ESCUELA POLITECNICA NACIONAL

FACULTAD DE INGENIERIA ELECTRICA

DEPARTAMENTO DE ELECTRONICA Y CONTROL

TESIS DE GRADO

PREVIA A LA OBTENCION DEL TITULO DE

INGENIERO EN ELECTRONICA Y CONTROL

IDENTIFICACION DE PARAMETROS UTILIZANDO UN ALGORITMO GENETICO DE OPTIMIZACION, APLICACION AL CIRCUITO EQUIVALENTE DE LA MAQUINA DE INDUCCION.

ANTONIO JAVIER QUINTANA MORENO

MARZO DE 1994

QUITO - ECUADOR

ANEXO C

ANEXO C

LISTADO DE PROGRAMAS.

El programa desarrollado en esta tesis ha sido escrito en el Lenguaje C, utilizando para su compilación la versión 1.01 de Turbo C++ de Borland. El listado completo del programa se ha dividido en 10 módulos, los mismos que se detallan a continuación:

1) **AG.C**, que contiene el programa principal del algoritmo genético.

2) **MENUS.C**, que contiene las funciones que permiten la presentación de los menús de opciones al usuario, así como aquellas que tienen relación con la presentación de las diferentes pantallas de datos del programa.

3) **AGEVOLUC.C**, que contiene la función que realiza el proceso de evolución artificial de la población inicial de parámetros de la máquina para identificar el grupo óptimo.

4) **GENERAC.C**, en el cual se incluyen las funciones desarrolladas para los procesos artificiales de reproducción, cruce y mutación.

5) **EVALUAR.C**, que contiene la función con la cual se evalúa la aptitud de los miembros de las diferentes poblaciones obtenidas durante el proceso de evolución artificial.

6) **COD_DEC.C**, en el cual se incluyen las funciones utilizadas para la codificación y decodificación de los valores de los parámetros de una población.

7) **POBINIC.C**, que contiene la función que permite obtener las poblaciones iniciales de parámetros de la máquina de inducción.

8) **REPORTE.C**, incluye las funciones desarrolladas para la generación de los reportes de resultados al usuario.

9) **GRAFICOS.C**, que contiene las funciones que permite visualizar en pantalla la forma como ha ocurrido la evolución artificial de los parámetros miembros de la población inicial.

10) **AGINCLUD.H**, archivo de encabezado del programa, desarrollado específicamente para esta tesis, en el cual se definen las principales constantes utilizadas en el programa así como las estructuras de datos utilizadas para el proceso de evolución artificial.

A continuación se presenta el contenido de cada uno de los módulos antes indicados.

C.1. MODULO AG.C.

```
/*                                                                      */
/* ESCUELA POLITECNICA NACIONAL.                                        */
/* FACULTAD DE INGENIERIA ELECTRICA.                                    */
/* DEPARTAMENTO DE ELECTRONICA Y CONTROL.                               */
/*                                                                      */
/* TESIS DE GRADO: Identificación de Parámetros utilizando un Algoritmo */
/*                 Genético de Optimización, Aplicación al Circuito     */
/*                 Equivalente de la Máquina de Inducción.              */
/*                                                                      */
/* AUTOR    : A. Javier Quintana M.                                     */
/* DIRECTOR : Dr. Hugo Banda Gamboa                                     */
/*                                                                      */
/* AG.C  : Programa principal desarrollado para realizar el proceso de  */
/*         Evolución Artificial. Para su funcionamiento requiere de siete */
/*         módulos adicionales y un archivo de inclusión especifico para */
/*         esta aplicación, estos son:                                  */
/*         - Reporte.c          - Cod_dec.c          - Graficos.c       */
/*         - Evaluar.c          - Generac.c          - Menus.c          */
/*         - Agevoluc.c         - Aginclud.h         - Pobinic.c        */
/*                                                                      */
/*         Todas las estructuras utilizadas en los diferentes módulos están */
/*         definidas en AGINCLUD.H                                      */
/*                                                                      */
/*         Este programa se desarrolló utilizando el Turbo C++ versión 1.01 */
/*         de BORLAND.                                                  */
/*                                                                      */
/* CREACION : 2 - XII - 1993                                            */

/*
** ARCHIVOS DE INCLUSION.
*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
#include <string.h>
#include "aginclud.h"


/*
** ALUSIONES A FUNCIONES.
*/

/* Funciones definidas en el módulo MENUS.C . */

extern void pantalla_presentacion ( void );
extern void presentar_valores ( void );
extern void menu_datos ( void );
extern void salir ( void );

/* Función definida en este módulo */

void inicializar_variables ( void );
```

```
/*
** DECLARACION DE VARIABLES GLOBALES.
*/

DATOS_ENTRADA valor; /* Estructura definida en AGINCLUD.H, y en la que se */
                     /* almacenan los valores ingresados para inicializar */
                     /* todas las variables empleadas en los módulos del  */
                     /* programa.                                         */


/*
** PROGRAMA PRINCIPAL DEL MECANISMO DE EVOLUCION ARTIFICIAL.
*/

void main ( void )
{
  int c;  /* Variable auxiliar para la lectura desde teclado. */

  /* Visualizar la pantalla de presentación del programa.  */

  pantalla_presentacion();

  /* Lazo de espera para la pulsación de una de las dos teclas ESC o ENTER. */

  while ( 1 )
  {
    c = getch();                    /* Espera que se presione una tecla.    */

    if(ESC == c )
    {                               /* Si la tecla presionada fue ESC:      */
      salir();                      /* entonces salir del programa.         */
    }

    if(ENTER == c )
    {                               /* Si la tecla presionada fue ENTER:    */
      inicializar_variables();      /* inicializar todas las variables, vi- */
      presentar_valores();          /* sualizar dichos valores en pantalla, */
      menu_datos();                 /* y acceder a un primer menú de opcio- */
    }                               /* nes.                                 */

  }
}    /* Fin de Programa Principal */


/*
** CODIGO DE LA FUNCION.
*/


/*                                                                         */
/* inicializar_variables (): Función que inicializa todos los valores de las*/
/*                           variables que se utilizan en los diferentes mó-*/
/*                           dulos.  Estos valores se almacenan en la es-   */
/*                           tructura "valor".                              */

void inicializar_variables ( void )
{
  /*  *****  Inicialización de los valores de la estructura "valor"  *****  */
```

```
  /* Código del método empleado para obtener la población inicial de pará-  */
  /* metros, ya sea el recomendado por la IEEE o el método BASICO.          */
  valor.metodo = M_BASICO;

  /* Número de pruebas realizadas en el laboratorio para la máquina en estu-*/
  /* dio.                                                                   */
  valor.num_pruebas = 100;

  /* Valor de la resistencia de estator, empleada por el método del IEEE.   */
  /* Este valor siempre debe ser ingresado, sin importar el método escogido.*/
  valor.r1 = 0.8936;

  /* Número máximo de generaciones que se pueden llevar a cabo. */
  valor.max_generaciones = 300;

  /* Probabilidades de cruce y mutación. */
  valor.p_cruce    = 1.00;
  valor.p_mutacion = 0.01;

  /* Valores de la prueba de carga.      */
  valor.torque        =   24.0;
  valor.deslizamiento =    0.0833;
  valor.voltaje       =  205.0;
  valor.corriente     =   16.5;
  valor.potencia      = 5300.0;

  /* Rangos de los parámetros del circuito equivalente de la M.I. */
  valor.r2_max  =   5.00;
  valor.r2_min  =   0.01;
  valor.x1_max  =   5.00;
  valor.x1_min  =   0.01;
  valor.xm_max  =  50.00;
  valor.xm_min  =   5.00;
  valor.rfe_max = 300.00;
  valor.rfe_min =  30.00;

  /* Constantes características de la función que evalúa el error. */
  valor.k1 = 0.3333;
  valor.k2 = 0.3333;
  valor.k3 = 0.3333;
  valor.m  = 1;

  /* Inicialización de la generación de números aleatorios. */
  randomize ();
  return;
}       /* Fin de la Función inicializar_variables() */
```

