UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD INGENIERÍA

APLICACIONES DE LOS MICROPROCESADORES TMS320CXX

PRÁCTICAS DE LABORATORIO

BOHUMIL PŠENIČKA
MAURICIO A. ORTEGA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

DEPARTAMENTO DE TELECOMUNICACIONES

PŠENIČKA, Bohumil y Mauricio A. Ortega. *Aplicaciones de los microprocesadores TMS320CXX. Prácticas de Laboratorio.* México, UNAM, Facultad de Ingeniería, 2000, 182 p.

Aplicaciones de los microcontroladores TMS320CXX.
Prácticas de laboratorio.

ISBN 968-36-8304-5

Primera edición, noviembre de 2000.

Impreso en México

Prólogo

La gran cantidad de aplicaciones que tiene el procesamiento digital de señales incluyen áreas, tales como: radar, sonar, voz, comunicaciones, telefonía, medicina, control, sismología, imágenes, etc. Las herramientas que han permitido obtener soluciones reales y eficientes son el desarrollo de las tecnologías de programación y los microprocesadores de procesamiento digital de señales.

El amplio crecimiento que ha tenido la industria digital se debe en gran medida al desarrollo de algoritmos de procesamiento digital de señales. Las áreas antes mencionadas requieren aplicaciones, tales como filtrado, compresión, análisis en frecuencia, entre otras. El uso de sistemas digitales permite realizar estas tareas con una computadora digital o un microprocesador.

Dentro de las herramientas de cómputo para el procesamiento digital de señales se encuentran los programas de simulación donde los datos pueden analizarse, aplicando filtros digitales o transformada de Fourier, y graficarse en una computadora. El paquete de cómputo MATLAB es un ejemplo de una poderosa herramienta para este tipo de simulaciones. Las tarjetas de microprocesadores *starter-kit* son otra ayuda para la ejecución de programas para procesamiento de señales. Estos programas pueden ejecutarse en el microprocesador, utilizando señales de un generador o señales reales de voz o audio, y observar la salida en un osciloscopio.

El propósito de este libro es presentar, a los alumnos de las carreras de Ingeniería en Telecomunicaciones, Electrónica y Computación de la Facultad de Ingeniería de la UNAM, aplicaciones del procesamiento digital de señales. Una gran variedad de ejemplos se incluyen en cada capítulo con el fin de que el alumno aplique los conocimientos teóricos de la materia de forma práctica a través de simulaciones en la computadora. El libro contiene dos partes: la primera está orientada a la implementación de programas en el simulador del microprocesador para procesamiento de señales de la compañía Texas Instruments TMS320C25 y en la segunda parte se presentan ejemplos de programas para procesamiento de señales mediante el paquete MATLAB, algunos de éstos requieren de la librería del *toolbox* de *Signal Processing* y el *toolbox* de *Image Processing*.

Los programas para el microprocesador se presentan de los capítulos del uno al ocho y los programas para MATLAB del nueve al trece. En los primeros tres capítulos el objetivo es que el alumno se familiarice con las instrucciones del microprocesador; además, que conozca la forma de utilizar el compilador y el ligador, como también el simulador para ejecutar las aplicaciones del microcontrolador, utilizando una computadora. En el capítulo 4 se presentan ejemplos de generación de señales con filtros tipo IIR y FIR. En el capítulo 5 se analizan ejemplos para realizar filtros tipo RC paso bajas y paso altas. Los algoritmos

para implementar la transformada rápida de Fourier y la transformada coseno se describen ampliamente en los capítulos 6 y 7. Finalmente, se presenta una introducción a los filtros adaptables en el capítulo 8. Este tipo de filtros tiene una importante aplicación en problemas de cancelación de ruido, ecualización del canal de comunicación e identificación de sistemas.

A partir del capítulo 9 se presentan programas para el paquete MATLAB y se incluye una introducción para que los alumnos aprendan el manejo de los programas. En los siguientes capítulos se describen programas de aplicación de acuerdo con los principales temas de la asignatura de procesamiento digital de señales. En el capítulo 10 se analizan los sistemas digitales, el muestreo de señales y la convolución mediante ejemplos que permiten entender claramente estos conceptos. El capítulo 11 describe las principales herramientas para el análisis en frecuencia, así como las propiedades de la transformada discreta de Fourier. El capítulo 12 presenta las funciones principales de MATLAB para el diseño de filtros. Finalmente, en el capítulo 13 el alumno podrá estudiar una introducción al procesamiento de imágenes, utilizando el paquete MATLAB.

Este libro representa un gran esfuerzo de los autores para contribuir a que los alumnos tengan un material para desarrollar aplicaciones del procesamiento digital de señales a algunos problemas reales. Es nuestro deseo que los alumnos encuentren en este material una motivación para el estudio del procesamiento digital de señales. Se agradece el apoyo a la Facultad de Ingeniería de la UNAM y a las personas que colaboraron en el proceso para la publicación de esta obra, particularmente a la Mtra. María Cuairán Ruidíaz, Jefa de la Unidad de Apoyo Editorial, y a la Lic. Amelia Guadalupe Fiel Rivera por la estructuración didáctica y corrección de estilo.

Dr. Bohumil Pšenička
M.C. Mauricio A. Ortega

Profesores de la Facultad
de Ingeniería de la UNAM

Contenido

Abreviaturas

ACC- Acumulador.
ACCH- Parte alta del acumulador.
ACCL- Parte baja del acumulador.
ARAU- Unidad aritmético lógica de los registros auxiliares.
ARB- Buffer de los registros auxiliares.
ARP- Apuntador de los registreos auxiliares.
ARx- Registros auxiliares x=0,1,...,7.
CALU- Unidad central aritmético lógica.
CNFD- Configura bloque B0 en la memoria de datos.
CNFP- Configura bloque B0 en la memoria de programa.
COFF- El archivo ejecutable para EVM de formato común.
DAB- Bus de datos.
DP- Apuntador de página.
DR- Pin del puerto de serie para recepción.
DRR- Registro para recepción en la forma serial.
DSP- Procesamiento digital de señales.
DSK- Archivo ejecutable para starter-kit.
DX- Pin del puerto de serie para transmisión.
DXR- Registro para transmisión en la forma serial.
EVM- Módulo de evaluación.
FIR- Filtros con la respuesta finita.
GREG- Registro de memoria global.
IE- Registro de habilitación.
IF- Registro de banderas.
IFR- Registro de bandera para interrupción.
IMR- Registro interrupción mascarable.
IIR- Filtros con la respuesta infinita.
IR0- Registro índice 0.
IR1- Registro índice 1.
MIPS- Millones de instrucciones por segundo.
MUX- Multiplexor.
PAB- Bus de programa.
PFC- Registro para direccionar B0.
PC- Contador de programa.
PR- Registro P.
Rx- Registro de precisión extendida x=0,1,...,7.
RE- Registro de dir. final de repetición.
RPTC- Registro de repetición.
RS- Registro de dir. inicial de repetición.
RSR- Registro de corrimiento para recepción serial.
ST0- Registro de estado 0.
ST1- Registro de estado 1.
TBC- Controlador de canal de pruebas.
TR- Registro T.
VLSI- Muy alta escala de integración.

Capítulo 1

Tarjetas de microprocesadores

El propósito de este capítulo es aprender cómo se trabaja con los *starter-kit* de Texas Instruments TMS320C25, TMS320C30 y TMS320C50. Los estudiantes tendrán la posibilidad de trabajar con una señal real. Asimismo, podrán conectar el starter-kit a una bocina, micrófono, generador de señales y ver en el osciloscopio la respuesta de un programa que se carga en la memoria del chip. Este programa puede ser un filtro digital o un algoritmo de la transformada rápida de Fourier.

En este capítulo se analizan, primeramente, las directivas más importantes utilizadas para compilar adecuadamente el programa. En seguida se describe la tarjeta del starter-kit y los chips que la integran y, por último, los comandos del ligador.

1.1 Microcontrolador TMS320C50

1.1.1 Directiva .set

La directiva **.set** iguala un símbolo con un valor constante. Es necesario escribir el símbolo como una etiqueta en la primera columna. En el ejemplo se muestra en qué forma se pueden escribir las constantes.

Ejemplo 1

Escriba el siguiente programa en un archivo const.asm.

```
;=============================================================
     .ps    .0a00h    ;se inicializa el contador del programa
;=============================================================
         .ps   0a00h        ;Se inicializa el contador del programa PC
K        .set  12           ;Se define K=12
K*2      .set  24           ;Se define la variable K*2=24
BIN      .set  0101010101b  ;BIN=055h
max_buf  .set  K*2          ;max_buf=k*2=24
```

```
FFT      .set  256          ;FFT=256
FFT-1    .set  255          ;FFT-1=255
         lar   AR0,#FFT     ;Al AR0 se carga 256
         lacl  #K           ;Al ACC se carga numero 12
         lacl  #K*2         ;Al ACC se carga numero 24
         lacl  max_buf      ;Al Acc se carga numero 24
         lacl  BIN          ;Al ACC se carga numero 55 hexa.
```

En el directorio donde está el archivo dsk5a.exe, se teclea **dsk5a.exe const.asm -l** y se obtiene el archivo const.lst, en el cual aparecerán en un listado los errores que se cometan en el programa.

```
00001 ---- ----  ;=============================================================
00002 ---- 0000       .ps    .0a00h    ;se inicializa el contador del programa
00003 ---- ----  ;=============================================================
00004 ---- 0a00            .ps    0a00h        ;Se inicializa PC contador
00005 ---- 000c  K         .set   12           ;Se define K=12
00006 ---- 0018  K*2       .set   24           ;Se define la variable K*2=24
00007 ---- 1041  BIN       .set   0101010101b  ;BIN=055h
00008 ---- 0018  max_buf   .set   K*2          ;max_buf=k*2=24
00009 ---- 0100  FFT       .set   256          ;FFT=256
00010 ---- 00ff  FFT-1     .set   255          ;FFT-1=255
00011 0a00 bf08            lar    AR0,#FFT     ;Al AR0 se carga 256
      0a01 0100
00012 0a02 b90c            lacl   #K           ;Al ACC se carga numero 12
00013 0a03 b918            lacl   #K*2         ;Al ACC se carga numero 24
00014 0a04 6918            lacl   max_buf      ;Al Acc se carga numero 24
00015 0a05 6941            lacl   BIN          ;Al ACC se carga numero 55h
>>>>> FINISHED READING ALL FILES
>>>>> ASSEMBLY COMPLETE: ERRORS:0   WARNINGS:0
```

1.1.2 Directivas .qxx y .lqxx

Las directivas **.qxx** y **.lqxx** generan los números fraccionarios y **xx** define en qué posición es punto decimal. Las directivas tienen la siguiente sintaxis:

$$\textbf{.qxx}\ \ valor1[, valor2, valor3, ...valorn]$$

$$\textbf{.lqxx}\ \ valor1[, valor2, valor3, ...valorn]$$

Ejemplo 2

Escriba el siguiente programa en el archivo frac.asm, utilizando **dsk5a frac.asm -l**, y cree el listado para observar cómo y dónde se guardan los números.

```
        .ds         400h      ;El numero que sigue se guarda en la direccion 400
        .q15        0.25
        .q15        .5
        .q15        0.75
        .q15        -.25,-.5,-.75
        .lq24       9,10,120
```

Del archivo frac.lst se observa que el número 0.25 se expresa en la forma hexadecimal 2000h y se guarda en la dirección 0400h; el número fraccionario -0.25 está ubicado en la dirección 0404h y tiene el valor de 16 bits e000h; mientras que el número 10 está expresado con 32 bits y ocupa dos palabras en las direcciones 0409h y 0404ah. El punto decimal está en la posición 24, después del número 9. 000009. Entonces, 6.4=24 bits.

```
00001 ---- 0400     .ds      400h ;El numero 0.25 se guarda en la direccion 400
00002 0400 2000     .q15     0.25
00003 0401 4000     .q15     .5
00004 0402 6000     .q15     0.75
00005 0403 e000     .q15     -.25,-.5,-.75
      0404 c000
      0405 a000
00006 0406 0000     .lq24       9,10,120
      0407 0900
      0408 0000
      0409 0a00
      040a 0000
      040b 7800
>>>>> FINISHED READING ALL FILES
>>>>> ASSEMBLY COMPLETE: ERRORS:0   WARNINGS:0


SYMBOLS
address  name                  address  name
--------  ----                 --------  ----
```

1.1.3 Directivas para secciones .ps .text .ds .data .bss .usect

El programa de un filtro o de la transformada rápida de Fourier incluye las instrucciones y las directivas. Las instrucciones son, por ejemplo: ADD para sumar, MPY para multiplicar o LTD para cargar el registro TR, sumar y correr los datos en la memoria de datos.

Las directivas dicen al microcontrolador, por ejemplo, donde guardar los datos en la memoria de datos **.ds 0a00h = .data** o en la memoria del programa guardar las instrucciones **.ps 0e00h=.text**.

La directiva **.entry** guarda en el contador del programa (PC) la dirección de donde se empieza a correr el programa. La sección es el bloque de los códigos o los datos que se guardarán separados en el mapa de la memoria. En general, tenemos tres secciones:

- **.text** (sección de textos)
- **.data** (sección de datos)
- **.bss** (sección para reservar el espacio)

En la figura 1.1 podemos ver que la sección de datos y la del programa están en la memoria del programa y la sección marcada **.bss** está en la memoria de datos en el chip RAM.

Las directivas **.bs** y **.usect** crean las secciones (*unitialized*) y los demás crean las secciones (*initialized*). Si en el programa no se especifica ninguna sección, el ensamblador junta todo en la sección **.text**. La sintaxis para estas directivas es la siguiente:

```
.bss      simbolo, el tamano, [la bandera]
.usect    "nombre de la seccion, el tamano, [la bandera]
.text
.data
.sect     "nombre de la seccion"
.asect    "nombre de la seccion", direccion
.ds       [direccion]
.ps       [direccion]
```

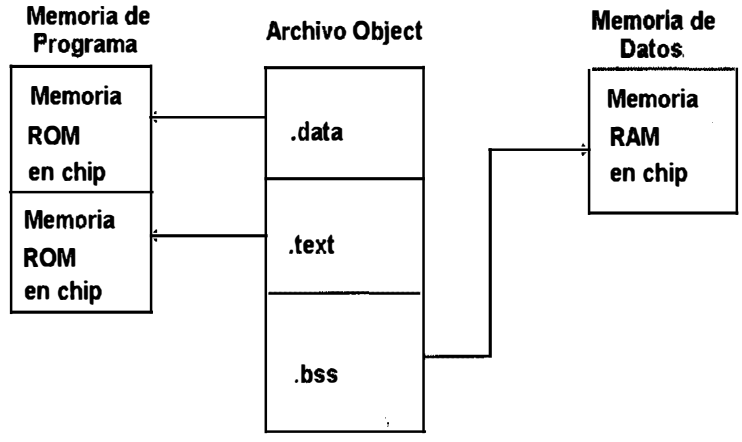


Figura 1.1. División de la memoria RAM en bloques

Ejemplo 3

Escriba el siguiente programa en el archivo section.asm y ensamble mediante el comando **dspa section.asm -l**.

```
*********************************************
**  Ensamblar los datos en la seccion .data  **
*********************************************
        .data
coef    .word           011h, 022h, 033h
*********************************************
**  Reservar el espacio para dos variables  **
*********************************************
        .bss            var2 ,1
        .bss            buffer, 10
*********************************************
**  Todavia ensamblar en la seccion de datos **
*********************************************
ptr     .word           0123h
*********************************************
**  Ensamblar los codigos en seccion .text  **
*********************************************
        .text
add:    lac             0Fh
aloop:  sblk            1
        blez            aloop
        sacl            var2, 0
************************************************
** Ensamblar otros datos en la seccion de datos **
************************************************
       .data
ivals  .word             0aah, 0bbh
*********************************************
**   Se define la nueva seccion "newvars"    **
*********************************************
var1   .usect           "newvars", 1
inbuf  .usect           "newvars", 7
*********************************************
**  Ensamblar mas codigos en seccion .text   **
*********************************************
       .text
       add              #00ffh
```

En el directorio obtenemos el archivo section.lst, listado donde se marcan los errores. Con **dir section.lst** aparece el siguiente listado:

```
DSP Fixed Point COFF Assembler Version 6.10     Fri Jan 16 10:55:09 1998
 Copyright (c) 1987-1990  Texas Instruments Incorporated

                                                                PAGE    1

 1                ***********************************************
 2                **  Ensamblar los datos en la seccion .data  **
 3                ***********************************************
 4 0000                   .data
 5 0000 0011      coef    .word          011h, 022h, 033h
   0001 0022
   0002 0033
 6                ***********************************************
 7                **  Reservar el espacio para dos variables   **
 8                ***********************************************
 9 0000                   .bss           var2 ,1
10 0001                   .bss           buffer, 10
11                ***********************************************
12                **  Todavia ensamblar a la seccion de datos **
13                ***********************************************
14 0003 0123      ptr     .word          0123h
15                ***********************************************
16                **  Ensamblar los codigos en seccion .text   **
17                ***********************************************
18 0000                   .text
19 0000 200f      add:    lac            0Fh
20 0001 d003      aloop:  sblk           1
   0002 0001
21 0003 f280              blez           aloop
   0004 0001
22 0005 6000              sacl           var2, 0
23                ************************************************
24                ** Ensamblar otros datos en la seccion de datos **
25                ************************************************
26 0004                  .data
27 0004 00aa      ivals   .word          0aah, 0bbh
   0005 00bb
28                ***********************************************
29                **   Se define la nueva seccion "newvars"    **
30                ***********************************************
31 0000           var1    .usect         "newvars", 1
```

```
32 0001        inbuf   .usect        "newvars", 7
33             ************************************************
34             **  Ensamblar mas codigos en seccion .text    **
35             ************************************************
36 0006                .text
37 0006 ccff           add           #00ffh

No Errors,  No Warnings
```

Desde el archivo section.lst podemos ver que la sección .text contiene siete palabras en el código de objeto; .data contiene cinco palabras; .bss reserva once palabras en la memoria para variables; y la sección que se llama newvars reserva en la memoria el espacio para ocho palabras. Esta sección fue creada mediante la directiva .usect.

En la segunda columna de la figura 1.2 el código está ensamblado en varias secciones y en la tercera las directivas son las que generan las secciones. En la misma figura se muestra el código section.obj de este ejemplo. El listado se divide en cuatro columnas. En la primera columna son líneas numeradas por el contador de líneas. En la segunda se crean las direcciones de las secciones. Cada sección tiene su propio contador de la sección. La tercera columna es el código y la cuarta es el original del programa en ensamblador.

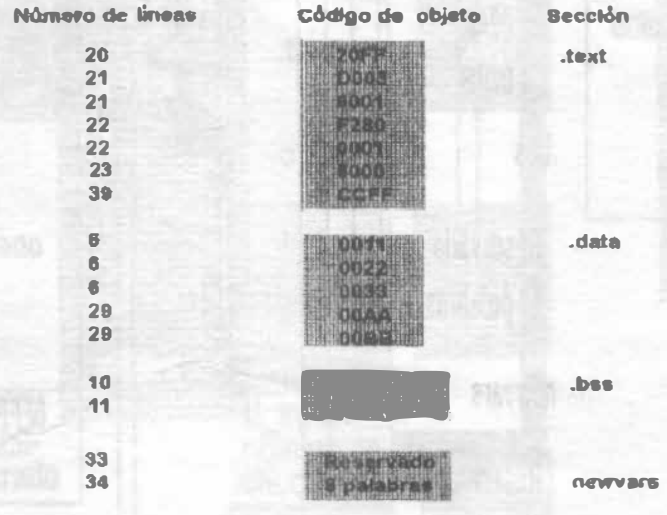


Figura 1.2. Código section.obj del ejemplo 3

En la figura 1.3 se muestra donde están distribuidas las secciones en la memoria RAM. Los códigos de las secciones del texto y los datos están repartidos en la memoria del programa y el espacio total para 19 palabras es reservado en la memoria de datos RAM. En la figura 1.4 se muestra cómo se reparten las secciones si con el ligador (dsplnk) juntamos dos archivos, por ejemplo, arch1.obj y arch2.obj.

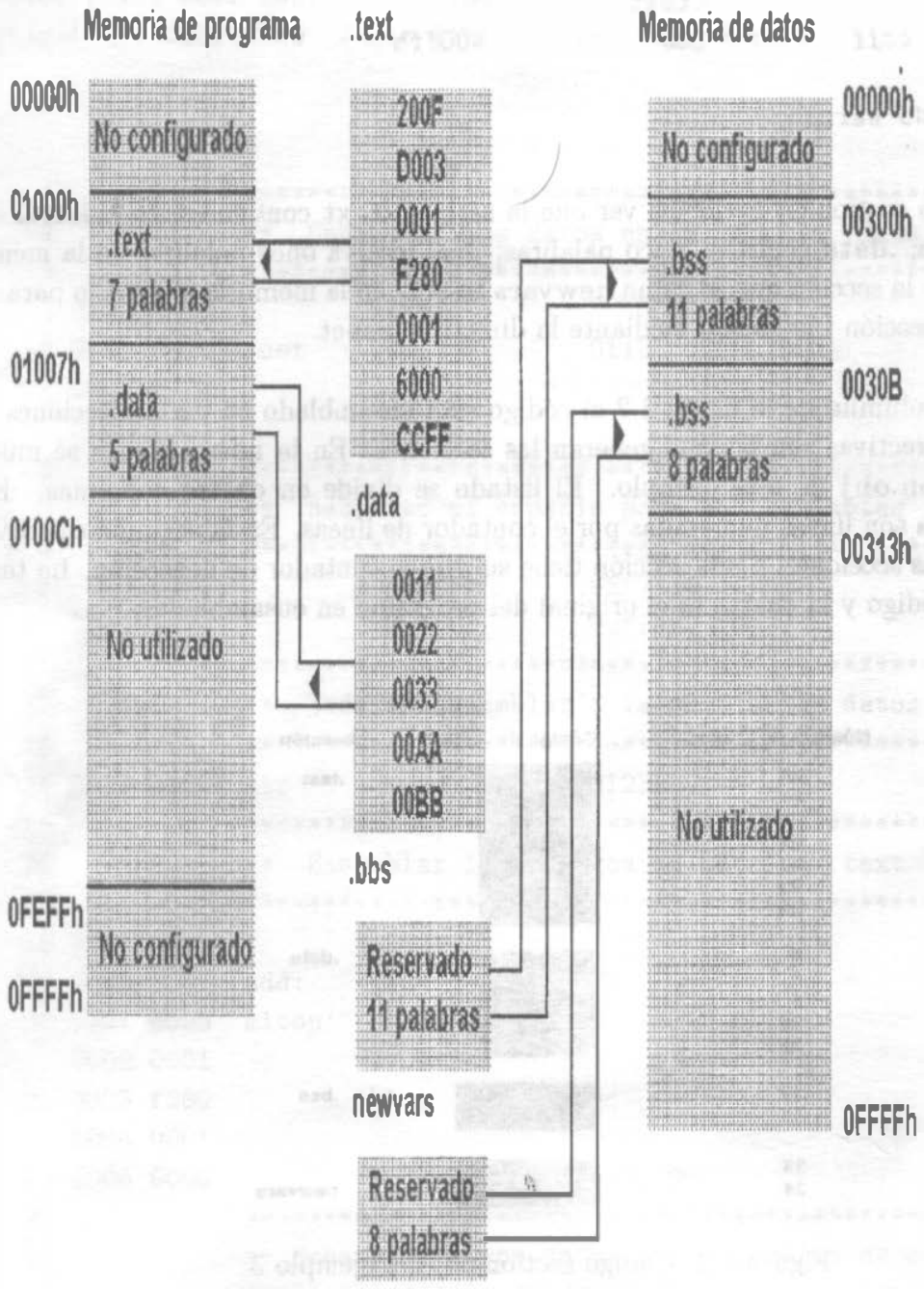


Figura 1.3. Distribución de las secciones en la memoria RAM

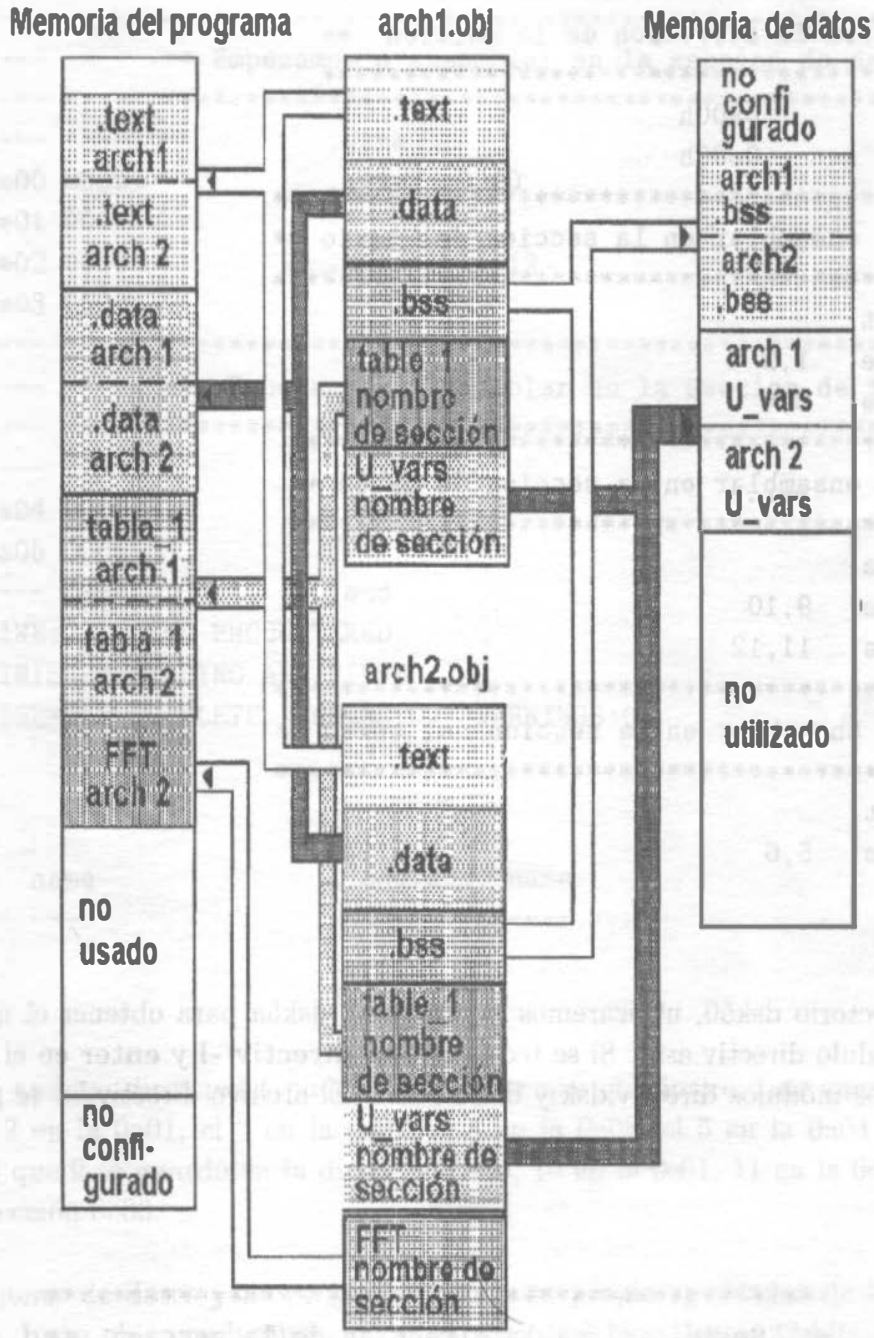


Figura 1.4. Distribución de las secciones de dos archivos en RAM

Ejemplo 4

Escriba el siguiente programa en el archivo directiv.asm.

```
************************************************
**  Se inicializa la direccion de la seccion  **
************************************************
          .ps         0a00h
          .ds         0e00h
************************************************
** Empezamos a ensamblar en la seccion del texto **
************************************************
          .text
          .byte   1,2
          .byte   3,4
************************************************
** Empezamos a ensamblar en la seccion de datos **
************************************************
          .data
          .byte   9,10
          .byte   11,12
************************************************
** Empezamos a ensamblar en la seccion del texto **
************************************************
          .text
          .byte   5,6
          .end
```

Ahora, en el directorio dsk50, utilizaremos el programa dsk5a para obtener el módulo directiv.dsk del módulo directiv.asm. Si se teclea **dsk5a directiv -l** y **enter** en el directorio dsk50, se crean los módulos directiv.dsk y directiv.lst. El archivo directiv.lst se presenta a continuación.

```
00001 ---- ----  ************************************************
00002 ---- ----  **  Se inicializa la direccion de la seccion  **
00003 ---- ----  ************************************************
00004 ---- 0a00            .ps         0a00h
00005 ---- 0e00            .ds         0e00h
00006 ---- ----  ************************************************
00007 ---- ----  ** Empezamos a ensamblar en la seccion del texto **
00008 ---- ----  ************************************************
```

```
00009 ---- ----            .text
00010 0a00 0001            .byte   1,2
      0a01 0002
00011 0a02 0003            .byte   3,4
      0a03 0004
00012 ---- ----  ************************************************
00013 ---- ----  ** Empezamos a ensamblar en la seccion de datos **
00014 ---- ----  ************************************************
00015 ---- ----            .data
00016 0e00 0009            .byte   9,10
      0e01 000a
00017 0e02 000b            .byte   11,12
      0e03 000c
00018 ---- ----  ************************************************
00019 ---- ----  **  Empezamos a ensamblar en la seccion de texto **
00020 ---- ----  ************************************************
00021 ---- ----            .text
00022 0a04 0005            .byte   5,6
      0a05 0006
00023 ---- ----            .end
>>>>> LINE:23  .END ENCOUNTERED
>>>>> FINISHED READING ALL FILES
>>>>> ASSEMBLY COMPLETE: ERRORS:0   WARNINGS:0


SYMBOLS
address  name                address  name
--------- ----               --------- ----
```

Desde el archivo directiv.lst podemos apreciar que el número 1 se guardó en la dirección 0a00, el 2 en la 0a01, el 3 en la 0a02, el 4 en la 0a03, el 5 en la 0a04 y el 6 en la 0a05, mientras que 9 se guardó en la dirección 0e00, 10 en la 0e01, 11 en la 0e02 y el número 12 en la dirección 0e03.

Las secciones de datos y de programa tienen su propio apuntador de las direcciones. La directiva **.byte** ubica en la dirección el número de 8 bits; **.long**, 32 bits; y **.float**, 32 bits.

1.1.4 Directivas .copy e .include

Las directivas **.copy** e **.include** dicen al ensamblador que tiene que leer las secciones del programa desde diferentes archivos. La sintaxis es **.copy " nombre de archivo"** o **.include "nombre de archivo"**. En el siguiente ejemplo se muestra cómo se utilizan esas directivas.

Ejemplo 5

Escriba el siguiente programa en el archivo fuente.asm.

```
.space    10h
.include  "byte.asm"
.space    20h
.end
```

Escriba el siguiente programa en el archivo fuente.asm.

```
.byte     'a',0ah, 32
.include  "word.asm"
.byte     11, 12, 13
```

Escriba el siguiente programa en el archivo fuente.asm.

```
.word     0abcdh, 56
```

Si tecleamos **dsk5a fuente -l**, obtenemos el siguiente listado:

```
00001 0000 0010          .space      10h
00002 ---- ----          .include    "byte.asm"
*****************************************
*     OPENING INCLUDE FILE byte.asm
*****************************************
00001 0001 00ff          .byte       'a', 0ah, 32
      0002 000a
      0003 0020
00002 ---- ----          .include    "word.asm"
*****************************************
*     OPENING INCLUDE FILE word.asm
*****************************************
00001 0004 abcd         .word   0abcdh, 56
      0005 0038
>>>>> FINISHED READING ALL FILES
*****************************************
*     CLOSING FILE word.asm
*****************************************
00003 0006 000b          .byte       11, 12 ,13
```

```
      0007 000c
>>>>> FINISHED READING ALL FILES
*****************************************
*     CLOSING FILE byte.asm
*****************************************
00003 0008 0020          .space      20h
00004 ---- ----          .end
>>>>> LINE:4   .END ENCOUNTERED
>>>>> FINISHED READING ALL FILES
>>>>> ASSEMBLY COMPLETE: ERRORS:0   WARNINGS:0


SYMBOLS
address  name                address  name
-------- ----                -------- ----
```

En el listado podemos observar que **a** se convierte en código de la letra a y se guarda en la dirección 0001h; después en la dirección de la memoria 0002h se guarda el número en la forma hexadecimal ah; y en 0003h se guarda el número hexadecimal 20, que es el número decimal 32h. En la dirección 0004h se guarda la palabra **abcd** y en la dirección 0005 el número 38=56h. En las siguientes direcciones se guardan los números 11=000bh, 12=000ch, 13=000dh y, por último, se reserva el espacio 20h = 32 bits, el cual es para dos palabras de 16 bits.

1.1.5 Programa para generar una señal rampa

En el siguiente ejemplo se muestra el programa que genera una señal rampa.

Ejemplo 6

Escriba el siguiente programa en el archivo rampa.asm.

```
*generador de la rampa
      .MMREGS
      .PS       0080ah
      B        RINT
      .PS       00a00h
      .ENTRY
      LDP       DXR
      LAMM        IMR
      OR        #10h
      SAMM      IMR
LOOP:   ADD       #10
```

```
      SACL      DXR,6
       IDLE
        B          LOOP
RINT:   RETE
        .END
```

Para crear un archivo con extensión .dsk, que es ejecutable para el starter-kit, es necesario primeramente obtener el archivo rampa.obj. Con la instrucción **dsk5a rampa -l** generamos el listado que se ve en el siguiente programa.

```
00001 ---- ----  *generador de la rampa
00002 ---- ----         .MMREGS
00003 ---- 080a         .PS       0080ah
00004 080a 7980         B        RINT
      080b 0000
00005 ---- 0a00         .PS       00a00h
00006 ---- ----         .ENTRY
>>>>> ENTRY POINT SET TO 0a00
00007 0a00 0d21         LDP       DXR
00008 0a01 0804         LAMM        IMR
00009 0a02 bfc0         OR        #10h
      0a03 0010
00010 0a04 8804         SAMM      IMR
00011 0a05 b80a  LOOP:    ADD       #10
00012 0a06 9621         SACL      DXR,6
00013 0a07 be22          IDLE
00014 0a08 7980           B         LOOP
      0a09 0a05
00015 0a0a be3a  RINT:    RETE
00016 ---- ----           .END
>>>>> LINE:16   .END ENCOUNTERED
>>>>> FINISHED READING ALL FILES
>>>>> ASSEMBLY COMPLETE: ERRORS:0   WARNINGS:0

SYMBOLS
address  name                address  name
--------  ----                --------  ----
00000a05 LOOP                00000a0a RINT
```

Ahora se conecta la salida del starter-kit con el osciloscopio, la fuente de 9 volts se conecta a un contacto de 110 volts y el puerto serie con el conector c2 o c1 en la computadora. Con los comandos **DSK5D, LD, RAMPA,ENTER** y **XG** se genera en la pantalla del osciloscopio la señal "rampa".

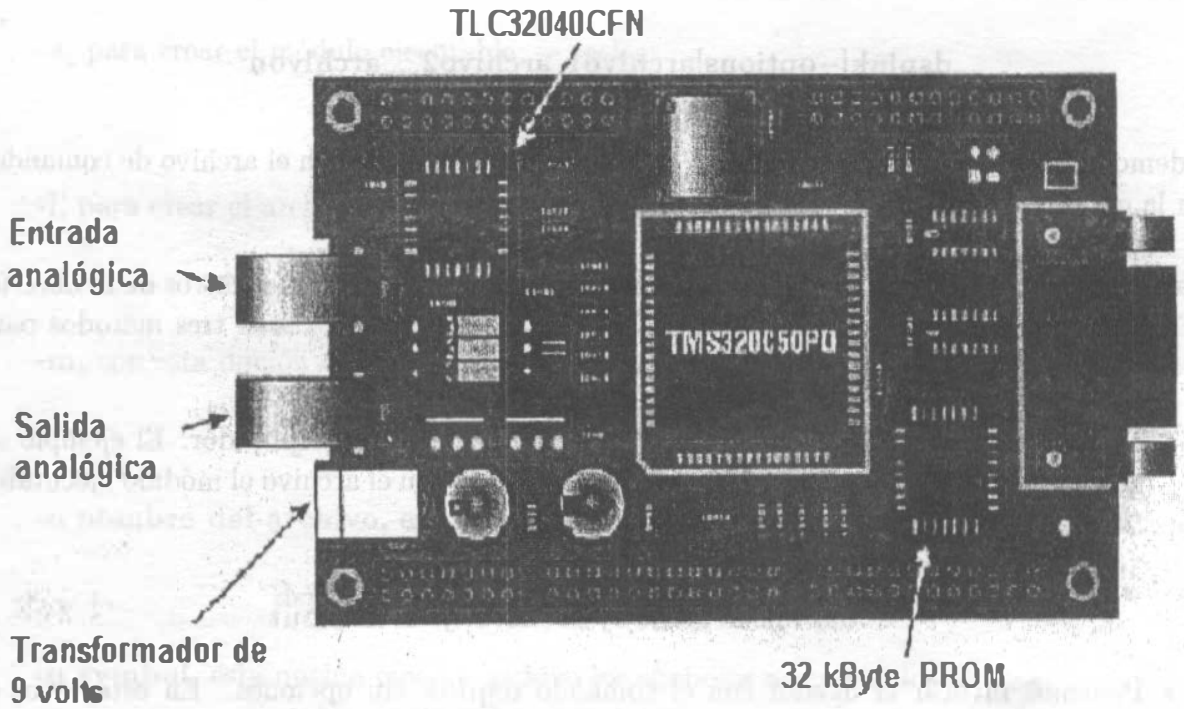


Figura 1.5. Tarjeta del *starter-kit* C50

1.2 Starter-kit de TMS320C50

La tarjeta del starter-kit que se muestra en la figura 1.5 tiene un puerto serial, una entrada y una salida analógicas, y una entrada de voltaje de 9 V. El puerto serial se conecta con el puerto c1 o c2 de la computadora; la salida (OUT) del puerto analógico, con el osciloscopio o con la bocina; y la entrada (IN) del puerto analógico, con el micrófono o con el generador de señales. La tarjeta posee el convertidor analog/digital y digital/analog (A/D y D/A) mediante el chip TCL32040CFN. La interfase analógica TLC32040AIC convierte la señal analógica en digital o la señal digital en analógica.

La tarjeta tiene el oscilador de 50.0 MHz PLE SQ3300. El procesador TMS325C50 es de 16 bits y trabaja con punto flotante. El huésped de interfase lógica (PAL) 22VV10Z tiene el array lógico programable con dos tri-state salidas (74ACT245). Este dispositivo conecta el DSP con la impresora. El regulador de voltaje LM7805 convierte el voltaje alterno de 9V de un transformador a -5V mediante el circuito LT1054. TLC32040 AIC requiere el voltaje negativo de -5V.

1.3 Ligador

En la figura 1.4 se observa cómo trabaja el ligador. El ligador acepta varios tipos de archivos como entrada. El ligador crea un archivo ejecutable COFF, módulo que se puede cargar a

varios módulos, como se ve en la figura 1.4. El ligador se invoca mediante el comando

dsplnk[−options]archivo1, archivo2, ...archivon

Podemos escribir las opciones [-options] en la línea de comandos o en el archivo de comandos con la extensión **.cmd**.

Los archivos pueden ser los que tengan la extensión **.obj** o **.cmd** o los archivos de la librería. La salida del ligador es un archivo que tiene la extensión **.out**. Existen tres métodos para invocar el ligador:

- El ligador se puede invocar desde la línea de comandos del depurador. El ejemplo siguiente liga dos archivos arch1.obj y arch2.obj y crea en el archivo el módulo ejecutable **fft.out**.

dsplnk arch1.obj arch2.obj -o link.out

- Podemos invocar el ligador con el comando **dsplnk** sin opciones. En este caso, el ligador responde con las preguntas:

Objectfile[*.obj*] :
Commandfiles :
Outputfile[*a.out*] :
Options :

En la primera línea se escriben todos los archivos de objeto y se separan con espacios o con una coma. En la línea Command files se escriben los nombres de los archivos de comandos, archivos con la extensión **.cmd**, por ejemplo, **fft.cmd**.

Si en la tercera fila Output file[a.out] respondemos con **enter**, entonces nuestro programa ejecutable se llamará **a.out**. Si queremos que el archivo ejecutable se denomine de manera diferente, por ejemplo, filter.out, es necesario teclear **filter** y **enter**.

