INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS CIUDAD DE MÉXICO

ESCUELA DE INGENIERÍA Y ARQUITECTURA

DEPARTAMENTO DE INGENIERÍA MECATRÓNICA

TOOLKIT DE PROCESAMIENTO DE VOZ PARA LABVIEW™

ROBERTO GARZA LAU

ASESOR: DR. ALFREDO VÍCTOR MANTILLA CAEIROS

México, D.F. Noviembre 2009

Instituto Tecnológico de Estudios Superiores Monterrey, Campus Ciudad de México.

Toolkit de procesamiento de voz para LabVIEW

Reporte Final.

Roberto Garza Lau A01122905
Noviembre 2009

Tabla de contenidos

Tabla de ilustraciones

Capítulo 1: Introducción

1.1. *El ser humano y el análisis digital de voz*

El ser humano ha desarrollado diferentes formas de comunicación a lo largo de la historia, una de las más comunes es la oral. Ésta utiliza la voz para generar sonidos que en conjunto crean palabras, frases, expresiones y oraciones.

La voz humana es producida en la laringe, cuya parte esencial, la glotis, constituye el verdadero órgano de fonación humano. El aire procedente de los pulmones es forzado durante la espiración a través de la glotis, haciendo vibrar los dos pares de cuerdas vocales, que se asemejan a dos lengüetas dobles membranáceas. Las cavidades de la cabeza, relacionadas con el sistema respiratorio y nasofaríngeo, actúan como resonadores.

El aparato de fonación puede ser controlado conscientemente por quien habla. La variación de la intensidad depende de la fuerza de la espiración. En el hombre las cuerdas vocales son más largas y más gruesas que en la mujer y el niño, por lo que produce sonidos más graves. (GA., 2003)

1.2. ¿Qué es LabVIEW?

LabVIEW es una herramienta gráfica para pruebas, control y diseño mediante la programación. El lenguaje que usa se llama lenguaje G, donde la G simboliza que es lenguaje Gráfico.

Este programa fue creado por National Instruments (1976) para funcionar sobre máquinas MAC y salió al mercado por primera vez en 1986. Ahora está disponible para las plataformas Windows, UNIX, Mac y Linux. La versión actual 8.6, liberada en Agosto de 2008, cuenta con soporte para Windows Vista.

Los programas desarrollados con *LabVIEW* se llaman Instrumentos Virtuales, o VIs, lo que da una idea de su uso en origen: el control de instrumentos. El lema de *LabVIEW* es: "La potencia está en el Software". Entre sus objetivos están el reducir el tiempo de desarrollo de aplicaciones de todo tipo (no sólo en ámbitos de Pruebas, Control y Diseño) y el permitir la entrada a la informática a programadores no expertos. Esto no significa que la empresa haga únicamente software, sino que busca combinar este software con todo tipo de hardware, tanto propio (tarjetas de adquisición de datos, PAC, Visión, y otro Hardware) como de terceras empresas. (National Instruments Corporation, 2009)

1.3. Problemática

Durante años el ser humano ha investigado y desarrollado diferentes técnicas para analizar, caracterizar, reconocer y sintetizar los diferentes patrones de voz. Todas estas técnicas son

cálculos matemáticos que pueden ser aplicados a diferentes dispositivos tales como computadoras, procesadores de aplicaciones específicas (DSP's), microcontroladores, entre otros.

Desgraciadamente, *LabVIEW* carece de herramientas para el procesamiento, análisis, clasificación y síntesis de voz. Y es precisamente en esta área donde se ha encontrado un espacio para el desarrollo de este proyecto. (National Instruments-Products and Services, 2009)

1.4. Objetivos

1.4.1. Objetivo general

Crear un conjunto de herramientas para conformar un *toolkit* de procesamiento de voz para *LabVIEW* que facilite la implementación de las diferentes fases del procesamiento de voz.

1.4.2. Objetivos específicos

1. Generar VI's sencillas y claras colocando el nombre de las entradas y salidas a la misma de modo que sea lo más descriptiva posible.
2. Obtener un excelente rendimiento para cualquier computadora donde pueda ejecutarse *LabVIEW* no excediendo los requerimientos mínimos de este programa.
3. Generar un *toolkit* que sea independiente en cierta medida, es decir, que no requiera de gran cantidad de elementos de *LabVIEW* que no se incluyan en el paquete base.
4. Las VI's desarrolladas podrán ser descargadas a sistemas de hardware ajenos a la computadora persona como tarjetas de desarrollo y sistemas basados en DSPs.

1.5. Justificación

El análisis de voz es utilizado en diferentes áreas como criminalística, automatización, seguridad, salud y más.

Al momento, *LabVIEW* no cuenta con paquete alguno que pueda simular de manera sencilla el procesamiento de voz, de modo que al realizar este proyecto se innovará en este aspecto. (National Instruments-Products and Services, 2009)

Con este compendio de herramientas se facilitará la simulación en el ambiente *LabVIEW* de sistemas que requieran alguna de las fases del procesamiento de voz, ya que contiene VI's preparadas para hacer las operaciones comunes del procesamiento de voz sin necesidad de conocimientos específicos de algoritmos por parte del usuario.

Capítulo 2: Marco teórico

2.1. Adquisición

Para comenzar a analizar la voz es necesario digitalizarla y almacenarla en un formato compatible con el ambiente a utilizar.

El muestreo digital es una de las partes que intervienen en la digitalización de las señales. Consiste en tomar muestras periódicas de la amplitud de una señal analógica, siendo el intervalo entre las muestras constante. El ritmo de este muestreo se denomina frecuencia o tasa de muestreo y determina el número de muestras que se toman en un intervalo de tiempo.

El muestreo está basado en el Teorema de Muestras, que es la base de la representación discreta de una señal continua en banda limitada. El muestreo de la señal implica pérdida de información respecto a la señal de entrada, ya que de un número infinito de valores posibles para la entrada sólo tenemos un valor finito de valores posibles para la salida. Por tanto es fundamental saber cuántas muestras hemos de tomar. (Babylon.com, 2007)

Esto depende del error medio admisible, el método de reconstrucción de la señal y el uso final de los datos de la conversión.

Independientemente del uso final, el error total de las muestras será igual al error total del sistema de adquisición y conversión más los errores añadidos por el ordenador o cualquier sistema digital.

2.2. Filtrado

El filtrado de la señal de voz adquirida es una etapa fundamental en el acondicionamiento de la señal. El objetivo del filtrado es la eliminación de ruido; uno de los problemas más importantes a resolver es la presencia de ruido de 60Hz en las grabaciones.

2.2.1. Filtros de pre-énfasis

Estos filtros tienen como objetivo resaltar determinadas componentes de frecuencia relevantes para un determinado procesamiento. En estos casos los filtros de pre-énfasis tienden a "blanquear" el espectro y hacerlo más plano removiendo componentes espectrales no deseadas. El factor de preénfasis se calcula como se muestra en la ecuación 1. (ppgb, 2003)

$$\alpha = e^{-2\pi F \Delta T}$$

Ecuación 1

donde Δt es el periodo de muestreo del sonido y F es la frecuencia por encima de la cual el espectro será aumentado 6dB/octava.

La nueva señal se calcula como se muestra en la ecuación 2:

$$y_i = x_i - \alpha x_{i-1}$$

Ecuación 2

2.2.2. Filtros de de-énfasis

El objetivo de estos filtros es contrarrestar el efecto del filtro de pree-énfasis, de modo que la señal regresa a la normalidad después de hacer sido procesada. (ppgb, 2003)

Para realizar este filtrado se utiliza la misma α que en el pre-énfasis pero ahora como se muestra en las ecuaciones 3 y 4:

$$y_1 = x_1$$

Ecuación 3

$$y_i = x_i + \alpha y_{i-1}$$

Ecuación 4

2.3. Segmentación

El proceso de segmentación de la señal de voz a analizar constituye una etapa necesaria en cualquier sistema de procesamiento de voz. La razón de la segmentación radica en el carácter no estacionario de la voz. Se puede considerar que en tramos pequeños que oscilan entre 10 y 30ms la señal de voz puede considerarse estacionaria o cuasi-estacionaria lo cual permite que todo el análisis matemático aplicable a señales estacionarias pueda ser utilizado para la voz en estos intervalos. (Mantilla, 2007)

2.4. Ventaneo

Una práctica común en los sistemas de acondicionamiento de la señal de voz es hacer pasar la señal de cada segmento por una función ventana. Este proceso llamado ventaneo permite eliminar errores debido a discontinuidades generadas en la segmentación provocadas por las transiciones de un segmento a otro. El ventaneo permite que las muestras en el centro de la ventana tengan un mayor peso que las que se encuentran en los bordes o transiciones. Las ventanas son funciones de cambios o transiciones suaves que se convolucionan con la señal original.

La duración de la ventana se elige generalmente mayor al tiempo de trama para prevenir la pérdida de información en las transiciones de una secuencia a la siguiente. Esto resuelve el problema de las transiciones entre tramas sin necesidad de realizar traslapes entre las mismas.

2.5. Coeficientes de predicción lineal

El método de predicción lineal toma como base un modelo del tracto vocal representado como un filtro (usualmente lineal) variable en el tiempo. Según este modelo se distinguen dos elementos separados en la producción de voz: la excitación y el tracto vocal. La onda de voz es

resultado de la convolución entre la excitación y el filtro (tracto vocal). Existen diversos métodos para lograr la separación fuente-filtro, uno de ellos es la predicción lineal.

2.5.1. Cálculo de LPC's

El método de predicción lineal caracteriza la forma del espectro de un segmento de voz con un número reducido de parámetros que permiten una codificación eficiente. La Codificación Lineal Predictiva, como también se llama a este método, predice una señal en el dominio del tiempo con base en una combinación de muestras previas, linealmente distribuidas como se muestra en la ecuación 5.

$$s'_n = \sum_{k=1}^{p} a_k s_{n-k}$$

Ecuación 5

Donde a_k (1<k<p) es un conjunto de constantes reales conocidas como coeficientes predictores, que necesitan ser calculados y p es el orden del predictor. Una de las ventajas de este método es que además de ser muy preciso, es muy adecuado para la implementación computacional, pues es sencillo y de rápida ejecución.

El error entre el valor real de la función y la función aproximada esta dado por la ecuación 6.

$$e_n = s_n - s'_n = s_n - \sum_{k=1}^{p} a_k s_{n-k}$$

Ecuación 6

El error total cuadrático es el que se muestra en la ecuación 7.

$$E = \sum_n e_n^2 = \sum_n (s_n s'_n)^2$$

Ecuación 7

El problema de la predicción lineal radica en encontrar los coeficientes predictores a_k que minimicen el error e_n. La condición para la minimización del error total cuadrático se obtiene estableciendo la derivada parcial del error total cuadrático E, con respecto a cada uno de los coeficientes predictores a_k, a cero. El resultado es el mostrado en la ecuación 8.

$$\sum_{k=1}^{p} a_k R_{i-k} = R_i$$

Ecuación 8

Donde R_i es la función de autocorrelación de la señal S_n. Por lo tanto para poder minimizar el error total cuadrático es necesario calcular los coeficientes de autocorrelación y después resolver la ecuación 8 para obtener los coeficientes predictores. La ecuación 8 es una ecuación matricial, existen dos métodos fundamentales de solución que son los de autocorrelación y covarianza.

En el método de autocorrelación se utilizan para la estimación únicamente los puntos dentro de la ventana definida para la señal. Se definen los límites de n suponiendo que el segmento de voz es cero fuera del intervalo $0 \leq n \leq N + p - 1$. De la solución por medio de mínimos cuadrados, resultan las siguientes ecuaciones (ecuaciones de Yule-Walker) expresadas en forma de matriz, es decir, la matriz de autocorrelación mostrada en la ecuación 9.

$$\begin{bmatrix} r(0) & \cdots & r(p-1) \\ \vdots & \ddots & \vdots \\ r(p-1) & \cdots & r(0) \end{bmatrix} \begin{bmatrix} r(1) \\ \vdots \\ r(p) \end{bmatrix} = \begin{bmatrix} r(1) \\ \vdots \\ r(p) \end{bmatrix}$$

Ecuación 9

Esta matriz tiene la estructura Toeplitz (es simétrica y los elementos de sus diagonales son iguales), lo que permite la aplicación de la recursión de Levinson-Durbin para resolver el sistema.

La recursión de Levinson-Durbin es un algoritmo que permite encontrar un filtro IIR a partir de una secuencia determinística de autocorrelación dada. El filtro producido por la recursión de Levinson-Durbin es de fase mínima.

El algoritmo de Levinson-Durbin encuentra de forma recursiva los coeficientes para el filtro de primer orden, que se denota por a_1^1 , donde el superíndice denota el orden del filtro, y el subíndice, el número del coeficiente para ese orden. Enseguida, el algoritmo encuentra los coeficientes para el filtro de segundo orden a_1^2 y a_2^2, y así sucesivamente hasta encontrar los coeficientes del filtro de orden p, a_1^p , a_2^p ,..., a_p^p.

Los coeficientes de predicción lineal (LPC's) constituyen uno de los métodos más utilizados en la extracción de parámetros de voz a partir de la información del tracto vocal. Las referencias consultadas utilizan los LPC's tanto en la caracterización de segmentos de voz como en algoritmos de compresión. (Childers, 1999)

2.5.2. Algoritmo de Levinson-Durbin

Para poder realizar el cálculo de los LPC's es necesario conocer el algoritmo recursivo de Levinson-Durbin. Se describe de la siguiente forma (ecuaciones 10 a 14):

Inicio de la recursión

$$\hat{a}_1^1 = -\frac{\hat{R}_{XX}(1)}{\hat{R}_{XX}(0)}$$

Ecuación 10

$$[E_T^2]^1 = (1 - (\hat{a}_1^1)^2)\hat{R}_{XX}(0)$$

Ecuación 11

Se incrementa el orden a *i=2*, recursión de Levinson.

$$\hat{a}_i^i = -\frac{\hat{R}_{XX}(i) + \sum_{j=1}^{i-1} \hat{a}_j^{i-1}\, \hat{R}_{XX}(i-j)}{(E_T^2)^{i-1}}$$

Ecuación 12

$$\hat{a}_j^i = \hat{a}_j^{i-1} + \hat{a}_i^i \hat{a}_{i-1}^{i-j}$$

Ecuación 13

$$(E_T^2)^i = \left(1 - \left(\hat{a}_i^i\right)^2\right)(E_T^2)^{i-1} \text{ para } j = 1, 2, \ldots, i-1$$

Ecuación 14

Se incrementa el orden en uno.

Si *orden = p+1*, termina. (Childers, 1999)

2.6. Formantes.

Son las frecuencias de resonancia del tracto vocal, es decir, frecuencias donde las armónicas tienen mayores amplitudes.

2.6.1. Extracción de formantes.

Los formantes son los picos de la envolvente del espectro de la señal de voz que representan las frecuencias de resonancia del tracto vocal. Las frecuencias a las que se producen los primeros formantes son muy utilizadas en los sistemas de caracterización de la voz. Las frecuencias formantes pueden variar de una persona a otra por edad, sexo y otros factores pero en sentido general se encuentran dentro de un rango establecido.

