Norwegian University of Life Sciences
Department of Mathematical Science and Technology

Master of Science 2014
60 credits

The Principal Component Transform (PCT)

Tensor-metamodel of biochemical synergistic (S) systems, a new approach for mathematical modelling of non-linear dynamical systems.

Sergio Haram Sarmiento

To Daniel Andrés & David Emanuel.

1 Acknowledgements

This work was carried out at the Norwegian University of Life Science from August 2013 to April 2014. Throughout each and every stage of this project there have been a great many people who have pushed me onwards, motivated me and guided me, and it is thanks to their support and guidance that we are reading this thesis today.
I am heartily grateful to my supervisor, Prof. Arkadi Ponossov, whose guidance and continual encouragement have been invaluable throughout the workings of this thesis. Prof Ponossov did not only lead me through the project; he truly inspired me and in sharing his knowledge both myself and my work have been immensely enriched.
I would also like to express my warmest thanks to Prof. Harald Martens, for his moral support, kindness and constructive ideas; this thesis being the result of one of these ideas. His constant enthusiasm have been both contagious and inspirational.
I would like to thanks to Prof Stig Omholt and Prof. Eberhard O. Voit, for their support and suggestions; this thesis bears the name proposed by Prof O. Voit, which is: the *principal component transform*.
My grateful thanks also goes to Rachel Davies, for her help in reading and correcting my terrible grammatical errors, which you will never find, thanks to Rachel.
Finally, I would like to express my gratitude to my family for their love and endless support, especially to my wife. Without her unwavering belief in me, I would never been able to achieve what I have achieved, and I am forever indebted to her.

2 Abstract

In this thesis, a new approach, the principal component transform (PCT)[1], is presented for mathematical modelling of synergistic biochemical systems (S-system) providing us with a simple and very accurate method for fitting these and other non-linear dynamical systems. The PCT is composed by two metamodels[2] one bi-linear metamodel and its improved version, a tensor-metamodel; both metamodels are based on the principal component analysis (PCA), which at the same time is based on the singular value decomposition (SVD). The tensor-metamodel is in addition based on mathematical attributes from tensor algebra, especially, the *Kronecker* product.
The PCT is inspired by the results of the PhD thesis of Julia Isaeva[3]. She and her co-authors presented how many non-linear models can be approximated by a single bi-linear metamodel. Isaeva's publications have provided valuable knowledge which has been applied in the implementation of the PCT as a tool which deals with multivariate power functions.
Referring to the PCT, the *loadings*, which are orthogonal, span a subspace, where, after projecting a simple time-series (given by the S-system), we are able to find a function which behaves in the same way as the non-linear dynamical system described by the S-system. The process of projecting the time-series vector to the span of the *loadings*-vectors is carried out by the linear regression method.
This thesis did explore the possibility of using the *scores*, which belong to the PCT as a *library*, meaning that after having computed the linear regression, we ought to have been able to find the same parameters in this *library*. After having tested this hypothesis in the scalar case we could not find any logical correspondence between the parameters from the *library* and those parameters found by means of the linear regression method. These fundings are in accordance with the results presented by the PhD candidate Valeriya Tafintseva[4], where she confirms that the S-system formalism has a sloppy structure in the scalar case, meaning that they have a neutral parameter-set. However, in higher dimensions, sloppiness seems to disappear, so that the data library can be constructed in an efficient way.
Regarding the bi-linear metamodel, it has been observed that the dimensionality of this method increases exponentially (n^3) and proportionally to the size of the S-system being studied. This leads, for big systems, to the formation of principal components belonging to a *null subspace*. This issue

[1]This terminology was suggested by Prof. Eberhard O. Voit; Prof. Voit is a Distinguished Professor in Biochemical Systems at the Georgia Technology Research Alliance.

[2]This concept and its applications are mostly due to by Prof. Harald Martens; Prof. Martens is a Professor at the Norwegian University of Science and Technology NTNU

[3]Julia Isaeva was a PhD student at the Norwegian University of Life Sciences from 2007-2011.

[4]Valeriya Tafintseva was a PhD student at the Norwegian University of Life Sciences from 2009-2013.

is solved in the tensor-metamodel, where the reduction of the dimensionality is given by the *Kronecker* product of the SVD.
Finally we can mention that the accuracy that each metamodel-phenome delivers is very good, meaning that more than 99% of the originally mathematical model is described by the PCT.

3 Sammendrag

I denne masteroppgaven presenterer jeg en ny tilnærming som kan brukes til matematisk modellering av synergistiske biokjemiske systemer (S-systemer), tilnærmingen kalles *the principal component transform* (PCT). PCT-modellen gir oss en enkel og svært nøyaktig metode for modellering av S-systemer og andre ikke-lineære systemer.
PCT modellen er satt sammen av to metamodeller: en bi-lineær metamodell og en forbedret versjon, kalt tensor-metamodell. Begge metamodellene baseres på prinsipal komponent analyse (PCA). Tensor-metamodellen er i tillegg basert på matematiske egenskaper fra tensor algebra, spesielt *Kronecker* produktet.
PCT-modellen er inspirert av doktorarbeidet til Julia Isaeva. Isaeva og hennes medforfattere presenterte hvordan ikke-lineære modeller kan tilnærmes ved en enkelt bi-lineær metamodell. Isaevas publikasjoner har gitt verdifull kunnskap som jeg har brukt for å bearbeide multivariate potensfunksjoner.
Med henvisning til PCA: *Scores* som er ortogonale, spenner ut et underrom der vi kan projisere en tidsrekke ved lineær regresjons metoden, tidsrekken er gitt av S-systemet. Dette gir en funksjon (en ODE-funksjon), som vil oppføre seg på samme måte som det opprinnelig S-systemet.
Denne oppgaven utforsker muligheten til å bruke *scores* som tilhører PCT-modellen som et bibliotek. Biblioteket kan brukes i forbindelse med parameterestimeringsproblemer. Min hypotese er at parametersettet funnet ved lineær regresjon burde finnes i *scores*-biblioteket. Etter å ha testet hypotesen for det skalare tilfellet fant jeg ikke logisk samsvar mellom parameterrommet og biblioteket. Dette er i samsvar med resultatene som presenteres av PhD kandidat Valeriya Tantseva. Tantseva bekreftet at S-systemer har en *sloppy* struktur i det skalare tilfellet. Det betyr at S-systemer har et nøytralt parameter-sett.
I høyere dimensjoner kan "sloppiness" likevel forsvinne slik at biblioteket kan konstrueres på en effektiv måte.
Nøyaktigheten som hver metamodell leverer er meget bra, noe som betyr at mer enn 99% av den opprinnelig matematiske modellen er beskrevet av PCT-modellen.

4 Introduction

March 2014 was a key date for the Norwegian Scientific Society, whereupon we celebrated the 150th anniversary of the only natural law found by Norwegian scientists: the *Law of Mass Action.* This law is a mathematical model which describes natural phenomenons in biochemistry given by solutions in dynamical equilibrium. Cato Maximilian Guldberg and Petter Waage proposed the *Law of Mass Action* in 1864, see Apendix A2.
This day we know that the attributes and properties of the *Law of Mass Action* can be applied not only to biochemistry, but to many other fields of science; such as physics (e.g. thermodynamic studies, semiconductor properties), physiology (e.g. extracellular physiological signals), pharmacology, psychology (e.g. psychological theories in neuro-psychology) and mathematics (e.g. dynamical systems, mathematical ecology, mathematical epidemiology).
The *Law of Mass Action* is also responsible for the relatively new mathematical theory called *power-law formalism*, which, together with their synergistic (S)-system representation of non-linear differential equations, describes satisfactorily the non-linearities of natural processes. This thesis is intended to be an attribute to this powerful and flexible natural *Law of Mass Action.* I will connect this natural law, which describes non-linear differential equations, to a method called metamodelling, which has been vastly used at the Center for Integrative Genetics (CIGENE) at the Norwegian University of Life Science, (section 5.1 describes what a metamodel is).
The main goal of the thesis is to find a new approach, which will be based on metamodelling, to non-linear systems appearing in biochemistry, in particular the S-system representation which goes back to the Gulberd and Waage model and its generalizations.
The mathematical theory which describes how the metamodel and the tensor-metamodel can be used as methods for modelling S-system is described in Sections: 8, 10 and 13.
Examples 2, 3, 4, and 5 present the fitting of (1x1), (2x2), (3x3) and (4x4) S-system to a bi-linear metamodel.
Section 13 presents the tensor-metamodel and Section 14 provides examples of (1x1), (2x2), (3x3) and (4x4) S-system being modelled by the tensor-metamodel.

Contents

List of Figures

5 Background

5.1 Model vs. metamodel

A model in mathematics is a simplified description of a system, and this description is made using mathematical language and concepts. I believe that a mathematical model has 3 main goals:

i) To help us to understand the system.
ii) To help us to explain and describe the system and the components which are involved in the system.
iii) To help us to make predictions about the behaviour of the system, and the effect each component has.

It is well known that mathematical models of complex systems are normally very difficult to analyze and time consuming to work with. On the other hand, a metamodel is a much simpler tool to work with. A metamodel is a model of the original model. A metamodel provides, just like a model, a description of the system. The advantage of working with metamodels is that they considerably reduce the complexity of the model and also reduce the dimensionality of the system, without loosing any important information. We can see in Figure 1 and Figure 2 a graphical representation of an *aposteriori* and an *apriori* model with its corresponding metamodel.

A model is known as *aposteriori* when the real data comes before the mode. The metamodel comes at the end.

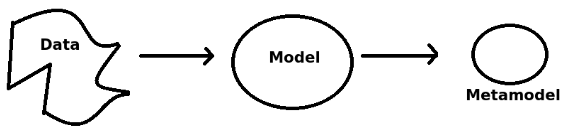

Figure 1: This figure has been extracted from [1].

The model is known as *a priori* when the data comes after the model. The metamodel comes at the end.

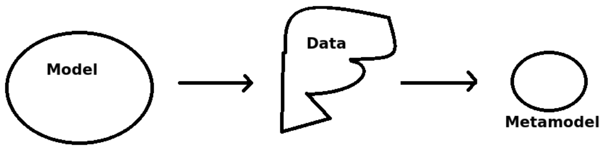

Figure 2: This figure has been extracted from [1].

Despite the fact that a metamodel is a simplification of the model, it does not lose any important information regarding the complex system. In other words, a metamodel will not fail to fulfil $i), ii)$ and $iii)$.
The metamodel will provide an insight, which in some cases is as deep as the model itself, of the complex system being analyzed.
In this thesis I will use the principal component analysis (PCA) as a metamodel. The PCA will be computed by means of the singular value decomposition.

5.2 The singular value decomposition (SVD)

According to [4], there are two matrices U and V which are orthogonal and a diagonal matrix Σ enabling the matrix $X_{(I\text{x}J)}$ to be factorized as follows:

$$X_{(I\text{x}J)} = U_{(I\text{x}I)}\Sigma_{(I\text{x}J)}V^t_{(J\text{x}J)} \tag{1}$$

where V^t is V transposed.
The matrix Σ contains the singular values of X on its diagonal. The left singular vectors U are joined together with Σ in order to obtain the *scores-matrix* T and the matrix V, also called right singular vectors which produce the *loadings-matrix* P:

$$X = (U\Sigma)V^t = (T)P^t + e \tag{2}$$

A more extensive definition of the singular value decomposition is provided in appendix A1.

5.3 The principal component analysis (PCA)

A principal component (PC) is a special type of variable which can not be measured directly. Instead, it can be computed as a linear combination of a set of input variables (e.g. a data set). These kinds of variables are called *latent variables*, see [2]. The author in [2] underlines that the principal component analysis (PCA) allows us to extract relevant information from big and sometimes confusing data setts with comparatively little effort. The PCA reduces the high dimensional data sett to a lower dimension giving few principal components that underlie the dynamics of the complex system. Authors in [3] acknowledge that the PCA is concerned with explaining the structure of the variance-covariance of a set of variables. The result of the analysis of these *latent variables* has shown, and also been considered by [1], that the principal components lie in the direction of the maximal variation of the data and they describe this variation in a descending order, thus: the first PC is found along the direction of the largest variation, the second PC in the direction of the second largest variation but orthogonal to the first, a pattern which is repeated for the third PC, the fourth PC and so on.

$$\text{Let } A = \text{optimal number of PCs}$$

When the remaining variation of the data in question is small, it is considered that A has been found and the information which is given by $A+1$ PCs is so small that it can be considered as noise (e).
In [1] we can read more about how the PCs reflect the largest eigenvalues of the co-variances, therefore the first few PCs can give an adequate description of the whole data set, these PCs form a new orthogonal basis of the variable space where the *scores* (T) and *loadings* (P) are projected onto it. The *scores* represent the relationship between the PCs and the *samples* (data rows) while the *loadings* represent the relationship between the PCs and the *variables* (data columns).
Assume we are given some random data in the matrix $X_{(I\text{x}J)}$, this data can be expressed in terms of the *loadings* and *scores* as shown below:

$$X_{(I\text{x}J)} = T_{(I\text{x}K)} P^t_{(K\text{x}J)} + e_{(I\text{x}J)} \tag{3}$$

Here are X, T, P and e matrices with known dimensions, and where P^t is P transposed.
(3) shows that $X_{(I\text{x}J)}$ is represented linearly by T and P, for this reason the PCA is referred to as the *bi-linear model.*
Next we have a graphical interpretation of (3).

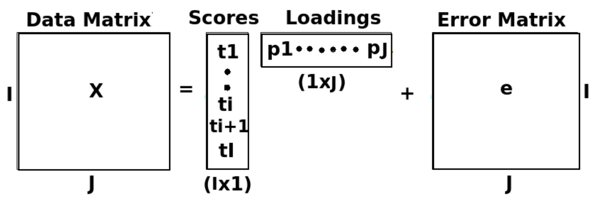

Figure 3: Illustrates how the structure of the PCA will be.

The full bi-linear model

The bi-linear model of the matrix $X_{(IxJ)}$, together with the optimal number of PCs (A) can be written in vector form as shown below:

$$X_{(IxJ)} = \overline{X} + \mathbf{t}_1\mathbf{p}_1^t + ... + \mathbf{t}_A\mathbf{p}_A^t + e_A \tag{4}$$

where $\mathbf{p}^t$ is $\mathbf{p}$ transposed.
The matrix $\overline{X}$ in (4) represents the main centred-valued matrix, the vectors $\mathbf{t}_1$,...$\mathbf{t}_A$ are the first A *scores* vectors, $\mathbf{p}_1$,...$\mathbf{p}_A$ are the first A *loadings* vectors, and e is the error.
Centring X before computing the PCA is a normal procedure in statistical analysis, but since this is a mathematical thesis, we are only interested in the mathematical behaviour of the bi-linear model. For this reason, we do not need to center X.
Just before we introduce the PCA as a metamodel for S-system, we give a formal introduction of S-system and the theory of the *power-law formalism.*

6 The power-law formalism and S-systems

6.1 Historical introduction

The *power-law formalism* and in particular the S-system representation within this non-linear formalism can trace their origins back to 150 years ago. It all started with two scientists named Peter Waage and Cato Maximilian Guldberg.
Peter Waage (June 29 1833 - January 13 1900), was a significant Norwegian chemist and professor at the Royal Frederick University. Waage, along with his brother in law Cato Maximilian Guldberg (11 August 1836 - 14 January 1902), developed the *Law of Mass Action* between 1864 and 1879. The *Law of Mass Action* in chemistry is a mathematical model that explains and predicts behaviours of solutions in dynamic equilibrium. Guldberg og Waage published three articles in total, the first one came in 1864, and was a paper in Norwegian called *Studier over Affiniteten*, (see appendix A2). It went largely unnoticed, as did the second paper in 1867, which was written in French. Almost fifteen years after the first paper Guldberg and Waage had published, 1877 saw a Dutch chemist called Jacobus Henricus van 't Hoff achieving similar results as those produced by Guldberd and Waage. Hoff's work was produced independently from that of Guldberg & Waage's and Hoff's had started to get recognition for these results. For this reason, the Norwegian scientists decided to write their last paper in 1879, in order to get credit for their work, an achievement which they they did manage to secure.
It is thanks to the scientific achievements of Guldberg & Waage in *Studier*

over Affiniteten (1864) that we have what we now call the *power-law formalism*, which is a sophisticated theory that studies dynamic equilibrium.
A representation of this formalism is given in systems of equations called S-systems. In [5] we can read that the S refers to *synergism* and *saturation* of the investigated system. The authors in [6] stress the fact that the *power-law formalism* is a theoretical framework for modelling and analysis of complex systems represented by differential S-system equations. Among the possibilities of this formalism, we would do well to mention:

* *Steady-state characterization of complex systems.*
* *Parameter estimation from experimental measurements.*
* *Classification of non-linear functions using recasting techniques.*
* *Development of efficient numerical algorithms for numerical simulations.*

6.2 The need of the power-law formalism and S-systems

In the analysis of complex technological systems or biological systems, methods based on linear mathematics very often fail to capture the essential non-linear characteristics from these processes. Concerning living systems and natural processes, non-linear responses such as synergism, saturation and oscillations are the rule rather than the exception. For instance, finding analytical solutions of non-linear differential equations is an art limited to a few special cases that it is insufficient for general non-linear system analysis. On the other hand, the power-law formalism with S-system differential equations shows potential for providing systematic methods for analysis of non-linear systems, see [6].

6.3 A formal mathematical description of S-systems

The author in [5] provides a very descriptive definition of the mathematical structure of S-systems. I will quote this definition with some terminology changed:

An S-system in biology represents the relative change in flux of a substance in a specific process. This relative change in flux is represented by the function X_i, where the variables have the same name and which implies that $\dot{X}_i$ denotes the derivative of X_i with respect to time. $\dot{X}_i$ is at the same time composed of two positive-valued and differentiable functions of X. One of these functions can be expressed as: $V^+(X_i)$ representing the in-flux and the second function: $V^-(X_i)$ representing the out-flux.
This gives:

$$\dot{X}_i = V_i^+(X_i) - V_i^-(X_i) \tag{5}$$

where $i = 1, 2, ..., n$.

From (5) we can understand that if the system is composed of $X_1, X_2, ..., X_n$ variables, the function that describes the production is given by $V_i^+(X_1, X_2, ..., X_n)$ and the function that describes the degradation is given by $V_i^-(X_1, X_2, ..., X_n)$, giving the following result:

$$\dot{X}_i = V_i^+(X_1, X_2, ..., X_n) - V_i^-(X_1, X_2, ..., X_n) \tag{6}$$

Considering now $\dot{X}_i$, where $i = 1, 2.., n$, this implies that there is only one differential equation for each dependent variable in the system being considered. Assume now that we have m independent variables and n dependent variables, this will give us n differential equations in the form of (6).
We let V_i^+ and V_i^- be defined now as follows:

$V_i^+(X_1, ..., X_n, X_{n+1}, ..., X_{n+m}) = \alpha_i X_1^{g_{i1}} X_2^{g_{i2}} ... X_n^{g_{in}} X_{n+1}^{g_{i,n+1}} ... X_{n+m}^{g_{i,n+m}}$

and

$V_i^-(X_1, ..., X_n, X_{n+1}, ..., X_{n+m}) = \beta_i X_1^{h_{i1}} X_2^{h_{i2}} ... X_n^{h_{in}} X_{n+1}^{h_{i,n+1}} ... X_{n+m}^{h_{i,n+m}}$

where $\alpha_i, \beta_i, g_{i,n+m}, h_{i,n+m}$ are parameters.
Then:

$V_i^+(X_1, ..., X_n, X_{n+1}, ..., X_{n+m}) = \alpha_i \prod_{j=1}^{n+m} X_j^{g_{i,j}}$

and

$V_i^-(X_1, ..., X_n, X_{n+1}, ..., X_{n+m}) = \beta_i \prod_{j=1}^{n+m} X_j^{h_{i,j}}$

This generalization allows us to re-write (6) in the following way:

$$\dot{X}_i = \alpha_i \prod_{j=1}^{n+m} X_j^{g_{ij}} - \beta \prod_{j=1}^{n+m} X_j^{h_{ij}} \tag{7}$$

with $X_i(0) = X_{i0}$ and for $i = 1, 2, ..., (n + m)$
The equation in (7) gives a canonical representation of non-linear ordinary differential equations. The coefficients $\alpha_i, \beta_i \in \mathbb{R}$ are called *rate constants* and the coefficients $g_{ij}, h_{ij} \in \mathbb{R}$ are referred to as *kinetic orders.*

6.4 Modelling with S-systems

From experimental trials conducted and presented by the author in [5], we know that near to steady-state solutions and relative changes in metabolites produces proportional changes in flux. S-system's representations of biochemical systems responds in the same way. For this reason we can consider S-systems as a suitable tool for modelling biochemical systems.

At the same time, we know from the theory presented in [6] that S-systems, thanks to its structural homogeneity, means that all steps required for analysis of complex biological systems are straightforward. We could mention: system representation, steady-state solutions, stability analysis and sensitivity analysis: these methods are efficient for evaluating the dynamic responses of the system, providing at the same time powerful tools for understanding the complex non-linear systems.
In the next chapter we will provide the theory that will allow us to compress these complex non-linear systems represented by S-systems. We will show as well that by compressing S-systems we don't lose any important information of the non-linear system.

7 Compressing power functions with PCA

7.1 Theoretical background

The theoretical background of this section is based on the research and work of Julia Isaeva. Julia took his Philosophiae Doctor Thesis at the Norwegian University of Life Sciences in 2011. I will use more specifically the results presented in her fourth paper *"The modelome of line curvature: Many non-linear models approximated by a single bi-linear model with verbal profiling"*, whilst at the same time I hope to take these results a little further.
Consider a matrix X of dimensions (I x J). Every entry in $X_{(IxJ)}$ will be given by a power function, this follows from the way X is defined, which is:

$$x_j^{\omega_i} : X_{(IxJ)} \longrightarrow \mathbb{R} \tag{8}$$

where $i = 1, .., I$ and $j = 1, .., J$.
This means that we can decompose $x_j^{\omega_i}$ by arbitrarily choosing values for J and I. If, for instance, we let: $j \in [1, ..., J]$ which is a close domain with J points, and $i \in [-1, ..., I]$ which is also a close domain with I points, we obtain the matrix $X \in \mathbb{R}^{I\text{x},J}$:

$$x_j^{\omega_i} : X = \begin{bmatrix} x_1^{\omega_1} & x_2^{\omega_1} & ... & x_J^{\omega_1} \\ x_1^{\omega_2} & x_2^{\omega_2} & ... & x_J^{\omega_2} \\ .. & .. & ... & .. \\ x_1^{\omega_I} & x_2^{\omega_I} & ... & x_J^{\omega_I} \end{bmatrix} \in \mathbb{R}^{(I\text{x}J)} \tag{9}$$

We can see now how the matrix $X_{(IxJ)}$ represents a collection of power functions, or, we could say that $X_{(IxJ)}$ is a multivariate power function matrix. Through this thesis I am going to work in detail with functions of the form $x_j^{\omega_i}$. This is because the structure of these functions is similar to the structure we find in the S-system representation. Since S-systems are related to biochemical systems, we need to define some constraints for the domain of $x_j^{\omega_i}$, this is needed in order to get valid results within this branch of chemistry.

ω_i:
ω_i will have a default close domain, $\omega_i \in [-1, 2]$. This is because [5] ensures that *kinetic order* reactions happen to occur inside this interval, namely $[-1, 2]$.

x_j:
The only constraint we have for the domain of x_j is that it cannot include the 0 value. This is because we could risk operating with singularities when dividing by zero.
According to [1] we know that such kinds of matrices ($X_{(IxJ)}$) can be compressed by a metamodel, the bi-linear model. If we recall equation (4), we are reminded of the fact that we can express $X_{(IxJ)}$ as a linear combination of the mean-centred valued matrix $\overline{X}$ and the first three principal components multiply with the first three *scores-vectors*:

$$X_{(IxJ)} = \overline{X} + \mathbf{t}_1\mathbf{p}_1^t + ... + \mathbf{t}_A\mathbf{p}_A^t + e_A \tag{10}$$

where $\mathbf{p}^t$ is $\mathbf{p}$ transposed.
Consequently, if we were to multiply $X_{(IxJ)}$ with a real-valued constant α, we would have:

$$\alpha X_{(IxJ)} = \alpha\overline{X}_{(IxJ)} + \alpha\mathbf{t}_1\mathbf{p}_1^t + ... + \alpha\mathbf{t}_A\mathbf{p}_A^t + \alpha e_A \tag{11}$$

Note that:

$$\mathbf{t} \in \mathbb{R}^I \text{ and } \mathbf{p} \in \mathbb{R}^J$$

the *scores* ($\mathbf{t}$) will be related to the rows of $X_{(IxJ)}$ while the *loadings* ($\mathbf{p}$) will be related to the columns of $X_{(IxJ)}$.