- Los comandos para el ligador también los podemos escribir en el archivo de comandos, por ejemplo, en el archivo fft.cmd. Si en el archivo fft.cmd están escritas las siguientes líneas:

−olink.out
*archivo*1*.obj*
*archivo*2*.obj*

Podemos invocar el ligador mediante el comando **fft.cmd** desde la línea del comandos del depurador, en la cual podemos teclear **dsplnk fft.cmd** o también **dsplnk -m link.map fft.cmd file3.obj**. En este caso, el ligador junta los archivos archivo1.obj, archivo2.obj y file3.obj, y crea el archivo ejecutable link.out, así como también el archivo link.map.

Algunas opciones del ligador son las siguientes:

-a, para crear el módulo ejecutable, se teclea:

dsplnk -a archivo1.obj archivo2.obj

-l, para crear el archivo de listado, se teclea:

dsplnk -l archivo1.obj archivo2.obj

-m, con esta opción se crea el archivo con la extensión .map:

dsplnk archivo1.obj archivo2.obj -m map.out

-o nombre del archivo, en el directorio aparece el archivo nombre.out, si se teclea:

dsplnk -o run.out archivo1.obj archivo2.obj

-u symbol, esta opción crea en archivo los símbolos no conocidos en una tabla de símbolos, si se teclea:

dsplnk -u symtab archivo1.obj archivo2.obj rts.lib

1.3.1 Archivos de comandos

Para invocar el ligador, mediante el archivo de comandos desde la línea de comandos del depurador, se utiliza la instrucción:

dsplnk nombre-del-archivo

En el archivo de comandos arch.cmd que está en ASCII, podemos escribir directivas y especificar la memoria y las secciones. Las directivas para memoria definen la configuración de la memoria y las directivas de secciones controlan cómo son creadas las secciones.

En la tabla 1.1 se muestra un ejemplo muy simple, un archivo de los comandos para el ligador, escrito en el archivo link.cmd. El ligador se invoca mediante:

dsplnk link.cmd

También podemos incluir otros parámetros en la línea de comandos del depurador:

dsplnk -r link.cmd c.obj d.obj

Otro ejemplo donde se utilizan las directivas del ligador se presenta en la tabla 1.2. En esta tabla se especifica la memoria y las secciones.

a.obj	/*El archivo primero */
b.obj	/*El archivo segundo */
-o progr.out	/*Opción que especifica archivo de salida*/
-m prog.map	/*Opción que especifica el mapa de la memoria*/

Tabla 1.1. Archivo de los comandos link.cmd

```
a.obj b.obj c.obj          /*Archivos de entrada*/
-o prog.out -m prog.map    /*Las opciones*/
MEMORY
{
RAM origin=100h            length=0100h
ROM origin=01000h          length=0100h
}
SECTIONS
{
.text:      ROM
.data:      ROM
.bss:       RAM
```

Tabla 1.2. Archivo de comandos con directivas de la memoria y secciones

Capítulo 2

Multiplicación de matrices

En este capítulo se presentan tres programas para ejecutarse en el simulador SIM5X: multiplicación de matrices sin lazo y con lazo. El alumno deberá corregir los programas y completar las instrucciones faltantes. Se presenta también un programa para resolver una ecuación cuadrática.

2.1 Multiplicación sin lazo

Modifique el programa lab4.asm para calcular la multiplicación de una matriz **A** por un vector **B** y, mediante el simulador SIM5X, calcule el vector **A.B**.

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 1 & 2 \\ 2 & 1 & 2 & 1 \\ 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \end{bmatrix} \tag{2.1}$$

$$\mathbf{B} = \begin{bmatrix} 10 \\ 11 \\ 12 \\ 13 \end{bmatrix} \tag{2.2}$$

Primeramente, con **dspa −v50 −ls lab4.asm** debe crear el módulo lab4.obj. Para utilizar el ligador es necesario que corrija el archivo lab4.cmd. Con **dsplnk lab4.cmd** cree el módulo lab4.out que es ejecutable para el simulador.

```
lab4.cmd
/*****************************************************************/
/*                                                               */
/* Usage:   dsplnk <obj files...> -o <out file> -m <map file> lab.cmd*/
/*                                                               */
/*****************************************************************/
/*---------------------------------------------------------------*/
/* Especificaciones de la memoria                                */
```

```
/* Bloque B0 esta configurado como la memoria de datos                */
/* (modo de microprocesador) Direccion de la memoria de datos 6h-5Fh */
/* las direcciones 80h--1FFh no estan configuradas.                   */
/*--------------------------------------------------------------------*/
????.obj
-o ?????.out
-m ?????.map
MEMORY
{
    PAGE 0 : VECS     : origin =    0h , length =   020h  /* PROGRAM */
             PROG     : origin =  020h , length =  0FE0h

    PAGE 1 : Regs     : origin =    0h , length =    5fh  /* MMRS    */
             B2       : origin =  060h , length =   020h
             B0       : origin = 0200h , length =  0100h  /* B0      */
             B1       : origin = 0300h , length =  0100h /* B1       */
}

/*--------------------------------------------------------------------*/
/* SECTIONS ALLOCATION                                                */
/*--------------------------------------------------------------------*/
SECTIONS
{
    vectors : { } > VECS      PAGE 0    /* Vector de interrupcion   */
    .text   : { } > PROG      PAGE 0    /* Codigo                   */
    .data   : { } > PROG      PAGE 0    /* Tabla de datos           */
    .bss    : { } > B1        PAGE 1    /* Seccion no inicializada */
}
```

;Lab 4 MULTIPLICACIÓN DE UNA MATRIZ POR EL VECTOR

```
;       SE MULTIPLICA LA MATRIZ 4x4 POR UN VECTOR 4x1

;       C = A * B

            .include   mmr.asm      ;define el mapa de memoria
            .sect      "vectors"
reset:      b           begin       ;Salto a la etiqueta begin
            .text
begin:
      ;INICIALIZAMOS PRIMERAMENTE EL ARREGLO

      ldp    #0             ;Se trabaja en la pagina 0
```

```
      mar    *, AR1          ;se activa registro auxiliar AR1
      lar    AR1, #a         ;AR1 se usa para apuntar la matriz a
      rpt    #tbl_siz
      blpd   #a_val, *+

      splk   #?, INDX        ;carga el registro de index
      lar    AR1, #a         ;AR1 apunta a(1,1)
      lar    AR2, #b         ;AR2 apunta b(1)
      lar    ar3, #c         ;AR3 apunta c(1)
      ;
      ;      PROGRAMA PARA MULTIPLICACION
      ;
      lt     *+, AR2         ;a11 ->T0
      mpy    *+, AR1         ;a11*b1 ->P
      ltp    *+, AR2         ;a11*b1 ->ACC, a12 ->T0
      mpy    *+, AR1         ;a12*b2 ->P
      lta    *+, AR2         ;ACC += a12*b2, a13 ->T0
      ??????????????
      ??????????????
      mpy    *0-, AR1        ;a13*b3 ->P, AR2 -= INDX
      lta    *+, AR3         ;ACC += a13*b3; a21 ->T0, AR3->ARP
      sacl   *+, AR2         ;ACC->c1
*****************************************************
      mpy    *+, AR1         ;a21*b1 ->P
      ltp    *+, AR2         ;a21*b1 ->ACC, a22 ->T0
      mpy    *+, AR1         ;a22*b2 ->P
      lta    *+, AR2         ;ACC += a22*b2, a23 ->T0
      ??????????????
      ??????????????
      mpy    *0-, AR1        ;a23*b3 ->P
      lta    *+, AR3         ;ACC += a23*b3, a31 -> T0, AR3 ->ARP
      sacl   *+, AR2         ;ACC ->c3
*****************************************************
      ????    ??,??
      ????    ??,??
      ????    ??,??
      ????    ??,??
      ????    ??,??
      ????    ??,??
      ????    ??,??
      ????    ??,??
      ????    ??,??
*****************************************************
      mpy     *+, AR1        ;a31*b1 ->P
```

```
      ltp     *+, AR2         ;a31*b1 ->ACC, a32 -> T0
      mpy     *+, AR1         ;a22*b2 ->P
      lta     *+, AR2         ;ACC += a32*b2, a33 -> T0
      ??????????????
      ??????????????
      mpy     *,  AR3         ;a33*b3 ->P
      apac                    ;ACC += a33*b3
      sacl    *               ;ACC -> c3
************************************************

done:  b      $

      .bss    a, ??
      .bss    b, ?
      .bss    c, ?

      ;son los valores de los vectores a,b y c

        .data
a_val:  .word   ???
        .word   ???
        .word   ???
        .word   ???
        .word   ???
        .word   ???
        .word   ???
        .word   ???
        .word   ???
        .word   ???
        .word   ???
        .word   ???
        .word   ???
        ?????   ???
        ?????   ???
        ?????   ???
        ?????   ???
b_val:  .word   ???
        .word   ???
        .word   ???
        .word   ???
c_val:  .word   0deadh
        .word   0deadh
        .word   0deadh
        .word   0deadh
```

```
tbl_siz .set    $-a_val-1
        .end
```

2.2 Multiplicación con lazo

Modifique el programa lab4.asm para multiplicar la matriz **A** por el vector **B**.

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 1 & 2 \\ 2 & 1 & 2 & 1 \\ 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \end{bmatrix} \tag{2.3}$$

$$\mathbf{B} = \begin{bmatrix} 10 \\ 11 \\ 12 \\ 13 \end{bmatrix} \tag{2.4}$$

Mediante el simulador SIM5X, calcule el vector **C= A.B**. Primeramente, es necesario con **dspa −v50 −ls lab4.asm** que cree el módulo lab4.obj y con **dsplnk lab4.cmd** el módulo lab4.out, el cual es ejecutable para el simulador.

```
;       Multiplicar la matriz por un vector
;       usando la repeticion del bloque
;
;       C = A * B
;

        .include  mmr.asm
        .version 50
        .sect      "vectors"
reset:  b       begin
        .text
begin:
        lar       AR1, #a
        mar       *, AR1
        ldp       #0

        rpt       #tbl_siz
        blpd       #a_val, *+
        splk       #?
```

```
        , INDX
        lar      AR1, #a        ;AR1 apunta  a(1,1)
        lar      AR2, #b        ;AR2 apunta  b(1)
        lar      AR3, #c        ;AR3 apunta  c(1)
        splk      #?, BRCR      ;el registro para repeticion
        lt       *+, AR2        ;a11 ->T0
        rptb      loop
        mpy      *+, AR1        ;aX1*b1 ->P
        ltp      *+, AR2        ;aX1*b1 ->ACC, aX2 ->T0
        mpy      *+, AR1        ;aX2*b2 ->P
        lta      *+, AR2        ;ACC += aX2*b2, aX3 ->T0
        ???     ???????          ;??????????????????????
        ???     ???????          ;??????????????????????
        mpy      *0-, AR1       ;aX3*b3 ->P, AR2 -= INDX
        lta      *+, AR3        ;ACC += aX3*b3; a21 ->T0, AR3->ARP
loop:   sacl     *+, AR2        ;ACC->c1

done:   b        $

       .bss      a, ??          ;reserva el espacio en la memoria de datos
       .bss      b, ?
       .bss      c, ?


       ;escriba los valores de la matriz

        .data
a_val:  .word      ???
        .word      ???
        .word      ???
        .word      ???
        .word      ???
        .word      ???
        .word      ???
        .word      ???
        .word      ???
        ?????      ???
        ?????      ???
        ?????      ???
        ?????      ???
        ?????      ???
        ?????      ???
        ?????      ???
```

```
        ; y del vector

b_val:  .word       ???
        .word       ???
        .word       ???
        ?????       ???

c_val:  .word       0deadh
        .word       0deadh
        .word       0deadh
        ?????       ??????

tbl_siz .set        $-a_val-1

        .end
```

2.3 Ecuación cuadrática

Calcule la ecuación (2.5) para 15 valores de x y para los coeficientes a=1, b=2, c=3.

$$y = a.x^2 + b.x + c \tag{2.5}$$

Los valores de x se almacenan en la memoria B2 desde la dirección 0300h y son los siguientes:

$$\{\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ a\ b\ c\ d\ e\ f\ \}$$

Corrija el programa lab5b.cmd para el ligador.

lab5b.cmd

```
/*************************************************************************/
/*                                                                       */
/* Usage:  dsplnk <obj files...> -o <out file> -m <map file> lab.cmd*/
/*                                                                       */
/*************************************************************************/
/*-----------------------------------------------------------------------*/
/*  Especificaciones de la memoria.                                      */
/*  Bloque B0 se configura como la memoria de datos                      */
/*  (modo de microprocesador). Direcciones de la memoria  6h--5Fh        */
/*  direcciones 80h--1FFh no estan configuradas.                         */
/*-----------------------------------------------------------------------*/
?????.obj
-o ?????.out
```

```
-m ?????.map
MEMORY
{
    PAGE 0 :   VECS   : origin =    0h , length =   020h  /* PROGRAM */
               PROG   : origin =  020h , length =  0FE0h

    PAGE 1 :   Regs   : origin =    0h , length =    5fh  /* MMRS    */
               B2     : origin =  060h , length =   0200
               B0     : origin = 0200h , length =  0100h  /* B0      */
               B1     : origin = 0300h , length =  0100h  /* B1      */
}

/*--------------------------------------------------------------------*/
/* UBICACION DE LAS SECCIONES                                         */
/*--------------------------------------------------------------------*/
SECTIONS
{
    vectors : { } > VECS   PAGE 0   /* El vector de interrupcion    */
    .text   : { } > PROG   PAGE 0   /* Codigo                       */
    .data   : { } > PROG   PAGE 0   /* La tabla de los datos        */
    .bss    : { } > B1     PAGE 1   /* Los datos no inicializados   */
}



* Lab 5b - Repeticion de bloques
* En este laboratorio vamos a mostrar el uso de RPT
* repeticion de bloques de instrucciones.
*
*
*
*  Esta rutina evalua repetidamente la ecuacion
*       y = ax^2 + bx + c
*
*  para 16 valores diferentes de x.  Los valores se guardan en la
*  variable y en la memoria.
*
* Los valores x son apuntados por AR1.
* Los valores y son apuntados por AR3.
* Los coeficientes a, b, y c son apuntados por AR2
;
        .mmregs
       .global x,y,TEMP,COEFF,BEGIN
       .text
```

```
BEGIN
                                    ; En el siguiente bloque se copian
                                    ; los valores x de la memoria de
                                    ; programa a la memoria de datos.
      LDP       #0
      MAR       *, AR1
      LAR       AR1,#x
      RPT       #count-1
      BLPD       #x_vals, *+
                                    ; En el siguiente bloque se cargan
                                    ; los coeficientes del programa a
                                    ; la memoria de datos.
      LAR       AR1, #COEFF
      RPT       #2
      BLPD       #c_vals, *+

      ; Los registros auxiliares AR1, AR2 y AR3 apuntan los datos x, a,  y c.

      LAR       AR1,#x              ; AR1 apunta los valores de x
      LAR       AR2,#COEFF          ; AR2 apunta los coeficientes b, a y c
      LAR       AR3,#y              ; AR3 apunta donde guardar coeficientes
      LDP       #0                  ; se utiliza la pagina 0 de memoria
                                    ; inicializa el apuntador AR a AR1
      MAR       *,AR1               ; ARP <- 1

      splk       #0FH,BRCR          ; se carga 15 al registro BRCR
                                    ; aqui empieza el lazo principal
      RPTB       END_LOOP-1         ; for 1=0;i<16;i++
      LAR       AR2,#COEFF          ; se apunta al primer coeficiente
      ZAP                           ; acc = preg = x^2
      SQRA       *+,AR2             ; treg 0=x preg = x^2
      SPL       TEMP                ; salva x^2
      MPY       *+                  ; preg = b*x
      LTA       TEMP                ; treg = x^2   acc = b*x
      MPY       *+                  ; preg = ax^2
      APAC                          ; acc = ax^2 + bx
      ADD       *,0,AR3             ; acc = ax^2 +bx + c
      SACL       *+,0,AR1           ; salva y al AR1 sin corrimiento
END_LOOP
      b      BEGIN
**********************************
* se reserva espacio para los datos*
**********************************
      .bss       x,??
```

```
      .bss      y,??
      .bss       COEFF,?
*************************************************************
* se cargan los datos iniciales en la memoria de programa**
*************************************************************
        .data
x_vals  .word      ?
        .word      ?
        .word      ?
        .word      ?
        .word      ?
        .word      ?
        .word      ?
        .word      ?
        .word      ?
        .word      ?
        .word      ?
        .word      ?
        .word      ?
        .word      ?
        .word      ?
        .word      ?
        .word      ?
        .word      ?
c_vals  .word      ?
        .word      ?
        .word      ?
TEMP    .set       63h             ; se almacena en TEMP el numero 63h
count   .set       16              ; se carga al count el numero 16
        .end
```

Capítulo 3

Códigos e inicialización del microcontrolador

Una herramienta muy útil para evaluar el programa del microcontrolador es a través del simulador del TMS320C25, lo cual se realiza con el programa SIM25.EXE. En este capítulo se analizan algunos programas y registros internos del microcontrolador, y se observa instrucción tras instrucción, cómo se modifican. Los programas para el microcontrolador TMS320C25 y TMS320C26 se compilan con el programa XAS25.EXE. El alumno compilará los programas presentados y observará los registros del microcontrolador en el simulador. El simulador utiliza como señal de entrada un archivo de datos y de igual forma envía la salida de datos a un archivo. Éstos pueden graficarse y observarse con el programa VIEW.EXE.

3.1 Códigos de las instrucciones del salto

Obtenga el código para cada una de las instrucciones y apúntelas en una hoja aparte. Escriba el programa en el archivo **codigo.asm** y compílelo con XAS25.EXE. Este programa ejecutable genera dos archivos, **codigo.mpo** y **codigo.lst**. Del archivo **codigo.mpo** usted puede comparar los códigos que escribió con los que generó el compilador. Ambos deben ser iguales.

```
       AORG 0020
       ZAC                ;ANULACION DEL ACUMULADOR
* EMPIEZA EL PROGRAMA EN ENSAMBLADOR
*
    BBZ  68
    B    12
    BACC
    BANZ 46
    BBNZ 28
    BBZ  38
    BC   26
    BGEZ 36
```

```
    BGZ  18
    BIOZ 25
    ADD *+,7
    ADD *0-,8
    RPTK 99
    ADLK 16384,2
    BNV 315
    ADD  100,4
```

3.2 Inicialización del microcontrolador

Escriba el siguiente programa en el archivo inic.asm y mediante el simulador verifique cómo se cambian los registros del simulador.

```
      TITL 'PROCESOR'
      IDT  'EJEMPLO'
      DEF   RESET,INT0,INT1,INT2
      DEF   TINT, RINT,XINT,USER
      REF   ISR0,ISR1,ISR2
      REF   TIME,RCV,XMT,PROC
*
*  SE ESPECIFICAN RESET Y EL VECTOR DE INTERRUPCION
*  BRINCOS PARA INTERRUPCION EXTERNA E INTERNA

*
      AORG   >0000
RESET  B   INIT
INT0   B   ISR0
INT1   B   ISR1
INT2   B   ISR2
      AORG  >0018
TINT   B   TIME
RINT   B   RCV
XINT   B   XMT
USER   B   PROC
*
INIT   ROVM
       LDPK  0
       LARP  7
       LACK  >3F
       SACL  4
*
```

```
*SE INICIALIZA LA MEMORIA DE DATOS INTERNA
*
      ZAC
      LARK   AR7,>60
      RPTK   3
      SACL   *+
*
      LRLK   AR7,>3
      RPTK   255
      SACL   *+
*
      LRLK AR7,>300
      RPTK 3
      SACL *+
*
      LALK 520
      SACL 3

      EINT
```

3.3 Registros DXR, STO y ST1

Escriba el siguiente programa para el simulador TMS320C25 en el archivo **reg-dxr.asm** y compile el programa mediante XAS25. Con el simulador SIM25 corra el programa **reg-dxr.asm** paso a paso y observe cómo se cambia el ACC y el registro DXR, así como qué bits contienen los registros de estado ST0 y ST1.

```
       AORG >0000
RESET  B    INIT
*
       AORG >0020
*
* INICIO DEL MICROCONTROLADOR
*
INIT   LDPK 0             ;Trabaja con bandera cero
       ZAC                ;Anulacion del acumulador
       LARP AR2           ;Actualiza registro auxiliar AR2
       LRLK AR2,>0060     ;Inicializa bloque  B2
       RPTK 3             ;Repite la instruccion que sigue 4 veces
       SACL *+            ;Anulacion de 4 direcciones en el bloque  B2
       LRLK AR2,>0060     ;Inicializa el bloque B2 en la direccion 0060
       RPTK 3             ;Repite la instruccion que sigue 4 veces
```

```
       BLKP COEF,*+      ;Transferencia de los coeficientes al bloque B2
*      SOVM
*
* DECLARACION DE LAS VARIABLES
*
A01     EQU  >0060        ;\
A11     EQU  >0061        ; \
A21     EQU  >0062        ;  > Coeficientes del filtro
A02     EQU  >0063        ; /
IMR     EQU  >0004        ;/
DXR     EQU  >0001

* EMPIEZA EL PROGRAMA EN ENSAMBLADOR*
OTROX  ADD A01,15      ;A la parte alta de ACC sume el numero en A01
       SACH DXR        ;En el registro DXR se carga el contenido de ACC
       B   OTROX       ;Se salta a la etiqueta OTROX
*
* DEFINICION DE LA CONSTANTE A01,A02 ETC
*
COEF   DATA 3276,3276,3276,3276
```

Capítulo 4

Generador de señales

Los generadores de señales tienen una importante aplicación en comunicaciones, telefonía, radar, etc. En este capítulo los alumnos generarán señales triangulares y senoidales a través de filtros digitales tipo FIR e IIR.

4.1 Diente de sierra con filtro FIR

Escriba el siguiente programa para el simulador TMS320C25 en el archivo **sierafir.asm** y compile el programa mediante XAS25. Utilice el simulador SIM25 para correr el programa **sierafir.mpo** y con el programa VIEW verifique si la respuesta obtenida está correcta. La estructura del circuito se observa en la figura 4.1

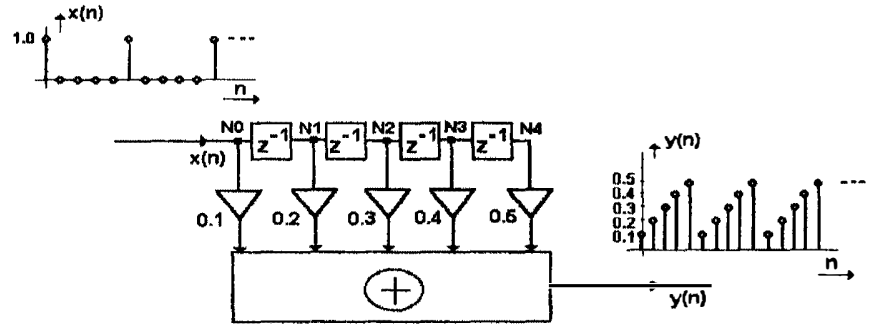


Figura 4.1. Generador de diente de sierra

```
*     FILTRO FIR COMO DIENTE DE SIERRA    *
*****************************************
*                                         *
*                 -1      -2     -3     -4 *
* H(z)  =  h0+h1*z +h2*z  +h3*z +h4*z     *
*                                         *
*****************************************
*
```

```
*
       AORG >0000
RESET  B    INIT
*
*
       AORG >0020
*
* INICIO DEL MICROCONTROLADOR
*
INIT
       LDPK 0             ;Trabaja con bandera cero
       ZAC                ;Anulacion del acumulador
       LARP AR2           ;Actualiza registro auxiliar AR2
       LRLK AR2,>0060     ;Inicializa el bloque B2
       RPTK 5             ;Repite la instruccion que sigue 6 veces
       SACL *+            ;Anulacion del bloque B2
       LRLK AR2,>0060     ;Inicializa el bloque B2 en la direccion 0060
       RPTK 4             ;Repite la instruccion que sigue 5 veces
       BLKP COEF,*+       ;Transferencia de los coeficientes al bloque B2
*
* DECLARACION DE LAS VARIABLES
*
H0     EQU  >0060         ;\
H1     EQU  >0061         ; \
H2     EQU  >0062         ;  > Coeficientes del filtro
H3     EQU  >0063         ; /
H4     EQU  >0064         ;/
YN     EQU  >0065         ;    Senal de salida
XN     EQU  >0066         ;\
N1     EQU  >0067         ;  \
N2     EQU  >0068         ;   >Variables internas del filtro
N3     EQU  >0069         ;  /
N4     EQU  >006A         ;/
*
* EMPIEZA EL PROGRAMA EN ENSAMBLADOR
*
OTRO   IN   XN,PA1        ;Entrada del muestreo
       LT   N4
       MPY  H4
       LTD  N3
       MPY  H3
       LTD  N2
       MPY  H2
       LTD  N1
```

```
        MPY  H1
        LTD  XN
        MPY  H0
        APAC
        SACH YN,1
        ZAC
        OUT  YN,PA2
        B    OTRO
*
* DEFINICION DE LA CONSTANTE DEL FILTRO
*
COEF    DATA 3276,6553,9830,13106,1638,19660
        END
```

4.2 Diente de sierra con filtro IIR

Escriba el programa para el filtro de la figura 4.2 en el archivo **sieraiir.asm**, compilarlo y simularlo con XAS25 y SIM25, respectivamente.

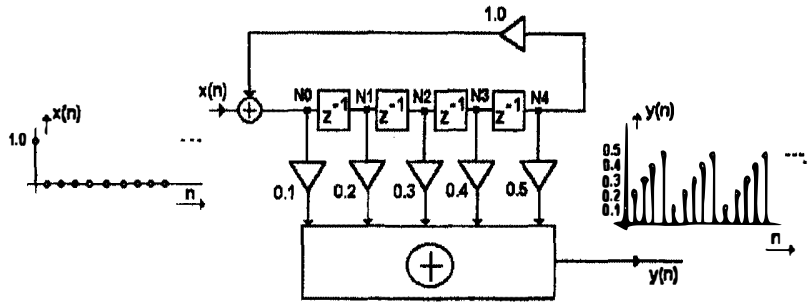


Figura 4.2. Generador de diente de sierra

```
*    FILTRO IIR COMO DIENTE DE SIERRA   *
*****************************************
*                                       *
*                -1     -2    -3     -4 *
*          h0+h1*z +h2*z  +h3*z +h4*z   *
*  H(Z)   ---------------------------   *
*                    -4                 *
*                1 - z                  *
*****************************************
*
        AORG >0000
*
```

```
RESET  B    INIT
*
       AORG >0020
*
* INICIO DEL MICROCONTROLADOR
*
INIT
       LDPK 0            ;Trabaja con bandera cero
       ZAC               ;Anulacion del acumulador
       LARP AR2          ;Actualiza registro auxiliar AR2
       LRLK AR2,>0060    ;Inicializa el bloque B2
       RPTK 4            ;Repite la instruccion que sigue 6 veces
       SACL *+           ;Anulacion del bloque B2
       LRLK AR2,>0060    ;Inicializa el bloque B2 en la direccion 0060
       RPTK 4            ;Repite la instruccion que sigue 5 veces
       BLKP COEF,*+      ;Transferencia de los coeficientes al bloque B2
*
*
* DECLARACION DE LAS VARIABLES
*
*
H0     EQU  >0060        ;\
H1     EQU  >0061        ; \
H2     EQU  >0062        ;  > Coeficientes del filtro
H3     EQU  >0063        ; /
A4     EQU  >0064        ;/
YN     EQU  >0065        ;    Senal de salida
XN     EQU  >0066        ;\
N      EQU  >0067        ; \
N1     EQU  >0068        ;  \
N2     EQU  >0069        ;   > Variables internas del filtro
N3     EQU  >006A        ;  /
N4     EQU  >006B        ; /
*
* EMPIEZA EL PROGRAMA EN ENSAMBLADOR
*
OTRO   IN   XN,PA1       ;Entrada de las muestras
       LT   N4
       MPY  A4
       APAC
       ADD  XN,15
       SACH  N,1
       ZAC
       LT  N3
```

```
        MPY   H3
        LTA   N2
        MPY   H2
        LTA   N1
        MPY   H1
        LTA   N
        MPY   H0
        APAC
        SACH YN,1
        LTD   N3
        LTD   N2
        LTD   N1
        LTD   N
        ZAC
        MPYK  0
        OUT   YN,PA2
        B     OTRO
*
* DEFINICION DE LA CONSTANTE DEL FILTRO
*
COEF    DATA 3277,6553,9830,13106,32767
        END
```

4.3 Generador de cosenos

La función de transferencia del circuito de la figura 4.3 es

$$H_1(z) = \frac{A_0 + A_1 z^{-1}}{1 - B_1 z^{-1} - B_2 z^{-2}} \tag{4.1}$$

Si elegimos los coeficientes

$$A_0 = 1 \qquad A_1 = cos(\omega_0.T) \qquad B_1 = 2 * cos(\omega_0.T) \qquad B_2 = 1$$

entonces los polos de la función de transferencia están ubicados en el círculo unitario y el filtro se comporta como un generador de cosenos.

Escriba en el archivo **cos.asm** el siguiente programa para compilarlo mediante XAS25 y simularlo con SIM25. Utilizando el programa VIEW verifique los resultados.

```
**********************
*GENERADOR DE COSENOS*
**********************
      AORG   >0000
*
```

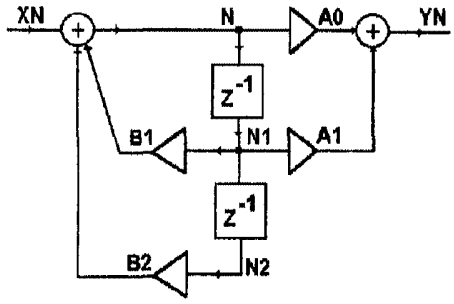


Figura 4.3. Generador de cosenos

```
RESET B      INIT
*
      AORG   >0020
*
*SE INICIALIZA LA MEMORIA DE DATOS INTERNA
*
*
INIT  SOVM
      LDPK 0
      ZAC
      LARP AR2
      LRLK AR2,>0060
      RPTK 4
      SACL *+
      LRLK AR2,>0060
      RPTK 4
      BLKP COEF,*+
*
*DECLARACION DE LOS COEFICIENTES
*
B0    EQU  >0060
B1    EQU  >0061
A1    EQU  >0062
A2    EQU  >0063
XN    EQU  >0064
ONE   EQU  >0065
YN    EQU  >0066
N     EQU  >0067
N1    EQU  >0068
N2    EQU  >0069
*
```

```
* PROGRAMA PARA GENERADOR COSENOIDAL
*
        B       ET2
ET1     ZAC                     ;0-->ACC
        SACH    XN,1
ET2     ZAC
        LT      N1
        MPY     A1
        MPYA    A1
        LTA     N2
        MPY     A2
        APAC
        ADD     XN,15
        SACH    N,1
        LTD     N1
        ZAC
        LAC     ONE,14
        MPY     B1
        LTD     N
        MPY     B0
        APAC
        SACH    YN,1
        OUT     YN,PA2
        ZAC
        LAC     ONE,14
        B       ET1
*
* DATOS EN EL ORDEN B0,B1,A1,A2
*
COEF  DATA >7FFF,>8644,>79BC,>8000,>0CCC
      END
```

4.4 Generador de senos

La salida de la estructura en la figura 4.4 es la respuesta $y(n) = sen(\omega_0.n.T)$. Calculando la transformada Z de $sen(\omega_0.n.T)$, obtenemos:

$$H(z) = \frac{z^{-1}sen(\omega_0.T)}{1 - 2z^{-1}cos(\omega_0.T) + z^{-2}} = \frac{A_1 z^{-1}}{1 + B_1 z^{-1} + B_2 z^{-2}} \tag{4.2}$$

La función de transferencia se realiza mediante el circuito de la figura 4.4.

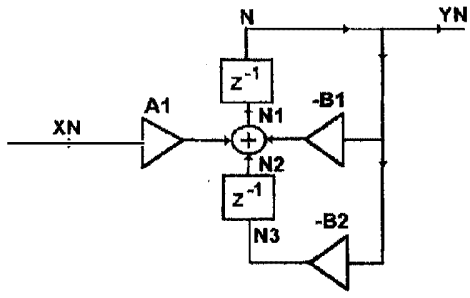


Figura 4.4. Generador de senos

Escriba el programa para el generador de senos de la figura 4.4 con los siguientes valores:

$$f_0 = 1000\ Hz \qquad f_m = 10000\ Hz$$

Escriba en el archivo **sen.asm** el siguiente programa. Compílelo con XAS25 y simúlelo con SIM25. Mediante el programa **view** grafique la respuesta.

```
*********************
*GENERADOR DE SENOS*
*********************
*
       AORG >0000
RESET  B    INIT
*
       AORG >0020
*
* INICIO DEL MICROCONTROLADOR
*
INIT   SOVM
       LDPK 0              ;Trabaja con bandera cero
       ZAC                 ;Anulacion del acumulador
       LARP AR2            ;Actualiza registro auxiliar AR2
       LRLK AR2,>0060      ;Inicializa bloque B2
       RPTK 31             ;Repite la instruccion que sigue 6 veces
       SACL *+             ;Anulacion del bloque B2
       LRLK AR2,>0060      ;Inicializa del bloque B2 en la direccion 0060
       RPTK 2              ;Repite la instruccion que sigue 5 veces
       BLKP COEF,*+        ;Transferencia de los coeficientes al bloque B2
       LACK 1
       SACL ONE
*
```

```
*
* DECLARACION DE LAS VARIABLES
*
B1      EQU  >0060          ; \
A1      EQU  >0061          ;  > Coeficientes del filtro
A2      EQU  >0062          ; /
YN      EQU  >0063          ;    Senal de salida
XN      EQU  >0064          ;\
N1      EQU  >0065          ; > Variables internas del filtro
N2      EQU  >0066
N3      EQU  >0067
N4      EQU  >0068
ONE     EQU  >0069
*
* EMPIEZA EL PROGRAMA EN ENSAMBLADOR
*
OTRO    IN   XN,PA1         ;Entrada del muestreo
        LT   XN             ;<N>-->ACC
        MPY  B1             ;<ACC>-->YN
        LTA  N4             ;<YN>-->PA2
        MPY  A1             ;0-->ACC
        MPYA A1             ;<XN>-->TR
        APAC                ;<A1*XN>-->PR
        ADD  N2,15          ;<N>-->TR; <A1*XN>-->ACC
        SACH N3,1           ;<N*B1>-->PR
        ZAC                 ;<A1*XN+B1*N>-->ACC
        LAC  ONE,14         ;<A1*XN+B1*N+N2>-->ACC
        MPY  A2             ;<A1*XN+B1*N+N2>-->N1
        APAC                ;0-->ACC
        SACH N1,1           ;<B2*N>-->PR
        OUT  N4,PA2         ;<N1>-->TR; <N1>-->N; <B2*N>-->ACC
        LTD  N3             ;<N3>-->N2
        LTD  N1
        ZAC                 ;0-->ACC
        LAC  ONE,14
        MPYK 0              ;0-->PR
        B    OTRO           ;Salto a la etiqueta OTRO
*
* DEFINICION DE LA CONSTANTE DEL FILTRO
*
COEF    DATA >4BC3,>678D,>8000
        END
```

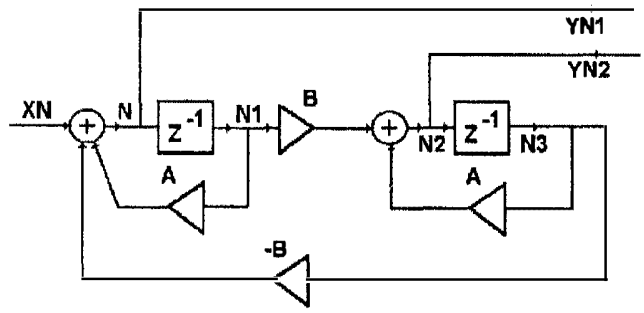


Figura 4.5. Generador de senos y cosenos

4.5 Generador de senos y cosenos

Escriba en el archivo **cossen.asm** el siguiente programa. Compílelo mediante el programa XAS25 y simúlelo con SIM25. Utilizando el programa VIEW verifique los resultados.

```
*******************************
*GENERADOR DE COSENOS Y SENOS*
*******************************

      AORG   >0000
*
RESET B      INIT
*
      AORG   >0020
*
*SE INICIALIZA LA MEMORIA DE DATOS INTERNA
*
*
INIT  SOVM
      LDPK 0
      ZAC
      LARP AR2
      LRLK AR2,>0060
      RPTK 4
      SACL *+
      LRLK AR2,>0060
      RPTK 2
      BLKP COEF,*+
      LAC  1
      SACL ONE
      ZAC
```

```
*
*DECLARACION DE LOS COEFICIENTES
*
B     EQU  >0060
A     EQU  >0061
XN    EQU  >0062
ONE   EQU  >0063
YN    EQU  >0064
N     EQU  >0065
N1    EQU  >0066
N2    EQU  >0067
N3    EQU  >0068
*
* PROGRAMA PARA GENERADOR DE COSENO Y SENOS
*
       B  ET2
ET1    ZAC
       SACH  XN
ET2    LT N3              ; <N3>->TR
       MPY  B             ; <N3*B>->PR
       LTA N1             ; <N1>->TR   y <N3*B>->ACC
       NEG                ; -<N3*B>->ACC
       MPY A              ; <N1*A>->PR
       APAC               ; <N1*A-N3*B>->ACC
       ADD XN,15          ; <XN+N1*A-N3*B>->ACC
       SACH N,1           ; <XN+N1*A-N3*B>->N
       OUT N,PA3          ; <N>-->PA3
       ZAC                ; 0->ACC
       LAC ONE,14         ; bit de redondeo a ACC
       MPY  B             ; <N1*B1>->PR
       LTA N3             ; <N3>->TR y <N1*B1>->ACC
       MPY A              ; <A*N3>->PR
       APAC               ; <A*N3+N1*B1>->ACC
       SACH N2,1          ; <A*N3+N1*B1>->N2
       LTD N
       LTD N2
       OUT N2,PA2         ; N2->PA2
       ZAC
       LAC ONE,14         ;Bit de redondeo al bit 14 de ACC
       MPYK  0            ;0->P
       B ET1
*
*DATOS EN EL ORDEN B,A
*
```

```
COEF  DATA 13804,30041,6553
      END
```

4.6 Generador de tonos

En esta sección se realizará un generador de tonos para la telefonía DTMF (*Dual-Tone-Multi-Frequency*). El generador de tonos para el teclado, en la figura 4.6, se realiza con dos generadores de cosenos en paralelo, que se muestran en la figura 4.7. Si se teclea el número 5, el oscilador produce la suma de los tonos con las frecuencias 770 [Hz] y 1336 [Hz].

$$\omega_{0L} = \frac{2 * pi * 770}{8000} = 0.60475$$

$$\omega_{0H} = \frac{2 * pi * 1336}{8000} = 1.049292$$

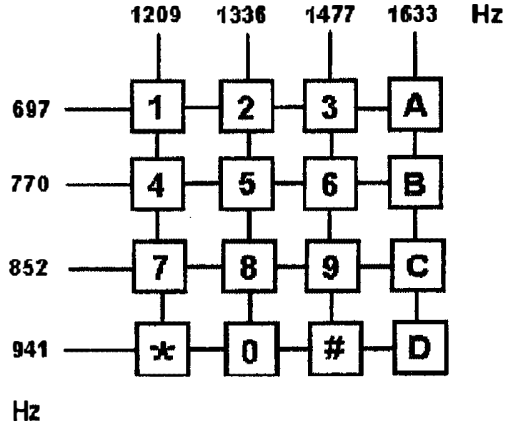


Figura 4.6. Teclado de los números por tonos

Los coeficientes de los multiplicadores en el circuito de la figura 4.7 toman los valores:

$$\begin{aligned}
2.cos(\omega_{0L}) &= 1.6452885 \\
2.cos(\omega_{0H}) &= 0.99637 = 32648_{Q_{15}} \\
cos\omega_L = cos(0.60475) &= 0.822644 = 26955_{Q_{15}} \\
cos\omega_H = cos(1.04929) &= 0.498185 = 16324_{Q_{15}}
\end{aligned}$$

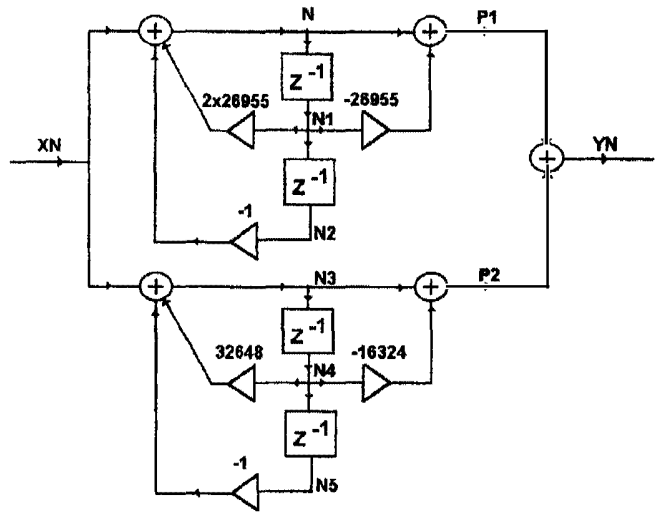


Figura 4.7. Generador de los números por tonos

```
      AORG   >0000
*
RESET B      INIT
*
      AORG   >0020
*
*SE INICIALIZA LA MEMORIA DE DATOS INTERNA
*
*
INIT  SOVM
      LDPK 0
      ZAC
      LARP AR2
      LRLK AR2,>0060
      RPTK 31
      SACL *+
      LRLK AR2,>0060
      RPTK 6
      BLKP COEF,*+
*
*DECLARACION DE LOS COEFICIENTES
*
A01    EQU  >0060
A11    EQU  >0061
B01    EQU  >0062
```

```
B02     EQU  >0063
B11     EQU  >0064
B12     EQU  >0065
XN      EQU  >0066
YN      EQU  >0067
ONE     EQU  >0068
N       EQU  >0069
N1      EQU  >006A
N2      EQU  >006B
N3      EQU  >006C
N4      EQU  >006D
N5      EQU  >006E
P1      EQU  >006F
P2      EQU  >0070
*
*PROGRAMA PARA GENERADOR COSENOIDAL
*
        B       ET2
ET1     ZAC                     ;0-->ACC
        SACH    XN,1
ET2     ZAC
        LT      N1
        MPY     B01
        MPYA    B01
        LTA     N2
        MPY     B02
        APAC
        ADD     XN,15
        SACH    N,1
        ZAC
        LTD     N1
        MPY     A01
        APAC
        ADD     N,15
        SACH    P1,1
        OUT     P1,PA2
        LTD     N1
        LTD     N
        ZAC
        LT      N4
        MPY     B11
        LTA     N5
        MPY     B12
        APAC
```

```
          ADD     XN,15
          SACH    N3,1
          ZAC
          LT      N4
          MPY     A11
          APAC
          ADD     N3,15
          SACH    P2,1
          OUT     P2,PA3
          LTD     N4
          LTD     N3
          ZAC
          LAC     P1,15
          ADD     P2,15
          SACH    YN,1
          OUT     YN,PA4
          ZAC
          LAC     ONE,14
          B       ET1
*
*DATOS EN EL ORDEN B0,B1,A1,A2
*
COEF  DATA -26955,-16324,26955,-32768
      DATA 32648,-32768,3267
      END
```

Capítulo 5

Filtros RC

En este capítulo, el alumno implementará dos sencillos filtros RC pasa-altas y pasa-bajas en el simulador. Para obtener la transformada de Fourier de la respuesta al impulso se utilizan los programas TOFRACT.EXE, que convierte los datos en hexadecimal (generados por el simulador) a datos en fraccional, y FR.EXE, que grafica amplitud y fase de la transformada de Fourier de los datos obtenidos.