Existen varios métodos para el cálculo de formantes de la señal de voz, uno de los más utilizados es a partir del filtro LPC explicado en el capítulo anterior. Si el espectro de una señal de voz puede ser aproximado únicamente por sus polos, entonces los formantes pueden ser obtenidos de los polos del filtro LPC.

Los polos de H(z) pueden ser calculados igualando el denominador a "0" y encontrando las raíces. La conversión al plano S se realiza sustituyendo *z* por e^{skT}, donde s_k es el polo en el plano *s*. Las raíces resultantes en su mayoría son pares complejos conjugados. Por lo tanto, los formantes pueden obtenerse de los picos del espectro LPC. (Mantilla, 2007)

2.7. Coeficientes Cepstrales o Cepstrum

Otra herramienta frecuentada para el análisis de voz son los coeficientes Cepstrales, mismos que se definen como la transformada inversa de Fourier del logaritmo del módulo del espectro. Mostrado a continuación en la ecuación 15.

$$c(t) = F^{-1}[\log(S(\omega))]$$

Ecuación 15

Desarrollando se obtiene:

$$c(t) = F^{-1}\{[\log(E(\omega))] + [\log(H(\omega))]\}$$

Ecuación 16

Esta ecuación muestra que el Cepstrum de una señal es la suma del Cepstrum de la excitación y el del filtro del tracto vocal. Las componentes cepstrales bajas corresponden a variaciones lentas del espectro, por lo que contienen información de la envolvente del espectro que se relaciona precisamente con el filtro que modela el tracto vocal. Las componentes altas del Cepstrum están relacionadas con la fuente de excitación.

Para propósitos de reconocimiento de voz es importante la información del tracto vocal más que la fuente de excitación por lo que los primeros coeficientes cepstrales contienen información importante para extracción y características de parámetros de voz. Las referencias consultadas utilizan los 10 ó 12 primeros coeficientes cepstrales sobre ventanas de 20 a 30ms para reconocimiento de voz.

Los coeficientes cepstrales pueden ser obtenidos a partir de los LPCs por procedimientos recursivos que facilitan su implementación computacional. Uno de estos algoritmos se muestra a continuación, en las ecuaciones 17 a 19.

$$c_0 = ln\sigma^2$$

Ecuación 17

$$c_m = a_m + \sum_{k=1}^{m-1} \left(\frac{k}{m}\right) c_k a_{m-k} \qquad 1 \leq m \leq p$$

Ecuación 18

$$c_m = \sum_{k=1}^{m-1} \left(\frac{k}{m}\right) c_k a_{m-k} \qquad m > p$$

Ecuación 19

Donde σ^2 es la ganancia del filtro LPC. Generalmente se utiliza una relación para coeficientes Cepstrales de 3/2 sobre los coeficientes LPC. Los coeficientes Cepstrales suelen ser más robustos para reconocimiento de voz que los de predicción lineal. (The MathWorks, 2009)

2.8. Redes neuronales artificiales

Debido a que para la validación de este sistema se utilizaron redes neuronales, se procederá a proporcionar una breve introducción acerca de las mismas.

Las redes neuronales artificiales fueron inspiradas en la forma de procesar información de nuestro cerebro. En la actualidad son utilizadas frecuentemente en reconocimiento de patrones, clasificación, aproximación de funciones y ciertos tipos de predicciones.

Las redes neuronales están formadas de unidades llamadas neuronas, así como sus interconexiones, generalmente se utiliza una capa de entrada, una o más capas intermedias u ocultas y una capa de salida.

En la última etapa de la neurona se encuentra la función de activación, misma que tiene la finalidad de entregar una salida acotada en términos de las entradas y sus pesos. Entre estas funciones se encuentran el escalón unitario, lineal a tramos, sigmoidal, gaussiana y tangente hiperbólica.

Uno de los algoritmos de entrenamiento para redes neuronales es el de "propagación hacia atrás", éste es uno de los más utilizados y fáciles de entender. La función de los algoritmos de entrenamiento es encontrar los pesos y los umbrales adecuados para cada neurona que minimicen el error cometido por la red y así, la salida obtenida sea lo más cercana posible a la deseada. (Mantilla, 2007)

2.9. Desarrollo del toolkit

Para la planeación del desarrollo de este proyecto se creó un diagrama con las etapas fundamentales del procesamiento digital de voz y sus principales algoritmos (diagrama 1). Una vez realizado el diagrama se ordenaron los algoritmos de mayor a menor prioridad, siendo los de mayor prioridad aquellos algoritmos necesarios para el correcto funcionamiento de los algoritmos consecuentes. Finalmente se desecharon los algoritmos que componen alguna herramienta existente para LabVIEW, de modo que el proyecto no resultara redundante, ejemplos de lo anterior es ventaneo, algunos filtros, transformadas, entre otros. Se acordó la siguiente lista de herramientas a desarrollar:

1. Segmentación
2. Filtrado
    a. Filtro de pre-énfasis
    b. Filtro de de-énfasis
3. Calculo de Coeficientes de Predicción Lineal
4. Cálculo de Formantes

5. Cálculo de Coeficientes Cepstrales

La metodología utilizada para el desarrollo de este proyecto se basó en un trabajo individual para cada herramienta, el tiempo invertido en cada una dependería de la dificultad del algoritmo pero la condición siempre fue que las herramientas estuvieran correctamente validadas. De este modo se desarrolló el sistema propuesto detallado en el siguiente capítulo.

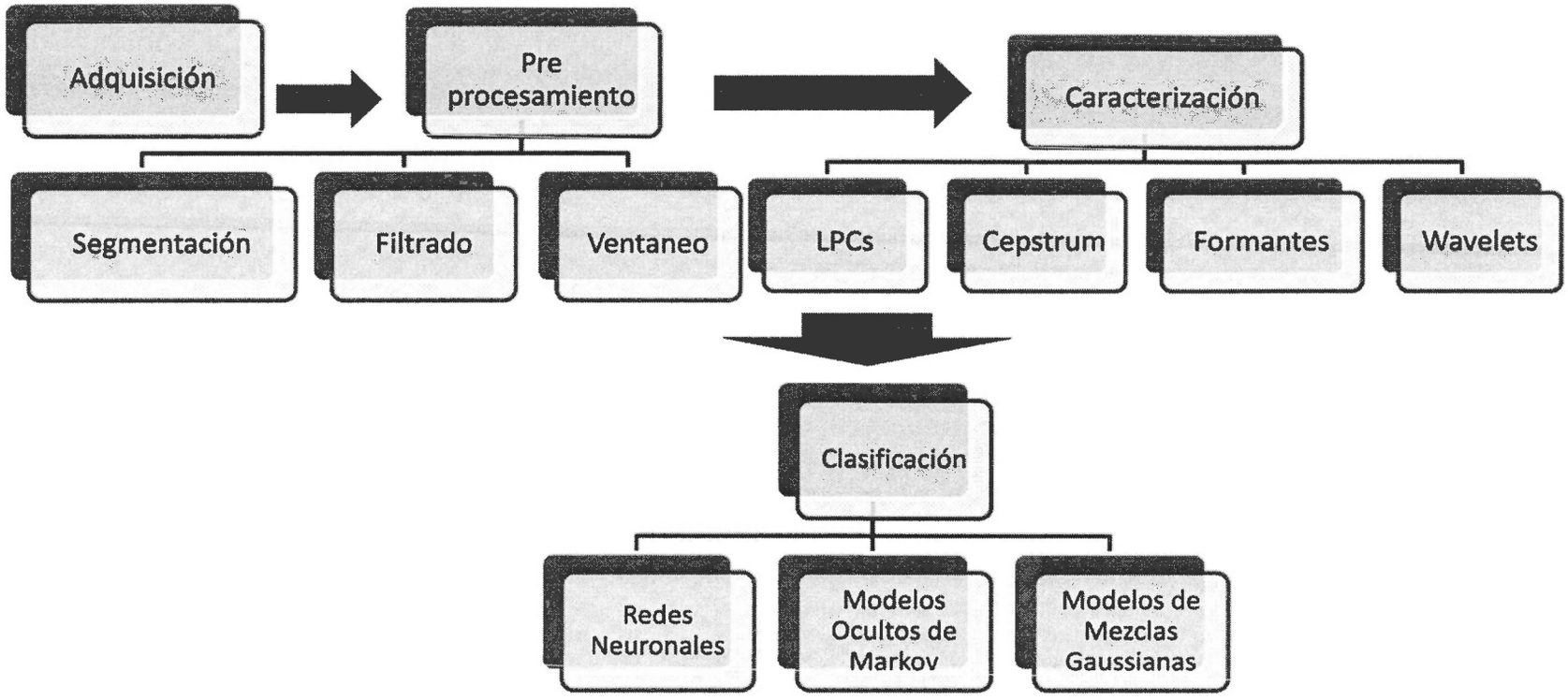


Diagrama 1: Esquema de procesamiento digital de voz.

Capítulo 3: Sistema propuesto

En este capítulo se describirán los sistemas propuestos para cada una de las herramientas que conforman el *toolkit*.

3.1. Segmentación.

Ícono: 

Descripción: Esta herramienta se encarga de segmentar una señal en segmentos del número de muestras especificado por el usuario. Esta herramienta debe funcionar dentro de un ciclo para que el índice sea aumentado en cada ciclo.

Entradas:

Input: recibe un arreglo de tipo *double* que es la señal a segmentar.

Samples: del tipo *integer*, es el número de muestras por segmento.

Index: es un entero que indica el índice de avance sobre la señal.

Salidas:

Output: arreglo de tipo *double* que contiene los segmentos de la señal, su longitud es el número de muestras.

Diagrama a bloques:

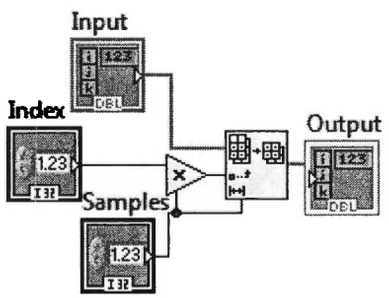


Diagrama 2: Diagrama a bloques de herramienta segmentación.

3.1.1. Metodología del algoritmo

El funcionamiento de esta herramienta se basa principalmente en la VI denominada *Array subset*, que obtiene un segmento de el arreglo determinado. La herramienta utiliza el valor del índice y de la cantidad de muestras para calcular la posición de inicio y la longitud del nuevo arreglo, esto por medio de una multiplicación. Finalmente se obtiene el segmento del arreglo de entrada de tamaño *Samples* a partir del índice *Index*.

3.2. Pre-énfasis

Ícono: 

Descripción: Entrega la señal de entrada pasada por un filtro de pre-énfasis con frecuencia de corte variable, misma que el usuario puede elegir.

Entradas:

F: del tipo *double*, frecuencia de corte del filtro de pre-énfasis en Hz.

Ts: del tipo *double*, es el periodo de muestreo de la señal de entrada.

Input: arreglo del tipo *double* que representa la señal de entrada al filtro.

Salidas:

Output: arreglo de tipo *double* que contiene la señal ya filtrada.

Diagrama a bloques:

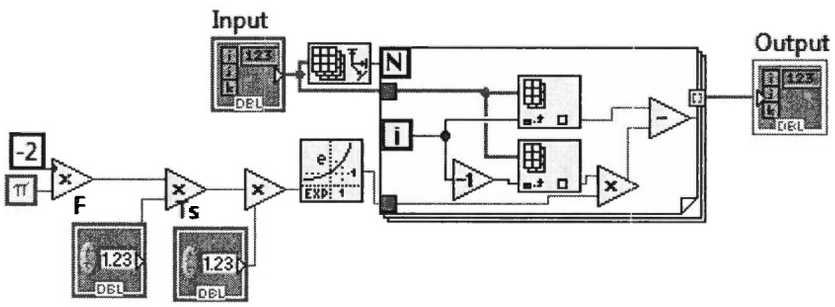


Diagrama 3: Diagrama a bloques de herramienta de pre-énfasis.

3.2.1. Metodología del algoritmo

De inicio se calcula el coeficiente del filtro de pre-énfasis denominado α de la siguiente forma:

$$\alpha = e^{-2\pi F T_s}$$

Ecuación 20

Por otro lado se obtiene la longitud del vector de entrada por medio de la herramienta *Array Size* y este valor es proporcionado a una estructura tipo *for* como número de iteraciones. Dentro de este ciclo se realiza el filtrado de la señal tomando el índice *i* del mismo como apuntador al arreglo de entrada y obteniendo los datos necesarios gracias a la herramienta *Index Array*, que nos entrega el valor del elemento apuntado por el índice. El cálculo de la señal filtrada se basa en la fórmula mostrada en la ecuación 21:

$$y_i = x_i - \alpha x_{i-1}$$ *donde i es el índice de los arreglos.*

Ecuación 21

3.3. De-énfasis

Ícono: 

Descripción: Entrega la señal de entrada pasada por un filtro de de-énfasis con frecuencia de corte variable, misma que el usuario puede elegir.

Entradas:

F: del tipo *double*, frecuencia de corte del filtro de de-énfasis en Hz.

Ts: del tipo *double*, es el periodo de muestreo de la señal de entrada.

Input: arreglo del tipo *double* que representa la señal de entrada al filtro.

Salidas:

Output: arreglo de tipo *double* que contiene la señal ya filtrada.

Diagrama a bloques:

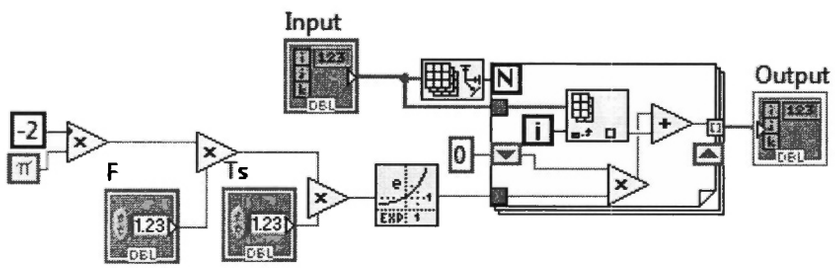


Diagrama 4: Diagrama a bloques de herramienta de-énfasis.

3.3.1. Metodología del algoritmo

Esta herramienta calcula el coeficiente α de la misma manera que el filtro de pre-énfasis, la diferencia entre aquella y ésta radica en la forma de calcular la nueva señal. Para este caso se utiliza un *shift register* que almacena el valor de la iteración anterior del ciclo, este valor se multiplica por el coeficiente del filtro y se le suma a la señal de entrada. Esta herramienta se basa en la ecuaciones 22 y 23.

$$y_1 = x_1$$ *Se inicializa el arreglo de salida.*

Ecuación 22

$$y_i = x_i + \alpha y_{i-1}$$ *donde i es el índice de los arreglos.*

Ecuación 23

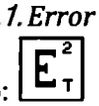
3.4. Coeficientes de predicción lineal

Esta herramienta fue desarrollada para calcular los coeficientes de predicción lineal a partir del vector de autocorrelación y por medio del algoritmo recursivo de Levinson-Durbin. Al realizar la recursión con *LabVIEW*, se pudo encontrar que dicho programa es extremadamente ineficiente para realizar recursiones. Por lo anterior se replanteó el problema para buscar una solución iterativa para el algoritmo recursivo de Levinson-Durbin. A continuación se presentan las herramientas obtenidas para ambos casos, siendo la forma iterativa la que será añadida al *toolkit*.