7.2 Further results: The scores-library

To reiterate one more time, we are only interested in the mathematical behaviour and structure of the bi-linear model, and for this reason we do not need to center the data in $X_{(IxJ)}$ before computing the PCA. The consequence of leaving $X_{(IxJ)}$ not-centred is that we need to work with 3 PCs instead of 2. We now have the following metamodel:

$$x_j^{\omega_i} : X_{(IxJ)} = \mathbf{t}_1\mathbf{p}_1^t + \mathbf{t}_2\mathbf{p}_2^t + \mathbf{t}_3\mathbf{p}_3^t + e_4 \tag{12}$$

where $\mathbf{p}^t$ is $\mathbf{p}$ transposed.
Another important result we are going to further explore is that the *scores* $\mathbf{t}_i$ ($i = 1, 2, 3$) from (12), can be brought together to form a *library*, which can be used in parameter estimation problems. The *scores-library* is introduced in the following section.

The scores-library

Let L refer to a library formed by the *scores* $\mathbf{t}_1$, $\mathbf{t}_2$ and $\mathbf{t}_3$, which can be found in (12).

$$L = [\mathbf{t}_1\ \mathbf{t}_2\ \mathbf{t}_3] \tag{13}$$

Before we proceed, please recall that the dimension of $\mathbf{t}_i$ $(i = 1, 2, 3)$, is directly related to the rows of $X_{(IxJ)}$, and where the rows of $X_{(IxJ)}$ are given by ω_i from the power function representation $x_j^{\omega_i}$, see (9).
Assume now that we are interested in a specific value called ω_s, which is inside ω_i and where $\omega_i \in [-1, 2]$. Assume this value $\omega_s = 3/2$, which gives the following function:

$$f(x) = x^{(3/2)}$$

Expressed graphically, we have:

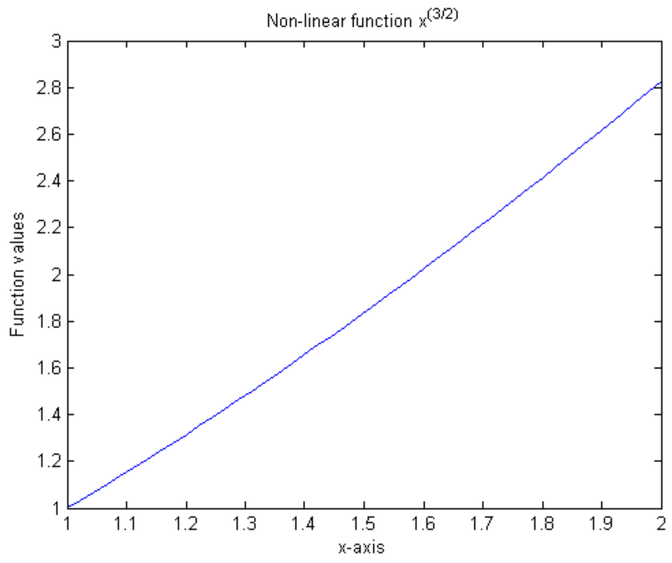

Figure 4

At the same time we construct a matrix $X \in \mathbb{R}^{(600\text{x}600)}$ where we let $x_j \in [0.1,\ 2]$ and $\omega_i \in [-1,\ 2]$. We can now compute the PCA to $X_{(600\text{x}600)}$ in order to get the *scores* and *loadings*.

$$x_j^{\omega_i} : X_{(600x600)} \ \rightarrow\ PCA \rightarrow \sum_{i=1}^{3} \mathbf{t}_i \mathbf{p}_i^t \approx X_{(600x600)}$$

where $\mathbf{p}^t$ is $\mathbf{p}$ transposed.
Now, since we know that $\omega_s \in \omega_i$ we can find the index position in ω_i where we have the value of 3/2 or its best approximation.

$$\omega_i \in [-1, \omega_2, \omega_3, ..., \underbrace{1.5}_{position\ 501}, ..., 2] \tag{14}$$

Once we have the index position of ω_s we can begin working with L. In this case we have the following library:

$$L = [\mathbf{t}_1 \ \mathbf{t}_2 \ \mathbf{t}_3] \ \in \mathbb{R}^{(600\text{x}3)}$$

Now, in order to find a metamodel that approximates our function $f(x) = x^{3/2}$ we need to find the values of L which correspond to the index position 501. This step is illustrated in the next Figure.

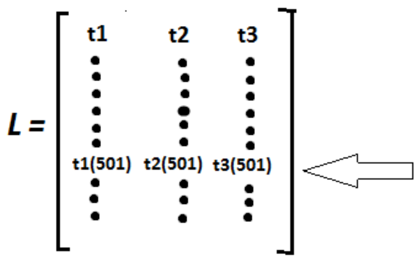

Figure 5: Representation of the *scores-library*

We use *matlab* to pick up these values and we get the following metamodel:

$$f(x) \approx \underbrace{L(501,1)\mathbf{p}_1(x) + L(501,2)\mathbf{p}_2(x) + L(501,3)\mathbf{p}_3(x)}_{\widehat{f}(x)} \qquad (15)$$

where:

$$t_1 = L(501,1) = -8.83^5$$
$$t_2 = L(501,2) = -1.15^5$$
$$t_3 = L(501,3) = -1.04^4$$

If we now plot both functions together, $f(x)$ and $\widehat{f}(x)$ we get the following result:

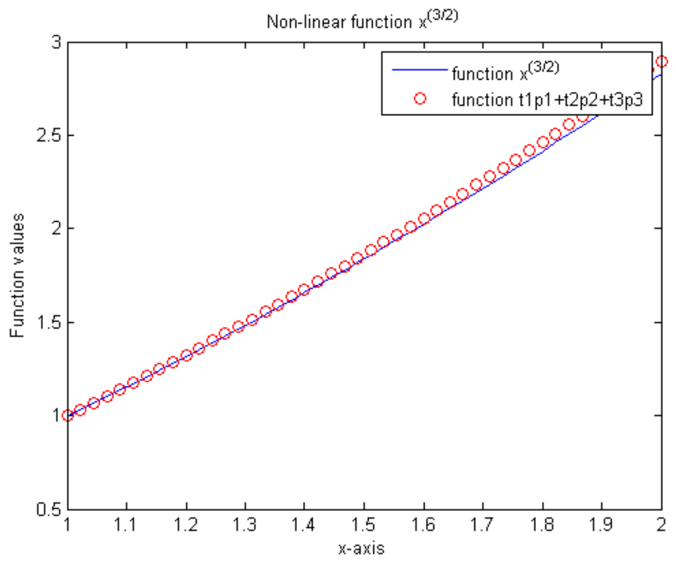

Figure 6: Shows the non-linear function $x^{1.5}$ being fitted by a metamodel where the parameters come from the *scores-library*

We can see that we get very good results.
Consider now the following non-linear function:

$$f(x) = x^{(3/2)} - x^{(1/2)} \tag{16}$$

The exponents fulfil the requirement of being inside $\omega_i \in [-1,\ 2]$.
We use the same matrix we had before, namely $X_{(600x600)}$ and we can compute the PCA one more time, meaning we get the same result as before:

$$x_j^{\omega_i} : X_{(600x600)} \ \rightarrow \ PCA$$

The difference this time will be given in the structure of the metamodel, which is given as follows:

$$f(x) \approx \sum_{i=1}^{3} \mathbf{t}_i \mathbf{p}_i^t - \sum_{i=1}^{3} \mathbf{t}_i \mathbf{p}_i^t \tag{17}$$

where $\mathbf{p}^t$ is $\mathbf{p}$ transposed.
Since we are using the same x_j in computing the metamodel for $x^{3/2}$ and $x^{1/2}$, this implies that $\mathbf{p}_i^t$ from (17) is the same in both sums and for this reason can be factorized. We can make the same supposition for the library L, which will be the same in both sums because we are using the same ω_i for the metamodel for $x^{3/2}$ and $x^{1/2}$. But the values we are querying in L come from different index-positions within L.
for $\omega(s_1) = 3/2$:

$$\omega_i \in [-1, \omega_2, \omega_3, ..., \underbrace{1.5}_{position\ 501}, ..., 2]$$

and for $\omega(s_2) = 1/2$:

$$\omega_i \in [-1, \omega_2, \omega_3, ..., \underbrace{0.5}_{position\ 301}, ..., 2]$$

This gives the following metamodel:

$$f(x) \approx [\underbrace{\omega(s_1)}_{L501,1} - \underbrace{\omega(s_2)}_{L301,1}]\mathbf{p}_1^t + [\underbrace{\omega(s_1)}_{L501,2} - \underbrace{\omega(s_2)}_{L301,2}]\mathbf{p}_2^t + [\underbrace{\omega(s_1)}_{L501,3} - \underbrace{\omega(s_2)}_{L301,3}]\mathbf{p}_3^t \qquad (18)$$

We pick up the six values from the common L which corresponds to these index-positions and we compute the difference:

$$\begin{aligned} t_1 &= L(501,1) - L(301,1) = -50.4210 \\ t_2 &= L(501,2) - L(301,2) = 11.4172 \\ t_3 &= L(501,3) - L(301,3) = -0.1087 \end{aligned}$$

Finally, we now have a metamodel with a structure similar to (12):

$$f(x) \approx t_1\mathbf{p}_1(x) + t_2\mathbf{p}_2(x) + t_3\mathbf{p}_3(x) + e_4 \qquad (19)$$

We plot both functions and we get:

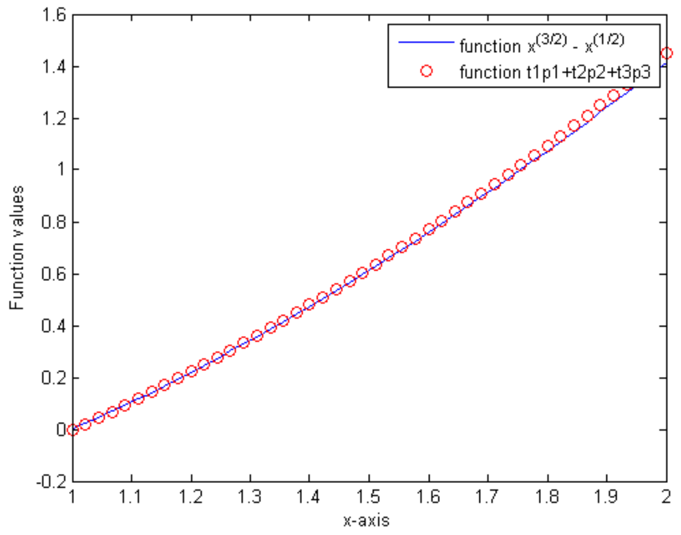

Figure 7

The results, as we can see, are very satisfying.
These examples ensure that L can indeed be used as a *library*. These results are important for further analysis, (see section 16, recasting).
In the next section of the thesis, a metamodel has been developed for approximate S-system differential equations, having a time-series given in the left hand side (LHS) of the system as the only information available.

8 Scalar case of S-systems and the bi-linear metamodel

A scalar case of S-system representation is given as follows:

$$\dot{X} = \underbrace{\alpha X^g - \beta X^h}_{f(x,t)} \tag{20}$$

one equation, four parameters and X common. Recall that $\alpha, \beta \in \mathbb{R}$ are called *rate constants* and $g, h \in \mathbb{R}$ are referred to as *kinetic orders.*
We will develop a metamodel which is simply an approximation of the original function, and we have called this metamodel $\widehat{f}(x,t)$.
As mentioned briefly previously, the right hand side (RHS) in (20) will be unknown. The only available information we are going to have is the LHS, which will be giving as a vector with function-values of $f(x,t)$ at some time t. In other words, $f(x,t)$ will be given discrete. First of all, we begin by deriving the metamodel.

8.1 Deriving the metamodel

Recall the results from section (**3**), then defined αX^g and βX^h as shown bellow:

$$\begin{aligned} \alpha X^g &\approx \alpha \boldsymbol{\lambda}_1 \mathbf{p}_1^t(x_j) + \alpha \boldsymbol{\lambda}_2 \mathbf{p}_2^t(x_j) + \alpha \boldsymbol{\lambda}_3 \mathbf{p}_3^t(x_j) + \alpha \epsilon_4 \\ \beta X^h &\approx \beta \boldsymbol{\mu}_1 \mathbf{p}_1^t(x_j) + \beta \boldsymbol{\mu}_2 \mathbf{p}_2^t(x_j) + \beta \boldsymbol{\mu}_3 \mathbf{p}_3^t(x_j) + \beta \epsilon_4 \end{aligned} \tag{21}$$

where $\mathbf{p}^t$ is $\mathbf{p}$ transposed; $\boldsymbol{\lambda}_1$, $\boldsymbol{\lambda}_2$, $\boldsymbol{\lambda}_3$ and $\boldsymbol{\mu}_1$,$\boldsymbol{\mu}_2$, $\boldsymbol{\mu}_3$ are vector-spaces, α and β are just constants $\in \mathbb{R}$.
Since we know that x_j is common for both metamodels, $\mathbf{p}_i^t$ can be factorized giving the following result:

$$\alpha X^g - \beta X^h \approx (\alpha \boldsymbol{\lambda}_1 - \beta \boldsymbol{\mu}_1)\mathbf{p}_1^t + (\alpha \boldsymbol{\lambda}_2 - \beta \boldsymbol{\mu}_2)\mathbf{p}_2^t + (\alpha \boldsymbol{\lambda}_3 - \beta \boldsymbol{\mu}_3)\mathbf{p}_3^t + \epsilon_4 \tag{22}$$

In order to simplify notation we can write (22) as shown below:

$$\alpha X^g - \beta X^h \approx \boldsymbol{\Gamma}_1 \mathbf{p}_1^t(x_j) + \boldsymbol{\Gamma}_2 \mathbf{p}_2^t(x_j) + \boldsymbol{\Gamma}_3 \mathbf{p}_3^t(x_j) + \epsilon_4 \tag{23}$$

Equation (23) is the metamodel for the (1x1) S-system, where:
$\boldsymbol{\Gamma}_1 = (\alpha \boldsymbol{\lambda}_1 - \beta \boldsymbol{\mu}_1)$ is a vector space.
$\boldsymbol{\Gamma}_2 = (\alpha \boldsymbol{\lambda}_2 - \beta \boldsymbol{\mu}_2)$ is a vector space.
$\boldsymbol{\Gamma}_3 = (\alpha \boldsymbol{\lambda}_3 - \beta \boldsymbol{\mu}_3)$ is also a vector space.

Notice that the metamodel in (23) is a general representation of the function $\widehat{f}(x,t)$ which approximates the (1x1) S-system for any given combination of parameters α, β, g, h within a valid domain.
For an explicit (1x1) S-system, the metamodel becomes:

$$\alpha X^g - \beta X^h \approx \Gamma_1 \mathbf{p}_1^t(x_j) + \Gamma_2 \mathbf{p}_2^t(x_j) + \Gamma_3 \mathbf{p}_3^t(x_j) + \epsilon_4 \tag{24}$$

where Γ_i $(i = 1, 2, 3)$, are real-valued constants. These constants, also called parameters for the metamodels are unknown.
Equation (24) can be considered as a parameter estimation problem. We now need to estimate these unknown parameters, which is explained in the next section.

8.2 Metamodel-parameter estimation problem

The process in finding the Γ_1, Γ_2 and Γ_3 parameters from a time-series will be explained through a series of *six steps* in a simple algorithm.

Step 1: Solving the S-system

Assume the (1x1) S-system is a known explicit, this means that:

$$\dot{X} = \underbrace{\widehat{\alpha} X^{\widehat{g}} - \widehat{\beta} X^{\widehat{h}}}_{\text{test function}}, \qquad x(0) = x_o \tag{25}$$

where $\widehat{\alpha}, \widehat{\beta}, \widehat{g}$ and $\widehat{h}$ are real-valued constants.
At the same time we choose to call the function from (25) *test function*.
The first step consists in solving the system in (25). Different methods can be applied, in *matlab* or other computational languages. I will use a built-in function from *matlab* called *ODE45*[5].
The result after solving (25) is a vector which we will refer to from now on as *test solution*. The *test solution* will be given as follows:

$$\underbrace{[x(t_k)]_{k_1=x_o}^{k_K=x_o+x}}_{\text{test solution}} \tag{26}$$

the first value of the vector is the initial condition-value x_o and the last value is $x_o + x$.
What is more I will only use $x(t_k)$ when referring to the *test solution* in (26), this is done in order to simplify notation.
The time interval for which the ODE from (25) is solved can be arbitrarily chosen. The domain of $x(t_k) \in [x_o,\ x_o + x]$ will depend entirely on the behaviour of the solution. This interval gives us information about where $x(t_k)$ is defined. Furthermore, we can find the *max* and the *min* value for the *test solution* within this interval. We are very interested in this interval, which will help us to decide the domain for x_j from the power-function representation $x_j^{\omega_i}$ used in assembling the multivariate power function matrix $X_{(IxJ)}$.

[5] *ode45* is a numerical tool used in solving differential equation problems. It applies the fourth and fifth order *Runge-Kutta* method. (*MathWorks-DocumentationCenter-ode45*)

Step 2: Computing the PCA

Recall that $x_j^{\omega_i}$ will give a matrix X of size (IxJ). The domain for the columns of X will vary accordingly to the domain of x_j alone. This is because the domain of ω_i (rows of X) has been set to the default value $= [-1,\ 2]$.
We have:

$$x_j^{\omega_i} : X_{(IxJ)} = \begin{bmatrix} x_1^{-1} & x_2^{-1} & ... & x_J^{-1} \\ x_1^{\omega_2} & x_2^{\omega_2} & ... & x_J^{\omega_2} \\ .. & .. & ... & .. \\ x_1^{2} & x_2^{2} & ... & x_J^{2} \end{bmatrix} \tag{27}$$

The domain of x_j needs to fulfil two constrains:
a) x_j has to be chosen in such way that $x(t_k)$ becomes a subset of x_j. This is possible if we take a close look to the domain of $x(t_k)$ and we simply chose a wider interval for x_j, perhaps if:

$$x(t_k) \in [x_o,\ x_o + x] \quad \forall\ x > 0$$

we choose:

$$x_j \in [x_o - x,\ x_o + 2x] \quad \forall\ x > 0$$

b) The step-size of x_j has to be much smaller than the step-size of $x(t_k)$. Assume that $k = 50 \Rightarrow x(t_k) \in \mathbb{R}^{50x1}$, thus, we chose $j = 10 * k \Rightarrow x_j \in \mathbb{R}^{500x1}$.
If $X_{(IxJ)}$ fulfils *a)* and *b)* we can compute the PCA to $X_{(IxJ)}$ and proceed to the next step.

Step 3: Superposition

The constrains *a)* and *b)* given on *step 2* ensure that we can write $x(t_k) \subset\ x_j$ and that x_j has smaller step-size than $x(t_k)$. A graphical representation is provided below.

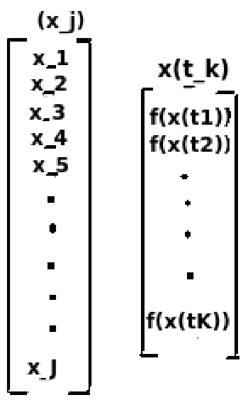

Figure 8

It is now possible to approximate values from x_j to $x(t_k)$ in the following three ways:

$$x_{j31} \approx x(t_1) \tag{28a}$$

$$x_{j67} \approx x(t_2) \tag{28b}$$

$$x_{j2} = x(t_3) \tag{28c}$$

$$x_{j72} = x(t_4) \tag{28d}$$

$$x_{j7} \approx x(t_5) \tag{28e}$$

$$x_{j7} \approx x(t_6) \tag{28f}$$

* *Some values from x_j (of index position 31 and 67) approximate the first and second value of $x(t_k)$, (28a, 28b).*
* *Some values from x_j (of index position 2 and 72) are equal to the third and fourth value of $x(t_k)$, (28c, 28d).*
* *Only one value from x_j (of index position 7) is similar to the fifth and sixth value of $x(t_k)$, (28e, 28f).*

As we can see, the process of approximate values from x_j to $x(t_k)$ will give us a very important result, which is a vector we will call $[idx]$ vector.
This vector contains the index positions of the values of x_j that approximate or are equal to the values of $x(t_k)$. These index positions are the small

numbers which are beside the $j's$ in (28a-28f).

$$idx = \begin{bmatrix} 31 \\ 67 \\ 2 \\ 72 \\ 7 \\ 7 \end{bmatrix} \tag{29}$$

Note that $[idx] \in \mathbb{R}^K$.

Superposition:

We are going to use the $[idx]$ vector to find a composed function $\mathbf{p}_i(x_j)$ $(i = 1, 2, 3)$, of $x(t_k)$. The result will be the composed function $\mathbf{p}_i(x(t_k))$ where $(i = 1, 2, 3)$.
For this job a *matlab-code* has been implemented, the code records the index positions ($[idx]$) of the approximated values of x_j to $x(t_k)$, then it goes through every element in $\mathbf{p}_i(x_j)$ and chooses the values that are in the index positions given in $[idx]$. At the end, the code rearranges the values of $\mathbf{p}_i(x_j)$ accordingly to $[idx]$ and the result is a composed function $\mathbf{p}_i(x(t_k)) \in \mathbb{R}^K$.
To illustrate this step let us consider (28a-28f):
Assuming now that we are done with the approximation step and we have our $[idx]$ vector (29). We will now focus on the $\mathbf{p}_i(x_j)$ vector, for $(i = 1, 2, 3)$, and pick up the value form the index position 31. This is the first value in $[idx]$ and for this reason this value takes the first position in the composed function $\mathbf{p}_i(x(t_k))$. If we repeat the process again we have to pick up the value from $\mathbf{p}_i(x)$ from the index position 67 which is the second value in $[idx]$ and for this reason, this value takes the second position in $\mathbf{p}_i(x(t_k))$. When we have gone through all the elements in $[idx]$ we have a $\mathbf{p}(x(t_k)) \ \in \mathbb{R}^K$.

Step 4: Taking the derivative of the test solution

This step is straightforward. *matlab* is applied to compute the derivative of the *test solution* giving: $\dot{x}(t_k) = dx/dt$ where $dx/dt \in \mathbb{R}^{K-1}$. Notice that we lose one dimension[6] when computing the derivative of the *test solution*.
The step where we differentiate $x(t_k)$ is necessary because we are going to apply linear regression in order to find the unknown parameters $\Gamma_{1,2,3}$, and linear regression requires that we work with time-series.

