Internship report: voice activity detection, report on methods implemented

Joshua Winebarger

Report on Methods Implemented

Joshua Winebarger
Geostat

2010-06-20

Abstract

As of the time of this writing, the goal of my work at Geostat has been, as a preliminary step, to reproduce the work of Almpanidis in voice activity detection or phone segementation using the DISTBIC-Γ method. For a review of this method, please see my previous works.

In this report I will describe the methods implemented or which were attempted in working towards a reproduction of the work of Almpanidis.

The DISTBIC-Γ method involves several steps. Most of these steps were implemented early in this project in C++, then translated into Matlab, which was a quick process. However, one step – the estimation of the parameters of the Generalized Γ distribution, has proven quite difficult to master and consequently has consumed most of my work. Because the other steps have already been described in my previous reports almost exactly as they are currently implemented, I will focus in this report on the methods used to estimate the GΓD.

0.1 Gradient Descent Algorithm

0.1.1 Theoretical Overview

This procedure is described in (Almpanidis, 2006, 2008) and (Shin, Chang, 2005). etc. as the method used by these researchers to estimate the parameters η β and γ for the GΓD.

Assume data are mutually independent
Calculate the following statistics in order to estimate γ

$$S_1(i) = (1-\zeta)S_1(i-1) + \zeta|x_i|^{\hat{\gamma}(i)} \tag{1}$$

$$S_2(i) = (1-\zeta)S_2(i-1) + \zeta \log|x_i|^{\hat{\gamma}(i)} \tag{2}$$

$$S_3(i) = (1-\zeta)S_2(i-1) + \zeta|x_i|^{\hat{\gamma}(i)} \log|x_i|^{\hat{\gamma}(i)} \tag{3}$$

where the statistics in question are given by the following:

$$S_1 = (1/N)\sum_{i=1}^{N}|x_i|^{\hat{\gamma}} \tag{4}$$

$$S_2 = (1/N)\sum_{i=1}^{N}\log|x_i|^{\hat{\gamma}} \tag{5}$$

$$S_3 = (1/N)\sum_{i=1}^{N}|x_i|^{\hat{\gamma}}\log|x_i|^{\hat{\gamma}} \tag{6}$$

update γ as:

$$\hat{\gamma}(i+1) = \hat{\gamma}(i) + \mu\left(\frac{1}{\hat{\eta}(i)} + S_2(i) - \frac{S_3(n)}{S_1(i)}\right) \tag{7}$$

Where ζ is a forgetting factor and μ is the learning rate.
By default, $\hat{\gamma}(1) = 1$.
The parameters used by Shin & Chang for forgetting and learning were:

$$\gamma = 0.0004, \mu = 0.0001$$

η and β are estimated by:

$$\psi_0(\hat{\eta}(i)) - \log\hat{\eta}(i) = S_2(i) - \log S_1(i) \tag{8}$$

$$\hat{\beta}(i) = \frac{\hat{\eta}(i)}{S_1(i)} \tag{9}$$

The left part of the first expression is monotonically increasing, so it is simple to do reverse lookup in a table to find η

0.1.2 Implementation

This was the first method I tried since it is used in the source papers.

I implemented this solely in Matlab. I could not make the algorithm converge properly.

The main fault was that certain parameters would change only a very small amount over the course of the iterations, and in any case even when initialized to the correct value would still exhibit the same "drifting" behavior.
I didn't pursue this method for more than a week before moving to other methods.

Take

0.2 SISE Equations Method

0.2.1 Introduction

In an attempt to circumvent the problems encountered with the Gradient Descent method, I sought other works in the literature which treated the problem of estimating the parameters. I found a paper published by Song in 2008 which purported to show a new method which was fast and globally convergent. This method used special equations called SISE to estimate the parameters using the Newton-Raphson wero-finding algorithm.

0.2.2 Theoretical Overview

(Re)-Introduction of the Generalized Γ Distribution and the traditional approach to its solution

The Generalized Γ Distribution pdf used in this method differs from the one used in previous methods:

PDF of Generalized Γ Distribution:

$$f(x;\theta) = \frac{\beta x^{\beta\lambda-1}}{\sigma^{\beta\lambda}\Gamma(\lambda)} exp\Big\{-(\frac{x}{\sigma})^{\beta}\Big\}. \quad \text{for} x \in R^{+} := (0,\infty) \tag{10}$$

ML estimator of GΓD: $\tilde{\theta}_n := (\tilde{\sigma}_n, \tilde{\beta}_n, \tilde{\lambda}_n)$
satisfies the following equations:

$$\tilde{\sigma}_n := \left(\frac{\sum_{i=1}^{n} X_i^{\tilde{\beta}_n}}{n\tilde{\lambda}_n}\right)^{1/\tilde{\beta}_n}$$

$$\tilde{\lambda}_n := \left[\beta_n \left(\frac{\sum_{i=1}^{n} X_i^{\tilde{\beta}_n}\log(X_i)}{\sum_{i=1}^{n} X_i^{\tilde{\beta}_n}}\right)\right]^{-1}$$

$$\frac{\tilde{\beta}_n}{n}\sum_{i=1}^{n}\log X_i - \log\sum_{i=1}^{n} X_i^{\tilde{\beta}_n} + \log(n\tilde{\lambda}_n) - \psi_0(\tilde{\lambda}) = 0$$

Where $\tilde{\sigma}_n$ is the shape parameter, We see that σ_n is an expression in terms of β_n and λ_n;
λ_n is expressed in terms of β_n

Solve the single equation resulting from substituting $\tilde{\lambda}_n$ in (2) into (3)
(3) is ill-behaved and difficult to solve numerically

SISE Equations

We will call the true parameter vector $\theta_0 = (\sigma_0, \beta_0, \lambda_0)$ Probability & expectation with respect to $f(x;\theta)$: P_0 and E_0.

