```
/*************************************************************************/
/***           RECONOCIMIENTO DE FONEMAS MEDIANTE RED NEURONAL           ***/
/***                                                                     ***/
/***      Marco Aurelio Alzate Monroy. Universidad Distrital F.J.C.      ***/
/***      Programa de Maestría en Teleinformática.   Junio de 1995.      ***/
/***                                                                     ***/
/***   Este programa captura tramas de señales de voz,  detectando las   ***/
/***   correspondientes  a fonemas  vocalizados para  encontrarles los   ***/
/***   parámetros propios de un modelo predictivo lineal: Coeficientes   ***/
/***   LPC, coeficientes de correlación parcial, coeficientes de razón   ***/
/***   de área logarítmica, coeficientes de par de espectro en línea y   ***/
/***   coeficientes  de seno inverso.  Cada uno de  estos conjuntos de   ***/
/***   coeficientes  se usan como  entrada a una  red neuronal  de una   ***/
/***   sola capa con cinco elementos de procesamiento, cada uno de los   ***/
/***   cuales posee 10 entradas para intentar clasificar la trama como   ***/
/***   correspondiente a un alófono de alguna de las cinco vocales del   ***/
/***   español (al menos del español hablado por el autor del proyecto)  ***/
/*************************************************************************/

/*************************************************************************/
// 33 de las funciones utilizadas se encuentran en archivos de encabezado:
#include <stdio.h>             // FILE, fopen, fscanf, fprintf, fclose
#include <stdlib.h>            // exit, ltoa, rand
#include <conio.h>             // clrscr, kbhit, getchar
#include <dos.h>               // FP_OFF, FP_SEG, MK_FP
#include <graphics.h>          // DETECT, initgraph, closegraph, cleardevice,
                               // setcolor, moveto, lineto, outtextxy, line,
                               // setviewport, clearviewport
#include <complex.h>           // complex, fabsl (<math.h>)
#include <string.h>            // strlen, strcpy, strcat
#include <alloc.h>             // farmalloc, farcoreleft, farfree
/*************************************************************************/
#define  UnSh unsigned short   // Se usarán muchos moldes (unsigned short)
#define iterando         0     // Estados posibles del proceso Raices:
#define dentrodetol      1     // Durante el proceso de búsqueda de raices
#define maxiteralcanzado 2     // complejas de los polinomios LSP pueden
#define casicero         3     // sucederse cada una de estas condiciones
#define muyplana         4
#define Jcasicero        5
typedef double vector[20];     // Coeficientes de polinomios (para Raices)
const  PI     = 3.141592654,   // Sin comentario
       fs     = 10000,         // Frecuencia de muestreo
       LTrama = 256;           // Tamano de la trama
unsigned char *BloqueD;        // Bloque de datos retornado por mallocSB()
unsigned char *Muestras;       // El mismo bloque, alineado en un párrafo
                               // para almacenar muestras de voz mediante DMA
unsigned char Direccion;       // Dirección del bloque alineado (Muestras)
double w[12][6];               // Coeficientes de la red neuronal
FILE  *ft;                     // Archivo de datos para el entrenamiento
/*************************************************************************/

/*************************************************************************/
/***************    RUTINAS DE CONTROL DE LA SOUNBLASTER    **************/
/*************************************************************************/

/*************************************************************************/

```

```
58  void EscribeDSP (unsigned char c)
59  // Esta funcion permite escribir un byte al puerto DSPWriteData/Command
60  {
61    while(inportb(0x022C)&0x80); // Espera que el DSPWriteBufferStatus se
62    outportb(0x022C,c);          // desactive antes de sacar el dato solicitado
63  }
64  /*************************************************************************/
65  unsigned char LeeDSP(void)
66  // Esta funcion permite leer un byte del puerto DSPReadData
67  {
68    while(!(inportb(0x22E)&0x80)); // Espera que se active el DSPDataAvailable
69    return(inportb(0x22A));        // antes de retornar el contenido del puerto
70  }
71  /*************************************************************************/
72  void IniciaSB(void)
73  // Esta función permite inicializar la tarjeta SoundBlaster
74  {
75      UnSh x;
76      inportb(0x022E);        // Ejecuta una secuencia de Reset:
77      outportb(0x0226,0x01);  //      Escribe un uno al registro de Reset
78      inportb(0x0226);        //      Espera mas de 3 microsegundos
79      inportb(0x0226);        //      Escribe un cero al registro de Reset
80      inportb(0x0226);        // Al finalizar la secuencia de reset la tarjeta
81      inportb(0x0226);        // debe responder con 0xAA dentro de los próximos
82      outportb(0x0226,0x00);  // milisegundos.
83      for(x=0;x<100;x++)      // Intenta durante cien lecturas:
84      {
85        if(inportb(0x022E)&0x80) // Si hay un dato disponible...
86        {
87            if(inportb(0x022A)==0xAA) break; // ... y es 0xAA : Todo ok
88        }
89      }
90      if(x==100)  // Si pasaron cien lecturas y la tarjeta no respondió con
91      {           // 0xAA es porque no existe una SoundBlaster en el equipo
92        closegraph();
93        printf("No existe una Sound Blaster en la Base de E/S 0220h\n");
94        exit(1);
95      }
96  }
97  /*************************************************************************/
98  void far *SBfarmalloc(long l)
99  // Reserva l bytes y retorna el puntero donde se encuentran.
100 // Si no hay suficiente memoria, detiene la ejecución del programa porque
101 // seria un error fatal.
