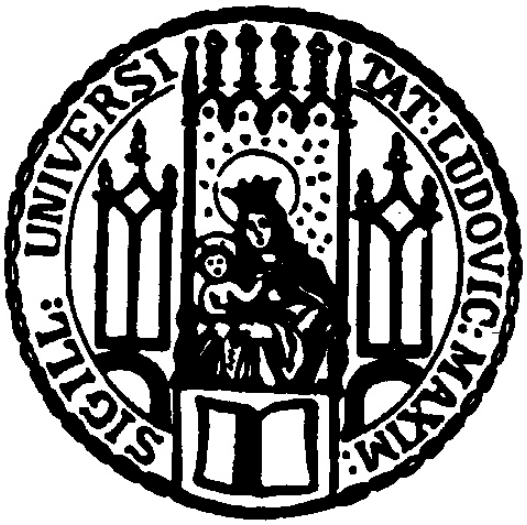
Ludwig-Maximilians-Universität München

Institut für Statistik



Analysis of Functional Phonetic Data

Analyse von funktionalen Daten aus der Phonetik

Master Thesis

vorgelegt von: Ivan Kondofersky

Studienbereich: Statistik

Betreuung: Prof. Dr. Friedrich Leisch

Contents

List of Figures

List of Tables

1. Introduction

Datasets produced in the field of phonetics have often a complex structure. Many of the variables are functional and therefore classic statistical approaches such as linear models or generalized linear models do not fulfill the arising demands.

In this thesis exploration techniques for functional data and functional linear models will be used to analyze phonetic data. Ramsay and Silverman have derived a bundle of methods for the analysis of functional data ([1], [2], [3]). This methods will be extended and adapted especially for the needs of phonetic data structures. Gubian et al. [4] and Gubian [5] used the available methods for analyzing phonetic data e. g. on a dataset where speakers pronounced the french word c'était. They used mainly functional principle component analysis. Applications of functional linear models with a functional response and non-functional covariates were further topics of discussion. This approach will be extended and modeling of functional response with functional covariates will be introduced as a standard technique in the analysis of phonetic data.

The derived methods and implemented functions will be used to analyze an actual phonetic dataset provided by the Institute of Phonetics and Speech Processing, University of Munich (Prof. Jonathan Harrington). The voice frequency and physiological movements of sensors placed in the mouth area of test speakers were recorded during the pronunciation of german sentences. Arising questions such as the influence of the tongue movements on the voice frequency are topics discussed in this thesis.

Moreover implemented functions are discussed in detail. The structure of the functions allows the analysis of further phonetic datasets without great rearrangements considering the software. Reducing computational time by sparse programming allows the application of functional models as an analysis technique on a regular basis.

2. Theoretical background of functional data analysis

This chapter provides an overview on functional data analysis. The theoretical background is required in order to improve understanding of the functions and models used in the later sections. Most of the theoretical structures in this chapter are introduced by Ramsay et al. [1, 2]. Some excerpts of the phonetic dataset (chapter 3) will be used to visualize the methods of this section.

2.1. Aspects of functional data

The term functional data describes a specific type of data in which each replication represents a part of a curve. This differs from most cases in which solely an ordinary data point is presented. Usually a curve is measured in equal or unequal (time) intervals and it is never possible to measure the exact curve. Of course the accuracy can be sharpened if e.g. the time intervals in which a new point of the curve is measured are reduced. However it is still impossible to examine the exact curve due to the difficulty of observing and storing an uncountable number of values for describing even the shortest interval of a real curve.
Figure 1 shows one observation from the phonetic dataset. The voice frequency of a test person pronouncing the vowel "A" was recorded in equal intervals of 0.005 seconds. The recorded points shown in Figure 1 are interpolated which might not be the correct displaying method of the real curve because the exact movement of the curve between two observation points could be much more complicated than a straight line.
The interpolation technique in this example shows just one possibility of displaying and approximating the curve in the unobserved region. More variable approximation properties can be achieved by using e.g. basis functions to represent the data.
Another feature of functional data is two "neighbor" data points having a higher correlation than two data points of the same curve that are further apart.
Moreover it is important to note that one observation or one replication represents a group of data points that all belong to the same curve. For example

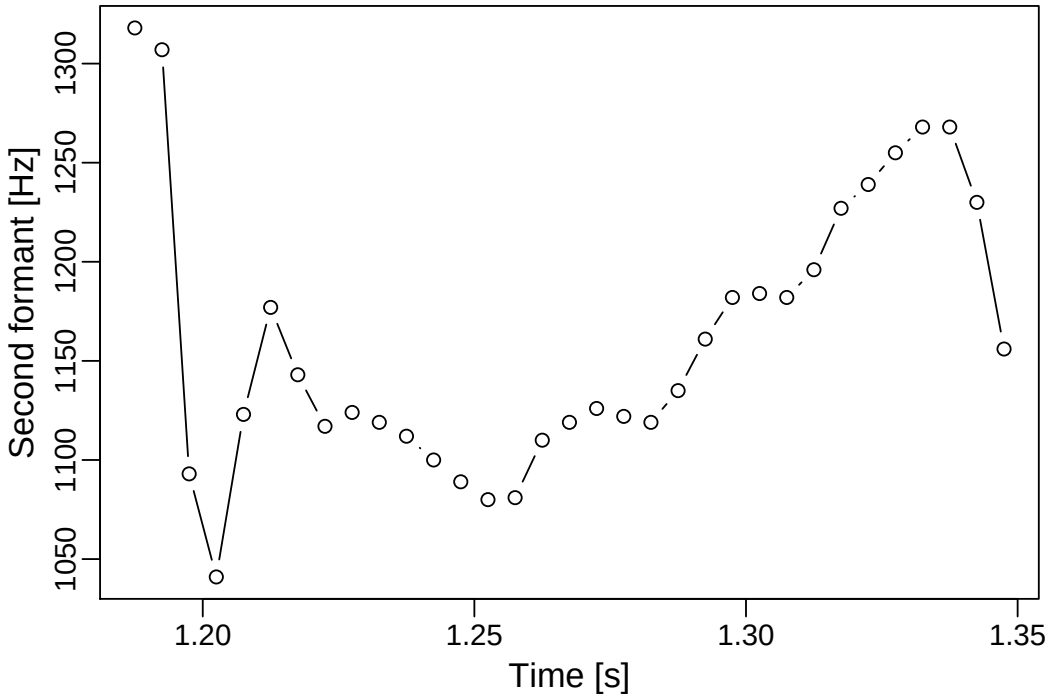


Figure 1: Sample voice frequency while pronouncing the vowel "A" in the word "gepape"

if we speak of 100 observations, the data matrix itself could contain 2000 lines if each curve is measured at 20 spots. This fast enlarging of functional data concerns the computational aspects of dealing with such huge samples so that sparse programming is an important keyword in this context.

2.2. From functional data to smooth functions

Turning raw discrete data into smooth functions is a complex task which can be solved by using linear combinations of basis functions. Many types of basis functions exist, each with different parameters that have to be adjusted. For instance one has to choose which order of polynomial is best for the given data by using polynomial bases. By choosing another alternative - the B-spline basis - one has to look for the best combination of order, knot placement and knot count in order to create the best approximation for the data. However the concept of approximating curves with basis functions remains the same. In

functional context the observed data vector $\mathbf{y} = (y_1, \ldots, y_n)$ of a curve can be expressed as follows:

$$y_j = x(t_j) + \epsilon_j \tag{2.1}$$

with x as a latent function or the data generating process that has to be approximated, $\mathbf{t} = (t_1, \ldots, t_n)$ as the times at which a snapshot of x is taken and $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_n)$ as a noise, error or some other disturbance on the real latent function.
Using vector notation for the same expression as in (2.1) leads to a much cleaner notation without using indices:

$$\mathbf{y} = x(\mathbf{t}) + \boldsymbol{\epsilon} \tag{2.2}$$

In this equation $\mathbf{y}$, $\mathbf{t}$ and $\boldsymbol{\epsilon}$ are all column vectors of the same length.
A basis function system is a set of known functions ϕ_k that are mathematically independent of each other. In general it is possible to approximate any function exactly just by taking a weighted sum from a sufficiently large number K basis functions. Of course exact representation of the curve is not desired in most cases, due to the large number of basis functions leading to a rise of the computational time and also to overfitting the model. The latent function x from (2.2) can be represented by basis functions.

$$x(t) = \sum_{k=1}^{K} c_k \phi_k(t) = \mathbf{c}'\boldsymbol{\phi} \tag{2.3}$$

As mentioned before, K describes the number of basis functions used and ϕ_k is a single basis function. The new parameter c_k is the k-th coefficient belonging to the k-th basis function. Instead of coefficient one can also use the term weight to express it. Figure 2 shows three different types of basis functions, all scaled on the interval $[0, 1]$. Panel A displays the B-spline basis functions of order $p = 4$. A property of the B-spline functions is the recursive definition of

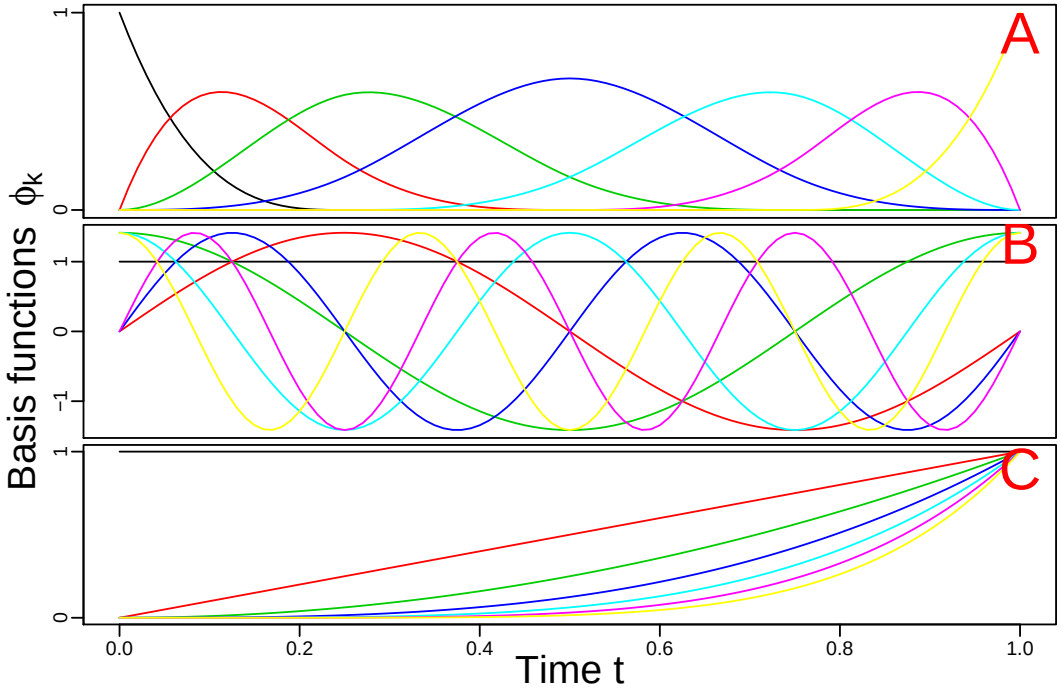


Figure 2: Seven basis functions scaled equally on the interval [0, 1]. A: B-spline, B: Fourier, C: Polynomial.

the different order of splines. The recursion formula from de Boor [6] can be used to compute the exact functions.

$$\begin{aligned} \phi_{k,1}(t) &= \begin{cases} 1 & \text{if} \quad t_k \leq t < t_{k+1} \\ 0 & \text{otherwise} \end{cases} \\ \phi_{k,p}(t) &= \frac{t - t_k}{t_{k+p-1} - t_k}\phi_{k,p-1}(t) + \frac{t_{k+p} - t}{t_{k+p} - t_{k+1}}\phi_{k+1,p-1}(t) \end{aligned} \tag{2.4}$$

Panel B displays a sequence of periodic Fourier basis functions (with parameter ω which determines the period $\frac{2\pi}{\omega}$):

$$\phi_k = \begin{cases} \cos(\frac{k-1}{2}\omega t) & \text{if} \quad k \quad \text{odd} \\ \sin(\frac{k}{2}\omega t) & \text{otherwise} \end{cases} \tag{2.5}$$

Panel C respresents a set of polynomial or also called monomial basis functions:

$$\phi_k(t) = t^{k-1} \tag{2.6}$$

All equations above use an index k with $k = 1, \ldots, K$.
Various different types of basis function systems exist, such as constant bases, wavelets, exponential bases etc. Each one has different advantages, e. g. the Fourier basis will have a better effect on periodic data, whereas the constant basis extremely simplifies the model. The most commonly used type of basis functions in literature is the B-spline basis. In this thesis the main focus concerns primarily the B-splines, however some usage of Fourier, polynomial and constant bases will be presented as well.
After considering the different possible basis functions, equations (2.2) and (2.3) can be combined as follows:

$$\begin{aligned} \mathbf{y} &= \sum_{k=1}^{K} c_k \phi_k(\mathbf{t}) + \boldsymbol{\epsilon} \\ &= \mathbf{c}'\boldsymbol{\phi} + \boldsymbol{\epsilon} \end{aligned} \tag{2.7}$$

It is interesting to point out the fact that the observed data $\mathbf{y}$ and the basis functions $\boldsymbol{\phi}$ in (2.7) are known and $\boldsymbol{\epsilon}$ can be assumed to have zero mean, as in other statistical analyses. Thus the unknown parameters are in the coefficient vector $\mathbf{c}$. In order to find the best set of coefficients $\mathbf{c}$ a use of the ordinary least squares criterion should be carried out.

$$LS(\mathbf{y} \mid \mathbf{c}) = \sum_{j=1}^{n} \Big[y_j - \sum_{k=1}^{K} c_k \phi_k(t_j) \Big]^2 \tag{2.8}$$

After defining the $(n \times K)$-matrix $\boldsymbol{\Phi}$, which contains the values of $\phi_k(t_j)$, equation (2.8) can be expressed with matrix notation.

$$\begin{aligned} LS(\mathbf{y} \mid \mathbf{c}) &= (\mathbf{y} - x(\mathbf{t}))'(\mathbf{y} - x(\mathbf{t})) \\ &= (\mathbf{y} - \boldsymbol{\Phi}\mathbf{c})'(\mathbf{y} - \boldsymbol{\Phi}\mathbf{c}) \end{aligned} \tag{2.9}$$

Simple algebraic operations such as taking the derivative of equation (2.9) and setting the resulting term to zero, lead to the set of coefficients that minimize the least squares criterion.

$$\begin{aligned} \frac{\partial LS(\mathbf{y} \mid \mathbf{c})}{\partial \mathbf{c}} &= 2\boldsymbol{\Phi}'\boldsymbol{\Phi}\mathbf{c} - 2\boldsymbol{\Phi}'\mathbf{y} \stackrel{!}{=} 0 \\ &\Leftrightarrow \quad \hat{\mathbf{c}} = (\boldsymbol{\Phi}'\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}'\mathbf{y} \end{aligned} \tag{2.10}$$

As mentioned earlier, this approach is based on the assumption that the residuals ϵ_j have a zero mean and are independent and identically distributed. Especially for functional data this assumption is often unrealistic due to the fact that two points that are close to each other are higher correlated than two points further apart. In order to deal with this kind of problem one can extend equation (2.9) by the matrix $\mathbf{W}$.

$$\begin{aligned} WLS(\mathbf{y} \mid \mathbf{c}) &= (\mathbf{y} - x(\mathbf{t}))'\mathbf{W}(\mathbf{y} - x(\mathbf{t})) \\ &= (\mathbf{y} - \mathbf{\Phi c})'\mathbf{W}(\mathbf{y} - \mathbf{\Phi c}) \end{aligned} \tag{2.11}$$

$\mathbf{W}$ is used to weight unequal squares and products of residuals. It should be taken into account that by using $\mathbf{W} = \mathbf{I}$ the weighted and unweighted least squares criterions are also equal. Using similar algebraic operations like the ones used in (2.10), the new estimated minimizer of (2.11) can be derived.

$$\hat{\mathbf{c}} = (\mathbf{\Phi}'\mathbf{W}\mathbf{\Phi})^{-1}\mathbf{\Phi}'\mathbf{W}\mathbf{y} \tag{2.12}$$

The next question arising is how to estimate $\mathbf{W}$. In the case of a known variance-covariance matrix Σ_ϵ of the residuals $\mathbf{W}$ is simply set to Σ_ϵ^{-1}. In the more general case of an unknown variance-covariance matrix, Σ_ϵ should be estimated from the data. Usually this involves an estimation of $\frac{n(n-1)}{2}$ different parameters which require a large number of observations, that are not given in most cases.

Techniques such as kernel smoothing and local polynomial fitting are other alternatives for the smoothing with basis functions. However a detailed description would go beyond the scope of this thesis.

After choosing an appropriate system of basis functions, a question considering the smoothness of the fitted curves arises. Two methods can be used in order to control the degree of smoothness for the fitted curves. On the one hand, the number of basis functions could be varied. Theoretically a sufficiently large number of basis functions lead to a perfect fit of every point sequence. This, however, is not desired in most cases since the error of the individual point would be disregarded. By choosing a moderate number of basis functions, a

good and smooth representation of the data can be achieved. Choosing a very small number of basis functions would yield almost constant lines which go through the mean value of the point sequence. On the other hand it is possible to use a roughness penalty approach in order to achieve smoothness. Usually the squared second derivative of the approximated curve is penalized, so large differences in the curvature of the fitted curve would lead to a higher least squares criterion. This automatically leads to smoother curves since the smoother a curve the smaller the (squared) second derivative. Smoothing with a roughness penalty is more effective and easier to handle than smoothing through increasing the number of basis functions. The roughness penalty approach can be controlled by just one parameter λ whereas the ideal number of basis functions depend on more parameters (knot placement, order of splines etc.).
In order to establish the roughness penalty approach to equation (2.12) the new penalization term should be added.

$$PENWLS_\lambda(x \mid \mathbf{y}) = (\mathbf{y} - x(\mathbf{t}))'W(\mathbf{y} - x(\mathbf{t})) + \lambda \underbrace{\int [D^2x(s)]^2 ds}_{\mathrm{PEN}(x)} \qquad (2.13)$$

In this equation λ stands for the smoothing parameter and $D^2(\cdot) = D^2\cdot$ denotes the second derivative of a function $(\cdot)$. The estimate of the function is obtained by finding the best $\hat{x}$ that minimizes $PENWLS_\lambda(x)$ over the space of functions x for which $\mathrm{PEN}(x)$ is defined. The smoothing parameter λ plays a key role in finding the estimate. If $\lambda \to \infty$, then the estimated function x will be a straight line that has no curvature at all and so $\mathrm{PEN}(x) = 0$. If $\lambda \to 0$ the new minimizer is a curve that fits the data points perfectly. However, even in this case the interpolating curve is not arbitrarily variable, because it is the smoothest twice differentiable curve that fits the data exactly.
Equation (2.13) has some unpleasant parts like the integral and taking the

derivative of a function. That is why some further attention should be attracted to PEN(x) before trying to derive the minimizer of (2.13).

$$\begin{aligned}
\text{PEN}(x) &= \int [D^2 x(s)]^2 ds \\
&= \int [D^2 \mathbf{c}'\phi(s)]^2 ds \\
&= \int (D^2 \mathbf{c}'\phi(s))(D^2 \mathbf{c}'\phi(s))' ds \\
&= \int D^2 \mathbf{c}'\phi(s) D^2 \phi(s)' \mathbf{c} ds \\
&= \mathbf{c}' \int D^2 \phi(s) D^2 \phi(s)' ds \mathbf{c} \\
&= \mathbf{c}' \mathbf{R} \mathbf{c} \qquad (2.14)
\end{aligned}$$

The usefull aspect of rephrasing the equation as in (2.14) is mainly because of the possibility of taking the derivative of PEN(x) without considering the complex structure of the new matrix $\mathbf{R}$, called the roghness penalty matrix.

$$\begin{aligned}
PENWLS_\lambda(x \mid \mathbf{y}) &= (\mathbf{y} - x(\mathbf{t}))' W (\mathbf{y} - x(\mathbf{t})) + \lambda \text{PEN}(x) \\
&= (\mathbf{y} - \mathbf{\Phi c})' \mathbf{W} (\mathbf{y} - \mathbf{\Phi c}) + \lambda \mathbf{c}' \mathbf{R} \mathbf{c} \\
\\
\frac{\partial PENWLS_\lambda(x \mid \mathbf{y})}{\partial \mathbf{c}} &= -2\mathbf{\Phi}'\mathbf{W}\mathbf{y} + 2\mathbf{\Phi}'\mathbf{W}\mathbf{\Phi}\mathbf{c} + 2\lambda\mathbf{R}\mathbf{c} \stackrel{!}{=} 0 \\
&\Leftrightarrow (\mathbf{\Phi}'\mathbf{W}\mathbf{\Phi} + \lambda\mathbf{R})\mathbf{c} \stackrel{!}{=} \mathbf{\Phi}'\mathbf{W}\mathbf{y} \\
&\Leftrightarrow \hat{\mathbf{c}} = (\mathbf{\Phi}'\mathbf{W}\mathbf{\Phi} + \lambda\mathbf{R})^{-1}\mathbf{\Phi}'\mathbf{W}\mathbf{y} \qquad (2.15)
\end{aligned}$$

Please note, that with $\lambda = 0$ and $W = I$ equation (2.15) equals the very first approach (2.10).
The roughness penalty matrix $\mathbf{R}$ has two difficult tasks that have to be considered before obtaining $\hat{c}$. First of all the basis functions $\phi(\mathbf{t})$ have to be differentiated with respect to $\mathbf{t}$. This goal is usually rather easy to accomplish if the system of basis functions represents the Fourier system (2.5), the polynomial system (2.6) or a number of other systems which are not mentioned in this thesis. In the case of a B-spline basis system differentiating is not trivial, however it can be derived by using the de Boor equations [6]. Having managed the differentiation problem, an integration difficulty appears immediately. Once again most of the systems of basis functions can handle integration very well. The integration of a B-spline function can not be written down, because

of an infinite number of basis functions would be needed [7]. More details on the numerical computation of the roughness penalty matrix are described by Ramsey et al. [1].

2.3. Explorative techniques for functional data

2.3.1. Mean, Variance and Correlation functions

After the accomplishments on how to represent discrete data by smooth functions, some explorative techniques will be presented in this chapter. Such techniques are important instruments which offer a possibility for the user to explore the data prior to more complicated analysis. Among the first things which a user does while analyzing new data is computing the mean and variance. The mean function is the average of the functions point-wise across replications.

$$\bar{x}(t) = \frac{1}{n}\sum_{i=1}^{n} x_i(t) \tag{2.16}$$

By using the mean function, the variance function is easily computed,

$$\mathrm{var}_x(t) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i(t) - \bar{x}(t))^2 \tag{2.17}$$

and the standard deviation function is the square root of the variance function.

$$\mathrm{sd}_x(t) = \sqrt{\mathrm{var}_x(t)} \tag{2.18}$$

Figure 3 shows the tongue tip movement while pronouncing the vowel “O” in context of the constructed word “gekoke” (phonetic data). Seven speakers pronounced the word in different sentences several times. Single observations are marked in gray color, whereas the mean function as well the mean function with added standard deviation are presented in red. By starting to speak out the “O”, the tongue tip is in a certain position (mean function at ca. $-3\ \mathrm{cm}^{-1}$). Subsequently it goes down reaching the minimum at approximately 60% of the recording time. Afterwards the tongue goes up again in order to pronounce the following consonant “K”. The mean function is a very useful summarizing method in this case. It smooths e.g. some time shifts on the individual level of the speakers by returning a clearly visible “U”-structure.

