BACHELOR

Multiplicative principal component analysis and a comparison with the standard case

van Roosmalen, J.

Award date:
2012

Multiplicative Principal Component Analysis

and a Comparison with the Standard Case

Jarno van Roosmalen (0587578)

22nd April 2012

Abstract

In this report the theory of multiplicative calculus is investigated. This calculus is based on the substitution of multiplication for the role of addition in normal calculus. It is concluded that this gives a powerful theoretical framework with most of the tools from normal calculus. Application of these principles on the method of Principal Component Analysis gives a new tool. it turns out from the conducted tests, that in certain situations this new method can be beneficiary.

Contents

Chapter 1

Introduction

The idea behind this project is to investigate the theory and application of multiplicative calculus in particular, and multiplicative algorithms in general. The concepts of multiplicative calculus were first raised by Volterra in 1887 [21]. In the following decades it seems to be forgotten by the mathematical community. Over the years a few papers have surfaced which discuss the topic again. In recent years the interest in the topic has grown again as there seem to be applications of multiplicative calculus in for example image analysis. In this report the following approach is followed.

First an overview is given of multiplicative calculus, based on the current literature. Starting from definitions and ending with some powerful theorems. Also included is a list of calculation rules, and examples of multiplicative derivatives and integrals.

This is followed by an overview of the algorithm behind Principal Component Analysis (PCA). This will serve as an example to be generalised to a multiplicative interpretation. A multiplicative variant of PCA is introduced and discussed.

The PCA is implemented in MATLAB, and in chapter 3.2, the results are shown of multiple tests that were conducted. Tests will conducted with completely random data, and with a parameterised class of functions.

Chapter 2

Multiplicative Calculus

In this chapter an overview of the theory of multiplicative calculus will be given. . This chapter is based on [18, 17, 3, 4, 19, 9]

2.1 Background

The key idea behind multiplicative calculus dates back to Volterra in 1887 [21]. In normal calculus a lot of concepts (e.g. linearity, derivatives, etc.) are based on addition and subtraction. The idea of multiplicative calculus is to substitute those operations by respectively multiplication and division. Normal scalar multiplication then becomes exponentiation. The resulting calculus doesn't focus on linear growth but multiplicative growth. In normal calculus it is natural to consider the special linear function. Linear functions have a constant derivative and in each time unit the same amount of growth is added (like in a population growth). In the multiplicative case we will see that exponentials are the natural functions with a constant derivative. This means that each time unit gives a constant growth factor, for example interest on a bank account.

Note that in a normal product due to commutativity and symmetry there is no difference between the role of the factors in a product ax , with $a, x \in \mathbb{R}$. But how to create the exponentiation? there are obviously two possibilities, $ax \to x^a$ or $ax \to a^x$. To ease the generalisation later to non scalar functions, it is good to define the structure now.

Definition 2.1.1. *The multiplicative structure is defined using the following rules (cf. [9, sec 2.1]). Consider a vector space V of dimension n. For any $u, v, w \in V$ and $\lambda, \mu \in \mathbb{R}$:*

1. $(uv)w = u(vw)$,
2. *there exists an element* $1_V \in V$ *such that* $1_V u = u = u1_V$,
3. *there exists an element* $u^{-1} \in V$ *such that* $uu^{-1} = 1_V = u^{-1}u$,
4. $u^{\lambda\mu} = (u^\lambda)^\mu$,
5. $u^1 = u$.

Note that the unit element $1_V \in V$ is not $1 \in \mathbb{R}$ unless $V = \mathbb{R}$. There are no assumptions made on commutativity. If applicable one can add commutative rules: $uv = vu$ and $(uv)^\lambda = u^\lambda v^\lambda$.

2.2 Differentiation

The core of multiplicative calculus centrers around the new definition for a derivative. To ease the definitions we require for now the functions to be positive and differentiable in the *normal* sense.

Definition 2.2.1. *We look at continuously differentiable functions defined on a subset of a field F (e.g. $\mathbb{R}$ or $\mathbb{R}^n$) to the positive functions:*

$$V = \{f \in C^1(F, \mathbb{R}) | f(x) > 0, \quad \forall x \in F\} \tag{2.1}$$

where $C^1(F, \mathbb{R})$ are the continuously differentiable functions from F to $\mathbb{R}$.

The normal derivative is based on creating a tangent to the function. It looks at how much is added in a short interval. By taking the limit to of the interval size we get the instantaneous velocity. The usual definition is:

$$f'(x) = \frac{d}{dx} f(x) \equiv \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \quad \text{for } f \in V \tag{2.2}$$

In the multiplicative case we look at the ratio of the function value over a short interval. This gives a average growth factor. By taking again the limit we get now the instantaneous growth factor.

Definition 2.2.2. *The multiplicative derivative f^* is defined as:*

$$f^*(x) \equiv \lim_{h \to 0} \left(\frac{f(x+h)}{f(x)} \right)^{1/h} \qquad \textit{for } f \in V. \tag{2.3}$$

In the literature there are different notations used. The notation f^* for the multiplicative derivative of a function is widely used. The other notation is used by Spivey[17]. This somewhat usual notation might seem strange here, but it is analog to a notation used for the product integral later on. It easy to see that $f(x) > 0$ implies $f^*(x) > 0$.

One can easily show the connection between the multiplicative derivative and the normal derivative. By using exponentials one gets:

Theorem 2.2.1. *The multiplicative derivative f^* is related to the normal derivative $f'(x)$ by*

$$\ln f^*(x) = (\ln f)'(x) \quad \textit{,for } f \in V \tag{2.4}$$

or by taking the exponential

$$f^*(x) = \exp(\frac{d}{dx} \ln(f(x))) = e^{\ln(f(x))'} \quad \textit{,for } f \in V. \tag{2.5}$$

Proof. Suppose $f^*(x)$ exists then

$$\begin{aligned} \ln f^*(x) = & \ln \left(\lim_{h \to 0} \left(\tfrac{f(x+h)}{f(x)} \right)^{\frac{1}{h}} \right) \\ = & \lim_{h \to 0} \left(\ln \left(\tfrac{f(x+h)}{f(x)} \right)^{\frac{1}{h}} \right) \\ = & \lim_{h \to 0} \left(\tfrac{\ln(f(x+h)) - \ln(f(x))}{h} \right) \\ = & (\ln(f(x)))' \end{aligned}$$

The interchange of the limit and the logarithm is allowed as the logarithm is continuous.

Corollary. *It is easy to see that we now have*

$$f^*(x) = e^{f'(x)/f(x)} \tag{2.6}$$

This shows there is an easy transfer from the normal derivative to the multiplicative derivative. It also implies that if a function is differentiable and positive is is automatically also multiplicatively differentiable.

2.2.1 Rules for product differentiation

A lot of rules for calculating derivatives have an analogue version for multiplicative derivatives. The list given below is based on [18, 3, 17]. All statements can easily be verified using the definition.

1. $(cf)^*(x) = f^*(x)$, where $c \neq 0$ a constant,
2. $(fg)^*(x) = f^*(x)g^*(x)$, so the product rule is now trivial
3. $(f/g)^*(x) = f^*(x)/g^*(x)$,
4. $(f^h)^*(x) = f^*(x)^{h(x)} \cdot f(x)^{h'(x)}$
5. $(f \circ h)^*(x) = f^*(h(x))^{h'(x)}$
6. $(f+g)^*(x) = f^*(x)^{\frac{f(x)}{f(x)+g(x)}} \cdot g^*(x)^{\frac{g(x)}{f(x)+g(x)}}$

The different papers also generalise some other important basic calculus theorems. First we give a mean value theorem for product derivatives.

Theorem 2.2.2. *If $f \in V$ is continuous on $[a, b]$, multiplicative differentiable on (a, b) and $f(a) \neq 0 \neq f(b)$. then there exists some $c \in (a, b)$ such that*

$$f^*(c) = \left(\frac{f(b)}{f(a)}\right)^{1/(b-a)} \tag{2.7}$$

The r.h.s. is the average multiplicative change on the interval. So the theorem states that the multiplicative derivative is at least at one point equal to the average change.

2.2.2 Taylor Series & Products

It's well known in normal calculus that all smooth functions can be approximated by a Taylor series. The order k Taylor series around the point $a \in \mathbb{R}$ of a $k+1$ times differentiable function $f : \mathbb{R} \to \mathbb{R}$ is given by:

$$f(x) = \sum_{i=0}^{k} f^{(i)}(c)\frac{(x-c)^i}{i!} + f^{(k+1)}(\xi)\frac{(x-c)^{k+1}}{(k+1)!} \tag{2.8}$$

, for some number ξ between x and c.

In the multiplicative case the sum becomes a product but the the approximation is again possible[4].