102 {
103   void far *Puntero;
104   Puntero = farmalloc(l);   // Intenta reservar l bytes
105   if(Puntero==NULL)         // Reporta en caso de falta de memoria
106   {
107     closegraph();
108     printf("Error de asignacion de memoria:\n");
109     printf("  Quedaban %li bytes y pidieron %li",farcoreleft(),l);
110     exit(1);
111   }
112   return Puntero;       // Devuelve un puntero al bloque de memoria reservado
113 }
114 /*************************************************************************/
115 void MallocSB(void)
```

```
116  // Esta funcion permite disponer de un bloque de memoria donde almacenar
117  // las muestras de sonido
118  {
119      unsigned long DirFisica; // Direccion fisica de 20 bits donde se almace-
120                               // naran las muestras de sonido
121      BloqueD=(unsigned char *)SBfarmalloc(131071L);   // Reserva memoria
122      DirFisica=(( unsigned long)FP_OFF(BloqueD)) +    // Calcula la direccion
123             (((unsigned long)FP_SEG(BloqueD))<<4);   // del bloque reservado
124      DirFisica+=0x0FFFFL; // Encuentra un segmento a partir del cual se pue-
125      DirFisica&=0xF0000L; // dan direccionar 64K con las l6 lineas menos sig-
126      Direccion=(DirFisica>>16)&15; // nificativas del bus de direcciones
127      // Crea un puntero de muestras para ser usado por el DMA en el programa
128      Muestras=(unsigned char *)MK_FP(((UnSh)Direccion<<12)&0xF000,0);
129  }
130  /*************************************************************************/
131  void EstableceTC(unsigned char tc)
132  // Esta función permite establecer la constante de tiempo, calculada
133  // segun la tasa de muestreo asi: tc = 256 - 1000000/tasa_de_muestreo)
134  {
135      inportb(0x022E);    // Lectura "profilactica"
136      EscribeDSP(0x40);   // Comando para escribir la constante de tiempo
137      EscribeDSP(tc);     // Constante de tiempo
138  }
139  /*************************************************************************/
140  void DetieneDMA(void)
141  // Esta funcion detiene cualquier proceso de E/S mediante DMA
142  {
143      EscribeDSP(0xD0);
144  }
145  /*************************************************************************/
146  void DesactivaParlante(void)
147  // Esta funcion desconecta el parlante de la salida del DAC
148  {
149      EscribeDSP(0xD3);   // Comando de desactivacion del parlante
150  }
151  /*************************************************************************/
152  void GrabaSB(UnSh NumMuestras)
153  // Esta funcion captura NumMuestras bytes del conversor ADC y las almacena
154  // en *Muestras (deben ser menos de 65535)
155  {
156      NumMuestras--;           // Desde cero hasta NumMuestras-1
157      outportb(0x0A,0x05);     // Enmascara el canal 1 de DMA
158      outportb(0x0C,0x00);     // Borra el puntero
159      outportb(0x0B,0x45);     // Modo de transferencia: por bytes, inrementan-
160                               // do direccion, de IO a memoria, canal 1
161      outportb(0x02,0);        // Direccion base, parte baja
162      outportb(0x02,0);        //                 parte alta
163      outportb(0x83,Direccion);// Pagina de direcciones para el canal 1
164      outportb(0x03,(unsigned char)(NumMuestras&0xFF));      // Numero de bytes
165      outportb(0x03,(unsigned char)((NumMuestras>>8)&0xFF)); // (lo,hi)
166      outportb(0x0A,0x01);     // Desenmascara el canal 1 del DMA
167      EscribeDSP(0x24);        // Conversion ADC de 8 bits en modo DMA
168      EscribeDSP((unsigned char)(NumMuestras&0xFF)); // Numero de muestras para
169      EscribeDSP((unsigned char)((NumMuestras>>8)&0xFF)); // el DSP
170  }
171  /*************************************************************************/
172  UnSh EstadoDMA(void)
173  // Esta funcion retorna el estado del DMA: El bit 1 del registro de estado
```

```
// se pone en uno cuando el canal uno alcanzo el numero de transferencias
// solicitado.
{
    return(inportb(0x0008)&2);
}
/*************************************************************************/
void Capturar(long longitud)
// Esta funcion permite la captura de una senal de voz a traves del ADC
{
  char c;                       // Constante de tiempo
  IniciaSB();                   // Programa la SoundBlaster
  c = 256 - 1000000L/fs;        // Calcula la constante de tiempo
  EstableceTC(c);               // y programa la tasa de muestreo de la SB
  DesactivaParlante();          // Condición para captura de señales
  GrabaSB(longitud);            // Ordena la transferencia DMA
  EstadoDMA();                  // Espera la terminación de la transferencia
  while(!EstadoDMA());
  DetieneDMA();
}
/*************************************************************************/
long captura(void)
// Esta funcion captura LTrama muestras de voz y retorna su energia,
// para determinar si se trata de un fonema vocalizado o no.
{
  int j,m;                              // Indice de las muestras, muestra
  long Energia=0;                       // Energia total de la trama
  Capturar(LTrama);                     // Captura una trama
  for(j=0;j<LTrama;j++)                 // Calcula la energia de cada
  {                                     // muestra y se la anade a la
    m = (int)(*(Muestras + j) - 125);   // energia total.
