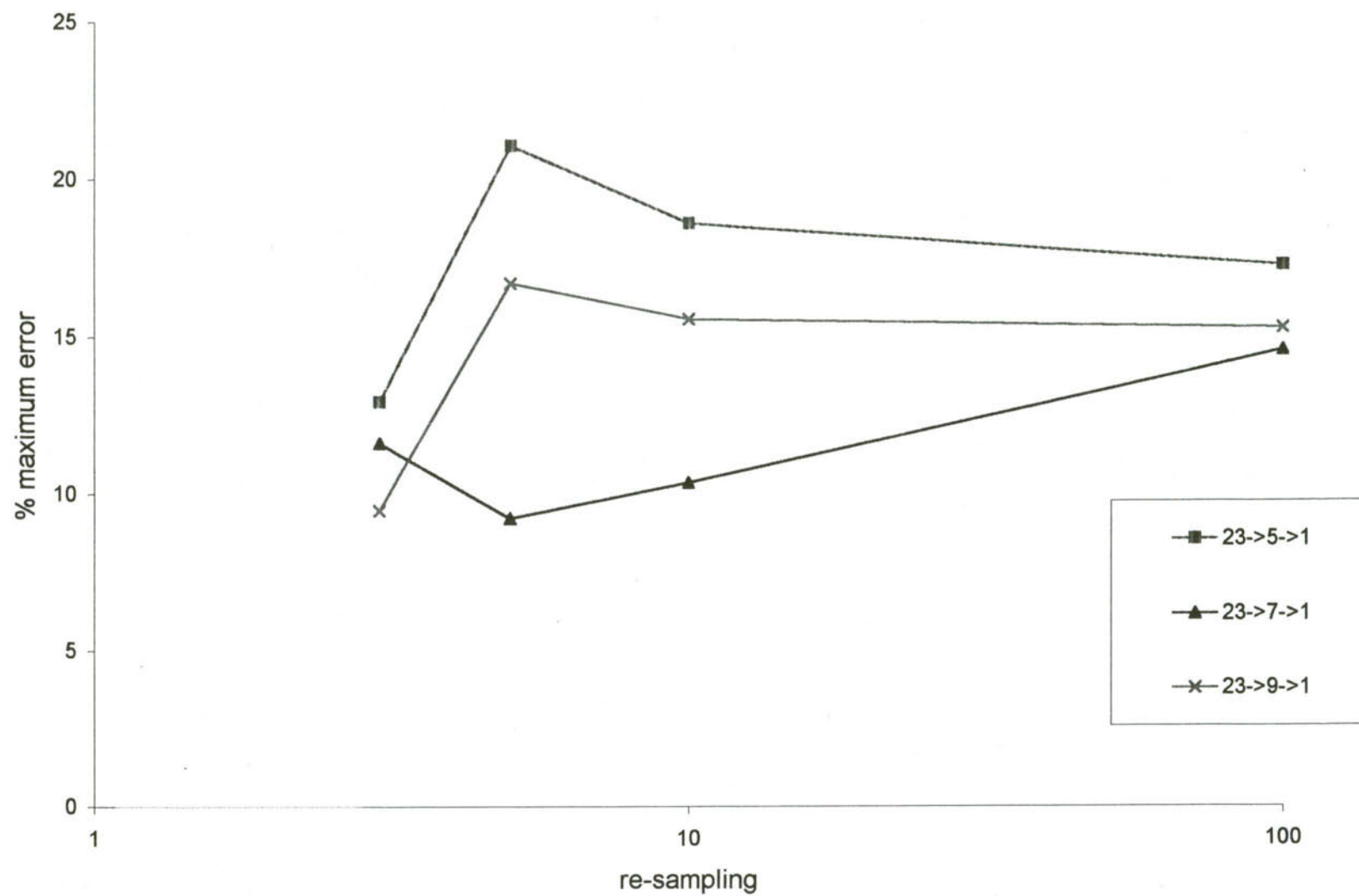
maximum error for compound 5



average error for compound 6



maximum error for compound 6



Relevant codes:

1) nlin.f90	 xliii
2) experr4.f90	 xlvii
3) straighten.f90	 xlix
4) syndata.f90	 l

```

!$=====
PROGRAM nlin
!
! title: nlin
! file: c:\wgn\nnf99\linear\f\nlin.f90
! description:
!   program reads in files containing the aver
! INPUT:
!   avgout.txt - file containing the concentrations for each mixture in
!               a 32x6 matrix that must be transposed when read
!   bmatrix.txt- file containing the mean spectrum (non-normalized) for each
!               component as a vector in a 23x6 matrix.
!               NOTE: negative values are set to ZERO
!   avgin.txt  - contains the spectra for each mixture in a 32x23 matrix that
!               is transposed when read
!
!   the files are read in and the linear combination of the basis
!   vectors is calculated
!                $S = \text{bmatrix} \times \text{alpha}$ 
!   the difference matrix is then calculated
!                $dS = S - S0$ 
!   the transpose of the matrix is written into dSmatrix.out for training
!   another NN. in order to consider the difference in flow rates
!   the vectors are normalized to consider the difference in linearity.
!
! MODIFICATION:
!   this program calculates the errors without replacing negative heights
!   by zero. the errors are again calculated as fractions.
!
! author: wgn                      date: 1/27/00
!
!
IMPLICIT NONE

REAL, DIMENSION(23,32) :: S, S0, dS
REAL, DIMENSION(6,32)  :: alpha
REAL, DIMENSION(23,6)  :: bmatrix
REAL, DIMENSION(32)     :: sum, avg, norm, std, max
REAL, DIMENSION(23)     :: compmn, compstd
REAL :: normmn, stdmn, avgmn, dev

INTEGER :: i, j, under

OPEN( unit = 1, file = 'avgout.txt' )      ! concentrations alpha' -> alpha
OPEN( unit = 2, file = 'bmatrix.txt' )     ! basis vectors (23x6)
OPEN( unit = 3, file = 'avgin.txt' )       ! observed mixture spectra
OPEN( unit = 4, file = 'nout.out' )        ! output
OPEN( unit = 5, file = 'ndSmatrix.out' )   ! file containing dS matrix

under = 0

DO j = 1, 32                                ! read in concentrations
  READ(1,105) alpha(:,j)                   ! alpha'(32x6) -> alpha(6x32)
105  format(6f13.7)
END DO

DO i = 1, 23                                ! read in basis matrix B(23x6)
  READ(2,*) bmatrix(i,1), bmatrix(i,2), bmatrix(i,3), &
    & bmatrix(i,4), bmatrix(i,5), bmatrix(i,6)

```

```

!      DO j = 1, 6
!      IF( bmatrix(i,j) < 0. )      THEN
!      IF( bmatrix(i,j) < -0.1 )      THEN
!      WRITE(4,207) i, j, alpha(i,j)
!207      format( 'underflow warning: alpha(',I5,', ',I5,') = '&
!      & ' ',f13.7,',' )
!      under = under + 1
!      END IF
!      bmatrix(i,j) = 0.
!      END IF
!      END DO
END DO

DO j = 1, 32
  READ(3,115) S0(:,j)      ! READ in original Spectra S0'->S0
115  format(23f13.7)
END DO

S = MATMUL( bmatrix, alpha )

max = 0

DO j = 1, 32      ! find maximum of each vector
  DO i = 1, 23
    IF( S(i,j) > max(j) )      THEN
      max(j) = S(i,j)
    END IF
  END DO
END DO

DO j = 1, 32      ! normalize each vector
  DO i = 1, 23
    S(i,j) = S(i,j) / max(j)
  END DO
END DO

max = 0

DO j = 1, 32      ! find maximum of each vector
  DO i = 1, 23
    IF( S0(i,j) > max(j) )      THEN
      max(j) = S0(i,j)
    END IF
  END DO
END DO

DO j = 1, 32      ! normalize each vector
  DO i = 1, 23
    S0(i,j) = S0(i,j) / max(j)
  END DO
END DO

dS = ( S - S0 )      ! '/ S0' commented out from nout.bak

DO j = 1, 32
  WRITE(5,215) dS(:,j)      ! output diff matrix for training
215  format(23f13.7,/)
END DO