Theorem 2.2.3. *Let A be a interval of $\mathbb{R}$, and $f : A \to \mathbb{R}^+$ a $(k+1)$ times multiplicatively differentiable function. Then the function can be approximated in point $c \in A$ by*

$$f(x) = \prod_{i=0}^{k} \left(f^{*(i)}(c) \right)^{\frac{(x-c)^i}{i!}} \cdot \left(\left(f^{*(k+1)}(\xi) \right)^{\frac{(x-c)^{k+1}}{(k+1)!}} \right) \tag{2.9}$$

for some number ξ between x and c.

The fact that a Taylor theorem exists for multiplicative calculus is very important. It means that one can approximate function by a product of basis functions instead of a sum. Depending on the function this might result in simpler approximations. For example the most simple approximation in the linear calculus is a linear approximation around a point $x = a$:

$$f(x) \approx f(a) + (x - a)f'(a) \tag{2.10}$$

For any function with a constant derivative (i.e. linear functions) the approximation is exact. In the multiplicative calculus we have the following analogue approximation around $x = a$:

$$f(x) \approx f(a)f^*(a)^{(x-a)}. \tag{2.11}$$

In this case the approximation is exact for functions with a constant multiplicative derivative, which are the exponential functions. This suggest that for related function this kind of approximations might be more suitable then long linear Taylor series.

2.2.3 Examples

Table 2.1: *A list of common functions and both their normal and their multiplicative derivatives and integrals. For integrals the integration constant has been omitted for readability.*

$f(x)$	$f'(x)$	$f^*(x)$	$\int f(x)dx$	$\prod f(x)^{dx}$
C	0	1	Cx	e^C
e^x	e^x	e	e^x	$e^{x^2/2}$
Ce^{ax}	Cae^{ax}	e^a	$\frac{C}{a}e^{ax}$	$C^x e^{ax^2/2}$
a^x	$\ln(a)a^x$	a	$\frac{a^x}{\ln(a)}$	$a^{x^2/2}$
ax	a	$e^{1/x}$	$\frac{a}{2}x^2$	$e^{-x}(ax)^x$
$ax + b$	a	$e^{\frac{a}{ax+b}}$		
ax^n	nax^{n-1}	$e^{n/x}$	$\frac{a}{n+1}x^{n+1}$	$e^{-nx}(ax^n)^x$
$\ln(ax)$	$\frac{1}{x}$	$e^{1/(x\ln(ax))}$		
$\sin(ax)$	$a\cos(ax)$	$e^{a\cos(ax)/\sin(ax)}$		
$\cos(ax)$	$-a\sin(ax)$	$e^{-a\sin(ax)/\cos(ax)}$		
$\tan(ax)$	$\frac{a}{\cos^2(ax)}$	$e^{\frac{2a}{\sin 2ax}}$		
$\sin^2(ax)$	$2a\cos(ax)\sin(ax)$	$e^{2a\cos(ax)/\sin(ax)}$		

In table 2.1 gives a list with both the normal and the multiplicative derivative for a set of common functions, cf. [18]. This can also be used for calculating the multiplicative derivative by hand. In the table the following symbols are used: $C, a, b, m \in \mathbb{R}$.

2.3 Multiplicative Integrals

There are in literature to definitions for multiplicative integrals or product integrals as they are commonly called.

The first possible definition is the natural one as the anti-derivative with respect to the multiplicative derivative.[18, 9, 17]

Definition 2.3.1. *Let $F \in V$ be a multiplicative differentiable function with $F^* = f$, and $c \in \mathbb{R}^+$ a constant. Then the multiplicative integral is defined by:*

$$\prod f(x)^{dx} = cF(x) \tag{2.12}$$

The infinitesimal quantity dx is placed as an exponent instead of as an multiplier in the normal case.

This is a definition for an indefinite integral. For definite integrals a form of the multiplicative integral can also be introduced in a way analogue to the Riemann sum. In the normal case we have:

$$\int_a^b f(x)dx = \lim_{h_i \to 0} \sum_{i=1}^{N} f(\xi_i)h_i \tag{2.13}$$

with $\xi_i \in [x_{i-1}, x_i]$, $x_0 = a$, $x_N = b$ and $h_i = x_i - x_{i-1}$.

In the multiplicative case this formula becomes:

$$\prod_a^b f(x)^{dx} = \lim_{h_i \to 0} \prod_{i=1}^{N} f(\xi_i)^{h_i} \tag{2.14}$$

with $\xi_i \in [x_{i-1}, x_i]$, $x_0 = a$, $x_N = b$ and $h_i = x_i - x_{i-1}$.

2.3.1 Properties of the multiplicative integral.

From Spivey[17] we have a useful lemma:

Lemma 2.3.1. *(Comparison of multiplicative integrals). Let f, g be multiplicatively integrable functions on $[a, b]$ and $f(x) \leq g(x), \quad \forall x \in [a, b]$, then $\prod_a^b f(x)^{dx} \leq \prod_a^b g(x)^{dx}$.*

The proof follows that for each value c_k representing a subinterval in (2.14) we have $f(c_k) \leq g(c_k)$ making each term smaller, and as such also the product.

The relation between the anti-derivative from (2.12) and the Riemann integral (2.14) is given by

Theorem 2.3.2. *(Fundamental Theorem of Multiplicative Calculus) Let $f \in V$ and let f^* be a continuous and multiplicatively integrable function on $[a, b]$ then*

$$\prod_a^b f^*(x)^{dx} = \frac{f(b)}{f(a)}. \tag{2.15}$$

And in a other form. Let f be a multiplicative continuous and integrable function and let

$$F(x) = \prod_a^x (f(t))^{dt}, \quad a \leq x \leq b. \tag{2.16}$$

Then F is multiplicatively differentiable at c and $F^*(c) = f(c)$ for all $c \in (a, b)$.

If we are working on a commutative field we have the following relation between the normal additive integral and the multiplicative integral.[4].

Theorem 2.3.3. *Let f be a positive and Riemann integrable function on $[a,b]$ then, it is also multiplicatively integrable on $[a,b]$ and*

$$\prod_a^b f(x)^{dx} = \exp\left(\int_a^b \ln(f(x))dx\right). \tag{2.17}$$

And also for the multiplicative integral a lot of properties exist. Here is a list distilled from the various papers.

1. Integrating over an interval of length 0 gives $\prod_a^a f(x)^{dx} = 1$, compare this to $\int_a^a f(x)dx = 0$ in the additive case.
2. $\prod_a^b f(x)^{dx} = \left(\prod_b^a f(x)^{dx}\right)^{-1}$ (compare to $\int_a^b f(x)dx = -\int_b^a f(x)dx$).
3. $\prod_a^b (f(x)^\lambda g(x)^\mu)^{dx} = \left(\prod_a^b f(x)^{dx}\right)^\lambda \left(\prod_a^b g(x)^{dx}\right)^\mu$
4. $\prod_a^b \left(\frac{f(x)}{g(x)}\right)^{dx} = \frac{\prod_a^b f(x)^{dx}}{\prod_a^b g(x)^{dx}}$
5. $\prod_a^b (f(x)^p)^{dx} = \left(\prod_a^b f(x)^{dx}\right)^p, \quad p \in \mathbb{R},$
6. $\prod_a^b f(x)^{dx} = \prod_a^c f(x)^{dx} \cdot \prod_c^b f(x)^{dx}, \quad a \le c \le b,$
7. Multiplicative integration by parts: $\prod_a^b (f^*(x)^{g(x)})^{dx} = \frac{f(b)^{g(b)}}{f(a)^{g(a)}} \frac{1}{\prod_a^b (f(x)^{g'(x)})^{dx}}$, requiring f to be multiplicative differentiable, g to be differentiable and f^g to be multiplicatively integrable.
8. Positivity if f is continuous and multiplicatively integrable (so positive) on $[a,b]$ then $\prod_a^b f(x)^{dx} > 0$

2.3.2 Examples

Some selected examples are given in table 2.1

2.3.3 Alternate definition

Slavik [16] and other use a different version of the product integral. It is related to differential equations of Matrix functions.

Definition 2.3.2. *Let $A : [a,b] \to \mathbb{R}^{n\times n}$ be a matrix function, and I the identity matrix. We have the partition of the interval $[a,b]$ in $a = t_0 \le\le t_1 \le \cdots \le t_{m-1} \le t_m = b$. With*

$\Delta t_i = t_i - t_{i-1}$ *and points* $\xi_i \in [t_{i-1}, t_i]$ *for* $i = 1, 2, \cdots, m$. *Then the product integral is defined as:*

$$\prod_1^m (I + A(t)dt) = \lim_{\Delta t_i \to 0} \prod_{i=m}^1 (I + A(\xi_i)\Delta t_i) \tag{2.18}$$

Note that the order of the product is of importance due to non commutativity. A lot of properties can be described for this types of integrals. This version is defined in most papers based on a differential equation of matrix functions. As this integral does NOT correspond to the anti-derivative of the multiplicative derivative it is not described in this report. For more information see the resources in the next section.