Let $\mathcal{M}_0 : t \to E_0\{X^t\}$ First derivative: $\dot{\mathcal{M}}_0$
Define: $\dot{\mathcal{R}}_0 \doteq \frac{\dot{\mathcal{M}}_0}{\mathcal{M}_0}$ and $\dot{\mathcal{R}}_0(0) \doteq \lim_{t\to 0} \dot{\mathcal{R}}_0(t)$
From this and a few other equations not shown here, we develop:

$$\begin{aligned}\mathcal{S}(\beta) \doteq \log(\mathcal{M}_0(2\beta)) - 2\log(\mathcal{M}_0(\beta)) \\ - \log(1 + \beta(\dot{\mathcal{R}}_0(\beta) - \dot{\mathcal{R}}_0(0))) = 0 \end{aligned} \tag{11}$$

$$\mathcal{T}(\beta) \doteq \frac{\mathcal{M}_0(2\beta)}{\mathcal{M}_0^2(\beta)} - (1 + (\dot{\mathcal{R}}_0(\beta) - \dot{\mathcal{R}}_0(0))) = 0 \tag{12}$$

Since we generally do not know β_0, we can use instead a sample mapping:

$$\mathcal{M}_n : t \to \sum_{i=1}^{n} X_i^t / n \tag{13}$$

For the derivative we have:
$\dot{\mathcal{M}}_n(t) = \sum_{i=1}^{n} X_i^t \log(X_i)/n$ which estimates: $\dot{\mathcal{M}}_0(t) = E_0\{X^t\}\log X$
and:

$$\dot{\mathcal{R}}_n \doteq \frac{\dot{\mathcal{M}}_n(t)}{\mathcal{M}_n(t)} \tag{14}$$

$$\dot{\mathcal{R}}_n(0) \doteq \lim_{t\to 0} \dot{\mathcal{R}}_n(t) = \sum_{i=1}^{n} \log(X_i)/n \tag{15}$$

the sample SISE equations are:

$$\begin{aligned}\mathcal{S}(\beta) \doteq \log(\mathcal{M}_n(2\beta)) - 2\log(\mathcal{M}_n(\beta)) \\ - \log(1 + \beta(\dot{\mathcal{R}}_n(\beta) - \dot{\mathcal{R}}_n(0))) = 0 \end{aligned} \tag{16}$$

$$\mathcal{T}(\beta) \doteq \frac{\mathcal{M}_n(2\beta)}{\mathcal{M}_n^2(\beta)} - (1 + (\dot{\mathcal{R}}_n(\beta) - \dot{\mathcal{R}}_n(0))) = 0 \tag{17}$$

Estimation of Parameters with unknown shape index parameter λ_0

Step 1: Compute the estimator $\hat{\beta}_n$ by means of NR method based on either (16) or (17)
Step 2: Compute the estimators $\hat{\sigma}_n$ and $\hat{\lambda}_n$ by using $\hat{\beta}_n$ found in the previous

step using the expressions

$$\hat{\sigma}_n = \left(\frac{\mathcal{M}_n(\hat{\beta}_n)}{\hat{\lambda}_n} \right) \text{ and}$$
$$\hat{\lambda}_n = (\hat{\beta}_n(\dot{\mathcal{R}}_n(\hat{\beta}_n) - \dot{\mathcal{R}}_n(0)))^{-1}.$$

In regards to step 1, the NR root-finding equations are:

$$\hat{\beta}_{n,k+1} = \hat{\beta}_{n,k} - \frac{\mathcal{S}_n(\hat{\beta}_{n,k})}{\dot{\mathcal{S}}_n(\hat{\beta}_{n,k})} \tag{18}$$

$$\hat{\beta}_{n,k+1} = \hat{\beta}_{n,k} - \frac{\mathcal{T}_n(\hat{\beta}_{n,k})}{\dot{\mathcal{T}}_n(\hat{\beta}_{n,k})} \tag{19}$$

Where

$$\mathcal{S}_n(t) = \log(\mathcal{M}_n(2t)) - \log(\mathcal{M}_n(t)) - log(1 + \beta(\dot{\mathcal{R}}_n(t) - \dot{\mathcal{R}}_n(0))) = 0; \tag{20}$$

In the Appendix the author gives the derivative for $\mathcal{S}_0$:

$$\dot{\mathcal{S}}_0(t) = 2(\dot{\mathcal{R}}_0(2t) - \dot{\mathcal{R}}_0(t)) - \tag{21}$$

$$\frac{\dot{\mathcal{R}}_0(t) - \dot{\mathcal{R}}_0(0) + t(\mathcal{R}_0^{(2)}(t) - \dot{\mathcal{R}}_0^2(t))}{t(\dot{\mathcal{R}}(t) - \dot{\mathcal{R}}(0)) + 1} \tag{22}$$

I will simply assume we can use the same expression for n rather than 0.

$$\dot{\mathcal{S}}_n(t) = 2(\dot{\mathcal{R}}_n(2t) - \dot{\mathcal{R}}_n(t)) -$$
$$\frac{\dot{\mathcal{R}}_n(t) - \dot{\mathcal{R}}_n(0) + t(\mathcal{R}_n^{(2)}(t) - \dot{\mathcal{R}}_n^2(t))}{\beta(\dot{\mathcal{R}}(t) - \dot{\mathcal{R}}(0)) + 1} \tag{23}$$

$\mathcal{S}_n(t)$ and $\dot{\mathcal{S}}_n(t)$ depend on four terms:

$$\dot{\mathcal{R}}_n(t) = \frac{\dot{\mathcal{M}}_n(t)}{\dot{\mathcal{M}}_n} \tag{24}$$

$$\dot{\mathcal{R}}_n(0) = \lim_{t \to 0} \dot{\mathcal{R}}_n(t) \dot{\mathcal{R}}_n^{(2)}(t) = \frac{\mathcal{M}_n^{(2)}(t)}{\dot{\mathcal{M}}_n} \qquad \text{Second derivative} \tag{25}$$

$$\dot{\mathcal{R}}_n^2(t) = \left(\frac{\dot{\mathcal{M}}_n(t)}{\dot{\mathcal{M}}_n} \right)^2 \tag{26}$$

$$\text{Square} \tag{27}$$

$$F_n(\gamma) = \tag{28}$$

$$\psi_0\left(\frac{\frac{1}{N}\sum i=1^N|x_i|^\gamma}{\frac{1}{N^2}\sum_{i=1}^N\sum_{j=1}^N|x_i|^\gamma\log\frac{|x_i|^\gamma}{|x_j|^\gamma}}\right)$$

$$+\log\left(\frac{1}{N^2}\sum_{i=1}^N\sum j=1^N|x_i|^\gamma\log\frac{|x_i|^\gamma}{|x_j|^\gamma}\right)$$

$$-\frac{1}{N}\sum_{i=1}^N\log|x_i|^\gamma = 0 \tag{29}$$

We use the Newton-Raphson root-finding algorithm, which is:

$$\hat{\gamma}(k+1) = \hat{\gamma}(k) - \frac{F_n(\hat{\gamma}(k))}{\dot{F}_n(\hat{\gamma}(k))} \tag{30}$$

taking care to reject negative values of $\hat{\gamma}$ and to monitor its progress so that it reduces F_n. Shin & Chang note that the solution to (36) tends to diverge and cannot be obtained with approximately $N < 400$.

η and β are estimated by:

$$\psi_0(\hat{\eta}(k)) - \log\hat{\eta}(k) = S_2(k) - \log S_1(k) \tag{31}$$

$$\hat{\beta}(k) = \frac{\hat{\eta}(k)}{S_1(k)} \tag{32}$$

where the statistics in question are given by the following:

$$S_1 = (1/N)\sum_{i=1}^N|x_i|^{\hat{\gamma}} \tag{33}$$

$$S_2 = (1/N)\sum_{i=1}^N\log|x_i|^{\hat{\gamma}} \tag{34}$$

The left part of (38) is monotonically increasing, so it is simple to do a reverse lookup in a table to find η.