```

```
sum = 0.  
norm = 0.  
std = 0.  
avg = 0.
```

! vector analysis

```
WRITE(4, 201)  
201 format('vector analysis')
```

```
DO j = 1, 32  
  DO i = 1, 23  
    sum(j) = sum(j) + dS(i,j)  
    norm(j) = norm(j) + dS(i,j) * dS(i,j)  
  END DO  
  
  avg(j) = sum(j) / 23  
  norm(j) = SQRT( norm(j) )  
  dev = 0  
  
  DO i = 1, 23  
    dev = dev + ( avg(j) - dS(i,j) ) * ( avg(j) - dS(i,j) )  
  END DO
```

```
std(j) = SQRT( (dev) / 22 )      ! WGN 1/26/00
```

```
normmn = normmn + norm(j)  
avgmn = avgmn + avg(j)  
stdmn = stdmn + std(j)
```

```
WRITE(4,210) j, norm(j), avg(j), std(j)  
210 format(' vector:',I5,' norm:',f13.7,' avg:',f13.7,' std:',f13.7)  
END DO
```

```
normmn = normmn / 32  
avgmn = avgmn / 32  
stdmn = stdmn / 32
```

```
WRITE(4,211)  
WRITE(4,212) normmn  
WRITE(4,213) avgmn  
WRITE(4,214) stdmn  
WRITE(4,*)  
WRITE(4,*)
```

```
211 format('For the complete set of vectors the mean &  
      & values for the following:')
```

```
212 format('vector norm:           ', f13.7)  
213 format('average componential deviation: ', f13.7)  
214 format('relative standard deviation:   ', f13.7)
```

! component analysis

```
WRITE(4,202)  
202 format('analysis by component:')
```

```
compmn = 0  
compstd = 0
```

```
DO i = 1, 23
```

```

      DO j = 1, 32
        compmn(i) = compmn(i) + dS(i,j)
      END DO
    END DO

    compmn = compmn / 23

    DO i = 1, 23
      DO j = 1, 32
        compstd(i) = compstd(i) + ( compmn(i) - dS(i,j) ) ** 2
      END DO
    END DO

    compstd = SQRT( (compstd) / 31 )      ! WGN 1/26/00

    DO i = 1, 23
      WRITE(4,222) i, compmn(i), compstd(i)
222    format('component:',I5,' mean: ',f13.7,' std: ',f13.7)
    END DO

    END PROGRAM

!$=====

```

```
! file: c:\wgn\NNf99\experr4.f90      author: wgn
! date: 4/7/00
! description:
!     program analyzes the standard deviations of the triplicate
!     measurements of each m/z peak for each mixture.  this data
!     is output in a matrix for further analysis
!     UPDATE: the deviations relative to the absolute magnitudes
!     are computed in this version
!     UPDATE: program computes avg. component of deviation vector
```

```
PROGRAM experr
```

```
    IMPLICIT NONE
```

```
    REAL, DIMENSION(:, :), ALLOCATABLE :: triple, diffmatrix
```

```
    REAL, DIMENSION(:), ALLOCATABLE :: avgvec, diffvec
```

```
    REAL :: dev, avg
```

```
    INTEGER :: i, j, k
```

```
    open( unit = 1, file = 'c:\wgn\NNf99\in' )
```

```
    open( unit = 2, file = 'c:\wgn\NNf99\experr4.out' )
```

```
!    open( unit = 2, file = 'c:\wgn\NNf99\experr.out' )
```

```
    open( unit = 3, file = 'c:\wgn\NNf99\avgdev.out' )
```

```
    ALLOCATE( triple(3,23) )
```

```
    ALLOCATE( diffmatrix(32,23) )
```

```
    ALLOCATE( avgvec(23), diffvec(23) )
```

```
    DO i = 1, 32
```

```
        DO j = 1, 3
```

```
            read(1,*) (triple(j,k), k=1,23)
```

```
        END DO
```

```
        avg = 0
```

```
        DO k = 1, 23
```

```
            avgvec(k) = triple(1,k) + triple(2,k) + triple(3,k)
```

```
            avgvec(k) = avgvec(k) / 3
```

```
            diffvec(k) = 0
```

```
            DO j = 1, 3
```

```
                diffvec(k) = ( triple(j,k) - avgvec(k) ) ** 2 + diffvec(k)
```

```
            END DO
```

```
            diffvec(k) = SQRT ( diffvec(k) / 2 )
```

```
            diffvec(k) = diffvec(k) / avgvec(k)
```

```
            diffmatrix(i,k) = diffvec(k)
```

```
            avg = avg + diffvec(k)
```

```
        END DO
```

```
        avg = avg / 23
```

```
        write(2,57) ( diffmatrix(i,k), k=1,23 )
```

```
        write(3,*) i, avg
```

```
!        write(2,55) (diffvec(k), k=1,23)
```

```
!        write(2,56) dev
```

```
!        print *, dev
```

```
    END DO
```

```
55 format(' ', 5f13.7)
```

```
56 format('magnitude:',f13.7)
```

```
57 format(' ', 23f13.7)
```

```
END PROGRAM
```

```

!$=====
PROGRAM straighten
!
! title: straighten
! file: c:\wgn\NNf99\linear\f\rep\straighten.f90
! description:
!     program reads in input vectors from file 'in' which
!     contains all 96 input vector organized in paragraphs
!     and outputs this data as row vectors for ease in input
!     into my linear analysis programs.
!           c:\wgn\NNf99\linear\f\rep\linrep.f90
!
! author: wgn                      date:2/9/00
!
!
      IMPLICIT NONE

      REAL, DIMENSION(96,23) :: data
      INTEGER :: i, j

      OPEN( unit = 1, file = 'in')
      OPEN( unit = 2, file = 'rowvec.out' )

      DO i = 1, 96
        DO j = 0, 3
          read(1,*) data(i,5*j+1), data(i,5*j+2), data(i,5*j+3), &
            & data(i,5*j+4), data(i,5*j+5)
! 57      format(5f13.7)
        END DO

        read(1,*) data(i,21), data(i,22), data(i,23)
! 58      format(3f13.7)
        read(1,*)
        END DO

        DO i = 1, 96
          write(2,105) data(i,:)
105      format(23f13.7)
        END DO

      END PROGRAM

!$=====