    Energia += m*m;
  }
  return Energia;                       // Retorna la energia total
}
/*************************************************************************/

/*************************************************************************/
/*************    RUTINAS AUXILIARES DE TRATAMIENTO NUMERICO    ************/
/*************************************************************************/

/*************************************************************************/
// Esta función busca un factor cuadrático del polinomio a[], de la forma
// x2 - ux - v.
void Raices(vector  a,        // Coeficientes del polinomio original
            double  u,        // Valor propuesto de u
            double  v,        // Valor propuesto de v
            int     n,        // Grado del polinomio
            double  tol,      // Tolerancia en los valores de u y v
            int     maxiter,  // Numero maximo de iteraciones
            double& unueva,   // Nuevo valor de u
            double& vnueva,   // Nuevo valor de v
            int&    numdeiter,// Numero de iteraciones
            vector& b,        // Coeficientes del polinomio residuo
            int&    resultado)// Indica si hubo exito en la busqueda
{
  double cero = 1.0e-20,      // Entre -1E-20 y +1E-20 se considera 0
         r, s,                // Coeficientes del residuo lineal (---> 0)
         b0, b1,              // Primeros coeficientes del residuo total
```

```
          c0, c1, c2,          // Derivadas de s, r y b0 respecto a u
          J,                   // Jacobiano de (r,s) con (u,v)
          du, dv;              // Correcciones para u y v en cada iteración
  int     contiter,            // Contador de iteraciones
          i,                   // Indice de los coeficientes
          estado;              // Indica el estado del proceso

  contiter = 0;                       // El proceso consiste en expresar el
  estado   = iterando;                // polinomio a(x) como
  do                                  //
  {                                   // a(x) = [x^2-u*x-v]*b(x) + r*[x-u] + s
    ++contiter;                       //
    b0 = a[n]; r = a[n-1] + u*b0;     // y buscar iterativamente los valores
    c1 = b0; c0 = r + u*c1;           // de u y v que hacen (r,s) ---> (0,0),
    for(i=n-2;i>0;i--)                // de manera que la expresion cuadratica
    {                                 // [x^2-u*x-v] sea un factor del poli-
      b1 = b0; b0 = r;                // nomio original a(x), y b(x) sea el
      r = a[i] + u*b0 + v*b1;         // residuo del polinomio respecto a
      c2 = c1; c1 = c0;               // dicho factor cuadrático.
      c0 = r + u*c1 + v*c2;           // (Se conoce como método Bairstow).
    }
    s = a[0] + u*r + v*b0;
    J = c0*c2 - c1*c1;                // Si el jacobiano de la transformación
    if(fabsl(J)<cero)                 // (r,s) en (u,v) se hace muy pequeño
       estado = Jcasicero;            // no es posible continuar
    else
    {
      du = (r*c1 - s*c2)/J;           // Calcula los incrementos de corrección
      dv = (s*c1 - r*c0)/J;           // de u y v.
      u += du; v += dv;
      if((fabsl(u)<cero)||(fabsl(v)<cero))  // "bajoflujo"
       estado = casicero;
      else if((fabsl(du/u)<tol)&&(fabsl(dv/v)<tol)) // Listo! Se satisfizo la
       estado = dentrodetol;                        // prueba de error relativo
      else if(contiter>maxiter)       // Demasiadas iteraciones
       estado = maxiteralcanzado;
    }
  } while(estado == iterando);
  resultado = estado;                 // Indica si hubo exito o no
  numdeiter = contiter;               // Informa cuantas iteraciones hizo
  unueva = u;                         // Retorna los coeficientes del
  vnueva = v;                         // factor cuadratico  y los
  b[n-2]=a[n];                        // coeficientes del polinomio
  b[n-3]=a[n-1]+u*b[n-2];             // residuo.
  if(n>3)
  {
    for(i=n-4;i>=0;i--)
      b[i] = a[i+2] + u*b[i+1] + v*b[i+2];
  }
}
/*****************************************************************************/

```

```
284   void bitreverse(complex z[])
285   // Esta funcion reordena las muestras de acuerdo con el algoritmo de
286   // reverso de bits entre los indices de las muestras.
287   {
288     int     i,     // indice de las muestras
289             m,     // indice i con los bits invertidos
290             l;     // variable auxiliar para el calculo de m
291     complex ro;    // variable temporal para intercambio de muestras
292
293     m=0;
294     for(i=1;i<=LTrama - 3;i++) // Recorre todas las muestras
295     {
296       l=(LTrama / 2);
297       while ((m+l)>=LTrama) l=(l / 2);
298       m=(m % l) + l;           // Calcula el indice inverso
299       if(i<m)
300       {                        // Si no ha hecho el cambio,
301         ro=z[i];               // intercambia estas dos muestras
302         z[i]=z[m];
303         z[m]=ro;
304       }
305     }
306   }
307   /*************************************************************************/
308   void butters(int signo, complex z[])
309   // Esta funcion realiza las operaciones 'mariposa' sobre las muestras
310   // previamente invertidas. Si signo es +1, se obtiene la FFT. Si signo
311   // es -1 se obtiene la IFFT.
312   {
313     int i,l,m;                     // Indices de las muestras
314     complex a,e;                   // Muestra de intercambio, termino exponenc
315     l=1;
316     while(l <= (LTrama / 2))       // l = 1, 2, 4, 8, 16, 32, 64, ...