[6] $Y = diff(X)$ calculates differences between adjacent elements of X along the first array dimension whose size does not equal 1. If X is a vector of length m, then $Y = diff(X)$ returns a vector of length $m - 1$. (*MathWorks-DocumentationCenter-diff*)

Step 5: Linear regression

This step will give us three real-valued constants: Γ_1, Γ_2 and Γ_3, which are the three unknown parameters from our metamodel:

$$\widehat{f}(x,t) = \Gamma_1 \mathbf{p}_1(x_j) + \Gamma_2 \mathbf{p}_2(x_j) + \Gamma_3 \mathbf{p}_3(x_j) + e_4$$

The linear system we are going to solve is given as:

$$M\ \Gamma = dx/dt \tag{30}$$

where

$$\begin{aligned} M &= [\mathbf{p}_1(x(t_k)), \mathbf{p}_2(x(t_k)), \mathbf{p}_3(x(t_k))] \in \mathbb{R}^{K\text{x}3} \\ \Gamma &\ \in \mathbb{R}^{3\text{x}1} \\ dx/dt &\ \in \mathbb{R}^{(K-1)\text{x}1} \end{aligned}$$

As we observe, the system in (30) is not consistent. The RHS has a lower dimension than the LHS. I believe the best option to solve this problem is by deleting the last row from the matrix $M_{(KX3)}$. The reason I suggest such an option is because we lose the last element from $x(t_k)$ during differentiation. We have now:

$$\begin{aligned} M &= [\mathbf{p}_1(x(t_k)), \mathbf{p}_2(x(t_k)), \mathbf{p}_3(x(t_k))] \in \mathbb{R}^{(K-1)\text{x}3} \\ \Gamma &\ \in \mathbb{R}^{3\text{x}1} \\ dx/dt &\ \in \mathbb{R}^{(K-1)\text{x}1} \end{aligned}$$

which is a consistent linear system. We can now compute:

$$\Gamma = M^t\ dx/dt \tag{31}$$

The vector of M forms a basis for the *column space*, the vector dx/dt is not in the *column space* so we need to project it onto it, as a result of this projection we get three real-valued constants: Γ_1, Γ_2 and Γ_3.

Step 6: The metamodel

Once we have the parameters Γ_1, Γ_2 and Γ_3 we can write the metamodel for the (1x1) S-system as shown below:

$$\dot{X} = \underbrace{\Gamma_1 \mathbf{p}_1(x_j) + \Gamma_2 \mathbf{p}_2(x_j) + \Gamma_3 \mathbf{p}_3(x_j)}_{\widehat{f}(x,t)} \tag{32}$$

where $\widehat{f}(x,t) \approx f(x,t)$.
To be able to judge the accuracy of the metamodel in (32), we are going to use the *square error* method:

$$max\ (|x(t_k) - \widehat{x}(t_k)|^2)\ 100\% \tag{33}$$

This method basically computes the difference from the exact solution $x(t_k)$ and the approximated solution $\widehat{x}(t_k)$. The *max* implies that we are only going to get the biggest result after the difference has been computed, then we square our answer and multiply it by 100 in order to have the result in percentage form.
Before we can compute the *square error* method we need to solve the first order differential equation given in (32). The challenge here is that we have our metamodel-function in discrete form and the *ODE45* method from *matlab* does not accept discrete functions. However this matter can be easily solved by another built-in function offered by *matlab*, which is *linear interpolation*. Applying *linear interpolation* to (32) will give a curve where the values $f(t_1), f(t_2), f(t_3), ..., f(t_K)$, and $t_1, t_2, t_3, ..., t_K$ (which are known) are fitted by *matlab*. We can now solve the ODE from (32) using *ODE45*.
This is hugely significant because the *test function* has been solved by means of *ODE45*. We now need to apply the same numerical method to the equation in (32) if we are going to be fair in comparing their solutions.

8.3 Example 1: (1x1) S-system and its bi-linear metamodel

Consider the following (1x1) S-system:

$$\dot{X} = 3X^{0.5} - 2X^{1.5}, \quad x(0) = x_o \tag{34}$$

The equation in (34) is given in explicit form, where $\alpha = 3,\ \beta = 2,\ g = 0.5$ and $h = 1.5$. This means that we can find a metamodel that approximates (34) if we go through the *steps 1- 6*. By doing this we will obtain a metamodel for the scalar case S-system:

$$\dot{X} \approx \Gamma_1 \mathbf{p}_1(x_j) + \Gamma_2 \mathbf{p}_2(x_j) + \Gamma_3 \mathbf{p}_3(x_j) \tag{35}$$

Note that parameters $\Gamma_1,\ \Gamma_2$ and Γ_3 will be different for different initial conditions (x_o).
Let the ODE in (34) have $x_o = 0.2,\ [tspan] \in [0\ 1]$. At the same time we let $\omega_i \in [-1,\ 2]$ and $x_j \in [0.1,\ 5]$, and we then assemble $X_{(IxJ)}$ as follows:

$$x_j^{\omega_i} :\ X_{(600x600)}\ \rightarrow PCA$$

Once we compute the superposition and linear regression, we obtain the following parameter-values:
$\Gamma_1 = 591.5121$
$\Gamma_2 = 35.1352$
$\Gamma_3 = 107.5798$
These results give us the following:

$$\dot{X} \approx 591.5121\mathbf{p}_1(x_j) + 35.1352\mathbf{p}_2(x_j) + 107.5798\mathbf{p}_3(x_j)$$

where $\mathbf{p}_i(x_j) \in \mathbb{R}^{(600X1)}$ for $i = 1, 2, 3$.
Next, we plot both results and compare their solutions.

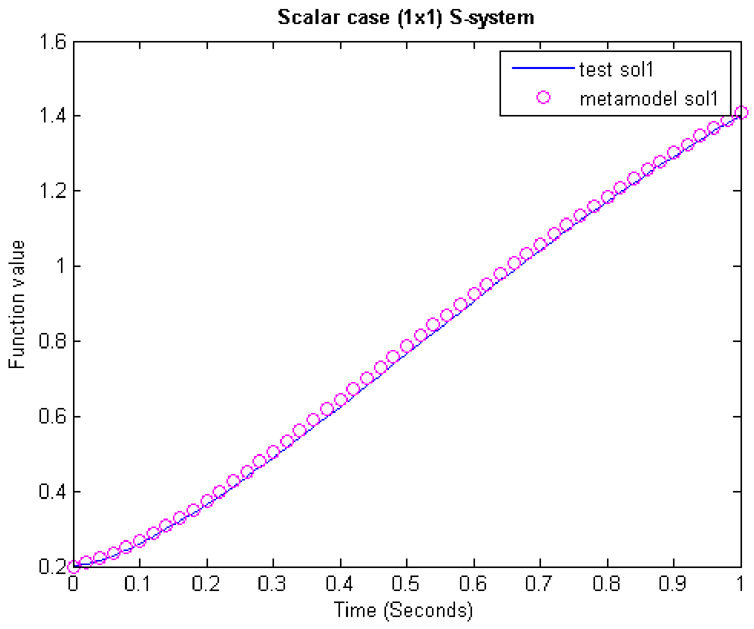

Figure 9

Finally, the *square error* method which is also computed in *matlab* gives:

$$(max|x(t_k) - \widehat{x}(t_k)|^2)\ 100 \approx 0.048\%$$

which is a very accurate result.
In order to show how stable the metamodel is, we are going to solve (34) at $x_o = 2,\ x_o = 2.5,\ x_o = 3$ and $x_o = 3.5$. We will compute a metamodel for each case, which will give different Γ parameters although we are using the same $X_{(600x600)}$ matrix to approximate each ODE.
The results, which are very satisfactory, can be studied in the next graph.

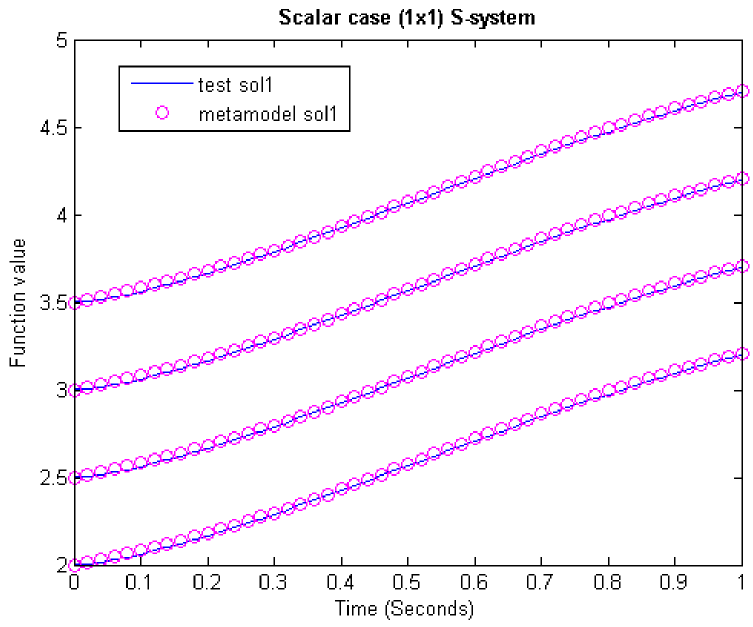

Figure 10

where the *square error* is:
- for $x_o = 2 \approx 0.042\%$.
- for $x_o = 2.5 \approx 0.043\%$.
- for $x_o = 3 \approx 0.043\%$.
- for $x_o = 3.5 \approx 0.042\%$.

Some very important aspects we need to consider now are: How trustworthy are these results? How reliable is the procedure we have employed in solving the discrete ODE given by the metamodel, considering that we used linear interpolation before *ODE45*? and, Is *ODE45* a stable numerical method? These are very relevant questions that will be considered in the next section.

9 Numerical analysis

Remember that we are *interpolating* the metamodel-function before solving it with *ODE45*. The following section will show that, whilst another numerical-integration methods can let us deal directly with ODEs given in discrete form, the methods of interpolation and *ODE45* are stable and provide reliable results.
Before we consider the methods offered by *matlab*, and assuming one wants to apply another numerical method, we will consider some important aspects regarding numerical-integration methods that we should bear in mind when solving ODEs.

As mentioned before, there are many numerical-integration methods that can allow us to deal directly with discrete-ODEs, to name but a few: the *Euler method*, the *midt point method*, the *Heun's method*, the *Runge-kutta method* and many others. A good numerical method has to provide two things: efficiency and accuracy and clearly not all of the methods mentioned above deliver on these fronts. In many cases, one has to decide if time-computing is more important than accuracy or vice versa.
Next, we will consider two methods to show that some methods are more reliable than others, namely the *Euler method* and the *Runge-Kutta* method.

9.1 Euler method

We can find a very good theoretical description of the *Euler method* in [12] and [13].
The *Euler method* is a numerical method used for solving initial value problems. The method provides an approximation of the solution to the initial value problem.
For any

$$x' = f(t, x) \qquad x(t_o) = x_o$$

the *forward euler method* gives:

$$x_{(k+1)} \approx x_k + hf(t_k,\ x_k) \quad \text{where } h = \underbrace{t_{(k+1)} + t_k}_{\Delta t} \tag{36}$$

The algorithm in (36) calculates approximate values $x_0, x_1, x_2, x_3, x_4, .., x_n, ..$ of the unique solution $f(x(t))$ at the set points $t_0 < t_1 < t_2 < t_3 < ... < t_n....$
Notice that $f(t, x)$ satisfies: $x'(t) = f(t, x(t))$.
The *Euler method* can be applied in order to solve the ODE given by the metamodel. Indeed, we have all we need: we have the time interval given by x_j, which are the points where the metamodel has been evaluated -this is Δt in (36)-, we have the initial condition $x_0 = x_k$ in and we have the function values for the metamodel, which corresponds to $f(t_k, x_k)$ in (36).
The problem with this method arises under *error analysis*. As we can observe, the *Euler method* is a step-by-step method, for this reason one can anticipate that by reducing the step-size, we get more accurate results. Broadly speaking, the smaller step-size yield the smaller the error. Such error is known in the field of numerical analysis as truncation error. The truncation error can be measured locally or globally. The local truncation error measures the error at specific points, found by taking the difference between the exact solution and the computed solution at specific points, while the global truncation error measures the difference between the exact solution $f(t)$ and the approximated solution $\widehat{f}(t)$. It is the latter which is of particular interest to us.
In [11], [12] and [13] we find that the *Euler method* has a global truncation

error of *first order* $h(O)$. This mean that the *error* $\to 0$ linearly when the step size $\Delta t \to 0$. This convergence of the error to zero is very slow, which implies that this method is not very accurate, although one has to acknowledge that it is very fast. In order to get good results one should consider a numerical method which has an error with a high convergence rate to zero, this means a higher power of $h(O)$.
According to [13], the global truncation error of the *midt point method* and the *Hun's method* is of second order $h(O)^2$, which implies more accurate results, but if we want accuracy one of the best methods to consider is the *fourth-order Runge-Kutta method.*

9.2 Fourth-order Runge-Kutta method (RK4)

Also called, *classical Runge-Kutta.* It is called fourth-order because its global truncation error is of fourth order $h(O)^4$, which is far quicker than the other methods previously mentioned.

The classical Runge-Kutta scheme

$$x_{(i+1)} = x_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)/h \tag{37}$$

where

$$\begin{aligned} k_1 &= f(t_i, x_i) \\ k_2 &= f(t_i + \frac{1}{2}h,\ x_i + \frac{1}{2}k_1 h) \\ k_3 &= f(t_i + \frac{1}{2}h,\ x_i + \frac{1}{2}k_2 h) \\ k_4 &= f(t_i + h,\ x_i + k_3 h) \end{aligned}$$

Notice that the $k's$ are recurrently related. That is, k_1 appears in the equation of k_2, which appears in the equation of k_3, and so forth. Because each k is a functional evaluation, this recurrence makes the RK4 method very efficient. [13] provides a formal representation of the *global truncation error* of the RK4 method, the most important result of this representation being that this truncation error is of fourth-order $h(O)^4$.
Notice that $h(O)^4$ is achieved with four evaluations of f for each integration-step. We can read in [11] that this situation does not extend to higher order RK methods. For instance, an eight-order RK method ($h(O)^8$), may require twelve evaluations per step, which is quite a high price to pay, time-wise, and implies more function evaluations per step to achieve slightly better results. This simply may not be worthwhile. For this reason the fourth-order RK method is considered the most popular among the *Runge-Kutta* methods.

We can conclude by stating that if one chooses to work with numerical-integration methods, the RK4 is one of the best options.

9.3 ODE45

We can read in [11] and [13] that the *matlab* function *ODE45* employs the *Runge-kutta* method of fourth and fifth order. This is a positive start for this numerical method. At the same time *ODE45* uses a variable step-size algorithm, also called an *adaptive step-size controller* in *matlab* language. This algorithm works as follows: for a given step-size interval the program makes six evaluations of *f*. These values allow evaluations of two *Runge-Kutta* formulas which allow us both to estimate the actual truncation error and proper step-size adjustment to control accuracy. If the estimated error is too large, the step-size decreases until the error tolerance is satisfied. On the contrary, if the computed error is small, the step-size is increased for the next step. This makes the algorithm fast and precise.
The fact that *ODE45* makes so many evaluations for each step-size makes it a very good candidate in solving ODEs. This method will provide very accurate results. It does, however, come with a price: more time spent computing.
We have thus far discovered two excellent candidates for solving ODEs, but before we compare them we are going to study interpolation.
One could be led to believe that interpolation will introduce an error which will propagate through the algorithm and impoverish the results by making them less accurate. This, however, is not the case. I will now share an example to show that interpolation is a reliable and stable buit-in function offered by *matlab*.

9.4 Interpolation

The best way to examine if interpolation is a suitable tool is through looking at an example. We are going to interpolate the trigonometric function: $f(x) = sin(x)$. This function $f(x)$ can be considered as an exact function.
First, we are going to compute *f(x)* at some specific x-values. Secondly, we will interpolate *f(x)* and plot both functions together. We call the interpolated function (πf).
Consider:

$$f(x) = sin(x_i) \text{ where } x \in [0,\ 2\pi] \text{ and } x_i = 1000 \tag{39}$$

This means that $f(x)$ has been evaluated at 1000 points between $[0,\ 2\pi]$.

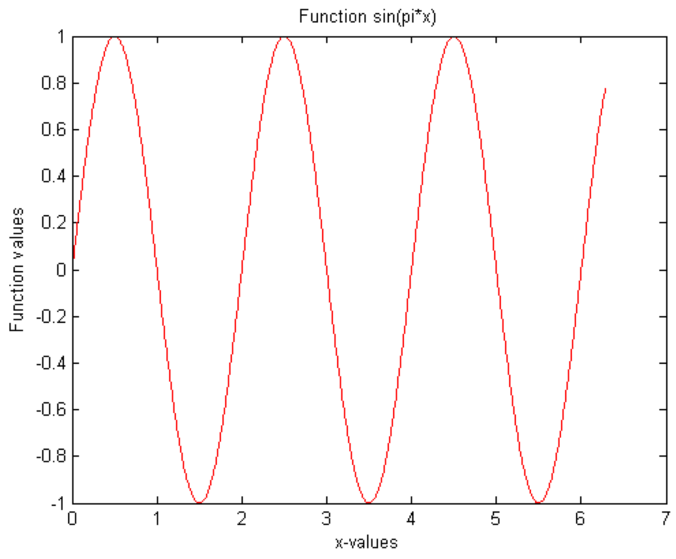

Figure 11

Let us interpolate $f(x)$ at only 15 points:

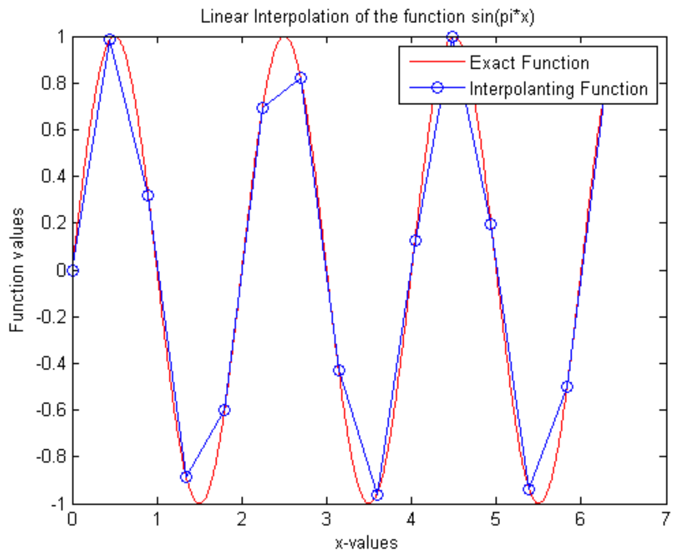

Figure 12

The result, as expected, is terrible. But notice that we are using only 15 points. Consider the next Figure where we increase the steps of $\pi f(x)$ to 50 points:

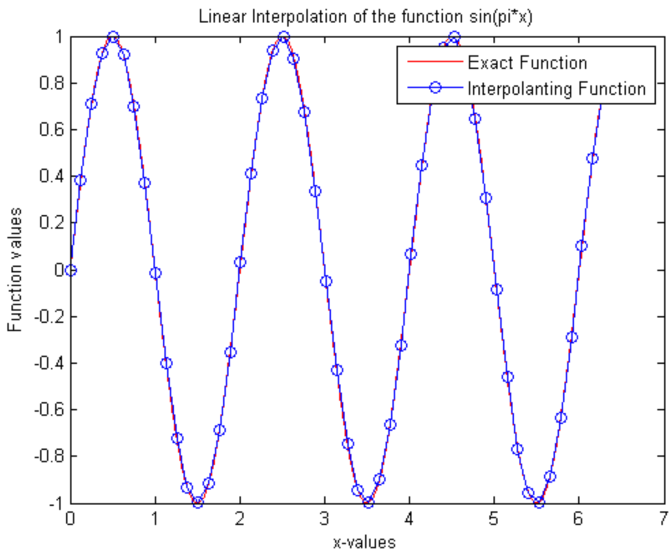

Figure 13

We can see immediately that $\pi f(x)$ improves. The blue circles are the points where we are interpolating.
Let us now interpolate using 1000 points, which are the all points where $f(x)$ has been evaluated at: (39).

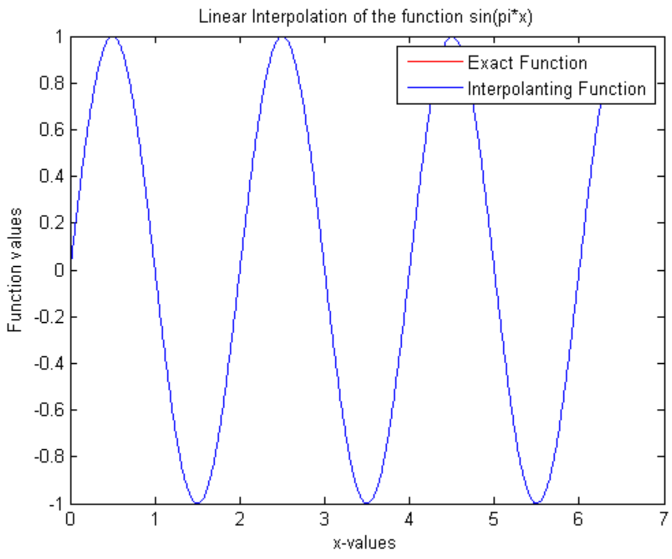

Figure 14: The blue circles have been omitted.

We can not see any signs of a red curve anymore; this is because the interpolating function matches the exact function perfectly.

We can now be sure that interpolation is a reliable tool.
We are now ready for the last part of this section, where we will compare the RK4 method and *ODE45* method.

9.5 ODE45 vs. fourth-order Runge-kutta

Consider the following nonlinear ODE:

$$\dot{r} = 4e^{0.8t} - 0.5r(t) \tag{40}$$

The exact solution for this equation is:

$$r(t) = \frac{4}{1.3}[e^{0.8t} - e^{-0.5t}] + 2e^{-0.5t} \tag{41}$$

We are going to compare precision and measure the time each numerical method uses in order to find a solution. The results will be presented graphically. We will use the absolute error to compare results and the *tic toc* function form *matlab* will help us to find out the elapse time.
We begin by evaluating the RK4 method.
Let (40) have $r(0) = 2$ and $tspan \in [0, 1]$ We find that:

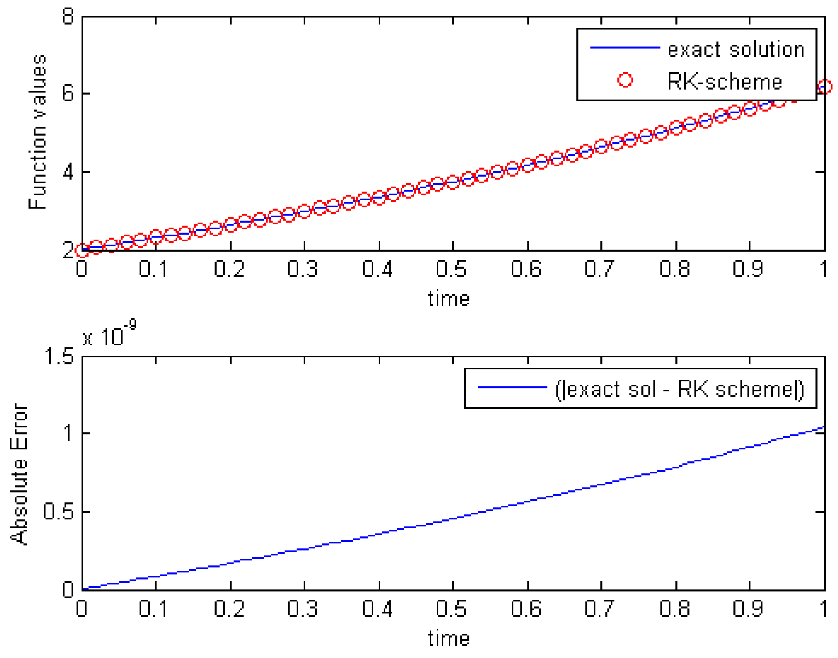

Figure 15

From *matlab* we find that the elapse time is 0.01859 seconds and the absolute error is $(1 * 10^-9)$.