5.1 Filtro RC pasa-altas

En muchas ocasiones los amplificadores introducen una componente de directa, conocida como offset, a la señal. Esto no es nada deseable, ya que dicha componente puede ocasionar errores. Para solucionar este problema, se acostumbra utilizar un filtro pasa-altas, eliminando así la componente de directa, figura 5.1.

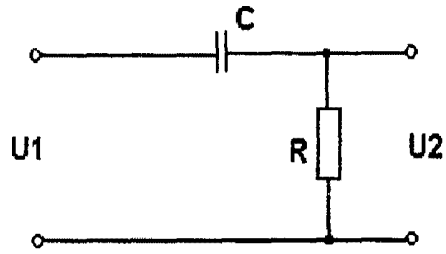


Figura 5.1. Filtro RC pasa-altas

La función de transferencia del filtro pasa-altas, mostrado en la figura 5.1, es

$$\frac{U_2}{U_1} = \frac{R}{R + \frac{1}{pc}} = \frac{p}{p + \frac{1}{CR}} \tag{5.1}$$

Utilizando la ecuación bilineal

$$p = c\frac{z-1}{z+1}$$

obtenemos la función de transferencia del filtro digital pasa-altas H(z)

$$H(z) = \frac{K - K.z^{-1}}{1 + b_1.z^{-1}} \tag{5.2}$$

donde

$$K = \frac{c}{c + \omega_c}$$

$$b_1 = \frac{\omega_c - c}{\omega_c + c}$$

$$c = 2.f_s \qquad \omega_c = \frac{1}{RC}$$

$$\omega_c = 2.pi.fc$$

La estructura que realiza la función de transferencia (5.2) está en la figura 5.2.

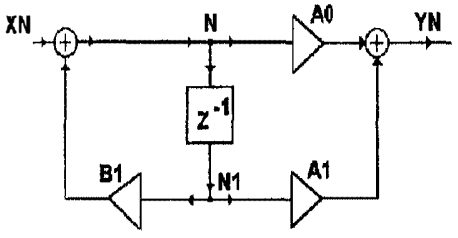


Figura 5.2. Filtro digital pasa-altas

Escriba el programa para el filtro pasa-altas y, mediante el simulador de Texas Instruments, calcule la respuesta a un impulso unitario y a un tren de impulsos. La frecuencia del muestreo que sea de 8000 [Hz]. Con la transformada rápida de Fourier (programa FR.EXE), calcule el espectro del filtro RC.

```
      AORG   >0000
RESET  B    INIT
      AORG  >0020
*
INIT  SOVM
      LDPK   0
      ZAC
      LARP   AR2
      LRLK   AR2,>0060
      RPTK   31
      SACL   *+
      LRLK   AR2,>0060
      RPTK   3
```

```
      BLKP   COEF,*+
*
*      EINT
*****************************
*DECLARACION DE LAS VARIABLES
*****************************
A0    EQU  >0060
A1    EQU  >0061
B1    EQU  >0062
N     EQU  >0063
N1    EQU  >0064
YN    EQU  >0065
XN    EQU  >0066
***************************
*EL PROGRAMA PARA FILTRO RC
***************************
START   IN   XN,PA1           ;<PA1>-->XN
        LT   N1               ;<N1>-->TR
        MPY  B1               ;<N1*B1>-->PR
        ZAC                   ;<0>-->ACC
        APAC                  ;<N1*B1>-->ACC
        ADD  XN,15            ;<XN+B1*N1>-->ACC
        SACH N,1              ;<XN+B1*N1>-->N
        ZAC                   ;0-->ACC
        MPY  A1               ;<N1*A1>-->PR
        LTD  N                ;<N>-->N1, <N>-->TR, <A1*N1>-->ACC
        MPY  A0               ;<N*A0>-->PR
        APAC                  ;<A1*N1+A0*N>-->ACC
        SACH YN,1             ;<ACC>-->YN
        OUT  YN,PA2           ;<YN>-->PA2
        ZAC                   ;0-->ACC
*************************************************************************
*EL PROGRAMA NECESITA EN TOTAL 625 CICLOS PARA REALIZAR LA FRECUENCIA     *
*DE MUESTREO DE 8000 Hz. EL PROGRAMA HASTA ESTE MOMENTO TIENE 15 CICLOS.  *
*ENTONCES ES NECESARIO CON LA INSTRUCCION NOP TODAVIA REALIZAR 610 CICLOS.*
*************************************************************************
        LARP  AR1
        LARK  AR1,152
        NOP
PAUSE   NOP
        NOP
        BANZ  PAUSE
        B     START
***************************
```

```
*COEFICIENTES DEL FILTRO RC*
****************************
COEF   DATA   31529,-31529,30293
       END
```

La respuesta al impulso unitario se obtiene en forma hexadecimal. Dicha respuesta está formada por una serie de números que representan las muestras de la señal en el dominio del tiempo.

Mediante el programa TOFRACT.EXE se convierten los números hexadecimales de la respuesta al impulso en números decimales.

Al correr el programa TOFRACT.EXE, éste pregunta: en primer término, ¿Cuántos números queremos convertir de hexadecimal a decimal?; después, ¿En qué archivo se encuentran los números que vamos a convertir?; y por último, ¿En qué archivo vamos a guardar los números ya convertidos?

Es necesario apuntar estos números en una hoja, ya que serán usados para la obtención del espectro. Se utiliza el programa FR.EXE, donde indicaremos la frecuencia de muestreo y los coeficientes de las muestras en forma decimal. Las muestras se escriben como un solo polinomio y entonces, para especificar el orden del denominador, tecleamos 0 y el coeficiente 1. Para poder observar el espectro en la pantalla, se oprime la tecla "a".

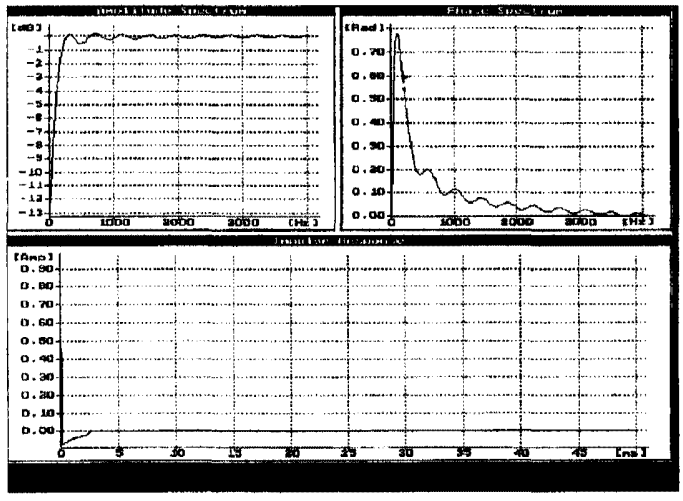


Figura 5.3. Espectro del filtro RC

La respuesta al impulso unitario de este filtro es:

$$h(n) = \{0.9622 - 0,07265 - 0.06716...\}$$

Estos números se obtienen también si se divide el numerador entre el denominador de la función de transferencia H(z):

$$H(z) = \frac{0.9622 + 0.9622z^{-1}}{1 - 09245z^{-1}} = 0.9622 - 0.07265z^{-1} - 0.06716z^{-2}...$$

Si los números obtenidos, mediante el simulador y la división de la función de transferencia H(z), son idénticos, entonces el programa está bien escrito.

Para obtener el espectro, que se muestra en la figura 5.3, utilizaremos el programa FR.EXE para la transformada rápida de Fourier, tecleando 20 muestras obtenidas mediante SIM25.EXE.

5.2 Filtro RC pasa-bajas

Escriba el programa en ensamblador para el circuito RC de la figura 5.4. Mediante el simulador, calcule la respuesta al impulso unitario y el espectro, como en el ejemplo anterior.

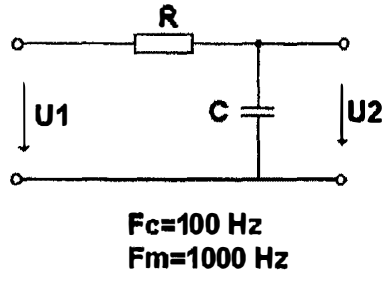


Figura 5.4. Filtro RC pasa-bajas

Capítulo 6

Transformada rápida de Fourier

En este capítulo se analiza el algoritmo de la transformada rápida de Fourier (TRF) para N=8. El alumno analizará las ecuaciones de este algoritmo y más tarde escribirá el programa para el simulador del TMS320C25 y comparará el resultado con los datos obtenidos anteriormente.

Este programa calcula el espectro de la señal mediante la transformada rápida de Fourier para N=8. Se utiliza el *algoritmo de decimación en el tiempo.* La mariposa radix 2 se muestra en la figura 6.1.

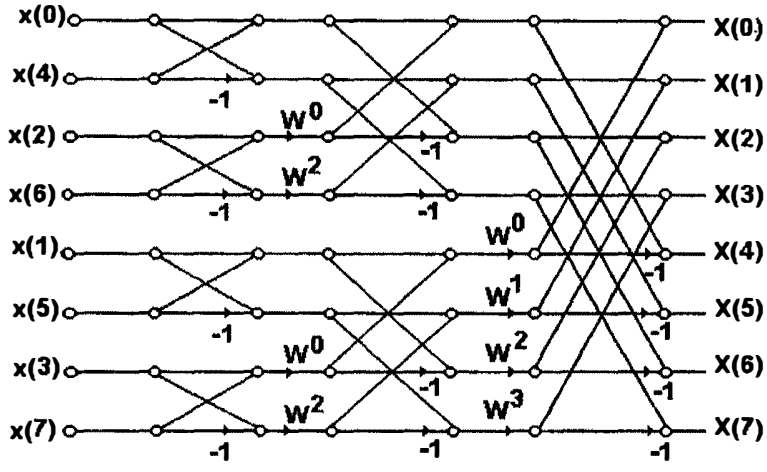


Figura 6.1. Mariposa para TRF N=8

De la mariposa en la figura 6.1 podemos escribir las siguientes ecuaciones para la primera etapa:

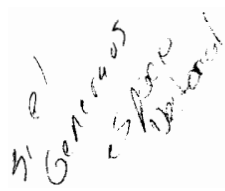
$$
\begin{aligned}
x(0) &= [x(0) + x(4)] \\
x(4) &= [x(0) - x(4)] \\
x(2) &= [x(2) + x(6)].W^0 \\
x(6) &= [x(2) - x(6)].W^2 \\
x(1) &= [x(1) + x(5)] \\
x(5) &= [x(1) - x(5)] \\
x(3) &= [x(3) + x(7)].W^0 \\
x(7) &= [x(3) - x(7 =].W^2
\end{aligned}
$$

Las ecuaciones para la segunda etapa de la mariposa:

$$
\begin{aligned}
x(0) &= [x(0) + x(2)] \\
x(4) &= [x(4) + x(6)] \\
x(2) &= [x(0) - x(2)] \\
x(6) &= [x(4) - x(6)] \\
x(1) &= [x(1) + x(3)].W^0 \\
x(5) &= [x(5) + x(7)].W^1 \\
x(3) &= [x(1) - x(3)].W^2 \\
x(7) &= [x(5) - x(7)].W^3
\end{aligned}
$$

Las muestras X(k) en el dominio de la frecuencia se obtienen en la tercera etapa de la mariposa y son las siguientes:

$$
\begin{aligned}
X(0) &= x(0) + x(1) \\
X(1) &= x(4) + x(5) \\
X(2) &= x(2) + x(3) \\
X(3) &= x(6) + x(7) \\
X(4) &= x(0) - x(1) \\
X(5) &= x(4) - x(5) \\
X(6) &= x(2) - x(3) \\
X(7) &= x(6) - x(7)
\end{aligned}
$$



Ejemplo 1

Calcule, mediante las ecuaciones anteriores, el espectro **X(k)** de la señal **x(n)**:

$$x(n) = \{\ 0.05,\ 0.05,\ 0.05,\ 0.05\ 0,\ 0,\ 0,\ 0\ \}$$

Si realizamos los cálculos paso a paso después de la primera etapa, tenemos para $x_0 = x_1 = x_2 = x_3 = 0.05$ y $x_4 = x_5 = x_6 = x_7 = 0$, entonces obtenemos:

$$
\begin{aligned}
x(0) &= 0.05 + 0.00 = 0.05 & \qquad x(1) &= 0.05 + 0.00 = 0.05 \\
x(2) &= 0.05 + 0.00 = 0.05 & \qquad x(3) &= 0.05 + 0.00 = 0.05 \\
x(4) &= (0.05 - 0.00) = 0.05 & \qquad x(5) &= (0.05 - 0,00) = 0.05 \\
x(6) &= (0.05 + 0.00).(-j) = -j0.05 & \qquad x(7) &= (0.05 + 0.00).(-j) = -j0.05
\end{aligned}
$$

Después de la segunda etapa tenemos:

$$x(0) = 0.05 + 0.05 = 0.1 \qquad x(1) = 0.05 + 0.05 = 0.1$$
$$x(2) = 0.00 - 0.00 = 0.0 \qquad x(3) = (0.05 - 0.05)W^2 = 0.00$$
$$x(4) = 0.05 - j0.05 \qquad x(5) = (0.05 - j0.05)(0.707 - j0.707) = -j0.0707$$
$$x(6) = 0.05 + j0.05 \qquad x(7) = (0.05 + j0.05)(-0.707 - j0.707) = -j0.0707$$

El resultado se obtiene, si se calcula la tercera etapa:

$$X(0) = 0.1 + 0.1 = 0.2 \qquad X(1) = 0.05 - j0.1207$$
$$X(2) = 0.00 \qquad X(3) = 0.05 - j0.0207$$
$$X(4) = 0.00 \qquad X(5) = 0.05 + j0.027$$
$$X(6) = 0.0 \qquad X(7) = 0.05 + j0.1207$$

Otro método para realizar la transformada rápida de Fourier es el de *decimación en la frecuencia*, el cual procesa las muestras de entrada en pares mediante N/2 2 punto transformadas, servidas por dos N/2 punto transformadas que generan los elementos pares e impares de la secuencia de salida.

De nueva cuenta se obtienen las ecuaciones para la primera, segunda y tercera etapas de la mariposa, que se muestran en la figura 6.2.

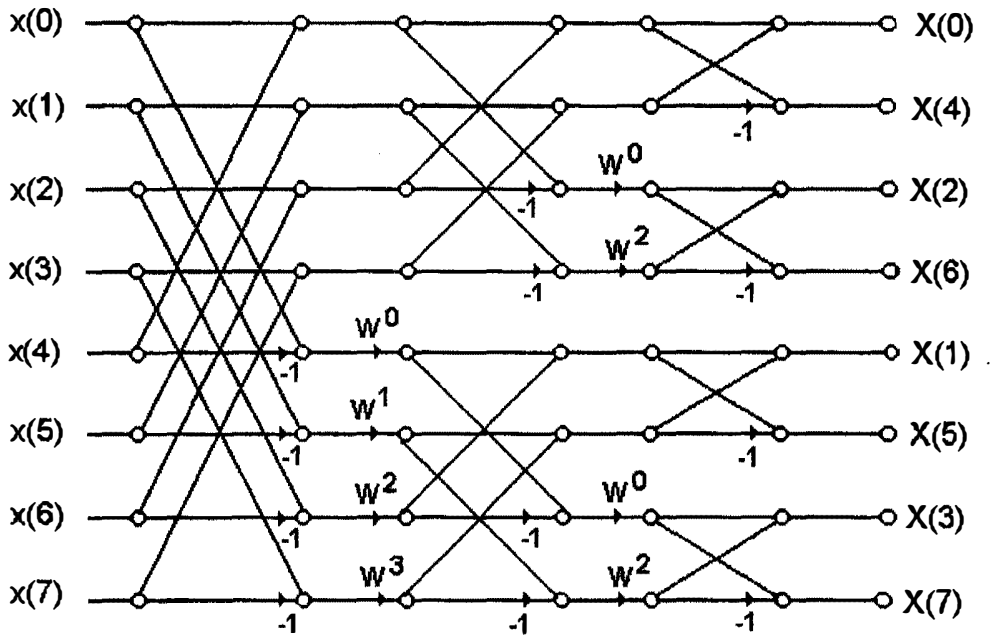


Figura 6.2. Mariposa de la TRF para N=8, decimación en la frecuencia

ETAPA 1	ETAPA 2	ETAPA 3
$x(0) = x(0) + x(4)$	$x(0) = x(0) + x(2)$	$X(0) = x(0) + x(1)$
$x(1) = x(1) + x(5)$	$x(1) = x(1) + x(3)$	$X(4) = x(0) - x(1)$
$x(2) = x(2) + x(6)$	$x(1) = [x(0) - x(2)].W^0$	$X(2) = x(2) + x(3)$
$x(3) = x(3) + x(7)$	$x(3) = [x(1) - x(3)].W^2$	$X(6) = x(2) - x(3)$
$x(4) = [x(0) - x(4)].W^0$	$x(4) = x(4) + x(6)$	$X(1) = x(4) + x(5)$
$x(5) = [x(1) - x(5)].W^1$	$x(5) = x(5) + x(7)$	$X(1) = x(4) + x(5)$
$x(6) = [x(2) - x(6)].W^2$	$x(6) = [x(4) - x(6)].W^0$	$X(3) = x(6) + x(7)$
$x(7) = [x(3) - x(7)].W^3$	$x(7) = [x(5) - x(7)].W^2$	$X(7) = x(6) - x(7)$

Ejemplo 2

Para las muestras de entrada

$$x(0) = x(2) = x(4) = x(6) = 0.1$$

$$x(1) = x(3) = x(5) = x(7) = 0.0$$

calcule el espectro X(k), utilizando la mariposa decimación en la frecuencia.

Solución:

Sustituyendo las ecuaciones de las etapas 1, 2 y 3, obtenemos el espectro en la forma:

$x(0) = 0.2$	$x(0) = 0.4$	$X(0) = 0.4$
$x(1) = 0.0$	$x(1) = 0.0$	$X(4) = 0.4$
$x(2) = 0.2$	$x(2) = 0.0$	$X(2) = 0.0$
$x(3) = 0.0$	$x(3) = 0.0$	$X(6) = 0.0$
$x(4) = 0.0$	$x(4) = 0.0$	$X(1) = 0.0$
$x(5) = 0.0$	$x(5) = 0.0$	$X(5) = 0.0$
$x(6) = 0.0$	$x(6) = 0.0$	$X(3) = 0.0$
$x(7) = 0.0$	$x(7) = 0.0$	$X(7) = 0.0$

Ejemplo 3

Para las muestras x(n)

$$x(0) = x(1) = x(2) = x(3) = 0.1$$

$$x(4) = x(5) = x(6) = x(7) = 0.0$$

calcule mediante el algoritmo decimación en las frecuencias el espectro X(k).

Solución:

Para la primera, segunda y tercera etapas, obtenemos:

$x(0) = 0.1$	$x(0) = 0.2$	$X(0) = 0.4$
$x(1) = 0.1$	$x(1) = 0.2$	$X(4) = 0.0$
$x(2) = 0.1$	$x(2) = 0.0$	$X(2) = 0.0$
$x(3) = 0.1$	$x(3) = 0.0$	$X(6) = 0.0$
$x(4) = 0.1$	$x(4) = -0.1 - j0.1$	$X(1) = 0.1 - j0.2414$
$x(5) = 0.0707 - j0.0707$	$x(5) = -j0.1414$	$X(5) = 0.1 + j0.0414$
$x(6) = -j0.1$	$x(6) = 0.1 + j0.1$	$X(3) = 0.1 - j0.0414$
$x(7) = -0.0707 - j0.0707$	$x(7) = -j0.1414$	$X(7) = 0.1 + j0.2414$

Práctica de laboratorio

Mediante el siguiente programa y el simulador, calcule los espectros de los ejemplos anteriores. No olvide que los datos de entrada se modifican en el archivo COEF, localizado al final del programa, y que las mismas locaciones de la memoria de las muestras de entrada son utilizadas para almacenar los resultados intermedios de las distintas etapas. Por lo tanto, para observar la secuencia de salida tendrá que abrir el mapa de memoria RAM, mediante el comando RAMI dentro del paquete del simulador SIM25.EXE.

```
*************************************************************************
*************************************************************************
**  PROGRAMA DE LA TRF PARA 8 MUESTRAS, DECIMACION EN LA FRECUENCIA  **
**                                                                     **
**  COMPLEMENTADO Y REVISADO POR AHMED CONCHA y ERYX DE LA TORRE A.   **
**                                                                     **
*************************************************************************
*************************************************************************
*
*
RESET  B    INIT
*
         AORG >0010
*
* INICIO DEL MICROCONTROLADOR
*
INIT   SOVM              ;Trabaja en saturacion
       LDPK 0            ;Trabaja con bandera cero
       ZAC               ;Anulacion del acumulador
       LARP AR2          ;Actualiza registro auxiliar  AR2
       LRLK AR2,>0060    ;Inicializa bloque B2
       RPTK 31           ;Repite la instruccion que sigue 32 veces
       SACL *+           ;Anulacion del bloque  B2
```

```
       LRLK AR2,>0060     ;Inicializa del bloque B2 en el direccion 0060
       RPTK 23            ;Repite la instruccion que sigue 23 veces
       BLKP COEF,*+       ;Transferencia de los coeficientes al bloque N2

*  CONSTANTES W
*
*COS0    DATA  32767
*COS1    DATA  23170
*COS2    DATA  0
*COS3    DATA -23170
*SEN0    DATA  0
*SEN1    DATA -23170
*SEN2    DATA -32767
*SEN3    DATA -23170

* MUESTRAS (PARTE REAL)

XR0     EQU  >0060
XR1     EQU  >0062
XR2     EQU  >0064
XR3     EQU  >0066
XR4     EQU  >0068
XR5     EQU  >006A
XR6     EQU  >006C
XR7     EQU  >006E

* MUESTRAS (PARTE IMAGINARIA)

XI0     EQU  >0061
XI1     EQU  >0063
XI2     EQU  >0065
XI3     EQU  >0067
XI4     EQU  >0069
XI5     EQU  >006B
XI6     EQU  >006D
XI7     EQU  >006F
*
* LOCALIDADES PARA ALMACENAR LOS VALORES DEL COSENO Y DEL SENO
*
C0      EQU   >0070
C1      EQU   >0071
C2      EQU   >0072
C3      EQU   >0073
S0      EQU   >0074
```

```
S1      EQU   >0075
S2      EQU   >0076
S3      EQU   >0077
TR1     EQU   >0078
TI1     EQU   >0079

**** DEFINICION DE LAS RUTINAS DE LOS MACROS
* MACRO CON LOS BITS AL REVES

BITR    $MACRO R1,I1,R2,I2
        LAC   :R1.S:
        SACL  TR1
        LAC   :R2.S:
        SACL  :R1.S:
        LAC   TR1
        SACL  :R2.S:
        LAC   :I1.S:
        SACL  TR1
        LAC   :I2.S:
        SACL  :I1.S:
        LAC   TR1
        SACL :I2.S:
        $END BITR

* MARIPOSA SIN TERMINOS SENO Y COSENO
FREEBF  $MACRO R1,I1,R2,I2
        LAC   :R1.S:
        SUB   :R2.S:
        SACL  TR1
        LAC   :I1.S:
        SUB   :I2.S:
        SACL  TI1
        LAC   :R1.S:
        ADD   :R2.S:
        SACL  :R1.S:
        LAC   :I1.S:
        ADD   :I2.S:
        SACL  :I1.S:
        LAC   TR1
        SACL  :R2.S:
       LAC   TI1
SACL  :I2.S:
        $END  FREEBF
```

```
* MARIPOSA CON TERMINOS SENO(K2) Y COSENO(K1)
CONSBF  $MACRO R1,I1,R2,I2,K1,K2
        LAC   :R1.S:
        SUB   :R2.S:
        SACL  TR1
        LAC   :I1.S:
        SUB   :I2.S:
        SACL  TI1
        LAC   :R1.S:
        ADD   :R2.S:
        SACL  :R1.S:
        LAC   :I1.S:
        ADD   :I2.S:
        SACL  :I1.S:
***********
ZAC
LT    TR1
MPY   :K1.S:
LTA   TI1
MPY   :K2.S:
SPAC
SACH  :R2.S:,1
ZAC
LT    TR1
MPY   :K2.S:
LTA   TI1
MPY   :K1.S:
APAC
SACH   :I2.S:,1
$END  CONSBF

**********ETAPAS**************
**** PRIMERA ETAPA (ITERACION)****

FIR FREEBF XR0,XI0,XR4,XI4
    CONSBF XR1,XI1,XR5,XI5,C1,S1
    CONSBF XR2,XI2,XR6,XI6,C2,S2
    CONSBF XR3,XI3,XR7,XI7,C3,S3
*******************************


*****SEGUNDA ETAPA (ITERACION)****
    FREEBF XR0,XI0,XR2,XI2
```

```
    CONSBF XR1,XI1,XR3,XI3,C2,S2
    FREEBF XR4,XI4,XR6,XI6
    CONSBF XR5,XI5,XR7,XI7,C2,S2
*******************************

*****TERCERA ETAPA (ITERACION)*****
    FREEBF XR0,XI0,XR1,XI1
    FREEBF XR2,XI2,XR3,XI3
    FREEBF XR4,XI4,XR5,XI5
    FREEBF XR6,XI6,XR7,XI7
*********************************

**** BIT REVERSAL
  BITR XR1,XI1,XR4,XI4
  BITR XR3,XI3,XR6,XI6
************************************
*OBTENCION DEL ESPECTRO
*

COEF    DATA 3276,3276,3276,3276
        DATA 3276,3276,3276,3276
        DATA 3276,-3276,3276,-3276
        DATA 3276,-3276,3276,-3276
DATA 32767,23170,0,-23170
        DATA 0,-23170,-32767,-23170
*
*  EN LA SECCION COEF SE INTRODUCEN LOS VALORES DE
*  LAS 8 MUESTRAS, RECUERDE QUE ESTOS VALORES SE
*  GUARDAN EN LA MEMORIA A PARTIR DE LA DIRECCION
*  0060 A 006F Y QUE LOS VALORES REALES VAN EN LAS
*  DIRECCIONES PARES Y LOS VALORES IMAGINARIOS EN
*  LAS DIRECCIONES IMPARES.

        END
```

Capítulo 7

Transformada de cosenos

En este capítulo los alumnos aprenderán la manera en que se construye la mariposa correspondiente a la transformada de cosenos para N=4. Asimismo, construirán un algoritmo para la transformada rápida de cosenos. Dentro del programa se utilizarán macros en el lenguaje ensamblador. Por último, se incluye un ejemplo para calcular la transformada de cosenos por etapas.

7.1 Transformada discreta de cosenos

La transformada discreta de cosenos (TDC-1) está definida por la ecuación (7.1),

$$X(k)^{C(1)} = \left(\frac{2}{N}\right)^{\frac{1}{2}} \alpha(k) \sum_{n=0}^{N-1} x(n) \cos(\frac{k\pi n}{N}) \tag{7.1}$$

y la transformada inversa de cosenos por la ecuación (7.2),

$$x(n) = \left(\frac{2}{N}\right)^{\frac{1}{2}} \alpha(n) \sum_{k=0}^{N-1} X(k)^{C(1)} \cos(\frac{k\pi n}{N}) \tag{7.2}$$

donde k,n=0, 1, 2, ... ,N.

La transformada discreta de cosenos (TDC-2) está definida por la ecuación (7.3),

$$X(k)^{C(2)} = \left(\frac{2}{N}\right)^{\frac{1}{2}} \alpha(k) \sum_{n=0}^{N-1} x(n) \cos(\frac{k\pi(2n+1)}{2N}) \tag{7.3}$$

y la transformada inversa de cosenos (TDCI-2) por la ecuación (7.4),

$$x(n) = \left(\frac{2}{N}\right)^{\frac{1}{2}} \alpha(n) \sum_{k=0}^{N-1} X(k)^{C(2)} \cos(\frac{k\pi(2n+1)}{2N}) \tag{7.4}$$

donde k,n=0,1,2, ... ,N.

En la literatura están definidas las transformadas de cosenos TDC-3 y TDC-4. El término $\alpha(k)$ y $\alpha(n)$ es el factor de escala.

Ejemplo 1

Calcule la matriz TDC-2 para N=4.

Si calculamos $X(k)^{C(2)}$ para n=0, 1, 2, 3, obtenemos, desarrollando la ecuación (7.3), la siguiente expresión:

$$X(k)^{C(2)} = \sqrt{\frac{2}{4}}\alpha(k)\left[x(0)cos(\frac{k\pi}{8}) + x(1)cos(\frac{3k\pi}{8}) + x(2)cos(\frac{5k\pi}{8}) + x(3)cos(\frac{7k\pi}{8})\right] \quad (7.5)$$

Desarrollando (7.5) para k=0, 1, 2, 3, obtenemos las ecuaciones:

$$X(0)^{C(2)} = \sqrt{\frac{1}{2}}\alpha(0)\left[x(0) + x(1) + x(2) + x(3)\right]$$

$$X(1)^{C(2)} = \sqrt{\frac{1}{2}}\alpha(1)\left[0.923x(0) + 0.382x(1) - 0.382x(2) - 0.923x(3)\right]$$

$$X(2)^{C(2)} = \sqrt{\frac{1}{2}}\alpha(2)\left[0.707x(0) - 0.707x(1) + 0.707x(2) - 0.707x(3)\right]$$

$$X(3)^{C(2)} = \sqrt{\frac{1}{2}}\alpha(3)\left[0.383x(0) - 0.923x(1) + 0.923x(2) - 0.382x(3)\right]$$

Si el factor de escala es $\sqrt{2}$, obtenemos la ecuación matricial:

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0.923 & 0.382 & -0.382 & -0.923 \\ 0.707 & -0.707 & -0.707 & 0.707 \\ 0.383 & -0.923 & 0.923 & -0.382 \end{bmatrix} \times \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \quad (7.6)$$

En la ecuación matricial (7.6) cambiamos el vector $X(k) = \{X(0), X(1), X(2), X(3)\}$ por el vector $X(k) = \{X(0), X(2), X(1), X(3)\}$ y obtenemos:

$$\begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0.707 & -0.707 & -0.707 & 0.707 \\ 0.923 & 0.382 & -0.382 & -0.923 \\ 0.383 & -0.923 & 0.923 & -0.382 \end{bmatrix} \times \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \quad (7.7)$$

Si en la ecuación (7.7) cambiamos el vector $x(n) = \{x(0), x(1), x(2), x(3)\}$ por el vector $x(n) = \{x(0), x(2), x(3), x(1)\}$, obtenemos:

$$\begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} = \left[\begin{array}{cc|cc} 1 & 1 & 1 & 1 \\ 0.707 & -0.707 & 0.707 & -0.707 \\ \hline 0.923 & -0.382 & -0.923 & 0.382 \\ 0.382 & 0.923 & -0.382 & -0.923 \end{array}\right] \times \begin{bmatrix} x(0) \\ x(2) \\ x(3) \\ x(1) \end{bmatrix} \quad (7.8)$$

De la última ecuación podemos escribir las submatrices **A**, **B**:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 0.707 & -0.707 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 0.923 & -0.382 \\ 0.382 & 0.923 \end{bmatrix} \tag{7.9}$$

Sustituyendo (7.8) en (7.9), obtenemos la ecuación matricial:

$$\begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} = \begin{bmatrix} [A] & [A] \\ [B] & -[B] \end{bmatrix} \times \begin{bmatrix} x(0) \\ x(2) \\ x(3) \\ x(1) \end{bmatrix} \tag{7.10}$$

La matriz **B** se puede expresar mediante la matriz **A** y se obtienen las ecuaciones recursivas.

Ejemplo 2

Con la transformada discreta de cosenos, calcule el espectro X(k) si se conocen las muestras en el dominio del tiempo x(n) y si N=4.

$$x(n) = \{1, 1, 1, 1\}$$

Solución:

Calculando la ecuación matricial, obtenemos el resultado $X(k) = \{4, 0, 0, 0\}$.

$$\begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0.707 & -0.707 & 0.707 & -0.707 \\ 0.923 & -0.382 & -0.923 & 0.382 \\ 0.383 & 0.923 & -0.382 & -0.923 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{7.11}$$

Ejemplo 3

Para N=8 calcule la matriz de TDC-2.

