Trabajo de Grado

"Algoritmos en SIGNAL
para el
procesamiento de señales de voz"

Autores:

Samantha G. Herrero

Guillermo F. Heroles

Director:

Ing. Oscar N. Bria

Co-Director:

Lic. Fernando G. Tinetti

Facultad de Ciencias Exactas

Universidad Nacional de La Plata

1996

A nuestros Padres...

Prefacio

El trabajo de investigación aplicada que se presenta a continuación tiene como objetivo profundizar en dos grandes temas como son el procesamiento de señales de voz y el lenguaje de especificación formal SIGNAL, para luego tratar de paralelizar ciertas rutinas inherentes a técnicas de análisis de señales de voz; específicamente aquellas relacionadas con el Análisis de Predicción Lineal.

Los capítulos iniciales detallan las características e hipótesis en las cuales se basa el lenguaje SIGNAL. Así, en el Capítulo 1 se incursiona en el área de las arquitecturas no convencionales, poniendo énfasis en aquellas Data-Flow, donde la ejecución de un programa depende de la disponibilidad de datos.

A continuación, el Capítulo 2 ofrece una interiorización sobre Sistemas Reactivos y de Tiempo Real, como así también sobre sus formas de especificación, ya sea a través de Formalismos Basados en Estados o Sistemas Recurrentes de Múltiples Relojes. Se ubica a SIGNAL dentro de este último estilo de modelización.

El Capítulo 3 describe SIGNAL. Se especifica cómo dicho lenguaje maneja el tiempo, su filosofía de procesamiento y los distintos operadores. Además se investiga el trabajo del compilador que asegura la ausencia de deadlock en un proceso dado, utilizando como herramientas básicas los sistemas de ecuaciones de reloj y los grafos de dependencias condicionales de dicho proceso.

Las características básicas del Procesamiento de señales de voz son presentadas en el Capítulo 4, focalizando la atención en características Short-Term y Long-Term.

En el capítulo 5 se detalla en qué consiste Análisis de Predicción Lineal como técnica de análisis de señales de voz. Puntualmente se estudia una solución al Problema Clásico de Mínimos Cuadrados pues mantiene una relación que involucra un conjunto de resultados intermedios y prácticos coincidentes con los

propios del Método de la Covarianza (uno de los métodos para hallar coeficientes de Predicción Lineal). Dicha solución es planteada recursiva en el tiempo.

El objetivo final de este trabajo de investigación se alcanza en el Capítulo 6 donde, aprovechando las facilidades del lenguaje SIGNAL, se obtiene una versión paralelizada de la solución al problema de Mínimos Cuadrados propuesta en el Capítulo 5. Además de algunas rutinas clásicas para el procesamiento de señales de voz (filtrados, cálculos de la medida de cruces por cero, etc.), también se implementa un proceso para la resolución de sistemas de ecuaciones vía sustitución backward, donde no es posible paralelizar completamente la tarea debido a la dependencia de datos que por naturaleza presenta el algoritmo.

Finalmente se adjunta un Apéndice que explica sintéticamente las llamadas Rotaciones de Givens, utilizadas para transformar una matriz general en triangular mediante rotaciones de plano ortonormales. Esta última información es aportada para proveer un mayor grado de entendimiento de la mecánica utilizada en el algoritmo de cálculo de Coeficientes de Predicción Lineal.

S.G.H

G.F.H

Agradecimientos

Nuestro especial agradecimiento para el Ing. Oscar N. Bria y el Lic. Fernando G. Tinetti, quienes contribuyeron con su dedicación y disponibilidad durante el desarrollo de este trabajo.

Queremos, además, dar gracias al Ing. Antonio A. Quijano por habernos permitido contar con toda la infraestructura del CeTAD y con todo el material bibliográfico utilizado en la investigación.

Finalmente, por tu paciencia, muchas gracias Claudia.

S.G.H.
G.F.H.

Indice

Capítulo 3: Programación Data-Flow Sincrónica: El lenguaje SIGNAL

Capítulo 4 Procesamiento de señales de voz

Capítulo 5: Análisis de Predicción Lineal

Capítulo 6: Algoritmos en SIGNAL

Capítulo 1

Arquitecturas no convencionales

1.1- Arquitecturas Data-Flow:

Un procesador basado en arquitecturas Data-Flow lleva a cabo el procesamiento controlado por la disponibilidad de datos (operandos).

Existen grandes diferencias entre estas arquitecturas y las de Von Neumann ya que en estas últimas la ejecución de las instrucciones está bajo el control del programa.

Veamos un ejemplo:

$$a = (b + 1) * (b - c).$$

En un modelo clásico secuencial, la ejecución sería la siguiente:

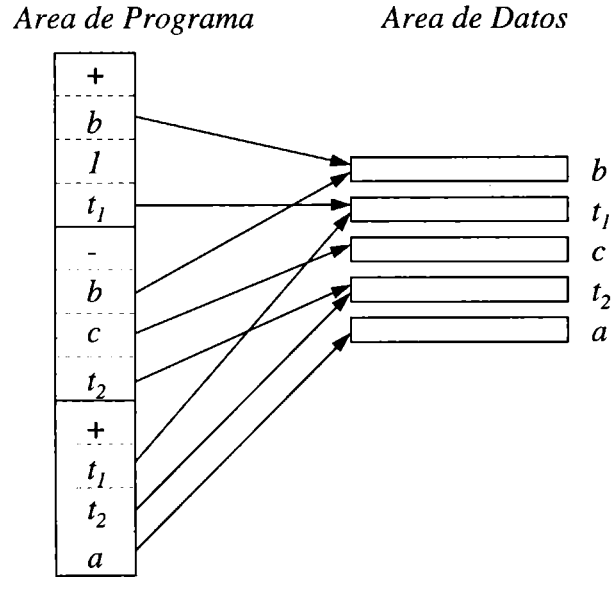


Fig. 1.1 Modelo secuencial.

Vemos que hay una sola dirección de control que va de instrucción en instrucción.

En un modelo Data-Flow la ejecución sería

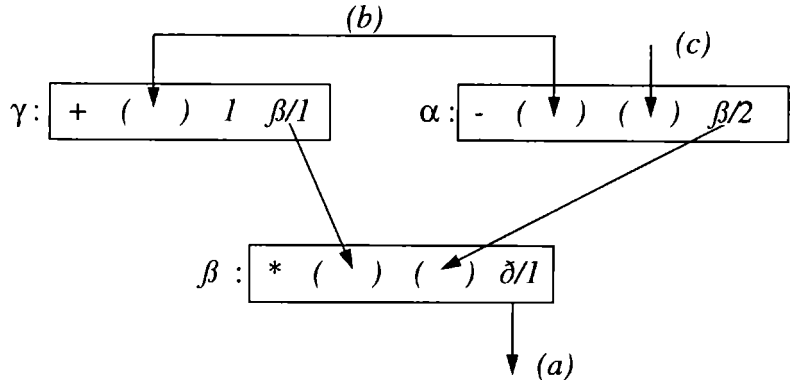


Fig. 1.2 Modelo Data-Flow.

gobernada por la disponibilidad de datos simbólicos (tokens) indicados por paréntesis. Los programas por flujos de datos se representan por grafos orientados que muestran el flujo de datos entre instrucciones.

Basándonos en este ejemplo podemos decir :

- los resultados finales o intermedios se transfieren como "data tokens" entre las instrucciones.
- no existe el concepto de almacenamiento de datos compartidos, como con la noción de variable.
- la secuencia de programa está sujeta a la dependencia de datos entre instrucciones.

Bajo esta idea muchas instrucciones pueden ejecutarse simultáneamente y asincrónicamente.

En estas máquinas tenemos dos tipos de información *"paquetes de operación "* y *"data tokens"*. Los primeros constan de *CODOP, OPERANDOS, CODINSTSIG;* mientras que los segundos están formados por *VALORRESULTANTE, DIRDEST.* Esto nos permite asumir una arquitectura de comunicación de paquetes, típico de una comunicación de multiprocesadores.

1.2 Arquitecturas estáticas y dinámicas

Considerando la forma en que se manejan los data tokens, estas arquitecturas se dividen en:

1.2.1 Arquitecturas estáticas

Veamos el siguiente diagrama:

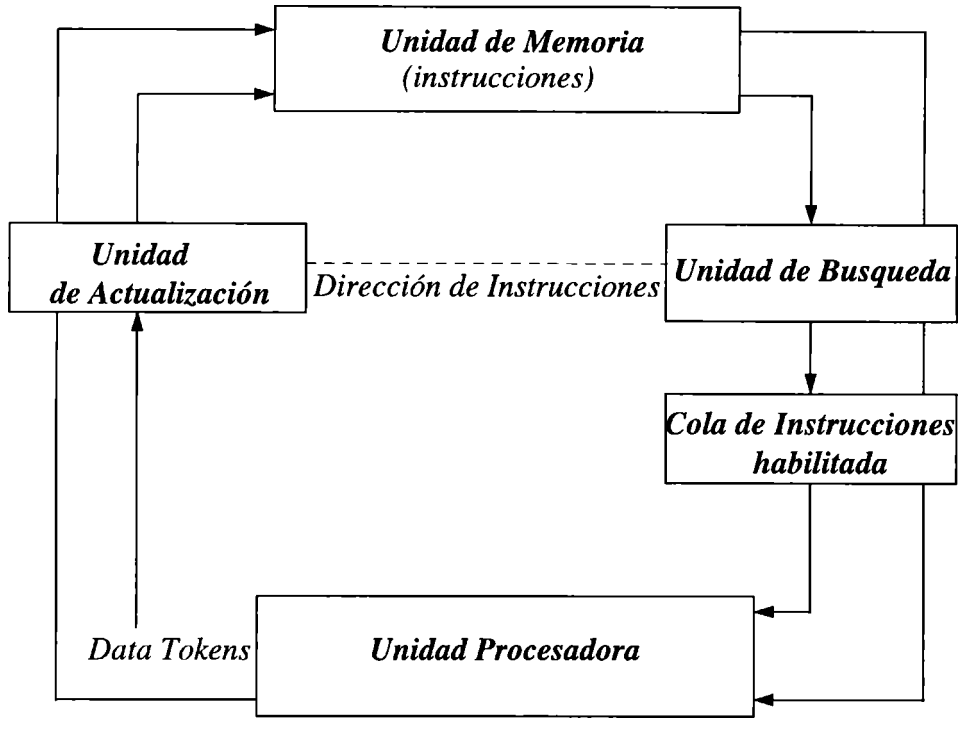


Fig.1.3 Arquitectura Data-Flow estática.

Sólo se permite que un token esté presente en todo arco en cualquier momento (instante), de lo contrario los conjuntos de tokens sucesivos no podrían distinguirse. Debido a esto se considera a la arquitectura, estática, ya que no se identifican los tokens y deben utilizarse tokens de control para lograr que los mencionados tokens de dato pasen de nodo a nodo en el momento adecuado.

En esta estructura los data tokens están en la entrada de la *Unidad de Actualización*, la cual transfiere los tokens a sus instrucciones de destino en la *Unidad de Memoria.*

Cuando una instrucción recibe todos sus operandos, es habilitada y pasa a la cola de instrucciones, siendo controlada por la *Unidad de Actualización.*

1.2.2 Arquitecturas Dinámicas

La organización dinámica utiliza tokens identificados de modo que más de un token puede existir en un arco. La identificación consiste en agregar a cada token un rótulo. De esta manera permite lograr un máximo de paralelismo. Este tipo de arquitectura puede ser ilustrada de la siguiente manera:

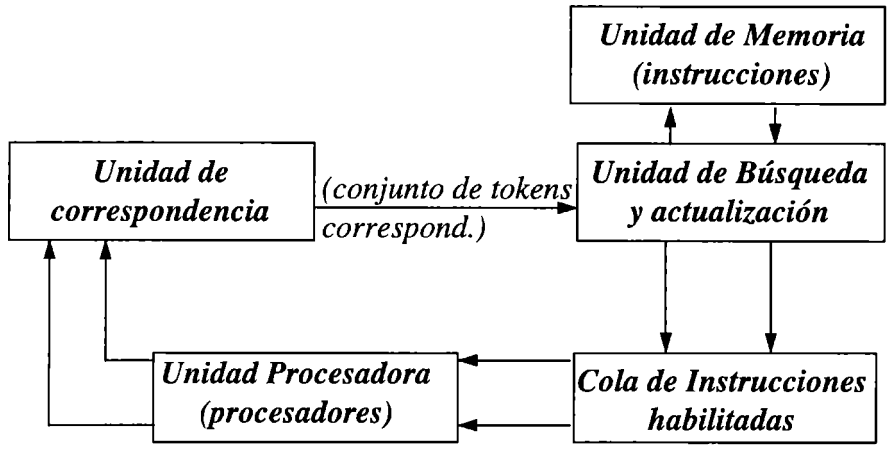


Fig. 1.4 Arquitectura Data-Flow dinámica.

Los tokens de datos están en la entrada de *Unidad de Correspondencia*, la cual los agrupa en pares o conjuntos y almacena temporariamente hasta que se comparan todos los operandos. A partir de ese momento los conjuntos de tokens correspondientes se transfieren a la *Unidad de Búsqueda y Actualización*, la cual forma las instrucciones habilitadas, distribuyendo los conjuntos de tokens.

1.2.3 Arquitetura resultante

Si se tiene en cuenta la sección de Entrada/Salida se puede obtener una estructura genérica de ambas configuraciones, la cual presenta una arquitectura de anillo en *pipeline* :

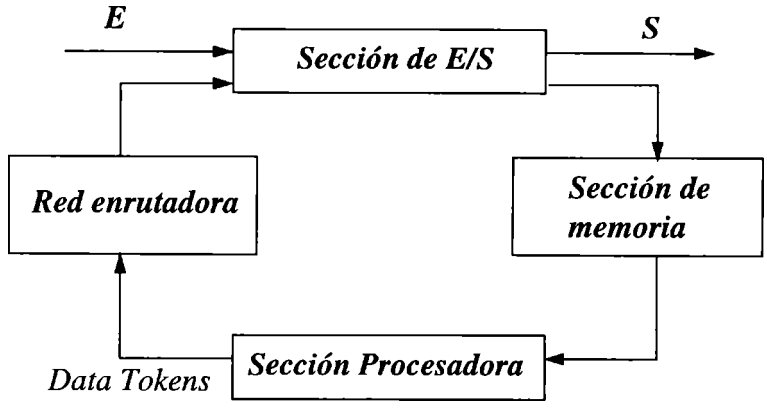


Fig. 1.5 Arquitectura Data-Flow genérica en anillo.

1.2 Lenguajes y Grafos Data-Flow

Así como en las máquinas secuenciales los programas suelen representarse por medio de diagramas de flujo, en aquellas Data-Flow, los mencionados programas se representan por grafos de flujos de datos, cuyos nodos y arcos corresponden a *operadores* y encadenamientos de *data tokens,* respectivamente.

1.2.1 Grafos de Flujos de Datos

Veamos un ejemplo:

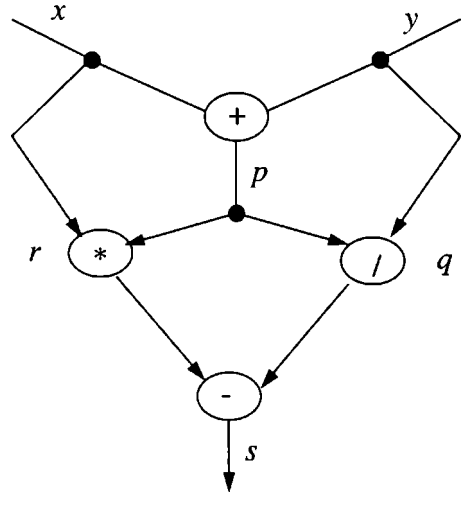
$$p := x + y$$

$$q := p / y$$

$$z := x * p$$

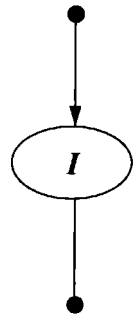
$$s := r - q$$

el grafo de flujo de datos es:

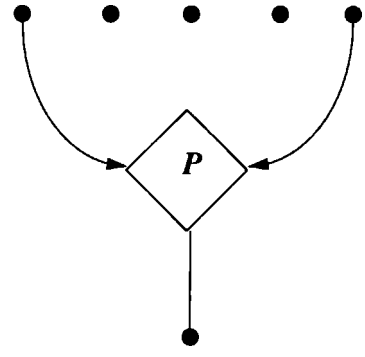


Otros elementos de un grafo son:

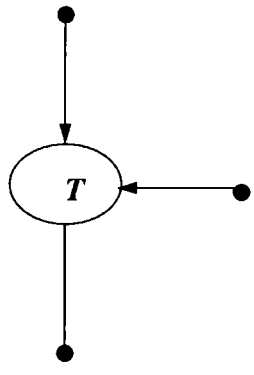
Identidad



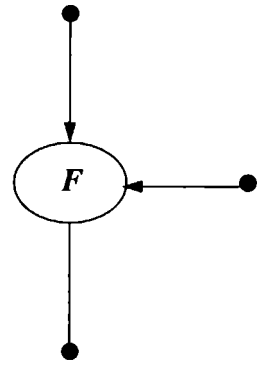
Comparador



Compuerta T (verdadera)

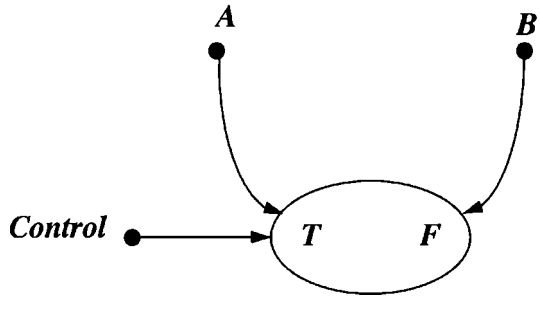


Compuerta F (falsa)



Si control es verdadero (falso) la compuerta (no) deja pasar el valor.

Distribuidor



Si control es T, entonces A; si control es F, entonces B.

1.2.2 Propiedades de los lenguajes

- Localidad: se deben evitar variable comunes y asignaciones globales, por lo tanto, la programación estructurada sigue teniendo vigencia *(ALGOL, PASCAL, etc).*

- Libre de efectos laterales: Debido a la localidad es posible evitar efectos laterales que pueden aparecer en procedimientos que modifican variables del programa principal. La llamada por valor hacen que la entrada y salida de un procedimiento estén totalmente aisladas.
- Asignación única: consiste en prohibir el uso de una variable a la izquierda de una sentencia. Veamos un ejemplo:

$$\begin{array}{lcl} x := p - q & & x:=p - q \\ x:=x * y & => & x_1:=x * y \\ w:=x - y & & w:= x_1 - y \end{array}$$

Desdoblamiento de iteraciones: las técnicas propuestas se basan en la identificación de los tokens. Un token así identificado consta de dos partes (*datos, nombre de la unidad de destino).* El segundo es un nombre único que identifica al contexto en el cual un bloque de código es llamado, el nombre de dicho bloque y el número de iteración asociado con el índice del loop. Así se desdoblarán los procesos iterativos en forma recurrente, ya que el propio contexto puede ser un operador. El hardware correspondería a una arquitectura dinámica.

Capítulo 2

Sistemas Reactivos y de Tiempo Real

2.1- Conceptos básicos

Un Sistema de Tiempo Real (STR) se puede considerar como un programa que recibe interrupciones externas o lee sensores conectados al mundo exterior, generando como salida comandos. Es controlado por software o firmware cumpliendo funciones restringidas temporalmente.

Por otro lado un Sistema Reactivo (SR) es aquel que mantiene una interacción con su ambiente; y si además está sujeto a restricciones de tiempo externas es considerado de Tiempo Real (TR).[4]

Ambos se caracterizan por no tener una secuencia de entradas temporales predeterminada lo cual conduce a *sincronizar eventos* en el tiempo, tratando de evitar *deadlocks* u *overloadings* . Además es necesario proveer un alto grado de confiabilidad, auto-recuperación de situaciones anormales, redundancia de hardware y software, etc. pues son sistemas infinitos.

Un programa de TR lógicamente correcto puede fallar si su ambiente controla que la salida no es producida a tiempo, por lo tanto no alcanza con *verificar* o *validar* funcionalmente; pues la correción operativa de un sistema es necesaria pero no suficiente para garantizar su utilización en condiciones reales.

2.2- Areas de aplicación

Tales áreas en orden de complejidad son:

- *Secuenciadores de tareas puras*: distintas secuencias pueden ocurrir en paralelo y cooperar vía eventos compartidos. El objetivo principal aquí es proveer un método formal para traducir una especificación en código eficiente.

- *Protocolos de comunicación:* son encontrados en distintas clases de redes.
- *Control de proceso industrial:* implica reguladores los cuales son supervisados a través de interrupciones generadas desde el propio sistema o recibidas desde el ambiente. El objeto principal es proveer una herramienta para soportar la especificación y garantizar que la implementación actual está basada en dicha especificación.
- *Sistemas de procesamiento de señales complejas:* los sistemas de radar por ejemplo utilizan pre-procesadores de señales seguida de una compresión y procesamiento de datos complejos. Todo esto describe claramente un sistema de tiempo real donde los eventos son generados, provocando nuevas operaciones.
- *Sistemas complejos de monitoreo y de control:* aquí se emplean ciertos sensores y conexiones entre ellos. Los datos son procesados en distintos modos de operación. Heurísticas de alta complejidad conforman el código en su gran mayoría.
- *Sistemas de comunicación, control y comandos:* es el caso de los sistemas militares y de control de tráfico aéreo; los cuales tienen por naturaleza arquitecturas distribuidas para su soporte.

2.3-Principales problemas y estrategias de solución

Estos sistemas son descompuestos en pequeños módulos concurrentes comunicantes. Todos los aspectos relacionados con la concurrencia son importantes:

* *comunicación:* que involucra el procesamiento de información que arriva en forma asincrónica teniendo en cuenta restricciones de tiempo.
* *sincronización:* diseñar condiciones de concurrencia en un conjunto apropiado de procesos concurrentes.

* *organización del flujo computacional:* manejo de estructuras de almacenamiento y protección de datos compartidos por procesos concurrentes.

Algunas estrategias adecuadas para cumplir con los aspectos antes mencionados son:

-Técnicas de uso modular y formal para especificar y verificar programas: las etapas de especificación e implementación de un problema son las más difíciles pues es allí donde se debe mostrar la arquitectura conceptual, manteniendo sumamente ligadas ambas etapas en el desarrollo de un sistema.

-Centrar las características reactivas en un "framework": sólo se justifica mantener varios *"framework"* si la interfase entre los mismos está bien definida, para evitar fallas en la cadena de comunicación global.

-Considerar problemas de velocidad: los tiempos de ejecución deberían ser predecibles, originando así un código ejecutable lo suficientemente eficiente como para evitar retardos en comunicaciones innecesarias.

-Mantener un cierto grado de determinismo: esto es que una secuencia de entradas produzca la misma secuencia de salidas. Desde que los programas determinísticos son los más fáciles de analizar, las herramientas utilizadas para su desarrollo no fuerzan el no determinismo.

-Manejo de arquitecturas distribuidas: necesarias sólo cuando se manejan restricciones geográficas en la aplicación o simples requerimientos de performance.

Estas estrategias abarcan la aplicación de técnicas como: la realización de conexiones clásicas utilizando primitivas del Sistema Operativo (caracterizado por ser no deteminístico); utilización de Redes de Petri, las cuales tienen como desventaja su utilización en aplicaciones pequeñas y la carencia de estructuras modulares y determinismo; el empleo de lenguajes de programación concurrente en los cuales es característico la concurrencia modularidad, asincronismo y determinismo *(OCCAM, ADA, etc)*; y la aplicación de autómatas finitos los cuales son determinísticos y pueden ser analizados por varios sistemas de

verificación en forma automática, aunque carecen de soporte para manejar concurrencia y diseño jerárquico.

2.4-Especificación, diseño e implementación de Sistemas Reactivos y de Tiempo Real : Aproximación sincrónica

La idea es considerar un SR que genera salidas sincrónicamente con sus entradas, donde su reacción toma tiempo nulo. En una aproximación sincrónica un sistema puede ser considerado como un conjunto de componentes concurrentes los cuales hacen al comportamiento general del mencionado sistema.

Para observar la hipótesis sincrónica de un sistema reactivo podemos plantear al mismo a través de diagramas de transición de estados y sistemas de ecuaciones recurrentes. Consideremos ahora un filtro digital de señales para ejemplificar dichas formas de especificación.

2.4.1 Filtrado digital : sistema de ecuaciones recurrentes

Este sistema es un modelo sincrónico el cual trata con un sistema de ecuaciones interconectadas (diagrama en bloque del procesamiento de la señal), o simplemente es una descripción de acuerdos.

La siguiente figura muestra un grafo de flujo de la señal de un filtro digital de segundo orden:

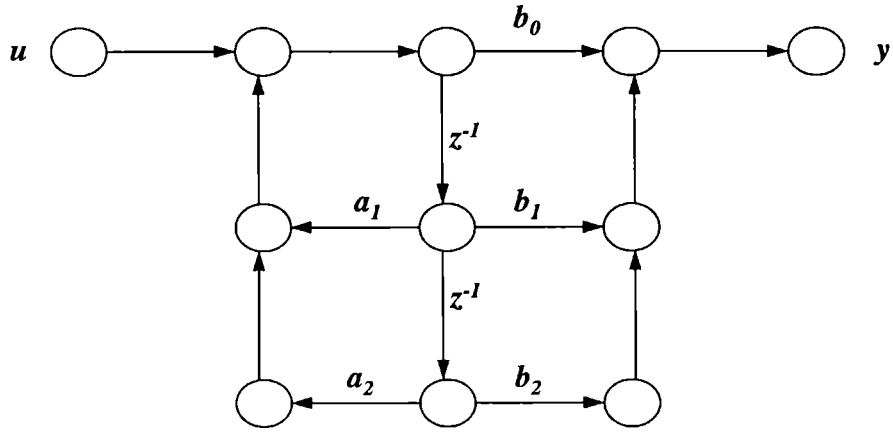


Fig. 2.1 Grafo de flujo de un filtro digital de segundo orden.

En los nodos de este grafo, las señales entrantes son adicionadas y el resultado es pasado a lo largo de los brazos salientes. Las etiquetas z^{-1} , a_i y b_j sobre los arcos denotan un "registro de corrimiento" y una multiplicación por las mencionadas constantes acrecentadas, respectivamente. El grafo representado anteriormente es una codificación de la siguiente fórmula:

$$w_n = u_n + a_1 * w_{n-1} + a_2 * w_{n-2} \qquad (2.1)$$

$$y_n = b_0 * w_n + b_1 * w_{n-1} + b_2 * w_{n-2}$$

donde n denota el índice del tiempo. Luego :

$$y_n = a_1 * y_{n-1} + a_2 * y_{n-2} + b_0 * u_n + b_1 * u_{n-1} + b_2 * u_{n-2} \qquad (2.2)$$

Esta expresión matemática puede ser ejecutada por una máquina la cual involucra un filtrado de acuerdo a los principios que a continuación se detallan:

1) los cambios de estados en cada uno de los módulos deberían ser considerados como sincrónicos con la recepción de la señal de entrada;
2) el resultado generado como salida debería ser sincrónico a la señal de entrada y
3) las comunicaciones deberían seguir el principio de *broadcast* instantáneo, así su recepción es sincrónica con su emisión.

La interconexión de filtros digitales es especificada por grafos de enlace. Por ejemplo el filtro antes mencionado puede ser redibujado en un modo más modular como sigue:

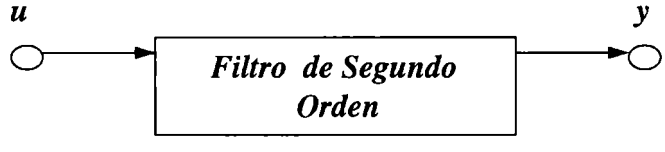


Fig. 2.2 Diagrama modular del filtro de segundo orden.

2.4.2 Filtrado digital: diagrama de transición de estados

Nos basaremos en un filtro más simple que (2.2):

$$y_n = a_1 * y_{n-1} + a_2 * y_{n-2} + u_n \qquad (2.3)$$

introducimos una señal vector :

$$\mathbf{X}_n = \begin{bmatrix} y_n \\ y_{n-1} \end{bmatrix} \tag{2.4}$$

y rescribimos la ecuación *(2.3)* así:

$$\mathbf{X}_n = \begin{bmatrix} a_1 & a_2 \\ 1 & 0 \end{bmatrix} * \mathbf{X}_{n-1} + \begin{bmatrix} u_n \\ 0 \end{bmatrix} \tag{2.5}$$

$$\mathbf{Y}_n = \begin{bmatrix} 1 & 0 \end{bmatrix} * \mathbf{X}_n \tag{2.6}$$

Un paso en el tiempo del sistema *(2.6)* será escrito en un lenguaje de programación secuencial, como sigue:

$$\mathbf{X}:= \begin{bmatrix} a_1 & a_2 \\ 1 & 0 \end{bmatrix} * \mathbf{X} + \begin{bmatrix} u \\ 0 \end{bmatrix} ; \tag{2.7}$$

$$y:= \begin{bmatrix} 1 & 0 \end{bmatrix} * \mathbf{X} \tag{2.8}$$

Este programa es de la forma *(2.7)*; *(2.8)*, esto es, está compuesto de dos instrucciones separadas por el secuenciador tipo *Pascal* , ";". Denotemos por α la acción ejecutada por este programa; entonces el diagrama de transición de estados para el sistema *(2.6)* es :

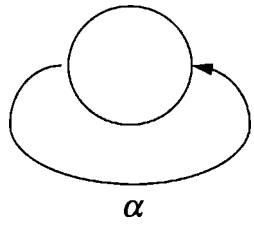


Fig. 2.3 Diagrama de transición de estados para (2.6).

El estado cuenta las ocurrencias de la señal de entrada. La tarea es resumida por la etiqueta α de la acción *(2.7)*; *(2.8)* la cual corresponde a una iteración simple de la ecuación recurrente.

2.4.3 Modelos sincrónicos para especificar una máquina ideal de TR

Dos modelos sincrónicos diferentes en estilo pero de forma equivalente pueden ser usados para especificar una máquina ideal de TR con las siguientes características :

1) la salida es sincrónica con la entrada, las acciones internas son instantáneas y las comunicaciones son ejecutadas vía *broadcasting*[1] instantáneo.
2) el *interleaving* global de las comunicaciones externas puede ser elegido parcialmente por el ambiente y es esencial en el análisis del comportamiento del sistema.

Los dos estilos son: *formalismos basados en estados* y *Sistemas Recurrentes de Múltiples Relojes* (SRMR) . El primer estilo es fácil y natural para usar en problemas donde predomina el flujo de control. El segundo estilo es una forma para describir los pasos legales de un sistema, y son generalizaciones de los modelos usuales de sistemas dinámicos usados en procesamiento de señales digitales o de control. El lenguaje SIGNAL se apoya sobre este estilo de modelización.

Los SRMR son claramente adaptables a problemas donde predomina Data-Flow, siendo procesamiento de señales un buen ejemplo. La composición de estos sistemas es fácil de definir desde que los mismos son sistemas de ecuaciones matemáticas estandar. Por otra parte, son débiles donde las aproximaciones basadas en estados son fuertes, esto es, cuando la complejidad está dada en función de los cambios de modo. Entonces el usuario debe manejar variables de control explícitas para almacenar el modo corriente, lo cual no es una tarea fácil.

En ambos estilos, las ecuaciones de comunicaciones pueden : *no tener solución*, cuando las restricciones se contradicen o existen ciclos de causalidad que no pueden ser resueltos usando algoritmos finitos, afectando al sistema entero o sólo a una parte del mismo; *tener infinitas soluciones,* cuando el tiempo de las distintas señales no es determinado por las entradas dadas, obteniendo así no determinismo; o *tener una solución simple,* lo cual es un mapeo de entradas y salidas del sistema, siendo el mismo de características determinísticas y con una ejecución apropiada.

[1] Envío de información simultánea a los distintos módulos del modelo.

