Fundamentals and applications of resampling methods for the analysis of speech production and perception data.

Olivier Crouzet

[1] Laboratoire de Linguistique de Nantes (LLING – UMR 6310, Université de Nantes / CNRS)
[2] University Medical Center Groningen (UMCG, ENT department, Reijksuniversiteit Groningen).

Workshop on "Statistical Methods in Phonetic Sciences", University of Cologne, June 11th 2017.

Talk outline

Talk outline

Talk outline

Talk outline

Aims of statistical analyses

- Estimating properties of a population or evaluating hypotheses on a population...
- ...from the observation of a random sample;

Specific applications

- Estimating a statistical parameter (central tendency, dispersion, correlation...) and computing associated confidence intervals...
- Hypothesis testing (comparing means...);

Approaches

- Asymptotic results (traditional inference approach);
- Resampling methods

  Bootstrap — Parameter estimation;
  Permutation tests — Hypothesis testing;
  Outlier detection Data cleanup (though one should consider the implications "definitely removing"obvervations from the data, computing confidence intervals may often be sufficient);
- Bayesian framework (not included in this presentation);

Asymptotic results

- Assumptions about the underlying distribution;
- A mathematical model of the underlying distribution is refered to;
- The sample is viewed as a random exemplar that is drawn from the underlying population;
- Computing a Confidence Interval requires a specific mathematical formula for each parameter (mean, median...);

Resampling approaches

- No assumptions about the underlying distribution;
- The mathematical model of the underlying distribution is replaced with a computational simulated estimation of the "population" by generating "bootstrap samples";
- The (original) sample is the source of this computating simulation;
- Computing a Confidence Interval is possible for any parameter without requiring specific formulas;

A Gaussian distributed variable

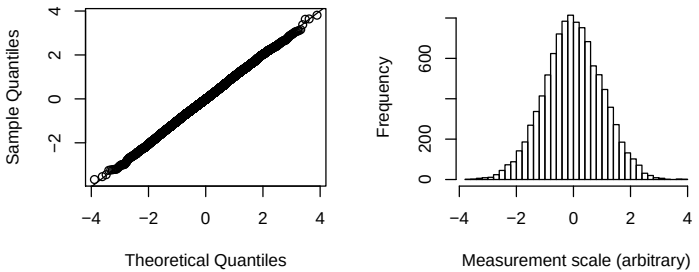


Figure 1: An illustration of a Gaussian distribution from which data may be randomly sampled. The QQ-plot on the left shows compatibility with the Gaussian assumption.

Confidence Intervals
Recalls

- We talk about 95%, 99%... Confidence Intervals (CIs);
- These mean that, in the long run, 95% (resp. 99%) of the computed CIs would contain the true value for the measured parameter;

Asymptotic framework
Estimating 95% CI for the mean

Estimating a 95% CI for a parameter's mean is done with the following formula:

$$\delta = \frac{1.96 * \text{SD}}{\sqrt{n}} \tag{1}$$

$$\text{CI} = \text{mean} \pm \delta \tag{2}$$

- Gaussian assumption: the formula is valid for a normally distributed variable;
- 95% of the area under a normal curve lies within the mean $\pm 1.96 \times$ sd;
- 99% of the area under a normal curve lies within the mean $\pm 2.58 \times$ sd.

Conventional CI for the mean

Function definition in R

```r
CI <- function(vector, targetprob = 0.95) {
    # CI for the mean

    # Compute the required percentile point from the target probability
    param <- qnorm(1 - ((1 - targetprob) / 2))

    # Estimate the delta
    delta <- ((param * sd(vector)) / (sqrt(length(vector))))

    # Generate the CI values
    ci <- c(mean(vector) - delta, mean(vector) + delta)

    # Give a name to the resulting vector values
    names(ci) <- as.character(
        c(
            paste0((1-targetprob)/2*100,"%"),
            paste0((1-(1-targetprob)/2)*100,"%")
        )
    )

    return(ci)
}
```