2.4 Further Reading

During the literature research also some other lines of thoughts were researched. The following related topics can be found in the referenced papers.

- On multiplicative calculus, I did found several papers each giving again a very similar overview of the subject, mostly in the scalar case: [18, 17, 3, 4] but also some interesting extra theory like [19]. Spivey ports Taylor series, L'hôpital, and integration by parts to Multiplicative calculus. While Bashirov also introduces multiplicative differential equations. The original ideas go back to Volterra [21]
- Combining this with matrix value functions one can find some general without details in [2].
- On Matrix valued functions in general some theory is given in [5, 13], based on 'normal' calculus.
- Product Integrals (the inverse of multiplicative derivatives) have been discussed more often in literature. It is often even defined for Matrix valued functions. The most comprehensive text is the book by Dollard [8]. Also Slavik [16] gives lots of details on product integration without mentioning explicitly the issues around commutativity.
- Some papers relating on multiplicative calculus are on product integrals from applications in highly specialised fields like Stochastic, or Quantum physics. Some examples are [10, 12, 11]. Unfortunately these papers are in general not very accessible for some one outside there respective fields.
- Searching on non-commutative on google scholar, also gives lot's of papers on non-commutative logic and/or geometry. Some examples are [1, 14]

Chapter 3

Principal Component Analysis

The Principal Component Analysis (PCA) [7] are a method for analysing the distribution of a set of n-tuples. The core idea is to find a structure in a set of numbers in such a way that the whole distribution of numbers can be described by a smaller number of parameters. The original data is usually correlated in an unknown way, and the PCA method tries to find a good approximation, so we can describe the complete variance of the data using a set of linearly uncorrelated parameters.

Remark. Initially a method called *Active Shape Models* where investigated. The ideas are rooted in image analysis. They try to train a model of certain shapes in images. Later this trained model is used for generating new shapes that have variations similar to the ones in the training set. This is done by creating a small set of parameters that represent the characteristics of the set of shapes.

From the Active Shape Model, only the essential part was distilled, leaving the parts specific to images out. It turned out that what was left is called Principal Component Analysis. This also explains why the cited literate is mostly on Active Shape Models and not PCA.

3.1 Theory

This theory can also be applied not to series of shapes but just series of n-tuples. First the normal linear case will be described of this algorithm. Then the algorithm will be generalised to use a multiplicative structure in the applicable vector space. At last both algorithms will be applied to generated and real data to compare the properties, (cf. [7, 20, 6, 15]).

3.1.1 Linear PCA

We start with a dataset containing m n-tuples. We denote them by column vectors

$$\mathbf{x_j} = (x_{j1}, x_{j2}, \ldots, x_{jn})^T, \quad \text{for } j = 1, 2, \ldots, m. \tag{3.1}$$

It is assumed that these vectors are already put in the basis and scale, so we don't need to align them, like it is done in[7].

Remark. In many practical applications it is essential that the data are aligned first. Usually this requires a registration step, and the application of the appropriate transforms. For

example if the data corresponds to coordinates in an image, the correct translation, rotation, etc. have to be found.

The purpose of the alignment/registration process is to establish a one to one correspondence between the points in different vectors. They must each represent exactly the same feature.

The second step is to calculate the average for each point in the tuple over the set of vectors:

$$\bar{\mathbf{x}} = \frac{1}{m}\sum_{j=1}^{m}\mathbf{x_j} \tag{3.2}$$

which is element-wise given by:

$$\bar{x}_i = \frac{1}{m}\sum_{j=1}^{m} x_{ji} \quad \text{for } j = 1, 2, \ldots, m \tag{3.3}$$

Now we have an average vector describing the n-tuple. We now need to look at the differences between each vector $\mathbf{x}_j$ and the average $\bar{\mathbf{x}}$. These deviations are given by

$$\delta\mathbf{x}_j = \mathbf{x}_j - \bar{\mathbf{x}} \tag{3.4}$$

These differences all together form a $n \times m$ difference matrix $\delta X = [\delta\mathbf{x}_1\delta\mathbf{x}_2 \cdots \delta\mathbf{x}_n]$.

The relevant properties are found by calculating the covariance matrix $S \in \mathbb{R}^{n\times n}$ given by the formula:

$$S = \frac{1}{N}\sum_{j=1}^{N}\delta\mathbf{x}_j\delta\mathbf{x}_j^T = \delta X\delta X^T \tag{3.5}$$

The covariance matrix describes the relations between the various coordinates in the n-tuple, and how they variate. By looking at the eigenpairs of this matrix we can get accurate descriptions of typical variations in the training set.

Remark. The eigenvalues of S can be calculated very fast using the singular value decomposition of DX.

So the eigenpairs $(\lambda_k, \mathbf{p}_k)$ are found, such that

$$S\mathbf{p}_k = \lambda_k\mathbf{p}_k \tag{3.6}$$

for the kth eigenvalue, $k = 1, \ldots, n$ and we assume $\lambda_k \geq \lambda_{k+1} > 0$ for all $k = 1, \ldots, n$. With $P_t = (\mathbf{p}_1, \mathbf{p}_1, \ldots, \mathbf{p}_t)$ we denote the $n \times t$ matrix of the t first eigenvectors.

In this case the largest direction of change is given by the first eigenvector, corresponding to the largest eigenvalue. The goal is to give a small set of parameters corresponding to characteristic changes in the data set. This is done by looking for a subset of eigenvalues representing a very large amount of the eigenspace. So we will determine t parameters such that

$$\sum_{i=1}^{t}\lambda_i \geq \eta\sum_{i=1}^{n}\lambda_i \tag{3.7}$$

where $\eta \in (0, 1)$is a parameter determining the size of the proportion of eigenspace that we cover. Alternatively one can choose t such that the error made by the approximations in (3.8), are small compared to a chosen threshold.

Any n-tuple can be reached by a linear combination of the eigenvectors as they span a basis of the vector space. In particular we can approximate every vector $\mathbf{x}$ using only the first t modes by:

$$\mathbf{x} \approx \bar{\mathbf{x}} + P_t \mathbf{c}, \tag{3.8}$$

where $\mathbf{c} = (c_1, c_2, \ldots, c_t)$ a vector of coefficients representing the weight of each eigenvector. These coefficients can be found by using the linear least squares method applied to this under determined system.

Remark. Note that to convert back from log space, we have to take the exponential of the reconstructed vector.

After this training of the model, it can be used to generate more vectors that are similar to the original ,with the same kind of variations. This is done by looking at the distribution of the coefficients $\mathbf{c}$. This will give limits detailing what values occur in the set, and by creating new vectors c one can create new data points. Another usage is to use the model to fit new data from for example other images. Because the model is trained to what kind of variations are allowed, it is more robust to noise and other interference.

To fit the model against new data points, and get some measure for the error, the following procedure can be used:

1. For the new data $\mathbf{y}_j$ we calculate the differences $\delta\mathbf{y}_j = \mathbf{y}_j - \bar{\mathbf{x}}$, and combine them in the matrix $\delta\mathbf{Y}$.

2. Solve the matrix equation
$$P_t\mathbf{C} = \delta\mathbf{Y}$$
for $\mathbf{C}$. This matrix $\mathbf{C}$ contains in each column the coefficients for the parameters of the model for each corresponding to each vector $\mathbf{y}_j$.

3. To calculate the error we create from the model the new vectors $\hat{\mathbf{y}}_j$ using the (3.8).
$$\hat{\mathbf{Y}} = \bar{\mathbf{x}} + P_t\mathbf{C}$$

4. For each vector $\mathbf{y}_j$ the error e_j is then calculated using
$$e_j = \|\hat{\mathbf{y}}_j - \mathbf{y}_j\|_2 \tag{3.9}$$

3.1.2 Multiplicative PCA

In this section an attempt is made to modify the active shape model in such a way that it utilises a multiplicative structure of the underlying data. The algorithm will be explained along the structure of section 3.1.1, and the focus will be on the justification of the modifications.

The following approach will be used. If one assumes that the input data follows and multiplicative structure, we can create a multiplicative version of PCA by using the following logic

$$\exp(PCA(\log(data)))$$

, so we take the logarithm on the input vectors, then apply the linear PCA, and finally take the exponential of any vectors we create using the model. Now we will analyse what this means for each of the steps in the model, and see how it could be rewritten directly.

Again we start with a set of vectors as defined in (3.1). The normal approach starts with taking the average for each coordinate

$$\begin{aligned} \bar{x}_i &= \frac{1}{m}\sum_{j=1}^{m}\log(x_{ji}) \\ &= \log\left(\left(\prod_{j=1}^{m} x_{ji}\right)^{1/m}\right), \quad \text{for } j = 1, 2, \ldots, m. \end{aligned} \tag{3.10}$$

It is clear that the arithmetic mean, is replaced by the geometric mean, which is logical to apply in a multiplicative case.