Let us now apply *ODE45*.
We let (40) have $r(0) = 2$ and $tspan \in [0, 1]$. We find that:

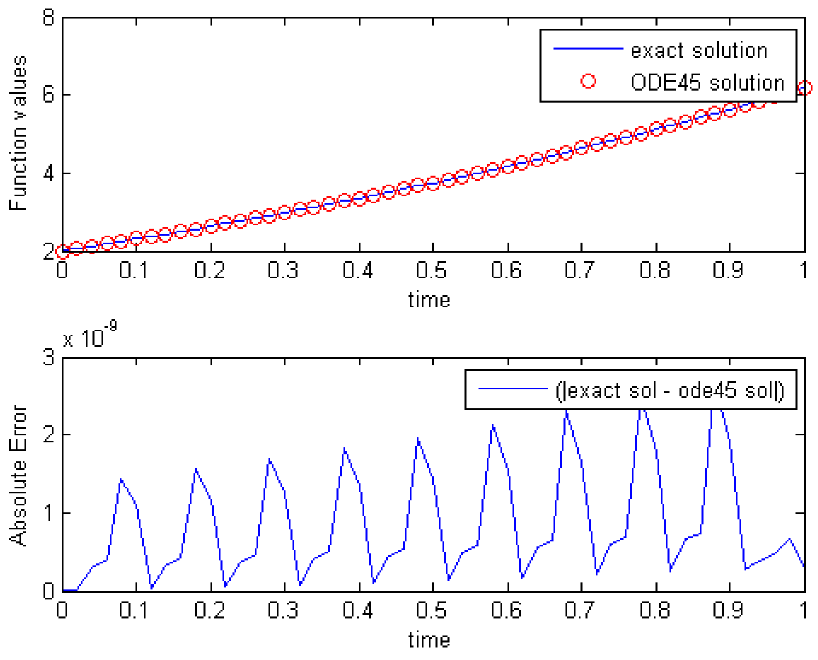

Figure 16

Here, the *tic toc* function is returning 0.532513 seconds, which is far more than the RK method. The absolute error, on the other hand, is much more accurate for the *ODE45* than for the RK4 method. This result can be difficult to observe, due to the fact that the absolute error from *ODE45* gives the appearance that it is oscillating in some way, but what we are really seeing are spikes due to the polynomial interpolation *matlab* uses to produce the points in between the true steps taken by *ODE45*. Consider the next figure, which compares the error produced by the RK4 and *ODE45* method.

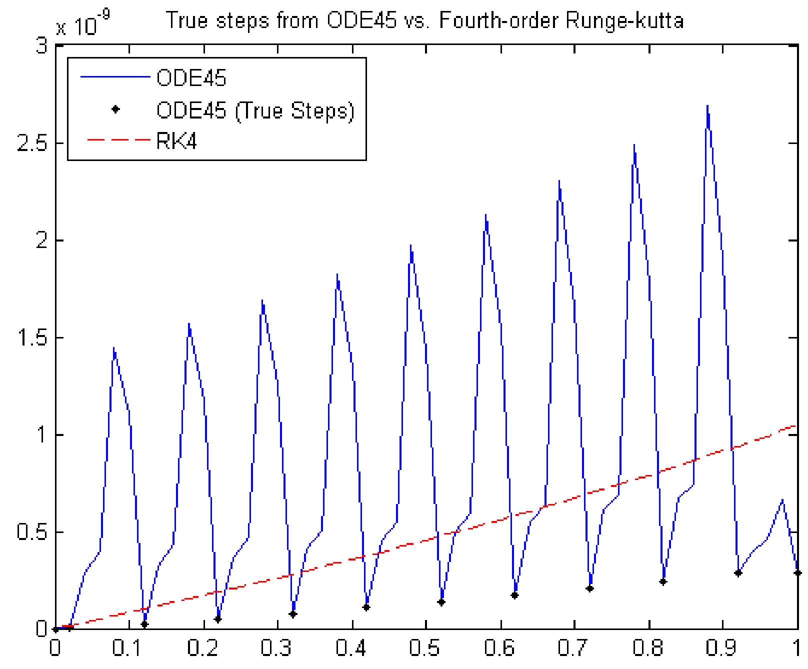

Figure 17: As we can observe, the error at the true steps taken by *ODE45* grows more slowly than the error for RK4. *ODE45* is effectively a higher order method than RK4.

This shows that *ODE45* is more precise but it requires more time to compute the answer if we compare it with the RK4 method.

10 Vector case of S-systems and the bi-linear metamodel

The vector case of S-system is given whenever we have an S-system of two or more equations. Before we begin working with the metamodel for a vector case S-system, we will cite a very important result derived from mathematical theory of systems of differential equations.

Corollary 1: Assume $\dot{X}_i = f(x,t)$ *represents a system of differential equations (for* $i \geq 2$*). If the parameters of* $f(x,t)$ *from every differential equation of the system are independent from each other, we can work with one equation at the time without affecting the main result of the system.*

Corollary 1 is very convenient, bearing in mind that the parameters that are included in the S-system are independent from each other:

$$\dot{X}_i = \alpha_i \prod_{j=1}^{n} X_j^{g_{ij}} - \beta_i \prod_{j=1}^{n} X_j^{h_{ij}}$$

meaning that we can work with i independent metamodels at a time, one for each differential equation of the S-system.

10.1 (2x2) S-systems

Consider:

$$\begin{aligned}\dot{X}_1 &= \alpha_1 X_1^{g_{11}} X_2^{g_{12}} - \beta_1 X_1^{h_{11}} X_2^{h_{12}} \\ \dot{X}_2 &= \alpha_2 X_1^{g_{21}} X_2^{g_{22}} - \beta_2 X_1^{h_{21}} X_2^{h_{22}}\end{aligned} \tag{42}$$

$x(0) = x_o$.

The metamodel for the (2x2) S-system will differ from the scalar case in two fundamental ways:

i) We will have one metamodel for each differential equation of the S-system of differential equations, (follows from Corollary 1); a (2x2) s-system will have 2 metamodels, a (4x4) s-system will have 4 metamodels and so forth.
ii) We have the multiplicity $X_1^{g_{ij}} X_2^{g_{ij}}$ and $X_1^{h_{ij}} X_2^{h_{ij}}$ for each differential equation.

Let us see if *i)* and *ii)* will affect the structure of the metamodel.

10.2 Deriving a metamodel for (2x2) S-systems

We are going to derive a metamodel only for $\dot{X}_i = \dot{X}_1$. The metamodel for $\dot{X}_2$ will be exactly the same.
Consider:

$$\dot{X}_1 = \alpha_1 X_1^{g_{11}} X_2^{g_{12}} - \beta_1 X_1^{h_{11}} X_2^{h_{12}} \tag{43}$$

We have four power functions and two monomials in (43), and we know that each of these power functions can be approximated by a metamodel, giving us the following results:

- for $X_1^{g_{11}} \rightarrow x_{1j}^{\omega_i} : X_1 \approx \Delta_1^1 \mathbf{p}_1^1(x_1) + \Delta_2^1 \mathbf{p}_2^1(x_1) + \Delta_3^1 \mathbf{p}_3^1(x_1)$

- for $X_2^{g_{12}} \rightarrow x_{2j}^{\omega_i} : X_2 \approx \Delta_1^2 \mathbf{p}_1^2(x_2) + \Delta_2^2 \mathbf{p}_2^2(x_2) + \Delta_3^2 \mathbf{p}_3^2(x_2)$

- for $X_1^{h_{11}} \rightarrow x_{3j}^{\omega_i} : X_1 \approx \gamma_1^1 \mathbf{p}_1^1(x_3) + \gamma_2^1 \mathbf{p}_2^1(x_3) + \gamma_3^1 \mathbf{p}_3^1(x_3)$

- for $X_2^{h_{12}} \rightarrow x_{4j}^{\omega_i} : X_2 \approx \gamma_1^2 \mathbf{p}_1^2(x_4) + \gamma_2^2 \mathbf{p}_2^2(x_4) + \gamma_3^2 \mathbf{p}_3^2(x_4)$

where the variable Δ_q^r represents the first monomial, q stays for the parameter number: $q = 1, 2, 3$ and $r = 1$ corresponds to the variable X_1 while $r = 2$ corresponds to X_2. For the second monomial we have γ_q^r where q corresponds to the parameter number: $q = 1, 2, 3$ and $r = 1, 2$ corresponds to X_1 and X_2. The domain of $\omega_i = [-1, 2]$ is default and the domain for x_1,x_2,x_3,x_4 can vary.
Let $\alpha = \beta = 1$, which means that:

$$X_1^{g_{11}} X_2^{g_{12}} - X_1^{h_{11}} X_2^{h_{12}} \approx$$
$$[\Delta_1^1 \mathbf{p}_1^1(x_1) + \Delta_2^1 \mathbf{p}_2^1(x_1) + \Delta_3^1 \mathbf{p}_3^1(x_1)][\Delta_1^2 \mathbf{p}_1^2(x_2) + \Delta_2^2 \mathbf{p}_2^2(x_2) + \Delta_3^2 \mathbf{p}_3^2(x_2)] -$$
$$[\gamma_1^1 \mathbf{p}_1^1(x_3) + \gamma_2^1 \mathbf{p}_2^1(x_3) + \gamma_3^1 \mathbf{p}_3^1(x_3)][\gamma_1^2 \mathbf{p}_1^2(x_4) + \gamma_2^2 \mathbf{p}_2^2(x_4) + \gamma_3^2 \mathbf{p}_3^2(x_4)]$$

In order to simplify notation, we let x_1,x_2,x_3 and x_4 be the same, this step will give us a common factor that we can then factorize:

$$[\Delta_1^1 \mathbf{p}_1^1(x) + \Delta_2^1 \mathbf{p}_2^1(x) + \Delta_3^1 \mathbf{p}_3^1(x)][\Delta_1^2 \mathbf{p}_1^2(x) + \Delta_2^2 \mathbf{p}_2^2(x) + \Delta_3^2 \mathbf{p}_3^2(x)] -$$
$$[\gamma_1^1 \mathbf{p}_1^1(x) + \gamma_2^1 \mathbf{p}_2^1(x) + \gamma_3^1 \mathbf{p}_3^1(x)][\gamma_1^2 \mathbf{p}_1^2(x) + \gamma_2^2 \mathbf{p}_2^2(x) + \gamma_3^2 \mathbf{p}_3^2(x)]$$

This step of factorization is the same as that demonstrated in (21) in Section 5.1.1.
After applying factorization we get the following:

$$[(\Delta_1^1 - \gamma_1^1)\mathbf{p}_1^1 + (\Delta_2^1 - \gamma_2^1)\mathbf{p}_2^1 + (\Delta_3^1 - \gamma_3^1)\mathbf{p}_3^1][(\Delta_1^2 - \gamma_1^2)\mathbf{p}_1^2 + (\Delta_2^2 - \gamma_2^2)\mathbf{p}_2^2 + (\Delta_3^2 - \gamma_3^2)\mathbf{p}_3^2]$$

We have reduced the expression to two brackets being multiplied by each other, and we can therefore observe that each bracket contains three elements. Thus, simple multiplication will give nine factors:

$$[(\Delta_1^1 - \gamma_1^1)(\Delta_1^2 - \gamma_1^2)\mathbf{p}_1^1(x)\mathbf{p}_1^2(x)] + ... + [(\Delta_3^1 - \gamma_3^1)(\Delta_3^2 - \gamma_3^2)\mathbf{p}_3^1(x)\mathbf{p}_3^2(x)] \quad (44)$$

We could simplify notation by letting:

$[(\Delta_1^1 - \gamma_1^1)(\Delta_1^2 - \gamma_1^2)] = \Gamma_1$
$[(\Delta_1^1 - \gamma_1^1)(\Delta_2^2 - \gamma_2^2)] = \Gamma_2$
$[(\Delta_1^1 - \gamma_1^1)(\Delta_3^2 - \gamma_3^2)] = \Gamma_3$
.
.
.
$[(\Delta_3^1 - \gamma_3^1)(\Delta_3^2 - \gamma_3^2)] = \Gamma_9$
where the parameters Γ_1, Γ_2, Γ_3..., Γ_9 are unknown vector-spaces.
By doing this, we can write (44) as follows:

$$\dot{X}_1 \approx \Gamma_1^1(\mathbf{p}_1^1(x)\mathbf{p}_1^2(x)) + \Gamma_2^1(\mathbf{p}_1^1(x)\mathbf{p}_2^2(x)) + ... + \Gamma_9^1(\mathbf{p}_3^1(x)\mathbf{p}_3^2(x)) \quad (45)$$

and we have our metamodel, namely equation (45).

The metamodel for $\dot{X}_2$ will have the same structure as the metamodel in (45), that is:
- Nine *principal components* and
- Nine unknown Γ parameters.
We present below the (2x2) metamodel:

$$\dot{X}_1 = X_1^{g_{11}} X_2^{g_{12}} - X_1^{h_{11}} X_2^{h_{12}} \approx \Gamma_1^1(\mathbf{p}_1^1(x)\mathbf{p}_1^2(x)) + ... + \Gamma_9^1(\mathbf{p}_3^1(x)\mathbf{p}_3^2(x)) \quad (46)$$
$$\dot{X}_2 = X_1^{g_{21}} X_2^{g_{22}} - X_1^{h_{21}} X_2^{h_{22}} \approx \Gamma_1^2(\mathbf{p}_1^1(x)\mathbf{p}_1^2(x)) + ... + \Gamma_9^2(\mathbf{p}_3^1(x)\mathbf{p}_3^2(x))$$

What if $\alpha,\ \beta\ \neq 1$? This does not affect the structure of the metamodel, due to the fact that $\alpha,\ \beta$ are constants, see Section 5.1.1.

10.3 Metamodel-parameter estimation

If we analyse the metamodel for (2x2) S-system:

$$\dot{X}_1 \approx \Gamma_1^1(\mathbf{p}_1^1(x)\mathbf{p}_1^2(x)) + ... + \Gamma_9^1(\mathbf{p}_3^1(x)\mathbf{p}_3^2(x)) \quad (47)$$
$$\dot{X}_2 \approx \Gamma_1^2(\mathbf{p}_1^1(x)\mathbf{p}_1^2(x)) + ... + \Gamma_9^2(\mathbf{p}_3^1(x)\mathbf{p}_3^2(x))$$

we can see that the LHS (time-series) is known, together with the *loading* product ($\mathbf{p}_i^1(x)\ \mathbf{p}_i^2(x)$) for $i = 1, 2, 3$; this product is possible because $\mathbf{p}_i^1,\ \mathbf{p}_i^2 \in \mathbb{R}^J$. The only unknown variables are the Γ parameters, which can be found in exactly the same way as in the case of the metamodel for (1x1) S-system. In other words, the *six steps* shown in Section 4, yields here in the very same way:

1.- We first solve the system in (42) using *ODE45*. The difference here is that we are going to obtain two solutions $(x_1(t_k), x_2(t_k))$.

2.- We compute the PCA to two matrices: X_1 and X_2. This will give us the principal components $\mathbf{p}_i^1(x_1)$ and $\mathbf{p}_i^2(x_2)$ for $i = 1, 2, 3$. Then we compute the product of the principal components as described in (47).

3.- We compute superposition to both solutions $x_1(t_k)$ and $x_2(t_k)$. The result will be two composed functions, $\mathbf{p}_i^1(x_1(t_k))$ and $\mathbf{p}_i^2(x_2(t_k))$, for $i = 1, 2, 3$. And where $\mathbf{p}_i^1(x_1(t_k))$ and $\mathbf{p}_i^2(x_2(t_k)) \in \mathbb{R}^K$.

4.- We differentiate the test solutions, $dx_1(t_k)/dt$, $dx_2(t_k)/dt$.

5.- We apply linear regression to both systems. This will give us nine parameters for the first metamodel, and another nine parameters for the second metamodel.

6.- Both metamodels are ready to be tested.

The *steps 1-6* are explained in detail in section 4, for this reason we are not going to derive them again.

10.4 Example 2: (2x2) S-system and its bi-linear metamodel

In the same way, we will now judge the metamodel for (2x2) S-system by comparing the exact solutions with the solutions produced by the metamodel:

For $\dot{X}_1$: $(max|x_1(t_k) - \widehat{x}_1(t_k)|^2)$

For $\dot{X}_2$: $(max|x_2(t_k) - \widehat{x}_2(t_k)|^2)$

where $x_1(t_k)$ and $x_2(t_k)$ are the exact solutions while $\widehat{x}_1(t_k)$ and $\widehat{x}_2(t_k)$ are the solutions proposed by the metamodel.
Consider now the following system:

$$\begin{aligned}\dot{X}_1 &= 2X_1 0.5X_2 - 2X_1 X_2^{1.2} \\ \dot{X}_2 &= 1.5X_1^{1.2} 3X_2 - X_1^{1.5} X_2\end{aligned} \tag{48}$$

$x(0) = x_o$
The (2x2) metamodel for (48) will be given as:

$$\begin{aligned}\dot{X}_1 &\approx \Gamma_1^1(\mathbf{p}_1^1(x)\mathbf{p}_1^2(x)) + ... + \Gamma_9^1(\mathbf{p}_3^1(x)\mathbf{p}_3^2(x)) \\ \dot{X}_2 &\approx \Gamma_1^2(\mathbf{p}_1^1(x)\mathbf{p}_1^2(x)) + ... + \Gamma_9^2(\mathbf{p}_3^1(x)\mathbf{p}_3^2(x))\end{aligned} \tag{49}$$

We solve (48) and (49) at $x(0) = 0.75$. The values for the Γ parameters for both metamodels are given below:

For the first metamodel:	For the second metamodel:
$\Gamma_1^1 = 75945$	$\Gamma_1^2 = 72.5172$
$\Gamma_2^1 = -59910$	$\Gamma_2^2 = 58655$
$\Gamma_3^1 = 9430$	$\Gamma_3^2 = -7927$
$\Gamma_4^1 = -59910$	$\Gamma_4^2 = 58655$
$\Gamma_5^1 = 2162$	$\Gamma_5^2 = -1356$
$\Gamma_6^1 = 1183$	$\Gamma_6^2 = -1772$
$\Gamma_7^1 = 9430$	$\Gamma_7^2 = -7927$
$\Gamma_8^1 = 1183$	$\Gamma_8^2 = -1772$
$\Gamma_9^1 = 1311$	$\Gamma_9^2 = -956.9559$

The size of X_1 and X_2 used for computing the *loadings* are $X_1,\ X_2 \in \mathbb{R}^{600\text{x}600}$.
The result is shown in the Figure below.

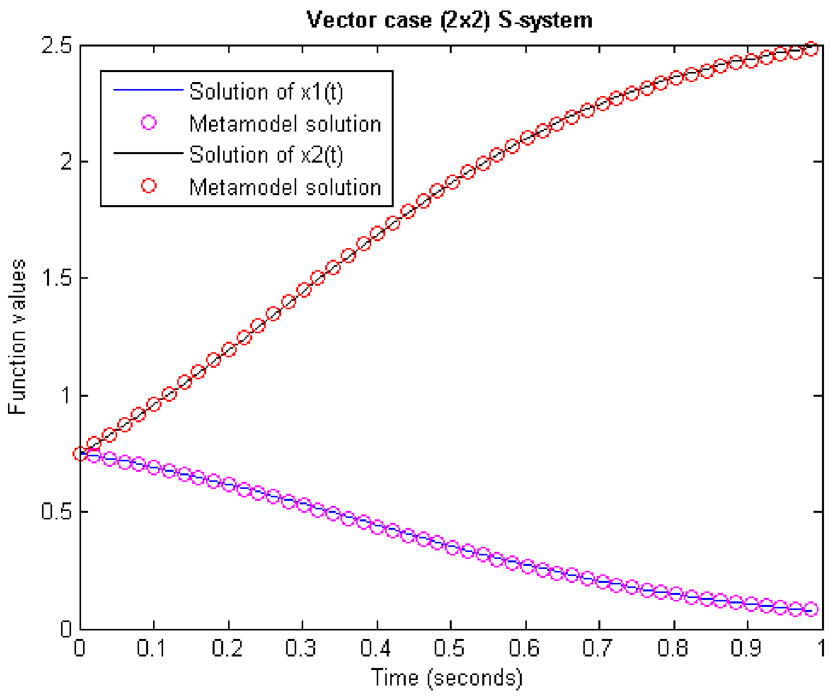

Figure 18

The absolute error between solutions are:

$$max(|x_1(t_k) - \widehat{x_1}(t_k)|^2) = 3.14e^{-5} \approx 0.003\%$$

and

$$max(|x_2(t_k) - \widehat{x_2}(t_k)|^2) = 5.12e^{-5} \approx 0.005\%$$

The (2x2) metamodel can be solved for different initial conditions. The results will maintain the accuracy we have seen up until now.
Consider now (48) and (49) being solved at $x(0) = 0.5,\ 0.75,$ and 1, $X_1,\ X_2 \in \mathbb{R}^{600\text{x}600}$ remains unchanged.

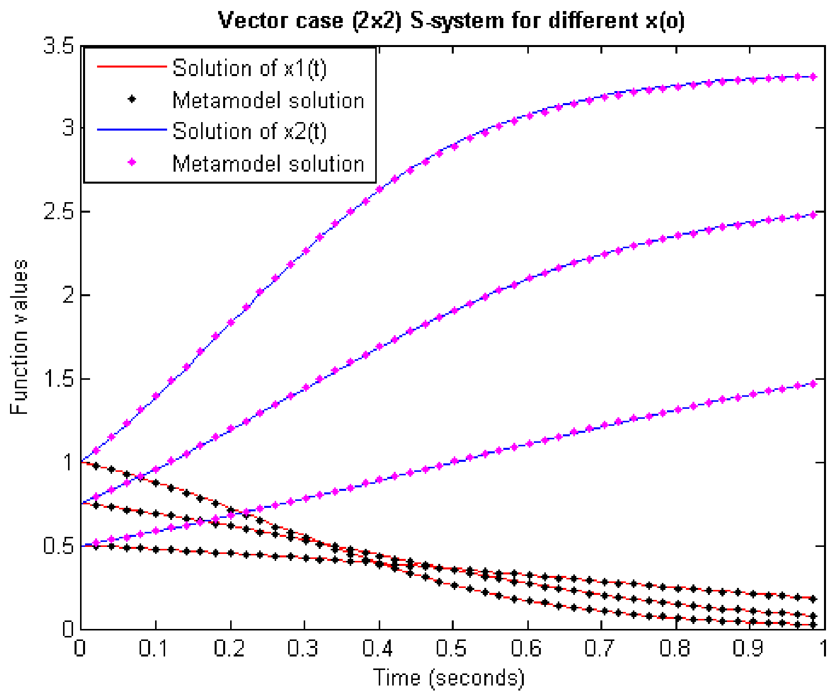

Figure 19

The results are very satisfactory.

Deriving a metamodel for a (3x3), (4x4) and even higher order S-system can be done in the same way as for the (2x2) S-system from Section 7. Nevertheless, there are three important facts we need to emphasize:

1.- The number of metamodels are related to the size of the system. This follows from Corollary 1.

2.- The product

$$\prod_{j=1}^{j=n} X_j^{g_{ij}}, \ \prod_{j=1}^{j=n} X_j^{h_{ij}} \tag{50}$$

will increase proportional to the size of the system, giving an exponential increase (3^n) in the number of PCs and unknown parameters.

3.- We can simplify any desired (NxN)-metamodel (for $N = 2,3,\ldots$) by computing the PCs of the approximation matrices:

$$x_{j1}^{\omega_i} : X_1, \quad x_{j2}^{\omega_i} : X_2 \ , \ldots, \quad x_{jM}^{\omega_i} : X_M$$

on the same x_j-domain.

These three facts, together with the *six steps* from Section 5.2 can help us to find any required (NxN) metamodel. For instance, we know from the (2x2) S-system that we will have two metamodels in order to be able to approximate this system. This result is according to *fact 1*. From *fact 2* we know that the (2x2) S-system has two monomials, and each monomial has the following multiplication: $X_1^{g_{ij}} X_2^{g_{ij}}$ and $X_1^{h_{ij}} X_2^{h_{ij}}$. This implies that we will have 3^2 PCs and 3^2 $-\Gamma-$ parameters for each metamodel. The last point (*fact 3*) allows us to simplify notation due to the factorization of common factors, see Section 7.2.
Let us consider the (3x3) S-system.