En la mayoría de los SR y STR es importante poder verificar formalmente las propiedades de *liveness*[2] *o safety*[3] de un programa respecto de especificaciones totales o parciales. En formalismos SRMR, tales como SIGNAL y LUSTRE, restricciones sobre propiedades pueden ser especificadas como ecuaciones dinámicas que deben ser implicadas por el sistema dado. Entonces no hay distinción profunda entre un programa con propiedad *safety* y un compilador de programa estandar que actue como verificador.

2.4.4 Hardware de soporte

Existen máquinas en la actualidad para las cuales el modelo sincrónico ideal es realístico; hardware sincronizado fuertemente o arquitecturas VLSI son tales que acciones internas y comunicaciones ocurren dentro de un ciclo de reloj. La única diferencia es que las salidas son entregadas al ambiente al final del ciclo y no sincrónicamente con las entradas. Desde que el ciclo de reloj es muy corto, digamos 100 ns, ésta es la mejor aproximación que podemos obtener.

Los STR son implementados sobre arquitecturas distribuidas, esto es, conjunto de procesadores conectados por medios asincrónicos.

Los modelos sincrónicos, como fueron introducidos antes pueden ser considerados como reales o válidos para estas arquitecturas.

2.4.5 Alternativas de implementación para el filtro digital.

Para elaborar una implementación realista presentamos dos alternativas :

- implementación puramente secuencial derivada del grafo de señales de la fig. *(2.1)*: se considera previamente el grafo de dependencias, cortando los brazos etiquetados con un retardo (z^{-1}) y se obtiene un grafo dirigido acíclico. Desmembrando este grafo, removiendo primero los nodos de entrada y luego los

[2] Propiedad que asegura a un programa entrar en buen estado, esto es, uno en el cual todas las variables tienen valores deseados.

[3] Propiedad qye asegura a un programa nunca entrar en un estado malo, esto es, uno en el cual alguna variable tenga valores indeseados.

subsecuentes da un esquema de ejecución secuencial de cada paso simple del sistema:

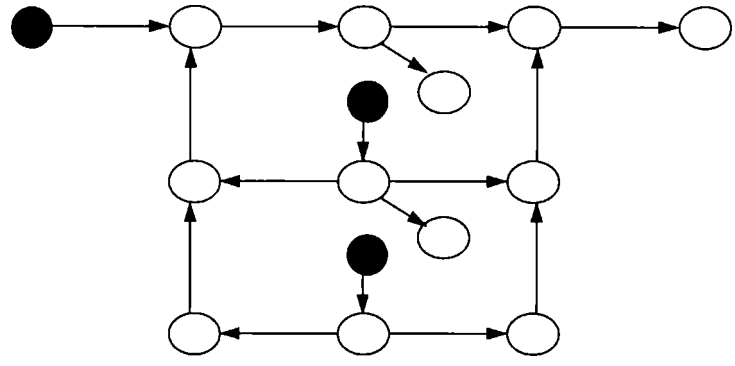


Fig. 2.4 Desmembramiento del grafo de la figura (2.1).

- Una ejecución Data-Flow (asincrónica):derivada interpretando cada nodo y brazo en la Fig. *(2.1)* de acuerdo al mecanismo de Data-Flow presentado en el Capítulo 1. Lo importante aquí es que conocemos antes de la ejecución que este mecanismo de *tokens* será no bloqueante y con archivos acotados.

Esta habilidad para validar ejecuciones sincrónicas de nuestras máquinas sincrónicas ideales permite para los sistemas reactivos que nosostros podamos modelar con nuestra aproximación.

2.4.5 Proyecciones de los formalismos sincrónicos sobre la programación en tiempo real

Aunque cuentan con una semántica matemática clara, además de estar basados en conceptos avanzados y potentes, los formalismos sincrónicos carecen de una apropiada metodología de programación e interfase de usuario.

Algunos formalismos son puramente gráficos (STATECHARTS), algunos puramente textuales (Esterel) y otros presentan interfases textuales o gráficas indistintamente (SIGNAL, LUSTRE). SIGNAL tiene una interfase orientada a Data-Flow, STATECHARTS es orientado a estados.

Los diagramas orientados a estado son pobres para el procesamiento de señales y los diagramas en bloque, para máquinas de estado; por lo tanto ninguna de estas elecciones cubren el área completa de STR y SR.

En cuanto a la metodología de programación, la mayoría de los usuarios se orientan a proceso más que a la ciencia de la computación, pues comunmente los STR son pensados utilizando un razonamiento basado en estados más que a partir de la manipulación de estados.

Es importante notar que los lenguajes sincrónicos no estan ligados al no determinismo. Algunos de ellos aceptan programas no determinísticos como módulos pero sin embargo rehusan producir código determinístico fuera de ellos. Los módulos no determinísticos pueden ser muy útiles para modelar el ambiente o los procesos físicos controlados. Esto podría ser la base para una metodología de diseño de software de TR, basado en una manipulación conjunta de la aplicación y de un modelo del proceso físico.

Capítulo 3

Programación Data-Flow sincrónica: El lenguaje SIGNAL

3.1- Preliminares

En una aplicación en Tiempo Real (TR) se manejan secuencias de valores y se tiene en cuenta la idea de tiempo haciendo depender los resultados de dicha aplicación de un orden temporal. En las aproximaciones sincrónicas, mencionadas en el Capítulo 2, se permite considerar al tiempo como un valor relevante dentro de la información de la aplicación, el cual define el orden parcial de la ocurrencia de *eventos*[1] que darán lugar a posteriores cálculos. De esta manera, el tiempo se caracteriza a través de:

- orden parcial de eventos;
- simultaneidad de eventos y
- retardo entre eventos.

En un framework sincrónico, el tiempo es una cronología donde las duraciones son restricciones a ser verificadas en la implementación; de esta manera es posible establecer que los cálculos tienen duración nula y expresar recurrencia sólo temporalmente.

El comportamiento complejo de muchas aplicaciones de TR necesita un conjunto de herramientas de especificación para asistir en el diseño de problemas de tales características.

SIGNAL[2] ha sido definido desde un conjunto de operadores para el diseño de programas en TR que se ejecutan en un ambiente eventualmente distribuido. Este lenguaje declarativo está basado en principios Data-Flow sincrónicos. El hecho de ser declarativo le permite realizar transformaciones útiles a programas, es decir, asignar eficientemente algoritmos distribuidos sobre procesadores a

[1] Conjunto de comunicaciones simultáneas cada una de las cuales asocia en un instante lógico del programa, un valor a una variable. Un evento involucra al menos una comunicación.
[2] Lenguaje de especificación formal desarrollado por el INRIA (Institut National de Recherche en Informatique et en Automatique), Rocquencourt, Francia.

través de operadores de composición asociativos y conmutativos. Además, se pueden describir restricciones de tiempo entre las señales manejadas por las distintas partes de una aplicación y disponer de un compilador para traducir un scheduler mínimo.

3.2- El lenguaje SIGNAL

El lenguaje SIGNAL está definido a partir de un conjunto de operadores, los cuales constituyen su núcleo *(kernel)*. Esto permite definir semánticas formales simples y derivar en nuevos operadores.

El diseño de SIGNAL está ligado al concepto de *flujo (o "historia")* como es definido en la semántica de lenguajes Data-Flow. La principal diferencia está en la construcción de *flujos* a partir de *pistas (traces)*, donde una *pista* es un *flujo* en el cual el valor *bottom (⊥, estado para no evento)* puede ocurrir entre los valores definidos [3].

El estilo de los lenguajes sincrónicos puede ser clasificado en :

- *Imperativos*: basados en los modelos de las familias de máquinas de transición de estados.
- *Ecuacionales*: basados en los modelos de sistemas dinámicos de múltiples relojes interconectados.

A este último estilo pertenece SIGNAL, y debido a que su programación está vista como una especificación de restricciones o relaciones sobre señales, su compilador ejecuta cálculos formales sobre dependencias de datos, lógicas y de sincronización para chequear la correctitud del programa y producir código ejecutable [8].

La hipótesis de sincronismo se traduce en dos aspectos :

- en lo que concierne a los mecanismos internos del sistema: cada acción es instantánea (duración nula).
- en lo que concierne a las comunicaciones con el mundo exterior: las entradas y las salidas son fijadas por adelantado. El conjunto de

instantes donde los valores están disponibles en las puertas de entrada es totalmente ordenado.

La duración nula de la ejecución y el orden total de eventos aseguran el determinismo. Por otro lado la característica Data-Flow de SIGNAL, permite describir fácilmente el paralelismo. El conjunto de cálculos a efectuar está representado por un grafo dirigido en el cual los arcos son los caminos de los datos y los nodos las operaciones. Entonces en la ejecución, un nodo se activa cuando todas sus entradas están disponibles. Dentro de este contexto el concepto de variable es reemplazado por el de flujo; es por ello que SIGNAL maneja secuencias de datos (posiblemente infinitas) con temporarización implícita, las cuales se llaman *señales*. Los valores de cada una de ellas pertenecen a un mismo dominio predefinido (señales puras, booleanas, enteras, reales, complejas) o definidas en el programa (vectores) . Por ejemplo x denota la secuencia infinita $\{x_t\}_{t>0}$ donde el índice de tiempo t está asociado a esta señal.

Consideremos el siguiente ejemplo en algun lenguaje Data-Flow:

if $a > 0$ *then* $x := a$;

$y := x + a$

si los arcos son considerados FIFO[3] y "a" una secuencia con valores negativos, la cola de "a" crecerá por siempre. Además la cola de "x" será vacía, a pesar de que la cola de a no lo es.

Si cada cola FIFO consiste de una celda simple, tan pronto como un valor negativo aparece en la entrada, en un corto plazo la ejecución no podrá seguir, pues habrá *deadlock*. Esto es usualmente representado por el valor indefinido $\perp$.

Por lo anterior es necesario verificar estáticamente las propiedades de tiempo, entonces el valor $\perp$ será manipulado cuando razonemos acerca del tiempo.

[3] First In First Out.

En un ambiente Data-Flow sincronizado, $\perp$ corresponde a la ausencia de valor, en un instante lógico dado, para una señal. La resincronización se logra insertando $\perp$ entre dos valores definidos de una señal.

El principal propósito de Data-Flow sincronizado es que la sincronización sea manipulada en tiempo de compilación para que la fase de ejecución no tenga que tratar con el valor $\perp$.

En SIGNAL, una *señal* puede tener el estado *ausente* (representado por $\perp$) o *presente*. El reloj de una señal x es representado sintácticamente por P(x) especificando el conjunto de instantes relativos en los cuales la señal se encuentra presente o no. Si b es una señal booleana, entonces T(b) representa el conjunto de instantes donde la señal está presente y es verdadera. Si en un instante dado dos señales están *presentes,* entonces se dirá que estas poseen el mismo *reloj,* caso contrario sus relojes serán diferentes. Así dos señales serán sincrónicas si sus eventos son 2 a 2 simultáneos (presente al mismo tiempo) [19], esto es, si su índice de tiempo t es el mismo o si están relacionadas lógicamente de la siguiente manera:

> *$\forall t$ el t-ésimo token de la primer señal de entrada es evaluado con el t-ésimo token de la segunda señal de entrada para producir el t-ésimo token de la señal salida.*

Esto es precisamente la noción de simultaneidad. Además dados dos tokens en una señal, cronológicamente se dice que están uno después del otro. Entonces en la aproximación sincrónica, un evento es un conjunto de cálculos instantáneos o de comunicaciones instantáneas.

De esta manera los relojes pueden ser considerados como clases de equivalencia de señales que están siempre presentes simultáneamente. Así la noción de tiempo no hace referencia a un reloj universal sino que es local a cada subconjunto particular de señales, es decir a cada ambiente [5]. Por esta razón los operadores SIGNAL fueron pensados para relacionar relojes, como así también

valores de las distintas señales involucradas en sistemas dinámicos llamados comunmente Sistemas Recurrentes de Múltiples Relojes (SRMR).

Antes de introducirnos en el detalle de cada uno de los operadores del lenguaje, se mencionarán los tres tipos de relaciones que SIGNAL permite describir entre señales:

1. una relación de precedencia entre los eventos de una misma señal para que puedan ser contados y ubicados sus valores en el tiempo respecto de su llegada.
2. una relación de simultaneidad entre eventos de diferentes señales, para saber si ellas coinciden o son distintas.
3. relaciones de dependecias operativas cuya certeza está condicionada a la definición de las señales de salida resultantes de operaciones efectuadas sobre señales de entrada.

3.2.1 Operadores sobre señales

En esta sección describiremos informalmente los operadores que SIGNAL utiliza para la manipulación de señales, clasificándolos en dos grupos: *operadores monochronous* (operandos con reloj único) y *operadores polychronous* (operandos con relojes distintos).

3.2.1.1 Operadores monochronous

Las funciones del lenguaje deben ser fuertemente sincronizadas, es decir, en cualquier instante el t-ésimo elemento del resultado de dicha función se encuentra definido desde los t-ésimos elementos de los argumentos. Las funciones antes mencionadas no llevan tiempo, solamente una referencia del mismo es manipulada, la cual define un proceso *monochronous*. Ahora podemos clasificar a estos operadores en:

estáticos: están definidos por extensiones directas de funciones instantáneas en funciones actuando sobre flujos. Es decir, son extensiones de secuencias de operadores lógicos y aritméticos (+, *, -, /, **, =, and, or, not, etc.).

Formalmente, definamos *f* un símbolo que denota una función n-aria dada [|*f*|] sobre valores instantáneos, luego la expresión SIGNAL

$$a_{n+1} := f(a_1, \ldots\ldots\ldots\ldots, a_n)$$

define un proceso elemental

$$\forall t \quad [|f|]\ (a_1(t), \ldots\ldots\ldots\ldots, a_n(t)) = a_{n+1}(t)$$

donde $a_i(t)$ denota el t-ésimo elemento de la secuencia descripta por a_i.

Consideremos ahora un ejemplo:

$$y := u + v$$

se corresponde con $\forall n \quad y_n := u_n + v_n$ con manejo implícito del índice de tiempo *n*.Todas las señales referidas deben estar presentes simultáneamente. La definición formal antes expuesta expresa una relación entre valores y no realmente una función.

***dinámicos*:** surgen cuando en una referencia de tiempo dada, valores recibidos antes del instante actual deben ser considerados. El operador de retardo de SIGNAL define una señal de salida cuyo t-ésimo elemento define el t-ésimo - 1 de la entrada, en cualquier instante excepto en el primero donde toma un valor de inicialización. Por ejemplo:

$$x := y\ \$1$$

es la codificación de $\forall n, \quad n>0, \quad x_n = y_{n-1}$, donde y_0 es el valor de inicialización. A partir de allí *x* posee los valores previos de *y*. Las señales involucradas deben estar presentes simultáneamente.

El comportamiento de este retardo con una inicialización en 0 sería:

y: 2 5 1 0 4 3 7 1 ....

x: 0 2 5 1 0 4 3 7 ....

3.2.1.2 Operadores polychronous

Distintas referencias de tiempo tienen lugar cuando los operadores SIGNAL correspondientes a compuertas Data-Flow *(gate)* o distribuidor *(merge)*

[ver capítulo 1] son usados. Dichos operadores definen procesos de múltiples relojes.

Operador "when": este operador se corresponde con la compuerta True Data-Flow, la cual tiene como entrada un dato y un "control" boolean. Cuando una de las entradas toma el valor ⊥, entonces la salida también es ⊥. En cualquier instante lógico en que las señales de entrada se encuentren bien definidas (distintas de ⊥), la salida será distinta de ⊥. Veamos el siguiente proceso SIGNAL como ejemplo:

$$x := y \textbf{ when } b$$

aquí x e y son señales, y b es una señal boolean que actúa como control. En un instante dado si b está presente y verdadera, entonces y toma el valor de x; en cualquier otro caso y toma el valor ⊥.

El operador ***when*** permite especificar una operación de muestreo a partir de una condición boolean, definiendo a la señal de salida como una extracción de la señal de entrada.

Con el siguiente diagrama presentamos el comportamiento de este operador:

y:	⊥	3	2	1	⊥	9	6
b:	T	F	T	T	T	⊥	F
x:	⊥	⊥	2	1	⊥	⊥	⊥

Operador "default": para la distribución Data-Flow, la cual necesita una entrada de control para seleccionar un dato de entrada, corresponde una distribución determinística con 2 (dos) datos de entrada, llamada ***default***. Este operador es considerado Data-Flow aunque no es el caso para el operador de "distribución" convencional del esquema Data-Flow, el cual es Data-Flow y Control-Flow: El valor de entrada de control de la distribución determina cuál valor de entrada (dato) es pasado por el arco de salida.

El proceso SIGNAL :

$$x := y \textbf{ default } z$$

determina que la señal x es una mezcla de las señales y y z con prioridad de y sobre z cuando ambas están presentes simultáneamente. Es decir, x toma los valores de y siempre y cuando esta última esté presente; toma los valores de z cada vez que esta última esté disponible e y esté ausente; y , finalmente, x es $\perp$ cuando ninguna de las 2 (dos) señales de entrada se encuentren disponibles.

El operador ***default*** puede ser asociativo, lo cual evita el uso de paréntesis. El operador ***when*** antes descripto es distributivo sobre el ***default,*** o sea:

$$x := y \textbf{ default } z \textbf{ when } b$$

es equivalente a

$$x := (y \textbf{ when } b) \textbf{ default } (z \textbf{ when } b)$$

Gráficamente el comportamiento del operador ***default*** podría describirse a través del siguiente diagrama:

y: 5 3 1 $\perp$ 2 1 1 $\perp$ 5

z: $\perp$ 1 3 $\perp$ $\perp$ 1 5 6 7

x: 5 3 1 $\perp$ 2 1 1 6 5

3.2.1.3 Operador de composición

Las ecuaciones anteriores definen procesos elementales; la composición:

$$P_1 | P_2$$

define un nuevo proceso donde los nombres comunes se refieren a señales iguales a través de las cuales P_1 y P_2 se comunican. Esta es justo la noción de unión ($\cup$) de ecuaciones utilizada en matemáticas. Es por ello que este operador es asociativo y conmutativo. El conjunto de procesos SIGNAL es un monoide conmutativo:

dados los procesos P_1 , P_2 y P_3 se cumple

$$(P_1 | P_2) | P_3 = P_1 | (P_2 | P_3)$$

$$(P_1 | P_2) = (P_2 | P_1)$$

$$P_1 | 1 = P_1$$

donde 1 es el proceso que carece de entrada y de salida (por lo tanto nunca se comunica).

Por lo antes mencionado se define a un programa en SIGNAL como una composición recursiva de procesos elementales los cuales son una expresión definiendo una señal. Dicha composición de procesos da lugar a una especificación jerárquica que representa la dependencia estática de datos (señales). La dependencia operatoria se experimenta en un estilo Data-Flow lo que permite describir el potencial paralelismo de los algoritmos de la aplicación.

3.2.1.4 Operador de restricción

Además de los cinco constructores básicos existen otros utilizados en el diseño de algoritmos: dados P un proceso y x_i señales

a) P !! x_1,....., x_p

b) P / x_1,....., x_n

a)- Restricción del proceso P a la lista de señales (x_1,....., x_p): otras señales involucradas en P son locales y no son visibles cuando es considerada una comunicación.

b)- Este operador pemite considerar como locales un subconjunto de señales definidas en un proceso dado (x_1,....., x_n). Así se define un nuevo proceso donde las formas de comunicación son aquellas de P excepto x_1,....., x_n .

3.2.1.5 Manejo del tiempo

En un proceso muchas veces es importante conocer si una señal está presente o no independientemente del valor que ésta tome. El operador unario ***event*** utilizado en los procesos SIGNAL:

x:= ***event*** b

define una señal tipo *event*[4] o señal pura *x* cuyas ocurrencias son simultáneas con aquellas de b; representa el reloj de b.

En términos de relojes:

[4] Señal boolean cuyas ocurrencias son siempre *true*.

$$P(x) = P(b)$$

Otro operador unario

x:= ***when*** b

define una señal x de tipo *event* la cual está presente siempre que la señal boolean b esté presente y tenga el valor *true*; y está ausente ($\perp$) en caso contrario. Es equivalente a :

x:= b ***when*** b

los relojes asociados serían $P(x)=T(b)$

Es posible sincronizar explícitamente dos o más señales, esto es, asignarles el mismo reloj. El proceso

synchro { $x_1,...,x_n$ }

implica

$$P(x_1)= P(x_2)= ... = P(x_n)$$

Otras notaciones utilizadas son :

$x\hat{}= y$ $\Rightarrow$ x e y tienen el mismo reloj, esto es $P(x) = P(y)$

$x\hat{} < y$ $\Rightarrow$ x no es más frecuente que y , lo cual es equivalente a:

$x\hat{} = x$ ***when (event*** y)

3.2.1.6 Ventanas deslizantes

Asi como el operador de retardo ***$*** k (sección 3.2.1.1) permite preservar el valor retardado en k instantes de una señal, es también posible manejar la historia de los últimos k valores de una señal lo que da lugar a la posibilidad de manipulación de arreglos en SIGNAL; herramienta de gran utilidad en el manejo de señales.

El proceso

ws:= s ***window*** k

define una ventana deslizante de longitud k sobre la señal s, donde

$ws[k]= s$; $ws[k-1]= s \$ 1$; ....; $ws[1]= s \$ k-1$

Las señales ws y s son sincrónicas, lo cual significa:

$$P(ws) = P(s)$$

Es posible combinar este proceso con el operador de retardo, y así

ws:= *s* ***$*** *n* ***window*** *k*

donde *k* es la longitud de la ventana *ws* y se cumple

$ws[i]= s\$ (n + k - y); \qquad 1 \leq i \leq k$

Este constructor y sus diversas combinaciones son de gran utilidad para el estudio de "*frames*" de señales, lo cual será discutido con posterioridad.

3.2.1.7 Arreglos de procesos

Esta estructura es útil cuando se especifican algoritmos de características *sistólicas*[5] o cuando describimos arquitecturas regulares. Veamos ahora como ejemplificación la extensión a vectores de un operador dado:

array I ***to*** N ***of*** V:= V1[I] * V2[I] ***end***

es la extensión del producto vectorial como es representado en la siguiente figura:

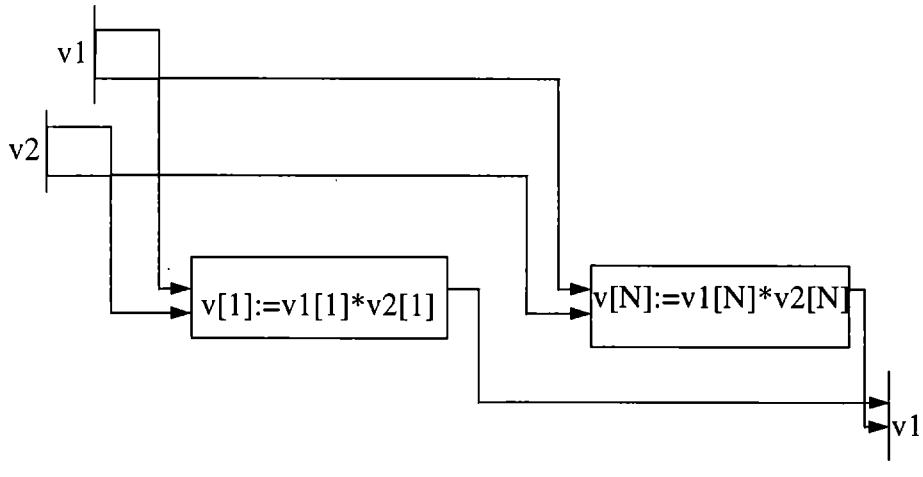


Fig 3.1 Diagrama de la semántica del producto vectorial.

Semántica[6][9]

El proceso PN definido por la expresión:

array I ***to*** N ***of*** P ***with*** *indexación* ***end***,

está constituído por la sucesión de procesos P_i construídos sobre el modelo de P.

[5] Los arreglos sistólicos son arquitecturas paralelas compuestas por elementos de procesamiento interconectados en forma regular o local. Los problemas clásicos para los cuales se utiliza un acercamiento sistólico son problemas lineales, producto de matrices, solución a un sistema de ecuaciones lineales, etc.

[6] El motivo de este detalle explicativo del comportamiento del proceso, es debido a que presenta una complejidad en su semántica superior a los restantes operadores.

- Para toda salida *x* de P de un tipo de datos en particular, PN posee una salida *X* de tipo vectorial (*X[1..N]*) del mismo tipo de la señal de *x*, tal que *X[i]* es la salida *x* del proceso P_i .
- Si *a* es una entrada de P que no está presente en las indexaciones, entonces es difundida a las N células, por el contrario si está presente en las indexaciones con la forma *a,* cada entrada de la célula P_i que posee su nombre es indexada por I.
- Para toda entrada *x* de P indexada y no conectada PN posee una entrada *X [1..N]* , tal que *X[i]* es la entrada *x* del proceso P_i .
- Cada entrada indexada *a* es conectada a la salida del mismo nombre con el mismo índice.

Una entrada *a* de índice 0 (respectivamente N+1) dada por *a[0]:b* se vuelve la entrada *b* del proceso PN.

3.2.1.8 Extensiones

El siguiente operador es una derivación de los operadores previamente descriptos, y es utilizado para especificar una memoria sincronizada : el proceso SIGNAL

y := *x* ***cell*** *b*

determina que los valores de *y* serán los de *x* cuando esta última esté presente o el último valor recibido de *x* cuando la señal boolean *b* esté presente y *true*.

El mencionado proceso es equivalente a :

y:= *x* ***default*** (*y* ***$1***)

| *y*^=(***event*** *x*) ***default*** (***when*** *b*)

3.2.2 Generando un programa SIGNAL

Un programa SIGNAL es un conjunto (composición) de procesos elementales o subprocesos que expresan relaciones funcionales y temporales

entre todas las señales involucradas en el mismo. Es considerado un *autómata*[7] cuyas variables son señales. Generalmente expresan restricciones sobre el comportamiento de sus señales involucradas, lo cual es posible a través del uso de la composición (sección 3.2.1.3) de procesos. Dicha composición da lugar a una taxonomía de diagramas en bloques, donde los procesos son representados por cajas, y las interconexiones entre las puertas de I/O de cada proceso; por líneas.

Un *modelo* encapsula un conjunto de ecuaciones, por lo tanto el mismo es una declaración de un proceso que permite al programador aislar definiciones locales, proveer descripciones parametrizadas, definir *submodelos* y variables externas o libres. Los parámetros de un modelo son constantes de los posibles valores de una señal.

Un *modelo* o declaración de un proceso puede ser definido en el exterior de un programa; él no es entonces visible sino a través de su interfase.

La invocación a un *modelo* es equivalente a reemplazar dicha invocación por el sistema de ecuaciones que el mismo representa (macrosustitución).

Como hemos dicho ya ,un programa SIGNAL puede ser usado para especificar STR y como tal debe respetar el principio de *causalidad*, es decir, el valor de una señal en un instante dado no debe depender del valor de la misma señal en un instante futuro.

3.2.2.1 Estructura de un modelo (programa) SIGNAL

La estructura general de un *modelo* es básicamente:

```
process identificador = ( lista de parámetros ) (? señales de entrada
                                                 ! señales de salida )
   (| declaraciones
    . . .
    |)
```

[7] Estructura que posee un conjunto de estados, un alfabeto de comunicación, una función de proximo estado y un estado inicial.

where

declaración de variables, funciones externas y subprocesos

end

3.2.2.2 El compilador

Para ser capaz de verificar si un programa está libre de *deadlock* o tiene una ejecución efectiva, y bajo estas suposiciones, qué *scheduling* puede ser calculado estáticamente para una implementación multiprocesador, dos herramientas son usadas antes de que dicho programa se ejecute sobre una arquitectura dada, a saber:

- modelización de las relaciones de sincronización en $\mathcal{F}_3$ por medio de polinomios con coeficientes en $\mathbf{Z/Z_3}$.
- grafo dirigido de dependencias de datos.

Sistema de cálculo formal ($\mathcal{F}_3$)

SIGNAL maneja secuencia de datos con tiempo implícito las cuales pueden estar ausentes ($\perp$) o presentes. Las instrucciones de dicho lenguaje son pensadas para relacionar relojes asi como valores de las distinta señales involucradas en un sistema dado.

Consideremos procesos SIGNAL restringidos al dominio simple de valores booleanos. La condición:

$c_3 := c_1$ ***when*** c_2

dice que

- si c_1 está definida y c_2 está definida y es *true*, entonces c_3 está definida y $c_3 = c_1$
- si c_1 no está definida, o c_2 está definida, o c_2 está definida y es *false*, entonces c_3 no está definida

Por lo tanto es útil saber si, siendo c una señal:

1. c está definida y *true*

2. c está definida y *false*

3. c no está definida.

Esto puede ser representado en el campo finito de $\mathbf{Z}/\mathbf{Z}_3$ (enteros módulo 3) de la siguiente forma:

-1 (presente y *false*), **1** (presente y *true*), **y 0** (ausente)

Entonces si v es el valor codificado de la señal c, la presencia de dicha señal puede ser representada por $\mathbf{v}^2$. Esta representación de un valor indeterminado de c (*true* o *false*) permite una inmediata generalización para valores no booleanos; donde la presencia de los mismos es codificada como **1** y su ausencia como **0.**

Así, $\mathbf{c}^2$ puede ser considerado como el reloj de la señal c; por lo tanto se puede asociar a cualquier señal c una variable **c** en $\mathcal{C}_3$.

Este principio es utilizado para representar relaciones de sincronización expresadas a través de programas SIGNAL.

La codificación de los operadores elementales bajo este sistema de representación es deducida a partir de su definición:

1. para $c_{n+1} := f(c_1, \ldots\ldots\ldots, c_n)$ tenemos entonces

$$c^2_{n+1} = c^2_1 = \ldots\ldots = c^2_n,$$

las relaciones booleanas pueden ser codificadas en $\mathcal{C}_3$ como sigue :

2. para $c_3 := \textbf{\textit{not}}\ c_1$ *corresponde*

$$c_3 = -c_1 .$$

Veamos que si c_1 es *true* entonces $c_1 = \mathbf{1}$ y $-c_1 = \mathbf{-1}$ lo cual implica asociar a c_3 el valor booleano *false.*

3. para $c_3 := c_1$ ***when*** c_2 con c_1, c_2 , y c_3 señales booleanas se asocia la ecuación

$$c_3 = c_1 (- c_2 - c^2_2),$$

lo cual puede ser interpretado como sigue: c_3 mantiene el mismo valor que c_1 ($c_3 = c_1$) cuando c_2 es *true* ($- c_2 - c^2_2 = \mathbf{1}$) Ahora cuando las señales en cuestión excepto c_2 , no son booleanas la ecuación asociada es $c^2_3 = c^2_1 (- c_2 - c^2_2)$.

4. para $c_3 := c_1$ ***default*** c_2 con todas las señales booleanas, la ecuación ligada es la siguiente:

$$c_3 = c_1 + (1 - c^2_1)\, c_2 .$$

Así, c_3 toma valores cuando:

- c_1 está definida, es decir $c^2_1 = 1$, tomando c_3 el valor de c_1.

  Así $c_3 = c^2_1\, c_1 = c_1$ pues $(1 - c^2_1)\, c_2 = 0$ donde c^2_1 representa la presencia de c_1 .

- c_2 está definida, pero c_1 no está definida, en términos de ecuaciones sería: $(1 - c^2_1)\, c_2 = 1$ y $c_1 = 0$, tomando c_3 el valor de c_2.

  Expresamos a $c_3 = (1 - c^2_1)\, c^2_2\, c_2$ donde: $(1 - c^2_1)$ es la ausencia de c_1 y c_2 es la presencia de c_2.

5. para P | Q, donde P y Q son procesos SIGNAL, las ecuaciones de reloj asociadas al proceso de composición son exactamente la unión de las ecuaciones de P y Q respectivamente.