Solución:

Mediante la ecuación (7.3), obtenemos la matriz:

$$\begin{bmatrix} X(0) \\ X(4) \\ X(2) \\ X(6) \\ X(1) \\ X(5) \\ X(3) \\ X(7) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ a & -a & a & -a & a & -a & a & -a \\ b & -c & -b & c & b & -c & -b & c \\ c & b & -c & -b & c & b & -c & -b \\ d & e & -g & f & -d & -e & g & -f \\ e & g & -f & d & -e & -g & f & -d \\ f & -d & -e & g & -f & d & -e & -g \\ g & f & d & e & -g & -f & -d & -e \end{bmatrix} \times \begin{bmatrix} x(0) \\ x(2) \\ x(4) \\ x(6) \\ x(7) \\ x(5) \\ x(3) \\ x(1) \end{bmatrix} \tag{7.12}$$

donde

$a = \sqrt{\frac{1}{2}} = 0.707$ $\quad b = C_8^1 = 0.923$ $\quad c = S_8^1 = 0.382$

$d = C_{16}^1 = 0.9807$ $\quad e = S_{16}^3 = 0.555$ $\quad f = C_{16}^3 = 0.831$

$g = S_{16}^1 = 0.195$ $\quad C_k^i = cos(\frac{\pi i}{k})$ $\quad S_k^i = sen(\frac{\pi i}{k})$

7.2 Transformada rápida de cosenos

Para obtener la mariposa de la transformada rápida de cosenos es necesario dividir la matriz de cosenos en submatrices que se puedan obtener recursivamente. Por ejemplo, la matriz de cosenos TDC-2 se calcula mediante la ecuación (7.3),

$$X(k) = \alpha(k)\left[x(0)\cos\frac{\pi k}{4} + x1\cos\frac{3k\pi}{4}\right]$$

y para k=0, 1 obtenemos:

$$X(0) = \alpha(0)\left[x(0) + x(1)\right]$$
$$X(1) = \alpha(1)\left[0.707x(0) - 0.707x(1)\right]$$

Si $\alpha_{(1)} = 1$ y $\alpha_{(0)} = \frac{1}{\sqrt{2}}$, la matriz $\mathbf{A}_2$ toma la forma:

$$\mathbf{A}_2 = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0.707 & -0.707 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ C_4^1 & C_4^3 \end{bmatrix}$$

donde

$$C_4^1 = \cos\frac{\pi}{4} \qquad C_4^3 = \cos\frac{3\pi}{4}$$

al igual como para n=2, obtenemos para n=4 la matriz $\mathbf{A_4}$:

$$\mathbf{A_4} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ C_8^1 & C_8^3 & C_8^5 & C_8^7 \\ C_8^2 & C_8^6 & C_8^6 & C_8^2 \\ C_8^3 & C_8^7 & C_8^1 & C_8^5 \end{bmatrix} \tag{7.13}$$

donde

$$C_k^i = \cos\left(\frac{\pi i}{k}\right)$$

De la propiedad de la simetría de la transformada de cosenos

$$C_8^3 = -C_8^5 \qquad C_8^6 = C_4^3 \qquad C_8^2 = C_4^1 \qquad C_8^7 = -C_8^1$$

y obtenemos la matriz (7.14):

$$\mathbf{A_4} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ C_8^1 & C_8^3 & -C_8^3 & -C_8^1 \\ C_4^1 & C_4^3 & C_4^3 & C_4^1 \\ C_8^3 & -C_8^1 & C_8^1 & -C_8^3 \end{bmatrix} \tag{7.14}$$

Podemos escribir la matriz $\mathbf{A_4}$ como producto de dos matrices:

$$\mathbf{A_4} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ C_4^1 & C_4^3 & C_4^3 & C_4^1 \\ C_8^3 & -C_8^1 & C_8^1 & -C_8^3 \\ C_8^1 & C_8^3 & -C_8^3 & -C_8^1 \end{bmatrix} \tag{7.15}$$

La primera matriz es la de permutación, que crea arriba del vector $\mathbf{X(k)}$ las muestras pares en orden ascendente X(0), X(2), ... y abajo reordena de manera descendente las muestras impares ... X(5), X(3), X(1), como se observa en la siguiente ecuación:

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} X(0) \\ X(2) \\ X(3) \\ X(1) \end{bmatrix}$$

La ecuación matricial (7.15) se puede escribir en la siguiente forma:

$$\mathbf{A_4} = \left[\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \hline 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array}\right] \times \left[\begin{array}{cc|cc} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ C_4^1 & C_4^3 & 0 & 0 \\ \hline 0 & 0 & -C_8^1 & C_8^3 \\ 0 & 0 & C_8^3 & C_8^1 \end{array}\right] \times \left[\begin{array}{cc|cc} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ \hline 0 & 1 & -1 & 0 \\ 1 & 0 & 0 & -1 \end{array}\right] \tag{7.16}$$

La ecuación matricial (7.16) se puede escribir en una ecuación más compacta:

$$\mathbf{A_4} = \mathbf{P_4} \times \begin{bmatrix} \mathbf{A_2} & \mathbf{0} \\ \mathbf{0} & \mathbf{P_2} \end{bmatrix} \times \mathbf{B_4} \tag{7.17}$$

El elemento A_{12} ($\mathbf{A_2}$) es la matriz de la transformada de cosenos TDC-2 para N=2; $\mathbf{P_4}$, la de permutación; y $\mathbf{B_4}$, la de mariposa. La matriz de permutación crea las muestras pares del vector X(k) en forma ascendente y, en la parte baja, las muestras impares en forma descendente. El objetivo de la transformada rápida de cosenos es expresar la matriz $\mathbf{A_4}$ recursivamente por la de $\mathbf{A_2}$, mientras que la de mariposa $\mathbf{B_4}$ se puede expresar por la matriz

$$\mathbf{B_4} = \begin{bmatrix} \mathbf{I_2} & \mathbf{I_2^T} \\ \mathbf{I_2^T} & -\mathbf{I_2} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & -\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{bmatrix} \tag{7.18}$$

De la matriz (7.16) podemos ver, que todas las submatrices tienen dos columnas y dos renglones. En general, para N^2 se puede escribir la ecuación matricial (7.16) en la forma:

$$\mathbf{A}_N = \mathbf{P}_N \times \begin{bmatrix} \mathbf{A}_{\frac{N}{2}} & 0 \\ 0 & \mathbf{R}^T_{\frac{N}{2}} \end{bmatrix} \times \mathbf{B}_N \tag{7.19}$$

La matriz de mariposa $\mathbf{B}_N$ se puede expresar por las matrices de identidad $\mathbf{I}_{\frac{N}{2}}$

$$\mathbf{B}_N = \begin{bmatrix} \mathbf{I}_{\frac{N}{2}} & \mathbf{I}^T_{\frac{N}{2}} \\ \mathbf{I}^T_{\frac{N}{2}} & -\mathbf{I}_{\frac{N}{2}} \end{bmatrix} \tag{7.20}$$

y la matriz $\mathbf{R}^T_N$ se obtiene, si se cambian las columnas y renglones, de la matriz $\mathbf{R}_N$. El elemento i, k de la matriz $\mathbf{R}_N$ se obtiene mediante la ecuación (7.21):

$$\mathbf{R}_{N_{ik}} = \cos\left[\frac{(2i+1)(2k+1)\pi}{4N}\right] \qquad i,k = 0,1,2,3,...N-1. \tag{7.21}$$

La mariposa de la transformada rápida de cosenos TRC, obtenida mediante este algoritmo, se muestra en la figura 7.1 para la decimación en el tiempo DET TRC-2. De la mariposa en la figura 7.1, podemos escribir las siguientes ecuaciones para cada etapa:

Fase 1	*Fase 2*	*Fase 3*	*Fase 4*	*Fase 5*	*Fase 6*	*Fase 7*
$x_0 = x_0$	$x_0 = x_0$	$x_0 = x_0 + x_2$	$x_0 = x_0 + x_1$	$x_0 = x_0$	$x_0 = x_0 + x_3$	$X_0 = 2^{-1}x_0$
$x_0 = x_0 + x_1$	$x_1 = x_2$	$x_1 = x_1$	$x_1 = x_0 - x_1$	$x_1 = x_2$	$x_1 = x_1 + x_k$	$X_1 = x_1(2C_2)^{-1}$
$x_2 = x_1 + x_2$	$x_2 = x_k$	$x_2 = x_0 - x_2$	$x_2 = x_2$	$x_2 = x_1$	$x_2 = x_2$	$X_2 = x_2(2C_4)^{-1}$
$x_3 = x_2 + x_3$	$x_3 = x_1$	$x_3 = x_3 + x_k$	$x_3 = x_3$	$x_3 = x_3$		
$x_k = x_3$	$x_k = x_3$	$x_k = x_3 - x_k$	$x_k = x_k.C_4$	$x_k = x_k$	$x_k = x_1 - x_k$	$X_3 = x_k(2C_6)^{-1}$

Ejemplo 4

Calcule, mediante las fases de la transformada de cosenos, el vector X(k) si se conoce $x(n) = \{1, 0, 1, 0\}$. El resultado está en la tabla 7.1:

Entradas	1	2	3	4	5	6	7	Resultados
$x_0 = 1$	1	1	1	2	2	4	2	$X_0 = 2$
$x_1 = 0$	1	1	1	0	1	1	0.541	$X_1 = 0.541$
$x_2 = 1$	1	0	1	1	0	0	0	$X_2 = 0$
$x_3 = 0$	1	1	2	2	2			$X_3 = 0.382$
x_k	0	1	0	0	0	1	0.3826	

Tabla 7.1. Resultados en las etapas parciales

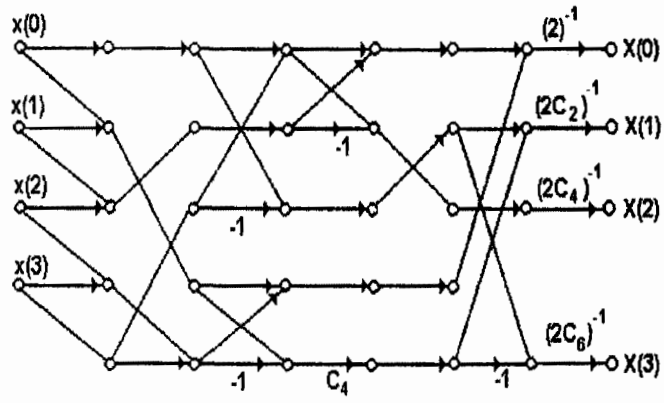


Figura 7.1. Mariposa DET TCR-2 para N=4

Ejemplo 5

G.- 907102

Mediante la mariposa de la figura 7.1, calcule el vector X(k) si se conocen las muestras x(n) en el dominio del tiempo $x(n) = \{\ 1\ 1\ 1\ 1\ \}$.

Solución:

De la mariposa en la figura 7.1, podemos escribir las ecuaciones:

$$\begin{aligned}
X(0) &= 2x(0).2^{-1} + x(1).2^{-1} + 2x(2)^{-1} + 2x(3).2^{-1} && = 4\\
X(1) &= (2C_2)^{-1}.[x(0) + x(0).C_4 + x(1).C_4 - x(2).C_4 - x(3) - x(3).C_4] && = 0\\
X(2) &= (2C_4)^{-1}.[x(0) - x(1) - x(2) + x(3)] && = 0\\
X(3) &= (2C_6)_{-1}.[-x(0).C_4 + x(0) - x(1).C_4 + x(2).C_4 + x(3).C_4 - x(3)] && = 0
\end{aligned}$$

El mismo resultado, lo obtuvimos mediante el cálculo de la transformada discreta de cosenos en el ejemplo 2.

Ejemplo 6

Mediante el siguiente programa para el TMS320C25, calcule X(k) para la señal en la figura 7.2.

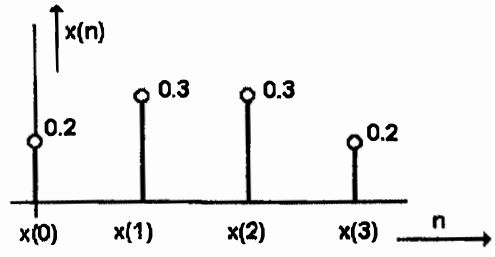


Figura 7.2. Señal en el dominio del tiempo

Solución:

```
* Programa para calcular la transformada de cosenos
* usando el TMS320C25x  Semestre 98-I
* Concha Dimas Ahmed - De la Torre Alatriste Eryx

       AORG >0000
RESET  B    INIT

       AORG >0020
*
* INICIO DEL MICROCONTROLADOR
*
INIT   ROVM              ;Trabaja en saturacion
       LDPK 0            ;Trabaja con bandera cero
       ZAC               ;Inicializacion del acumulador
       LARP AR2          ;Actualiza registro auxiliar AR2
       LRLK AR2,>0060    ;Inicializa bloque B2
       RPTK 31           ;Repite la instruccion que sigue 32 veces
       SACL *+           ;Anulacion del bloque B2
       LRLK AR2,>0060    ;Inicializa el bloque B2 en la direccion 0060
       RPTK 9            ;Repite la instruccion que sigue 10 veces
       BLKP COEF,*+      ;Transferencia de los coeficientes al bloque B2
*
* DECLARACION DE LAS VARIABLES
*
X0     EQU  >0060
X1     EQU  >0061
X2     EQU  >0062
X3     EQU  >0063
X4     EQU  >0064
K1     EQU  >0065
K2     EQU  >0066
K3     EQU  >0067
K4     EQU  >0068
K5     EQU  >0069
T      EQU  >006A
* Declaracion de las macros

MUEVE       $MACRO X1,X2  ;Las etiquetas, como MUEVE, empiezan
        LAC  :X1.S:       ;en la primera columna
        SACL :X2.S:
        $END MUEVE
```

```
MULTI   $MACRO X1,X2
        LT   :X1.S:
        MPY  :X2.S:
        PAC
        SACH :X1.S:,1
        $END MULTI

SUMA    $MACRO X1,X2
        LAC   :X1.S:
        ADD   :X2.S:
        SACL  :X1.S:
        $END  SUMA

RESTA   $MACRO X1,X2
        LAC   :X1.S:
        SUB   :X2.S:
        SACL  :X2.S:
        $END  RESTA

* Empieza el programa en ensamblador.

* FASE I
      MUEVE X3,X4
      SUMA X3,X2
      SUMA X2,X1
      SUMA X1,X0

* FASE II
      MUEVE X4,T
      MUEVE X3,X4
      MUEVE X1,X3
      MUEVE X2,X1
      MUEVE T,X2

* FASE III
      MUEVE  X0,T
      SUMA  X0,X2
      RESTA T,X2
      MUEVE  X3,T
      SUMA  X3,X4
      RESTA T,X4

* FASE IV
      MUEVE  X0,T
```

```
      SUMA  X0,X1
      RESTA T,X1
      MULTI  X4,K1

* FASE V
      MUEVE X1,T
      MUEVE X2,X1
      MUEVE T,X2

* FASE VI
      SUMA  X0,X3
      MUEVE  X1,T
      SUMA  X1,X4
      RESTA T,X4

*FASE VII
        MULTI  X0,K2
        MULTI  X1,K3
        MULTI  X2,K4
        MULTI  X4,K5   ;Es solo la mitad, ya que 1/(2*C6)es mayor que 1
        LAC    X4
        ADD    X4
        SACL   X3
*Coeficientes de x(n) en el orden x(0),x(1),x(2) y x(3)
COEF DATA 3276,0,3276,0
     DATA 0
*Coeficientes C4, .5 , 1/(2.C2), 1/(2.C4), 1/(2.C6)
     DATA 23169,16383,17734,23169
     DATA 21406,0
     END
```

Capítulo 8

Filtros adaptables

En este capítulo se analizan los filtros adaptables y la forma de programarlos en el TMS320CXX. El alumno aplicará el filtro adaptable a un problema de cancelación de ruido, utilizando el simulador SIM25. En los resultados obtenidos se deberá analizar la rapidez de convergencia del filtro adaptable al modificar la variable μ.

8.1 Introducción a los filtros adaptables

Los *filtros adaptables* tienen la propiedad de variar los coeficientes del filtro de acuerdo con las variaciones estadísticas del ambiente en el cual operan. Los filtros adaptables tienen aplicación en problemas, tales como ecualización del canal de comunicación, cancelación de ruido, identificación de sistemas, entre otros.

Al buscar una solución estadística se requiere de consideraciones estadísticas. La adaptación de los coeficientes se realiza de manera que una determinada señal de error es minimizada. Un criterio comúnmente utilizado es minimizar la media cuadrada del error, definiendo al error como la diferencia entre alguna señal deseada y la señal obtenida a la salida del filtro. Para un sistema estacionario, la solución es llamada *filtro Wiener*. Al graficar el error *vs* los parámetros del filtro se tiene una superficie donde el valor mínimo representa la solución Wiener. En el caso de señales no estacionarias, el filtro óptimo también es variante en el

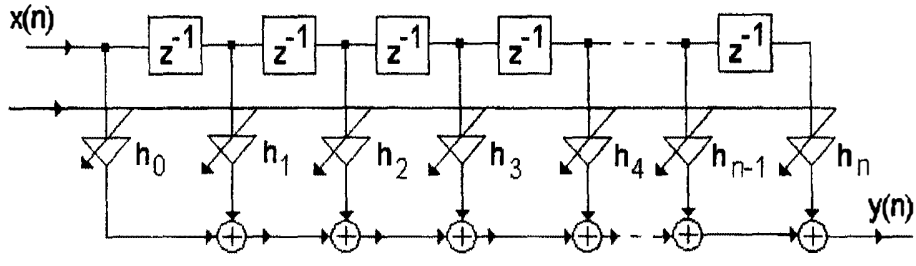


Figura 8.1. Filtro FIR transversal adaptable

tiempo, una solución adecuada la presenta el filtro de *Kalman*.

El filtro adaptable se basa en algoritmos recursivos que comienzan a partir de ciertas condiciones iniciales y convergen a una solución óptima. Algunas propiedades que se deben tomar en cuenta al diseñar algoritmos adaptables son las siguientes:

- Velocidad de convergencia a la solución óptima
- Desajuste entre el valor final del error y el valor mínimo
- Detección de variaciones estadísticas
- Requerimientos computacionales

Dos métodos que se emplean para derivar algoritmos recursivos son: gradiente de aproximación estocástica y estimación mediante mínimos cuadrados.

8.1.1 Gradiente de aproximación estocástica

El desempeño del filtro está basado en minimizar una función de pesos definida por la media del error cuadrado, es decir, la media cuadrada de la diferencia entre la respuesta ideal y la salida del filtro.

Esta función de pesos es una función de segundo orden y depende de los coeficientes del filtro, de manera que puede verse como una superficie al graficar el error *vs* los parámetros del filtro y es llamada *superficie de desempeño,* donde los pesos correspondientes al punto mínimo de la superficie equivalen a la solución Wiener óptima.

Un algoritmo comúnmente utilizado es el LMS *(least mean square)*, que actualiza los coeficientes de la siguiente forma:

```
(COEF. DEL FILTRO)=(COEF. DEL FILTRO)+(CONSTANTE)(DATOS)(ERROR)
      N+1                  N
```

8.1.2 Estimación mediante mínimos cuadrados

Otra forma de realizar un filtrado adaptable es utilizando como función de pesos la suma de los errores cuadrados. Un método comúnmente empleado es el algoritmo RLS *(recursive least squares)*, que se podría considerar como un caso particular del filtro de Kalman. Este algoritmo tiene la siguiente estructura:

```
COEF. DEL FILTRO)=(COEF. DEL FILTRO)+(GANANCIA KALMAN)(VECTOR DE INNOVACION)
      N+1                  N
```

En la figura 8.2 se muestra la estructura con un filtro adaptable. A diferencia de los filtros comunes, se tienen dos señales por analizar. Primeramente la entrada al filtro, $x[n]$, y una señal de referencia conocida, $d[n]$. El error se obtiene al comparar la salida del filtro con la señal de referencia, que se realimenta al algoritmo.

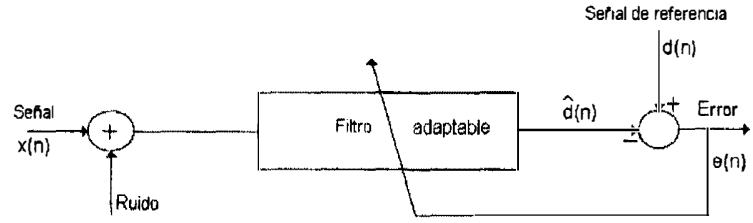


Figura 8.2. Circuito con filtro adaptable

8.2 Algoritmo LMS

El algoritmo LMS minimiza la media cuadrada del error y con la técnica *steepest descent* actualiza los coeficientes iteración tras iteración. Sea la media del error:

$$J = E\{e^2(n)\} \tag{8.1}$$

donde E representa el valor esperado o esperanza matemática y el error está dado por

$$e(n) = d(n) - \hat{d}(n) \tag{8.2}$$

donde $d(n)$ es la señal deseada y $\hat{d}(n)$ es la salida del filtro adaptable (filtro FIR), figura 8.2. El filtro FIR puede expresarse como una multiplicación vectorial de la siguiente forma:

$$\hat{d}(n) = \bar{w}^T \bar{x}(n) \tag{8.3}$$

donde $\bar{w}$ es el vector que contiene los coeficientes del filtro adaptable y puede escribirse como: $\{w_1(n), w_2(n), ..., w_L(n)\}$ y $\bar{x}(n)$ es el vector de datos $\{x(n), x(n-1), ..., x(n-L+1)\}$, para un filtro de orden L. Combinando las ecuaciones anteriores, se obtiene que:

$$J = \sigma - 2\bar{w}^T \bar{p} + \bar{w}^T R \bar{w} \tag{8.4}$$

donde R es la matriz de autocorrelación de x dada por $R = E\{\bar{x}(n)\bar{x}(n)^T\}$ y $\bar{p}$ es el vector de cross-correlación entre d y x, es decir, $\bar{p} = E\{d(n)\bar{x}(n)\}$. La ecuación anterior tiene la forma $Q(\bar{x}) = \bar{x}^T A \bar{x}$, que representa una curva de superficie. El valor mínimo se obtiene cuando

$$\frac{\delta J}{\delta \bar{w}} = 0 \tag{8.5}$$

lo que implica que

$$-2\bar{p} + 2\bar{w}R = 0 \tag{8.6}$$

$$\bar{w}_{opt} = R^{-1}\bar{p} \tag{8.7}$$

llamada ecuación *Wiener-Hopf*. Para actualizar los coeficientes, se emplea la técnica *steepest descent* con la función gradiente del error:

$$\bar{w}(n+1) = \bar{w}(n) + \mu(-\nabla J) \tag{8.8}$$

El algoritmo complcto se resume en las siguientes ecuaciones:

$$\hat{d}(n) = \bar{w}^T(n) * \bar{x}(n) \tag{8.9}$$

$$e(k) = d(n) - \hat{d}(n) \tag{8.10}$$

$$\bar{w}(n+1) = \bar{w}(n) + 2 * \mu * \bar{x} * e(n) \tag{8.11}$$

El parámetro μ representa la ganancia del filtro adaptable y controla la rapidez con la que el filtro se aproxima al valor deseado.

8.3 Implementación del algoritmo LMS en el DSP TMS32C25

Instrucciones tales como **MPYA** o **MPYS** y **ZALR** permiten implementar de manera óptima un filtro FIR con coeficientes variables, esto reduce el tiempo de ejecución del programa. Además, para cada coeficiente del filtro en un instante de tiempo dado, el factor $2 * \mu * e(k)$ es una constante y puede calcularse una sola vez y almacenar el resultado en el registro T. De esta manera, el cálculo se vuelve únicamente una instrucción multiplicación/acumulación más redondeo. El bloque *B1* puede contener el vector de datos y el bloque *B0* los coeficientes.

El programa del filtro adaptable se presenta a continuación. En la parte de inicialización de variables se observa que el vector de coeficientes (bloque *B0*) se inicializa con ceros y el vector de datos (bloque *B1*) con $\{x(1), x(2), ..., x(L)\}$, repitiendo L veces la instrucción **IN**, donde L es el orden del filtro.

Enseguida se realiza el filtrado FIR, el valor en la salida del filtro se resta a la señal de referencia (obtenida del puerto PA2) y así se genera el error, que es enviado al puerto de salida PA3.

La constante **BETA** contiene el valor $2 * \mu$, de manera que al multiplicar por el error se obtiene $2 * \mu * e(k)$, por lo que esta constante queda almacenada en el registro T.

Finalmente, se actualizan los coeficientes del filtro (ecuación 8.11), donde el iésimo elemento se calcula: $w_i(n+1) = w_i(n) + beta * x_i(n)$. La instrucción **MPYA** realiza la multiplicación $beta * x_i(n)$, guardando el resultado en el registro P, y en la misma instrucción acumula el producto previo *i-1*.

```
*********************************
**  PROGRAMA PARA ALGORITMO LMS **
*********************************
```

```
        AORG   >0000
RESET   B      INIT
        AORG   >0020

** INICIALIZACION DE VARIABLES **
ONE     EQU    >0060
BETA    EQU    >0061
ERR     EQU    >0062
ERRF    EQU    >0063
YN      EQU    >0064
XN      EQU    >0065
DN      EQU    >0066
ORDER   EQU    20
FRSTAP  EQU    >300
LASTAP  EQU    768+ORDER
COEFFP  EQU    >FF00
COEFFD  EQU    >0200
INIT    SOVM
        LDPK   0
        LACK   1
        SACL   ONE
        ZAC
        SACL   YN
        SACL   BETA
        SACL   ERR
        SACL   ERRF
        SACL   DN
        LARP   AR4
        LRLK   AR4,>61
        BLKP   MU2,*
        LRLK   AR4,>300
        RPTK   ORDER-1
        IN     *+,PA1
        RPTK   ORDER-2
        IN     DN,PA2
        IN     DN,PA2
        ZAC
        LRLK   AR4, 512+ORDER
        RPTK   ORDER-1
        SACL   *-

** EJECUCION DEL FILTRO FIR **
CICLO   CNFP
        MPYK   0
```

```
        LAC     ONE,14
        LARP    AR3
        LRLK    AR3,LASTAP
        RPTK    ORDER-1
        MACD    COEFFP,*-
        CNFD
        APAC
        SACH    YN,1
        NEG
        ADD     DN,15
        SACH    ERR,1
        OUT     ERR,PA3

** ADAPTACION DE LOS COEFICIENTES **
        LT      ERR
        MPY     BETA
        PAC
        ADD     ONE,14
        SACH    ERRF,1
        MAR     *+
        IN      XN,PA1
        LAC     XN
        SACL    *
        LRLK    AR1,ORDER-1
        LRLK    AR2,COEFFD
        LRLK    AR3,LASTAP+1
        LT      ERRF
        MPY     *-,AR2
        SPM     1
ADAPT   ZALR    *,AR3
        MPYA    *-, AR2
        SACH    *+,AR2
        BNZ     ADAPT,*-,AR1
        CALL    NUEVO
        B       CICLO
NUEVO   IN      DN,PA2
        SPM     0
        RET

*** Valor 2MU ***
MU2     DATA    >1999
```

A continuación se aplica el filtro adaptable LMS a un problema de cancelación de ruido. Se tiene una señal que es contaminada con una senoidal de cierta frecuencia (60 Hz por

ejemplo). Se conoce una versión del ruido, aun cuando posee diferente amplitud y fase. La señal contaminada con ruido es la entrada al sistema:

$$d(n) = s(n) + A_0 cos(w_0 n + \phi_0)$$

y la referencia conocida es:

$$u(n) = Acos(w_0 n)$$

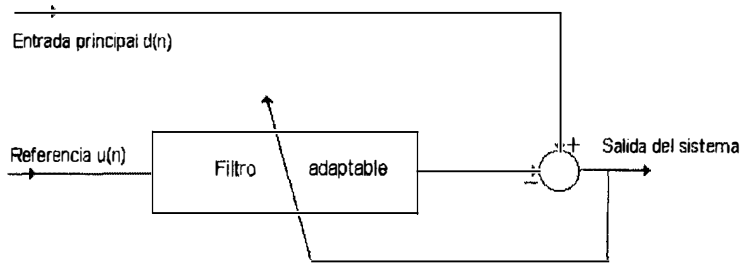


Figura 8.3. Cancelación de ruido

La figura 8.3 presenta un diagrama de un cancelador de ruido adaptable. El anexo presentado al final del capítulo contiene dos secuencias de datos utilizados por el simulador del TMS32C25 para analizar el filtro adaptable. La primera columna presenta la entrada principal al sistema $d(n)$ y la segunda columna la señal de referencia. En la figura 8.4a se observa la señal contaminada con ruido y en la figura 8.4b la salida del filtro adaptable junto con la señal original, que es una senoidal de 200Hz, y el ruido es una señal de 60 Hz, asumiendo una frecuencia de muestreo de 1000Hz. En la figura 8.5 se observa el espectro de frecuencia de la señal antes y después del aplicar el filtro adaptable.

Ejercicios

- Genere dos archivos **datos.dat** y **ruido.dat** que contengan cada uno las secuencias de la tabla presentada al final de este capítulo.

- Investigue en la bibliografía el significado de la constante μ y encuentre un valor apropiado para este ejemplo.

- Utilizando los archivos generados como entrada y referencia del filtro adaptable, obtenga la salida del filtro adaptable.

- Grafique la TRF de las señales de entrada y salida del filtro adaptable.

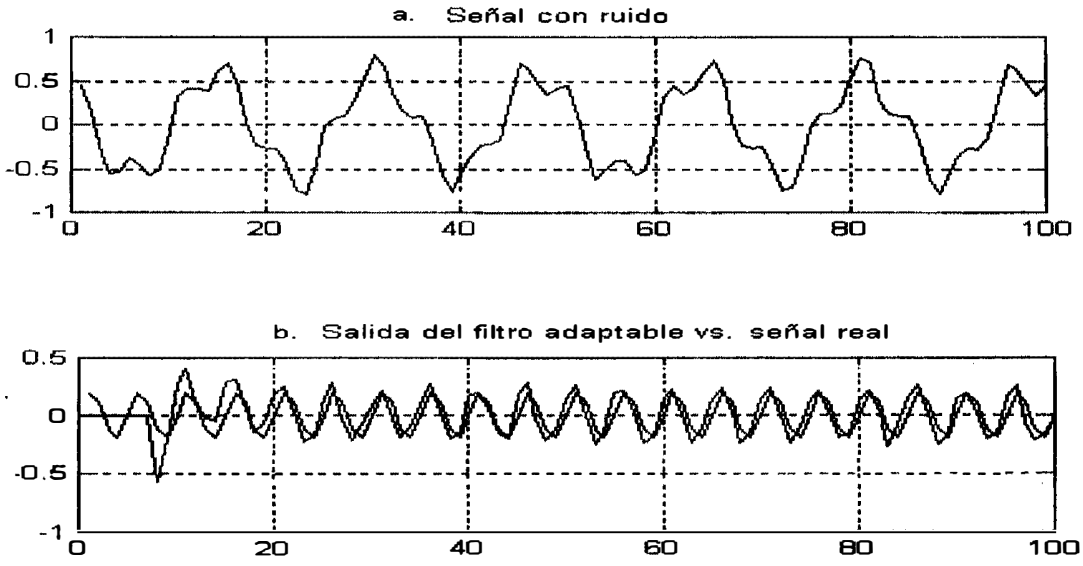


Figura 8.4. Entrada y salida del filtro adaptable

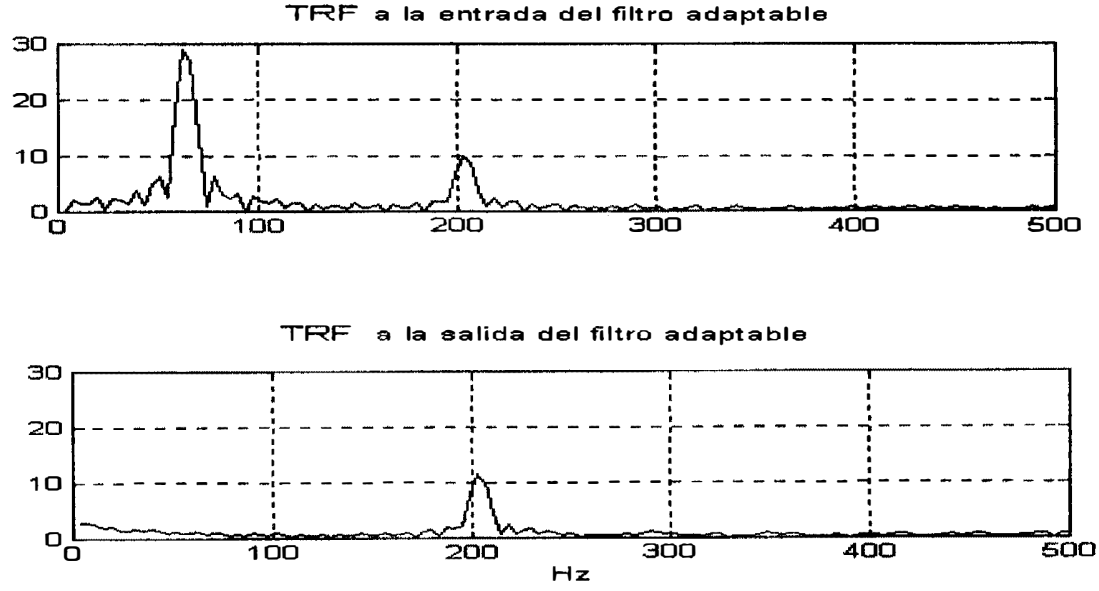


Figura 8.5. TRF de las señales a la entrada y salida del filtro adaptable

8.4 Algoritmo RLS

Este es un algoritmo recursivo que realiza una estimación de los coeficientes del filtro de orden N. La señal de error es la suma de los errores cuadrados y para minimizarla es preciso invertir la matriz de autocorrelación en cada iteración, lo que no sería computacionalmente eficiente. La matriz inversa se puede obtener, a partir de la inversa en la iteración anterior, únicamente multiplicando por un escalar, de esta forma el algoritmo RLS se vuelve tan atractivo. A continuación se describe el algoritmo.

Se tienen las variables:

m $\bar{d}(n)$, señal de referencia expresada en forma vectorial, es decir:

$$\bar{d}(n)^T = \{d(n), d(n-1), ..., d(1)\}$$

$\bar{w}(n)$, coeficientes del filtro, que en forma vectorial:

$$\{w_1(n), w_2(n), ..., w_N(n)\}$$

$\mathbf{X}$(n), datos expresados en forma matricial $n \times N$ en la iteración n:

$$\begin{pmatrix} \bar{x}^T(n) \\ \bar{x}^T(n-1) \\ \bar{x}^T(n-2) \\ \cdot \\ \cdot \\ \cdot \\ \bar{x}^T(1) \end{pmatrix}$$

donde $\bar{x}(n)^T = \{x(n), x(n-1), ..., x(n-N+1)$.

Esta técnica se basa en una minimización exacta de la suma cuadrática del error en cada iteración. El error está dado por:

$$e(n) = d(k) - \bar{w}(n)^T \bar{x}(k) \tag{8.12}$$

donde:

$e(n)$: error en la iteración n
$d(n)$: señal de referencia
$\bar{w}(n)$: coeficientes del filtro
$\bar{x}(n)$: vector de datos

Para minimizar el error se requiere que:

$$\bar{d}(k) = \mathbf{X}(\mathrm{n})\bar{\mathrm{w}}(\mathrm{n}) \tag{8.13}$$

que tiene la forma

$$Ax = B$$

Para el caso sobredeterminado, $n > N$, la solución se obtiene multiplicando ambos lados de la ecuación por la matriz $\mathbf{X}(\mathrm{n})^{\mathrm{T}}$, lo cual nos lleva a la siguiente ecuación:

$$\mathbf{X}(\mathrm{n})^{\mathrm{T}}\mathbf{X}(\mathrm{n})\bar{\mathrm{w}}(\mathrm{n}) = \mathbf{X}(\mathrm{n})^{\mathrm{T}}\bar{\mathrm{d}}(\mathrm{n}) \tag{8.14}$$

y la solución para $w(n)$ es:

$$\bar{w}_{opt}(n) = R^{-1}p(n) \tag{8.15}$$

donde:

$$R = \mathbf{X}(\mathrm{n})^{\mathrm{T}}\mathbf{X}(\mathrm{n})$$
$$p(n) = \mathbf{X}(\mathrm{n})^{\mathrm{T}}\bar{\mathrm{d}}(\mathrm{n})$$

Para resolver la ecuación (8.15) de manera recursiva, se calcula $\bar{w}(n+1) = f(\bar{w}(n))$, y utilizando inversión de matriz, se obtiene la nueva inversa de la matriz a la iteración $n+1$ a partir de la inversa a la iteración n, multiplicando únicamente por un escalar, de otra forma sería preciso invertir la matriz en cada iteración. De esta manera se llega a las ecuaciones que definen el algoritmo RLS:

$$K(n+1) = \frac{R^{-1}(n)\bar{x}(n+1)}{\gamma + 1} \tag{8.16}$$

donde $K(n)$ se define como la ganancia de Kalman mediante $K(n) = R^{-1}(n)\bar{x}(n)$ y γ está dada por $\gamma = \bar{x}^T(n+1)R^{-1}(n)\bar{x}(n+1)$. La matriz inversa se obtiene con la ecuación:

$$R^{-1}(n+1) = R^{-1}(n) - K(n+1)R^{-1}\bar{x}^T(n+1)R^{-1}(n) \tag{8.17}$$

y, finalmente, los coeficientes del filtro se actualizan:

$$\bar{w}(n+1) = \bar{w}(n) + K(n+1)e(n+1) \tag{8.18}$$

Señal	de	datos	Señal	de	referencia
382F	EA88	E732	3B81	D734	0BFE
D5EF	588A	9C95	1B40	0405	DDB5
B88B	53C6	B4F3	0405	1B40	CC39
BB0D	3C69	D1C6	EC39	2EA7	C203
D01C	2D79	DDA2	D734	3B81	C081
C4F1	34AD	DBD1	C7EB	4000	C7EB

B6D7	387E	EDF0	C081	3B81	D734
BE4E	0F30	248C	C203	2EA7	EC39
F629	D27B	5784	CC39	1B40	0405
2A16	B1A6	50DC	DDB5	0405	1B40
35BA	BFAD	3D07	F402	EC39	2EA7
3404	CBFE	2AB6	0BFE	D734	3B81
301C	CA55	3948	224B	C7EB	4000
502A	B5E9		33C7	C081	
5984	BF9C		3DFD	C203	
3847	F66E		3F7E	CC39	
0477	293D		3815	DDB5	
E1A6	39C4		28CB	F402	
DDBD	2CDE		13C7	0BFE	
DE05	3511		FBFB	224B	
C678	4CD8		E4C0	33C7	
A0D6	5C72		D159	3DFD	
9AD3	40AF		C47F	3F7E	
C7C5	069D		C000	3815	
FE3E	E1B8		C47F	28CB	
08E1	DCA8		D159	13C7	
0B62	DF2B		E4C0	FBFB	
24BC	C4BC		FBFB	E4C0	
4565	9FFB		13C7	D159	
6594	A4C1		28CB	C47F	
550C	CCFB		3815	C000	
2EE4	F9R9		3F7E	C47F	
1540	0FD3		3DFD	D159	
0A7A	102D		33C7	E4C0	
0BD2	1B14		224B	FBFB	
E83C	44D8		0BFE	13C7	
B5DC	6196		F402	28CB	
9E37	58CB		DDB5	3815	
B7E2	2545		CC39	3F7E	
D177	1128		C203	3DFD	
E1B2	0D27		C081	33C7	
E2B5	OBB2		C7EB	224B	

Tabla 8.1. Señal de datos y de referencia

Capítulo 9

Introducción al programa MATLAB

9.1 Operaciones básicas con MATLAB

MATLAB es un programa de cómputo basado en operaciones matriciales y orientado al cálculo numérico por computadora. Este programa resulta muy útil para científicos e ingenieros ya que puede resolver problemas numéricos de una manera más sencilla que utilizando lenguajes de programación, tales como Fortran o C. El nombre MATLAB se deriva de MATrix LABoratory (laboratorio de matrices).

El presente capítulo da una introducción al uso de MATLAB. Se describen brevemente las principales funciones para generar matrices y vectores, funciones aritméticas y ejemplos para resolver problemas prácticos.[1] La ayuda en línea que presenta el mismo programa MATLAB resulta también de gran utilidad, para lo cual el comando **help** *namefunction* proporciona toda la información referente a la función deseada.

MATLAB se encuentra disponible para una gran variedad de ambientes: Sun /Apollo /VAXstation /HPworkstations, VAX, Micro VAX, Gould, PC y AT compatibles, 80386 y 80486, y Macintosh.[2]

En la mayoría de los sistemas se ejecuta MATLAB mediante el comando **matlab** en el sistema operativo y se finaliza la ejecución con los comandos **quit** o **exit**. También puede ser accesado desde un menú o al activar el ícono correspondiente. Si el ambiente de trabajo lo permite (por ejemplo UNIX o Windows), es conveniente tener activo MATLAB simultáneamente con un editor de texto.

Después de ejecutar MATLAB, todo aquello que sea introducido por el teclado de la computadora será interpretado y evaluado por MATLAB. Las variables se generan al asignarles una cierta expresión:

$$variable = expresión$$

[1] Para una mayor consulta se recomienda los manuales *User's Guide* y *Reference Guide*.

[2] Existe una edición para estudiantes a un precio accesible. Mayor información puede obtenerse vía e-mail en: *info@mathworks.com*.

donde las expresiones pueden ser operaciones, funciones u otras variables. Al evaluar las expresiones se genera una matriz que contiene el resultado y es desplegada en la pantalla. Al incluir al final de la expresión un ";" el resultado no se muestra en la pantalla y solamente se asigna a la variable seleccionada. Todas las variables generadas permanecen en la memoria dentro del espacio de trabajo hasta finalizar la ejecución de MATLAB. Si únicamente se le proporciona a MATLAB la expresión que se evaluará, automáticamente se asigna el resultado a una variable denominada **ans**.

Toda expresión se evalúa después de teclear RETURN. El comando **who** o **whos** despliega la información sobre las variables que se encuentran en el actual espacio de trabajo. Con el comando **clear** *nombre* se borra del espacio de trabajo la variable deseada, o bien, **clear** por sí solo borra todas las variables.

Al finalizar la ejecución de MATLAB se pierden todas las variables del espacio de trabajo, sin embargo, el comando **save** permite salvar todas las variables en un archivo llamado **matlab.mat**. Al ejecutar MATLAB nuevamente, se recuperarán las variables con el comando **load**.

9.1.1 Funciones con matrices

MATLAB trabaja y realiza operaciones numéricas esencialmente con un solo tipo de objetos: matrices con elementos complejos. En algunas situaciones matrices de 1 por 1 se interpretan como escalares y matrices de un solo renglón o columna como vectores. Las matrices son introducidas en diferentes formas:

- Por teclado, listando todos los elementos
- Mediante funciones que generan matrices
- Utilizando un editor de texto
- Cargando el correspondiente archivo de datos

A continuación se presentan dos formas de generar la matriz **A**:

```
A = [2 5 4; 3 6 8; 9 1 7]
```

o bien,

```
A = [
2 5 4
3 6 8
9 1 7];
```

Para generar matrices complejas:

```
A = [1 3; 6 4] + i*[5 7; 8 9]
A = [1+5i 3+7i; 6+8i 4+9i]
```

Si se ha utilizado i o j como variables, será necesario generar una nueva unidad imaginaria, por ejemplo: **ii=sqrt(-1)**.

Para generar matrices de un gran tamaño es recomendable hacerlo mediante un editor ASCII, de manera que puedan corregirse los errores fácilmente. El comando **load** *name.ext* asigna a la variable **name** el contenido del archivo *name.ext*.

Las siguientes operaciones aritméticas con matrices se encuentran disponibles en MATLAB:

+	suma
-	substracción
*	multiplicación
∧	potencia
\	división por izquierda
/	división por derecha

Si el tamaño de las matrices es incompatible con la operación por realizar, se obtendrá un mensaje de error. Estas operaciones se aplican a escalares o matrices de 1×1.

La operación de división de matrices se interpreta de la siguiente forma:

$x = A\backslash b$ es la solución de $A * x = b$
$x = b/A$ es la solución de $x * A = b$

Las operaciones de suma y substracción se realizan sobre matrices de iguales dimensiones y sumando o restando término a término. De igual forma es posible realizar operaciones, tales como *, ∧, \ y /, término a término, precediendo la operación con un punto. Por ejemplo: [1,2,3,4].∧2 producirá [1,4,9,16].

Algunas funciones útiles para la construcción de matrices son las siguientes:

`eye`	genera la matriz identidad
`zeros`	genera matriz de ceros
`ones`	genera matriz de unos
`diag`	crea o extrae la diagonal de una matriz
`triu`	parte triangular superior de una matriz
`tril`	parte triangular inferior de una matriz
`rand`	genera una matriz con datos aleatorios
`hilb`	matriz Hilbert
`magic`	genera un cubo mágico
`toeplitz`	véase `help toeplitz`

Funciones para construcción de matrices

Por ejemplo, **zeros(m,n)** genera una matriz de tamaño m por n de ceros, y **zeros(n)** una de tamaño n por n. Si A es una matriz, **zeros(size(A))** produce una matriz de ceros del mismo tamaño de A.

9.1.2 Instrucciones for, while, if y relaciones de comparación

Estas declaraciones de control operan de manera similar que la mayoría de los lenguajes de cómputo.

