A Functional PCA Session in Detail

Michele Gubian

Centre for Language and Speech Technology
Radboud University Nijmegen
The Netherlands

Interspeech 2011
27 August 2011, Florence, Italy

Outline I

1. Data Preparation
   - The Case Study
   - Input Data
   - Preparing Data in R
2. Smoothing
   - Smoothing as Trade-Off
   - Generalised Cross-Validation (GCV)
   - Maximum Rate of F_0 Change (Xu & Sun, 2002)
   - Code
3. Landmark Registration
   - Principles
   - Code
4. Fuctional PCA

Outline II

- Theory
- Code
- Understanding the Principal Components

5 Robustness

- ‘Wrong’ Smoothing Parameters
- Skipping Landmark Registration
- Ordinary PCA on B-Splines Coefficients
- Voiceless sounds and f_0 tracker errors

Question/Statement Opposition in Neapolitan Italian

In Neapolitan Italian the Q/S opposition is expressed solely by intonation.

Example

“A<u>me</u>lia dorme da <u>no</u>nna (?)” = *Amelia sleeps at grandma’s (?)*

- <u>Underlined</u> syllables carry an accent
- focus is on the first accent, realised as F_0 peak
- we expect that peak to shift in time according to Q/S

Material

- Read speech
- 3 male speakers
- 3 sentences with identical syllabic structure
- 2 modalities (Q/S)
- 5 repetitions per speaker/sentence/modality
- total: 87 utterances (3 discarded)

F_0 Extraction

- F_0 is extracted from each utterance
- store each F_0 contour in text format in a separate file
- If you use Praat, you can iterate a code like this:

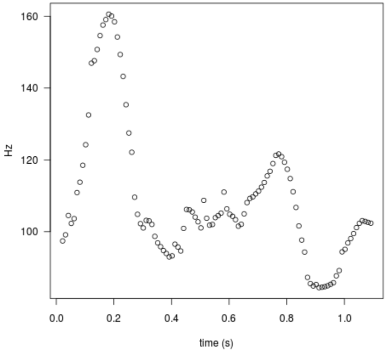
```
Read from file... your_file.wav
select Sound your_file
To Pitch (ac)...  0 75 15 no  0.03 0.45 0.01 0.35 0.14 600
Down to PitchTier
Write to headerless spreadsheet file... your_file.txt
select Sound your_file
plus Pitch your_file
plus PitchTier your_file
Remove
```

F_0 Extraction

- F_0 is extracted from each utterance
- store each F_0 contour in text format in a separate file
- If you use Praat, you can iterate a code like this:

```
Read from file... your_file.wav
select Sound your_file
To Pitch (ac)...  0 75 15 no  0.03 0.45 0.01 0.35 0.14 600
Down to PitchTier
Write to headerless spreadsheet file... your_file.txt
select Sound your_file
plus Pitch your_file
plus PitchTier your_file
Remove
```

A Look at the Raw Data




```
0.021678004535147441 97.520668167596128
0.031678004535147443 101.08220705922977
0.041678004535147445 99.24533666065679
0.05167800453514744 98.724662819231213
0.061678004535147442 99.302246322973005
0.071678004535147444 100.84047831333012
0.081678004535147439 102.00821916185417
0.091678004535147448 102.75790484538685
0.10167800453514744 106.61468810043968
0.11167800453514744 110.98996349766793
0.12167800453514745 115.95315262762335
0.13167800453514744 123.07736510420415
0.14167800453514745 140.08431597335237
0.15167800453514743 139.7256329584217
0.16167800453514744 143.9872809191987
0.17167800453514745 146.49693861531492
0.18167800453514743 151.02849979126009
0.19167800453514744 154.69488003510585
0.20167800453514745 157.12971112297956
0.21167800453514746 157.77626192507461
0.22167800453514744 157.20544666108123
.....
```

... and Metadata

	filename	type	speaker	sentence	v1_beg	v1_end	v2_beg	v2_end
1	QNF_3_a_AS	Q	AS	3	0.125	0.267	0.760	0.867
2	QNF_3_a_SC	Q	SC	3	0.130	0.263	0.884	0.992
3	QNF_3_b_AS	Q	AS	3	0.117	0.252	0.752	0.858
....								
87	SNF_3_e_DC	S	DC	3	0.104	0.349	0.885	1.042

- Onset and offset of the two accented vowels will be used as landmarks
- we expect F_0 to be synchronised with respect to those points
- Alternatively, we could use every syllable or phone boundary
- an ASR can be used to obtain those boundaries by forced alignment

... and Metadata

	filename	type	speaker	sentence	v1_beg	v1_end	v2_beg	v2_end
1	QNF_3_a_AS	Q	AS	3	0.125	0.267	0.760	0.867
2	QNF_3_a_SC	Q	SC	3	0.130	0.263	0.884	0.992
3	QNF_3_b_AS	Q	AS	3	0.117	0.252	0.752	0.858
....								
87	SNF_3_e_DC	S	DC	3	0.104	0.349	0.885	1.042

- Onset and offset of the two accented vowels will be used as landmarks
- we expect F_0 to be synchronised with respect to those points
- Alternatively, we could use every syllable or phone boundary
- an ASR can be used to obtain those boundaries by forced alignment

... and Metadata

	filename	type	speaker	sentence	v1_beg	v1_end	v2_beg	v2_end
1	QNF_3_a_AS	Q	AS	3	0.125	0.267	0.760	0.867
2	QNF_3_a_SC	Q	SC	3	0.130	0.263	0.884	0.992
3	QNF_3_b_AS	Q	AS	3	0.117	0.252	0.752	0.858
....								
87	SNF_3_e_DC	S	DC	3	0.104	0.349	0.885	1.042

- Onset and offset of the two accented vowels will be used as landmarks
- we expect F_0 to be synchronised with respect to those points
- Alternatively, we could use every syllable or phone boundary
- an ASR can be used to obtain those boundaries by forced alignment

The fda R package

- We use the R package fda, by J. O. Ramsay, Hadley Wickham, Spencer Graves and Giles Hooker
- download it (any operating system) at:
  `http://cran.r-project.org/web/packages/fda/index.html`
- our first line of R code will be:
  ```
  library(fda)
  ```
- an alternative MATLAB version is also freely available, maintained by the same authors

Preparing F_0 Contours

- Each F_0 contour has a different duration
- the number and time location of F_0 samples vary across the 87 contours
- we have to use lists (and not matrices) to represent them
- Use semitones (and not Hz) to reduce excursion variation
- Subtract the average value across time from each F_0 contour to reduce speaker dependent variability
- save this average in case you think it is relevant to your analysis and reintroduce it after PCA

Preparing F_0 Contours

- Each F_0 contour has a different duration
- the number and time location of F_0 samples vary across the 87 contours
- we have to use lists (and not matrices) to represent them
- Use semitones (and not Hz) to reduce excursion variation
- Subtract the average value across time from each F_0 contour to reduce speaker dependent variability
- save this average in case you think it is relevant to your analysis and reintroduce it after PCA

Preparing F_0 Contours

- Each F_0 contour has a different duration
- the number and time location of F_0 samples vary across the 87 contours
- we have to use lists (and not matrices) to represent them
- Use semitones (and not Hz) to reduce excursion variation
- Subtract the average value across time from each F_0 contour to reduce speaker dependent variability
- save this average in case you think it is relevant to your analysis and reintroduce it after PCA

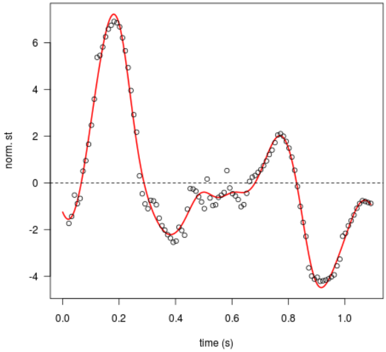
Preparing F_0 Contours

```
for (i in 1:n_contours) {
f0_contour =
read.table(paste(data_dir,metadata$filename[i],'.txt',sep=''),h=F)
```

show metadata   show f0_contour

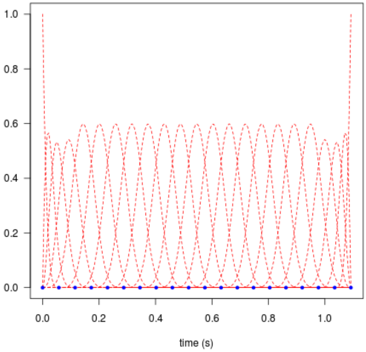
```
time_list[[i]] = f0_contour[,1]
f0_list[[i]] =  f0_contour[,2]
# turn Hz into semitones
st_list[[i]] = 12 * logb(f0_list[[i]], base = 2)
# save the mean value and subtract it away from the contour
mean_st = c(mean_st, mean(st_list[[i]]))
norm_st_list[[i]] = st_list[[i]] - mean_st[i]
# save duration and number of samples for each f0 contour
len = c(len, length( f0_sample[,1]))
duration = c(duration,max(time_list[[i]])
}
```

Smoothing



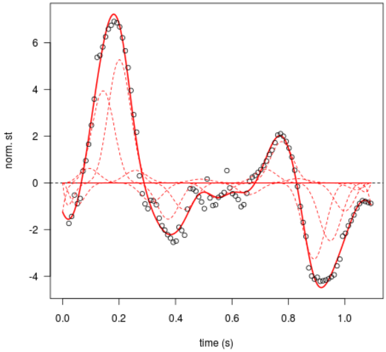
- smoothing is the operation of representing a sampled contour by a smooth curve expressed by a mathematical function