317     {                              // Despues de log2(LTrama) descomposiciones
318       for(m=0;m<l;m++)             // se obtiene la DFT total como combinacion
319       {                            // lineal de varias DFTs de 2 terminos
320         e=complex(0,-PI*((double)((double)m/(double)l))*signo);
321         e=exp(e);
322         i=m;
323         while(i<=(LTrama-l))
324         {
325         a = z[i+l]*e;              // Operacion "mariposa" : DFT de 2 términos
326         z[i+l] = z[i] - a;
327         z[i] += a;
328         i+=2*l;
329         }
330       }
331       l=2*l;
332     }
333   }
334   /*************************************************************************/
335   void espectro(int signo, complex z[])
336   // Esta funcion grafica la amplitud del espectro en escala logaritmica (dB)
337   // Si signo es -1, no se considera z[] sino 1/z[] (filtro inverso)
338
```

```
{
  const  LT = LTrama/2;      // Solo las primeras muestras son significativas
  double a[LT+10],           // Amplitud logaritmica
         ro;                 // Variable auxiliar
  int    i,                  // Indice
         amp[LT+10],         // Amplitud escalizada para la pantalla
         x, y;               // Coordenadas en pantalla
  char   dato[80];           // Escala en pantalla

  setviewport(0,0,639,479,1); // Trabaja con toda la pantalla
  setcolor(LIGHTRED);
  line(128,130,639,130);      // Traza el cero del osciloscopio
  moveto(128,270);            // Traza un marco
  lineto(639,270); lineto(639,470);
  lineto(128,470); lineto(128,270);
  if(signo<0) setcolor(LIGHTGREEN);
  else setcolor(YELLOW);           // El color distingue la envolvente
  for(i=0;i<LT;i++)
  {
    ro=abs(z[i]);                  // Espectro de amplitud
    if(ro<0.0001) ro=0.0001;       // Evita sobreflujos (hasta -80 dB)
    a[i] = signo*20.0*(log10(ro)); // Signo menos para el filtro inverso
    if(signo<0) a[i]+=35;          // Superpone la envolvente
    if(a[i]>100.0) a[i]=100.0; if(a[i]<0.0) a[i]=0.0; // Controla picos
    amp[i]=(int)floor(470.5 - 2.0*a[i]); // Escaliza para la pantalla
  }
  ro = 128.0;
  moveto(128,amp[0]);              // Se ubica en el punto inicial
  for(i=1;i<LT;i++)
  {
      ro += 512.0 / (double)LT;    // Ajusta LTrama/2 muestras a 512 pixels
      x = (int)floor(ro+0.5);
      lineto(x,amp[i]);
  }
  setcolor(LIGHTRED);              // Escribe la escala del espectroscopio
  outtextxy(102,270,"100");        // (en dBs)
  outtextxy(118,462,"0");
}
/*************************************************************************/
void FFT(int signo, double *ventana)
// Calcula la FFT de la trama escogida y presenta  la  amplitud  espectral
// Si signo es -1, grafica la respuesta del filtro (1/ventana[])
{
  complex z[LTrama+5];
  for(int i=0;i<LTrama;i++)    // Transfiere las muestras al arreglo complejo
    z[i]=complex(ventana[i],0);
  bitreverse(z);               // Reordena las muestras
  butters(1,z);                // Efectua el algoritmo FFT
  espectro(signo, z);          // Traza la grafica
}
/*************************************************************************/
void polinomioLPC(double *parametros, double *polinomio, int NCoeficientes)
// Esta funcion calcula NCoeficientes predictores de la ventana de LTrama
// muestras que se encuentra en Muestras, retornando los coeficientes pre-
// dictores en polinomio[]. En parametros[] puede devolver los coeficientes
// Parcor, LSP, IS ó LAR
{
```

```
    int   i,j,k,l;           // Indices auxiliares
    double *ro;              // Autocorrelacion
    double *alfa;            // Vector de coeficientes LPC
    double *alfa_;           // Vector de coeficientes LPC de grado menor
    double Energia;          // Energía del residuo de la prediccion
    double G;                // Ganancia del filtro inverso
    double *Ventana;         // Muestras vistas a traves de la ventana
    double *parcor;          // Coeficientes de correlacion parcial
    double *LSP_P, *LSP_Q;   // Polinomios para calculo de los parámetros LSP
    int    grado;            // Grado de los residuos de P(z) y Q(z) -LSP-
    vector a,b;              // Polinomio y residuo en el calculo de raices
    double u,v;              // Coeficientes de los factores cuadráticos
    double Formanto[3];      // Tres principales frecuencias de resonancia
    int    numdeiter,        // Numero de iteraciones realizadas en Raices()
           resultado;        // Posibles estado finales de la funcion Raices()

    NCoeficientes++;         // Grado del polinomio mas termino independiente
    Ventana = (double *)SBfarmalloc((LTrama+2)*sizeof(double));       // Reserva
         ro = (double *)SBfarmalloc((NCoeficientes+2)*sizeof(double));// memoria
       alfa = (double *)SBfarmalloc((NCoeficientes+2)*sizeof(double));
      alfa_ = (double *)SBfarmalloc((NCoeficientes+2)*sizeof(double));
     parcor = (double *)SBfarmalloc((NCoeficientes+2)*sizeof(double));
     LSP_P  = (double *)SBfarmalloc((NCoeficientes+5)*sizeof(double));
     LSP_Q  = (double *)SBfarmalloc((NCoeficientes+5)*sizeof(double));
    Ventana[0] = (double)(Muestras[0]) - 125.0; // Elimina el nivel dc del ADC
    for(j=1;j<LTrama;j++)                       // Pre-énfasis: H(z)=1-0.95z^-1
      Ventana[j]=((double)(Muestras[j])-125)-0.95*((double)(Muestras[j-1])-125);
    for(j=0;j<LTrama;j++)                       // Ventana Hamming
      Ventana[j] *= (0.54 - 0.46*cos(2*PI*(double)j/(double)(LTrama-1)));
    for(k=0;k<NCoeficientes;k++)                // Calcula la autocorrelacion
    {
      ro[k]=0;
      for(l=0;l<LTrama-k;l++) ro[k] += Ventana[l]*Ventana[l+k];
    }
    Energia   =  ro[0];                    // Obtiene los coeficientes LPC por
    parcor[1] =  ro[1]/Energia;            // el método de la autocorrelación
    alfa[1]   =  parcor[1];                // evaluado mediante el algoritmo
    Energia  *= (1 - parcor[1]*parcor[1]); // Levinson-Durbin, el cual permite
    for(l=2; l<NCoeficientes;l++)          // obtener de una vez los parámetros
    {                                      // de reflexión o correlacion par-
      parcor[l] = ro[l];                   // cial, Parcor. Este algoritmo a-
      for(i=1;i<l;i++) parcor[l] -= alfa[i]*ro[l-i]; // provecha la caracterís-
      parcor[l] /= Energia;                // tica Toeplitz de la matriz de
      alfa[l] = parcor[l];                 // autocorrelaciones para no tener
      for(i=1; i<l; i++) alfa_[i] = alfa[i] - parcor[l]*alfa[l-i]; // que
      for(i=1; i<l; i++) alfa[i] = alfa_[i]; // invertirla.