For

Genera un vector de tamaño n, por ejemplo:

```
x=[];
for i=1:n,
   x=[x,i^2],
end
```

Para generar el mismo vector en orden inverso:

```
x=[];
for i=n:-1:1,
    x=[x,i^2],
end
```

Las siguientes instrucciones generan la matriz *Hilbert* de tamaño n por n:

```
for i = 1:m
  for j = 1:n
    H(i,j) = 1/(i+j-1);
  end
end
H
```

While

La forma general de un ciclo utilizando **while** es la siguiente:

while *relación o comparación*
  comandos y declaraciones
end

Las declaraciones se ejecutarán mientras la comparación sea verdadera. El siguiente ejemplo muestra el menor número entero positivo n, tal que $2^n \geq a$:

```
n = 0;
while 2^n < a
    n = n + 1;
end
n
```

If

La forma general de la declaración **if** es la siguiente:

```
if comparación
   comandos y declaraciones
end
```

Las declaraciones se ejecutan únicamente si la comparación es verdadera.

Relaciones de comparación

Los operadores de relación en MATLAB son:

$<$	menor que
$>$	mayor que
$<=$	menor o igual que
$>=$	mayor o igual que
$==$	igual
$\sim=$	desigual

En una declaración de asignación se utiliza "=", mientras que "==" para comparación. Las operaciones de relación generan por sí solas un resultado: 1 o 0, dependiendo de si son verdadero o falso; por ejemplo, `3<5, 3>5` producen 1 y 0, respectivamente. En el caso de matrices, el resultado es una matriz con unos y ceros, de acuerdo con la relación entre los correspondientes elementos. Las operaciones lógicas se realizan mediante:

&	and
|	or
$\sim$	not

9.1.3 Funciones escalares, vectoriales y matriciales

Funciones escalares

Existen ciertas funciones que operan esencialmente en escalares y aplicarlas a matrices es similar a efectuar la operación sobre cada elemento de la matriz; las más comunes son las siguientes:

`abs`	valor absoluto o módulo de un número complejo
`angle`	fase del número complejo
`sqrt`	raíz cuadrada
`real`	parte real del número complejo
`imag`	parte imaginaria del número complejo
`conj`	conjugado del número complejo
`round`	redondeo de un número
`fix`	truncamiento
`sign`	signo de un número
`rem`	residuo de división
`exp`	función exponencial
`log`	logaritmo natural
`log10`	logaritmo base 10

Operaciones elementales

`sin`	función seno
`asin`	arco seno
`cos`	función coseno
`acos`	arco coseno
`tan`	función tangente
`atan`	arco tangente
`atan2`	arco tangente en los 4 cuadrantes
`sinh`	seno hiperbólico
`cosh`	coseno hiperbólico
`tanh`	tangente hiperbólica
`asinh`	seno hiperbólico inverso
`acosh`	coseno hiperbólico inverso
`atanh`	tangente hiperbólica inversa

Funciones trigonométricas

Funciones vectoriales

Estas funciones operan esencialmente sobre un vector (columna o renglón) y, al aplicarlas sobre una matriz m por n ($m \geq 2$), generan un vector renglón cuyos elementos son el resultado de aplicar la función a cada columna. Para afectuar la operación renglón por renglón se debe utilizar la transpuesta. A continuación se presentan algunas de las funciones más importantes.

`min`	valor mínimo
`max`	valor máximo
`sort`	ordena de manera ascendente
`sum`	suma de los elementos
`prod`	producto de los elementos
`median`	mediana
`mean`	media
`std`	desviación estándar
`any`	verdadero si cualquier elemento es no-cero
`all`	verdadero si todos los elementos son no-cero

Funciones vectoriales

Para calcular el máximo elemento de una matriz se utiliza **max(max(A))**.

Funciones matriciales
Gran parte de la potencialidad de MATLAB proviene de las funciones matriciales, como las siguientes:

`eig`	eigenvalores y eigenvectores
`chol`	factorización Cholesky
`svd`	descomposición en valores singulares
`inv`	inverso de una matriz
`lu`	factorización LU
`qr`	factorización QR
`hess`	forma Hessenberg
`schur`	descomposición schur
`rref`	forma echelon de renglón reducido
`expm`	exponencial de una matriz
`sqrtm`	raíz de una matriz
`poly`	polinomio característico
`det`	determinante
`size`	tamaño de una matriz
`norm`	normalización
`cond`	condición
`rank`	rango

Funciones matriciales

9.1.4 Submatrices y vectores

Utilizar submatrices y vectores en operaciones permite minimizar el uso de ciclos **for end** y así hacer más eficiente a MATLAB. La expresión `1:5` equivale al vector `[1 2 3 4 5]`. En

resumen, el número antes de los dos puntos indica el índice inicial y el segundo número el valor final con un incremento = 1. Para tener incrementos diferentes a la unidad:

```
0.2:0.2:1.2
```

genera `[0.2, 0.4, 0.6, 0.8, 1.0, 1.2]`.

De esta manera, para generar, por ejemplo, una tabla de senos sin necesidad de un ciclo:

```
x = [0.0:0.1:2.0]'
y = sin(x)
[x y]
```

La notación ":" puede utilizarse para generar submatrices a partir de una matriz. Algunos ejemplos:

`A(1:4,3)` es el vector columna que contiene los primeros cuatro elementos de la tercera columna de A.

`A(:,3)` es la tercera columna de A, y `A(1:4,:)` es una matriz formada por los primeros cuatro renglones de A.

`A(:,[2 4])` utiliza un vector como subíndice y genera las columnas 2 y 4 de A.

`A(:,[2 4 5]) = B(:,1:3)` reemplaza las columnas 2, 4 y 5 de A con las primeras tres columnas de B. También es posible realizar operaciones con índices:

```
A(:,[2,4]) = A(:,[2,4])*[1 2;3 4]
```

Ejemplo 1

Se tiene un vector de datos **x** de longitud N. En diferentes aplicaciones del procesamiento digital de señales se desean obtener ventanas del vector **x** de tamaño L a intervalos K, permitiendo inclusive un traslape. Esto se logra con los siguientes comandos (las variables x, **K**, **L**, **N** han sido definidas previamente):

```
index=0;
while (index*K+L)<=N
   XW(:,index+1)=x(index*K+1:index*K+L);
   index=index+1;
end
```

9.2 Generación de archivos .m

Todos los comandos y funciones analizadas pueden introducirse a MATLAB desde el teclado y ser ejecutados en línea; sin embargo, existen archivos con extensión .m que pueden contener todos los comandos ejecutados por MATLAB. Así, por ejemplo, el siguiente archivo llamado **datos.m** puede utilizarse para generar la matriz A.

```
  A = [
1 2 3 4
5 6 7 8
];
```

Para generar la matriz A basta con teclear **datos** como un comando dentro de MATLAB. De esta forma es más sencillo efectuar correcciones sobre el archivo **datos.m** en lugar de introducir nuevamente todos los elementos de la matriz. Dentro de un archivo .m se puede llamar cualquier otro archivo .m o función de MATLAB.

Es posible crear nuevas funciones dentro de MATLAB para resolver un problema específico, aplicando a su vez las funciones ya existentes dentro del mismo programa. Todas las variables utilizadas dentro de la función son locales (se pierden al finalizar la ejecución de la función), pero pueden definirse variables globales. Una función en MATLAB es también un archivo con extensión .m, con la diferencia de que la primera línea del archivo debe contener la declaración de la función. La declaración de cualquier función es del tipo:

```
function a = nombre (b)
```

en donde **b** es el parámetro de entrada y **a** el parámetro de salida. Todas las variables intermedias se perderán después de ejecutada la función. Los comentarios sobre la función se escriben inmediatamente después de la declaración, seguidos del símbolo % y se presentarán como ayuda al emplear el comando **help nombre**.

Ejemplo 2

Se desea una función que genere la distribución normal de probabilidad definida por:

$$\frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{1}{2}(\frac{x-\mu}{\sigma})^2)$$

```
function [gss,axs]=gaussia(var,mean)
%  Function [gss,axs]=gaussia(var,mean)
%  Mauricio Ortega 1995
```

```
% Funcion que produce una distribucion de probabilidad gaussiana
% de parametros  variance = var  y   mean
% gss es el vector que contiene la distribucion
% axs es el vector que contiene la escala
% La funcion utiliza 512 bins

limit1=mean-3*sqrt(var);
limit2=mean+3*sqrt(var);
axs=[limit1:abs(limit2-limit1)/512:limit2];
gss=(1/(var*sqrt(2*pi))).*exp(-(1/(var*2))*((axs-mean).^2));
```

La función anterior genera la distribución gaussiana de probabilidad de parámetros **var,mean**. El resultado se escribe en dos variables, **gss** y **axs**, que contienen la distribución y la escala apropiada para la graficación. La función anterior, evaluada para valores de variancia = 1 y media = 0, se obtendrá tecleando:

```
[gss, axx] = gaussia(1,0).
```

En caso de que el número de parámetros de entrada sea variable y desconocido por el programador es posible utilizar el comando **nargin,** el cual tomará como valor el número de argumentos de entrada a la función. La siguiente función genera una matriz de tamaño m por n cuyos elementos poseen una distribución de probabilidad uniforme entre a y b. En caso de omitirse estos parámetros de entrada, se seleccionará automáticamente una distribución uniforme entre 0 y 9.

```
function y = randint(m,n,a,b)
% randint matriz de elementos enteros aleatorios
% y = randint(m,n,a,b)
% calcula una matriz de orden m por n con elementos escritos en vector a y b

if nargin<3, a = 0; b = 9; end
y = floor((b-a+1)*rand(m,n))+ a;
```

De esta forma se pueden añadir diferentes funciones a las ya existentes dentro de MATLAB, incrementando la potencialidad de este programa. Los *toolbox* de MATLAB son precisamente librerías que contienen funciones para aplicaciones específicas: procesamiento de imágenes, control, redes neuronales, *fuzzy logic*, *wavelets*, etc.

9.3 Entrada de datos y de texto

Para hacer más cómoda y amigable la presentación de datos al usuario es posible incluir cadenas de texto, por ejemplo:

```
s = 'Esta es una prueba'
```

asigna una cadena de caracteres a la variable **s**. Para mostrar un determinado texto se puede utilizar la función **disp**, por ejemplo:

```
disp('este mensaje sera desplegado')
```

Los mensajes de error se pueden desplegar con la función **error**:

```
error('Error en las dimensiones de la matriz')
```

Al encontrar la función **error** dentro de un archivo .m a la ejecución del archivo. La entrada de datos se puede realizar con la función **input**, por ejemplo:

```
iter = input('Proporcione número de iteraciones:')
```

La ejecución se detendrá hasta introducir el dato solicitado.

9.4 Formatos de salida

Todos los cálculos realizados por MATLAB se efectúan con doble precisión, pero el formato con el que se despliegan se selecciona con los siguientes comandos:

`format short`	punto fijo con 4 decimales
`format long`	punto fijo con 14 decimales
`format short e`	notación científica con 4 decimales
`format long e`	notación científica con 15 decimales
`format hex`	formato hexadecimal

Por ejemplo, el vector `x = [4/3 1.2345e-5]` se mostrará de la siguiente forma:

```
format short:
1.3333    0.0000

format short e:
1.3333e+10     1.2345e-6

format long:
1.33333333333333     0.00000123450000
```

```
format long e:
1.333333333333333e+000      1.234500000000000e-005

format hex:
3ff5555555555555      3ee9e3abe16fc70d
```

9.5 Graficación con MATLAB

MATLAB permite realizar tanto curvas planas como superficies en tres dimensiones. Las principales funciones para graficar se resumen en la siguiente tabla:

`plot`	gráfica lineal x-y
`loglog`	gráfica x-y con ejes logarítmicos
`semilogx`	gráfica x-y con eje x logarítmico
`semilogy`	gráfica x-y con eje y logarítmico
`polar`	gráfica en coordenadas polares
`mesh`	gráfica en tres dimensiones
`contour`	curvas de superficie
`bar`	gráfica de barras
`stairs`	gráfica de barras
`title`	título de la gráfica
`xlabel`	unidades de la ordenada
`ylabel`	unidades de la abscisa
`text`	incluye texto dentro de la imagen
`gtext`	pone texto con mouse
`grid`	cuadricula la imagen
`axis`	modifica la escala de los ejes
`hold`	congela la gráfica en la pantalla
`clg`	borra la gráfica anterior
`subplot`	divide en varias subgráficas
`ginput`	datos desde teclado o mouse

Tabla de operaciones elementales de graficación

9.5.1 Curvas planas

El comando **plot** genera gráficas lineales x-y. Los vectores **X** y **Y** deben ser del mismo tamaño. El comando **plot(X,Y)** grafica **X** vs **Y**, donde **X** corresponde a las ordenadas y **Y** a las abscisas. Por ejemplo, para graficar una función seno (figura 9.1):

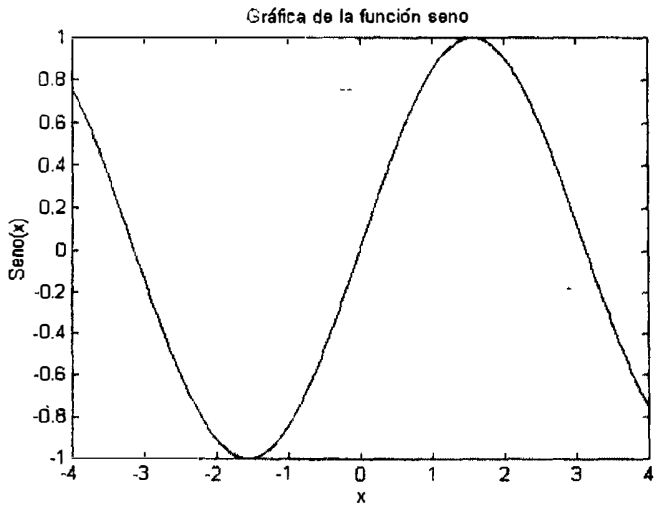


Figura 9.1. Función seno

```
x = -4:.01:4;
y = sin(x);
plot (x,y)
```

Para incluir el título y designar los ejes:

```
title('Grafica de la funcion seno');
xlabel('x')
ylabel('Sin(x)')
```

También es posible graficar varias señales en una sola (figura 9.2), por ejemplo:

```
x=0:.01:2*pi;
y1=sin(x);
y2=sin(2*x);
y3=sin(4*x);
plot(x,y1,x,y2,x,y3);
```

Otra forma de hacerlo es mediante el comando **hold on** que congela la imagen y las subsecuentes gráficas se encimarán a la gráfica inicial (figura 9.3). El comando **hold off** "descongela" la gráfica. Por ejemplo, para modificar el tipo de línea y los colores se puede usar:

```
x=0:.01:2*pi;
y1=sin(x);
y2=sin(2*x);
y3=sin(4*x);
plot(x,y1,'--',x,y2,':',x,y3,'+');
```

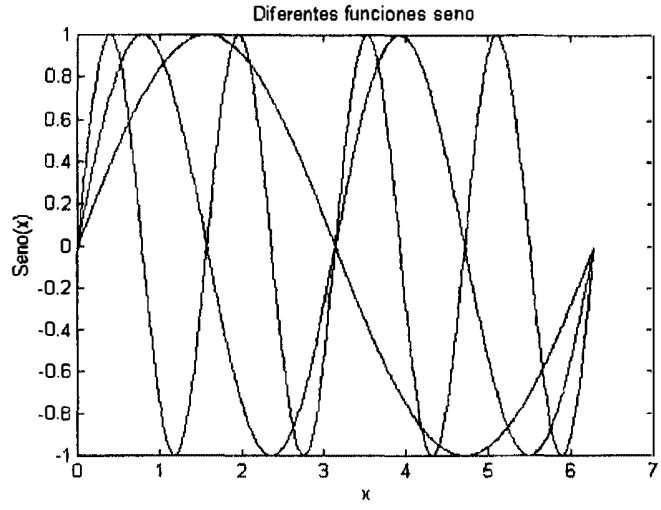


Figura 9.2. Varias funciones seno

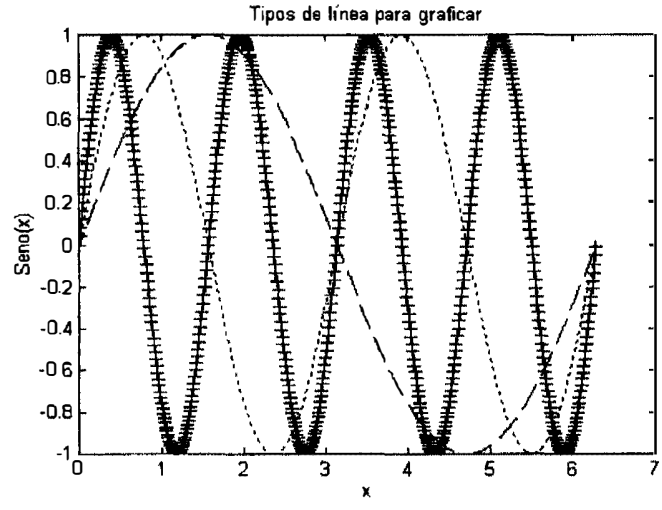


Figura 9.3. Tipos de línea para graficar

La ayuda que se anexa a la función **plot** muestra las diferentes opciones. Los tipos de línea, marca y color son:

Tipos de línea	**Tipos de marca**	**color**
sólida (-)	punto (.)	amarillo (y)
punteada (–)	más (+)	magenta (m)
puntos(:)	asterisco(*)	cyan (c)
punto-línea (-.)	círculo (o)	negro (b)
	marca-x (x)	rojo (r)
		azul (b)
		verde (g)
		blanco (w)

El comando **print** almacena la gráfica en un archivo con un determinado formato, el default es el formato PostScript. Véase **help print**.

Ejemplo 3

Se desea obtener la gráfica de las curvas equipotenciales generadas por un dipolo eléctrico. Dicho potencial se define como:

$$R = C_v\sqrt{\cos\theta}$$

donde R es la distancia o radio del dipolo a un punto dado y θ es el ángulo en radianes de la perpendicular que define al dipolo a dicho punto y C_v es una constante. De manera similar, el campo eléctrico definido por:

$$R = C_E \sin\theta^2$$

Los siguientes comandos generan las gráficas en coordenadas polares para $C_v = C_E = \pm 1$ (figura 9.4).

```
clear
theta=-pi/2:pi/64:pi/2;
N=length(theta);
Rpq=sqrt(cos(theta));
Rnq=-sqrt(cos(theta));
Rpe=(sin(theta))^2;
Rpe=(sin(theta)).^2;
Rne=-(sin(theta)).^2;
hold on
```

```
polar(theta,Rpq);
polar(theta,Rnq);
polar(theta,Rpe,'--');
polar(theta,Rne,'--');
hold off
```

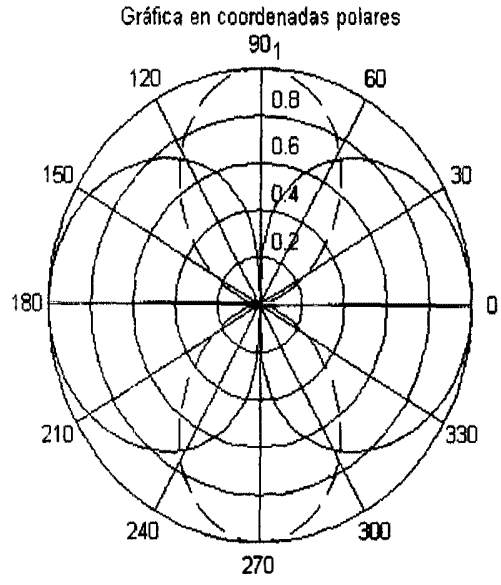


Figura 9.4. Gráfica polar

Ejemplo 4

Existen diferentes distribuciones de probabilidad analizadas por MATLAB. La función **hist(x)** de MATLAB permite observar el histograma del vector x, que puede aproximarse a una función de densidad de probabilidad, normalizando el área total a uno. La siguiente función obtiene la gráfica de la distribución de probabilidad del vector x:

```
function [N2,X]=pdf(x,N);
% function [N2,X]=pdf(x,N)
% Mauricio Ortega 1995
% Funcion para obtener la distribucion de probabilidad de x
%        x es la variable aleatoria
%        N numero de bins

delta=abs(max(x)-min(x))/N;
[N1,X]=hist(x,N);
```

```
area=sum(N1*delta);
N2=N1./area;
area2=sum(N2.*delta);
bar(X,N2)
```

La función **randn** genera números aleatorios con una distribución de probabilidad *normal* con parámetros media=0 y variancia 1. Los siguientes comandos realizan la gráfica comparativa entre una secuencia de datos generados aleatoriamente con distribución normal de probabilidad y la distribución teórica esperada (figura 9.5).

```
clear
data=randn(200,1);
pdf(x,20);
hold on
[g,ax]=gaussia(1,0);
plot(ax,g)
hold off
```

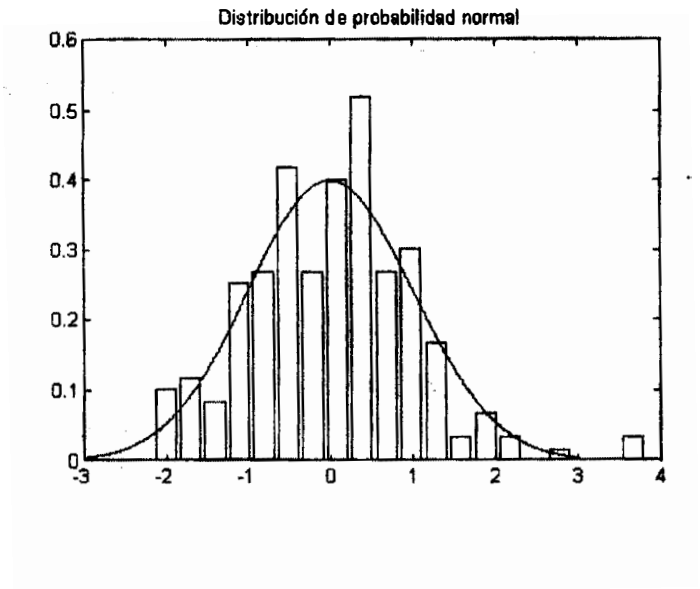


Figura 9.5. Distribución de probabilidad normal

Problema

Genere una matriz con distribución uniforme de probabilidad. Obtenga la media (función **mean**) de los renglones de la matriz para generar un vector. Grafique la distribución de probabilidad de este último vector y compare con la distribución normal. El *teorema del límite central* nos dice que debe tener una distribución normal.

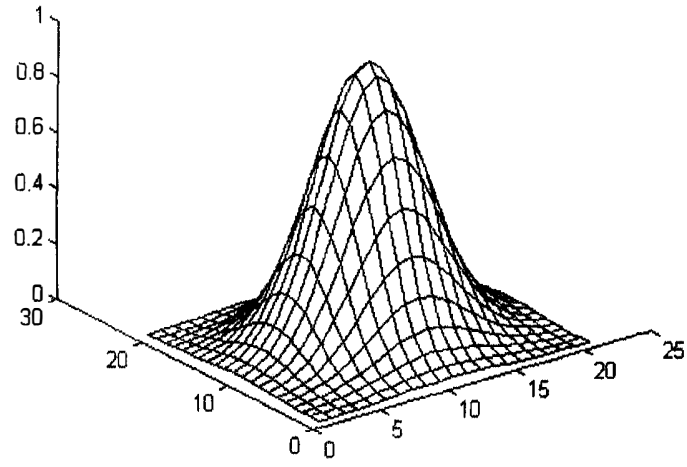


Figura 9.6. Figura en 3-D

9.5.2 Gráficas en tres dimensiones

El comando para gráficas en tres dimensiones **plot3** es similar a **plot**. Si x, y y z son vectores del mismo tamaño, entonces **plot3(x,y,z)** produce una gráfica en perspectiva que genera una curva lineal, pasando a través de los puntos cuyas coordenadas son los respectivos elementos de x, y y z. De igual forma, el título y los ejes se pueden añadir y para el eje z se utiliza el comando **zlabel** (figura 9.6).

Gráficas mediante superficies y malla

Gráficas de tipo malla en tres dimensiones pueden generarse mediante el comando **mesh(z)**. El comando **mesh(z)** genera una gráfica en perspectiva tridimensional de los elementos de la matriz z. De manera similar, una gráfica del tipo de superficie se genera con el comando **surf**.

Ejemplo 5

```
xx=-2:.2:2;
yy = xx;
[x,y]=meshgrid(xx,yy);
z=exp(-x.^2-y.^2);
mesh(z);
```

La función **meshgrid** permite generar los índices requeridos para producir gráficas en tercera dimensión (3-D) dentro de un cierto intervalo, véase **help meshgrid**.

Capítulo 10

Señales y sistemas

10.1 Señales analógicas y digitales

De manera general, una señal analógica es una función continua en el tiempo. Para procesar esta señal con una computadora, debe ser muestreada cada T segundos, generando así una secuencia de datos que se convierte en una señal digital. Dicha secuencia se representa usualmente como $x[n] = f(kT)$.

MATLAB sólo trabaja con secuencias digitales. La función `plot` grafica las secuencias discretas como señales continuas. Para observar la secuencia discreta como una serie de impulsos se puede utilizar la función de graficación `stem`.

Ejemplo 1

Este ejemplo permite comparar entre una señal analógica y una digital, usando las funciones `plot` y `stem` de MATLAB, como se observa en la figura 10.1.

```
t=linspace(0,4,128);
f=exp(-0.5*t).*sin(2*pi*t);
subplot(2,1,1),plot(t,f),title('Formas Digital & Analogica');
xlabel('Senal digital'), ylabel('F(k)');
subplot(2,1,2), stem(t(1:4:128),f(1:4:128))
subplot(2,1,1),plot(t,f),title('Formas Digital & Analogica');
xlabel('Senal analogica'),ylabel('F(t)');
subplot(2,1,2), stem(t(1:4:128),f(1:4:128))
xlabel('Senal digital'), ylabel('F(k)');
```

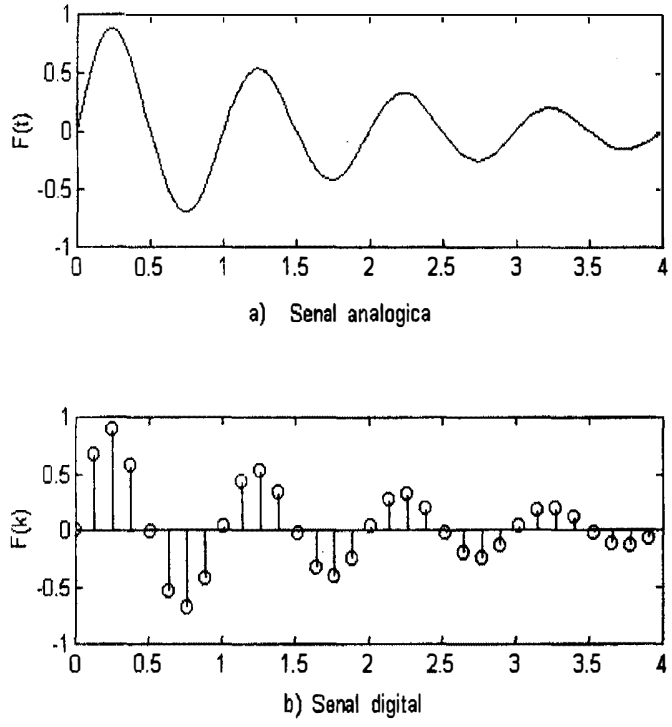


Figura 10.1. Señales analógica y digital

10.2 Secuencias discretas mediante MATLAB

A continuación se generan algunas secuencias discretas importantes utilizando MATLAB. Una variable de MATLAB no puede contener índices negativos o cero, sino únicamente índices positivos. Para generar un impulso asignamos `x(1)=1` en lugar de `x(0)=1`. Para el escalón unitario podemos establecer los valores unitarios a partir de un cierto índice (la mitad de la secuencia) que corresponde al índice cero. Las figuras 10.2 y 10.3 muestran estas secuencias.

10.2.1 Impulso unitario

El programa para obtener el impulso unitario es el siguiente:

```
x=zeros(50,1);
x(1)=1;
stem(x);
```

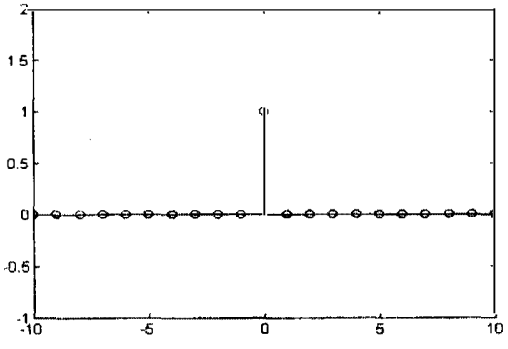


Figura 10.2. Impulso unitario

10.2.2 Escalón unitario

El programa para crear el escalón unitario es el siguiente:

```
x=zeros(100,1);
x(51:100)=ones(50,1);
indice=-50:49;
stem(indice,x)
```

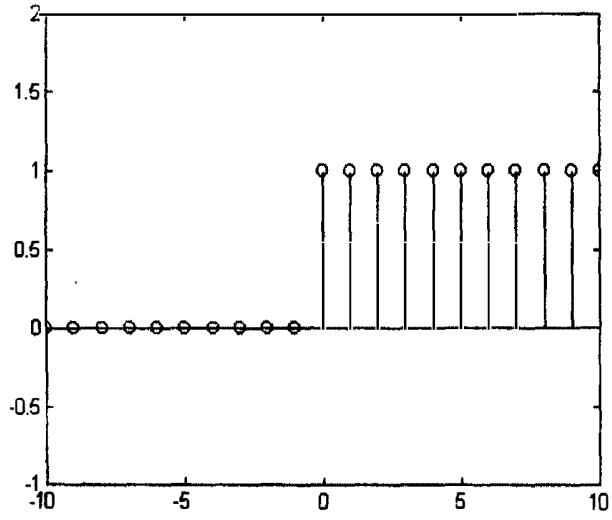


Figura 10.3. Escalón unitario

10.2.3 Senoidal

Otra secuencia discreta importante es la senoidal. Los parámetros de la senoidal son la frecuencia de muestreo y la frecuencia de la senoidal.

Ejemplo 2

Genere dos senoidales de 100 y 50 Hz. Utilice una frecuencia de muestreo de 500 Hz.

```
n=1:100;  %indice
T=1/500;  %periodo de muestreo=500 hz
y=cos(2*pi*100*n*T); %senoidal de 100 hz
plot(n*T,y);
x=cos(2*pi*50*n*T);  %senoidal de  50 hz
plot(n*T,x);
```

Problema

Una señal modulada en amplitud (AM) es comúnmente utilizada en comunicaciones. Esta señal tiene la siguiente forma:

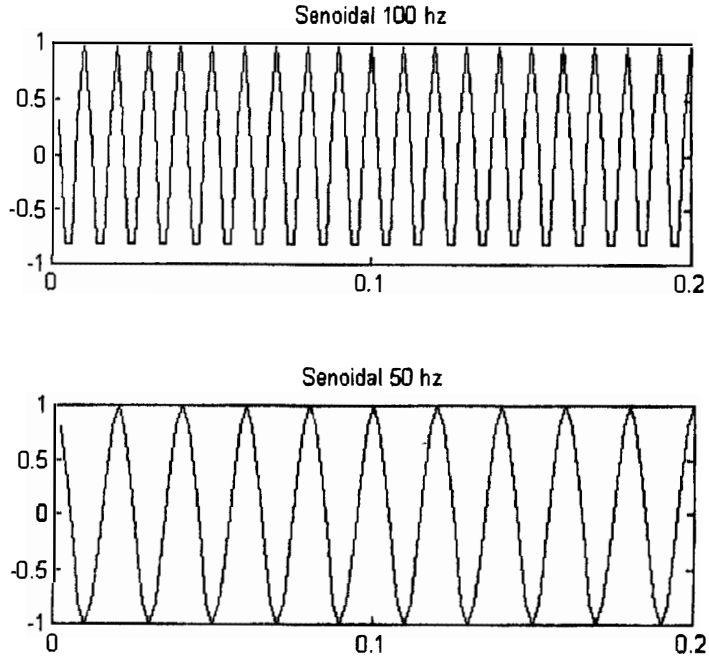


Figura 10.4. Senoidal

$$x_a(t) = A_c[1 + \mu m(t)]cos(2\pi f_c t)$$

donde $m(t)$ es el mensaje por transmitir, f_c es la frecuencia de la portadora y μ es el llamado *índice de modulación*. En el caso especial de que el mensaje sea una senoidal:

$$x_a(t) = A_c[1 + \mu sin(2\pi f_m t)]cos(2\pi f_c t)$$

Una secuencia discreta de esta señal se puede obtener al muestrear la señal anterior a intervalos de tiempo T, esto es, evaluando la función para nT. La versión discreta es la siguiente:

$$x(n) = A_c[1 + \mu sin(2\pi f_m nT)]cos(2\pi f_c nT)$$

Genere y grafique 1024 muestras de la secuencia $x(n)$, utilizando como parámetro $f_c = 1$ KHz, $f_m = 100$ Hz, $\mu = 0.7$, $A_c = 1$ y $T = 10^{-5}$.

10.3 Muestreo de señales

Una secuencia digital $x(n)$ se obtiene a partir de una señal analógica $x(t)$, mediante la siguiente expresión:

$$x(n) = x(kT)$$

donde $T = \frac{1}{f_s}$ es el intervalo de muestreo y f_s es la frecuencia de muestreo.

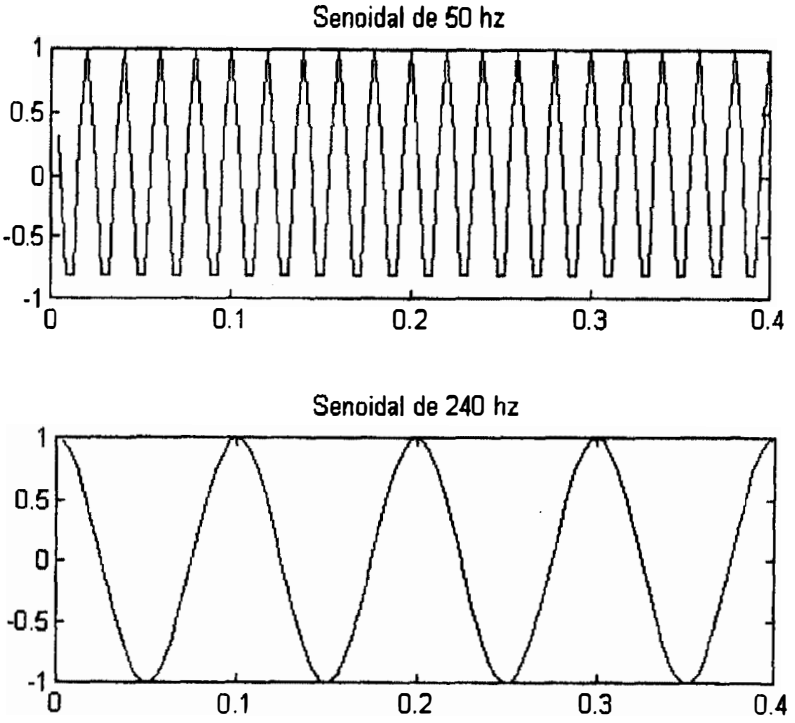


Figura 10.5. Problema de aliasing

El teorema del muestreo nos indica el valor que debe tener f_s para evitar el fenómeno denominado *aliasing*. Este teorema nos dice que la frecuencia de muestreo debe ser mayor al doble de la máxima frecuencia contenida dentro de la señal.

Ejemplo 3

Genere dos senoidales de 50 y 240 Hz, utilizando una frecuencia de muestreo de 250 Hz. Las siguientes instrucciones grafican ambas señales, observe y analice el resultado.

```
n=1:100;  %indice
T=1/250;  %periodo de muestreo=250 hz
y=cos(2*pi*50*n*T); %senoidal de 50 hz
plot(n*T,y);
pause
x=cos(2*pi*240*n*T); %senoidal de 240 hz
plot(n*T,x);
```

10.4 Convolución

La operación de convolución permite evaluar la respuesta de un sistema $h(n)$ a partir de una entrada $x(n)$ mediante la siguiente relación:

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$

MATLAB contiene la función `conv(a,b)` que realiza la operación de convolución de las secuencias discretas `a` y `b`, descritas mediante la ecuación anterior. A continuación se presenta el programa para calcular la convolución de las secuencias:

```
x(n)={1/4 1 1/2 1/2}
h(n)={1 1 1/2 1/2}
```

```
 a=[1/4 1/2 1 1/4]
   0.2500    0.5000    1.0000    0.2500
 b=[1 1 1/2 1/2]
b =
    1.0000    1.0000    0.5000    0.5000
 conv(a,b)
ans =
    0.2500    0.7500    1.6250    1.6250    1.0000    0.6250    0.1250
```

10.5 Función de transferencia de sistemas discretos

La función de transferencia de un sistema lineal e invariante en el tiempo nos representa la respuesta en frecuencia del sistema en amplitud y fase. La función `freqz(b,a)` de MATLAB permite obtener la respuesta en frecuencia de un sistema discreto. Los parámetros de entrada de esta función son los coeficientes de la función de transferencia del sistema, dados por los vectores `b` y `a`. El vector `b` contiene los coeficientes del numerador y el vector `a` del denominador de la función. Cuando no se tienen coeficientes en el denominador (llamado filtro FIR), el vector `a` debe ser: `a=[1]`. Algunos ejemplos de sistemas discretos se presentan a continuación.