Smoothing with B-Splines



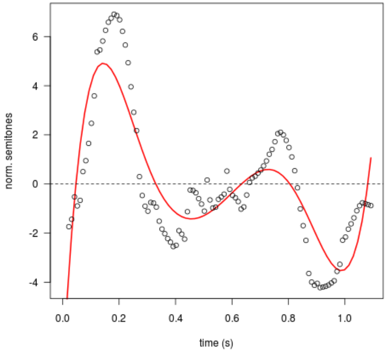
- The function is chosen from a specific set of functions called basis
- a good general-purpose basis is called B-splines
- B-splines are overlapping ‘humps’ that can approximate many shapes
- the number and position of humps is specified by the knots (in blue)

Smoothing with B-Splines



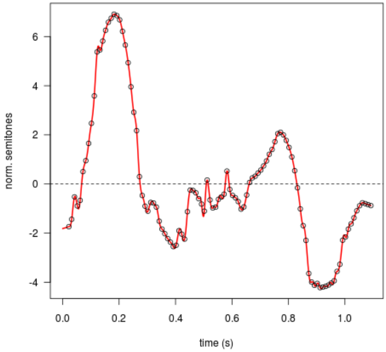
- Choosing one function from a B-splines basis means to determine the hump weights
- the weighted humps composed (summed) together form the smoothing function

Smoothing as Trade-Off: Underfitting



- When smoothing an F_0 contour you want to capture enough of its shape features

Smoothing as Trade-Off: Overfitting



- But you don't want to follow every F_0 sample, because F_0 trackers make errors and because you may not be interested in microprosodic effects

One Principle, Two Parameters

- One Principle: smoothing with roughness penalty

$$\min\{\mathit{Fitting\ Error} + \lambda \cdot \mathit{Roughness}\}$$

  - small *Fitting Error* but high *Roughness* (overfitting) example
  - small *Roughness* but high *Fitting Error* (underfitting) example
- Two Parameters:
  1. number of B-splines humps (or number of knots, k)
  2. smoothing parameter λ

Three Ways to Find a Good Trade-off

1. Use qualitative prior knowledge (eye inspection)
2. Use quantitative model selection
   - Generalised Cross-Validation GCV
3. Use quantitative prior knowledge
   - *"Maximum speed of pitch change and how it may relate to speech"* by Yi Xu and Xuejing Sun, JASA 2002 Xu & Sun

Smoothing Code

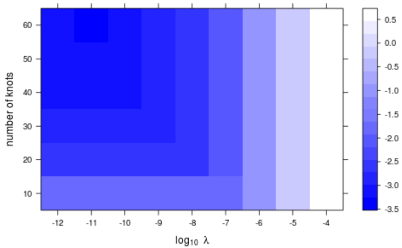
Generalised Cross-Validation (GCV)

- CV is an empirical method to estimate a 'fair' fitting error
  - smooth a contour using only a part of the samples
  - calculate fitting error on the left out samples
  - reiterate on different sample splittings
- CV helps preventing to choose parameter combinations (λ, k) that lead to overfitting
- CV is computationally expensive
- GCV is a lighter approximation of CV

Generalised Cross-Validation (GCV)

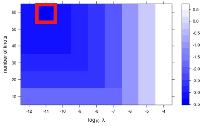
- CV is an empirical method to estimate a 'fair' fitting error
  - smooth a contour using only a part of the samples
  - calculate fitting error on the left out samples
  - reiterate on different sample splittings
- CV helps preventing to choose parameter combinations (λ, k) that lead to overfitting
- CV is computationally expensive
- GCV is a lighter approximation of CV

GCV Procedure

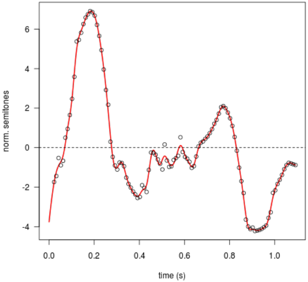


- Try many parameter combinations (λ, k), usually on a grid
- possibly refine the grid around interesting values
- choose a value

GCV Procedure

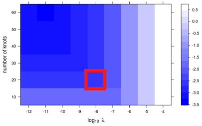


$$\lambda = 10^{-11}$$
$$k = 60$$

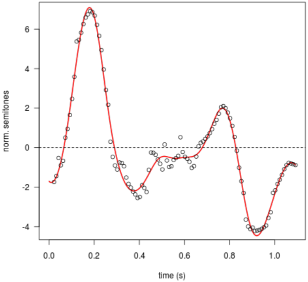


- Usually choosing the smallest GCV error is not a good idea
- GCV error tends to favour overfitting

GCV Trade-off Solution



$$\lambda = 10^{-8}$$
$$k = 20$$



- Take also the solution complexity into account
- among solutions (λ, k) with equal GCV error choose the one with minimum complexity (k) and maximum roughness penalization (λ)

Summary

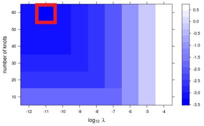
- Generalised Cross-Validation helps choosing good trade-off parameters for smoothing
- however, it cannot be used ‘blindly’ (i.e. pure GCV error minimization)
- what a good trade-off means actually depends on the phenomenon at study

Maximum Rate of F_0 Change (Xu & Sun, 2002)

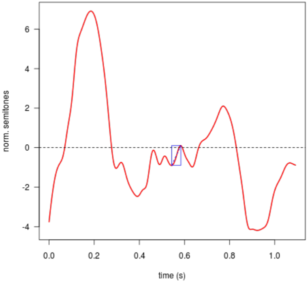
- Empirically obtained linear equations relate the F_0 excursion in a voluntary gesture to its corresponding maximum achievable F_0 change rate
- for a rising gesture, average and top F_0 change rate do not exceed the following values:

$$\begin{aligned} \text{mean } F_0 \text{ change rate [st/s]} &\leq 10.8 + 5.6 \cdot \text{excursion [st]} \\ \text{top } F_0 \text{ change rate [st/s]} &\leq 12.4 + 10.5 \cdot \text{excursion [st]} \end{aligned}$$

Check mean F_0 change rate

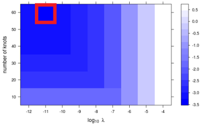


$$\lambda = 10^{-11}$$
$$k = 60$$

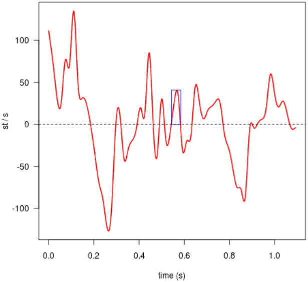


mean F_0 change rate:
24.4 st/s
max mean F_0 change rate:
16.4 st/s

Check top F_0 change rate

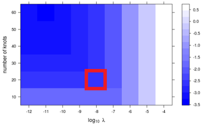


$$\lambda = 10^{-11}$$
$$k = 60$$



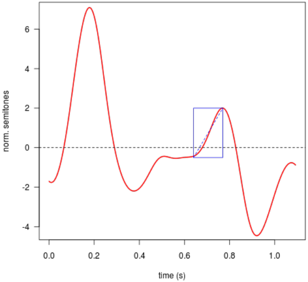
top F_0 change rate:
41 st/s
max top F_0 change rate:
22.9 st/s

Check mean F_0 change rate

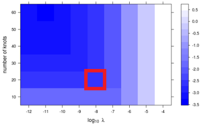


$$\lambda = 10^{-8}$$
$$k = 20$$

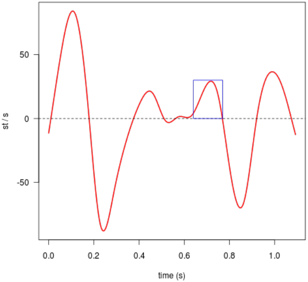
mean F_0 change rate:
19.2 st/s
max mean F_0 change rate:
24.8 st/s



Check top F_0 change rate



$$\lambda = 10^{-8}$$
$$k = 20$$



top F_0 change rate:
30 st/s
max top F_0 change rate:
38.6 st/s

✓

Summary

- In this case, the parameter setting obtained applying prior knowledge was in line with the one obtained using GCV
- In general, there is no automatic 'infallible' method to select smoothness parameters

Smoothing One F_0 Contour

```
# curve index (from 1 to 87)
i = 1
# B-splines basis time interval
range = c(0,duration[i])
# number of knots
k = 20
# evenly spaced knots
knots = seq(0,duration[i],length.out = k)
# roughness is measured as:  ∫ [ D² f(t) ]² dt
Lfdobj = 2
# order of polynomials used in B-splines
norder = 2 + Lfdobj
# a fixed relation that holds for B-splines
nbasis = k + norder - 2
# create basis object
basis = create.bspline.basis(range, nbasis, norder, knots)
```

Smoothing One F_0 Contour

```
# curve index (from 1 to 87)
i = 1
# B-splines basis time interval
range = c(0,duration[i])
# number of knots
k = 20
# evenly spaced knots
knots = seq(0,duration[i],length.out = k)
# roughness is measured as:  ∫ [ D² f(t) ]² dt
Lfdobj = 2
# order of polynomials used in B-splines
norder = 2 + Lfdobj
# a fixed relation that holds for B-splines
nbasis = k + norder - 2
# create basis object
basis = create.bspline.basis(range, nbasis, norder, knots)
```