C.2. MODULO MENUS.C.

```
/*                                                                         */
/* ESCUELA POLITECNICA NACIONAL.                                           */
/* FACULTAD DE INGENIERIA ELECTRICA.                                       */
/* DEPARTAMENTO DE ELECTRONICA Y CONTROL.                                  */
/*                                                                         */
```

```
/* TESIS DE GRADO: Identificación de Parámetros utilizando un Algoritmo     */
/*                 Genético de Optimización, Aplicación al Circuito         */
/*                 Equivalente de la Máquina de Inducción.                  */
/*                                                                          */
/* AUTOR    : A. Javier Quintana M.                                         */
/* DIRECTOR : Dr. Hugo Banda Gamboa                                         */
/*                                                                          */
/* MENUS.C  : Módulo que contiene las funciones necesarias para la presen-  */
/*            tación en pantalla de menús, valores de variables y reportes. */
/*                                                                          */
/* CREACION : 2 - XII - 1993                                                */

/*
** ARCHIVOS DE INCLUSION.
*/

#include <conio.h>
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <bios.h>
#include "aginclud.h"


/*
** ALUSIONES A FUNCIONES.
*/

/* Funciones definidas en este módulo. */

void pantalla_presentacion ( void );
void presentar_valores ( void );
void modificar_valores ( void );
void menu_datos ( void );
void ver_resultados ( void );
void menu_resultados ( void );
void ventana_proceso ( int color );
void salir ( void );
void dibujar_caja ( int xmin, int ymin, int xmax, int ymax,
                    int tipo, int color );
int leer_ent ( int valor_ant, int minimo, int maximo, int digitos,
               int xcoord, int ycoord );
float leer_flot ( float valor_ant, float minimo, float maximo, int digitos,
                  int xcoord, int ycoord );

/* Función definida en AGEVOLUC.C . */

extern int evolucion_artificial ( void );

/* Función definida en GRAFICOS.C */

extern void graficar ( void );

/* Función definida en POBINIC.C .  */

extern void menu ( void );
```

```
/*
** CODIGO DE LAS FUNCIONES.
*/


/*                                                                    */
/* pantalla_presentacion () : Función que visualiza la pantalla de presen-  */
/*                            tación en el momento en que se corre el pro-  */
/*                            grama AG.EXE.                           */

void pantalla_presentacion ( void )
{
   int j;        /* Variable auxiliar. */

   /* Define la ventana de visualización como la pantalla completa, y selec-*/
   /* ciona el color del fondo y del texto.                              */
   clrscr(); window(1,1,80,25); textattr ( COLOR_NORMAL ); clrscr();

   /* Dibuja el borde de la ventana principal.   */
   dibujar_caja ( 1,  1, 80, 24, DOBLE, BORDE_NORMAL );

   /* Imprime encabezado.    */
   gotoxy ( 26, 5 ); cprintf("ESCUELA POLITECNICA NACIONAL");
   gotoxy ( 24, 6 ); cprintf("FACULTAD DE INGENIERIA ELECTRICA");
   gotoxy ( 22, 7 ); cprintf("DEPARTAMENTO DE ELECTRONICA Y CONTROL");

   /* Dibuja cuadro en el que se incluye el título de la Tesis de Grado.    */

   dibujar_caja ( 6, 11, 74, 15, DOBLE, BORDE_NORMAL );
   textcolor ( YELLOW ); gotoxy ( 32, 11 ); cprintf(" TESIS DE GRADO ");

   textcolor ( WHITE );  /* Selecciona el color del texto. */

   /* Imprime el título de la Tesis de Grado.     */
   gotoxy ( 10, 12 );
   cprintf("Identificación de Parámetros utilizando un Algoritmo Genético");
   gotoxy ( 10, 13 );
   cprintf("de Optimización, Aplicación al Circuito Equivalente de la");
   gotoxy ( 10, 14 );
   cprintf("Máquina de Inducción.");

   /* Imprime en pantalla los nombres del autor y del director de la Tesis. */
   gotoxy ( 25, 18 ); cprintf("AUTOR    : A. JAVIER QUINTANA M.");
   gotoxy ( 25, 19 ); cprintf("DIRECTOR: Dr. HUGO BANDA GAMBOA");
   gotoxy ( 33, 22 ); cprintf("Febrero de 1994");

   /* Presenta en pantalla las opciones de que dispone en este punto.       */
   window(1,25,80,25);
   textattr(COLOR_MENU);
   gotoxy ( 1,1 ); for( j = 1; j < 81; j++ ) putch(' ');
   gotoxy ( 10, 1 ); cprintf(" ESC");
   gotoxy ( 55, 1 ); cprintf(" ENTER");
   textcolor(BLACK);
   gotoxy ( 15, 1 ); cprintf("Salir");
   gotoxy ( 62, 1 ); cprintf("Continuar");
   _setcursortype( _NOCURSOR );    /* Elimina el cursor. */
   return;
}        /* Fin de la Función pantalla_presentación () . */
```

```
/*                                                                    */
/* presentar_valores () :  Función que visualiza en pantalla los valores de */
/*                         las variables que definen las condiciones bajo   */
/*                         las cuales se va a desarrollar el proceso de e-  */
/*                         volución artificial.                             */

void presentar_valores ( void )
{
  /* Declaración de variables. */

  extern DATOS_ENTRADA valor;  /* Estructura declarada en AG.C . */
  int j;                       /* Variable auxiliar.             */

  /* Define la ventana de visualización con sus respectivos atributos.      */
  clrscr(); window(1,1,80,25); textattr ( COLOR_NORMAL ); clrscr();

  /* Dibuja el marco de la ventana principal.    */
  dibujar_caja ( 1,1,80,23,DOBLE,BORDE_NORMAL );

  /* Dibuja los marcos internos de la ventana principal. */
  dibujar_caja (  4, 5,47, 8,SIMPLE,BORDE_NORMAL );
  dibujar_caja (  4, 9,47,13,SIMPLE,BORDE_NORMAL );
  dibujar_caja (  4,14,47,21,SIMPLE,BORDE_NORMAL );
  dibujar_caja ( 50, 5,76,15,SIMPLE,BORDE_NORMAL );
  dibujar_caja ( 50,16,76,22,SIMPLE,BORDE_NORMAL );

  /* Imprime los títulos de todas las cajas.  */
  textcolor ( YELLOW );
  gotoxy ( 25, 2 ); cprintf("PROCESO DE EVOLUCION ARTIFICIAL");
  gotoxy ( 18,15 ); cprintf("Prueba de Carga");
  gotoxy ( 53, 6 ); cprintf("Rangos de Parámetros");
  gotoxy ( 55,17 ); cprintf("Función de Error");
  gotoxy ( 22, 6 ); cprintf("B");
  gotoxy ( 33, 6 ); cprintf("I");

  textcolor( WHITE );
  gotoxy ( 32, 3 ); cprintf("Datos de Entrada");
  gotoxy (  6, 6 ); cprintf("METODO : ");
  gotoxy ( 23, 6 ); cprintf("ASICO");
  gotoxy ( 34, 6 ); cprintf("EEE");
  gotoxy (  6, 7 ); cprintf("                         ");

  /* Si el método escogido es el BASICO imprime una B. */

  if ( M_BASICO == valor.metodo )
  {
    gotoxy (  6, 7 ); cprintf("                         ");
    gotoxy ( 15, 6 ); cprintf("B");
  }

  /* Si el método escogido es el IEEE imprime una I, y el valor de R1. */
  if ( M_IEEE == valor.metodo )
  {
    gotoxy ( 15, 6 ); cprintf("I");
    gotoxy (  6, 7 ); cprintf("r1 = %6.4f ohmios", valor.r1 );
  }
```

```
/* Imprime en pantalla las condiciones del proceso de evolución artificial*/

gotoxy (  6,10 ); cprintf("Número de generaciones   : ");
gotoxy ( 33,10 ); cprintf("%d", valor.max_generaciones );

gotoxy (  6,11 ); cprintf("Probabilidad de Mutación : ");
gotoxy ( 33,11 ); cprintf("%5.3f", valor.p_mutacion );

gotoxy (  6,12 ); cprintf("Probabilidad de Cruce    : ");
gotoxy ( 33,12 ); cprintf("%5.3f", valor.p_cruce );

/* Visualiza los valores de la prueba de carga. */

gotoxy (  6,16 ); cprintf("Deslizamiento      : ");
gotoxy ( 31,16 ); cprintf("%6.4f", valor.deslizamiento );
gotoxy ( 44,16 ); cprintf("pu");

gotoxy (  6,17 ); cprintf("Torque en el eje   : ");
gotoxy ( 30,17 ); cprintf("%5.2f", valor.torque );
gotoxy ( 44,17 ); cprintf("Nm");

gotoxy (  6,18 ); cprintf("Voltaje de Fase    : ");
gotoxy ( 29,18 ); cprintf("%6.2f", valor.voltaje );
gotoxy ( 45,18 ); cprintf("V");

gotoxy (  6,19 ); cprintf("Corriente de Fase  : ");
gotoxy ( 30,19 ); cprintf("%5.2f", valor.corriente );
gotoxy ( 45,19 ); cprintf("A");

gotoxy (  6,20 ); cprintf("Potencia Activa 3 Φ: ");
gotoxy ( 28,20 ); cprintf("%6.1f", valor.potencia );
gotoxy ( 45,20 ); cprintf("W");

/* Visualiza los rangos permitidos de variación de los parámetros.        */

gotoxy ( 52, 7 ); cprintf("R2 max  = ");
gotoxy ( 63, 7 ); cprintf("%5.2f", valor.r2_max );

gotoxy ( 52, 8 ); cprintf("R2 min  = ");
gotoxy ( 63, 8 ); cprintf("%5.2f", valor.r2_min );

gotoxy ( 52, 9 ); cprintf("X1 max  = ");
gotoxy ( 63, 9 ); cprintf("%5.2f", valor.x1_max );

gotoxy ( 52,10 ); cprintf("X1 min  = ");
gotoxy ( 63,10 ); cprintf("%5.2f", valor.x1_min );

gotoxy ( 52,11 ); cprintf("Xm max  = ");
gotoxy ( 63,11 ); cprintf("%5.2f", valor.xm_max );

gotoxy ( 52,12 ); cprintf("Xm min  = ");
gotoxy ( 63,12 ); cprintf("%5.2f", valor.xm_min );

gotoxy ( 52,13 ); cprintf("Rfe max = ");
gotoxy ( 62,13 ); cprintf("%6.2f", valor.rfe_max );

gotoxy ( 52,14 ); cprintf("Rfe min = ");
gotoxy ( 62,14 ); cprintf("%6.2f", valor.rfe_min );
```

```
  for ( j = 7; j < 15; j++ )
  {
    gotoxy( 72, j ); cprintf("ohm");
  }

  /* Visualiza las constantes características de la función de error.       */

  gotoxy ( 52,18 ); cprintf ("K1 = ");
  gotoxy ( 57,18 ); cprintf ("%6.4f", valor.k1 );

  gotoxy ( 52,19 ); cprintf ("K2 = ");
  gotoxy ( 57,19 ); cprintf ("%6.4f", valor.k2 );

  gotoxy ( 52,20 ); cprintf ("K3 = ");
  gotoxy ( 57,20 ); cprintf ("%6.4f", valor.k3 );

  gotoxy ( 52,21 ); cprintf ("m  = ");
  gotoxy ( 57,21 ); cprintf ("%d", valor.m );

  return;
}       /* Fin de la Función presentar_valores (). */


/*                                                                          */
/* menu_datos () : Visualiza el menú de opciones disponibles una vez que se */
/*                 han presentado en pantalla los valores de las variables  */
/*                 que definen las condiciones del proceso de E.A.          */

void menu_datos ( void )
{
   /* Declaración de variables. */

   extern DATOS_ENTRADA valor;  /* Estructura declarada en AG.C.      */

   char pantalla_datos [4096];  /* Buffer para guardar una pantalla.  */

   /* Arreglo en que se definen las opciones del menú. */
   char opcion[3][12] = { " Continuar ",
                          " Modificar ",
                          " Población " };

   /* Arreglo en que se definen los mensajes respectivos de cada opción. */
   char mensaje[3][45] = {" Iniciar el proceso de Evolución Artificial.",
                          " Cambiar los valores indicados en pantalla. ",
                          " Obtener la población inicial de parámetros." };

   int j;              /* Variable auxiliar. */
   int eleccion = 0;   /* Puntero de la opción seleccionada. */
   int xi = 5;         /* Posición en X de la primera opción del menú. */
   char c;             /* Variable auxiliar para leer un caracter del teclado*/
   int error_referencia = 0;

   /* Visualiza la barra del menú con sus opciones.  */

   textattr(COLOR_MENU);
   gotoxy ( 1,24 );
   for( j = 1; j < 81; j++ ) putch(' ');
```

```
gotoxy ( 28, 24 );
cprintf(opcion[1]);

gotoxy ( 51, 24 );
cprintf(opcion[2]);

textcolor(RED);
gotoxy ( 69,24 ); cprintf("ESC");
textcolor(BLACK);
gotoxy ( 66,24 ); cprintf("|");
gotoxy ( 73,24 ); cprintf("Salir");

textattr(REVERSO_MENU);
gotoxy (  5,24 );
cprintf(opcion[0]);

textattr(COLOR_NORMAL);
gotoxy (  1, 25 );
cprintf(mensaje[0]);

/* Lazo de selección de una de las opciones disponibles. */

while(1)
{
  c = getch();         /* Lee un caracter del teclado. */

  /* Se permite el movimiento de una barra de selección con el TAB, a    */
  /* través de la barra del menú.                                        */
  if ( TAB == c )
  {
    ++ eleccion;
    gotoxy(xi,24);
    textattr(COLOR_MENU);
    cprintf(opcion[eleccion-1]);

    if(eleccion > 2)
    {
       eleccion = 0;
       xi  = 5;
    }
    else  xi += 23;

    gotoxy( xi,24 );
    textattr(REVERSO_MENU);
    cprintf(opcion[eleccion]);
    gotoxy(  1,25 ); textattr(COLOR_NORMAL); cprintf(mensaje[eleccion]);

  }


  /* Si una vez posicionado en la opción deseada, se presiona ENTER, se  */
  /* ejecuta(n) la(s) accion(es) respectiva(s).                          */
  if ( ENTER == c )
  {
    /* Opcion Población */
    if ( eleccion == 2 )
    {
       gettext(1,1,80,25,pantalla_datos);  /* Guarda la pantalla actual. */
```

```
            menu();
            clrscr(); window(1,1,80,25); textattr ( COLOR_NORMAL ); clrscr();
            puttext(1,1,80,25,pantalla_datos);  * Presenta pantalla original. */

            /* Si el método escogido es el BASICO imprime una B. */
            if ( M_BASICO == valor.metodo )
            {
              gotoxy (  6, 7 ); cprintf("                          ");
              gotoxy ( 15, 6 ); cprintf("B");
            }

            /* Si el método escogido es el IEEE imprime una I,y el valor de R1*/
            if ( M_IEEE == valor.metodo )
            {
              gotoxy ( 15, 6 ); cprintf("I");
              gotoxy (  6, 7 ); cprintf("r1 = %6.4f ohmios", valor.r1 );
            }

          }

          /* Opción Modificar. */
          if (eleccion == 1) modificar_valores();

          /* Opción Continuar. */
          if (eleccion == 0)
          {
            gettext(1,1,80,25,pantalla_datos);  /* Guarda la pantalla actual. */
            ventana_proceso( COLOR_PROCESO );
            error_referencia = evolucion_artificial();
            if ( error_referencia == 0 )
              ver_resultados();
            clrscr(); window(1,1,80,25); textattr ( COLOR_NORMAL ); clrscr();
            puttext(1,1,80,25,pantalla_datos);  /* Presenta pantalla original.*/
          }
        }

        if ( ESC == c )
          salir();
      }

} /* Fin de Función menu_datos (). */



/*                                                                        */
/* modificar_valores () : Función que permite la modificación de los valores*/
/*                        de las variables que definen las condiciones bajo */
/*                        cuales se va a realizar el proceso de E.A.        */


void modificar_valores ( void )
{
  /* Declaración de variables. */

  extern DATOS_ENTRADA valor;  /* Estructura declarada en AG.C . */
  char c ;             /* Variable auxiliar para leer un caracter del teclado.*/
  int flag = 0;        /* Bandera que permite modificar todos los valores.    */
```

```
int flag1 = 0;      /* Bandera que permite seleccionar el método empleado, */
int flag2 = 0;      /* y aceptar las modificaciones realizadas.            */

_setcursortype( _NORMALCURSOR );  /* Visualiza el cursor. */

/* Lazo principal para la modificación de los valores. */
do
{
  /* Lazo secundario para seleccionar el método a utilizarse. */
  do
  {
    flag2 = 0;
    do
    {
       fflush ( stdin );
       gotoxy ( 15, 6 ); c = getch();
       if ( isalpha(c) )   flag2 = 1;
       if ( ENTER == c )   flag2 = 1;
    } while ( !flag2 );

    if ( ENTER == c )
    {
       if ( valor.metodo == M_BASICO )
       {
         valor.metodo = M_BASICO;
         gotoxy ( 15, 6 ); cprintf("B");
         gotoxy (  6, 7 ); cprintf("                          ");
         flag1 = 1;
       }

      if ( valor.metodo == M_IEEE )
       {
         valor.metodo = M_IEEE;
         gotoxy ( 15, 6 ); cprintf("I");
         flag1 = 1;
       }
    }

    if ( isalpha(c) )
    {
       if ( islower ( c ) )  c = toupper ( c );
       gotoxy ( 15, 6 ); putch(c);
       fflush ( stdin );
       if ( c == 'B')
       {
         valor.metodo = M_BASICO;
         gotoxy ( 15, 6 ); cprintf("B");
         gotoxy (  6, 7 ); cprintf("                          ");
         flag1 = 1;
       }
       if ( c == 'I')
       {
         valor.metodo = M_IEEE;
         gotoxy ( 15, 6 ); cprintf("I");
         flag1 = 1;
       }
    }
  } while ( flag1 == 0 );
```

```
/* Lectura de todas las variables involucradas en el proceso de E.A.    */

if ( valor.metodo == M_IEEE )
{
  gotoxy (  6, 7 ); cprintf("r1 = %6.4f ohmios", valor.r1 );
  valor.r1 = leer_flot(valor.r1,0.01,9.9,6,11,7);
  gotoxy ( 11, 7 ); cprintf("%6.4f",valor.r1);
}

valor.max_generaciones = leer_ent(valor.max_generaciones,1,300,3,33,10);
gotoxy ( 33,10 ); cprintf("%3d", valor.max_generaciones);

valor.p_mutacion = leer_flot(valor.p_mutacion,0.0,1.0,5,33,11);
gotoxy ( 33,11 ); cprintf("%5.3f", valor.p_mutacion);

valor.p_cruce = leer_flot(valor.p_cruce,0.0,1.0,5,33,12);
gotoxy ( 33,12 ); cprintf("%5.3f", valor.p_cruce);

valor.deslizamiento = leer_flot(valor.deslizamiento,0.001,1.0,6,31,16);
gotoxy ( 31,16 ); cprintf("%6.4f", valor.deslizamiento);

valor.torque = leer_flot(valor.torque,1.0,99.99,5,30,17);
gotoxy ( 30,17 ); cprintf("%5.2f", valor.torque);

valor.voltaje = leer_flot(valor.voltaje,100.0,300.0,6,29,18);
gotoxy ( 29,18 ); cprintf("%6.2f", valor.voltaje);

valor.corriente = leer_flot(valor.corriente,1.0,99.99,5,30,19);
gotoxy ( 30,19 ); cprintf("%5.2f", valor.corriente);

valor.potencia = leer_flot(valor.potencia,100.0,9999.9,6,28,20);
gotoxy ( 28,20 ); cprintf("%6.1f", valor.potencia);

valor.r2_max = leer_flot(valor.r2_max,0.01,9.99,4,64,7);
gotoxy ( 64, 7 ); cprintf("%4.2f", valor.r2_max);

valor.r2_min = leer_flot(valor.r2_min,0.01,valor.r2_max,4,64,8);
gotoxy ( 64, 8 ); cprintf("%4.2f", valor.r2_min);

valor.x1_max = leer_flot(valor.x1_max,0.01,9.99,4,64,9);
gotoxy ( 64, 9 ); cprintf("%4.2f", valor.x1_max);

valor.x1_min = leer_flot(valor.x1_min,0.01,valor.x1_max,4,64,10);
gotoxy ( 64,10 ); cprintf("%4.2f", valor.x1_min);

valor.xm_max = leer_flot(valor.xm_max,0.01,99.99,5,63,11);
gotoxy ( 63,11 ); cprintf("%5.2f", valor.xm_max);

valor.xm_min = leer_flot(valor.xm_min,0.01,valor.xm_max,5,63,12);
gotoxy ( 63,12 ); cprintf("%5.2f", valor.xm_min);

valor.rfe_max = leer_flot(valor.rfe_max,10.0,999.99,6,62,13);
gotoxy ( 62,13 ); cprintf("%6.2f", valor.rfe_max);

valor.rfe_min = leer_flot(valor.rfe_min,1.0,valor.rfe_max,6,62,14);
gotoxy ( 62,14 ); cprintf("%6.2f", valor.rfe_min);
```

```
    valor.k1 = leer_flot(valor.k1,0.01,1.0,6,57,18);
    gotoxy ( 57,18 ); cprintf("%.4f", valor.k1);

    valor.k2 = leer_flot(valor.k2,0.01,1.0,6,57,19);
    gotoxy ( 57,19 ); cprintf("%.4f", valor.k2);

    valor.k3 = leer_flot(valor.k3,0.01,1.0,6,57,20);
    gotoxy ( 57,20 ); cprintf("%.4f", valor.k3);

    valor.m = leer_ent(valor.m,1,3,1,57,21);
    gotoxy ( 57,21 ); cprintf("%d", valor.m);

    flag1 = 0;



    /* Lazo secundario para aceptar las modificaciones realizadas. */

    do
    {
      gotoxy ( 10,22 ); cprintf("Aceptar ( S/N ) ? ");

      do
      {
         fflush ( stdin );
         gotoxy ( 28,22 ); c = getch();
      } while ( !isalpha(c) );

      if ( islower ( c ) )  c = toupper ( c );
      gotoxy ( 28,22 ); putch(c);
      fflush ( stdin );

      if ( c == 'S')
      {
         gotoxy ( 10,22 ); cprintf("                          ");
         flag1 = 1;
         flag = 1;
      }

      if ( c == 'N')
      {
         gotoxy ( 10,22 ); cprintf("                         ");
         flag1 = 1;
         flag = 0;
      }

    } while ( flag1 == 0 );

  } while ( flag == 0 );


  _setcursortype( _NOCURSOR );   /* Oculta el cursor. */

  return;
}       /* Fin de la Función modificar_valores () . */
```

```
/*                                                                    */
/* ver_resultados () : Función que visualiza las condiciones de la población*/
/*                     inicial y de la población resultante del proceso de  */
/*                     evolución artificial.                          */

void ver_resultados ( void )
{
  extern RESULTADOS resultado [MAX_GEN];/*Estructura declarada en REPORTES.C*/
  extern DATOS_ENTRADA valor;          /* Estructura declarada en AG.C .   */
  int j;                               /* Variable auxiliar.               */
  int max = valor.ultima_generacion;   /* Número de generaciones.          */

  /* Define ventana de visualización de resultados.  */
  clrscr(); window(1,1,80,25); textattr ( COLOR_NORMAL ); clrscr();

  /* Dibuja el borde principal de la ventana de visualización. */
  dibujar_caja ( 4, 3, 76, 21, SIMPLE, BORDE_NORMAL );

  /* Imprime el marco de la tabla de resultados. */

  textcolor(YELLOW);
  gotoxy (  5, 5 );

  for ( j = 5; j < 76; j++ ) cprintf("─");

  for ( j = 4; j < 21; j++ )
  {
    gotoxy ( 37, j ); cprintf("│");
  }
  for ( j = 4; j < 21; j++ )
  {
    gotoxy ( 57, j ); cprintf("│");
  }

  gotoxy (  4, 5 ); putchar('├');
  gotoxy ( 37, 5 ); putchar('┼');
  gotoxy ( 57, 5 ); putchar('┼');
  gotoxy ( 76, 5 ); putchar('┤');
  gotoxy ( 37, 3 ); putchar('┬');
  gotoxy ( 57, 3 ); putchar('┬');
  gotoxy ( 37,21 ); putchar('┴');
  gotoxy ( 57,21 ); putchar('┴');

  /* Imprime el título general de la pantalla. */
  textcolor(YELLOW);
  gotoxy ( 17, 2 ); cprintf("RESULTADOS DEL PROCESO DE EVOLUCION ARTIFICIAL");

  /* Imprime los títulos de las columnas de la tabla. */
  textcolor(WHITE);
  gotoxy ( 16, 4 ); cprintf("PARAMETROS");
  gotoxy ( 40, 4 ); cprintf("COND. INICIALES");
  gotoxy ( 62, 4 ); cprintf("RESULTADOS");

  /* Visualiza los valores correspondientes a cada columna.  */

  gotoxy (  6, 6 ); cprintf("Aptitud Mínima");
  gotoxy ( 43, 6 ); cprintf("%g", resultado[0].aptitud.minimo);
  gotoxy ( 62, 6 ); cprintf("%g", resultado[max].aptitud.minimo);
```

```
gotoxy (  6, 7 ); cprintf("Aptitud Promedio");
gotoxy ( 43, 7 ); cprintf("%g", resultado[0].aptitud.promedio);
gotoxy ( 62, 7 ); cprintf("%g", resultado[max].aptitud.promedio);

gotoxy (  6, 8 ); cprintf("Aptitud Máxima");
gotoxy ( 43, 8 ); cprintf("%g", resultado[0].aptitud.maximo);
gotoxy ( 62, 8 ); cprintf("%g", resultado[max].aptitud.maximo);

gotoxy (  6, 9 ); cprintf("Suma de Aptitudes");
gotoxy ( 43, 9 ); cprintf("%g", resultado[0].aptitud.sumatorio);
gotoxy ( 62, 9 ); cprintf("%g", resultado[max].aptitud.sumatorio);

gotoxy (  6,10 ); cprintf("Error Mínimo");
gotoxy ( 43,10 ); cprintf("%g", resultado[0].error_minimo);
gotoxy ( 62,10 ); cprintf("%g", resultado[max].error_minimo);

gotoxy (  6,11 ); cprintf("R1                    [ohmios]");
gotoxy ( 42,11 ); cprintf("%7.4f", resultado[0].parametro.r1);
gotoxy ( 61,11 ); cprintf("%7.4f", resultado[max].parametro.r1);

gotoxy (  6,12 ); cprintf("R2                    [ohmios]");
gotoxy ( 42,12 ); cprintf("%7.4f", resultado[0].parametro.r2);
gotoxy ( 61,12 ); cprintf("%7.4f", resultado[max].parametro.r2);

gotoxy (  6,13 ); cprintf("X1 = X2               [ohmios]");
gotoxy ( 42,13 ); cprintf("%7.4f", resultado[0].parametro.x1);
gotoxy ( 61,13 ); cprintf("%7.4f", resultado[max].parametro.x1);

gotoxy (  6,14 ); cprintf("Xm                    [ohmios]");
gotoxy ( 42,14 ); cprintf("%7.4f", resultado[0].parametro.xm);
gotoxy ( 61,14 ); cprintf("%7.4f", resultado[max].parametro.xm);

gotoxy (  6,15 ); cprintf("Rfe                   [ohmios]");
gotoxy ( 41,15 ); cprintf("%6.2f", resultado[0].parametro.rfe);
gotoxy ( 60,15 ); cprintf("%6.2f", resultado[max].parametro.rfe);

gotoxy (  6,16 ); cprintf("Impedancia Real       [ohmios]");
gotoxy ( 41,16 ); cprintf("%8.4f", resultado[0].impedancia.real);
gotoxy ( 60,16 ); cprintf("%8.4f", resultado[max].impedancia.real);

gotoxy (  6,17 ); cprintf("Impedancia Imaginaria [ohmios]");
gotoxy ( 41,17 ); cprintf("%8.4f", resultado[0].impedancia.imag);
gotoxy ( 60,17 ); cprintf("%8.4f", resultado[max].impedancia.imag);

gotoxy (  6,18 ); cprintf("Torque en el eje      [N-m]");
gotoxy ( 41,18 ); cprintf("%8.4f", resultado[0].torque);
gotoxy ( 60,18 ); cprintf("%8.4f", resultado[max].torque);

gotoxy (  6,19 ); cprintf("Número de Mutaciones");
gotoxy ( 43,19 ); cprintf("%ld", resultado[0].n_mutaciones);
gotoxy ( 62,19 ); cprintf("%ld", resultado[max].n_mutaciones);

gotoxy (  6,20 ); cprintf("Número de Cruces");
gotoxy ( 43,20 ); cprintf("%ld", resultado[0].n_cruces);
gotoxy ( 62,20 ); cprintf("%ld", resultado[max].n_cruces);

gotoxy ( 44,22 ); cprintf("Z Real medida [ohmios] = ");
gotoxy ( 69,22 ); cprintf("%7.3f", valor.z_real);
```

```
  gotoxy ( 44,23 ); cprintf("Z Imag medida [ohmios] = ");
  gotoxy ( 69,23 ); cprintf("%7.3f", valor.z_imag);

  gotoxy (  5,23 ); cprintf("Torque medido [N-m]    = ");
  gotoxy ( 30,23 ); cprintf("%7.3f", valor.torque);

  textcolor ( YELLOW );
  gotoxy ( 5,22 ); cprintf("Resultados esperados : ");
  textcolor ( WHITE );

  /* Activa un menú de opciones disponibles en este punto. */
  menu_resultados();
  return;
}       /* Fin de la Función ver_resultados () . */



/*                                                                    */
/* menu_resultados () : Función que define otras opciones para visualizar  */
/*                      los resultados obtenidos después de la E.A..  */

void menu_resultados ( void )
{
  /* Declaración de variables. */

  char pantalla_resultados [4096]; /* Buffer para almacenar la pantalla. */

  /* Opciones del menú. */
  char opcion[2][11] = { " Imprimir ",
                         " Graficar " };

  /* Mensajes correspondientes a cada una de las opciones del menú. */
  char mensaje[2][58] = {
           " Imprimir el reporte indicado en pantalla.              ",
           " Presentar gráficos del proceso de Evolución Artificial. " };

  int j;              /* Variable auxiliar. */
  int eleccion = 0;   /* Puntero de selección de una opción del menú. */
  int xi = 10;        /* Posición en X, de la primera opción del menú.*/
  char c;             /* Variable auxiliar que lee un caracter del teclado. */
  int estado;         /* Estado de la impresora. */

  /* Visualiza la barra del menú, con sus opciones. */

  textattr(COLOR_MENU);
  gotoxy ( 1,24 );
  for( j = 1; j < 81; j++ ) putch(' ');

  gotoxy ( 37, 24 );
  cprintf(opcion[1]);

  textcolor(RED);
  gotoxy ( 64,24 ); cprintf("ESC");
  textcolor(BLACK);
  gotoxy ( 68,24 ); cprintf("Menú Previo");
  gotoxy ( 62,24 ); cprintf("|");
```

```
textattr(REVERSO_MENU);
gotoxy ( 10, 24 );
cprintf(opcion[0]);

textattr(COLOR_NORMAL);
gotoxy (  1, 25 );
cprintf(mensaje[0]);


/* Lazo de selección de una opción del menú. */

while(1)
{
  c = getch();  /* Lee un caracter del teclado. */

  /* La tecla TAB permite el movimiento de una barra de selección a traves*/
  /* de la barra del menú.                                                */

  if ( TAB == c )
  {
    ++ eleccion;
    gotoxy(xi,24);
    textattr(COLOR_MENU);
    cprintf(opcion[eleccion-1]);

    if ( eleccion > 1 )
    {
       eleccion = 0;
       xi  = 10;
    }
    else  xi += 27;

    gotoxy( xi,24 );
    textattr(REVERSO_MENU);
    cprintf(opcion[eleccion]);
    gotoxy(  1,25 ); textattr(COLOR_NORMAL);
    cprintf(mensaje[eleccion]);
  }


  /* Una vez escogida la opción, si se aplica la tecla ENTER, se ejecuta  */
  /* el proceso correspondiente.                                          */

  if ( ENTER == c )
  {
    /* Opción Graficar. */
    if ( eleccion == 1 )
    {
       _setcursortype ( _NORMALCURSOR ); /* Presenta cursor. */
       gettext( 1,1, 80, 25, pantalla_resultados );/* Guarda pantalla. */
       /* Grafica en pantalla los parámetros y su evolución. */
       graficar();
       clrscr(); window(1,1,80,25); textattr ( COLOR_NORMAL ); clrscr();
       puttext(1,1,80,25,pantalla_resultados); /* Recupera pantalla. */
       gotoxy ( 64,24 );
       _setcursortype ( _NOCURSOR );     /* Oculta cursor. */
    }
```

```
      /* Opción Imprimir. */
      if ( eleccion == 0 )
      {
         gettext(1,1,80,25,pantalla_resultados); /* Guarda pantalla. */
         ventana_proceso( COLOR_PROCESO);

         gotoxy ( 2,2 ); putch ('\a');
         cprintf("Prepare la impresora y pulse una tecla para continuar.");
         getch();
         estado = biosprint(2,0,0);

         if ( estado & 0x08 )
         {
           putch('\a');
           gotoxy ( 2, 2 );
           cprintf("Prepare la impresora y pulse una tecla para continuar.");
           getch();
         }

         gotoxy ( 2, 3 ); cprintf ( "Imprimiendo reporte...." );
         system ( "type agrepor1.prn > prn " );
         putch('\a');
         gotoxy ( 2, 4 ); cprintf ( "Presione una tecla para continuar...." );
         getch();
         clrscr(); window(1,1,80,25); textattr ( COLOR_NORMAL ); clrscr();
         gotoxy ( 10,24 );
         puttext(1,1,80,25,pantalla_resultados); /* Recupera pantalla. */
      }
    }
    if ( ESC == c )
      return;
  }
}  /* Fin de la Función menu_resultados () . */


/*                                                                    */
/* salir () : Función empleada para terminar el programa de una manera nor- */
/*            mal.                                                    */

void salir ( void )
{
   clrscr();                        /* Borra pantalla . */
   window(1,1,80,25);               /* Define ventana de visualización. */
   _setcursortype( _NORMALCURSOR );/* Presenta cursor. */
   textattr ( DOS_COLOR );          /* Define colores normales del DOS. */
   clrscr();                        /* Borra la pantalla. */
   exit(1);                         /* Termina la ejecución del programa. */
}          /* Fin de la Función salir () . */


/*                                                                    */
/* ventana_proceso () : Función que visualiza en pantalla una ventana en la */
/*                      cual se imprimem mensajes de un proceso determinado.*/

void ventana_proceso ( int color )
{
   dibujar_caja ( 12, 10, 68, 16, DOBLE, color );
   window ( 13,11,67,15 ); textattr ( color ); clrscr();
```

```
   gotoxy( 12, 1 ); cprintf ("PROCESO DE EVOLUCION ARTIFICIAL.");
   return;
}        /* Fin de la Función ventana_proceso () . */



/*                                                                    */
/* dibujar_caja () : Dibuja un marco en una región determinada.       */
/*                   Parámetros:                                      */
/*                 - xmin,ymin --> Esquina superior izquierda del marco.  */
/*                 - xmax,ymax --> Esquina inferior derecha del marco.    */
/*                 - tipo      --> Tipo de línea del marco.           */
/*                 - colores   --> Color de las líneas del marco.     */

void dibujar_caja ( int xmin, int ymin, int xmax, int ymax,
                    int tipo, int colores )
{
  /* Declaración de variables. */

  /* Arreglo en el cual se almacenan los caracteres gráficos necesarios para*/
  /* construir el marco.                                                */

  static char car_caja [2][6] = {
     { '┌', '─', '┐', '│', '└', '┘' },
     { '╒', '═', '╕', '║', '╚', '╝' }  };


  int i;             /* Variable auxiliar. */
  int oldatrib;      /* Atributo anterior de la pantalla. */
  char hcar, vcar;   /* Caracter horizontal y vertical a imprimirse. */
  TEXEL t;           /* Estructura en la cual se almacenan las caracterís-*/
                     /* de una sola celda de texto de la pantalla. Defini-*/
                     /* en AGINCLUD.H .                                  */
  struct text_info ti; /* Estructura definida en GRAPHICS.H, en la cual se */
                       /* guardan las características de la ventana actual.*/


  /* Dibuja el tipo de marco especificado, a menos que no se desee un marco.*/

  if ( tipo )
  {
    gettextinfo ( &ti );      /* Lee y guarda el atributo anterior de la ven-*/
    oldatrib = ti.attribute; /* tana de visualización.                    */

    textattr ( colores );     /* Define el color del marco. */

    hcar = car_caja[tipo-1][1];   /* Escoge el caracter horizontal. */
    vcar = car_caja[tipo-1][3];   /* Escoge el caracter vertical.   */

    /* Dibuja las líneas horizontales. */

    gotoxy ( xmin+1, ymin );
    for( i = xmin + 1; i < xmax; i++ )  putch(hcar);
    gotoxy ( xmin+1, ymax );
    for( i = xmin + 1; i < xmax; i++ )  putch(hcar);

    /* Dibuja las líneas verticales. */
```

```
    for( i = ymin + 1; i < ymax; i++ )
    {
      gotoxy ( xmin, i ); putch(vcar);
      gotoxy ( xmax, i ); putch(vcar);
    }
    /* Dibuja las esquinas del marco. */
    gotoxy ( xmin, ymin ); putch( car_caja[tipo-1][0] );
    gotoxy ( xmax, ymin ); putch( car_caja[tipo-1][2] );
    gotoxy ( xmin, ymax ); putch( car_caja[tipo-1][4] );
    /* Dibuja la esquina inferior derecha del marco. */
    gettext(xmax,ymax,xmax,ymax,&t);
    t.ch = car_caja[tipo-1][5]; t.attr = colores;
    puttext(xmax,ymax,xmax,ymax,&t);
    textattr(oldatrib);  /* Devuelve a la pantalla el atributo anterior. */
  }
  return;
}         /* Fin de la Función dibujar_marco () . */


/*                                                                        */
/* leer_flot () : Función utilizada para leer un valor en punto flotante, */
/*                dentro de cierto rango predeterminado.                  */
/*                Parámetros:                                             */
/*              - valor_ant     --> Valor anterior de la variable.        */
/*              - minimo,maximo --> Rango permitido para la variable.     */
/*              - digitos       --> Número de dígitos a leerse.           */
/*              - xcoord,ycoord --> Coordenadas del punto de lectura.     */
/*                Si se presiona ENTER se mantiene el valor anterior.     */

float leer_flot ( float valor_ant, float minimo, float maximo, int digitos,
                  int xcoord, int ycoord )
{
  /* Declaración de variables. */

  char pantalla[1024];     /* Buffer que almacena una parte de la pantalla. */
  int j, flag, flag1;      /* Variables auxiliares.                         */
  char valor[10];          /* Arreglo que guarda los dígitos leídos.        */
  char c;                  /* Variable auxiliar para lectura de un caracter.*/
  char *puntero;           /* Puntero al arreglo " valor ".                 */
  float nuevo_valor;       /* Nuevo valor de la variable.                   */

  puntero = &valor[0];     /* Inicialización del puntero al arreglo "valor".*/

  /* Lazo principal de lectura de la variable en forma de string. */
  do
  {
    flag = 1;  /* Bandera que indica si se ha leído un número correcto.     */

    /* Lectura del primer dígito. Si se presiona ENTER se devuelve el valor */
    /* original de la variable.                                             */

    do
    {
      fflush ( stdin );
      gotoxy ( xcoord, ycoord );
      c = getch();
```

```
    if ( c == ENTER )
    {
       gotoxy ( xcoord, ycoord );
       return valor_ant;
    }
  } while ( !isdigit(c) );

  gotoxy ( xcoord, ycoord );
  putch(c);
  gotoxy ( xcoord + 1, ycoord ); cprintf ("      ");
  valor[0] = c;

  /* Lazo de lectura de los demás dígitos. */
  for ( j = 1; j < digitos; j++ )
  {
    do
    {
       fflush ( stdin ); flag1 = 0;
       gotoxy ( xcoord + j, ycoord );
       c = getch();
       if ( isdigit(c) ) flag1 = 1;
       if ( c == '.' )   flag1 = 1;
       if ( ENTER == c ) flag1 = 1;
    } while ( flag1 == 0 );

    if ( ENTER == c ) break;

    gotoxy ( xcoord + j, ycoord );
    putch(c);
    valor[j] = c;
  }
  valor[j] = '\0';

  /* Conversión del string leído a un número en punto flotante. */
  nuevo_valor = atof(puntero);

  /* Validación del valor leído. Si existe un error despliega un mensaje y*/
  /* vuelve a leer el valor.                                              */
  if ( (nuevo_valor > maximo) || (nuevo_valor < minimo) )
  {
    putch('\a'); flag = 0;
    gettext ( 9,1,71,8,pantalla );
    dibujar_caja ( 10,2, 70,7, DOBLE, COLOR_ERROR );
    window ( 11,3,69,6 ); textattr ( COLOR_ERROR ); clrscr();
    gotoxy ( 22,1 ); cprintf ("VALOR FUERA DE RANGO");
    gotoxy ( 2,2 );
    cprintf ("Ingrese un valor entre %.4f y %.4f ", minimo, maximo );
    gotoxy ( 2,3 ); cprintf ("valor ingresado = %.4f",nuevo_valor );
    gotoxy ( 2,4 ); cprintf ("Presione una tecla para continuar...");
    getch();
    clrscr();window(1,1,80,25);textattr(COLOR_NORMAL);
    puttext ( 9,1,71,8, pantalla );
    gotoxy ( xcoord, ycoord );
  }
 } while ( flag == 0 );
 gotoxy ( xcoord, ycoord );
 return nuevo_valor;        /* Retorna el valor leído. */
}                    /* Fin de la Función leer_flot (). */
```

```
/*                                                                    */
/* leer_ent () :  Función utilizada para leer un valor entero dentro de  */
/*                cierto rango predeterminado.                        */
/*                Parámetros:                                         */
/*              - valor_ant     --> Valor anterior de la variable.    */
/*              - minimo,maximo --> Rango permitido para la variable. */
/*              - digitos       --> Número de dígitos a leerse.       */
/*              - xcoord,ycoord --> Coordenadas del punto de lectura. */
/*                Si se presiona ENTER se mantiene el valor anterior. */

int leer_ent ( int valor_ant, int minimo, int maximo, int digitos,
               int xcoord, int ycoord )
{
  /* Declaración de variables. */

  char pantalla[1024];   /* Buffer en que se almacena la pantalla.   */
  int j, flag, flag1;    /* Variables auxiliares.                    */
  char valor[10];        /* Arreglo en que se guarda el valor leído. */
  char c;                /* Caracter leído del teclado.              */
  char *puntero;         /* Puntero al arreglo " valor ".            */
  int nuevo_valor;       /* Nuevo valor de la variable leída.        */

  puntero = &valor[0];    /* Inicialización del puntero al arreglo "valor". */

  /* Lazo principal de lectura de la variable en forma de string. */

  do
  {
    flag = 1;      /* Bandera que indica que se ha leído un valor correcto. */

    /* Lazo de lectura del primer dígito. Si se presiona ENTER se devuelve  */
    /* el valor original de la variable.                                    */
    do
    {
      fflush ( stdin );
      gotoxy ( xcoord, ycoord );
      c = getch();
      if ( c == ENTER )
      {
        gotoxy ( xcoord, ycoord );
         return valor_ant;
      }
    } while ( !isdigit(c) );

    gotoxy ( xcoord, ycoord );
    putch(c);
    gotoxy ( xcoord + 1, ycoord ); cprintf ("      ");
    valor[0] = c;


    /* Lazo de lectura de los demás dígitos.*/
    for ( j = 1; j < digitos; j++ )
    {
      do
      {
         fflush ( stdin ); flag1 = 0;
         gotoxy ( xcoord + j, ycoord );
```

```
        c = getch();
        if ( isdigit(c) ) flag1 = 1;
        if ( ENTER == c ) flag1 = 1;
      } while ( flag1 == 0 );

      if ( ENTER == c ) break;

      gotoxy ( xcoord + j, ycoord );
      putch(c);
      valor[j] = c;
    }

    valor[j] ='\0';

    /* Conversión del string en un número decimal. */
    nuevo_valor = atoi(puntero);


    /* Validación del valor leído. Si existe un error se despliega un mensa-*/
    /* je y se vuelve a leer la variable.                                    */
    if ( (nuevo_valor > maximo) || (nuevo_valor < minimo) )
    {
      putch('\a'); flag = 0;
      gettext ( 9,1,71,8,pantalla );
      dibujar_caja ( 10,2, 70,7, DOBLE, COLOR_ERROR );
      window ( 11,3,69,6 ); textattr ( COLOR_ERROR ); clrscr();
      gotoxy ( 22,1 ); cprintf ("VALOR FUERA DE RANGO");
      gotoxy ( 2,2 );
      cprintf ("Ingrese un valor entre %d y %d ", minimo, maximo );
      gotoxy ( 2,3 ); cprintf ("valor ingresado = %d",nuevo_valor );
      gotoxy ( 2,4 ); cprintf ("Presione una tecla para continuar...");
      getch();
      clrscr();window(1,1,80,25);textattr(COLOR_NORMAL);
      puttext ( 9,1,71,8, pantalla );
      gotoxy ( xcoord, ycoord );
    }

  } while ( flag == 0 );

  gotoxy ( xcoord, ycoord );

  return nuevo_valor;       /* Retorna el nuevo valor de la variable. */
}             /* Fin de la Función leer_ent () .  */
```