Conventional CI for the mean
Function application

```
samplesize <- 10
set.seed(1)
vecn <- rnorm(samplesize, mean = 0);
vecn

 [1] -0.626  0.184 -0.836  1.595  0.330 -0.820  0.487  0.738  0.576 -0.305

CI(vecn)

  2.5%  97.5%
-0.352  0.616

CI(vecn, targetprob = .99)

  0.5%  99.5%
-0.504  0.768
```

```
par(mfrow=c(1,1), cex=0.85)
hist(vecn, breaks=40, main = "", xlab = "Measurement scale (arbitrary)")
abline(v = CI(vecn), col = "red")
```

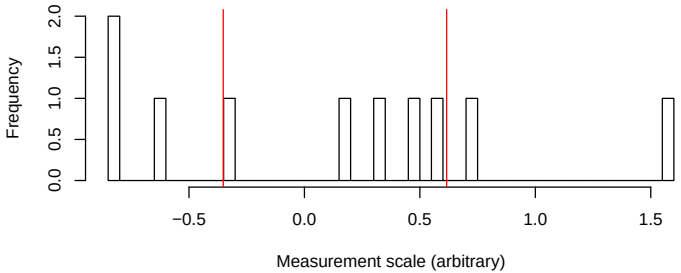


Figure 2: Conventional Confidence Interval for the mean.

Conventional CI for the mean

Function application

```
samplesize <- 50
set.seed(1)
vecn <- rnorm(samplesize, mean = 0);
vecn
```

```
 [1] -0.6265  0.1836 -0.8356  1.5953  0.3295 -0.8205  0.4874  0.7383
 [9]  0.5758 -0.3054  1.5118  0.3898 -0.6212 -2.2147  1.1249 -0.0449
[17] -0.0162  0.9438  0.8212  0.5939  0.9190  0.7821  0.0746 -1.9894
[25]  0.6198 -0.0561 -0.1558 -1.4708 -0.4782  0.4179  1.3587 -0.1028
[33]  0.3877 -0.0538 -1.3771 -0.4150 -0.3943 -0.0593  1.1000  0.7632
[41] -0.1645 -0.2534  0.6970  0.5567 -0.6888 -0.7075  0.3646  0.7685
[49] -0.1123  0.8811
```

```
CI(vecn)
```

```
  2.5%  97.5%
-0.130  0.331
```

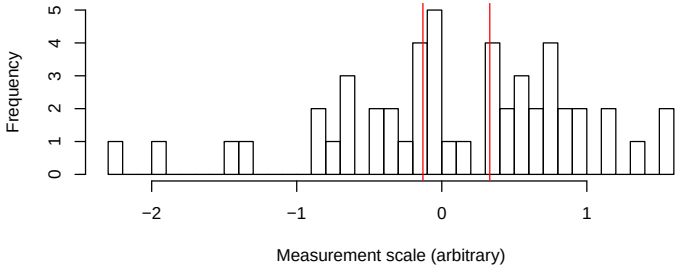


Figure 3: Conventional Confidence Interval for the mean.

Conventional CI for the mean
Function application

```
samplesize <- 1000
set.seed(1)
vecn <- rnorm(samplesize, mean = 0);
CI(vecn)

   2.5%   97.5%
-0.0758  0.0525
```

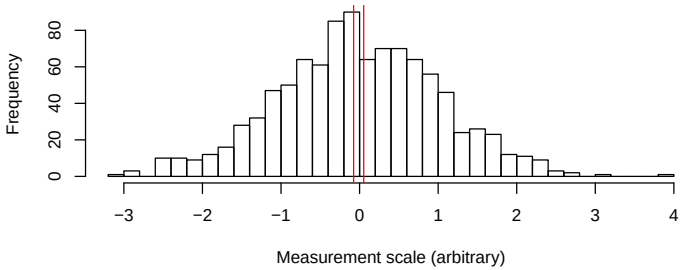


Figure 4: Conventional Confidence Interval for the mean.

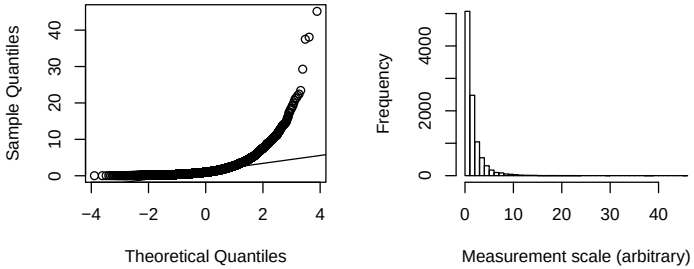


Figure 5: An illustration of a (strongly) non-Gaussian distribution from which data may be randomly sampled. The QQ-plot on the left shows strong departure from the Gaussian assumption. This distribution will be used as an example for the computation of Confidence Intervals in both the asymptotic and the resampling framework.