3.4.1. LPC's algoritmo recursivo

Para el cálculo de LPC's con algoritmo recursivo fue necesario generar tres herramientas distintas, una que calculara el error, otra los coeficientes de recursión de Levinson-Durbin y la última que calculara el vector de LPC's. Se describirán a continuación cada una de ellas.

3.4.1.1. Error recursivo

Ícono: 

Descripción: Calcula el error cuadrático para la recursión de Levinson-Durbin, se llama a sí misma y a la herramienta recursiva de coeficientes *a*.

Entradas:

Order: del tipo *integer*, orden del error que se desea calcular.

Rxx: arreglo del tipo *double*, es el vector de autocorrelación de la señal.

Salidas:

Et: del tipo *double*, es el error calculado

Diagrama a bloques:

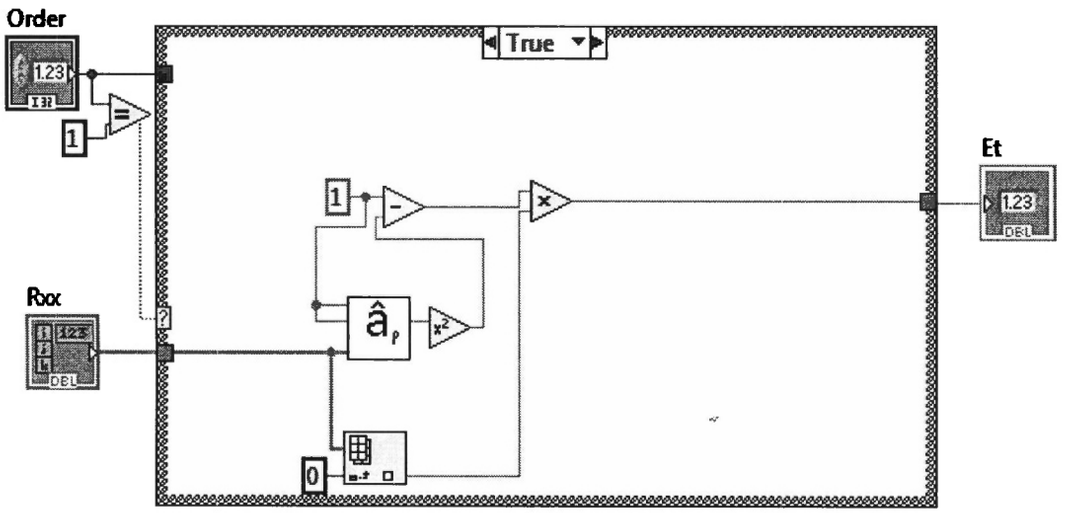


Diagrama 5: Diagrama a bloques de herramienta error caso 1

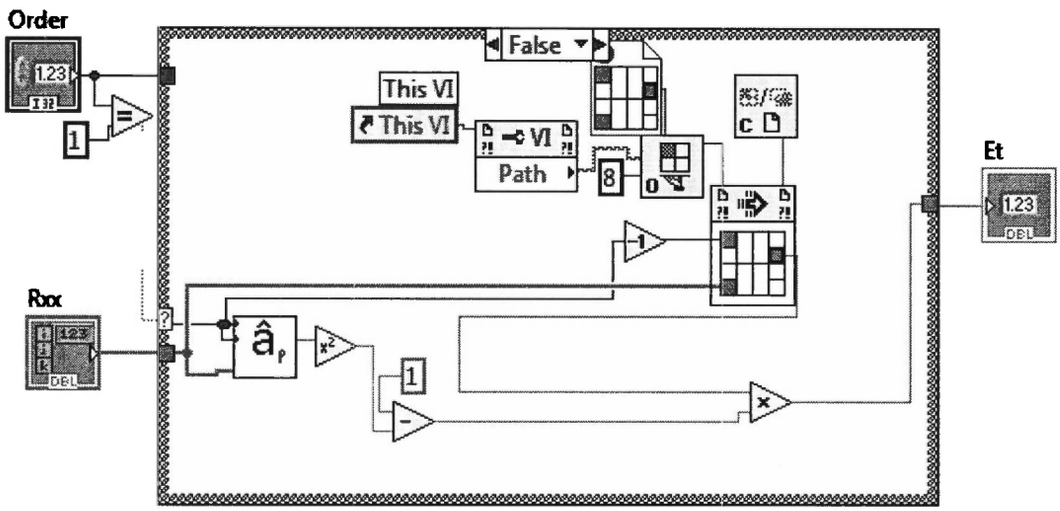


Diagrama 6: Diagrama a bloques de herramienta error caso 2

3.4.1.1.1. Metodología del algoritmo recursivo

Esta herramienta está dedicada a calcular el factor de error para la recursión de Levinson-Durbin, la recursión se basa en llamar a la VI desde ella misma (lo que se puede notar en los diagramas anteriores) generando clones en la memoria con entradas y salidas independientes.

Tiene dos casos, uno que inicia la recursión con la operación mostrada en el capítulo 2 y la segunda que implica la llamada a esta misma herramienta.

Además, esta herramienta requiere hacer una llamada a la VI que calcula los coeficientes de la recursión de Levinson-Durbin, misma que a su vez llama a una nueva copia de esta herramienta.

3.4.1.2. Coeficientes a recursivos

Ícono: 

Descripción: Calcula el coeficiente *a* del orden e índice proporcionados por las entradas de la misma. Esta función se llama a sí misma y llama a la función de error para realizar los cálculos.

Entradas:

Order: del tipo *integer*, orden del filtro que se desea calcular.

Index: del tipo *integer*, índice del coeficiente que se desea calcular.

Rxx: arreglo del tipo *double*, es el vector de autocorrelación de la señal.

Salidas:

a: del tipo *double,* es el coeficiente calculado.

Diagrama a bloques:

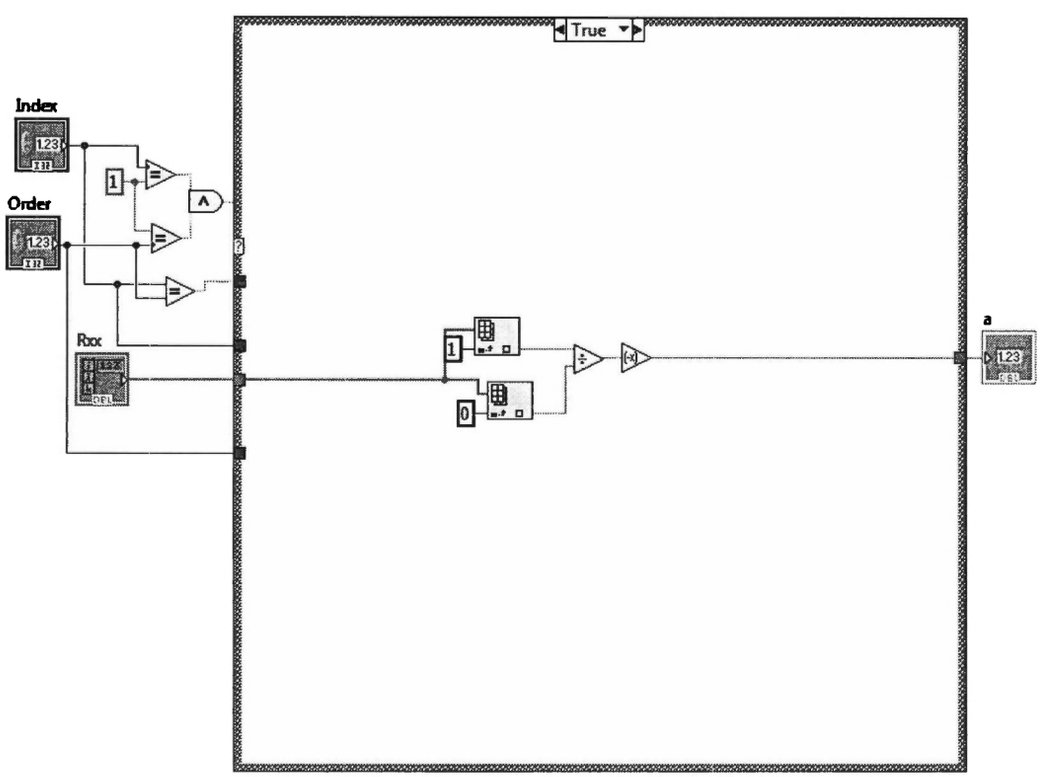


Diagrama 7: Diagrama a bloques de herramienta coeficiente caso 1

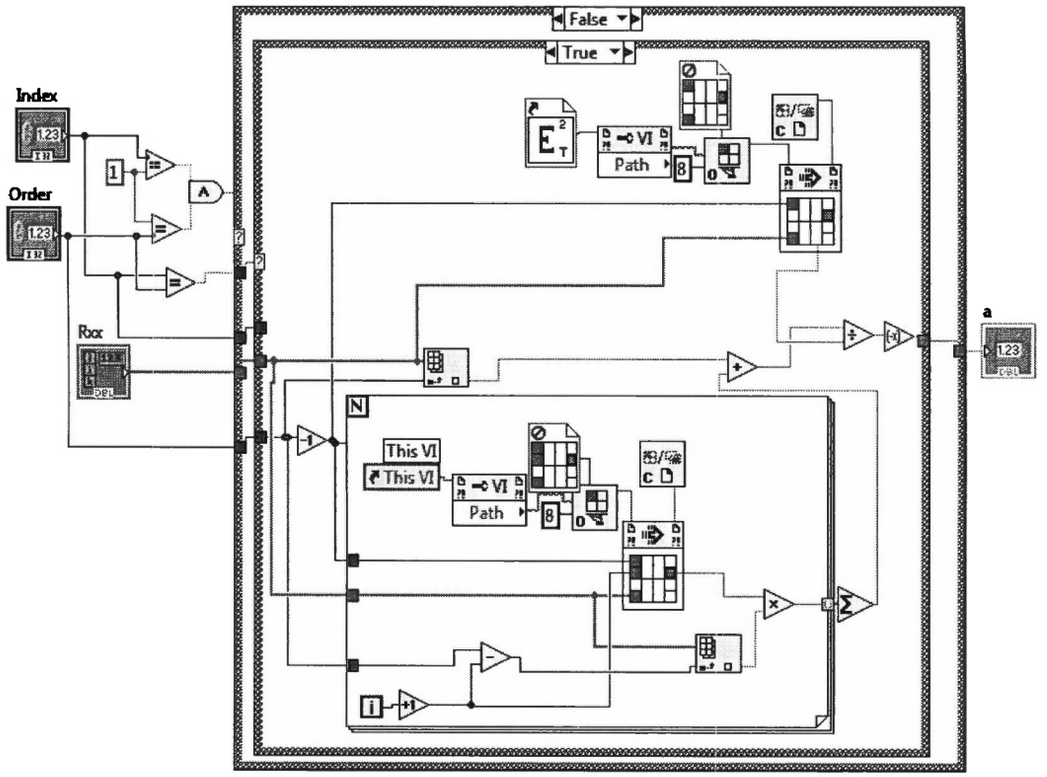


Diagrama 8: Diagrama a bloques de herramienta coeficientes caso 2

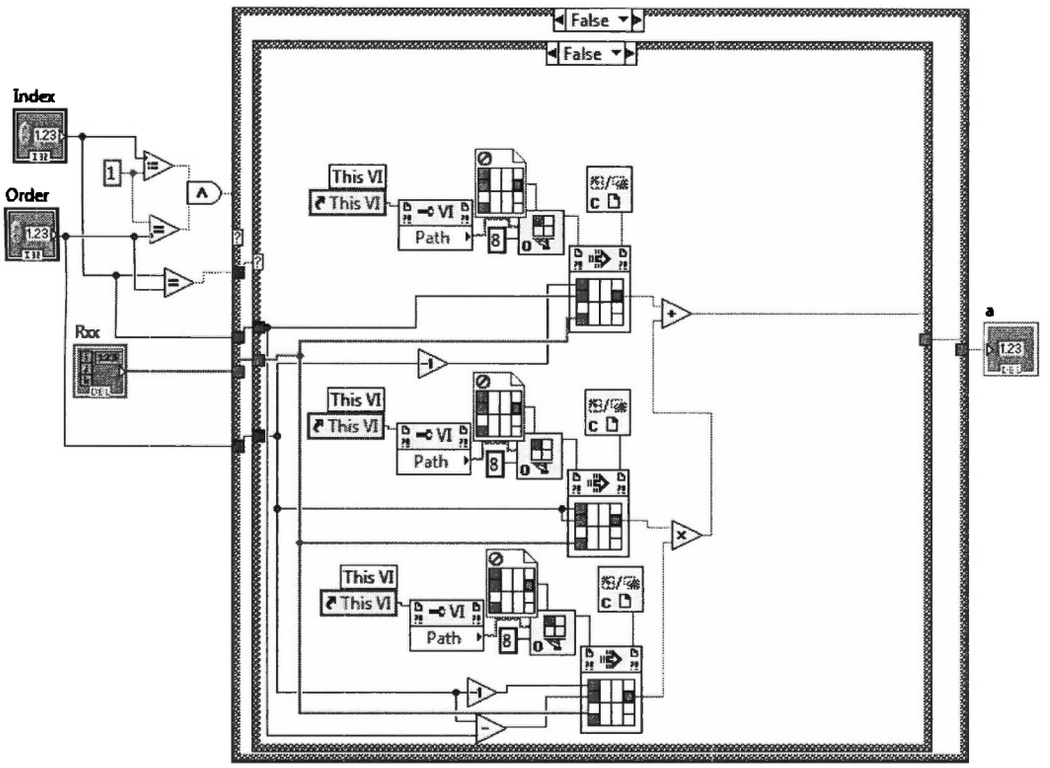


Diagrama 9: Diagrama a bloques de herramienta coeficientes caso 3

3.4.1.2.1. Metodología del algoritmo de coeficientes

En esta herramienta existen tres casos distintos, los cuales serán explicados a continuación:

1) ***Index=1, Order=1.*** Este es el inicio de la recursión y por lo tanto sólo se realiza la división del elemento 1 del vector de autocorrelación entre el elemento 0 del mismo. Finalmente se añade un signo negativo al resultado. Éste es enviado como resultado a la salida de la VI.
2) ***Order=Index.*** En este caso, según el algoritmo recursivo, se debe hacer lo siguiente:

$$\hat{a}_i^i = -\frac{\hat{R}_{XX}(i) + \sum_{j=1}^{i-1} \hat{a}_j^{i-1}\, \hat{R}_{XX}(i-j)}{(E_T^2)^{i-1}}$$

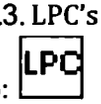
Ecuación 24

Como podemos observar, se toma el valor en la posición *i* por medio de la VI denominada *Index Array* y ese valor es sumado al resultado de realizar el ciclo que ejecuta esta misma VI para obtener los coeficientes de los filtros de órdenes menores, los multiplica por el valor del vector de autocorrelación en la posición *i-j* y hace la suma de todos estos elementos (VI *Sum*). Finalmente entrega el resultado como salida de la herramienta.