Going one step further in explorative analysis of functional data, it might be

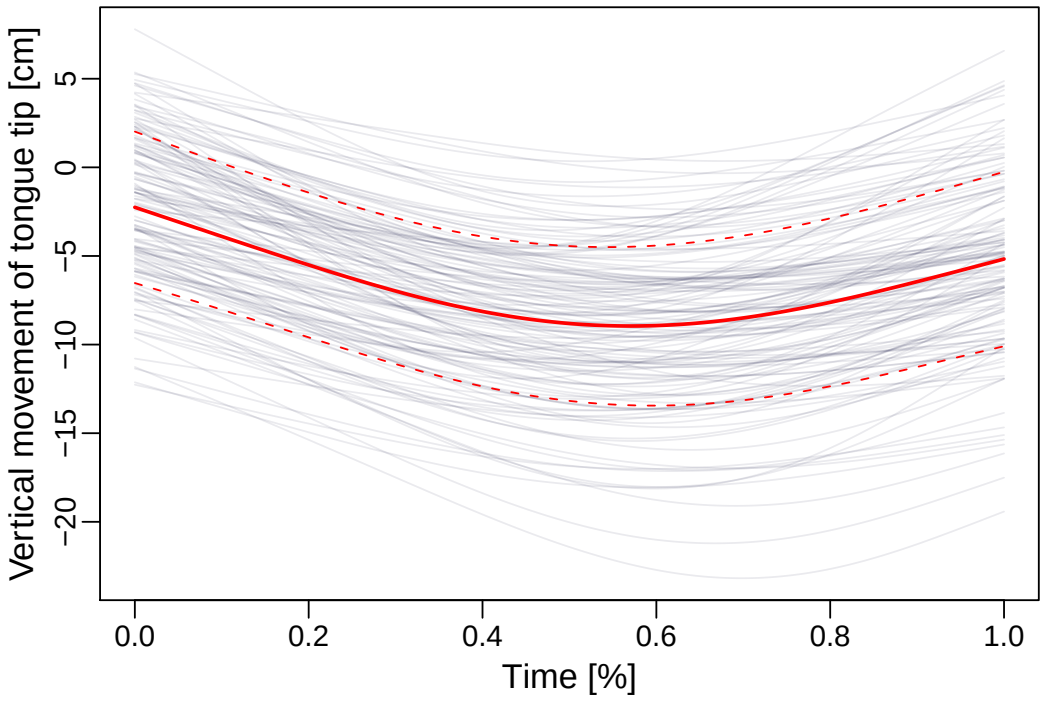


Figure 3: Several observations of the vertical tongue tip movement while pronouncing "O" in the constructed word "gekoke". Gray: single observations, red: mean function (solid line) and mean function with added standard deviation functions (dashed line)

interesting to inspect the covariances and correlations between two different time points contained in the data.

$$\mathrm{cov}_x(t_1, t_2) = \frac{1}{n-1} \sum_{i=1}^{n} (x_i(t_1) - \bar{x}(t_1))(x_i(t_2) - \bar{x}(t_2)) \tag{2.19}$$

$$\mathrm{cor}_x(t_1, t_2) = \frac{\mathrm{cov}_x(t_1, t_2)}{\sqrt{\mathrm{var}_x(t_1)\mathrm{var}_x(t_2)}} \tag{2.20}$$

2.3.2. Cross-variance and cross-correlation functions

Equations (2.19) and (2.20) are similar to their analogues from the multivariate data analysis which simplifies their usage. Thinking ahead to chapter 2.4 where functional linear models are being introduced, relations between different time points of one functional variable become intriguing alongside with relations

between two different functional variables. Ramsay et al. [1] refer to this type of dependency as cross-correlation or cross-covariance.

$$\mathrm{cov}_{x,y}(t_1, t_2) = \frac{1}{n-1} \sum_{i=1}^{n} (x_i(t_1) - \bar{x}(t_1))(y_i(t_2) - \bar{y}(t_2)) \tag{2.21}$$

$$\mathrm{cor}_{x,y}(t_1, t_2) = \frac{\mathrm{cov}_{x,y}(t_1, t_2)}{\sqrt{\mathrm{var}_x(t_1)\mathrm{var}_y(t_2)}} \tag{2.22}$$

Equations (2.19) to (2.22) all try to demonstrate some relations between different time points either for one ore two variables. Figure 4 presents the different types of correlation on two variables from the phonetic data. Panels A and D show correlation plots of the second formant (response variable) and the horizontal movement of the lower lip. Panels B and C show the cross-correlation functions which seem to be very similar. A more precise look at equation (2.22) suggests a notable conclusion.

$$\begin{aligned}
\mathrm{cor}_{x,y}(t_1, t_2) &= \frac{\mathrm{cov}_{x,y}(t_1, t_2)}{\sqrt{\mathrm{var}_x(t_1)\mathrm{var}_y(t_2)}} \\
&= \frac{1}{n-1} \sum_{i=1}^{n} (x_i(t_1) - \bar{x}(t_1))(y_i(t_2) - \bar{y}(t_2)) \frac{1}{\sqrt{\mathrm{var}_x(t_1)\mathrm{var}_y(t_2)}} \\
&= \frac{1}{n-1} \sum_{i=1}^{n} (y_i(t_2) - \bar{y}(t_2))(x_i(t_1) - \bar{x}(t_1)) \frac{1}{\sqrt{\mathrm{var}_y(t_2)\mathrm{var}_x(t_1)}} \\
&= \frac{\mathrm{cov}_{y,x}(t_2, t_1)}{\sqrt{\mathrm{var}_y(t_2)\mathrm{var}_x(t_1)}} \\
&= \mathrm{cor}_{y,x}(t_2, t_1)
\end{aligned}$$

This yields that both panels are transposes of one another at the diagonal $t_1 = t_2$. It also explains the similarity of both plots. The correlation plots first attract attention due to the second formant having a lower correlation between the beginning and the end of the time scale. The lowest correlation averages at about 0.8, and suggests that knowing the beginning of the curve, it's possible to make a good prediction on the curve position at the end of the measured time. The tongue movement has a considerably higher correlation at the edges. It barely decreases below 0.95. This demonstrates a lower variability of this functional variable. The cross correlation plots show a negative correlation

higher than -0.5 throughout the whole time scale. The strongest correlation is achieved at about 30% of the time scale and has a value of -0.65.

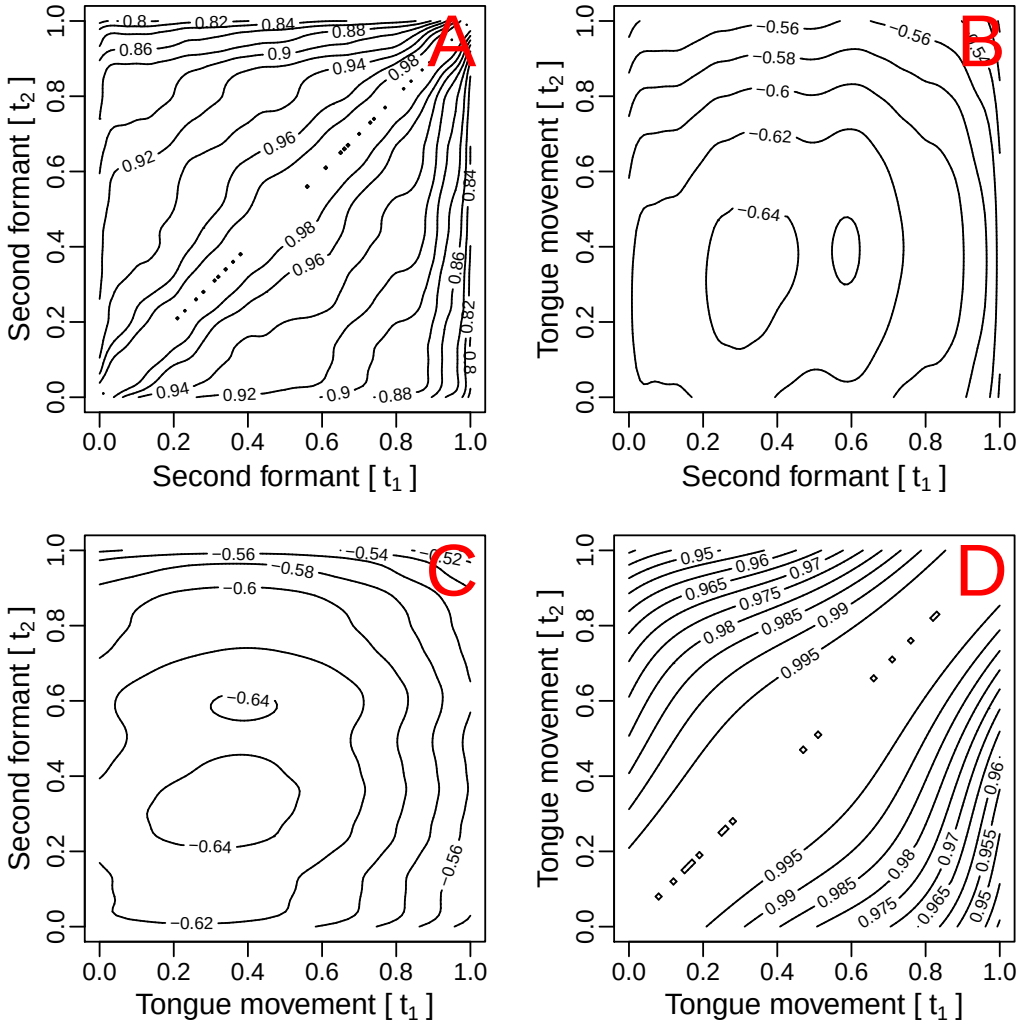


Figure 4: Correlations and cross-correlations of second formant and horizontal tongue back sensor. The second formant seems to have a higher variability than the tongue sensor. Both variables have a negative correlation near -0.6 throughout the time. The highest correlation is at about 30% of the time and is about -0.65

2.3.3. Principal Component Analysis for functional data

Another useful exploration technique in multivariate analysis is provided by principal component analysis (PCA). It can help discover some sources of variation which can not be recognised through analysis of variance-covariance structures. PCA of multivariate data consists of two major steps.

1. Finding the weight vector $\boldsymbol{\xi}_1$ for which

   $$f_{i1} = \boldsymbol{\xi_1}' x_i$$

   has the largest mean square $\frac{1}{n}\sum_i f_{i1}^2$ with respect to the constraint

   $$|| \boldsymbol{\xi}_1 ||^2 = 1$$

2. Finding the new weight vector $\boldsymbol{\xi}_m$, $(m = 2, \ldots, M)$ for which

   $$f_{im} = \boldsymbol{\xi_m}' x_i$$

   has the largest mean square $\frac{1}{n}\sum_i f_{im}^2$ with respect to the constraint

   $$|| \boldsymbol{\xi}_m ||^2 = 1$$

   and the $m-1$ additional constraints

   $$\boldsymbol{\xi}_k \boldsymbol{\xi}_m = 0, \quad k < m$$

By maximizing the mean square in step 1 the strongest source of variation is being identified and the constraint condition is needed to confirm that the resulting weight vector is well defined. In step 2 the strongest source of variation is examined once again. This time the new constraints asure that a different weight vector is identified because an orthogonal solution to all previous is required.
The values of the linear combinations f_{im} are called principal component scores and are often very useful in describing what these components of variation imply.
As mentioned earlier, the mean is an important aspect of data. Nevertheless an easy technique for its identification already exists. A subtraction of the

mean from each variable before applying PCA is recommended. Subsequently, maximizing the mean square of the principal component scores corresponds to maximizing their sample variance.
In order to use PCA for functional data, some mild modifications of the multivariate case need to be done. The term $\boldsymbol{\xi_m}'x_i$ uses x_i with discrete data points. Since this has to be adjusted for curves the term is being modified to

$$f_{im} = \int \boldsymbol{\xi}_m(s)x_i(s)ds.$$

The notation $|| \cdot ||^p$ denotes the norm in multivariate analysis. In functional context this also has to be computed with an integral so the new constraint becomes

$$|| \boldsymbol{\xi}_m ||^2 = \int \xi_m(s)^2 ds = 1.$$

Finally the orthogonality conditions become

$$\int \boldsymbol{\xi}_k(s)\boldsymbol{\xi}_m(s)ds = 0, \quad k < m.$$

Figure 5 illustrates PCA on simulated data. The functions $g_1(x) = -(x - 1.5)^2 + 1$, $g_2(x) = (x - 1.5)^2$ and $g_3(x) = 0$ were evaluated at 20 equally placed points on the interval $[0, 3]$. At each point a random normal distributed error with zero mean and standard deviation of 0.5 was added. $g_2(x)$ and $g_3(x)$ were evaluated 40 times, whereas $g_1(x)$ was evaluated 20 times. Consequently the sampled points were smoothed by a B-spline basis with 10 knots and an order of 4. The original functions g_1, g_2, g_3 are plotted with thick lines in the top left panel in different colors. The sampled curves form each function are displayed in the same color with thinner lines. The top right and bottom left panels show the first two principal components. The first component identifies the highest source of variability and is very similar to $g_2(x)$. Since $g_3(x)$ was constructed to have a very small variability and only 20 curves were sampled from $g_1(x)$, this was the expected principal component. The second principal component identifies a curve similar to $g_1(x)$. Due to the fact that the new solution has to be orthogonal to the previous one, it is obvious that the second strong source of variation originates from $g_1(x)$. Both principal components describe approximately 86% of the total variation, so further principal components are suppressed in this figure. The bottom right panel displays the principal

component scores f_{i1} and f_{i2}. As expected the green points corresponding to $g_2(x)$ have a high score on the first principal component and a value of nearly 0 for the second component. The red points which represent curves sampled from $g_1(x)$ have a high positive score on the second component and a high negative score on the first. This result is not surprising since a negative second

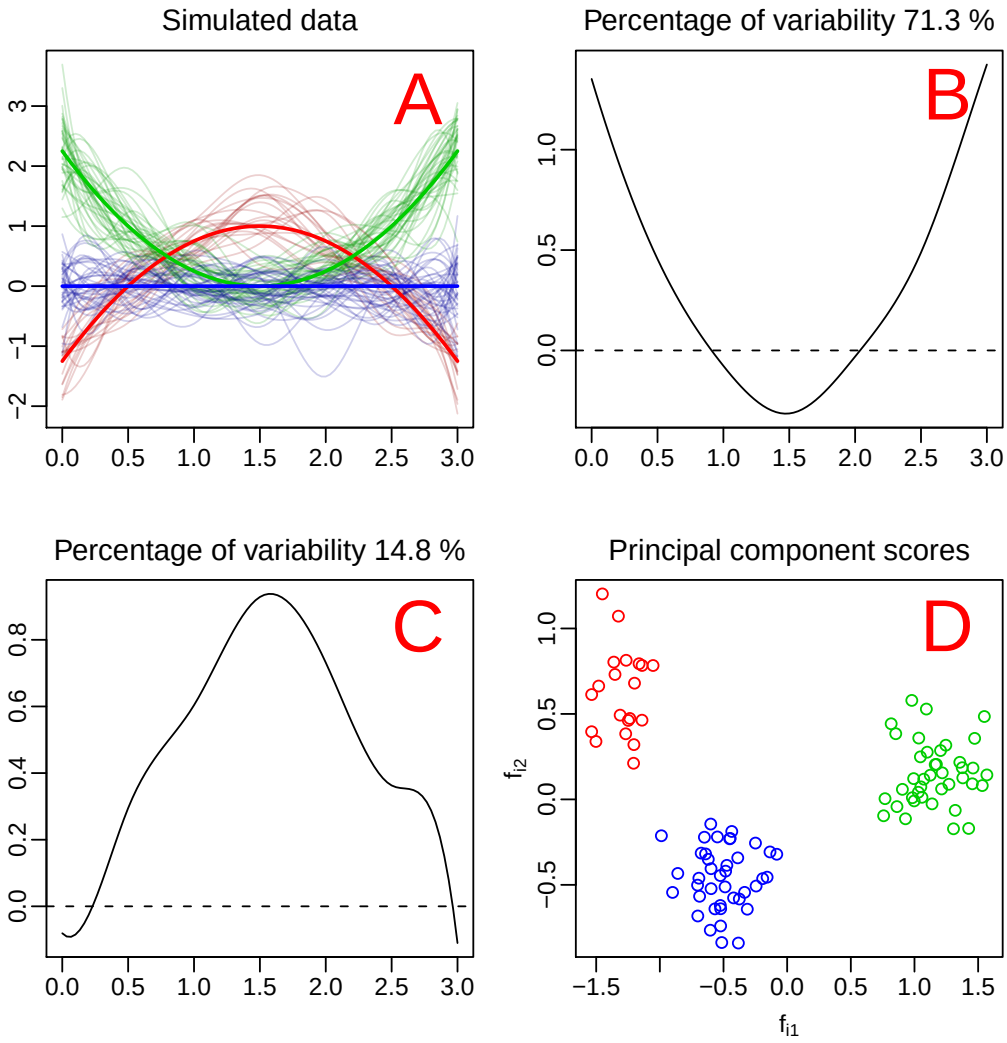


Figure 5: PCA on some simulated data. A: simulated functions, colors denote the three different sampling functions which consider errors; B: first principal component; C: second principal component; D: principal component scores.

principal component is very similar to the first principal component. The blue points corresponding to the constant function have scores near zero for both

principal components which indicates the accuracy of the presented method.

In conclusion instruments such as the mean function, the variance function and functional principal component analysis proved to be useful for the task of displaying and exploring functional data. They offer excellent possibilities for becoming acquainted with the data before making more complex analysis such as functional linear models. Moreover they can also be of great help in analyzing e. g. functional residuals produced in functional linear models. The next subsection provides an overview on the theory considering such models using many of the instruments described in the present subsection.

2.4. Functional linear models

2.4.1. Modeling functional variables

The inconstancy of a functional variable was explored without analyzing how much of its variation is explainable by other variables in the first part of this chapter. Consequently one should consider the use of covariates. In classical statistics, linear models serve this purpose so extending the notion of a linear model to the functional context is a highly recommended feature while analyzing functional data.
Ramsay et al. [1] provide an excellent overview on functional linear models. They distinguish three different cases in which an ordinary linear model is not sufficient.

1. Non-functional response and functional independent variables
2. Functional response and non-functional independent variables
3. Functional response and functional independent variables

Since the phonetic dataset has a functional response (second formant) and functional independent variables (tongue movements), the third and most complex case should be applied. The model can be expressed as follows:

$$\begin{aligned} \mathbf{y}(t) &= \mathbf{x}_1(t)\beta_1(t) + \ldots + \mathbf{x}_{p_1}(t)\beta_{p_1}(t) + \boldsymbol{\epsilon}(t) + \\ &\quad \beta_0 + \mathbf{x}_{p_1+1}\beta_{p_1+1} + \ldots + \mathbf{x}_{p_1+p_2}\beta_{p_1+p_2} \end{aligned} \tag{2.23}$$

Equation (2.23) is split into two lines. The upper line contains functional variables which are multiplied by functional coefficients and a functional error. The lower line contains the non-functional independent variables which can be of a numeric, categorical or constant type. They are multiplied by scalar coefficients. The notation in the equation above separates the independent covariates into functional and non-functional. A great simplification for the further equations is the fact that the scalar coefficients alongside with the non-functional covariates can be expressed as functional by using a system of constant basis functions.

$$\mathbf{x}_j\beta_j = x_{ij}\beta_j = x_{ij}\beta_j \cdot 1(t) \tag{2.24}$$

$1(t)$ represents a constant function that equals 1 for each t. After fitting the model, scalar coefficients can be extracted easily from equation (2.24). Equation (2.23) can be formulated as follows (the intercept term is suppressed for reasons of simplicity):

$$\begin{aligned} \mathbf{y}(t) &= \mathbf{x}_1(t)\beta_1(t) + \ldots + \mathbf{x}_p(t)\beta_p(t) + \boldsymbol{\epsilon}(t) \\ &= \sum_{j=1}^{p} \mathbf{x}_j(t)\beta_j(t) + \boldsymbol{\epsilon}(t) \\ &= \mathbf{X}(t)\boldsymbol{\beta}(t) + \boldsymbol{\epsilon}(t) \end{aligned} \tag{2.25}$$

Since the coefficients are functions that depend on t, a basis function expansion has to be estimated for each β_j. This involves choosing a type of basis function and a roughness penalty as well. Since some of the coefficient functions could have a different degree of smoothness than others a penalty term λ_j corresponding to each β_j has to be adjusted. Consequently a roughness penalty definition evolves for each basis function separately.

$$\mathrm{PEN}_j = \lambda_j \int \left[D^2\beta_j(t)\right]^2 dt \tag{2.26}$$

for each basis function separately. Therefore the functional fitting criterion becomes

$$LMSSE(\boldsymbol{\beta}) = \int \mathbf{r}(t)'\mathbf{r}(t)dt + \sum_{j=1}^{p} \mathrm{PEN}_j \tag{2.27}$$

with

$$\mathbf{r}(t) = \mathbf{y}(t) - \mathbf{X}(t)\boldsymbol{\beta}(t) \tag{2.28}$$

It is possible to choose different types of basis systems for each coefficient function β_j. The number of basis functions associated with each β_j can vary as well. For instance one may choose a high number of basis functions for an independent covariate that is assumed to have a high frequency variability. By choosing a lower number of basis functions only some general effects of a functional predictor can be described. Assuming that the j-th covariate is approximated by K_j basis functions, $\beta_j(t)$ can now be expressed as follows:

$$\beta_j(t) = \sum_{j=1}^{K_j} b_{kj}\theta_{kj}(t) = \boldsymbol{\theta}_j(t)'\mathbf{b}_j \tag{2.29}$$

Equations (2.25) and (2.27) can be presented in matrix notation which leads to cleaner and shorter expressions. Subsequently the construction of some super matrices is required. First the total number of used basis functions is defined as follows:

$$K_\beta = \sum_{j=1}^{p} K_j$$

The construction of vector $\mathbf{b}$ with length K_β is achieved by stacking the j shorter vectors $\mathbf{b}_j$ verticaly.

$$\mathbf{b} = (\mathbf{b}_1', \mathbf{b}_2', \ldots, \mathbf{b}_p')' \tag{2.30}$$

The $p \times K_\beta$ matrix Θ contains the basis functions and has a block diagonal form:

$$\Theta = \begin{bmatrix} \boldsymbol{\theta}_1' & 0 & \ldots & 0 \\ 0 & \boldsymbol{\theta}_2' & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & \boldsymbol{\theta}_p' \end{bmatrix} \tag{2.31}$$

Using the long constructs (2.30) and (2.31) the functional model (2.25) can now be expressed in matrix notation:

$$\mathbf{y}(t) = \mathbf{X}(t)\Theta(t)\mathbf{b} + \boldsymbol{\epsilon}(t) \tag{2.32}$$

Combining the two matrices $\mathbf{X}(t)$ and $\Theta(t)$ allows a reformulation of the model to

$$\mathbf{y}(t) = \mathbf{X}^*(t)\mathbf{b} + \boldsymbol{\epsilon}(t) \tag{2.33}$$

This last reformulation does not contribute much to the simplification of the model due to the fact that the design matrix now contains K_β columns instead of p which is usually much shorter. The main reason for reformulating the model as in (2.33) is showing that a functional linear model in fact has K_β parameters that have to be adjusted. Since $\mathbf{y}$ is also a functional variable the degrees of freedom of the error ϵ can be derived. Assuming that each functional replication y_i is approximated by a system of K_y basis functions df_ϵ becomes

$$\mathrm{df}_\epsilon = n \cdot K_y - K_\beta \tag{2.34}$$

The roughness penalties PEN_j are the last thing to consider before deriving an estimate for the functional linear model. Once again a super matrix is constructed. It has a symmetric block diagonal form and dimensions $K_\beta \times K_\beta$:

$$\mathbf{R} = \begin{bmatrix} \mathbf{R}_1 & 0 & \dots & 0 \\ 0 & \mathbf{R}_2 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \mathbf{R}_p \end{bmatrix} \tag{2.35}$$

with

$$\mathbf{R}_j = \lambda_j \int D^2\boldsymbol{\theta}_j(t) D^2\boldsymbol{\theta}_j(t)' dt \tag{2.36}$$

The bigger version of the matrices and vectors required to estimate the set of coefficients minimizing the least squares criterion (2.27) can now be used for deriving $\hat{\mathbf{b}}$. Since the basis functions $\boldsymbol{\theta}_j$ which define the coefficient functions

β_j are known, estimating $\hat{\mathbf{b}}$ would lead immediately to $\hat{\beta}_j$. Consequently the new fitting criterion becomes $LMSSE(\mathbf{b})$.

$$\begin{aligned} LMSSE(\mathbf{b}) &= \int (\mathbf{y}(t) - \mathbf{X}(t)\Theta(t)\mathbf{b})'(\mathbf{y}(t) - \mathbf{X}(t)\Theta(t)\mathbf{b})dt + \mathbf{b}'\mathbf{R}\mathbf{b} \\ &= \int \mathbf{y}(t)'\mathbf{y}(t) - 2\mathbf{b}'\Theta(t)'\mathbf{X}(t)'\mathbf{y}(t) + \\ &\quad +\mathbf{b}'\Theta(t)'\mathbf{X}(t)'\mathbf{X}(t)\Theta(t)\mathbf{b}dt + \mathbf{b}'\mathbf{R}\mathbf{b} \\ &= \int \mathbf{y}(t)'\mathbf{y}(t)dt - 2\mathbf{b}' \int \Theta(t)'\mathbf{X}(t)'\mathbf{y}(t)dt + \\ &\quad +\mathbf{b}' \int \Theta(t)'\mathbf{X}(t)'\mathbf{X}(t)\Theta(t)dt\mathbf{b} + \mathbf{b}'\mathbf{R}\mathbf{b} \end{aligned} \tag{2.37}$$

Differentiating equation (2.37) with respect to $\mathbf{b}$ and setting the resulting term to $\mathbf{0}$ leads to $\hat{\mathbf{b}}$:

$$\begin{aligned} &\frac{\partial LMSSE(\mathbf{b})}{\partial \mathbf{b}} = \\ &-2 \int \Theta(t)'\mathbf{X}(t)'\mathbf{y}(t)dt + 2 \int \Theta(t)'\mathbf{X}(t)'\mathbf{X}(t)\Theta(t)dt\mathbf{b} + \mathbf{R}\mathbf{b} \stackrel{!}{=} \mathbf{0} \\ &\Leftrightarrow \Big[\int \Theta(t)'\mathbf{X}(t)'\mathbf{X}(t)\Theta(t)dt + \mathbf{R}\Big]\mathbf{b} = \int \Theta(t)'\mathbf{X}(t)'\mathbf{y}(t)dt \\ &\Leftrightarrow \hat{\mathbf{b}} = \Big[\int \Theta(t)'\mathbf{X}(t)'\mathbf{X}(t)\Theta(t)dt + \mathbf{R}\Big]^{-1}\Big[\int \Theta(t)'\mathbf{X}(t)'\mathbf{y}(t)dt\Big] \end{aligned} \tag{2.38}$$

The amount of numerical integration involved in these expressions is quite manageable. The most complicated structure is enclosed in the integral $\int \Theta(t)'\mathbf{X}(t)'\mathbf{X}(t)\Theta(t)dt$. In this expression the inner product of two basis functions $\boldsymbol{\theta}_j$ and $\boldsymbol{\theta}_k$ is weighted by some scalar combination of data points. Computing these inner products by numerical integration should be almost a routine procedure.