      Energia *= (1 - parcor[l]*parcor[l]);
    }
    polinomio[0] = 1;                          // Construye el polinomio pre-
    LSP_P[0] = 1;  LSP_P[NCoeficientes] =  1;  // dictor A(z) y los polinomios
    LSP_Q[0] = 1;  LSP_Q[NCoeficientes] = -1;  // P(z) = A(z) + A(1/z)*z^-(p+1)
    for(j=1;j<NCoeficientes;j++)               // Q(z) = A(z) - A(1/z)*z^-(p+1)
```

```
449     {                                          // los cuales tienen NC raices
450       polinomio[j]  = -alfa[j];                // complejas conjugadas, cuyos
451       LSP_P[j] = polinomio[j] + polinomio[NCoeficientes-j]; // argumentos son
452       LSP_Q[j] = polinomio[j] - polinomio[NCoeficientes-j]; // los parametros
453     }                                                       // LSP
454     for(j=0;j<=NCoeficientes;j++) a[j]=LSP_P[j]; // Calcula todas las raices
455     grado = NCoeficientes;                       // complejas conjugadas de
456     u = 1.0; v = 0.0;                            // P(z) encontrando sus fac-
457     i = 0;                                       // tores cuadráticos
458     while(grado>2)                               // (x^2 - u*x - v)
459     {
460       do                                         // Itera hasta encontrar un
461       {                                          // par (u,v) que converja
462         Raices(a,u,v,grado,0.001,80,u,v,numdeiter,b,resultado);
463         if(resultado!=1)                         // Si despues de 80 itera-
464         {                                        // ciones no ha convergido o
465         u = 1.0 - 2.0*((double)rand())/((double)RAND_MAX); // se ha presentado
466         v = sqrt(1 - fabsl(u*u));                // inestabilidad numérica, es-
467         }                                        // coge otro punto de partida
468       } while(resultado!=1);
469       LSP_P[i] = sqrt(fabsl(u*u + 4*v))/2.0;     // Solución a la ecuación
470       LSP_P[i] = arg(complex(u/2,LSP_P[i]));     // cuadrática, cuyo argumento
471       i++; grado -=2;                            // es el parametro LSP deseado
472       for(resultado=0;resultado<=grado;resultado++) a[resultado]=b[resultado];
473     }
474     for(j=0;j<=NCoeficientes;j++) a[j]=LSP_Q[j]; // Repite el procedimiento
475     grado = NCoeficientes;                       // para las raices de Q(z).
476     u = -1.0; v = 0.0;                           // P(z) y Q(z) tienen, en con-
477     while(grado>2)                               // junto NCoef-1 factores cua-
478     {                                            // draticos que corresponden a
479       do                                         // NCoef-1 frecuencias de re-
480       {                                          // sonancia.
481         Raices(a,u,v,grado,0.001,80,u,v,numdeiter,b,resultado);
482         if(resultado!=1)
483         {
484         u = 1.0 - 2.0*((double)rand())/((double)RAND_MAX);
485         v = sqrt(1 - fabsl(u*u));
486         }
487       } while(resultado!=1);
488       LSP_P[i] = sqrt(fabsl(u*u + 4*v))/2.0;
489       LSP_P[i] = arg(complex(u/2,LSP_P[i]));
490       i++; grado -=2; // Transfiere el residuo para hallar sus factores cuadráti.
491       for(resultado=0;resultado<=grado;resultado++) a[resultado]=b[resultado];
492     }
493     for(j=0;j<NCoeficientes-2;j++)       // Ordena los coeficientes LSP
494      for(i=j+1;i<NCoeficientes-1;i++)    // mediante el metodo sencillo
495       if(LSP_P[i]<LSP_P[j])              // de la burbuja (el arreglo es
496       {                                  // suficientemente pequeno para
497         u = LSP_P[i];                    // justificar el metodo).
498         LSP_P[i] = LSP_P[j];
499         LSP_P[j] = u;
500       }
501     for(j=0;j<NCoeficientes-2;j++)  // Calcula distancias entre raices adyacentes
502         LSP_Q[j] = LSP_P[j+1] - LSP_P[j];
503
```

```
  for(j=0;j<3;j++)                // Busca las tres más "pegadas"
  {
    u=LSP_Q[0]; k=0;
    for(i=1;i<NCoeficientes-2;i++)
    if(LSP_Q[i]<u)
    {
      u = LSP_Q[i];
      k=i;
    }
    Formanto[j]=(LSP_P[k] + LSP_P[k+1])/2;
    LSP_Q[k] = 1.0e+20;
  }
  for(j=0;j<2;j++)
   for(i=j+1;i<3;i++)
    if(Formanto[i]<Formanto[j])
    {
           u = Formanto[i];
      Formanto[i] = Formanto[j];
      Formanto[j] = u;
    }
  for(j=0;j<3;j++) fprintf(ft,"%lf\n",Formanto[j]);
/*
  for(j=1;j<NCoeficientes;j++)                  // Almacena los resultados en
  {                                             // un archivo para entrenam.