```

```

=====
PROGRAM syndata
=====
file: c:\wgn\NNf99\net\syndata\sdata.f90
description:
  INPUT:
    basis: basis spectra for pure components
    eps0: mean % deviation from linearity for each component
    std: standard deviation of eps0 for each component

  OUTPUT
    S: synthetic spectra for training NN
    alpha: fractional composition of each spectra

program reads in basis matrix (23x6) of spectra for pure
components. alpha matrix (6xN) of components is generated
by call to random number generator. the alpha matrix is
then scaled so that the sum of the components of each column
vector (each mixture) is 1. i.e. alpha is the fractional
composition.
the synthetic spectra, S, are then generated by matrix mult.
      S = basis x alpha
to each spectra random noise is added. the noise is added
in a gaussian distribution. the center of the distribution
is the mean fractional error, eps1, for the component. the
standard deviation of the gaussian distribution is the
input standard deviation of the differences from linearity
that is input as std.
random noise of a gaussian distribution is created by a
standard rejection method from a window of height 1 and
a width of 10 standard deviations, s. RANDOM_NUMBER generates
      nu0: 0 < nu0 < 1
which is scaled
      ups0 = 10s(nu0 - .5): -5s < ups0 < 5s
a second random number nul is generated by RANDOM_NUMBER
      nul: 0 < nul < 1
a gaussian function, fcn(x), of standard deviation, s, is
then evaluated at nu0. if nul < fcn(nu0) then nu0 is
accepted as the noise and is added to epsi
      eps = epsi + nul
the spectral component is then updated by
      S(i,j) = S(i,j) + eps * S(i,j)
the spectra is then once again normalized relative to the
max peak.

SUBROUTINE data reads in basis spectra for pure components
from file bmatrix.txt
SUBROUTINE normalize then calculates the normalization
constant for the gaussian distribution
SUBROUTINE output formats the resultant spectra S(:,j) as
rows in synin.out and the fractional composition
vectors alpha(:,j) as rows in synout.out

NOTE: the program utilizes the error function, erf(x) from
the microsoft library, MSIMSL

author: wgn date: 1/21/00
=====
USE MSIMSL