Equation (2.38) is an important result of functional data analysis. It can be considered as the most commonly used equation for analyzing functional data since it contains all aspects that have to be considered in the functional linear model. It is interesting to point out that equation (2.33) describes a very specific type of dependency between $\mathbf{y}(t)$ and $\mathbf{X}(t)$. Since $\mathbf{X}$ only influences $\mathbf{y}(t)$ through its value $\mathbf{X}(t)$ at time t it is often called a concurrent or point-wise model. Hastie and Tibshirani refer to a slightly different version of this model as a varying coefficient model [8]. Further improvements of the model could

involve bivariate coefficient functions which describe an influence of the independent variables on the response over a wide interval of time. Although such models have a much higher number of coefficient parameters to fit and a much more complex becoming interpretation they can improve the quality of the model. The main reason why they are not considered in this thesis is the fact that phonetic data and especially the response variable (second formant) has a low variability regarding time. Figure 4 shows the high correlation between the beginning and the end of the time scale of the response variable. This suggests another reason for using the advantageous point-wise model (2.33). Consequently the final estimate becomes:

$$\hat{\mathbf{y}}(t) = \mathbf{X}(t)\hat{\boldsymbol{\beta}}(t) = \mathbf{X}(t)\Theta\hat{\mathbf{b}} \tag{2.39}$$

2.4.2. Assessing goodness of fit

In the last subsection a method on how to find an estimate for the coefficient functions was presented. This estimate is found by minimizing the quadratic distance (2.37). The next step in data analysis is qualifying this fit. In classical statistics a commonly used criterion is the coefficient of determination R^2 as summarized e. g. by Nagelkerke [9]. Taking this approach into account a functional type of determination coefficient can be derived. It is possible to extend the concept of assessing the goodness of fit by R^2 in three ways:

1. Computation of n different R_i^2 for each replication which determines the quality of fit for the single observation
2. Computation of a function $R^2(t)$ which qualifies the fit of the model over time
3. Computation of a global R^2 which qualifies the total fit of the model

The first approach ascertain which observations can be well explained by the fitted model and which not.

$$R_i^2 = 1 - \frac{\int(\hat{y}_i(t) - y_i(t))^2 dt}{\int(y_i(t) - \bar{y}(t))^2 dt} \tag{2.40}$$

A variation from classical statistics is that the single observation can produce values of R_i^2 which negative. This means that for those observations the mean

function $\bar{y}(t)$ represents a better fit than the predicted values from the model. The second approach determines the quality of the fit concerning the time scale. By using $R^2(t)$ it is possible to discover areas on the time scale in which observations are predicted very accurately as well as areas in which the fit is less reliable.

$$R^2(t) = 1 - \frac{\sum_{i=1}^{n}(\hat{y}_i(t) - y_i(t))^2}{\sum_{i=1}^{n}(y_i(t) - \bar{y}(t))^2} \tag{2.41}$$

Usually areas with high variation of the response function are expected to have a lower R^2 value. Smoother areas of a function which are easier to fit should have a higher coefficient of determination.
The third approach determines the quality of the model and is defined as the mean of $R^2(t)$ over the time t. It is being computed numerically by evaluating $R^2(t)$ on a sufficiently large number N_R of equally placed times. Afterwards the mean of those values is to be computed.

$$R^2 = \frac{1}{N_R}\sum_{i=1}^{N_R} R^2(t_i) \tag{2.42}$$

Another aspect of functional data analysis is that functional covariates are associated rather with coefficient functions than with scalar coefficients. In classical data analysis it is possible to measure the significance of the effects which covariates imply the response variable. In functional data analysis such techniques can not be applied. In order to anyhow assess the effects of the coefficient curves bootstrap techniques can be applied. One of the most basic bootstrap procedures is deriving percentile intervals. The following algorithm describes the steps required to derive such intervals.

Bootstrap percentile intervals for coefficient functions

1. Let $\mathbf{z}_i(t)$ be the combination of the i-th response and covariates $(y_i(t), x_{1i}(t), \ldots, x_{pi}(t))$ and let $\mathbf{Z}$ be the whole data matrix containing all response and covariate functions
2. Draw B bootstrap samples $\mathbf{Z}^1, \ldots, \mathbf{Z}^B$ with repetition
3. Compute $\hat{\boldsymbol{\beta}}^1, \ldots, \hat{\boldsymbol{\beta}}^B$ with the associated data

4. Evaluate each $\hat{\boldsymbol{\beta}}^b$ at a sufficiently large number N_β of time points and store the values in a $N_\beta \times p$ matrix

5. Sort each column separately and extract the $\lceil B \cdot \alpha \rceil$-th and the $\lfloor B \cdot (1-\alpha) \rfloor$-th from the sorted sequence of values

6. The two developed vectors describe the lower and upper point-wise $(1-2\alpha)$-confidence limits

---

The bootstrap procedure described above is new in the field of functional data analysis, so default values for the hyperparameters N_β, B and α do not exist. Generally it is a good idea to choose a moderate number of evaluation points such as $N_\beta = 100$. This leads to vectors which if plotted, describe a smooth curve. The number of bootstrap samples B depends on the confidence level α. If one chooses a confidence level of 0.005, an appropriate choice for the bootstrap samples would be $B \geq 1000$. By choosing an even lower confidence level such as 0.01, $B \geq 10000$ should be considered.

Looking at the residual functions a further way of assessing the goodness of fit is presented. As obvious from e. g. (2.32) the residuals produced by the model are functional. This fact complicates the usage of the residuals as goodness of fit instrument. Nevertheless the usage of the residual functions is still possible if the original data from which the functional observations were approximated is still available. In this case a residual sum of squares (RSS) can be computed. It can be used to compare two different models fitted with the same original data, due to a lower RSS denoting a better fit to the data.

3. Phonetic Dataset

3.1. Data collection

The dataset of interest in this thesis is presented by the Institute of Phonetics and Speech Processing, University of Munich (Prof. Jonathan Harrington). Seven test speakers were asked to pronounce german sentences. While pronouncing, the voice frequency and tongue, lip and jaw movements were

Table 1: Frequency of the pronounced words for each speaker. Column names denote the abbreviations of the different speakers.

	bk	ck	fs	hp	ht	mh	ta
gepape	19	20	20	22	20	20	20
gepepe	20	20	21	19	20	20	20
gepipe	20	20	21	21	20	19	20
gepope	20	19	22	22	20	19	20
gepupe	20	22	22	16	20	19	20
gepype	20	20	20	20	20	20	20
getate	20	22	21	21	20	20	20
getete	21	20	20	20	20	20	20
getite	19	21	21	21	20	19	20
getote	19	24	22	20	20	21	20
getute	19	20	20	21	20	20	20
getyte	20	20	20	21	20	20	20
gekake	20	20	20	23	21	22	21
gekeke	20	22	21	22	20	20	19
gekike	19	20	22	21	20	20	20
gekoke	20	21	20	20	20	19	20
gekuke	19	22	21	21	20	21	20
gekyke	20	22	21	20	20	20	20

recorded. The spoken sentences were varied in only one word and were all of the type “Ich habe geCVCe gesagt.”, which can be translated as “I said geCVCe.”. The letters written in uppercase in the target word “geCVCe” are placeholders for consonants (C) and vowels (V). The three different consonants K, T, P and six different vowels U, I, A, O, Y, (german umlaut “Ü”) and E were used to construct a total of 18 different artificial words (e. g. gekoke, getate etc.). The words before and after the target word were used to distract

the test person in order to reduce the concentration on the target word. Each speaker repeated one sentence several times, varying e.g. speed and tensity. Table 1 summarizes the frequency of the 18 words corresponding to the speaker. The minimum frequency of the words is for the word “gepupe” pronounced 16 times by speaker “hp”. Most of the words were pronounced roughly about 20 or 21 times. The total number of replications amounts to 2556. The voice frequency provides multiple formants. For the data analysis the second formant is to be used as the functional response. It was recorded in equal intervals of 0.005 seconds (see e. g. figure 1). Only the recorded frequency of the vowels in the artificial word “geCVCe” will be used. The pronunciation of a single letter depends on e. g. the speed of speech or the test person. Since the formant is recorded in equal intervals, each replication has a different length. For example, if the vowel “A” is spoken in 0.1 seconds, a total of 20 discrete data points would be recorded.
In addition to the voice frequency physical movements of sensors, placed in the mouth area of the speakers were recorded. For this purpose six sensors

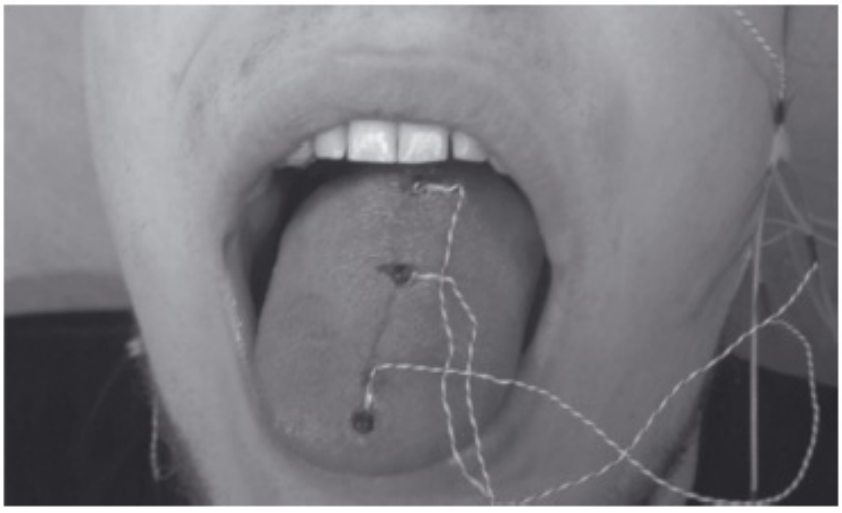


Figure 6: The tongue tip, tongue mid and tongue back sensors glued with dental cement to the surface of the tongue.

were responsible for physical movement measures. These were glued with dental cement to the tongue tip (TTIP), in the middle of the tongue (TMID), at the back of the tongue (TBACK), at the tongue dorsum which is further back than the TBACK sensor (TDORS), at the lower lip (LLIP) and on the jaw (JAW). All movements were recorded by the so-called “5D” system that

has been developed at the IPS Munich ([10], [11]). Figure 6 shows the mouth area of a speaker with three sensors already attached to the tongue. Each of these sensors produce one vertical and one horizontal record, except for the jaw sensor which measures only in the horizontal direction. This means that a total of 11 functional objects are being generated by pronouncing one sentence. Contrary to the voice frequency these functional objects are recorded every 0.004 seconds. This means that if a speaker needs 0.1 seconds to speak out a letter, 25 discrete data points corresponding e.g. to the horizontal tongue movement will be recorded.
A distinguishment between the time the voiced vowel is spoken and the aspiration phase shortly before the vowel begins is possible as well. In this case both types of functional data – voice frequency and physical movements – show great differences. On the one hand, the physical movements are being recorded very accurately and reliably in both phases. On the other hand the voice frequency is difficult to measure in the aspiration phase due to lack of any sound during this period. A marker called "von" is attached to the data, so the exact time of the beginning of the viced vowel can be computed.

3.2. Data modifications

After receiving the data from IPS Munich, mild modifications were performed on some of the replications. First of all, 13 observations were discarded due to a very low amount of discrete data per replication (less than four data points). These records seemed unrealistic because they would mean the pronunciation of a vowel in less than 0.02 seconds. After removing them the new sample size decreased to 2543 observations. The second modification involved the correction of "zero errors". Such kind of errors are based on a problem regarding the recording of the voice frequency. In such cases the frequency drops without an explainable reason for a short period of time to zero and goes up to the previous level afterwards. This kind of errors occurred in approximately 10% of the obtained data. The easiest way of correcting such errors is to manually interpolate the two non-zero data points which surround the "zero error". If the error is at the end or the beginning of the spoken vowel than a straight line between the nearest non-zero point and the end / the beginning is interpolated.

4. Results

This chapter gives an insight into a functional linear model with voice frequency as a response as well as different functional and non-functional covariates is fitted according to the data. For functional covariates coefficient curves are adapted and non-functional covariates are associated with scalar coefficients. Bootstrapping is used to compute confidence intervals of all covariates. This procedure helps classifying the importance of a covariate for the response. In addition the coefficients of the best fitted model are used to predict the voice frequency in the aspiration phase prior to the voiced vowel. As mentioned in chapter 3, the voice frequency in the aspiration phase is very unstable and can not be recorded properly. Contrariwise the covariates are all recorded properly in the aspiration phase so they can be well used for a prediction.

4.1. Functional linear models with response voice frequency

4.1.1. Data preparation - finding the best set of hyperparameters suited for the phonetic dataset

Table 2 summarizes the available data for the functional linear models. A total of 11 functional and 3 non-functional, categorical covariates can be used to predict the functional response.
The first step towards a functional linear model is choosing a common time scale on which the data is recorded. The original time scale for the phonetic data depends on several different circumstances, such as the time at which the speaker is starting the sentence. To eliminate such differences a time interval of $[0, 1]$ is chosen for the data. These time values can be interpreted as a percentage with 0 representing the beginning of the spoken vowel and 1 as the end of the time scale.
Furthermore discrete data has to be prepared. In order to represent discrete data by smooth curves, the proper system of basis functions and smoothing parameters have to be chosen. In general it is possible to choose a different basis function and smoothing parameter for each functional variable. This would involve many different possibilities due to the variety of additional hyperparameters for each basis function such as knot count, order and so forth. Figure 7 displays the sixth replication of the data as already presented in figure 1.

Table 2: Phonetic dataset: available covariates and response for the functional linear model

Name	Description	Type
FM	Voice frequency (second formant)	functional
TTIPX	Tongue tip sensor in horizontal direction	functional
TTIPY	Tongue tip sensor in vertical direction	functional
TMIDX	Middle tongue sensor in horizontal direction	functional
TMIDY	Middle tongue sensor in vertical direction	functional
TBACKX	Tongue back sensor in horizontal direction	functional
TBACKY	Tongue back sensor in vertical direction	functional
TDORSX	Tongue dorsum sensor in horizontal direction	functional
TDORSY	Tongue dorsum sensor in vertical direction	functional
LLIPX	Lower lip sensor in horizontal direction	functional
LLIPY	Lower lip sensor in vertical direction	functional
JAWX	Jaw sensor in horizontal direction	functional
s.l	Speaker	non-functional
k.l	Consonant	non-functional
t.l	Tensity	non-functional

In figure 7 the time scale is set to $[0, 1]$ and different alternatives for representing the data with a curve are added. The red curve which is based on a B-spline basis with 5 interior knots and an order of 4 smooths the data excessively and does not make a good representation of the data especially in the first half of the time scale. In contrast the green curve (15 interior knots) is hardly smoothing the data, so almost every single discrete point is being fitted without any error. The third blue curve is based on a constant basis. It runs exactly through the mean of the second formant for this replication.

Having chosen the right combination of basis functions with appropriate hyperparameters, an improvement can be achieved by using smoothing parameters. Theoretically, each functional variable, regardless whether it is a response or a covariate, could have a different smoothing parameter. This would lead to a high number of possible variations. A good way to reduce some source of variation is to choose only two different smoothing parameters – one for the functional response and one for the functional covariates.

A further possibility to improve the quality of the model fit is a variation of the parameters corresponding to the coefficient curves. Since those are also

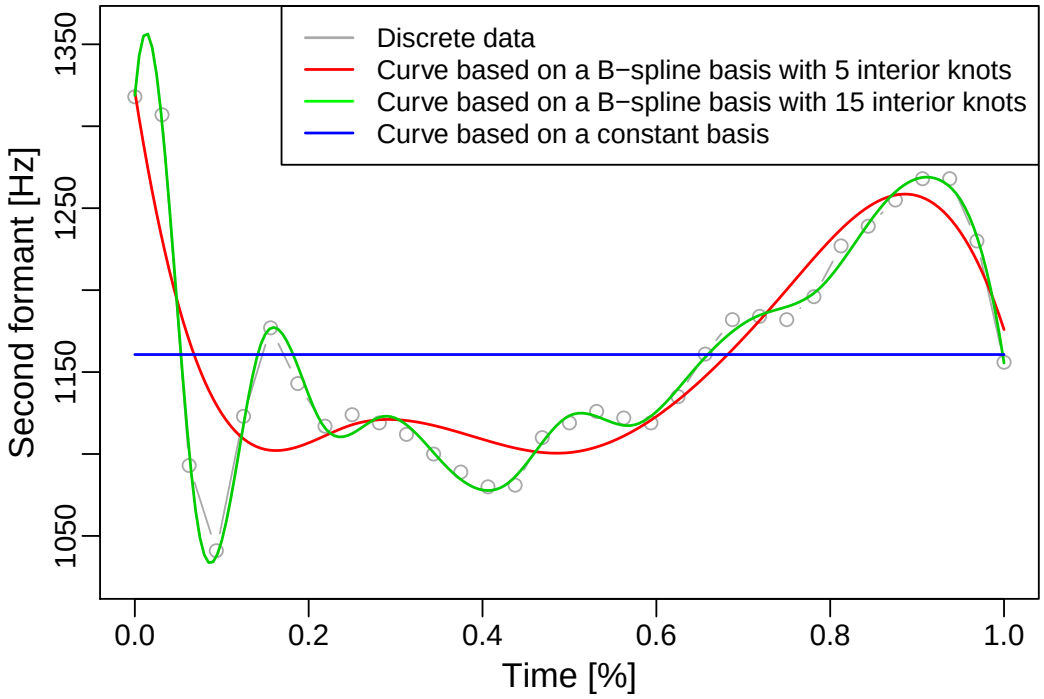


Figure 7: Three different types of basis functions used to represent discrete data by curves.

curves, the same technique with basis functions and smoothing parameter can be used. In order to create comparable coefficient curves and reduce the varied parameters, the basis system and smoothing parameter should be specified for all functional covariates.

This great discrepancy created by the large number of possible variations is one problem that occurs while modeling the data. Similar functional analysis lack in the literature especially for phonetic datasets. This is a further reason for choosing default parameters cautiously. One possible way of finding the best set of parameters that suit the given data is making a simulation and trying different combinations. Preliminary tests suggested B-spline basis functions and smoothing parameters in the range of $[0, 1]$. Table 3 outlines the varied parameters. The last row of the table summarizes the number of different variations per column. In order to try every single possible combination $16 \cdot 4 \cdot 4 \cdot 16 \cdot 4 = 16384$ different functional linear models should be computed. Each model has 11 different functional covariates and a scalar intercept that have to be adjusted. In order to qualify the adapted models two different measures are worth considering. On the one hand the discrete data could be used

Table 3: Simulation design phonetic dataset: finding the best set of coefficients.

Basis functions data			λ_y	λ_x	Basis functions coefficients			λ_{coef}
Type	Order	Knots			Type	Order	Knots	
B-spline	2	5	10^{-1}	10^{-1}	B-spline	2	5	10^{-1}
B-spline	2	10	10^{-3}	10^{-3}	B-spline	2	10	10^{-3}
B-spline	2	15	10^{-5}	10^{-5}	B-spline	2	15	10^{-5}
B-spline	3	5	10^{-7}	10^{-7}	B-spline	3	5	10^{-7}
B-spline	3	10			B-spline	3	10	
B-spline	3	15			B-spline	3	15	
B-spline	4	5			B-spline	4	5	
B-spline	4	10			B-spline	4	10	
B-spline	4	15			B-spline	4	15	
Polynomial	3	-			Polynomial	3	-	
Polynomial	5	-			Polynomial	5	-	
Polynomial	7	-			Polynomial	7	-	
Fourier	3	-			Fourier	3	-	
Fourier	5	-			Fourier	5	-	
Fourier	7	-			Fourier	7	-	
Constant	-	-			Constant	-	-	
Total number of variations:								
16			4	4	16			4

to compute a residual sum of squares. This is achieved by using the squared distance between the discrete data point and the fitted curve evaluated on the same time scale as the discrete data point. Instead of the squared distance one can also use the absolute value (RSA). On the other hand an approach borrowed from classical statistics would be the comparison of R^2 values. This approach fails however for this type of data because systems of constant basis functions produced the highest coefficients of determination. The structure of (2.41) and (2.42) suggests that a constant line is much easier to predict than a possibly high variable curve.

The models were computed on a computer with a 2.00 GHz CPU and 4 GB RAM. The computation time was 8325 minutes which equals approximately 140 hours. Most of the time was used for turning the discrete data points into curves. The computation of the functional linear models was much faster and took about 5% of the computation time.

After examining the results of the simulation, the variation of the parameters of the coefficient curves proved to play a minor role in decreasing the residual sum of squares. The combination of the remaining parameters however showed

Table 4: Simulation 1: Ten parameter combinations with lowest residual sum of absolute values.