Conventional CI for the mean
Function application

```
samplesize <- 1000
set.seed(1)
vec <- rlnorm(samplesize, meanlog = 0);
CI(vec)

 2.5% 97.5%
 1.54  1.83
```

```
par(mfrow=c(1,1), cex=0.85)
hist(vec, breaks=40, main = "", xlab = "Measurement scale (arbitrary)")
abline(v = CI(vec), col = "red")
```

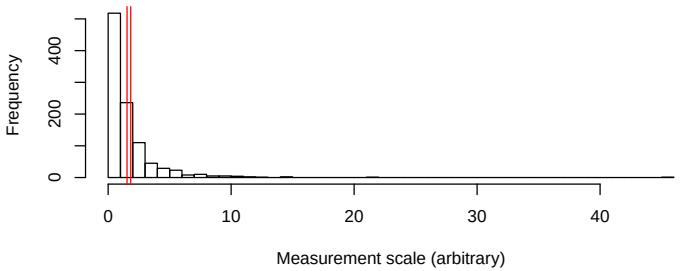


Figure 6: Conventional Confidence Interval for the mean.

Conventional CI for the mean
Function application

```
samplesize <- 50
set.seed(1)
vec <- rlnorm(samplesize, meanlog = 0);
CI(vec)

 2.5% 97.5%
 1.17  1.77
```

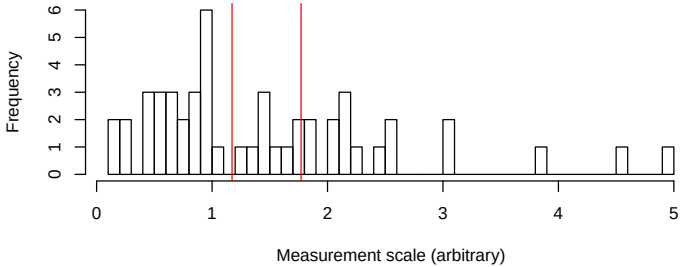


Figure 7: Conventional Confidence Interval for the mean.

Conventional CI for the mean
Function application

```
samplesize <- 10
set.seed(1)
vec <- rlnorm(samplesize, meanlog = 0);
CI(vec)

 2.5% 97.5%
0.688 2.345
```

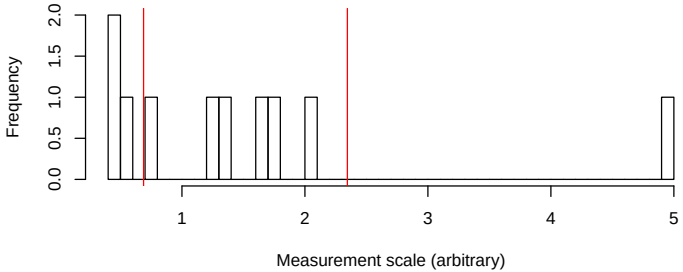


Figure 8: Conventional Confidence Interval for the mean.

Issues with conventional CIs

- They rely on distributional assumptions;
- These distributional assumptions imply that estimating different parameters involves different formulas;

Resampling or "bootstrap" framework

The "bootstrap principle"

- "The sample is to the population...
- what the bootstrap sample is to the sample";

Resampling or "bootstrap" framework

The "bootstrap principle"

- We can then use this principle to "build" a population of bootstrap samples;
- Principle: Draw random samples from the original sample (with replacement) *a very high number of times*;
- This can be done for *any* parameter (mean, median, linear regression parameter...);

Resampling or "bootstrap" framework

Drawing a single bootstrap sample

```
vec
```

```
 [1] 0.534 1.202 0.434 4.930 1.390 0.440 1.628 2.092 1.779 0.737
```

```
median(vec)
```

```
[1] 1.3
```

Resampling or "bootstrap" framework

Drawing a single bootstrap sample (n° 1)

