The STM R package

Santiago Barreda and T. Florian Jaeger

# Introduction

This package is meant to facilitate the simulation of vowel perception and the creation of related visualizations. First install the package if necessary, and load it. The finalized version of the package will be available on CRAN, it is only on GitHub for now.

# Examples of Usage

## Data setup

We load the STM package and the h95\_data (Hillenbrand et al. 1995, henceforth H95) dataset. This is the complete data asociated with the H95 paper that was once hosted on James Hillenbrand’s academic website, now down (and distributed with his permission). The data loaded below contains: formant data at a ‘stable part’ of the vowel, formant values at different time points, listener classifications for each judgement, and two estimates of : psi\_1 (based on the stable part of the formants) and psi\_2 (based on measurements at 20% and 80%).

library (STM)  
data (h95\_data)  
head (h95\_data)

file dur f0 f1 f2 f3 f1\_1 f2\_1 f3\_1 f1\_2 f2\_2 f3\_2 f1\_3 f2\_3 f3\_3  
1 b01ae 257 238 630 2423 3166 625 2388 3174 651 2413 3115 675 2463 3011  
140 b01ah 212 241 831 1676 2602 820 1688 2511 845 1684 2583 845 1681 2606  
279 b01aw 242 247 725 1384 2642 722 1411 2669 725 1384 2642 733 1384 2662  
418 b01eh 184 214 713 2095 3129 611 2121 3242 643 2071 3078 672 2079 3120  
557 b01ei 222 230 534 2690 3335 582 2454 3223 621 2573 3260 579 2632 3287  
696 b01er 227 240 608 1733 2159 567 1599 2255 566 1664 2180 579 1709 2168  
 f1\_4 f2\_4 f3\_4 f1\_5 f2\_5 f3\_5 f1\_6 f2\_6 f3\_6 f1\_7 f2\_7 f3\_7 f1\_8 f2\_8 f3\_8  
1 687 2391 2926 683 2295 2888 696 2187 2888 793 2074 2866 806 2049 2961  
140 858 1690 2600 863 1696 2576 875 1745 2569 845 1858 2657 807 1980 2893  
279 731 1385 2681 748 1388 2682 758 1458 2690 772 1579 2739 805 1699 2777  
418 700 2109 3162 703 2079 3044 682 2133 3092 687 2124 3119 672 2140 3128  
557 534 2690 3335 498 2797 3299 480 2883 3294 455 2956 NA 440 3012 3318  
696 635 1746 2103 644 1678 2046 634 1672 2054 633 1778 2078 623 1781 2157  
 type speaker vowel ipa correct psi\_1 psi\_2 ae ah aw eh ei er ih iy oa  
1 b b01 ae æ 95 7.317269 7.315179 19 0 0 1 0 0 0 0 0  
140 b b01 ah ɑ 95 7.317269 7.315179 0 19 0 0 0 0 0 0 0  
279 b b01 aw ɔ 10 7.317269 7.315179 0 9 2 0 0 0 0 0 0  
418 b b01 eh ɛ 100 7.317269 7.315179 0 0 0 20 0 0 0 0 0  
557 b b01 ei e 100 7.317269 7.315179 0 0 0 0 20 0 0 0 0  
696 b b01 er ɝ 100 7.317269 7.315179 0 0 0 0 0 20 0 0 0  
 oo uh uw vote  
1 0 0 0 ae  
140 1 0 0 ah  
279 0 9 0 ah  
418 0 0 0 eh  
557 0 0 0 ei  
696 0 0 0 er

We first impute missing values in the formant data using the impute\_NA function. This function uses a linear model to impute, and optionally adds error to estimates (not used here but useful when using resampling methods bootstrapping).

h95\_data[,c(10:12,28:30)] =   
 impute\_NA (log(h95\_data[,c(10:12,28:30)]), h95\_data$speaker, h95\_data$vowel)

We will focus on the 20% and 80% time points, for the first 3 formants. We collect the formants, f0, psis, classification information, and category names.

ffs = log(h95\_data[,c(10:12,28:30)])  
f0s = log(h95\_data[,3])  
psis = h95\_data$psi\_2  
  
data(h95\_classifications)  
classifications = h95\_classifications  
vs = colnames (classifications)