C.3. MODULO AGEVOLUC.C

```
/*                                                                      */
/* ESCUELA POLITECNICA NACIONAL.                                        */
/* FACULTAD DE INGENIERIA ELECTRICA.                                    */
/* DEPARTAMENTO DE ELECTRONICA Y CONTROL.                               */
/*                                                                      */
/* TESIS DE GRADO: Identificación de Parámetros utilizando un Algoritmo */
/*                 Genético de Optimización, Aplicación al Circuito     */
/*                 Equivalente de la Máquina de Inducción.              */
```

```
/*                                                                    */
/* AUTOR    : A. Javier Quintana M.                                   */
/* DIRECTOR : Dr. Hugo Banda Gamboa                                   */
/*                                                                    */
/* AGEVOLUC.C  : Este módulo contiene las funciones necesarias para realizar*/
/*               el proceso de Evolución Artificial de una población inicial*/
/*               de grupos de parámetros del circuito equivalente de la M.I.*/
/*               para obtener una población resultante, que contenga a un   */
/*               grupo de parámetros óptimos, de dicha máquina.             */
/*                                                                    */
/* CREACION : 2 - XII - 1993                                          */

/*
** ARCHIVOS DE INCLUSION.
*/

#include <stdio.h>
#include <conio.h>
#include <math.h>
#include "aginclud.h"

/*
** ALUSIONES A FUNCIONES.
*/

/* Funciones definidas en el módulo REPORTE.C . */

extern void reportes ( void );
extern void guardar_resultados ( int generacion );

/* Funciones definidas en el módulo COD_DEC.C . */

extern void cod_parametros ( void );
extern void decod_parametros ( void );

/* Funciones definidas en el módulo GENERAC.C . */

extern void igualar_poblaciones ( POBLACION pob_destino[], POBLACION
pob_fuente[] );
extern void reproduccion ( void );
extern void cruce ( void );

/* Función definida en el módulo EVALUAR.C .    */

extern void evaluar_aptitud ( POBLACION pob[] );

/* Función definida en MENUS.C . */

extern void salir ( void );

/* Función definida en este módulo. */

int  evolucion_artificial ( void );
void leer_pob_inicial ( void );
int  impedancia_prueba ( void );
void adecuar_poblacion ( POBLACION pob[] );
void estadisticas ( POBLACION pob[] );
```

```
/*
** DECLARACION DE VARIABLES GLOBALES.
*/

POBLACION pob_vieja [TAMANIO_POB];/* Estructura que almacena la población */
                                  /* de parámetros de la M.I. anterior al */
                                  /* proceso de evolución artificial.     */
POBLACION pob_nueva [TAMANIO_POB]; /* Estructura que almacena la población  */
                                   /* de parámetros de la M.I. posterior al */
                                   /* proceso de evolución artificial.      */
ESTADISTICAS indicador;            /* Estructura definida en AGINCLUD.H y   */
                                   /* en la cual se almacenan los indices   */
                                   /* estadísticos de los miembros de una   */
                                   /* población de grupos de parámetros del */
                                   /* circuito equivalente de la M.I.       */
extern DATOS_ENTRADA valor;        /* Estructura declarada en AG.C y que    */
                                   /* contiene los valores de todas las     */
                                   /* variables que se utilizan en el prog. */

/*
** CODIGO DE LAS FUNCIONES.
*/


/* evolucion_artificial () : Función desarrollada para realizar el proceso  */
/*                           de Evolución Artificial. Esta función lee la   */
/*                           la población inicial de parámetros del circuito*/
/*                           equivalente de la M.I. , y, sobre ella aplica  */
/*                           los operadores genéticos de Reproducción, Cruce*/
/*                           y Mutación, para obtener una nueva población   */
/*                           que contenga los grupos de parámetros óptimos  */
/*                           de la máquina en estudio.                      */

int evolucion_artificial ( void )
{
  int error_referencia = 0;
  int n_generacion = 0;  /* Contador del número de generaciones. */
  indicador.n_cruces = 0;      /* Resetea los contadores de cruces y */
  indicador.n_mutaciones = 0;  /* de mutaciones.                     */

  /* ***** ***** ***** ***** Inicialización ***** ***** ***** ***** */
  /* Lectura de la población inicial de parámetros del circuito equivalente */
  /* de la máquina de inducción ( M.I. ).                                   */
  gotoxy ( 2, 2 ); cprintf("Leyendo la Población Inicial...");
  leer_pob_inicial ( );

  gotoxy ( 2, 3 );
  cprintf("Procesando la información...Generación No. %d", n_generacion);

  /* Cálculo de la impedancia compleja de entrada para la prueba de carga.  */
  error_referencia = impedancia_prueba ( );
  if ( error_referencia == -1 )
     return -1;

  /* Conversión de los parámetros de la M.I. de su valor decimal,a un nuevo */
  /* código binario de 10 bits cada uno.                                    */
  cod_parametros ( );
```

```
/* Completar la población inicial de parámetros de la M.I. con los valores*/
/* de la resistencia de estator ( r1 ), dependiendo del método utilizado. */
adecuar_poblacion ( pob_vieja );

/* Evaluación de la aptitud que tienen todos los grupos de parámetros de  */
/* la población inicial.                                                  */
evaluar_aptitud ( pob_vieja );

/* Cálculo de los índices estadísticos de la población inicial.           */
estadisticas ( pob_vieja );

/* Almacenamiento en memoria de los resultados obtenidos en la generación */
/* inicial.                                                               */
guardar_resultados ( n_generacion );


/* *****  ***  *****  Proceso de Evolución Artificial  *****  ***  *****  */
/* Este proceso se repite hasta que el contador de generaciones alcance el*/
/* valor máximo de generaciones permitidas.                               */

do
{
  ++n_generacion;   /* Incrementa en uno el contador de generaciones. */
  gotoxy ( 45, 3 ); cprintf ( "%d ", n_generacion );

  /* Proceso de Reproducción de la población anterior de parámetros.      */
  reproduccion ( );

  /* Proceso de Cruce y Mutación de la población anterior de parámetros,  */
  /* para obtener una nueva población de grupos de parámetros.            */
  cruce ( );

  /* Conversión de los parámetros de la M.I. de su código binario de 10   */
  /* bits cada uno a su correspondiente valor en punto flotante.          */
  decod_parametros ( );

  /* Completar la nueva población de parámetros de la M.I. con los valores*/
  /* de la resistencia de estator ( r1 ), dependiendo del método empleado.*/
  adecuar_poblacion ( pob_nueva );

  /* Evaluación de la aptitud que tienen todos los grupos de parámetros de*/
  /* la nueva población.                                                  */
  evaluar_aptitud  ( pob_nueva );

  /* Cálculo de los índices estadísticos de la nueva población.           */
  estadisticas ( pob_nueva );

  /* Conversión de la nueva población obtenida en la población anterior.  */
  igualar_poblaciones ( pob_vieja, pob_nueva );

  /* Almacenamiento en memoria, de los resultados obtenidos en cada gen.  */
  guardar_resultados ( n_generacion );

} while( (indicador.aptitud_maxima < 10000.0)
      && (n_generacion < valor.max_generaciones));

valor.ultima_generacion = n_generacion;
```

```
  /* Generación de los reportes de todos los resultados obtenidos durante el*/
  /* proceso de Evolución Artificial.                                        */
  gotoxy ( 2, 4 ); cprintf("Creando los Reportes de Resultados...");
  reportes ( );

  putch ('\a');
  gotoxy ( 2, 5 ); cprintf("Presione una tecla para continuar....");
  getch();

  return 0;
}           /* Fin de la Función evolucion_artificial(). */



/*                                                                          */
/* leer_pob_inicial () : Esta función lee la población inicial de grupos de */
/*                       parámetros del circuito equivalente de la M.I., y  */
/*                       los almacena en la estructura "pob_vieja".         */
/*                       Esta lectura se la realiza ya sea del archivo      */
/*                       M_BASICO.DAT o del archivo M_IEEE.DAT, según el    */
/*                       método que se utilizó para evaluar la población    */
/*                       inicial de parámetros.                             */

void leer_pob_inicial ( void )
{
  /* Declaración de variables. */

  FILE *fpin;  /* Puntero al archivo de entrada de datos. */
  int  j;      /* Variable auxiliar. */

  /* Apertura del archivo que contiene la población inicial correspondiente */
  /* al método BASICO.                                                      */
  /* Si existe algún error, se imprime un mensaje en pantalla y se da por   */
  /* terminada la ejecución del programa.                                   */
  if ( valor.metodo == M_BASICO )
  {
    if ((fpin = fopen("M_BASICO.DAT","r")) == NULL )
    {
      putch ('\a');
      gotoxy( 2, 3 );
      cprintf("No se puede abrir archivo de entrada M_BASICO.DAT");
      gotoxy ( 2, 4 );
      cprintf("Presione una tecla para salir....");
      getch();
      salir();
    }
  }


  /* Apertura del archivo que contiene la población inicial correspondiente */
  /* al método IEEE.                                                        */
  /* Si existe algún error, se imprime un mensaje en pantalla y se da por   */
  /* terminada la ejecución del programa.                                   */

  if ( valor.metodo == M_IEEE )
  {
    if ((fpin = fopen("M_IEEE.DAT","r")) == NULL )
    {
```

```
      putch ('\a');
      gotoxy ( 2, 3 );
      cprintf("No se puede abrir archivo de entrada M_IEEE.DAT");
      gotoxy ( 2, 4 );
      cprintf("Presione una tecla para salir....");
      getch();
      salir();
    }
  }

  fflush (stdin);  /* Limpieza del buffer de entrada. */


  /* Lectura de la población inicial de parámetros del circuito equivalente */
  /* de la máquina de inducción, una vez abierto el archivo respectivo.     */

  for ( j = 0; j < valor.num_pruebas; j++ )
  {
    fscanf(fpin,"%f",&pob_vieja[j].r2);  /* Lee r2.  */
    fscanf(fpin,"%f",&pob_vieja[j].x1);  /* Lee x1.  */
    fscanf(fpin,"%f",&pob_vieja[j].xm);  /* Lee xm.  */
    fscanf(fpin,"%f",&pob_vieja[j].rfe); /* Lee rfe. */
  }

  fflush (stdin);  /* Limpieza del buffer de entrada. */

  fclose(fpin);    /* Cierra el archivo desde el cual se ingresó los datos. */

  return;
}       /* Fin de la Función leer_pob_inicial() */



/*                                                                          */
/* impedancia_prueba () : Esta función calcula la impedancia compleja de    */
/*                        entrada a la máquina de inducción, para una prueba*/
/*                        de carga definida en la estructura "valor", y los */
/*                        resultados son almacenados en la misma estructura.*/
/*                        Esta función se emplea en evolucion_artificial(). */

int impedancia_prueba ( void )
{
  int j;
  float psi, mod_z; /* Variables auxiliares. */
  float cos_phi;

  /* Cálculo del ángulo característico de la impedancia de entrada.  */
  cos_phi = valor.potencia/(valor.voltaje*valor.corriente*sqrt(3.0));

  if ( cos_phi > 1.0 )
  {
    putch ('\a');

    for ( j = 2; j < 6; j++ )
    {
       gotoxy ( 2,j );
       cprintf ("                                                  ");
    }
```

```
    gotoxy ( 2,2 );
    cprintf ("ERROR : Los valores de voltaje,corriente y potencia,");
    gotoxy ( 2,3 );
    cprintf ("ingresados como datos de la prueba de carga, impiden");
    gotoxy ( 2,4 );
    cprintf ("la obtención de los valores de referencia.");
    gotoxy ( 2,5 );
    cprintf ("Presione una tecla para continuar....");
    getch();

    return -1;
  }

  psi = acos (cos_phi);

  /* Cálculo del módulo de la impedancia de entrada. */
  mod_z = valor.voltaje/( valor.corriente*sqrt(3.0) );

  /* Cálculo de la parte real de la impedancia de la máquina de inducción.  */
  valor.z_real = mod_z*cos( psi );

  /* Cálculo de la parte imaginaria de la impedancia de la M.I.             */
  valor.z_imag = mod_z*sin( psi );
  return 0;
}       /* Fin de la Función impeddancia_prueba() */


/*                                                                          */
/* adecuar_poblacion ( ) : Función que completa una estructura con el valor */
/*                         de la resistencia de estator, dependiendo del    */
/*                         método utilizado para obtener la población ini-  */
/*                         cial de parámetros del circuito equivalente de   */
/*                         la máquina de inducción.                         */
/*                         Parámetro :                                      */
/*                       - pob --> Estructura a ser completada.             */
/*                         Esta función se emplea en evolucion_artificial().*/

void adecuar_poblacion ( POBLACION pob[] )
{
  int j;  /* Variable auxiliar. */

  /* Si el método utilizado es el recomendado por el IEEE, se completa la   */
  /* población con el valor de la resistencia de estator medida experiment. */
  if ( valor.metodo == M_IEEE )
  {
    for ( j = 0; j < valor.num_pruebas; j++ )
      pob[j].r1 = valor.r1;
  }
  /* Si el método utilizado es el BASICO, se completa la población con su   */
  /* correspondiente valor de la resistencia de rotor.                       */
  if ( valor.metodo == M_BASICO )
  {
    for ( j = 0; j < valor.num_pruebas; j++ )
      pob[j].r1 = pob[j].r2;
  }
  return;
}       /* Fin de la Función adecuar_poblacion() */
```

```
/*                                                                    */
/* estadisticas () : Esta función se emplea para encontrar la aptitud máxi- */
/*                   ma, promedio y mínima de una población de parámetros  */
/*                   de la M.I., así como su sumatoria. Estos valores se   */
/*                   almacenan en la estructura "indicador", que es del ti- */
/*                   po ESTADISTICAS.                                      */
/*                   Parámetros :                                          */
/*                 - pob --> Población de la cual se desean obtener estas  */
/*                           estadísticas.                                 */

void estadisticas ( POBLACION pob[] )
{
  int j;  /* Variable auxiliar. */

  /* Inicialización de los indicadores estadísticos de la población. */

  indicador.aptitud_sumatoria = pob[0].aptitud;
  indicador.aptitud_minima    = pob[0].aptitud;
  indicador.aptitud_maxima    = pob[0].aptitud;


  /* Evaluación de los índices estadísticos. */

  for ( j = 1; j < valor.num_pruebas; j++)
  {
    indicador.aptitud_sumatoria += pob[j].aptitud;   /* Suma de aptitudes.  */

    if ( pob[j].aptitud > indicador.aptitud_maxima ) /* Aptitud máxima.     */
      indicador.aptitud_maxima = pob[j].aptitud;

    if ( pob[j].aptitud < indicador.aptitud_minima ) /* Aptitud mínima.     */
      indicador.aptitud_minima = pob[j].aptitud;
  }
                                                     /* Aptitud promedio.   */
  indicador.aptitud_promedio = indicador.aptitud_sumatoria/valor.num_pruebas;

  return;
}       /* Fin de la Función estadisticas () */
```