Note that the call to `set.seed(n)` is used only to enforce reproducibility in a pedagogical setting. It should not be used in real settings as we really need to get random samples.

```
set.seed(10)
n <- length(vec) # Bootstrap sample size
samB <- sample(vec, n, replace = TRUE)

samB

 [1] 0.440 4.930 1.390 1.628 0.534 0.434 0.434 0.434 1.628 1.390

median(samB)

[1] 0.962

vec

 [1] 0.534 1.202 0.434 4.930 1.390 0.440 1.628 2.092 1.779 0.737
```

Comparing the sample and a given bootstrap sample

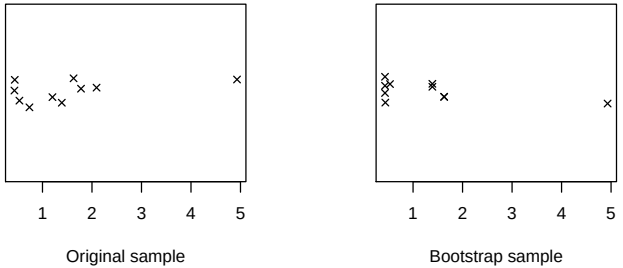


Figure 9: Comparing the original and a bootstrap sample.

Resampling or "bootstrap" framework

Drawing a single bootstrap sample (n° 2)

```
set.seed(20)
n <- length(vec) # Bootstrap sample size
samB <- sample(vec, n, replace = TRUE)

samB

 [1] 1.779 2.092 0.434 0.440 0.737 0.737 0.534 0.534 4.930 4.930

median(samB)

[1] 0.737

vec

 [1] 0.534 1.202 0.434 4.930 1.390 0.440 1.628 2.092 1.779 0.737
```

Comparing the sample and a given bootstrap sample

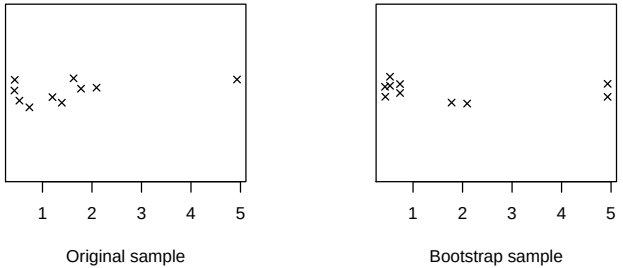


Figure 10: Comparing the original and a bootstrap sample.

Resampling or "bootstrap" framework

Drawing a single bootstrap sample (n° 3)

```
set.seed(30)
n <- length(vec) # Bootstrap sample size
samB <- sample(vec, n, replace = TRUE)

samB

 [1] 0.534 1.390 4.930 1.390 4.930 1.202 1.779 0.434 0.737 1.202

median(samB)

[1] 1.3

vec

 [1] 0.534 1.202 0.434 4.930 1.390 0.440 1.628 2.092 1.779 0.737
```

Comparing the sample and a given bootstrap sample

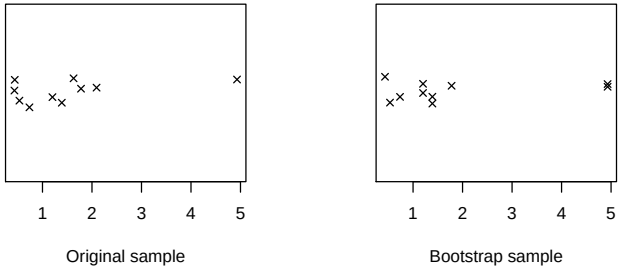


Figure 11: Comparing the original and a bootstrap sample.

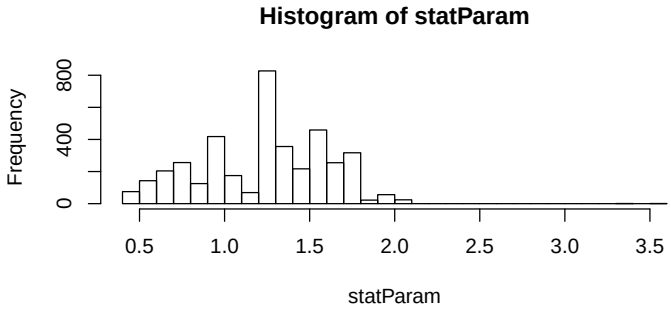
Performing a boostrap estimation

- Define the number of replications;
- Generate a loop and repeat the following for each replication / iteration:
  1. Generate a bootstrap sample;
  2. Compute the required statistical parameter on this bootstrap sample;
  3. Store the result in a vector;
- Then compute the distribution of these results (the parameter distribution);
- Estimate the relevant quantiles in order to compute the CI;