0.2.3 Implementation

I implemented this method in Matlab. In spite of the apparent promise of the method as perceived from the paper, I could not cause this method to correctly converge. While convergence was acheived, the algorithm (apparently) did not converge to the correct parameters of synthetic data generated from gaussian and gamma distributions.
However, in light of the fact that the pdf of this method has a different form than that of the others, and also considering the fact that perhaps at this point I had not learned to properly generate arbitrary pdfs using the inverse-CDF method, it is possible that I was mistaken in my assessment of this algorithm's accuracy. Thus this algorithm may merit a second, more rigorous evaluation,

especially if it is seen to be faster and less prone to error than the methods more recently tried.

0.3 Newton-Raphson Algorithm for Parameter Estimation

0.3.1 Introduction & Description of Method

Regardless of the success or failure of the Song method, it inspired me to try the Newton-Raphson method in a different way. I recalled that in the papers of Almpanidis and Shin & Chang the stated goal of the gradient descent algorithm was to find the zeros of the following equation:

$$F_n(\gamma) = \tag{35}$$
$$\psi_0\left(\frac{\frac{1}{N}\sum i = 1^N |x_i|^\gamma}{\frac{1}{N^2}\sum_{i=1}^N \sum_{j=1}^N |x_i|^\gamma \log \frac{|x_i|^\gamma}{|x_j|^\gamma}}\right)$$
$$+\log\left(\frac{1}{N^2}\sum_{i=1}^N \sum j = 1^N |x_i|^\gamma \log \frac{|x_i|^\gamma}{|x_j|^\gamma}\right)$$
$$-\frac{1}{N}\sum_{i=1}^N \log |x_i|^\gamma = 0 \tag{36}$$

We use the Newton-Raphson root-finding algorithm, which is:

$$\hat{\gamma}(k+1) = \hat{\gamma}(k) - \frac{F_n(\hat{\gamma}(k))}{\dot{F}_n(\hat{\gamma}(k))} \tag{37}$$

taking care to reject negative values of $\hat{\gamma}$ and to monitor its progress so that it reduces F_n. Shin & Chang note that the solution to (36) tends to diverge and cannot be obtained with approximately $N < 400$.

η and β are estimated by:

$$\psi_0(\hat{\eta}(k)) - \log \hat{\eta}(k) = S_2(k) - \log S_1(k) \tag{38}$$

$$\hat{\beta}(k) = \frac{\hat{\eta}(k)}{S_1(k)} \tag{39}$$

where the statistics in question are given by the following:

$$S_1 = (1/N)\sum_{i=1}^N |x_i|^{\hat{\gamma}} \tag{40}$$

$$S_2 = (1/N)\sum_{i=1}^N \log |x_i|^{\hat{\gamma}} \tag{41}$$

The left part of (38) is monotonically increasing, so it is simple to do a reverse lookup in a table to find η.

0.3.2 Implementation

I implemented this method in Matlab.
This method gave good results on synthetic data geerated using the inverse-CDF method in finding the correct values of the parameters. However it has the disadvantage of being so slow as to be unusable.

0.4 Newton-Hessian Method

0.4.1 Introduction

In talking with Nizar Bouguila about the previous method, he recommended that I use the Newton method with the Hessian matrix to solve for the three parameters η β, and γ simultaneously.
The first step in implementing this was to derive the second derivatives of the log-likelihood function. Dr. Bouguila and I did this independently and he later confirmed my results.

0.4.2 Theoretical Overview

As a reminder, the log-likelihood for the generalized gamma distribution is:

$$L = \log f_{\mathrm{x}}(\mathrm{x};\eta,\beta,\gamma) = \log\left(\frac{\gamma\beta^{\eta}}{2\Gamma(\eta)}\right) + (\eta\gamma - 1)\sum_{i=1}^{N}\log|x_i| - \beta\sum_{i=1}^{N}|x_i|^{\gamma}$$

We want to maximize this expression through a numerical solution.

Newton-Raphson method with Hessian matrix: Let $\theta = (\eta, \beta, \gamma)$. Apply the following recurrence relation:

$$\theta_{k+1} = \theta_k - \frac{\partial L}{\partial \theta_k}\left(\frac{\partial^2 L}{\partial^2 \theta}\right)^{-1}$$

Gradient: Vector of first derivatives with respect to each parameter in θ.
Hessian: Matrix of second derivatives with respect to each parameter in θ.

$$\frac{\partial L}{\partial \eta} = N(\log\beta - \psi_0(\eta)) + \gamma\sum_{i=1}^{N}\log|x_i|$$

$$\frac{\partial L}{\partial \beta} = N\frac{\eta}{\beta} - \sum_{i=1}^{N}|x_i|^{\gamma}$$

$$\frac{\partial L}{\partial \gamma} = N\frac{N}{\gamma} + \eta\sum_{i=1}^{N}\log|x_i| - \beta\sum_{i=1}^{N}|x_i|^{\gamma}\log|x_i|$$

$$\frac{\partial^2 L}{\partial \eta^2} = -N\psi^{(1)}(\eta) \qquad \frac{\partial^2 L}{\partial \beta^2} = -\frac{N\eta}{\beta^2}$$
$$\frac{\partial^2 L}{\partial \eta \partial \beta} = \frac{N}{\beta} \qquad \frac{\partial^2 L}{\partial \beta \partial \gamma} = -\sum_{i=1}^{N} |x_i|^N \log |x_i|$$
$$\frac{\partial^2 L}{\partial \eta \partial \gamma} = \sum_{i=1}^{N} \log |x_i| \quad \frac{\partial^2 L}{\partial \gamma^2} = -\frac{N}{\gamma^2} - \beta \sum |x_i|^\gamma (\log |x_i|)^2 \tag{42}$$

0.4.3 Implementation & Practical efforts

The basic method was implemented in Matlab. This method worked quite well on synthetic data generated using the inverse-CDF method. It was relatively fast and accurate. However, numerical problems were encountered when applying the method to real speech data feature vectors and even when applying it to concatenated series of synthetic feature vectors generated by inverse-CDF. The latter is explained as follows:

Generation of Synthetic Feature Data

Our speech feature vectors were of dimension 10 (10 DCT coefficients.) We had analysis window shifts of 10 ms and statistical windows of 500ms to 2s, giving us 50 to 200 feature vectors fo each statistical window and half that for each subwindow. For an experiment, we generated 30 GΓ Distributions. Starting with three base distributions - Gaussian, Gamma, and Laplacian, we generated 10 distributions from each base distribuion by perturbing the parameters with a random uniform number of appropriate scale. Then we generate a vector of random numbers drawn from each of the 30 distributions. This vector is of length approximately 3 times the length of a subwindow, or approximately 750ms to 3 s ==¿ 75 features to 300 features. There are ten such vectors for each base distribution, so we end up with three matrices of size 10x75 to 10x300, each drawn from random perturbations of the base distributions. In other words, we have one matrix corresponding to 10 variations of the Gaussian, another with 10 variations of the Gamma, and another with 10 variations of the Laplacian.