    if(parcor[j]>=1.0)  parcor[j]= 0.9999;      // Evita sobreflujos en el
    if(parcor[j]<=-1.0) parcor[j]=-0.9999;      // calculo LAR e IS
    fprintf(ft,"%lf\n",polinomio[j]);           // Parámetros LPC
    fprintf(ft,"%lf\n",parcor[j]);              // Parámetros Parcor
    fprintf(ft,"%lf\n",2*asin(parcor[j])/PI);   // Parámetros IS
    G = (1.0 + parcor[j])/(1.0 - parcor[j]);
    if(G<=0.0) G=1e-10;
    fprintf(ft,"%lf\n",0.5*log(G));             // Parámetros LAR
    fprintf(ft,"%lf\n",LSP_P[j-1]);             // Parámetros LSP
  }
*/
  for(j=1;j<NCoeficientes;j++)
     parametros[j] = parcor[j];    // Devuelve los coeficientes ParCor
  farfree(Ventana);                // Libera la memoria reservada
  farfree(ro);
  farfree(alfa);
  farfree(alfa_);
  farfree(parcor);
  farfree(LSP_P);
  farfree(LSP_Q);
}
/*************************************************************************/

/*************************************************************************/
/*************         RUTINAS DE CLASIFICACION NEURONAL        ************/
/*************************************************************************/

/*************************************************************************/
void RedNeural(void)
// Esta funcion carga los pesos de las conexiones sinapticas para
// reconocimiento de vocales mediante red neural.

```

```
{
  int  i,                          // Indice de las entradas sinapticas
       j;                          // Indice de las neuronas
  FILE *fpt;                       // Archivo de coeficientes de la red neural
  clrscr();                        // Si no existe el archivo, reporta el
  if(!(fpt=fopen("pesosNN.dat","r"))) // error y termina la ejecución pues
  {                                // no hay nada que hacer sin él.
    printf("No se pudo abrir pesosNN.dat\n");
    exit(1);
  }
  for(i=0;i<11;i++)                // Se trata de un perceptron de una
     for(j=0;j<5;j++)              // sola capa con conexion completa
      fscanf(fpt,"%lf",&w[i][j]); // entre 10 entradas y 5 neuronas, cada
  fclose(fpt);                     // una de las cuales tiene una entrada
}                                  // de polarización -w[10][j];
/*****************************************************************************/
void reconoce(void)
// Esta funcion hace un analisis predictivo lineal de la trama de voz
// capturada, encontrando los coeficientes que se usaran como entradas
// a una red neural para reconocimiento de vocales. Luego evalúa la red
// para clasificar la trama de voz capturada.
{
  const  grado = 10;               // Grado del polinomio predictor
  double ventana[LTrama],          // Ventana Hamming con las muestras
         r[25],                    // Polinomio predictor
         p[25],                    // Coeficientes para reconocimiento (LSP,...)
         y4,y3,y2,y1,y0;           // Neuronas de reconocimiento
  int    j;                        // Indice auxiliar
  char   dato[6];                  // Resultado del reconocimiento

  for(j=0;j<LTrama;j++) ventana[j] = (double)(*(Muestras+j)-125);
  FFT(1,ventana);                  // Despliega el espectro
  polinomioLPC(p,r,grado);         // Calcula los coeficientes LPC y los
                                   // parametros para reconocimiento
  y4 = y3 = y2 = y1 = y0 = 0.0;    // Cinco neuronas, una para cada vocal
  for(j=0;j<grado;j++)
  {
    y4 += w[j][4]*p[j+1];          // Pondera las conexiones sinapticas
    y3 += w[j][3]*p[j+1];          // de los parámetros de entrada
    y2 += w[j][2]*p[j+1];
    y1 += w[j][1]*p[j+1];
    y0 += w[j][0]*p[j+1];
  }
  y4 -= w[grado][4];               // Anade la polarizacion
  y3 -= w[grado][3];
  y2 -= w[grado][2];
  y1 -= w[grado][1];
  y0 -= w[grado][0];

  strcpy(dato,"");
  if(y4>0) strcat(dato,"U");       // Funcion de activacion de las neuronas:
  if(y3>0) strcat(dato,"O");       //                +---------------
  if(y2>0) strcat(dato,"I");       //                |
  if(y1>0) strcat(dato,"E");       //  --------------+
  if(y0>0) strcat(dato,"A");       //                0
  if(!strlen(dato)) strcpy(dato,"?"); // No pudo reconocer ninguna vocal
  for(j=0; j<=grado; j++) ventana[j] = r[j];      // Exita el filtro inverso
```

```
617     for(j=grado+1; j<LTrama; j++) ventana[j] = 0.0; // con un impulso para gra-
618     y0=0;                                           // ficar su respuesta es-
619     FFT(-1,ventana);                                // pectral de amplitud
620     outtextxy(44-4*strlen(dato),300,dato); // Informa el resultado del recono-
621     getch();                              // cimiento y espera una tecla para
622     setviewport(90,270,639,470,1);        // continuar
623     clearviewport();                      // Borra el espectro pero deja el
624     setcolor(LIGHTRED);                   // marco del espectroscopio
625     line(38,0,549,0);
626     line(38,0,38,200);
627     line(38,200,549,200);
628     line(549,200,549,0);
629     setviewport(0,299,90,309,1);
630     clearviewport();               // Borra el resultado del reconocimiento
631     setviewport(129,1,638,254,1);  // Establece la ventana del osciloscopio
632     setcolor(WHITE);
633   }
634   /*************************************************************************/
635
636   /*************************************************************************/
637   /******              RUTINAS DE PRESENTACION GRAFICA                 *******/
638   /*************************************************************************/
639
640   /*************************************************************************/
641   void modografico(void)
642   // Esta funcion establece el modo  grafico en la pantalla y muestra el
643   // formato de presentacion del osciloscopio y el espectroscopio
644   {
645     int driver=DETECT, modo;       // Parámetros de InitGraph
646     double mx;                     // Extremo derecho en el osciloscopio
647     char dato[80];                 // Para presentacion de mensajes
648     initgraph(&driver,&modo,"");   // Establece el modo grafico en la pantalla