We normalize the log-transformed formant data using the normalize function, which employs the regression-based method of log-mean normalization outlined in Barreda and Nearey (2018).

nffs = normalize (h95\_data[,c(10:12,28:30)],h95\_data$speaker,h95\_data$vowel)

And calculate templates for the dialect based on the available data. Templates contains category means, covariance matrices, and precision matrices.

template = create\_template (ffs-psis, h95\_data$vowel,shared\_covar = FALSE)

## Example Analysis

If is known, formants may be normalized as usual and the STM function may be used. In addition, this function may be used on templates trained on unnormalized data for classification without normalization. The STM function calculates posterior probabilities for each category based on the template and token acoustic properties. It returns a matrix of posterior probabilities, one row for each observation and one column for each candidate category.

STM (nffs[1,1:6], template = template)

This function works by calculating the log density of the formant vector for each category, and then calculating posteriors based on these densities. correctOUflow.internal is a function that changes 0 and 1 to .Machine$double.xmin and .9999999 respectively, in order to avoid problems when calculating the log likelihood.

STM

function (formants, template, vowel\_priors = NULL, correctOUflow = TRUE)   
{  
 n\_samples = nrow(formants)  
 n\_classes = nrow(template$means)  
 log\_vowel\_density = matrix(0, n\_samples, n\_classes)  
 for (j in 1:n\_classes) {  
 log\_vowel\_density[, j] = dmvnorm\_fast(formants, template$means[j,   
 ], template$covariance[[j]], log = TRUE)  
 }  
 log\_vowel\_density = log\_vowel\_density - apply(log\_vowel\_density,   
 1, function(x) mean(x))  
 if (!is.null(vowel\_priors))   
 log\_vowel\_density = sweep(log\_vowel\_density, 2, log(vowel\_priors),   
 `+`)  
 vowel\_densities = exp(log\_vowel\_density)  
 posterior\_probabilities = vowel\_densities/rowSums(vowel\_densities)  
 if (correctOUflow)   
 posterior\_probabilities = correctOUflow\_internal(posterior\_probabilities)  
 colnames(posterior\_probabilities) = rownames(template$means)  
 posterior\_probabilities = STM\_posteriors(posterior\_probabilities)  
 return(posterior\_probabilities)  
}  
<bytecode: 0x000002c761219a48>  
<environment: namespace:STM>

The BSTM function takes in a formant vector, an (optional depending on the method) f0 value, and a dialectal template. It returns information regarding the posterior probabilities for each category, the likelihoods, and the priors.

BSTM (ffs[1,],f0s[1], template = template)

posterior\_mu posterior\_sd posterior\_density posterior\_probability rounded\_pp  
ae 7.296 0.024 2.826 0.996 0.996  
ah 7.395 0.028 -24.474 0.000 0.000  
aw 7.489 0.030 -58.967 0.000 0.000  
eh 7.330 0.024 -2.693 0.004 0.004  
ei 7.354 0.026 -26.650 0.000 0.000  
er 7.541 0.031 -25.733 0.000 0.000  
ih 7.350 0.022 -9.490 0.000 0.000  
iy 7.262 0.030 -63.778 0.000 0.000  
oa 7.464 0.029 -33.754 0.000 0.000  
oo 7.406 0.026 -20.620 0.000 0.000  
uh 7.392 0.024 -21.023 0.000 0.000  
uw 7.457 0.031 -17.017 0.000 0.000  
 likelihood\_mu likelihood\_sd likelihood\_density prior\_mu prior\_sd  
ae 7.287 0.026 11.044 7.338 0.056  
ah 7.415 0.033 -15.894 7.338 0.056  
aw 7.549 0.035 -45.966 7.338 0.056  
eh 7.328 0.027 5.197 7.338 0.056  
ei 7.358 0.029 -18.722 7.338 0.056  
er 7.634 0.038 -8.187 7.338 0.056  
ih 7.352 0.024 -1.587 7.338 0.056  
iy 7.231 0.036 -54.588 7.338 0.056  
oa 7.512 0.035 -22.356 7.338 0.056  
oo 7.425 0.029 -11.789 7.338 0.056  
uh 7.404 0.027 -12.570 7.338 0.056  
uw 7.508 0.037 -5.889 7.338 0.056  
 prior\_density  
ae 1.822  
ah 1.822  
aw 1.822  
eh 1.822  
ei 1.822  
er 1.822  
ih 1.822  
iy 1.822  
oa 1.822  
oo 1.822  
uh 1.822  
uw 1.822