Based on these synthetic data, we perform the fitting of 10 GΓDs and find the log-likelihoods and GLRT distances.

Numerical Problems

It was in doing this that we ran into numerical problems. Namely, the local estimate of the parameters would occasionally tend to having two of the parameters of very large scale (¿50) with the other parameter approaching zero. This caused the Hessian matrix to become singular and the log-likelihood to become infinite in scale (probably an artifact of the hugely inflated parameters.)

Another numerical problem encountered was that, in the course of the iterations of the Newton algorithm, sometimes taking the full Newton step led to

a negative η β, or γ. The log likelihood function cannot handle negative data or negative parameters.

Another problem is that sometimes, especially with small sample sizes encountered when using small subwindows, the algorithm was ill-behaved, wherein taking the full Newton step did not increase the log-likelihood but rather decreased it.

These three problemes were tackled using an implementation of backtracking as I understand it from the book "Numerical Recipes".

Outline of Newton-Hessian Algorithm as Implemented in Matlab

The algorithm is written as a Matlab function. It accepts as input the data and three control parameters. The first control parameter, λ, controls the scale of a term used to determine whether to run backtracking in a given iteration. The second, ρ, is used to control the scale of a measure which determines whether to stop the algorithm. The third, d, was once used as a control to stop the algorithm.

The Newton method is run within a while loop. The conditions for exiting the loop are the following:
(1) $\sqrt[2]{|\text{diff} \cdot \text{diff}|} < 1 \times 10^{-6}$
or
(2) # iterations > maxits

The program starts with parameters η β, and γ set to 1, 1, and 1, respectively. The algorithm takes the full Newton step computed as $\alpha \left(\frac{\partial^2 L}{\partial^2 \theta}\right)^{-1} \frac{\partial L}{\partial \theta_k}$, where by default α is set to 1 at the beginning of each loop/iteration.
Next, the program goes into a while loop which checks if any of the parameters has gone negative with the step taken. If so, it records the current value of α as α_0, and applies $\alpha = \alpha - \alpha_0/10$. It then re-estimates the parameters and if any is still negative, repeats until either the parameters are all positive or it reaches a maximum-loop-iteration counter. This limit is usually set at 25.

The next backtracking routine is as follows.

1. The program computes the following: $term = \lambda \frac{\partial L}{\partial \theta_k}(\theta_{k+1} - \theta_k)$ Originally, I had set this term as $\frac{\partial L}{\partial \theta_0}$, where this is the initial gradient on the first step of the algorithm. However this did not seem to work as well so I changed it to the current form.
2. $\alpha_0 = \alpha$
3. The program enters into a while loop. The loop runs while the updated log-likelihood J_{k+1} is less than or equal to $J_k - term$, or while J_{k+1} is infinite or NaN. The loop terminates when these conditions are no longer satsified or when it reaches a limit on a loop counter. While the conditions of the loop are active, it repeats much the same procedure as the backtracking routine used to prevent the parameters from becoming negative: $\alpha = \alpha - \alpha_0/10$. Then we compute temporary parameters according

to: $\theta_{\text{temp}} = \theta_k - \alpha \left(\frac{\partial^2 L}{\partial^2 \theta} \right)^{-1} \frac{\partial L}{\partial \theta_k}$ and compute a temporary updated log-likelihood J_{temp}. If J_{temp} is in fact less than the previous estimated log likelihood taking the full Newton step (i.e. J_{k+1},) or if J_{temp} is infinite or NaN, or if any temporary parameter is less than zero, we break out of the while loop and discard the temporary parameters and temporary updated log-likelihood. Otherwise, $J_{k+1} = J_{\text{temp}}$ and $\theta_{k+1} = \theta_{\text{temp}}$ (the updated log-likelihood and parameters get overrwitten with the new values taking a smaller step.)

We repeat until $J_{k+1} > J_k - term$ or J_{k+1} is not infinite or NaN.

Previously, these backtracking routines did not subtract a fixed fraction of α, but rather divided α by 2. This, however, could not resolve some cases. One backtracking routine which used this method and is no longer used in the present form of the NR-Hessian algorithm worked as follows:

1. We check if the updated likelihood J_{k+1} is less than J_k and if, at the same time, $J_k <= J_{k-1}$.
2. If so, we apply $\alpha = \alpha/2$.
3. $\theta_{\text{temp}} = \theta_{k-1} - \alpha \left(\frac{\partial^2 L}{\partial^2 \theta} \right)^{-1} \frac{\partial L}{\partial \theta_{k-1}}$
4. Re-check the first condition and if true repeat steps 2 and 3.

After the loop completes, we apply the following

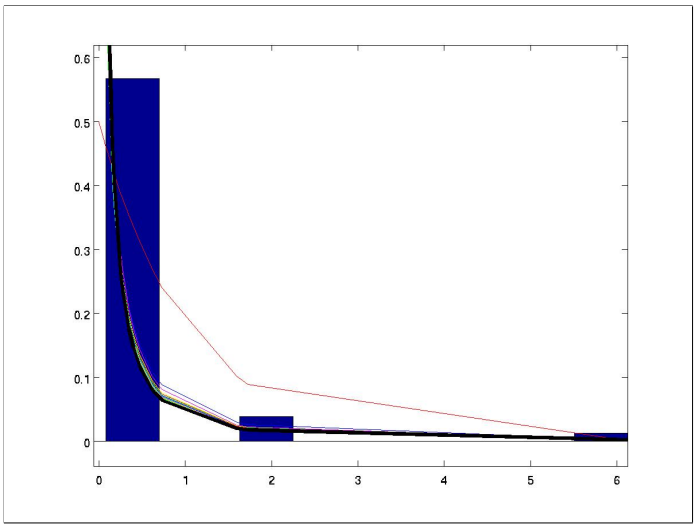
1. $\theta_k = \theta_{\text{temp}}$.
   Note that in this step we step back one step and revise θ_k by taking a fractional Newton step from θ_{k-1}.
2. We revise J_k accordingly
3. Erase θ_{k+1} and J_{k+1}
4. *Decrement* k by 1.

After much use and modification, and more importantly after modification to the other backtracking routines, I felt this routine was unneccessary and it has been commented-out in recent revisions of the NR-Hessian algorithm.