The next step in the algorithm is to calculate the differences between the vectors and the mean.

$$\delta\mathbf{x}_j = \log(\mathbf{x}_j) - \bar{\mathbf{x}} \tag{3.11}$$

These differences all together form again a difference matrix $\delta X^* = [\delta\mathbf{x}_1^*\delta\mathbf{x}_2^*\cdots\delta\mathbf{x}_n^*]$. For calculating the covariance matrix, we use the normal linear approach. (3.5),

$$S^* = \frac{1}{N}\sum_{j=1}^{N}\delta\mathbf{x}_j\delta\mathbf{x}_j^T = \delta X^*\delta X^{*T} \tag{3.12}$$

The rest of the algorithm is almost identical to the section 3.1.1, but given again for completion. So the eigenpairs $(\lambda_k, \mathbf{p}_k)$ are found, such that

$$S\mathbf{p}_k = \lambda_k\mathbf{p}_k \tag{3.13}$$

for the kth eigenvalue, $k = 1, \ldots, n$ and we assume $\lambda_k \geq \lambda_{k+1} > 0$ for all $k = 1, \ldots, n$. With $P_t = (\mathbf{p}_1, \mathbf{p}_1, \ldots, \mathbf{p}_t)$ we denote the $n \times t$ matrix of the t first eigenvectors.

In this case the largest direction of change is given by the first eigenvector, corresponding to the largest eigenvalue. The goal is to give a small set of parameters corresponding to characteristic changes in the data set. This is done by looking for a subset of eigenvalues representing a very large amount of the eigenspace. So we will determine t parameters such that

$$\sum_{i=1}^{t}\lambda_i \geq \eta\sum_{i=1}^{n}\lambda_i \tag{3.14}$$

where $\eta \in (0, 1)$is a parameter determining the size of the proportion of eigenspace that we cover. Alternatively one can choose t such that the error made by the approximations in (3.15), are small compared to a chosen threshold.

Any n-tuple can be reached by a linear combination of the eigenvectors as they span a basis of the vector space. In particular we can approximate every vector $\mathbf{x}$ using only the first t modes by:

$$\mathbf{x} \approx \exp\left(\bar{\mathbf{x}} + P_t\mathbf{c}\right), \tag{3.15}$$

where $\mathbf{c} = (c_1, c_2, \ldots, c_t)$ a vector of coefficients representing the weight of each eigenvector. These coefficients can be found by using the linear least squares method applied to this under determined system.

To fit the model against new data points, and get some measure for the error, the procedure of section 3.1.1 is modified:

1. For the new data $\mathbf{y}_j$ we calculate the differences component wise $\delta\mathbf{y}_j = \log(\mathbf{y}_j) - \bar{\mathbf{x}}$, and combine them in the matrix $\delta\mathbf{Y}$.

2. Solve the matrix equation
$$P_t\mathbf{C} = \delta\mathbf{Y}$$
for $\mathbf{C}$. This matrix $\mathbf{C}$ contains in each colum $\mathbf{c}_j$ the coefficients for the parameters of the model for each corresponding to each vector $\mathbf{y}_j$.

3. To calculate the error we create from the model the new vectors $\hat{\mathbf{y}}_j$ we use
$$\hat{\mathbf{y}}_\mathbf{j} = \exp(\bar{\mathbf{x}} + P_t\mathbf{c}_j)$$
.

4. For each vector $\mathbf{y}_j$ the error e_j is then calculated using
$$e_j = \|\hat{\mathbf{y}}_j - \mathbf{y}_j\|_2 \tag{3.16}$$

Remark. If one is using only the multiplicative PCA, it would be interesting to investigate of alternative norms for calculating the error. This is explicitly not done in this report as, it is intended to compare the linear and multiplicative variants of PCA in the following section. For the comparison to work, the same norm has to be used in both cases, so the values mean the same thing.

Another possibility for further research would be to reformulate the eigensystem and it algebra directly to a multiplicative algebra. This goes well beyond the scope of this project.

3.2 Implementation & tests

Both the linear as the multiplicative PCA are implemented in MATLAB. For the code the reader is referred to appendix A. Two different tests have been devised and executed. In the first random vectors are created using different probability distributions. For the second test two classes of parameterised functions are created and analysed using PCA¿

3.2.1 Test using random data vectors

Table 3.1: *A list of different probability distributions and the associate results for the PCA implementations. The numbers listed here are averages over 1000 runs, and the uncertainty is taken at a 0.95 confidence level. It is calculate with* $\eta = 0.15$

Distribution	# Parameters Linear PCA	# Parameters Mult. PCA	Error lin. PCA	Error mult. PCA
Uniform	1	1	$0.0801 \pm 5E-4$	$0.0829 \pm 5E-4$
Normal	9.64 ± 0.08	12.5 ± 0.09	$0.1433 \pm 3E-4$	$0.1407 \pm 4E-4$
Binomial	4.67 ± 0.05	8.58 ± 0.07	$0.1449 \pm 2E-4$	$0.1460 \pm 2E-4$
Exponential	14.34 ± 0.09	19.07 ± 0.02	$0.1391 \pm 4E-4$	$0.104 \pm 3E-3$
Geometric	14.25 ± 0.07	16.75 ± 0.06	$0.1353 \pm 6E-4$	$0.1276 \pm 9E-4$
Poisson	9.07 ± 0.05	13.55 ± 0.05	$0.1437 \pm 2E-4$	$0.1428 \pm 3E-4$

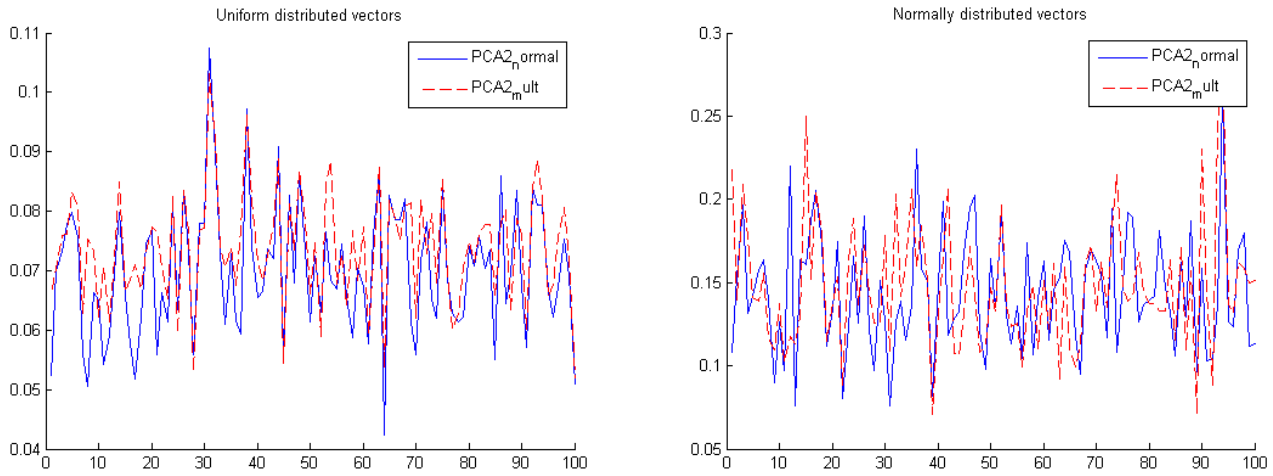

Figure 3.1: Plot of Uniform & normal errors for both linear and multiplicative PCA.

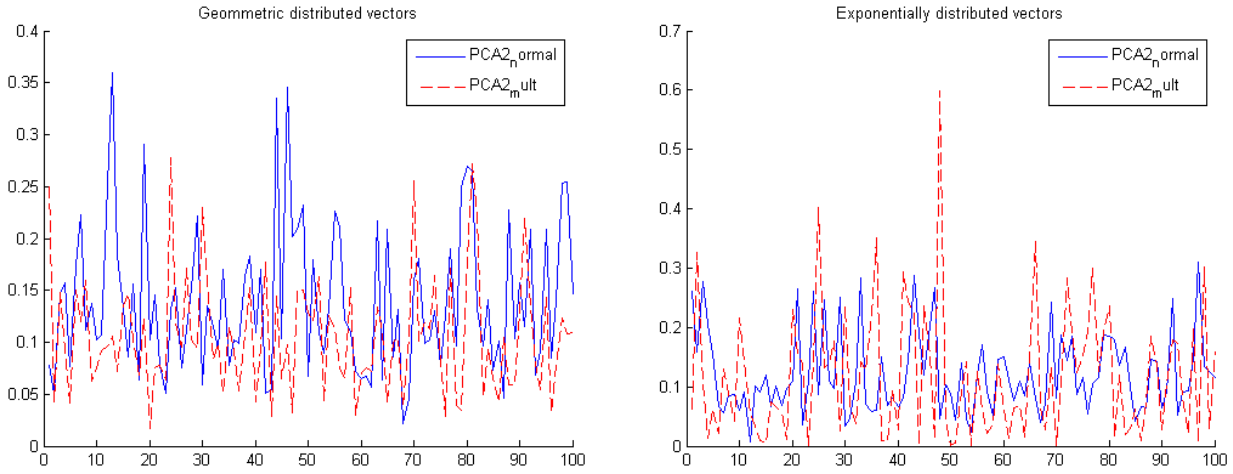

Figure 3.2: Plot of Geometric & exponential errors for both linear and multiplicative PCA.