Smoothing One F_0 Contour

```
# curve index (from 1 to 87)
i = 1
# B-splines basis time interval
range = c(0,duration[i])
# number of knots
k = 20
# evenly spaced knots
knots = seq(0,duration[i],length.out = k)
# roughness is measured as:  ∫ [ D^2 f(t) ]^2 dt
Lfdobj = 2
# order of polynomials used in B-splines
norder = 2 + Lfdobj
# a fixed relation that holds for B-splines
nbasis = k + norder - 2
# create basis object
basis = create.bspline.basis(range, nbasis, norder, knots)
```

Smoothing One F_0 Contour

```
# curve index (from 1 to 87)
i = 1
# B-splines basis time interval
range = c(0,duration[i])
# number of knots
k = 20
# evenly spaced knots
knots = seq(0,duration[i],length.out = k)
# roughness is measured as:  ∫ [ D² f(t) ]² dt
Lfdobj = 2
# order of polynomials used in B-splines
norder = 2 + Lfdobj
# a fixed relation that holds for B-splines
nbasis = k + norder - 2
# create basis object
basis = create.bspline.basis(range, nbasis, norder, knots)
```

Smoothing One F_0 Contour

```
# curve index (from 1 to 87)
i = 1
# B-splines basis time interval
range = c(0,duration[i])
# number of knots
k = 20
# evenly spaced knots
knots = seq(0,duration[i],length.out = k)
# roughness is measured as:  ∫ [ D^2 f(t) ]^2 dt
Lfdobj = 2
# order of polynomials used in B-splines
norder = 2 + Lfdobj
# a fixed relation that holds for B-splines
nbasis = k + norder - 2
# create basis object
basis = create.bspline.basis(range, nbasis, norder, knots)
```

Smoothing One F_0 Contour

```
# curve index (from 1 to 87)
i = 1
# B-splines basis time interval
range = c(0,duration[i])
# number of knots
k = 20
# evenly spaced knots
knots = seq(0,duration[i],length.out = k)
# roughness is measured as:
Lfdobj = 2
# order of polynomials used in B-splines
norder = 2 + Lfdobj
# a fixed relation that holds for B-splines
nbasis = k + norder - 2
# create basis object
basis = create.bspline.basis(range, nbasis, norder, knots)
```

$$\int \left[D^2 \; f(t) \right]^2 \; dt$$

Smoothing One F_0 Contour

```
# curve index (from 1 to 87)
i = 1
# B-splines basis time interval
range = c(0,duration[i])
# number of knots
k = 20
# evenly spaced knots
knots = seq(0,duration[i],length.out = k)
# roughness is measured as:  ∫ [ D² f(t) ]² dt
Lfdobj = 2
# order of polynomials used in B-splines
norder = 2 + Lfdobj
# a fixed relation that holds for B-splines
nbasis = k + norder - 2
# create basis object
basis = create.bspline.basis(range, nbasis, norder, knots)
```

Smoothing One F_0 Contour

```
# curve index (from 1 to 87)
i = 1
# B-splines basis time interval
range = c(0,duration[i])
# number of knots
k = 20
# evenly spaced knots
knots = seq(0,duration[i],length.out = k)
# roughness is measured as:  ∫ [ D^2 f(t) ]^2 dt
Lfdobj = 2
# order of polynomials used in B-splines
norder = 2 + Lfdobj
# a fixed relation that holds for B-splines
nbasis = k + norder - 2
# create basis object
basis = create.bspline.basis(range, nbasis, norder, knots)
```

Smoothing One F_0 Contour

```
# smoothing parameter  λ
lambda = 10^(-8)
# all information about smoothing with roughness penalty
# wrapped in one object
fdPar = fdPar(basis, Lfdobj, lambda)
# carry out smoothing
y_fdSmooth = smooth.basis(time_list[[i]],norm_st_list[[i]],fdPar)
```

show lists

- the object `y_fdSmooth` contains a `fd` object and other information (e.g. GCV)
- the `fd` object contains:
  - an object describing the basis (`basisobj`)
  - a `nbasis`-by-n. contours matrix of basis weights (`coefs`)
    `coefs` (here n. contours is 1)

Smoothing One F_0 Contour

```
# smoothing parameter  λ
lambda = 10^(-8)
# all information about smoothing with roughness penalty
# wrapped in one object
fdPar = fdPar(basis, Lfdobj, lambda)
# carry out smoothing
y_fdSmooth = smooth.basis(time_list[[i]],norm_st_list[[i]],fdPar)
```

show lists

- the object `y_fdSmooth` contains a `fd` object and other information (e.g. GCV)
- the `fd` object contains:
  - an object describing the basis (`basisobj`)
  - a `nbasis`-by-n. contours matrix of basis weights (`coefs`)

example (here n. contours is 1)

Smoothing One F_0 Contour

```
# smoothing parameter  λ
lambda = 10^(-8)
# all information about smoothing with roughness penalty
# wrapped in one object
fdPar = fdPar(basis, Lfdobj, lambda)
# carry out smoothing
y_fdSmooth = smooth.basis(time_list[[i]],norm_st_list[[i]],fdPar)
```

show lists

- the object `y_fdSmooth` contains a `fd` object and other information (e.g. GCV)
- the `fd` object contains:
  - an object describing the basis (`basisobj`)
  - a `nbasis`-by-n. contours matrix of basis weights (`coefs`) (here n. contours is 1)

Smoothing One F_0 Contour

```
# smoothing parameter  λ
lambda = 10^(-8)
# all information about smoothing with roughness penalty
# wrapped in one object
fdPar = fdPar(basis, Lfdobj, lambda)
# carry out smoothing
y_fdSmooth = smooth.basis(time_list[[i]],norm_st_list[[i]],fdPar)
```

show lists

- the object `y_fdSmooth` contains a `fd` object and other information (e.g. GCV)
- the `fd` object contains:
  - an object describing the basis (`basisobj`)
  - a `nbasis`-by-n. contours matrix of basis weights (`coefs`) (here n. contours is 1)

Smoothing One F_0 Contour

```
# smoothing parameter  λ
lambda = 10^(-8)
# all information about smoothing with roughness penalty
# wrapped in one object
fdPar = fdPar(basis, Lfdobj, lambda)
# carry out smoothing
y_fdSmooth = smooth.basis(time_list[[i]],norm_st_list[[i]],fdPar)
```

show lists

- the object `y_fdSmooth` contains a `fd` object and other information (e.g. GCV)
- the `fd` object contains:
  - an object describing the basis (`basisobj`)
  - a `nbasis`-by-n. contours matrix of basis weights (`coefs`) example (here n. contours is 1)

Smoothing All F_0 Contours

```
# a common time interval for all contours
range = c(0,mean(duration))
k = 20; lambda = 10^(-8)
knots = seq(0,mean(duration),length.out = k)
Lfdobj = 2
norder = 2 + Lfdobj
nbasis = k + norder - 2
basis = create.bspline.basis(range, nbasis, norder, knots)
fdPar = fdPar(basis, Lfdobj, lambda)
# a nbaasis-by-n_contours matrix will store the basis coefs
coef = matrix(nrow = nbasis, ncol = n_contours)
for (i in 1:n_contours) {
# normalise time
t_norm = ( time_list[[i]] / duration[i] ) * mean(duration)
coef[,i] = c(smooth.basis(t_norm,norm_st_list[[i]],fdPar)$fd$coefs)
}
# all contours in a common fd object (same basis)
norm_st_fd = fd(coef=coef,basisobj=basis)
```

Obtaining GCV errors

```
# define your (k,lambda) grid
k_vec = seq(10,60,10)
lambda_vec = 10^(seq(-12 , -4, 1))
# use a subset of your data, to save computation time
i_sample = sample(1:n_contours,20)
# smooth the contours for each grid value
for (kind in 1:length(k_vec)) {
for (lind in 1:length(lambda_vec)) {
k = k_vec[kind]; lambda = lambda_vec[lind]
for (i in i_sample) {
# ... same smoothing code as before ...
# store gcv error for each contour i in i_sample
gcv_err[i,lind,kind] =
smooth.basis(t_norm,norm_st_list[[i]],fdPar)$gcv
}}}
# average gcv values for each grid point over all contours
GCV_log_err = log( apply(gcv_err,2:3,mean,trim=0.05))
```

Obtaining GCV errors

```
# define your (k,lambda) grid
k_vec = seq(10,60,10)
lambda_vec = 10^(seq(-12 , -4, 1))
# use a subset of your data, to save computation time
i_sample = sample(1:n_contours,20)
# smooth the contours for each grid value
for (kind in 1:length(k_vec)) {
for (lind in 1:length(lambda_vec)) {
k = k_vec[kind]; lambda = lambda_vec[lind]
for (i in i_sample) {
# ... same smoothing code as before ...
# store gcv error for each contour i in i_sample
gcv_err[i,lind,kind] =
smooth.basis(t_norm,norm_st_list[[i]],fdPar)$gcv
}}}
# average gcv values for each grid point over all contours
GCV_log_err = log( apply(gcv_err,2:3,mean,trim=0.05))
```

Obtaining GCV errors

```
# define your (k,lambda) grid
k_vec = seq(10,60,10)
lambda_vec = 10^(seq(-12 , -4, 1))
# use a subset of your data, to save computation time
i_sample = sample(1:n_contours,20)
# smooth the contours for each grid value
for (kind in 1:length(k_vec)) {
for (lind in 1:length(lambda_vec)) {
k = k_vec[kind]; lambda = lambda_vec[lind]
for (i in i_sample) {
# ... same smoothing code as before ...
# store gcv error for each contour i in i_sample
gcv_err[i,lind,kind] =
smooth.basis(t_norm,norm_st_list[[i]],fdPar)$gcv
}}}
# average gcv values for each grid point over all contours
GCV_log_err = log( apply(gcv_err,2:3,mean,trim=0.05))
```