6. para $c_2 := c_1$ $1 , en este caso los registros de desplazamiento boolean son codificados como sigue:

$$\xi_{n+1} = (1 - c^2_1)\, \xi_n + c_1 \quad , \xi_0 = c_{2,0}$$

$$c_2 = c^2_1\, \xi_n$$

En esta ecuación , ξ_n es el estado corriente, y ξ_{n+1} es el próximo valor de acuerdo a cualquier reloj, el cual es más frecuente que el reloj de c_1 ($\xi_0 = c_{2,0}$ es el valor inicial). Esto es un sistema dinámico no lineal sobre $\mathcal{Z}_3$. El registro de desplazamiento no boolean es codificado via la igualdad de relojes:

$$c^2_2 = c^2_1$$

Finalmente concluimos con la forma general para codificar cualquier programa SIGNAL:

$$\xi_{n+1} = \mathbf{A}\,(\xi_n, \mathbf{Y}_n)$$

$$0 = \mathbf{B}\,(\xi_n, \mathbf{Y}_n)$$

$$0 = \mathbf{C}\,(\xi_0, \mathbf{Y}_0)$$

En este sistema, ξ e **Y** son vectores con componentes en el campo finito $\mathcal{F}_3$. **A**, **B** y **C** son polinomios cuyas componentes son $\xi(i)$ e **Y**(j). Los componentes de ξ son estados de los registros boolean, y los de **Y** son la codificación en $\mathcal{F}_3$ de todas las señales **Y**(j) involucradas en el programa. El índice de tiempo *n* es el de algún reloj más frecuente que los relojes de las componentes de **Y**.

La codificación algebraica antes descripta de las relaciones de sincronización es utilizada para:

Detectar errores de sincronización: para ilustrarlo consideremos el siguiente ejemplo:

$$x := c > 0$$
$$| \; y := c \; \textbf{\textit{when}} \; x$$
$$| \; z := y + c$$

esto significa adicionar a *c*, *c* cuando ésta es positva. Su codificación en $\mathcal{F}_3$ sería:

$$\mathbf{x}^2 = \mathbf{c}^2$$
$$\mathbf{y}^2 = \mathbf{c}^2 \, (\, -\mathbf{x} - \mathbf{x}^2 \,)$$
$$\mathbf{z}^2 = \mathbf{c}^2 = \mathbf{y}^2$$

lo cual aplicando pasos algebraicos simples resulta en:

$$\mathbf{x}^2 = \mathbf{c}^2 = \mathbf{z}^2 = \mathbf{y}^2 = \mathbf{c}^2(\, -\mathbf{x} - \mathbf{x}^2 \,)$$

entonces $\mathbf{x}^2 = \mathbf{x}^2 \, (-\mathbf{x} - \mathbf{x}^2 \,)$, luego **x** es 1 ó 0. Pero *x* es el resultado de la evaluación de una señal no boolean *c*. La codificación en $\mathcal{F}_3$ no permite la representación de valores no booleanos, y debido a que **x** tiene dos valores posibles para una misma solución el programa será rechazado por el compilador.

Organizar el control del programa: una relación de orden puede ser definida sobre los relojes, donde si para dos relojes dados $\mathbf{h}^2$ y $\mathbf{k}^2$, $\mathbf{h}^2 \geq \mathbf{k}^2$,significa que $\mathbf{h}^2$ es un sobremuestreo de $\mathbf{k}^2$,es decir que el conjunto de instantes de la señal *k* está incluido en el conjunto de instantes de la señal *h*.

El conjunto de relojes con esta relación forman un *reticulado*, por lo tanto, habrá un reloj que es cota superior denominado *master*, y se podrá definir cada reloj de dicho reticulado por medio de una expresión de cálculo en términos del reloj *master*, esto es definir a cada reloj como un submuestreo del conjunto de instantes del reloj *master*.

Para que un programa sea correcto el orden parcial inducido por la relación $\leq$ de instantes restringidos a los submuestreos de una determinada condición boolean, debe ser un árbol. Esto es necesario para obtener un programa determinístico. La raíz de dicho árbol es el reloj más frecuente *(master),* cualquier ecuación de reloj puede ser reducida recursivamente a una suma de monomios donde cada uno de ellos es un producto de submuestreos, caso contrario el reloj es la raíz.

Grafo de dependencias

Otra herramienta usada para verificar un programa SIGNAL es el Grafo de Dependencias de Datos a partir del cual será posible definir subgrafos que podrán ser distribuidos sobre diferentes procesadores.

Un grafo Data-Flow no representa la dependencia de un programa SIGNAL pues los relojes pueden ser diferentes y por lo tanto las dependencias no ser constantes. Así nuestro grafo representa dependencias condicionales, donde las condiciones son los relojes sobre los cuales las dependencias se hacen efectivas. La siguiente relación tiene que ser considerada:

" *para cualquier señal 'a', el valor de 'a'* (**a**) *no puede ser conocido antes que su reloj* ($\mathbf{a}^2$), *o sea que 'a' depende de* $\mathbf{a}^2$.

Este grafo, calculado por el compilador SIGNAL para un programa dado, es dirigido con etiquetas donde:

- los vértices son las señales más la variables de reloj;
- los arcos representan las relaciones de dependencia;

- las etiquetas son polinomios con coeficientes en $\mathcal{Z}/3$ que representan los relojes en los cuales las relaciones son válidas.

Para procesos elementales tenemos lo siguiente : la notación $\mathbf{x}^2 : y \rightarrow z$ significa que z depende de y exactamente cuando $\mathbf{x}^2 = 1$. Debe ser considerado que los procesos que involucran solo señales boolean no generan dependencia de datos. Entonces se tendrán en cuenta solo aquellos procesos en los que se definen señales no boolean.

$a_{n+1} := f(a_1, \ldots\ldots, a_n)$	$a^2_{n+1} : a_1 \rightarrow a_{n+1}, \ldots, a^2_{n+1} : a_n \rightarrow a_{n+1}$
$a_1 := a_3$ ***when*** a_2	$a^2_1 : a_3 \rightarrow a_1$ $a^2_1 : a_2 \rightarrow a^2_1$
$a_4 := a_5$ ***default*** a_6	$a^2_5 : a_5 \rightarrow a_4$, $a^2_6 - a^2_5 a^2_4 : a_6 \rightarrow a_4$

Este grafo es utilizado para detectar dependencias incorrectas, la cual aparece cuando existe un ciclo en un grafo. Aún así, desde que las dependencias son etiquetadas por relojes algunos circuitos pueden no ocurrir jamás, teniendo en cuenta que un circuito es efectivo si el producto de los labels no es nulo.

Formalmente, un grafo de dependecias generado por el compilador SIGNAL es un $\mathcal{G}$ (p) asociado a un proceso (programa SIGNAL) p, el cual es también considerado una cuádrupla $G = (N, \Gamma, I, O)$ donde:

- N es el conjunto finito de nodos, cada uno de los cuales representa la expresión definiendo la señal.
- $\Gamma \subset N \times N$ es el conjunto de dependencias entre las expresiones, el cual se encuentra incluído en el conjunto total de pares de nodos $N \times N$ por ser una dependencia una relación entre dos nodos de un grafo.
- $I \subset N$ es el conjunto de nodos de entrada visible.
- $O \subset N$ es el conjunto de nodos de salida visible.

Ya establecimos que dos señales tienen el mismo reloj si y sólo si ellas están asociadas a polinomios equivalentes en $\mathcal{Z}/3$.

Una implementación distribuída de un proceso SIGNAL p está definido por un conjunto de de subprocesos de p comunicados en la siguiente expresión :

$$p_1 \mid \ldots\ldots \mid p_n$$

Cada p_i es ejecutado sobre un procesador distinto y representado por un subgrafo SIGNAL $\mathcal{G}(p_i)$ del grafo general SIGNAL $\mathcal{G}(p)$.

Definamos ahora el subgrafo $\mathcal{G}(p_i)$ de $\mathcal{G}(p)$ como un cuádrupla $G_i = (N_i, \Gamma_i, I_i, O_i)$ tal que para cualquier arco cortado (x,y) en $\mathcal{G}(p)$ donde x está en G_i e y está en G_j un nodo (x,y) es adicionado en G_i y en G_j. Más aún Γ_i tiene la dependencia $(x, (x,y))$ y Γ_j a la dependencia $((x,y), y)$. Cualquier nodo (arco) de $\mathcal{G}(p)$ es adicionado como se describió antes. Los conjuntos N y Γ son particionados en subconjuntos N_i y Γ_i.

Las particiones de un grafo SIGNAL en subgrafos debe ser tal que la composición de los subgrafos es el grafo inicial.

La composición $G_i \circ G_j$ de subgrafos G_i, G_j está definida por al unión de nodos (arcos) G_i, G_j con la siguiente regla especial:

si $(x,(x,y))$ es un arco de G_i y $((x,y),y)$ es un arco de G_j luego el arco (x,y) es adicionado en el grafo compuesto y el nodo (x,y) es borrado.

Es fácil ver que $\mathcal{G}(p_1) \circ \ldots \circ \mathcal{G}(p_n) = \mathcal{G}(p_1 \circ \ldots \circ p_n)$

3.2.2.3 El trabajo del compilador

Dado que esta codificación presentada es híbrida por naturaleza, pues dos álgebras distintas son utilizadas: por un lado el álgebra de expresiones polinomiales en variables $\mathcal{F}_3$ y por el otro los grafos dirigidos con etiquetas; es que podemos considerar a SIGNAL como un formalismo adecuado para sistemas híbridos permitiendo especificar sus características claves: operaciones, eventos y lógica, restricciones de tiempo, y sus interacciones mutuas.

Utilizando las herramientas de codificación mencionadas en la sección previa, y con la ayuda de la teoría polinomial ideal, es posible responder a preguntas acerca de las propiedades de un programa dado. Veamos algunas de ellas:

1. *¿ el programa exhibe contradicciones?* ejemplo:

(| x:= y ***when*** $(y>0)$

| k:= y ***when not*** $(y>0)$

| c:= $x + k$

|)

escribiendo β en lugar de $y>0$ su cálculo de reloj da :

$x^2 = y\ (-\beta\ -\beta^2)$

$k^2 = y\ (\beta\ -\beta^2)$

$c^2 = x^2 = k^2$

$\Rightarrow\ -\beta - \beta^2 = \beta - \beta^2$

entonces $\beta = 0$, esto es, y deberá estar siempre ausente lo cual hace que el programa rechace su entrada y no haga nada.

2) *¿ hay circuitos cortos ?* ejemplo

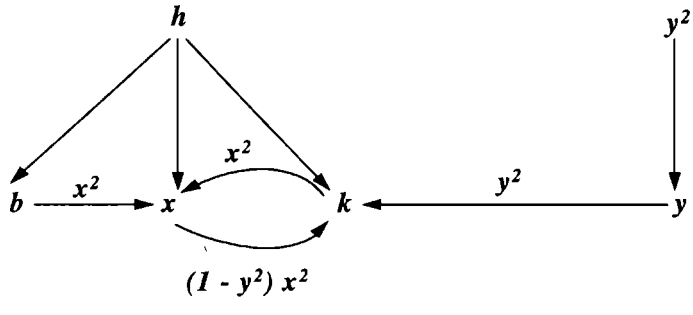
(| x:= $sin\{k\} + b$

| k:= y ***default*** x

|)

el cálculo de reloj y el grafo de dependencias condicionales es:

$x^2 = k^2 = b^2 = y^2 + (1 - y^2)\ x^2$



Debido al circuito corto incluyendo x y k, este programa se encuentra bloqueado a menos que el reloj de dicho circuito esté siempre ausente, es decir,$(1 - y^2)\ x^2 = 0$. En consecuencia $k^2 = y^2$ y así este programa implementa:

(| k:= y

| k:= $sin\ \{y\} + b$

|)

3)¿ está el programa estableciendo restricciones sobre sus entrada?

(| *k*:= *y when y>0*

| *x*:= *y* + *k*

|)

escribiendo α en lugar de *y>0,* el cálculo de reloj es:

$\mathbf{x^2 = y^2 = k^2}$, $\mathbf{x^2 = y^2 (-\alpha - \alpha^2)}$, $\alpha^2 = y^2$

lo cual fuerza a $\alpha^2 = 0$ ó $1+\alpha^2 +\alpha^2 = 0$, esto es (en $\mathbb{F}_3$), $\alpha=1$

así cuando *y* esté presente, deberá ser mayor que cero (0), caso contrario el programa entrará en deadlock. No obstante SIGNAL no puede razonar sobre los tipos de datos no boolean.

Por lo tanto considerando que α es la salida de una función no boolean (testeando *y >0*), la restricción $\alpha^2(1-\alpha)=0$ es reemplazada por una más fuerte $\alpha^2=0$ lo cual no involucra el valor *true* o *false* de α : *y* es entonces rechazado y el programa no trabaja.

4) ¿es el programa determinístico, esto es, es una función?

consideremos un programa que especifica un contador con reset externo:

process p = {? reset ! t}

(|nt:=(0 when reset) default t+1

t:= nt $1

|)

Su cálculo de reloj da :

$\mathbf{nt^2 = t^2 = reset^2 + (1-reset^2)t^2}$

lo cual es equivalente a $\mathbf{t^2 \geq reset^2}$: si *reset* es la entrada especificada, el reloj de la salida *t* no es una función de cualquier señal externa. Así este programa no es una función. Insertando la siguiente ecuación de sincronización *reset=s default u* (con *u* otra entrada), especifica completamente el tiempo y obtenemos una función.

El compilador usa un algoritmo eficiente para construir una jerarquía de relojes con respecto a las siguientes reglas:

- si *c* es una señal boolean libre, (o sea resulta de la evaluación de una función con argumentos no boolean); es una señal de entrada del programa o es el estado de una memoria boolean, entonces el reloj definido para el valor *true* de *c (when c)* y el valor definido para el valor *false (when not c)* están puestos bajo el reloj de *c* en la jerarquía; ambos son llamados menos frecuentes (downsamplings).
- si un reloj **k** se ubica bajo un reloj **h** cualquier reloj ubicado bajo **k** lo estará bajo **h**.
- sea **H** un reloj definido como una función de submuestreo (downsampling) $\mathbf{h}_1...\mathbf{h}_n$ si todos estos $\mathbf{h}_i$ están bajo un reloj **k**, entonces **H** también está bajo **k**.

La jerarquía resultante es una colección de árboles interconectados. El orden parcial para estas colecciones representa dependencias entre relojes : el valor actual de un reloj **h** puede ser necesitado para calcular el valor actual de un reloj **k** dado, sólo si **h** se encuentra antes que **k** de acuerdo a este orden parcial. Ninguna jerarquía es definida sobre las raices de los árboles, pero pueden existir restricciones. Cuando estas colecciones reducen a un árbol simple existe un reloj *master* (ya mencionado cuando nos referimos al Sistema de cálculo formal), y de éste derivan los relojes restantes. En este caso el programa puede ser ejecutado en modo maestro, esto es, requiriendo los datos del ambiente. Si surgen distintos árboles el ambiente deberá proveer sincronización adicional.

El grafo de dependencias condicionales es adicionado a esta estructura de la siguiente manera: las señales disponibles (las expresiones definiendo las mismas) para un reloj dado son adicionadas a este reloj. El *grafo jerárquico condicional*, así obtenido es la base para la generación de código, tanto secuencial como paralelo. La sintaxis propia de SIGNAL puede ser usada para representar este grafo. Para este propósito, el compilador reescribe las expresiones de reloj como expresiones SIGNAL boolean[8] : el operador "default" representa la cota superior de relojes (suma) y el operador "when" la cota inferior

[8]El código que transcribe éste grafo para un programa en particular puede ser obtenido compilando la especificación con la opción **tra.**

(producto); entonces cualquier expresión de reloj puede ser recursivamente reducida a una suma de monomiales, donde cada monomial es un producto de submuestreos (c.c. el reloj es una raíz). Las definiciones de las señales también son reescritas para hacer explícitos los relojes de los cálculos que las definen. El programa reescrito es equivalente al inicial con la diferencia de que el cálculo de relojes y las dependencias han sido resueltas, y los relojes manipulados en el programa han sido explicitados precisamente. El proceso así obtenido será llamado la forma *resuelta* del programa considerado.

3.2.2.4 Notación alternativa para el cálculo de relojes

Así como el compilador representa las ecuaciones de un programa mediante polinomios con coeficientes en $\mathcal{Z}/3\mathcal{Z}$ para el cálculo de relojes, es posible utilizar una notación basada en la teoría de conjuntos para verificaciones prácticas.

Esta notación ya fue introducida en la sección 3.2 cuando se definió para una señal x dada a $P(x)$ como el conjunto de instantes (relativos a un ambiente) en el cual la señal está presente; esto es, el reloj de x. De la misma manera se denotará con $T(b)$ al conjunto de instantes en que la señal boolean b se encuentra presente y es *true*. Teniendo en cuenta estas consideraciones la notación asociada a los operadores elementales ya vistos es [19]:

Proceso elemental	**Ecuaciones de reloj**	**Dependencia instantaneas entre señales**
y:= f(x)	P(y) = P(x)	y depende de x.
zx := x $1	P(zx) = P(x)	zx no depende de x.
y := x when b	$P(y) = P(x) \cap T(b)$	y depende de x cuando b está presente y *true*.
y:= x default z	$P(y) = P(x) \cup P(z)$	y depende de x cuando está presente, o de z cuando x está ausente y z presente.

Cualquier proceso SIGNAL es asociado a un sistema de esta ecuaciones de reloj, el cual si puede ser resuelto, asegura que el proceso no tendrá inconsistencias temporales entre señales. Informalmente un sistema de ecuaciones de relojes se considera resuelto si cada señal intermedia y de salida es expresada como una función de los relojes de señales de entrada.

Las siguientes reglas son usadas con el objeto de resolver las ecuaciones de reloj:

1) dependendiendo del tipo de la señal:
   - si b es una señal boolean entonces T(b) $\subseteq$P(b), esto es el conjunto de instante en los que la señal b está presente y es *true* está incluído en el conjunto de instantes en que la señal b está presente.
   - si b es una señal pura o de tipo *event,* entonces T(b) = P(b).
2) teniendo en cuenta el álgebra de conjuntos, dados *a* y *b* conjuntos:
   - $a \subseteq b$, entonces $a \cap b = a$

     $a \subseteq b$, entonces $a \cup b = b$
   - $a \cap b = a$, entonces $a \subseteq b$

     $a \cup b = b$, entonces $a \subseteq b$
   - $a \supseteq (a \cap b) \subseteq b$

     $a \subseteq (a \cup b) \supseteq b$

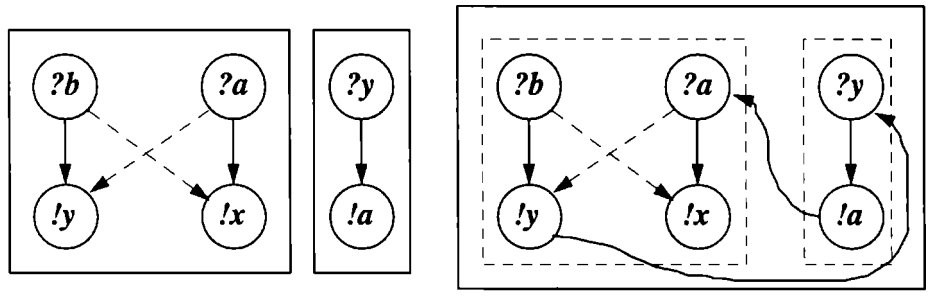
3.2.3 Aproximación a la implementación paralela

Una implementación distribuida de un programa SIGNAL , P, consiste en:

$$P= (|P_1| ... | P_n |)$$

un conjunto de módulos P_1 ,..,P_n , a cada uno de los cuales se les asigna un procesador. Con el sistema de ecuaciones que el compilador SIGNAL genera los módulos P_i pueden ser construidos *top-down* o *bottom-up*. Así tenemos construido un método para serializar tales módulos, evitando posibles deadlocks y proveyendo una herramienta para probar la eficiencia de la implementación, reduciendo la sobrecarga debido al proceso de scheduling.

3.2.3.1 Esquematización



a) Objeto p b) Objeto r c) deadlock

Fig. 3.2

Las siguientes figuras especifican programas a través de una notación que luego se detallará:

Aquí las flechas continuas denotan dependencias de datos inducidas por programa, las flechas punteadas indican un orden adicional que resulta de una implementación dada. Por ejemplo, en la fig.*(3.2 a)* el programa especifica que ***a*** debe ser recibido antes de generar ***x*** (lo mismo ocurre con ***b*** e ***y***). Las flechas punteadas también expresan que en la implementación planteada primero se espera por ***a*** y ***b***, y luego se generan ***x*** e ***y***. Como conclusión si se adicionan flechas punteadas dentro de un grafo de dependencias se aumentará el orden en dicho grafo.

Veamos el siguiente ejemplo :

P=(| $y := g(b)$
| $x := f(a)$
|)

este programa es correcto desde el punto de vista de la ejecución siempre que se disponga de señales de entrada cada vez que sean necesitada. La descripción de cada paso es mostrado en la fig.*(3.2 a)*.

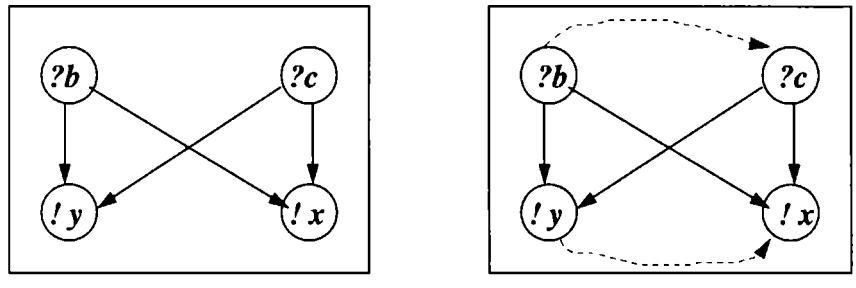
Vaemos ahora el siguente programa:

R=(| $a := h(y)$
|)

donde ***h*** es una función. Su único esquema de ejecución correcto es la entrada de la señal ***y*** seguidas de las ***a*** generadas; así por siempre. Este programa es explicado en la fig. *(3.2 b)*.

Si generamos la siguiente composición P | R , la misma sería correcta. No obstante desde el punto de vista concurrente (ambiente multitarea) de sus implementaciones secuenciales *obj-p* y *obj-r* , dicha composición incurre en *deadlock (3.2 c)*): *obj-p* está esperando por ***a*** , pero para producir ***a*** *obj-r* necesita ***y*** lo cual no puede ser retornado por *obj-p* , esto es mostrado por el ciclo en la fig. *(3.2 c)*.

Ahora consideramos las siguientes figuras:



a) Q ***b) obj-Q***

Fig. 3.3

que representan el siguiente programa :

$$\mathbf{Q}= (\ |\ y := g(a, b)\ |\ x := f(a, b)\ |\ )$$

entonces para cualquier programa R´ tal que ***x*** o ***y*** sean necesitados para calcular ***a*** o ***b*** , el programa **Q** | R´ es incorrecto. De esta manera hace falta un orden parcial como el que es especificado en la fig. *(3.3 a)*. para garantizar que la ejecución será correcta. Por ejemplo la secuencia { obtener ***b***, obtener ***a***, sacar ***y***, sacar ***x*** } repetida por siempre no causa deadlock adicionales cualquiera sea el ambiente. Esta implementación es mostrada por las líneas punteadas en la fig. *(3.3 b)*. Dicha figura muestra un mejoramiento del orden del grafo ilustrado en la fig. *(3.3 a)* , siendo éste una clave para la distribución de código y una herramienta para eliminar posibles deadlock que el programa representado pueda tener con el ambiente en el cual se desarrolle.

3.2.3.2 Interfase del grafo condicional

La interfase se plantea utilizando la siguiente dos reglas:

1. *regla de series* :

$$x \xrightarrow{h} y \xrightarrow{k} z \Longrightarrow x \xrightarrow{hk} z$$

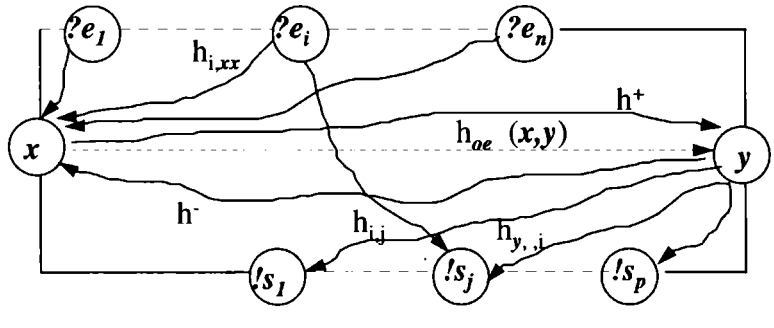
x precede a y, en el instante en que h=1, e y precede a z en el instante en que k=1, por lo tanto x precede a z cada vez que h=k=1.

2. *regla de paralelo:*

$$\left.\begin{array}{l} x \xrightarrow{h} y \\ x \xrightarrow{k} y \end{array}\right\} \Longrightarrow x \xrightarrow{h \vee k} y$$

donde h v k = h + (1-h) k denota el supremo entre los dos relojes h y k (funciones polinomiales con coeficientes en $\mathcal{F}_3$ tomando 0 ó 1 como únicos valores). Esta regla implica que x precede a y cada vez que h=1 ó k=1.

Sucesivas aplicaciones de estas reglas da la clase de grafos mostrada como brazos sólidos en la siguiente figura:



3- Mejora de orden

Fig. 3.4

Mejoras de Orden

Considerando dos señales de interfase x e y como las de la figura precedente, y denominando $h_{oe}(x, y)$ al reloj de alguna mejora de orden legal que pone a x antes que a y , las condiciones que deben ser satisfechas por h_{oe} son las siguientes:

-ningún ciclo interno resultará a partir de la adición del reloj $h_{oe}(x, y)$ *en el grafo.*

-ninguna posibilidad de un ciclo adicional debido a los resultados del ambiente $h_{oe}(x, y)$ *; esto da las desigualdades:*

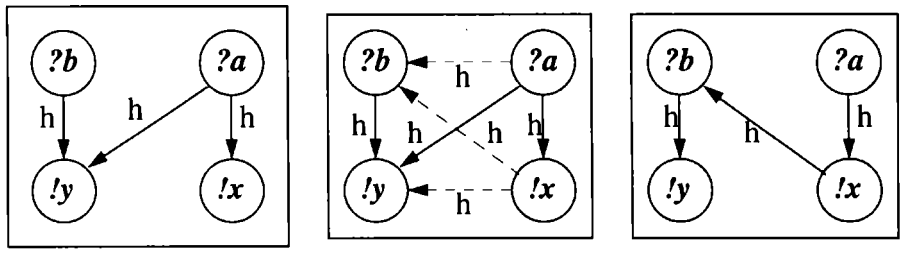
$$\forall i, j \; h_{i,x} \; h_{oe}(x, y) \; h_{y,j} \leq h_{i,j}$$

cualquier entrada e_i que precede a x también precede a cualquier salida s_j que precede a y: esto asegura que en cualquier contexto ninguna dependencia desde una salida s_j a una entrada e_i pueda ser introducida lo cual crearía un deadlock.

Se dice que un grafo de dependencia condicional G_1 es menor que otro G_2 si y sólo sí ellos tienen los mismos nodos y cada vez que $x \rightarrow y$ ocurre en G_1 cuando su etiqueta $h_1=1$, entonces $x \rightarrow y$ ocurre en G_2 ($h_2=1$); así $h_1 \leq h_2$. Aplicando mejora de orden resulta un grafo donde cada $h_{oe}(x, y)$ toma su valor maximal. Por lo tanto este grafo no es el de un orden parcial pero si la cota superior de las mejoras de orden maximales.

3.2.3.3 Obtención de un esquema de ejecución

Sea h el reloj de todos los brazos sólidos de la fig. *(3.2 a)*. El grafo original coincide con la interfase del grafo condicional.

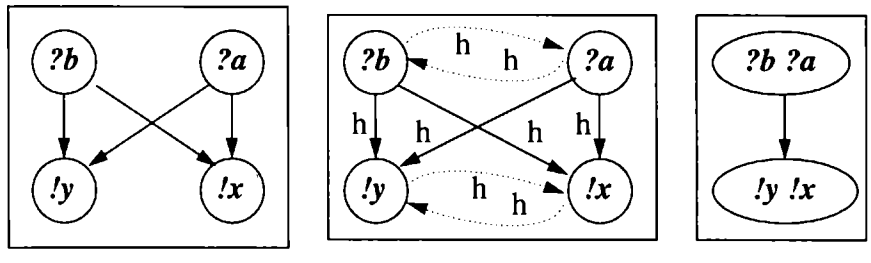


a) S b) aplicando mejoras de orden c) esquema de ejecución

Fig. 3.5

Sea S algún programa cuyo grafo de dependencia condicional es mostrado en la figura *(3.5 a)*; la mejora resultante es mostrado en *(3.5 b)*. S tiene el esquema de ejecución secuencial único mostrado en *(3.5 c)* el cual es obtenido tomando el subgrafo de los brazos sólidos o punteados que representa un camino y cubre todos los nodos.

Para algunos programas la mejora de orden puede resultar en un grafo cíclico como muestra la figura *(3.6 b)*. Tales ciclos no expresan que han sido creados deadlocks pero indican que comunicaciones externas dentro del ciclo pueden ser ejecutadas en algún orden arbitrario dependiendo de las ofertas o requerimientos del ambiente en un instante en particular. Por ejemplo podemos igualmente primero recibir ***a*** y luego ***b*** o viceversa; esto lo demuestra la fig *(3.6 c)*:



a) Q b) aplicando mejoras de orden c) esquema de ejecución

Fig. 3.6

3.2.4 Un programa como ejemplo

El compilador SIGNAL produce un código intermedio en lenguaje C, el cual debe ser compilado para obtener un programa ejecutable que simule la funcionalidad de la aplicación. Cada señal de entrada (respectivamente de salida) es simulada por un archivo conteniendo los valores sucesivos de la señal, siendo el nombre de dicho archivo construido de la siguiente manera: para archivos de entrada, es el nombre de la señal prefijado por la letra ***R***; y para archivos de salida lo único que cambia es el prefijo (***W***) . En ambos casos la extensión es ***.dat***.

Supongamos que deseamos construir un programa donde una señal de entrada de tipo entero es sobremuestrada para producir una salida de tipo lógico donde los valores sucesivos representan los 8 bit de codificación binaria, o sea :

entrada:1 19 255

salida : T F F F F F F F T T F F T F F F T T T T T T T T.

El proceso SIGNAL asociado con este programa es el siguiente:

```
process conversión={? entrada ! salida}  %(1)%
     (|zcontador:= contador $1   %(2)%
      |parar:= (zcontador = 0)   %(3)%
       |contador:=(7  when  parar)default(zcontador -
1)%(4)%
      |synchro{contador, salida} %(5)%
      |synchro{entrada, when parar}  %(6)%
      |zacum:= acum $1  %(7)%
      |acum:= entrada default zacum/2  %(8)%
      |salida:= (acum-2*(acum/2))=1    %(9)%
      |)
     where
         logical    parar;%";"para    separar    tipos
distintos%
         integer contador,zcontador init[9] 0,
         integer acum,zacum init 0;
end %conversión%
```

El cálculo de ecuaciones de reloj nos permite verificar que la señal de salida es en realidad un sobremuestreo de la señal de entrada, como se deseaba:

(1) P(*salida*) ⊃ P(*entrada*)

(2) P(*zcontador*) = P(*contador*)

(3) P(*parar*) = P(*zcontador*)

[9] Palabra reservada para inicializar señales que representan retardos.

(4) P(*contador*) = T(*parar*) ∪ P(*zcontador*) ⇒ P (*contador*) = P(*zcontador*)

pues T(*parar*) ⊂ P(*parar*) = P(*zcontador*)

(5) P(*contador*) = P(*salida*)

(6) P(*entrada*) = T(*parar*)

(7) P(*zacum*) = P(*acum*)

(8) P(*acum*) = P(*entrada*) ∪ P(*zacum*) ⇒ P (*acum*) ⊇ P(*entrada*),

(9) P(*salida*) = P (*acum*)

Lo que nos permite concluir:

P (*entrada*) = T(*parar*) por (6), luego

T(*parar*) ⊂ P (*parar*) aplicando una de las reglas mencionadas en la secciòn 3.2.2.4 y considerando que *parar* es una señal boolean.

Finalmente por (7), (9), (2), (3)

P(*parar*) = P (*zcontador*) = P (*contador*) = P (*salida*) = P (*zacum*) =

= P (*acum*)

y así en (1) se tiene :

P(*entrada*) ⊂ P(*salida*)

3.2.5 Conclusiones

Después de esta presentación informal del lenguaje, se puede concluir que SIGNAL es un lenguaje para especificación de sistemas de TR orientado a Data-Flow, sincrónico y libre de efectos laterales.