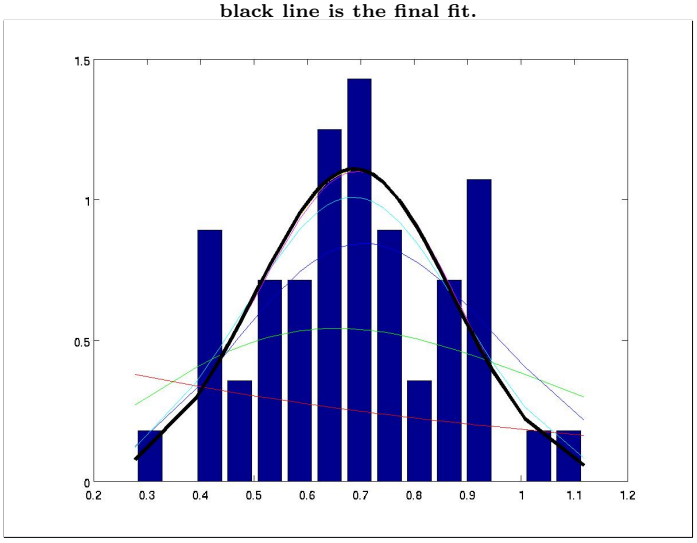
0.4.4 Conclusion

I have still not been able to force this algorithm to work perfectly with the speech data. Nizar Bouguila suggested the use of the **gammaln** Matlab function rather than taking the logarithm of **gamma(η)** in the log-likelihood expression. This was supposed to help avoid infinite log likelihoods, but it doesn't seem to have helped much. The best I can do to prevent singular Hessian matrices and parameters and log-likelihoods which go to infinity is to emplace a condition which tells the **while** loop to quit if the magnitude of the Hessian becomes too great. However any such limit may prove too high depending on the kind of data presented.

:

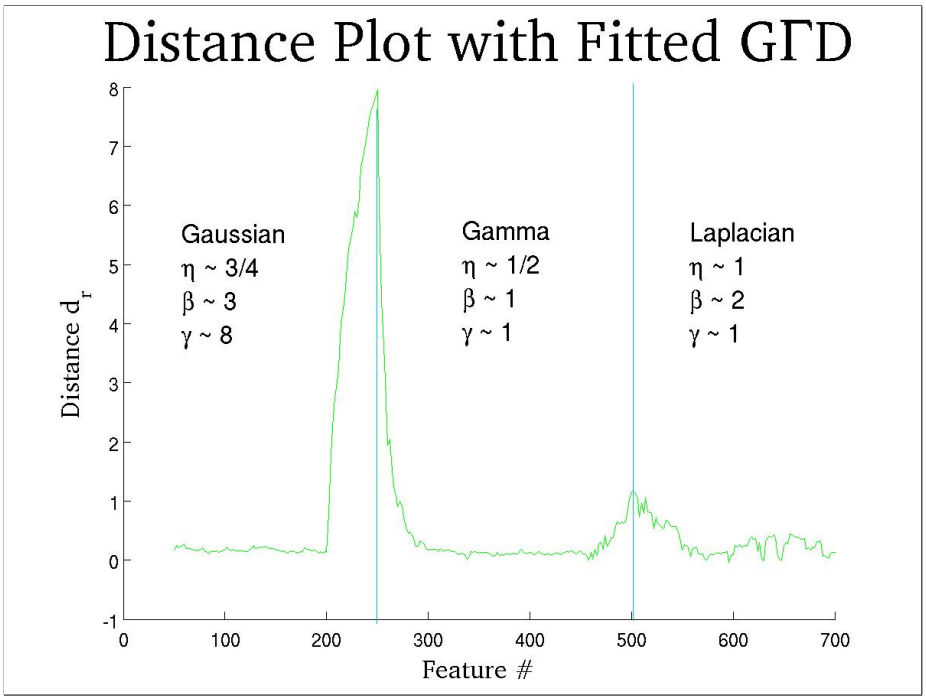


An example of the kind of data with which this algorithm has trouble. The blue bars represent the empirical pdf. The colored lines are successive fits of the $G\Gamma D$. The black line is the final fit.



An example of the kind of data with which this algorithm excels

Numerical problems aside, the algorithm performs well on synthetic data drawn from idealized GΓD distributions when the window size is rather large (not on the order of what would be used for phoneme segmentation.)



GLRT distance computed from log-likelihoods of GΓD's fitted by the NR-Hessian method. Statistical window size in terms of number of features is large.

0.5 Matlab Optimization Toolbox Methods

Several Matlab Optimization Toolbox methods were tried. These are divided into variations on the use of **fminunc** and the use of **fminsearch**. **fminsearch** is a non-derivative based secant method, whereas **fminunc** uses either a user-supplied analytical or finite-difference numerical gradient with the option of a user-supplied analytical or finite-difference numerical Hessian. **fminunc** is divided into two modes of operation: Medium-Scale and Large Scale. Medium-Scale uses a Quasi-Newton method, whereas Large Scale uses a Trust-Region Newton method.

0.5.1 Perfomance

While these methods all perform equivalently in terms of accuracy on synthetic data generated from idealized GΓD's with a large number of samples, we once again encounter numerical problems with the Medium-Scale **fminunc** and with **fminsearch.** We encounter two very familiar scenarios:

1. the algorithm leads the parameters to be negative
2. the algorithm leads the parameters to be very large resulting in infinite Log-Likelihood

The first problem is largely resolved through a fix in the routine which estimates the Log-likelihood. This adds a small increment to the parameters until

they are non-negative and returns the log-likelihood of those "fudged" parameters.

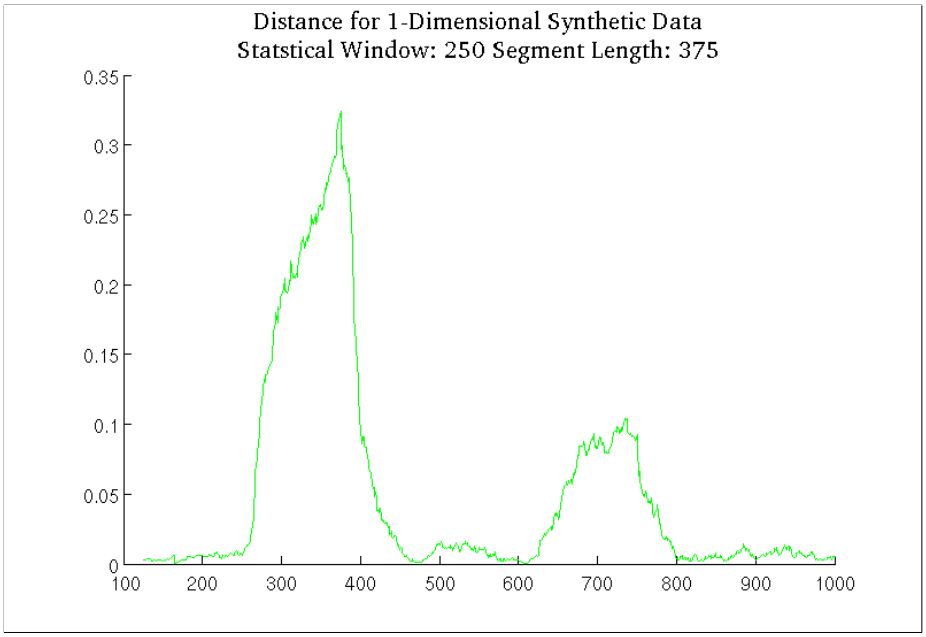
The second problem is not easily tractable. We can try to avoid it by limiting the number of function evaluations and by raising the tolerance which ends the loop. This will, however, theoretically result in a loss of accuracy.