Obtaining GCV errors

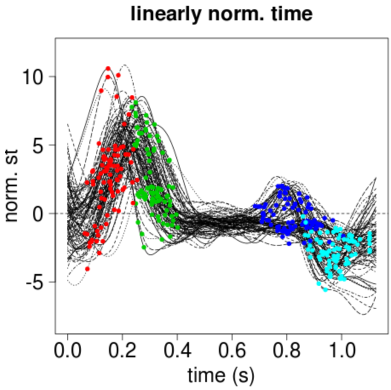
```
# define your (k,lambda) grid
k_vec = seq(10,60,10)
lambda_vec = 10^(seq(-12 , -4, 1))
# use a subset of your data, to save computation time
i_sample = sample(1:n_contours,20)
# smooth the contours for each grid value
for (kind in 1:length(k_vec)) {
for (lind in 1:length(lambda_vec)) {
k = k_vec[kind]; lambda = lambda_vec[lind]
for (i in i_sample) {
# ... same smoothing code as before ...
# store gcv error for each contour i in i_sample
gcv_err[i,lind,kind] =
smooth.basis(t_norm,norm_st_list[[i]],fdPar)$gcv
}}}
# average gcv values for each grid point over all contours
GCV_log_err = log( apply(gcv_err,2:3,mean,trim=0.05))
```

Obtaining GCV errors

```
# define your (k,lambda) grid
k_vec = seq(10,60,10)
lambda_vec = 10^(seq(-12 , -4, 1))
# use a subset of your data, to save computation time
i_sample = sample(1:n_contours,20)
# smooth the contours for each grid value
for (kind in 1:length(k_vec)) {
for (lind in 1:length(lambda_vec)) {
k = k_vec[kind]; lambda = lambda_vec[lind]
for (i in i_sample) {
# ... same smoothing code as before ...
# store gcv error for each contour i in i_sample
gcv_err[i,lind,kind] =
smooth.basis(t_norm,norm_st_list[[i]],fdPar)$gcv
}}}
# average gcv values for each grid point over all contours
GCV_log_err = log( apply(gcv_err,2:3,mean,trim=0.05))
```

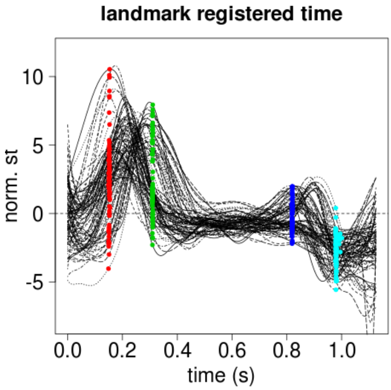
▸ GCV_log_err

Landmark Registration



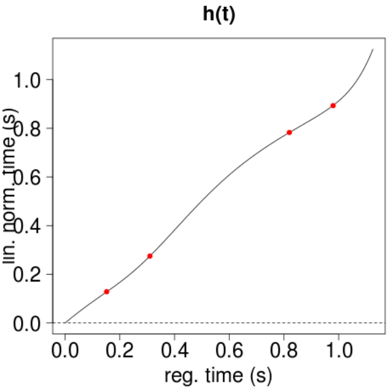
- Events in F_0 are synchronised with the segmental material and not with (absolute) time
- But FDA internal operations use the time axis as reference for comparing contours

Landmark Registration



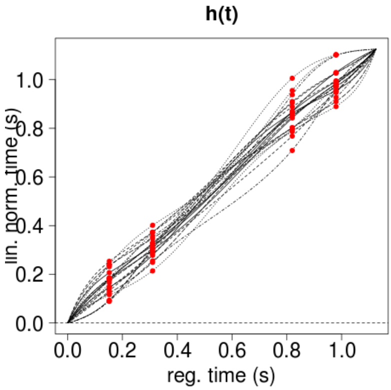
- Landmark Registration allows to warp the time axis of each contour on user-defined synchronization events
- For prosody, those events can be phone/syllable/word boundaries

Landmark Registration as Smoothing



- Landmark registration is also carried out by smoothing with roughness penalty
- the function $h(t)$ is smoothed around the position of landmarks in the (reg. time, lin. norm. time) plain

Landmark Registration as Smoothing



- Landmark registration is also carried out by smoothing with roughness penalty
- the function $h(t)$ is smoothed around the position of landmarks in the (reg. time, lin. norm. time) plain

Landmark Registration - Code

```
# put landmarks in a matrix and normalize to mean(duration)
land = matrix(nrow = n_contours, ncol = 4)
for (i in 1:n_contours) {
land[i,1] = metadata$v1_beg[i]/duration[i]
land[i,2] = metadata$v1_end[i]/duration[i]
land[i,3] = metadata$v2_beg[i]/duration[i]
land[i,4] = metadata$v2_end[i]/duration[i]
}
land = land * mean_dur
# position of the aligned landmarks
land_mean = colMeans(land,na.rm =T)
```

Landmark Registration - Code

```
# put landmarks in a matrix and normalize to mean(duration)
land = matrix(nrow = n_contours, ncol = 4)
for (i in 1:n_contours) {
land[i,1] = metadata$v1_beg[i]/duration[i]
land[i,2] = metadata$v1_end[i]/duration[i]
land[i,3] = metadata$v2_beg[i]/duration[i]
land[i,4] = metadata$v2_end[i]/duration[i]
}
land = land * mean_dur
# position of the aligned landmarks
land_mean = colMeans(land,na.rm =T)
```

Landmark Registration - Code

```
# put a knot on every landmark
knots = c(0,land_mean,mean_dur)
# smoothing procedure
norder = 4
nbasis = length(knots) + norder - 2
basis = create.bspline.basis(c(0,mean_dur),nbasis,norder,knots)
# start by using the same lambda used for contour smoothing
# then correct by visual inspection (see pics. later)
lambda = 1e-8
fdPar = fdPar(basis,2,lambda)
# the registration command (can be quite slow)
warp_y_fd = landmarkreg(norm_st_fd,land,land_mean,fdPar, monwrd=T)
```

Landmark Registration - Code

```
# put a knot on every landmark
knots = c(0,land_mean,mean_dur)
# smoothing procedure
norder = 4
nbasis = length(knots) + norder - 2
basis = create.bspline.basis(c(0,mean_dur),nbasis,norder,knots)
# start by using the same lambda used for contour smoothing
# then correct by visual inspection (see pics. later)
lambda = 1e-8
fdPar = fdPar(basis,2,lambda)
# the registration command (can be quite slow)
warp_y_fd = landmarkreg(norm_st_fd,land,land_mean,fdPar, monwrd=T)
```

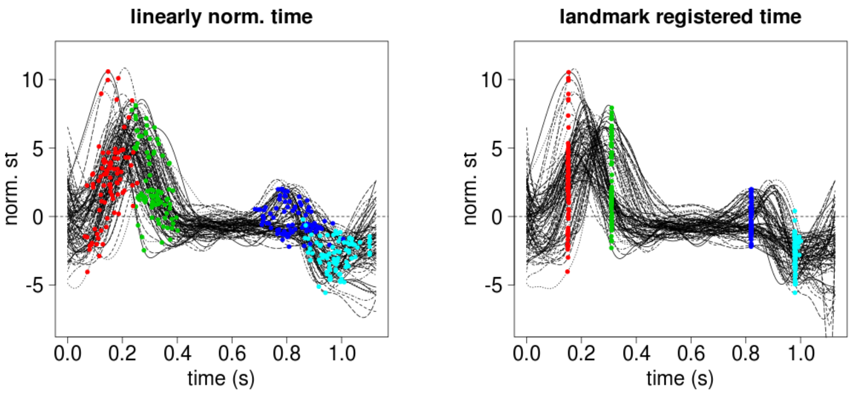
Landmark Registration - Code

```
# put a knot on every landmark
knots = c(0,land_mean,mean_dur)
# smoothing procedure
norder = 4
nbasis = length(knots) + norder - 2
basis = create.bspline.basis(c(0,mean_dur),nbasis,norder,knots)
# start by using the same lambda used for contour smoothing
# then correct by visual inspection (see pics. later)
lambda = 1e-8
fdPar = fdPar(basis,2,lambda)
# the registration command (can be quite slow)
warp_y_fd = landmarkreg(norm_st_fd,land,land_mean,fdPar, monwrd=T)
```

Landmark Registration - Code

```
# put a knot on every landmark
knots = c(0,land_mean,mean_dur)
# smoothing procedure
norder = 4
nbasis = length(knots) + norder - 2
basis = create.bspline.basis(c(0,mean_dur),nbasis,norder,knots)
# start by using the same lambda used for contour smoothing
# then correct by visual inspection (see pics. later)
lambda = 1e-8
fdPar = fdPar(basis,2,lambda)
# the registration command (can be quite slow)
warp_y_fd = landmarkreg(norm_st_fd,land,land_mean,fdPar, monwrd=T)
```

Inspecting Registration Result



Ordinary PCA

- In ordinary PCA the input data are N fixed size column vectors v_n, $n = 1, \dots, N$
- The first principal component is the vector P_1 of norm one that produces the largest possible variance of the inner product $P_1 \cdot v_n$ across the N vectors v_n
- The k-th principal component is the vector P_k is obtained in the same way, but with the further constraint of being orthogonal to the previous principal components P_1 to P_{k-1}
- In ordinary PCA the inner product $P_k \cdot v_n$ is the classic row by column product

$$P_k \cdot v_n = P_k^T v_n$$

Functional PCA

- In Functional PCA the input data are N functions $f_n(t)$ defined on the same interval $[0, T]$
- The inner product $P_k \cdot f_n$ is redefined as the following integral:

$$P_k \cdot f_n \triangleq \int_0^T P_k(t) f_n(t) dt$$

- Note that the definition is independent of the basis used to represent $f_n(t)$
- Moreover, it compares contours $f_n(t)$ synchronously on t