Este lenguaje especifica naturalmente restricciones entre las señales involucradas en una aplicación, por lo tanto tiene un estilo ecuacional. Esta última característica hace que el programador sólo tenga que especificar restricciones de sincronización local; la síntesis de la sincronización completa es tarea del compilador, el cual provee verificación de correctitud del tiempo derivando en un grafo de dependencias que es de sumo interés cuando el objetivo final es la implementacón sobre arquitecturas multiprocesador.

Podemos decir que SIGNAL es su propio sistema de prueba pues las propiedades que se expresan a través de un programa son procesadas por el compilador por medio de la generación de un sistema de ecuaciones (cálculo de relojes) para asegurar la ausencia de deadlock, y un grafo de dependencias asociado.

Considerando que se combinan dos álgebras diferentes para la verificación de un programa, SIGNAL es un formalismo que puede encuadrarse dentro de los sistemas híbridos.

Varios servicios, prueba, compilación e implementación distribuida, son soportados por SIGNAL.

El compilador SIGNAL produce un nivel intermedio de generación de código C, FORTRAN u OCCAM (para sistemas multitransputer). El producto de esta precompilación puede ser utilizado por un compilador C para obtener un ejecutable del programa que simule la funcionalidad deseada.

Por otra parte, una interfase gráfica SynDEx, distribuye automáticamente (aplicando una heurística) programas SIGNAL sobre arquitecturas multiprocesador propuestas; siendo su entrada el grafo jerárquico condicional generado por el compilador.

Como corolario, dada la características de los operadores del lenguaje y la forma en que pueden manipularse señales unidimensionales se puede asegurar que es una herramienta adecuada para el procesamiento de señales de voz como se discutirá en los capítulos posteriores.

Capítulo 4

Procesamiento de señales de Voz

4.1- Introducción

Las señales de voz están compuestas de una secuencia de sonidos que sirven como representación simbólica de un pensamiento que un locutor desea transmitir a un oyente. La disposición de estos sonidos es gobernada por las reglas asociadas al *lenguaje*.

La ciencia que estudia las características de la producción de sonidos humanos especialmente para descripción, clasificación y transcripción de señales de voz es llamada *fonética*.[21]

La ciencia que estudia al lenguaje y el modo en que las reglas dentro del mismo son usadas es denominada *lingüística.*

Desde un punto de vista computacional la manipulación de estas señales es posible sólo si previamente son *digitalizadas;* esto es, el procesamiento de señales digitales de voz comienza con una señal de voz cuantizada en el tiempo, la cual para la computadora será una secuencia de valores digitales. Básicamente el procesamiento consiste en la aplicación de operadores sobre las señales para:

- Extraer parámetros o características de una secuencia
- Producir una secuencia similar con características adicionales
- Restablecer una secuencia desde un estado anterior
- Codificar o comprimir una secuencia

Para nosotros una señal de voz discreta en el tiempo (speech), digamos $s(n)$, será indexada por enteros [20]. Dicha señal representa las muestras de una forma de onda analógica, digamos $s_a(t)$, en algún período de muestreo T,

$$s(n) = s_a(nT) = s_a(t)\big|_{t=nT} \qquad n = \ldots,0,1,2,\ldots \qquad (4.1)$$

donde n es el índice de las muestras.

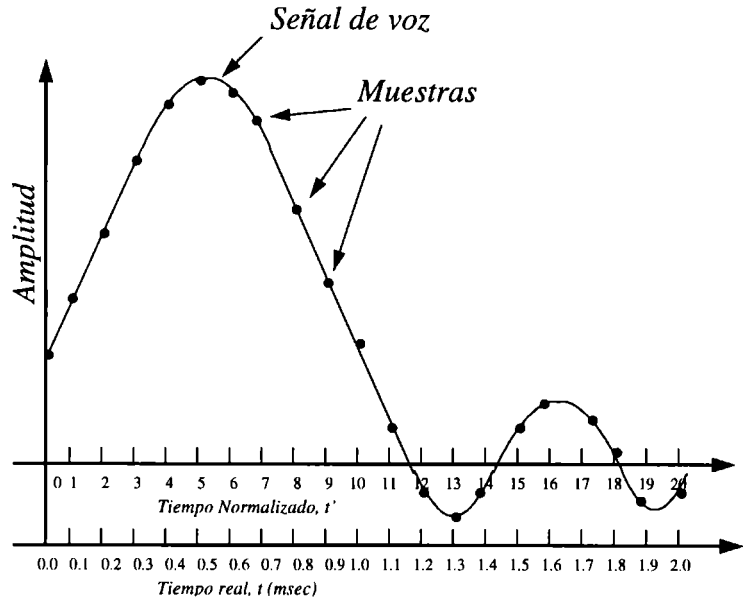


fig. 4.1: Señal de voz discreta en el ti empo.

Hay dos clases básicas de sonidos de speech, vocalizados y no vocalizados. Los primeros están caracterizados por formas de onda acústicas determinísticas, mientras que los segundos corresponden a formas de onda aleatorias. De allí entonces que la teoría de procesos aleatorios será necesaria para analizar señales no vocalizadas, es más, aún en el caso de sonidos vocalizados será muy útil emplear técnicas analíticas motivadas a través de la teoría de procesos aleatorios, principalmente la función de autocorrelación.

Un proceso aleatorio discreto y real está definido como una colección de variables aleatorias, cada una indexadas por un punto en tiempo discreto [6]. El siguiente conjunto describe un proceso aleatorio:

$$\{\ldots,\underline{x}(-1),\underline{x}(0),\underline{x}(1),\ldots\} = \{\underline{x}(n),\quad n\in(-\infty,\infty)\},$$

donde cada variable aleatoria representa un modelo para la generación de valores en su correspondiente tiempo.

La relación entre un proceso aleatorio y un problema físico puede ser vista de la siguiente manera: supongamos que se define un experimento simple en el cual un entero que representa una de L formas de onda de speech es seleccionado en forma aleatoria. Cada tiempo es gorbernado por una variable aleatoria, digamos $\underline{x}(n)$ para el tiempo n, y la colección de estas variables aleatorias es el proceso aleatorio $\underline{x}$. Una vez finalizado el experimento, cada variable aleatoria

mapeará la salida a un nivel de amplitud correspondiente a esa salida. La totalidad de las variables aleatorias producirá una forma de onda particular desde la salida experimental y cada variable aleatoria será responsable de un punto. Esta forma de onda es llamada *realización* del proceso aleatorio. La colección de todas las realizaciones es denominada *ensamble*.

Esta claro que, si seleccionamos un tiempo obtenemos una variable aleatoria, si seleccionamos una salida experimental obtenemos una realización, y si seleccionamos a ambos, un número que es el resultado del mapeo de esa salida a la linea real a través de la variable aleatoria en el tiempo seleccionado.

Dos características básicas de un proceso aleatorio son la correlación y covarianza: sean $\underline{x}(n_1)$ y $\underline{x}(n_2)$ dos variables aleatorias de un proceso aleatorio $\underline{x}$, la correlación de las mismas es $\mathcal{E}\{\underline{x}(n_1)\underline{x}(n_2)\}$. Desde que las dos variables pertenecen al mismo proceso la correlación es denominada autocorrelación y se la caracteriza con una notación especial:

$$r_{\underline{x}}(n_1,n_2) \overset{def}{=} \mathcal{E}\{\underline{x}(n_1)\underline{x}(n_2)\}. \qquad (4.2)$$

Cualquier proceso aleatorio es estacionario de i-ésimo orden si la función de densidad de probabilidad no cambia al considerar cualquier conjunto de i variables aleatorias de dicho proceso con el mismo espaciado relativo que el conjunto original.

A partir de esta definición, si un proceso aleatorio $\underline{x}$ es estacionario al menos de segundo orden el valor de la autocorrelación no depende de cuál de las dos variables son seleccionadas de él, sino más bien de su separación en el tiempo. Por lo tanto una notación a adoptar es:

$$\begin{aligned} r_s(\eta) \overset{def}{=} & \textit{ autocorrelacion de 2 variables aleatorias cualquiera} \\ & \textit{en } \underline{x}\textit{, separadas por } \eta \textit{ en el tiempo.} \\ = & \ \mathcal{E}\{\underline{x}(n)\underline{x}(n-\eta)\} \textit{ para cualquier } \eta. \end{aligned}$$

Todo proceso aleatorio estacionario de i-ésimo orden lo es también de (i-1)-ésimo orden . Así un proceso estacionario de segundo orden lo es también de

primer orden y tiene una media constante ($\mu_{\underline{x}} = \mathcal{E}\{\underline{x}(n)\}$ *para cualquier n.*) , con lo cual se obtiene la siguiente definición: un proceso aleatorio es estacionario en sentido amplio o débilmente estacionario si:

1. *Su autocorrelación es una función de la diferencia de tiempo y*
2. *Su media es constante.*

Finalmente es importante considerar que si $\underline{x}$ es de correlación ergódica entonces la autocorrelación puede ser calculada usando un promedio temporal:

$$r_{\underline{s}}(\eta) = \ell\{s(n)s(n-\eta)\} = \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} s(n)s(n-\eta) \qquad (4.3)$$

El motivo de la presentación de estos conceptos es que la mayoría de las propiedades que se describirán en adelante, utilizarán términos de la teoría de probabilidades desde que se asume que una señal de speech es modelable a través de un proceso aleatorio.

4.2- Ventanas y Bloques (Frames)

En ciertos casos es necesario trabajar con tramos cortos o "frames" de una señal, a menos que la señal sea de corta duración. Esto es especialmente utilizado cuando usamos técnicas de análisis convencional sobre señales, donde es necesario seleccionar una porción de la señal que se comporte en forma estacionaria.

Una "ventana" (dominio en el tiempo), *w(n)*, es una secuencia de longitud finita utilizada para seleccionar un frame deseado, a través de un simple proceso de multiplicación. Por consistencia asumiremos que las ventanas serán secuencias causales y simétricas en el tiempo.

Hay distintos tipos de ventanas como:*rectangulares, de Kaiser, Hamming, Hanning y Blackman.*

Un frame de una señal *s(n)* de longitud *N*, finalizando en tiempo *m* es obtenido de la siguiente manera:

$$f_s(n;m) = s(n)w(m-n) \qquad (4.4)$$

Practicamente un frame es un "trozo" de speech recuperado por una ventana deslizante en el tiempo. Y formalmente es una nueva secuencia sobre n donde se comporta como una señal nula fuera de los rangos $[m\text{-}N+1, m]$.

4.3- Características Short-Term desde conceptos Long-Term

Si deseamos extraer información acerca de un frame de speech en un rango $n=m\text{-}N+1, \ldots, m$, podriamos basarnos intuitivamente en los conceptos Long-Term [6]. Por ejemplo, sea χ_s, una característica Long-Term, y debido a que contamos con un conjunto de las mismas, las diferenciaremos a través de un índice λ, es decir, $\chi_s(\lambda)$. Dichas características son calculadas desde $s(n)$ como sigue:

$$\chi_s(\lambda) = \tilde{\Im}(\lambda)\{s(n)\} = \ell\{\Im(\lambda)\{s(n)\}\}, \qquad (4.5)$$

donde $\tilde{\Im}(\lambda)$ es alguna operación generalmente no lineal y dependiente de λ que se puede descomponer en $\tilde{\Im}(\lambda)=\ell\circ\Im(\lambda)$, siendo $\Im(\lambda)$ una operación que produce una nueva secuencia sobre n y ℓ el operador promedio temporal.

Una forma intuitiva de calcular una característica digamos $\chi_s(\lambda;m)$ utilizando solamente el rango $n\in [m\text{-}N+1, m]$ es mostrada en el siguiente principio de construcción:

1. *Seleccionar la longitud N del frame de s(n) utlizando una ventana, w(n),*

$$f(n;m) = s(n)w(m-n).$$

2. *Aplicar al frame una operación " tipo* $\tilde{\Im}(\lambda)$ *" a la cual llamaremos* $\tilde{S}(\lambda)$:

$$\chi_s(\lambda;m) = \tilde{S}(\lambda)\{s(n)w(m-n)\} = \frac{1}{N}\sum_{n=-\infty}^{\infty} S(\lambda)\{s(n)w(m-n)\} =$$

$$= \frac{1}{N}\sum_{n=-\infty}^{\infty} S(\lambda)\{f(n;m)\},$$

donde se asume que $\tilde{S}(\lambda)$ *puede ser descompuesta como* $(1/N)\sum_{n=-\infty}^{\infty}\circ S(\lambda)$ *al igual que* $\tilde{\Im}(\lambda)$ *en Long-Term.*

Será notado que $\mathcal{S}(\lambda)$ es frecuentemente la misma operación que $\Im(\lambda)$ y por lo tanto nos quedaremos solamente con aquellos casos, obteniendo:

$$\chi_s(\lambda;m)=\frac{1}{N}\sum_{n=-\infty}^{\infty}\Im(\lambda)\{s(n)w(n-m)\}. \qquad (4.6)$$

Lo que hemos hecho hasta aquí es cortar un trozo de speech (creando un frame) y le hemos aplicado una operación similar a aquella que empleamos en la versión Long-Term de la característica. Esta porción de speech fue seleccionada utilizando una ventana la cual ha sido deslizada en el tiempo de manera que termine en el tiempo deseado *m*.

Teniendo en cuenta el principio de arriba podemos definir al *potencia promedio Short-Term* sobre un frame de longitug *N* terminando en tiempo *m* como:

$$P_s(m)=\frac{1}{N}\sum_{n=-\infty}^{\infty}\{s(n)w(n-m)\}^2. \qquad (4.7)$$

Ningún parámetro λ se involucra aquí y $\Im\{\cdot\}=\{\cdot\}^2$.

Otro ejemplo que involucra un parámetro es la *autocorrelación*. Para Long-Term se había definido como:

$$r_s(\eta)=\tilde{\Im}(\eta)\{s(n)\}=\ell\{\Im(\eta)\{s(n)\}\}=\ell\{s(n)s(n-\eta)\}. \qquad (4.8)$$

Vemos que $\Im$ depende del parámetro η que corresponde al orden de la *autocorrelación*, y $\Im(\eta)\{s(n)\}=s(n)s(n-\eta)$.

Aplicando el principio de construcción visto con anterioridad obtenemos el estimador de la *autocorrelación* Short-Term para orden η sobre un frame de longitud *N* terminando en tiempo *m*, de la siguiente manera:

$$\begin{aligned} r_s(\eta;m) &= \frac{1}{N}\sum_{n=-\infty}^{\infty}\Im(\eta)\{s(n)w(m-n)\} \\ &= \frac{1}{N}\sum_{n=-\infty}^{\infty}s(n)w(m-n)s(n-\eta)w(m-n+\eta) \qquad (4.9) \\ &= \frac{1}{N}\sum_{n=m-N+1}^{m}s(n)w(m-n)s(n-\eta)w(m-n+\eta) \end{aligned}$$

El operador $\Im(\lambda)$ en tales problemas debe tener la propiedad de producir una nueva secuencia sobre n desde la secuencia sobre la cual él opera.

Para la elección de una ventana adecuada habrá que tener en cuenta las siguientes consideraciones: en el caso de un N fijo juegan un rol importante la necesidad de evitar la ditorsión de los puntos de la forma de onda y la necesidad de suavizar las discontinuidades abruptas que puede presentar una señal.

En la elección del N hay nuevamente dos factores a tener en cuenta; para un tipo de ventana fijo un N creciente mejora la resolución espectral para un m dado, proveyendo más información para el cálculo; sin embargo debido a que las ventanas se deslizan a través del tiempo, cuando son largas o variables provocan que las cotas fonéticas pierdan definición y que los eventos no sean resueltos a tiempo.

Como una regla podemos adoptar el hecho de que el speech se comporte en forma estacionaria dentro de un frame considerando al mismo de un tamaño de 20 msg.

4.3.1 Estimación de la autocorrelación Short-Term

Podemos interpretar la autocorrelación como un indicador del grado de relación lineal que existe entre dos variables aleatorias cualquiera espaciadas η en un proceso aleatorio estacionario. Para un proceso ergódico se infiere que la autocorrelación define al grado de relación lineal esperado que existe entre dos puntos cualesquiera espaciados por η en el tiempo, en una forma de onda simple.

Aquí se considera el caso Short-Term, al cual se llega por medios puramente heurísticos, y se espera que $r_s(\eta;\ m)$ indique el grado de relación esperado que existe entre puntos separados η sobre una ventana terminando en tiempo m.

La secuencia de autocorrelación fue calculada con anterioridad de la siguiente manera :

$$r_s(\eta) = \tilde{\Im}\ (\eta)\ \{s(n)\} = \ell\ \{\Im\ (\eta)\ \{s(n)\}\} = \ell\ \{s(n)s(n-\eta)\}. \qquad (4.10)$$

Desde que $r_s(\eta)$ es una función uniforme de η es también verdad que:

$$r_s(\eta) = \ell\{s(n)s(n-|\eta|)\}. \tag{4.11}$$

Si queremos calcular la autocorrelación Short-Term para N puntos finalizando en tiempo m. De acuerdo a la sección precedente el estimador tiene la siguiente forma:

$$\begin{aligned} r_s(\eta;m) &= \frac{1}{N}\sum_{n=-\infty}^{\infty}\Im(\eta)\{s(n)w(m-n)\} \\ &= \frac{1}{N}\sum_{n=-\infty}^{\infty} s(n)w(m-n)s(n-|\eta|)w(m-n+|\eta|). \end{aligned} \tag{4.12}$$

Ahora podríamos calcular el mismo estimador pero utilizando convolución con $s_{\Im}(n;\eta) = s(n)s(n-|\eta|)$ y $w_{\Im}(n;\eta) = w(n)w(n-|\eta|)$ teniendo en cuenta que la ventana es rectangular. Resulta entonces el siguiente estimador:

$$r_s(\eta;m) = \frac{1}{N}\sum_{n=m-N+1+|\eta|}^{m} s(n)w(m-n). \tag{4.13}$$

4.3.2 Estimación de la covarianza Short-Term

Considere por un momento que la secuencia $s(n)$ es una realización de un proceso aleatorio, $\underline{s}$. En general la autocorrelación de un proceso aleatorio es una esperanza estadística que es función de dos parámetros rotados, α y β:

$$r_{\underline{s}}(\alpha;\beta) = \mathcal{E}\{\underline{s}(n-\alpha)\underline{s}(n-\beta)\}, \tag{4.14}$$

donde $\underline{s}(n)$ es la variable aleatoria en tiempo n en $\underline{s}$.Solamente cuando $\underline{s}$ es "estacionario en sentido amplio" (WSS : Wide Sense Stationary), $r_{\underline{s}}$ se vuelve una función de la diferencia de tiempo (absoluta), $\eta = \alpha - \beta$, y en este caso puede ser escrito con un único argumento:

$$\begin{aligned} r_{\underline{s}}(\alpha;\beta) &= \mathcal{E}\{\underline{s}(n-\alpha)\underline{s}(n-\beta)\} \\ &= \mathcal{E}\{\underline{s}(n)\underline{s}(n-\eta)\} = r_{\underline{s}}(0;\eta) \end{aligned} \tag{4.15}$$

Por comodidad nos referimos a $r_{\underline{s}}(\eta)$ para indicar $r_{\underline{s}}(0;\eta)$.

Cuando $\underline{s}$ no es estacionario es necesario involucrar los dos argumentos en el cálculo, sin embargo trabajar con la versión de $r_{\underline{s}}$ con dos argumentos es

matemáticamente correcto y la correspondiente función de autocorrelación temporal de dos argumentos se define como:

$$r_{\underline{s}}(\alpha;\beta) = \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} s(n-\alpha)s(n-\beta) = \ell\{s(n-\alpha)s(n-\beta)\}. \quad (4.16)$$

Generalizando el operador $\Im$ para dos parámetros tenemos $\Im(\alpha;\beta)$ y la definición Short-Term de (4.15) es :

$$\varphi_s(\alpha,\beta;m) = \frac{1}{N} \sum_{n=-\infty}^{\infty} s_{\Im}(n;\alpha,\beta)w(m-n) \quad (4.17)$$

donde $s_{\Im}(n;\alpha,\beta) = \Im(\alpha,\beta)\{s(n)\} = s(n-\alpha)s(n-\beta)$ Definiendo a $w(n)$ como una ventana rectangular de longitud N dada por :

$$w(n) = \begin{cases} 1, & n = 0,1,\ldots,N-1 \\ 0, & \text{caso contrario} \end{cases}$$

la función de covarianza Short-Term resulta en:

$$\varphi_s(\alpha,\beta;m) = \frac{1}{N} \sum_{n=m-N+1}^{m} s(n-\alpha)s(n-\beta). \quad (4.18)$$

Tanto la definición de covarianza como la de correlación son utilizadas incorrectamente en el procesamiento de señales. Por ejemplo, es frecuente referenciar al uso de la covarianza en el caso no estacionario, cuando en realidad si consideramos que el speech es no estacionario por naturaleza se debe considerar a la misma como la autocorrelación de dos argumentos en el caso de un proceso aleatorio que no es WSS.

Por otro lado con un estimador de uno o dos argumentos se asume que la señal es estacionaria Long-Term y se intenta estimar propiedades Long-Term (autocorrelación) sobre el frame Short-Term. Si no se modelara el problema de esa forma, la forma de onda no podría ser asumida ergódica[1] y muchos usos como el promedio temporal no tendrían sentido.

Para N grandes, todos los estimadores precedentes se vuelven equivalentes en el sentido de que son aproximaciones a la Long-Term $r_{\underline{s}}(|\eta|)$.

[1] Una señal es ergódica si promedios adel ensamble pueden ser reemplazados por promedios temporales.

4.3.3 Estimación de la cros-correlación Short-Term

Desde cualquiera de los estimadores Short-Term de autocorrelación, se puede inferir una función similar Short-Term denominada *"croscorrelación"* de la siguiente manera:

$$r_{xy}(\eta;m)=\frac{1}{N}\sum_{n=m-N+1+|\eta|}^{m}x(n)y(n-|\eta|) \qquad (4.20)$$

para dos secuencias $x(n)$ e $y(n)$ como siempre consideradas estacionarias.

4.3.4 Cálculo de la medida de cruces por cero

El número de cruces por cero (número de veces que la secuencia cambia de signo) es una característica muy útil en análisis de speech. Formalmente es definido en Long-Term como:

$$Z_s=\ell\left\{\frac{\left|sgn\{s(n)\}-sgn\{s(n-1)\}\right|}{2}\right\}, \qquad (4.21)$$

donde

$$sgn\{s(n)\}=\begin{cases}1, & s(n)\geq 0\\ -1, & s(n)<0\end{cases}. \qquad (4.22)$$

La medida de cruce de cero para un intervalo de longitud N terminando en $n=m$ será:

$$Z_s(m)=\frac{1}{N}\sum_{n=m-N+1}^{m}\frac{\left|sgn\{s(n)\}-sgn\{s(n-1)\}\right|}{2}w(n-m) \qquad (4.23)$$

La tasa de cruces por cero es un estadístico que nos permite extraer determinados parámetros de una señal de voz, como por ejemplo, establecer la presencia de sonidos fricativos[2] en una pronunciación.

4.4- Señales de fase mínima

[2] Producidos mediante la excitación del tracto vocal con una corriente de aire uniforme que se vuelve turbulenta en algún punto de contracción (bilabial, labio-dental, interdental, alveolar, palatal, velar o glotal). Son ejemplos de fricativas /v, d, z, t, f, s/.

Desde que la fase negativa para una frecuencia en particular (en el espectro) está relacionada con la cantidad de retardos temporales de una componente en esa frecuencia, se puede inferir que una señal de mínima fase es aquella que para un espectro de magnitud dado, tiene un mínimo retardo de cada componente de frecuencia en el espectro. Dicha señal, además, tiene la mayor concentración de energía cerca del tiempo $n=0$ que cualquier señal con el mismo espectro de magnitud. En otras palabras si $x_{min}(n)$ es la señal de mínima fase e $y(n)$ cualquier señal con el mismo espectro de magnitud, ocurre que:

$$E_{x_{min}}(m) \geq E_y(m) \qquad E_x(m) \stackrel{def}{=} \sum_{n=0}^{m} x^2(n), \qquad \forall m \qquad (4.23)$$

Otro modo de ver una señal de mínima fase, particularmente cuando representa la respuesta impulsiva de un sistema es la siguiente: si $h(n)$ representa la respuesta impulsiva de mínima fase de un sistema causal y estable, luego la función de sistema en el dominio transformado z , $H(z)$, tendrá todos sus polos y ceros dentro del círculo unitario. Así existirá un sistema inverso causal y estable $H^{-1}(z)$ tal que:

$$H(z)H^{-1}(z) = 1, \qquad (4.24)$$

en cualquier punto en el plano z. En el caso de que al menos un cero se encuentre fuera del círculo unitario en $H(z)$, entonces no existe la inversa estable, desde que al menos un polo en la inversa estaría ubicado fuera del círculo unitario. La existencia de la transformada z inversa, estable y causal de $H(z)$ es condición suficiente para asegurar que *la señal h(n) es mínima en fase.*

Los conceptos de señales y sistemas de mínima fase juegan un rol importante en la teoría de predicción lineal, de ahí que se han presentado previo a la introducción en dicho tema.

Capítulo 5

Análisis de Predicción Lineal

5.1- Predicción Lineal "Long-Term"

5.1.1 Modelo de sólo polos

Un modelo de producción de voz discreto en el tiempo durante un frame estacionario de una señal de voz, puede ser caracterizado idealmente por una función de sistema de polos y ceros de la siguiente forma :

$$\Theta(z) = \Theta_0 \frac{1 + \sum_{i=1}^{L} b(i) z^{-1}}{1 - \sum_{i=1}^{R} a(i) z^{-1}} \qquad (5.1)$$

el cual es manejado por una secuencia de excitación:

$$e(n) = \begin{cases} \sum_{q=-\infty}^{\infty} \delta(n - qP), & \text{caso vocalizado} \\ \text{media cero, varianza unitaria,} & \\ \text{ruido no correlacionado} & \text{caso no vocalizado} \end{cases} \qquad (5.2)$$

y el diagrama en bloque asociado es:

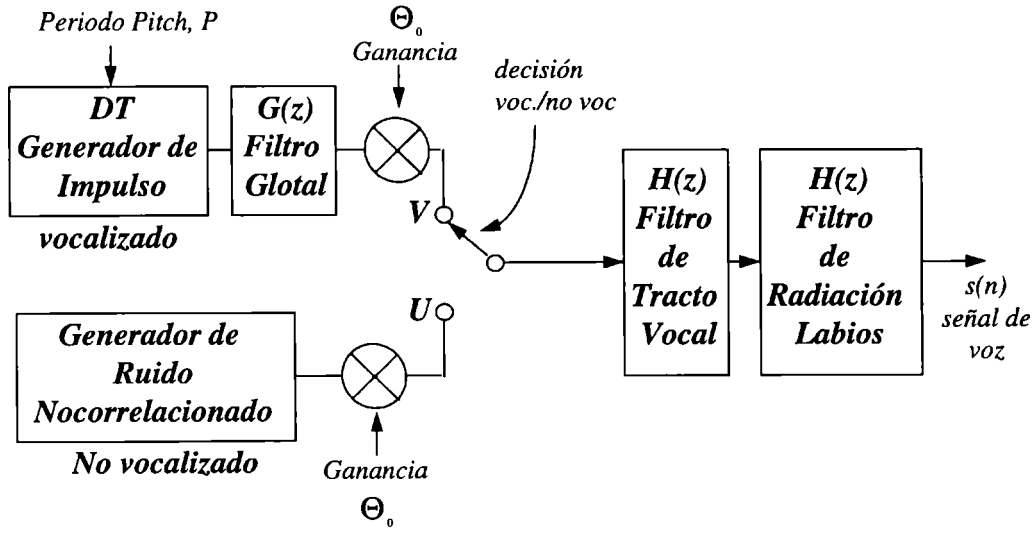


Fig 5. 1 Diagrama en bloques del modelo de producción de speech. Modelo ideal.

No obstante cuando se diseñan algoritmos de predicción lineal, el objetivo es encontrar los parámetros asociados con una función de sistema de sólo polos que sirve para la construcción de un modelo que es la estimación del modelo ideal $\Theta(z)$. Dicha función de sistema de sólo polos es la siguiente:

$$\hat{\Theta}(z) = \frac{1}{1 - \sum_{i=1}^{M} \hat{a}(i) z^{-i}} \qquad (5,3)$$

y el diagrama en bloque para el modelo a ser estimado utilizando análisis de predicción lineal es:

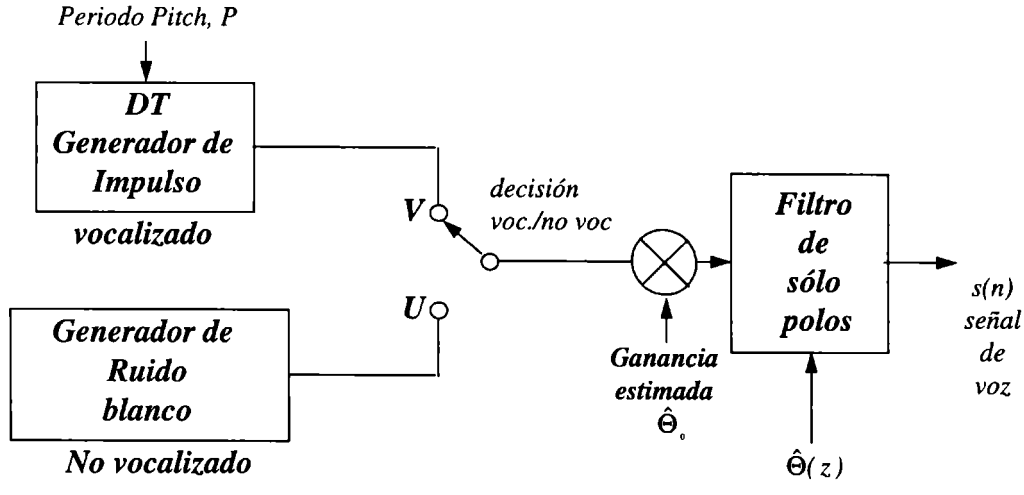


Fig 5. 2 Modelo a ser estimado utilizando análisis de Predicción Lineal.

El motivo por el cual un sistema de sólo polos es usado es que ciertas clases de fonemas, como las vocales, involucran configuraciones del tracto vocal que son de resonancia acústica y modelables a través de estructuras de sólo polos (Fant, 1956, 1960; Flanagan, 1972). Por otro lado, los fonemas nasales y fricativos pueden ser modelados por la inclusión de cavidades acústicas en el tracto vocal lo cual contendrá espectro nulo, matemáticamente hablando, se corresponde con ceros en la función de sistema.

La necesidad de este modelo es básicamente analítica. Los parámetros de Predicción Lineal (PL) pueden ser determinados utilizando estrategias (resultando en simples ecuaciones lineales) sobre una escaza información acerca del sistema ideal, esto es, su salida ($s(n)$).

La necesidad de los ceros depende de si el modelo PL desea preservar la información acústica en la señal de voz (speech) sin estar ligado a la relación temporal, esto es, las relaciones de fase entre los componentes de speech no tienen efecto en la percepción pues el oído humano es insensible a la fase. No obstante la magnitud del espectro es la que define el nivel de audibilidad de la señal de voz. Además dicha magnitud espectral, es decir la información en la señal de voz, es perfectamente modelable a partir de un sistema de sólo polos. Si el objetivo es codificar, almacenar, resintetizar, etc., sólo es necesario las características de magnitud espectral para poder modelar perfectamente.

La naturaleza de sólo polos de la representación PL restringe al modelo a ser de mínima fase con lo cual se aparta de las verdaderas características de la señal que está siendo codificada.

5.1.2 Ecuaciones de Predicción Lineal

El objetivo es estimar los parámetros $a(i)$ de $\Theta(z)$, a los cuales los llamaremos $\hat{a}(i)$ por consecuencia al nuevo modelo estimado $\hat{\Theta}(z)$. Estos números (estimadores) obtenidos constituyen una representación o codificación de la forma de onda conteniendo toda la información de la magnitud espectral excepto el pitch[1] y ganancia. Estos M estimadores son más eficientes que manejar datos de gran tamaño, para almacenar, transmitir, reconocer, o síntesizar junto con otros pocos parámetros más.