3) ***Else.*** Es el caso en el que ninguna de las anteriores condiciones se cumple, para obtener el resultado para estas condiciones, la VI se ejecuta a sí misma con el orden reducido en 1, obtiene el resultado y le suma la multiplicación del resultado de esta VI (con el mayor índice) por esta misma con el índice y orden reducido.

Al analizar el algoritmo observamos que el número de llamadas recursivas es grande y en consecuencia, la memoria se satura de VI's clonadas que esperan por un resultado para continuar su ejecución. Problema que en otros lenguajes de programación no sería grave.

3.4.1.3. LPC's utilizando herramientas recursivas

Ícono: 

Descripción: Calcula los coeficientes de predicción lineal del orden especificado por el usuario para la señal de entrada, llama a la función de coeficientes *a*.

Entradas:

Order: del tipo *integer*, orden del filtro de predicción lineal que se calculará.

Input: arreglo del tipo *double*, es la señal de entrada.

Salidas:

LPCoef: arreglo del tipo *double,* son los coeficientes para el filtro de predicción lineal.

Diagrama a bloques:

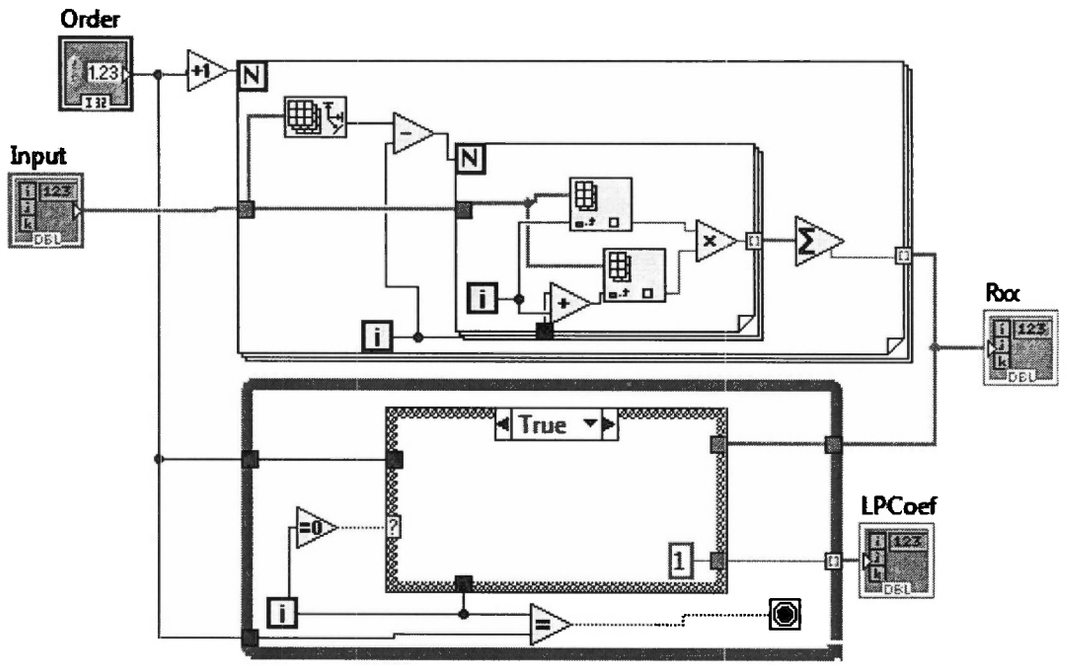


Diagrama 10: Diagrama a bloques de herramienta LPC caso 1.

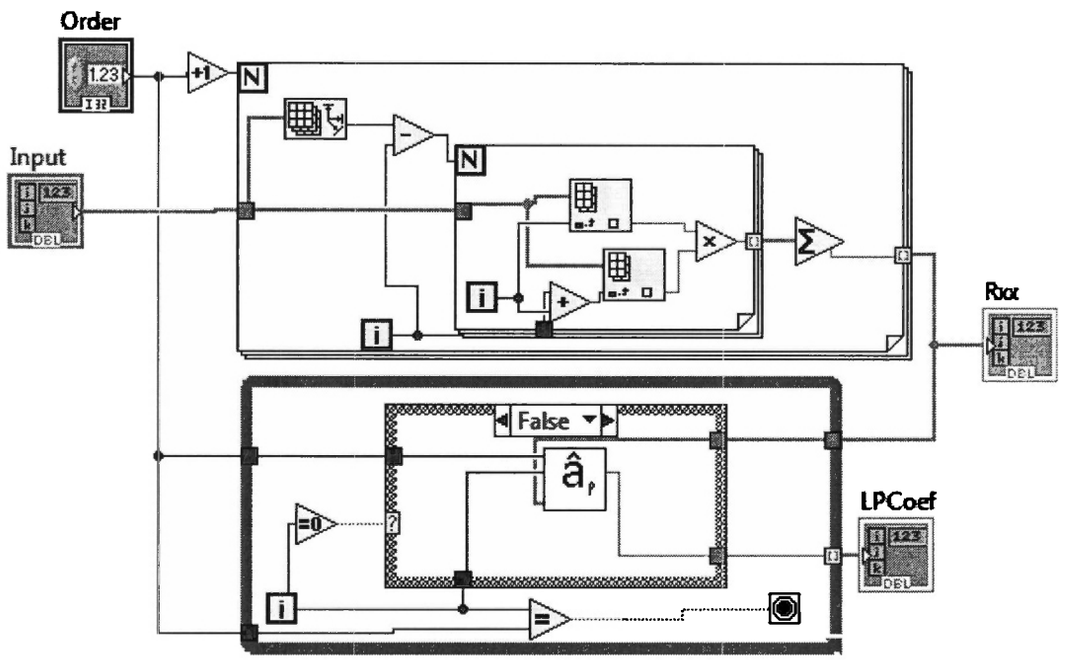


Diagrama 11: Diagrama a bloques de herramienta LPC caso 2.

3.4.1.3.1. Metodología del algoritmo de LPC's

Esta VI se encarga de llamar a las dos VI's mencionadas anteriormente después de generar el vector de autocorrelación (cuadro superior). También forma el vector de coeficientes de predicción lineal inicializándolo con un 1 en la primera posición.

La desventaja principal de este sistema radicó en que sólo se podían generar unas cuantas recursiones, después de las cuales la memoria de la computadora se saturaba y ésta se tornaba cada vez más lenta hasta dejar de trabajar.

La solución iterativa logró corregir estos problemas ya que se obtuvieron resultados eficientes incluso para orden 15 y mayores. Se omitieron también las herramientas que calculaban el error y los coeficientes *a*, gracias a que el cálculo de los mismos se incluyó en la iteración que calcula los LPC's. A continuación se describe la herramienta iterativa.

3.4.2. LPC's utilizando herramienta iterativa

Ícono: 

Descripción: Calcula los coeficientes de predicción lineal del orden especificado por el usuario para la señal de entrada a través de iteraciones.

Entradas:

Order: del tipo *integer*, orden del filtro de predicción lineal que se calculará.

Input: arreglo del tipo *double*, es la señal de entrada.

Salidas:

LPCoef: arreglo del tipo *double,* son los coeficientes para el filtro de predicción lineal.

Diagrama a bloques: (Se colocó en las siguientes páginas para poder apreciarlo correctamente)

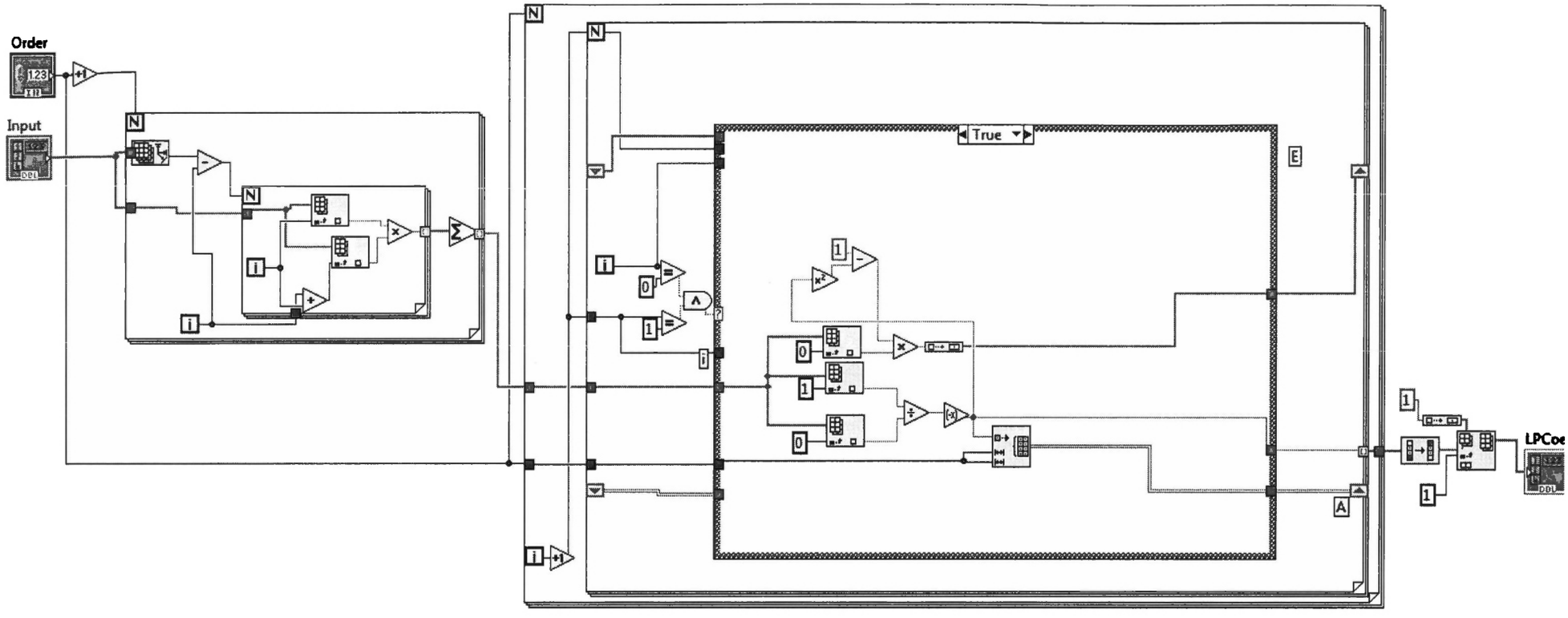


Diagrama 12: Diagrama a bloques de herramienta LPC final caso1.

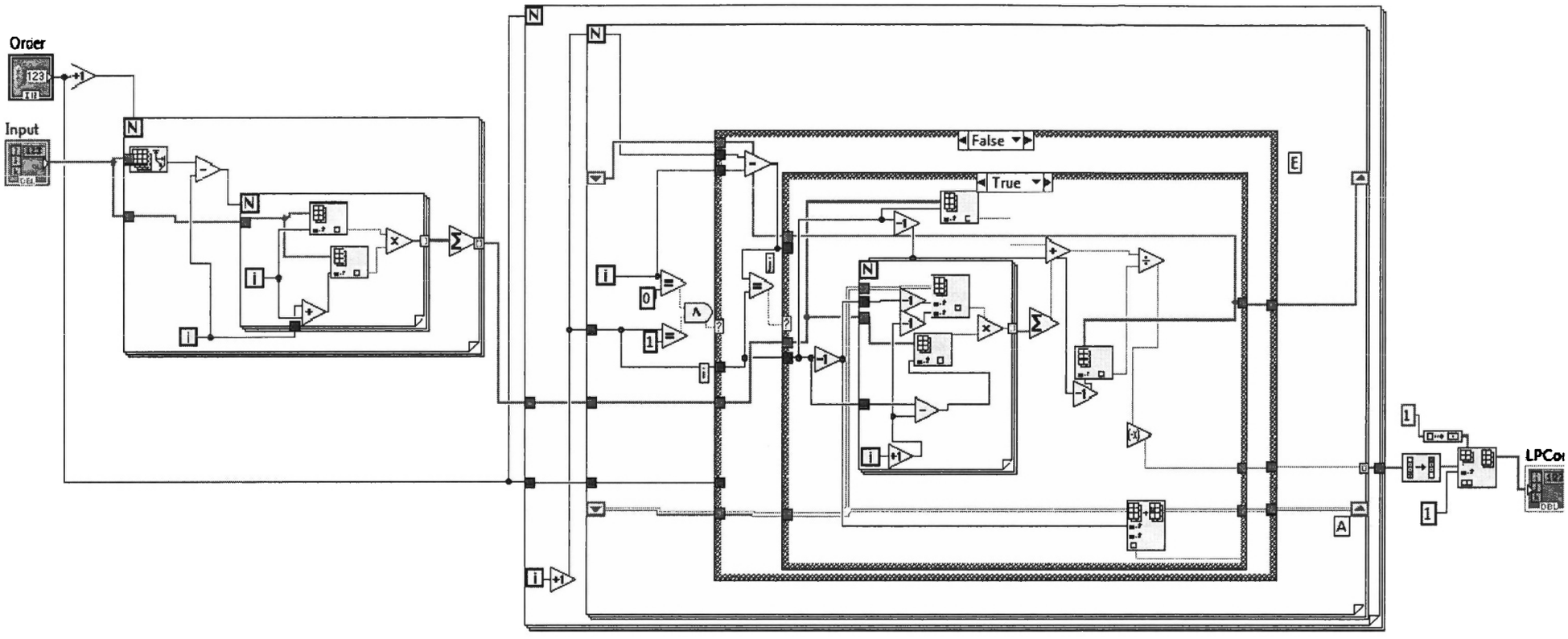


Diagrama 13: Diagrama a bloques de herramienta LPC final caso2.