Functional PCA

- In Functional PCA the input data are N functions $f_n(t)$ defined on the same interval $[0, T]$
- The inner product $P_k \cdot f_n$ is redefined as the following integral:

$$P_k \cdot f_n \triangleq \int_0^T P_k(t) f_n(t) dt$$

- Note that the definition is independent of the basis used to represent $f_n(t)$
- Moreover, it compares contours $f_n(t)$ synchronously on t

Functional PCA

- In Functional PCA the input data are N functions $f_n(t)$ defined on the same interval $[0, T]$
- The inner product $P_k \cdot f_n$ is redefined as the following integral:

$$P_k \cdot f_n \triangleq \int_0^T P_k(t) f_n(t) dt$$

- Note that the definition is independent of the basis used to represent $f_n(t)$
- Moreover, it compares contours $f_n(t)$ synchronously on t

Finding PC with Smoothing

- Choose a basis for $P_k(t)$
- For each component k, find function $P_k(t)$ that maximises:

$$\mathrm{var}_n \left\{ \int_0^T P_k(t) f_n(t) dt \right\}$$

- Apply smoothing with roughness penalty, i.e. maximise but 'not too much'

Functional PCA - Code

```
# assign k and lambda the same values used to smooth f0 contours
# because PC contours have dynamic properties similar to them
k = 20; lambda = 1e-8
# the following code should look familiar now ...
range = c(0,mean(duration))
knots = seq(0,mean(duration),length.out = k)
Lfdobj = 2
norder = 2 + Lfdobj
nbasis = k + norder - 2
basis = create.bspline.basis(range, nbasis, norder, knots)
fdPar = fdPar(basis, Lfdobj, lambda)
# compute the first 3 Principal Components (aka 'harmonics')
# (y_fd contains all the 87 registered contours)
y_pcafd = pca.fd(y_fd, nharm=3, fdPar)
```

Functional PCA - Code

```
# assign k and lambda the same values used to smooth f0 contours
# because PC contours have dynamic properties similar to them
k = 20; lambda = 1e-8
# the following code should look familiar now ...
range = c(0,mean(duration))
knots = seq(0,mean(duration),length.out = k)
Lfdobj = 2
norder = 2 + Lfdobj
nbasis = k + norder - 2
basis = create.bspline.basis(range, nbasis, norder, knots)
fdPar = fdPar(basis, Lfdobj, lambda)
# compute the first 3 Principal Components (aka 'harmonics')
# (y_fd contains all the 87 registered contours)
y_pcafd = pca.fd(y_fd, nharm=3, fdPar)
```

Functional PCA - Code

```
# assign k and lambda the same values used to smooth f0 contours
# because PC contours have dynamic properties similar to them
k = 20; lambda = 1e-8
# the following code should look familiar now ...
range = c(0,mean(duration))
knots = seq(0,mean(duration),length.out = k)
Lfdobj = 2
norder = 2 + Lfdobj
nbasis = k + norder - 2
basis = create.bspline.basis(range, nbasis, norder, knots)
fdPar = fdPar(basis, Lfdobj, lambda)
# compute the first 3 Principal Components (aka 'harmonics')
# (y_fd contains all the 87 registered contours)
y_pcafd = pca.fd(y_fd, nharm=3, fdPar)
```

Functional PCA - Code

- The `y_pcafd` object contains:
  - `meanfd`: a `fd` object giving the mean function
  - `harmonics`: a `fd` object giving the PCs
  - `scores`: a `n_contours`-by-`nharm` matrix giving PC scores
  - `varprop`: a vector giving the proportion of variance explained by each PC
- to plot PC1:

  `plot(y_pcafd$harmonics[1])`

- to get the PC1 score of the i-th f0 contour:

  `y_pcafd$scores[i,1]`

Functional PCA - Code

- The `y_pcafd` object contains:
  - `meanfd`: a `fd` object giving the mean function
  - `harmonics`: a `fd` object giving the PCs
  - `scores`: a `n_contours`-by-`nharm` matrix giving PC scores
  - `varprop`: a vector giving the proportion of variance explained by each PC
- to plot PC1:

```
plot(y_pcafd$harmonics[1])
```

- to get the PC1 score of the i-th f0 contour:

```
y_pcafd$scores[i,1]
```

Functional PCA - Code

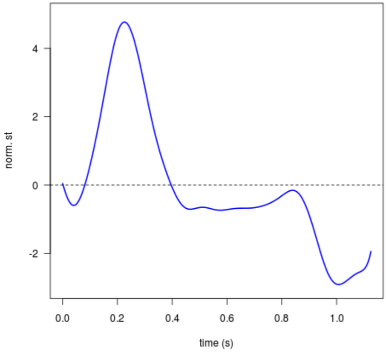
- The `y_pcafd` object contains:
  - `meanfd`: a `fd` object giving the mean function
  - `harmonics`: a `fd` object giving the PCs
  - `scores`: a `n_contours`-by-`nharm` matrix giving PC scores
  - `varprop`: a vector giving the proportion of variance explained by each PC
- to plot PC1:

```
plot(y_pcafd$harmonics[1])
```

- to get the PC1 score of the i-th f0 contour:

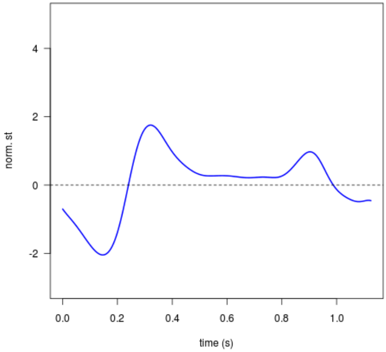
```
y_pcafd$scores[i,1]
```

The Mean Contour



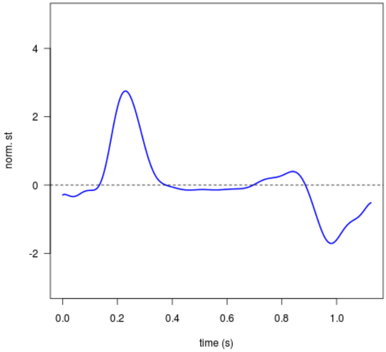

```
plot(y_pcafd$meanfd,...)
```

PC1



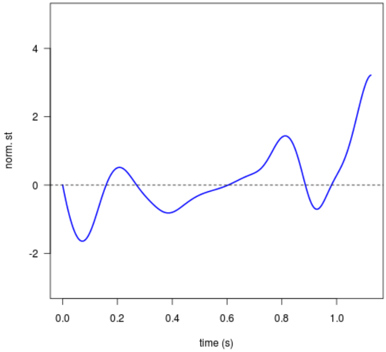

```
plot(y_pcafd$harmonics[1],...)
```

PC2



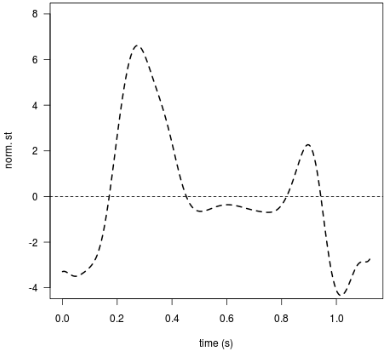

```
plot(y_pcafd$harmonics[2],...)
```

PC3



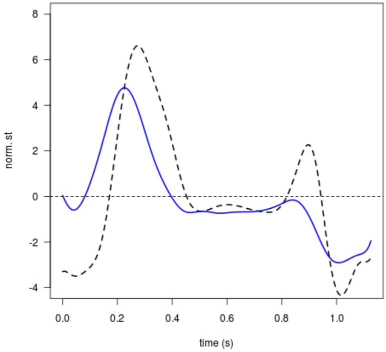

```
plot(y_pcafd$harmonics[3],...)
```

PC-Based Reconstruction



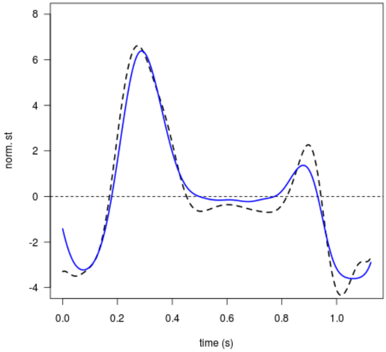

```
i = 67  # a question
plot(y_fd[i],...)
```

PC-Based Reconstruction



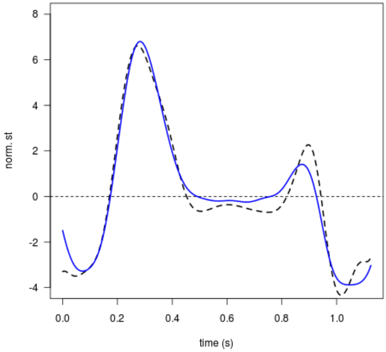

```
i = 67  # a question
plot(y_fd[i],...)

lines(y_pcafd$meanfd,...)
```