En el dominio del tiempo:

$$s(n) = \sum_{i=1}^{M} a(i)s(n-i) + \Theta_0 e'(n) \qquad (5.4)$$

lo cual puede ser expresado en forma matricial como

$$s(n) = \mathbf{a}^{\mathrm{T}}\mathbf{s}(n) + \Theta_0 e'(n) \qquad (5.5)$$

con Θ_0 una constante relacionada a las singularidades de $\Theta(z)$ y $e'(n)$ una versión alterada en fase de $e(n)$.

[1]Frecuencia fundamental de fonación, esto es la tasa de vibraciones de las cuerdas vocales en una unidad de tiempo.

La salida *s(n)* puede ser predecida usando una combinación lineal de sus *M* valores pasados, estadísticamente hablando puede ser llamada regresiva sobre si misma y por lo tanto el modelo ser *autoregresivo* (típicamente llamado modelo A R de orden *M* debido a sus *M* parámetros o polos). Los *a(i)* son los coeficientes de la ecuación predictora, y por lo tanto sus estimadores obtenidos mediante análisis PL son frecuentemente llamados codificación predictiva lineal.

A lo largo de esta presentación los estimadores de los números *a(i)* $i=1, 2, ..., M$ denotados con $\hat{a}(i)$ serán considerados como *parámetros "a", parámetros* $\hat{a}(i)$, *parámetros PL o coeficientes PL* y se tendrá en cuenta la siguiente notación:

$$\mathbf{a} \overset{def}{=} [a(1)a(2)...a(M)]^T \qquad (5.6)$$

$$\mathbf{s}(n) \overset{def}{=} [s(n-1)s(n-2)...s(n-M)]^T$$

el superíndice *T* es utilizado para hacer referencia a la traspuesta de una matriz dada.

Es posible encontrar los parámetro PL resolviendo cuatro problemas interpretativos diferentes los cuales concluyen en el mismo resultado y permiten interpretar el modelo resultante en una forma distinta. Estas interpretaciones del modelo PL son:

- Identificación de sistema
- Filtrado inverso
- Predicción lineal
- Flattening[2] espectral

5.1.2.1 Interpretación del modelo PL vía identificación de sistema

Dado un sistema no conocido con entrada conocida y salida como muestra la Fig. 5.3 un método para deducir un modelo es atribuirle una respuesta

[2] Allanamiento.

impulsiva (RI) que minimice el error cuadrático medio de salida. Sea $\hat{\Theta}(z)$ la representación del modelo; se trata de encontrar $\hat{\theta}(n)$ tal que:

$$\lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} \tilde{s}^2(n) \qquad (5.7)$$

sea mínimo. Considerando que la señal de voz es un proceso aleatorio estacionario, la aplicación de este límite es adecuada.

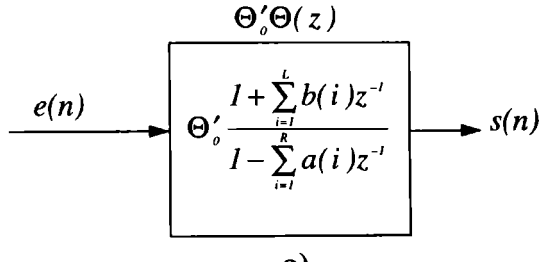
Este método falla desde que los parámetros $\hat{a}(i)$ están relacionados en forma no lineal con $\hat{\theta}(n)$. Si "damos vuelta el problema" como en el inciso c) de la Fig.5.3 y aplicamos la RI al modelo inverso en el mismo sentido, la RI, digamos $\alpha(n)$, $n=0,1,...,M$ está claramente relacionada a los parámetros deseados, desde

$$\hat{A}(z) = \alpha(0) + \sum_{i=1}^{\infty} \hat{\alpha}(i) z^{-i} \qquad (5.8)$$

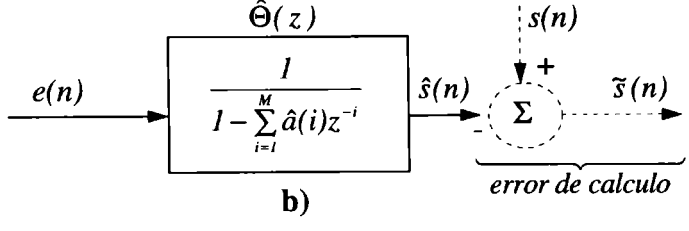
y la forma del modelo deseado es:

$$\hat{A}(z) = 1 - \sum_{i=1}^{M} \hat{a}(i) z^{-i} \qquad (5.9)$$

Si comparamos ambas ecuaciones vemos que la forma del modelo presenta ciertas restricciones sobre la RI como $\alpha(0)=1$, debido a la carencia de conocimiento de la ganancia del sistema; y $\alpha(n)=0$ para $n>M$.



a)



b)

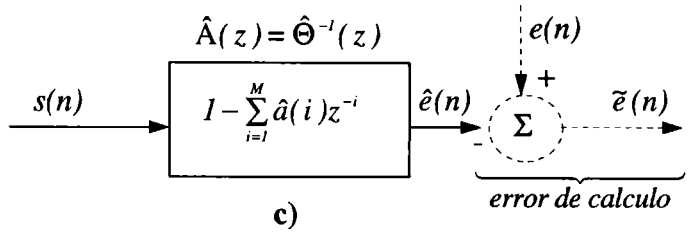


Fig 5. 3 a) Modelo ideal **b)** Modelo estimado **c)** Modelo inverso estimado.

A partir de estas retricciones el modelo buscado será obtenido resolviendo el siguiente problema interpretativo que nos permite encontrar los parámetros PL deseados.

Problema: *Diseñar el filtro de respuesta de impulso finita (FIR) de longitud M+1, con la restricción $\alpha(0)=1$, que minimize*

$$\lim_{N\to\infty}\frac{1}{2N+1}\sum_{n=-N}^{N}\tilde{e}^2(n) \qquad (5.10)$$

cuando s(n) es entrada (los parámetros PL están dados por $\hat{a}(i)=-\alpha(i)$, $i=1, 2, ..., M$.

Veamos la solución al problema planteado:

A partir de la Fig.5.3 y si denominamos

$$\ell\left\{\tilde{e}^2(n)\right\} \qquad (5.11)$$

al límite antes mencionado se tiene

$$\ell\left\{\tilde{e}^2(n)\right\}=\ell\left\{\left[\sum_{i=0}^{M}\alpha(i)s(n-i)-e(n)\right]^2\right\}. \qquad (5.12)$$

Diferenciando con respecto $\alpha(\eta)$, $\eta=1, 2, ..., M$ e igualando el resultado a cero obtenemos

$$2\ell\left\{\left[\sum_{i=0}^{M}\alpha(i)s(n-i)-e(n)\right]s(n-\eta)\right\}=0 \qquad (5.13)$$

y luego

$$\ell\left\{\sum_{i=0}^{M}\alpha(i)s(n-i)s(n-\eta)\right\}-\ell\{e(n)s(n-\eta)\}=0 \qquad \eta=1,\ 2,\ \ldots,\ M, \qquad (5.14)$$

lo cual si aplicamos la propiedad del límite con respecto a la suma y teniendo en cuenta que $\alpha(0)=1$ resulta en

$$\ell\{s(n)s(n-\eta)\}+\ell\left\{\sum_{i=1}^{M}\alpha(i)s(n-i)s(n-\eta)\right\}-\ell\{e(n)s(n-\eta)\}=0 \qquad \eta=1,\ 2,\ \ldots,\ M, \qquad (5.15)$$

donde de acuerdo a las definiciones del capítulo anterior ,

$$r_s(\eta)=\ell\{s(n)s(n-\eta)\} \quad y \quad r_{es}(\eta)=\ell\{e(n)s(n-\eta)\} \qquad (5.16)$$

es decir $r_s(\eta)$ es la autocorrelación temporal de $s(n)$ y $r_{es}(\eta)$ es la croscorrelación temporal de las secuencias $e(n)$ y $s(n)$, las cuales son realizaciones de procesos aleatorios ergódicos de segundo orden WSS.

Reescribiendo (5.15) tenemos

$$r_s(\eta)+\sum_{i=1}^{M}\alpha(i)\,r_s(\eta-i)-r_{es}(\eta)=0 \qquad (5.17)$$

Si consideramos a $s(n)$ una realización de un proceso aleatorio $\underline{s}$ WSS con variables aleatorias $\underline{s}(n)$, entonces de acuerdo al *principio de ortogonalidad* (Gray y Davisson, 1986, pag. 204-205) el filtro:

$$P(z)=\sum_{i=1}^{M}\hat{a}(i)z^{-i} \qquad (5.18)$$

produce un error cuadrático medio mínimo si y sólo si la variable aleatoria $\underline{\hat{e}}(n)$ (perteneciente al proceso aleatorio $\underline{\hat{e}}$) es ortogonal a la variable aleatoria $s(l)$ para $l=n-M,\ \ldots,\ n-1$, o sea $\mathcal{E}\{\underline{\hat{e}}(n)\underline{s}(l)\}=0$ para cualquier n. Esto puede ser escrito como:

$$\mathcal{E}\{\underline{\hat{e}}(n)\underline{s}(n-\eta)\}=0 \qquad para\ \eta=1,\ \ldots,\ M \qquad (5.19\,a)$$

o

$$\ell\{\underline{\hat{e}}(n)\underline{s}(n-\eta)\}=r_{\hat{e}s}(\eta)=0 \qquad para\ \eta=1,\ \ldots,\ M \qquad (5.19\,b)$$

asumiendo que $\underline{s}$ es un proceso aleatorio ergódico.

Así como $\underline{e}$ es ortogonal a $\underline{s}$, $r_{es}(\eta)=0 \quad para\ \eta=1,\ \ldots,\ M$

Luego de esta consideración, a partir de (5.17) podemos obtener

$$\sum_{i=1}^{M}\alpha(i)r_s(\eta-i)=-r_s(\eta),\quad \eta=1,\ 2,\ \ldots.M \qquad (5.20)$$

y recordando que $\hat{a}(i)=-\alpha(i),\ i=1,\ 2,\ \ldots,\ M$ la igualdad anterior se transforma en

$$\sum_{i=1}^{M}\hat{a}(i)r_s(\eta-i)=r_s(\eta),\quad \eta=1,\ 2,\ \ldots,\ M \qquad (5.21)$$

Así con estas M ecuaciones (para los distintos η) es posible calcular los parámetros PL convencionales, y por comodidad se los quiere tener empaquetados como una ecuación de vectores y matrices de la siguiente manera:

$$\begin{bmatrix} r_s(0) & r_s(1) & r_s(2) & \cdots & r_s(M-1) \\ r_s(1) & r_s(0) & r_s(1) & \cdots & r_s(M-2) \\ r_s(2) & r_s(1) & r_s(0) & \cdots & r_s(M-3) \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ r_s(M-1) & r_s(M-2) & r_s(M-3) & \cdots & r_s(0) \end{bmatrix} * \begin{bmatrix} \hat{a}(1) \\ \hat{a}(2) \\ \hat{a}(3) \\ \vdots \\ \hat{a}(M) \end{bmatrix} = \begin{bmatrix} r_s(1) \\ r_s(2) \\ r_s(3) \\ \vdots \\ r_s(M) \end{bmatrix}$$

lo cual en forma compacta lo denotaremos como

$$\mathbf{R}_s\hat{\mathbf{a}}=\mathbf{r}_s \Rightarrow\ \hat{\mathbf{a}}=\mathbf{R}_s^{-1}\mathbf{r}_s. \qquad (5.22)$$

5.2- Análisis de Predicción Lineal "Short-Term"

Hay dos soluciones para PL Short-Term: el método de autocorrelación y el de la covarianza.

5.2.1 Método de Autocorrelación

Este método tiene dos ventajas con respecto al otro y es que se asegura al menos en forma teórica que el modelo se mantiene estable y emplea un sólo estimador para la autocorrelación.

Se trabajará sobre N datos tomados hasta el tiempo m, es decir $s(m-N+1)$, $s(m-N+2)$, ..., $s(m)$; para estimar los parámetros de predicción lineal. Al vector de los M parámetros estimados los denotaremos como $\hat{\mathbf{a}}(m)$. Dichos parámetros se encuentran asociados a la secuencia de datos $s(n)$.

Teniendo en cuenta la definición de autocorrelación para el caso *Long-Term* dada en la sección precedente, un estimador natural para los parámetros de PL que utilizan sólo los datos especificados es aquel que resulta de la inserción de uno de los estimadores *Short-Term* para la autocorrelación que vimos en el capítulo anterior, es decir:

$$\sum_{i=1}^{M} \hat{a}(i;m) r_s(\eta - i;m) = r_s(\eta;m), \quad \eta = 1, 2, \ldots M \qquad (5.23)$$

y cuya representación matricial es la siguiente:

$$\mathbf{R}_s(m)\hat{\mathbf{a}}(m) = \mathbf{r}_s(m) \Rightarrow \hat{\mathbf{a}}(m) = \mathbf{R}_s^{-1}(m)\mathbf{r}_s(m). \qquad (5.24)$$

donde $\mathbf{R}_s(m)$ es la matriz de autocorrelación Short-Term, la que se considera como un operador Toeplitz ya que todos los elementos de sus diagonales son iguales. La solución a la última ecuación planteada es lo que llamamos *método de autocorrelación* para calcular o estimar los coeficientes de Predicción Lineal.

La mayoría de la literatura dedicada al cálculo de coeficientes de PL presentan el *método de autocorrelación* resolviendo el siguiente problema:

Problema: *Dado el frame de speech f(n;m)= s(n)w(m-n), encontrar el predictor lineal de f(n;m), que minimice el error cuadrático total en predicción para todo n. (Tener en cuenta que el error será mínimo para todo n y no para el rango seleccionado por la ventana).*

Antes de presentar una solución al problema incorporaremos algunas notaciones útiles. Denotaremos con $\hat{f}(n;m)$ al valor predecido del punto $f(n;m)$ y $\varepsilon(n;m)$ el error de predicción al instante n (el cual no surge de un frame del error long-term $\hat{e}(n)$, esto es, $\varepsilon(n;m) \neq \hat{e}(n)w(m-n)$),

$$\varepsilon(n;m) = f(n;m) - \hat{f}(n;m). \qquad (5.25)$$

Y $N\xi(m)$[3] al error cuadrático de predicción total para todo n, relacionado con el análisis del frame que finaliza al instante m.

Lo que se quiere minimizar es:

[3] N ha sido introducido como un factor extra y es la longitud de la ventana.

$$\xi\ (m) = \frac{1}{N} \sum_{n=-\infty}^{\infty} \varepsilon^2(n;m) = \frac{1}{N} \sum_{n=-\infty}^{\infty} \left[f(n;m) - \sum_{i=1}^{M} \hat{a}(i;m) f(n-i;m) \right]^2 . \qquad (5.26)$$

para lo cual diferenciamos $\xi\ (m)$ respecto de $\hat{a}(\eta\ ;m)$ e igualamos a cero obteniendo:

$$\frac{\partial\, \xi\,(m)}{\partial\, \hat{a}(\eta\ ;m)} = \frac{2}{N} \sum_{n=-\infty}^{\infty} \left[f(n;m) f(n-\eta\ ;m) - \sum_{i=1}^{M} \hat{a}(i;m) f(n-i;m) f(n-\eta\ ;m) \right] = 0. \qquad (5.27)$$

Si dividimos por 2 y distribuimos la sumatoria junto con el factor 1/N resulta en:

$$\frac{1}{N} \sum_{n=-\infty}^{\infty} \left[f(n;m) f(n-\eta\ ;m) \right] - \frac{1}{N} \sum_{n=-\infty}^{\infty} \left[\sum_{i=1}^{M} \hat{a}(i;m) f(n-i;m) f(n-\eta\ ;m) \right] = 0, \qquad (5.28)$$

y si consideramos que:

$$f(n;m) = s(n) w(m-n) \qquad (5.29)$$

podemos concluir que :

$$\frac{1}{N} \sum_{n=-\infty}^{\infty} \left[s(n) w(m-n) s(n-\eta) w(m-n+\eta) \right] = r_s(\eta;m)$$

$$\frac{1}{N} \sum_{n=-\infty}^{\infty} \left[\sum_{i=1}^{M} \hat{a}(i;m) s(n-i) w(m-n+i) s(n-\eta) w(m-n+\eta) \right] = \sum_{i=1}^{M} \hat{a}(i;m) r_s(\eta - i;m) \qquad (5.30)$$

y así se llega a que:

$$\sum_{i=1}^{M} \hat{a}(i;m) r_s(\eta - i;m) = r_s(\eta;m), \quad \eta = 1, 2, \ldots, M \qquad (5.31)$$

considerando la ventana especificada en el problema y la definición de autocorrelación Short-Term dada en el Capítulo 4.

5.2.2 Método de Covarianza

Otra forma de obtener estimadores de los parámetros PL de un frame,

s(m-N+1), ..., s(m), es insertar la función de covarianza Short-Term, presentada en el Capítulo 4, en las ecuaciones utilizadas para calcular los parámetros PL Long-Term; como una estimación de la autocorrelación:

$$\sum_{i=1}^{M} \hat{a}(i;m)\,\varphi_s(i,v;m) = \varphi_s(0,v;m), \quad v = 1, 2, \ldots, M. \tag{5.32}$$

Su forma matricial es la siguiente:

$$\Phi_s(m)\hat{\mathbf{a}}(m) = \varphi_s(m) \Rightarrow \hat{\mathbf{a}}(m) = \Phi_s^{-1}(m)\varphi_s(m), \tag{5.33}$$

donde $\Phi_s(m)$ es una matriz de M x M donde el elemento (η, v) es $\varphi_s(\eta,v;m)$; y

$$\varphi_s(m) \overset{def}{=} \left[\varphi_s(0,1;m)\varphi_s(0,2;m)\cdots\varphi_s(0,M;m)\right]^T. \tag{5.34}$$

La matriz $\Phi_s(m)$ es llamada *matriz de covarianza Short-Term* ; es simétrica pero no Toeplitz. La ecuación antes presentada es la solución al *método de la covarianza.*

Es interesante notar que el *método de la covarianza* hace uso de algunos puntos de datos fuera del rango $n \in [m - N + 1, m]$ cuando calcula $\varphi_s(\eta,v; m)$; esto es debido a que se consideran datos sobre los cuales la técnica minimiza la energía del error predecido, como muestra la solución al siguiente problema:

Problema: *Encontrar el predictor lineal de s(n) que minimiza el error cuadrático medio en el rango n∈ [m-N+1, m].*

La solución trata de minimizar:

$$\xi\,(m) = \frac{1}{N}\sum_{n=m-N+1}^{m}\left[s(n) - \hat{s}(n)\right]^2 = \frac{1}{N}\sum_{n=m-N+1}^{m}\varepsilon^2(n;m). \tag{5.35}$$

Si tenemos en cuenta que

$$\hat{s}(n) = \sum_{i=1}^{M}\hat{a}(i;m)s(n-i), \tag{5.36}$$

reemplanzando en (5.35) resulta:

$$\xi\,(m) = \frac{1}{N}\sum_{n=m-N+1}^{m}\left[s(n) - \sum_{i=1}^{M}\hat{a}(i;m)s(n-i)\right]^2. \tag{5.37}$$

Luego si diferenciamos $\xi\,(m)$ respecto de $\hat{a}(v;m)$ e igualamos a cero, tenemos:

$$\frac{\partial \xi(m)}{\partial \hat{a}(v;m)} = \frac{2}{N} \sum_{n=m-N+1}^{m} \left[s(n)s(n-v) - \sum_{i=1}^{M} \hat{a}(i;m)s(n-i)s(n-v) \right] = 0, \qquad (5.38)$$

donde si dividimos por 2 y distribuimos la sumatoria externa y el factor $1/N$ obtenemos :

$$\frac{1}{N} \sum_{n=m-N+1}^{m} [s(n)s(n-v)] - \frac{1}{N} \sum_{n=m-N+1}^{m} \left[\sum_{i=1}^{M} \hat{a}(i;m)s(n-i)s(n-v) \right] = 0, \qquad (5.39)$$

y si nos remitimos a las definiciones de covarianza del Capítulo 4 podemos reemplazar esta ecuación por:

$$\varphi_s(0,v;m) - \sum_{i=1}^{M} \hat{a}(i,m)\varphi_s(i,v;m) = 0, \quad v = 1, 2, ..., M. \qquad (5.40)$$

Efectuando un pasaje de términos se llega a la ecuación que soluciona el método de la covarianza, es decir el resultado que se deseaba obtener:

$$\sum_{i=1}^{M} \hat{a}(i,m)\varphi_s(i,v;m) = \varphi_s(0,v;m), \qquad v = 1, 2, ..., M. \qquad (5.41)$$

En este método la predicción se realiza respecto de la señal sin alteraciones, esto es, no se obtiene primero un *frame de speech.* Por esta razón concluimos que: mientras que en el método de la autocorrelación primero se "ventanea" la señal y luego se trata de minimizar el error en la predicción de la señal "ventaneada" para todo *n*, en el método de la covarianza se trata de minimizar el error en la predicción de la señal sobre un rango especificado de puntos.

5.2.3 El problema clásico de los Mínimos Cuadrados

El método de la covarianza mantiene una relación que involucra un conjunto de resultados intermedios y prácticos con la solución al problema de los mínimos cuadrados: *dadas una serie de N muestras s(1), s(2), ..., s(N), y un conjunto de vectores relacionados* **s**(1), s(2), ..., **s**(N), *se quiere encontrar el predictor lineal que los relaciona, digamos* $\hat{\mathbf{a}}(N)$, *que minimice el error cuadrático medio; esto es:*

$$\xi\ (N) = \frac{1}{N}\sum_{n=1}^{N}[s(n) - \hat{s}(n)]^2, \qquad (5.42)$$

donde $\hat{s}(n)$ es la predicción de la muestra n-ésima,

$$\hat{s}(n) \overset{def}{=} \sum_{i=1}^{M}\hat{a}(i;N)s(n-i). \qquad (5.43)$$

Aquí asumiremos que la ventana tiene una longitud de N, comenzando en 1, con $m=N$ y $m-N+1=1$.

En términos matriciales el problema sería encontrar la solución ($\hat{\mathbf{a}}(N)$) al siguiente sistema de ecuaciones sobredeterminado[4] :

$$\mathbf{S}(N)\hat{\mathbf{a}}(N) = \bar{\mathbf{s}}(N), \qquad (5.44)$$

donde $\bar{\mathbf{s}}(N)$ trata de diferenciarse de $\mathbf{s}(N)$ que identifica a los M valores pasados de la señal a partir del instante N.

Según [13], la solución es la siguiente:

$$\frac{1}{N}\mathbf{S}^{\mathbf{T}}(N)\mathbf{S}(N)\hat{\mathbf{a}}(N) = \frac{1}{N}\mathbf{S}^{\mathbf{T}}(N)\bar{\mathbf{s}}(N), \qquad (5.45)$$

siendo :

$$\mathbf{S}^{\mathbf{T}}(N) \overset{def}{=} [\mathbf{s}(1)\mathbf{s}(2)\cdots\mathbf{s}(N)]$$
$$\bar{\mathbf{s}}(N) \overset{def}{=} [s(1)s(2)\cdots s(N)]^T. \qquad (5.46)$$

Comparando las siguientes ecuaciones:

$$a)\quad \mathbf{\Phi}_{\mathbf{s}}(m)\hat{\mathbf{a}}(m) = \varphi_s(m)$$
$$b)\quad \frac{1}{N}\mathbf{S}^{\mathbf{T}}(N)\mathbf{S}(N)\hat{\mathbf{a}}(N) = \frac{1}{N}\mathbf{S}^{\mathbf{T}}(N)\bar{\mathbf{s}}(N), \qquad (5.47)$$

y considerando que $N=m$ vemos que :

$$\mathbf{\Phi}_{\mathbf{s}}(m) = \frac{1}{N}\mathbf{S}^{\mathbf{T}}(N)\mathbf{S}(N)$$
$$\varphi_s(N) = \frac{1}{N}\mathbf{S}^{\mathbf{T}}(N)\bar{\mathbf{s}}(N). \qquad (5.48)$$

[4] Denotación para sistemas con mayor número de ecuaciones que incógnitas.

de modo que (5.47) *b)* es precisamente la solución al método de la covarianza.

Agregando la noción de *"peso"* a esta solución, se obtiene:

$$\frac{1}{N}\mathbf{S}^{\mathrm{T}}(N)\Lambda(N)\mathbf{S}(N)\hat{\mathbf{a}}(N)=\frac{1}{N}\mathbf{S}^{\mathrm{T}}(N)\Lambda(N)\bar{\mathbf{s}}(N), \qquad (5.49)$$

donde $\Lambda(N)$ es una matriz diagonal cuyo i-ésimo elemento de la diagonal es $\lambda(i)$, valor no negativo que constituye el *"peso"* a aplicar.

Volviendo a las definiciones (5.48) y a partir del concepto de *"peso"* llamaremos *matriz de covarianza pesada* a:

$$\Phi'_s(m)=\frac{1}{N}\mathbf{S}^{\mathrm{T}}(N)\Lambda(N)\mathbf{S}(N)$$

y (5.50)

$$\varphi'_s(N)=\frac{1}{N}\mathbf{S}^{\mathrm{T}}(N)\Lambda(N)\bar{\mathbf{s}}(N).$$

Estamos ahora en condiciones de redefinir la ecuación que nos permite encontrar la solución $\hat{\mathbf{a}}(N)$ buscada:

$$\Phi'_s(N)\hat{\mathbf{a}}(N)=\varphi'_s(N) \Rightarrow \hat{\mathbf{a}}(N)=[\Phi'_s(N)]^{-1}\varphi'_s(N). \qquad (5.51)$$

Llamaremos a esta técnica *método de la covarianza pesada Short-Term.* Esta técnica permite dar mayor significancia o descartar datos dependiendo de la elección del *peso* $\lambda(n)$. No obstante no existe una teoría general acerca de cuál es el *peso* más adecuado a utilizar para obtener una mayor calidad en la estimación de los coeficientes PL. La estrategia pesada ha sido utilizada para identificación adaptiva (Deller y Hsu, 1987) en algoritmos para deconvolución de formas de onda glotal y estimación de formantes.

5.2.4 Buscando soluciones óptimas

Siempre que se trabaje con el método de la autocorrelación o el de la covarianza, lo que se busca es la solución a un problema de vectores y matrices de la forma **Ax=b,** donde **A** es una matriz cuadrada y **x** es el vector solución. En principio numerosas técnicas clásicas pueden ser utilizadas, las cuales figuran en libros de álgebra lineal básica (Nobel, 1969; Golub y Van Loan, 1989).

Hemos notado que tanto el método de la covarianza como el de la autocorrelación producen matrices simétricas y que a su vez la matriz de la autocorrelación es Toeplitz adicionalmente. También es importante tener en cuenta la definición positiva de las matrices.

A partir del aprovechamiento de estas propiedades, trataremos de crear algoritmos eficientes para hallar la solución a los dos métodos, enfocando nuestra atención en la paralelización de un algoritmo que resuelve el problema de mínimos cuadrados, y en un conjunto de conceptos de cálculo numérico.

Nuestro objetivo final es obtener una especificación de la solución a uno de los métodos utilizando el lenguaje SIGNAL, presentado en los capítulos iniciales, como herramienta adecuada para modelizar este tipo de algoritmos sistólicos.

5.2.4.1 Solucion recursiva al problema de mínimos cuadrados pesado

La recursión de Levinson-Durbin[5] es un algoritmo recursivo para encontrar la solución a las ecuaciones de autocorrelación. Aquí presentaremos métodos para resolver las ecuaciones de la covarianza recursivamente en el tiempo. Estudiaremos dos algoritmos diferentes para calcular una solución *de mínimos cuadrados recursiva pesada* (MCRP) para los parámetros PL, la cual es equivalente a la solución de la *covarianza pesada.*

El motivo de la utilización de recursión en el tiempo, en el análisis de PL de speech es la necesidad de cambiar los parámetros estimados en el tiempo. La forma convencional MCRP no era usada en ingeniería de procesamiento de speech pero recurrieron a métodos para cambiar las estimaciones en el tiempo. El motivo de esto es que el MCRP tiene un orden de ejecución de $O(M^2)$ por muestra de una señal de voz cuando se ejecuta sobre una máquina secuencial. Aquí trataremos con métodos que tienen un orden de ejecución de $O(M)$,

[5] Levinson (1947) presentó una solución para **Ax=b** ,.teniendo los elementos de **b** una especial relación con los de la matrtiz **A** (Toeplitz,simétrica y positiva). Años más tarde, en 1960, Durbin elaboró una mejora a este algoritmo y es por ello que en ingeniería de procesamiento de voz se adjudica el algoritmo a ambos autores.

particularmente nos concentraremos en una versión sistólica del algoritmo MCRP, el cual corriendo sobre arquitecturas sistólicas y otros tipos de procesadores paralelos han hecho posible bajar el orden de ejecución $O(M^2)$ a escalas muchos más rápidas que "tiempo real".

MCRP convencional

Aquí ejecutaremos la recursión en el tiempo, incrementando la longitud de la ventana en cada iteración, es decir, la ventana comenzará en tiempo $n=1$ y finalizará en $n=N$. Si la recursión comienza en tiempo $n=1$, en cada instante $N \geq M$ (M es la cantidad de parámetros PL) producirá una estimación equivalente a la producida sobre la misma ventana con el método de la covarianza.

El algoritmo MCRP consiste de dos recursiones, una para la matriz de covarianza y otra para el vector de parámetros PL.

Existen varias formas de calcular la matriz de covarianza $\Phi'_s(N)$; tomaremos aquella que surge de la siguiente ecuación:

$$\Phi'_s(N) = \frac{1}{N} \sum_{n=1}^{N} \lambda(n) \mathbf{s}(n) \mathbf{s}^T(n). \qquad (5.52)$$

Ahora si multiplicamos por N y sacamos el último término de la sumatoria resulta en:

$$N\Phi'_s(N) = (N-1)\Phi'_s(N-1) + \lambda(N) \mathbf{s}(N) \mathbf{s}^T(N). \qquad (5.53)$$

Sea:

$$\Psi_s(N) \overset{def}{=} \left[N\Phi'_s(N)\right]^{-1}, \qquad (5.54)$$

luego tenemos

$$\Psi_s(N) = \left[(N-1)\Phi'_s(N-1) + \lambda(N) \mathbf{s}(N) \mathbf{s}^T(N)\right]^{-1}. \qquad (5.55)$$

Utilizando el resultado del lema *de inversión de matrices* (lema de Woodbury) que dice:

-Dadas **A** y **C**, *matrices cuya inversa existe, y una matriz* **B**, *tal que* $\mathbf{BCB}^T$ *tiene la misma dimensión que* **A**, *entonces:*

$$\left[\mathbf{A}+\mathbf{BCB}^{\mathrm{T}}\right]^{-1}=\mathbf{A}^{-1}-\mathbf{A}^{-1}\mathbf{B}\left[\mathbf{B}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{B}+\mathbf{C}^{-1}\right]^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{A}^{-1}, \tag{5.56}$$

(5.55) puede ser escrita de la forma (reemplazando N por n) :

$$\Psi_s(n)=\Psi_s(n-1)-\lambda(n)\frac{\Psi_s(n-1)\mathbf{s}(n)\mathbf{s}^T(n)\Psi_s(n-1)}{1+\lambda(n)\mathbf{s}^T(n)\Psi_s(n-1)\mathbf{s}(n)}. \tag{5.57}$$

Esta ecuación es utilizada para actualizar recursivamente la inversa de la matriz de covarianza pesada.

La segunda recursión actualiza los estimadores $\hat{\mathbf{a}}(n)$ a partir de la inversa $\Psi_s(n)$, una vez que ésta es encontrada:

$$\hat{\mathbf{a}}(n)=\hat{\mathbf{a}}(n-1)+\lambda(n)\Psi_s(n)\mathbf{s}(n)\varepsilon(n;n-1), \tag{5.58}$$

donde $\varepsilon(n;n-1)$ es el error de predicción en el instante n basado en el filtro diseñado sobre la ventana terminando en $n-1$.