Alternatively, the function can be run on a matrix of formant vectors, and a vector of f0 values. In this case, a list of results is returned as an STM\_output\_list object.

analysis = BSTM (ffs[1:5,],f0s[1:5], template = template)  
  
analysis

STM\_output\_list object with the following elements:  
  
$list: a list of 5 STM\_output objects  
$winners: a data frame of the winning psi and vowel for each token  
$posterior: a data frame of the posterior probabilities for each token

Below, we show the results for the fifth observation in the dataset.

analysis[[5]]

posterior\_mu posterior\_sd posterior\_density posterior\_probability rounded\_pp  
ae 7.311 0.024 -31.228 0.000 0.000  
ah 7.357 0.028 -58.905 0.000 0.000  
aw 7.406 0.030 -75.977 0.000 0.000  
eh 7.290 0.024 -25.085 0.000 0.000  
ei 7.327 0.026 2.728 1.000 1.000  
er 7.489 0.031 -27.010 0.000 0.000  
ih 7.347 0.022 -19.607 0.000 0.000  
iy 7.338 0.030 -15.008 0.000 0.000  
oa 7.420 0.029 -68.827 0.000 0.000  
oo 7.420 0.026 -32.553 0.000 0.000  
uh 7.414 0.024 -35.136 0.000 0.000  
uw 7.460 0.031 -21.367 0.000 0.000  
 likelihood\_mu likelihood\_sd likelihood\_density prior\_mu prior\_sd  
ae 7.307 0.026 -25.799 7.325 0.056  
ah 7.369 0.033 -53.289 7.325 0.056  
aw 7.439 0.035 -69.108 7.325 0.056  
eh 7.282 0.027 -19.458 7.325 0.056  
ei 7.327 0.029 8.116 7.325 0.056  
er 7.563 0.038 -15.366 7.325 0.056  
ih 7.351 0.024 -14.131 7.325 0.056  
iy 7.343 0.036 -9.583 7.325 0.056  
oa 7.456 0.035 -61.450 7.325 0.056  
oo 7.446 0.029 -25.315 7.325 0.056  
uh 7.435 0.027 -28.174 7.325 0.056  
uw 7.519 0.037 -11.774 7.325 0.056  
 prior\_density  
ae 1.918  
ah 1.918  
aw 1.918  
eh 1.918  
ei 1.918  
er 1.918  
ih 1.918  
iy 1.918  
oa 1.918  
oo 1.918  
uh 1.918  
uw 1.918

The get\_winners function returns the category with the highest posterior probability for each observation, and the corresponding estimate of .

get\_winners(analysis)

psi\_hat vowel\_hat  
1 7.296313 ae  
2 7.302488 ah  
3 7.329890 aw  
4 7.284129 eh  
5 7.326943 ei

And the get\_posterior function returns the posterior probability for each category for each observation.

get\_posterior (analysis)

ae ah aw eh ei er ih iy oa oo uh uw  
1 0.996 0.000 0.000 0.004 0 0 0 0 0 0.000 0.000 0  
2 0.000 0.977 0.016 0.000 0 0 0 0 0 0.000 0.007 0  
3 0.000 0.292 0.422 0.000 0 0 0 0 0 0.000 0.286 0  
4 0.008 0.000 0.000 0.991 0 0 0 0 0 0.001 0.000 0  
5 0.000 0.000 0.000 0.000 1 0 0 0 0 0.000 0.000 0

To use a different estimation method we use the method parameter in BSTM as shown below.