Performing a boostrap estimation

```r
n <- length(vec) # Bootstrap sample size
nreps <- 4000 # Number of replications
statParam <- rep(NA, nreps) # Storage vector for the estimate

for (i in 1:nreps) {
    samB <- sample(vec, n, replace = TRUE)
    statParam[i] <- median(samB)
}
```

Performing a boostrap estimation

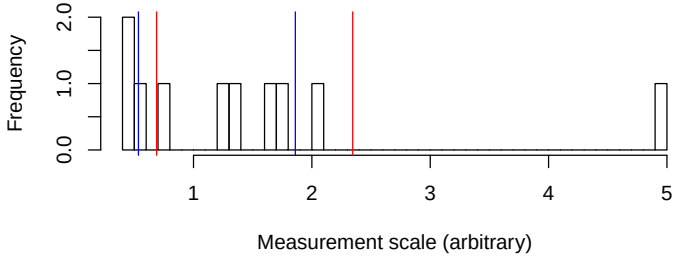


Performing a boostrap estimation

```
bCI <- quantile(statParam, prob = c(2.5, 97.5)/100)
bCI

 2.5% 97.5%
0.534 1.860
```

Compare with the original CI (for the mean):

```
CI(vec)

 2.5% 97.5%
0.688 2.345
```


Frequency
0.0
1.0
2.0
1
2
3
4
5
Measurement scale (arbitrary)

Issues with the standard bootstrap

- The number of replication samples is choosen in order to reach relative stability of the estimate. Some time must be spent on evaluating the adequate number of replications;
- Standard bootstrap interval estimates are *inaccurate*: they will include the true value less often than the predicted probability;
- They are *imprecise*: they will include more erroneous values than is desirable (Good, 2005a);
- Using the R boot library provides CI computation functions with methods to deal with these errors;

Issues with sample size

- Issues may arise concerning the applicability of bootstrap methods to small initial sample sizes;
- As mentionned *supra*, it has been shown that the standard bootstrap generates *inacurrate* and *imprecise* CI end-points;
- There are several solutions that are available in order to solve this issue;
- Efron (1987) describes the "non-parametric BC$_a$" (*Bias Corrected accelerated*) Confidence Interval (see also DiCiccio & Efron, 1996);
- See also Ho & Lee (2005) for evaluations of various solutions (among which *parametric* bootstraps);

The boot library

The boot library is made available by Canty & Ripley (2016). If it is not already installed:

```
install.packages("boot")
```

Then load the library:

```
library(boot)
```

Bootstrapping with the boot library

- The bootstrap parameter estimation must be defined in a home-made function;
- Then the `boot()` function calls this home-made function;

Bootstrapping with the boot library

Defining the parameter estimation function

The parameter estimation function takes 2 arguments:

1. The data object;
2. The indexing vector in the data object;

```
SPar <- function(data, index) {
  res <- median(data[index])
  return(res)
}
```

Bootstrapping with the boot library

It is useful to verify the function application

```
SPar(vec, 1:length(vec))
```

```
[1] 1.3
```

Confirm that it is equal to:

```
median(vec)
```

```
[1] 1.3
```

Bootstrapping with the boot library

Performing the bootstrap

```
nreps = 2000
#bootRes <- boot(vec, statistic = SPar, R = nreps, sim = "ordinary", stype = "i")
bootRes <- boot(vec, statistic = SPar, R = nreps)
str(bootRes)

List of 11
 $ t0       : num 1.3
 $ t        : num [1:2000, 1] 1.628 1.202 1.703 0.534 1.415 ...
 $ R        : num 2000
 $ data     : num [1:10] 0.534 1.202 0.434 4.93 1.39 ...
 $ seed     : int [1:626] 403 84 356515316 1424289583 -339859737 -1122151017 963274428 -22198097 -430865073 -146139
 $ statistic:function (data, index)
  ..- attr(*, "srcref")=Class 'srcref'  atomic [1:8] 1 9 4 1 9 1 1 4
  .. .. ..- attr(*, "srcfile")=Classes 'srcfilecopy', 'srcfile' <environment: 0x7da0930>
 $ sim      : chr "ordinary"
 $ call     : language boot(data = vec, statistic = SPar, R = nreps)
 $ stype    : chr "i"
 $ strata   : num [1:10] 1 1 1 1 1 1 1 1 1 1
 $ weights  : num [1:10] 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
 - attr(*, "class")= chr "boot"
```