```

IMPLICIT NONE

```

INTEGER, PARAMETER :: N = 1000                ! no. examples
REAL, PARAMETER     :: PI = 3.14159

REAL, DIMENSION(:, :), ALLOCATABLE :: S
REAL, DIMENSION(:, :), ALLOCATABLE :: alpha
REAL, DIMENSION(23,6) :: basis
REAL, DIMENSION(23)   :: eps0, std, norm
REAL :: k, sigma, eps, epsi, ups, ups0, nu0, nul, max, sum

INTEGER :: i, j

OPEN( unit = 1, file = 'bmatrix.txt' )      ! basis (23,6)
OPEN( unit = 2, file = 'char.txt' )         ! eps0 (23); std (23)
OPEN( unit = 3, file = 'synin.out' )
OPEN( unit = 4, file = 'synout.out' )

ALLOCATE( S(23,N) )
ALLOCATE( alpha(6,N) )

CALL data( basis, eps0, std )                ! get input

CALL normalize( std, norm )                  ! eval. gaussian
                                           ! scaling

CALL RANDOM_NUMBER( alpha )

DO j = 1, N                                ! make fractional comp.
    sum = 0

    DO i = 1, 6
        sum = alpha(i,j) + sum
    END DO

    alpha(:,j) = alpha(:,j) / sum
END DO

S = MATMUL( basis, alpha )                  ! lin. comb. of basis vect.

DO i = 1, 23                                ! get gaussian err.
    sigma = std(i)
    k = norm(i)
    epsi = eps0(i)

    DO j = 1, N
        CALL RANDOM_NUMBER( nu0 )           ! 0 < nu0 < 1
        ups0 = 10 * sigma * (nu0 - .5)      ! -5s < nu0 < 5s
        CALL RANDOM_NUMBER( nul )           ! 0 < nul < 1
        IF( nul .le. fcn(ups0, sigma, k) ) THEN
            ups = ups0
            goto 15
        END IF
    END DO

15     eps = epsi + ups                      ! center around mean

    S(i,j) = ( eps + 1 ) * S(i,j)
END DO
END DO

DO j = 1, N                                ! rescale rel. to 1

```

```

max = 0

DO i = 1, 23
  IF( S(i,j) .ge. max) THEN
    max = S(i,j)
  END IF
END DO

S(:,j) = S(:,j) / max
END DO

CALL output( S, alpha, N )

CONTAINS

SUBROUTINE data( basis, eps0, std )
! get basis spectra, mean deviations, standard deviations
  REAL, INTENT(OUT), DIMENSION(:, :) :: basis
  REAL, INTENT(OUT), DIMENSION(:) :: eps0, std

  INTEGER :: i

  DO i = 1, 23
    READ(1,*) basis(i,1), basis(i,2), basis(i,3), &
      & basis(i,4), basis(i,5), basis(i,6)
  END DO

  READ(2,*)

  DO i = 1, 23
    READ(2,*) eps0(i), std(i)
  END DO
END SUBROUTINE

SUBROUTINE normalize( std, norm )
! evaluate normalization constant for gaussian distribution
  REAL, INTENT(IN), DIMENSION(:) :: std
  REAL, INTENT(OUT), DIMENSION(:) :: norm

  INTEGER :: i

  DO i = 1, 23
    norm(i) = SQRT(2*PI) * std(i) * erf( 5*std(i) )
  END DO
END SUBROUTINE

SUBROUTINE output( S, alpha, N )
! output spectra and composition matrices for training
  REAL, INTENT(IN), DIMENSION(:, :) :: S
  REAL, INTENT(IN), DIMENSION(:, :) :: alpha
  INTEGER, INTENT(IN) :: N

  DO i = 1, N
    WRITE(3,105) S(:,i)
    format(23f13.7)
  END DO

  DO i = 1, N
    WRITE(4,106) alpha(:,i)
    format(6f13.7)
  END DO

```

END DO
END SUBROUTINE

FUNCTION fcn(x, std, norm)
 REAL fcn
 REAL, INTENT(IN) :: x, std, norm

 fcn = (1/norm) * exp(-1*(x*x)/(2*std*std))
END FUNCTION

END PROGRAM

!\$=====

m/z peaks input to the neural networks

peak no.	m/z
1	13
2	14
3	15
4	16
5	26
6	27
7	28
8	29
9	30
10	37
11	38
12	39
13	40
14	41
15	42
16	43
17	44
18	50
19	51
20	53
21	55
22	56
23	58