10.5 (3x3) S-systems

A (3x3) S-system is given as shown below:

$$\begin{aligned}
\dot{X}_1 &= \alpha_1 X_1^{g_{11}} X_2^{g_{12}} X_3^{g_{13}} - \beta_1 X_1^{h_{11}} X_2^{h_{12}} X_3^{h_{13}} \\
\dot{X}_2 &= \alpha_2 X_1^{g_{21}} X_2^{g_{22}} X_3^{g_{23}} - \beta_2 X_1^{h_{21}} X_2^{h_{22}} X_3^{h_{23}} \\
\dot{X}_3 &= \alpha_3 X_1^{g_{31}} X_2^{g_{32}} X_3^{g_{33}} - \beta_3 X_1^{h_{31}} X_2^{h_{32}} X_3^{h_{33}}
\end{aligned} \tag{51}$$

Fact 1 implies that we will have three metamodels for the system in (51). According to *fact 2*, we will have the following multiplication of the variable X_j:

$$n = 3 \Rightarrow \prod_{j=1}^{j=3} X_j^{g_{ij}}, \ \prod_{j=1}^{j=3} X_j^{h_{ij}}$$

This implies that we are going to have 3^3 PCs and 3^3 $-\Gamma-$ parameters. Consider the first monomial from the first equation in (51), $\alpha_1 = 1$:

$$\underbrace{X_1^{g_{11}}}_{x_{j1}^{\omega_i} \approx [\Delta_1^1 p_1^1 + \Delta_2^1 p_2^1 + \Delta_3^1 p_3^1]} \quad \underbrace{X_2^{g_{12}}}_{x_{j2}^{\omega_i} \approx [\Delta_1^2 p_1^2 + \Delta_2^2 p_2^2 + \Delta_3^2 p_3^2]} \quad \underbrace{X_3^{g_{13}}}_{x_{j3}^{\omega_i} \approx [\Delta_1^3 p_1^3 + \Delta_2^3 p_2^3 + \Delta_3^3 p_3^3]} \tag{52}$$

We know that each power function can be approximated by a metamodel, and in this case we have three metamodels for this first monomial. Thus, simple multiplication will give a total of 27 PCs.
We have exactly the same situation for the second monomial from the first equation in (51), $\beta_1 = 1$:

$$\underbrace{X_1^{h_{11}}}_{x_{j4}^{\omega_i} \approx [\gamma_1^1 p_1^1 + \gamma_2^1 p_2^1 + \gamma_3^1 p_3^1]} \quad \underbrace{X_2^{h_{12}}}_{x_{j5}^{\omega_i} \approx [\gamma_1^2 p_1^2 + \gamma_2^2 p_2^2 + \gamma_3^2 p_3^2]} \quad \underbrace{X_3^{h_{13}}}_{x_{j6}^{\omega_i} \approx [\gamma_1^3 p_1^3 + \gamma_2^3 p_2^3 + \gamma_3^3 p_3^3]} \tag{53}$$

Therefore here, in the very same way, simple multiplication will give 27 PCs.

If we let (for $i = 1, 2, 3$):

$$\begin{gathered}\mathbf{p}_i^1(x_1),\ \mathbf{p}_i^2(x_2),\ \mathbf{p}_i^3(x_3) \text{ in } (52) \\ \text{and} \\ \mathbf{p}_i^1(x_4),\ \mathbf{p}_i^2(x_5),\ \mathbf{p}_i^3(x_6) \text{ in } (53)\end{gathered}$$

be defined on the same x_j-domain, we will have *fact 3*, which allows us to simplify notation because we choose:

$$x_1 = x_2 = x_3 = x_4 = x_5 = x_6 = x$$

Thus, we have a common factor in (52) and (53).
Then (for $i = 1, 2, 3$):

$$\mathbf{p}_i^1(x),\ \mathbf{p}_i^2(x),\ \mathbf{p}_i^3(x)$$

can be factorized giving the following expression:

$$\begin{aligned}\dot{X}_1 &\approx \Gamma_1^1(\mathbf{p}_1^1(x)\mathbf{p}_1^2(x)\mathbf{p}_1^3(x)) + ... + \Gamma_{27}^1(\mathbf{p}_3^1(x)\mathbf{p}_3^2(x)\mathbf{p}_3^3(x)) \\ \dot{X}_2 &\approx \Gamma_1^2(\mathbf{p}_1^1(x)\mathbf{p}_1^2(x)\mathbf{p}_1^3(x)) + ... + \Gamma_{27}^2(\mathbf{p}_3^1(x)\mathbf{p}_3^2(x)\mathbf{p}_3^3(x)) \\ \dot{X}_3 &\approx \Gamma_1^3(\mathbf{p}_1^1(x)\mathbf{p}_1^2(x)\mathbf{p}_1^3(x)) + ... + \Gamma_{27}^3(\mathbf{p}_3^1(x)\mathbf{p}_3^2(x)\mathbf{p}_3^3(x))\end{aligned} \tag{54}$$

which is the (3x3) metamodel.
Recall that *fact 3*, which is letting $\mathbf{p}_i^1(x)$, $\mathbf{p}_i^2(x)$ and $\mathbf{p}_i^3(x)$ be defined on the same x_j-domain, is done so as to simplify notation. In real life, the domain for $x_1, x_2, ..., x_6$ can and will be different.
Let us consider an example for the (3x3) metamodel.

10.6 Example 3: (3x3) S-system and its bi-linear metamodel

Consider the following (3x3) S-system:

$$\begin{aligned}\dot{X}_1 &= X_1X_2X_3 - 2X_1X_2X_3 \\ \dot{X}_2 &= 1.5X_1X_2X_3 - X_1^{1.5}X_2X_3 \\ \dot{X}_3 &= 1.2X_1^{1.2}X_22X_3 - 0.8X_14.2X_2^{0.5}X_3\end{aligned} \tag{55}$$

We are going to solve this system at two different initial conditions.
We are going to use $X_1, X_2, X_3 \in \mathbb{R}^{(600x600)}$ to produce the *loadings*. The procedure we employ in finding the 27 $-\Gamma-$ parameters is the same as before, namely the *six steps* from Section 5.2.
Letting $x(0) = 1.3$ in (54) and (55) gives the following results:

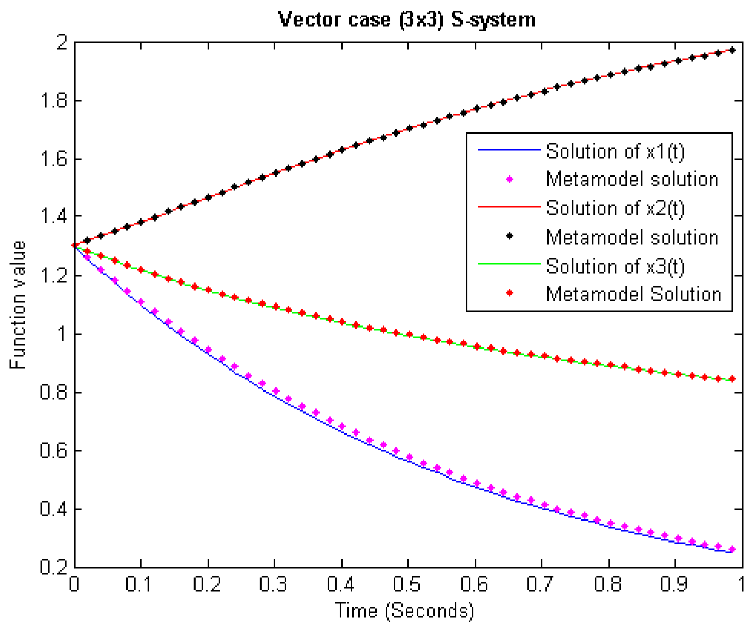

Figure 20: We can observe that the metamodel fits the exact solution very satisfactorily. The biggest of the three square errors is $\approx 0.004\%$

Consecutively $x(0) = 1.6$ in (54) and (55) gives:

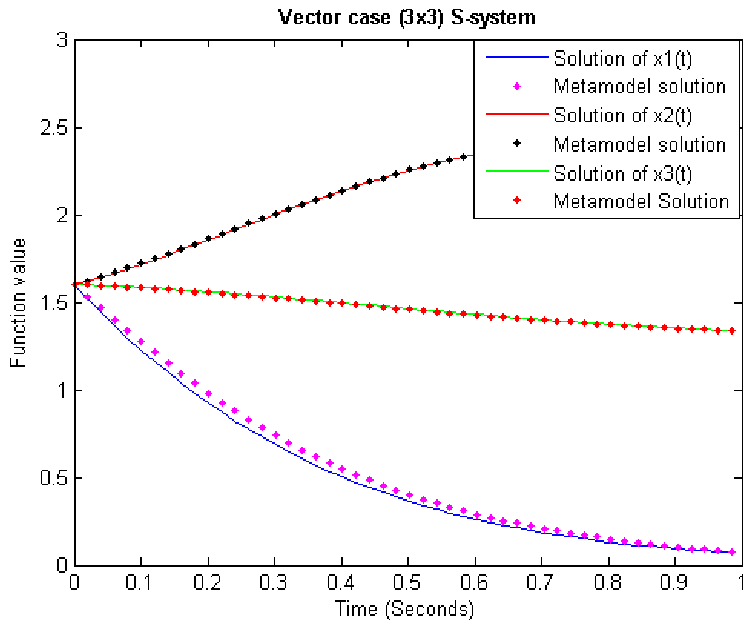

Figure 21: The biggest of the three square errors is $\approx 0.09\%$

The results for the (3x3) metamodel are very good.
The last metamodel we will consider in this thesis is the (4x4) metamodel.

10.7 (4x4) S-systems

A canonical (4x4) S-system is given as follows:

$$\begin{aligned}
\dot{X}_1 &= \alpha_1 X_1^{g_{11}} X_2^{g_{12}} X_3^{g_{13}} X_4^{g_{14}} - \beta_1 X_1^{h_{11}} X_2^{h_{12}} X_3^{h_{13}} X_4^{h_{14}} \\
\dot{X}_2 &= \alpha_2 X_1^{g_{21}} X_2^{g_{22}} X_3^{g_{23}} X_4^{g_{24}} - \beta_2 X_1^{h_{21}} X_2^{h_{22}} X_3^{h_{23}} X_4^{h_{24}} \\
\dot{X}_3 &= \alpha_3 X_1^{g_{31}} X_2^{g_{32}} X_3^{g_{33}} X_4^{g_{34}} - \beta_3 X_1^{h_{31}} X_2^{h_{32}} X_3^{h_{33}} X_4^{h_{34}} \\
\dot{X}_4 &= \alpha_4 X_1^{g_{41}} X_2^{g_{42}} X_3^{g_{43}} X_4^{g_{44}} - \beta_4 X_1^{h_{41}} X_2^{h_{42}} X_3^{h_{43}} X_4^{h_{44}}
\end{aligned} \tag{56}$$

In analogy to *fact 1* and *fact 2*, the (4x4) S-system can be approximated by four metamodels, one for each equation and in addition we will have a fourth order multiplicity of the X_j variable, which implies that we will have 3^4 PCs and $3^4 - \Gamma-$ parameters.
Consider the first monomial from the first equation in (56) with $\alpha = 1$:

$$\underbrace{X_1^{g_{11}}}_{x_{j1}^{\omega_i}\approx[\Delta_1^1 p_1^1+\Delta_2^1 p_2^1+\Delta_3^1 p_3^1]} \quad \underbrace{X_2^{g_{12}}}_{x_{j2}^{\omega_i}\approx[\Delta_1^2 p_1^2+\Delta_2^2 p_2^2+\Delta_3^2 p_3^2]} \quad \underbrace{X_3^{g_{13}}}_{x_{j3}^{\omega_i}\approx[\Delta_1^3 p_1^3+\Delta_2^3 p_2^3+\Delta_3^3 p_3^3]} \quad \underbrace{X_4^{g_{14}}}_{x_{j4}^{\omega_i}\approx[\Delta_1^4 p_1^4+\Delta_2^4 p_2^4+\Delta_3^4 p_3^4]}$$

In the same way, the second monomial with $\beta = 1$ gives:

$$\underbrace{X_1^{h_{11}}}_{x_{j5}^{\omega_i}\approx[\gamma_1^1 p_1^1+\gamma_2^1 p_2^1+\gamma_3^1 p_3^1]} \quad \underbrace{X_2^{h_{12}}}_{x_{j6}^{\omega_i}\approx[\gamma_1^2 p_1^2+\gamma_2^2 p_2^2+\gamma_3^2 p_3^2]} \quad \underbrace{X_3^{h_{13}}}_{x_{j7}^{\omega_i}\approx[\gamma_1^3 p_1^3+\gamma_2^3 p_2^3+\gamma_3^3 p_3^3]} \quad \underbrace{X_4^{h_{14}}}_{x_{j8}^{\omega_i}\approx[\gamma_1^4 p_1^4+\gamma_2^4 p_2^4+\gamma_3^4 p_3^4]}$$

Thus, simple multiplication will give a total of 81 principal components.
Here as well we choose to simplify notation by letting the *approximation matrices* be defined on the same x_j-domain (*fact 3*).
Thereby factorization of common factors will give the (4x4) metamodel:

$$\begin{aligned}
\dot{X}_1 &\approx \Gamma_1^1(\mathbf{p}_1^1(x)\mathbf{p}_1^2(x)\mathbf{p}_1^3(x)\mathbf{p}_1^4(x)) + ... + \Gamma_{81}^1(\mathbf{p}_3^1(x)\mathbf{p}_3^2(x)\mathbf{p}_3^3(x)\mathbf{p}_3^4(x)) \\
\dot{X}_2 &\approx \Gamma_1^2(\mathbf{p}_1^1(x)\mathbf{p}_1^2(x)\mathbf{p}_1^3(x)\mathbf{p}_1^4(x)) + ... + \Gamma_{81}^2(\mathbf{p}_3^1(x)\mathbf{p}_3^2(x)\mathbf{p}_3^3(x)\mathbf{p}_3^4(x)) \\
\dot{X}_3 &\approx \Gamma_1^3(\mathbf{p}_1^1(x)\mathbf{p}_1^2(x)\mathbf{p}_1^3(x)\mathbf{p}_1^4(x)) + ... + \Gamma_{81}^3(\mathbf{p}_3^1(x)\mathbf{p}_3^2(x)\mathbf{p}_3^3(x)\mathbf{p}_3^4(x)) \\
\dot{X}_4 &\approx \Gamma_1^4(\mathbf{p}_1^1(x)\mathbf{p}_1^2(x)\mathbf{p}_1^3(x)\mathbf{p}_1^4(x)) + ... + \Gamma_{81}^4(\mathbf{p}_3^1(x)\mathbf{p}_3^2(x)\mathbf{p}_3^3(x)\mathbf{p}_3^4(x))
\end{aligned} \tag{57}$$

Let us consider an example.

10.8 Example 4: (4x4) S-system and its bi-linear metamodel

Consider:

$$\begin{aligned}
\dot{X}_1 &= X_1X_2X_3 - 2X_1X_21.5X_4 \\
\dot{X}_2 &= 1.5X_1X_20.9X_3X_4 - X_1^{1.5}X_23X_3 \\
\dot{X}_3 &= 1.2X_1X_22X_3 - 0.8X_1X_2^{1.5}5X_4 \\
\dot{X}_4 &= 2X_1^{0.2} - X_4
\end{aligned} \tag{58}$$

The tensor-metamodel for (4x4) S-system is given in (57). We can now solve both systems.
Let $x_o = 1$:

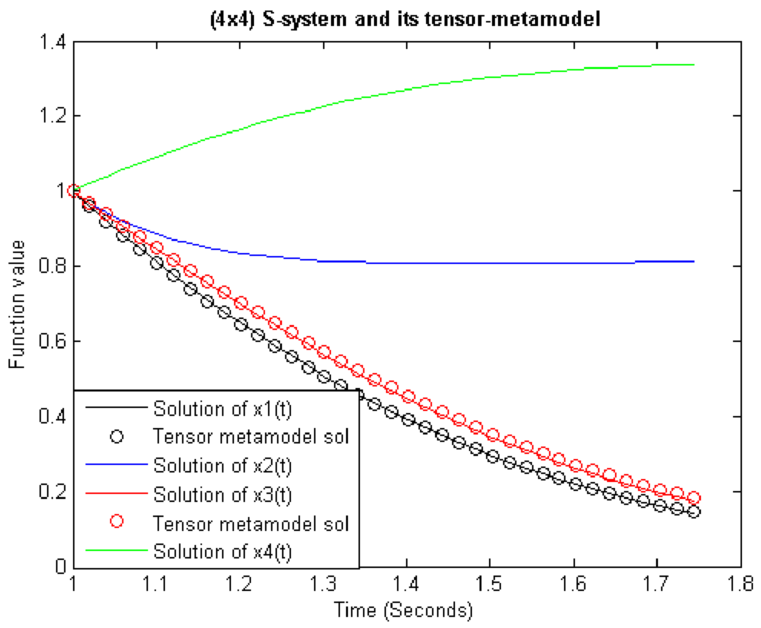

Figure 22

We observe in Figure 24 that only two of the four solutions are fitted by the metamodel, the reason being that some of the solutions of the (4x4) S-system become negative. These kind of results are meaningless when working with real biochemical systems.
The *square error* of the approximated solutions is given below:

$|x_1(t) - \widehat{x}_1(t)|^2 100 \approx 0.0008\%$
$|x_3(t) - \widehat{x}_3(t)|^2 100 \approx 0.007\%$

It is important to bear in mind that whenever we find an approximation to only some of the solutions of the S-system, belonging to the vector case, we should investigate if the system's solutions are entirely positive or not. On the other hand, if we only have a portion of the approximated solution, we need to expand the x_j-domain.

11 Issue with the bi-linear metamodel of big size S-system: Producing the span for the null space

We are to give careful consideration to the product:

$$\prod_{j=1}^{j=n} X_j^{g_{ij}}, \prod_{j=1}^{j=n} X_j^{h_{ij}}$$

If we recall the theory from the principal component analysis, and remember how a matrix X can be approximated by the first three PCs and *scores*, we also know that the last PC ($\mathbf{p}_3(x)$), is the one containing the smallest information of the *variance* of X. Thus, it is logical to imply that should we multiply $\mathbf{p}_3(x)$ n times with itself, the result will be very small values as a result.
Consider:

$$x_j^{\omega_i} : X_{(7x7)} \ \rightarrow \ PCA \ \Rightarrow \ X \approx \mathbf{t}_1\mathbf{p}_1^t + \mathbf{t}_2\mathbf{p}_2^t + \mathbf{t}_3\mathbf{p}_3^t$$

where:

$$\mathbf{p}_3^t(x) = [0.6724\ 0.0768\ 0.3654\ 0.3895\ 0.2342\ 0.0540\ 0.4461]^t$$

then the product

$$\mathbf{p}_3^t(x)\ \mathbf{p}_3^t(x)\ \mathbf{p}_3^t(x)\ \mathbf{p}_3^t(x) = [\mathbf{p}_3^t(x)]^4 \tag{59}$$

will produce:

$$[\mathbf{p}_3^t(x)]^4 = [0.2044\ 0.0000\ 0.0000\ 0.0230\ 0.0030\ 0.0000\ 0.0396]^t$$

and if we go even further:

$$[\mathbf{p}_3^t(x)]^5 = [0.1374\ 0.0000\ 0.0000\ 0.0090\ 0.0007\ 0.0000\ 0.0100]^t$$

which is a vector very close to the *zero-vector*.
We could generalize (59) by letting x_j have different domains:

$$\mathbf{p}_3^t(x_1)\ \mathbf{p}_3^t(x_2)\ \mathbf{p}_3^t(x_3)\ \mathbf{p}_3^t(x_4) \tag{60}$$

The outcome will still be the same. We are producing a vector close to *zero*. Therefore, a natural implication of this result is the fact that: at least one of the PCs from the (2x2), (3x3) or (4x4) metamodel is unnecessary.

Recall that:
the (2x2) metamodel contains 9 PC.
the (3x3) metamodel contains 27 PC.
the (4x4) metamodel contains 81 PC.

The two very important question now remaining to be addressed are:

1.- How many PCs do we actually need for each metamodel?

2.- Which one of the PCs is superficial?

The answer is found in tensor algebra. Tensor algebra approaches the multiplicity of X_j in a more elegant way that the bi-linear model, which is, the tensor product.

12 The tensor product

Also called the *kronecker product* ($\otimes$), it is defined as follows
Let

$$X_1 = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix}, \quad X_2 = \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix}$$

then:

$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \otimes \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} = \begin{bmatrix} a_{1,1}\begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} & a_{1,2}\begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} \\ \\ a_{2,1}\begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} & a_{2,2}\begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} \end{bmatrix}$$

or:

$$X_1 \otimes X_2 = \begin{bmatrix} a_{1,1}b_{1,1} & a_{1,1}b_{1,2} & a_{1,2}b_{1,1} & a_{1,2}b_{1,2} \\ a_{1,1}b_{2,1} & a_{1,1}b_{2,2} & a_{1,2}b_{2,1} & a_{1,2}b_{2,2} \\ a_{2,1}b_{1,1} & a_{2,1}b_{1,2} & a_{2,2}b_{1,1} & a_{2,2}b_{1,2} \\ a_{2,1}b_{2,1} & a_{2,1}b_{2,2} & a_{2,2}b_{2,1} & a_{2,2}b_{2,2} \end{bmatrix}$$

In general if:

$$X_1 \in \mathbb{R}^{(mxn)}, \; X_2 \in \mathbb{R}^{(pxq)}, \quad \text{then} \quad X_1 \otimes X_2 \in \mathbb{R}^{(mpxnq)} \tag{61}$$

Note that $X_1 \otimes X_2 \neq X_2 \otimes X_1$
As we can observe, the tensor product in (61) multiplies all the possible combinations X_1 and X_2 have. This result will still be valid if:

$$x_{j1}^{\omega_i} : X_1 \to \mathbb{R} \quad \otimes \quad x_{j2}^{\omega_i} : X_2 \to \mathbb{R}$$

13 The tensor-metamodel

It has been observed after systematic study that if we take the *Kronecker product* of the power-function matrices $X_1, X_2, \ldots$ before we compute the PCA, we get very interesting results regarding the dimensionality of the PCs. We are going to explore these results more carefully by considering the tensor-metamodel for the (2x2), (3x3) and (4x4) S-system.