Bootstrapping with the boot library
Accessing the information

The `boot()` function returns a list object which contains the following information (among others):

- **t0** Contains the original sample's value for the statistical parameter;
- **t** Contains the boostrapped values (as many as there are replications);
- **R** The number of replications;
- **data** The original sample's data;

Bootstrapping with the boot library

- It is then possible to use the library to compute various (uncorrected and corrected) estimates of a Confidence Interval;

Bootstrapping with the boot library
Computing a Confidence Interval

For example,

```
CIs <- boot.ci(bootRes, conf = 0.95, type = c("norm", "basic", "bca"))
str(CIs)

List of 6
 $ R     : int 2000
 $ t0    : num 1.3
 $ call  : language boot.ci(boot.out = bootRes, conf = 0.95, type = c("norm", "basic",      "bca"))
 $ normal: num [1, 1:3] 0.95 0.61 2.11
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:3] "conf" "" ""
 $ basic : num [1, 1:5] 0.95 1950.97 50.03 0.732 2.057
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:5] "conf" "" "" "" ...
 $ bca   : num [1, 1:5] 0.95 29.15 1919.61 0.487 1.779
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:5] "conf" "" "" "" ...
 - attr(*, "class")= chr "bootci"
```

Bootstrapping with the boot library
Computing a Confidence Interval

For example, Efron (1987)'s "non-parametric BC$_a$" Confidence Interval is available:

```
CIs$bca[4:5]

[1] 0.487 1.779
```

Compare with what we found:

```
bCI

 2.5% 97.5%
0.534 1.860

CI(vec)

 2.5% 97.5%
0.688 2.345
```

Bootstrapping a linear regression from real data

- We will use a subset of a dataset that was generated from a speech production study in which locus equations in Jordanian Arabic were investigated (Abuoudeh & Crouzet, 2014);
- In order to replicate these analyses, you will need to download the corresponding dataset extract from:
  - `https://osf.io/j8pys/download`
- and then load the corresponding file in R:

```
load("locusdata.Rdata")
```

Bootstrapping a linear regression from real data

Data are usually stored in 2D datasets (dataframes in R);

```
     C V position num locuteur atburst F2ons F2mid F3mid duration length
2422 d a   attaque 623       Mo    1593  1619  1464  2480     81.2 courte
2443 d i   attaque 644       Mo    1759  1922  1964  2791     68.8 courte
2463 d u   attaque 664       Mo    1705  1580  1326  2192     87.5 courte
2489 d u   attaque 691       Mo      NA  1450    NA  2368     75.0 courte
2506 d i   attaque 708       Mo    1724  1754  1852  2721     93.8 courte
2518 d a   attaque 720       Mo    1595  1588  1596  2524     87.5 courte
     intervalsize sex
2422          101   m
2443           93   m
2463          101   m
2489           96   m
2506          114   m
2518          101   m
```

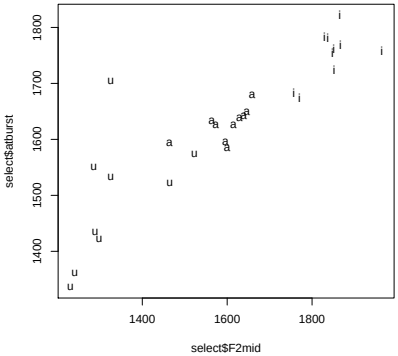
Bootstrapping a linear regression from real data