Basis functions data			λ_y	λ_x	Basis functions coefficients			λ_{coef}	RSA
Type	Order	Knots			Type	Order	Knots		
B-spline	3	10	10^{-7}	10^{-1}	B-spline	4	15	10^{-1}	$5.1657 \cdot 10^5$
B-spline	3	10	10^{-5}	10^{-1}	B-spline	4	15	10^{-1}	$5.1657 \cdot 10^5$
B-spline	3	10	10^{-3}	10^{-1}	B-spline	4	15	10^{-1}	$5.1658 \cdot 10^5$
B-spline	3	15	10^{-3}	10^{-1}	B-spline	4	10	10^{-3}	$5.1659 \cdot 10^5$
B-spline	3	5	10^{-3}	10^{-1}	B-spline	4	10	10^{-1}	$5.1659 \cdot 10^5$
B-spline	3	15	10^{-5}	10^{-1}	B-spline	4	10	10^{-3}	$5.1659 \cdot 10^5$
B-spline	3	15	10^{-7}	10^{-1}	B-spline	4	10	10^{-3}	$5.1659 \cdot 10^5$
B-spline	3	5	10^{-5}	10^{-1}	B-spline	4	10	10^{-1}	$5.1659 \cdot 10^5$
B-spline	3	5	10^{-7}	10^{-1}	B-spline	4	10	10^{-1}	$5.1659 \cdot 10^5$
B-spline	3	10	10^{-1}	10^{-1}	B-spline	4	15	10^{-3}	$5.1669 \cdot 10^5$

great variation of the residual sum of squares. Table 4 displays 10 combinations of parameters which had the smallest RSA (last column in table). All ten parameter combinations have B-spline basis functions of order 3 for the data and B-spline basis functions of order 4 for the coefficient curves. The best interior knot count is around 10 and the smoothing parameter for the functional response λ_y is between 10^{-7} and 10^{-3}. All ten parameter combinations have a smoothing parameter of the functional covariates λ_x which equal 10^{-1}. Since this is at the edge of the grid, further improvement could be achieved if some higher values of λ_x are used. Taking this fact into account a second simulation is started with slightly changed simulation settings. Table 5 shows further combinations of parameters that were used to obtain more improved models. As mentioned above the different parameter combinations of the coefficient functions did not take a lot of computational time. That is why almost all combinations were computed again. In the second simulation run a total of 2080 functional models were adapted. The computational time of 17 hours was much shorter than the first simulation run due to a reduced number of models computed.

Table 6 shows the 10 parameter combinations with the smallest RSA after combining both simulation runs. Rows originating from the second simulation run are marked with a green background. Three of the best ten parameter combinations originate from the second simulation run. The combination of parameters with the lowest RSA has B-spline basis functions for the data of

Table 5: Simulation design phonetic dataset: a finer selection grid based on the first simulation run.

Basis functions data			λ_y	λ_x	Basis functions coefficients			λ_{coef}
Type	Order	Knots			Type	Order	Knots	
B-spline	3	10	10^{-3}	10^{2}	B-spline	2	10	10^{-1}
B-spline	3	15	10^{-5}	10^{1}	B-spline	2	15	10^{-3}
B-spline	4	10		10^{0}	B-spline	3	10	10^{-5}
B-spline	4	15		10^{-2}	B-spline	3	15	10^{-7}
Constant	-	-			B-spline	4	10	
					B-spline	4	15	
					Polynomial	3	-	
					Polynomial	5	-	
					Polynomial	7	-	
					Fourier	3	-	
					Fourier	5	-	
					Fourier	7	-	
					Constant	-	-	
Total number of variations:								
5			2	4		13		4

order 4, 15 interior knots, $\lambda_y = 10^{-5}$ and $\lambda_x = 10^{-2}$. The best combination for the coefficient curves are B-spline basis functions for the data of order 4, 10 interior knots and $\lambda_{coef} = 10^{-3}$. Please note that computational time saving reasons the models were fitted only by using 11 functional covariates and a scalar intercept. Further improvements can be achieved by using the rest of the available non-functional covariates.

Those parameters lead to the smallest tested RSA. Nevertheless it would be rather unlikely that this would be the best combination of hyperparameters for the phonetic dataset. The obtained combination can be considered as sufficiently accurate given the computational time needed.

Explorative graphics are presented in the appendix. Figures 13 to 24 display the adapted curves alongside with mean and standard deviations. Figures 25 to 35 display cross correlation plots between the response variable and single functional covariates. Functional principal component analysis is not described for the given data. The obtained hyperparameters led to curves with low variation. Therefore the first principal component explained more than 95% of the variation regardless the functional variable considered. The first principal component usually equals the mean function in cases with high percentage of explained variance.

Table 6: Simulation 1+2: Ten parameter combinations with lowest residual sum of absolute values.

Basis functions data			λ_y	λ_x	Basis functions coefficients			λ_{coef}	RSA
Type	Order	Knots			Type	Order	Knots		
B-spline	4	15	10^{-5}	10^{-2}	B-spline	4	10	10^{-3}	$5.1656 \cdot 10^5$
B-spline	3	10	10^{-7}	10^{-1}	B-spline	4	15	10^{-1}	$5.1657 \cdot 10^5$
B-spline	3	10	10^{-5}	10^{-1}	B-spline	4	15	10^{-1}	$5.1657 \cdot 10^5$
B-spline	3	10	10^{-3}	10^{-1}	B-spline	4	15	10^{-1}	$5.1658 \cdot 10^5$
B-spline	4	15	10^{-3}	10^{-2}	B-spline	2	10	10^{-7}	$5.1658 \cdot 10^5$
B-spline	4	10	10^{-3}	10^{-2}	B-spline	4	10	10^{-1}	$5.1658 \cdot 10^5$
B-spline	3	15	10^{-3}	10^{-1}	B-spline	4	10	10^{-3}	$5.1659 \cdot 10^5$
B-spline	3	5	10^{-3}	10^{-1}	B-spline	4	10	10^{-1}	$5.1659 \cdot 10^5$
B-spline	3	15	10^{-5}	10^{-1}	B-spline	4	10	10^{-3}	$5.1659 \cdot 10^5$
B-spline	3	15	10^{-7}	10^{-1}	B-spline	4	10	10^{-3}	$5.1659 \cdot 10^5$

The following models in this chapter are based on the simulated parameters which are also set as default parameters in the R-functions (refer to chapter 5).

4.1.2. Model computation and interpretation

Using the obtained hyperparameters which minimize the residual sum of squares and residual sum of absolute values one can now compute the functional linear model. For the first model, only functional covariates will be applied. This procedure is helpful regarding the discovery of tendencies and behaviors of coefficient curves throughout different parts of the time scale. Figure 8 shows the 11 coefficient curves and the value of the scalar intercept. Pointwise 95% confidence intervals were computed for every covariate. They are based on 2000 bootstrap samples using the percentile method. The functional variables `tbackx`, `tbacky`, `ttipy`, `jawx`, `llipy`, `tdorsx` and `tmidx` have a negative effect on the second formant through the whole time scale. This means e. g. for `ttipy` that if the tongue tip moves up in vertical direction than the voice frequency tends to get lower. Contrariwise the covariates `ttipx`, `llipx` and `tmidy` have a positive effect on the second formant. The covariate `tdorsy` has a minor effect on the response, due to the fact that the zero line is completely between the confidence limits. An interesting structure of a coefficient curve is presented by the covariate `jawx`. During the first 60% of the time it has almost a constant value of approximately -17. After that the curve is decreasing fast, reaching

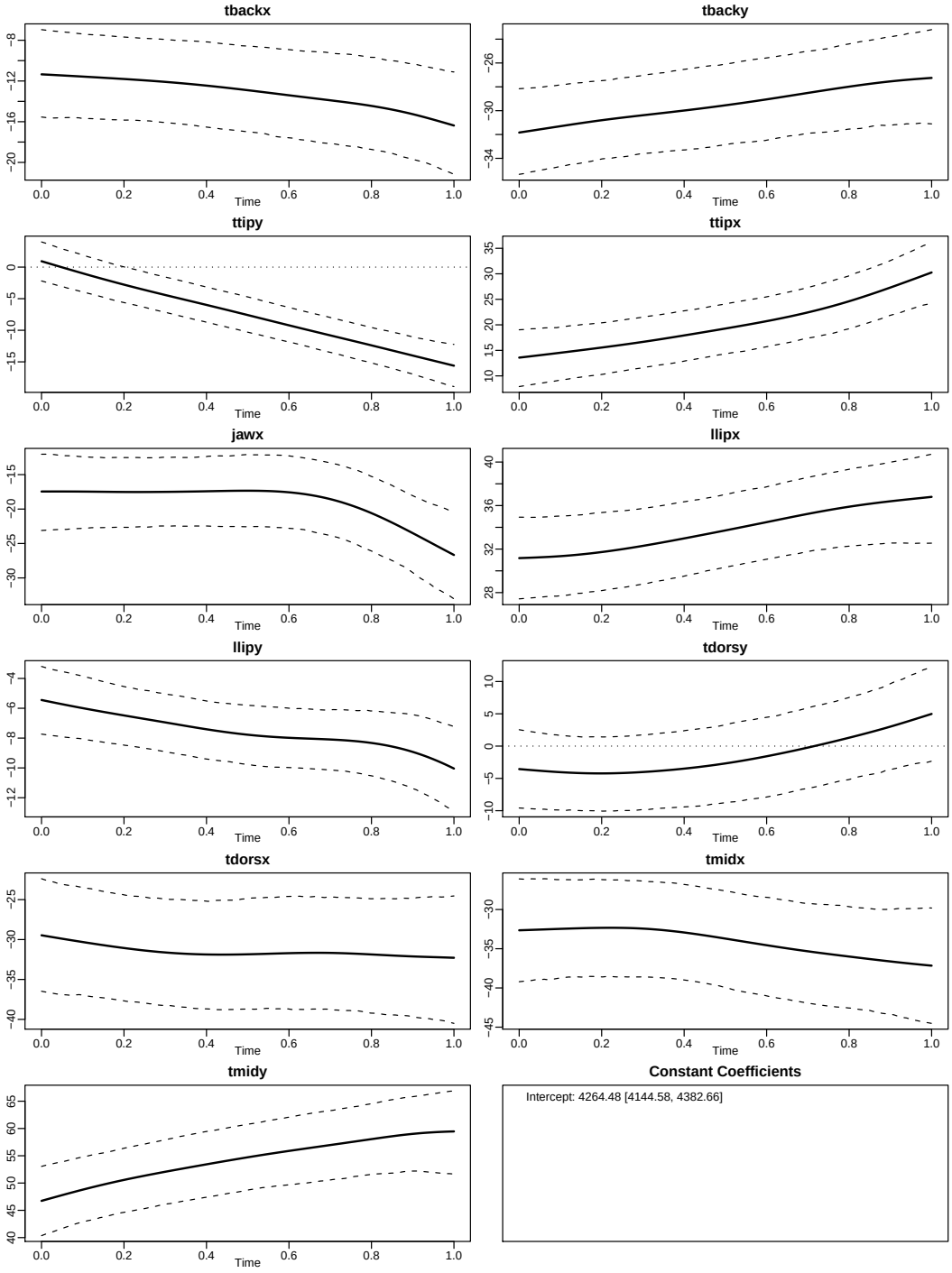


Figure 8: Model 1: Coefficient curves and intercept. Pointwise confidence intervals (95%) based on 2000 bootstrap samples

a minimum at the end of the time scale at approximately -27. This means that the negative effect which `jawx` is having on the second formant is greater at the end of the time scale. If the speaker is moving the jaw sensor forth in horizontal direction than the voice frequency is expected to decrease mildly in the first 60% of the spoken vowel. After that an even greater drop is expected to occur at the end of the spoken vowel.
A further way of looking at the model coefficients as well as other model information is presented in table 7. The table is split into three parts. In the first part non-functional coefficients are presented. Since the intercept is the only non-functional variable which is used in this model, it is displayed in this section of the table. The second part of the table shows a summary of the functional coefficients. Mean, median, minimum, maximum and standard deviation of the coefficient curves is presented. The computation of these values is done by evaluating the curves at 100 equally placed spots between the two ends of the time scale. After that, it is a trivial task to compute several statistics. The last part of the table shows further numbers concerning the functional model. In this data 2543 observations are used for the computation. The mean R squared is approximately 0.75. This is achieved by evaluating the R squared curve at 18 equally placed points and computing the mean of those points after that. The residual sum of squares produced by the model is $1.5 \cdot 10^8$. The last value in the table shows that the mean residual is at approximately 203. The last three values are different ways in assessing goodness of fit. A variation of some hyperparameters would lead to different numbers which then can be helpful in comparing the two different models. Generally this kind of summary provides a good way in comparing the intensity of the effects. While the interpretation of the non-functional covariates does not differ from classical statistics, one should be careful in interpreting the values for the functional covariates. If one considers only the mean and the standard deviation e. g., it is a common approach to favor covariates with a large absolute value of the mean and a rather small standard deviation. The problem of this approach is that a functional variable could have e. g. strong positive effects at one end of the time scale and strong negative effects at the other. This could theoretically lead to a zero mean although throughout most of the time scale the curve would not be near the zero line. If the minimum and maximum of the curve are considered too, then this problems should not occur. In the given case only the covariates

Table 7: Model 1: Summarizing the coefficients and further statistics.

Estimated numeric coefficients:

	Class	Coefficient
Intercept	numeric	4264.476

Estimated functional coefficients (summarized):

	Class	Mean	Median	Minimum	Maximum	Std. Dev.
tbackx	functional	-13.167	-12.906	-16.375	-11.345	1.399
tbacky	functional	-29.465	-29.549	-31.836	-27.252	1.370
ttipy	functional	-7.553	-7.592	-15.609	0.931	4.764
ttipx	functional	20.098	19.277	13.576	30.262	4.706
jawx	functional	-18.955	-17.513	-26.668	-17.335	2.584
llipx	functional	33.806	33.717	31.171	36.798	1.881
llipy	functional	-7.563	-7.775	-10.043	-5.445	1.099
tdorsy	functional	-1.522	-2.682	-4.230	4.982	2.808
tdorsx	functional	-31.494	-31.738	-32.284	-29.465	0.690
tmidx	functional	-34.131	-33.710	-37.160	-32.329	1.669
tmidy	functional	54.263	54.720	46.734	59.462	3.734

Number of observations: 2543
Mean R squared: 0.752839
Residual sum of squares: 150942505
Mean residual: 203.1990

`ttipy` and `tdorsy` are associated with curves which cross the zero line. The strongest negative effect of the covariate `ttipy` has a value of -15.6, which indicates that this covariate explains much of the response at least at one part of the time scale. The strongest effects are found for the covariates `tmidx` and `tmidy` with -34.1 and 54.3 respectively. Both covariates are associated with a sensor placed in the middle part of the tongue.
The mean R squared value of 0.75 suggests a good quality of the fit. In order to extend the idea of the R squared statistics to functional data, it is also possible to examine a R squared curve. It provides information on fit behavior during the different parts of the time scale. The curve in figure 9 shows a very good fit with a R squared value of around 0.8 during the middle part of the time scale. At both ends it drops to approximately 0.75 (start of time scale) and 0.65 (end of time scale). This drop at the end of the time scale can be explained by the adapted curves being more variable at the end of the time scale.

An even better fit can be expected using more non-functional covariates than

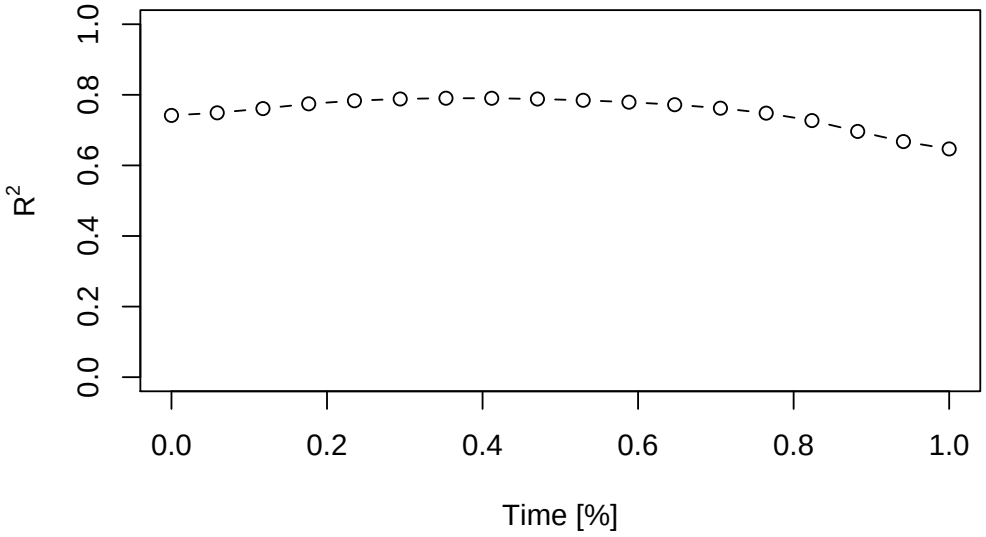


Figure 9: Model 1: R squared curve. The evaluation of the curve at 18 equally placed spots lead to a mean R squared value of 0.75.

an intercept. Table 2 suggests that further effects such as the speaker effect, consonant effect and tensity effect could be applied. For simplification reasons the same hyperparameters as in the previous model will be used. This approach could be justified if taken into account that those hyperparameters concern only the functional covariates. At the same time, the new model is extended by only non-functional covariates. On the other hand the statistic that is being minimized by the models is the residual sum of squares. Since there are interactions between the (new) non-functional covariates and the functional ones, it would be highly unusual that the same hyperparameters would minimize the RSS in both models. However the simulation for model 1 showed that the differences in the RSS between the best combinations of hyperparameters are very small. Considering all these facts, the simulated parameters are absolutely eligible to used once again for the new model.

Table 8 summarizes the fitted coefficients. The non-functional covariates which are included in this model are displayed in the first part of the table. The intercept effect is slightly higher than in model 1 without making a big difference in the interpretation. The next six rows denote the speaker effects where `s.l`

Table 8: Model 2: Summarizing the coefficients and further statistics.

Estimated numeric coefficients:

	Class	Coefficient
Intercept	numeric	4360.519
s.lck	numeric	174.438
s.lfs	numeric	-54.741
s.lhp	numeric	-15.126
s.lht	numeric	-339.377
s.lmh	numeric	-398.554
s.lta	numeric	-198.163
k.lP	numeric	-13.979
k.lT	numeric	-52.840
t.l+	numeric	13.185

Estimated functional coefficients (summarized):

	Class	Mean	Median	Minimum	Maximum	Std. Dev.
tbackx	functional	-26.642	-26.464	-29.611	-24.821	1.321
tbacky	functional	-20.576	-20.367	-22.748	-19.526	0.906
ttipy	functional	1.466	1.286	-4.992	8.538	3.690
ttipx	functional	-0.455	-0.603	-6.184	6.493	3.568
jawx	functional	-4.470	-3.512	-9.911	-3.277	1.839
llipx	functional	40.179	40.308	38.234	41.862	1.306
llipy	functional	-15.124	-15.125	-18.727	-12.240	1.714
tdorsy	functional	19.336	18.195	16.872	27.519	2.868
tdorsx	functional	-37.851	-37.947	-38.768	-35.765	0.747
tmidx	functional	4.818	5.754	0.369	6.941	2.051
tmidy	functional	32.389	33.682	24.816	35.386	3.112

Number of observations: 2543
Mean R squared: 0.8122898
Residual sum of squares: 114293396
Mean residual: 178.0303

stands for the speaker covariate and the two letters after that are representing the speaker abbreviation. The speaker with the abbreviation `bk` is alphabetically first and for that reason in the reference category with an effect equaling zero. The covariates `k.lP` and `k.lT` stand for the consonants “P” and “T” which enclose the observed vowel. The reference category here is the consonant “K”. The last numeric coefficient fitted is the tensity effect. `t.l+` denotes present tensity and `t.l-` is the reference category.

For some functional covariates there has been a noticeable change after including all covariates. There was no complete alternation of the effect in terms of a negative covariate effect in model 1 becoming positive in model 2 or vice versa. However the effect intensity has changed for almost all functional covariates.

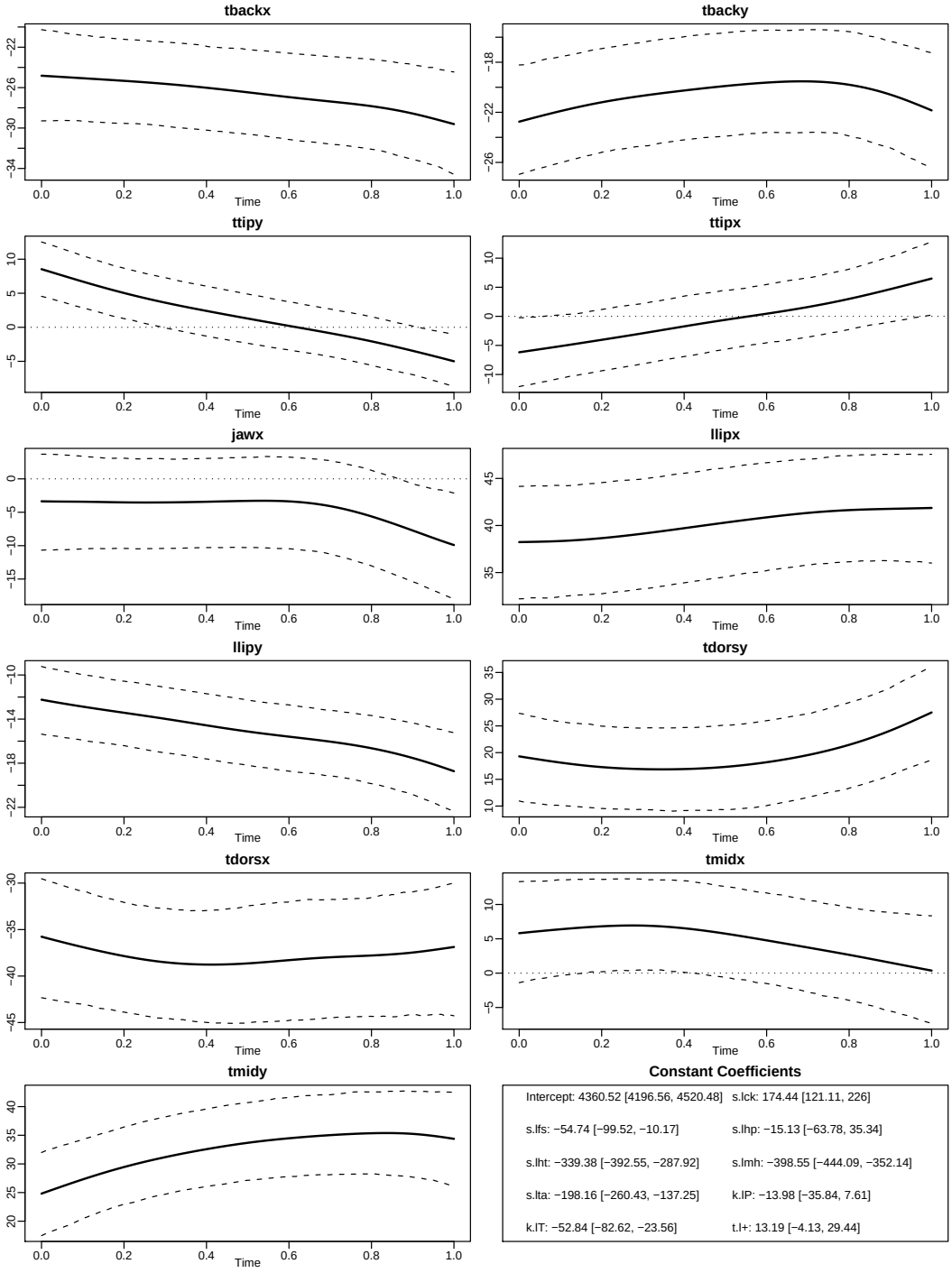


Figure 10: Model 2: Coefficient curves and scalar coefficients. Pointwise confidence intervals (95%) based on 2000 bootstrap samples

`tmidy` e. g. still has a positive effect but also a greatly decreased mean value of 32.4 (mean in model 1 was 54.3). The covariates `ttipy`, `ttipx`, `jawx` and `tmidx` have all mean values near zero. They do not give much information on the response.
The number of observations in this model equals the number of observations in model 1 (2543). The mean R squared value is higher and has a value of 0.81. The RSS was further lowered by the non-functional covariates and is now approximately 76% of the RSS from model 1. The mean residual value has also dropped to 178 which is a decrease of about 12%.
The R squared curve (figure 11) for this model is slightly higher than the corresponding curve for model 1. The overall R squared mean has a value of 0.81 which is 0.06 higher. The biggest difference is observed at the very end of the time scale where the R squared value for model 2 is approximately 0.08 higher.
Figure 10 shows the coefficients of model 2 alongside with confidence bands

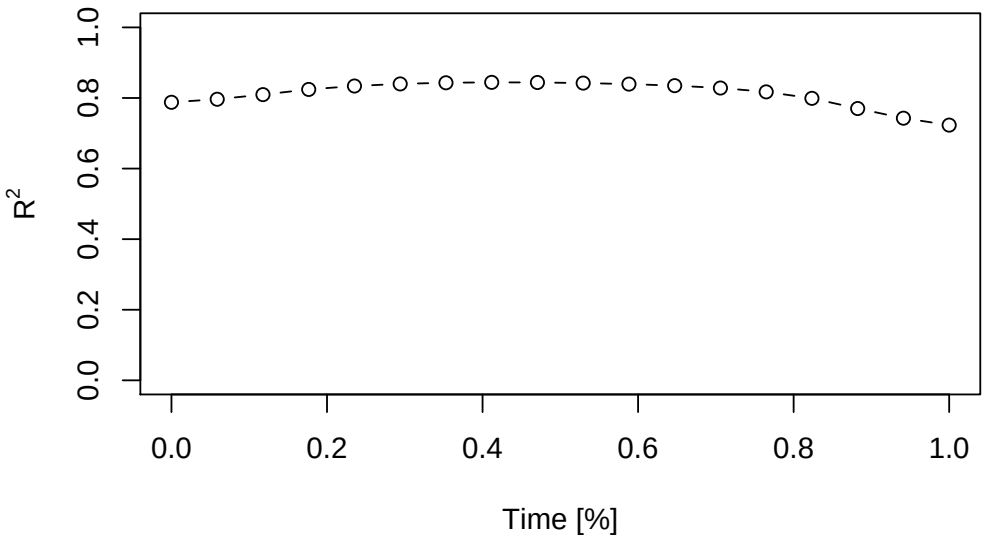


Figure 11: Model 2: R squared curve. The evaluation of the curve at 18 equally placed spots lead to a mean R squared value of 0.81.

which once again are based on 2000 bootstrap samples. Looking at the coefficient curves and scalar coefficients reveals that some covariate effects are weaker than others. The 95%-confidence bands corresponding to `ttipx`, `jawx`

and `tmidx` all embrace the zero line. This means that this covariates have a minor contribution for the explanation of the response variable. `ttipy` has a greater effect on the response only during the first 20% of the time scale. The non-functional covariate t.l representing the tensity has the zero included in the confidence interval produced with the bootstrap technique. Therefore this covariate has also a minor effect. The rest of the covariates all have strong effects and a greater contribution in explaining the functional response.