13.1 Tensor-metamodel for (2x2) S-systems

Assume we have two power-function matrices:

$$x_{1j}^{\omega_i} : X_{1(IxJ)} \to \mathbb{R}, \quad x_{2j}^{\omega_i} : X_{2(IxJ)} \to \mathbb{R}$$

then, if we compute

$$X_{1(IxJ)} \otimes X_{2(IxJ)} = T_1 \ \to \ PCA \tag{62}$$

and if we let X_1 and X_2 vary, we will get different results. These results are directly related to the dimension of the matrices $X_{1(IxJ)}$ and $X_{2(IxJ)}$.
For instance, if $X_1 \in \mathbb{R}^{(10x10)}$ and $X_2 \in \mathbb{R}^{(10x10)}$, we get $T_1 \in \mathbb{R}^{(100x100)}$ and if we compute the PCA to T_1 we find out that we only need 4 PCs in order to get at least 97% of the variance of T_1.
The interesting outcome occurs when we let X_1 and X_2 increase in dimension, because this gives an increment in the PCs too. But, when T_1 reaches a dimension of (2500x2500), (this value is possible if $X_1, X_2 \in \mathbb{R}^{(50x50)}$), then the number of PCs stabilizes to 6 PCs even though T_1 continues to grow. In other words the number of PCs are converging, as can be seen in the next Figure:

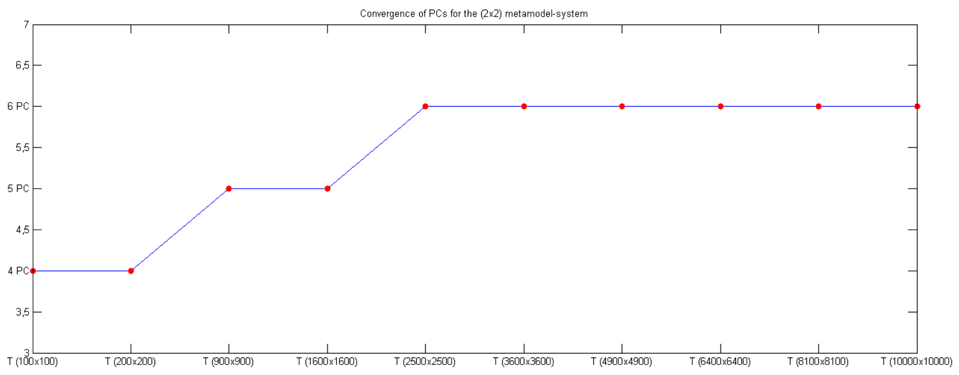

Figure 23

Where:

$$\begin{aligned}
T(100x100) &\text{ is the tensor product of } X_1, X_2 \in \mathbb{R}^{(10x10)},\ 4\ PC \\
T(200x200) &\text{ is the tensor product of } X_1, X_2 \in \mathbb{R}^{(20x20)},\ 4\ PC \\
T(900x900) &\text{ is the tensor product of } X_1, X_2 \in \mathbb{R}^{(30x30)},\ 5\ PC \\
T(1600x1600) &\text{ is the tensor product of } X_1, X_2 \in \mathbb{R}^{(40x40)},\ 5\ PC \\
T(2500x2500) &\text{ is the tensor product of } X_1, X_2 \in \mathbb{R}^{(50x50)},\ 6\ PC \\
T(3600x3600) &\text{ is the tensor product of } X_1, X_2 \in \mathbb{R}^{(60x60)},\ 6\ PC \\
T(4900x4900) &\text{ is the tensor product of } X_1, X_2 \in \mathbb{R}^{(70x70)},\ 6\ PC \\
T(6400x6400) &\text{ is the tensor product of } X_1, X_2 \in \mathbb{R}^{(80x80)},\ 6\ PC \\
T(8100x8100) &\text{ is the tensor product of } X_1, X_2 \in \mathbb{R}^{(90x90)},\ 6\ PC \\
T(10000x10000) &\text{ is the tensor product of } X_1, X_2 \in \mathbb{R}^{(100x100)},\ 6\ PC
\end{aligned}$$

We can see how the number of PCs reaches an apex ($T_{(2500x2500)}$), and stabilizes at this point. These results provide us with valuable information that we will apply to the general (2x2) metamodel.
Recall the (2x2) metamodel:

$$\begin{aligned}
\dot{X}_1 &\approx \Gamma^1_1(\mathbf{p}^1_1(x)\mathbf{p}^2_1(x)) + ... + \Gamma^1_9(\mathbf{p}^1_3(x)\mathbf{p}^2_3(x)) \\
\dot{X}_2 &\approx \Gamma^2_1(\mathbf{p}^1_1(x)\mathbf{p}^2_1(x)) + ... + \Gamma^2_9(\mathbf{p}^1_3(x)\mathbf{p}^2_3(x))
\end{aligned} \tag{63}$$

Due to tensor product, the tensor-metamodel will have less PCs and the simple multiplication in (63) is replaced with the *Kronecker-product* giving us the following system:

$$\begin{aligned}
\dot{X}_1 &\approx \Gamma^1_1(\mathbf{p}^1(x) \otimes \mathbf{p}^2(x)) + ... + \Gamma^1_6(\mathbf{p}^1(x) \otimes \mathbf{p}^2(x)) \\
\dot{X}_2 &\approx \Gamma^2_1(\mathbf{p}^1(x) \otimes \mathbf{p}^2(x)) + ... + \Gamma^2_6(\mathbf{p}^1(x) \otimes \mathbf{p}^2(x))
\end{aligned} \tag{64}$$

Note that we only needs 6 PCs and not 9 as we had first thought. We must emphasize the fact that these 6 PCs guarantee at least 97% of the variance of T_1. Notice as well that we do not yet know the correct combination of the principal components which are being tensor multiplied, and for this reason, the PCs in (64) are not specified.
The fact that we are able to reduce the dimensionality of the PCs is an improvement which will save us time.
Next we are going to compute the tensor product for the (3x3) and (4x4) S-system in order to find the true dimensionality needed for the tensor-metamodel. We shall also show how these results affect the accuracy of the metamodel by testing these tensor-metamodels and we will explain how we can find the PCs that have to be removed from each tensor-metamodel-system.
But first, let us consider the (3x3) tensor-metamodel.

13.2 Tensor-metamodel for (3x3) S-systems

Based on the results from the (2x2) tensor-metamodel consider:

$$x_{1j}^{\omega_i} : X_{1(IxJ)} \to \mathbb{R},\ \ x_{2j}^{\omega_i} : X_{2(IxJ)},\ x_{3j}^{\omega_i} : X_{3(IxJ)} \to \mathbb{R}$$

then for the product:

$$X_1 \otimes X_2\ \otimes X_3 = T_2\ \to\ PCA$$

where X_1, X_2 and X_3 varies, we have the following results:

$$\begin{aligned}
T(1000x1000)\ &= X_1 \otimes X_2\ \otimes X_3 \text{ where } X_1, X_2, X_3 \in \mathbb{R}^{(10x10)},\ 7\ PC \\
T(3375x3375)\ &= X_1 \otimes X_2\ \otimes X_3 \text{ where } X_1, X_2, X_3 \in \mathbb{R}^{(15x15)},\ 9\ PC \\
T(8000x8000)\ &= X_1 \otimes X_2\ \otimes X_3 \text{ where } X_1, X_2, X_3 \in \mathbb{R}^{(20x20)},\ 7\ PC \\
T(15625x15625)\ &= X_1 \otimes X_2\ \otimes X_3 \text{ where } X_1, X_2, X_3 \in \mathbb{R}^{(25x25)},\ 11\ PC \\
T(27000x27000)\ &= X_1 \otimes X_2\ \otimes X_3 \text{ where } X_1, X_2, X_3 \in \mathbb{R}^{(30x30)},\ 11\ PC \\
T(42875x42875)\ &= X_1 \otimes X_2\ \otimes X_3 \text{ where } X_1, X_2, X_3 \in \mathbb{R}^{(35x35)},\ 12\ PC \\
T(64000x64000)\ &= X_1 \otimes X_2\ \otimes X_3 \text{ where } X_1, X_2, X_3 \in \mathbb{R}^{(40x40)},\ 12\ PC \\
T(91125x91125)\ &= X_1 \otimes X_2\ \otimes X_3 \text{ where } X_1, X_2, X_3 \in \mathbb{R}^{(45x45)},\ 13\ PC \\
T(125000x125000)\ &= X_1 \otimes X_2\ \otimes X_3 \text{ where } X_1, X_2, X_3 \in \mathbb{R}^{(50x50)},\ 13\ PC \\
T(166375x166375)\ &= X_1 \otimes X_2\ \otimes X_3 \text{ where } X_1, X_2, X_3 \in \mathbb{R}^{(55x55)},\ 14\ PC \\
T(216000x216000)\ &= X_1 \otimes X_2\ \otimes X_3 \text{ where } X_1, X_2, X_3 \in \mathbb{R}^{(60x60)},\ 14\ PC \\
T(274625x274625)\ &= X_1 \otimes X_2\ \otimes X_3 \text{ where } X_1, X_2, X_3 \in \mathbb{R}^{(65x65)},\ 14\ PC \\
T(343000x343000)\ &= X_1 \otimes X_2\ \otimes X_3 \text{ where } X_1, X_2, X_3 \in \mathbb{R}^{(70x70)},\ 14\ PC \\
T(1000000x1000000)\ &= X_1 \otimes X_2\ \otimes X_3 \text{ where } X_1, X_2, X_3 \in \mathbb{R}^{(100x100)},\ 14\ PC \\
T(1728000x1728000)\ &= X_1 \otimes X_2\ \otimes X_3 \text{ where } X_1, X_2, X_3 \in \mathbb{R}^{(120x120)},\ 14\ PC \\
T(2197000x2197000)\ &= X_1 \otimes X_2\ \otimes X_3 \text{ where } X_1, X_2, X_3 \in \mathbb{R}^{(130x130)},\ 14\ PC \\
T(2744000x2744000)\ &= X_1 \otimes X_2\ \otimes X_3 \text{ where } X_1, X_2, X_3 \in \mathbb{R}^{(140x140)},\ 14\ PC
\end{aligned}$$

The optimal number of PCs needed for the (3x3) tensor-metamodel are 14 PCs. Note that these 14 PC give at least 97% of the variance of T_2.
We recall the (3x3) metamodel:

$$\begin{aligned}
\dot{X}_1 &\approx \Gamma_1^1(\mathbf{p}_1^1(x)\mathbf{p}_1^2(x)\mathbf{p}_1^3(x)) + ... + \Gamma_{27}^1(\mathbf{p}_3^1(x)\mathbf{p}_3^2(x)\mathbf{p}_3^3(x)) \\
\dot{X}_2 &\approx \Gamma_1^2(\mathbf{p}_1^1(x)\mathbf{p}_1^2(x)\mathbf{p}_1^3(x)) + ... + \Gamma_{27}^2(\mathbf{p}_3^1(x)\mathbf{p}_3^2(x)\mathbf{p}_3^3(x)) \\
\dot{X}_3 &\approx \Gamma_1^3(\mathbf{p}_1^1(x)\mathbf{p}_1^2(x)\mathbf{p}_1^3(x)) + ... + \Gamma_{27}^3(\mathbf{p}_3^1(x)\mathbf{p}_3^2(x)\mathbf{p}_3^3(x))
\end{aligned} \qquad (65)$$

If we now apply the results from above we get the (3x3) tensor-metamodel:

$$\begin{aligned}\dot{X}_1 &\approx \Gamma_1^1(\mathbf{p}^1(x)\otimes\mathbf{p}^2(x)\otimes\mathbf{p}^3(x)) + ... + \Gamma_{14}^1(\mathbf{p}^1(x)\otimes\mathbf{p}^2(x)\otimes\mathbf{p}^3(x))\\ \dot{X}_2 &\approx \Gamma_1^2(\mathbf{p}^1(x)\otimes\mathbf{p}^2(x)\otimes\mathbf{p}^3(x)) + ... + \Gamma_{14}^2(\mathbf{p}^1(x)\otimes\mathbf{p}^2(x)\otimes\mathbf{p}^3(x))\\ \dot{X}_3 &\approx \Gamma_1^3(\mathbf{p}^1(x)\otimes\mathbf{p}^2(x)\otimes\mathbf{p}^3(x)) + ... + \Gamma_{14}^3(\mathbf{p}^1(x)\otimes\mathbf{p}^2(x)\otimes\mathbf{p}^3(x))\end{aligned} \tag{66}$$

Notice how advantageous the Kronecker product is, seeing as we have been able to omit 13 PCs from the (3x3) tensor-metamodel.

13.3 Tensor-metamodel for (4x4) S-systems

This time we are considering the following tensor product:

$$X_1 \otimes X_2 \ \otimes X_3 \otimes X_4 = T_3 \ \rightarrow \ PCA$$

The convergence pattern is given next:

$$\begin{aligned}
T(50625x50625) &= X_1\otimes X_2 \ \otimes X_3\otimes X_4,\ X_1,X_2,X_3,X_4\in\mathbb{R}^{(15x15)},\ 18\ PC\\
T(160000x160000) &= X_1\otimes X_2 \ \otimes X_3\otimes X_4,\ X_1,X_2,X_3,X_4\in\mathbb{R}^{(20x20)},\ 21\ PC\\
T(390625x390625) &= X_1\otimes X_2 \ \otimes X_3\otimes X_4,\ X_1,X_2,X_3,X_4\in\mathbb{R}^{(25x25)},\ 25\ PC\\
T(810000x810000) &= X_1\otimes X_2 \ \otimes X_3\otimes X_4,\ X_1,X_2,X_3,X_4\in\mathbb{R}^{(30x30)},\ 27\ PC\\
T(1500625x1500625) &= X_1\otimes X_2 \ \otimes X_3\otimes X_4,\ X_1,X_2,X_3,X_4\in\mathbb{R}^{(35x35)},\ 29\ PC\\
T(2560000x2560000) &= X_1\otimes X_2 \ \otimes X_3\otimes X_4,\ X_1,X_2,X_3,X_4\in\mathbb{R}^{(40x40)},\ 30\ PC\\
T(4100625x4100625) &= X_1\otimes X_2 \ \otimes X_3\otimes X_4,\ X_1,X_2,X_3,X_4\in\mathbb{R}^{(45x45)},\ 31\ PC\\
T(6250000x6250000) &= X_1\otimes X_2 \ \otimes X_3\otimes X_4,\ X_1,X_2,X_3,X_4\in\mathbb{R}^{(50x50)},\ 31\ PC\\
T(9150625x9150625) &= X_1\otimes X_2 \ \otimes X_3\otimes X_4,\ X_1,X_2,X_3,X_4\in\mathbb{R}^{(55x55)},\ 32\ PC\\
T(12960000x12960000) &= X_1\otimes X_2 \ \otimes X_3\otimes X_4,\ X_1,X_2,X_3,X_4\in\mathbb{R}^{(60x60)},\ 32\ PC\\
T(24010000x24010000) &= X_1\otimes X_2 \ \otimes X_3\otimes X_4,\ X_1,X_2,X_3,X_4\in\mathbb{R}^{(70x70)},\ 33\ PC\\
T(31640625x31640625) &= X_1\otimes X_2 \ \otimes X_3\otimes X_4,\ X_1,X_2,X_3,X_4\in\mathbb{R}^{(75x75)},\ 33\ PC\\
T(40960000x40960000) &= X_1\otimes X_2 \ \otimes X_3\otimes X_4,\ X_1,X_2,X_3,X_4\in\mathbb{R}^{(80x80)},\ 33\ PC\\
T(52200625x52200625) &= X_1\otimes X_2 \ \otimes X_3\otimes X_4,\ X_1,X_2,X_3,X_4\in\mathbb{R}^{(85x85)},\ 33\ PC\\
T(65610000x65610000) &= X_1\otimes X_2 \ \otimes X_3\otimes X_4,\ X_1,X_2,X_3,X_4\in\mathbb{R}^{(90x90)},\ 33\ PC\\
T(100000000x100000000) &= X_1\otimes X_2 \ \otimes X_3\otimes X_4,\ X_1,X_2,X_3,X_4\in\mathbb{R}^{(100x100)},\ 33\ PC\\
T(174900625x174900625) &= X_1\otimes X_2 \ \otimes X_3\otimes X_4,\ X_1,X_2,X_3,X_4\in\mathbb{R}^{(115x115)},\ 33\ PC
\end{aligned}$$

We only need 33 PCs for the (4x4) tensor-metamodel, this is less than a half

of the original metamodel.
Finally, the (4x4) tensor-metamodel is given as follows:

$$\begin{aligned}
\dot{X}_1 &\approx \Gamma_1^1(\mathbf{p}^1(x) \otimes \mathbf{p}^2(x) \otimes \mathbf{p}^3(x) \otimes \mathbf{p}^4(x)) + ... + \Gamma_{33}^1(\mathbf{p}^1(x) \otimes \mathbf{p}^2(x) \otimes \mathbf{p}^3(x) \otimes \mathbf{p}^4(x)) \\
\dot{X}_2 &\approx \Gamma_1^2(\mathbf{p}^1(x) \otimes \mathbf{p}^2(x) \otimes \mathbf{p}^3(x) \otimes \mathbf{p}^4(x)) + ... + \Gamma_{33}^2(\mathbf{p}^1(x) \otimes \mathbf{p}^2(x) \otimes \mathbf{p}^3(x) \otimes \mathbf{p}^4(x)) \\
\dot{X}_3 &\approx \Gamma_1^3(\mathbf{p}^1(x) \otimes \mathbf{p}^2(x) \otimes \mathbf{p}^3(x) \otimes \mathbf{p}^4(x)) + ... + \Gamma_{33}^3(\mathbf{p}^1(x) \otimes \mathbf{p}^2(x) \otimes \mathbf{p}^3(x) \otimes \mathbf{p}^4(x)) \\
\dot{X}_4 &\approx \Gamma_1^4(\mathbf{p}^1(x) \otimes \mathbf{p}^2(x) \otimes \mathbf{p}^3(x) \otimes \mathbf{p}^4(x)) + ... + \Gamma_{33}^4(\mathbf{p}^1(x) \otimes \mathbf{p}^2(x) \otimes \mathbf{p}^3(x) \otimes \mathbf{p}^4(x))
\end{aligned}$$

We have therefore rid ourselves of 48 PCs!
We can now move forward and find the right combination of PCs for each tensor-metamodel.

14 The tensor singular value decomposition

We have provided in Section 2.3 the method we have employed to compute the PCA, which is the SVD.
Equations (1) and (2) show how these two are connected:

$$SVD : X = U\Sigma V^t \tag{1}$$
$$PCA : X = (U\Sigma)V^t = (T)P^t + e \tag{2}$$

Recall that we have been taking the *Kronecker-product* to X before we computed the PCA in order to find the optimal number of PCs.
We cite now a *Theorem* from tensor algebra concerning the SVD.

Theorem 1

Let $X_1 \in \mathbb{R}^{(mxn)}$ have a singular value decomposition $(U_{X_1}\Sigma_{X_1}V_{X_1}^t)$ and let $X_2 \in \mathbb{R}^{(pxq)}$ have a singular value decomposition $(U_{X_2}\Sigma_{X_2}V_{X_2}^t)$. Then:

$$(U_{X_1} \otimes\ U_{X_2})(\Sigma_{X_1} \otimes\ \Sigma_{X_2})(V_{X_1}^t \otimes\ V_{X_2}^t) \tag{67}$$

yields a singular value decomposition of $X_1 \otimes X_2$, after a simple reordering of the diagonal elements of $(\Sigma_{X_1} \otimes \Sigma_{X_2})$ and the corresponding right and left singular vectors. [10].

We are going to use *Theorem 1* to find the right combination of tensor products each tensor-metamodel has. To do so, we need to focus only on the Σ matrices. This is because the *singular values*, which are the values of Σ, are responsible for giving information about the variance of the PCs, see Appendix A1
We now know that the (2x2) tensor-metamodel needs 6 PCs instead of 9, so we can now apply *Theorem 1* to find the last 3 unnecessary PCs.
Let:

$$X_1 = U_{X_1}\Sigma_{X_1}V_{X_1}^t$$

$$X_2 = U_{X_2}\Sigma_{X_2}V_{X_2}^t$$

where

$$\Sigma_{X_1} = \begin{bmatrix} \sigma_1^1 \\ \\ \sigma_2^1 \\ \\ \sigma_3^1 \end{bmatrix} \quad \Sigma_{X_2} = \begin{bmatrix} \sigma_1^2 \\ \\ \sigma_2^2 \\ \\ \sigma_3^2 \end{bmatrix}$$

then:

$$\Sigma_{X_1} \otimes \Sigma_{X_2} = \begin{bmatrix} \sigma_1^1 \begin{bmatrix} \sigma_1^2 \\ \sigma_2^2 \\ \sigma_3^2 \end{bmatrix} \\ \sigma_2^1 \begin{bmatrix} \sigma_1^2 \\ \sigma_2^2 \\ \sigma_3^2 \end{bmatrix} \\ \sigma_3^1 \begin{bmatrix} \sigma_1^2 \\ \sigma_2^2 \\ \sigma_3^2 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \sigma_1^1\sigma_1^2 \\ \sigma_1^1\sigma_2^2 \\ \sigma_1^1\sigma_3^2 \\ \sigma_2^1\sigma_1^2 \\ \sigma_2^1\sigma_2^2 \\ \sigma_2^1\sigma_3^2 \\ \sigma_3^1\sigma_1^2 \\ \sigma_3^1\sigma_2^2 \\ \sigma_3^1\sigma_3^2 \end{bmatrix} = T1_{(\Sigma X_1 \otimes \Sigma X_2)}$$

The tensor product of $\Sigma_{X_1} \otimes \Sigma_{X_2}$ produces a total of nine PCs, as was expected. We can now choose the first six elements of $T1_{(\Sigma X_1 \otimes \Sigma X_2)}$. Therefore, we can conclude that we do indeed need these particular six PCs for the (2x2) tensor-metamodel. The last three PCs can be discarded. We can observe as well that $T1_{(\Sigma X_1 \otimes \Sigma X_2)}$ gives the structure/combination of these 6 PCs.
Using the same procedure we can find the right combination of PCs for the tensor product in the (3x3), (4x4) and higher order tensor-metamodels.

15 Testing the tensor-metamodel

15.1 Example 5: (2x2) S-system and its tensor-metamodel

The (2x2) tensor-metamodel is given as follows:

$$\begin{aligned} \dot{X}_1 &\approx \Gamma_1^1(\mathbf{p}_1^1(x) \otimes \mathbf{p}_1^2(x)) + ... + \Gamma_6^1(\mathbf{p}_2^1(x) \otimes \mathbf{p}_3^2(x)) \\ \dot{X}_2 &\approx \Gamma_1^2(\mathbf{p}_1^1(x) \otimes \mathbf{p}_1^2(x)) + ... + \Gamma_6^2(\mathbf{p}_2^1(x) \otimes \mathbf{p}_3^2(x)) \end{aligned} \tag{68}$$

where the right combination of PCs is given by the tensor product of $(\Sigma_{X_1} \otimes \Sigma_{X_2})$.
We are going to use (68) to solve the same system from Example 3.
Recall the S-system from Example 3:

$$\dot{X}_1 = 2X_1 0.5X_2 - 2X_1 X_2^{1.2}$$

$$\dot{X}_2 = 1.5X_1^{1.2} 3X_2 - X_1^{1.5} X_2$$

$x(0) = 0.75$
We get the following result:

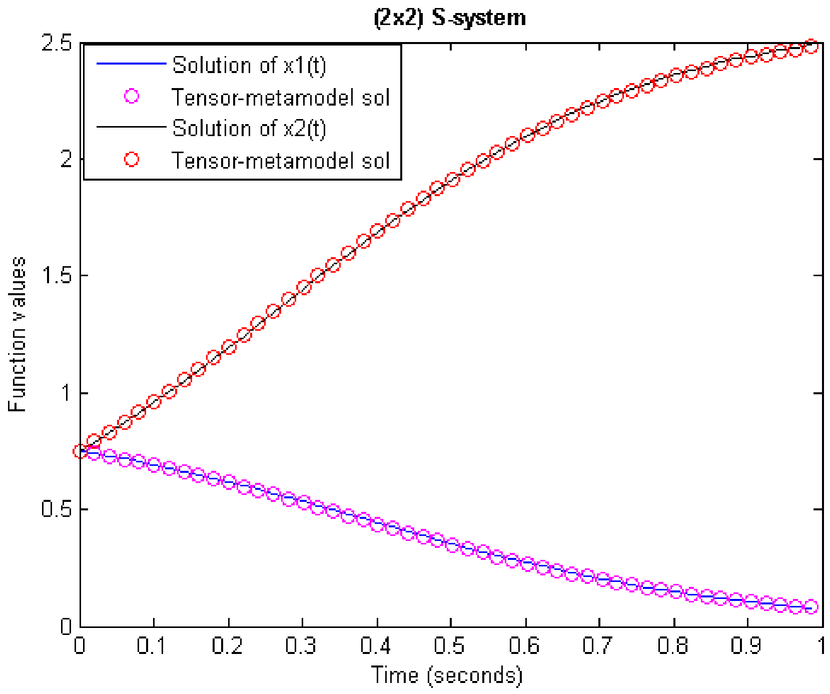

Figure 24

The *absolute error* between solutions is:

$$max(|x_1(t_k) - \widehat{x_1}(t_k)|^2) = 3.74e^{-5} \approx 0.003\%$$

and

$$max(|x_2(t_k) - \widehat{x_2}(t_k)|^2) = 4.64e^{-5} \approx 0.005\%$$

which are just as good as the results for the metamodel containing 9 PCs.
We get the same satisfactory results for different initial conditions.
Let $x(0) = 0.5$ and 1:

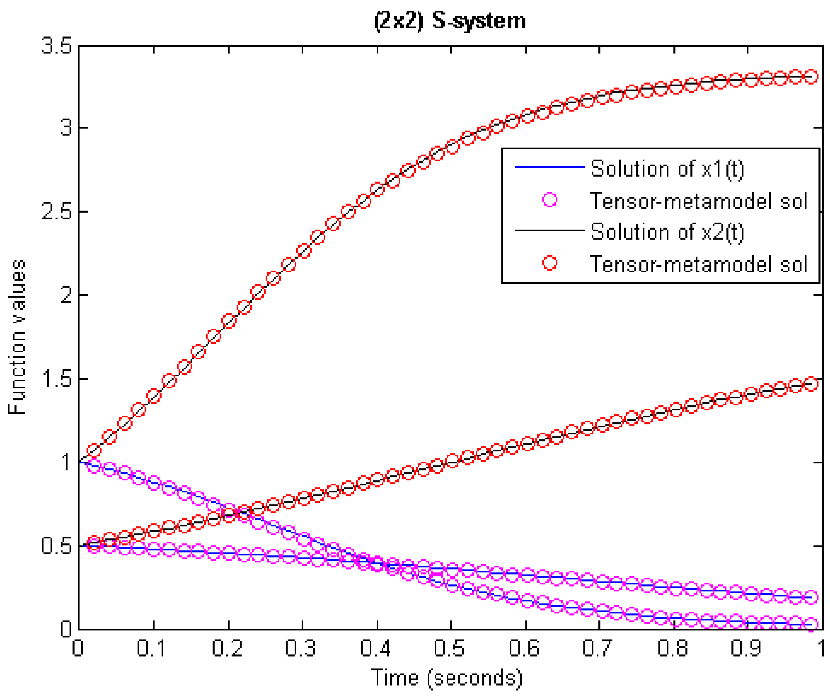

Figure 25

We can see clearly that the tensor-metamodel succeeds in producing excellent results.