The large-scale method performs approximately 15% slower than the medium-scale method.

0.5.2 Results with the Matlab Toolbox Methods

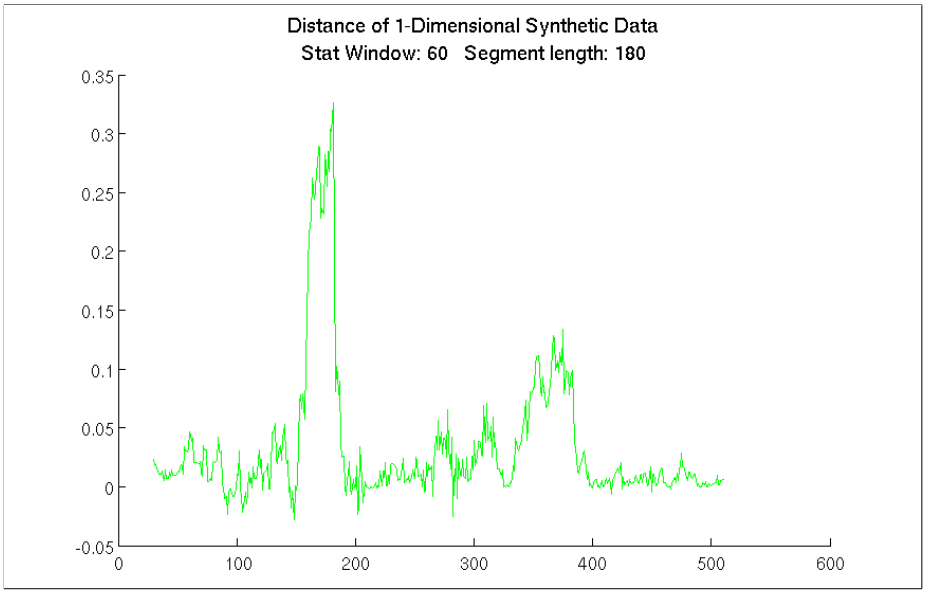
Synthetic Data

We tested the algorithms on the same synthetic data setup used before. We started with large window sizes applied to unidimensional data.

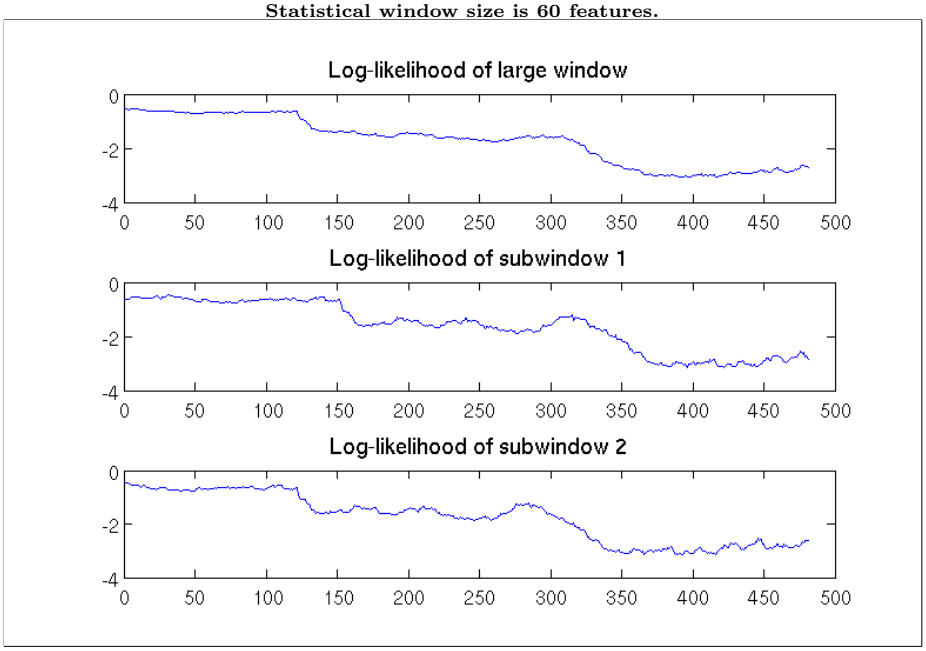


GLRT distance computed from log-likelihoods of GΓD's fitted by the *fminunc* method. Statistical window size is 275 features.

Next we moved to smaller statistical window size and smaller segment lengths.

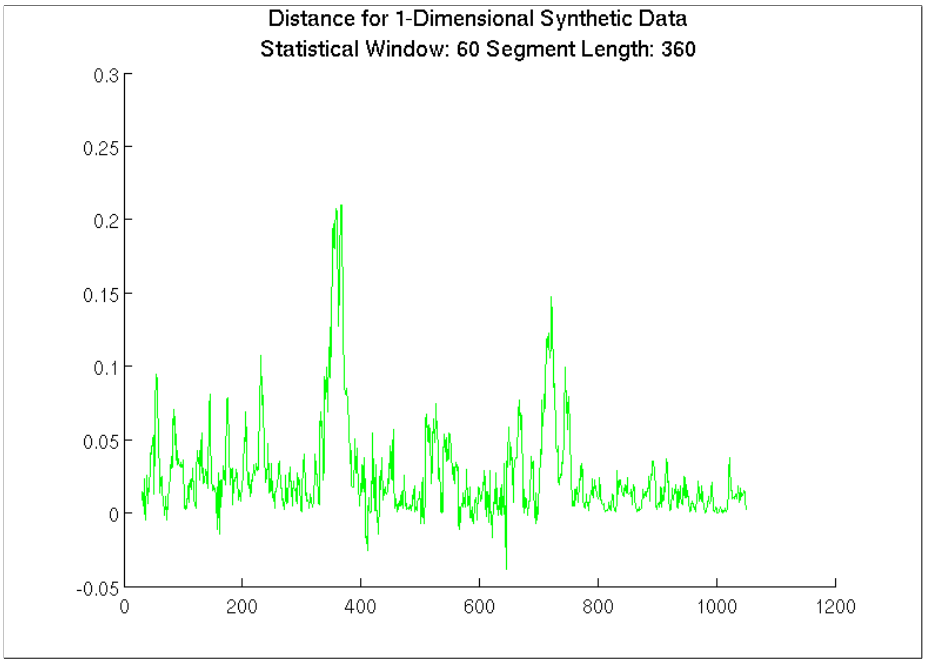


GLRT distance computed from log-likelihoods of GΓD's fitted by the *fminunc* method. Statistical window size is 60 features.