4.2. Predicting the voice frequency in the aspiration phase prior to the voiced vowel

In the last subsection the dependency between voice frequency and physiological movements of tongue, lips and jaw were described via functional models. The fit was improved by adding additional non-functional covariates. In the present part of the thesis the gained information will be used to predict the voice frequency during the aspiration phase. As mentioned in chapter 3, it is possible to distinguish very accurately between the aspiration phase and the voiced vowel pronunciation. The voice frequency can not be recorded properly in the aspiration phase. In the meantime the covariates are recorded accurately during both phases. The assumption that the voice frequency has a similar structure during both periods is an important requirement for applying the functional models from chapter 4.1.2. If this assumption holds true then a prediction of the voice frequency with the coefficients of the functional models is possible.

Figure 12 considers the voice frequency during the aspiration phase of 9 randomly chosen replications. The discrete data points are marked with black points. Red curves are adapted directly to the points. The green curves indicate the prediction for each replication. For this particular figure the coefficients and coefficient curves of model 2 are used. Obviously the red curves are highly variable due to the recording difficulties mentioned in chapter 3. The green curves on the other hand have little variation and seem to display a smooth version of the data. The region in which the data points are expected is well predicted. For replication 832 the predicted curve lies higher than the recorded discrete data points. This may be an outlier for which the model is not well suited. A further explanation for the gap between the green curve and

the data points lies in the possibly false recording. In this case the green curve would indicate the right position of the data points. Since there is no method to test the suitability of the curves this remains a task for future analyses.

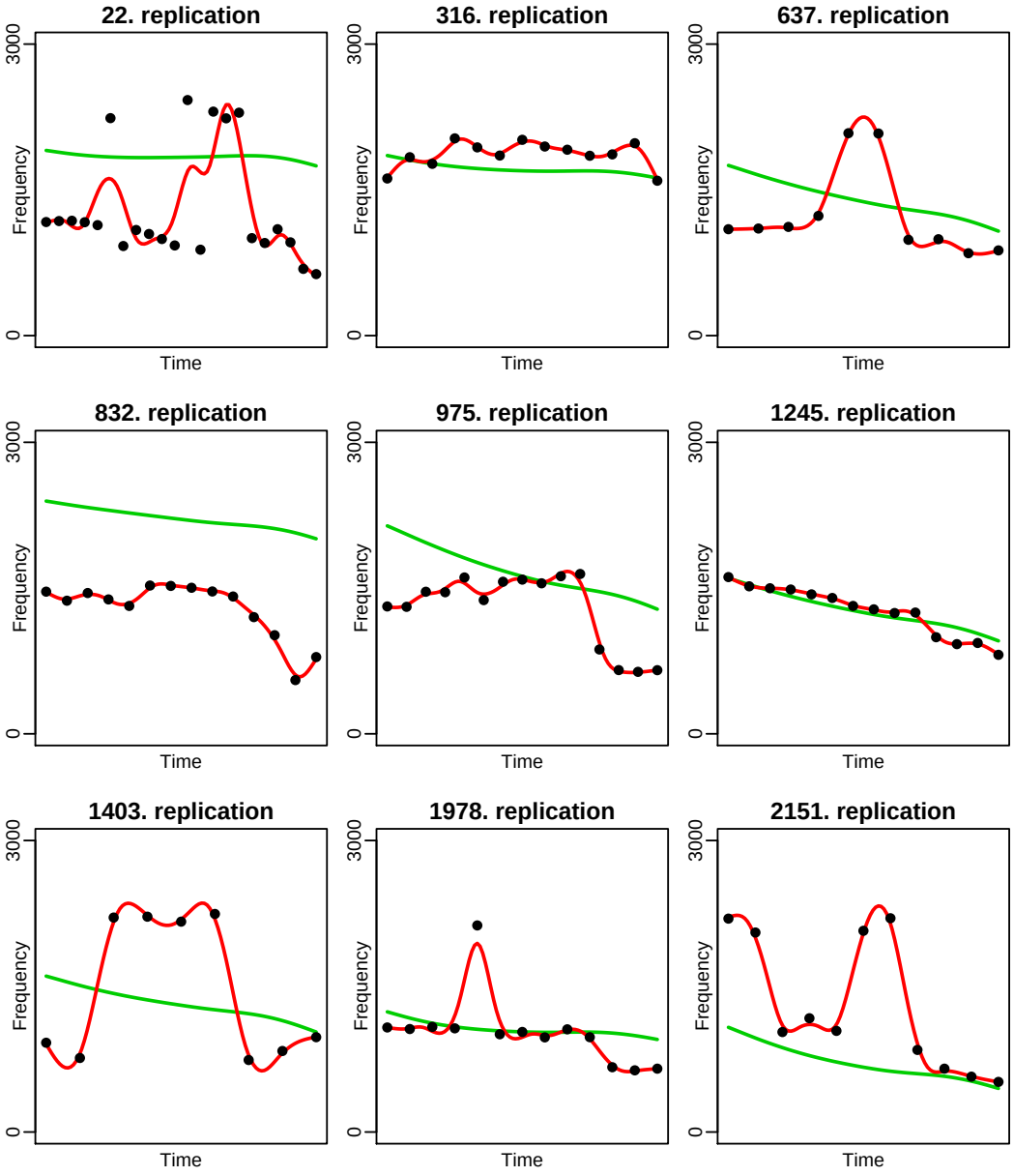


Figure 12: Voice frequency during the aspiration phase of 9 randomly chosen replications. Black points indicate recorded data points. Smoothing the data directly produces highly variable curves (red). The predicted curves using the functional and non-functional covariates of the aspiration phase and coefficients from model 2 are marked in green.

5. R functions and code

The analysis of the phonetic dataset as well as further simulations in this thesis were all computed with the software `R` ([12]). One of the features this thesis offers are all functions and codes written in a general way, so they can be easily applied to further phonetic data. In this section the key R functions are presented.
In order to properly use most of them one should install the packages `emu` ([13], [14]), `fda` ([15]) and `dummies` ([16]). R ([12]) is a free software environment for statistical computing and graphics. Additional packages can be easily installed with the help of functions like `install.packages('`*PACKAGENAME*`')`.
The most important functions are presented in this section in detail. Further functions are presented in the appendix. The functions in the appendix are mostly of generic type and can be used for further analysis using an object returned by the function `flm` (refer to section 2.4).

5.1. Function: fda_prep

The function `fda_prep` is used to convert raw discrete data into smooth curves. The input parameter `vars` expects a variable name or a character vector, containing the variable names. The parameter `trackdata` expects a list with the functional variables named like in `vars`. `lambdavec` is a vector containing the smoothing parameters – possibly different for every functional variable. `bas` expects an object of class `basisfd` which constructs the system of basis functions used. The parameter `min.len` expects an integer and denotes the minimum amount of discrete points that are used for the preparation of the data. If one replication has less discrete points than `min.len` then this replication is omitted and a warning message is displayed. The parameter `trace` expects a logical input whether to display information during the computation time. `ind` expects an index vector with logical entries and a length which equals the observation size. The observations which should be transformed into smooth curves are marked with `TRUE` and those who are not wanted as smooth curves and should be omitted are marked with `FALSE`. The `...` argument is not used at this stage of the programming.

```
fda_prep <- function(vars, trackdata, lambdavec = NULL, bas,
                     min.len = 4, trace = FALSE, ind = NULL, ...){
  if(missing(bas))
    bas <- create.bspline.basis(c(0, 1), norder = 4, breaks = seq
        (0, 1, 1/10))
  if(is.null(lambdavec)) lambdavec <- rep(1e-2, length(vars))
  L <- list()
  j <- 1
  if(trace) cat("Converting data points to curves:\n")
  for(V in vars){

    if(trace)  cat("Variable ", j, "/" , length(vars), "\n")

    # fdPar object containing basis phi_k and smoothing parameter
        lambda
    temp_basis <- fdPar(bas, lambda = lambdavec[match(V, vars)])

    # load corresponding trackdata object
    x <- trackdata[[V]]
    n <- nrow(x)

    # keep obervations with a minimum number of discrete points
    if(is.null(ind)){
      l <- rep(0, n)
      for(i in 1:n) l[i] <- length(tracktimes(x[i]))
      ind <- l >= min.len
      if(sum(!ind) > 0 && sum(!ind) <= 25)
      warning("Observations ", toString(which(!ind)), "\n",
              "  contain less than ", min.len,
              " discrete data points and will be omitted!")
      else if(sum(!ind) > 25)
      warning(sum(!ind), " observations contain less than ", min.
          len,
              " discrete data points and will be omitted!")
    }
    x <- x[ind]
    n <- nrow(x)

    # First iteration (initializing 'ret')
    x1 <- seq(0, 1, length.out = length(frames(x[1])))
    x2 <- frames(x[1])
```

```
39      smx <- smooth.basis(x1, x2, temp_basis)

41      ret <- list(coefs = matrix(NA, ncol = n, nrow = bas$nbasis),
                    basis = smx$fd$basis, fdnames = paste("rep", 1:n))
43      ret$coefs[, 1] <- smx$fd$coefs
        ret$fdnames <- NULL
45      ret$fdnames$reps <- paste("rep", 1:n, sep = "")
        ret$fdnames$value <- "value"
47      ret$fdnames$time <- seq(0, 1, length.out = nrow(smx$fd$coefs))

49      # 2:n iterations
        for(i in 2:n){
51        x1 <- seq(0, 1, length.out = length(tracktimes(x[i])))
          x2 <- frames(x[i])
53        smx <- smooth.basis(x1, x2, temp_basis)
          ret$coefs[, i] <- smx$fd$coefs
55      }

57      # class attribute and saving ret in L
        class(ret) <- "fd"
59      L[[j]] <- ret
        j <- j + 1
61    }

63    # naming L and returning L and ind
      names(L) <- vars
65    return(list(L = L, ind = ind))
  }
```

In the first five lines of the function code the input parameters are declared and some default values for basis functions and smoothing parameters are set. In lines 6 – 8 further variables like the output list `L` and a counting variable `j` is set to the value 1. If the user requests trace information during the computation, `j` is used to display which functional variable is being converted at the moment. Lines 9 – 61 contain the most important commands, wrapped into a for-loop. During this loop every variable is taken separately and the discrete data is converted to curves. In lines 21 – 32 it is taken into account that only observations with a minimum number of discrete data points should be converted to curves. In order to save some computation time these lines are being computed only one time while converting the first functional variable of the

data. The vector `ind` is created and then reused for the following variables. If `ind` is supplied as an input parameter then these lines are skipped during the first iteration too. In the lines 36 – 47 the return argument `ret` is prepared. `ret` is a list with the slots `coefs`, `basis` and `fdnames`. The most important slot is `coefs`, a matrix with a column size equaling the replication size and line numbers equaling the number of basis functions. In the very first iteration only the first column is filled with the weigths "c_k" (cf. (2.7)) from the first replication. The lines 49 – 55 fill the rest of the coefficient matrix. This part is the most time consuming part of the function. This can be explained by the function `smooth.basis`. Since this function has to be called as often as the observation size it has a considerable high contribution to the overall computation time. In the lines 57 – 60 the list object `ret` is classified with the class "`fd`" and saved in a slot of `L`. The counting parameter `j` is increased by 1. The last part of the function (lines 65 – 67) consists of naming the list object `L` with the associated functional variable names and returning `L` alongside the constructed logical vector `ind`.

5.2. Function: fdareg

The function `fdareg` is used to compute a regression model. The input parameter `resp` expects a functional variable with associated class "`fd`" denoting the response variable. The parameter `x` expects a list with the functional and non-functional covariates. Each slot of the list should contain a single variable of class "`fd`", "`numeric`" or "`integer`". `bas` expects an object of class `basisfd` which constructs the system of basis functions used. `l` stands for the smoothing parameter of the coefficient curves λ_{coef}. `ind` and `emuobj` are used to construct a plot function which plots single fitted curves and the discrete data of the same replication.

```
fdareg <- function(resp, x, bas, l, ind = NULL, emuobj = NULL){
2
  # fdPar construction for coefficient curves
4 betafdPar <- fdPar(bas, lambda = l)
  betalist <- list()
6
```

```
  # for functional variables, coefficient curves with betafdpar
      are fitted
  # for constant variables, one scalar coefficient is fitted
  # other types are not allowed
  for(i in 1:length(x)){
    if(class(x[[i]]) == "fd") betalist[[i]] <- betafdPar
    else if(class(x[[i]]) %in% c("integer", "numeric"))
      betalist[[i]] <- create.constant.basis(c(0, 1))
    else stop(paste("Variable ", i, " has wrong class!", sep = "")
        )
  }

  # main function for regression
  fRegressList <- fRegress(resp, x, betalist)

  # fitted curves
  predhatfd <- fRegressList$yhatfd$fd
  # estimated coefficients
  betaestlist <- fRegressList$betaestlist

  # matrix of fitted points (dimension nbasis x nobs)
  predhatmat <- eval.fd(seq(0, 1, length.out = nrow(resp$coefs)),
      predhatfd)

  # residual matrix
  resmat <- predhatmat - resp$coefs

  # R2 computation (Ramsay Silverman: p. 285)
  # R2 = 1 - (hat(y) - y)^2 / (y - bar(y))^2
  predmeanvec <- as.matrix(rowMeans(resp$coefs))
  resmat0 <- resp$coef - predmeanvec %*% matrix(1, 1, ncol(resp$
      coefs))
  SSE0 <- apply(resmat0 ^ 2, 1, sum)
  SSE1 <- apply(resmat ^ 2, 1, sum)
  Rsqr <- (SSE0 - SSE1) / SSE0

  # plot function for plotting the fitted curves
  fdaplot <- function(n, emuobj = emuobj, ind = ind, ...){
    n2 <- which.max(cumsum(ind) == n)
    xaxis <- seq(0, 1, length.out = length(tracktimes(emuobj[n2]))
        )
    plot(xaxis, frames(emuobj[n2]),
```

```
44          ylab = "Frequenz", xlab = "Zeit",
            ylim = c(min(min(predhatmat), min(frames(emuobj))),
46                   max(max(predhatmat), max(frames(emuobj)))),
            main = paste(n, ". Beobachtung", sep = ""), pch = 19,
                ...)
48      lines(xaxis, eval.fd(resp, xaxis)[,n], col = 3)
        times <- seq(0, 1, length.out = 100)
50      lines(times, eval.fd(predhatfd[n], times), col = 2, lty = 2)
    }
52
    # returning a list and giving an class attribute
54  ret <- list(reg = fRegressList, Rsqr = Rsqr, fdaplot = fdaplot,
        resp = resp)
    class(ret) <- 'emu_fd'
56  return(ret)
}
```

In lines 4 – 15 `betalist` and `betafdPar` are constructed using the given input information. They are used in the regression step and denote what kind of coefficient curves or scalar coefficients should be fitted. In line 18 the main regression functions is executed using the response `resp`, the covariates `x` and the properties of the coefficients stored in `betalist`. This is the computationally most intensive part of the given function. In lines 21 – 37 the computation of the functional R squared is executed. Lines 40 – 51 contain the construction of the internal plot function which can be used to plot fitted curves of single observations alongside with the discrete data points. This could be used to visually qualify the model fit, e. g. when plotting several random observations. In lines 54 – 56 the return object is constructed and returned. It is a list with four slots containing the regression object, the functional R squared values, the plot function and the response argument.

5.3. Function: flm

The function `flm` is the main function which is presented to the user. The usage is similar to functions like `lm` or `glm`. The first input parameter `formula` which expects a formula argument of the type `y ~ 1 + x1 + x2`. On the left side of the tilde the response argument should be inserted. On the right side of the formula covariates are separated with "+" signs. An intercept usage is being denoted by `1`. The input parameter `data` is expected as a list with containing

discrete data points, adapted functional curves and a logical vector denoting which observations to use for the functional regression model. Normally this argument will be set to `NULL` and the data preparation will be done within the function. If the data argument is presented the computational time is reduced severely. This is especially useful when some of the control arguments are replaced and at the same time no data transformation needs to be done. The input parameter `return.prepared.data` is logical and it indicates whether the prepared data should be returned or not. The last input parameter is `control` which expects a list as returned by the function `flm.control` (refer to section 5.4).

```
flm <- function(formula,
                data = NULL,
                return.prepared.data = TRUE,
                control = flm.control()
                ){


  y <- strsplit(deparse(formula, width = 500), " \\~ ")[[1]][1]
  x <- strsplit(strsplit(deparse(formula, width = 500), " \\~ ")
      [[1]][2], " \\+ ")[[1]]
  if("1" %in% x){
    x <- x[x != "1"]
    intercept <- TRUE
  }
  vars <- c(y, x)

  if(is.null(data)){
    regdata <- list()
    data[[1]] <- eval(as.name(y))
    types <- "trackdata"
    for(i in 1:length(x)){
      dat <- eval(as.name(x[i]))
      if(class(dat) %in% c("factor", "character")){
        dat <- as.data.frame(as.matrix(dat))
        colnames(dat) <- vars[i+1]
      }
      data[[i+1]] <- dat
      types[[i+1]] <- class(data[[i+1]])
```

```
    }
29  names(data) <- vars

31  num_data <- NULL
    for(i in 1:length(vars)){
33    if(types[i] == "numeric") num_data <- cbind(num_data, data[[
          i]])
      else if(types[i] == "data.frame")
35      num_data <- cbind(num_data, dummy(vars[i], data[[i]])[,
            -1, drop = FALSE])
    }
37
    fun_vars <- vars[types == "trackdata"]
39  fun_vars_loc <- match(fun_vars, vars)
    if(length(fun_vars) > 0){
41    temp <- fda_prep(vars = fun_vars, trackdata = data,
                       lambadavec = control$fd_smooth, bas =
                           control$fd_basis,
43                     min.len = control$min.len, trace = control$
                           trace)
      func <- temp$L
45    ind <- temp$ind
      for(i in 1:length(fun_vars)){
47      regdata[[i]] <- func[[i]]
        names(regdata)[[i]] <- fun_vars[i]
49    }
    }
51  else ind <- rep(TRUE, length(data[[1]]))

53  if(!is.null(num_data)){
      for(i in 1:ncol(num_data)){
55      regdata[[length(fun_vars) + i]] <- (num_data[, i])[ind]
        names(regdata)[[length(fun_vars) + i]] <- colnames(num_
            data)[i]
57    }
    }
59
    if(intercept){
61    inter <- rep(1, sum(ind))
      for(i in length(regdata):2){
63      regdata[[i+1]] <- regdata[[i]]
        names(regdata)[[i+1]] <- names(regdata)[[i]]
```

```
65        }
          regdata[[2]] <- inter
67        names(regdata)[[2]] <- "Intercept"
        }
69
        if(return.prepared.data)
71        data <- list(regdata = regdata, ind = ind, emudata = data)
        else
73        data <- list(ind = ind, emudata = data)

75    }
      else{
77      if(!all(c("regdata", "ind", "emudata") %in% names(data)))
          stop("Argument 'data' can't be used! Please provide slots\n"
              ,
79              "  named c('regdata', 'ind', 'intercept', 'emudata')")
        regdata <- data$regdata
81      ind <- data$ind
        if(!return.prepared.data)
83        data <- list(ind = ind, emudata = data$emudata)
      }
85
      response <- regdata[[match(y, names(regdata))]]
87    pred <- regdata[-match(y, names(regdata))]
      model <- fdareg(response, pred, control$coef_basis, control$coef
          _smooth,
89                    data$ind, data$emudata[[1]])

91    ret <- list(model = model, data = data, call = formula, control
          = control)
      class(ret) <- "flm"
93    return(ret)

95
}
```

Lines 8 – 14 the formula argument is being deparsed. Intercept usage and the separation of covariates and response are made in this first lines too. Lines 16 – 75 are skipped if a `data` argument is presented. In the more usual case of `data` argument being passed as `NULL` the discrete data points are being converted to smooth curves in this lines. The user can use functional, numer-

ical and categorical predictors. Only a functional response is allowed. In lines 17 – 37 the non-functional covariates are prepared. Since the functions from the `fda` package which are used for the regression in the following steps do not allow categorical covariates, this is being taking care of in this steps. The function `dummy` originating from the `dummies` package is very helpful for transforming a categorical variable into multiple dummy variables. In lines 38 – 51 the functional covariates and the functional response are turned into curves and the index vector `ind` denoting which replications to be used for the regression is being constructed. The numerical covariates and the intercept are attached to the functional variables in lines 53 – 68. If the input parameter `return.prepared.data` is `TRUE` then all prepared data is attached to the list `data` in a slot `regdata`. If `return.prepared.data` is set to `FALSE` this step is skipped (lines 70 – 74). If a data argument is presented it should contain the slots `regdata`, `ind` and `emudata`. If some of those slots are missing then the function is stopped and a custom error message is presented to the user asking to provide the missing slots. If the three slots are provided than in lines 76 – 84 this information is extracted from the `data` argument. In lines 86 – 89 the prepared data and the information about the type of coefficient curves are used to compute the model. A list containing model information, data, formula call and control arguments is returned in the last lines of the function.
Please refer to the appendix for generic functions which can be applied to an object returned by the function `flm`. `residuals`, `fitted`, `coef`, `summary`, `print` and `plot` all provide useful information for the user. The functions `bootstd` and `plot.bootstd` are designed to compute and plot bootstrap based confidence intervals for the coefficient curves.