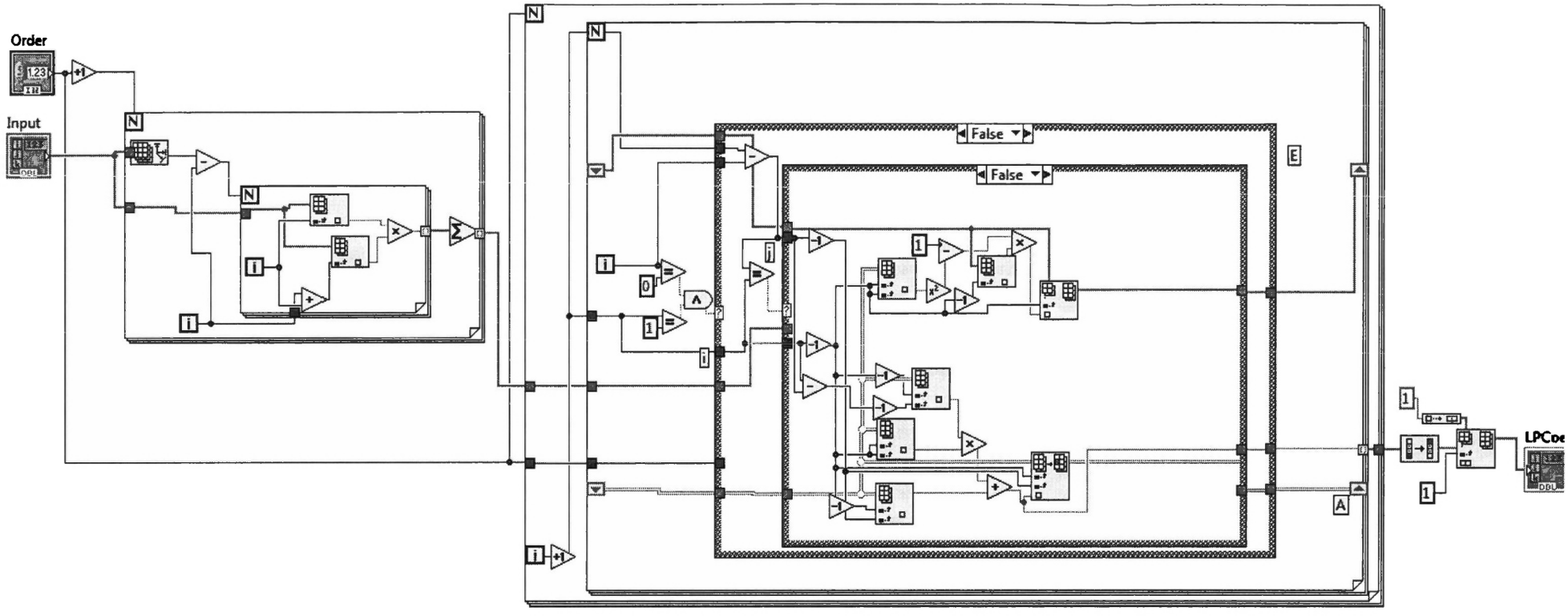


Diagrama 14: Diagrama a bloques de herramienta LPC final caso3.

3.4.2.1. Metodología del algoritmo

Como solución a los problemas de recursión se implementó este algoritmo basado en la recursión de Levinson-Durbin pero aplicada de forma iterativa.

Para analizarla tomaremos en cuenta el primer bloque (extremo izquierdo). Estos ciclos se encargan de generar el vector de autocorrelación. El ciclo interno realiza la multiplicación de un elemento por otro con desplazamiento variable sobre el vector y al finalizar realiza la suma de todos estos elementos. Esto genera el primer coeficiente del vector de autocorrelación. El ciclo externo se encarga de realizar eta operación el número de veces indicada por *Order* y almacenar los resultados en un arreglo, de modo que el vector de autocorrelación está completo.

En el segundo bloque (lado derecho) encontramos dos ciclos anidados con dos estructuras *case* dentro. Se explicará cada caso posible a continuación:

1) ***Index=1, Order=1.*** Como en la versión recursiva explicada previamente, en el inicio de la recursión se realiza la división del elemento 1 del vector de autocorrelación entre el elemento 0 del mismo. La diferencia con la anterior radica en que el cálculo del error se realiza en este mismo ciclo, ahorrando tiempo y espacio en memoria. Una vez calculados los valores, son transformados a arreglos y se almacenan en un *shift register* que funcionará como entidad de memoria para escribir datos futuros y leer anteriores. El *shift register* superior conforma la memoria del error (etiquetado con una E) y el inferior la memoria de los coeficientes (etiquetado con una A).

2) ***Order=Index.*** En este caso, se debe realizar la misma operación que en la herramienta recursiva, pero con la variante del almacenamiento de los arreglos. En el diagrama a bloques, la etiqueta denominada *i* nos indica el índice y el orden, ya que en este caso ambos son iguales. Dentro de la estructura *case* interna encontramos un ciclo cuya función es determinar la sumatoria de los coeficientes de los filtros de orden anterior. Como en esta herramienta ya no contamos con la recursión, el algoritmo accede al *shift register* de los coeficientes a en la posición indicada por el índice de este ciclo, obteniendo así los valores previamente calculados. Fuera del ciclo encontramos otras VI's, la función de éstas es calcular el error para este valor de orden. Se calcula accediendo a los valores de error previamente obtenidos y almacenados en el *shift register*.

3) ***Else (ambas condiciones en falso).*** Cuando las dos condiciones se encuentran en falso, es decir *Order* e *Index* son diferentes entre sí y diferentes de 1, se realiza el cálculo accediendo nuevamente a los valores almacenados en el *shift register* de coeficientes a. Cabe aclarar que en esta parte del algoritmo, el acceso se debía realizar del mayor orden hacia el menor, puesto que se buscaba en el arreglo los coeficientes del mayor índice en un determinado orden para poder calcular los de menor índice. Lo anterior explica la resta localizada cerca de la etiqueta *j*.

Finalmente, para facilitar la comprensión del algoritmo, es necesario explicar que el *shift register* de coeficientes *a* almacena un arreglo de dos dimensiones: una de ellas el índice y la otra el orden. De esta forma, al querer acceder a un determinado coeficiente, basta con utilizar la herramienta *Array Index* (en su versión de dos dimensiones), entregar los valores de orden e índice y a la salida obtendremos el coeficiente requerido.

Para concluir, el algoritmo obtiene el vector de coeficientes para el orden requerido (ya que en el *shift register* se encuentran todos los filtros de orden menor o igual a *Order*) a través del ciclo interno. Se puede observar en el diagrama a bloques que, además de almacenar los valores en el *shift register,* el algoritmo envía cada valor final de coeficientes del orden requerido hacia afuera de los ciclos. Una vez que terminan los ciclos, el arreglo es pasado hacia el exterior, donde se encuentran diferentes VI's que tienen como propósito ordenar los coeficientes, agregar un 1 en la primer posición del vector de salida y entregar éstos a la salida de la VI.

3.4.3. Diferencias entre algoritmos y transformación de recursivo a iterativo

El algoritmo recursivo antes mencionado realizaba llamadas a las funciones de "error" y "coeficientes a", a su vez se llamaba a ella misma y así sucesivamente hasta llegar al menor índice del ciclo. Al llegar a este momento, la cantidad de VI's cargadas en la memoria ocasionaba que la computadora comenzara a paginar en disco duro y se alentara el proceso, de modo tal que el sistema se detenía.

En este momento era imprescindible realizar un algoritmo iterativo así que se consideraron las siguientes condiciones:

- Un algoritmo recursivo requiere calcular el último valor del arreglo antes de poder calcular los primeros.
- Las llamadas en un algoritmo recursivo se realizan del menor índice al mayor.
- Es necesario realizar los cálculos en una sola VI para evitar saturar la memoria.

Una vez establecido lo anterior se encontró que la nueva VI debía tener las siguientes características:

- Iniciar el cálculo a partir del último valor del arreglo.
- Calcular simultáneamente el error y lo coeficientes para evitar llamar otra VI.
- Continuar el cálculo en sentido inverso (del mayor índice al menor) hasta llegar al índice 0.

Con lo anterior se logró realizar una VI con algoritmo iterativo que cumple con las características necesarias para conformar este toolkit.

3.5. Formantes

Ícono: 

Descripción: Calcula los formantes a partir del vector de coeficientes de predicción lineal P(x).

Entradas:

P(x): arreglo de tipo *double* que contiene el vector de predicción lineal del segmento.

fs: del tipo *double*, frecuencia de muestre de la señal de entrada.

n: del tipo *integer*, es el número de formantes que se desea obtener.

Salidas:

Formants: arreglo del tipo *double,* son los formantes obtenidos, su longitud es igual a *n.*

Diagrama a bloques: (Se encuentra en la siguiente página para mejorar su apreciación).

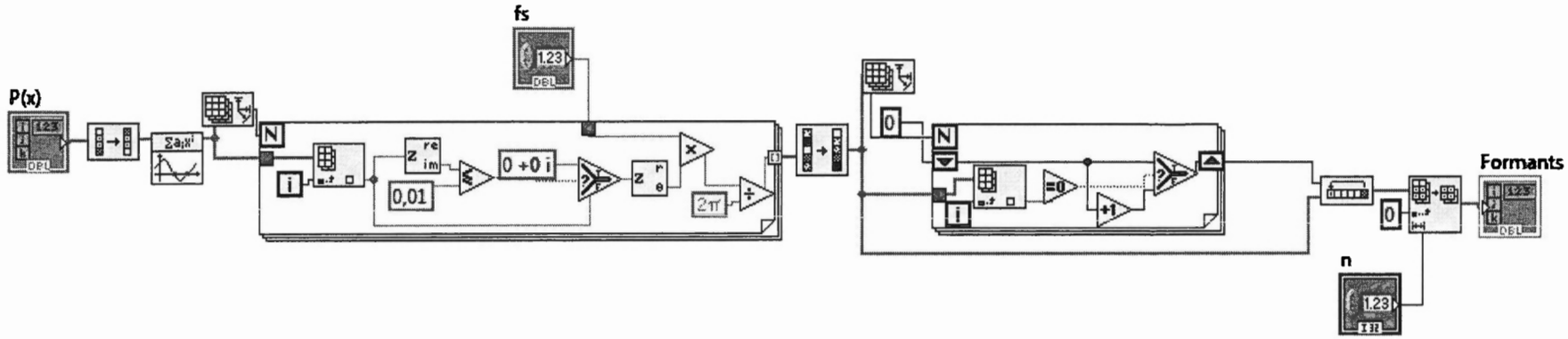


Diagrama 15: Diagrama a bloques de herramienta formantes.

3.3.1. Metodología del algoritmo

Una vez solucionados los problemas del algoritmo de Levinson-Durbin y el cálculo de LPC's se pudo continuar con el desarrollo de herramientas.

La herramienta para calcular formantes recibe el vector de coeficientes de predicción lineal que entrega la VI denominada *LPC's*, a continuación lo invierte para ordenar los coeficientes en orden ascendente de potencia. Este nuevo arreglo se introduce a la VI llamada *Roots*, que calcula las raíces del polinomio representado por el vector de LPC's.

El siguiente bloque es un ciclo que se ejecuta tantas veces como raíces en el polinomio. Dentro de este ciclo se obtienen uno a uno los valores de las raíces, se extrae la parte imaginaria y se pregunta si es menor o igual a 0,01, en caso afirmativo se elimina esa raíz y se coloca un cero en su posición. En caso contrario, se obtiene la fase de la raíz, se multiplica por la frecuencia de muestreo y se divide entre 2π. Todos los valores de este ciclo se almacenan en un arreglo y se ordenan en orden ascendente.

El segundo ciclo (derecha del diagrama) se encarga de contar el número de ceros que contiene el arreglo, esto para eliminar los ceros por medio de varios desplazamientos en el arreglo. Finalmente se tiene un arreglo que contiene los formantes en orden ascendente y los ceros al final. Esta herramienta extrae los *n* formantes de este arreglo y los entrega a la salida de la misma.

3.4. Cepstrum

Ícono: 

Descripción: Calcula los coeficientes cepstrales a partir del vector de coeficientes de predicción lineal A.

Entradas:

A: arreglo de tipo *double* que contiene el vector de predicción lineal del segmento.

LPC gain: del tipo *double*, ganancia del filtro de predicción lineal.

p: del tipo *integer*, es el número de coeficientes cepstrales que se desea obtener.

Salidas:

C: arreglo del tipo *double*, contiene los coeficientes cepstrales del segmento. Su longitud es igual a *p*.

Diagrama a bloques: (se presenta en las siguientes páginas para cada uno de sus casos)

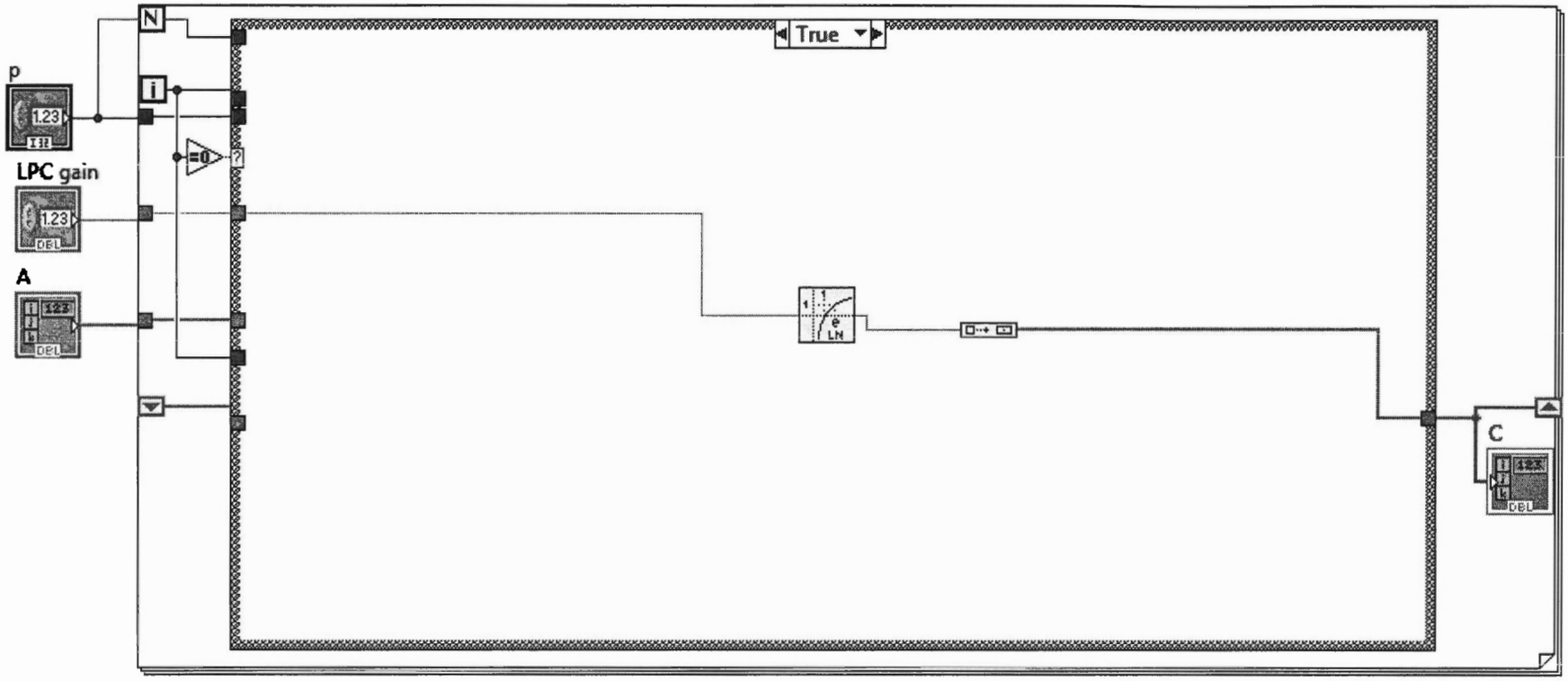


Diagrama 16: Diagrama de herramienta Cepstrum caso 1.