649     cleardevice();                 // Borra la pantalla
650     setcolor(LIGHTRED);            // Selecciona el color
651     moveto(128,0);                 // Borde del osciloscopio
652     lineto(639,0); lineto(639,255);
653     lineto(128,255); lineto(128,0);
654
655     moveto(128,270);                 // Borde del espectroscopio
656     lineto(639,270); lineto(639,470);
657     lineto(128,470); lineto(128,270);
658
659     outtextxy(102,0,"130");          // Escala de amplitud del osciloscopio
660     outtextxy(94,247,"-125");
661     outtextxy(118,127,"0");
662
663     outtextxy(128,257,"0");          // Escala de tiempo del osciloscopio
664     mx = ((double)LTrama)*(1000.0/(double)fs);    // (ms)
665     sprintf(dato,"%5.2lf",mx);
666     outtextxy(610,257,dato);
667
668     outtextxy(128,472,"0");          // Escala de frecuencia del espectroscopio
669     mx = (double)fs / 2.0;           // (Hz)
670     sprintf(dato,"%7.2lf",mx);
671     outtextxy(592,472,dato);
672
673     line(12,139,84,139);             // Borde del titulo del proyecto
674     line(12,165,84,165);
```

```
675     line(12,139,12,165);
676     line(84,139,84,165);
677
678     line(0,5,94,5);                  // Borde del titulo general
679     line(0,245,94,245);
680     line(0,5,0,245);
681     line(94,5,94,245);
682
683     outtextxy(16,143,"Proyecto");  outtextxy(16,153,"SINVOZES"); // Mensajes
684
685     setcolor(LIGHTBLUE);
686     outtextxy(4,13,"UNIVERSIDAD"); outtextxy(12,23,"DISTRITAL");
687     outtextxy(24,33,"F.J.C.");
688
689     outtextxy(16,63,"Programa");   outtextxy(40,73,"de");
690     outtextxy(16,83,"Maestría");   outtextxy(40,93,"en");
691     outtextxy(8,103,"Teleinfor-"); outtextxy(24,113,"mática");
692
693     outtextxy(36,183,"RED");       outtextxy(24,193,"NEURAL");
694     outtextxy(32,203,"PARA");      outtextxy(12,213,"RECONOCI-");
695     outtextxy(12,223,"MIENTO DE"); outtextxy(36,233,"VOZ");
696
697     outtextxy(16,410," Marco");    outtextxy(16,420,"Aurelio");
698     outtextxy(12,430," Alzate");   outtextxy(12,440," Monroy");
699     outtextxy(12,470,"Junio/95");
700
701     setcolor(WHITE);
702     setviewport(129,1,638,254,1);      // Ventana del osciloscopio
703   }
704   /*************************************************************************/
705   void grafica(void)
706   // Esta funcion grafica en el osciloscopio la trama de muestras de voz
707   {
708     int i,                     // indice
709         x=0,y;                 // coordenadas
710     double ro;                 // incremento fraccional de x
711
712     clearviewport();           // Borra el osciloscopio
713     moveto(x,255 - *Muestras); // Se ubica en la posicion de la primera muestra
714     ro = 0;                    // y traza lineas hasta la posicion de cada una
715     for(i=1;i<LTrama;i++)      // de las siguientes muestras
716     {
717       ro += 512.0 / (double)LTrama;
718       x = (int)ro;
719       y = 255 - *(Muestras+i);
720       lineto(x,y);
721     }
722   }
723
```

```
/*************************************************************************/

/*************************************************************************/
/*******************       PROGRAMA PRINCIPAL        *********************/
/*************************************************************************/
void main(void)
// Programa principal: Esta permanentemente capturando tramas y presen-
//                     tandolas en la pantalla hasta que se  oprima una
//                     tecla.  Si una trama tiene una energia total su-
//                     perior a 120 unidades por muestra,  se clasifica
//                     segmento vocalizado y  se muestra su espectro de
//                     amplitud en dB.  Durante el calculo del espectro
//                     se hace un analisis lineal predictivo con el que
//                     se alimenta una red neuronal para reconocimiento
//                     de vocales.
{
  const Nivel = 120*LTrama;
  long Energia,                    // Energia de la trama actual
       Eant=0;                     // Energia de la trama anterior
  ft=fopen("polinomx.dat","w");    // Archivo para el entrenamiento
  MallocSB();
  RedNeural();                     // Carga la red neural para reconocimiento
  modografico();                   // Establece el modo grafico
  do
  {
    Energia = captura();           // Captura una trama y calcula su energia
    grafica();                     // La presenta en pantalla
    if(Eant>Nivel)                 // Si la anterior trama supero el nivel
    {                              // de energia, se espera que haya sido
      reconoce();                  // el inicio de un segmento vocalizado
      Energia=0;                   // que se continuo en la actual trama,
    }                              // que es la que se analiza.
    Eant=Energia;
  } while(!kbhit());               // Repite hasta que se oprima una tecla
  closegraph();                    // Retorna al modo de texto
  getch();                         // Retira del buffer la tecla oprimida
  farfree(BloqueD);                // Libera la memoria reservada
  fclose(ft);                      // Cierra el archivo de entranamiento
}
```

```
/*************************************************************************/
/**     ENTRENAMIENTO DE UNA RED NEURAL PARA RECONOCIMIENTO DE FONEMAS     **/
/**                                                                       **/
/**        Marco Aurelio Alzate Monroy. Universidad Distrital F.J.C.      **/
/**        Programa de Maestria en Teleinform tica.    Junio de 1995      **/
/**  Este programa entrena una red neural sencilla para reconocer vocales  **/
/**  a partir de los coeficientes LPC, ParCor,  LAR, LSP e IS,  obtenidos  **/
/**  de un analisis predictivo lineal.  La red forma un perceptron de una  **/
/**  sola capa con cinco elementos  de diez entradas cada uno y los datos  **/
/**  de entrenamiento se obtienen de un archivo grabado previamente.       **/
/*************************************************************************/
// Archivos de encabezado para funciones predefinidas.