La derivación de la ecuación antes presentada es la siguiente: el vector de la covarianza auxiliar puede ser escrito como:

$$\varphi'_s(N)=\frac{1}{N}\sum_{n=1}^{N}\lambda(n)\mathbf{s}(n)s(n)=\frac{N-1}{N}\varphi'_s(N-1)+\frac{\lambda(N)}{N}\mathbf{s}(N)s(N). \tag{5.59}$$

Si lo combinamos con (5.51) del método de la covarianza pesada queda

$$\Phi'_s(N)\hat{\mathbf{a}}(N)=\frac{N-1}{N}\Phi'_s(N-1)\hat{\mathbf{a}}(N-1)+\frac{\lambda(N)}{N}\mathbf{s}(N)s(N). \tag{5.60}$$

Ahora si sumamos y restamos el vector columna

$$\left[\lambda(N)/N\right]\mathbf{s}(N)\mathbf{s}^T(N)\hat{\mathbf{a}}(N-1) \tag{5.61}$$

obtenemos:

$$\begin{aligned}\Phi'_s(N)\hat{\mathbf{a}}(N)&=\frac{N-1}{N}\Phi'_s(N-1)\hat{\mathbf{a}}(N-1)+\frac{\lambda(N)}{N}\mathbf{s}(N)s(N)+\\&+\frac{\lambda(N)}{N}\mathbf{s}(N)\mathbf{s}^T(N)\hat{\mathbf{a}}(N-1)-\frac{\lambda(N)}{N}\mathbf{s}(N)\mathbf{s}^T(N)\hat{\mathbf{a}}(N-1)\\&=\frac{N-1}{N}\Phi'_s(N-1)\hat{\mathbf{a}}(N-1)+\frac{\lambda(N)}{N}\mathbf{s}(N)\times\\&\times\left[s(N)-\mathbf{s}^T(N)\hat{\mathbf{a}}(N-1)\right]+\frac{\lambda(N)}{N}\mathbf{s}(N)\mathbf{s}^T(N)\hat{\mathbf{a}}(N-1).\end{aligned} \tag{5.62}$$

Se puede reconocer al término entre corchetes como $\varepsilon(N; N-1)$ y el primero y el tercer término sumados como $\Phi'_s(N)$ de acuerdo a (5.53). Así tenemos

$$\Phi'_s(N)\hat{\mathbf{a}}(N) = \Phi'_s(N)\hat{\mathbf{a}}(N-1) + \frac{\lambda(N)}{N}\mathbf{s}(N)\varepsilon(N; N-1). \qquad (5.63)$$

Multiplicando a ambos lados por $[\Phi'_s(N)]^{-1}$ y reemplazando N por n se concluye con la ecuación cuya derivación queríamos obtener, la cual constituye la segunda recursión del algoritmo MCRP.

Antes de presentar el algoritmo deberemos tener en cuenta algunas consideraciones acerca de la inicialización del mismo. Para $n=1$ la matriz $\Psi_s(0)$ necesita ser calculada y esto es la inversa de la matriz nula. En la práctica lo que se hace es definir la matriz inicial como:

$$\Psi_s(0) = \frac{1}{\mu}\mathbf{I} \qquad (5.64)$$

donde $\mathbf{I}$ es la matriz identidad y μ un número pequeño, clásicamente 10^{-6}.

La inicialización de $\hat{\mathbf{a}}(0)$ es algo arbitrario (Albert y Gardner, 1967, pp. 109-115); sin embargo si comenzamos en el instante n_0 debería inicializarse $\hat{\mathbf{a}}(n_0)$ para la solución al método de la covarianza sobre el intervalo $[1,n_0]$. Esto no es utilizado en la práctica y comunmente es elegido $\hat{\mathbf{a}}(0)=0$.

Algoritmo MCRP

1. *Inicialización: Setear* $\hat{\mathbf{a}}(0)=0$ y $\Psi_s(0)$ de acuerdo a su definición precedente.
2. *Recursión:* **For** $n=1, 2, \ldots,$

   Actualizar $\Psi_s(n)$ *de acuerdo a la ecuación que la define.*

   Actualizar $\hat{\mathbf{a}}(n)$ *de acuerdo a la ecuación que la define.*

   Detenerse en un punto predeterminado de acuerdo a algún criterio.

   end.

Versión sistólica del algoritmo MCRP.

El método MCRP alternativo para resolver el problema de la covarianza deriva de una revisión del problema de mínimos cuadrados presentado en la sección 5.2.3. Recordemos que el problema es resolver el sistema de ecuaciones sobredeterminado (5.44) sujeto a criterio de error pesado. Aquí estableceremos el mismo problema de un modo levemente diferente: resolver el sistema sobredeterminado de N ecuaciones y M incógnitas,

$$\overline{\Lambda}^{1/2}(N)\begin{bmatrix}\mathbf{0}_{M\times M} & \\ \mathbf{s}^T(1) & \rightarrow \\ \mathbf{s}^T(2) & \rightarrow \\ \vdots & \\ \mathbf{s}^T(N) & \rightarrow\end{bmatrix}\hat{\mathbf{a}}(N)=\overline{\Lambda}^{1/2}(N)\begin{bmatrix}\mathbf{0}_{M\times 1}\\ s(1)\\ s(2)\\ \vdots\\ s(N)\end{bmatrix} \qquad (5.65)$$

sujeto a la restricción de error pesado donde $\overline{\Lambda}^{1/2}(N)$ es la matriz diagonal:

$$\overline{\Lambda}^{1/2}(N)\overset{def}{=}\begin{bmatrix}\mathbf{I}_{M\times M} & \mathbf{0}\\ \mathbf{0} & \Lambda^{1/2}(N)\end{bmatrix} \qquad (5.66)$$

con $\mathbf{I}_{M\times M}$ una matriz identidad de $M\times M$ y $\Lambda^{1/2}(N)$ la matriz diagonal con i-ésimo elemento de la diagonal $\sqrt{\lambda(i)}$, y $\{\lambda(n)\}$ algún conjunto de pesos no negativos. Hasta aquí lo único que hemos hecho es mover los pesos fuera de los criterios de error y modificado las ecuaciones para compensar los pesos "perdidos". Otro pequeño cambio con respecto a la declaración original del problema es que hemos adicionado M "ecuaciones nulas" de la forma $\mathbf{0}^T\hat{\mathbf{a}}(N)=0$ en el tope del sistema. Arbitrariamente, hemos asignado pesos unitarios a estas ecuaciones. La forma matricial de este sistema es:

$$\overline{\Lambda}^{1/2}(N)\overline{\mathbf{S}}(N)\hat{\mathbf{a}}(N)=\overline{\Lambda}^{1/2}(N)\overline{\mathbf{s}}(N), \qquad (5.67)$$

donde la única diferencia entre $\overline{\mathbf{S}}(N)$ y $\mathbf{S}(N)$, $\overline{\mathbf{s}}(N)$ y $\mathbf{s}(N)$ son los ceros al tope.

Considerando (5.44) y (5.45) la solución al presente sistema de ecuaciones es:

$$\frac{1}{N}\bar{\mathbf{S}}^T(N)\left[\bar{\Lambda}^{1/2}(N)\right]^T\bar{\Lambda}^{1/2}(N)\bar{\mathbf{S}}(N)\hat{\mathbf{a}}(N)=\frac{1}{N}\bar{\mathbf{S}}^T(N)\left[\bar{\Lambda}^{1/2}(N)\right]^T\bar{\Lambda}^{1/2}(N)\bar{\bar{\mathbf{s}}}(N). \tag{5.68}$$

Este resultado es equivalente al obtenido en (5.45), considerando que las ecuaciones nulas no tienen efecto sobre la solución.

Hasta aquí para calcular la solución $\hat{\mathbf{a}}(N)$ hemos formado siempre, de alguna u otra manera, la matriz de covarianza $\Phi_s(N)$. Nuestra propuesta ahora es encontrar la solución sin calcular directamente dicha matriz; para ello trabajaremos sobre el sistema de ecuaciones (5.65) o (5.67) utilizando técnicas de *triangularización ortogonal* de la matriz de "coeficientes" $\bar{\mathbf{S}}(N)$ basadas en los trabajos originales de Givens (1958) y Householder (1958). La aplicabilidad de esta técnica al problema de mínimos cuadrados fue presentada por primera vez por publicaciones de Householder, mientras que los primeros algoritmos numéricos fueron realizados por Golub y Businger (1965a, 1965b).

Enfocaremos nuestra atención en la metodología de Givens para desarrollar nuestro algoritmo [6]. Dicha técnica para transformar una matriz general en triangular mediante el cálculo de rotaciones unitarias en el plano es presentada en detalle en el ***Apéndice.***

Luego de aplicar las rotaciones de Givens, el sistema transformado obtenido es el siguiente:

$$\begin{bmatrix}\mathbf{T}(N)\\ \mathbf{0}_{N\times M}\end{bmatrix}\hat{\mathbf{a}}(N)=\begin{bmatrix}\mathbf{d}_1(N)\\ \mathbf{d}_2(N)\end{bmatrix}, \tag{5.69}$$

donde $\mathbf{T}(N)$ es una matriz triangular de $M\times M$. El sistema:

$$\mathbf{T}(N)\hat{\mathbf{a}}(N)=\mathbf{d}_1(N), \tag{5.70}$$

puede ser resuelto usando un algoritmo de sustitución *backward* para obtener los estimadores de mínimos cuadrados $\hat{\mathbf{a}}(N)$.

De aquí en más mostraremos este método en un modo recursivo donde cada ecuación es utilizada para obtener una estimación actualizada de los parámetros PL.

Si un conjunto de matrices de rotación de plano es utilizado para eliminar filas de $\bar{\mathbf{S}}(N)$ desde la izquierda hacia la derecha, comenzando con la fila $M+1$, es decir la primera ecuación debajo de los ceros, cada fila requiere de una secuencia de M rotaciones de plano ortonormales. Si denotamos al producto de las M matrices de rotación de plano aplicado a la ecuación n (fila $M+n$) por $\mathcal{R}_n$, la transformación del sistema de ecuaciones es de la forma:

$$\mathcal{R}_n\mathcal{R}_{n-1}\ldots\mathcal{R}_1\bar{\mathbf{S}}(N)\hat{\mathbf{a}}(N)=\mathcal{R}_n\mathcal{R}_{n-1}\ldots\mathcal{R}_1\bar{\bar{\mathbf{s}}}(N). \qquad (5.71)$$

Definamos como matriz de la transformación global a:

$$\prod_N \overset{def}{=} \mathcal{R}_n\mathcal{R}_{n-1}\ldots\mathcal{R}_1. \qquad (5.72)$$

Luego de esta definición el sistema antes mencionado tiene la misma forma que el de (5.69) .

Para ir hacia una solución recursiva adicionaremos al sistema una nueva ecuación dejando por el momento las matrices de rotación de Givens. Supongamos que deseamos encontrar el mismo sistema de N ecuaciones más una nueva representando el tiempo $N+1$,

$$\begin{bmatrix}\bar{\Lambda}^{1/2}(N)\bar{\mathbf{S}}(N)\\ \sqrt{\lambda(N+1)}\mathbf{s}^T(N+1)\end{bmatrix}\hat{\mathbf{a}}(N+1)=\begin{bmatrix}\bar{\Lambda}^{1/2}(N)\bar{\bar{\mathbf{s}}}(N)\\ \sqrt{\lambda(N+1)}s(N+1)\end{bmatrix} \qquad (5.73)$$

o

$$\bar{\Lambda}^{1/2}(N+1)\bar{\mathbf{S}}(N+1)\hat{\mathbf{a}}(N+1)=\bar{\Lambda}^{1/2}(N+1)\bar{\bar{\mathbf{s}}}(N+1). \qquad (5.74)$$

Estamos ahora con una versión de $N+1$ ecuaciones del mismo problema. Aplicando una vez más el concepto de transformaciones de Givens, esta vez denotando a la matriz que opera sobre la fila n como Q_i , se tiene que si las $\mathcal{R}_i$ eran de dimensión $(M+N)\times(M+N)$ en un problema de N ecuaciones entonces las Q_i serán de dimension $(M+N+1)\times(M+N+1)$ considerando que el problema es ahora de $N+1$ ecuaciones. A través del siguiente lema[6] podremos distinguir que las operaciones hechas sobre las primeras N ecuaciones en el problema de $N+1$

[6] La demostración de este lema surge de la definición y no es de nuestro interés considerarla aquí.

ecuaciones son idénticas a aquellas practicadas sobre el problema de N ecuaciones.

Lema: $$Q_N Q_{N-1} \cdots Q_1 = \begin{bmatrix} \prod_N & \mathbf{0}_{N\times 1} \\ \mathbf{0}^T{}_{N\times 1} & \mathbf{1} \end{bmatrix}, \qquad (5.75)$$

Aplicando este lema a las $N+1$ ecuaciones de (5.73) está visto que luego de N transformaciones de filas en el problema de $N+1$ ecuaciones, el sistema obtenido es el siguiente:

$$\begin{bmatrix} \prod_N \bar{\Lambda}^{1/2}(N)\bar{\mathbf{S}}(N) \\ \sqrt{\lambda(N+1)}\mathbf{s}^T(N+1) \end{bmatrix} \hat{\mathbf{a}}(N+1) = \begin{bmatrix} \prod_N \bar{\Lambda}^{1/2}(N)\bar{\bar{\mathbf{s}}}(N) \\ \sqrt{\lambda(N+1)}s(N+1) \end{bmatrix} \qquad (5.76)$$

las primeras M filas son exactamente el sistema representado en 5160 con la única diferencia que ahora se le introdujo el concepto de "*peso*".

Note que para resolver el problema de las $N+1$ ecuaciones es suficiente aplicar las transformaciones $Q_{N+1}\mathcal{R}_N\,\mathcal{R}_{N-1}\ldots\mathcal{R}_1$ a las filas con la restricción de que las $\mathcal{R}_i$ sólo operen sobre las N ecuaciones de $\bar{\Lambda}^{1/2}(N+1)\bar{\mathbf{S}}(N+1)$. Por lo tanto podríamos definir un conjunto $\{\mathcal{S}_i\}$, $i=1, \ldots, N+1$ donde cada elemento es una matriz de dimensión $(M+n)\times(M+n)$ y tal que $\mathcal{S}_{N+1}= Q_{N+1}$, $\mathcal{S}_j= \mathcal{R}_j$ con $j=1, \ldots, N$ y $\mathcal{S}_{N+1}\mathcal{S}_N\ldots\ \mathcal{S}_1$ la productoria de las matrices de transformación aplicadas a las $N+1$ ecuaciones del problema en cuestión, notando así que resolver el problema de $N+1$ ecuaciones incluye la resolución del problema de N ecuaciones. Concluímos así en una solución recursiva para hallar los $\hat{\mathbf{a}}(N)$.

Si consideramos la aplicación de $\mathcal{S}_{N+1}$ al sistema de (5.76), encontraremos que las filas y columnas desde la $M+1$ hasta N sirven para preservar las filas de ceros (causadas por la eliminación de ecuaciones desde 1 hasta N) en la matriz de la izquierda (matriz de coeficientes) y en el vector $\mathbf{d}_2(N)$. Desde que estas filas no cumplen ninguna función en la solución pueden ser eliminadas junto con las filas y columnas de $\mathcal{S}_{N+1}$. Por lo tanto cada $\mathcal{S}_i$ es así una matriz de dimensión $(M+1)\times(M+1)$.

Algoritmo MCRP "sistólico"

Por varios años se han utilizado recursiones basadas en Rotaciones de Givens (RG) en procesamiento de señales de voz sobre máquinas de propósito general en un modo monoprocesador (no paralelas) [23-26]. El algoritmo que presentamos a continuación puede ser utilizado para calcular un estimador recursivo temporal de parámetros PL de speech. Un parámetro actualizado muestra a muestra es muy útil en aplicaciones de speech [25-26][14]. El método RG produce una solución terminal la cual es teóricamente equivalente a la solución de la covarianza [14-15] sobre el mismo bloque de datos.

La solución batch (usando todas las ecuaciones a la vez) al problema de mínimos cuadrados de un sistema de la forma (5.65) mediante el empleo de transformaciones de Givens puede ser encontrado en [13 y 16]. Un algoritmo recursivo requiriendo $(M+1)^2$ espacios de memoria es dado en [17].

El algoritmo básicamente es el siguiente:

Dedicar $(M+1)\times M$ celdas de memoria para los elementos de la matriz sobre el lado izquierdo de la ecuación (5.73), y $(M+1)\times 1$para el vector auxiliar sobre el lado derecho. Concatenar ambas estructuras en una matriz de trabajo $\mathbf{W}$ *de dimensión $(M+1)\times (M+1)$.* $\mathbf{W}(n)$ *denotará la matriz de trabajo en el instante n (incluyendo la n-ésima ecuación), antes de la aplicación de* S_n; *y* $\mathbf{W}'(n)$ *a la matriz de post-rotación.*

1. *Inicialización:*

$$\mathbf{W}(0)=\mathbf{W}'(0)=\left[\mathbf{0}_{(M+1)\times M}\middle|\mathbf{0}_{(M+1)\times 1}\right].$$

2. *Recursión:* **For** $n = 1, 2, ...,$

- *Reemplazar la última fila de* $\mathbf{W}'(n-1)$ *por* $\left[\sqrt{\lambda(n)}\mathbf{s}^T(n)\middle|\sqrt{\lambda(n)}s(n)\right]$, *donde* $\lambda(n)$ *denota el n-ésimo peso, para formar* $\mathbf{W}'(n)$.
- *"Aplicar* S_n*" Esto requiere alguna operaciones escalares:*

  For $j=1, 2, ..., M$

  $\theta = \mathbf{arctan}[\mathbf{W}(M+1, j; n) / \mathbf{W}(j, j; n)]$

$\mathbf{W}'(M+1, j; n) = 0$

For $k = j, j+1, ..., M+1$

$\mathbf{W}'(j, k; n) = \mathbf{W}(j, k; n)\cos\theta + \mathbf{W}(M+1, k; n)\,sen\theta$

For $k = j+1, j+2, ..., M+1$

$\mathbf{W}'(M+1, k; n) = -\mathbf{W}(j, k; n)\,sen\theta + \mathbf{W}(M+1, k; n)\cos\theta$

3. *Las primeras M filas y columnas de* $\mathbf{W}'(n)$ *contienen* $\mathbf{T}(n)$, *y las primeras M filas de la última columna son* $\mathbf{d}_1(n)$.
4. *Parar en algún punto predeterminado o de acuerdo a algún criterio.*

El método MCRP involucra un manejo flexible de la información lo cual permite una fácil exclusión de los datos, ventaneo, y minimización del error predecido en forma backward.

Este método es simple de programar, no sufre inicialización ni cálculo de la inversa de una matriz.

Es dificil compararlo con las soluciones convencionales al método de la covarianza usadas en procesamiento de speech, por haber asumido la necesidad de un "peso". En este caso "pesado", la solución convencional requiere un orden de ejecución de $O(M^3)$ u operaciones por punto, mientra las rotaciones de Givens requieren sólo un orden de $O(M^2)$. No obstante los cálculos de la covarianza no pesada pueden ser mucho más eficientes, es decir, de orden O(M). Esta ventaja es discutida si el proceso de RG es implementado sobre un procesador rápido utilizando operaciones paralelas.

La motivación primaria que hace interesantes estas soluciones es la disponibilidad de arquitecturas de cómputo paralelo que permiten ejecutar el método MCRP basado en rotaciones de Givens en procesos de O(M). Más detalles de cómo cálculo paralelo puede ser aplicado a este método es encontrado en los trabajos de Gentleman y Kung (1981); McWhirter (1983); y Deller, Odeh y Luk (1991, 1989, 1989).

Capítulo 6

Algoritmos en SIGNAL

6.1 Introducción

Después de esta reseña sobre procesamiento de señales presentada en los capítulos previos, ahora nuestro objetivo es mostrar la aplicabilidad del lenguaje SIGNAL para la especificación de las rutinas clásicas mencionadas. Luego concluiremos con la elaboración de un algoritmo para el cálculo de coeficientes de predicción lineal tratando de paralelizar su implementación.

6.2 Filtro transversal

En las secciones 3.2.1.6 y 3.2.1.7 fueron presentados los operadores de SIGNAL para la manipulación de ventanas deslizantes y arreglos de procesos, cuya aplicación es inmediata cuando se trata de construir un filtro digital donde el número de coeficientes serán pasados como parámetros y cuya salida *s* se obtiene de la entrada *e* de la siguiente manera:

$$s(t) = \sum_{i=0}^{N-1} k_i e(N-i+1) \qquad (6.1)$$

considerando N coeficientes k_i

Su implementación en SIGNAL sería la siguiente:

```
process filtro ( integer N; [N] real k )={? real e
                                          ! real s}
    (|ventanae := e window N
     |array I to N of
        calculo := calculo + ventanae[N-I+1]
        with calculo[0]:cero
      end
     |s:= calculo [N]
     |cero:= 0.0
```

```
  |)
where
  [N]real calculo, ventanae init ({to N-1}: 0.0)
end
```

Gráficamente, el proceso "filtro" especifica lo siguiente:

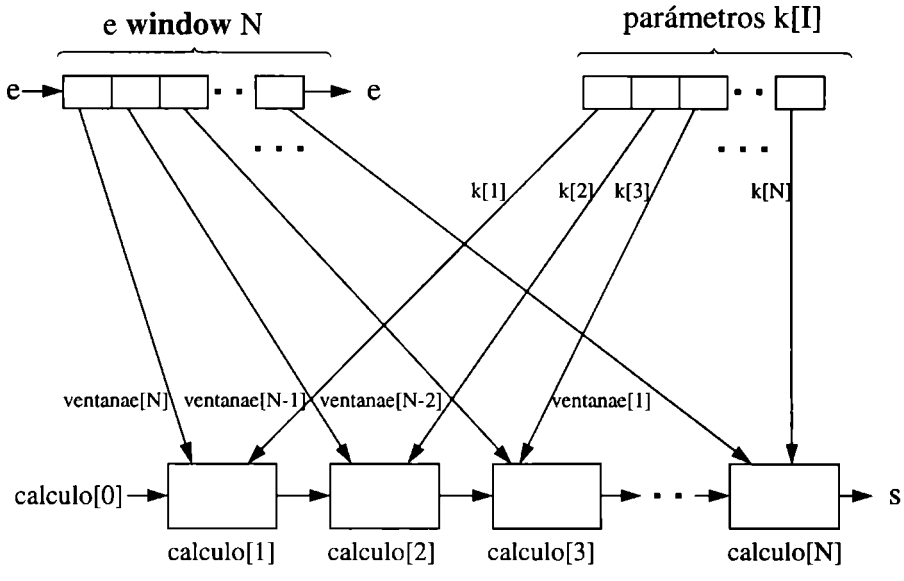


Fig 6.1 Comportamineto del proceso filtro.

Es claro que el comportamiento del constructor **array** con la cláusula **with** (que involucra a la señal "cálculo") es netamente sistólico y permite la paralelización del algoritmo como por ejemplo asignando 1 unidad de procesamiento para cada uno de los procesos construidos sobre el modelo del cuerpo del **array**.

Desde el punto de vista temporal la señal "s" es sincrónica con la señal "e", es decir, que sus relojes son coincidentes; ya que ninguno de los dos operadores, ni las operaciones aritméticas producen alteraciones en los relojes de las señales involucradas, como fue previamente explicado.

Sistema de ecuaciones de reloj:

1) P(ventanae) = P(e)

2) P(calculo) = P(ventanae)

3) P(s) = P(calculo)

4) P(s) = P(ventanae) de 2) y 3).

5) Finalmente P(s) = P(e) de 4) y 1).

6.3 Covarianza Short-Term de orden M

Este algoritmo calcula el vector de covarianza (órdenes de 0 a M, ecuación 5.34) descripto en la sección 5.2.2. Cuando se hizo mención al método de la covarianza, se destacó el uso de algunos puntos de datos fuera del rango $n \in [m-N+1, m]$ al calcular $\varphi_s(\eta,v; m)$. La alternativa que plantea el algoritmo es evitar la situación anterior tomando los puntos de datos (en el frame) a partir de la muestra M+1-i para obtener la covarianza de orden i, calculando la función de covarianza para un frame de longitud N de la siguiente manera:

$$\varphi_s(0,i;N) = \frac{1}{N-M} \sum_{j=M+1}^{N} s(j)s(j-i+1) \qquad (6.2)$$

A partir de esta ecuación el siguiente proceso SIGNAL surge:

```
process covarianza ( integer M,N )= {? [N]real frame
                                     ! [M+1] real covar }
(|cero := 0 e 0 when (event frame)
(|array I to M+1 of
  (|synchro{frame,cálculo}
   |array J to N of
         cálculo:=(cálculo+(frame[J]*frame[J-I+1]))
                  when(J>M) default cálculo
    with cálculo[.]:cero
   end
   |covar:= cálculo[1]/(N-M)
   |)/cálculo
  end
 |)!! covar
```

```
where
   [N]real cálculo;
   real cero
end
```

Una interpretación gráfica del mismo es como sigue:

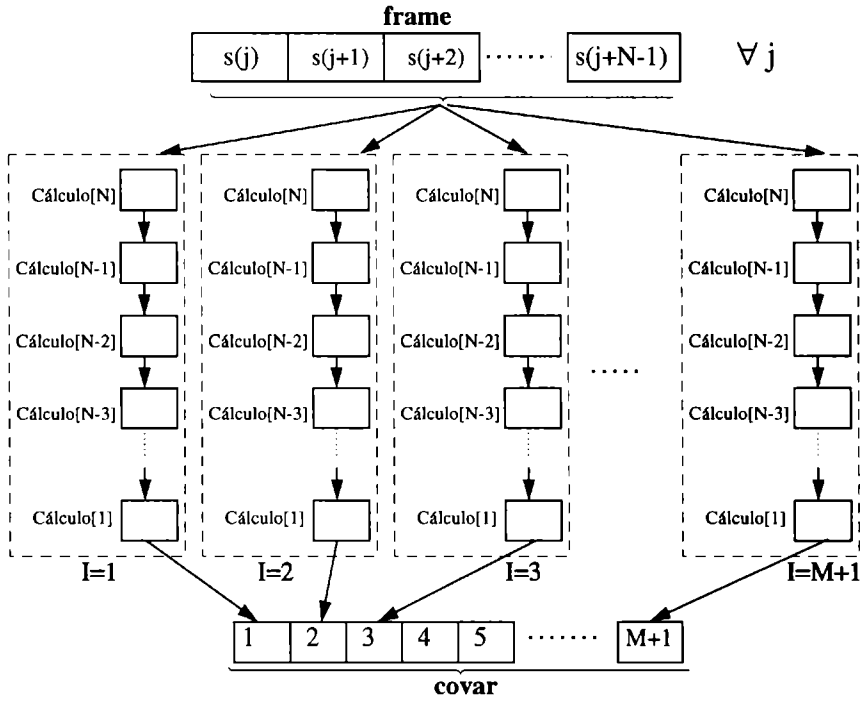


Fig 6. 2 Diagrama de proceso del cáculo de covarianza de orden M.

Como podemos ver cada una de las covarianzas (covar[i]) de orden i es dejada como resultado en la primera posición de la señal de tipo array "cálculo" a la salida del bucle interno. Esto se debe a la declaración en la cláusula **with**, la cual indica que la dependencia es decremental, esto es, cálculo[N+1] se inicializa con 0.0 (cero) y luego cálculo[i] depende de cálculo[i+1]. Veamos un ejemplo: si J varía en un rango [1, 4] y M=2 (independientemente del bucle exterior, pues sólo es utilizado para el acceso a la señal "frame"), entonces tendremos que:

cálculo[5]=0.0

cálculo[4]=cálculo[5] + t_1

cálculo[3]=cálculo[4] + t_2

cálculo[2]=cálculo[3]

cálculo[1]=cálculo[2]

por lo tanto cálculo[1] = 0.0 + t_1+ t_2, siendo t_1 y t_2 los términos que involucran referencias a "frame" y se ajustan a la sumatoria planteada inicialmente.

Sistema de ecuaciones de reloj:

1) P(cero)=P(frame)

2) P(frame)=P(cálculo)

3) P(covar)=P(cálculo)

4) Finalmente P(frame)=P(covar) por 2) y 3)

6.4 Cálculo de la medida de cruces por cero

Este proceso calcula el número de pasajes por cero (*nrocruces*) sobre una ventana de señal de voz (*frame*) de longitud N. Todas las señales son sincrónicas a *frame*. Este proceso efectúa una pasada simple desde 1 hasta N. La señal *cambiosigno* es incrementada a cada cambio de signo de un muestra respecto de su anterior, caso contrario mantiene el valor. La señal *nrocruces* recupera el valor final de *cambiosigno*.

Veamos su implementación SIGNAL:

```
process cruces=(integer N)(?[N] real frame
                           !real nrocruces)
(|synchro(frame,cambiosigno)
 |cero := 0e0 when (event frame)
 |array I to N of
  cambiosigno:=((cambiosigno+1)
              when(frame[I]>0e0 and frame[I-1]<0e0
               andI>1) or (frame[I]<0e0
               and frame[I-1]>0e0) and I>1)
                 default (cambiosigno)
  with cambiosigno [0]:cero
 end
```

```
|nrocruces:= cambiosigno[N]
|)!! nrocruces
where
   [N]real cambiosigno;
   real cero
end
```

Sistema de ecuaciones de reloj:

1) P(frame)=P(cambiosigno)

2) P(cero)=P(cambiosigno)

3) P(nrocruces)=P(cambiosino)

4) finalmente P(nrocruces)=P(frame) por 1) y 3)

La tasa de cruces por cero es una medida razonablemente buena de la presencia o no de speech no vocalizado, pues será mucho más alta en regiones no vocalizadas (indicando altas frecuencias)[ref Rabiner and Sambur].

El algoritmo "LPC10" para codificación de speech (Kang 1979 y Tremain 1982) utiliza la medida de cruces por ceros para tomar desiciones de vocalización.

6.5 Generación de segmentos homogéneos de señal

Buscando el desarrollo de un ambiente para el procesamiento de señales con determinadas características, es que se construyó un proceso que permite la generación de una señal caracterizada por parámetros especificados.

El proceso SIGNAL "Generación" utiliza un mecanismo de sobremuestreo para generar la señal de salida "*s*" a partir de tres señales de entrada (*a, longitudes* y *varianzas*) y de un proceso "Sobremuestreo" el cual recibe como entrada la señal *longitudes*. Si representamos con "hl" a la señal que denota la presencia de *longitudes*, y con "v" a su valor en un instante *t*, entonces se dice que el mecanismo de sobremuestreo permite crear (v-1) instantes adicionales entre dos presencia de "hl", obteniendo así un reloj más rápido que el de *longitudes*, el cual pertenecerá a la señal "contador" (salida de sobremuestreo).

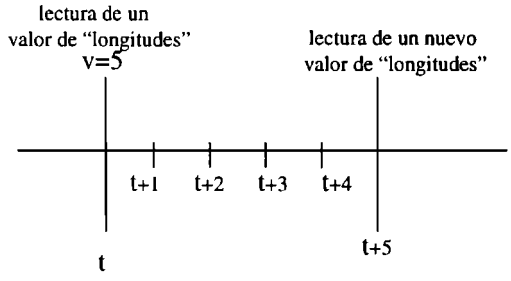


Fig 6. 3 Ejemplo de sobremuestreo con v=5.

El texto del proceso "Sobremuestreo" es el siguiente:

```
process sobremuestreo = { ? integer longitudes
                          ! integer contador}
(|synchro{longitudes,hl}
 |contador := longitudes default (zcontador-1)
 |zcontador := contador $1
 |hl := true when (zcontador=1)
 |)/zcontador,hl
 where
  logical hl;
  integer zcontador
end
```

Sistema de ecuaciones de reloj:

1) P(longitudes)=P(hl)

2) P(contador)=P(longitudes) ∪ P(zcontador)

3) P(zcontador)=P(contador)

4) P(hl)=T(zcontador)⊆P(zcontador)

5) entonces P(hl)⊆P(contador) por 3) y 4)

6) por lo tanto P(longitudes)⊆P(contador) por 1) y 5)

Con lo cual es posible verificar que el proceso se comporta de acuerdo a lo deseado.

La creación de la señal "s" se logra a través de la función "Random"[1] que genera números aleatorios siguiendo una ley Gaussiana (N(0, 1)). Su construcción puede corresponder a un proceso autorregresivo Gaussiano de orden p,

$$s_t = \sum_{i=1}^{p} a_i s_{t-i} + e_t \sigma \qquad \forall t;$$

La función "Random" genera una señal "ruido" (e_t); el valor final de "s" es una función de las señales "ruido", "a" (a_i) y "varianzas" (σ). Su reloj es el mismo que el de *contador*.