- These data originate from speech recordings aimed at investigating "locus equations";
- "locus equations" are linear regressions expressing the relation between the frequencies of F_2 at the burst of a consonant and at the middle of a coarticulated vowel (e.g. in a CV sequence);
- A linear function of the form $y = ax + b$ (with a the slope and b the intercept) is usually described as an indicator of the degree of coarticulation between the consonant and the vowel;

```
'data.frame': 30 obs. of  13 variables:
 $ C           : Factor w/ 5 levels "b","d","g","k",..: 2 2 2 2 2 2 2 2 2 2 ...
 $ V           : Factor w/ 6 levels "a","a:","i","i:",..: 1 3 5 5 3 1 1 3 5 5 ...
 $ position    : Factor w/ 2 levels "attaque","finale": 1 1 1 1 1 1 1 1 1 1 ...
 $ num         : int  623 644 664 691 708 720 761 765 791 802 ...
 $ locuteur    : Factor w/ 7 levels "Ah","Al","As",..: 5 5 5 5 5 5 5 5 5 5 ...
 $ atburst     : int  1593 1759 1705 NA 1724 1595 1639 1755 1434 1550 ...
 $ F2ons       : int  1619 1922 1580 1450 1754 1588 1609 1879 1550 1660 ...
 $ F2mid       : int  1464 1964 1326 NA 1852 1596 1629 1848 1289 1286 ...
 $ F3mid       : int  2480 2791 2192 2368 2721 2524 2528 2552 2332 2282 ...
 $ duration    : num  81.2 68.8 87.5 75 93.8 ...
 $ length      : Factor w/ 2 levels "courte","longue": 1 1 1 1 1 1 1 1 1 1 ...
 $ intervalsize: int  101 93 101 96 114 101 100 108 100 98 ...
 $ sex         : Factor w/ 1 level "m": 1 1 1 1 1 1 1 1 1 1 ...
```

Bootstrapping a linear regression from real data

Let's take a LE for the voiced alveolar stop /d/ in various vocalic contexts (Jordanian Arabic, short vowels only):

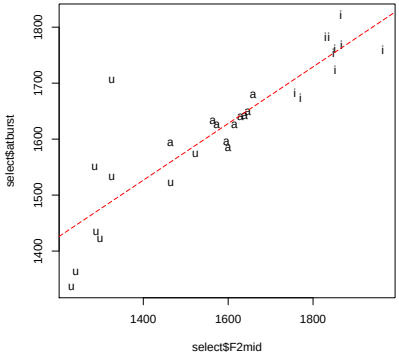


Bootstrapping a linear regression from real data

Computing Locus Equations

```
## Compute LE = (simple) linear regression
model <- lm(select$atburst ~ select$F2mid)
```

Bootstrapping a linear regression from real data

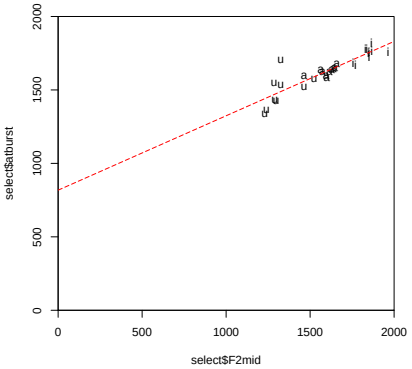


Bootstrapping a linear regression from real data

```
## Extract LE parameters
slope <- model$coefficients[2]
intercept <- model$coefficients[1]

slope

select$F2mid
       0.506

intercept

(Intercept)
        818
```

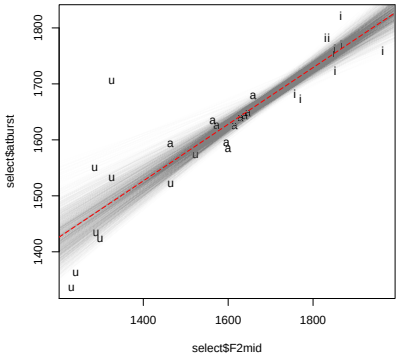
Bootstrapping a linear regression from real data

$$y = 0.506 \times x + 817.709 \tag{3}$$



Bootstrapping a linear regression from real data

Only for illustrating the process, one may plot the results of linear regressions over all bootstrap samples:



Bootstrapping a linear regression from real data

Using the boot library

Define the parameter estimation function:

```
bslope <- function(data, index) {
  slope <- lm(data[index, ]$atburst ~ data[index, ]$F2mid)$coefficients[2]
  return(slope)
}

bintercept <- function(data, index) {
  intercept <- lm(data[index, ]$atburst ~ data[index, ]$F2mid)$coefficients[1]
  return(intercept)
}
```