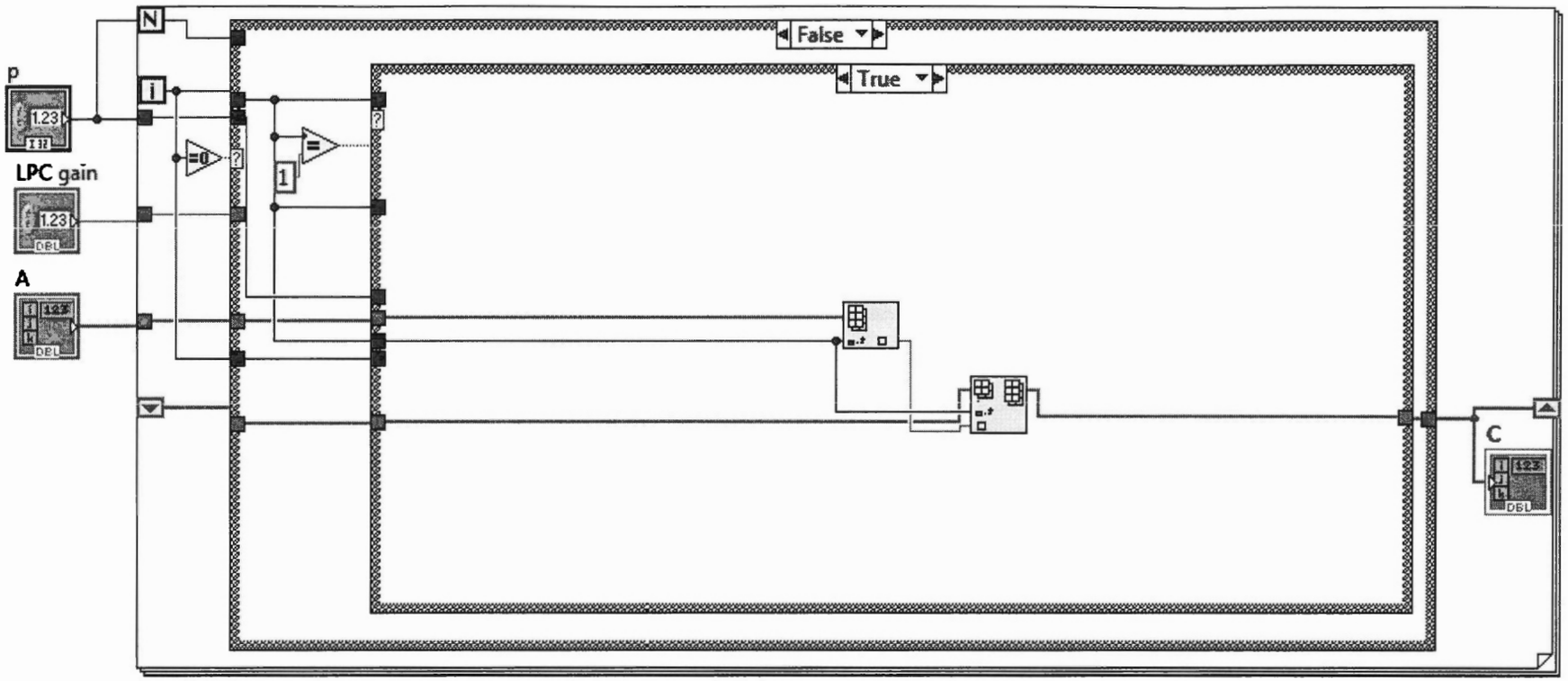


Diagrama 17: Diagrama de herramienta Cepstrum caso 2.

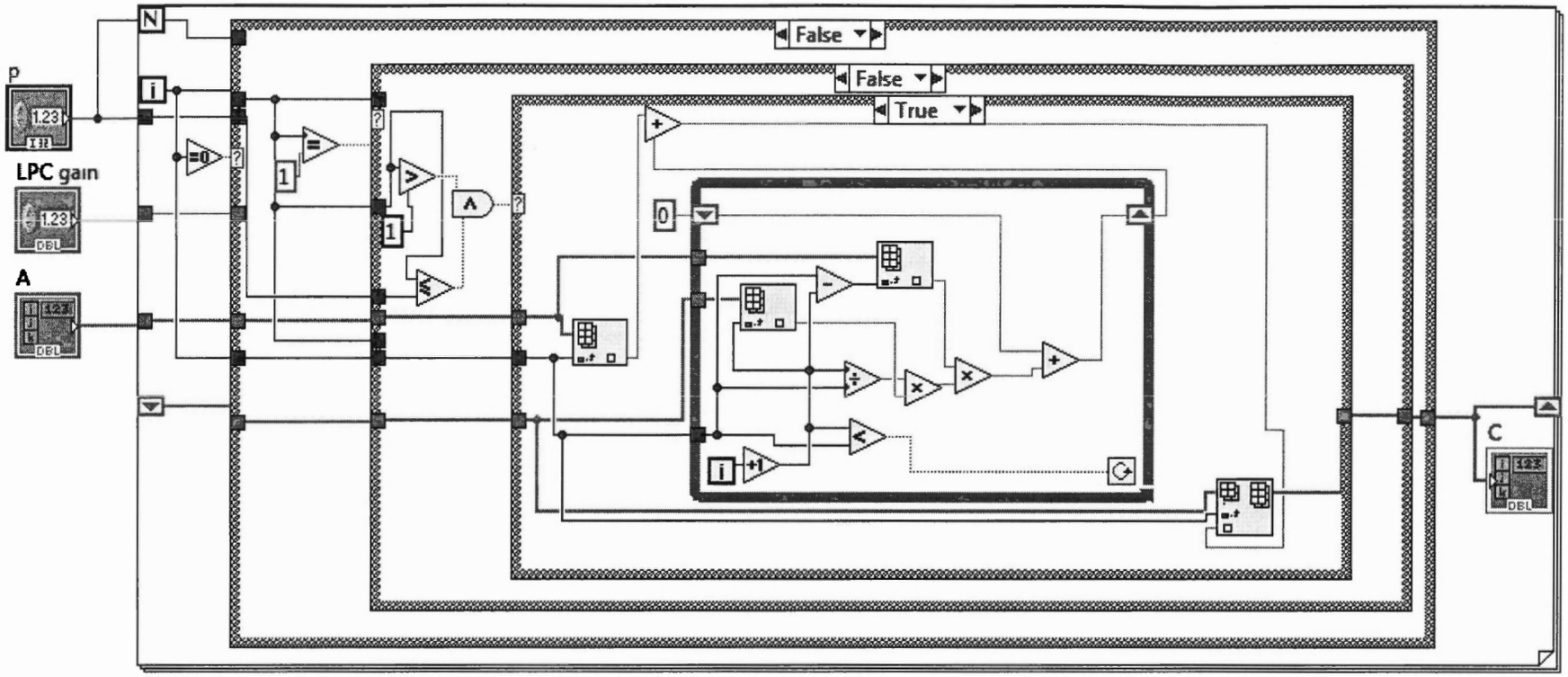


Diagrama 18: Diagrama de la herramienta Cepstrum caso 3.

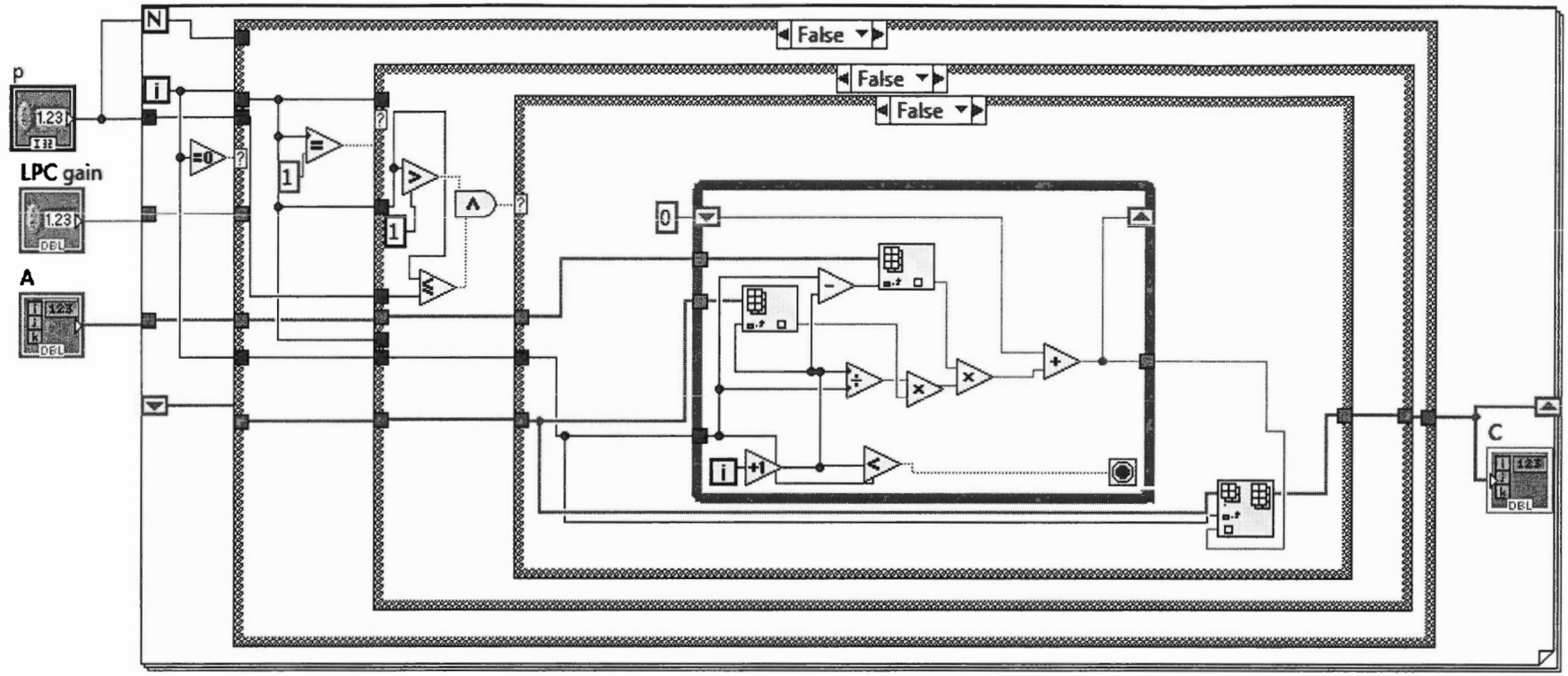


Diagrama 19: Diagrama de herramienta Cepstrum caso 4.

3.4.1. Metodología del algoritmo

Esta herramienta se basa en el algoritmo computacional presentado en el marco teórico de este documento.

El primer caso que se tiene es cuando el índice es cero, es decir, en la primera iteración. En este caso el algoritmo toma el valor de ganancia del filtro LPC y calcula el logaritmo natural del mismo. Finalmente este valor es agregado al arreglo de salida en la posición cero.

El segundo caso sucede cuando el índice es uno. La herramienta toma el valor del vector de LPCs en la posición 1 y lo transfiere al arreglo de salida en la misma posición.

El tercer caso resulta cuando el índice es mayor a uno y menor o igual que p. El algoritmo toma el valor del coeficiente de predicción lineal y le suma el resultado de la ecuación 25:

$$\sum_{k=1}^{m-1} \left(\frac{k}{m}\right) c_k a_{m-k}$$

Ecuación 25

El resultado se anexa al arreglo de salida en la posición indicada por el índice del ciclo.

El último caso sucede cuando el índice ha rebasado el valor de p. Se realiza la misma operación que en el caso anterior pero sin la suma del valor de coeficiente lineal.

Al finalizar, el algoritmo entrega un arreglo del tipo *double* que contiene los valores de los coeficientes Cepstrales, su tamaño es igual a p.

Capítulo 4: Resultados

En este capítulo se mostrarán los resultados obtenidos al utilizar cada una de las herramientas propuestas. Para algunas de ellas se realizará una comparación con los resultados obtenidos en diferentes funciones de *Matlab.*

4.1. Resultados de segmentación

Esta herramienta, aunque sencilla, es útil en el procesamiento de voz, pues como ya se explicó es necesario segmentar las señales para considerarlas estacionarias. A continuación se realiza una prueba para comprobar el funcionamiento de la herramienta de segmentación.