BSTM (ffs[1,],f0s[1], template = template, method = method2)

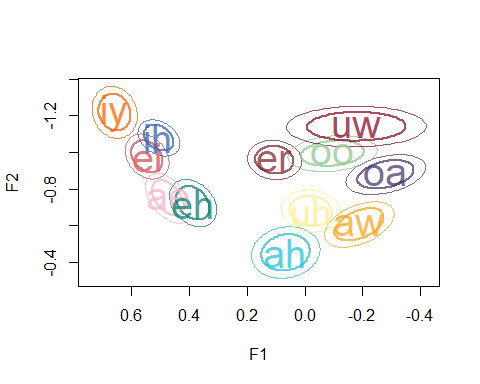
posterior\_mu posterior\_sd posterior\_density posterior\_probability rounded\_pp  
ae 7.287 0.026 2.729 0.997 0.997  
ah 7.415 0.033 -24.208 0.000 0.000  
aw 7.549 0.035 -54.280 0.000 0.000  
eh 7.328 0.027 -3.117 0.003 0.003  
ei 7.358 0.029 -27.036 0.000 0.000  
er 7.634 0.038 -16.502 0.000 0.000  
ih 7.352 0.024 -9.902 0.000 0.000  
iy 7.231 0.036 -62.903 0.000 0.000  
oa 7.512 0.035 -30.671 0.000 0.000  
oo 7.425 0.029 -20.103 0.000 0.000  
uh 7.404 0.027 -20.884 0.000 0.000  
uw 7.508 0.037 -14.204 0.000 0.000  
 likelihood\_mu likelihood\_sd likelihood\_density prior\_mu prior\_sd  
ae 7.287 0.026 11.044 NA NA  
ah 7.415 0.033 -15.894 NA NA  
aw 7.549 0.035 -45.966 NA NA  
eh 7.328 0.027 5.197 NA NA  
ei 7.358 0.029 -18.722 NA NA  
er 7.634 0.038 -8.187 NA NA  
ih 7.352 0.024 -1.587 NA NA  
iy 7.231 0.036 -54.588 NA NA  
oa 7.512 0.035 -22.356 NA NA  
oo 7.425 0.029 -11.789 NA NA  
uh 7.404 0.027 -12.570 NA NA  
uw 7.508 0.037 -5.889 NA NA  
 prior\_density  
ae NA  
ah NA  
aw NA  
eh NA  
ei NA  
er NA  
ih NA  
iy NA  
oa NA  
oo NA  
uh NA  
uw NA

Note that the prior information is set to NA because this method places no a priori constraints on values of .

## Plotting

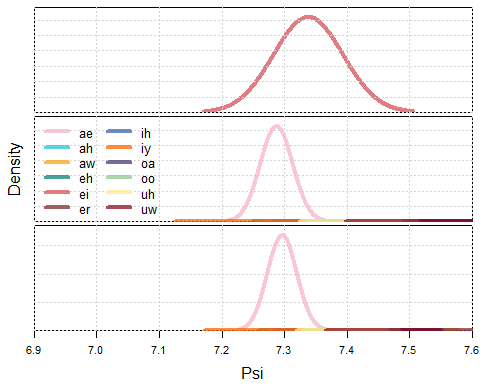
The package contains plotting functions defined for the STM\_template and STM\_output\_list objects. The plot function for the STM\_template object plots the means of the formant values for each category, and ellipses enclosing one and two standard deviations.

plot (template)



The plot function for STM\_output objects compares the prior probabilities of , the likelihood of of the formant pattern given different values of and different vowel categories, and the posterior probabilities of for each category.

plot (analysis[[1]])



# Walkthrough of the the BSTM function

We will describe the sequence of steps underlying the BSTM function, relying on the steps carried out in section **?@sec-setup**.

The BSTM function takes in a formant vector, an (optional depending on the metho) f0 value, and a template. It returns information regarding the posterior probabilities for each category, the likelihoods, and the priors.

BSTM (ffs[1,],f0s[1], template = template)

I am going to walk through the way this function works. BSTM and PSTM are basically a wrapper for the method functions in the estimation\_methods.R file. The method functions are the ones that actually calculate the likelihoods and priors. method6 implements the method 6 algorithm as seen below:

method6

function (ffs, f0, template, PSI\_prior\_mean = 7.233, PSI\_prior\_sd = 0.1284,   
 f0\_hat\_sd = 0.1327, f0\_hat\_intercept = -10.32, f0\_hat\_slope = 2.145,   
 vowel\_priors = vowel\_priors, correctOUflow = TRUE, type = "BSTM",   
 ...)   
{  
 n\_classes = nrow(template$means)  
 likelihoods = t(sapply(1:n\_classes, function(j) {  
 estimate\_likelihood(ffs, template$means[j, ], template$covariance[[j]],   
 template$precision[[j]])  
 }))  
 rownames(likelihoods) = rownames(template$means)  
 priors = estimate\_f0\_psi\_prior(f0)  
 priors = matrix(priors, n\_classes, 3, byrow = TRUE)  
 colnames(priors) = c("prior\_mu", "prior\_sd", "prior\_density")  
 if (!is.null(vowel\_priors))   
 priors$prior\_density = priors$prior\_density + log(vowel\_priors)  
 posterior = find\_posterior(likelihoods, priors, type = type)  
 if (correctOUflow)   
 posterior[, 4] = correctOUflow\_internal(posterior[, 4])  
 stimulus = c(ffs, f0)  
 parameters = c(PSI\_prior\_mean = PSI\_prior\_mean, PSI\_prior\_sd = PSI\_prior\_sd,   
 f0\_hat\_sd = f0\_hat\_sd, f0\_hat\_intercept = f0\_hat\_intercept,   
 f0\_hat\_slope = f0\_hat\_slope)  
 output = STM\_output(cbind(posterior, likelihoods, priors),   
 stimulus, template, parameters, vowel\_priors)  
 return(output)  
}  
<bytecode: 0x000002c76147e268>  
<environment: namespace:STM>

The other methods (2 and 3) include subsets of the process above, so I will not go through method6. First, the estimate\_likelihood function is called for each category. This function calculates the log likelihood of the formant vector given the category mean and covariance. This is done using the method in Terry’s notes which yields the mean and standard deviation. The density at the mean is calculated using the density function. We will in general collect: the mu, the sd, the peak (log) density.

estimate\_likelihood

function (ffs, means, covariance, precision = solve(covariance))   
{  
 intercept\_template = sum(unlist(ffs - means) %\*% precision)  
 slope\_template = -sum(precision)  
 likelihood\_mu = (-intercept\_template/slope\_template)  
 likelihood\_sd = sqrt(-1/slope\_template)  
 likelihood\_density = dmvnorm\_fast(ffs - likelihood\_mu, means,   
 covariance, log = TRUE)  
 output = c(likelihood\_mu = likelihood\_mu, likelihood\_sd = likelihood\_sd,   
 likelihood\_density = likelihood\_density)  
 return(output)  
}  
<bytecode: 0x000002c761481f58>  
<environment: namespace:STM>

We then calculate the joint prior pf and f0 using the estimate\_f0\_psi\_prior function. This function calculates the prior for the f0 value given the psi value. The mean and sd are calculated the way described by our derivation. The main addition is that we find the value of the density at the prior mean.

estimate\_f0\_psi\_prior

function (f0, PSI\_prior\_mean = 7.233, PSI\_prior\_sd = 0.1284,   
 f0\_hat\_sd = 0.1327, f0\_hat\_intercept = -10.32, f0\_hat\_slope = 2.145,   
 ...)   
{  
 intercept\_f0\_given\_psi = ((f0\_hat\_slope \* f0)/(f0\_hat\_sd^2) -   
 (f0\_hat\_intercept \* f0\_hat\_slope)/(f0\_hat\_sd^2))  
 slope\_f0\_given\_psi = ((-f0\_hat\_slope^2/(f0\_hat\_sd^2)))  
 f0\_given\_psi\_mu = (-intercept\_f0\_given\_psi/slope\_f0\_given\_psi)  
 f0\_given\_psi\_sd = sqrt(-1/slope\_f0\_given\_psi)  
 f0\_hat = f0\_hat\_intercept + f0\_hat\_slope \* f0\_given\_psi\_mu  
 f0\_given\_psi\_density = stats::dnorm(f0, f0\_hat, f0\_hat\_sd,   
 log = TRUE)  
 psi\_prior\_density = stats::dnorm(PSI\_prior\_mean, PSI\_prior\_mean,   
 PSI\_prior\_sd, log = TRUE)  
 prior\_density = f0\_given\_psi\_density + psi\_prior\_density  
 tmp = combine\_gaussians(f0\_given\_psi\_mu, f0\_given\_psi\_sd,   
 PSI\_prior\_mean, PSI\_prior\_sd, f0\_given\_psi\_density, psi\_prior\_density)  
 output = c(prior\_mu = tmp[1], prior\_sd = tmp[2], prior\_density = tmp[3])  
 return(output)  
}  
<bytecode: 0x000002c76155f510>  
<environment: namespace:STM>