C.4. MODULO GENERAC.C.

```
/*                                                                    */
/* ESCUELA POLITECNICA NACIONAL.                                      */
/* FACULTAD DE INGENIERIA ELECTRICA.                                  */
/* DEPARTAMENTO DE ELECTRONICA Y CONTROL.                             */
/*                                                                    */
/* TESIS DE GRADO: Identificación de Parámetros utilizando un Algoritmo */
/*                 Genético de Optimización, Aplicación al Circuito   */
/*                 Equivalente de la Máquina de Inducción.            */
/*                                                                    */
/* AUTOR    : A. Javier Quintana M.                                   */
/* DIRECTOR : Dr. Hugo Banda Gamboa                                   */
/*                                                                    */
```

```
/* GENERAC.C:  Este módulo contiene las funciones que permiten llevar a cabo*/
/*             el proceso de Evolución Artificial, con sus respectivas fun- */
/*             ciones auxiliares. El mecanismo de Evolución Artificial com- */
/*             prende los procesos de Reproducción, Cruce y Mutación.       */
/*             Estos procesos se realizan sobre el código binario de los    */
/*             grupos de parámetros de la Máquina de Inducción.             */
/*             Con estos procesos se genera una nueva población de grupos   */
/*             de parámetros, con mejores características de aptitud.       */
/*                                                                          */
/* CREACION : 2 - XII - 1993.                                               */


/*
** ARCHIVOS DE INCLUSION.
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "aginclud.h"

/*
** ALUSIONES A FUNCIONES.
*/

/* Funciones utilizadas en el proceso de Reproducción. */
void reproduccion ( void );
void definir_ruleta ( POBLACION pob[] );
int  buscar_aptitud ( POBLACION pob[], float valor );
void igualar_poblaciones ( POBLACION pob_destino[], POBLACION pob_fuente[] );

/* Funciones utilizadas en los procesos de Cruce y Mutación. */
void cruce ( void );
void selec_parejas ( POBLACION pob[] );
int  mutacion ( int valor_allele );
int  lanzar_dados ( float probabilidad );



/*
** DECLARACION DE VARIABLES GLOBALES.
*/

extern ESTADISTICAS indicador;/* Estructura declarada en AGEVOLUC.C, en la  */
                              /* que se almacenan los índices estadísticos  */
                              /* de la población que está siendo procesada. */
extern DATOS_ENTRADA valor; /* Estructura declarada en INICIALZ.C, en la que*/
                            /* se guardan los valores ingresados para todas */
                            /* las variables que se emplean en el programa. */


/*
** FUNCIONES UTILIZADAS EN EL PROCESO DE REPRODUCCION.
*/
```

```
/*                                                                    */
/* reproduccion () : Esta función lleva a cabo el proceso de Reproducción de*/
/*                   los grupos de parámetros óptimos de la población que se*/
/*                   procesa en ese instante.  Para ello es necesario defi- */
/*                   nir, primeramente, el número de veces que debe reprodu-*/
/*                   cirse cada grupo de parámetros de dicha población, lo  */
/*                   cual depende de su aptitud.                            */

void reproduccion ( void )
{
  /* Declaración de variables. */

  extern POBLACION pob_vieja[TAMANIO_POB]; /* Estructura declarada en AG.C, */
                                           /* en la que se guardan los có-  */
                                           /* digos binarios de los grupos  */
                                           /* de parámetros de la M.I.      */
  POBLACION pob_aux[TAMANIO_POB];          /* Estructura auxiliar.          */
  int j, j1;        /* Variables auxiliares. */
  int offset = 0;   /* Contador que ayuda en el proceso de reproducción.    */

  /* Determinación del número de veces que debe reproducirse cada grupo de  */
  /* parámetros de la población, según su aptitud.                          */
  definir_ruleta ( pob_vieja );

  /* *****  *****  *******  Proceso de Reproducción  *******  *****  *****  */
  /* Cada grupo de parámetros se copia un número de veces anteriormente de- */
  /* finido, dependiendo de su aptitud, en una población auxiliar.          */

  for ( j = 0; j < valor.num_pruebas; j++ )
  {
    for ( j1 = 0; j1 < pob_vieja[j].ruleta; j1++ )
      pob_aux[ offset + j1 ] = pob_vieja[j];

    offset += pob_vieja[j].ruleta;
  }

  /* Se transfiere el contenido de la población auxiliar a la población que */
  /* está siendo procesada actualmente.                                     */
  igualar_poblaciones ( pob_vieja, pob_aux );

  return;
}       /* Fin de la Función reproduccion () */




/*                                                                    */
/* definir_ruleta () : Esta función se utiliza para determinar el número de */
/*                     veces que debe reproducirse cada grupo de parámetros */
/*                     dependiendo de su aptitud. Esto es, a mayor aptitud  */
/*                     mayor es el número de veces que debe reproducirse un */
/*                     grupo de parámetros.                                 */
/*                     Parámetros :                                         */
/*                   - pob --> Población de parámetros para la cual se desea*/
/*                             definir el índice de reproducción.         */
/*                     Esta función se emplea en la función reproduccion(). */
```

```
void definir_ruleta ( POBLACION pob[] )
{
  /* Declaración de variables. */

  int    j, j1;          /* Variables auxiliares. */
  double x;              /* Variable auxiliar.    */
  int    ruleta;         /* Contador del número total de copias que deben   */
                         /* realizarse de todos los grupos de parámetros de */
                         /* la población.                                   */
  int    diferencia;     /* Indicador de la diferencia que existe entre el  */
                         /* número total de copias y el tamaño de la pobla- */
                         /* ción de grupos de parámetros.                   */
  int    minimo, maximo; /* Punteros que indican los grupos de parámetros   */
                         /* de la población, que tienen la menor y la mayor */
                         /* aptitud, respectivamente.                       */
  float  menor_aptitud;  /* Variable en la que se almacena la menor aptitud */
                         /* encontrada en la población.                     */

  /* Inicialización de las variables. */

  ruleta     = 0;
  diferencia = 0;
  minimo     = 0;
  maximo     = 0;

  /* Determinación del grupo de parámetros que tiene la mayor aptitud. */
  maximo = buscar_aptitud ( pob, indicador.aptitud_maxima );

  /* Determinación del número de veces que debe reproducirse un grupo de    */
  /* parámetros, según la aptitud que tengan.                               */

  for ( j = 0; j < valor.num_pruebas; j++ )
  {
    x =  (pob[j].aptitud / indicador.aptitud_promedio) + 0.5;
    pob[j].ruleta = (int) floor( x ); /* Número de veces que debe reprodu-  */
                                      /* cirse un cierto grupo de parámetros*/
    ruleta += pob[j].ruleta;          /* Incremento del contador de copias. */
  }

  /* Si el contador de copias es menor que el tamaño de la población, el nú-*/
  /* mero de reproducciones del grupo de parámetros que tiene la mayor apti-*/
  /* tud, debe incrementarse el número necesario de veces hasta que el con- */
  /* tador de copias se iguale con el tamaño de la población.               */

  if ( ruleta < valor.num_pruebas )
  {
    diferencia = valor.num_pruebas - ruleta;/*Número de copias a aumentarse.*/

    /* Equiparación del contador de copias con el tamaño de la población.   */

    for ( j1 = 0; j1 < diferencia; j1++ )
     {
      ++pob[maximo].ruleta; /* Incremento en el número de reproducciones del*/
                            /* grupo de parámetros con la mayor aptitud.    */
      ++ruleta;             /* Incremento del contador de copias.           */
     }
  }
```

```
  /* Si el contador de copias es mayor que el tamaño de la población, debe  */
  /* reducirse el número de reproducciones de los grupos de parámetros que  */
  /* tengan la menor aptitud, siempre y cuando este número sea mayor que 0, */
  /* hasta que el contador de copias sea igual al tamaño de la población.   */

  if ( ruleta > valor.num_pruebas )
  {
    diferencia = ruleta - valor.num_pruebas;/* Número de copias a reducirse.*/

    /* Equiparación del contador de copias con el tamaño de la población.   */

    for ( j = 0; j < diferencia; j++ )
    {
      menor_aptitud = pob[maximo].aptitud;
      minimo = maximo;

      /* Determinación del grupo de parámetros que tiene la menor aptitud y */
      /* que además tiene un número de reproducciones mayor que 0.          */

      for ( j1 = 0; j1 < valor.num_pruebas; j1++ )
      {
        if ( pob[j1].ruleta > 0 )
        {
          if ( pob[j1].aptitud <= menor_aptitud )
          {
            menor_aptitud = pob[j1].aptitud;
            minimo = j1; /* Indicador del grupo de parámetros seleccionado. */
          }
        }
      }

      /* Reducción del número de reproducciones del grupo de parámetros que */
      /* ha sido seleccionado.                                              */
      --pob[minimo].ruleta;

      /* Reducción del contador de copias. */
      --ruleta;
    }
  }

  return;
}       /* Fin de la Función definir_ruleta () */




/*                                                                         */
/* buscar_aptitud () : Esta función determina cual grupo de parámetros, en */
/*                     una población dada, tiene un cierto valor de aptitud.*/
/*                     Parámetros:                                         */
/*                   - pob   --> Población en la cual se desea buscar.     */
/*                   - valor --> Aptitud a encontrarse en un grupo de pará- */
/*                               metros de la población.                   */
/*                     Esta función se emplea en la función definir_ruleta()*/
```

```
int buscar_aptitud ( POBLACION pob[], float valor_aptitud )
{
  /* Declaración de variables. */

  int j;         /* Variable auxiliar. */
  int seleccion; /* Indicador del grupo de parámetros seleccionado. */

  /* Búsqueda del grupo de parámetros que tiene una cierta aptitud. */

  for ( j = 0; j < valor.num_pruebas; j++ )
  {
    if ( pob[j].aptitud == valor_aptitud )
    {
      seleccion = j; /* Grupo de parámetros seleccionado. */
      break;
    }
  }

  return seleccion;
}                  /* Fin de la Función buscar_aptitud () */


/*                                                                    */
/* igualar_poblaciones () : Esta función copia el contenido de una estructu-*/
/*                          ra tipo POBLACION en otra similar.        */
/*                          Parámetros :                              */
/*                        - pob_destino --> Estructura en la cual se desea  */
/*                                          copiar.                   */
/*                        - pob_fuente  --> Estructura a copiarse.    */
/*                          Esta función se utiliza en AG.C y en la función */
/*                          reproduccion().                           */

void igualar_poblaciones ( POBLACION pob_destino[], POBLACION pob_fuente[] )
{
  int j;  /* Variable auxiliar. */

  /* Proceso de copiado de una estructura tipo POBLACION en otra similar.   */

  for ( j = 0; j < valor.num_pruebas; j++ )
    pob_destino[j] = pob_fuente[j];

  return;
}        /* Fin de la Función igualar_poblaciones () */



/*
** FUNCIONES UTILIZADAS EN LOS PROCESOS DE CRUCE Y MUTACION.
*/



/*                                                                    */
/* cruce () : Función utilizada para realizar el proceso de Cruce y Mutación*/
/*            en pares de grupos de parámetros de una población, para gene- */
/*            rar otra con mejores características de aptitud. Para ejecutar*/
/*            este proceso, es necesario que primeramente se definan dichas */
/*            parejas de grupos de parámetros.                        */
```

```
void cruce ( void )
{
  /* Declaración de variables. */

  extern POBLACION pob_vieja[TAMANIO_POB]; /* Estructura declarada en AG.C, */
                                           /* y que constituye la población */
                                           /* de parámetros sobre la cual se*/
                                           /* va a trabajar.                */
  extern POBLACION pob_nueva[TAMANIO_POB]; /* Estructura declarada en AG.C, */
                                           /* y que constituye la nueva po- */
                                           /* blación que se genera.        */
  int bandera [TAMANIO_POB];  /* Vector de banderas que indican las parejas */
                              /* con las cuales ya se realizó el Cruce.     */
  int j, j1;          /* Variables auxiliares. */
  int jcruce;         /* Puntero que indica el lugar del cromosoma en el cual  */
                      /* se va a realizar el Cruce.                            */
  int par1, par2;     /* Punteros que indican cuales son los grupos de paráme- */
                      /* tros entre los que se va a realizar el Cruce y la Mu- */
                      /* tación.                                               */

  /* Inicialización de banderas */

  for ( j = 0; j < valor.num_pruebas; j++)
    bandera[j] = 0;

  /* Selección de las parejas de grupos de parámetros entre los cuales se va*/
  /* a realizar el cruce.                                                    */

  selec_parejas ( pob_vieja );



  /* *****  *****  *****  Proceso de Cruce y Mutación  *****  *****  *****  */


  for ( j = 0; j < valor.num_pruebas; j++ )
  {
    par1 = j;                       /* Definición del par de grupos de pará- */
    par2 = pob_vieja[par1].pareja; /* metros entre los cuales se va a reali-*/
                                    /* zar el Cruce y la Mutación.           */

    /* Proceso de Cruce y Mutación entre dos pares de grupos de parámetros, */
    /* que no hayan sido sometidos anteriormente a estos procesos.          */

    if ( (!(bandera[par1])) || (!(bandera[par2])) )
    {
      bandera[par1] = 1;  /* Habilitación de banderas que indican que este  */
      bandera[par2] = 1;  /* par de grupos de parámetros ha sido seleccio-  */
                          /* para los procesos de Cruce y Mutación.         */

      /* Si existe la probabilidad de que ocurra un cruce, se realizan los  */
      /* procesos de Cruce y Mutación.                                      */

      if ( lanzar_dados ( valor.p_cruce ) )
      {
        /* Selección del punto de Cruce en los cromosomas del par grupos de */
        /* parámetros.                                                      */
```

```
        do
          jcruce = random ( L_CROMOSOMA );
        while ( !jcruce );

        ++indicador.n_cruces;/* Incremento del contador del número de cruces*/

        /* Ejecución de los procesos de Cruce y Mutación, propiamente dichos*/

        for ( j1 = 0; j1 < jcruce; j1++ )
        {
          pob_nueva[par1].crom[j1] = mutacion ( pob_vieja[par1].crom[j1] );
          pob_nueva[par2].crom[j1] = mutacion ( pob_vieja[par2].crom[j1] );
        }

        for ( j1 = jcruce; j1 < L_CROMOSOMA; j1++ )
        {
          pob_nueva[par1].crom[j1] = mutacion ( pob_vieja[par2].crom[j1] );
          pob_nueva[par2].crom[j1] = mutacion ( pob_vieja[par1].crom[j1] );
        }

      }

      /* Si no existe la probabilidad de que ocurra un cruce, se realiza    */
      /* únicamente el proceso de Mutación.                                 */

      else
      {
         jcruce = L_CROMOSOMA;

         for ( j1 = 0; j1 < jcruce; j1++ )
        {
          pob_nueva[par1].crom[j1] = mutacion ( pob_vieja[par1].crom[j1] );
          pob_nueva[par2].crom[j1] = mutacion ( pob_vieja[par2].crom[j1] );
        }
      }

      /* Almacenamiento de ciertas características de estos procesos en la  */
      /* estructura que contiene la nueva población generada.               */

      pob_nueva[par1].pareja  = par2;                    /* Pareja. */
      pob_nueva[par1].ruleta  = pob_vieja[par1].ruleta; /* # reproducciones.*/
      pob_nueva[par1].cruce   = jcruce;                  /* Punto de cruce.  */

      pob_nueva[par2].pareja  = par1;
      pob_nueva[par2].ruleta  = pob_vieja[par2].ruleta;
      pob_nueva[par2].cruce   = jcruce;
    }
  }
  return;
}        /* Fin de la Función cruce () */



/*                                                                        */
/* selec_parejas () : Función que se utiliza para encontrar las parejas de  */
/*                    grupos de parámetros entre las cuales se van a reali- */
/*                    zar los procesos de Cruce y Mutación.               */
/*                    Parámetros:                                         */
```

```
/*                  - pob --> Población en la cual se va a realizar esta    */
/*                            selección.                                    */
/*                    Esta función se emplea en la función cruce().         */

void selec_parejas ( POBLACION pob[] )
{
  int bandera[TAMANIO_POB]; /* Vector de banderas que indican si un grupo   */
                            /* de parámetros ya ha sido seleccionado.       */
  int par; /* Puntero que indica una pareja. */
  int j;   /* Variable auxiliar. */

  /* Inicialización de banderas. */
  for ( j = 0; j < valor.num_pruebas; j++)
     bandera[j] = 0;

  /* Selección de la pareja del primer grupo de parámetros. */
  do
     par = random ( valor.num_pruebas );
  while ( !par );

  pob[0].pareja   = par; /* Indicación de la pareja. */
    bandera[0]    = 1;   /* Bandera que indica la selección. */
  pob[par].pareja = 0;
    bandera[par]  = 1;

  /* Selección de las parejas de los restantes grupos de parámetros. */

  for ( j = 1; j < valor.num_pruebas; j++ )
  {
    /* Si un grupo de parámetros no tiene asignada una pareja, se la escoge */
    /* en este momento.                                                     */
    if ( !(bandera[j]) )
    {
      bandera[j] = 1;   /* Habilitación de bandera que indica la selección. */

      /* Búsqueda de una pareja. */

      do
       par = random ( valor.num_pruebas );
      while ( bandera[par] );

      pob[j].pareja   = par; /* Indicación de las parejas. */
      pob[par].pareja = j;
      bandera[par]  = 1;   /* Habilitación de bandera que indica selecc.  */
    }
  }
  return;
}       /* Fin de la Función select_parejas () */



/*                                                                          */
/* mutacion () : Función que se emplea para mutar un bit de un cromosoma, si*/
/*               existe la probabilidad de que esto pueda ocurrir. La muta- */
/*               ción de un bit, quiere decir cambiar 1 por 0, o viceversa. */
/*             - valor_allele --> Bit que se desea mutar, y que forma parte */
/*                                del cromosoma.                            */
/*               Esta función se emplea en la función cruce().              */
```

```
int mutacion ( int valor_allele )
{
  int    mutar;  /* Variable que indica si existe la probabilidad de que    */
                 /* ocurra una mutación.                                     */

  /* Determinación de la existencia de una mutación. */

  mutar = lanzar_dados ( valor.p_mutacion );

  /* Se realiza la mutación en el bit, si existe la probabilidad de que és- */
  /* ta ocurra.                                                               */

  if ( mutar )
  {
    ++indicador.n_mutaciones; /* Incremento del contador de mutaciones. */

    if ( valor_allele )
      return FALSO;     /* Si el bit es igual a 1, lo convierte en 0. */

    if ( !valor_allele )
      return VERDAD;    /* Si el bit es igual a 0, lo convierte en 1. */
  }

  /* Si no existe la probabilidad de que ocurra una mutación, se deja al bit*/
  /* como estaba.                                                             */

  else
  {
    if ( valor_allele )
      return VERDAD;    /* Si el bit es igual a 1, se lo deja como 1. */

    if ( !valor_allele )
      return FALSO;     /* Si el bit es igual a 0, se lo deja como 0. */
  }

  return 0;
}          /* Fin de la Función mutacion () */




/*                                                                          */
/* lanzar_dados () : Función que se utiliza para determinar la ocurrencia   */
/*                   de un evento, de acuerdo con una probabilidad dada.    */
/*                   Parámetros :                                           */
/*                 - probabilidad --> Probabilidad de ocurrencia de un      */
/*                                    evento. No debe indicarse como %.     */
/*                   Esta función se utiliza en las funciones mutacion() y  */
/*                   cruce().                                               */

int lanzar_dados ( float probabilidad )
{
  if ( probabilidad == 1.0 )
    return VERDAD;   /* Si la probabilidad es 1, el evento siempre ocurre.  */

  if ( probabilidad == 0.0 )
    return FALSO;    /* Si la probabilidad es 0, el evento nunca ocurre.    */
```

```
  /* Determinación de la ocurrencia de un evento. */

  if ( ( random(1001)/1000.0 ) <= probabilidad )
    return VERDAD;
  else
    return FALSO;
}               /* Fin de la Función lanzar_dados () */
```