Se generó la siguiente VI:

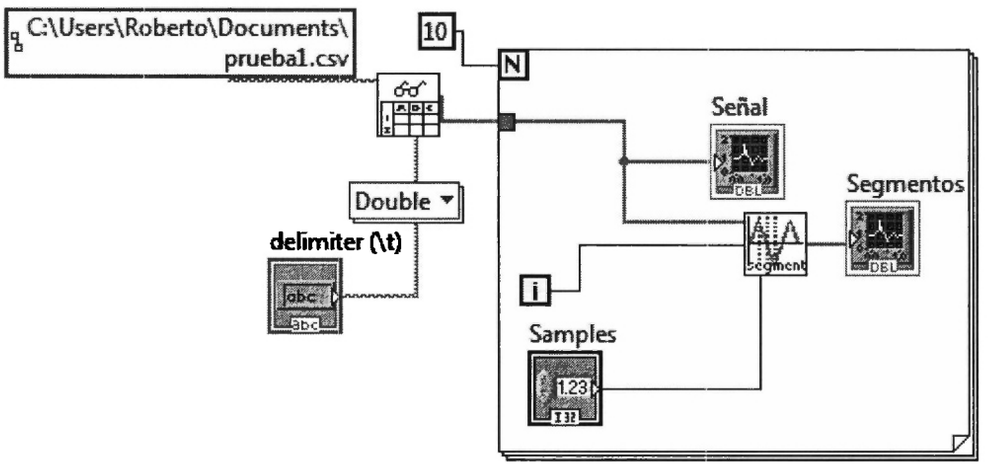
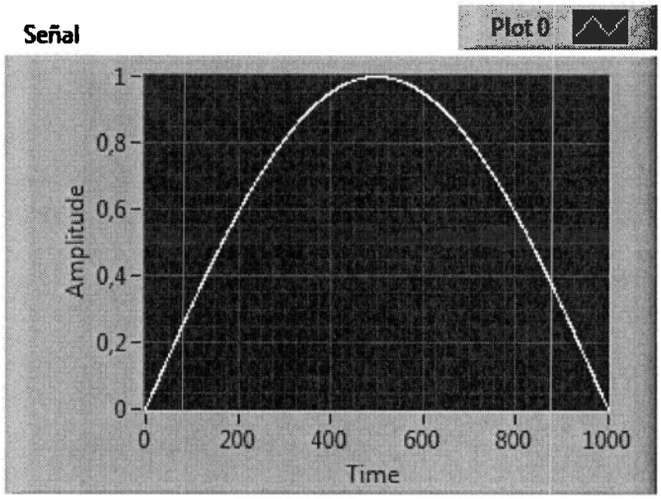


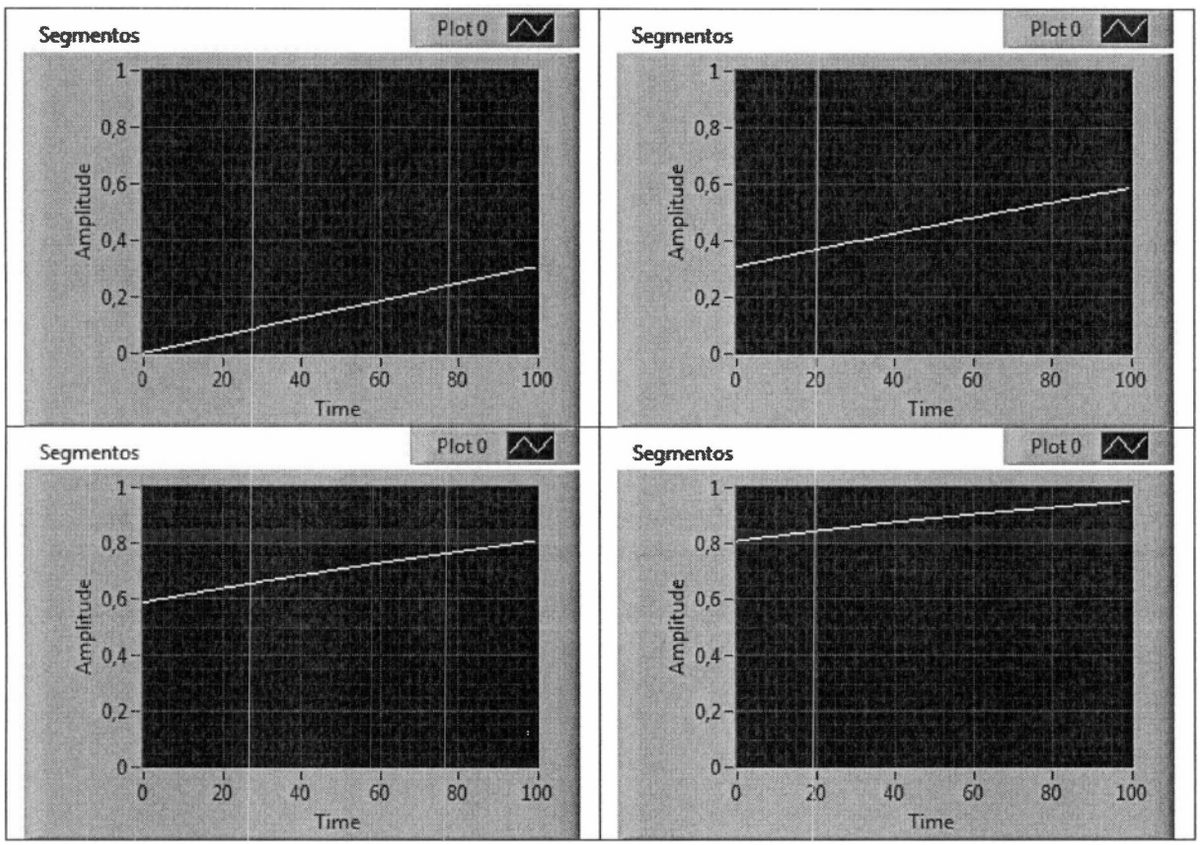
Diagrama 20: Diagrama a bloques de prueba segmentación.

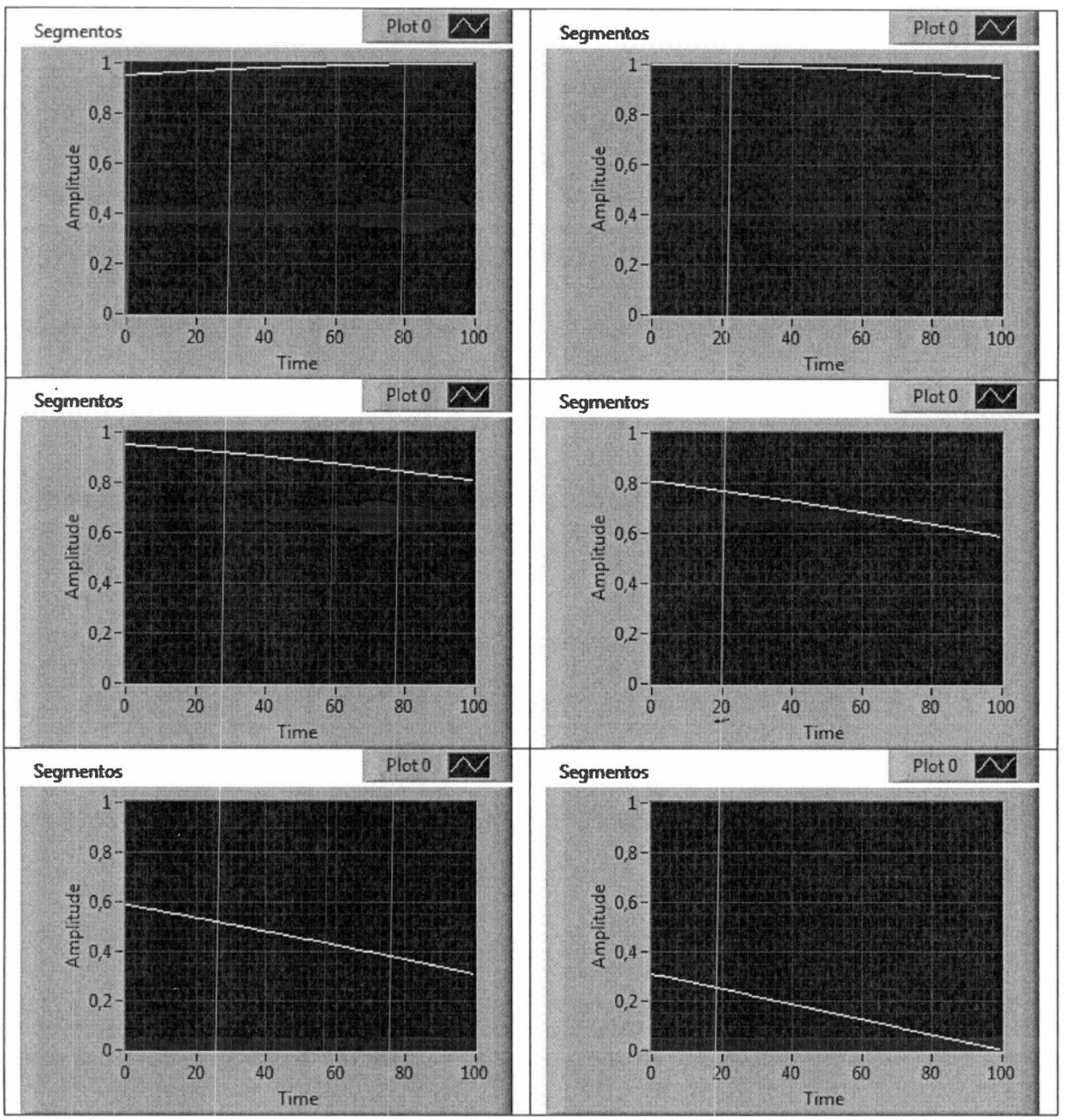
La entrada a esta VI es un archivo *.csv* que contiene valores de medio periodo de un seno, su gráfica se muestra a continuación.



Gráfica 1: Señal de entrada al sistema.

Al realizar el ciclo las 10 veces se obtienen las siguientes gráficas a la salida:





Se realizó esta prueba para diez señales de diferentes características y se obtuvieron resultados correctos. Cada una de las gráficas consecuentes representan uno de los segmentos de la señal. Con esto queda demostrado que la herramienta de segmentación cumple satisfactoriamente su propósito.

4.2. Resultados de filtro de pre-énfasis

Para demostrar el funcionamiento de ésta herramienta, se tomó como referencia que debe ser un filtro pasa altas, de modo que se generó una señal de ruido a la entrada para observar el comportamiento en muchas frecuencias. El diagrama a bloques es el siguiente:

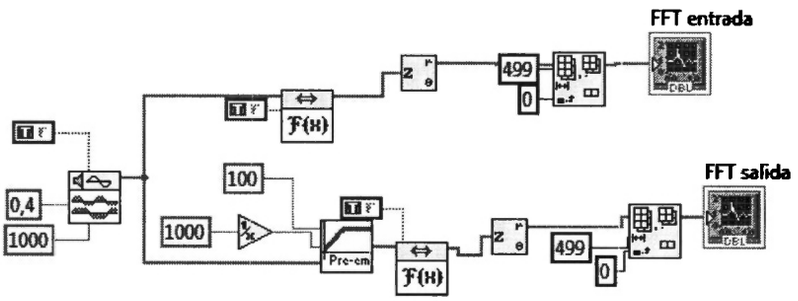
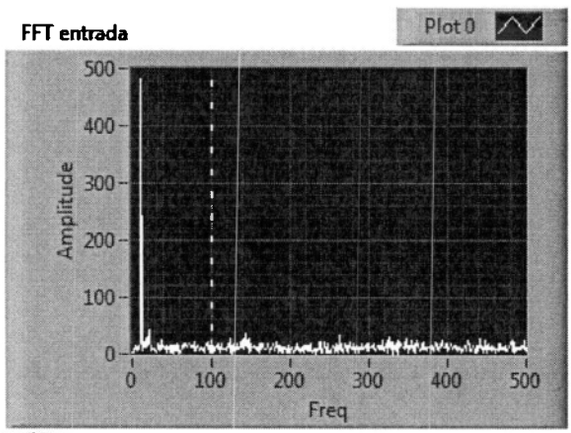
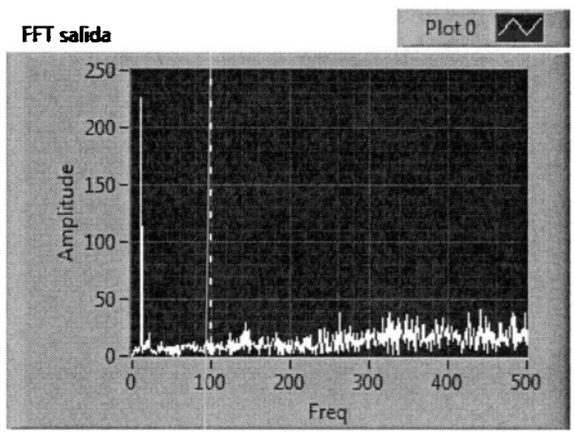


Diagrama 21: Diagrama a bloques de prueba pre-énfasis

Se obtuvo la gráfica de las transformadas de Fourier de las señales de entrada y salida y se observó lo siguiente:



Gráfica 2: FFT de la señal de entrada.



Gráfica 3: FFT de la señal de salida.

En este caso el filtro se creó con frecuencia de corte de 100Hz y podemos observar en las gráficas que la señal se encuentra filtrada. Es notorio que las frecuencias por debajo de los 100Hz fueron atenuadas y las mayores se amplificaron. La prueba se repitió para diferentes señales de ruido Gaussiano y algunas senoidales y el comportamiento de la herramienta fue correcto. Finalmente podemos concluir que la herramienta está funcionando como se esperaba, filtrando señales como un pasa-altas.

4.3. Resultados de filtro de de-énfasis

Como en la herramienta anterior, se realizó una prueba de filtrado. En este caso se espera obtener un filtro pasa-bajas de la frecuencia elegida por el usuario. El diagrama a bloques es el siguiente:

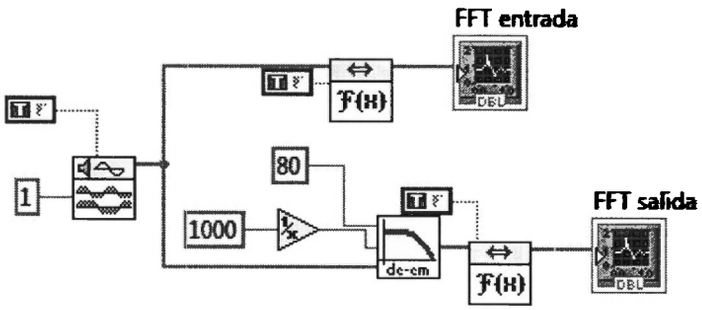
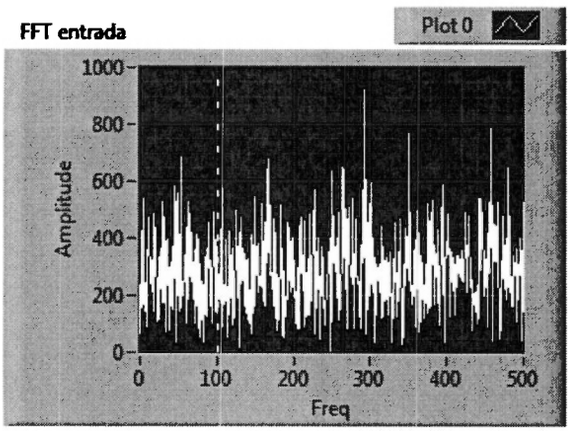
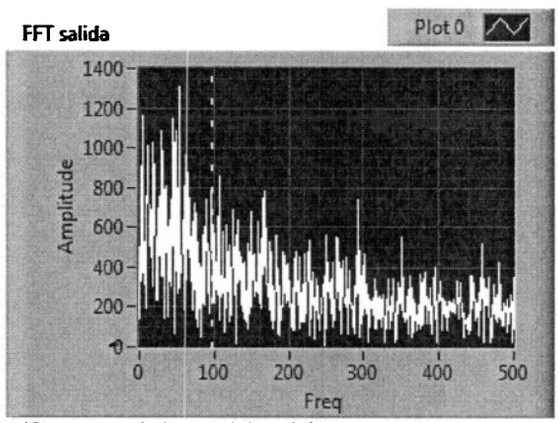


Diagrama 22: Diagrama a bloques de prueba de-énfasis.

Al graficar las transformadas de Fourier de la entrada y la salida obtenemos lo siguiente:



Gráfica 4: FFT de la señal de entrada.