15.2 Example 6: (3x3) S-system and its tensor-metamodel

Let us now consider the tensor metamodel for (3x3) S-systems. The tensor metamodel for (3x3) S-systems is given as follows:

$$\dot{X}_1 \approx \Gamma_1(\mathbf{p}_1^1(x_1) \otimes \mathbf{p}_1^2(x_2) \otimes \mathbf{p}_1^3(x_3)) + ... + \Gamma_{14}(\mathbf{p}_3^1(x_1) \otimes \mathbf{p}_3^2(x_2) \otimes \mathbf{p}_3^3(x_3))$$
$$\dot{X}_2 \approx \Gamma_1(\mathbf{p}_1^1(x_1) \otimes \mathbf{p}_1^2(x_2) \otimes \mathbf{p}_1^3(x_3)) + ... + \Gamma_{14}(\mathbf{p}_3^1(x_1) \otimes \mathbf{p}_3^2(x_2) \otimes \mathbf{p}_3^3(x_3))$$
$$\dot{X}_3 \approx \Gamma_1(\mathbf{p}_1^1(x_1) \otimes \mathbf{p}_1^2(x_2) \otimes \mathbf{p}_1^3(x_3)) + ... + \Gamma_{14}(\mathbf{p}_3^1(x_1) \otimes \mathbf{p}_3^2(x_2) \otimes \mathbf{p}_3^3(x_3))$$

We can now approximate the same system we had in Section 8.2 (55), as shown below:

$$\dot{X}_1 = X_1X_2X_3 - 2X_1X_2X_3$$
$$\dot{X}_2 = 1.5X_1X_2X_3 - X_1^{1.5}X_2X_3$$
$$\dot{X}_3 = 1.2X_1^{1.2}X_22X_3 - 0.8X_14.2X_2^{0.5}X_3$$
$$x(0) = 1.6$$

The next Figure presents the results.

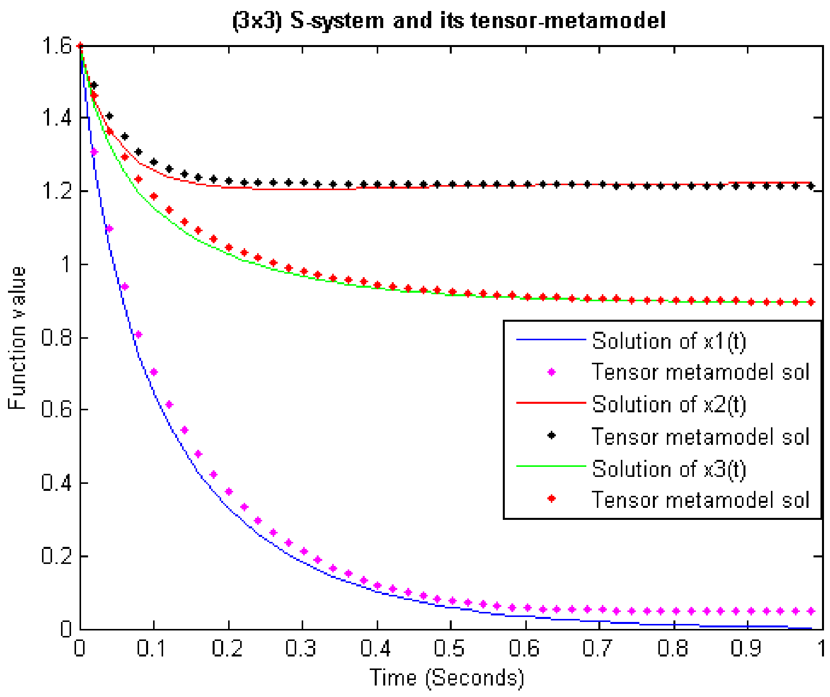

Figure 26

The absolute error for the approximation to the solutions is given as:

$max(|x_1(t_k) - \widehat{x_1}(t_k)|^2) \approx 0.3\%$
$max(|x_2(t_k) - \widehat{x_2}(t_k)|^2) \approx 0.1\%$
$max(|x_3(t_k) - \widehat{x_3}(t_k)|^2) \approx 0.1\%$

Finally let us consider the tensor-metamodel for the (4x4) S-system.

15.3 Example 7: (4x4) S-system and its tensor-metamodel

Consider:

$$\begin{aligned}\dot{X}_1 &= X_1X_2X_3 - 2X_1X_21.5X_4 \\ \dot{X}_2 &= 1.5X_1X_20.9X_3X_4 - X_1^{1.5}X_23X_3 \\ \dot{X}_3 &= 1.2X_1X_22X_3 - 0.8X_1X_2^{1.5}5X_4 \\ \dot{X}_4 &= 2X_1^{0.2} - X_4\end{aligned} \tag{69}$$

Solving this system and the tensor-metamodel for this system give us the following results:
let $x_o = 1$:

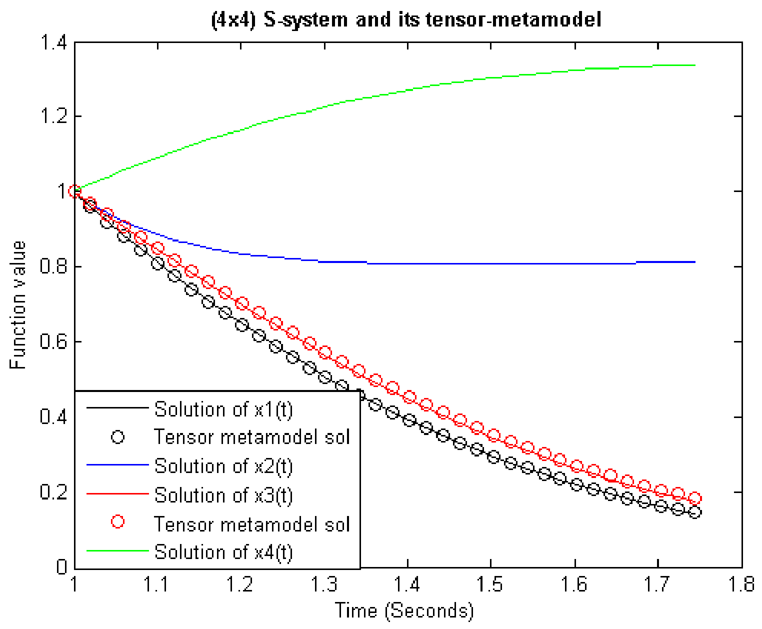

Figure 27

We observe the same result as those shown in Figure 24, which is the approximation to only two of the four solutions. The *square error* of the approximated solutions is given below:

$|x_1(t) - \widehat{x}_1(t)|^2 100 \approx 0.008\%$
$|x_3(t) - \widehat{x}_3(t)|^2 100 \approx 0.07\%$

These are very accurate results.

16 Applications of the tensor-metamodel

We have seen through examining theories and from looking at examples that the tensor-metamodel is successful in reaching the rather ambitious goal of approximate globally non-linear differential equations. The metamodel and its improved version, the tensor-metamodel, can be used in addition to approximate S-system differential equations, to find steady-state solutions or equilibrium points, in stability analysis, in extrapolation solutions of empirical data, recasting nonlinear differential equations and much more.
Regardless of the fact that this thesis consists in presenting the mathematical theory upon which the metamodel and tensor-metamodel is based, I will discuss two of the many applications mentioned above: steady state analysis and recasting.

16.1 Steady state analysis for the tensor-metamodel

Although this is only an example of one of many applications the tensor-metamodel is capable of, the results obtained have proven to be very good.

Steady state of S-system

We can read in [6] concerning how many important aspects of any given non-linear system can be analyzed far more easily when they are close to a steady state. These aspects are of great importance since most of the biochemical systems in nature operate close to a steady state, in which all in-flux is in equilibrium with all out-flux. We can also see in [6] the approach the author employs in order to find the values of X that will give us a steady state. Let us consider the same example the author uses in [6]:
Consider the following (2x2) S-system:

$$\begin{aligned} \dot{X}_1 &= 0.4X_2^{-2} - 2X_1 \\ \dot{X}_2 &= 2X_1 - X_2^{0.5} \end{aligned} \tag{70}$$

We are interested in the values of X_1, X_2 for which the system is in a steady state. This will imply that under these conditions, the system in (70) will not change with time. Hence $\dot{X}_1 = \dot{X}_2 = 0$, and this yields:

$$\begin{aligned} 0 &= 0.4X_2^{-2} - 2X_1 \\ 0 &= 2X_1 - X_2^{0.5} \end{aligned} \tag{71}$$

The equations from (71) can be simplified mathematically:

$$\begin{aligned} 0.4X_2^{-2} &= 2X_1 \\ 2X_1 &= X_2^{0.5} \end{aligned} \tag{72}$$

the next step is making the substitution of X_1, X_2 from the cartesian space to the logarithmic space. This is possible if we define:

$$y_1 = ln(X_1)$$
$$y_2 = ln(X_2)$$

then (72) becomes:

$$\begin{aligned} ln(0.4) - 2y_2 &= ln(2) + y_1 \\ ln(2) + y_1 &= 0.5y_2 \end{aligned} \tag{73}$$

which simplifies to:

$$y_1 = -ln(5) - 2y_2 \tag{74}$$
$$y_1 = -ln(2) + 0.5y_2 \tag{75}$$

we solve for y_2:

$$y_2 = \frac{ln(2) - ln(5)}{2.5} = -0.36652 \tag{76}$$

this gives $y_1 = -0.87641$
Finally we compute:

$$X_1 = exp(y_1) = 0.41$$
$$X_2 = exp(y_2) = 0.70$$

So, the substitution of X_1 and X_2 in (71) does indeed give us zero.

Steady state analysis of the tensor-metamodel

Let us first find a tensor-metamodel for the system in (70).
We choose $x(0) = 0.65$
After running the *matlab-code* at the given initial condition we obtain the following parameters:

$\Gamma_1 = 4.8 * 10^5$	$\Gamma_1 = 5.4 * 10^7$
$\Gamma_2 = 6.6 * 10^4$	$\Gamma_2 = 2.9 * 10^7$
$\Gamma_3 = 1.9 * 10^5$	$\Gamma_3 = 4.7 * 10^7$
$\Gamma_4 = 6.6 * 10^4$	$\Gamma_4 = 2.9 * 10^7$
$\Gamma_5 = -1.7 * 10^4$	$\Gamma_5 = -6.3 * 10^6$
$\Gamma_6 = 7.5 * 10^3$	$\Gamma_6 = 3.6 * 10^6$

The tensor-metamodel gives the following approximation to the solutions of the system from (70):

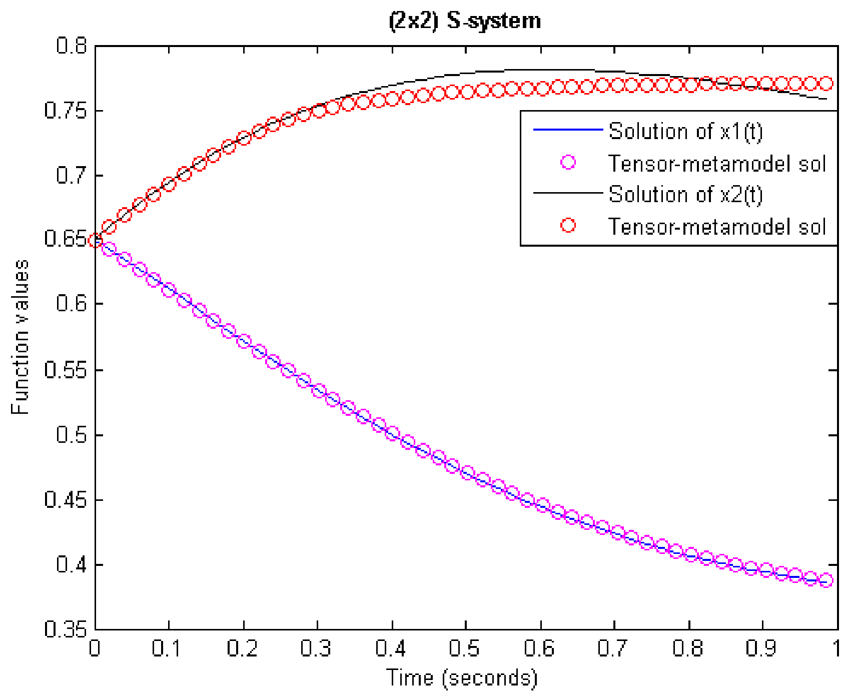

Figure 28: The *max square error* is: 0.00015% for the first solution and 0.021% for the second solution

Once we have solved the system, which is of course necessary in order to find the steady-state solution, we can find the x-value in (77) that gives us the approximation to the zero-value:

$$0 \approx \Gamma_1^1(\mathbf{p}_1^1(x) \otimes \mathbf{p}_1^2(x)) + ... + \Gamma_6^1(\mathbf{p}_2^1(x) \otimes \mathbf{p}_3^2(x)) \tag{77}$$

$$0 \approx \Gamma_1^2(\mathbf{p}_1^1(x) \otimes \mathbf{p}_1^2(x)) + ... + \Gamma_6^2(\mathbf{p}_2^1(x) \otimes \mathbf{p}_3^2(x))$$

Once this x-value is found, we store its index-position. Seeing as we are considering a (2x2) S-system, we are going to have two index-positions, as shown below:

$$\textit{index1} = 38$$

$$\textit{index2} = 98$$

and we return to:

$$x_j^{\omega_i}$$

Here we need to find the x_j-value which corresponds to the two index-positions we have recently discovered.
The last part consists in finding the real-values of these two index-positions in x_j, as follows:

$$\widehat{X}_1 = 0.37$$

$$\widehat{X}_2 = 0.78$$

We can now compare the results with the analytical solutions:

$$\widehat{X}_1 = 0.37 \quad vs. \quad X_1 = 0.41$$

$$\widehat{X}_2 = 0.78 \quad vs. \quad X_2 = 0.70$$

Finally we can compute:

$$\dot{X}_1 = 0.4\widehat{X}_2^{-2} - 2\widehat{X}_1$$

$$\dot{X}_2 = 2\widehat{X}_1 - \widehat{X}_2^{0.5}$$

which gives:

$$0 \approx 0.08 = 0.4\widehat{X}_2^{-2} - 2\widehat{X}_1$$

$$0 \approx 0.01 = 2\widehat{X}_1 - \widehat{X}_2^{0.5}$$

16.2 Recasting

Recasting nonlinear differential equations as S-system.

The theory of recasting nonlinear differential equations as S-system was introduced by *Michael A. Savageau*[7] and *Eberhard O. Voit* in a scientific paper published in 1987, see [7].
Recasting is a mathematical process in which we can reconstruct any given nonlinear ordinary differential equation as S-system differential equations. In [7] we can observe that the original system of differential equations is equivalent to the recasted S-system, meaning that both systems, the original and the recasted system, behave equally under steady state analysis. This is a remarkable breakthrough for the theory of S-system.
Accordingly to *Savageau* and *Voit*:

THEOREM:
Let:

$$\dot{Z} = f_i(Z_1,\ Z_2,\ ...,\ Z_n)\ where\ Z_i(0) = Z_{i0}\ for\ i = 1,2,3,...,n \tag{78}$$

be a set of differential equations where each f_i is composed of sums and products of elementary functions, or nested elementary functions of elementary functions.
Then there is a smooth change of variables that recast equations (78) into an S-system as follows:

$$\dot{X}_i = \alpha_i \prod_{j=1}^{m} X_j^{g_{ij}} - \beta \prod_{j=1}^{m} X_j^{h_{ij}},\ X_i(0) = X_{i0}\ for\ i = 1,2,3,...,(m) \tag{79}$$

[7]Michael A. Savageau is a Distinguished Professor in Biomedical Engineering at the University of California.

There are n-m constrains that are generated in this recasting process, they occur as:

$$\phi_k(X_1, X_2, ..., X_m) = 0 \tag{80}$$

the ϕ_k are elementary functions or nested elementary functions of elementary functions, this is the fist constrain of the recasting precess. The second constrain is between new and old variables that take the form:

$$X_i X_j = Z_k$$

The X_i and X_j variables always occur together as a product in the logarithmic derivatives $[d(logX_i)/dt]$ of the recast system. The definition of the new variables, as part of the recasting process, generates the constraints, which are automatically incorporated into equations (79). (Michael A. Savageau & Eberhard O. Voit. Recasting Nonlinear Differential Equations as S-Systems: A Canonical Nonlinear Form. Mathematical Biosciences 87:83-115 (1987) page: 89-90).

The recasting algorithm is based mainly on three transformations:

* *Translation of variables to the positive orthant.*
* *Decomposition of composite functions.*
* *Reduction of sums and products of power-law functions.*

Consider this example:

$$\dot{Z} = exp[log(Z)^2] \;\; Z(0) = 2 \tag{81}$$

which is not an S-system. This equation, which is considered by [7], can be recast in two cycles of decomposition as follows. Rename Z as X_1, define the single composite factor (f_{111}) as a new variable X_2, and add the differential equation for X_2 to the existing set:

$$\begin{aligned} \dot{X}_1 &= X_2 & X_1(0) &= 2 \\ \dot{X}_2 &= 2X_1^{-1}X_2^2 \log(X_1) & X_2(0) &= exp[(log(2))^2] \end{aligned}$$

Then define a new variable X_3 as $log(X_1)$, the third fact of the last equation, f_{213}, which is the only factor that is not already a power-law function, and add the corresponding differential equation to the existing set:

$$\dot{X}_1 = X_2 \qquad X_1(0) = 2 \tag{82a}$$

$$\dot{X}_2 = 2X_1^{-1}X_2^2X_3 \qquad X_2(0) = exp[(log(2))^2] \tag{82b}$$

$$\dot{X}_3 = X_1^{-1}X_2 \qquad X_3(0) = log(2) \tag{82c}$$

The result is the canonical S-system form.

As we can see, the recasting procedure published by *Savageau & Voit* is not parametrized, meaning that any of the functions $\dot{Z} = f_i(Z_1,\ Z_2,\ ...,\ Z_n)$ *Savageau & Voit* considered had unknown parameters. On the other hand the tensor-metamodel is meant to be used in parameter estimation problems. To be able to build a bridge between metamodels and recast-models, we have to first develop a new theory of parametrized recast models. In as far as the author of this thesis has attempted to find something close to this theory, nothing has as yet been discovered.

Parametrized recast-tensor-metamodel

Prof. Stig W. Omholt[8] suggested to apply the idea of recasting combined with tensor-metamodelling to the general problem of parameter estimation for non-linear dynamical systems. I do not address this problem in my thesis in full, but I show via examples that this is a possible and that the ides seems to be have great potential.
In this (at the time of writing, still non-existing) theory of parametrized recast-models we must begin to consider generic systems of non-linear differential equations with parameters. Once this is done we can apply the three elementary transformations from the *recasting algorithm* provided by *Savageau & Voit*.
Consider for example:

$$\dot{Z} = exp[\omega(log(Z))^2] \ where\ \omega \in \mathbb{R} \qquad Z(0) = 2 \tag{83}$$

which is the same example *Savageau & Voit* presented in their paper. There is nonetheless a slight difference, in the form of an unknown parameter (ω). Let us now apply the three elementary transformations we find in *Savageau & Voit*'s paper.
We define:

$$\begin{aligned} X_1 &= Z \\ X_2 &= exp[log(Z)^2] \\ X_3 &= log(Z) \end{aligned} \tag{84}$$

this will imply:

$$\dot{X}_1 = X_2^{\omega} \qquad X_1(0) = 2 \tag{85a}$$
$$\dot{X}_2 = 2X_1^{-1}X_2^2X_3 \qquad X_2(0) = exp[(log(2))^2] \tag{85b}$$
$$\dot{X}_3 = X_1^{-1}X_2 \qquad X_3(0) = log(2) \tag{85c}$$

[8]Prof. Stig W. Omholt is a distinguished professor at the Norwegian University of Science and Technology (NTNU)

We have now recast the equation (83) as a (3x3) S-system where the first equation contains an unknown parameter, namely ω.
If we recall the theory from metamodeling, we now have:

$$X_2^{\omega} \approx \sum_{i=1}^{3} t_i p_i^t \tag{86}$$

this implies:

$$\dot{X}_1 = X_2^{\omega} \approx \Gamma_1 \mathbf{p}_1(x_2) + \Gamma_2 \mathbf{p}_2(x_2) + \Gamma_3 \mathbf{p}_3(x_2) \tag{87}$$

An important detail to notice is that the domain for (x_2) in (87) depends on the domain of (x_1), this is because $X_2 = exp[log(X_1)^2]$, see (84).
Assume now that the parameter ω is a known real-value $\omega_s = 1.5$, we can run the implemented matlab-algorithm for the tensor-metamodel and find the three real-valued Γ parameters:

$$X_2^{\omega_s=1.5} \approx \widehat{\Gamma}_1 \mathbf{p}_1(x_2) + \widehat{\Gamma}_2 \mathbf{p}_2(x_2) + \widehat{\Gamma}_3 \mathbf{p}_3(x_2) \tag{88}$$

where $\widehat{\Gamma}_i$, $(i = 1, 2, 3)$ are known values.
The challenge now will consist in finding the correspondence or mapping between $\omega_s = 1.5$ and the triple-set $\widehat{\Gamma}_1$, $\widehat{\Gamma}_2$ and $\widehat{\Gamma}_3$. If we succeed in this step, the results will imply that the tensor-metamodel can also be used in parameter estimation problems of generic parametrized recast-models. This means, in other words, that if we have given a system of ODEs, we can first recast the system to S-system. This recasted system will contain one or more unknown parameters, the next step will be to find the corresponding tensor-metamodel that approximates the recasted system and by employing a mapping from the triple-set $\widehat{\Gamma}_1$, $\widehat{\Gamma}_2$ and $\widehat{\Gamma}_3$ to ω_i, where $\omega_i \Leftarrow x_j^{\omega_i}$, we are able to reconstruct the original S-system.
The reason for wanting to find a mapping from $\widehat{\Gamma}_i$, $i = 1, 2, 3$ to ω_i is closely related to Section 4.3. In Section 4.3 we have presented how the *score-matrix* can be used as a library L:

$$L = [\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3]$$

Equation (17) shows that it is possible to use specific triple from L in order to approximate a specific non-linear function $(x^{1.5})$. We have shown as well that the values of L come from the index-position ω_s has in ω_i, this is because $\omega_s \in \omega_i$:

$$\omega_i \in [-1, \omega_2, \omega_3, ..., \underbrace{1.5}_{\omega_s = position\ 501}, ..., 2]$$

A natural implication of these results will imply that the $\widehat{\Gamma}_1$, $\widehat{\Gamma}_2$ and $\widehat{\Gamma}_3$ parameters, which are found by means of the last square method, are values that are equal or close to a specific triple-set from L. This assumption is made because:

$$X_2^{\omega_s=1.5} \approx \widehat{\Gamma}_1\mathbf{p}_1(x_2) + \widehat{\Gamma}_2\mathbf{p}_2(x_2) + \widehat{\Gamma}_3\mathbf{p}_3(x_2) \tag{89}$$

and

$$X_2^{\omega_s=1.5} \approx L(501,1)\mathbf{p}_1(x_2) + L(501,2)\mathbf{p}_2(x_2) + L(501,3)\mathbf{p}_3(x_2) \tag{90}$$

which will imply:

$$\begin{aligned} L(501,1) &\approx \widehat{\Gamma}_1 \\ L(501,2) &\approx \widehat{\Gamma}_2 \\ L(501,3) &\approx \widehat{\Gamma}_3 \end{aligned} \tag{91}$$

Unfortunately this is not the case. After many computational trials we could not find any logical correspondence between the triple-set from L and the $\widehat{\Gamma}_i$, $i = 1, 2, 3$ parameters despite both sets giving the same solution to the system. The reason is not discussed in this thesis but this problem could be due to over-parametrization. Nevertheless, this is an issue that is worthy exploring further.