C.5. MODULO EVALUAR.C

```
/*                                                                      */
/* ESCUELA POLITECNICA NACIONAL.                                        */
/* FACULTAD DE INGENIERIA ELECTRICA.                                    */
/* DEPARTAMENTO DE ELECTRONICA Y CONTROL.                               */
/*                                                                      */
/* TESIS DE GRADO: Identificación de Parámetros utilizando un Algoritmo */
/*                 Genético de Optimización, Aplicación al Circuito     */
/*                 Equivalente de la Máquina de Inducción.              */
/*                                                                      */
/* AUTOR    : A. Javier Quintana M.                                     */
/* DIRECTOR : Dr. Hugo Banda Gamboa                                     */
/*                                                                      */
/* EVALUAR.C : Módulo que contiene una función que evalúa la aptitud de los */
/*             grupos de parámetros de una población. Esta aptitud es el    */
/*             inverso del error que existe entre los valores de impedancia */
/*             y torque obtenidos en la prueba de carga y los obtenidos a   */
/*             partir del circuito equivalente de la Máquina de Inducción.  */
/*             Esta función utiliza como parámetro de entrada el nombre de  */
/*             la población en la cual se desea evaluar la aptitud.         */
/*                                                                      */
/* CREACION : 2 - XII - 1993.                                           */

/*
** ARCHIVOS DE INCLUSION.
*/

#include <stdio.h>
#include <math.h>
#include <conio.h>
#include "aginclud.h"


/*
** ALUSION A LA FUNCION.
*/

void evaluar_aptitud ( POBLACION pob[] );



/*
** CODIGO DE LA FUNCION.
*/
```

```
void evaluar_aptitud ( POBLACION pob[] )
{
  /* Declaración de variables. */

  extern DATOS_ENTRADA valor;   /* Estructura declarada en INICIALZ.C, y que*/
                                /* contiene los valores de las variables de */
                                /* entrada empledas en el programa.         */

  float zm_real, zm_imag;       /* Variables auxiliares empleadas para sim- */
  float s, v_prueba, torque_m;  /* plificar las ecuaciones, y en las cuales */
  float k1, k2, k3;             /* se almacenan los valores de la prueba de */
  int   m;                      /* carga y las constantes de la función de  */
                                /* error contenidos en la estructura "valor"*/


  float torque, z_real, z_imag, ws; /* Variables para torque, impedancia y  */
                                    /* velocidad de sincronismo.            */
  float r1, r2, x1, x2, xm, rfe;    /* Variables de los parámetros del C.E. */
  float error_z_real, error_z_imag; /* Variables de error total, de error en*/
  float error_torque, error;        /* torque e impedancias.                */
  float r, x, a, b, c;              /* Variables auxiliares.                */
  int   n;                          /* Exponente de la función de error.    */
  int   j;                          /* Variable auxiliar.                   */

  /* *****  *****  Inicialización de las variables empleadas.  *****  ***** */

  /* Datos de la Prueba de Carga.  */

  zm_real  = valor.z_real;          /* Impedancia real.       */
  zm_imag  = valor.z_imag;          /* Impedancia imaginaria. */
  torque_m = valor.torque;          /* Torque en el eje.      */
  v_prueba = valor.voltaje;         /* Voltaje de entrada.    */
  s        = valor.deslizamiento;   /* Deslizamiento.         */
  ws = 60*PI;                       /* Velocidad de sincronismo en rad/seg.  */

  /* Constantes de la Función de Error.  */

  k1 = valor.k1; /* Constantes de la función de error. */
  k2 = valor.k2;
  k3 = valor.k3;
  m  = valor.m;
  n  = 2*m;      /* Exponente de la función de error.  */


  /*  *****  Cálculo de la Aptitud de los miembros de la Población  *****   */


  for ( j = 0; j < valor.num_pruebas; j++ )
  {
    r2 = pob[j].r2;     /* Lectura de los valores de un grupo de parámetros */
    r1 = pob[j].r1;     /* del circuito equivalente de la M.I.              */
    x1 = pob[j].x1;
    x2 = x1;
    xm = pob[j].xm;
    rfe = pob[j].rfe;
```

```
    /* Cálculo de la impedancia real e imaginaria, según el C.E. */

    a = rfe*r2/s - x2*xm;
    b = rfe*(xm+x2) + xm*r2/s;
    c = xm*rfe/(a*a + b*b);

    z_real = r1 + c*( b*r2/s - a*x2 );
    z_imag = x1 + c*( a*r2/s + b*x2 );

    /* Cálculo del Torque desarrollado, según el C.E. */

    r  = r1 + r2/s;
    x  = x1 + x2;

    torque = v_prueba*v_prueba*(r2/s)/(ws*(r*r + x*x));

    /* Cálculo del error en las impedancias y en el torque. */

    error_z_real = fabs(zm_real - z_real);
    error_z_imag = fabs(zm_imag - z_imag);
    error_torque = fabs(torque_m - torque);

    /* Cálculo del error total. */
    error = k1*pow(error_z_real,n) + k2*pow(error_z_imag,n) +
          k3*pow(error_torque,n);

    if ( error <= 0.0 )          /* Seguridad que evita un overflow. */
       error = 0.000000000001;

    /* Almacenamiento de los resultados obtenidos. */

    pob[j].aptitud = 1/error;    /* Aptitud.                */
    pob[j].z_real = z_real;      /* Impedancia real.        */
    pob[j].z_imag = z_imag;      /* Impedancia imaginaria. */
    pob[j].torque = torque;      /* Torque desarrollado.    */

  }
  return;
}       /* Fin de la Función evaluar_aptitud() */
```