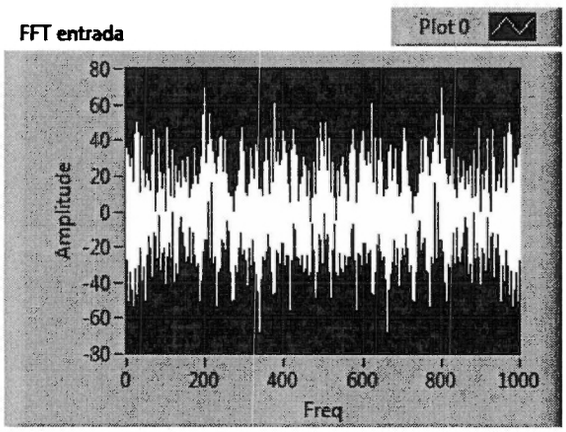


Gráfica 5: FFT de la señal de salida

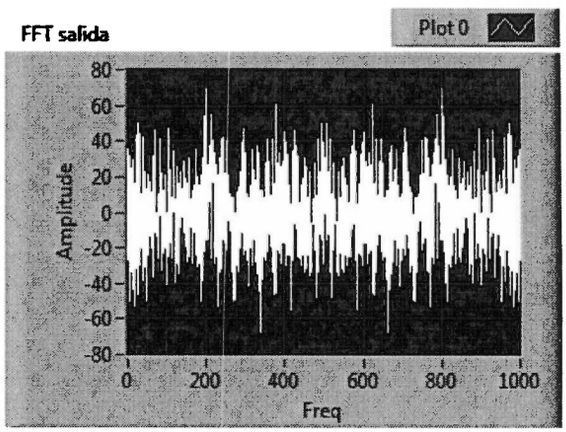
En este caso la frecuencia de corte es de 100Hz, podemos observar que las frecuencias menores a este valor se amplifican y las mayores se atenúan. El comportamiento se repitió para diferentes señales por lo que se concluye que la herramienta funciona correctamente.

4.4. Prueba de complemento de ambos filtros

Sabemos que el filtro de de-énfasis debe realizar el filtrado opuesto al de pre-énfasis, de modo que si hacemos pasar una señal por el filtro de pre-énfasis y a continuación por el de de-énfasis, debemos obtener la señal inicial. Al realizar esta prueba se obtiene lo siguiente:



Gráfica 6: FFT de la señal de entrada.



Gráfica 7: FFT de la señal de salida.

Como podemos observar, ambas señales son idénticas y podemos concluir que ambas herramientas se complementan.

4.5. Resultados de LPC's

Para validar esta herramienta, se recurrió a la función *lpc* de *Matlab*. Se utilizaron diferentes señales para realizar pruebas en el algoritmo. Un ejemplo de ellas es la siguiente: tanto en *LabVIEW* como en *Matlab* se adquiere la misma señal senoidal utilizada en la prueba de la segmentación. El diagrama a bloques es el siguiente:

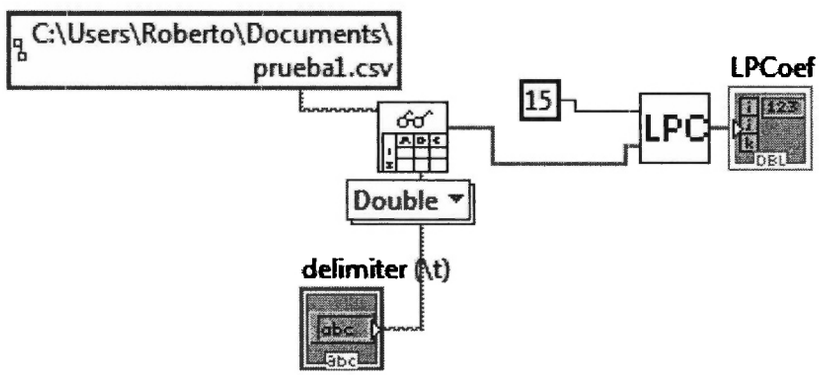


Diagrama 23: Diagrama a bloques de prueba LPC's.

Resultados:

LabVIEW	*Matlab*
1	1.0000
-0,0581128	-0.0581
-0,102098	-0.1021
-0,100373	-0.1004
-0,0617551	-0.0618
-0,0757648	-0.0758
-0,111302	-0.1113
-0,103631	-0.1036
-0,0168119	-0.0168
-0,0315505	-0.0316
-0,0992341	-0.0992
-0,06927	-0.0693
-0,061532	-0.0615
-0,0482911	-0.0483
-0,0134113	-0.0134
-0,0217201	-0.0217

Tabla 1: Comparación de resultados de LPC's.

Podemos notar que los valores entregados por Matlab y los entregados por la herramienta desarrollada coinciden. La herramienta funcionó de la misma manera que Matlab para 10 señales diferentes elegidas al azar. Finalmente se demostró que la herramienta desarrollada funciona de la misma manera que la función *lpc* de *Matlab*, por lo tanto queda validada su correcta implementación.

4.6. Resultados de Cepstrum

Debido a que Matlab no cuenta con una herramienta similar a esta, para la validación de la misma se utilizó la función *LPC to/from Cepstral Coefficients* de Simulink. Ambos sistemas son mostrados a continuación.

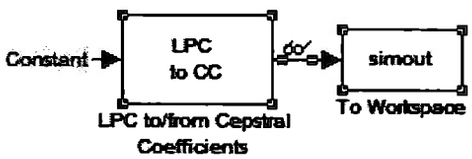


Diagrama 24: Programa para obtención de Coeficientes Cepstrales en Simulink

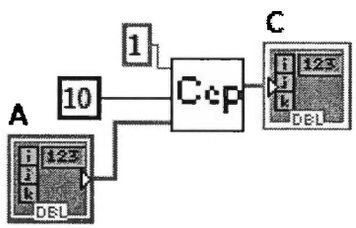


Diagrama 25: Diagrama a bloques para obtener Coeficientes Cepstrales en LabView

Se realizaron 10 pruebas diferentes introduciendo distintas señales. Para cada prueba se introdujo el mismo vector en ambas herramientas, los resultados de una de ellas se presentan a continuación.

LabView	Simulink
C	0
0	-2.0000
-2	-1.0000
-1	-0.6667
-0,666667	-0.5000
-0,5	-0.4000
-0,4	-0.3333
-0,333333	-0.2857
-0,285714	-0.2500
-0,25	-0.2222
-0,222222	

Tabla 2: Resultados de Coeficientes Cepstrales

Nuevamente notamos que los valores entregados por la herramienta coinciden con los entregados por Simulink, lo que nos indica que la herramienta tiene un correcto funcionamiento.

4.7. Resultados de Formantes

Para validar la herramienta de formantes, se utilizó una función en *Matlab* que había sido validada previamente (véase anexo). En *LabVIEW* también se generó una VI que utiliza la herramienta de LPC's y la de Formantes, esto porque el algoritmo de formantes los obtiene a partir del vector de LPC's. Para comprobar se introdujeron 10 diferentes vectores, un ejemplo de ellos fue uno con números del 1 al 7 en orden ascendente. El diagrama a bloques es el siguiente:

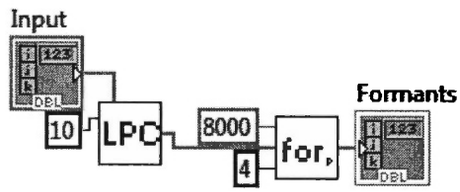


Diagrama 26: Diagrama a bloques de prueba formantes.

Los resultados obtenidos se muestran a continuación:

LabVIEW	*Matlab*
Formants 0 221,064 714,193 1850,27 2884,02	221.1 714.2 1850.3 2884.0

Tabla 3: comparación de resultados de formantes.

Nuevamente podemos comprobar que la herramienta funciona a la perfección al compararla con la función ya validada. El mismo resultado se obtuvo para las 10 distintas señales.

4.8. Sistema de identificación de vocales

Una vez que las herramientas fueron validadas correctamente se realizó un sistema capaz de identificar vocales por medio de los coeficientes de predicción lineal y Cepstrales. Para lograrlo se utilizaron herramientas de redes neuronales pertenecientes al *toolkit* de inteligencia artificial desarrollado por el Dr. Pedro Ponce. El objetivo de este sistema es demostrar una de las aplicaciones de las herramientas desarrolladas en este toolkit.

Los controles de la interfaz gráfica son los siguientes:

Ilustración 1: interfaz gráfica del sistema de identificación

El usuario elige el archivo que desea procesar, elige entre reconocimiento por Cepstrum o LPCs y ejecuta el programa. A continuación se reproduce el archivo de audio a reconocer y en el campo "Vocales presentes" se muestran las vocales reconocidas en el archivo antes elegido.

El sistema adquiere los datos del archivo de audio, en seguida utiliza la herramienta de segmentación de este *toolkit* para obtener segmentos de 30ms, mismos que son introducidos en la herramienta de LPCs con un orden 12. Se obtienen 13 coeficientes de predicción lineal, los cuales, para el caso de identificación por LPCs son la entrada a la red neuronal. Si se elige el reconocimiento por Cepstrum, los coeficientes de predicción lineal son introducidos a la herramienta llamada *Cepstrum* y se obtienen 18 coeficientes Cepstrales para cada muestra de 30 ms.

Para entrenar la red neuronal del sistema se utilizó la herramienta de entrenamiento por propagación hacia atrás (Backpropagation). Para el caso de LPCs se diseñó una red con dos capas ocultas, una de 36 y otra de 10 neuronas y se obtuvo un error menor a 0.2. En el caso de coeficientes Cepstrales se diseñó una red de la misma cantidad de capas intermedias y número de neuronas, el error obtenido fue cercano a 0.5.

4.8.1. Resultados del sistema

A continuación se muestran los resultados obtenidos para ambos métodos y con los mismos archivos de audio.

Palabra introducida	Resultados con LPCs	Resultados con Cepstrum
aeiou	AEIOU	AEIOU
bebido	EIO	EII
adicto	AIO	AI
ahilar	AIA	AIA
bala	AA	A
besar	EA	EA
cupo	UO	UI
abeja	AE	AE
abrigo	IU	UII
foco	OU	O
un almacén de equipos electrónicos	UAAEEIEE	IAEEEIIEUII

Tabla 4: Resultados del sistema para 11 archivos de prueba.

Como se puede apreciar, el sistema basado en LPCs generó resultados con mayor fiabilidad que el realizado con Cepstrum. Lo anterior va en contra de la teoría y puede deberse a que la discriminación del sistema con Coeficientes Cepstrales es mayor, de modo que es más complicado lograr que dos vocales concuerden totalmente. A pesar de que el error obtenido en el entrenamiento fue bajo, no pudo lograrse un mejor resultado que el dado por Coeficientes de Predicción Lineal. Lo anterior no es de gran importancia debido a que el propósito de este sistema era simplemente ilustrar una de las funcionalidades de las herramientas desarrolladas para este *toolkit*.

Capítulo 5: Conclusiones y trabajo futuro

5.1. Conclusiones

En este documento se presentó la información necesaria para dar a conocer el procedimiento del desarrollo del "toolkit de procesamiento de voz para LabVIEW", así como validar el correcto funcionamiento de cada una de las herramientas desarrolladas. Retomando los objetivos iniciales se puede determinar que se han alcanzado.

Se han generado herramientas que al usuario se presentan sencillas y claras, ya que sólo basta una sección de este documento para que el toolkit pueda utilizarse. De igual modo, las herramientas son intuitivas gracias a los nombres de las entradas y los íconos.

Después de un retraso en el avance del proyecto, gracias a los problemas de *LabVIEW* con recursiones, se pudo lograr que todas las VI's desarrolladas sean eficientes y no generen carga excesiva de procesamiento, cumpliendo con el objetivo de la eficiencia en cuanto al uso de recursos computacionales.

Como ya se mostró en los diagramas a bloques, se utilizan herramientas básicas de *LabVIEW*, de modo que la versión más simple de éste pueda ejecutar todas las VI's presentadas en este documento. Así mismo, estas herramientas pueden ser descargadas a sistemas de hardware externos a la computadora como tarjetas de desarrollo y sistemas basados en DSPs.

Finalmente, con los diagramas presentados de cada una de las herramientas y las ilustraciones de su aplicación, podemos notar que este toolkit es de gran utilidad para evitar gastos innecesarios de tiempo al realizar una aplicación orientada al procesamiento de voz.

5.2. Trabajo futuro

Desgraciadamente, un año es insuficiente para desarrollar un toolkit que posea todas las herramientas que conforman al procesamiento de voz, por lo que este proyecto quedará abierto a recibir nuevas propuestas y desarrollo de herramientas requeridas en el campo del procesamiento de voz.

Entre las herramientas propuestas para un futuro se encuentran métodos de caracterización más complejos que los aquí presentados como *Wavelets, Máquinas de Soporte Vectorial,* entre otras. En cuanto a métodos de clasificación de proponen *Modelos Ocultos de Markov, Modelos de Mezclas Gaussianas* y más.

Así mismo se propone descargar las herramientas desarrolladas a plataformas de hardware basadas en DSPs o FPGAs para completar la investigación y comprobar su correcto funcionamiento en entornos distintos a la computadora personal.

Bibliografía

Babylon.com. (2007). *Muestreo digital.* Recuperado el 23 de Marzo de 2009, de Dictionary and translation by Babylon: http://www.babylon.com/definition/muestreo_digital/Spanish

Childers, D. (1999). *Speech Processing and Synthesis Toolboxes.* Wiley.

GA. (2003). *La voz humana.* Recuperado el 25 de marzo de 2009, de La voz humana: http://www.ehu.es/acustica/espanol/musica/vohues/vohues.html

Mantilla, A. (2007). *Análisis, reconocimiento y síntesis de voz esofágica.* México, D.F.

National Instruments Corporation. (2009). *¿Qué es LabVIEW?* Recuperado el 3 de Abril de 2009, de NAtional Instruments: http://www.ni.com/labview/whatis/esa/

National Instruments-Products and Services. (2009). Recuperado el 17 de abril de 2009, de What Is LabVIEW?: http://www.ni.com/labview/whatis/

ppgb. (9 de marzo de 2003). *Sound.* Recuperado el 17 de abril de 2009, de Sound: Filter (pre-emphasis): http://uvafon.hum.uva.nl/praat/manual/Sound__Filter__pre-emphasis____.html

The MathWorks, I. (2009). *Documentation.* Recuperado el 6 de abril de 2009, de Signal processing blockset: http://www.mathworks.com/access/helpdesk/help/toolbox/dspblks/index.html?/access/helpdesk/help/toolbox/dspblks/ref/lpctofromcepstralcoefficients.html

Anexos.

Anexo 1: código en Matlab.

Función "formant" para cálculo de formantes.

```
function [ffreq]=formant(x1)
ffreq=zeros(5,1);
x=x1;
fs=8000;
%get Linear prediction filter
ncoeff=2+fs/1000;        % rule of thumb for formant estimation
a=lpc(x,ncoeff);
[h,f]=freqz(1,a,512,fs);
% find frequencies by root-solving
r=roots(a);        % find roots of polynomial a
r=r(imag(r)>0.01);        % only look for roots >0 up to fs/2
ffreq=sort(atan2(imag(r),real(r))*fs/(2*pi));% convert to Hz and sort
```

Prueba para cálculo de LPC's.

```
input = csvread('prueba2.csv',0);
output = lpc(input,15)
```