5.4. Function: flm.control

The function `flm.control` is used to control hyperparameters while computing a functional model. `coef\_basis` expects an object of class `basisfd` which contains additional information about the type of the coefficient curves like knot placement, order or time scale of the basis functions. `coef\_smooth` controls the smoothing parameter for the coefficient curves. If the parameter `trace` is set to `TRUE` the user is presented with some details on the computation and is able to make a rough estimate on the computation time. `min.len` denoted

the minimum number of discrete data which have t be contained in each replication. `fd\_basis` is used to control the functional variables. Type of basis functions, knot placement, order and time scale are possible variations of the basis functions. The smoothing parameter of the functional data can be controlled with `fd\_smooth` which can be a vector or a scalar.

```
flm.control <- function(
  coef_basis <- create.bspline.basis(c(0, 1), norder = 4,
                                     breaks = seq(0, 1, 1/10)),
  coef_smooth <- 1e-3,
  trace <- TRUE,
  min.len <- 4,
  fd_basis <- create.bspline.basis(c(0, 1), norder = 4,
              breaks = seq(0, 1, 1/15)),
  fd_smooth <- NULL
){

  return(list(coef_basis = coef_basis, coef_smooth = coef_smooth,
              trace = trace, min.len = min.len, fd_basis = fd_
                  basis,
              fd_smooth = fd_smooth))
}
```

6. Concluding remarks

The main topic of this thesis concerned the provision of functional linear models used for the analysis of phonetic data. Since many measurable variables in the field of phonetics like voice frequency or tongue movements are indeed functional, this approach should prove useful for future analyses.
In section 2 of this thesis the theoretical background for analysis of functional data was presented and some additional equations were derived. Many explorative tools for the description of functional datasets were presented as well as more complicated methods like functional principal component analysis or functional linear models. The derivation of bootstrap based confidence bands for the coefficient curves played a main role in assessing goodness of fit. Further useful statistics such as (functional) residual sum of squares or different variations of the coefficient of determination were also discussed.
In section 4 the presented methods were used to analyze the phonetic dataset described in 3. Different functional models with voice frequency or more precise the second formant as a functional response and different functional and non-functional covariates were discussed. The final model adapted had an overall mean R squared value of 0.81 which indicated a good description of the complex response. In order to find the best hyperparameters for all functional variables, computationally intensive simulations were performed.
The information gained through the functional models was used for a prediction of voice frequency during pronunciation periods of not very accurate recording. The results seem highly rational and can be interpreted well. Nevertheless methods for discovering the correctness of this approach have to be researched in future.
Section 5 provides implemented functions which can be used to analyze a general phonetic dataset. The functions are user friendly and may be used in a similar way as in the cases of `lm()` and `glm()`. Generic functions can also be used for further exploration of the adapted models. This should guarantee a troublefree and unproblematic usage of the functions by a large number of R users.

Room for future research lies in the further development of functional models. The models used in this thesis can be qualified as concurrent. A functional

covariate only influences the functional response at a fixed time t_j. More complicated models with dependencies modeled throughout the whole time scale and coefficient curves replaced by coefficient surfaces are possible. At the same time this approach would increase the need of a larger number of discrete data points per functional observation.
The bootstrap based confidence bands derived in this thesis can also be extended in future research. For now only the percentile method can be used. Further methods like bootstrap-t-intervals, bootstrap-BC_α-intervals or even intervals based on parametric bootstrap seem to be possible alternatives.
Another futuristic idea concerns variable selection techniques for functional models (partly in [17]) as well as functional mixed models [18]. This would allow an even deeper exploration of data structures by possibly using less covariates and automatically saving computation time. Mixed model approaches could be used to differ e. g. between different speakers with random intercept or random slopes.
Future research could also be useful for the implementation of the R functions. The inclusion of those functions in the `emu` package could make them easily accessible for R users. A separate R package is also a further possibility to promote the functions. While implementing the R code sparse programming was of great interest. Even faster results could be achieved if some of the functions with (nested) `for` loops were implemented in C.

Many of the methods and techniques proved to be very useful for functional data analysis. Nevertheless a great number of hyperparameters have to be adjusted prior to the actual analysis. This requires either a priori knowledge on the data structure or some criterion that has to be minimized (like the RSS). Since in most cases this criterion is computed during more complicated analysis, exploration techniques normally applied during the first analysis stage are neglected.
The long duration of the computational time for finding the best set of coefficients and bootstrapping confidence bands is also problematic. The user is not able to compute a fast model if only some minor exploration of the data is aimed. By relying on the default set of parameters delicate dependencies may not be discovered. This difficulty may be partly remedied if C programming is

used.
The best functional model discussed in this thesis had a mean residual absolute value of approximately 178. The data could be analyzed with ordinary linear models. This would be possible if every functional replication is replaced by one single value, which typically would be the mean or the median of the discrete data points. This approach was tested and the resulting mean residual absolute values equaled 206. This corresponds to a reduction of about 14% by using the functional linear model approach. This reduction does not justify the greater effort put in the new approach. Nevertheless the computation of coefficient curves reveals tendencies that an ordinary least squares approach would never detect. The variation of the effect power of functional covariates throughout different parts of the time scale is a feature which is presented only by functional models. Furthermore the small decrease of the absolute residual values may be a feature which is hidden in the dataset itself. Further research with simulated data could reveal the real potential of the presented models.

The first steps into introducing functional models as a standard technique in analyzing phonetic data were made with this thesis. Many different possibilities of further research exist and are yet to be investigated.

References

[1] J. O. Ramsay and B. W. Silverman. *Functional data analysis*. Springer, New York, 2005.

[2] J. O. Ramsay, G. Hooker, and S. Graves. *Functional Data Analysis with R and MATLAB*. Springer Verlag, 2009.

[3] J.O. Ramsay and B.W. Silverman. *Applied functional data analysis: methods and case studies*. Springer Verlag, 2002.

[4] M. Gubian. Functional Data Analysis as a tool for Phonetic Detail analysis. 2010.

[5] M. Gubian, F. Torreira, H. Strik, and L. Boves. Functional data analysis as a tool for analyzing pronunciation variation-a case study on the french word c'était. In *Proceedings of INTERSPEECH*, pages 6–10, 2009.

[6] C. De Boor. On calculating with B-splines. *Journal of Approximation Theory*, 6(1):50–62, 1972.

[7] G.E. Farin, J. Hoschek, and M.S. Kim. *Handbook of computer aided geometric design*, chapter 6. North-Holland, 2002.

[8] T. Hastie and R. Tibshirani. Varying-coefficient models. *Journal of the Royal Statistical Society, Series B*, 55:757–796, 1990.

[9] Nagelkerke N. A Note on a General Definition of the Coefficient of Determination. *Biometrika*, 78(3):691–692, 1991.

[10] Ph. Hoole, A. Zierdt, and C. Geng. Beyond 2D in articulatory data acquisition and analysis. In *Proc. 15th ICPhS*, pages 265–268, Barcelona, Spain, 2003.

[11] Ph. Hoole and A. Zierdt. Five-dimensional articulography. *Stem-, Spraak- en Taalpathologie 14, Supplement June 2006*, page 57, 2006.

[12] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. ISBN 3-900051-07-0.

[13] Jonathan Harrington, others, and IPS LMU Muenchen & IPDS CAU Kiel. *emu: Interface to the Emu Speech Database System*, 2009. R package version 4.1.

[14] Institute of Phonetics and Speech Processing LMU Munich. http://emu.sourceforge.net/, 2010.

[15] J. O. Ramsay, Hadley Wickham, Spencer Graves, and Giles Hooker. *fda: Functional Data Analysis*, 2009. R package version 2.1.2.

[16] Christopher Brown. *dummies: Create dummy/indicator variables flexibly and efficiently*, 2009. R package version 1.05-1.

[17] J.H. Friedman and B.W. Silverman. Flexible parsimonious smoothing and additive modeling. *Technometrics*, 31(1):3–21, 1989.

[18] J.S. Morris and R.J. Carroll. Wavelet-based functional mixed models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(2):179–199, 2006.

A. R functions

A.1. Generic functions for class flm

Listing 1: fitted() function for class emu_fd

```
fitted.emu_fd <- function(object, type, emuobject = NULL, ind =
    NULL, ...){

  # type emupoints
  # for every discrete point recorded a fitted point is being
      computed
  if(type == "emupoints"){
    emuobject <- emuobject[ind]
    ret <- nam <- rep(NA, length(tracktimes(emuobject)))
    j = 1
    for(i in 1:nrow(emuobject)){
      n <- length(tracktimes(emuobject[i]))
      nam[j:(j+n-1)] <- rep(i, n)
      xaxis <- seq(0, 1, length.out = length(tracktimes(emuobject[
          i])))
      yhat <- eval.fd(object$reg$yhatfd$fd, xaxis)[, i]
      ret[j:(j+n-1)] <- yhat
      j = j + n
    }
    names(ret) <- nam
    return(ret)
  }

  # type curves
  # the fitted curves are returned
  if(type == "curves"){
    return(object$reg$yhatfdobj$fd)
  }

}
```

Listing 2: predict() function for class emu_fd

```
predict.emu_fd <- function(object, type = 'curves', newdata = NULL
    , ...){

  # return fitted object if newdata = NULL
```

```
  if(is.null(newdata)) return(fitted(object, type))

 # some comaprisons to see if newdata and olddata are of same type
  else{
    if(length(object$reg$xfdlist) != length(newdata))
      stop("New data has not the same length as test data!")
    for(i in 1:length(object$reg$xfdlist)){
      if((nrow(object$reg$xfdlist[[i]]$coefs) == 1 &&
          !(class(newdata[[i]]) %in% c('integer','numeric')))
          ||
          (nrow(object$reg$xfdlist[[i]]$coefs) > 1 &&
          class(newdata[[i]]) != 'fd')){
        stop(paste('Variable ', names(object$reg$xfdlist)[i],
          ' has a wrong type of class in newdata!', sep = ''))
      }
    }

    # adding curves and constants together
    ret <- const <- 0
    for(i in 1:length(object$reg$xfdlist)){
      if(class(newdata[[i]]) == 'fd')
        ret <- ret + object$reg$betaestlist[[i]]$fd * newdata[[i]]
      else const <- const + coef(object$reg$betaestlist[[i]]) *
          newdata[[i]]
    }

    for(i in 1:nrow(ret$coefs)) ret$coefs[i, ] <- ret$coefs[i, ] +
        const

    # unimplemented: return also other types, not just curves
    return(ret)
  }
}
\end{lstlisitng}

\begin{lstlisting}[caption = plot() function for class emu\_fd]
plot.emu_fd <- function(x, y, ...){

  # marking the constant and functional variables
  cons <- NULL
  fds <- NULL
  for(i in 1:length(x$reg$xfdlist)){
```

```
    if(nrow(coef(x$reg$betaestlist[[i]]$fd)) == 1)
45    cons <- c(cons, i)
    else
47    fds <- c(fds, i)
  }
49
  # defining how many plots to plot and how to arrange them
51  nr_plots <- length(fds) + as.numeric(!is.null(cons))
  if(nr_plots == 1)  op <- par(mfrow = c(1, 1), mgp = c(1, .5, 0),
      mar = c(2, 2, 2, 0))
53  else if(nr_plots == 2)  op <- par(mfrow = c(1, 2), mgp = c(1,
      .5, 0), mar = c(2, 2, 2, 0))
  else if(nr_plots <= 14)  op <- par(mfrow = c(ceiling(nr_plots/2)
      ,2), mgp = c(1,.5,0), mar = c(2,2,2,0))
55  else stop("Too many variables. Please plot manually!")
  on.exit(par(op))
57
  # at what points to plot the curves
59  time <- seq(x$reg$yfdPar$fd$basis$rangeval[1],
              x$reg$yfdPar$fd$basis$rangeval[2],
61            length.out = 100)

63  # plot functional data
  for(i in fds){
65    plot(time, eval.fd(time, x$reg$betaestlist[[i]]$fd), type="l",
        lwd = 2,
         xlab = "Time", ylab = "", main = names(x$reg$xfdlist)[i])
67    abline(h = 0, lty = 2)
  }
69
  # plot constant coefficients
71  plot(0:1, 0:1, type="n", main = "Constant Coefficients", xlab =
      "",
       ylab = "", xaxt = "n", yaxt = "n")
73  yax <- rep(seq(.95, 0.05, length.out = length(cons)/2), each =
      2)
  for(i in 1:length(cons)){
75    if(i %% 2 == 1)
    text(0, yax[i], pos = 4, labels =
77    paste(names(x$reg$xfdlist)[cons[i]], ": ",
          round(coef(x$reg$betaestlist[[cons[i]]]$fd), 2), sep = "
              "))
```

```
79      else
        text(0.5, yax[i], pos = 4, labels =
81      paste(names(x$reg$xfdlist)[cons[i]], ": ",
              round(coef(x$reg$betaestlist[[cons[i]]]$fd), 2), sep = "
                  "))
83    }
  }
```

Listing 3: residuals() function for class emu_fd

```
  residuals.emu_fd <- function(object, emuobject, type = curves, ind
      , ...){
2
    emuobject <- emuobject[ind]
4   if(type == 'emupoints'){
      ret <- nam <- rep(NA, length(tracktimes(emuobject)))
6     j = 1
      for(i in 1:nrow(emuobject)){
8       n <- length(tracktimes(emuobject[i]))
        nam[j:(j+n-1)] <- rep(i, n)
10      xaxis <- seq(0, 1, length.out = length(tracktimes(emuobject[
            i])))
        y <- frames(emuobject[i])
12      yhat <- eval.fd(xaxis, object$reg$yhatfd$fd)[, i]
        ret[j:(j+n-1)] <- y - yhat
14      j = j + n
      }
16    names(ret) <- nam
    }
18  if(type == 'fd_samples'){
      n <- nrow(emuobject)
20    ret <- rep(NA, n)
      nam <- 1:n
22    for(i in 1:n){
        xaxis <- seq(0, 1, length.out = length(tracktimes(emuobject[
            i])))
24      y <- frames(emuobject[i])
        yhat <- eval.fd(xaxis, object$reg$yhatfd$fd)[, i]
26      ret[i] <- mean(abs(y - yhat))
      }
28    names(ret) <- nam
    }
30  if(type == 'curves'){
```

```
    ret <- object$resp - object$reg$yhatfdobj$fd
  }
  return(ret)
}
```

Listing 4: coef() function for class emu_fd

```
coef.emu_fd <- function(object, ...){
  ret <- list()
  for(i in 1:length(object$reg$xfdlist)){
    co <- drop(coef(object$reg$betaestlist[[i]]))
    if(length(co) == 1) ret[[i]] <- co
    else ret[[i]] <- object$reg$betaestlist[[i]]$fd
  }
  names(ret) <- names(object$reg$xfdlist)
  ret
}
```

Listing 5: summary() function for class emu_fd

```
summary.emu_fd <- function(object, emuobject, ind, ...){
  co <- coef(object)
  cla <- sapply(co, class)
  num <- NULL
  means <- NULL
  medians <- NULL
  mins <- NULL
  maxs <- NULL
  sds <- NULL
  for(i in 1:length(co)){
    if(cla[i] == "numeric") num <- c(num, round(co[[i]], 3))
    else{
      fd <- eval.fd(seq(0, 1, length.out = 100), co[[i]])
      means <- c(means, round(mean(fd),3))
      medians <- c(medians, round(median(fd),3))
      mins <- c(mins, round(min(fd),3))
      maxs<- c(maxs, round(max(fd),3))
      sds <- c(sds, round(sd(fd),3))
    }
  }
  ret1 <- as.data.frame(matrix(NA, ncol = 2, nrow = length(num)))
  ret1[, 1] <- rep("numeric", length(num))
  ret1[, 2] <- num
```

```
  rownames(ret1) <- names(co[cla == "numeric"])
  colnames(ret1) <- c("          Class", "    Coefficient")
  ret2 <- as.data.frame(matrix(NA, ncol = 6, nrow = length(means))
      )
  ret2[, 1] <- rep("functional", length(means))
  ret2[, 2] <- means
  ret2[, 3] <- medians
  ret2[, 4] <- mins
  ret2[, 5] <- maxs
  ret2[, 6] <- sds
  rownames(ret2) <- names(co[cla == "fd"])
  colnames(ret2) <- c("          Class", "            Mean", "
          Median",
                      "        Minimum", "        Maximum", "Std.
                          Deviation")

  nobs <- ncol(object$reg$yhatfdobj$fd$coef)
  RSS <- sum(residuals(object, emuobject = emuobject, type="fd_
      samples", ind = ind)^2)
  MR <- mean(residuals(object, emuobject = emuobject, type="fd_
      samples", ind = ind))
  R <- mean(object$R)

  return(list(numeric_coef = ret1, functional_coef = ret2, nobs =
      nobs,
              RSS = RSS, MR = MR, Rsq = R))

}
```

Listing 6: fitted() function for class flm

```
fitted.flm <- function(object, type = 'curves', ...){
  fitted(object$model, type = type, object$data$emudata[[1]],
      object$data$ind, ...)
}
```

Listing 7: predict() function for class flm

```
predict.flm <- function(object, type = 'curves', newdata = NULL,
    ...){
  predict(object$model, type = type, newdata = newdata, ...)
}
```

Listing 8: plot() function for class flm

```
plot.flm <- function(x, y, ...){
  plot(x$model)
}
```

Listing 9: residuals() function for class flm

```
residuals.flm <- function(object, type = "curves", ...){
  residuals(object$model, object$data$emudata[[1]], type = type,
      object$data$ind, ...)
}
```

Listing 10: coef() function for class flm

```
coef.flm <- function(object, ...){
  coef(object$model, ...)
}
```

Listing 11: summary() function for class flm

```
summary.flm <- function(object, ...){
  ret <- summary(object$model, emuobject = object$data$emudata
      [[1]],
                 ind = object$data$ind, ...)
  ret$call <- object$call
  class(ret) <- "summary.flm"
  return(ret)
}
```

Listing 12: print() function for class summary.flm

```
print.summary.flm <- function(x, ...){
  cat("\n")
  cat("Functional regression of phonetic data.\n\n")
  cat("Call:", deparse(x$call), "\n\n")
  cat("Estimated numeric coefficients:\n")
  print(x$numeric_coef)
  cat("\n")
  cat("Estimated functional coefficients (summarized):\n")
  print(x$functional_coef)
  cat("\n")
  cat("Number of observations:", x$nobs, "\n" )
  cat("Mean R squared:", x$Rsq, "\n" )
  cat("Residual sum of squares:", x$RSS, "\n" )
```

```
  cat("Mean residual:", x$MR, "\n" )
}
```

Listing 13: print() function for class flm

```
print.flm <- function(x, ...){
  cat("\n")
  cat("  Functional linear model of phonetic data. Please use
      generic functions such as\n\n")
  cat("  \t summary(), plot(), residuals(), coef(), predict() or
      fitted()\n\n")
  cat("  for further data analysis.\n\n")
}
```

A.2. Computation and display of bootstrap based confidence intervals

Listing 14: Computation of bootstrap based confidence intervals

```
bootstd <- function(object, B = 1000, alpha = 0.05){
  x <- object$model$reg$xfdlist
  y <- object$model$reg$yfdPar$fd
  n <- ncol(object$model$reg$yhatfdobj$fd$coef)
  time <- seq(0, 1, length.out = 100)
  fd_mat <- matrix(NA, ncol = 100, nrow = B)
  num_vec <- rep(NA, B)
  co <- coef(object)
  cla <- sapply(co, class)
  beta <- list()
  for(i in 1:length(x)){
    if(cla[i] == "numeric") beta[[i]] <- num_vec
    if(cla[i] == "fd") beta[[i]] <- fd_mat
  }

  for(b in 1:B){
    print(b)
    indb <- sample(n, replace = TRUE)
    regdatab <- lapply(object$data$regdata, function(x, ind) x[ind
        ], ind = indb)
    indb2 <- object$data$ind[indb]
    emudatab <- NULL                          # emudata not needed
        here
```

```
22      datab <- list(regdata = regdatab, ind = indb2, emudata =
            emudatab)
        newreg <- flm(object$call, data = datab, control = object$
            control)
24      cob <- coef(newreg)
        for(i in 1:length(x)){
26        if(cla[i] == "numeric") beta[[i]][b] <- cob[[i]]
          else beta[[i]][b, ] <- eval.fd(time, cob[[i]])
28      }
    }
30  names(beta) <- names(x)

32  lo <- ceiling(alpha/2*B)
    up <- floor((1-alpha/2)*B)
34  perz <- list()
    for(i in 1:length(x)){
36    if(cla[i] == "numeric") perz[[i]] <- sort(beta[[i]])[c(lo, up)
          ]
      else{
38      perz[[i]] <- matrix(NA, ncol = 100, nrow = 2)
        for(j in 1:100) perz[[i]][, j] <- sort(beta[[i]][, j])[c(lo,
            up)]
40    }
    }
42

    ret <- list(beta = beta, perz = perz, flmobject = object)
44  class(ret) <- "flmboot"
    return(ret)
46  }
```

Listing 15: Plotting of bootstrap based confidence intervals

```
    plot.flmboot <- function(x, y, ...){
2

      xsave <- x
4     perz <- x$perz
      x <- x$flmobject$model
6     # marking the constant and functional variables
      cons <- NULL
8     fds <- NULL
      for(i in 1:length(x$reg$xfdlist)){
10      if(nrow(coef(x$reg$betaestlist[[i]]$fd)) == 1)
          cons <- c(cons, i)
```

```
    else
      fds <- c(fds, i)
  }

  # defining how many plots to plot and how to arrange them
  nr_plots <- length(fds) + as.numeric(!is.null(cons))
  if(nr_plots == 1)  op <- par(mfrow = c(1, 1), mgp = c(1, .5, 0),
      mar = c(2, 2, 2, 0.3))
  else if(nr_plots == 2)  op <- par(mfrow = c(1, 2), mgp = c(1,
     .5, 0), mar = c(2, 2, 2, 0.3))
  else if(nr_plots <= 14)  op <- par(mfrow = c(ceiling(nr_plots/2)
     ,2), mgp = c(1,.5,0), mar = c(2,2,2,0.3))
  else stop("Too many variables. Please plot manually!")
  on.exit(par(op))

  # at what points to plot the curves
  time <- seq(x$reg$yfdPar$fd$basis$rangeval[1],
              x$reg$yfdPar$fd$basis$rangeval[2],
              length.out = 100)

  # plot functional data
  for(i in fds){
    plot(time, eval.fd(time, x$reg$betaestlist[[i]]$fd), type="l",
        lwd = 2,
         xlab = "Time", ylab = "", main = names(x$reg$xfdlist)[i],
         ylim = range(perz[[i]]))
    lines(time, perz[[i]][1,], lty = 2)
    lines(time, perz[[i]][2,], lty = 2)
    abline(h = 0, lty = 3)
  }

  # plot constant coefficients
  plot(0:1, 0:1, type="n", main = "Constant Coefficients", xlab =
     "",
       ylab = "", xaxt = "n", yaxt = "n")
  yax <- rep(seq(.95, 0.05, length.out = length(cons)/2), each =
     2)
  for(i in 1:length(cons)){
    if(i %% 2 == 1)
    text(0, yax[i], pos = 4, labels =
    paste(names(x$reg$xfdlist)[cons[i]], ": ",
          round(coef(x$reg$betaestlist[[cons[i]]]$fd), 2),
```

```
48            " [", round(perz[[cons[i]]][1], 2), ", ",
              round(perz[[cons[i]]][2], 2), "]",
50            sep = ""))
    else
52    text(0.5, yax[i], pos = 4, labels =
    paste(names(x$reg$xfdlist)[cons[i]], ": ",
54            round(coef(x$reg$betaestlist[[cons[i]]]$fd), 2),
              " [", round(perz[[cons[i]]][1], 2), ", ",
56            round(perz[[cons[i]]][2], 2), "]",
              sep = ""))
58  }
}
```

B. Graphics

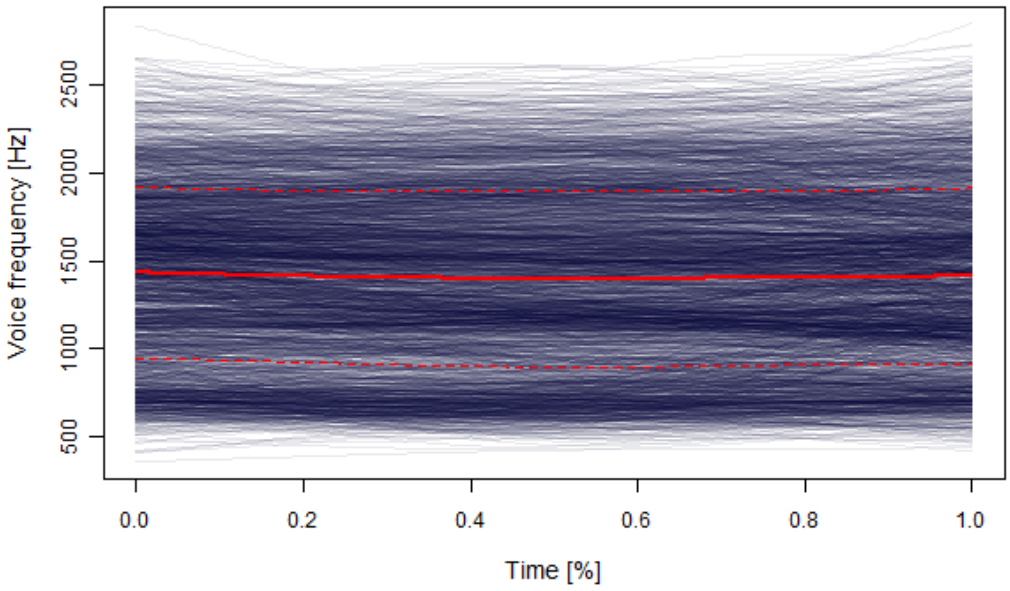


Figure 13: All replications of the voice frequency with mean and standard deviation