C.6. COD_DEC.C.

```
/*                                                                    */
/* ESCUELA POLITECNICA NACIONAL.                                      */
/* FACULTAD DE INGENIERIA ELECTRICA.                                  */
/* DEPARTAMENTO DE ELECTRONICA Y CONTROL.                             */
/*                                                                    */
/* TESIS DE GRADO: Identificación de Parámetros utilizando un Algoritmo */
/*                 Genético de Optimización, Aplicación al Circuito   */
/*                 Equivalente de la Máquina de Inducción.            */
/*                                                                    */
```

```
/* AUTOR    : A. Javier Quintana M.                                        */
/* DIRECTOR : Dr. Hugo Banda Gamboa                                        */
/*                                                                         */
/* COD_DEC.C : Módulo que contiene las funciones necesarias para convertir */
/*             los Parámetros del Circuito Equivalente de la Máquina de    */
/*             Inducción, en:                                              */
/*             - Números binarios de 10 bits, a partir de valores en punto */
/*               flotante.                                                 */
/*             - Números en punto flotante, a partir de números binarios de */
/*               10 bits.                                                  */
/*                                                                         */
/* CREACION : 2 - XII - 1993.                                              */


/*
** ARCHIVOS DE INCLUSION.
*/

#include <stdio.h>
#include <math.h>
#include "aginclud.h"

/*
** ALUSIONES A FUNCIONES.
*/

/* Funciones necesarias para codificar un número decimal en otro binario.  */

void  cod_parametros ( void );
int   cod_numero ( float y, float ymax, float ymin );
void  cod_crom   ( int codigo, int *crom );

/* Funciones necesarias para decodificar un número binario en otro decimal. */

void  decod_parametros ( void );
float decod_crom   ( int *crom );
float decod_numero ( float x, float ymax, float ymin );


/*
** DECLARACION DE VARIABLES GLOBALES.
*/

extern DATOS_ENTRADA valor;  /* Estructura declarada en "inicialz.c", y que */
                             /* contiene los valores de todas las variables */
                             /* que se utilizan en el programa.             */
static float xmax = 1023.0;  /* Variables globales utilizadas solo en este  */
static float xmin =    0.0;  /* módulo.  Corresponden a los valores decima- */
                             /* les, máximo y mínimo, que se pueden obtener */
                             /* a partir de un número binario de 10 bits.   */

/*
** FUNCIONES UTLIZADAS EN LA CODIFICACION.
*/
```

```
/*                                                                    */
/* cod_parametros() : Esta función convierte los parámetros del circuito    */
/*                    equivalente de la máquina de inducción, desde su      */
/*                    código inicial en punto flotante a un nuevo código    */
/*                    binario de 10 bits.                                   */
/*                    Los valores de los parámetros en punto flotante deben */
/*                    estar almacenados en la estructura "pob_vieja", de la */
/*                    misma manera, los códigos binarios se guardan en dicha*/
/*                    estructura.                                           */

void cod_parametros ( void )
{
  /* Declaración de variables. */

  extern POBLACION pob_vieja[TAMANIO_POB];/* Estructura declarada en "ag.c",*/
                                          /* que contiene la población de   */
                                          /* de parámetros a codificar.     */
  int r2_cod, x1_cod;     /* Variables que almacenan un      */
  int xm_cod, rfe_cod;     /* código entero intermedio.       */
  int j;                  /* Variable auxiliar.              */


  /* *****  ***** *****    Proceso de Codificación    ***** *****  ***** */


  for ( j = 0; j < valor.num_pruebas; j++ )
  {
    /* Conversión de los valores en punto flotante a un código intermedio,  */
    /* de tipo entero, comprendido entre xmax y xmin.                       */

    r2_cod  = cod_numero ( pob_vieja[j].r2, valor.r2_max, valor.r2_min );
    x1_cod  = cod_numero ( pob_vieja[j].x1, valor.x1_max, valor.x1_min );
    xm_cod  = cod_numero ( pob_vieja[j].xm, valor.xm_max, valor.xm_min );
    rfe_cod = cod_numero ( pob_vieja[j].rfe, valor.rfe_max, valor.rfe_min );

    /* Conversión de los códigos enteros intermedios a sus nuevos códigos   */
    /* binarios de 10 bits. Estos códigos se guardan en pob_vieja[j].crom.  */

    cod_crom ( r2_cod,  &pob_vieja[j].crom[0] );
    cod_crom ( x1_cod,  &pob_vieja[j].crom[10] );
    cod_crom ( xm_cod,  &pob_vieja[j].crom[20] );
    cod_crom ( rfe_cod, &pob_vieja[j].crom[30] );
  }

  return;
}       /* Fin de la Función cod_parametros() */



/*                                                                    */
/* cod_numero ():  Función utilizada para convertir un número en punto      */
/*                 flotante a un código entero, comprendido entre xmax y    */
/*                 xmin, empleando la ecuación de una recta.                */
/*                 Parámetros:                                              */
/*               - y         --> Número en punto flotante a ser codificado. */
/*               - ymax,ymin --> Rango de variación de y.                   */
/*                 Esta función se emplea en cod_parametros().              */
```

```
int cod_numero ( float y, float ymax, float ymin )
{
  /* Declaración de variables. */

  double x;        /* Variable auxiliar  */
  int    codigo;   /* Variable en la cual se almacena el código entero */

  /* Ecuación de la recta utilizada para convertir en un código entero un   */
  /* número en punto flotante, aproximandolo al número inmediato superior.  */

  x = ((xmax - xmin)/(ymax - ymin))*(y - ymin) + xmin + .5;
  codigo = (int) floor ( x );

  return codigo;
}              /* Fin de la Función cod_numero() */



/*                                                                          */
/* cod_crom ():  Función utilizada para convertir un número entero en un    */
/*               número binario de 10 bits. Se emplea el método de las      */
/*               divisiones sucesivas del número entero para 2, en el cual  */
/*               los residuos pasan a conformar el número binario.          */
/*.              Parámetros:                                                */
/*             - codigo --> Número entero a ser codificado.                 */
/*             - *crom  --> Puntero utilizado para guardar el número bin.   */
/*               Esta función se emplea en cod_parametros().                */

void cod_crom ( int codigo, int *crom )
{
  /* Declaración de variables. */

  int j;          /* Variable auxiliar. */
  int cuociente; /* Variable en la que se almacena el cuociente de la div.  */
  int residuo;   /* Variable en la que se almacena el residuo de la div.    */

  /* Conversión de un número entero en un número binario de 10 bits, */
  /* por medio de la división sucesiva de un número entero para 2.   */

  for ( j = 0; j < L_PARAMETRO; j++ )
  {
    residuo   = (int)(codigo % 2);           /* Cálculo del residuo.  */
    cuociente = (int)((codigo - residuo)/2); /* Cálculo del cuociente */

    *(crom + j) = residuo;  /* Conformación del número binario. */

    codigo = cuociente;     /* Siguiente número a dividir.      */
  }

  return;
}       /*  Fin de Función cod_crom() */



/*
** FUNCIONES UTILIZADAS EN LA DECODIFICACION.
*/
```

```
/*                                                                    */
/* decod_parametros(): Función utilizada para convertir los valores de los  */
/*                     parámetros del circuito equivalente de la máquina de */
/*                     inducción, desde sus códigos binarios de 10 bits a   */
/*                     sus correspondientes valores en punto flotante.      */
/*                     Tanto el código binario como sus respectivos valores */
/*                     en punto flotante, se almacenan en la estructura     */
/*                     "pob_nueva".                                         */

void decod_parametros ( void )
{
  /* Declaración de variables. */

  extern POBLACION pob_nueva[TAMANIO_POB];/* Estructura declarada en "ag.c",*/
                                          /* que contiene el código binario.*/
  int j;                    /* Variable auxiliar. */
  float r2_cod, x1_cod;     /* Variables que almacenan un código */
  float xm_cod, rfe_cod;    /* flotante intermedio.              */

  /* *****  *****  *****  Proceso de Decodificación  *****  *****  ***** */

  for ( j = 0; j < valor.num_pruebas; j++ )
  {
    /* Conversión de los números en código binario de 10 bits a un código   */
    /* intermedio en punto flotante, comprendido entre xmax y xmin.         */

    r2_cod  = decod_crom ( &pob_nueva[j].crom[0] );
    x1_cod  = decod_crom ( &pob_nueva[j].crom[10] );
    xm_cod  = decod_crom ( &pob_nueva[j].crom[20] );
    rfe_cod = decod_crom ( &pob_nueva[j].crom[30] );

    /* Conversión de los códigos intermedios a sus correspondientes valores */
    /* verdaderos en punto flotante.                                        */

    pob_nueva[j].r2  = decod_numero ( r2_cod, valor.r2_max, valor.r2_min );
    pob_nueva[j].x1  = decod_numero ( x1_cod, valor.x1_max, valor.x1_min );
    pob_nueva[j].xm  = decod_numero ( xm_cod, valor.xm_max, valor.xm_min );
    pob_nueva[j].rfe = decod_numero ( rfe_cod, valor.rfe_max, valor.rfe_min );
  }

  return;
}       /* Fin de la Función decod_parametros() */




/*                                                                    */
/* decod_crom(): Función utilizada para convertir un número binario en su   */
/*               valor decimal respectivo.  Se emplea el método de sumar el */
/*               producto de 1s y 0s multiplicados por su correspondiente   */
/*               potencia de 2, la cual se determina según la ubicación de  */
/*               los mismos dentro de la cadena binaria.                    */
/*               Parámetros:                                                */
/*             - *crom --> Puntero referido al número binario.              */
/*               Esta función se emplea en decod_parametros().              */
```

```
float decod_crom ( int *crom )
{
  /* Declaración de variables. */

  int   j;      /* Variable auxiliar. */
  float acum;   /* Variable en la que se almacena el valor decimal.     */
  float potde2; /* Potencia de 2 correspondiente para cada elemento del */
                /* número binario.                                      */

  /* Inicialización de variables. */

  acum   = 0.0;
  potde2 = 1.0;

  /* Conversión del número binario de 10 bits en su valor decimal.     */

  for ( j = 0; j < L_PARAMETRO; j++)
  {
    if ( *( crom + j ) )  /* Si en el número binario existe un 1, */
      acum += potde2;     /* se suma el valor de la potencia de 2 */
                          /* correspondiente a su ubicación, y se */
    potde2 = 2*potde2;    /* incrementa la potencia de 2.         */
  }                       /* Si se tiene un 0, solo se incrementa */
                          /* la potencia de 2.                    */
  return acum;
}             /* Fin de la Función decod_crom() */



/*                                                                    */
/* decod_numero(): Función utilizada para convertir el número decimal en su */
/*                 verdadero valor en punto flotante.  Para conseguirlo se  */
/*                 utiliza la ecuación de una recta.                        */
/*                 Parámetros :                                             */
/*               - x         --> Número a ser decodificado.                 */
/*               - ymax,ymin --> Límites del número ya decodificado.        */
/*                 Esta función se emplea en decod_parametros().            */

float decod_numero ( float x, float ymax, float ymin )
{
  float y; /* Variable en la cual se almacena el número decodificado. */

  /* Ecuación de la recta utilizada para decodificar el número x,     */
  /* xmax y xmin, definen el rango de variación de x.                 */
  y = (( ymax - ymin )/( xmax - xmin ))*( x - xmin ) + ymin;
  return y;
}          /* Fin de la Función decod_numero() */
```