To test the implementation and to compare both methods some tests were conducted with random data. For this test we generate 100 vectors of length 20. First for each of the 20 coordinates the average is randomly chosen. Then the vectors are generated by creating for each coordinate 100 random numbers according to some distribution. These test vectors were fed to the PCA implementations. The model is run to determine dynamically how many parameters are needed to create a certain accuracy level. After creating the model, the model is fitted on the test vectors and the results compared to the input. For each of the test vectors this gives us error numbers. These errors are plotted in figures 3.1,3.2 for different distributions of randomly chosen coordinates.

As this test is conducted with random numbers it would be preferred to repeat the whole process a number of times to get an average error. For this we need one error number per test, and the average number of parameters used. The test was then ran a 1000 times and the results were averaged and a uncertainty was calculated. The results can be found in table 3.1.

3.2.2 Test with function classes

To test the application of the PCA, two classes of parameterised functions is created. The second class will be created by taking the exponential from the first class. As such we will create a multiplicative structure in the function class.

For the first class of functions f, the following considerations were taken into account:

- We look at simple functions, so we will use polynomials
- We create functions on the interval $[0, 1]$ using 5 parameters. By sampling these functions in 20 points, we can create vectors of length 12, which are characterised by an underlying structure of only 5 parameters. Therefore it will be very appropriate to apply PCA.
- To keep the functions bounded we demand that $f(0) = f(1) = 0$.
- The function should be linear in the parameters.

The most logical form is to use the following polynomial:

$$f_{a,b,c,d,e,f,g}(x) = a + bx + cx^2 + dx^3 + ex^4 + fx^5 + gx^6 \tag{3.17}$$

We need to satisfies the boundary conditions. The first $f(0) = 0$ immediately gives $a = 0$. The second bc $f(1) = 0$ gives $b+c+d+e+f+g = 0$. This means that one of the coefficients will depend on the others. To be able to vary both high and low powers of the polynomial the following class is used:

$$f_{a_1,a_2,a_3,a_4,a_5}(x) = a_1x + a_2x^2 - (a_1 + a_2 + a_3 + a_4 + a_5)x^3 + a_3x^4 + a_4x^5 + a_5x^6 \tag{3.18}$$

To create multiplicative function class, we will use the exponential of this function class .

$$g_{a_1,a_2,a_3,a_4,a_5}(x) = \exp(a_1x + a_2x^2 - (a_1 + a_2 + a_3 + a_4 + a_5)x^3 + a_3x^4 + a_4x^5 + a_5x^6) \tag{3.19}$$

To generate functions from these classes we will use random coefficients. All 5 parameters will be normally distributed around 0, but with different standard deviations.

For each of these classes the following testing procedure is followed:

Table 3.2: *This table list the number of parameters used by the models, and the errors in different situations*

Function Class & situation	# par lin	# par mult	Err Lin	Err Mult
f, model vectors	2	10	0.019	0.044
f, new vectors + noise			0.34	1.12
g, model vectors	6	2	0.038	0.018
g, new vectors + noise			1.13	0.26

1. First we define the standard deviation for each parameter. This was chosen arbitrarily to be:
$$\sigma_1 = 6, \sigma_2 = 2, \sigma_3 = 2, \sigma_4 = 2, \sigma_5 = 5.$$

2. 50 vectors of length 20 are generated by creating 50 normally distributed numbers with the correct standard deviation for each of the parameters. This was then used to evaluate the function f or g at 20 equidistant points in $(0, 1)$, resulting in 50 vectors.

3. These 50 vectors are used to train both the normal and the multiplicative PCA model, with a target error of 0.05.

4. For testing a new set of 1000 vectors are generated from the functions using the same approach as in step 2. On these testing vectors random noise is added. For the first class we add Gaussian noise with standard deviation 1, in each point independently. For the second class we multiply by the exponential of a normal distributed random noise, again with standard deviation 1.

5. The models are used to create the best fit for each vector, As the model forces a certain shape, it should be possible even with a lot of noise.

6. The error for each vector is determined by comparing the fitted vector from the model, to the original generated test vector (out of the 1000) without the noise.

The results of this test can be seen in figures 3.3,3.4 and the errors are detailed in table 3.2.

To show how the model behaves in fitting the test vector, an example vector is plotted in figure 3.5 and 3.6. One can clearly see that the random perturbations significantly alter the shape of the functions. However in either case the model most appropriate for the class has only a few degrees of freedom, and manages to make an excellent fit. The other model has so many parameters that is completely confused by the noise.

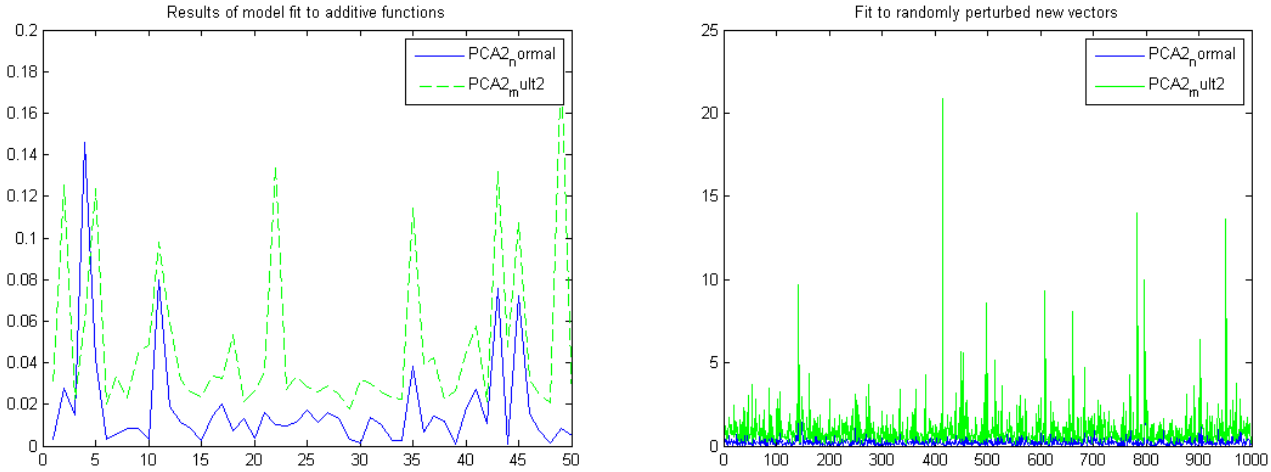

Figure 3.3: Plot of errors for model fit, and to randomly perturbed test vectors. For function class f

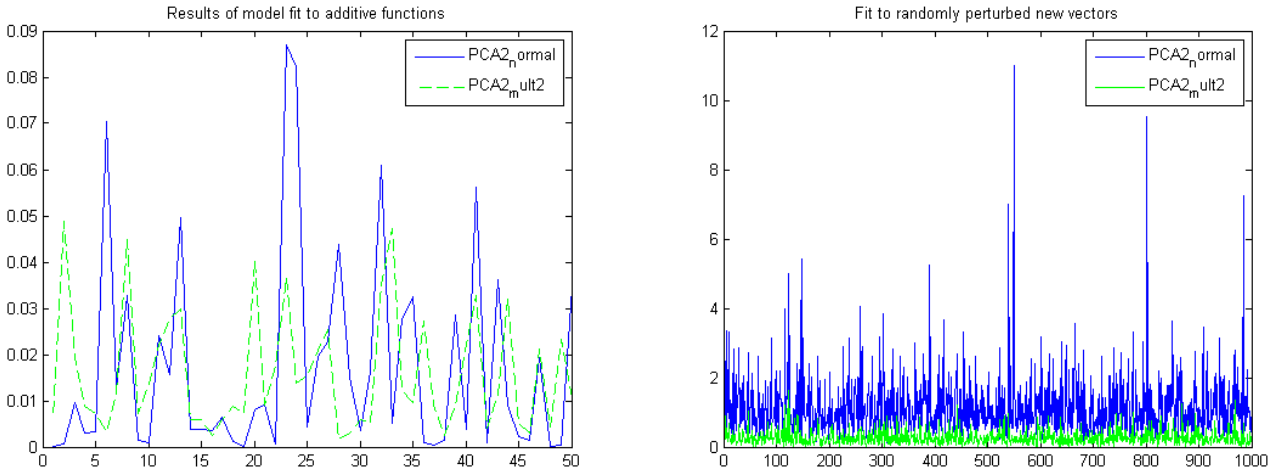

Figure 3.4: Plot of errors for model fit, and to randomly perturbed test vectors. For function class g