Los instantes de cambio de modelo son dados por la señal "longitudes". La implementación SIGNAL del proceso "Generación" es:

```
process generacion=(integer n1,n2){?integer longitudes
                                   real a, varianzas
                                  !real s}
(|synchro{longitudes, a, varianzas}
 |sobremuestreo{longitudes, contador}
 |varianzatop:= varianzas cell (event contador)
 |atop:= a cell (event contador)
 |s:= (atop*sretar)+(varianzatop*ruido)
 |ruido:= random{n1,n2}-0.5e0
 |sretar:= s $1
 |)!! s
 where
   real varianzatop, atop, ruido, sretar;
   function random ={?integer n1, n2 ! real n}
end
```

[1] Para más detalles de esta función véase [11]

Notemos que el algoritmo genera señales para un orden p=1. Por otro lado que cada muestra de la señal de salida es producida con los valores de varianza (varianzatop) y de coeificiente (atop) últimos ocurridos. Esto se debe que tanto "atop" como "varianzatop" toman el valor de "a" y "varianza" respectivamente o bien en el momento en que estas dos señales ocurren, o bien repiten (arrastran) los últimos valores recibidos. Vemos entonces que este comportamiento del proceso lo logramos utlizando el constructor **cell.**

Sistema de ecuaciones de reloj:

1) P(longitudes)=P(a)=P(varianzas)
2) P(longitudes)$\subseteq$P(contador por el resultado del sistema de ecuaciones del proceso "sobremuestreo" descripto anteriormente.
3) P(varianzatop)=P(varianzas)$\cup$P(contador)
4) P(atop)= P(a)$\cup$P(contador)
5) P(s)=P(varianzatop)=P(atop)=P(sretar)
6) P(sretar)=P(s)
7) P(s)$\supseteq$P(varianza)=P(a)=P(longitudes) Por 1),3),4),5), con lo cual la salida está en función de las señales de entrada.

6.6 Algoritmo MCRP

Este proceso implementa el algoritmo MCRP explicado en la sección 5.2.4.1 "versión sistolica del algoritmo MCRP". El mismo calcula una matriz, $\mathbf{W}'(n)$, de dimensiones $(M+1)\times(M+1)$ donde las primeras M columnas y filas de dicha matriz representan la matriz $\mathbf{T}(n)$ y las primeras M filas de la última columna muestran al vector columna $\boldsymbol{d}_1(n)$. Tanto $\mathbf{T}(n)$ como $\boldsymbol{d}_1(n)$ fueron debidamente definidos en la sección antes mencionada.

A continuación mostraremos dos diferentes implementaciones de este algoritmo.

6.6.1 Implementación secuencial de MCRP

Este proceso SIGNAL calcula la matriz $W'(n)$ en forma secuencial para cada entrada de una muestra de la señal "s". Dicha matriz es tratada como un arreglo unidimensinal, lo cual gráficamente sería:

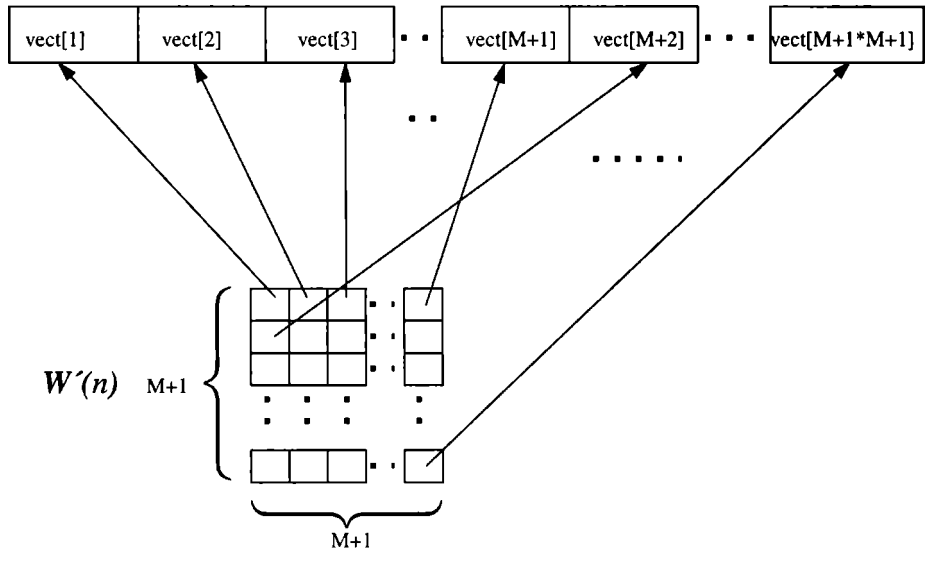


Fig 6.4. Reperesentacion vectorial de $W'(n)$.

El proceso SIGNAL correspondiente es:

```
process matrizw =(integer M, N){?dpreal s
                                  ![M*M]dpreal wprima}
(|wauxiliar:= calmat(M){s}%1%
 |wprima:=  wauxilar when fin%2%
 |relwauxiliar:= #(event wauxiliar)%3%
 |fin:= ((relwauxiliar-(N*(relwauxiliar/N)))=0)%4%
 |)
 where
   integer relauxiliar;
   logical fin;
   [M*M]dpreal wauxiliar
```

```
process calmat =(integer M)(?dpreal s
                                !![M*M]dpreal wsali)
(|retardow:= retardowsali cell (event veces)%5%
 |retardowsali:= wsali $1%6%
 |ventanas:= s window M %7%
 |histventanas:= ventanas cell(event veces)%8%
 |veces:= 1when(comienzo) default retardoveces+1
                                                  %9%
 |retardoveces:= veces $1 %10%
 |comienzo:= (retardoveces=(M-1))%11%
 |tita:=atan2(histventanas[M-veces],
             retardow[((veces-1)*M)+veces])%12%
 |wprimp:= matwprim $1 %13%
 |synchro(matwprim, veces, retardow) %14%
 |(matwprim):= rotgivens(M)(retardow, wprimp,
                     tita, veces, histventanas)%15%
 |fin:=(retardoveces=(M-2))%16%
 |synchro(s, when comienzo)%17%
 |wsali:= matwprim when fin %18%
 |)
 where
   logical fin;
   [M*M]dpreal retardow, matwprim;
   [M*M]dpreal wprimp init [(to((M*M)-1)):0.0d0];
   [M*M] dpreal retardowsali init [ (to((M*M)-
                                    1)):0.0d0];
   [M]dpreal retardos;
   [M]dpreal ventanas init[(to(M-1)):0.0001d0];
   dpreal tita;
   integer veces, retardoveces init (M-1)
```

```
process rotgivens = (integer M) {?[M*M]dpreal retarw,
                                                w2primp;
                                    dpreal tita1;
                                    integer j;
                                    [M]dpreal ventas
                                    ![M*M]dpreal matwp}
      (|coseno:= cos{tita1}%19%
       |seno:= sin{tita1}%20%
       |array I to (M*M)of
         (|matw:=    0.0d0
             when(I=((M*(M-1))+j))default
                     ((retarw[I]*coseno)+
                     (ventas[M-(I-(M*(I/M)))]*seno))
              when(I<=(j*M))and(I>=((j-1)*M)+j)default
                      ((-1)*retarw[I-((M-j)*M)]*seno)+
                   (ventas[(M-(I-N(M*(I/M))))]*coseno)
              when(I>=((M*(M-1))+j+1)default
                      (w2primp[I])%21%
          |)with matw
        end
       |)
       where
         dpreal coseno, seno
         function sin{?dpreal t !dpreal x};
         function cos{?dpreal t !dpreal y}
      end; (rotgivens)
      function atan2{?dpreal t, x !dpreal y}
   end (calmat)
end (matrizw)
```

Este proceso está compuesto por 2 subprocesos; *calmat* y *rotgivens*. El segundo realiza una rotación de Givens colocando un cero(0) en la posición (M+1,j) y ajustando las correspondientes filas rotadas (siempre manejando los índices de acuerdo a la forma en que fue construida la matriz; línea %21%).Toda esta rotación es efectuada en paralelo con el uso del constructor **`Array`** . El subproceso *calmat* es el que invocando sucesivas veces a *rotgivens* logra aniquilar una fila de la matriz (o sea realiza una aplicación de $\mathcal{R}_i$ como en (5.71)). Para realizar estas sucesivas llamadas se utiliza una señal *veces* (linea %9%) que es pasada como parámetro a *rotgivens* para indicar cual es la columna (de la última fila) en la que se está colocando un cero(0). Esta señal se inicializa ante la llegada de una nueva muestra y se incrementa M-1 veces, en este punto se obtiene la salida del proceso *calmat* que es la matriz *wsali* (linea %18%) y ocurre la "llegada" de una nueva muestra (linea %19%). Es importante notar que durante las rotaciones dentro de *calmat* la matriz utilizada para generar los parámetros de la rotación es siempre la misma y no la que se obtiene luego de una rotación, es decir para aniquilar una fila se utiliza la matriz *retardow* durante todo el proceso *calmat*, de la misma manera la última fila de la matriz (M+1) es simulada por una señal de retardo de longitud M (la señal *ventanas*, la cual es mantenida durante todas las rotaciones como *histventanas* (línea%8%), con la utilización del constructor **`cell`**). Por cada matriz retornada por el proceso *calmat* una nueva muestra ingresa y dependiendo de la cantidad de muestras que han ingresado, esto es, una vez que se ha llegado a N muestras se toma la salida de *calmat* como salida del proceso principal *matrizw* (línea %2%) ,caso contrario esta matriz (*wsali)* es asignada a *retardow* (líneas %5%, %6%) para ser utilizada en la rotacion futura ante el arrivo de la nueva muestra. Es importante destacar que el parámetro M que recibe el proceso *matrizw* es el valor M+1 de todos los desarrollos comentados en el Capítulo 5. Gráficamente, el procesamiento de *calmat* puede verse como:

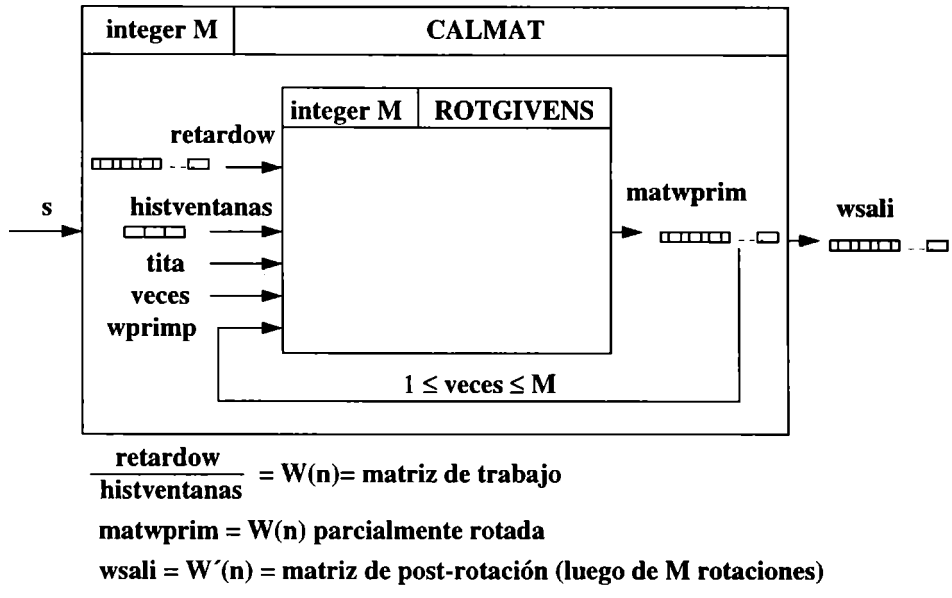
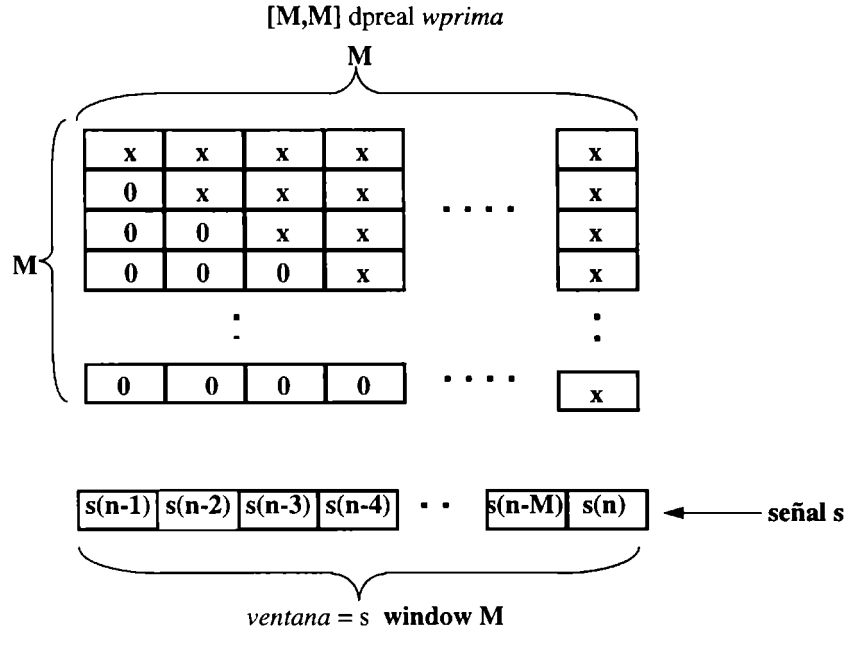


Fig 6. 5 Esquematización de los procesos *calmat* y *rotgivens.*

6.6.2 Implementación paralela de MCRP

Este proceso calcula también la matriz $W'(n)$, para cada muestra de la señal "s". Aquí se realizan M rotaciones en paralelo poniendo ceros en la fila M+1.

Dado que en la sección 5.2.4.1 justificamos la eliminación de ciertas filas del sistema lineal de ecuaciones original, nos manejamos con la información estructurada de la siguiente manera:



La fila M+1 en realidad está representada por la señal "ventaneada" *s* y la matriz mostrada arriba de dimensión M (M+1, para el algoritmo original) corresponde a la salida del proceso *Givpar*, que a continuación se detalla:

```
process givpar=(integer M, N) { ? dpreal s
                                ! [M, M] dpreal wprima}

(| ventana := s window M%1%
 | ns:= #(event s)%2%
 | aux := givens(M){ventana,zaux} %3%
 | synchro{s, aux} %4%
 | zaux := (aux $1)%5%
 | wprima:= asignar(M){ aux}when((ns-(N*(ns/N)))=0)%6%
 |)
 where
     [M, M] dpreal aux,zaux init[{to M,to M}:0.0 d 0];
     [M] dpreal ventana init [{to (M-1)}:0.0001 d 0];
     integer ns
     process asignar =(integer M){? [M, M] dpreal resu
                                  ! [M, M] dpreal salida}
     (| array I to M of
         array J to M of
            salida:= resu[I,J]%7%
         end
        end
      |)
      end;
     process givens = ( integer M) { ? [M] dpreal ws;
                                       [M,M] dpreal w
                                   ! [M, M] dpreal wp}
     (| array I to M-1 of
          tita:= atan2 { ws[M-I],w[I,I]}%8%
        end
      | array I to M-1 of
          (|c:=cos{tita[I]}%9%
           |s:=sin{tita[I]}%10%
           |)
        end
      | array I to M of
         array J to M of
          wp:= 0.0 d 0
                  when((I=M)and(J<M)) default
                  ((w[I,J]*c[I])+(ws[M-J]*s[I])) when
                  ((J>=I)and(I<M)) default
          (((-1)*(w[I,J]*s[I-1]))+(ws[M-J]*c[I-1]))when
```

```
                    ((I=M)and(J=M)) default
              w[I,J] %11%
          end
         end
        |)
        where
         [M-1] dpreal tita,c,s
         function cos = { ? dpreal t ! dpreal x};
         function sin = { ? dpreal t ! dpreal y};
         function atan2 = {? dpreal a,b ! dpreal c}
       end
end
```

El proceso *Givpar* está compuesto por dos subprocesos (los cuales son considerados macros): 1- *Givens*

2-*Asignar*

La función del primero es la siguiente: dada una matriz de trabajo *w* (*w)* en cualquier instante y ante una nueva muestra de la señal *s* realizar las rotaciones de Givens correspondientes para eliminar los elementos de la fila M+1 (representada por *ws*, ventaneo de *s*, línea %1%), dejando los resultados en *w´ (wp).* Para ello calcula en paralelo los ángulos para cada una de las rotaciones, dejando los resultados sobre *tita* que es un arreglo de tamaño M (línea %8%).

Al mismo tiempo se obtienen los cosenos y senos involucrados sobre los arreglos *c* y *s* (líneas %9,10%). Simultáneamente con estos resultados se va actualizando la matriz resultante ,*wp* , (línea %11%) eliminando los elementos de la fila M+1 según el criterio de triangularización de matrices planteado por Givens, el cual se explica en el ***Apéndice***.

Estos cálculos sobre la matriz *wp* y la obtención de ángulos, senos y cosenos en paralelo es posible mediante la utilización combinada del constructor **array** y de la composición de procesos elementales (|).

El proceso *Givpar* calcula por cada muestra de *s* una matriz *w (aux)* invocando a *Givens* (línea %3%) la cual se comporta como salida de dicho proceso sólo si ya ha recibido un frame de *s*. Esto se realiza asignando *aux* a *wprima* (a través del proceso *asignar,* línea %6%,) cuando la cantidad de muestras recibidas *ns,* es igual a N (tamaño del frame) o un múltiplo de la misma.

Para lograr que por cada *s* se produzca una matriz *aux* en forma simultánea, hacemos hacemos que las variables mencionadas sean sincrónicas a través de la sentencia **synchro** (línea %4%).

Una representación gráfica de este proceso sería:

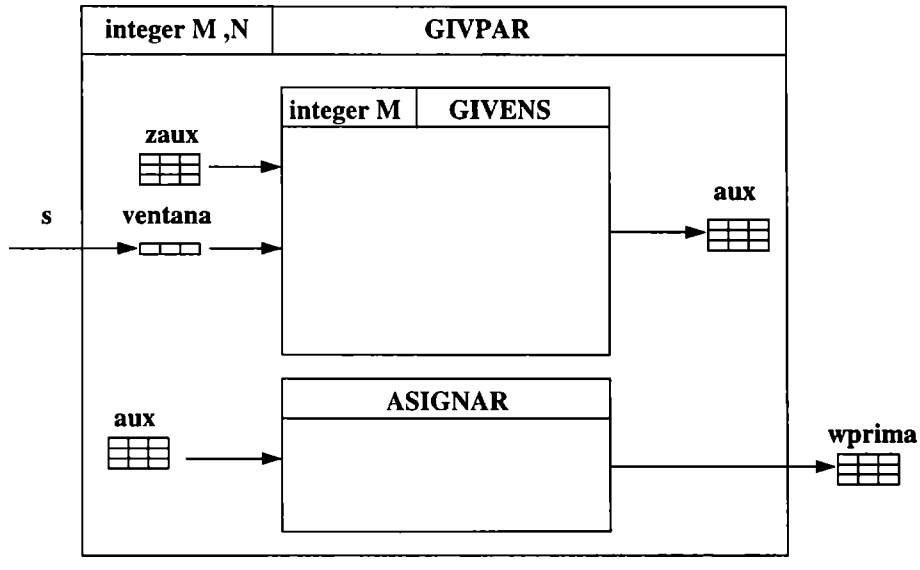


Fig 6.6 Esquematización del proceso *givpar*.

donde el proceso *"Givens"* es el que realiza la paralelización de la triangularización de la matriz de la siguiente manera:

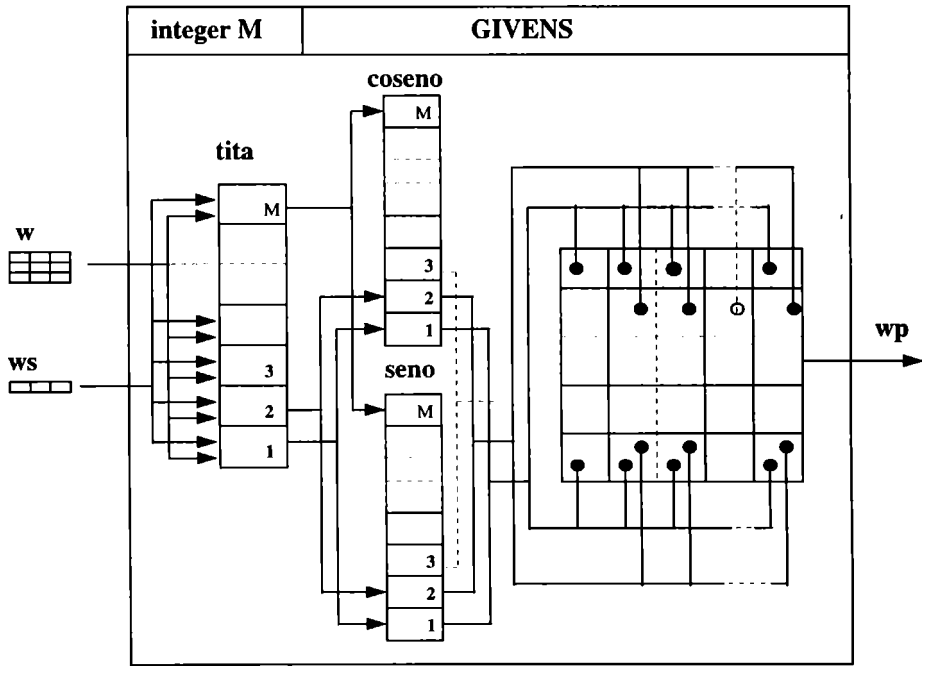
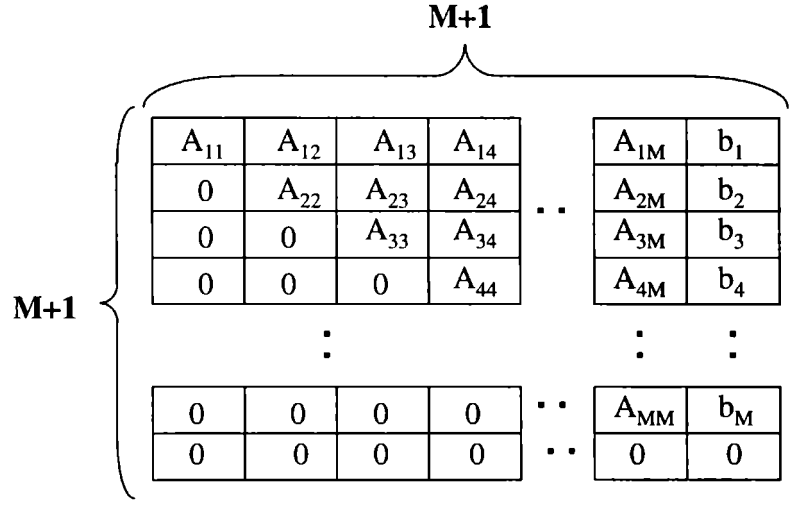


Fig 6. 7 Esquema detallado del proceso *givens* (paralelizado).

Aquí se puede observar la independencia de datos que permite realizar el cálculo en forma paralela.

6.6.3 Obtención de los coeficientes de Predicción Lineal

Una vez realizada la triangularización de la matriz que contiene concatenados coeficientes y términos independientes (*wprima*)



es posible obtener el vector de incógnitas que solucionan al sistema de ecuaciones, mediante la utilización de un método clásico de resolución de sistemas de ecuaciones como es el de *resolución backward.*

Dada una matriz de coeficientes, $\mathbf{A} \in R^{MxM}$, y un vector de términos independientes, $\mathbf{b} \in R^{Mx1}$, este método permite obtener la solución al sistema a partir de la siguiente expresión:

$$x_k = \frac{b_k - \sum_{j=k+1}^{M} A_{kj} x_j}{A_{kk}} \qquad (6.1)$$

Notar que para calcular la k-ésima incógnita es necesario conocer las restantes desde k+1 hasta M, por lo tanto existe una dependencia de datos que impide una paralelización completa del proceso de resolución.

Una posible aproximación que permite efectuar el método de resolución *backward* está representada por el siguiente proceso SIGNAL, parcialmente paralelizado:

```
process backward =(integer M) { ? [M, M] dpreal wprima
                                ! [M-1] dpreal
coeficientes}
 (|veces:= (M-1) when inicio default (zveces-1)%1%
  |zveces:= veces $1 %2%
  |inicio:= (zveces=1)%3%
  |{nuevamat,incógnita}:=
coefic(M){veces,matrizcoef}%4%
  |retarbufercoef:= bufercoef $1 %5%
  |array I to (M-1) of
      bufercoef:= (incógnita when (I=veces)) default
                  retarbufercoef[I] %6%
   end
  |retarnuevamat:= nuevamat $1 %7%
  |matrizcoef:= wprima default retarnuevamat %8%
  |fin:=(veces=1) %9%
  |coeficientes:= bufercoef when fin %10%
  |synchro{wprima,when inicio} %11%
  |synchro{nuevamat, incógnita,matrizcoef} %12%
  |)
   where
    logical fin,inicio;
    integer veces,zveces init 1;
    [M-1] dpreal retarbufercoef init [{to (M-2)}:0.0 d
0],
    bufercoef ;
    dpreal incógnita;
    [M,M] dpreal nuevamat , retarnuevamat,matrizcoef

    process coefic =(integer M) {? integer k;
                                   [M,M] dpreal matriz
                                 ! [M,M] dpreal
matmodif;
                                   dpreal lpc}
    (|lpc:= (matriz[k,M]-(sumatoria(M){k,matriz}))/
            /matriz[k,k] %13%
     |array I to M of
       array J to M of
        matmodif:=((matriz[I,J]*lpc) when
((J=k)and(I<k)))
```

```
                    default matriz[I,J] %14%
        end
       end
      |)
       where
        process sumatoria=(integer M) (? integer i;
                                          [M,M] dpreal
mat
                                         ! dpreal
resultado)
         (|zero:=0.0 d 0
          |array J to M of
            calculo := (calculo + (mat[i,(i+J)])) when
                       (J<=((M-1)-i)) default zero %15%
            with calculo[0]:zero
           end
          |resultado:= calculo[((M-1)-i)]
          |)
           where
            [M] dpreal calculo;
            dpreal zero
       end
   end
end
```

El proceso *backward* está compuesto por un sólo subproceso *coefic*; y en conjunto permiten calcular los coeficientes de Predicción Lineal (vector solución obtenido a través de sustitución backward) a partir de la matriz de coeficientes y términos independientes, *wprima*, obtenida del proceso *Givpar* antes descripto.

Backward recibe como parámetros la cantidad de coeficientes (orden de estimación) , M, y la matriz *wprima*; e invoca M veces (control efectuado por la señal *veces,* línea %1%) en forma secuencial (por la naturaleza del método de resolución) al subproceso *coefic* (línea %4%) el cual calcula una incógnita y la retorna (*incógnita*), ajusta la matriz de coeficientes con la misma (en forma paralela) y retorna la matriz modificada (*nuevamat*).

Cuando se han calculado todos los coeficientes el proceso retorna un arreglo (*coeficientes*) el cual fue obtenido asignando los elementos de a uno en cada invocación a *coefic* (línea %6%).

Es importante considerar que este proceso trabaja en paralelo con el proceso *Givpar* retornando por cada N muestras de un frame los coeficientes de PL necesarios para estimar las muestras de dicho frame.

Un diagrama del proceso es el siguiente:

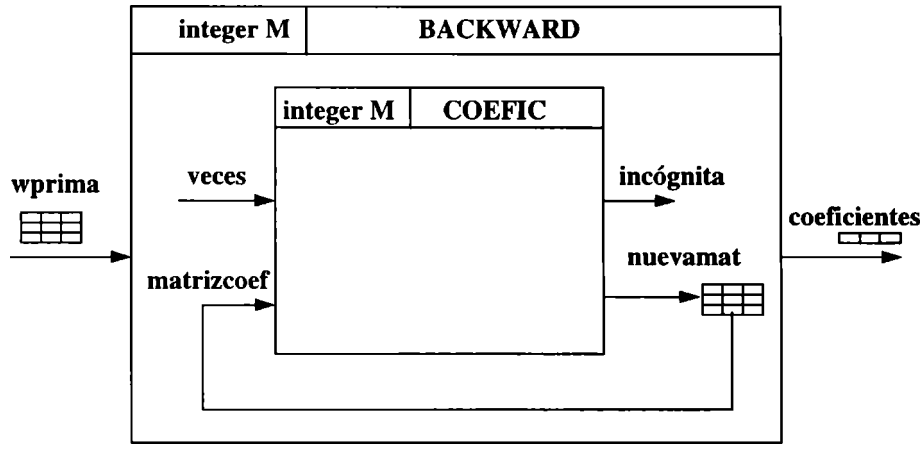
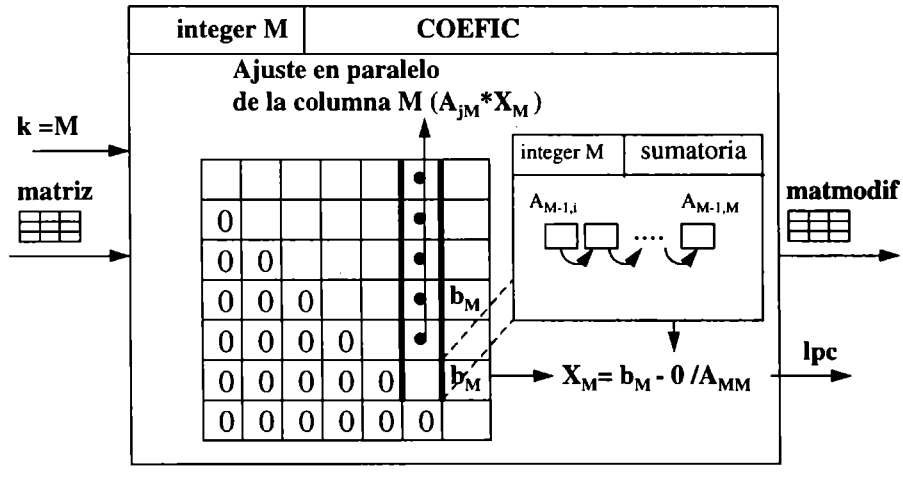


Fig 6. 8 Esquematización del proceso *backward.*

El comportamiento del proceso *coefic* pude ser observado a través de la siguiente ilustración que muestra como se transforma la matriz de entrada y se obtienen los coeficientes (incógnitas del sistema) a cada invocación:



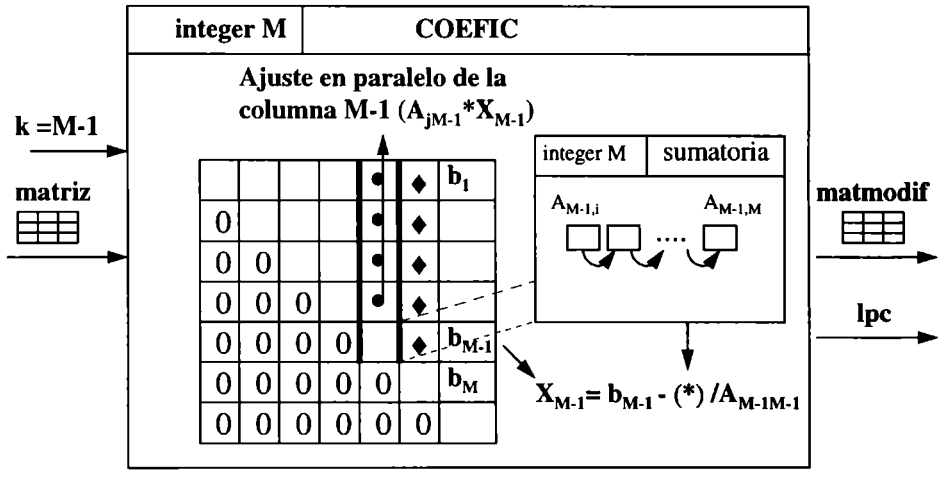


Fig 6.9 Dos etapas de las M que ejecuta *coefic* para obtener los coeficientes.