We also call the combine\_gaussians function which combines gaussian curves of any given peak density as follows:

combine\_gaussians

function (mu\_1, sd\_1, mu\_2, sd\_2, d\_1 = NULL, d\_2 = NULL)   
{  
 if (is.null(d\_1))   
 d\_1 = 1/(sqrt(2 \* pi \* sd\_1^2))  
 if (is.null(d\_2))   
 d\_2 = 1/(sqrt(2 \* pi \* sd\_2^2))  
 mu = (mu\_1/sd\_1^2 + mu\_2/sd\_2^2)/(1/sd\_1^2 + 1/sd\_2^2)  
 sd = sqrt(1/(1/sd\_1^2 + 1/sd\_2^2))  
 d = dlike(mu, mu\_1, sd\_1, d\_1, log = TRUE) + dlike(mu, mu\_2,   
 sd\_2, d\_2, log = TRUE)  
 return(cbind(mu, sd, d))  
}  
<bytecode: 0x000002c761556b28>  
<environment: namespace:STM>

Finally (in terms of important steps) we call the find\_posterior function to combine our priors and likelihoods. This functions simply combines the priors and likelihoods and then scales the posterior probabilities. by calling scale\_posterior.

find\_posterior

function (likelihoods, priors, type = "BSTM")   
{  
 n\_vs = nrow(likelihoods)  
 posterior = matrix(0, n\_vs, 5, byrow = TRUE)  
 posterior[, 1:3] = combine\_gaussians(likelihoods[, 1], likelihoods[,   
 2], priors[, 1], priors[, 2], likelihoods[, 3], priors[,   
 3])  
 posterior = scale\_posterior(posterior, type = type)  
 colnames(posterior) = c("posterior\_mu", "posterior\_sd", "posterior\_density",   
 "posterior\_probability", "rounded\_pp")  
 rownames(posterior) = rownames(likelihoods)  
 return(posterior)  
}  
<bytecode: 0x000002c761571698>  
<environment: namespace:STM>

The scale\_posterior function scales the posterior probabilities to sum to 1. This is done by integration of the posterior distributions of for each category and summing across all categories (if type=BSTM), or by exponentiating and summing posterior densities (if type=PSTM).

scale\_posterior

function (posterior, type = "BSTM")   
{  
 if (type == "BSTM") {  
 vowel\_integrals = posterior[, 3] + log(posterior[, 2]) +   
 (log(pi) + log(2)) \* (0.5)  
 log\_sum\_vowel\_integrals = log(sum(exp(vowel\_integrals)))  
 posterior[, 3] = posterior[, 3] - log\_sum\_vowel\_integrals  
 posterior[, 4] = exp(vowel\_integrals - log\_sum\_vowel\_integrals)  
 }  
 if (type == "PSTM") {  
 posterior[, 4] = exp(posterior[, 3])/sum(exp(posterior[,   
 3]))  
 }  
 posterior[, 5] = round(posterior[, 4], 4)  
 return(posterior)  
}  
<bytecode: 0x000002c76157a5b0>  
<environment: namespace:STM>

# Comparison to Grid Search parameter estimation

As a sanity check we will compare the output of the BSTM function to the values we can calculate more directly using a grid search method. We use the same parameters as in methods 2, 3 and 6, and use the first observation from our data (any other can be used):

tmp\_ffs = ffs[1,]  
tmp\_f0 = f0s[1]  
  
psis = seq(6.8,7.6,.0001)  
PSI\_prior\_mean = 7.233  
PSI\_prior\_sd = 0.1284  
f0\_hat\_sd = 0.1327  
f0\_hat\_intercept = -10.32  
f0\_hat\_slope = 2.145

To estimate the method 6 prior:

# psi prior normal density  
psiprior = dnorm (psis,PSI\_prior\_mean,PSI\_prior\_sd)  
  
# predicted f0 for each psi  
f0\_hat = f0\_hat\_intercept + f0\_hat\_slope\*psis  
  
# density of f0 given psi  
f0\_given\_psi = dnorm (tmp\_f0, f0\_hat, f0\_hat\_sd)  
  
# joint density of f0|psi and psi  
prior = f0\_given\_psi \* psiprior

And likelihood:

# matrix of formant vector, repeated  
formant\_matrix = matrix (unlist(tmp\_ffs),length(psis),6,byrow=TRUE)  
  
# matrix of candidate psis, repeated across columns  
psi\_matrix = matrix (unlist(psis),length(psis),6)  
  
# for each category, density of pattern - psi candidate  
v\_likelihoods = matrix (0,length(psis),12)  
for (i in 1:12)  
 v\_likelihoods[,i] =   
 dmvnorm\_fast(formant\_matrix - psi\_matrix,   
 unlist(template$means[i,]),  
 template$covariance[[i]])

The posterior (before scaling) is the product of the likelihood and the prior:

posterior = v\_likelihoods \* matrix (prior,length(psis),12)

We numerically estimate the integral of all posteriors and divide the densities by this number to scale the total integral across all categories to 1:

integrals = rep(0,12)  
for (i in 1:12) integrals[i] = integrate\_numerical (psis, posterior[,i])  
  
posterior = posterior / sum(integrals)

And confirm that the total integral across categories is now 1:

integrals\_posterior = rep(0,12)  
for (i in 1:12) integrals\_posterior[i] = integrate\_numerical (psis, posterior[,i])  
sum (integrals\_posterior)

[1] 1

We will compare to the output of the BSTM function:

BSTM\_output = BSTM (tmp\_ffs, tmp\_f0, template = template)$df  
t(BSTM\_output[1,])

ae  
posterior\_mu 7.29631335  
posterior\_sd 0.02353500  
posterior\_density 2.82624049  
posterior\_probability 0.99592058  
rounded\_pp 0.99590000  
likelihood\_mu 7.28726847  
likelihood\_sd 0.02596349  
likelihood\_density 11.04363315  
prior\_mu 7.33799095  
prior\_sd 0.05573306  
prior\_density 1.82247956

The posterior mean corresponds to the MAP psi value:

BSTM\_output[1,'posterior\_mu']

[1] 7.296313

psis[which.max(posterior[,1])]

[1] 7.2963

the posterior probability corresponds well to the numerically estimated integral:

BSTM\_output[1,'posterior\_probability']

[1] 0.9959206

integrate\_numerical(psis, posterior[,1])

[1] 0.9959206

The posterior sd corresponds to what we can estimate using the second derivative of the posterior density:

log\_posterior <- log(posterior[,1])  
log\_posterior\_dx = diff(log\_posterior) / diff(psis)  
log\_posterior\_dx\_dx = mean (diff(log\_posterior\_dx) / diff(psis[-1]))  
  
  
BSTM\_output[1,'posterior\_sd']

[1] 0.023535

sqrt (-1/log\_posterior\_dx\_dx)

[1] 0.023535

and the maximum posterior log density also corresponds to our estimated value:

BSTM\_output[1,'posterior\_density']

[1] 2.82624

log(max(posterior[,1]))

[1] 2.82624

We can do the same estimates for our prior, seeing that these match our function outputs:

unlist(BSTM\_output[1,9:11])

prior\_mu prior\_sd prior\_density   
 7.33799095 0.05573306 1.82247956

psis[which.max(prior)]

[1] 7.338

log\_prior <- log(prior)  
log\_prior\_dx = diff(log\_prior) / diff(psis)  
log\_prior\_dx\_dx = mean (diff(log\_prior\_dx) / diff(psis[-1]))  
sqrt (-1/log\_prior\_dx\_dx)

[1] 0.05573306

log(max(prior))

[1] 1.82248

and for our likelihood:

unlist(BSTM\_output[1,6:8])

likelihood\_mu likelihood\_sd likelihood\_density   
 7.28726847 0.02596349 11.04363315

psis[which.max(v\_likelihoods[,1])]

[1] 7.2873

log\_likelihood <- log(v\_likelihoods[,1])  
log\_likelihood\_dx = diff(log\_likelihood) / diff(psis)  
log\_likelihood\_dx\_dx = mean (diff(log\_likelihood\_dx) / diff(psis[-1]))  
sqrt (-1/log\_likelihood\_dx\_dx)

[1] 0.02596349

log(max(v\_likelihoods[,1]))

[1] 11.04363

Q.E.D.!