Bootstrapping a linear regression from real data

Test the function

```
bslope(select, 1:length(select))

data[index, ]$F2mid
              0.353

bintercept(select, 1:length(select))

(Intercept)
       1087
```

Bootstrapping a linear regression from real data

Perform the bootstrap (separately on the slope / intercept)

```
nreps = 2000
bootSl <- boot(select, statistic = bslope, R = nreps)
bootSl


ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = select, statistic = bslope, R = nreps)


Bootstrap Statistics :
    original   bias    std. error
t1*    0.506 9.45e-05       0.068
```

Bootstrapping a linear regression from real data

Perform the bootstrap (separately on the slope / intercept)

```
bootInt <- boot(select, statistic = bintercept, R = nreps)
bootInt


ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = select, statistic = bintercept, R = nreps)


Bootstrap Statistics :
    original  bias    std. error
t1*      818   -2.98         113
```

Bootstrapping a linear regression from real data

Compute the boostrapped CIs

```
CIS1 <- boot.ci(bootS1, conf = 0.95, type = "bca")
CIs$bca[4:5]

[1] 0.487 1.779

CIInt <- boot.ci(bootInt, conf = 0.95, type = "bca")
CIInt$bca[4:5]

[1]  642 1118
```

Bootstrap procedures: There's more to discover

- We've only adressed parameter estimation (partially);
- It may also be used for hypothesis testing (comparing means for continuous variables, comparing frequencies for categorical variables) in so-called "permutation tests";
- Though it is then still part of the NHST (*Null-Hypothesis Significance Testing*) framework, it may also help (me) understanding "parts of" Bayesian approaches to statistics;

(incomplete) Suggested readings

- Good, P. I. (2005c). *Resampling Methods: A Practical Guide to Data Analysis*. Birkhäuser, 3rd ed.
- Good, P. (2005b). *Permutation, Parametric and Bootstrap Tests of Hypotheses*. Springer Series in Statistics, New-York, USA: Springer-Verlag Inc., 3rd ed.
- Robert, C., & Casella, G. (2010). *Introducing Monte Carlo Methods with R*. UseR!, New-York, USA: Springer-Verlag.
- Carpenter, J., & Bithell, J. (2000). Bootstrap confidence intervals: when, which, what? A practical guide for medical statisticians.
- ...

Concerning the specific issues associated with the computation of Confidence Intervals, several interesting sources are available (DiCiccio & Efron, 1996; Efron, 1987; Ho & Lee, 2005).

Bibliographie I

Abuoudeh, M., & Crouzet, O. (2014). Vowel length impact on locus equation parameters: An investigation on Jordanian Arabic. in *Interspeech 2014 – 15th Annual Conference of the International Speech Communication Association*, pp. 184–188, Singapore: Chinese and Oriental Languages Information Processing Society – COLIPS, 2014, 14th–18th September.

Canty, A., & Ripley, B. D. (2016). *boot: Bootstrap R (S-Plus) Functions*. R package version 1.3-18.

Carpenter, J., & Bithell, J. (2000). Bootstrap confidence intervals: when, which, what? A practical guide for medical statisticians.

DiCiccio, T. J., & Efron, B. (1996). Bootstrap confidence intervals. *Statistical Science*, *11*(3), 189–228.

Efron, B. (1987). Better Bootstrap Confidence Intervals. *Journal of the American Statistical Association*, *82*(397), 171–185.

Good, P. (2005a). *Introduction to Statistics through Resampling Methods and R/S-Plus*. NJ: Hoboken, USA: Wiley.

Good, P. (2005b). *Permutation, Parametric and Bootstrap Tests of Hypotheses*. Springer Series in Statistics, New-York, USA: Springer-Verlag Inc., 3rd ed.

Good, P. I. (2005c). *Resampling Methods: A Practical Guide to Data Analysis*. Birkhäuser, 3rd ed.

Ho, Y. H. S., & Lee, S. M. S. (2005). Iterated smoothed bootstrap confidence intervals for population quantiles. *The Annals of Statistics*, *33*(1), 437–462.

Robert, C., & Casella, G. (2010). *Introducing Monte Carlo Methods with R*. UseR!, New-York, USA: Springer-Verlag.