PC-Based Reconstruction



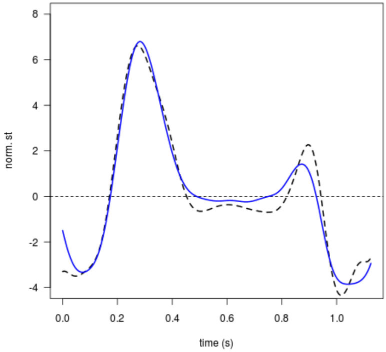

```
i = 67  # a question
plot(y_fd[i],...)

lines(y_pcafd$meanfd +
y_pcafd$scores[i,1] *
y_pcafd$harmonics[1],...)
```

PC-Based Reconstruction



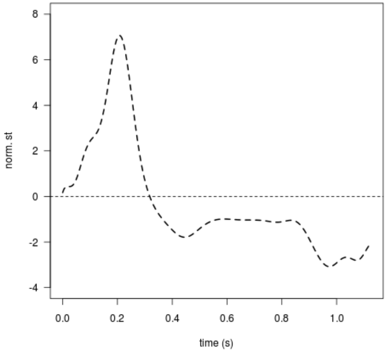

```
i = 67  # a question
plot(y_fd[i],...)

lines(y_pcafd$meanfd +
y_pcafd$scores[i,1] *
y_pcafd$harmonics[1] +
y_pcafd$scores[i,2] *
y_pcafd$harmonics[2],...)
```

PC-Based Reconstruction



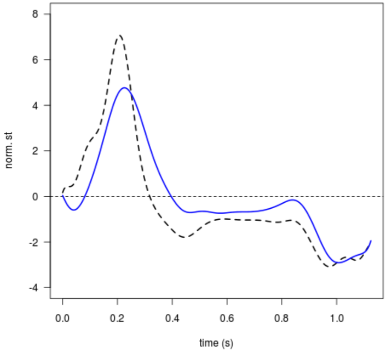

```
i = 67  # a question
plot(y_fd[i],...)

lines(y_pcafd$meanfd +
y_pcafd$scores[i,1] *
y_pcafd$harmonics[1] +
y_pcafd$scores[i,2] *
y_pcafd$harmonics[2] +
y_pcafd$scores[i,3] *
y_pcafd$harmonics[3],...)
```

PC-Based Reconstruction



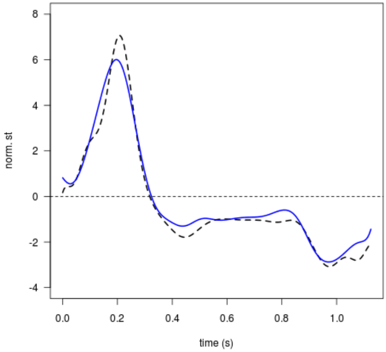

```
i = 78   # a statement
plot(y_fd[i],...)
```

PC-Based Reconstruction



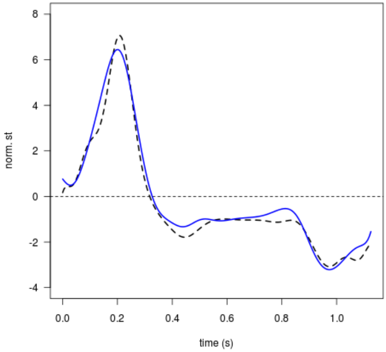

```
i = 78  # a statement
plot(y_fd[i],...)

lines(y_pcafd$meanfd,...)
```

PC-Based Reconstruction



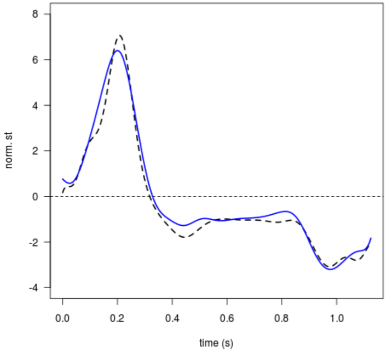

```
i = 78  # a statement
plot(y_fd[i],...)

lines(y_pcafd$meanfd +
y_pcafd$scores[i,1] *
y_pcafd$harmonics[1],...)
```

PC-Based Reconstruction



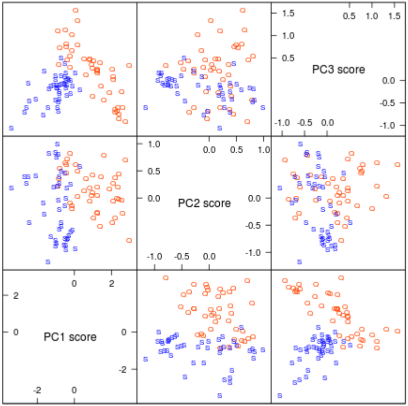

```
i = 78   # a statement
plot(y_fd[i],...)

lines(y_pcafd$meanfd +
y_pcafd$scores[i,1] *
y_pcafd$harmonics[1] +
y_pcafd$scores[i,2] *
y_pcafd$harmonics[2],...)
```

PC-Based Reconstruction



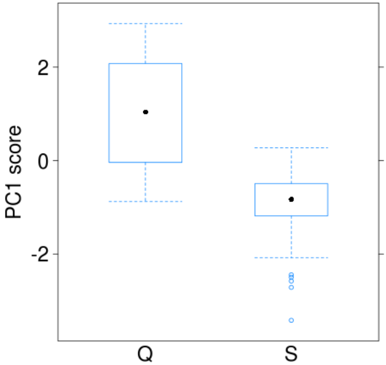

```
i = 78  # a statement
plot(y_fd[i],...)

lines(y_pcafd$meanfd +
y_pcafd$scores[i,1] *
y_pcafd$harmonics[1] +
y_pcafd$scores[i,2] *
y_pcafd$harmonics[2] +
y_pcafd$scores[i,3] *
y_pcafd$harmonics[3],...)
```

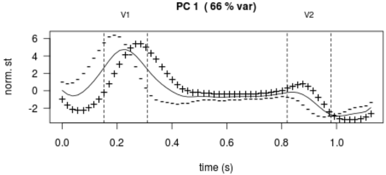
Results



Scatter Plot Matrix

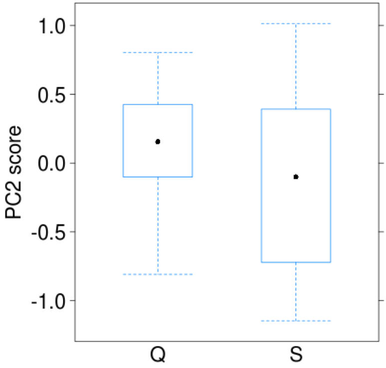
Results - PC1

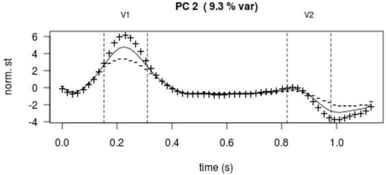




without land. reg. PCA on splines coefs.

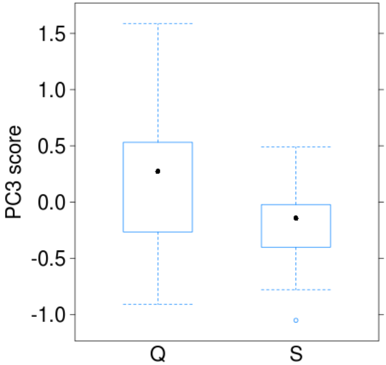
Results - PC2

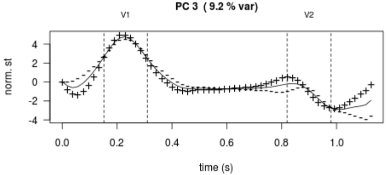




▸ without land. reg. ▸ PCA on splines coefs.

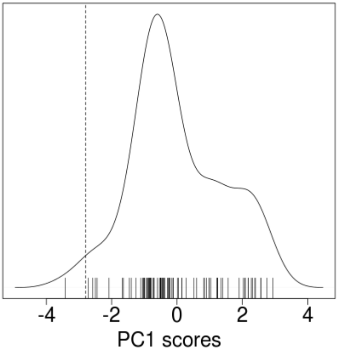
Results - PC3

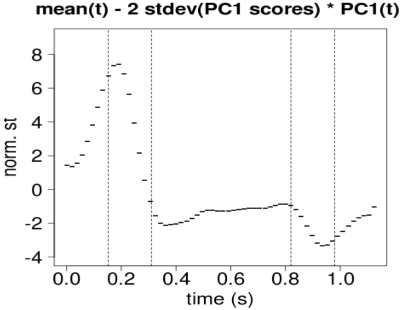




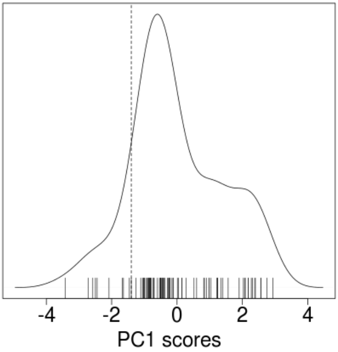
▸ without land. reg. ▸ PCA on splines coefs.