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <time.h>
const ng =  12;               // Numero de grupos de vocales en el archivo
/*************************************************************************/
/*******************        Programa Principal       **********************/
/*************************************************************************/
void main(void)
{
  FILE   *fpt;                // Archivo de coeficientes (In) y pesos (Out)
  double x[ng][5][10],        // Los datos de entrenamiento consisten en
                              // NG grupos de cinco vocales, cada una con
                              // diez coeficientes Parametricos
         w[11][5],            // Pesos de las diez entradas de cada una de
                              // las cinco neuronas mas la polarizacion
         t=5.0,               // Ponderaci¢n de las correcciones de los pesos
         e,                   // Numero de errores
         tasa,                // Tasa de aciertos
         y0, y1, y2, y3, y4,  // Salidas deseables de las neuronas
         y0_,y1_,y2_,y3_,y4_; // Salidas reales    de las neuronas
  unsigned int i,j,k,l,m;     // Indices auxiliares
  char   c;                   // Lectura del teclado
  clrscr();
  fpt=fopen("polinomi.dat","r");         // Abre el archivo de coeficientes
  for(i=0;i<ng;i++)                      // Numero de ensayo
    for(j=0;j<5;j++)                     // Numero de vocal
       for(k=0;k<10;k++)                 // Numero de coeficiente
          fscanf(fpt,"%lf",&x[i][j][k]); // Lee cada coeficiente
  fclose(fpt);
  if(!(fpt=fopen("pesosNN.dat","r")))    // Si no existe el archivo de pesos,
    for(i=0;i<11;i++)                    // inicia aleatoriamente sus valores
      for(j=0;j<5;j++)
      w[i][j] = 0.5 - ((double)random(32767))/32766.0;
  else
  {                                      // En otro caso parte de los resul-
    for(i=0;i<11;i++)                    // tados del entrenamiento anterior
      for(j=0;j<5;j++)
        fscanf(fpt,"%lf",&w[i][j]);
    fclose(fpt);
  }
  for(j=0;j<10000;j++)                   // Numero de iteraciones
  {
    e = 0.0;                             // Numero de errores
    gotoxy(1,1); printf("%d : ",9999-j); // Reporta el numero de iteracion
    for(m=0;m<ng;m++)                    // Recorre cada grupo de vocales
```

```
    {
      for(l=0;l<5;l++)                  // Recorre cada vocal del grupo
      {
        if (l==4) y4=1; else y4=0;      // Salidas deseables de las
        if (l==3) y3=1; else y3=0;      // neuronas para esta vocal
        if (l==2) y2=1; else y2=0;
        if (l==1) y1=1; else y1=0;
        if (l==0) y0=1; else y0=0;
        y4_ = y3_ = y2_ = y1_ = y0_ = 0; // Calcula las salidas reales
        for(k=0;k<10;k++)               // Suma los coeficientes ponderados
        {
          y4_ += w[k][4]*x[m][l][k];
          y3_ += w[k][3]*x[m][l][k];
          y2_ += w[k][2]*x[m][l][k];
          y1_ += w[k][1]*x[m][l][k];
          y0_ += w[k][0]*x[m][l][k];
        }
        y4_ -= w[10][4];                // Anade la polarizacion
        y3_ -= w[10][3];
        y2_ -= w[10][2];
        y1_ -= w[10][1];
        y0_ -= w[10][0];
        if(y4_>0.0) y4_ = 1.0; else y4_ = 0.0; // Alinealidad de las neuronas
        if(y3_>0.0) y3_ = 1.0; else y3_ = 0.0;
        if(y2_>0.0) y2_ = 1.0; else y2_ = 0.0;
        if(y1_>0.0) y1_ = 1.0; else y1_ = 0.0;
        if(y0_>0.0) y0_ = 1.0; else y0_ = 0.0; // Verifica si hubo errores
        if((y4!=y4_)||(y3!=y3_)||(y2!=y2_)||(y1!=y1_)||(y0!=y0_)) e += 1.0;
        for(k=0;k<10;k++)               // Ajusta los pesos segun el error
        {
          w[k][4] += (y4 - y4_)*x[m][l][k]/t;
          w[k][3] += (y3 - y3_)*x[m][l][k]/t;
          w[k][2] += (y2 - y2_)*x[m][l][k]/t;
          w[k][1] += (y1 - y1_)*x[m][l][k]/t;
          w[k][0] += (y0 - y0_)*x[m][l][k]/t;
        }
        w[10][4] -= (y4 - y4_)/t;       // Ajusta el peso de la polarizaci¢n
        w[10][3] -= (y3 - y3_)/t;
        w[10][2] -= (y2 - y2_)/t;
        w[10][1] -= (y1 - y1_)/t;
        w[10][0] -= (y0 - y0_)/t;
        t += 2.0e-5;                    // Ajusta el factor de ponderaci¢n
      }
    }
    tasa = ((5.0*ng - e)*100.0)/(5.0*ng);// Tasa de aciertos en el reconcimie.
    printf(" %3.0lf (%5.2lf%)",e,tasa);
    if(tasa>98.5) j=20000;              // Suficiente precision
    if(kbhit()) if(getch()==27) j=20000;
  }
  fpt=fopen("pesosNN.dat","w");         // Genera el archivo de pesos
  for(i=0;i<11;i++)                     // sinapticos para ser usado
     for(j=0;j<5;j++)                   // por el programa de recono-
        fprintf(fpt,"%lf\n",w[i][j]);   // cimiento
    fclose(fpt);
}
```