C.7. MODULO POBINIC.C.

```
/*                                                                    */
/* ESCUELA POLITECNICA NACIONAL.                                      */
/* FACULTAD DE INGENIERIA ELECTRICA.                                  */
/* DEPARTAMENTO DE ELECTRONICA Y CONTROL.                             */
```

```
/*                                                                    */
/* TESIS DE GRADO: Identificación de Parámetros utilizando un Algoritmo  */
/*                 Genético de Optimización, Aplicación al Circuito      */
/*                 Equivalente de la Máquina de Inducción.               */
/*                                                                    */
/* AUTOR    : A. Javier Quintana M.                                   */
/* DIRECTOR : Dr. Hugo Banda Gamboa                                   */
/*                                                                    */
/* POBINIC.C  : Módulo desarrollado para calcular la población inicial de  */
/*              grupos de parámetros del circuito equivalente de la Máquina */
/*              de Inducción, siguiendo el procedimiento recomendado por el */
/*              IEEE std-112 ( 1991 ) o el procedimiento BASICO.  El progra-*/
/*              ma lee los valores de las pruebas realizadas a la máquina en*/
/*              estudio, almacenados en un archivo, dependiendo de cual de  */
/*              cual de los procedimientos antes indicados ha sido escogido.*/
/*                                                                    */
/* CREACION : 18 - XII - 1993                                         */


/*
** ARCHIVOS DE INCLUSION.
*/

#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <ctype.h>
#include <string.h>
#include "aginclud.h"

/*
** ALUSIONES A FUNCIONES.
*/

/* Funciones definidas en el módulo MENUS.C . */

extern void salir ( void );
extern void ventana_proceso ( int color );
extern void dibujar_caja ( int xmin, int ymin, int xmax, int ymax,
                           int tipo, int colores );
extern float leer_flot ( float valor_ant, float minimo, float maximo, int
                     digitos, int xcoord, int ycoord );
extern int leer_ent ( int valor_ant, int minimo, int maximo, int digitos,
                      int xcoord, int ycoord );

/* Funciones definidas en este módulo. */

void menu ( void );
void escoger_metodo ( void );
void archivo_entrada ( void );
void evaluar_pob_inicial ( void );
void leer_datos ( FILE *fpin );
void calc_parametros_ieee ( void );
void calc_parametros_basico ( void );
void verificar_datos ( void );
void guardar_parametros ( void );
```

```
/*
** DECLARACION DE VARIABLES GLOBALES.
*/

static float dato_prueba [TAMANIO_POB][NUM_DATOS_PRUEBA_IEEE];
static float parametro [TAMANIO_POB][NUM_PARAMETROS];
extern DATOS_ENTRADA valor;


/*
** CODIGO DE LAS FUNCIONES.
*/


/*                                                                    */
/* menu () : Visualiza el menú de opciones disponibles, para el cálculo de  */
/*           la población inicial.                                    */

void menu ( void )
{
   /* Declaración de variables. */

   char pantalla_datos [4096];        /* Buffer para guardar una pantalla.  */

   /* Arreglo en que se definen las opciones del menú. */
   char opcion[2][12] = { " Calcular ",
                          " Modificar " };
   /* Arreglo en que se definen los mensajes respectivos de cada opción. */
   char mensaje[2][45] = { " Evalúar la población inicial de parámetros.",
                           " Cambiar los valores indicados en pantalla. " };

   int j;              /* Variable auxiliar. */
   int eleccion = 0;   /* Puntero de la opción seleccionada. */
   int xi = 10;        /* Posición en X de la primera opción del menú. */
   char c;             /* Variable auxiliar para leer un caracter del teclado*/


   /* Define la ventana de visualización como la pantalla completa, y selec-*/
   /* ciona el color del fondo y del texto.                              */
   clrscr(); window(1,1,80,25); textattr ( COLOR_NORMAL ); clrscr();

   /* Dibuja el borde de la ventana principal.   */
   dibujar_caja ( 1, 1, 80, 23, DOBLE, BORDE_NORMAL );

   /* Imprime encabezado.    */
   gotoxy ( 12, 6 );
   cprintf("OBTENCION DE LA POBLACION INICIAL DE GRUPOS DE PARAMETROS");
   gotoxy ( 12, 7 );
   cprintf("  .       DE LA MAQUINA DE INDUCCION TRIFASICA. ");

   /* Imprime las características de ejecución. */

   textcolor ( YELLOW );
   gotoxy ( 31,11 ); cprintf("Datos del Programa");
   textcolor ( WHITE );

   dibujar_caja ( 10, 10, 70, 17, SIMPLE, BORDE_NORMAL );
   gotoxy ( 12,12 ); cprintf("METODO  : ");
```

```
if ( valor.metodo == M_IEEE )
{
  gotoxy ( 12,14 ); cprintf("R1 = %6.4f  ohmios", valor.r1);
  gotoxy ( 22,12 ); cprintf("I");
  strcpy(valor.archivo,"M_IEEDAT.DAT");
  gotoxy ( 12,15 ); cprintf("ARCHIVO DE ENTRADA : %s",valor.archivo);
  gotoxy ( 12,16 ); cprintf("ARCHIVO DE SALIDA  : M_IEEE.DAT  ");
}

if ( valor.metodo == M_BASICO )
{
  gotoxy ( 22,12 ); cprintf("B");
  gotoxy ( 12,14 ); cprintf("                                  ");
  strcpy(valor.archivo,"M_BASDAT.DAT");
  gotoxy ( 12,15 ); cprintf("ARCHIVO DE ENTRADA : %s",valor.archivo);
  gotoxy ( 12,16 ); cprintf("ARCHIVO DE SALIDA  : M_BASICO.DAT");
}

gotoxy ( 26,12 ); cprintf("ASICO");
gotoxy ( 36,12 ); cprintf("EEE");

textcolor ( YELLOW );
gotoxy ( 25,12 ); cprintf("B");
gotoxy ( 35,12 ); cprintf("I");
textcolor ( WHITE );

gotoxy ( 12,13 ); cprintf("# de Grupos de Pruebas : %d  ",
                     valor.num_pruebas);


/* Visualiza la barra del menú con sus opciones.  */

textattr(COLOR_MENU);
gotoxy ( 1,24 );
for( j = 1; j < 81; j++ ) putch(' ');

gotoxy ( 37, 24 );
cprintf(opcion[1]);

textcolor(RED);
gotoxy ( 65,24 ); cprintf("ESC");
textcolor(BLACK);
gotoxy ( 69,24 ); cprintf("Menú Previo");
gotoxy ( 62,24 ); cprintf("|");

textattr(REVERSO_MENU);
gotoxy ( 10, 24 );
cprintf(opcion[0]);

textattr(COLOR_NORMAL);
gotoxy (  1, 25 );
cprintf(mensaje[0]);

/* Lazo de selección de una de las opciones disponibles. */

while(1)
{
  c = getch();        /* Lee un caracter del teclado. */
```

```
    /* Se permite el movimiento de una barra de selección con el TAB, a     */
    /* través de la barra del menú.                                         */

    if ( TAB == c )
    {
      ++ eleccion;
      gotoxy(xi,24);
      textattr(COLOR_MENU);
      cprintf(opcion[eleccion-1]);

      if(eleccion > 1)
      {
        eleccion = 0;
        xi   = 10;
      }
      else  xi += 27;

      gotoxy( xi,24 );
      textattr(REVERSO_MENU);
      cprintf(opcion[eleccion]);
      gotoxy(  1,25 ); textattr(COLOR_NORMAL); cprintf(mensaje[eleccion]);

    }


    /* Si una vez posicionado en la opción deseada, se presiona ENTER, se  */
    /* ejecuta(n) la(s) accion(es) respectiva(s).                           */

    if ( ENTER == c )
    {
      /* Opción Modificar. */
      if (eleccion == 1) escoger_metodo();

      /* Opción Calcular. */
      if (eleccion == 0)
      {
        gettext(1,1,80,25,pantalla_datos);   /* Guarda la pantalla actual. */
        ventana_proceso( COLOR_PROCESO );
        evaluar_pob_inicial ();
        clrscr(); window(1,1,80,25); textattr ( COLOR_NORMAL ); clrscr();
        puttext(1,1,80,25,pantalla_datos);   /* Presenta pantalla original.*/
      }
    }

    if ( ESC == c )  return;

  }

} /* Fin de Función menu (). */



/*                                                                          */
/* escoger_metodo () :  Función empleada para seleccionar el método con el  */
/*                      cual se desea obtener la población inicial de pará- */
/*                      metros a utilizarse en la evolución artificial.     */
```

```
void escoger_metodo ( void )
{
  /* Declaración de variables. */

  int flag2 = 0;
  int flag1 = 0;
  char c;

  _setcursortype ( _NORMALCURSOR ); /* Presentación del cursor. */

  /* Lazo de selección del método a emplearse. */

  do
  {
    flag2 = 0;

    /* Lee una tecla. */


    do
    {
      fflush ( stdin );
      gotoxy ( 22,12 ); c = getch();
      if ( isalpha(c) )   flag2 = 1;
      if ( ENTER == c )   flag2 = 1;
    } while ( !flag2 );

    /* Si la tecla presionada es ENTER se conserva la opción anterior. */

    if ( ENTER == c )
    {
      if ( valor.metodo == M_BASICO )
      {
        valor.metodo = M_BASICO;
        gotoxy ( 22,12 ); cprintf("B");
        gotoxy ( 12,14 ); cprintf("                          ");
        flag1 = 1;
      }
      if ( valor.metodo == M_IEEE )
      {
        valor.metodo = M_IEEE;
        gotoxy ( 22,12 ); cprintf("I");
        gotoxy ( 12,14 ); cprintf("R1 = %6.4f  ohmios", valor.r1);
        flag1 = 1;
      }
    }

    /* Lazos para seleccionar uno de los métodos, según la tecla presionada.*/

    if ( isalpha(c) )
    {
      if ( islower ( c ) )  c = toupper ( c );
      gotoxy ( 22,12 ); putch(c);
      fflush ( stdin );
      if ( c == 'B')
      {
        valor.metodo = M_BASICO;
        gotoxy ( 22,12 ); cprintf("B");
```

```
      gotoxy ( 12,14 ); cprintf("                          ");
      flag1 = 1;
    }
    if ( c == 'I')
    {
      valor.metodo = M_IEEE;
      gotoxy ( 22,12 ); cprintf("I");
      gotoxy ( 12,14 ); cprintf("R1 = %6.4f  ohmios", valor.r1);
      flag1 = 1;
    }
  }
} while ( flag1 == 0 );

/* Definición del número de pruebas realizadas en el laboratorio.  */

valor.num_pruebas = leer_ent(valor.num_pruebas,10,100,3,37,13);
gotoxy ( 12,13 ); cprintf("# de Grupos de Pruebas : %d  ",
                          valor.num_pruebas);

/* Se lee la información adicional que requiere el método IEEE.   */

if ( valor.metodo == M_IEEE )
{
  gotoxy ( 12,14 ); cprintf("R1 = ");
  valor.r1 = leer_flot ( valor.r1,0.01,9.99,6,17,14 );
  gotoxy ( 17,14 ); cprintf("%6.4f  ohmios", valor.r1);
  strcpy ( valor.archivo,"M_IEEDAT.DAT" );
  gotoxy ( 33,15 ); cprintf("%s",valor.archivo);
  archivo_entrada();
  gotoxy ( 33,15 ); cprintf("%s",valor.archivo);
  gotoxy ( 12,16 ); cprintf("ARCHIVO DE SALIDA  : M_IEEE.DAT  ");
}

/* Se lee la información adicional que requiere el método BASICO.   */

if ( valor.metodo == M_BASICO )
{
  gotoxy ( 12,14 ); cprintf("                                ");
  strcpy ( valor.archivo,"M_BASDAT.DAT" );
  gotoxy ( 33,15 ); cprintf("%s",valor.archivo);
  archivo_entrada();
  gotoxy ( 33,15 ); cprintf("%s",valor.archivo);
  gotoxy ( 12,16 ); cprintf("ARCHIVO DE SALIDA  : M_BASICO.DAT");
}
_setcursortype ( _NOCURSOR );

}  /* Fin de la Función escoger_metodo (). */



/*                                                                  */
/* archivo_entrada () : Función empleada para leer, caracter por caracter,  */
/*                      el nombre del archivo que contiene los datos de las */
/*                      que contiene los datos tomados en el laboratorio.   */
```

```
 archivo_entrada ( void )

Declaración de variables. */

r c;
 xcoord = 33;
 ycoord = 15;
 flag1  = 0;
 j;

Lectura de la primera letra.Si se presiona ENTER se devuelve el valor*/
original de la variable.                                              */


fflush ( stdin );
gotoxy ( xcoord, ycoord );
c = getch();
if ( c == ENTER )  return;
while ( !isalpha(c) );

toxy ( xcoord, ycoord );
( islower(c) ) c = toupper(c);
tch(c);
toxy ( xcoord + 1, ycoord ); cprintf("                    ");
lor.archivo[0] = c;

* Lazo de lectura de las demás letras. */
or ( j = 1; j < 21; j++ )

do
{
   fflush ( stdin ); flag1 = 0;
   gotoxy ( xcoord + j, ycoord );
   c = getch();
   if ( isalpha(c) ) flag1 = 1;
   if ( c == '_' )   flag1 = 1;
   if ( c == '.' )   flag1 = 1;
   if ( ENTER == c ) flag1 = 1;
} while ( flag1 == 0 );

if ( ENTER == c ) break;

gotoxy ( xcoord + j, ycoord );
if ( islower(c) ) c = toupper(c);
putch(c);
valor.archivo[j] = c;

alor.archivo[j] = '\0';

eturn;
       /* Fin de la Función archivo_entrada() */


                                                                      */
evaluar_pob_inicial () :  Función empleada para desarrollar todo el pro- */
                          ceso a seguirse, para obtener la población ini-*/
                          cial, con cualesquiera de los métodos propues- */
                          tos.                                          */
```

```
void evaluar_pob_inicial ( void )
{
  FILE *fpin;

  /* Apertura del archivo que tiene los valores de las pruebas.            */
  /* Si existe algún error, se imprime un mensaje en pantalla y se da por  */
  /* terminada la ejecución del programa.                                  */
  gotoxy ( 11,1 ); cprintf("OBTENCION DE LA POBLACION INICIAL");

  if ((fpin = fopen( valor.archivo,"r")) == NULL )
  {
    putch ('\a');
    gotoxy( 2, 2 );
    cprintf("No se puede abrir archivo de entrada %s",valor.archivo);
    gotoxy ( 2, 3 );
    cprintf("Presione una tecla para salir....");
    getch();
    return;
  }

  /* Proceso a seguirse cuando el método seleccionado es el IEEE */

  if ( valor.metodo == M_IEEE )
  {
    gotoxy ( 2,2 ); cprintf("Leyendo los valores de las pruebas...");
    leer_datos ( fpin );
    fclose ( fpin );
    verificar_datos ();
    gotoxy ( 2,3 ); cprintf("Obteniendo la población inicial...");
    calc_parametros_ieee ();
  }

  /* Proceso a seguirse cuando el método seleccionado es el BASICO */

  if ( valor.metodo == M_BASICO )
  {
    gotoxy ( 2,2 ); cprintf("Leyendo los valores de las pruebas...");
    leer_datos ( fpin );
    fclose ( fpin );
    verificar_datos ();
    gotoxy ( 2,3 ); cprintf("Obteniendo la población inicial...");
    calc_parametros_basico ();
  }

  gotoxy ( 2,4 ); cprintf("Almacenando la población inicial...");
  guardar_parametros ();
  putch('\a');
  gotoxy ( 2,5 ); cprintf("Presione una tecla para continuar...");
  getch();
  return;
}       /* Fin de la Función evaluar_pob_inicial () */



/*                                                                       */
/* leer_datos ()  : Función utilizada para leer los datos del archivo de */
/*                  entrada " fpin ", que contiene los datos de las pruebas */
/*                  realizadas en la máquina.                            */
```

```
void leer_datos ( FILE *fpin )
{

/* Declaración del tipo de variables */

  int j;

/* Entrada de datos de las pruebas realizadas */

  for ( j = 0; j < valor.num_pruebas; j++ )
  {

/* Entrada de datos de prueba de vacio */

    fscanf (fpin,"%f",&dato_prueba [j][0]);
    fscanf (fpin,"%f",&dato_prueba [j][1]);
    fscanf (fpin,"%f",&dato_prueba [j][2]);

/* Entrada de datos de prueba de rotor bloqueado */

    fscanf (fpin,"%f",&dato_prueba [j][3]);
    fscanf (fpin,"%f",&dato_prueba [j][4]);
    fscanf (fpin,"%f",&dato_prueba [j][5]);

/* Entrada de datos de prueba de vacio, a deslizamiento de plena carga.     */
/* Estos datos se leen únicamente cuando se emplea el método IEEE.          */

    if ( valor.metodo == M_IEEE )
    {
      fscanf (fpin,"%f",&dato_prueba [j][6]);
      fscanf (fpin,"%f",&dato_prueba [j][7]);
      fscanf (fpin,"%f",&dato_prueba [j][8]);
      fscanf (fpin,"%f",&dato_prueba [j][9]);
    }
  }

  return;
}       /* Fin de la Función leer_datos () */




/*                                                                          */
/* calc_parametros_ieee () : Función empleada para evaluar los parámetros   */
/*                           del circuito equivalente de un motor de induc- */
/*                           ción trifásico, según el IEEE.                 */


void calc_parametros_ieee ( void )
{

/* Definición del tipo de variables */

  float x1_x2 = 1.0;            /* relación para rotor bobinado */
  float x1,xm,x1_xm,x1_xm_aux; /* valores de reactancias */
  float r2,rfe;                 /* valores de resistencias */
```

```
  float vo,io,po,varo;          /* valores para prueba de vacio */
  float vb,ib,pb,varb;          /* valores para prueba de rotor bloqueado */
  float v1,i1,p1,nr;            /* valores para prueba de vacío especial  */
  float v2,i2,i_21,i_22;        /* variables auxiliares */
  float z2,z_11,z_12,theta_1,theta_2;
  float ife,ie,s,wh;
  int j;
  int m = 3;                    /* Número de fases */

/* Evaluación de los parámetros del circuito equivalente */

  for ( j = 0; j < valor.num_pruebas; j++ )
  {

/* Asignación de valores */

    vo =  dato_prueba [j][0];
    io =  dato_prueba [j][1];
    po =  dato_prueba [j][2];
    vb =  dato_prueba [j][3];
    ib =  dato_prueba [j][4];
    pb =  dato_prueba [j][5];
    v1 =  dato_prueba [j][6];
    i1 =  dato_prueba [j][7];
    p1 =  dato_prueba [j][8];
    nr =  dato_prueba [j][9];

/* Definición de los valores iniciales de x1 y x1_xm */

    x1 = 1;
    x1_xm = 0.1;

/* Acondicionamiento de valores para equivalente monofásico */
    vo = vo/sqrt(3.0);
    vb = vb/sqrt(3.0);
    v1 = v1/sqrt(3.0);
    s = (1800.0 - nr)/1800.0;

/* Cálculo de potencias reactivas de vacio y de rotor bloqueado */

    varo = sqrt(pow((m*vo*io),2)-po*po);
    varb = sqrt(pow((m*vb*ib),2)-pb*pb);

/* Lazo de cálculo de x1 y xm */

    do
    {
      x1_xm_aux = x1_xm;
      xm = (m*vo*vo)/((varo-m*io*io*x1)*pow((1+x1_xm),2));
      x1 = (varb*(x1_x2+x1_xm))/(m*ib*ib*(1+x1_x2+x1_xm));
      x1_xm = x1/xm;
    }
    while (x1_xm_aux - x1_xm > 0.0001);

/* Cálculo de rfe */

    wh = po-m*io*io*valor.r1;
    rfe = (m*vo*vo)/(wh*pow((1+x1_xm),2));
```

```
/* Cálculo de r2 */

    theta_1 =  acos(p1/(m*v1*i1));
    z_11 =  x1*sin(theta_1) + valor.r1*cos(theta_1);
    z_12 =  x1*cos(theta_1) - valor.r1*sin(theta_1);
    v2 =  sqrt(pow((v1 - i1*z_11),2) + pow ((i1*z_12),2));
    theta_2 =  atan((i1*z_12)/(v1 - i1*z_11));
    wh =  p1-m*i1*i1*valor.r1;
    ife =  wh/(m*v2);
    ie =  v2/xm;
    i_21 =  i1*cos(theta_2)+ie*sin(theta_2)-ife*cos(theta_2);
    i_22 =  i1*sin(theta_2)-ie*cos(theta_2)+ife*sin(theta_2);
    i2 =  sqrt(i_21*i_21 + i_22*i_22);
    z2 =  v2/i2;
    r2 =  s*sqrt(z2*z2 - x1*x1);

/* Asignación de valores al arreglo correspondiente */

    parametro [j][0] = r2;
    parametro [j][1] = x1;
    parametro [j][2] = xm;
    parametro [j][3] = rfe;

  }

  return;
}       /* Fin de la Función calc_parametros_ieee () */



/*                                                                    */
/* calc_parametros_basico () : Función empleada para evaluar los parámetros */
/*                             del circuito equivalente del motor de induc- */
/*                             ción trifásico, según el método BASICO.      */


void calc_parametros_basico ( void )
{

/* Definición del tipo de variables */

  float x1,xm;          /* valores de reactancias */
  float r2,rfe;         /* valores de resistencias */
  float vo,io,po,yo;    /* valores para prueba de vacio */
  float vb,ib,pb,z;     /* valores para prueba de rotor bloqueado */
  int j;

/* Evaluación de los parámetros del circuito equivalente */

  for ( j = 0; j < valor.num_pruebas; j++ )
  {

/* Asignación de valores */

    vo = dato_prueba [j][0];
    io = dato_prueba [j][1];
    po = dato_prueba [j][2];
```

```
    vb = dato_prueba [j][3];
    ib = dato_prueba [j][4];
    pb = dato_prueba [j][5];

/* Acondicionamiento de valores para equivalente monofásico */

    vo = vo/sqrt(3.0);
    vb = vb/sqrt(3.0);

/* Cálculo de rfe y xm */

    rfe = po/(3.0*pow(vo,2));
    yo  = io/vo;
    xm  = sqrt(yo*yo - rfe*rfe);
    rfe = 1/rfe;
    xm  = 1/xm;

/* Cálculo de r2 ( = r1 ) y x1 ( = x2 ) */

    r2 = pb/(3.0*pow(ib,2));
    z  = vb/ib;
    x1 = sqrt(z*z - r2*r2);
    r2 = r2/2.0;
    x1 = x1/2.0;

/* Asignación de valores al arreglo correspondiente */

    parametro [j][0] = r2;
    parametro [j][1] = x1;
    parametro [j][2] = xm;
    parametro [j][3] = rfe;

  }

  return;
}       /* Fin de la Función calc_parametros_basico () */



/*                                                                    */
/* verificar_datos () : Función utilizada para comprobar el ingreso de datos*/
/*                      adecuados, mayores o iguales a cero.          */


void verificar_datos ( void )
{

/* Declaración del tipo de variables */

  int i,j;
  int num_datos_prueba;

  if ( valor.metodo == M_BASICO )
    num_datos_prueba = NUM_DATOS_PRUEBA_BASICO;

  if ( valor.metodo == M_IEEE )
    num_datos_prueba = NUM_DATOS_PRUEBA_IEEE;
```

```
/* Verificación de datos de las pruebas */

  for ( j = 0; j < valor.num_pruebas; j++ )
  {
    for ( i = 0; i < num_datos_prueba; i++ )
    {
      if ( dato_prueba [j][i] <= 0.0 )
      {
        putch('\a');
        gotoxy ( 2,3 ); cprintf("Datos de las pruebas inconsistentes....");
        gotoxy ( 2,4 ); cprintf("Presione una tecla para salir...");
        getch();
        salir();
      }
    }
  }
  return;
}         /* Fin de Función verificar_datos () */


/*                                                                    */
/* guardar_parametros () : Función para almacenar los valores de los pará- */
/*                         metros calculados, que forman la población ini- */
/*                         cial, en un archivo " fpout ".             */

void guardar_parametros ( void )
{
/* Declaración del tipo de variables */
  FILE *fpout;
  int j;

  if ( valor.metodo == M_BASICO )
  {
    if ((fpout = fopen("M_BASICO.DAT","w")) == NULL )
    {
      putch ('\a');
      gotoxy( 2, 4 );
      cprintf("No se puede abrir archivo de salida M_BASICO.DAT");
      gotoxy ( 2, 5 );
      cprintf("Presione una tecla para salir....");
      getch();
      salir();
    }
  }
  if ( valor.metodo == M_IEEE )
  {
    if ((fpout = fopen("M_IEEE.DAT","w")) == NULL )
    {
      putch ('\a');
      gotoxy( 2, 4 );
      cprintf("No se puede abrir archivo de salida M_IEEE.DAT");
      gotoxy ( 2, 5 );
      cprintf("Presione una tecla para salir....");
      getch();
      salir();
    }
  }
```

```
/* Impresión de los valores calculados de los parámetros en el archivo de
   salida */
  for ( j = 0; j < valor.num_pruebas; j++ )
  {
    fprintf (fpout,"%4.4f\t\t%4.4f\t\t%4.4f\t\t%4.4f\n",
                parametro [j][0],
                parametro [j][1],
                parametro [j][2],
                parametro [j][3]);
  }
  fclose ( fpout );
  return;
}       /* Fin de Función guardar_parametros () */
```