Example1

Let:

$$\dot{Z} = exp[(1.5)(log(Z))^2] \qquad Z(0) = 2 \tag{92}$$

this implies $\omega_s = 1.5$

We defined:

$$\begin{aligned} X_1 &= Z \\ X_2 &= exp[log(Z)^2] \\ X_3 &= log(Z) \end{aligned}$$

Furthermore:

$$\dot{X}_1 = X_2^{\omega_s=1.5} \qquad X_1(0) = 2 \tag{93a}$$

$$\dot{X}_2 = 2X_1^{-1}X_2^2\log(X_1) \qquad X_2(0) = exp[(log(2))^2] \tag{93b}$$

$$\dot{X}_3 = X_1^{-1}X_2 \qquad X_3(0) = log(2) \tag{93c}$$

We know from Section 4.3 that the index-position (in ω_i) which corresponds to the value of $\omega_s = 1.5$ is in the position 501.

Thus, we have the following triple-set:

$$\begin{aligned} L(501,1) &= -8.8 * 10^5 \\ L(501,2) &= -1.1 * 10^5 \\ L(501,3) &= -1.5 * 10^4 \end{aligned} \tag{94}$$

which is the same as that of Section 4.3
If we now compute the approximation to the exact function, where the exact function is given as:

$$\dot{X}_1 = X_2^{(\omega_s=1.5)} \qquad (95)$$
$$where\ X_2 = exp((log(x_1))^2)$$

meaning that we get the following ODE:

$$\dot{X}_1 = (-8.8 * 10^5)\mathbf{p}_1(x_2) + (-1.1 * 10^5)\mathbf{p}_2(x_2) + (-1.5 * 10^4) * \mathbf{p}_3(x_2) \quad (96)$$

If we now solve the ODEs from (95) and (96) at $x(0) = 1$ and compare their solutions, we can conclude that the *max-square error* is:

$$max|x(t) - \widehat{x}(t)| \approx 0.5\%$$

If we now used the same matrix $X_{(600x600)}$ employed in producing the library L and we apply the *last square method* to the tensor-metamodel, we find the following Γ parameters:

$$\begin{aligned} \widehat{\Gamma}_1 &= -2.1 * 10^{14} \\ \widehat{\Gamma}_2 &= 0.1 * 10^{14} \\ \widehat{\Gamma}_3 &= -0.6 * 10^{11} \end{aligned} \qquad (97)$$

These values are completely different from the triple-set we found in L. We then have the following tensor-metamodel:

$$X_2^{\omega_s=1.5} \approx \widehat{\Gamma}_1\mathbf{p}_1(x_2) + \widehat{\Gamma}_2\mathbf{p}_2(x_2) + \widehat{\Gamma}_3\mathbf{p}_3(x_2)$$

The function $\dot{X}_1$ is the same as before. Solving these two ODEs at $x(0) = 1$ and comparing their solutions gives:

$$max|x(t) - \widehat{x}(t)| \approx 0.7\%$$

which is also a good result.

Example 2

Assuming that we have a more complex ODE than the one from example 1:

$$\dot{Z} = \alpha Z^g - \beta exp[(h)(log(Z))^2] \qquad Z(0) = 2 \qquad (98)$$

Here er $\alpha, \beta\ \in \mathbb{R}$ and $g,\ h\ \in [-1, 2]$
Let:

$$\begin{aligned} X_1 &= Z \\ X_2 &= exp[log(Z)^2] \\ X_3 &= log(Z) \end{aligned}$$

The recasting to a canonical form will give:

$$\begin{aligned}
\dot{X}_1 &= \alpha X_1^g - \beta X_2^h & X_1(0) &= 2 \\
\dot{X}_2 &= 2X_1^{-1}X_2^2 \log(X_1) & X_2(0) &= exp[(log(2))^2] \\
\dot{X}_3 &= X_1^{-1}X_2 & X_3(0) &= log(2)
\end{aligned} \tag{99a}$$

The first equation is an S-system with unknown parameters, and we can solve this equation by means of the tensor-metamodel. We must, however, be more careful with these equations, due to the fact that x_1 and x_2 will have different domains.
Consider:

$$\begin{aligned}
X_1 &= \alpha[\Gamma_1^1\mathbf{p}_1^1(x_1) + \Gamma_2^1\mathbf{p}_2^1(x_1) + \Gamma_3^1\mathbf{p}_3^1(x_1)] \\
X_2 &= \beta[\Gamma_1^2\mathbf{p}_1^2(x_2) + \Gamma_2^2\mathbf{p}_2^2(x_2) + \Gamma_3^2\mathbf{p}_3^2(x_2)]
\end{aligned} \tag{100}$$

So the metamodel is given as:

$$X_1^g - X_2^h \approx$$

$$\alpha[\Gamma_1^1\mathbf{p}_1^1(x_1) + \Gamma_2^1\mathbf{p}_2^1(x_1) + \Gamma_3^1\mathbf{p}_3^1(x_1)] - \beta[\Gamma_1^2\mathbf{p}_1^2(x_2) + \Gamma_2^2\mathbf{p}_2^2(x_2) + \Gamma_3^2\mathbf{p}_3^2(x_2)]$$

Assume now that $\alpha = \beta = 1$ and $g = 0.5,\ h = 1.5$. This gives:

$$\dot{X}_1 = X_1^{0.5} - X_2^{1.5}$$

where

$$X_2 = exp(log(X_1)^2)$$

If we find the Γ parameters for the metamodel, we get the following values:

$$\begin{aligned}
\Gamma_1^1 &= 311.6285 & \Gamma_1^2 &= 1180 \\
\Gamma_2^1 &= -134.3277 & \Gamma_2^2 &= -606.2249 \\
\Gamma_3^1 &= 169.2220 & \Gamma_3^2 &= 223.5313
\end{aligned} \tag{101}$$

If we solve both ODEs at $x(0) = 1.5$ we reach the conclusion that the metamodel approximates the exact solution with an accuracy of:

$$(1 - max|x(t) - \widehat{x}(t)|)\ 100 \approx 98.9\%$$

On the other hand, if we want to use the values from the library L, we are going to need two independent libraries. The first one corresponding to $X_1^{0.5}$ and the second one corresponding $X_2^{1.5}$.
This gives the following expression:

$$\alpha X_1^g - \beta X_2^h \approx \alpha \sum_{i=1}^{3} t_i^1 p_i^{1t} - \beta \sum_{i=1}^{3} t_i^2 p_i^{2t} \tag{102}$$

where

$$\begin{aligned} t_1^1 &= -33.7908 & t_1^2 &= -65.2256 \\ t_2^1 &= -5.7801 & t_2^2 &= 3.4524 \\ t_3^1 &= 0.5367 & t_3^2 &= 0.2610 \end{aligned} \tag{103}$$

Finally we solve both ODEs at $x(0) = 1$ and we are therefore able to conclude that the metamodel approximates the exact solution with an accuracy of:

$$(1 - max|x(t) - \widehat{x}(t)|)\ 100 \approx 99.3\%$$

17 Conclusions

From a technical point of view, the greatest challenge in writing this thesis was to connect metamodelling to non-linear systems appearing in biochemistry to in some way pay homage to the *Law of Mass Actions* discovered 150 years ago by the Norwegian scientists: C. M. Gulberg and P. Waage. This challenge has been met and addressed in this thesis.
In this thesis, I have presented a new approach for mathematical modelling of biochemical reaction networks represented by S-systems applying a modern method used by scientists at NMBU, namely metamodelling.
More specifically, I have, by finding latent variables in these non-linear dynamical systems, compressed S-systems by means of a bi-linear metamodel and its improved version, a tensor-metamodel. Both methods have then been brought together to form the principal component transform (PCT).
I have shown that each power-function representation in:

$$\dot{X}_i = \alpha_i \prod_{j=1}^{n} X_j^{g_{ij}} - \beta_i \prod_{j=1}^{n} X_j^{h_{ij}}$$

can be approximated by three PCs, giving for a common x-domain an approximation to the S-system.

$$\dot{X}_i \approx \Gamma_1 \mathbf{p}_1^t + \Gamma_2 \mathbf{p}_2^t + \Gamma_3 \mathbf{p}_3^t$$

$\mathbf{p}^t$ is $\mathbf{p}$ transposed, the three PCs are the latent variables and Γ_1, Γ_2 and Γ_3 are unknown parameters which can be found by the linear regression method.
I have presented in sections 8 and 10 a number of numerical experiments where (1x1), (2x2), (3x3) and (4x4) S-system have being modelled by the bi-linear metamodel, and where the results are extremely accurate.
In section 11 it was established that the bi-linear metamodel produces, for big S-systems, PCs which span a *null space*. This issue is solved in the tensor-metamodel, where, by applying the *Kroneker* product to the singular values

of the matrix X, and computing the PCA to it, we are able to significantly reduce the dimensionality of the tensor-metamodel-system, (see section 13). Section 15 presents numerical examples where (1x1), (2x2), (3x3) and (4x4) S-systems are modelled by the tensor-metamodel, also giving very accurate results.
Finally, in section 16 we name some of the applications for the PCT, at the same time providing some examples where we find the steady-state of a (2x2) tensor-metamodel-system (section 16.1) and we show how recasting of parametrized non-linear differential equations can be related to the PCT. We have attempt to use the *scores* belonging to the PCT as a *library* in order to use the PCT in parameter estimation problems. However, we have unfortunately not succeed in doing this. This could be, according to [19], because the system has a sloppy structure, meaning that the system has a neutral parameter-set. However, in higher dimensions sloppiness seems to disappear, so that the *library* can be constructed in an efficient way.
This issues are to be investigated more thoroughly in further studies.

18 Appendix A1: The singular value decomposition

The singular value decomposition (SVD) is intimately related to the theory of diagonalization of symmetric matrices with one fundamental difference; the matrix **M** is not a square matrix (n x n) but a rectangular matrix (m x n); where $m < n$. Nevertheless, the singular value decomposition is based on a property from ordinary diagonalization that can be imitated by any rectangular matrix (D. Lay [4] s.487-497). Assume **M** is symmetric, then the absolute values of the eigenvalues of **M** measures the amount that **M** stretches or shrinks.
If $\mathbf{Mx} = \lambda\mathbf{x}$ and $||\mathbf{x}|| = 1$, then:

$$||\mathbf{Mx}|| = ||\lambda\mathbf{x}|| = |\lambda|||\mathbf{x}|| = |\lambda|$$

Consequently, if λ_1 is the greatest eigenvalue for **M**, then its corresponding eigenvector $\mathbf{v}_1$ is the responsible of maximizing $||\mathbf{Mx}||$.
Let now **M** be a (m x n) matrix; where $m < n$. Then $\mathbf{M}^t\mathbf{M}$ is a symmetric matrix (n x n) that can be orthogonally diagonalized (D. Lay [4] s.487-497). Let $\lambda_1, ..., \lambda_n$ be the eigenvalues of $\mathbf{M}^t\mathbf{M}$ with its corresponding eigenvectors $\mathbf{v}_1, ..., \mathbf{v}_n$. The singular values -$\sigma_i$- are the square roots of the eigenvalues of the matrix product $\mathbf{M}^t\mathbf{M}$, that is, $\sigma_i = \sqrt{\lambda_i}$ for $i = 1, .., n$ (D. Lay [4] s.487-497).
The singular values -σ_i- are the length of $\mathbf{Mv}_1, ..., \mathbf{Mv}_n$ and they are per definition all ways > 0, because:

$$||\mathbf{Mv}_i||^2 = (\mathbf{Mv}_i)^t\mathbf{Mv}_i = \mathbf{v}_i^t\mathbf{M}^t\mathbf{Mv}_i = \mathbf{v}_i^t(\lambda_i\mathbf{v}_i) = \lambda_i$$

There are two matrices: **U** and **V**, which are orthogonal and a diagonal matrix Σ meaning that $\mathbf{M}_{mxn}$ can be factorized as follows: $\mathbf{M} = \mathbf{U}\Sigma\mathbf{V}^t$.
Assume we have a matrix $\mathbf{M}_{mxn}$ with rank r. The *"diagonal"* matrix Σ is given as:

$$\Sigma_{mxn} = \begin{bmatrix} \mathbf{D} & 0 \\ 0 & 0 \end{bmatrix}$$

where **D** is:

$$\mathbf{D} = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & ... & 0 \\ 0 & 0 & \sigma_n \end{bmatrix}$$

This means that for the matrix Σ: the $m - r$ rows and the $n - r$ columns will all be zeros and **D** (which is now diagonal) will be given by the first r singular values of **M** [4] –implying that Σ_{nxn}–.
The orthogonal matrix $\mathbf{U}_{mxm}$ is given as:

$$\mathbf{U} = \begin{bmatrix} \mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, ..., \mathbf{u}_m \end{bmatrix}$$

Where:

$$\mathbf{u}_i = \frac{1}{||\mathbf{Mv}_i||}\mathbf{Mv}_i = \frac{1}{\sigma_i}\mathbf{Mv}_i, \ (i = 1, ..m)$$

The columns of $\mathbf{U}$ are called left singular vectors, and from them we obtain an orthonormal basis $\{\mathbf{u}_1, ..., \mathbf{u}_m\} \in \mathbb{R}^m$.
Finally the matrix $\mathbf{V}_{(nxn)}$ is given as shown previously by the orthonormal vectors $\mathbf{v}_i$ where $i = 1, ..., n$ which are the eigenvectors of $\mathbf{M}^t\mathbf{M}$.
This means that:

$$\mathbf{M}_{(mxn)} = \begin{bmatrix}\mathbf{u}_1, ..., \mathbf{u}_m\end{bmatrix}_{(mxm)} \begin{bmatrix}\sigma_1 & 0 & 0 \\ 0 & \sigma_n & 0 \\ 0 & 0 & 0\end{bmatrix}_{(mxn)} \begin{bmatrix}\mathbf{v}_1, ..., \mathbf{v}_n\end{bmatrix}^t_{(nxn)}$$

(D. Lay [4] s.487-497)

19 Appendix A2: The Law of Mass Action

Peter Waage (June 29 1833 - January 13 1900), was a significant Norwegian chemist and professor at the Royal Frederick University. Along with his brother in law, Cato Maximilian Guldberg (11 August 1836 - 14 January 1902), they co-discovered and developed the Law of Mass Action between 1864 and 1879. Guldberg og Waage published three articles, in 1864, 1867 and 1879. Their first paper in Norwegian "Studier over Affiniteten" went largely unnoticed, as did their second paper in French (1867). Almost fifteen years after the first paper, in 1877 van 't Hoff came to similar conclusions but as his work had been conducted independently from Guldberg and Waagw's work, the Norwegian scientists decided to write and publish their last paper (1879) in order to get credit for their work. (Store Norske Leksikon; translated by Sergio Haram Sarmiento).
I will not even attempt to explain the law of Mass Action in my own words, because I am afraid of ruining the magnificent work produced by these two brilliant scientist. On the contrary, I believe that the best way to present Guldberg & Waage's work is by simply presenting their work as it appeared in it's original form:

"Støttebde os dels til tidligere af andre Kemikere udførte Forsøg og dels til vore egne og ledet af den ovenfor udviklede Gang i de kemiske Processer, fremsætte vi følgende to Love, nemlig Loven om Massernes Virkning og Loven om Volumets Virkning, hvoraf da udledes Ligevægtsbetingelsen for de i Systemet virkende Kræfter.

1.

Massernes Virkning

Substitutionskraften er under forøvrigt samme Forholde direkte proportional med Produktet af Maserne, efterat hver er ophøjet i en bestemt Exponent. Betegnes Mængderne af de to Stoffer, der idirekte paa hinanden med M og N, saa er Substitutionskraften for disse

$$\alpha(M^a N^b)$$

Koefficienterne α, a og b ere Konstanter der under forøvigt samme Forholde alene afheænge af Stoffernes Natur.

2.

Volumets Virkning

Naar de samme Masser af de indvirkende Stoffer befinde sig under forskjellige Volumina, da ere disse Massers Virkning omvendt proportional med Volumet. Betegner som ovenfor M og N Mængdem af de to Stoffer og er det samlede Volum af Systemet i to forskjellige Tilfælde V og V' saa er Substitutionskraften i det ene Tilfælde udtrykt ved

$$\alpha(\frac{M}{V})^a(\frac{N}{V})^b$$

og i det andet ved

$$\alpha(\frac{M}{V'})^a(\frac{N}{V'})^b$$

(Guldberg & Waage [14] s. 5-6)

Relying in part on earlier experiments carried out by other chemists and in part on our own experiments and guided by the development in chemical processes named above, we present two laws, namely the **law of mass action** and **the law of volume action**, from which the **equilibrium condition** is derived for the forces acting in the system.

1 Action of Mass

The substitution force is directly proportional to the product of the masses when each is raised to a particular exponent. Designating each of the quantities of the two substances by M and N, then the substitution force for these are given by:

$$\alpha(M^a N^b)$$

The coefficients α, a and b are constants depending only on the nature of the substances.

2 The Action of Volume

The action of the masses is inversely proportional to the volume if the same masses of the substances occur in different volumes. If we were to design as shown in the example above, M and N as the amount of the two substances and V and V' as the total volume of the system in two different cases, then is the substitution force in the first and second case would be expressed as:

$$\alpha(\frac{M}{V'})^a(\frac{N}{V'})^b$$

(Guldberg & Waage [14] s. 5-6. Translated by Sergio Haram Sarmiento.)

20 Appendix A3: Matlab-codes

In order compress the size of this section, I have not included all the *matlab* codes I have implemented.

20.1 Note about matlab codes for the bi-linear metamodel and tensor-metamodel

The matlab codes for the bi-linear metamodel and tensor metamodel for (1x1), (2x2), (3x3) and (4x4) S-system are not included because of each code's large size. I can nevertheless confirm that I have all these codes and I can vouch for the fact that the *matlab* codes applied in this thesis have been implemented by the author of this thesis.

20.2 Code for a multivatiave-power-function Matrix

```
%%% function [X, Xc] computes a multivariate-power-function matrix X %%%
%%% Description: %%%
%%% The Matrix X is composed of porwer-functions where %%%
%%% each entry is a power function given as a combination of %%%
%%% the rows and colummns %%%
%%% Xc is the centered matrix X, if this value is not needed %%%
%%% Xc can be replaced with ~ %%%

%%% By: Sergio Haram Sarmiento %%%

function [X, Xc] = matrix(rows,columns)
r = (0:1/rows:2); % r = rows, (R gives the alpha-domain)
c = (-1:2/columns:2); % c = columns (c gives the x_j-doamin)
r(1)=[]; % discard the zero value
c(1)=[]; % ommit the zero value
x = [ones(length(r), 1) * c];
for i = 1:length(r)
    X(i,:) = x(i,:).^r(i);
end
Xc = X-ones(size(X,1),1)*mean(X);
end
```

20.3 Computing the PCA

```
%%% Computes the principal component analysis (PCA) %%%
%%% by means of the singular value decomposition (SVD) %%%

%%% By: Sergio Haram Sarmiento %%%

function [SCR, LDS, VAR] = pca(X)
[U, S, V] = svd(X, 'econ');
var = diag(S);
VAR = var.^2; % Returns the variance each PC carries
SCR = U*S; % Returns the scores of X
LDS = V; % Returns the loadings of X
end
```

20.4 Tensor-singular value decomposition for finding the right combination of PCs for the tensor-metamodel

```
%%% The code computes the 4x4-tensor product of the singular values %%%
%%% of the matrix X. X is a multivariate-power-function matrix %%%
%%% cum returns the cumulative value for the variance each PC carries %%%
%%% In many cases it will not be needed to print more than the first %%%
%%% 50 values from cum. This can be done by computing %%%
%%% >> cum(50,:); on the command-window %%%

%%% By: Sergio Haram Sarmiento %%%

[X,~] = matrix(size_of_rows , size_of_columns);
[~,s,~] = svd(X);
var = diag(s);
t2 = kron(var,var);
t3 = kron(t2,var);
t4 = kron(t3,var);
VAR1 = sort(t4);
VAR2 = flipud(VAR1);
VAR = VAR2.^2;
andel = VAR/sum(VAR);
cum = 100*cumsum(andel);
```

References

[1] Julia Isaeva, *Multivariate Analysis as a Tool for Understanding and Reducing Complexity of Mathematical Models in Systems Biology.* Philosophiae Doctor (PhD) Thesis. Dept. of Chemistry, Biotechnology and Food Science Norwegian University of Life Sciences 2011; 1-9, 71-90.

[2] Harald Martens and Magni Marten, *Multivariate Analysis of Quality, An Introduction* John Wiley & Sons Ltd. 2001; 9-20, 93-94.

[3] Richard A. Johnson & Dean W. Wichern, *Applied Multivariate Statistical Analysis.* Pearson International Edition, sixth edition 2007; 430-431.

[4] David C. Lay. *Linear Algebra and Its Applications.* Pearson international Edition 2006; 487-497.

[5] Eberhard O Voit *Computational Analysis of Biochemical Systems. A Practical Guide for Biochemists and Molecular Biologists.* Cambridge University Press 2000; 37-41, 191-218.

[6] Eberhard O Voit *Canonical Nonlinear modeling. S-Systems Approach to Understanding Complexity* Published by Van Nostrand Reinhold 1991; 47-50;

[7] Michael A. Savageau and Eberhard O. Voit, *Recasting Nonlinear Differential Equations as S-Systems: A canonical nonlinear form differential equations and dynamical systems.* Department of microbiology and immunology, The University of Michigan 1987. Received 6 May 1987; revised 28 July 1987.

[8] Kohler and Johnson *Elementary Differential Equations, with Boundary Value Problems.* Pearson and Addison Wesley, second edition 2006; 391-458

[9] V. I. Arnold *Ordinary Differential Equations* The Massachusetts Institute of Technology 1973; 111-112

[10] Alan J. Laub *Matrix Analysis for Scientists and Engineers* Society for Industrial and Applied Mathematics 2004; 139-150

[11] Howard B. Wilson, Louis H. Turcotte, David Halpern *Advanced mathematics and mechanics applications using matlab* Chapman & Hall/CRC, third edition 2003; : 273

[12] Aslak Tveito, Hans Petter Langtangen, Bjørn Frederik Nielsen, Xing Cai *Elements of Scientific Computing.* Springer 2010; 31-69.

[13] Steven C. Chapra *Applied Numerical Methods whit Matlab for Engineers and Scientists* Mc Graw-Hill, third edition 2012; 553-583

[14] Cato Maximilian Guldberg and Peter Waage *Studier over Affiniteten I, II, III* Videnskapsselskap forhandlinger for 1864, Christiania 1865, og 'Om den chemiske Affinitet' Vdenskapsselskap forhandlinger for 1879, nr

[15] John wyller *Notes in Dynamical Systems.* Notes for the Course MATH301 at the Norwegian University of Life Sciences.

[16] Arkadi Ponossov *Lectures in Dynamical Systems* Lectures for the Course MATH301 at the Norwegian University of Life Science 2013.

[17] John wyller *Notes in Dynamical Systems.* Notes for the Course MATH301 at the Norwegian University of Life Sciences.

[18] Arkadi Ponossov *Lectures in Dynamical Systems* Lectures for the Course MATH301 at the Norwegian University of Life Sciences 2013.

[19] Valeriya Tafintseva, Philosophiae Doctor (PhD) Thesis. Dept. of Mathematical Science and Technology. Norwegian University of Life Sciences 2013; 83-109.