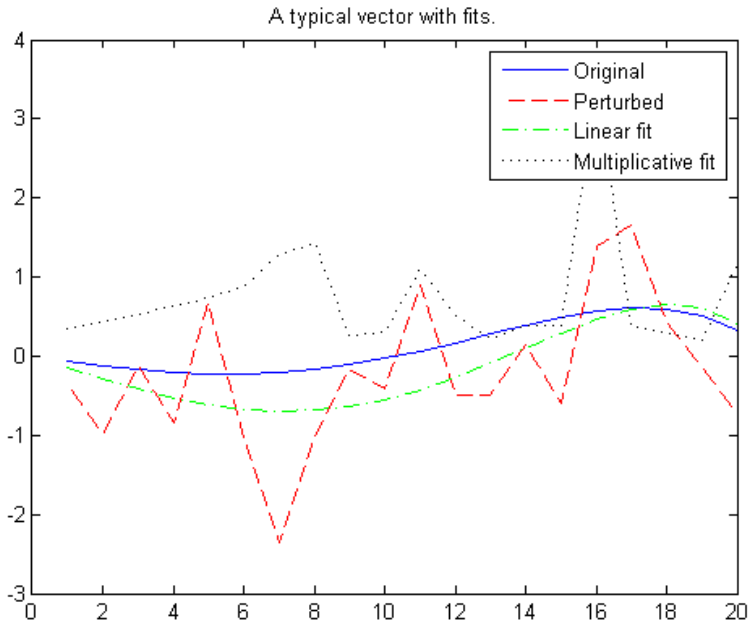

Figure 3.5: Plot of a test vector, with noise, and the fits, for the class f

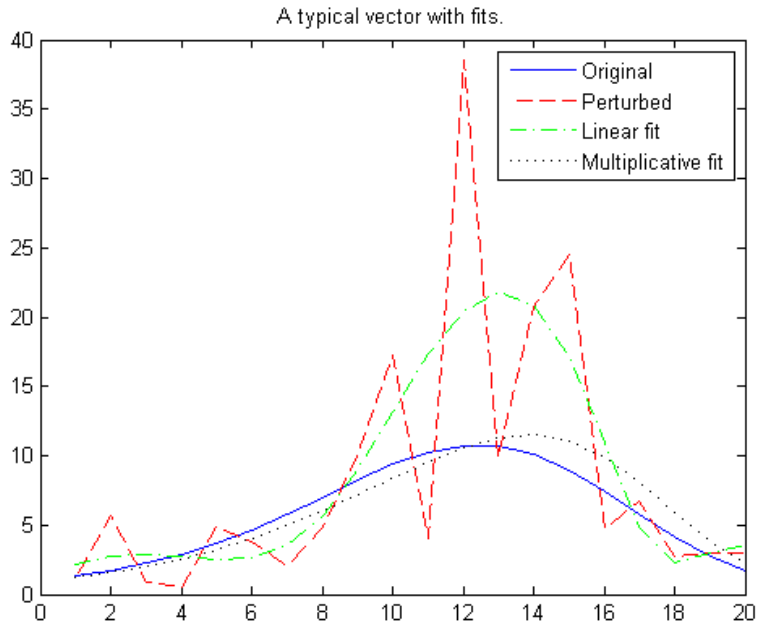

Figure 3.6: Plot of a test vector, with noise, and the fits, for the class f

Chapter 4

Conclusion & discussion

4.1 Theory

It has been argued in this paper that multiplicative calculus provides an almost as rich a toolbox as the normal calculus every mathematician is used to. The theory is very extensive and numerous theorems could be generalised and used also in the multiplicative case.

Secondly it must be concluded that in the commutative case, this theory can easily be constructed by transformations from normal calculus. It seems to be a topic for further research to identify the complications from applying this in a non commutative setting.

4.2 Random tests

From the first artificial test, it can be concluded at a minimum that it seems feasible to apply PCA in a multiplicative way, resulting in sensible results. From the figures one can see that the quality of the differences really depend on the distribution of the vectors fed to the model. This suggest that depending on the underlying relation between the coordinates and vectors, that multiplicative PCA can be a useful analysis tool. In general it can also be seen that the multiplicative variant seems to use less parameters to describe the variations in the model, although the errors seem a bit higher. If this can be tuned better, that it uses some extra parameters the model should be expected to perform better.

Comparing the linear and the multiplicative variants, one sees that for the uniform, the exponential and the geometric distributions the two models have the closed results. It is expected to find similar results for the exponential and the geometric distributions as they are both a "memory-less" probability distribution. In fact they are the only memory-less distributions for respectively the continuous and discrete case. As the probabilities from the geometric distribution form a geometric series, it is not surprising that the multiplicative PCA performs well under this kind of numbers.

4.3 Function class test

This test has some very promising results. From the results we can conclude that as expected depending on the function class either the linear or the multiplicative variant of PCA has the best results. If the models are programmed to choose the numebr of parameters such that a given error is reached, we see that the most appropriate model needs the least amount of

parameters for the approximation. Secondly we see that in the case of class f the linear model is more robust to the noise then the multiplicative model. This can be logically expected, as the first model has less parameters and therefore forces the fits to be more in line with the model. At the same time the multiplicative model has as many as 10 degrees of freedom, therefore allowing it to fit to the noise, be send astray very easily.

In the case of function class g the roles are reversed. Here the multiplicative model is performing much better then the linear ones. As the multiplicative model is implemented by using the logarithm and the linear model, is is expected to the results should be similar to the ones from the linear model with function class f. However this is not completely true, for function class g the multiplicative model is even more robust to noise than the linear one in case of class f. The figures 3.5 and 3.6 both show that using the correct model it can be very robust. Even if due to the random noise the function has become completely different, a nice smooth fit can be obtained. This fit lies close to the original function, thereby proving that the correct application of PCA can be very useful in the analysis of datasets.

4.4 Final conclusion

Reviewing the total report the following can be concluded.

- Multiplicative Calculus can be a very powerful toolbox for the analysis of different systems, problems etc. The toolbox contains most of the tools known from the classic Calculus. One might expect that with further research even more theorems and procedures can be adapted to the multiplicative case.

- It is relatively unknown in the mathematical community as can be seen on the low number of papers published on the subject in over a century. One explanation is that in most cases the same results can be found using classical Calculus and a simple transformation using exponentials and logarithms. However if it would be applied in a context where the multiplication is non commutative, great care would have to be taken to take into account any effects it would have. For the multiplicative integrals, the Riemann product would depend on the order of the terms.

- Principal component analysis can easily be transformed into what one could call a multiplicative variant.

- For PCA it is concluded from the numerical tests that it depends on the underlying relations in the data which version is best. If one creates datasets from function in a way that it is more natural to describe it my multiplicative parameters, it turns out the multiplicative version of PCA gives superior results.

- If PCA is trained with a reasonable basis set of vectors covering the possible variations, it is a very robust method, that can be used to fit new unknown vectors, even if there is a significant amount of noise.

- Based on the above it seems reasonable to start further research into the application of multiplicative tools for the analysis of different systems.

Bibliography

[1] V. Abrusci and P. Ruet, *Non-commutative logic i: the multiplicative fragment*, Annals of Pure and Applied Logic, 101 (1999), pp. 29–64.

[2] V. Arsigny, P. Fillard, X. Pennec, and N. Ayache, *Log-euclidean metrics for fast and simple calculus on diffusion tensors*, Magnetic Resonance in Medicine, 56 (2006), pp. 411–421.

[3] A. Bashirov, E. Misirli, Y. Tandogdu, and A. zyapici, *On modeling with multiplicative differential equations*, Applied Mathematics - A Journal of Chinese Universities, 26 (2011), pp. 425–438. 10.1007/s11766-011-2767-6.

[4] A. E. Bashirov, E. M. Kurpinar, and A. zyapici, *Multiplicative calculus and its applications*, Journal of Mathematical Analysis and Applications, 337 (2008), pp. 36 – 48.

[5] B. Burgeth, S. Didas, L. Florack, and J. Weickert, *A generic approach to diffusion filtering of matrix-fields*, Computing, 81 (2007), pp. 179–197. 10.1007/s00607-007-0248-9.

[6] T. Cootes, A. Hill, C. Taylor, and J. Haslam, *Use of active shape models for locating structures in medical images*, Image and Vision Computing, 12 (1994), pp. 355 – 365. ¡ce:title¿Information processing in medical imaging¡/ce:title¿.

[7] T. Cootes, C. Taylor, D. Cooper, J. Graham, et al., *Active shape models-their training and application*, Computer vision and image understanding, 61 (1995), pp. 38–59.