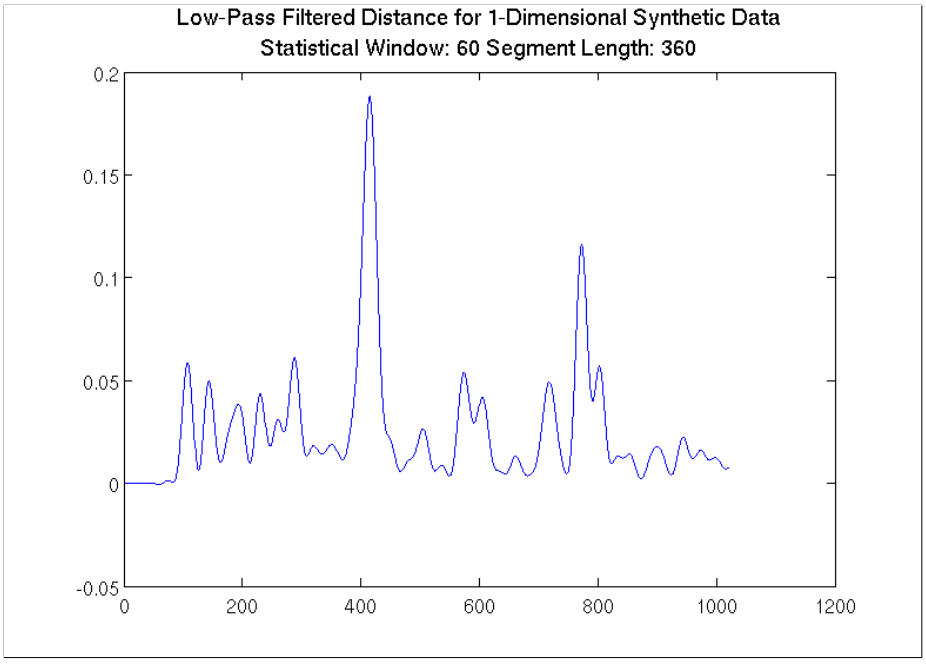
Log likelihoods of the fitted GΓD's for the large window and the two subwindows. Notice the delay in variation between the subwindow 1 and subwindow 2.

Then we move to larger segment sizes relative to the window size.



GLRT distance computed from log-likelihoods of GΓD's fitted by the *fminunc* method. Statistical window size is 60 features.

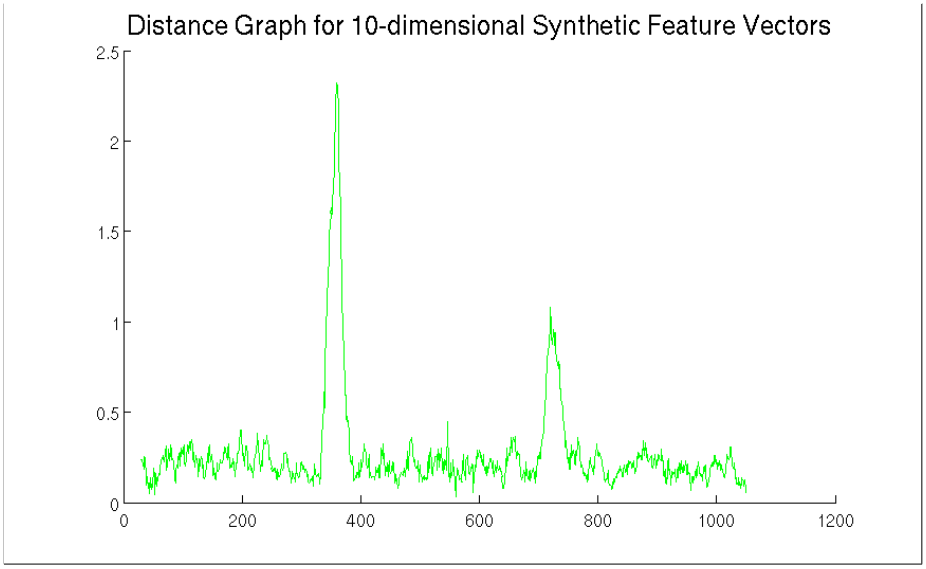
We also applied a low-pass filter for this step.



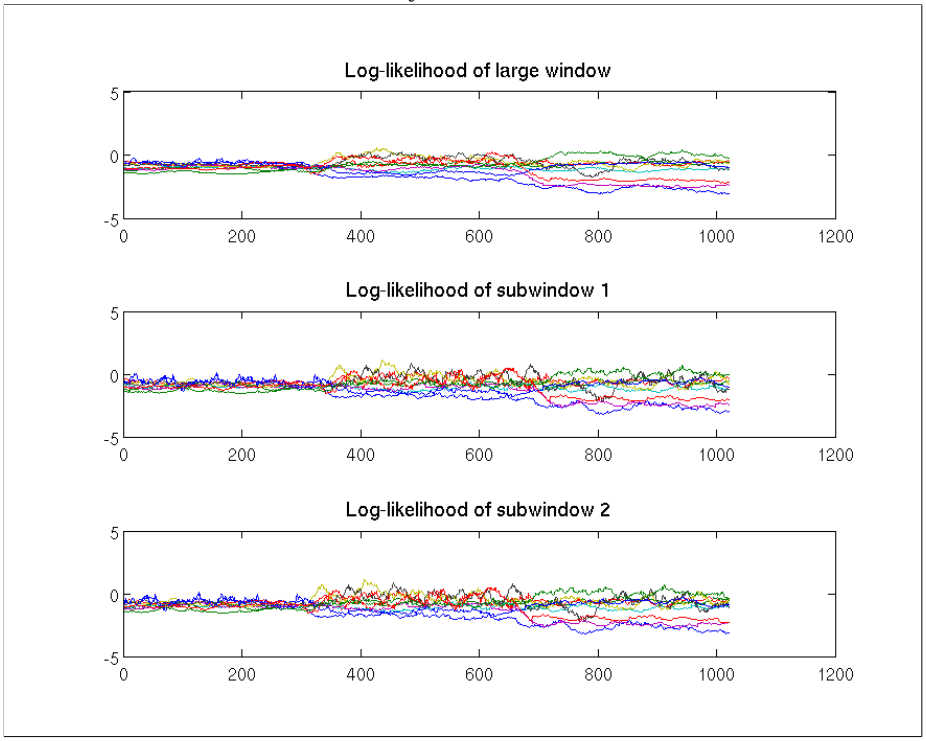
Low-pass filtered GLRT distance. Statistical window size is 60 features. Segment length is 360 features.