Filtro no recursivo de primer orden

La función de este filtro es:

$$h(n) = \frac{y(n)}{x(n)} = 1 - r_0 z^{-1}$$

Para $r_0 = -.95$,

```
b=[1 0.95];
a=[1];
freqz(b,a);
```

Filtro no recursivo de segundo orden

La función de transferencia de este filtro es:

$$h(n) = \frac{y(n)}{x(n)} = 1 - rcos(W_0)z^{-1} + r^2z^{-2}$$

Para $r_0 = 0.9$,

```
r=0.9;
b=[1 -2*r*cos(pi/4) r*r];
a=[1];
freqz(b,a);
```

Filtro recursivo de primer orden

La función de transferencia de este filtro es:

$$h(n) = \frac{y(n)}{x(n)} = \frac{1}{1 - r_0z^{-1}}$$

Para $r_0 = 0.9$,

```
b=[1];
a=[1 -0.9];
freqz(b,a);
```

Filtro recursivo de segundo orden

La función de transferencia de este filtro es:

$$h(n) = \frac{y(n)}{x(n)} = \frac{1 - rcos(W_0)z^{-1}}{1 - 2rcos(W_0)z^{-1} + r^2z^{-2}}$$

Para $r_0 = 0.9$,

```
r=0.9;
b=[1-r*cos(pi/4)];
a=[1 -2*r*cos(pi/4) r*r];
freqz(b,a)
```

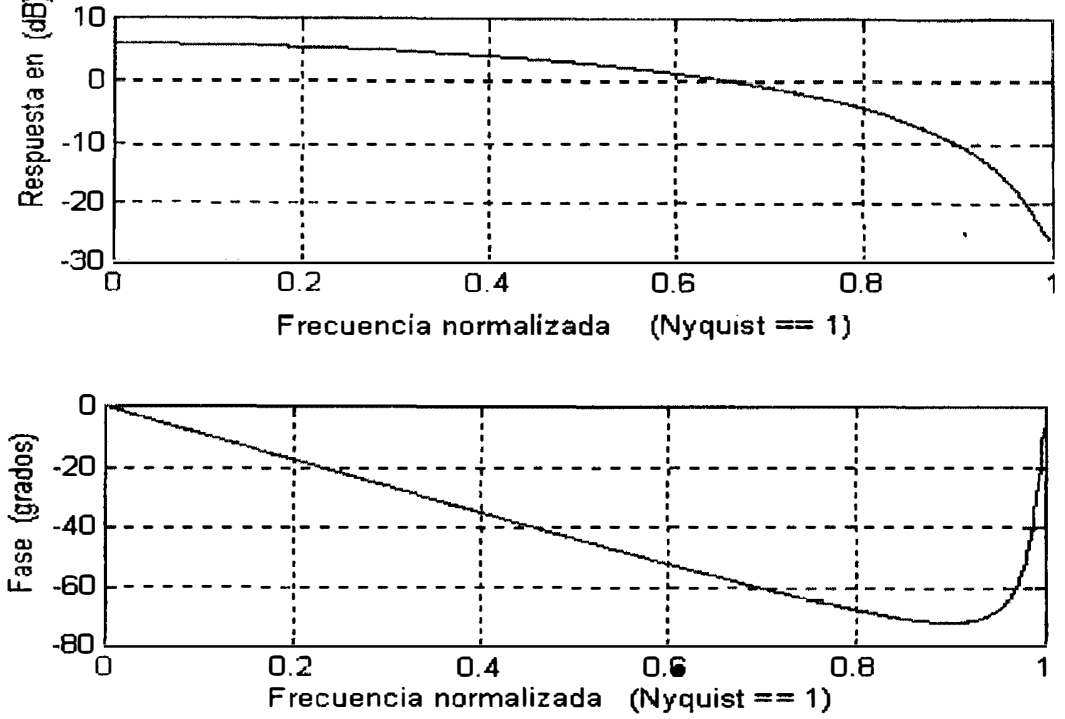


Figura 10.6. Filtro no recursivo de primer orden

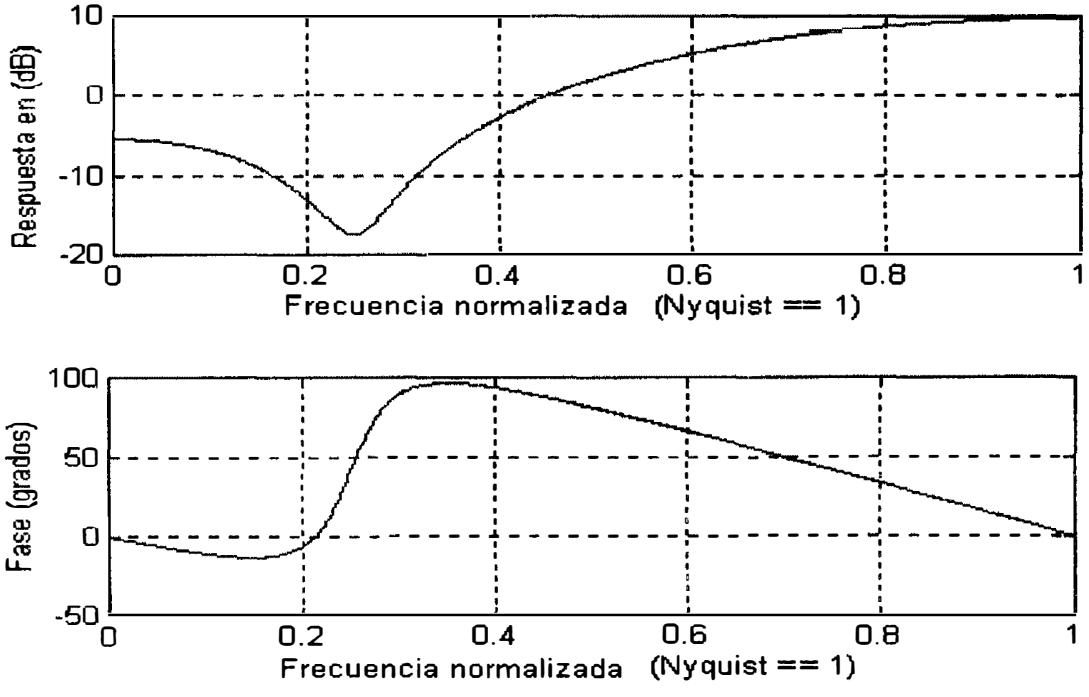


Figura 10.7. Filtro no recursivo de segundo orden

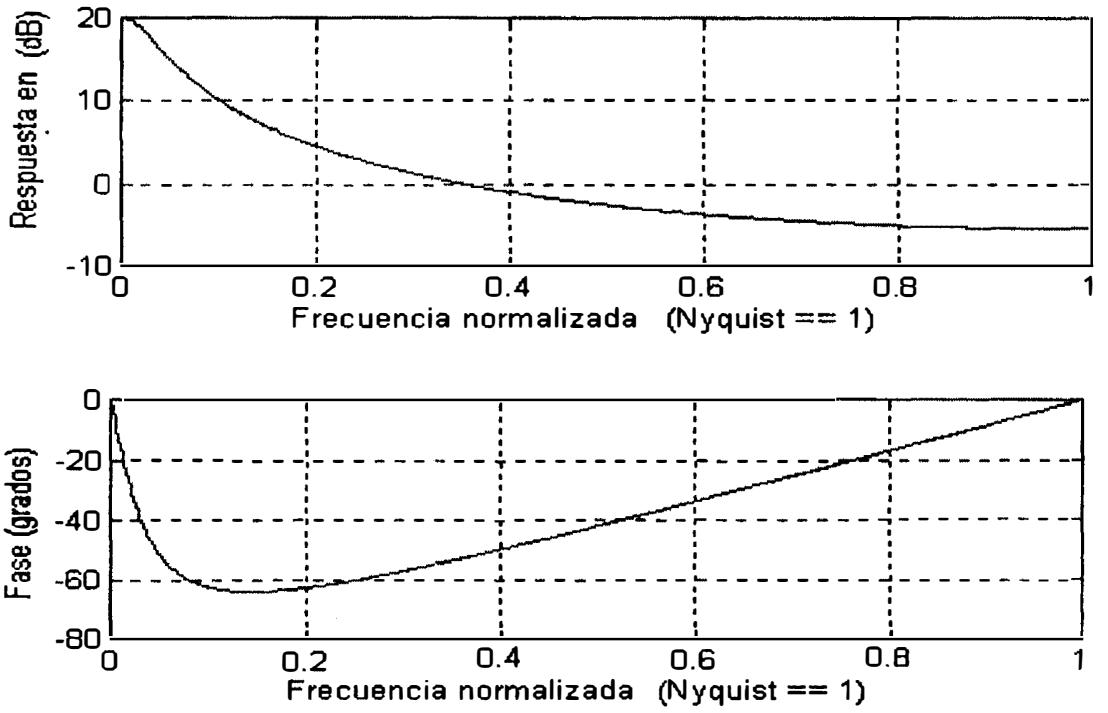


Figura 10.8. Filtro recursivo de primer orden

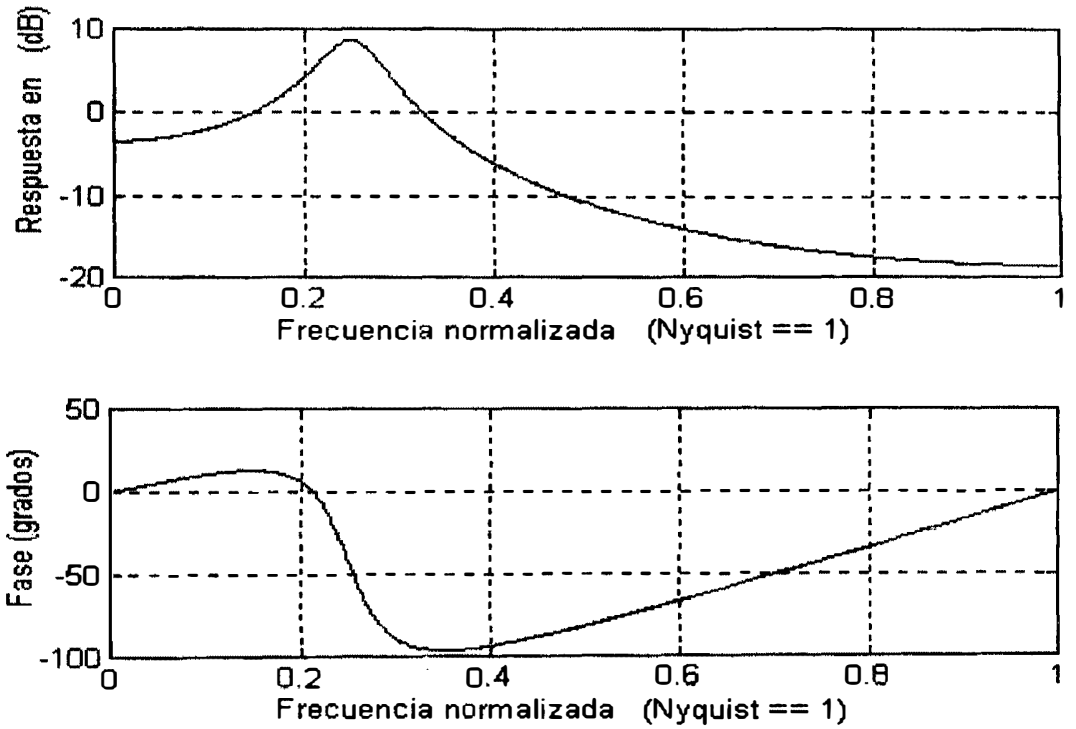


Figura 10.9. Filtro recursivo de segundo orden

Capítulo 11

Análisis de frecuencias

Las señales pueden ser analizadas en dos dominios: tiempo y frecuencia. En el dominio de la frecuencia, la señal se descompone en las componentes senoidales que integran la señal. Asimismo, se puede saber el contenido de frecuencias de dicha señal, o bien, determinar si la señal está limitada a un cierto ancho de banda. En el dominio del tiempo se puede extraer información tal como la media, la desviación estándar y la potencia.

Ejemplo 1

En el siguiente ejemplo se genera una señal f que se compone de una suma de senoidales de distintas frecuencias (5, 12.5, 20 y 35 Hz). Utilizando la transformada de Fourier, se analiza el contenido de frecuencias de la señal y la amplitud de cada componente.

```
N=256;
T=1/128;
k=0:N-1;
time=k*T;
f=0.25+2*sin(2*pi*5*k*T)+1*sin(2*pi*12.5*k*T)+...
1.5*sin(2*pi*20*k*T) + 0.5*sin(2*pi*35*k*T);
plot(time,f), title('Senal muestreada a 128 Hz');
xlabel('Tiempo (s)'), ylabel('Magnitud (volts)');
% Rutina FFT:
F=fft(f);
magF=abs([F(1)/N,F(2:N/2)/(N/2)]);
hertz=k(1:N/2)*(1/(N*T));
stem(hertz,magF), title('Componentes de frecuencia')
xlabel('Frecuencia (Hz)'), ylabel('Amplitud (volts)')
```

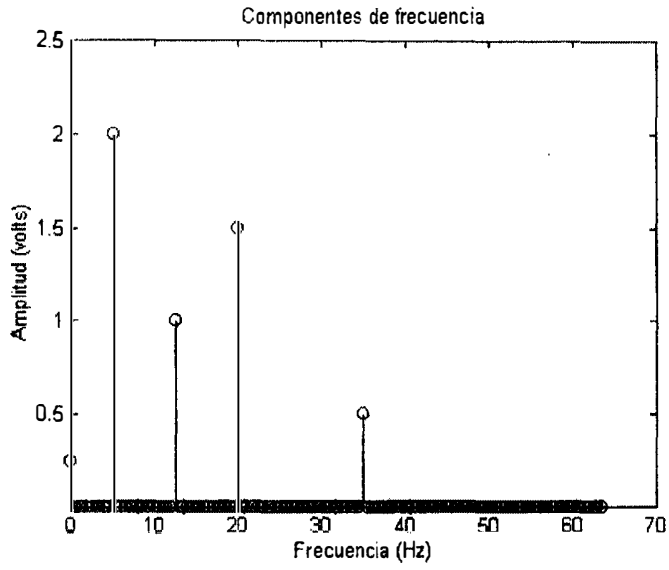


Figura 11.1. Componentes de frecuencia

11.1 Transformada de Fourier de secuencias discretas

La transformada de Fourier de una secuencia discreta se define como:

$$X(w) = \sum_{n=-\infty}^{\infty} x(n)e^{-jwn}$$

$X(w)$ es una función compleja y continua de la variable w, además, es una función con periodo 2π, es decir:

$$X(w) = X(w + 2\pi)$$

Las partes real e imaginaria de $X(w)$ son:

$$X(w) = X_r(w) + X_j(w)$$

La magnitud y fase de $X(w)$ se definen como:

$$\mid X(w) \mid = \sqrt{X_r(w)^2 + X_j(w)^2}$$

$$Arg[X(w)] = tan^{-1}\frac{X_j(w)}{X_r(w)}$$

Algunas propiedades importantes de la transformada de Fourier de secuencias discretas se presentan a continuación:

	Señal	Transformada
1	$x(n)$	$X(w)$
2	$x^*(n)$	$X^*(-w)$
3	Linealidad	
	$\alpha x_1(n) + \beta x_2(n)$	$\alpha X_1(w) + \beta X_2(w)$
4	Desplazamiento en tiempo	
	$x(n-M)$	$e^{-jwM}X(w)$
5	Inversión en tiempo	
	$x(-n)$	$X(-w)$
6	Desplazamiento en frecuencia	
	$e^{jw_0 n}x(n)$	$X(w-w_0)$
7	Modulación	
	$x(n)cos(w_0 n)$	$\frac{1}{2}X(w-w_0) + \frac{1}{2}X(w+w_0)$
8	Convolución	
	$x(n) \otimes h(n)$	$X(w)H(w)$
9	Diferenciación en frecuencia	
	$nx(n)$	$j\frac{dX(w)}{dw}$
10	Secuencia conjugada simétrica	
	$x(n) = x^*(-n)$	$X(w)$ real
11	Secuencia conjugada asimétrica	
	$x(n) = -x^*(-n)$	$X(w)$ imaginario
12	$x(n)$ real	$\mid X(w) \mid = \mid X(-w) \mid \; Arg[X(w)] = -arg[X(-w)]$

La función **freqz** puede ser utilizada para evaluar la transformada de Fourier de una secuencia discreta, expresada como una función racional de e^{jw} mediante la forma

$$X(e^{jw}) = \frac{P(e^{jw}}{D(e^{jw}} = \frac{p_0 + p_1 e^{jw} + ... + p_M e^{jwM}}{d_0 + d_1 e^{jw} + ... + d_M e^{jwM}}$$

y evaluada para ciertos valores de frecuencia $w = w_i$. Para obtener una buena aproximación se debe utilizar un gran número de datos.

Ejemplo 2

El siguiente programa evalúa y grafica las partes real e imaginaria, así como la magnitud y fase de la transformada de Fourier de una secuencia discreta, expresada como una función racional de e^{jw}. Los parámetros de entrada son el número de puntos **k** de frecuencia, para los cuales se evalúa la transformada, y los vectores de coeficientes del numerador y denominador.

```
clear
%Trans. Fourier de una secuencia discreta
%No. de datos de la TDF:
k = input('No. de puntos de frecuencia = ');
% coeficientes del numerador y denominador
num = input('Coeficientes del numerador = ');
den = input('Coeficientes del denominador = ');
% Calculo de la respuesta en frecuencia
w = 0:pi/k:pi;
h = freqz(num,den,w);
%Grafica la respuesta en frecuencia
subplot(2,2,1)
plot(w/pi,real(h));grid
title('Parte Real')
xlabel('Frecuencia angular normalizada')
ylabel('Amplitud')
subplot(2,2,2)
plot(w/pi,imag(h));grid
title('Parte imaginaria')
xlabel('Frecuencia angular normalizada')
ylabel('Amplitud')
subplot(2,2,3)
plot(w/pi,abs(h));grid
title('Magnitud')
xlabel('Frecuencia angular normalizada')
ylabel('Espectro de magnitud')
subplot(2,2,4)
plot(w/pi,angle(h));grid
title('Espectro de fase')
xlabel('Frecuencia angular normalizada')
ylabel('Fase en radianes')
pause
```

La figura 11.2 muestra las gráficas correspondientes para los siguientes valores:

```
k=256
num=[0.008 -0.033 0.05 -0.033 0.008]
den=[1 2.37 2.7 1.6 0.41]
```

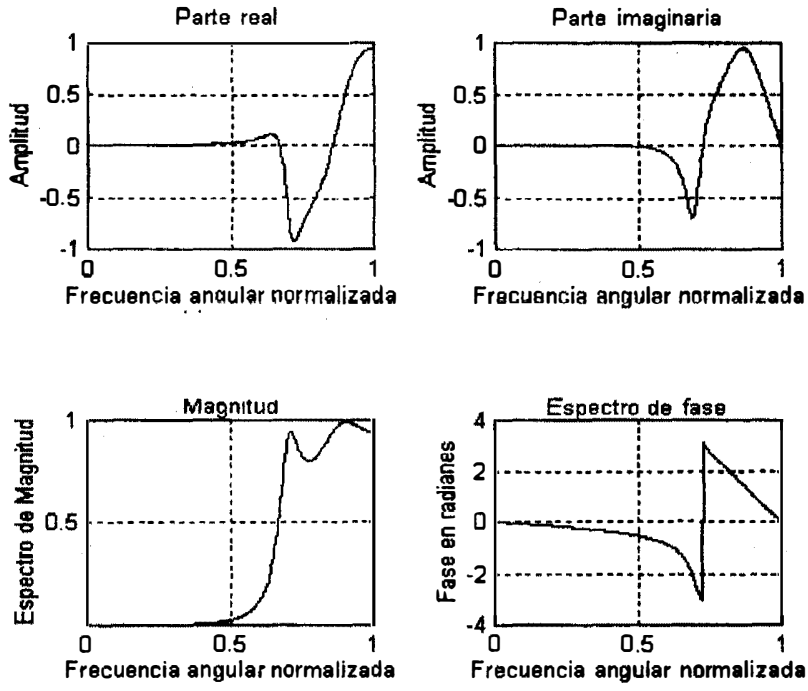


Figura 11.2. Gráficas de las partes real e imaginaria de la TDF

11.2 Transformada discreta de Fourier

La transformada de Fourier de una secuencia discreta es una función continua de w. Para evaluar la transformada de Fourier con una computadora digital, se requiere representar la transformada de Fourier como una secuencia discreta, lo que da origen a la *transformada discreta* de Fourier (TDF).

Sea $x(n)$ una secuencia discreta diferente de cero en el intervalo $0 \leq n \leq N-1$, la transformada discreta de Fourier se define como:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{\frac{-j2\pi nk}{N}}$$

para *k=0,...,N-1.*

Un algoritmo eficiente para el cálculo de la transformada discreta de Fourier es el de la transformada rápida de Fourier, implementado en MATLAB con la función **fft(x)**.

La transformada discreta de Fourier es el algoritmo que convierte la señal digital en el dominio del tiempo a una serie de puntos en el dominio de la frecuencia. Para una señal de N datos, el algoritmo calcula N valores complejos que representan la información en el dominio de la frecuencia. En el caso en que $N = 2^M$ un algoritmo especial llamado *transformada rápida de Fourier* (TRF) puede ser utilizado de manera que reduce enormemente el número de multiplicaciones, haciendo más eficiente el cálculo.

La señal digital es muestreada cada T segundos, es decir, a una razón de $\frac{1}{T}$ Hz. La selección de la frecuencia de muestreo es importante para evitar el llamado *aliasing* causado por una baja frecuencia de muestreo. La señal debe ser muestreada al doble del mayor contenido de frecuencia de la señal. La llamada frecuencia de *Nyquist* es igual a la mitad de la frecuencia de muestreo y representa el mayor contenido de frecuencias que puede contener la señal.

La función **fft(x)** en MATLAB realiza la transformada discreta de Fourier de la señal **x**. Si se utiliza un solo argumento de entrada, la salida es un vector del mismo tamaño. Si el tamaño del vector es una potencia de 2, se utiliza el algoritmo TRF, si es de otra forma, se usa simplemente la TDF. Al utilizar dos gumentos de entrada, el primero corresponde al vector de datos y el segundo es un número entero que representa el total de puntos del vector de salida.

Los valores en el dominio de la frecuencia tienen una resolución en frecuencia de $\frac{1}{NT}$ Hz; de manera que si se tienen 32 muestras de una señal analógica muestreada a 1000 Hz, los valores calculados en frecuencia corresponden a $F_i = i * 32.25$ Hz, donde $i = 0, 1, 2, .., 31$. La frecuencia de Nyquist es $\frac{1}{2T}$, o sea 500 Hz, y corresponde a F_{16}. Una de las propiedades de la TDF es la periodicidad; en consecuencia, los valores arriba de F_{16} son los complejos conjugados de los valores inferiores, no hay información adicional y, por lo general, solamente la primera mitad de la función **fft** es graficada.

11.3 Propiedades de la TDF

A continuación se analizan, mediante ejemplos de MATLAB, algunas de las propiedades de la transformada discreta de Fourier.

Ejemplo 3. Linealidad de la TDF

El siguiente ejemplo evalúa la propiedad de linealidad de la transformada discreta de Fourier de las señales x_1 y x_2, donde $x_1[n] = 0.8^n$ y $x_2 = u[n] - u[n-8]$. Primero se calcula la transformada de Fourier de la suma $x_1[n]+x_2[n]$ y enseguida se calcula la suma $X_1[w]+X_2[w]$. Se comparan únicamente los primeros quince puntos.

```
%PROPIEDAD LINEAL DE LA TDF
%exponencial decreciente:
n=0:99;
a=0.8;
x1=a.^n;
%escalon desplazado:
M=8;
x2=ones(1,100)-[zeros(1,M) ones(1,100-M)];
y=x1+x2;
Y=fft(y);
Ys=fft(x1)+fft(x2);
% Aqui se presentan algunas muestras de Y(w) y Ys(w)
Yt=[Y(1:15)' Ys(1:15)']
```

Ejemplo 4. Desplazamiento en tiempo de la TDF

El siguiente ejemplo muestra la magnitud y fase de las transformadas de Fourier de $x_2[n]$ y $x_3[n]$. Sea $x_2[n] = u[n] - u[n-3]$ y $x_3[n] = x_2[n-2]$. Se observa que no hay variación en la magnitud, pero si hay un cambio en la fase debido a que la transformada se multiplica por una exponencial compleja.

```
% PROPIEDAD DE DESPLAZAMIENTO EN TIEMPO
clear
clf
M=3;
% x2(n)=u(n)-u(n-3)
x2=ones(1,100)-[zeros(1,M) ones(1,100-M)];
N=2;
NN=5;
% x3(n)=x2(n-2)=u(n-2)-u(n-5)
x3=[zeros(1,N) ones(1,100-N)]-[zeros(1,NN) ones(1,100-NN)];
subplot(2,1,1)
plot(x2)
title('x2(n)=u(n)-u(n-3)')
subplot(2,1,2)
plot(x3)
title('x3(n)=n(n-2)-u(n-5)')
pause
X2=fft(x2);
X3=fft(x3);
subplot(2,1,1);
plot(abs(X2))
title('DESPL. EN TIEMPO   Magnitud de x2(w)')
subplot(2,1,2);
plot(abs(X3))
title('Magnitud de x3(w)')
pause
subplot(2,1,1);
plot(angle(X2))
title('Fase de x2(w)')
subplot(2,1,2);
plot(unwrap(angle(X3)))
title('Fase de x3(w)')
%Aqui se presentan algunas muestras de X2(w) y X3(w)
XW=[X2(1:15)' X3(1:15)']
```

Ejemplo 5. Inversión en tiempo de la TDF

Sea $x[n] = u[n] - u[n-3]$ y $w[n] = x[-n]$. La transformada de Fourier de $w[n]$ es similar a la transformada de $x[n]$ en magnitud y en fase, pero esta última es de signo contrario.

```
%INVERSION EN TIEMPO
clear
clf
% x(n)=u(n)-u(n-3)
M=3;
x=ones(1,100)-[zeros(1,M) ones(1,100-M)];
% w(n)=x(-n)
w=fliplr(x);
X=fft(x);
W=fft(w);
subplot(2,1,1);
plot(abs(X))
title('INV. EN TIEMPO    Magnitud de x(w)')
subplot(2,1,2);
plot(abs(W))
title('Magnitud de w(w)')
pause
subplot(2,1,1);
plot(angle(X))
title('Fase de x(w)')
subplot(2,1,2);
plot(unwrap(angle(W)))
title('Fase de w(w)')
```

Ejemplo 6. Modulación

El espectro de una señal multiplicada por una senoidal con frecuencia w_0 se observa desplazado en frecuencia precisamente por w_0 rad/s. La señal $x[n] = u[n] - u[n-8]$ se multiplica por $\cos(0.2\pi n)$. Observe el cambio en la magnitud de la transformada.

```
%MODULACION
clear
clf
%x(n)=u(n)-u(n-8)
clear
clf
M=8;
x=ones(1,100)-[zeros(1,M) ones(1,100-M)];
%w(n)=cos(.2*pi*n)
k=0:99;
w=cos(0.2*pi*k);
y=w.*x;
Y=fft(y);
X=fft(x);
```

```
subplot(2,1,1);
plot(abs(X))
title('MODULACION     Magnitud de x(w)')
subplot(2,1,2);
plot(abs(Y))
title('Magnitud de y(w)')
pause
subplot(2,1,1);
plot(angle(X))
title('Fase de x(w)')
subplot(2,1,2);
plot((angle(Y)))
title('Fase de y(w)')
pause
```

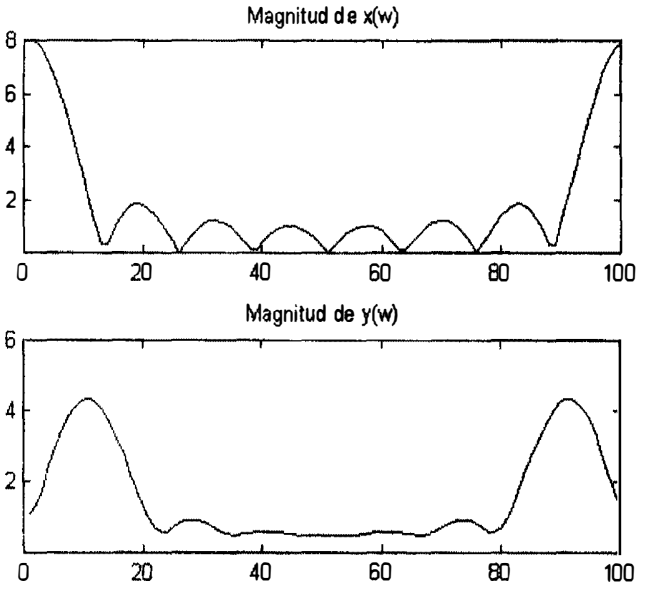


Figura 11.3. Propiedad de modulación de la TDF

Ejemplo 7. Desplazamiento circular

Sea $X[w]$ la TDF de la secuencia $x[n]$ de N puntos. Un desplazamiento circular de la señal en el tiempo de M muestras es equivalente a multiplicar la transformada por una exponencial compleja $e^{-j2\pi kM/N}$. A continuación se observa la señal $x[n] = 24u[n] - nu[n]$ desplazada circularmente cuatro muestras al multiplicar por $exp((-2 * 4 * i * pi * k)/N)$.

```
%DESPLAZAMIENTO CIRCULAR
clear
clf
k=0:24;
%x(n)=24u(n)-n*u(n)
```

```
x=24*ones(1,25)-k.*[ones(1,25)];
X=fft(x);
X1=X.*exp((-2*4*i*pi*k)/25);
x1=ifft(X1);
XX=[x' x1']

XX =

  24.0000             3.0000 + 0.0000i
  23.0000             2.0000 + 0.0000i
  22.0000             1.0000 + 0.0000i
  21.0000             0.0000 + 0.0000i
  20.0000            24.0000 + 0.0000i
  19.0000            23.0000 + 0.0000i
  18.0000            22.0000 + 0.0000i
  17.0000            21.0000 + 0.0000i
  16.0000            20.0000 - 0.0000i
  15.0000            19.0000 - 0.0000i
  14.0000            18.0000 + 0.0000i
  13.0000            17.0000 + 0.0000i
  12.0000            16.0000 - 0.0000i
  11.0000            15.0000 - 0.0000i
  10.0000            14.0000 - 0.0000i
   9.0000            13.0000 - 0.0000i
   8.0000            12.0000 - 0.0000i
   7.0000            11.0000 - 0.0000i
   6.0000            10.0000 - 0.0000i
   5.0000             9.0000 - 0.0000i
   4.0000             8.0000 - 0.0000i
   3.0000             7.0000 - 0.0000i
   2.0000             6.0000 - 0.0000i
   1.0000             5.0000 - 0.0000i
        0             4.0000 + 0.0000i
```

Ejemplo 8. Convolución

Considere las secuencias $x_1[n]$ y $x_2[n]$ y sus respectivas transformadas $X_1[w]$ y $X_2[w]$. La convolución de las señales puede expresarse como la multiplicación de las transformadas: $x_1[n] \otimes x_2[n] = X_2[w]X_2[w]$.

```
%CONVOLUCION LINEAL
clear
clf
x1=[3 4.2 1 1 0 7 -1 0 2 0 0 0 0 0 0];
x2=[1.2 3 0 -0.5 2 0 0 0 0 0 0 0 0 0 0];
```

```
X1=fft(x1);
X2=fft(x2);
convol=conv(x1,x2)';
[ifft(X1.*X2)' convol(1:15)]
```

```
   3.6000 + 0.0000i     3.6000
  14.0400 + 0.0000i    14.0400
  13.8000 - 0.0000i    13.8000
   2.7000 + 0.0000i     2.7000
   6.9000 + 0.0000i     6.9000
  16.3000 + 0.0000i    16.3000
  21.3000 - 0.0000i    21.3000
  -1.0000 - 0.0000i    -1.0000
  -1.1000 + 0.0000i    -1.1000
  20.5000 - 0.0000i    20.5000
  -2.0000 - 0.0000i    -2.0000
  -1.0000 + 0.0000i    -1.0000
   4.0000 - 0.0000i     4.0000
   0.0000 + 0.0000i          0
   0.0000 + 0.0000i          0
```

11.4 Transformada discreta de Fourier matricial

La TDF está definida por

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{\frac{-j2\pi nk}{N}}$$

Si hacemos $W = e^{\frac{-j2\pi}{N}}$, entonces podemos expresar:

$$X(k) = \sum_{n=0}^{N-1} x(n) W^{kn}$$

que en forma matricial es:

$$\begin{pmatrix} X(0) \\ X(1) \\ . \\ . \\ X(N-1) \end{pmatrix} = \begin{pmatrix} W^0 & W^0 & W^0 & . & . & W^0 \\ W^0 & W^1 & W^2 & & & W^{N-1} \\ W^0 & W^2 & W^4 & & & W^{2(N-1)} \\ . & & & & & . \\ . & & & & & . \\ W^0 & W^N - 1 & W^{-2(N-1)} & . & . & W^{(N-1)(N-1)} \end{pmatrix} \begin{pmatrix} x(0) \\ x(1) \\ . \\ . \\ x(N-1) \end{pmatrix}$$

Ejemplo 9

La siguiente rutina genera la matriz de orden N para la TDF en la variable **mat** dentro de MATLAB y se utiliza para determinar la transformada de la secuencia $x[n] = [12345678]$. Se compara este resultado con el obtenido con la función **fft** de MATLAB.

```
N=8;
W=exp(-i*2*pi/N);
for k=1:N
  for l=1:N
    mat(k,l)=W^((k-1)*(l-1));
  end
end

x=[1 2 3 4 5 6 7 8]';
R=[mat*x fft(x)]
R =

  36.0000              36.0000
  -4.0000 + 9.6569i    -4.0000 + 9.6569i
  -4.0000 + 4.0000i    -4.0000 + 4.0000i
  -4.0000 + 1.6569i    -4.0000 + 1.6569i
  -4.0000 - 0.0000i    -4.0000
  -4.0000 - 1.6569i    -4.0000 - 1.6569i
  -4.0000 - 4.0000i    -4.0000 - 4.0000i
  -4.0000 - 9.6569i    -4.0000 - 9.6569i
```

Ejemplo 10

El siguiente programa calcula el espectrograma de la señal de voz. En MATLAB hay un archivo de voz **mtlb.mat**. La señal de este archivo (figura 11.4) contiene 4001 muestras con la frecuencia de muestreo 7418 Hz. Vamos a utilizar la ventana de Hamming que tiene la longitud 256 con traslapamiento de 0 muestras. En la figura 11.4 a) se muestra el espectrograma obtenido mediante el siguiente programa:

```
%Espectrograma de la voz
%Numero de las muestras 4001
%La frecuencia del muestreo 7418 Hz
%La ventana de Hamming
%Numero de traslapes 50
load mtlb
n=1:4001;
plot(n-1,mtlb);
xlabel('Indice del tiempo n');ylabel('Amplitud');
```

```
pause
nfft=input('Teclee la longitud de la ventana= ');
ovlap=input('Teclee numero de traslapes= ');
specgram(mtlb,nfft,7418,hamming(nfft),ovlap)
```

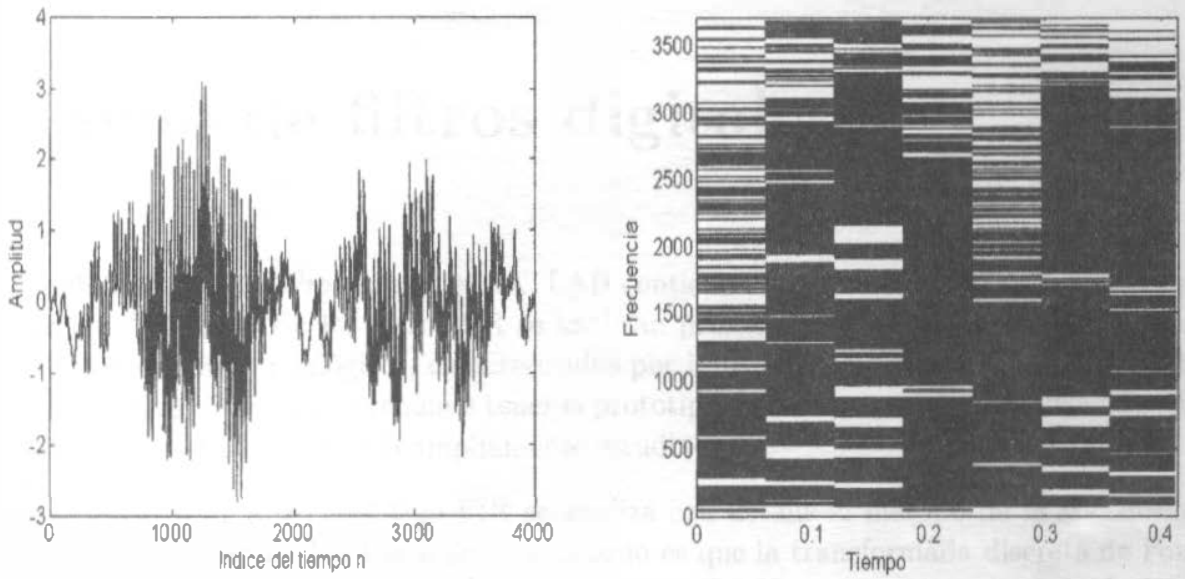


Figura 11.4. Espectrograma de la señal de voz

Capítulo 12

Diseño de filtros digitales

El toolbox de *Signal Processing* de MATLAB contiene una gran variedad de funciones para el diseño de filtros. En los filtros IIR se analizan primeramente algunos métodos de transformación de filtros analógicos, caracterizados por la función de transferencia $H(s)$ a filtros digitales $H(z)$. Para ello se requiere tener el prototipo de un filtro analógico adecuado, cuyos métodos de diseño hayan sido ampliamente estudiados.

Para el diseño de filtros del tipo FIR se analiza con detalle el método de la transformada inversa de Fourier. La limitante de este método es que la transformada discreta de Fourier involucra el uso de una ventana de datos de tamaño N que genera una serie de lóbulos laterales en la banda de rechazo en el espectro del filtro. Para disminuir dichos lóbulos se utilizan diferentes tipos de ventanas que serán analizadas.

Enseguida, se aplican las funciones de MATLAB para el diseño a partir de las especificaciones del filtro.

12.1 Diseño de filtros IIR mediante transformación de filtros analógicos

12.1.1 Invariancia de la respuesta al impulso

Un método para obtener el filtro digital a partir de un filtro analógico es conservando la correspondiente respuesta al impulso analógica. La respuesta al impulso del filtro digital es una versión muestreada del prototipo analógico, es decir:

$$h(n) = Tg(nT) \tag{12.1}$$

donde $h(n)$ y $g(t) = g(nT)$ son las respuestas al impulso de los filtros digital y analógico, respectivamente, y donde T es el intervalo de muestreo.

Como se estudió al analizar el muestreo de señales, el efecto de *aliasing* puede presentarse cuando el filtro analógico no está limitado en su ancho de banda. Eso restringe la aplicación de esta técnica a filtros pasa-bajas y algunos paso-banda.

Ejemplo 1

Sea

$$H(s) = \frac{1}{s + 1000}$$

grafique la respuesta del filtro analógico, obtenga el correspondiente filtro digital mediante el método de invariancia de la respuesta al impulso, utilizando la función **impinvar** de MATLAB. Finalmente, grafique la función de transferencia del filtro analógico con el programa FREQS. El resultado del programa se muestra en las figuras 12.1 y 12.2.

```
% INVARIANCIA DE LA RESPUESTA AL IMPULSO
%           1
%  H(s) = -----
%          s+a
A=[1 1000];
B=1;
freqs(B,A);
title('Filtro analogico')
pause
clf
T=pi/1000;
f=1/T;
[b,a]=impinvar(B,A,f)
freqz(b,a);
title('Correspondiente filtro digital')
```

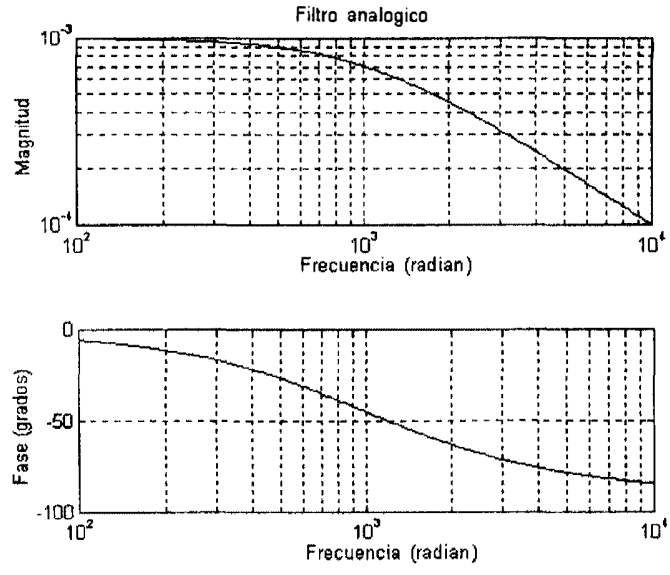


Figura 12.1. Magnitud y fase del filtro analógico

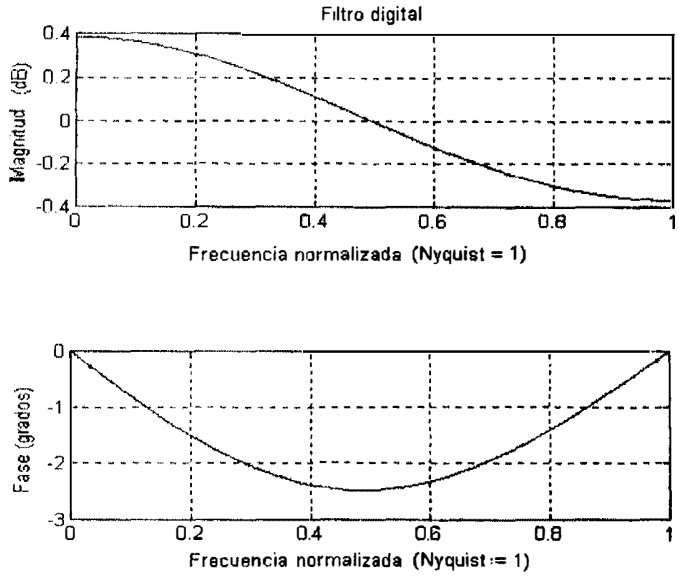


Figura 12.2. Magnitud y fase del filtro digital

Ejemplo 2

Utilizando la función butter, diseñe un filtro analógico tipo Butterworth de orden 4 y con frecuencia de corte de 1.021612 rad/s. Mediante el método de invariancia de la respuesta al impulso obtenga el correspondiente filtro digital. Grafique la respuesta de ambos filtros.

```
clear
% Filtro pasa-bajas Butterworth
% Invariancia de la respuesta al impulso
% ORDEN:
N=4;
% FREC. DE CORTE:
Wn=1.021612;
[num,den]=butter(N,Wn,'s');
freqs(num,den)
title('Filtro tipo Butterworth analogico')
pause
clf
% FREC. DE MUESTREO:
FT=1;
[b,a]=impinvar(num,den,FT);
[h,omega]=freqz(b,a,512);
mag=20*log10(abs(h));
plot(omega/pi,mag);grid;
title('Filtro tipo Butterworth digital')
xlabel('Frecuencia normalizada');
ylabel('Ganancia dB');
pause
```

El resultado del programa se muestra en las figuras 12.3 y 12.4.