[8] J. D. Dollard and C. N. Friedman, *Product integration with applications to differential equations*, vol. 10 of Encyclopedia of mathematics and its application, Addison-Wesley, 1979.

[9] L. Florack and H. van Assen, *Multiplicative calculus in biomedical image analysis*, Journal of Mathematical Imaging and Vision, (2011), pp. 1–12. 10.1007/s10851-011-0275-1.

[10] R. Gill and S. Johansen, *A survey of product-integration with a view toward application in survival analysis*, The annals of statistics, (1990), pp. 1501–1555.

[11] R. L. Hudson and S. Pulmannová, *Algebraic theory of product integrals in quantum stochastic calculus*, Journal of Mathematical Physics, 41 (2000), pp. 4967–4980.

[12] I. Kramer, *The product-integral calculus formulation in quantum mechanics*, Amer. J. Phys, 40 (1972), p. 1221. 10.1119/1.1986806.

[13] X. Pennec, P. Fillard, and N. Ayache, *A riemannian framework for tensor computing*, International Journal of Computer Vision, 66 (2006), pp. 41–66. 10.1007/s11263-005-3222-z.

[14] C. Retor, *Pomset logic: A non-commutative extension of classical linear logic*, in Typed Lambda Calculi and Applications, P. de Groote and J. Roger Hindley, eds., vol. 1210 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 1997, pp. 300–318. 10.1007/3-540-62688-3_43.

[15] S. Romdhani, S. Gong, A. Psarrou, et al., *A multi-view nonlinear active shape model using kernel pca*, in British Machine Vision Conference, vol. 99, 1999, pp. 483–492.

[16] A. Slavk, *Product Integration, its history and applications*, Matgyzpress, 2007.

[17] M. Z. Spivey, *A product calculus.* `http://math.pugetsound.edu/~mspivey/ProdCalc.pdf`.

[18] D. Stanley, *A multiplicative calculus*, PRIMUS, 9 (1999), pp. 310–326.

[19] A. B. Twist and M. Z. Spivey, *L'hôpital's rules and taylor's theorem for product calculus.* `http://math.pugetsound.edu/~mspivey/ProdInt.pdf`.

[20] B. van Ginneken, A. Frangi, J. Staal, B. ter Haar Romeny, and M. Viergever, *Active shape model segmentation with optimal features*, Medical Imaging, IEEE Transactions on, 21 (2002), pp. 924 –933.

[21] V. Volterra, *Sulle equazioni differenziali lineari*, Sulle equazioni differenziali lineari, 3 (1887), pp. 393–396.

Appendix A

Implementations

In this chapter the implementation code is given.

A.1 Models

A.1.1 PCA_normal.m

```
function [Evalues, Evectors, coef, Xnorm, num_par]=PCA_normal(X, p, verb)
% X are set of row vector, first index is i of vector, second is j within
% vector
if nargin <2
    p=0.1;
end;
if nargin <3
    verb=true;
end


%first determine mean per coordinate
x_mean=mean(X,2);
s=size(X);

%subtract mean from data
X2=bsxfun(@minus, X, x_mean);

%calculate Covariance:
S=X2*X2'/s(1);

%calculate eigenpairs
[Evectors, Evalues]=eig(S);
Evalues=diag(Evalues);

%sort eigenvalues and vectors
[~, ind]=sort(Evalues, 'descend');
Evalues=Evalues(ind);
Evectors=Evectors(:, ind);


%find number of parameters
num_par=0;
error=1;
```

```
while (error>p) && (num_par<=length(Evalues))
    num_par=num_par+1;

    %fit to original vectors
    coef=pinv(Evectors(:,1:num_par))*X2;

    %Determine norm of fit
    %create fitted vecs; A linear comb of eigenvectors, and added to the mean
    Xn=repmat(x_mean,1,s(2))+Evectors(:,1:num_par)*coef;

    %calculate norm of the results and create error vector.
    Xnorm=zeros(1,size(Xn,2));
    for i=1:size(Xn,2)
        Xnorm(i)=norm(Xn(:,i)-X(:,i),2)/norm(X(:,i),2);
    end;
    error=mean(Xnorm);

end;
    if verb
    fprintf('Number parameters: %i covering %g of eigenspace\n',num_par,sum(Evalues
        (1:num_par))/sum(Evalues));
    end;
```

A.1.2 PCA_mult.m

```
function [Evalues,Evectors,coef,Xnorm,Xnorm_2,Xnorm_3,num_par]=PCA_mult2(X,p,verb)
%implemented usign log expo:
Xorg=X;
X=log(X);

if nargin <2
    p=0.1;
end;
if nargin <3
    verb=true;
end


%first determine mean per coordinate
x_mean=mean(X,2);
s=size(X);

%subtract mean from data
X2=bsxfun(@minus,X,x_mean);

%calculate Covariance:
S=X2*X2'/s(1);

%calculate eigenpairs
[Evectors,Evalues]=eig(S);
Evalues=diag(Evalues);

%sort eigenvalues and vectors
[~,ind]=sort(Evalues,'descend');
Evalues=Evalues(ind);
Evectors=Evectors(:,ind);
```

```
%find number of parameters
num_par=0;
error=1;
while (error>p) && (num_par<=length(Evalues))
    num_par=num_par+1;

    %fit to original vectors
    coef=pinv(Evectors(:,1:num_par))*X2;

    %Determine norm of fit
    %create fitted vecs; A linear comb of eigenvectors, and added to the mean
    Xn=exp(repmat(x_mean,1,s(2))+Evectors(:,1:num_par)*coef);

    %calculate norm of the results and create error vector.
    Xnorm=zeros(1,size(Xn,2));Xnorm_2=Xnorm;Xnorm_3=Xnorm;
    for i=1:size(Xn,2)
        Xnorm(i)=norm_mult(Xn(:,i)./Xorg(:,i),2);
        Xnorm_2(i)=norm(Xn(:,i)./Xorg(:,i),2);
        Xnorm_3(i)=norm(Xn(:,i)-Xorg(:,i),2)/norm(Xorg(:,i),2);
    end;
        error=mean(Xnorm_3);
end;
    if verb
    fprintf('Number parameters: %i covering %g of eigenspace\n',num_par,sum(Evalues
        (1:num_par))/sum(Evalues));
    end;
```