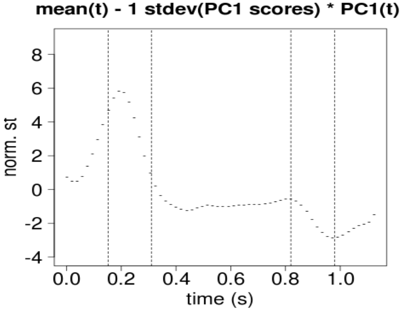
PC1-Based Reconstruction



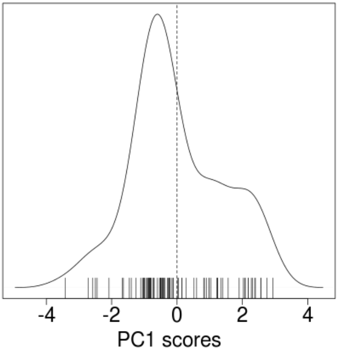


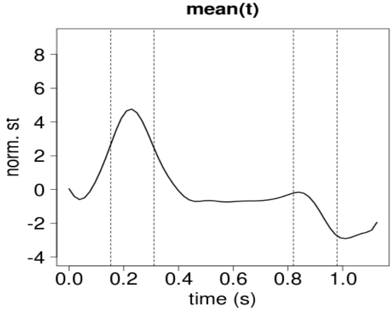
PC1-Based Reconstruction



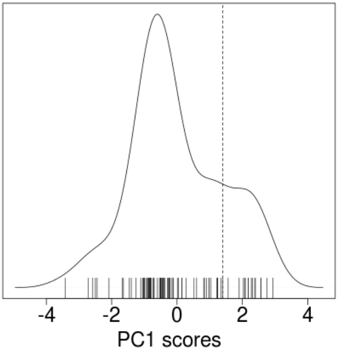


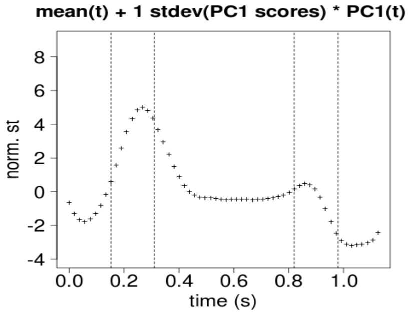
PC1-Based Reconstruction



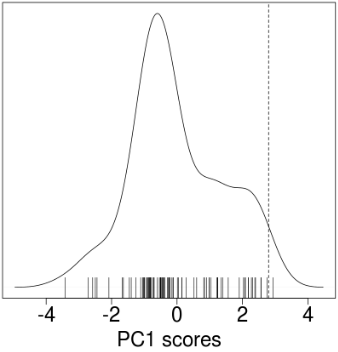


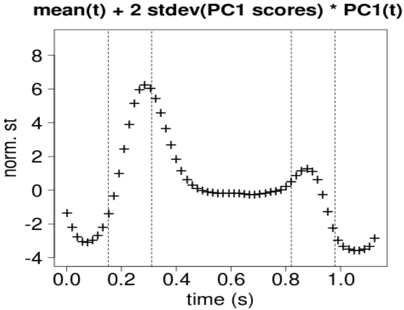
PC1-Based Reconstruction



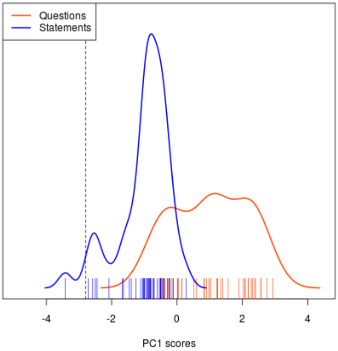


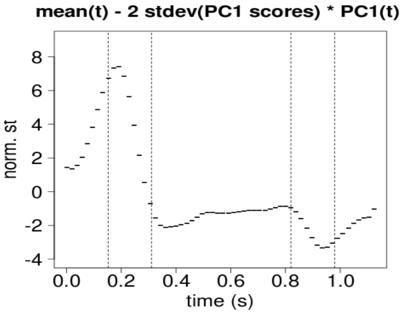
PC1-Based Reconstruction



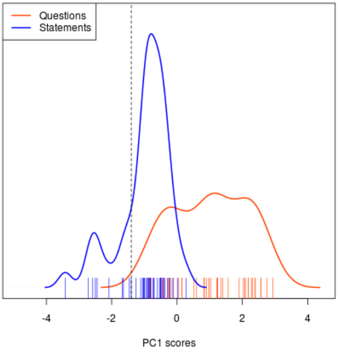


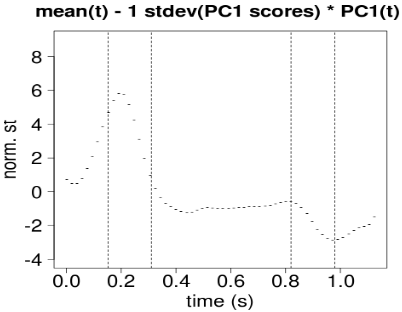
PC1-Based Reconstruction



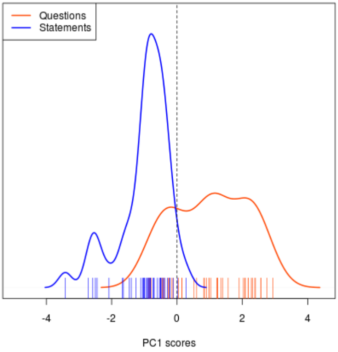


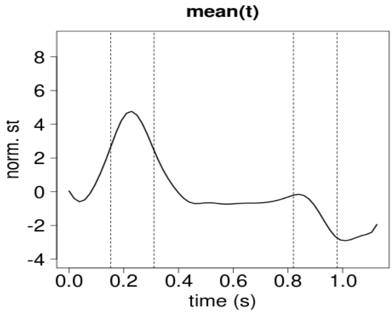
PC1-Based Reconstruction



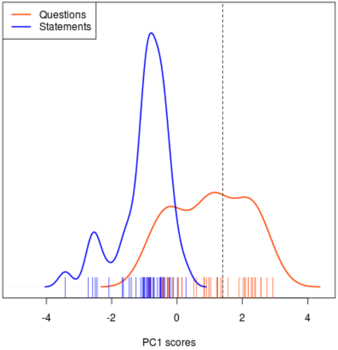


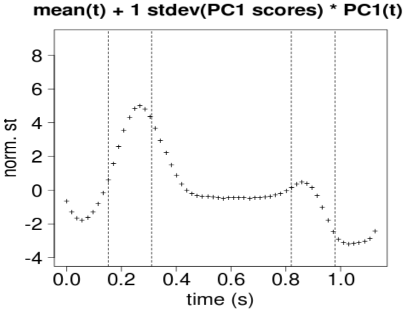
PC1-Based Reconstruction



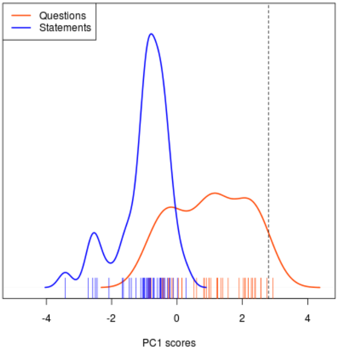


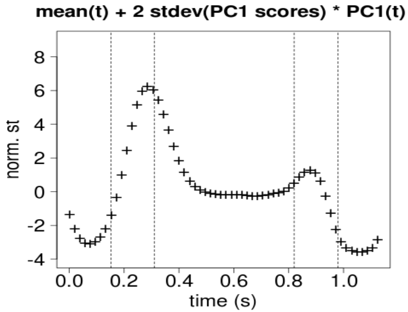
PC1-Based Reconstruction





PC1-Based Reconstruction

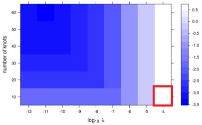




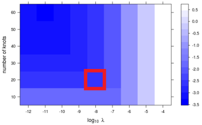
'Wrong' Smoothing Parameters

- How much does a poor choice of smoothing parameters affects functional PCA results?
  - case of underfitting example
  - case of overfitting example

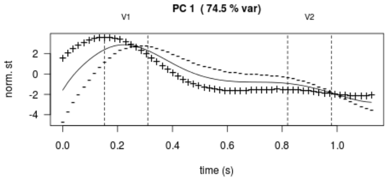
Underfitting Smoothing Parameters - PC1

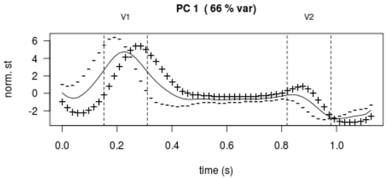


$$\lambda = 10^{-4}$$
$$k = 10$$

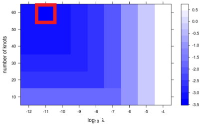


$$\lambda = 10^{-8}$$
$$k = 20$$

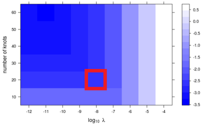




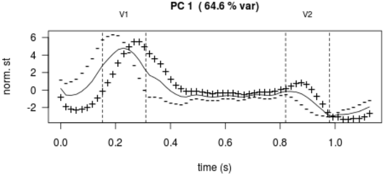
Overfitting Smoothing Parameters - PC1

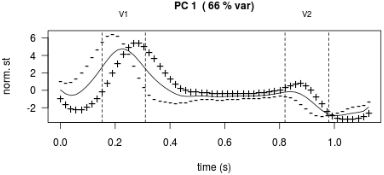


$$\lambda = 10^{-11}$$
$$k = 60$$



$$\lambda = 10^{-8}$$
$$k = 20$$





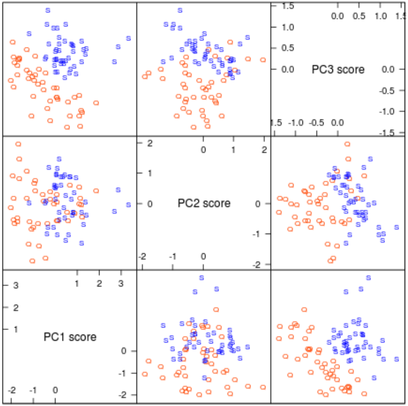
Summary

- Underfitting blurs too much detail with no recovery
- Overfitting brings decent PC curves in our case (just a bit 'shaky')
- Overfitting may become problematic when contours are more noisy
- Overfitting is always a waste of computational power

Skipping Landmark Registration

- How crucial is landmark registration?
- Let us repeat the whole analysis, just skipping registration

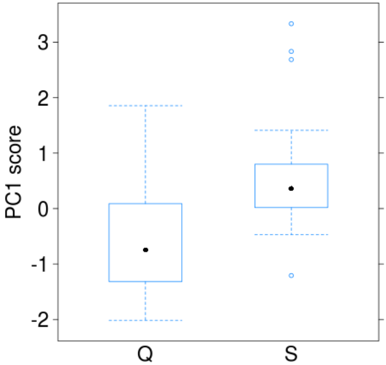
Results Without Landmark Registration

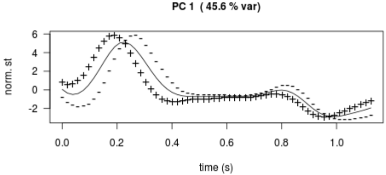


Scatter Plot Matrix

with land. reg.