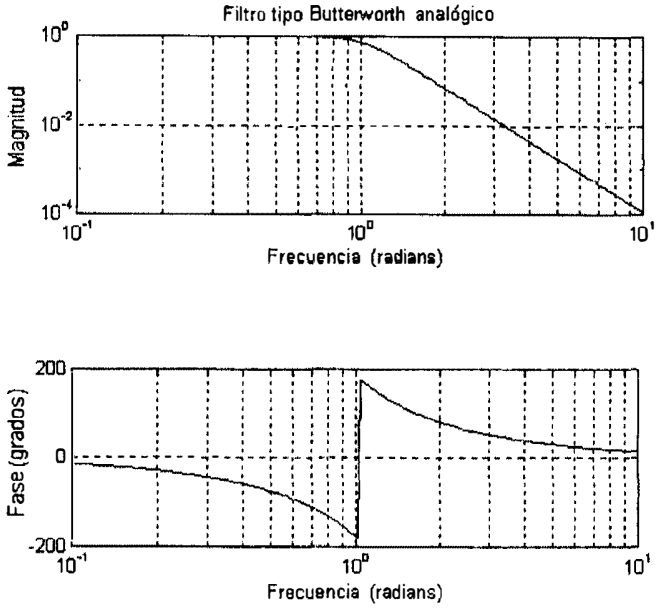


Figura 12.3. Magnitud y fase del filtro Butterworth analógico

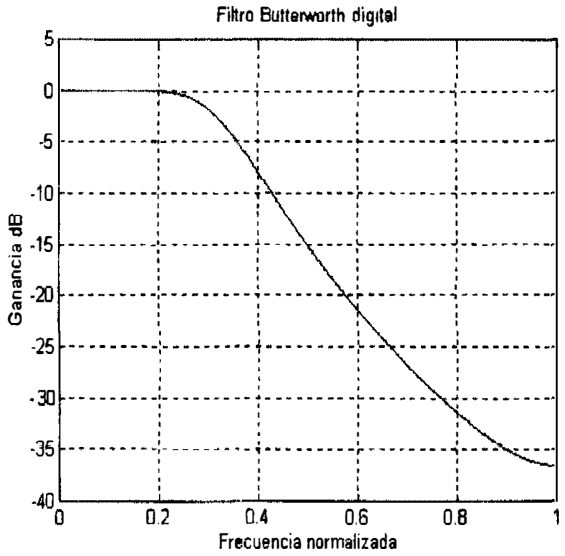


Figura 12.4. Magnitud del filtro Butterworth digital

12.1.2 Transformación bilineal

Para evitar el problema de aliasing se requiere de una transformación que realice un mapeo uno a uno, es decir, un punto en el plano s que corresponda a un único punto en el plano z y viceversa. Para ello se puede utilizar la transformación bilineal, representada mediante la siguiente ecuación:

$$s = \frac{2}{T}\frac{1 - z^{-1}}{1 + z^{-1}} \tag{12.2}$$

Sustituyendo la expresión anterior en $H(s)$, se obtiene $H(z)$.

Ejemplo 3

Sea

$$H(s) = \frac{\frac{1}{3}s}{s^2 + \frac{1}{3}s + 5}$$

Obtenga en papel la función de transferencia del filtro digital $H(z)$ mediante transformación bilineal y MATLAB con las siguientes instrucciones:

```
% TRANSFORMACION BILINEAL
%                1/3 s
%  H(s) = ---------------
%           2
%          s  + 1/3 s + 5
clear
A=[1 1/3 5];
B=[1/3 0];
f=1;
[a,b]=bilinear(B,A,f)
```

12.2 Funciones de MATLAB para el diseño de filtros IIR

MATLAB contiene funciones para el diseño de filtros de los cuatro tipos basados en diseños analógicos. Los filtros Butterworth tienen una respuesta plana en las bandas de paso y de rechazo. El filtro Chebyshev tipo I presenta ciertas variaciones en la banda de paso, llamadas rizo; el tipo II tiene rizo en la banda de rechazo y el elíptico en ambas bandas. Sin embargo, para un determinado orden, los filtros elípticos tienen la transición más pronunciada de todos los filtros. Las funciones para diseñar filtros IIR pasa-bajas, utilizando los prototipos analógicos, son las siguientes:

`[B,A]=butter(N,Wn)`	N	: orden del filtro
`[B,A]=cheby1(N,Rp,Wn)`	Rp	: rizo en banda de paso
`[B,A]=cheby2(N,Rs,Wn)`	Rs	: rizo en banda de rechazo
`[B,A]=ellip(N,Rp,Rs,Wn)`	Wn	: frecuencia de corte

Ejemplo 4

Diseñe un filtro pasa-bajas elíptico de orden 5 con una frecuencia de corte de 0.4, un rizo en la banda de paso de 0.5 dB y una atenuación mínima en la banda de rechazo de 40 dB. Grafique la respuesta del filtro. El resultado se muestra en la figura 12.5.

```
%DISE&O DE FILTROS
% Filtro pasa-bajas tipo eliptico
clear
N=input('Teclee el orden del filtro = ');
Rp=input('Teclee el rizo en la banda de paso = ');
Rs=input('Teclee la atenuacion minima en la banda de rechazo = ');
Wn=input('Teclee le frecuencia de corte de la banda de paso = ');
[b,a]=ellip(N,Rp,Rs,Wn);
disp(' Polinomio del numerador')
disp(b)
disp(' Polinomio del denominador')
disp(a)
w=0:0.01/pi:pi;
h=freqz(b,a,w);
gain=20*log10(abs(h));
plot(w/pi,gain);grid;
xlabel('Frecuencia normalizada');
ylabel('Ganancia, dB');
```

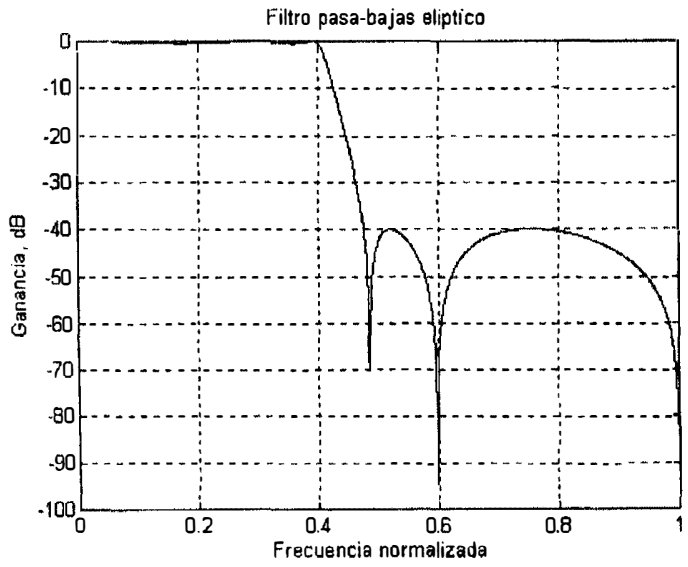


Figura 12.5. Filtro pasa-bajas elíptico

Ejemplo 5

Obtenga la función de transferencia y grafique la respuesta de un filtro pasa-altas de orden 4 tipo Chebyshev I con una frecuencia de corte 0.7 y rizo de 1 dB. El resultado del programa se muestra en la figura 12.6.

```
% Filtro pasa-altas tipo Chebyshev I
clear
N=input('Teclee el orden del filtro = ');
Rp=input('Teclee el rizo en la banda de paso = ');
Wn=input('Teclee le frecuencia de corte de la banda de paso = ');
[b,a]=cheby1(N,Rp,Wn,'high');
disp(' Polinomio del numerador')
disp(b)
disp(' Polinomio del denominador')
disp(a)
w=0:0.01/pi:pi;
h=freqz(b,a,w);
gain=20*log10(abs(h));
plot(w/pi,gain);grid;
xlabel('Frecuencia normalizada');
ylabel('Ganancia, dB');
```

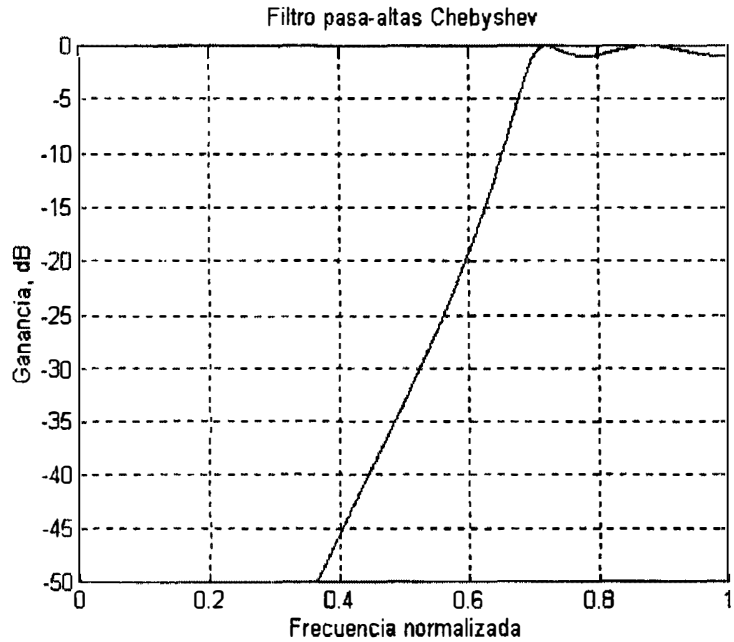


Figura 12.6. Filtro pasa-altas Chebyshev I

Ejemplo 6

Obtenga la función de transferencia y grafique la respuesta de un filtro Chebyshev II tipo pasa-altas de orden 6, con un rizo en la banda de rechazo de 20 dB y frecuencia de corte de 0.6. El resultado del programa se muestra en la figura 12.7.

```
% Filtro pasa-altas tipo Chebyshev II
clear
N=input('Teclee el orden del filtro = ');
Rp=input('Teclee el rizo en la banda de paso = ');
Wn=input('Teclee le frecuencia de corte de la banda de paso = ');
[B,A]=cheby2(N,Rp,Wn,'high')
[H,wT]=freqz(B,A,100);
T=.001;hertz=wT/(2*pi*T);
plot(hertz,abs(H)),title('Pasa-altas Chebyshev')
xlabel('Hz'),ylabel('Magnitud'),grid
```

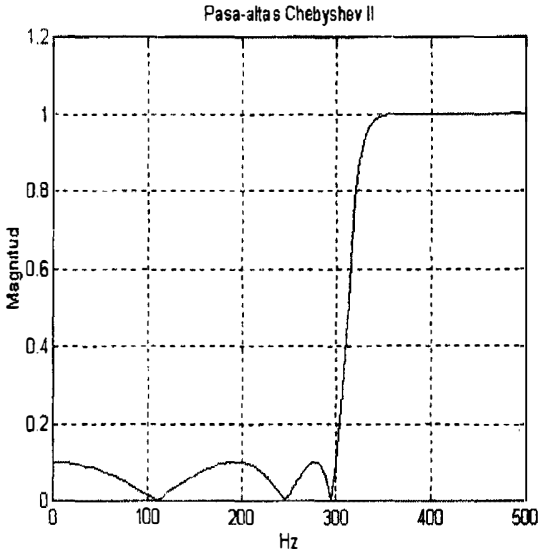


Figura 12.7. Filtro pasa-altas Chebyshev II

Ejemplo 7

Obtenga la función de transferencia y grafique la respuesta de un filtro paso-banda Butterworth de orden 8 con bandas de paso y rechazo de 0.4 y 0.7, respectivamente.

```
% Filtro paso-banda tipo Butterworth
clear
N=input('Teclee el orden del filtro = ');
M=N/2;
W1=input('Teclee la frecuencia de corte inferior de la banda de paso = ');
W2=input('Teclee la frecuencia de corte superior de la banda de paso = ');
```

```
Wn=[W1 W2];
[b,a]=butter(M,Wn);
disp(' Polinomio del numerador')
disp(b)
disp(' Polinomio del denominador')
disp(a)
w=0:0.01/pi:pi;
h=freqz(b,a,w);
gain=20*log10(abs(h));
plot(w/pi,gain);grid;
xlabel('Frecuencia normalizada');
ylabel('Ganancia, dB');
```

El resultado del programa se muestra en la figura 12.8.

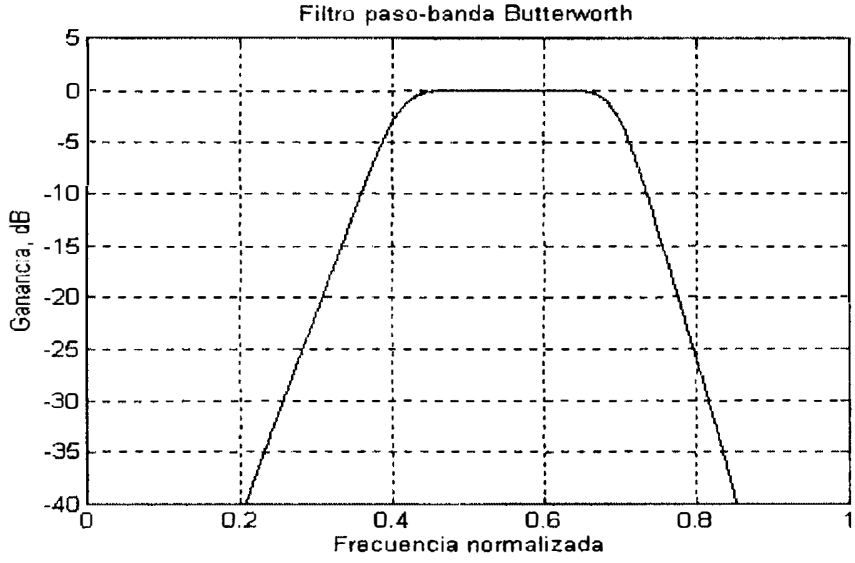


Figura 12.8. Filtro paso-banda Butterworth

El toolbox de MATLAB tiene implementadas las funciones para el diseño directo de filtros Yule-Walker. El comando es

```
[B,A]=yulewalk(n,f,m)
```

Los vectores **A** y **B** contienen los coeficientes del filtro de orden **n**. Los vectores **f** y **m** indican las características en frecuencia de la magnitud, utilizando un rango de frecuencias normalizado, donde 1 representa la frecuencia de Nyquist. Las frecuencias definidas en *f* empiezan en 0 y finalizan en 1, en orden ascendente. Las magnitudes, dadas en *m*, corresponden a la magnitud deseada del filtro para cada f.

```
% YULEWALK
m=[0 0 1 1 0 0 1 1 0 0];
f=[0 .1 .2 .3 .4 .5 .6 .7 .8 1];
[B,A]=yulewalk(12,f,m);
[H,wT]=freqz(B,A,100);
plot(f,m,'--',wT/pi,abs(H)),title('Filtro IIR con dos   bandas')
xlabel('Frecuencia normalizada'),ylabel('Magnitud'),grid
```

El resultado de este programa se muestra en la figura 12.9.

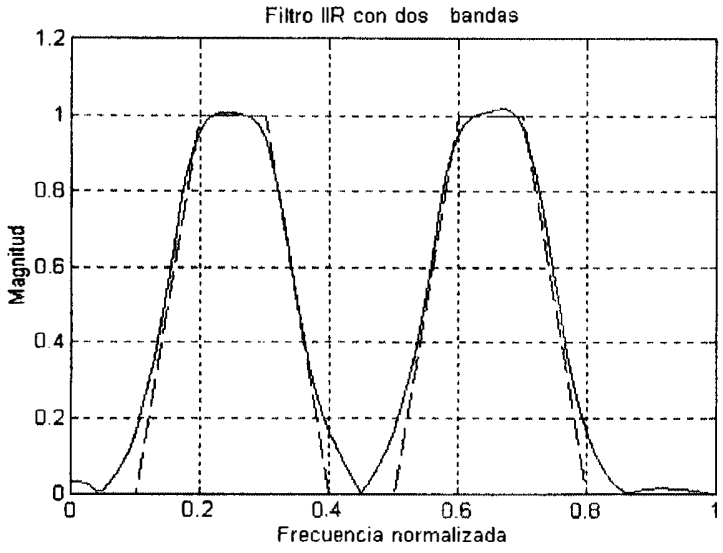


Figura 12.9. Filtro IIR con dos bandas

12.3 Diseño de filtros tipo FIR

Los filtros del tipo respuesta al impulso finita (FIR) se caracterizan por una función de transferencia del tipo:

$$h(n) = b_1 x(n) + b_2 x(n-1) + b_3 x(n-2) + .. + b_N x(n-N+1) \tag{12.3}$$

Una de las ventajas de los filtros tipo FIR es la posibilidad de tener una fase lineal, pero muchas veces el orden del filtro se incrementa considerablemente. Para una implementación en tiempo real, los filtros IIR son matemáticamente más eficientes y demandan menos recursos.

Diseño mediante series de Fourier

Un filtro pasa-bajas ideal puede caracterizarse por una función de transferencia de la forma:

$$H_d(w) = \begin{cases} 1 & \mid w \mid < w_c \\ 0 & w_c < \mid w \mid < \pi \end{cases} \tag{12.4}$$

Asumiendo que la ecuación anterior es la respuesta del filtro ideal, aplicamos entonces la transformada inversa de Fourier para encontrar la respuesta al impulso de este filtro ideal, es decir:

$$h_d(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_d(w)e^{jwn}dw \tag{12.5}$$

$$h_d(n) = \frac{1}{2\pi}\int_{-w_c}^{w_c} e^{jwn}dw \tag{12.6}$$

y finalmente:

$$h_d = \frac{sin(w_c n)}{\pi n} \tag{12.7}$$

Sin embargo, esta última ecuación es válida para n en el rango de $-\infty < n < \infty$; y un filtro FIR tiene una respuesta al impulso finita, por lo tanto, solamente se puede obtener una aproximación truncando la secuencia infinita. Esto afecta la respuesta de manera que se presenta una serie de lóbulos laterales no deseados, lo que se conoce como *oscilación Gibbs.*

El efecto de truncar la serie infinita es equivalente a convolucionar el espectro ideal con una ventana rectangular. Mediante el uso de distintas ventanas es posible reducir el efecto de los lóbulos laterales. La función **fir1** de MATLAB calcula la respuesta al impulso del filtro FIR, aplicándose como parámetros de entrada el orden N, la frecuencia de corte Wn y la ventana que se utilizará.

Ejemplo 8

Función de transferencia de filtros FIR de orden N=61 y frecuencia de corte $W_c = 0.3$, utilizando diferentes ventanas. Las siguientes instrucciones de MATLAB permiten diseñar cada uno de los filtros y obtener la respuesta al impulso $h(n)$ correspondiente. En la figura 12.10 se presenta únicamente la respuesta para ventanas rectangular y Hamming.

```
% ANALISIS DE VENTANAS
% FILTROS FIR
N=61;

%VENTANA RECTANGULAR
w=boxcar(N);
b=fir1(60,0.3,w);
[h,omega]=freqz(b,1,512);
magb=20*log10(abs(h));

%VENTANA HANNING
w=hanning(N);
b=fir1(60,0.3,w);
[h,omega]=freqz(b,1,512);
```

```
magn=20*log10(abs(h));

%VENTANA HAMMING
w=hamming(N);
b=fir1(60,0.3,w);
[h,omega]=freqz(b,1,512);
magm=20*log10(abs(h));

%VENTANA KAISER
w=kaiser(N,4.533514);
b=fir1(60,0.3,w);
[h,omega]=freqz(b,1,512);
magk=20*log10(abs(h));


plot(omega/pi,magb,omega/pi,magn,omega/pi,magm,omega/pi,magk);grid;
xlabel('Frecuencia normalizada');
ylabel('Ganancia, dB');
```

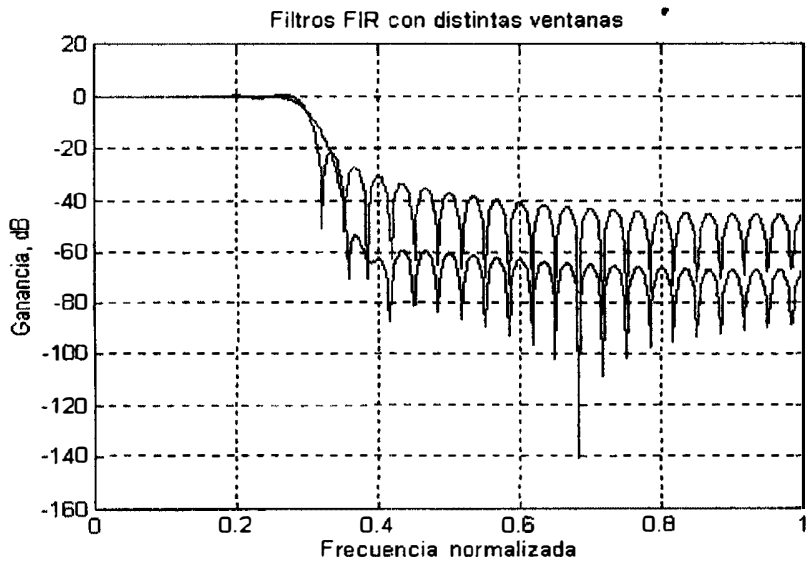


Figura 12.10. Filtros FIR con ventanas rectangular y Hamming

Capítulo 13

Procesamiento de imágenes

En este capítulo se describen los conceptos básicos para analizar y procesar imágenes. Asimismo, se estudian dos objetivos fundamentales al procesar una imagen: mejorar su calidad y comprimirla.

Una imagen es una función de dos dimensiones $I = f(x, y)$, donde x,y denotan coordenadas espaciales y el valor f es una variable proporcional al brillo o nivel de gris, el cual es una variable discreta. Una imagen digital ha sido discretizada en coordenadas e intensidad, y se representa por una matriz, y cada elemento de la matriz es llamado *pixel*.

13.1 Ecualización del histograma

El histograma es una función $p(r_k)$, donde r_k es el nivel de gris y se ubica en el intervalo $0 \leq r_k \leq 1$, y $p(r_k)$ es la probabilidad de cada nivel de gris, es decir,

$$\frac{n_k}{n} = No.$$

No es número de pixeles con nivel de gris k/total de pixeles.

El histograma proporciona una descripción general de la apariencia de la imagen. Es muy útil conocer el histograma, ya que al modificarlo en una forma específica se mejora la calidad de la imagen.

Para efectuar la ecualización de la imagen, realizamos una transformación de r de la siguiente forma:

$$s = T(r) \tag{13.1}$$

donde la función de transformación $T(r)$ es:

a) monotónicamente creciente y *uno a uno*
b) $0 \leq T(r) \leq 1$

además, $T^{-1}(s)$ existe y satisface a) y b).

Considere el nivel de gris como una variable aleatoria con función de densidad de probabilidad $p_r(r)$, entonces, la función de densidad de probabilidad de la variable de gris transformada es:

$$P_s(s) = [P_r(r)\frac{dr}{ds}] \tag{13.2}$$

Ahora, seleccionemos la transformación definida por la siguiente función:

$$s = T(r) = \int_0^r p_r(r)dr \tag{13.3}$$

es decir, la transformación es la función de distribución acumulada de r. Entonces,

$$\frac{ds}{dr} = \frac{ds}{dr}[\int_0^r P_r(r)dr] \tag{13.4}$$

y

$$P_s(s) = P_r(r)\frac{1}{P_r(r)} = 1 \tag{13.5}$$

de donde se observa que al utilizar como transformación la distribución acumulada de probabilidad, obtenemos una imagen con niveles de gris uniformes, esto significa que se incrementa el rango dinámico. Para el caso discreto:

$$P_r(r_k) = \frac{n_k}{n} \tag{13.6}$$

y la transformación está dada por:

$$S_k = T(r_k) = \Sigma_{j=0}^{k} P_r(r_j) \tag{13.7}$$

para $0 \leq r_k \leq 1$.

La ecuación anterior es llamada *ecualización del histograma.*

Como se explica en el capítulo 9, MATLAB cuenta con un toolbox para el procesamiento de imágenes. La función **histeq** permite realizar la ecualización de una imagen dada como una matriz **I**. Las siguientes instrucciones afectúan la ecualización y comparan el resultado de una imagen llamada **trees**. Mediante la instrucción **imshow** es posible graficar una imagen.

```
load trees
I=ind2gray(X,map);
J=histeq(I,32);
K=histeq(I,4);
subplot(2,2,1),imshist(J,32)
subplot(2,2,2),imshow(J,32)
subplot(2,2,3),imshist(K,32)
subplot(2,2,4),imshow(K,32)
```

Figura 13.1. Imagen analizada

La figura 13.1 muestra la imagen original y la 13.2, el correspondiente histograma. En la figura 13.3 se presenta la imagen ecualizada y en la 13.4, el histograma respectivo en el que se observa una mejor distribución de los tonos de gris.

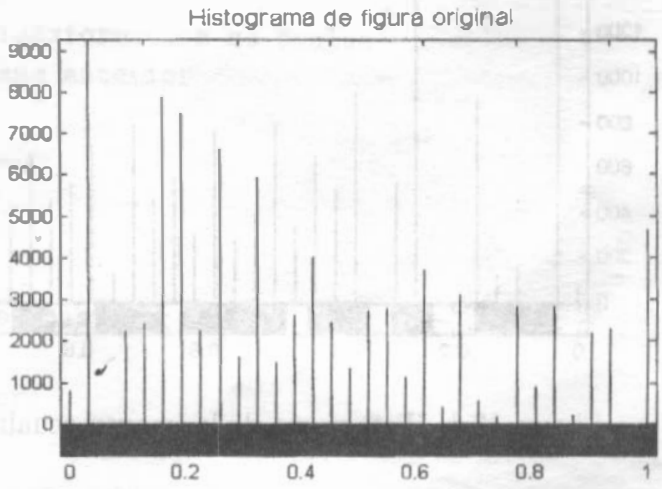


Figura 13.2. Histograma de la imagen original

Figura 13.3. Imagen ecualizada

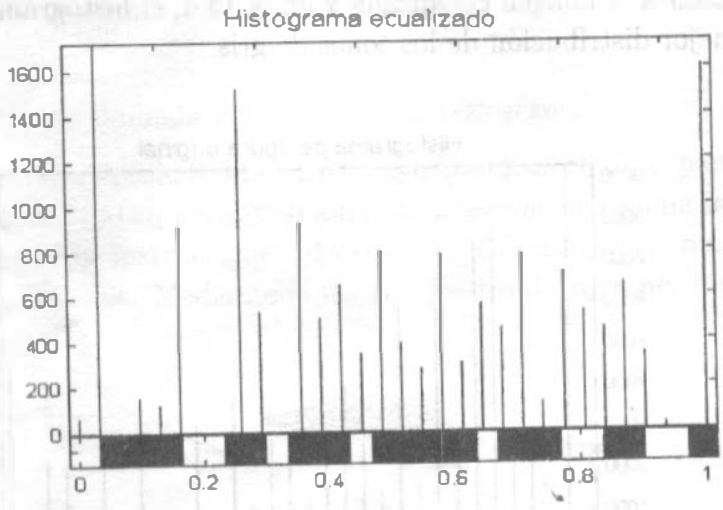


Figura 13.4. Histograma de la imagen ecualizada

13.2 Modificación del histograma

Para obtener un histograma en particular, esto es, realzar un determinado nivel de gris, podemos utilizar la técnica denominada *especificación del histograma.* Esto se realiza de la siguiente forma:

1. Ecualizar la imagen original.
2. Especificar la función de distribución de probabilidad (o histograma) deseado y obtener la función de transformación $G(z)$.
3. Aplicar la función de transformación inversa $G^{-1}(z)$ a los niveles obtenidos en 1.

La siguiente función realiza la transformación del histograma de la imagen **I** y el histograma final de esta imagen está dado por **hist**.

```
function II=espec(I,hist)
% Esta funcion modifica la imagen I
% en la imagen II cuyo histograma esta dado por hist
% El numero de pixeles esta dado por
% el numero de pixeles de histograma -> tamano(hist)
% Este programa modifica el histograma
% La imagen original esta ecualizada

Ip=histeq(I);

% El numero de pixeles  k y el nivel de gris
% estan dados por el tamano del histograma

k=size(hist,1)
graylev=0:1/k:1;

[M,N]=size(Ip);

% La funcion para transformacion se evalua
% usando el histograma anterior

trans(1)=0;
for i=1:k,
    tempor=0;
    for j=1:i,
        tempor=tempor+hist(j);
    end,
    trans(i+1)=tempor;
end,
Ip=Ip/max(max(Ip));

% Ahora la imagen se transforma usando
```

```
% el mapeo inverso de  trans y
% la nueva imagen es II


for i=1:M,
   for j=1:N,
     K=k+1;
     while Ip(i,j)<trans(K)
        K=K-1;
     end,
     II(i,j)=graylev(K);
   end,
end,
```

La figura 13.5 muestra histogramas de una imagen en la que se han aplicado tres diferentes tipos de transformación a las funciones **x*x** y **sqrt(x)**.

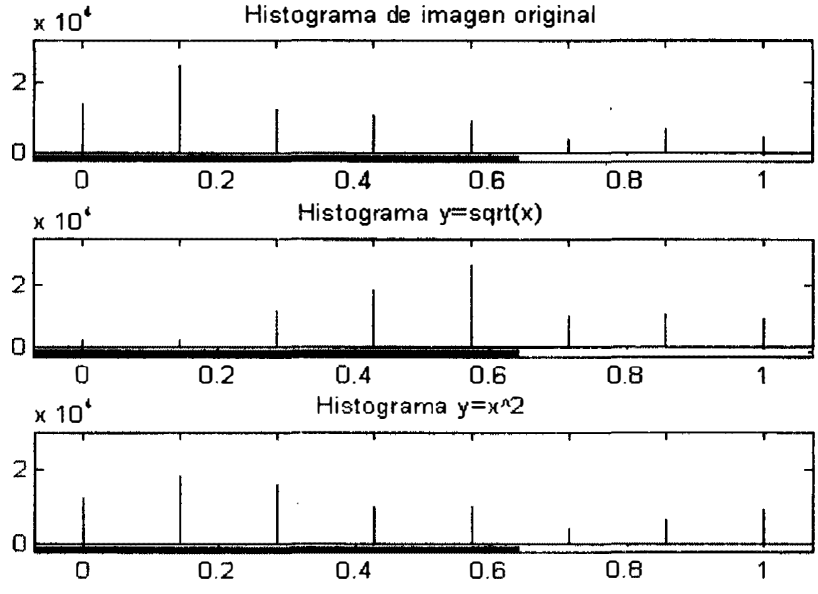


Figura 13.5. Histogramas de diferentes transformaciones

13.3 Detección de orillas

Para transformar una imagen, se asigna un nuevo valor a cada pixel que es una función del nivel de gris de z_5 y del nivel de gris de los pixeles vecinos. Por ejemplo, en el vecindario de pixeles definido por:

Z_1	Z_2	Z_3
Z_4	Z_5	Z_6
Z_7	Z_8	Z_9

El nuevo valor del pixel Z_5 puede obtenerse al realizarse un promedio de los pixeles a su alrededor. Esta operación se denomina *máscara*. A cada pixel se le asigna un peso. La ecuación del nuevo valor de Z_5 es:

$$Z = W_1Z_1 + W_2Z_2 + ... + W_9Z_9 \tag{13.8}$$

Para detectar las orillas de una imagen se calcula el gradiente basado en obtener las derivadas $\partial f/\partial x$, $\partial f/\partial y$. Una forma de implementar esta función es mediante el operador *Sobel*. Las máscaras correspondientes son:

$$G_x = (Z_7 + 2Z_8 + Z_9) - (Z_1 + 2Z_2 + Z_3) \tag{13.9}$$

$$G_y = (Z_3 + 2Z_6 + Z_9) - (Z_1 + 2Z_4 + Z7) \tag{13.10}$$

Utilizando diferentes operadores, las siguientes instrucciones de MATLAB encuentran las orillas de la imagen **I**.

```
load treess
I=ind2gray(X,map);
subplot(2,2,1),imshow(~edge(I),2)
subplot(2,2,2),imshow(~edge(I),'roberts'2)
subplot(2,2,3),imshow(~edge(I),'prewitt'2)
subplot(2,2,4),imshow(~edge(I),'marr'2)
```

En la figura 13.6 se observan las imágenes que contienen las orillas para cada operador. Los operadores *Sobel* y *Prewitt* detectan las orillas donde la primera derivada de la intensidad de la imagen es máxima (o mínima). El operador *Roberts* es sensible a orillas en diagonales.

Para obtener orillas a +45 y -45 grados es posible rotar las máscaras:

+45 grados

2	1	0
1	0	-1
0	-1	-2

-45 grados

0	1	2
-1	0	1
-2	-1	0

La siguiente función, llamada **g45.m**, obtiene estos gradientes:

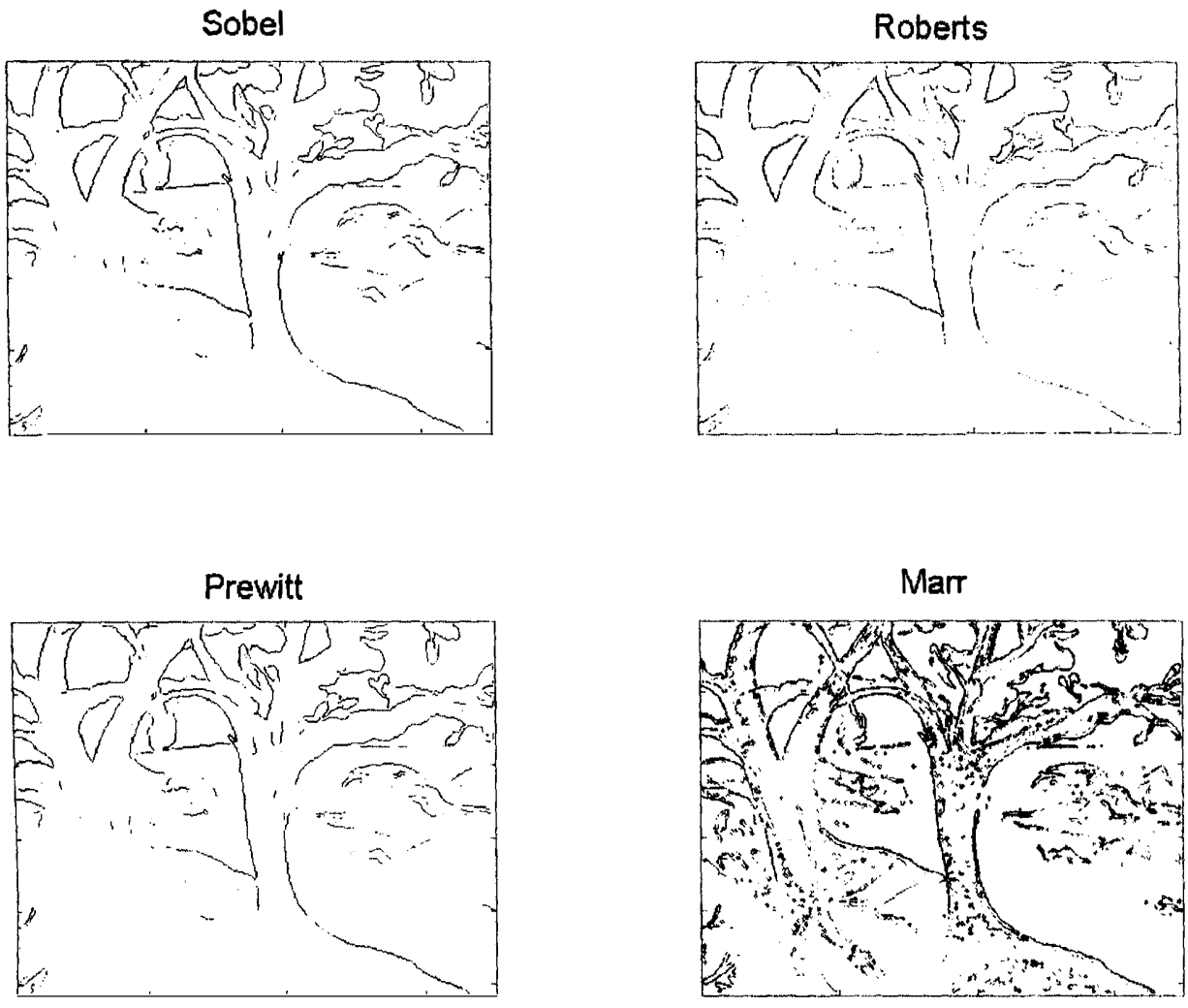


Figura 13.6. Operadores para detección de orillas

```
function [EDG45]=g45(I)
% [Gr]=g45(I)
% Esta funcion evalua el gradiente de la imagen

[x,y]=size(I);
M45=[ 0  1  2
     -1  0  1
     -2 -1  0];
m45=[ 2  1  0
      1  0 -1
      0 -1 -2];


GM=zeros(x,y);
Gm=zeros(x,y);
for j=2:x-1,
   for k=2:y-1,
      I2=I(j-1:j+1,k-1:k+1);
      GM(j-1,k-1)=abs(sum(sum(I2.*M45)));
      Gm(j-1,k-1)=abs(sum(sum(I2.*m45)));
   end,
end,

EDG45=sqrt((GM.^2+Gm.^2));
```

La figura 13.7 presenta una imagen con las orillas a 45 grados.

13.4 Transformada de Fourier

La transformada de Fourier permite conocer el contenido de frecuencia de una imagen. Para una sola variable, la transformada de Fourier está definida por:

$$X(k) = \frac{1}{N} \sum_{n=0}^{N-1} x(n) e^{\frac{-j2\pi kn}{N}} \tag{13.11}$$

$$x(n) = \sum_{k=0}^{N-1} X(k) e^{\frac{j2\pi kn}{N}} \tag{13.12}$$

Para el caso de dos variables:

$$X(k,l) = \frac{1}{MN} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x(n,m) e^{-j2\pi(\frac{kn}{M}+\frac{lm}{N})} \tag{13.13}$$

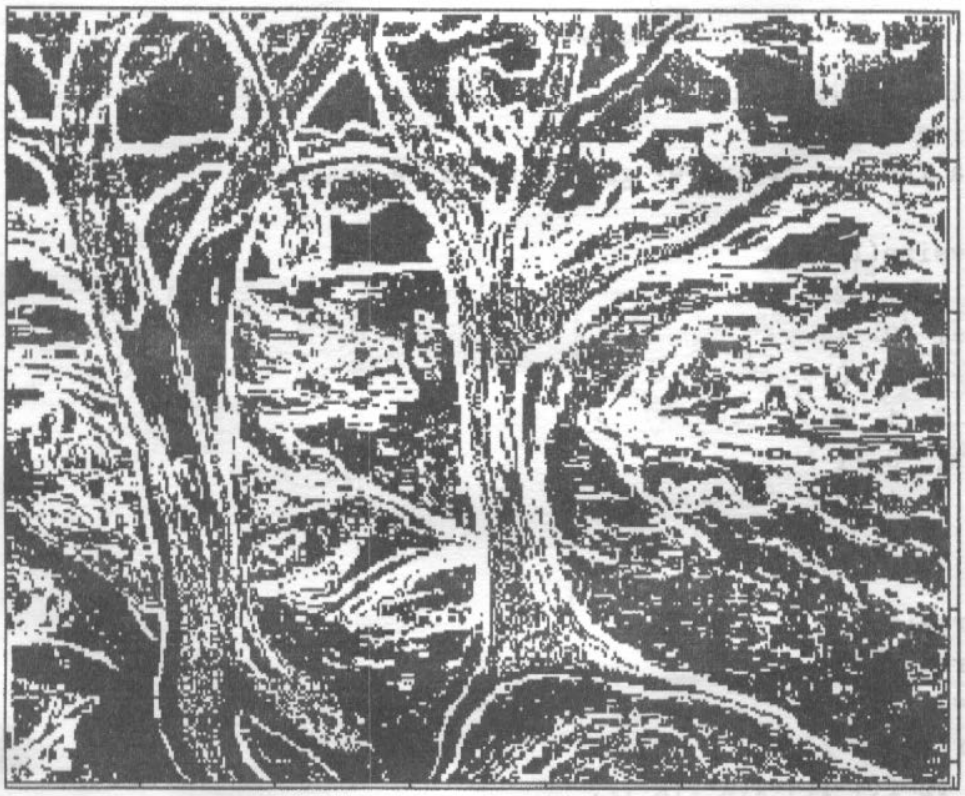


Figura 13.7. Orillas a 45 grados

$$x(n,m) = \sum_{k=0}^{N-1} \sum_{l=0}^{M-1} X(k,l) e^{j2\pi(\frac{kn}{M} + \frac{lm}{N})} \tag{13.14}$$

Para un arreglo donde $M = N$, se tiene:

$$X(k,l) = \frac{1}{N} \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} x(n,m) e^{\frac{-j2\pi(kn+lm)}{N}} \tag{13.15}$$

$$x(n,m) = \frac{1}{N} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} X(k,l) e^{\frac{j2\pi(kn+lm)}{N}} \tag{13.16}$$

Algunas propiedades importantes de la transformada de Fourier de dos variables son la separabilidad, la traslación y la periodicidad.