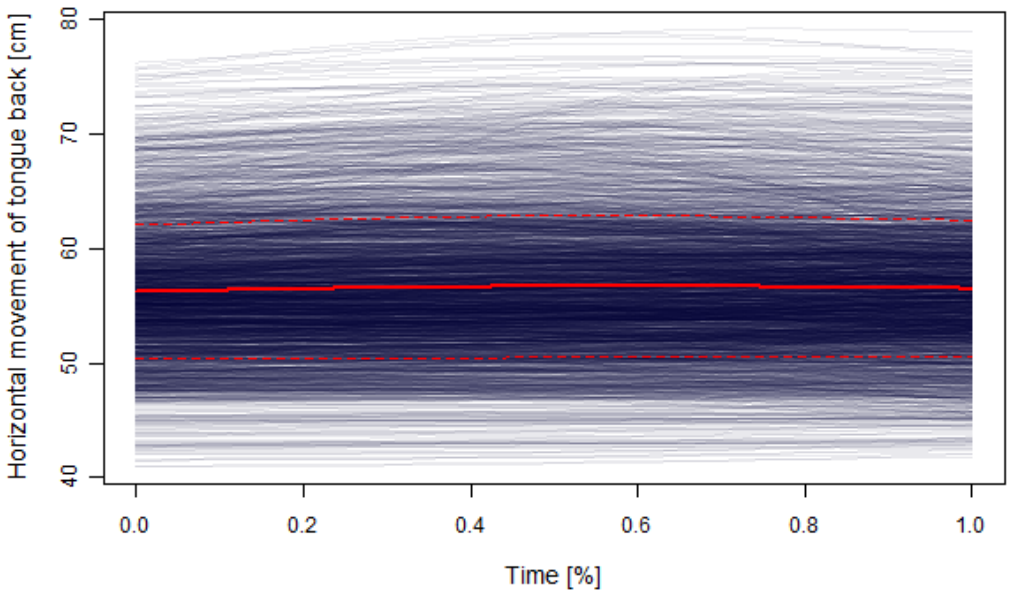


Figure 14: All replications of the horizontal movement of the tongue back with mean and standard deviation

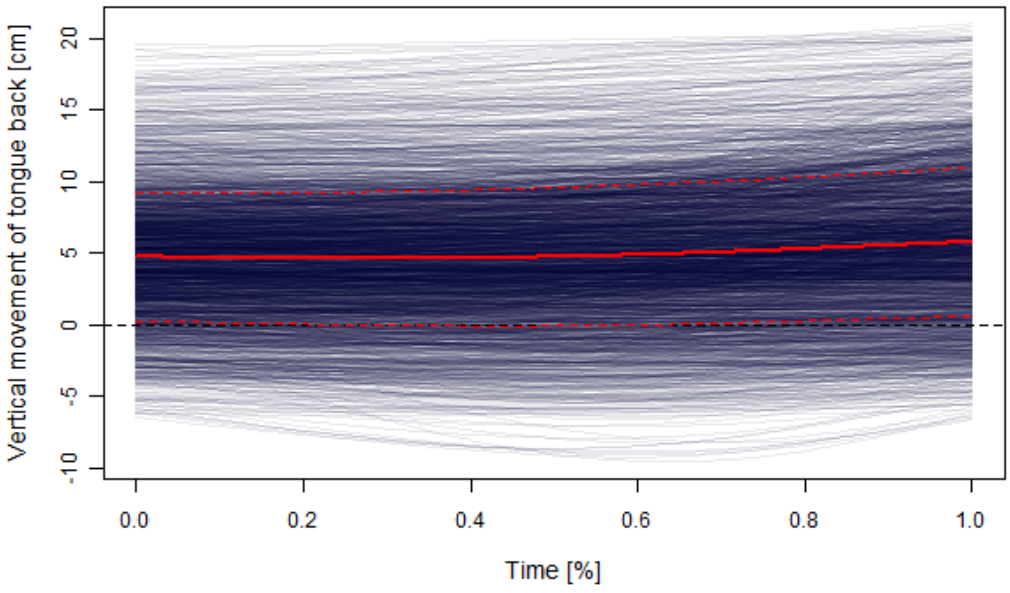


Figure 15: All replications of the vertical movement of the tongue back with mean and standard deviation

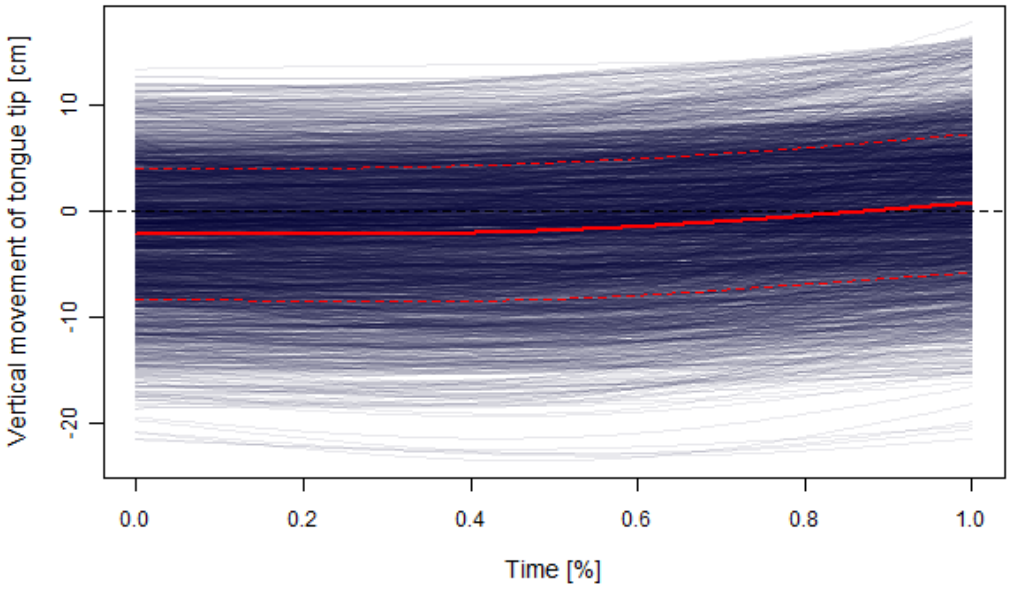


Figure 16: All replications of the vertical movement of the tongue tip with mean and standard deviation

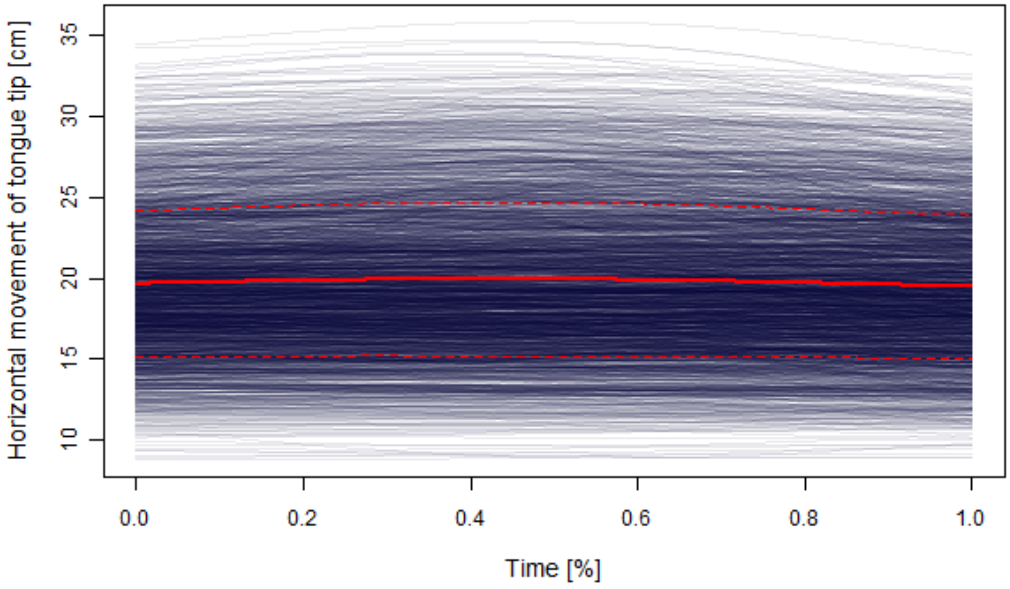


Figure 17: All replications of the horizontal movement of the tongue tip with mean and standard deviation

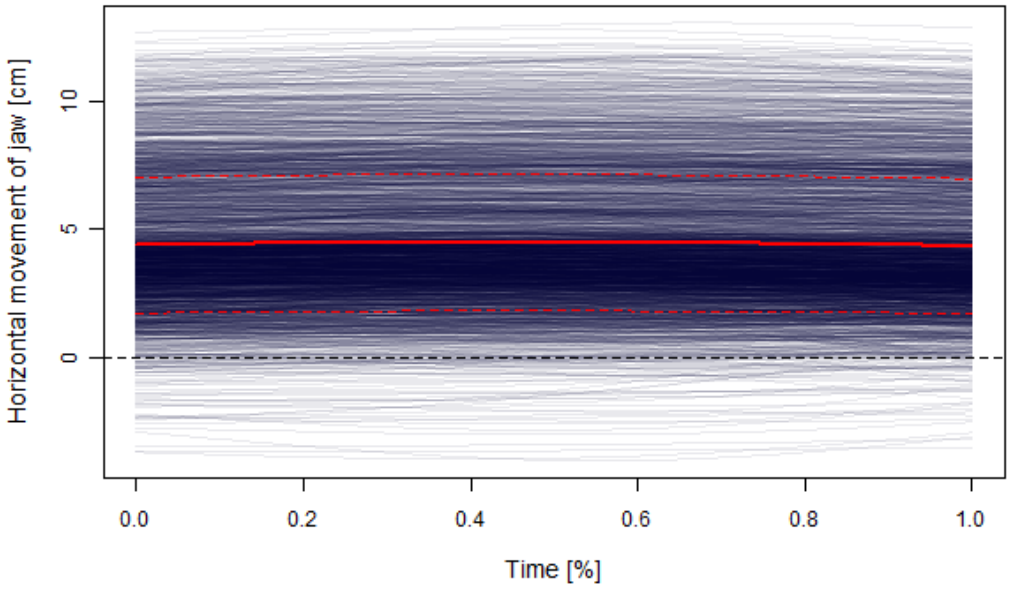


Figure 18: All replications of the horizontal movement of the jaw with mean and standard deviation

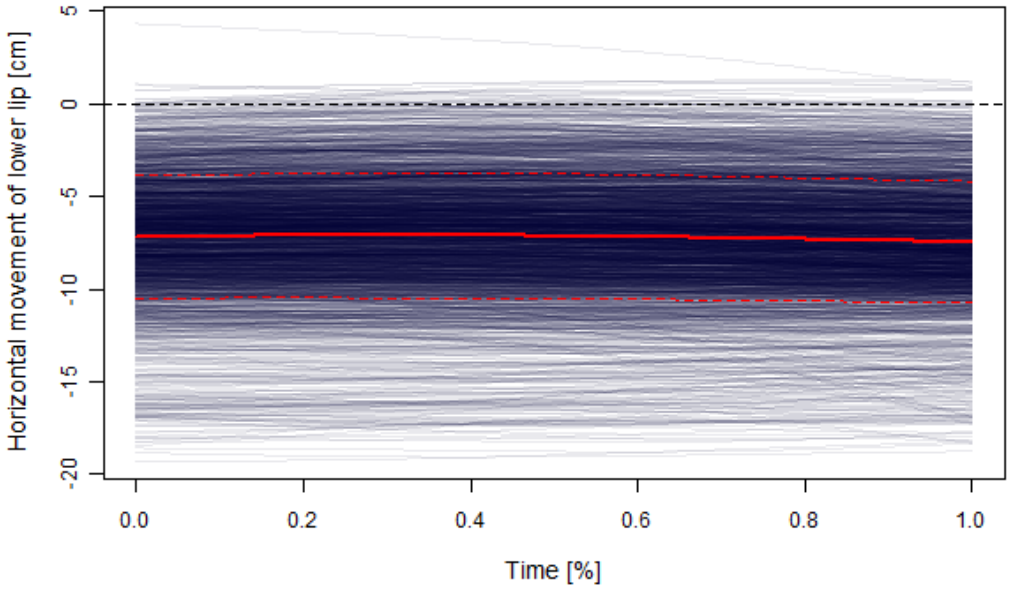


Figure 19: All replications of the horizontal movement of the lower lip with mean and standard deviation

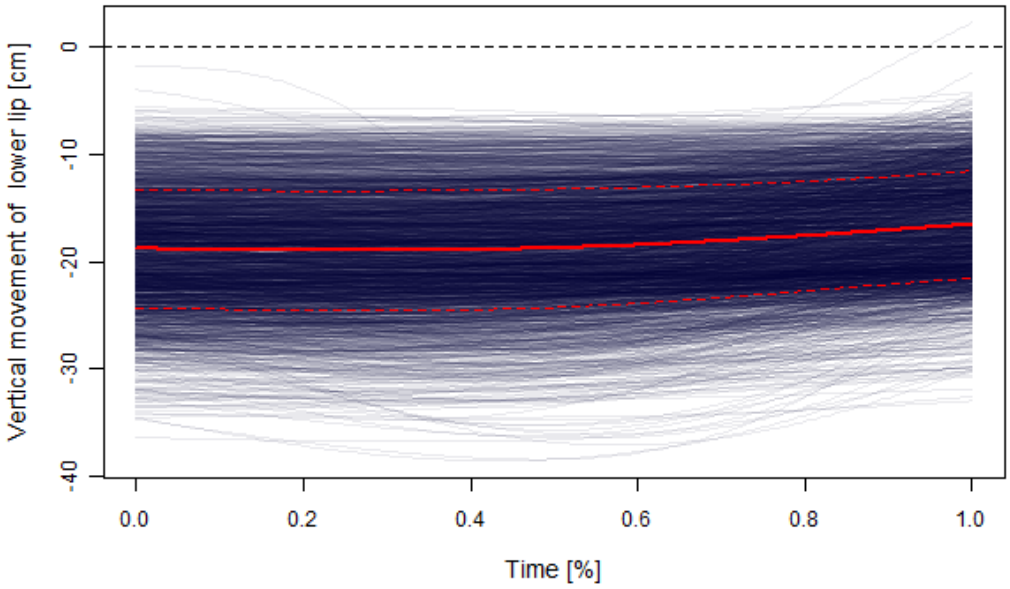


Figure 20: All replications of the vertical movement of the lower lip with mean and standard deviation

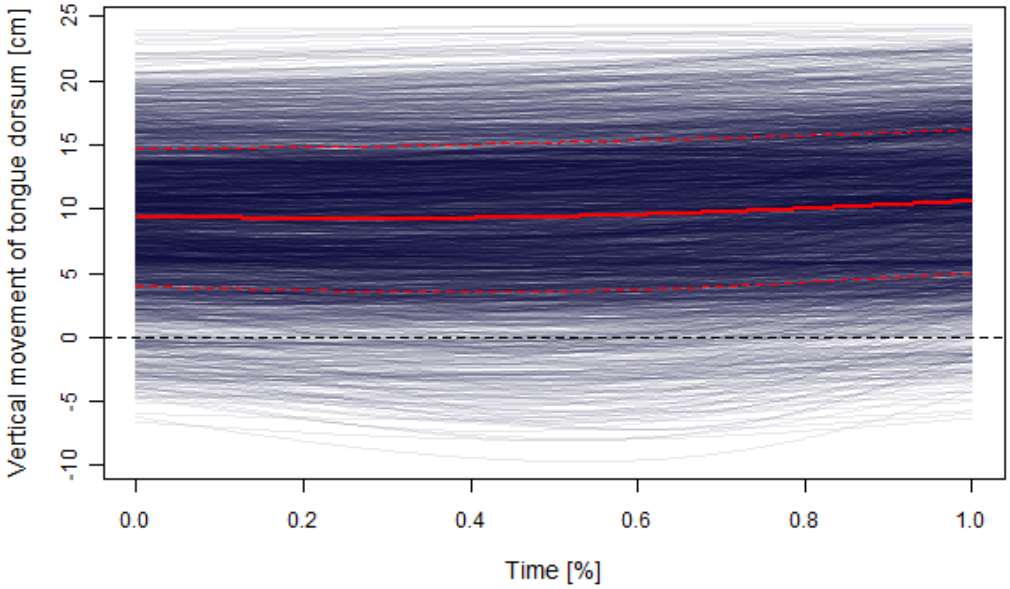


Figure 21: All replications of the vertical movement of the tongue dorsum with mean and standard deviation

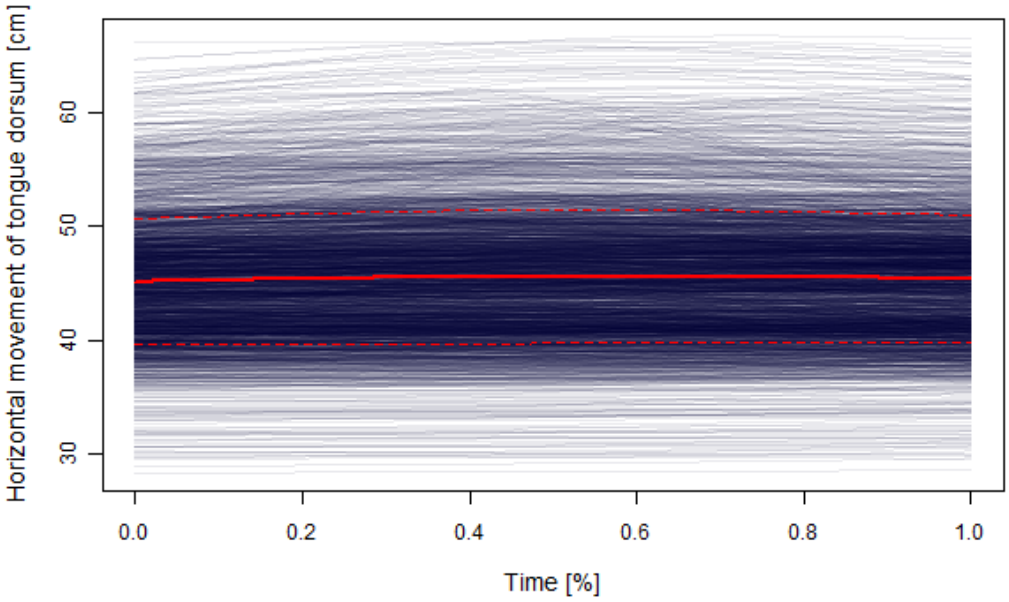


Figure 22: All replications of the horizontal movement of the tongue dorsum with mean and standard deviation

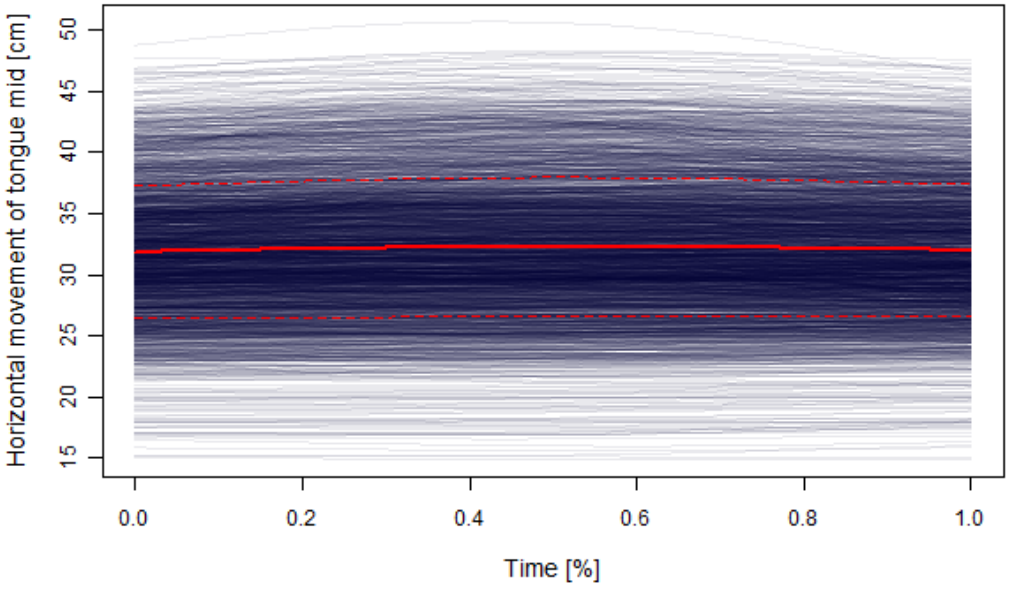


Figure 23: All replications of the horizontal movement of the tongue mid with mean and standard deviation

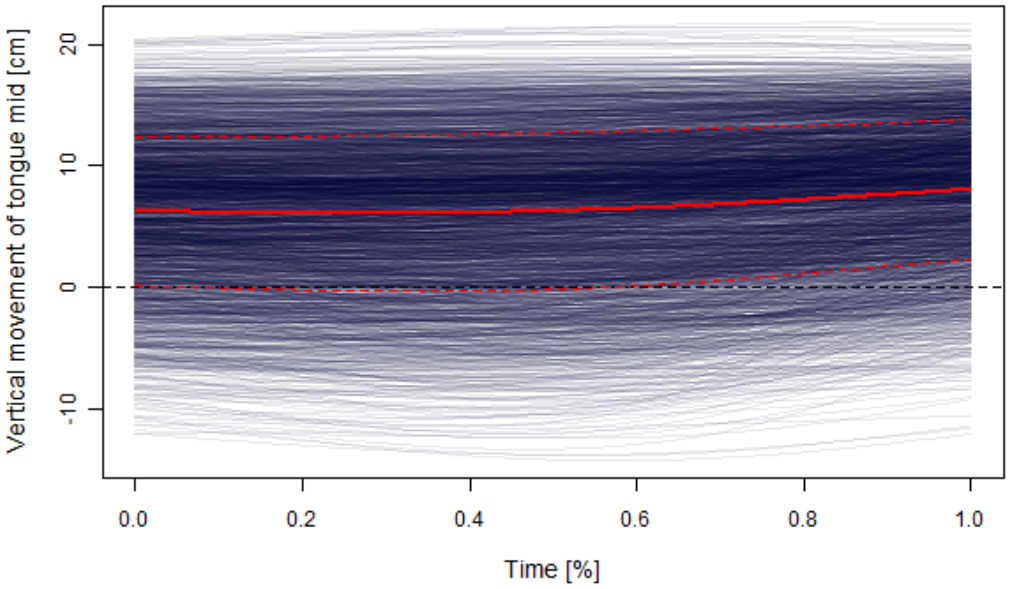


Figure 24: All replications of the vertical movement of the tongue mid with mean and standard deviation

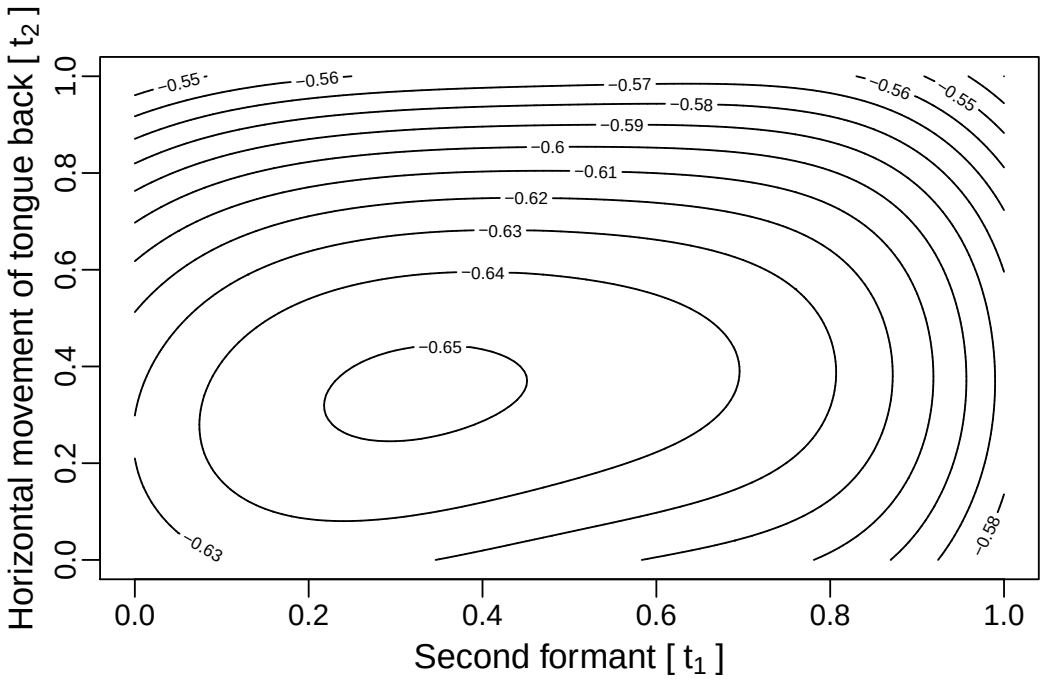


Figure 25: Cross correlation plot of voice frequency and horizontal movement of tongue back