A.2 Test scripts

A.2.1 test_script.m

```
clear all;close all;
%Generate data sets:
rng(37);
n=20;%length of vectors
m=100;%number of vectors;
p=0.15;
verb=true;
n_trials=1;


error_unif=zeros(2,n_trials);
error_norm=zeros(2,n_trials);
error_binom=zeros(2,n_trials);
error_exp=zeros(2,n_trials);
error_geo=zeros(2,n_trials);
error_pois=zeros(2,n_trials);
par_unif=zeros(2,n_trials);
par_norm=zeros(2,n_trials);
par_binom=zeros(2,n_trials);
par_exp=zeros(2,n_trials);
par_geo=zeros(2,n_trials);
par_pois=zeros(2,n_trials);
```

```
fprintf('Trial: 0000');
for k=1:n_trials
    avg=abs(randn(1,20)*4+7)+1;
    sd=abs(randn(1,20)+3);
    %unif
    X=zeros(n,m);
    for i=1:n
        X(i,:)=abs(rand(1,m)*sd(i)+avg(i))+0.01;
    end;
    %calculate PCA
    [~,~,~,nrm_n,n_n]=PCA2_normal(X,p,verb);
    [~,~,~,~,~,nrm_m3,n_m]=PCA2_mult2(X,p,verb);
    error_unif(:,k)=[mean(nrm_n),mean(nrm_m3)];
    par_unif(:,k)=[n_n,n_m];
    if verb, fprintf('Norms: %g\t\t%g\n',mean(nrm_n),mean(nrm_m3));
        %get statistics
        figure;hold on; plot(nrm_n,'b-');plot(nrm_m3,'r--');hold off;
        legend('PCA2_normal','PCA2_mult');title('Uniform distributed vectors');end;

    %Normal
    X=zeros(n,m);
    for i=1:n
        X(i,:)=abs(randn(1,m)*sd(i)+avg(i))+0.01;
    end;
    %calculate PCA
    [~,~,~,nrm_n,n_n]=PCA2_normal(X,p,verb);
    [~,~,~,~,~,nrm_m3,n_m]=PCA2_mult2(X,p,verb);
    error_norm(:,k)=[mean(nrm_n),mean(nrm_m3)];
    par_norm(:,k)=[n_n,n_m];

    if verb, fprintf('Norms: %g\t\t%g\n',mean(nrm_n),mean(nrm_m3));
        %get statistics
        figure;hold on; plot(nrm_n,'b-');plot(nrm_m3,'r--');hold off;
        legend('PCA2_normal','PCA2_mult');title('Normally distributed vectors');end;


    %Binomial
    X=zeros(n,m);
    for i=1:n
        X(i,:)=binornd(round(avg(i)*2),0.5,1,m)+1;
    end;
    %calculate PCA
    [~,~,~,nrm_n,n_n]=PCA2_normal(X,p,verb);
    [~,~,~,~,~,nrm_m3,n_m]=PCA2_mult2(X,p,verb);
    error_binom(:,k)=[mean(nrm_n),mean(nrm_m3)];
    par_binom(:,k)=[n_n,n_m];error_binom(:,k)=[mean(nrm_n),mean(nrm_m3)];

    if verb, fprintf('Norms: %g\t\t%g\n',mean(nrm_n),mean(nrm_m3));
        %get statistics
        figure;hold on; plot(nrm_n,'b-');plot(nrm_m3,'r--');hold off;
        legend('PCA2_normal','PCA2_mult');title('Binomial distributed vectors');end;

    %Exponential
    X=zeros(n,m);
    for i=1:n
```

```
        X(i,:)=exprnd(1/avg(i),1,m);
    end;
    %calculate PCA
    [~,~,~,nrm_n,n_n]=PCA2_normal(X,p,verb);
    [~,~,~,~,~,nrm_m3,n_m]=PCA2_mult2(X,p,verb);
    error_exp(:,k)=[mean(nrm_n),mean(nrm_m3)];
    par_exp(:,k)=[n_n,n_m];
    if verb, fprintf('Norms: %g\t\t%g\n',mean(nrm_n),mean(nrm_m3));
        %get statistics
        figure;hold on; plot(nrm_n,'b-');plot(nrm_m3,'r--');hold off;
        legend('PCA2_normal','PCA2_mult');title('Exponentially distributed vectors')
            ;end;


    %Geometric
    X=zeros(n,m);
    for i=1:n
        X(i,:)=geornd(1/avg(i),1,m)+1;
    end;
    %calculate PCA
    [~,~,~,nrm_n,n_n]=PCA2_normal(X,p,verb);
    [~,~,~,~,~,nrm_m3,n_m]=PCA2_mult2(X,p,verb);
    error_geo(:,k)=[mean(nrm_n),mean(nrm_m3)];
    par_geo(:,k)=[n_n,n_m];%get statistics

    if verb, fprintf('Norms: %g\t\t%g\n',mean(nrm_n),mean(nrm_m3));
        figure;hold on; plot(nrm_n,'b-');plot(nrm_m3,'r--');hold off;
        legend('PCA2_normal','PCA2_mult');title('Geommetric distributed vectors');
            end;

    %Poisson
    X=zeros(n,m);
    for i=1:n
        X(i,:)=poissrnd(avg(i),1,m)+1;
    end;
    %calculate PCA
    [~,~,~,nrm_n,n_n]=PCA2_normal(X,p,verb);
    [~,~,~,~,~,nrm_m3,n_m]=PCA2_mult2(X,p,verb);
    error_pois(:,k)=[mean(nrm_n),mean(nrm_m3)];
    par_pois(:,k)=[n_n,n_m];

    if verb, fprintf('Norms: %g\t\t%g\n',mean(nrm_n),mean(nrm_m3));
        %get statistics
        figure;hold on; plot(nrm_n,'b-');plot(nrm_m3,'r--');hold off;
        legend('PCA2_normal','PCA2_mult');title('Poisson distributed vectors');end;
    fprintf('\b\b\b\b%04i',k);
end;
stats=@(x,y) [mean(x,2),std(x,0,2)/sqrt(n_trials)*1.96,mean(y,2),std(y,0,2)/sqrt(
    n_trials)*1.96]
stats(error_unif,par_unif)
stats(error_norm,par_norm)
stats(error_binom,par_binom)
stats(error_exp,par_exp)
stats(error_geo,par_geo)
stats(error_pois,par_pois)
```

A.2.2 test_func.m

```
clear all;%close all;
%Test using creation of classes of functions
n=20; %sample number
x=(1/(n+1):1/(n+1):1-1/(n+1))';
m=50;% number of vectors to create model
m2=1000;% number of vectors to test on
p=0.05;
verb=true;
%standard deviations for different parameters
s1=6;
s2=2;
s3=2;
s4=2;
s5=5;
s_noise=1;


%create function last coordinate is to make sure boundary f(0)=f(1)=0
%f=@(x,a1,a2,a3,a4,a5) a1*x+a2*x.^2+a3*x.^3+a4*x.^4+a5*x.^5+(-a1-a2-a3-a4-a5)*x.^6;
f=@(x,a1,a2,a3,a4,a5) a1*x+a2*x.^2+(-a1-a2-a3-a4-a5)*x.^3+a3*x.^4+a4*x.^5+a5*x.^6;
g=@(x,a1,a2,a3,a4,a5) exp(f(x,a1,a2,a3,a4,a5));


for k=1:2
    if (k==1) h=f;fprintf('Additive functions\n');
    else h=g;fprintf('Multiplicative functions:\n');end;

%Create linear input vectors
X_in=zeros(n,m);
%create random parameters
a1=randn(1,m)*s1;
a2=randn(1,m)*s2;
a3=randn(1,m)*s3;
a4=randn(1,m)*s4;
a5=randn(1,m)*s5;
for i=1:m
    X_in(:,i)=h(x,a1(i),a2(i),a3(i),a4(i),a5(i));
end;

%Create PCA models
[~,Evec_n,~,nrm_n,n_n]=PCA2_normal(X_in,p,verb);
%[~,Evec_m,~,~,~,nrm_m3,n_m]=PCA2_mult(X_in,p,verb);
[~,Evec_m2,~,~,~,nrm_m3_2,n_m2]=PCA2_mult2(X_in,p,verb);
fprintf('Norms: %g\t\t%g\n',mean(nrm_n),mean(nrm_m3_2));
figure(-2+3*k); plot(nrm_n,'b-');hold on;%plot(nrm_m3,'r--');
plot(nrm_m3_2,'g--');hold off;
legend('PCA2_normal','PCA2_mult2');title('Results of model fit to additive functions
    ');


X_in=zeros(n,m2);
%create random parameters
clear a1 a2 a3 a4 a5
a1=randn(1,m2)*s1;
a2=randn(1,m2)*s2;
```

```
a3=randn(1,m2)*s3;
a4=randn(1,m2)*s4;
a5=randn(1,m2)*s5;
for i=1:m2
    X_in(:,i)=h(x,a1(i),a2(i),a3(i),a4(i),a5(i));
end;


%Generate new vectors + random noise
noise=randn(size(X_in))*s_noise;
if k==1
X_in2=X_in+noise;
else
    X_in2=X_in.*exp(noise);
end;

%X2=X.*(1+randn(size(X))*0.2);
%fit
s=size(X_in2);
coef_n=pinv(Evec_n(:,1:n_n))*bsxfun(@minus,X_in2,mean(X_in,2));
Xn=repmat(mean(X_in,2),1,s(2))+Evec_n(:,1:n_n)*coef_n;
nrm_n=zeros(1,size(Xn,2));
for i=1:size(Xn,2)
    nrm_n(i)=norm(Xn(:,i)-X_in(:,i),2)/norm(X_in(:,i),2);
end;
%{
coef_m=Evec_m(:,1:n_m)\bsxfun(@rdivide,X_in2,mean_mult(X_in,2));
Xm=repmat(mean_mult(X_in,2),1,s(2)).*(Evec_m(:,1:n_m)*coef_m);
nrm_m3=zeros(1,size(Xm,2));
for i=1:size(Xm,2)
    nrm_m3(i)=norm(Xm(:,i)-X_in2(:,i),2)/norm(X_in2(:,i),2);
end;
%}
coef_m2=pinv(Evec_m2(:,1:n_m2))*bsxfun(@minus,log(X_in2),mean(log(X_in),2));
Xm2=exp(repmat(mean(log(X_in),2),1,s(2))+(Evec_m2(:,1:n_m2)*coef_m2));
nrm_m3_2=zeros(1,size(Xm2,2));
for i=1:size(Xm2,2)
    nrm_m3_2(i)=norm(Xm2(:,i)-X_in(:,i),2)/norm(X_in(:,i),2);
end;


fprintf('Norms: %g\t\t%g\n',mean(nrm_n),mean(nrm_m3_2));
figure(-1+3*k); plot(nrm_n,'b-');hold on;%plot(nrm_m3,'r-');
plot(nrm_m3_2,'g-');hold off;
legend('PCA2_normal','PCA2_mult2');title('Fit to randomly perturbed new vectors');

%Plot function with fits to show behaviours
i=round(m2/2);
figure(3*k);plot(X_in(:,i),'b-');hold on;plot(X_in2(:,i),'r--');plot(Xn(:,i),'g-.');
    plot(abs(Xm2(:,i)),'k:');hold off;
legend('Original','Perturbed','Linear fit','Multiplicative fit');title('A typical
    vector with fits.')

end;
```