13.4.1 Separabilidad

La transformada de Fourier definida por

$$X(k,l) = \frac{1}{N} \sum_{n=0}^{N-1} e^{\frac{-j2\pi kn}{N}} \sum_{m=0}^{M-1} x(n,m) e^{\frac{-j2\pi lm}{N}} \tag{13.17}$$

$$x(n,m) = \frac{1}{N}\sum_{k=0}^{N-1} e^{\frac{j2\pi kn}{N}} \sum_{l=0}^{M-1} X(k,l)e^{\frac{j2\pi lm}{N}} \tag{13.18}$$

y por la propiedad de separabilidad puede expresarse como

$$X(k,l) = \frac{1}{N}\sum_{n=0}^{N-1} X(n,l)e^{\frac{-j2\pi ln}{N}} \tag{13.19}$$

donde

$$X(n,l) = \frac{1}{N}\sum_{m=0} N - 1e^{\frac{-2j\pi km}{N}} \tag{13.20}$$

13.4.2 Traslación

Esta propiedad se expresa como

$$x(n,m)e^{\frac{j2\pi(k_0n+l_0m)}{N}} \Longleftrightarrow X(k-k_0, l-l_0) \tag{13.21}$$

Esto significa que el origen de la transformada de Fourier puede moverse al centro de su correspondiente frecuencia MXN, multiplicando por $-1^{n,m}$.

13.4.3 Periodicidad

La transformada de Fourier discreta es periódica de periodo N. Para el caso de dos variables:

$$X(k,l) = X(k+N,l) = X(k,l+N) = X(k+N,l+N) \tag{13.22}$$

Por esta razón, para observar un periodo completo, debemos mover el origen de la transformada a un punto N/2, utilizando la función **fftshift** de MATLAB. Las siguientes instrucciones permiten graficar la transformada de Fourier de la imagen **I**. La magnitud de la transformada de Fourier de esta imagen se observa en la figura 13.8 y en la 13.9 después de mover las frecuencias al origen.

```
load trees
I=ind2gray(X,map);
IF=fft2(I);
IF2=log(ans(IF));
subplot(1,2,1),imshow(IF2),
subplot(1,2,2),imshow(fftshift(IF2))
```

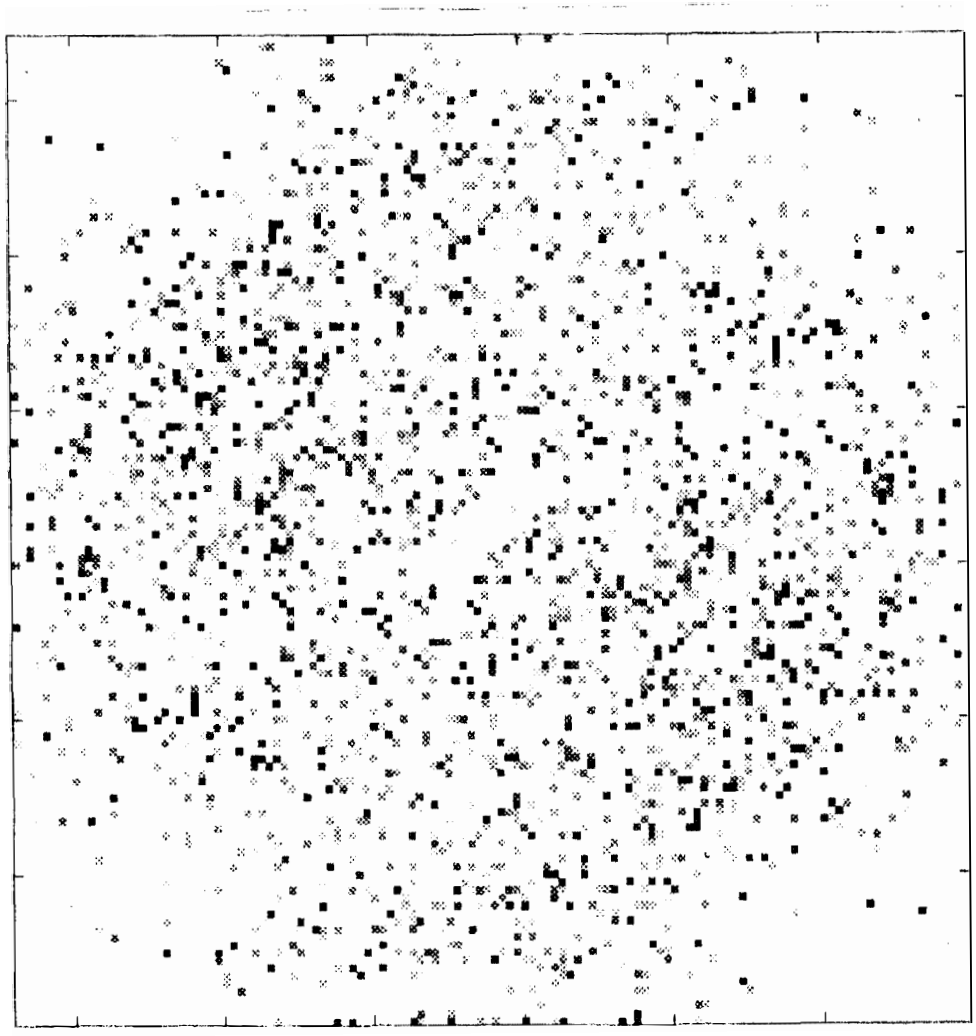


Figura 13.8. Transformada de Fourier de la imagen

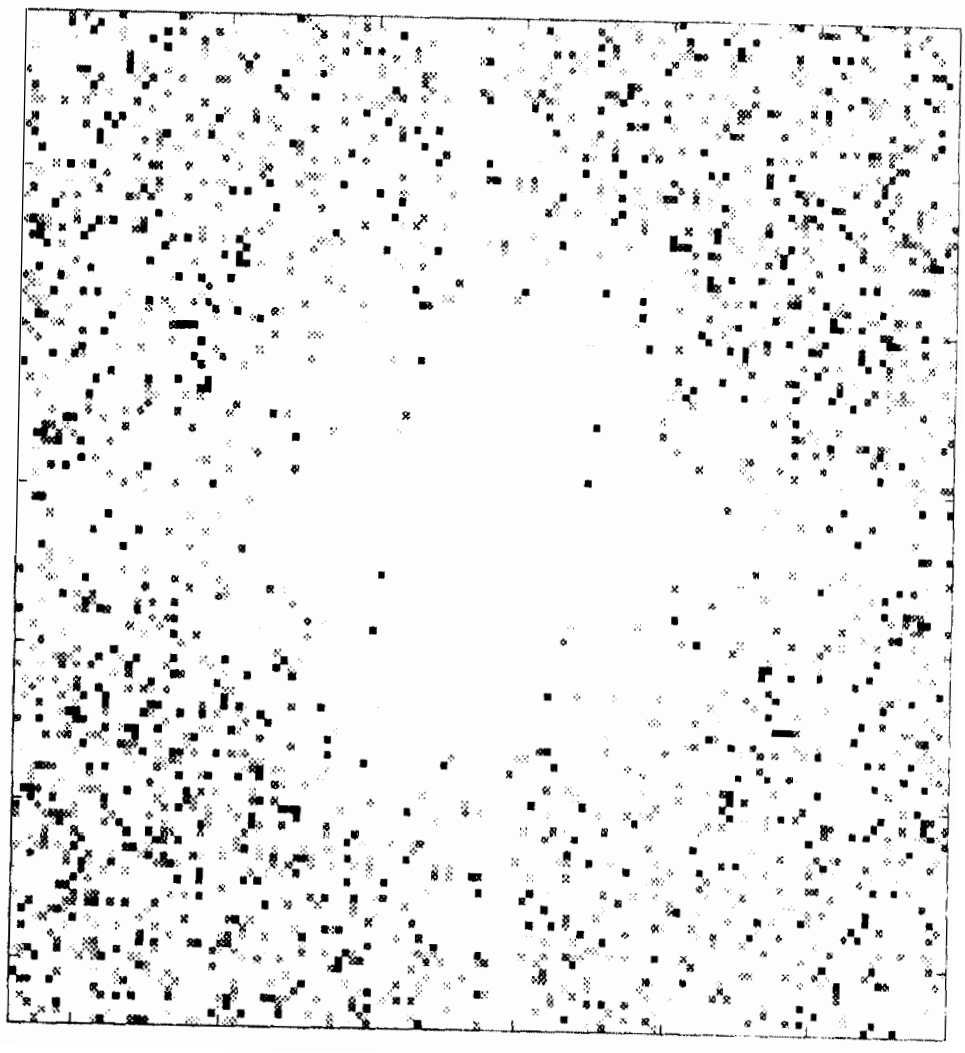


Figura 13.9. Transformada de Fourier centrada al origen

13.5 Transformada coseno

La transformada coseno de una imagen es una matriz que contiene toda la información concentrada en ciertos valores de manera que, despreciando los datos menores a un cierto umbral, podremos reducir la cantidad de información, es decir, comprimir la imagen.

La transformada coseno está definida por:

$$c(u) = \alpha(u) \sum_{x=0}^{N-1} f(x) cos(\frac{(2x+1)u\pi}{2N}) \tag{13.23}$$

para $u = 0, 1, 2, ..., N-1$.

Y la transformada inversa:

$$f(x) = \sum_{u=0}^{N-1} \alpha(u) c(u) cos(\frac{(2x+1)u\pi}{2N}) \tag{13.24}$$

para $x = 0, 1, 2, ..., N-1$,

donde

$$\alpha(u) = \sqrt{\frac{1}{N}} \qquad \text{para} \qquad u = 0$$

y

$$\alpha(u) = \sqrt{\frac{2}{N}} \qquad \text{para} \qquad u = 1, 2, ..., N-1$$

El caso de dos dimensiones (2-D) de la transformada coseno se expresa como:

$$c(k,l) = \alpha(k)\alpha(l) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) cos(\frac{(2x+1)k\pi}{2N}) cos(\frac{(2y+1lu\pi}{2N}) \tag{13.25}$$

para

$$k, l = 0, 1, 2, ..., N-1$$

y la transformada inversa es

$$f(x,y) = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} c(k,l)\alpha(k)\alpha(l) cos(\frac{(2x+1)k\pi}{2N}) cos(\frac{(2y+1lu\pi}{2N}) \tag{13.26}$$

La función **dct2** de MATLAB calcula la transformada coseno de dos dimensiones. Las siguientes instrucciones obtienen la transformada coseno, encuentran los valores menores a un cierto umbral y los eliminan haciéndolos ceros y, finalmente, obtienen la transformada coseno inversa.

```
load 'trees'
I=ind2gray(x,map);
J=dct2(I);
colormap(jet(64)),imagesc(log(abs(J)),colorbar
%se eliminan ahora valores menores a 10
nz=find(abs(J)<10);
J(nz)=zeros(size(nz));
k=idct2(J);
imshow(k,64),axis off
```

Este ejemplo nos ayuda a entender el proceso de compresión, asumiendo que únicamente se conservan los valores mayores al umbral establecido. Al incrementar el umbral, se obtienen tasas mayores de compresión pero una menor calidad de la imagen recuperada. Las imágenes resultantes para diferentes umbrales se observan en las figuras 13.10 y 13.11.

Figura 13.10. Imagen antes de comprimir

Figura 13.11. Imagen después de comprimir

Capítulo 14

Algoritmo para la eliminación de ruido impulsivo en imágenes

En este reporte [1] de aplicación se explica el funcionamiento general del algoritmo. Se mencionan los aspectos más importantes del funcionamiento del programa en C y se incluyen recomendaciones para que el programa pueda ejecutarse correctamente. Se presenta el programa en ensamblador para el TMS320C30 y, finalmente, algunas recomendaciones generales.

El algoritmo para eliminación de ruido fue propuesto por Eduardo Abreu, Michael Lightstone, Sanjit K. Mitra y Kaoru Arakawa, quienes lo publicaron en el artículo *A new efficent approach for the removal of impulse noise from highly corrupted images.* [2]

Este algoritmo fue diseñado para eliminar el ruido llamado "sal y pimienta", donde los niveles de ruido sólo toman dos valores, el máximo de blanco o el máximo de negro. Sin embargo, el algoritmo funciona con una eficiencia muy alta para ruidos con una distribución continua de probabilidad, como el ruido gaussiano.

14.1 Definición del algoritmo

El algoritmo se basa en el hecho de que los pixelcs vecinos referidos a un pixel central en una imagen sin ruido no tienen variaciones de valor muy grandes entre ellos. En caso de que el pixel central tenga un valor muy diferente al de los pixeles adyacentes, existe una probabilidad alta de que el pixel contenga un valor corrupto (tenga ruido).

En la figura 14.1 se muestra una ventana que identifica a cierto pixel central $x(n)$ y a sus ocho vecinos cercanos, de $x_1(n)$ a $x_8(n)$. Se define un vector $\mathbf{w}(\mathbf{n}) \in \Re^8$ que contiene los ocho elementos de observación alrededor de $x(n)$, tal que

$$\mathbf{w}(\mathbf{n}) = [x_1(n), x_2(n), ..., x_8(n)] \tag{14.1}$$

[1] Proyecto de investigación de B. Tovar, C. Estevez y R. Bustamante, Departamento de Electrónica, ITESM Campus Ciudad de México

[2] *IEEE Transactions on Image Processing, Vol. 5, No. 6, junio 1996.*

En este vector se ordenan las $x_i(n)$ de acuerdo con su valor numérico (en las imágenes esto quiere decir su nivel de gris):

$$\mathbf{r}(n) = [r_1(n), r_2(n), ..., r_8(n)] \tag{14.2}$$

Finalmente, se define el valor medio del vector ordenado (*R●M, rank-ordered mean*) como

$$m(n) = \frac{r_4(n) + r_5(n)}{2} \tag{14.3}$$

Un clasificador C identifica la ventana en una de M distintas clases por medio de la variable de estado s(n) = C(x(n), w(n)), que controla la función de filtrado F, definida como

$$y(n) = F(x(n), \mathbf{r}(n), s(n)) = \alpha_{s(n)}x(n) + \beta_{s(n)}m(n) \tag{14.4}$$

Donde α_i y β_i para i=1,...,M son los coeficientes correspondientes a cada uno de los estados posibles en los que la ventana puede ser clasificada. Para reducir el número de cálculos y la complejidad de diseño se elige:

$$\beta_i = 1 - \alpha_i, \qquad i = 1, ..., M \tag{14.5}$$

El clasificador C debe generar una función de estado s(n) tal, que sea un indicador probabilístico de que el pixel de la ventana esté corrupto. El clasificador C propuesto en el artículo opera sobre las diferencias entre el pixel de entrada x(n) y los demás pixeles del vector **r(n)**. Las diferencias se definen como

$$d_k(n) = \begin{cases} r_k(n) - x(n), & x(n) \leq m(n) \\ x(n) - r_{9-k(n)}, & x(n) > m(n) \end{cases} \tag{14.6}$$

para k = 1,...,4. En algunos casos la respuesta del algoritmo mejora considerablemente si se aplica de forma recursiva, es decir, se construye

$$\mathbf{w}(n) = [y_1(n), ..., y_4(n), x_5(n), ..., x_8(n)] \tag{14.7}$$

Con esto, los pixeles ya evaluados en ventanas anteriores que pudieron ser modificados se convierten en la información de las ventanas subsecuentes.

El caso más simple es M=2 y trabajar con sólo dos estados. Para esto se definen empíricamente cuatro umbrales $T_1 < T_2 < T_3 < T_4$, de los que el algoritmo detecta a x(n) como un pixel corrupto y asigna s(n) = 1 si cualquiera de las siguientes desigualdades es cierta:

$x_1(n)$	$x_2(n)$	$x_3(n)$
$x_4(n)$	x(n)	$x_5(n)$
$x_6(n)$	$x_7(n)$	$x_8(n)$

Figura 14.1. Ventana 3 × 3 de los pixeles vecinos de cierto pixel x(n)

$$d_k(n) > T_k, \quad k = 1, ..., 4. \tag{14.8}$$

De otra forma, el algoritmo supone que el pixel actual no está corrupto y asigna s(n)=2. El artículo original indica que, después de múltiples pruebas, los intervalos que se consideran para los umbrales son los siguientes:

$$T_1 \leq 15, \quad 15 \leq T_2 \leq 25, \quad 30 \leq T_3 \leq 50, \quad 40 \leq T_4 \leq 60 \tag{14.9}$$

Se define que para s(n)=1, $\alpha_1 = 0$ y para s(n)=2, $\alpha_2 = 1$. Con esto, si el pixel se considera corrupto, porque una de las diferencias ha sobrepasado uno de los umbrales, se reemplazará con la media del vector ordenado (ROM). En caso contrario, el pixel no se modificará.

La aproximación multiestado supone que dado que el pixel tiene ruido aditivo, todavía contiene información que se puede recuperar. Así, si por ejemplo cierto estado determina que $\alpha = 0.5$, entonces el pixel que se reemplazará será la mitad del actual más la mitad de la media del vector ordenado. Esto incrementa enormemente el número de cálculos, pero mejora la definición de la imagen final.

14.2 Programa en C

El programa está estructurado en cuatro funciones:

Función	**Descripción**
Recuperación	Guarda en tipo, reng, col, cmap y $z[][]$ el tipo de imagen, el número de renglones, columnas, colores y los pixeles de la imagen, respectivamente.
Ruido	Crea los pixeles corruptos a partir de los pixeles guardados en $z[][]$.
Orden	Ordena los pixeles de un vector, de **r** en este caso.
main	Realiza la operación del filtrado y crea la imagen final.

Los umbrales para las respectivas diferencias se definen al principio del programa como las contantes *T1, T2, T3, T4*. Se recomienda como experimentación cambiar sus valores para poder observar cómo afecta cada uno en la imagen final. También como constante se define *Maxt*, que es el número máximo de columnas o renglones que puede tener la imagen. Si el programa llegara a detenerse en su ejecución, tal vez el problema consiste en que la imagen tiene más renglones o columnas definidas por *Maxt*, por lo que bastará con cambiar su valor para probar imágenes más grandes.

El programa inicia preguntando por la imagen de prueba y por el nombre de la imagen final que se creará. Después se llama a la función *Recuperación*, que mejorará los parámetros de la imagen, como tamaño, número de colores, etc., y guardará todos los pixeles de la imagen en el arreglo bidimensional $z[][]$.

Una vez recuperada la imagen, la función *Ruido* hace uso de la generación de números aleatorios para simular el ruido con cierta distribución de probabilidad. La función viene comentada para variar el nivel de ruido en la imagen (por omisión de 20%).

La función principal (*main*) ejecuta el filtrado, teniendo como control de repetición el número de renglones y de columnas. Primero recupera la ventana por trabajar y la asigna a un arreglo (vector) w. Por comodidad en la programación, en w también se incluye el pixel central de la ventana, aunque en la definición del algoritmo no se definió así. Esta libertad se puede tomar, ya que la función de *orden* no tomará en cuenta este pixel. El arreglo w se ordena por la función *orden*, generando los valores del arreglo r.

Con los valores de r calculados, se procede a la determinación de m (en el algoritmo denominado como ROM). Es importante hacer notar que los subíndices de r utilizados son 3 y 4, y no 4 y 5 como se determinó en el algoritmo. Esto es porque los arreglos en C se numeran desde "cero" y no desde "uno", como en las ventanas del algoritmo.

Después se determinan las diferencias respectivas y se guardan en un arreglo d[]. En este punto, el programa se puede manipular para que funcione como biestado o como una propuesta de multiestado (en las líneas de código se incluyen los comentarios pertinentes para efectuar este cambio). Por omisión el programa correrá con una propuesta de multiestados, basada en coeficientes empíricos que encontramos que mejoraban el funcionamiento biestado. Para el lector que esté interesado en cómo producir una propuesta multiestados, se recomienda leer el artículo original, donde los autores proponen una aproximación por mínimos cuadrados, pero con coeficientes entrenados, lo que es una desventaja si se requiere que el algoritmo trabaje con imágenes de muy distinto contenido frecuencial. Nuestra propuesta empírica consiste en sumar "pedazos" de coeficiente a α, dependiendo de la magnitud de cada una de las diferencias.

El valor de α encontrado se utiliza para calcular el nuevo valor (si hubo algún cambio) del pixel en z[i][j]. El pixel actual z[i][j] se escribe en el archivo final de la imagen, para que el programa pruebe con la siguiente ventana. Dentro de la función de *main* los contadores i y j determinan el valor del renglón y de la columna actual, respectivamente.

Los resultados se pueden observar en la figuras 14.2, 14.3 y 14.4.

14.2.1 Recomendaciones generales y uso

El programa funciona con imágenes en formato ascii PGM. Este formato es muy común en sistemas operativos tipo UNIX, aunque aplicaciones manejadoras de imágenes comerciales en otros sistemas operativos también lo soportan. El programa se probó bajo *GNU Linux*, usando como compilador el *GNU C Compiler*. Para poder utilizar el formato en PGM, la imagen debe estar en escala de grises.

En algunas PC que corren otros sistemas operativos (aunque depende del compilador de C) se tienen problemas en el manejo de memoria de arreglos de variables grandes. Si el programa detiene su operación antes de terminar, probablemente sea un error de manejo de memoria en el sistema operativo, o bien, la constante *Maxt* que define el tamaño máximo

de imagen Maxt×Maxt tiene un valor pequeño respecto a la imagen que se probará. Si se tiene la aplicación XV [3], se podrán visualizar las imágenes automáticamente, al tiempo que se ejecuta el programa. La mayoría de los programas generadores de imágenes en formato PGM incluyen comentarios entre la línea que marca el tipo de imagen *P2* y los primeros datos numéricos. Los comentarios se denotan por un # como caracter inicial y deben de eliminarse del archivo antes de ejecutar el programa. El archivo PGM puede abrirse en cualquier editor de texto para eliminar los comentarios y debe guardarse como *sólo texto.*

Como el algoritmo no trabaja sobre los bordes (se puede realizar una aproximación), éstos se eliminan en la imagen final, es decir, se tendrán dos renglones y dos columnas menos.

En formato PPM el mismo algoritmo se puede implementar para imágenes a color. El formato PPM, denominado como tipo *P3,* guarda la imagen en forma RGB, por lo que cada pixel está compuesto de tres valores (cantidad de rojo, de verde y de azul). El programa debe modificarse para que se lean tres arreglos bidimensionales (uno para cada color) y no sólo $z[]$ como en escala de grises. Dado que los colores rojo, verde y azul son independientes entre sí, se puede aplicar la operación de filtrado a cada arreglo y después escribirlos en un archivo final. Una recomendación en este caso es volver la operación de filtrado una función aparte de *main* y que esta nueva función tenga como argumento un arreglo bidimensional. Así, esta función se llamaría tres veces, una para cada color.

```
/*********************PROGRAMA EN C*****************************/

#include <stdio.h>
#include <stdlib.h>
#define Maxt 650/*Maximo numero de renglones y columnas en la imagen*/
/* Definicion de los umbrales */

#define T1 5
#define T2 15
#define T3 30
#define T4 40

int reng, col, cmap;
int z[Maxt][Maxt]={0}; /* "z" guardara la imagen restaurada */
int w[9]={0}; /* Vector que guarda los pixeles de la ventana */
int r[8]={0}; /* Vector con los pixeles ordenados (rank) */
int d[4]={0}; /* Vector de diferencias */
FILE *imagen;
FILE *final;
char tipo[2];
char nombreo[20], nombref[20], tempc[25];
                /*Variables temporales para nombres de archivo*/
void ruido(int reng, int col, int cmap, char tipo[2]);
```

[3]Derechos de John Bradley

```
void recuperacion(void);
void orden(int rt[]);

main(){

  int i, j, k, l, p, temp, m;
  float alfa;

  printf("\n Imagen por probar: \t");
  scanf("%s", &nombreo);

  printf("\n Imagen final: \t \t");
  scanf("%s", &nombref);

  if((imagen=fopen(nombreo, "r"))==NULL)
    printf("\n Error: No se pudo encontrar el archivo\n");
  else
    final=fopen(nombref, "w");

  recuperacion();
  ruido(reng, col, cmap, tipo);

  fprintf(final, "%s\n%d %d\n", tipo, col-2, reng-2);
  fprintf(final, "%d\n", cmap);

  for(i=1; i<(reng-1); i++){
    for(j=1; j<(col-1); j++){
      temp=0;
      for(k=0; k<3; k++){
for(l=0; l<3; l++){
  w[temp]=z[i+k-1][j+l-1];
  temp++;
}
      }

      orden(w);

      m=(r[3]+r[4])/2;

      if (z[i][j]<=m){
for(p=0; p<4; p++)
  d[p]=r[p]-z[i][j];
      }
      else{
```

```
for(p=0; p<4; p++)
  d[p]=z[i][j]-r[7-p];
      }


      alfa=0;



      /**Para IMPLEMENTACION BIESTADO quitar comentarios a lo siguiente**/
      /**y comentar todas las lineas siguientes entre INICIO y FIN de**/
      /**PROPUESTA MULTIESTADOS *******/

      /*if ((d[0]>T1)||(d[1]>T2)||(d[2]>T3)||(d[3]>T4)){
z[i][j]=m;
alfa=1;
}*/

      /**********FIN IMPLEMENTACION BIESTADO*****/

      /**INICIO DE PROPUESTA MULTIESTADOS**********************************/

      if (d[0]>T1)
if (d[0]>(T1+20))
  alfa+=0.3333;
else
  if (d[0]>(T1+5))
    alfa+=0.1;
  else
    alfa+=0.05;

      if (d[1]>T2)
if (d[1]>(T2+20))
  alfa+=0.3333;
else
  if (d[1]>(T2+5))
    alfa+=0.1;
  else
  alfa+=0.05;

      if (d[2]>T3)
if (d[2]>(T3+30))
  alfa+=0.3333;
else
```

```
  if (d[2]>(T3+10))
    alfa+=0.2;
  else
    alfa+=0.1;

      if (d[3]>T4)
if (d[3]>(T4+30))
  alfa+=0.3333;
else
  if (d[3]>(T4+10))
    alfa+=0.2;
  else
  alfa+=0.1;


      if(alfa>1)
alfa=1;

      /**FIN DE PROPUESTA MULTIESTADOS**********************************/


z[i][j]=alfa*m+(1-alfa)*z[i][j];

fprintf(final, "%3d ", z[i][j]);
    }
  }

  fclose(imagen);
  fclose(final);

/**Quitar los comentarios a lo siguiente para visualizar automaticamente**/
/**las imageneS**/

  strcpy(tempc,"xv ");
  strcat(tempc, nombreo);
  strcat(tempc, " &");
  system(tempc);

  strcpy(tempc,"xv ");
  strcpy(tempc,"ruido.pgm ");
  strcat(tempc, " &");
  system(tempc);

  strcpy(tempc,"xv ");
```

```
  strcat(tempc, nombref);
  strcat(tempc, " &");
  system(tempc);

/*********************************/

}

void recuperacion(void){
  int i, j;
  fscanf(imagen, "%s", &tipo);
  fscanf(imagen, "%d", &col); /* No. de renglones de la imagen */
  fscanf(imagen, "%d", &reng); /* No. de columnas de la imagen */
  fscanf(imagen, "%d", &cmap);

  for(i=0; i<reng; i++)
    for(j=0; j<col; j++)
      fscanf(imagen, "%d", &z[i][j]);
}


void orden(int rt[]){
  int i, j, temp;

  r[0]=rt[0];
  r[1]=rt[1];
  r[2]=rt[2];
  r[3]=rt[3];
  r[4]=rt[5];
  r[5]=rt[6];
  r[6]=rt[7];
  r[7]=rt[8];

  for(i=0; i<8; i++){
    for(j=(i+1); j<8; j++){
      if(r[i]<r[j]){
temp=r[j]; r[j]=r[i];
r[i]=temp;
      }
    }
  }
}

void ruido(int reng, int col, int cmap, char tipo[2]){
```

```
  int i,j;
  FILE *ruido;

  srand(time());

  ruido=fopen("ruido.pgm", "w");

  fprintf(ruido, "%s\n%d %d\n", tipo, col, reng);
  fprintf(ruido, "%d\n", cmap);

  for(i=0; i<reng; i++)
    for(j=0; j<col; j++){
      if ((rand()%101)<20)/*Define ruido 20 = 20 por ciento */
if ((rand()%101)>50){/*Distribucion de 0 y 255. 50 = Partes iguales*/
  z[i][j]=255;
}
else{
  z[i][j]=0;
}
    fprintf(ruido,"%d ", z[i][j]);
    }
  fclose(ruido);

}
```

Figura 14.2. Praga, imagen original

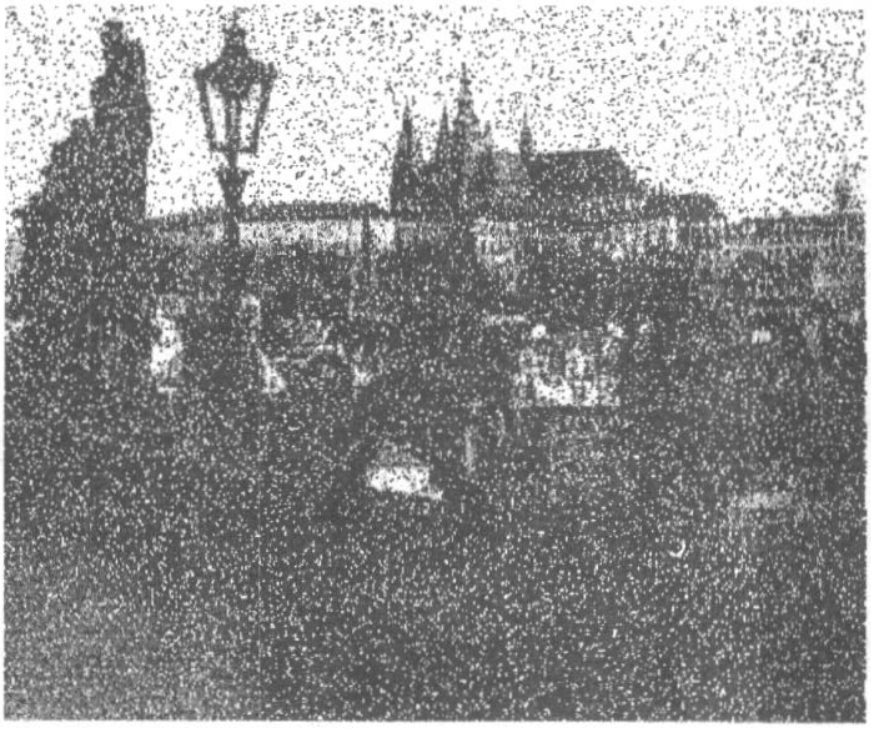


Figura 14.3. Praga, imagen con ruido al 20%

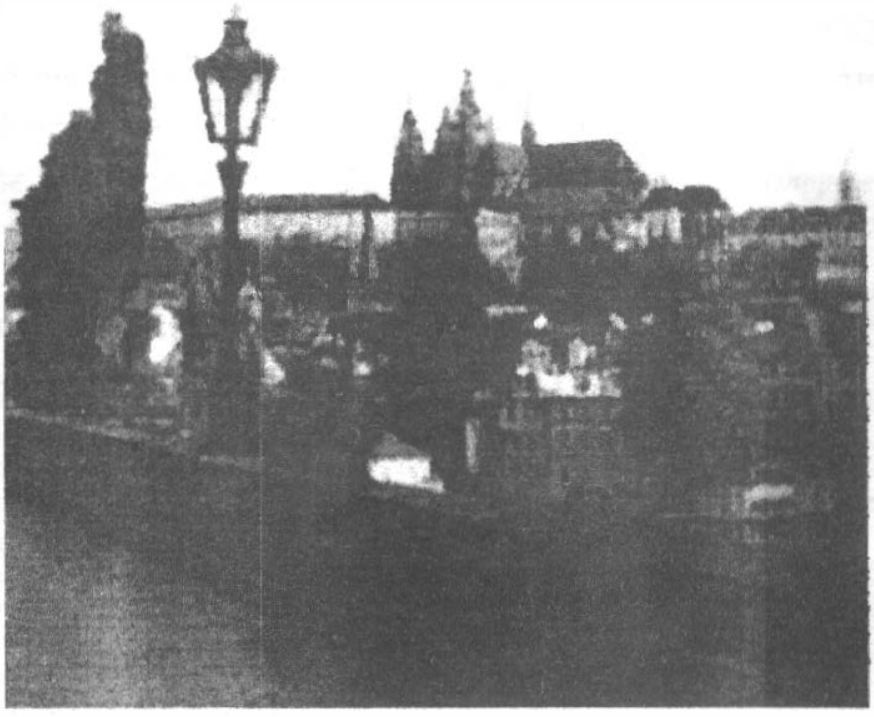


Figura 14.4. Praga, imagen recuperada

Bibliografía

BORGHESANI, C. and Ambardar, A.: *Mastering DSP Concepts Using Matlab.* New Jersey, Prentice Hall, 1998.

BRIGHAM, E.O.: *The Fast Fourier Transform*, New York, Prentice-Hall, 1983.

BURRUS, C.S. and Parks, T.W.: *DFT/FFT and Convolution.* New York, John Wiley and Sons, 1985.

CHASSAING, R. and Horning, D.W.: *Digital Signal Processing with the TMS320C25.* Toronto, John Wiley and Sons, 1990.

_____.: *Digital Signal Processing with the TMS320C30.* Toronto, John Wiley and Sons, 1992.

CHING, P.C. and Wu, S.W.: *Realtime Digital Signal Processing System Using a Parallel Processing Architecture.* Microprocessors and Microsystems, No.10, December 1989, pp. 653–658.

ELLIOTT, D.F. and Rao, K.R.: *Fast Transforms. Algorithms, Analyses, Applications.* New Jersey, Academic Press, 1982.

FETTWEIS, A.: *Digital Filters Structures Related to Clasical Filter Networks.* Arch. Elek. Uebertragung, vol. 25, pp. 79-89, Feb.1971.

FRANTZ, G.A.: *The Texas Instruments TMS320C25 Digital Signal Microcomputer.* IEEE Micro 6, No.6, 1986, pp. 10–28.

GÓMEZ, S. et al.: *An Application-specific FFT processor.* Electronic Engineering, No.6, June 1988, pp. 99–106.

GUNN, L.: *Chips Try Teaching Computers To Speak, Listen.* Electronic Design, No.11, May 1989, pp. 33–36.

HONIG, M.L. and Messerschmitt, C.G.: *Adaptive filters Structures, Algorithms and Applications.* New York, Kluwer Academic Publishers, 1986.

INGLE, V.K.: *Digital Signal Processing Using Matlab V.4.* Boston, PWS Publishing Company, ITP, 1997.

LACROIX, A.: *Digitale Filter.* 3. Auflage, Wien, Munchen, Oldenburg, 1988.

LACROIX, A. and Witte, K.H.: *Zeitdiskrete normierte Tiefpasse.* Heidelberg, Dr.Alfred Huthig Verlag, 1980.

LARIMORE, M.G.: *An Algorithm for Adapting IIR Digital Filters.* IEEE Trans. on ASSP, No.4, August 1980, pp. 428–440.

LARRY, E.S.:*Manual de laboratorio de procesamiento digital de señales.* México, UNAM, Facultad de Ingeniería, 2000.

LARRY, E.S.:*Arquitecturas de DSP's, Familia TMS320 y El TMS320C50.* México, UNAM, Facultad de Ingeniería, 2000.

LUIKUO, G., Fleming, M. and Magar, M.S.: *A 500 MOPS DSP Chip Set.* Electronic Engineering, No.6, June 1988, pp.106–113.

MARSHALL, D.F., Jenkins, W.K. and Murphy, J.J.: *The Use of Orthogonal Transforms for Improving Performance of Adaptive Filters.* IEEE Trans. on CAS, No.4, April 1989, pp.474–484.

MARVEN, G. and Ewers, G.: *A simple approach to Digital Signal Processing.* Oxford, Alden Press Limited, Texas Instruments, 1994.

McCLELLAN, J.H.: *The design of two-dimensional digital filter by transformation.* Proc. 7th. Annu. Princenton Conf. Information Sciences and Systems, 1973, pp. 247-251.

McCLELLAN, J.H., Schaffer, R.W. and Yoder M.A.: *DSP First a Multimedia Approach.* New York, Prentice Hall, 1998.

MILT, L.: *Signal–Processing Circuits.* Electronic Design. No.4, February 1990, pp. 101–108.

MITRA, S.K.: *Digital Signal Processing. A computer-Based Approach.* New York, McGraw-Hill Companies, 1998.

MITRA, S.K., Chacrabarti, S. and Abreu, E.: *The nonuniform Discrete Fourier Transform and its Signal Processing Applications.* Proc European Signal Processing Conference-EUSIPCO, Brussels, Belgium, August 1992, pp. 909-912.

MITRA, S.K. and Sherwood, R.J.: *Canonic realizations of digital filters using the continued fraction expansion.* IEEE Trans. Audio Electroacoustics, AU-20: August 1972, pp. 185-194.

_____.: *Digital Ladder Networks* IEEE Trans. Audio Electroacoustics, AU-21: February 1973, pp.30-36.

MITRA, S.K., Hirano, K., Furano, K. and Sherwood, R.J.: *Digital sine-coseno generator.* International Conference on Digital Signal processing, Florence, Italy, September 1975, pp. 142-149.

OPPENHEIM, A.V. and Schafer, R.W.: *Discrete-Time Signal Processing.* New Jersey, Prentice Hall, 1989.

_____.: *Digital Signal Processing.* New Jersey, Prentice Hall, 1985.

ORFANIDIS, S.J.: *Introduction to Signal Processing.* New Jersey, Prentice Hall, 1996.

PARKS, T.W. and Burrus, C.S.: *Digital Filter Design.* N.Y., John Wiley and Sons, 1987.

RAMÍREZ, F.M. y Moreno, J.A.: *Prácticas de procesamiento digital de Señales.* Centro de Investigación y de Estudios Avanzados del IPN, Departamento de Ingeniería Eléctrica, Octubre 1996.

RAO, K.R. and Yip, P.: *Discrete Cosene Transform.* San Diego, Academic Press, 1990.

SCHAFER, R.W. and Rabiner, L.R.: *Digital Representation of Speech Signal.* Proceedings of the IEEE, Vol.63, No.4, June 1975.

SHYNK, J.J.: *Adaptive IIR Filtering.* IEEE ASSP Magazine, No.2, April 1989.

SORENSEN, H.V. and Chen, J.: *A Digital Signal Processing Laboratory Using The TMS320C30.* New Jersey, Prentice Hall, Upper Sadle River, 1997.

STANLEY, W.D. and Dougherty, G.R.:*Digital Signal Processing.* San Diego, Reston Publishing corp., 1984.

STEARNS, S.D. and David, R.A.: *Signal Processing Algorithms In Matlab.* New Jersey, Prentice Hall, 1996.

STRAHLE, W.C.: *Adaptive Nonlinear Filter Using Fractal Geometry.* Electronic Letters, No.19, September 1988, pp. 1248–1250.

TEXAS INSTRUMENTS: *TMS320C25 Digital Signal Processor. Product Description.* Printed in USA, 1986.

_____.: *Second-Generation TMS320 User's Guide.* Printed in USA, 1987.

_____.: *TMS34020 User's Guide.* Printed in USA, 1987.

_____.: *TMS320C1x/TMS320C2x Assembly Language Tools User's Guide.* Printed in USA, 1987.

_____.: *TMS320C25 C Compiler Reference Guide.* Printed in USA, 1988.

_____.: *TMS320C5X Starter Kit Users Guide.* Printed in USA, 1995.

_____.: *TMS320 Family Development Suport-Reference Guide.* Printed in USA, 1994.

_____.: *TMS320C2X DSP Starter Kit Users Guide.* Printed in USA, 1993.

_____.: *TMS320C25 C Compiler Reference Guide.* Printed in USA, 1988.

VAN DEN ENDEN, A.W.M and Verhoeck, N.A.M.: *Discrete-time Signal Processing.* New Jersey, Prentice Hall, 1989.

WHITE, M.W. et al.: *New Strategies For Improving Spech Enhancement.* Int. Journal on Biomedical Computing, No. 25, 1990, pp. 101–124.

Índice analítico

Esta obra se terminó de imprimir
en enero de 2001
en el taller de imprenta del
Departamento de Publicaciones
de la Facultad de Ingeniería,
Ciudad Universitaria, México, D.F.
C.P. 04510

Secretaría de Servicios Académicos

El tiraje consta de 300 ejemplares
más sobrantes de reposición.