When using the Large-Scale method, we are able to obtain fits for the Generalized Γ even with window sizes as small as 30 feature vectors.

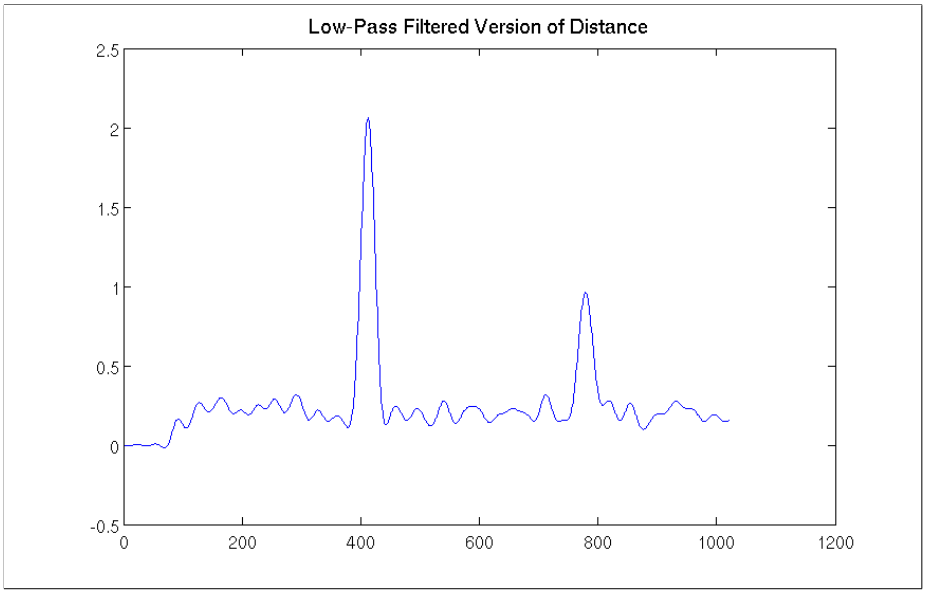
We then introduce 10-dimensional data to mirror the setup of speech features.



GLRT distance computed from log-likelihoods of GΓD's fitted by the *fminunc* method to synthetic data.



Log likelihoods of the fitted GΓD's for the large window and the two subwindows.



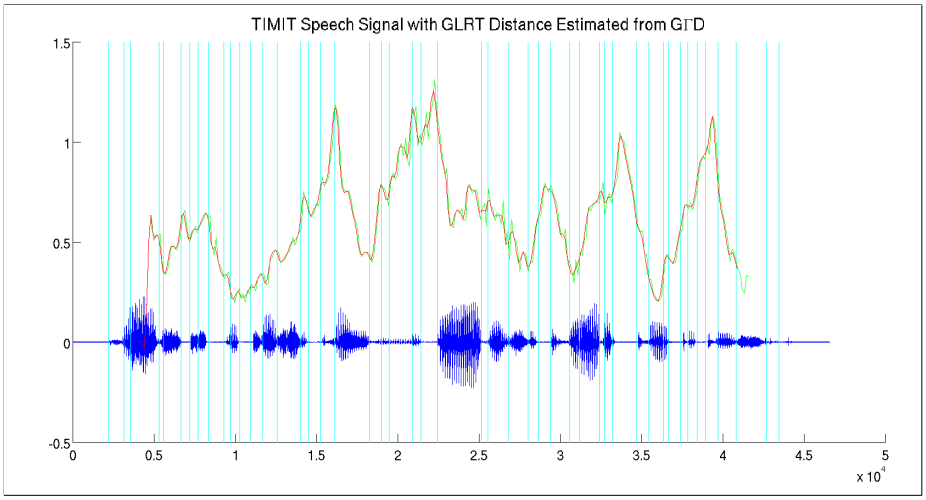
Low-pass filtered version of GLRT distance for 10-dimensional feature synthetic data.

The increase in the number of dimensions seems to increase the signal-to-noise ratio of the distance graph.

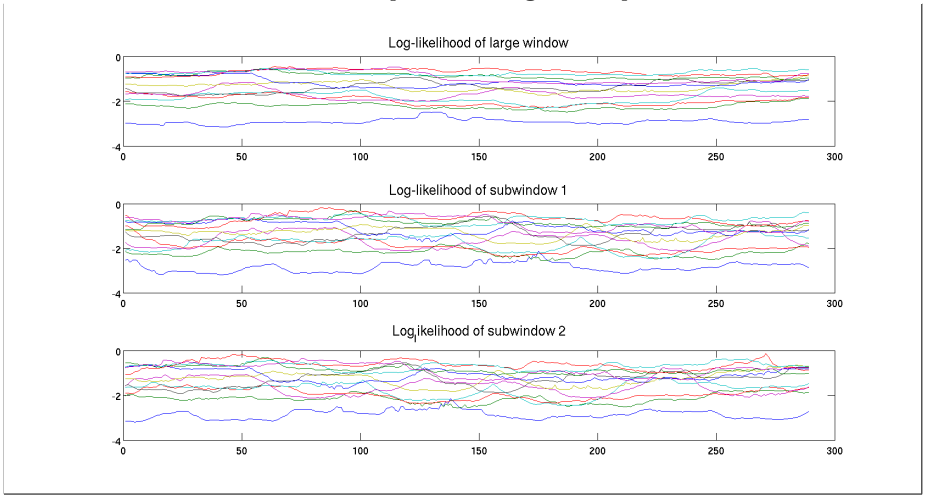
The configuration of settings we settled on is as follows. We use the Large-Scale algorithm and supply the gradient as a function, but not the Hessian matrix. We reduce the maxiumum number of function evaluations from 300 to 150, increse the Function Tolerance ("Function" here meaning both the objective function of the log-likelihood and the first-order optimality) to 2e-3, and increase the tolerance on X (the parameters of the GΓD being estimated) to 5e-3. We take these last three steps in an effort to reduce the time taken to fit a distribution to the data.

Speech Data

I have applied the method to speech data to obtain the GLRT distances. On one sample of speech, the result is the following distance graph.



GLRT distance computed from log-likelihoods of GΓD's fitted by the *fminunc* method to speech data. The blue curve is the speech signal. The cyan lines are the manually transcribed phoneme boundaries. The green line is the GLRT distance. The red line is the same distance passed through a low-pass filter.



Log likelihoods of the fitted GΓD's for the large window and the two subwindows for speech data.