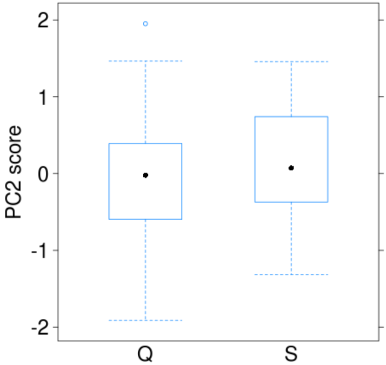
Results Without Landmark Registration - PC1

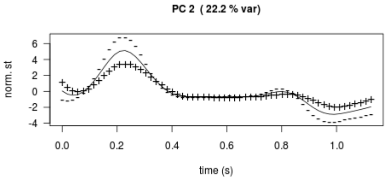




with land. reg.

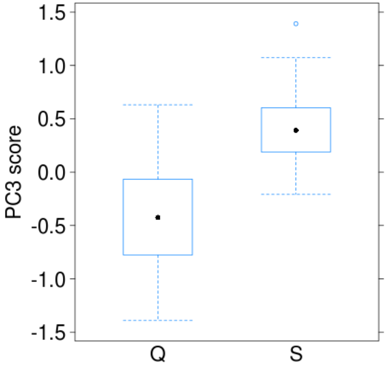
Results Without Landmark Registration - PC2

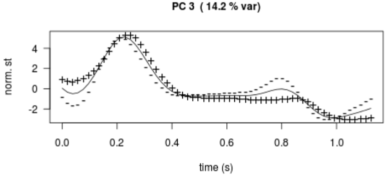




with land. reg.

Results Without Landmark Registration - PC3





with land. reg.

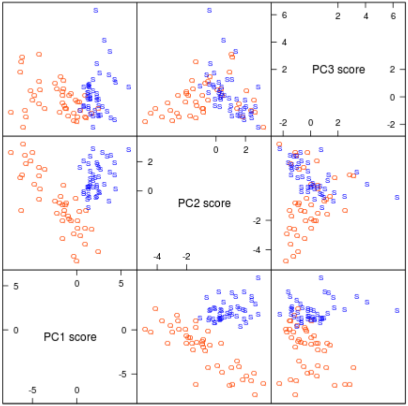
Summary

- PC contours don't change much with or without landmark registration
- PC scores correlation with linguistic variables can change a lot
  - systematic shifts in time are blurred by random shift of landmarks across contours
  - systematic amplitude variations are preserved

Ordinary PCA on B-Splines Coefficients

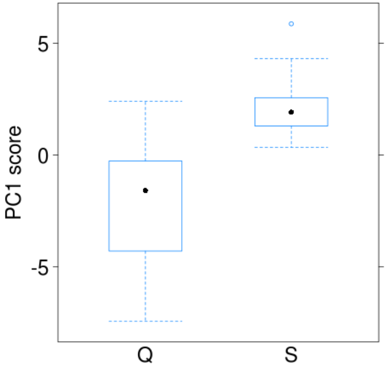
- Repeat smoothing and landmark registration as before
- instead of applying functional PCA, apply ordinary PCA to the (23-dim) B-splines coefficient vectors

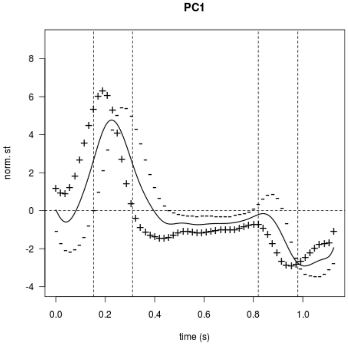
Results PCA on B-Splines Coefficients



Scatter Plot Matrix

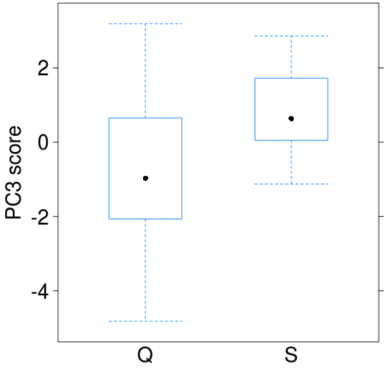
Results PCA on B-Splines Coefficients - PC1

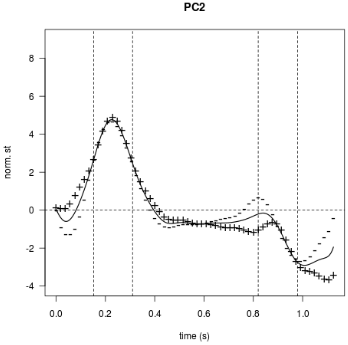




functional PCA

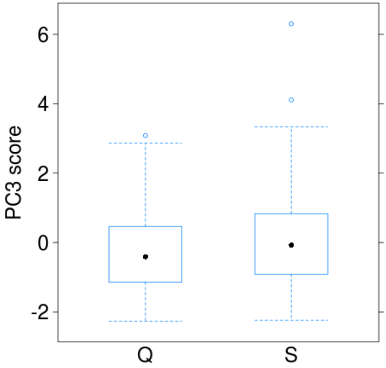
Results PCA on B-Splines Coefficients - PC2

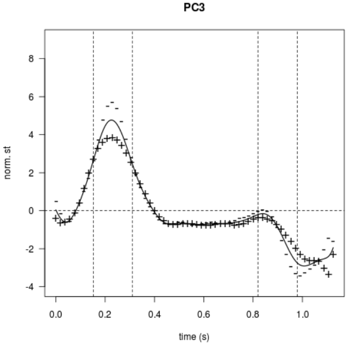




functional PCA

Results PCA on B-Splines Coefficients - PC3

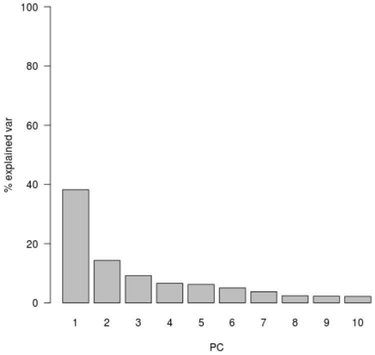




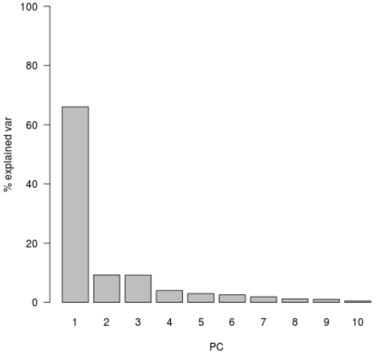
functional PCA

Percentage of Explained Variance

PCA on B-splines coefficients



functional PCA



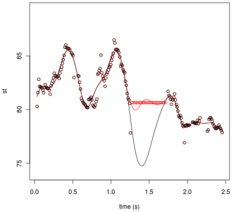
Summary

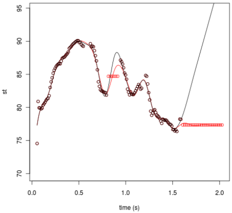
- Functional PCA can be substituted for ordinary PCA on your basis coefficients
- Results differ
- Functional PCA performs better in our example
- Functional PCA is inherently independent of the basis, while ordinary PCA operates directly on the basis coefficients
- Note that the two PCA concide only if the basis is orthogonal (but B-splines are not)

Voiceless sounds and f_0 tracker errors

- Speech material contains voiceless sounds, pauses, creacky voice, etc.
- f_0 trackers commit errors, like octave jumps
- How to handle all this in a FDA session?

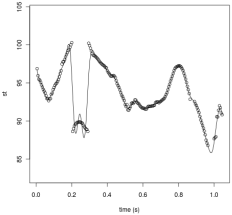
Voiceless sounds

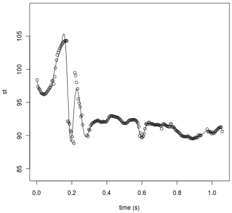




- Gaps in the f_0 sample sequence leave smoothing unconstrained
- Simple solution: padding gaps with extra samples (e.g. value = mean of surrounding samples)

Octave jumps





- An octave jump spanning more than a few samples has no recovery
- Smoothing would produce a highly disrupted curve, which would perturbate the whole analysis

Final remarks

- Isolated spurious samples have little impact on smoothing
- The use of a median filter can give some benefit
- Absence of signal (like in voiceless sounds) can be takled by sample padding
- Octave jumps have no recovery
- Creacky voice behaves somehow in between silence and octave jumps