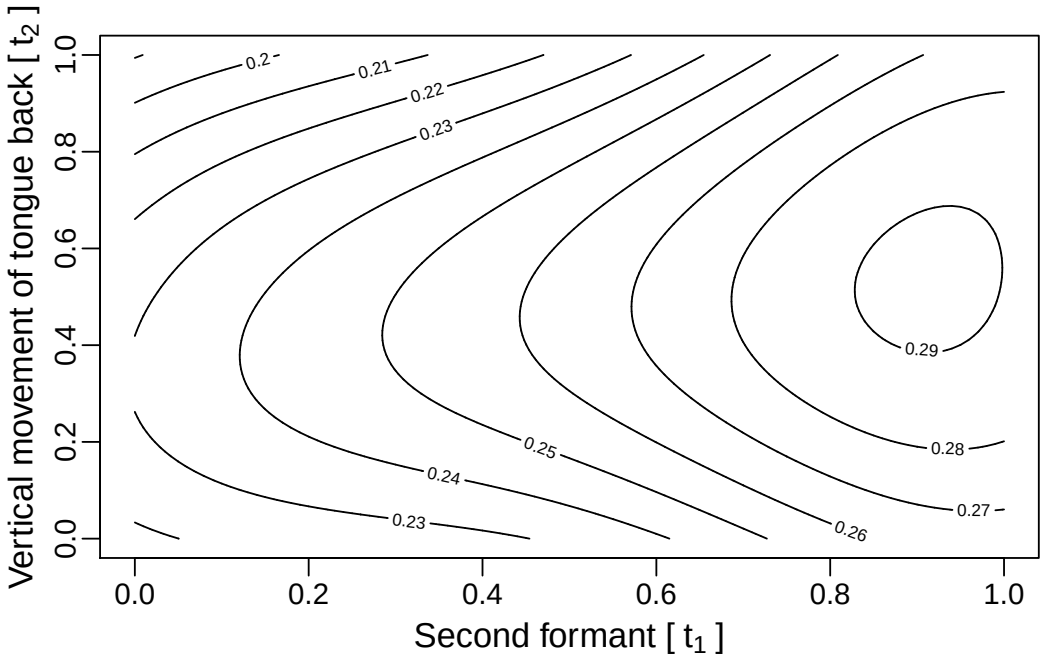


Figure 26: Cross correlation plot of voice frequency and vertical movement of tongue back

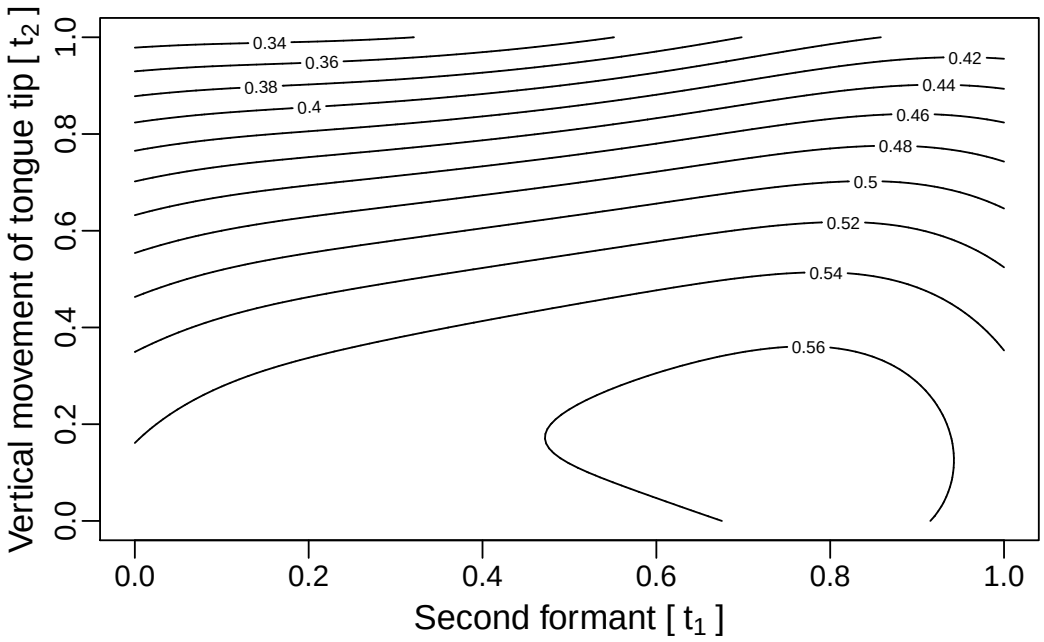


Figure 27: Cross correlation plot of voice frequency and vertical movement of tongue tip

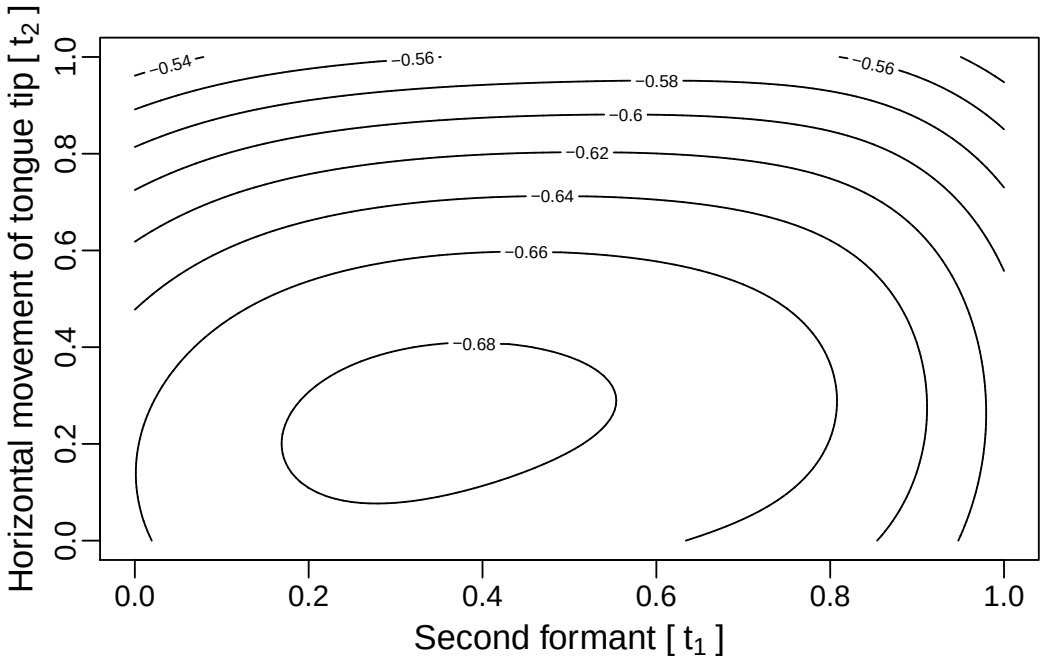


Figure 28: Cross correlation plot of voice frequency and horizontal movement of tongue tip

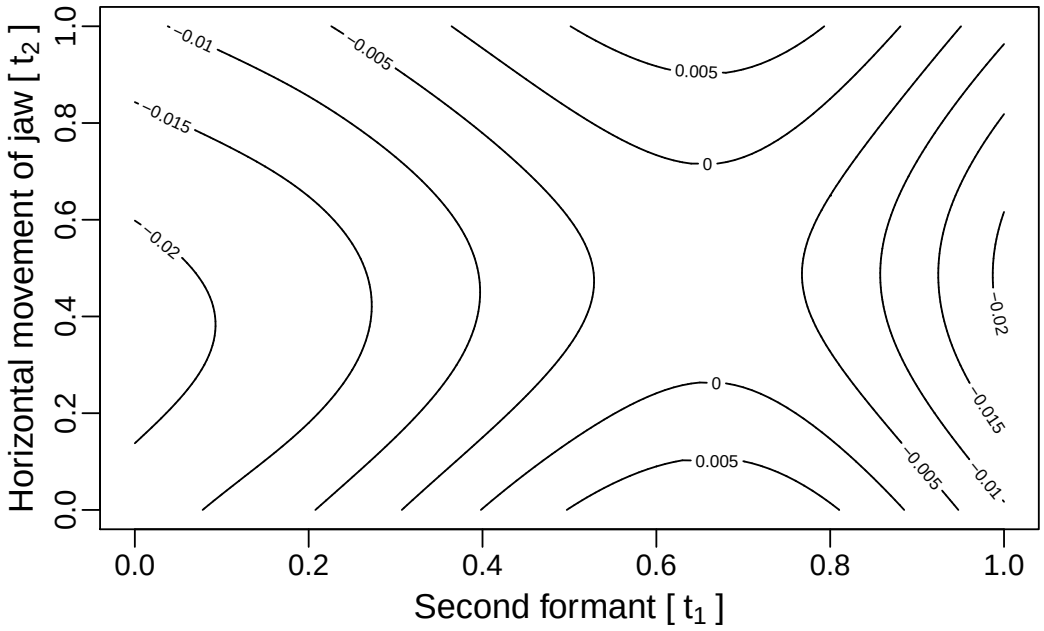


Figure 29: Cross correlation plot of voice frequency and horizontal movement of jaw

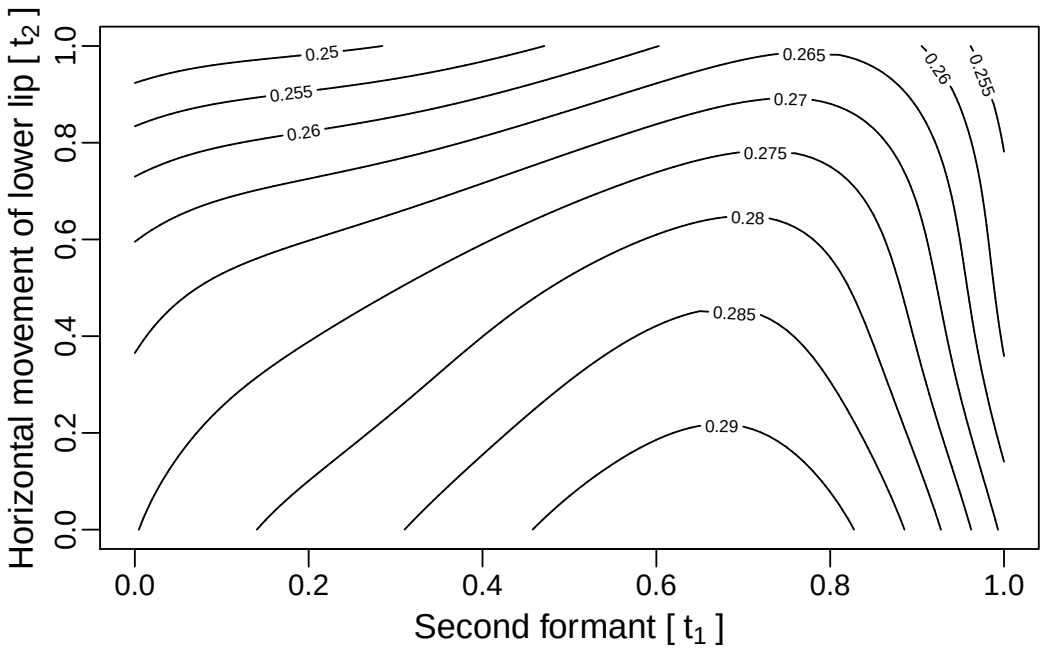


Figure 30: Cross correlation plot of voice frequency and horizontal movement of lower lip

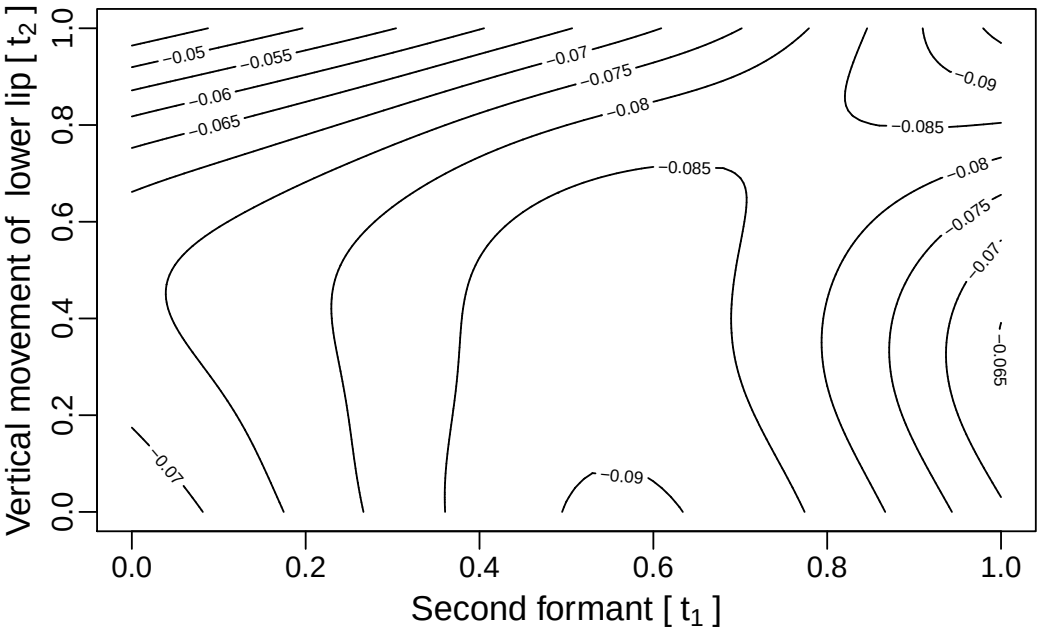


Figure 31: Cross correlation plot of voice frequency and vertical movement of lower lip

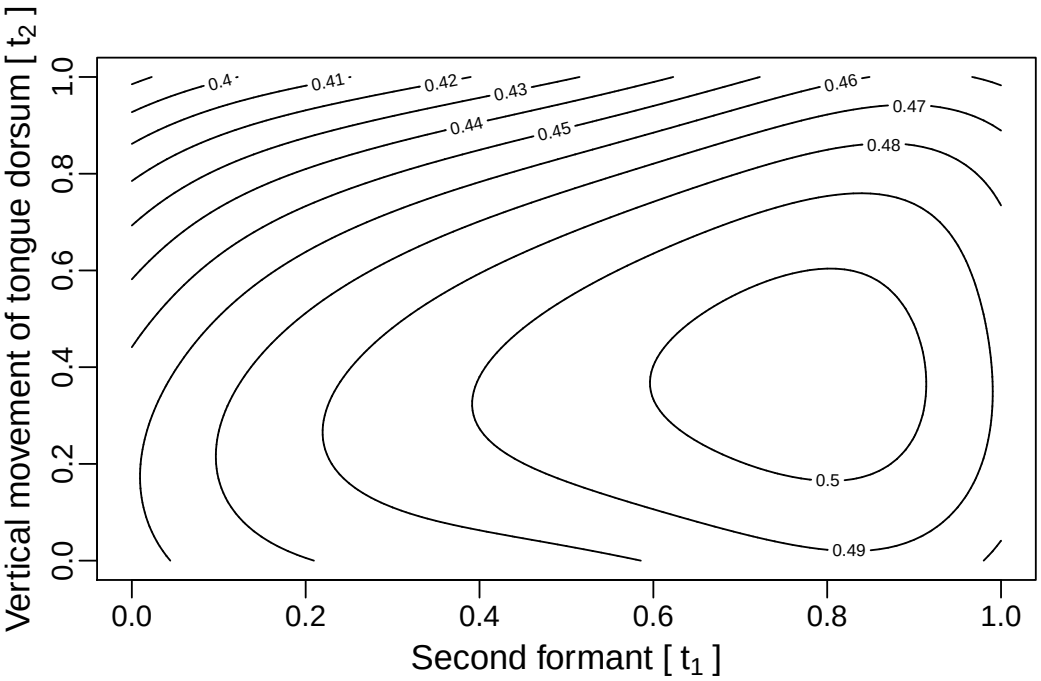


Figure 32: Cross correlation plot of voice frequency and vertical movement of tongue dorsum

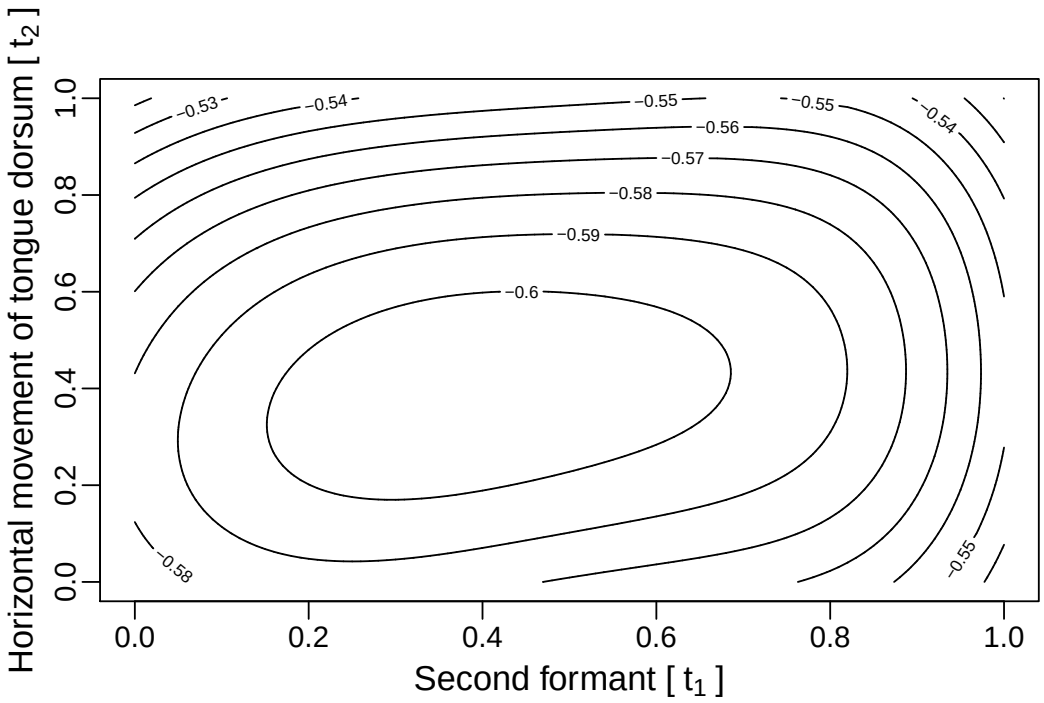


Figure 33: Cross correlation plot of voice frequency and horizontal movement of tongue dorsum

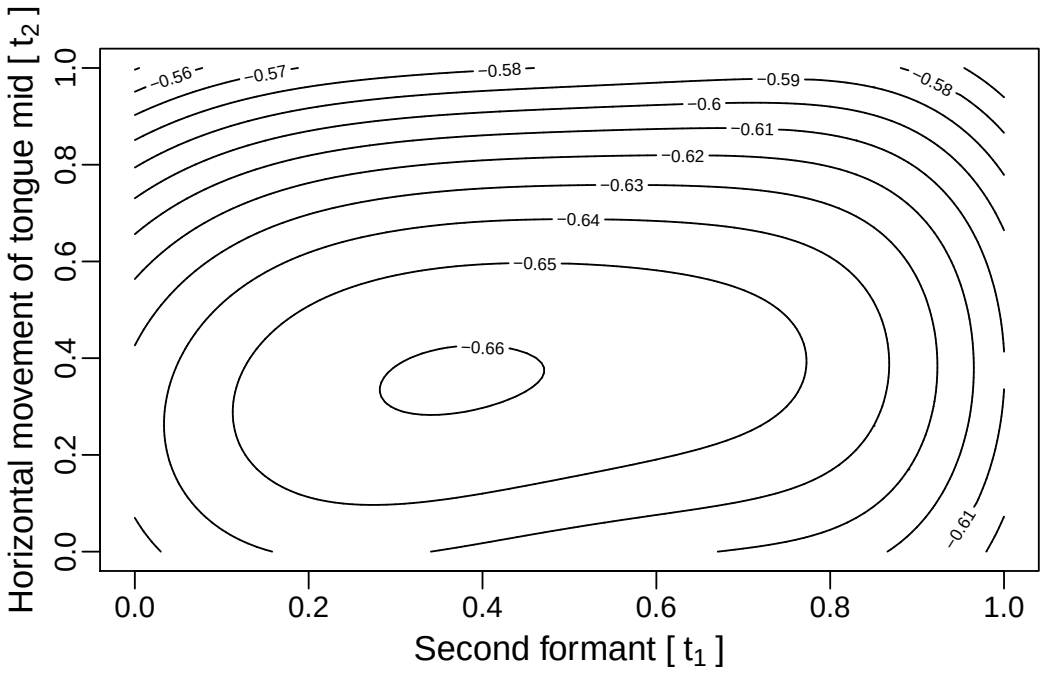


Figure 34: Cross correlation plot of voice frequency and horizontal movement of tongue mid

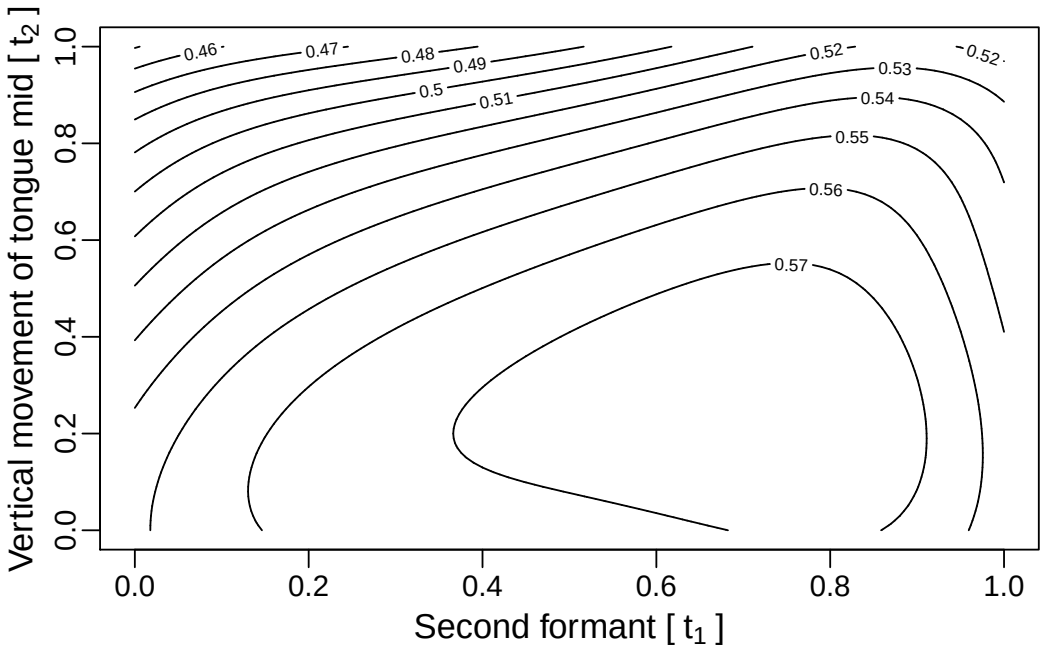


Figure 35: Cross correlation plot of voice frequency and vertical movement of tongue mid

Erklärung zur Urheberschaft

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe.

München, den 04.10.2010

(Ivan Kondofersky)