El proceso *sumatoria* (invocado por *coefic*) realiza la sumatoria a través del constructor **`array`** con la cláusula **`with`** (línea %15%).

Algunas estimaciones

Estos son ejemplos de estimaciones realizadas con el proceso de cálculo de coeficientes PL, habiendo compilado los procesos SIGNAL de manera de obtener un código fuente en lenguaje C a partir del cual se puede construir ejecutables para emulación. Los resultados fueron cotejados con aquellos obtenidos de la ejecución de las mismas rutinas de cálculo implementadas en MATLAB®, un utilitario que permite construir algoritmos de manipulación de tablas y vectores a partir de un conjunto de funciones predefinidas.

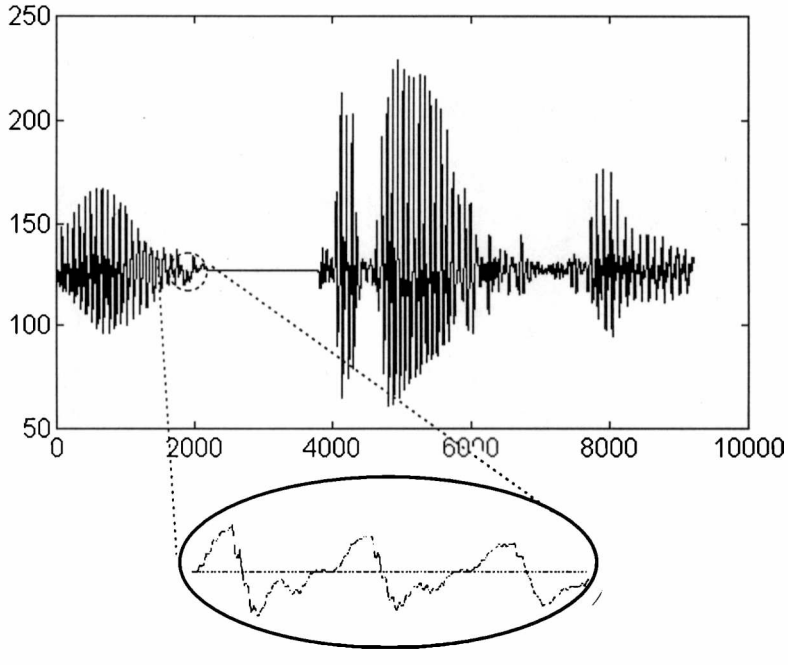
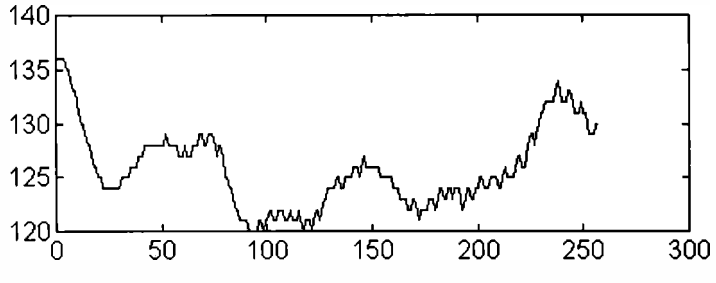


Fig 6.9 Forma de onda acústica de la palabra "estrofa"

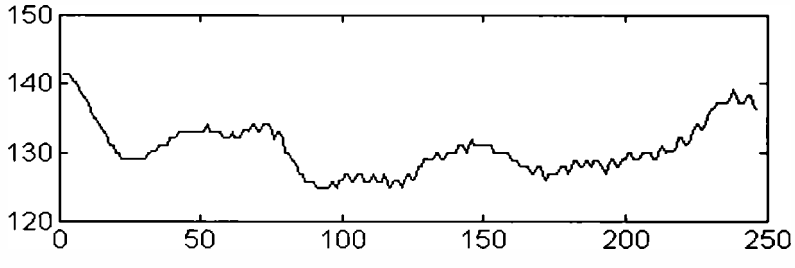
La figura de arriba presenta la forma de onda acústica de la palabra "*estrofa*", donde es posible distinguir zonas bien caracterizadas por los fonemas involucrados. La región ampliada corresponde a la zona de la "s" y es allí donde tomamos un frame de 256 muestras



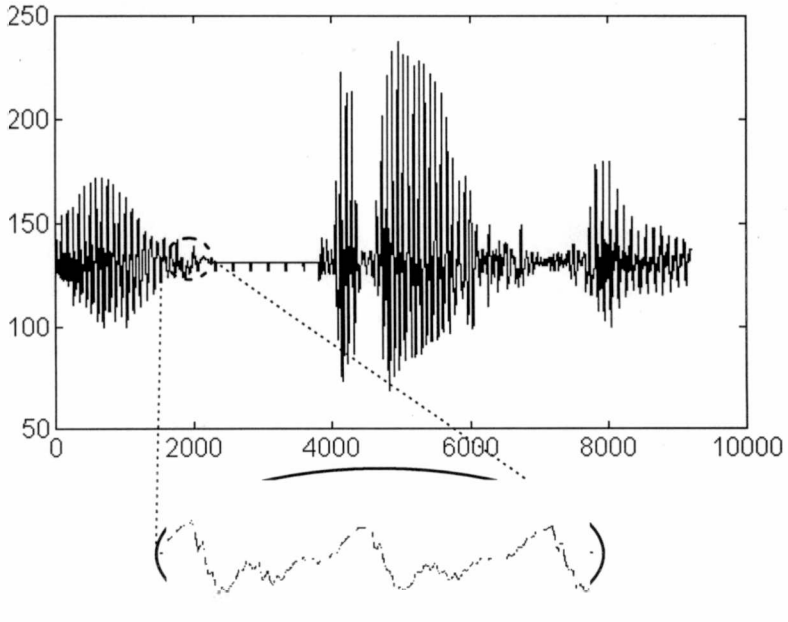
sobre el cual calculamos los coeficientes de Predicción y obtuvimos

COEFICIENTES	MATLAB	SIGNAL COMPILADO A LENG C
a_1	0.00453	0.004539
a_2	0.00455	0.004551
a_3	0.00459	0.004599
a_4	0.00462	0.004622
a_5	0.00463	0.004640
a_6	0.00465	0.004653
a_7	0.00464	0.004641
a_8	0.00466	0.004668
a_9	0.00465	0.004652
a_{10}	0.99830	0.998303

Con estos resultados, aplicando la ecuación (5.43), realizamos la estimación del frame y obtuvimos la siguiente onda:



Finalmente la estimación total de la señal es la siguiente:



Conclusiones

A través de la utilización del lenguaje SIGNAL, hemos comprobado que el mismo aporta ventajas al procesamiento de señales de voz, hecho este que consideramos de gran interés pues cualquiera sean los algoritmos destinados al procesamiento de voz, insumen gran cantidad de cálculos matemáticos los cuales deben realizarse en el menor tiempo posible por tratarse de procesos que se ejecutan en tiempo real.

El procesamiento de voz manipula las señales como una secuencia de valores reales (muestras) temporalmente consecutivos expresando relaciones y restricciones entre las mismas. SIGNAL es un lenguaje de especificación formal (o ecuacional) y puramente sincrónico; lo cual nos permite expresar las señales de voz como un conjunto de muestras más un reloj que le brinda una ubicación temporal a dicha señal, aportando así un alto grado de expresividad para establecer restricciones entre señales. Esta última bondad que ofrece SIGNAL, es por ser un lenguaje dedicado.

Además por su naturaleza sincrónica se considera cada acción como instantánea y las entradas y salidas de un proceso se asumen por adelantado, permitiendo así asegurar en tiempo de compilación la ausencia de deadlock en un determinado proceso.

Debemos remarcar la flexibilidad que ofrece SIGNAL para la representación y manipulación de valores numéricos, evitando por ejemplo desbordes en la realización de ciertos cálculos matemáticos.

Es importante también el hecho que el compilador SIGNAL genere código C paralelo y OCCAM listo para ser corrido sobre una determinada arquitectura mediante la utilización de una interfase gráfica (SynDEx), la cual recibiendo como parámetros el grafo de dependencia de un proceso SIGNAL generado por el mencionado compilador y la arquitectura de la que se dispone, aplica una heurística y obtiene un scheduling de ejecución adecuado.

Por otro lado incursionamos en el campo del procesamiento de señales de voz, particularmente en Predicción Lineal. Puntualmente estudiamos un método para hallar los coeficientes de Predicción Lineal denominado "método de la covarianza", el cual fue calculado utilizando la solución recursiva al Problema de Mínimos Cuadrados, que aplica las rotaciones de Givens para factorizar la matriz de coeficientes.

A partir de este estudio diseñamos un algoritmo paralelo de la versión original secuencial presentada en [6], la cual es recursiva en el tiempo y permite obtener la matriz de coeficientes y términos independientes del sistema de ecuaciones que, utilizando algún método de resolución, permite calcular los coeficientes de orden M que hacen posible la estimación de la n-ésima muestra de un frame determinado a partir de sus M muestras pasadas.

Esta paralelización no presentó mayores dificultades gracias a los constructores que ofrece el lenguaje SIGNAL para el manejo simultáneo de los elementos de tablas n-dimensionales, no obstante los mayores obstáculos surgieron a la hora de sincronizar las señales de entrada y locales, hecho éste que quedó reflejado en la previa implementación de una versión secuencial que presentaba un orden de ejecución superior.

La utilización del código fuente C, generado por el compilador SIGNAL, nos permitió verificaciones de los resultados obtenidos, no obstante los procesos diseñados no fueron corridos sobre arquitecturas paralelas que hubieran otorgado una mejor observación de las bondades del lenguaje en relación a lo especificado.

La utilización de SynDEx es de vital importancia para la asignación de procesos SIGNAL sobre arquitecturas multiprocesador. Sin embargo esta herramienta necesita para su manejo de un aprendizaje previo que no fue pautado dentro de los objetivos de esta investigación.

Los resultados obtenidos fueron cotejados con aquellos obtenidos de las mismas rutinas escritas secuencialmente en MATLAB® una herramienta que ofrece la posibilidad de diseñar algoritmos a partir de un conjunto de funciones predefinidas útiles para el manejo de matrices y vectores.

Como futuras extensiones a esta investigación, sería interesante la utilización de este algoritmo de prediccion lineal dentro de un proceso global que permita modelizar un sistema de producción de speech, adjuntando otros parámetros como pitch, ganancia, etc.

Apéndice

Rotaciones de Givens

La reducción de una matriz $\mathbf{A} \in \mathbb{R}^{n \times m}$ con $n \geq m$ a formas canónicas se realiza a través de transformaciones ortogonales. Las rotaciones de Givens es un ejemplo de dichas transformaciones.

Consideremos un vector de dos componentes $\mathbf{x} = [x_1, x_2]^T$, donde x_1 y x_2 son reales [18]. La longitud del vector es $r = \sqrt{x_1^2 + x_2^2}$ y el ángulo $\theta = tan^{-1}(x_2 / x_1)$. Así:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} r\cos\theta \\ r\,sen\theta \end{bmatrix}.$$

Supongamos que deseamos construir un vector **x** con el ángulo rotado como se muestra en la siguiente figura:

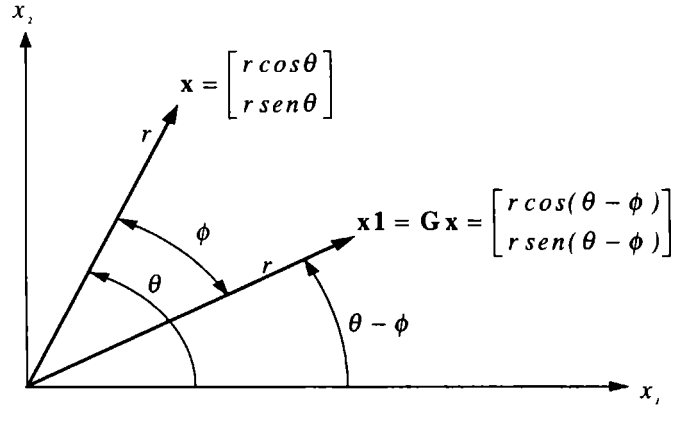


Fig.1 Rotación angular de un vector.

Es fácilmente verificable que seleccionando una matriz de transformación ortogonal **G** como:

$$\mathbf{G} = \begin{bmatrix} \cos\phi & sen\phi \\ -sen\phi & \cos\phi \end{bmatrix}$$

el vector **x1= Gx** es:

$$\mathbf{x1} = \begin{bmatrix} r\cos(\theta - \phi) \\ r\,sen(\theta - \phi) \end{bmatrix}$$

Notemos que $\mathbf{G}^T\mathbf{G}=\mathbf{I}$ así que los vectores columna en **G** están normalizados a la longitud unitaria.

Supongamos que elegimos una matriz de rotación angular tal que la segunda componente de **x1** sea cero, entonces:

$$-x_1\, sen\phi + x_2 \cos\phi = 0 \quad \text{y} \quad \phi = arctan(\, x_2 \,/\, x_1\,).$$

Así los elementos de **G** serán:

$$\mathbf{G} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$$

donde

$$c = \cos\phi = \frac{x_1}{\sqrt{x_1^2 + x_2^2}} \quad \text{y} \quad s = sen\phi = \frac{x_2}{\sqrt{x_1^2 + x_2^2}};$$

luego

$$\mathbf{Gx} = \begin{bmatrix} r \\ 0 \end{bmatrix}.$$

Este desarrollo demuestra que es posible eliminar un elemento en un vector bidimensional usando una rotación de Givens. Es posible extender la rotación elemental de Givens para aniquilar un elemento en un vector con N elementos. Esto puede ser hecho como sigue: para un vector N-dimensional: $\mathbf{x}_N=[x_1, \ldots, x_j, \ldots, x_k, \ldots, x_N]^T$ con $x_j<>0$ o $x_k<>0$, podemos construir una matriz ortogonal $\mathbf{G}_N$ de dimensión NxN tal que:

$$\mathbf{G_N} = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \cdots & \vdots & & \vdots \\ 0 & & c & \cdots & s & \cdots & 0 \\ \vdots & \cdots & \vdots & \ddots & \vdots & \cdots & \vdots \\ 0 & \cdots & -s & & c & \cdots & 0 \\ \vdots & & \vdots & \cdots & \vdots & \ddots & \vdots \\ 0 & & 0 & & 0 & \cdots & 1 \end{bmatrix} \begin{matrix} \\ \\ i \\ \\ j \\ \\ \\ \end{matrix}$$

$$\qquad\qquad i \qquad\quad j$$

donde $c = x_i \big/ \sqrt{|x_i|^2 + |xj|^2}$ y $s = x_j \big/ \sqrt{|x_i|^2 + |xj|^2}$. Se puede ver que:

$$\mathbf{G_N x_N} = \left[x_1,\ldots,\sqrt{|x_i|^2 + |x_j|^2},\ldots,0,\ldots,x_N \right]^T .$$

Desde que sólo los elementos x_j y x_i cambian, se dice que una rotación de Givens opera sobre el j-ésimo y k-ésimo elemento de $\mathbf{x_N}$. Si se trabaja con una matriz con N filas, se modificará solamente la i-ésima y la j-ésima fila.

Aplicando otra rotación de Givens sobre dos elementos del vector $\mathbf{G_N}$ $\mathbf{v_N}$ antes obtenido podemos generar un cero más en este vector. Así podemos eliminar todos excepto uno de los elementos del vector $\mathbf{x_N}$ utilizando a lo sumo N-1 rotaciones de Givens.

Como una consecuencia de la propiedad de preservación de la longitud de las transformaciones ortogonales, puede ser mostrado que el elemento residual es igual a la longitud de $\mathbf{x_N}$ (raiz cuadrada de la suma de los cuadrados de las componentes de $\mathbf{x_N}$).

Factorización QR

Existen distintas formas de descomponer la matriz de autocorrelación (mencionada en el Capítulo 4) y así proveer un medio para resolver las ecuaciones normales dadas en (*5.22*). Hay métodos para calcular la matriz de datos (A) directamente sin tener que obtener primero la matriz de autocorrelación.[13]

La factorización QR de una matriz de datos, $\mathbf{A} \in \mathbb{R}^{n \times m}$ con $n \geq m$, consiste en expresar a dicha matriz como un producto de las matrices $Q \in \mathbb{R}^{n \times n}$ ortogonal y $R \in \mathbb{R}^{m \times m}$ triangular superior tal que: $QA = \left[R \quad 0 \right]^T$, donde R tiene la siguiente forma:

$$R = \begin{bmatrix} & & \\ 0 & \ddots & \\ \vdots & \ddots & \\ \vdots & \vdots & \ddots \\ 0 & & 0 \end{bmatrix}$$

Así las transformaciones de Givens son utilizadas para realizar esta factorización.

Cualquier secuencia de rotaciones de Givens puede ser utilizada para triangularizar A, lo cual se puede realizar colocando 0 por columnas o por filas. Si el modo seleccionado es el segundo, el algoritmo es el siguiente:

```
for i = m:-1:2
    for j = 1:min{i-1,n}
        [c,s] = Givens(a_{i-1,j}, a_{ij})
        A = rot.fila(A,i-1,i,c,s)
    end
end
```

donde:

```
Function:[c,s]= Givens(a,b)
    if b=0
        c=1 , s=0
    else
        if |b|>|a|
            r=-a/b , s=1/(1+r^2)^1/2 , c=sr
        else
            r=-b/a , c=1/(1+r^2)^1/2 , s=cr
        end
    end
End Givens

Function:[A]= rot.fila(A,j,i,c,s)
    q= cols(A)
    for t = 1:q
        r1=A(j,t) , r2=A(i,t)
```

```
            A(j,t)= cr1-sr2 , A(i,t)= sr1+cr2
        end
    End rot.fila
```

Ejemplo de factorización

Después de las consideraciones presentadas retornemos al sistema de ecuaciones 5.65 y tomemos N=3 y M=2. El objetivo es factorizar las matrices $\bar{S}(N)$ y $\bar{\bar{s}}(N)$ utilizando rotaciones de Givens tal que exista una solución mediante sustitución *backward* para $Q\bar{\mathbf{S}}(N)\hat{\mathbf{a}}(N) = Q\,\bar{\bar{\mathbf{s}}}(N)$ con $Q = \mathcal{R}_3\mathcal{R}_2\mathcal{R}_1$ donde $\mathcal{R}_i$ es la productoria de las matrices de rotación utilizadas para colocar ceros en la fila i o sea la fila $2+i$.

Sea $s(1)=2$, $s(2)=4$, y $s(3)=5$, luego:

$$Q\begin{bmatrix}0 & 0\\ 0 & 0\\ 2 & 0\\ 4 & 2\\ 5 & 4\end{bmatrix}\begin{bmatrix}\hat{a}_1\\ \hat{a}_2\end{bmatrix} = Q\begin{bmatrix}0\\ 0\\ 2\\ 4\\ 5\end{bmatrix}$$

1. Eliminar la fila $i=1$ (fila 3) aplicando $\mathcal{R}_1$

1.1. Eliminar el elemento (3,1); $\theta=arctan(2/0)=1.5707$, $cos(\theta)=0$; $sen(\theta)=1$; por lo tanto la matriz de rotación será:

$$G(3,1) = \begin{bmatrix}\mathbf{0} & 0 & \mathbf{1} & 0 & 0\\ 0 & 1 & 0 & 0 & 0\\ \mathbf{-1} & 0 & \mathbf{0} & 0 & 0\\ 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 1\end{bmatrix}$$

luego :

$$G(3,1)\bar{\mathbf{S}}(3) = \begin{bmatrix}\mathbf{2} & \mathbf{0}\\ 0 & 0\\ \mathbf{0} & \mathbf{0}\\ 4 & 2\\ 5 & 4\end{bmatrix} \quad y \quad G(3,1)\bar{\bar{\mathbf{s}}}(3) = \begin{bmatrix}\mathbf{2}\\ 0\\ \mathbf{0}\\ 4\\ 5\end{bmatrix}$$

1.2. Eliminar el elemento (3,2); $\theta=arctan(0/0)=1.5707$, $cos(\theta)=0$; $sen(\theta)=1$; por lo tanto la matriz de rotación será:

$$G(3,2)=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{0} & \mathbf{1} & 0 & 0 \\ 0 & \mathbf{-1} & \mathbf{0} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

luego :

$$G(3,2)\bar{\mathbf{S}}(3)=\begin{bmatrix} 2 & 0 \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \\ 4 & 2 \\ 5 & 4 \end{bmatrix} \quad y \quad G(3,2)\bar{\bar{\mathbf{s}}}(3)=\begin{bmatrix} 2 \\ \mathbf{0} \\ \mathbf{0} \\ 4 \\ 5 \end{bmatrix}$$

2. Eliminar la fila $i=2$ (fila 4) aplicando $\mathcal{R}_2$

2.1. Eliminar el elemento (4,1); $\theta=arctan(4/2)=1.1071$, $cos(\theta)=0.4472$; $sen(\theta)=0.8944$; por lo tanto la matriz de rotación será:

$$G(4,1)=\begin{bmatrix} \mathbf{0.4472} & 0 & 0 & \mathbf{0.8944} & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \mathbf{-0.8944} & 0 & 0 & \mathbf{0.4472} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

luego :

$$G(4,1)\bar{\mathbf{S}}(3)=\begin{bmatrix} \mathbf{4.47} & \mathbf{1.78} \\ 0 & 0 \\ 0 & 0 \\ \mathbf{0} & \mathbf{0.89} \\ 5 & 4 \end{bmatrix} \quad y \quad G(4,1)\bar{\bar{\mathbf{s}}}(3)=\begin{bmatrix} \mathbf{4.47} \\ 0 \\ 0 \\ \mathbf{0} \\ 5 \end{bmatrix}$$

2.2. Eliminar el elemento (4,2); $\theta=arctan(0.89/0)=1.5707$, $cos(\theta)=0$; $sen(\theta)=1$; por lo tanto la matriz de rotación será:

$$G(4,2)=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{0} & 0 & \mathbf{1} & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & \mathbf{-1} & 0 & \mathbf{0} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

luego :

$$G(4,2)\overline{\mathbf{S}}(3)=\begin{bmatrix} 4.47 & 1.78 \\ \mathbf{0} & \mathbf{0,89} \\ 0 & 0 \\ \mathbf{0} & \mathbf{0} \\ 5 & 4 \end{bmatrix} \quad y \quad G(4,2)\overline{\overline{\mathbf{s}}}(3)=\begin{bmatrix} 4.47 \\ \mathbf{0} \\ 0 \\ \mathbf{0} \\ 5 \end{bmatrix}$$

3. Eliminar la fila $i=3$ (fila 5) aplicando $\mathcal{R}_3$

3.1. Eliminar el elemento (5,1); $\theta=arctan(5/4.47)=0.841$, $cos(\theta)=0.666$; $sen(\theta)=0.745$; por lo tanto la matriz de rotación será:

$$G(5,1)=\begin{bmatrix} \mathbf{0.666} & 0 & 0 & 0 & \mathbf{0.745} \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ \mathbf{-0.745} & 0 & 0 & 0 & \mathbf{0.666} \end{bmatrix}$$

luego :

$$G(5,1)\overline{\mathbf{S}}(3)=\begin{bmatrix} \mathbf{6.702} & \mathbf{4.170} \\ 0 & 0.894 \\ 0 & 0 \\ 0 & 0 \\ \mathbf{0} & \mathbf{1.331} \end{bmatrix} \quad y \quad G(5,1)\overline{\overline{\mathbf{s}}}(3)=\begin{bmatrix} \mathbf{6.702} \\ 0 \\ 0 \\ 0 \\ \mathbf{0} \end{bmatrix}$$

3.2. Eliminar el elemento (5,2); $\theta=arctan(1.331/4.170)=0.9793$, $cos(\theta)=0.557$; $sen(\theta)=0.830$; por lo tanto la matriz de rotación será:

$$G(5,2)=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{0.557} & 0 & 0 & \mathbf{0.830} \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & \mathbf{-0.830} & 0 & 0 & \mathbf{0.557} \end{bmatrix}$$

luego :

$$G(5,2)\bar{\mathbf{S}}(3)=\begin{bmatrix} 6.702 & 4.170 \\ \mathbf{0} & \mathbf{1.608} \\ 0 & 0 \\ 0 & 0 \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \quad y \quad G(5,1)\bar{\bar{\mathbf{s}}}(3)=\begin{bmatrix} 6.702 \\ \mathbf{0} \\ 0 \\ 0 \\ \mathbf{0} \end{bmatrix}$$

Vemos que las matrices obtenidas son triangular superior. Nuestro fin aqui fue mostrar el método y no el grado de estimación alcanzado, pues es claro que para ello es necesario contar con más valores de los considerados para el ejemplo.

El paso siguiente luego de la triangularización es la resolución backward del sistema alterado.

Problema de Mínimos cuadrados "Full Rank"

Consideremos el problema de resolver la siguiente ecuación matricial:

$$Ax = b$$

donde $A \in \mathbb{R}^{n \times m}$, $x \in \mathbb{R}^m$ y $b \in \mathbb{R}^n$ con $n \geq m$. Si x es el vector de incógnitas y b, el vector solución, estamos en presencia de un sistema con más ecuaciones que incógnitas denominado *sobredeterminado*. Usualmente este tipo de sistemas de ecuaciones lineales no tiene una solución exacta desde que b debe ser un elemento del rango(A), un subespacio propio de $\mathbb{R}^n$.

Para encontrar el vector x adecuado deberíamos tender a minimizar $\|Ax-b\|_p$ para algún p en particular. El problema de mínimos cuadrados, es decir $p = 2$,

$$\min_{x \in R^n} \|Ax - b\|_2$$

es más manejable por dos razones:

- $\phi(x)=\frac{1}{2}\|Ax-b\|_2^2$ es una función diferenciable de x, tal que los mínimos de esa función satisfacen la ecuación del gradiente, $\nabla\phi(x)=0$. Así el sistema lineal obtenido es simétrico y definido positivamente en el caso de que la matriz A sea full rank en columnas.
- además la norma-2 se preserva bajo la transformación ortogonal, es decir que podemos obtener una matriz Q, tal que el problema de minimizar $\|Q^TAx-Q^Tb\|_2$ sea fácil de resolver. Esto nos permite concluir que es posible trabajar con las matrices factorizadas, por ejemplo, utilizando en Q la productoria de n matrices de rotación de Givens de la forma:

$$Q = G_n G_{n-1} \cdots G_1$$

Dos conceptos aqui son importantes: espacio nulo y rango.[12]

Si A es una matriz singular, luego hay algún subespacio de x llamado espacio nulo, tal que $Ax=0$. La dimensión del espacio nulo es el número de vectores linealmente independientes de dicho espacio, la cual es denominada "*nulidad de A*". También hay algún subespacio de b que puede ser alcanzado desde A, es decir que existe algún x, tal que $Ax=b$. Este subespacio es llamado "*rango de A*". La dimensión de este subespacio es llamada *rank de A*.

Si A es no singular, luego todos los vectores del subespacio de x serán tales que $Ax=b$ por lo tanto el *rank(A)= número de columnas de A = m*.

Si A es singular, existe entonces al menos un vector x tal $Ax=0$; por lo tanto el *rank(A) < m*.

Una vez introducidos los conceptos precedentes, podemos afirmar que si A es full rank (es decir que la dimensión del espacio nulo de A es cero) se cumple lo siguiente:

existe una única solución al problema de los mínimos cuadrados (x_{ls}) y por lo tanto del sistema lineal positivo que representa (Véase [13]). *Dicho sistema es el siguiente:*

$$A^T A x_{ls} = A^T b.$$

Bibliografía

[1] Carlos J. Bogni, Luis A. Marrone, *Computadoras: Introducción a las arquitecturas paralelas,* UNICAM, 1986

[2] Armando E. De Giusti, *Descripción y verificación de hardware. Aplicaciones en Tiempo Real*

[3] P. Le Guernic, T. Gautier, *Data-Flow to Von Neumann: the SIGNAL approach,* Rapport de Recherche1229, INRIA, Mayo 1990.

[4] A. Benveniste, G. Berry, *The synchronous approach to reactive and real-time systems,* Rapport de Recherche 1445, INRIA, Junio 1991.

[5] A. Benveniste, M. Le Borgne, P. Le Guernic, *SIGNAL as a model for real-time and hibrid systems,* Rapport de Recherche 1608, INRIA, Febrero 1992.

[6] J. R. Deller, J. G. Proakis, J. H. L. Hansen, *Discrete-Time Processing of Speech Signals,* Macmillan, 1993.

[7] M. Cosnard, P. Quinton, Y. Robert, M. Tchuente, *Parallel Algorithms & Architectures,* Proc. of the International Workshop on Parallel Algorithms & Architectures, Luminy, France, Abril 1986.

[8] P. Le Guernic, M. Le Borgne, T. Gautier, C. Le Maire, *Programming real-time applications with SIGNAL,* Rapport de Recherche 1446, INRIA, Junio 1991.

[9] P. Bournai, B. Cheron, B. Houssais, P. Le Guernic, *Manuel SIGNAL,* Rapport techniques 128, INRIA, Abril 1991.

[10] J. R. Deller and G. P. Picaché, *Advantages of a Givens Rotations Approach to temporally recursive linear prediction analysis of speech,* IEEE Trans. on Speech and Signal Processing, Vol. 1 pp 509-512, 1988.

[11] W. Press, B. Flannery, S. Teukolsky and W. Vetterling, *Numerical Recipes in C,* Cambridge University Press, 1988.

[12] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis,* Springer-Verlag NY, 1989.

[13] G. H. Golub and C. Van Loan, *Matrix Computations,* Baltimore MD: Johns Hopkins Press, 1983, Sects. 2.3, 3.4, 5.6, 6.1, 6.3, 12.6.

[14] D. O'Shaughnessy, *Speech Communication: Human and Machine.* Reading, MA: Addisson-Wesley, 1987, Sects. 8.2, 8.4.

[15] J. Makhoul, *"Linear prediction : A tutorial review, "* Proc. IEEE, vol.63, pp 561-580, 1975. Reprinted in *Speech Analysis,* R. W. Schafer and J. D. Markel, Eds. NY: IEEE Press, 1978.

[16] C. L. Lawson R. J. Hanson, *Solving Least Squares Problems.* Englewood Cliffs, NJ: Prentice-Hall, 1974, Ch. 27.

[17] T. C. Luk and J. R. Deller, Jr., *"A nonclassical WRLS algorithm, "* in Proc. 23rd Annu. Allerton Conf. Commun., Contr., Comput., Champaign, IL, 1985.

[18] J. G. Proakis, Charles M. Rader, F. Ling, Ch. L. Nikias, *"Advanced Digital Signal Processing,"* Ch 5 y 7 , Macmillan Plublishing Company, 1992

[19] Yves Sorel, *"Le Langage SIGNAL,"*. INRIA Rocquencourt . Francia, 1994.

[20] Paul M. Embree, Bruce Kimble, *"C Language algorithms for digital signal processing,"* Cap. 1, Prentice-Hall, Inc., 1991.

[21] J.M.C. Thomas, L.Bouquiaux, F. Cloarec-Heiss, *"Iniciación a la fonética,"* Editorial Gredos, Madrid, 1986.

[22] Luis Rocha, *"Procesamiento de voz,"* I EBAI, Edit. Kapelusz S.A., Buenos Aires, 1987.

[23] J.P.Laebenes, *"Unbiased identifications of AR Systems with a class of correlated inputs,"* M.S. thesis, Illinois Inst. Thecnol., Chicago, 1982.

[24] J.P.Laebenes, J.R. Deller, Jr., K " *`SISI`- A silent input selective sequential identifier for AR systems,"* in Proc. GRETSI Neuvieme Colloque sur le Tratement du Signal et ses Applications, vol. 2, Nice, Francia, 1983, pag. 989-994.

[25] T.C.Luck, J.R. Deller, Jr., *"A nonclassical WRLS algorithm,"* in Proc. 23 rd. Annu. Allerton Conf. Commun., Contr., Comput., Champaign, IL, 1985.

[26] J.R. Deller, Jr., D. Hsu, *"An alternative adaptive sequential regression algorithm and its application to the recognition of cerebral palsy speech,"* IEEE Trans. Circuits Sys., vol CAS-34. pag. 782-787, Julio 1987.

[27] G.P.Picaché, *"A Givens rotations algorithm for single channel formant tracking and glottal waveform deconvolution,"* M.S. thesis, Boston, 1988.