C.8. MODULO REPORTE.C.

```
/*                                                                      */
/* ESCUELA POLITECNICA NACIONAL.                                        */
/* FACULTAD DE INGENIERIA ELECTRICA.                                    */
/* DEPARTAMENTO DE ELECTRONICA Y CONTROL.                               */
/*                                                                      */
/* TESIS DE GRADO: Identificación de Parámetros utilizando un Algoritmo */
/*                 Genético de Optimización, Aplicación al Circuito     */
/*                 Equivalente de la Máquina de Inducción.              */
/*                                                                      */
/* AUTOR    : A. Javier Quintana M.                                     */
/* DIRECTOR : Dr. Hugo Banda Gamboa                                     */
/*                                                                      */
/* REPORTE.C : Módulo que contiene las funciones necesrias para elaborar los*/
/*             reportes de los resultados obtenidos en el proceso de evo-   */
/*             lución artificial, y su almacenamiento en memoria y en disco.*/
/*                                                                      */
/* CREACION : 2 - XII - 1993.                                           */


/*
** ARCHIVOS DE INCLUSION.
*/

#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <time.h>
#include "aginclud.h"

/*
** ALUSIONES A FUNCIONES.
*/

void reportes ( void );
void reporte_1 ( FILE *fpout );
void guardar_resultados ( int generacion );
void almacenar ( FILE *fpout, int generacion );
```

```
/* Función definida y declarada en el módulo GENERAC.C . */
extern buscar_aptitud ( POBLACION pob[], float valor );

/* Función definida y declarada en el módulo MENUS.C . */
extern void salir ( void );


/*
** DECLARACION DE VARIABLES GLOBALES.
*/

RESULTADOS resultado[MAX_GEN];/* Estructura en la cual se almacenan los re- */
                              /* sultados de la evolución artificial.  Se   */
                              /* define en el archivo AGINCLUD.H .          */
extern ESTADISTICAS indicador;/* Estructura declarada en AGEVOLUC.C, y en la*/
                              /* se almacenan los índices estadísticos de la*/
                              /* población en uso.                          */
extern DATOS_ENTRADA valor;   /* Estructura declarada en AG.C, y en la cual */
                              /* se hallan almacenados los valores de todas */
                              /* las variables empledas en el programa.     */

/*
** CODIGO DE LAS FUNCIONES.
*/


/*                                                                          */
/* guardar_resultados () : Función que se emplea para almacenar en memoria, */
/*                         especificamente en la estructura "resultado",las */
/*                         características principales del grupo de paráme- */
/*                         tros óptimos de cada generación.                 */
/*                         Parámetros :                                     */
/*                       - generacion --> Generación cuyos resultados serán */
/*                                        almacenados.                      */
/*                         Esta función se utiliza en el módulo AG.C .      */

void guardar_resultados ( int generacion )
{
  /* Declaración de variables. */

  extern POBLACION pob_vieja[TAMANIO_POB]; /* Estructura declarada en AG.C, */
                                           /* de la cual se van a almacenar */
                                           /* las características del grupo */
                                           /* de parámetros óptimos.        */
  int maximo; /* Variable auxiliar que define el grupo de parámetros óptimos*/
              /* de la población en uso.                                     */

  /* Indicador del grupo de parámetros óptimos de la población en esta ge-  */
  /* neración.                                                               */
  maximo = buscar_aptitud ( pob_vieja, indicador.aptitud_maxima );


  /* *****  *****  *****  Almacenamiento de resultados  *****  *****  ***** */


  /* Guarda el número de la generación actual.  */
  resultado[generacion].generacion = generacion;
```

```
  /* Guarda los indices estadisticos de la población en esta generación.    */
  resultado[generacion].aptitud.minimo    = indicador.aptitud_minima;
  resultado[generacion].aptitud.promedio  = indicador.aptitud_promedio;
  resultado[generacion].aptitud.maximo    = indicador.aptitud_maxima;
  resultado[generacion].aptitud.sumatorio = indicador.aptitud_sumatoria;

  /* Guarda el error minimo de la población en esta generación.             */
  resultado[generacion].error_minimo = 1/indicador.aptitud_maxima;

  /* Guarda los parámetros óptimos del circuito equivalente de la máquina de*/
  /* inducción, de la generación actual.                                    */
  resultado[generacion].parametro.r1  = pob_vieja[maximo].r1;
  resultado[generacion].parametro.r2  = pob_vieja[maximo].r2;
  resultado[generacion].parametro.x1  = pob_vieja[maximo].x1;
  resultado[generacion].parametro.xm  = pob_vieja[maximo].xm;
  resultado[generacion].parametro.rfe = pob_vieja[maximo].rfe;

  /* Almacena los valores de impedancia y torque,calculados a partir de los */
  /* parámetros óptimos de la población, en la actual generación.           */
  resultado[generacion].impedancia.real = pob_vieja[maximo].z_real;
  resultado[generacion].impedancia.imag = pob_vieja[maximo].z_imag;
  resultado[generacion].torque          = pob_vieja[maximo].torque;

  /* Almacena el número de mutaciones y de cruces que ocurrieron es la ge-  */
  /* neración actual.                                                       */
  resultado[generacion].n_mutaciones = indicador.n_mutaciones;
  resultado[generacion].n_cruces     = indicador.n_cruces;

  return;
}       /* Fin de la Función guardar_resultados() */




/*                                                                          */
/* reportes () : Esta función almacena en disco, dos tipos de reportes:     */
/*               - AGREPOR1.PRN, que contiene las características del mejor */
/*                 grupo de parámetros de la población que se obtuvo en la  */
/*                 primera y en la última generación.                       */
/*               - AGREPOR2.PRN, que contiene las características del mejor */
/*                 grupo de parámetros de la población que se obtuvo en cada*/
/*                 una de las generaciones realizadas.                      */

void reportes ( void )
{
  /* Declaración de variables. */

  FILE *fpout;   /* Puntero al archivo del salida. */
  int j, limite; /* Variables auxiliares. */

  /* Apertura del archivo AGREPOR1.PRN, si existe algún error se imprime un */
  /* mensaje en pantalla y se termina la ejecución del programa.            */
  if ( ( fpout = fopen("AGREPOR1.PRN","w") ) == NULL )
  {
    putch ('\a');
    gotoxy ( 2, 4 );
    cprintf("No se puede abrir archivo de entrada AGREPOR1.PRN");
```

```
    gotoxy ( 2, 5 );
    cprintf("Presione una tecla para salir....");
    getch();
    salir();
  }
  fflush(stdin); /* Limpia el buffer de entrada. */

  /* Elabora reporte AGREPOR1.PRN, listo para impresión. */
  reporte_1 ( fpout );

  fclose ( fpout ); /* Cierra el archivo AGREPOR1.PRN . */

  /* Apertura del archivo AGREPOR2.PRN, si existe algún error se imprime un */
  /* mensaje en pantalla y se termina la ejecución del programa.            */
  if ( ( fpout = fopen("AGREPOR2.PRN","w") ) == NULL )
  {
    putch ('\a');
    gotoxy ( 2, 4 );
    cprintf("No se puede abrir archivo de entrada AGREPOR2.PRN");
    gotoxy ( 2, 5 );
    cprintf("Presione una tecla para salir....");
    getch();
    salir();
  }
  fflush(stdin); /* Limpia el buffer de entrada. */

  limite = valor.ultima_generacion+1; /* Límite del número de generaciones.*/

  /* Almacena las características del mejor grupo de parámetros obtenidos  */
  /* en la cada una de las generaciones.                                    */
  for ( j = 0; j < limite; j++ )
    almacenar ( fpout, j );

  fclose ( fpout ); /* Cierra el archivo AGREPOR2.PRN . */

  return;
}       /* Fin de la Función reportes () */



/*                                                                        */
/* reporte_1 () : Función que elabora un reporte listo para imprimir.  En */
/*                este reporte se incluyen los valores de las variables u-*/
/*                tilizadas en este proceso, así como las condiciones ini-*/
/*                ciales de la población y los resultados obtenidos después*/
/*                del proceso de Evolución Artificial.                    */
/*                Parámetros:                                             */
/*              - *fpout  --> Puntero al archivo de salida.               */
/*              - Esta función se emplea en la función reportes () .      */

void reporte_1 ( FILE *fpout )
{
  time_t tiempo;
  int max;

  tiempo = time ( NULL );
  max = valor.ultima_generacion;
```

```
fprintf ( fpout,"\n\n" );
fprintf ( fpout,
"                                                                pag. 1 de 2");
fprintf ( fpout,"\n\n" );

fprintf ( fpout,"\n\t\t    PROCESO DE EVOLUCION ARTIFICIAL" );
fprintf ( fpout,"\n\t\t         REPORTE DE RESULTADOS     " );
fprintf ( fpout,"\n\n\nELABORACION : %s\n\n\n", ctime ( &tiempo ) );

fprintf ( fpout,
"---------------------- CONDICIONES DE EJECUCION -----------------------");

if ( valor.metodo == M_BASICO )
  fprintf ( fpout,"\n\n\nMETODO UTILIZADO :  BASICO\n\n\n");

if ( valor.metodo == M_IEEE )
{
  fprintf ( fpout,"\n\n\nMETODO UTILIZADO :  IEEE");
  fprintf ( fpout,"\nR1 = %7.4f ohmios\n\n", valor.r1 );
}

fprintf (fpout,"\nNUMERO DE GENERACIONES   : %d", valor.ultima_generacion );
fprintf (fpout,"\nPROBABILIDAD DE CRUCE    : %5.3f", valor.p_cruce );
fprintf (fpout,"\nPROBABILIDAD DE MUTACION : %5.3f\n\n", valor.p_mutacion );

fprintf (fpout,"\nPRUEBA DE CARGA" );
fprintf (fpout,"\n---------------" );
fprintf (fpout,"\nDESLIZAMIENTO      = %6.4f ( pu )", valor.deslizamiento );
fprintf (fpout,"\nTORQUE EN EL EJE   = %5.2f  ( Nm )", valor.torque );
fprintf (fpout,"\nVOLTAJE DE FASE    = %6.2f ( V )", valor.voltaje );
fprintf (fpout,"\nCORRIENTE DE FASE  = %5.2f  ( A )", valor.corriente );
fprintf (fpout,"\nPOTENCIA ACTIVA 3f = %6.1f ( W )\n\n", valor.potencia );

fprintf ( fpout,"\nCARACTERISTICAS DE COMPARACION" );
fprintf ( fpout,"\n-----------------------------" );
fprintf ( fpout,"\nIMPEDANCIA REAL  = %7.4f ( ohmios )", valor.z_real );
fprintf ( fpout,"\nIMPEDANCIA IMAG  = %7.4f ( ohmios )", valor.z_imag );
fprintf ( fpout,"\nTORQUE EN EL EJE = %7.4f ( Nm )\n\n", valor.torque );

fprintf ( fpout,"\nRANGO PERMITIDO DE VARIACION DE LOS PARAMETROS" );
fprintf ( fpout,"\n--------------------------------------------" );
fprintf ( fpout,"\nR2 max  = %6.2f ( ohmios )", valor.r2_max );
fprintf ( fpout,"\nR2 min  = %6.2f ( ohmios )", valor.r2_min );
fprintf ( fpout,"\nX1 max  = %6.2f ( ohmios )", valor.x1_max );
fprintf ( fpout,"\nX1 min  = %6.2f ( ohmios )", valor.x1_min );
fprintf ( fpout,"\nXm max  = %6.2f ( ohmios )", valor.xm_max );
fprintf ( fpout,"\nXm min  = %6.2f ( ohmios )", valor.xm_min );
fprintf ( fpout,"\nRfe max = %6.2f ( ohmios )", valor.rfe_max );
fprintf ( fpout,"\nRfe min = %6.2f ( ohmios )\n\n", valor.rfe_min );

fprintf ( fpout,"\nCONSTANTES CARACTERISTICAS DE LA FUNCION DE ERROR" );
fprintf ( fpout,"\n----------------------------------------------" );
fprintf ( fpout,"\nK1        = %6.4f", valor.k1 );
fprintf ( fpout,"\nK2        = %6.4f", valor.k2 );
fprintf ( fpout,"\nK3        = %6.4f", valor.k3 );
fprintf ( fpout,"\nExponente = %d\n\n", valor.m );

fprintf ( fpout,"\n\n\n\n\n\n\n\n" ); /* Completa 1ra página */
```

```
  fprintf ( fpout,"\n\n\n");        /* Comienza 2da página */
  fprintf ( fpout,
  "                                                    pag. 2 de 2");
  fprintf ( fpout,"\n\n" );

  fprintf ( fpout,
  "--------------------------  RESULTADOS  ------------------------------");

  fprintf ( fpout,"\n\n\n\n\n" );

  fprintf ( fpout,
  "       PARAMETRO              COND. INICIAL            RESULTANTE       ");

  fprintf ( fpout,"\n" );

  fprintf ( fpout,
  "-------------------------------------------------------------------------");

  fprintf ( fpout,"\n" );

  fprintf ( fpout,"\nAPTITUD MINIMA    =\t\t%-12g\t\t%-12g",
        resultado[0].aptitud.minimo, resultado[max].aptitud.minimo );
  fprintf ( fpout,"\nAPTITUD PROMEDIO  =\t\t%-12g\t\t%-12g",
        resultado[0].aptitud.promedio, resultado[max].aptitud.promedio );
  fprintf ( fpout,"\nAPTITUD MAXIMA    =\t\t%-12g\t\t%-12g",
        resultado[0].aptitud.maximo, resultado[max].aptitud.maximo );
  fprintf ( fpout,"\nSUMA DE APTITUDES =\t\t%-12g\t\t%-12g\n",
        resultado[0].aptitud.sumatorio, resultado[max].aptitud.sumatorio );

  fprintf ( fpout,"\nERROR MINIMO      =\t\t%-12g\t\t%-12g\n",
        resultado[0].error_minimo, resultado[max].error_minimo );

  fprintf ( fpout,"\nR1    ( ohmios )  =\t\t%7.4f\t\t\t%7.4f",
        resultado[0].parametro.r1, resultado[max].parametro.r1 );
  fprintf ( fpout,"\nR2    ( ohmios )  =\t\t%7.4f\t\t\t%7.4f",
        resultado[0].parametro.r2, resultado[max].parametro.r2 );
  fprintf ( fpout,"\nX1=X2 ( ohmios )  =\t\t%7.4f\t\t\t%7.4f",
        resultado[0].parametro.x1, resultado[max].parametro.x1 );
  fprintf ( fpout,"\nXm    ( ohmios )  =\t\t%7.4f\t\t\t%7.4f",
        resultado[0].parametro.xm, resultado[max].parametro.xm );
  fprintf ( fpout,"\nRfe   ( ohmios )  =\t\t%8.4f\t\t%8.4f\n",
        resultado[0].parametro.rfe, resultado[max].parametro.rfe );

  fprintf ( fpout,"\nIMPEDANCIA REAL ( ohmios ) =\t%7.4f\t\t\t%7.4f",
        resultado[0].impedancia.real, resultado[max].impedancia.real );
  fprintf ( fpout,"\nIMPEDANCIA IMAG ( ohmios ) =\t%7.4f\t\t\t%7.4f",
        resultado[0].impedancia.imag, resultado[max].impedancia.imag );
  fprintf ( fpout,"\nTORQUE EN EL EJE ( Nm )    =\t%5.2f\t\t\t%5.2f\n",
        resultado[0].torque, resultado[max].torque );

  fprintf ( fpout,"\nNUMERO DE MUTACIONES   =\t%ld\t\t\t%ld",
        resultado[0].n_mutaciones, resultado[max].n_mutaciones );
  fprintf ( fpout,"\nNUMERO DE CRUCES       =\t%ld\t\t\t%ld\n",
        resultado[0].n_cruces, resultado[max].n_cruces );

  fprintf ( fpout,"\n\nFIN DEL REPORTE." );
  return;
}       /* Fin de la Función reporte_1 () */
```

```
/*                                                                    */
/* almacenar () : Esta función guarda en un archivo en disco las caracte-  */
/*                ticas del grupo de parámetros óptimos de la población en  */
/*                una generación específica.                          */
/*                Parámetros:                                         */
/*              - *fpout     --> Puntero al archivo de salida.        */
/*              - generacion --> Generación cuyos resultados se desean al-  */
/*                               macenar.                             */
/*                Esta función se emplea en la función reportes () .  */

void almacenar ( FILE *fpout, int generacion )
{

  fprintf ( fpout,"\n" );

  /* Guarda en un archivo el número de la generación actual.  */
  fprintf ( fpout,"%d",resultado[generacion].generacion );

  /* Guarda los índices estadísticos de la población en esta generación.    */
  fprintf ( fpout,"\t%f",resultado[generacion].aptitud.minimo );
  fprintf ( fpout,"\t%f",resultado[generacion].aptitud.promedio );
  fprintf ( fpout,"\t%f",resultado[generacion].aptitud.maximo );
  fprintf ( fpout,"\t%f",resultado[generacion].aptitud.sumatorio );

  /* Guarda el error mínimo de la población en esta generación.          */
  fprintf ( fpout,"\t%15.10f",resultado[generacion].error_minimo );


  /* Guarda los parámetros óptimos del circuito equivalente de la máquina de*/
  /* inducción, de la generación actual.                                 */
  fprintf ( fpout,"\t%f",resultado[generacion].parametro.r1 );
  fprintf ( fpout,"\t%f",resultado[generacion].parametro.r2 );
  fprintf ( fpout,"\t%f",resultado[generacion].parametro.x1 );
  fprintf ( fpout,"\t%f",resultado[generacion].parametro.xm );
  fprintf ( fpout,"\t%f",resultado[generacion].parametro.rfe );

  /* Almacena los valores de impedancia y torque,calculados a partir de los */
  /* parámetros óptimos de la población, en la actual generación.        */
  fprintf ( fpout,"\t%f",resultado[generacion].impedancia.real );
  fprintf ( fpout,"\t%f",resultado[generacion].impedancia.imag );
  fprintf ( fpout,"\t%f",resultado[generacion].torque );

  /* Almacena el número de mutaciones y de cruces que ocurrieron es la ge-  */
  /* neración actual.                                                    */
  fprintf ( fpout,"\t%ld",resultado[generacion].n_cruces );
  fprintf ( fpout,"\t%ld",resultado[generacion].n_mutaciones );

  return;
}       /* Fin de la Función almacenar () */
```

C.9. MODULO GRAFICOS.C.

```
/*                                                                    */
/* ESCUELA POLITECNICA NACIONAL.                                      */
/* FACULTAD DE INGENIERIA ELECTRICA.                                  */
/* DEPARTAMENTO DE ELECTRONICA Y CONTROL.                             */
/*                                                                    */
/* TESIS DE GRADO: Identificación de Parámetros utilizando un Algoritmo */
/*                 Genético de Optimización, Aplicación al Circuito   */
/*                 Equivalente de la Máquina de Inducción.            */
/*                                                                    */
/* AUTOR    : A. Javier Quintana M.                                   */
/* DIRECTOR : Dr. Hugo Banda Gamboa                                   */
/*                                                                    */
/* GRAFICOS.C  : Este módulo contiene las funciones necesarias para graficar*/
/*               la forma en la cual va evolucionando el grupo de parámetros*/
/*               óptimos de la M.I., en cada una de las generaciones desa- */
/*               rrolladas durante el proceso de Evolución Artificial.    */
/*                                                                    */
/* CREACION : 12 - XII - 1993                                         */

/*
** ARCHIVOS DE INCLUSION.
*/

#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "aginclud.h"

/*
** ALUSIONES A FUNCIONES.
*/

/* Función definida en el módulo MENUS.C . */

extern void ventana_proceso ( int color );

/* Funciones definidas en este módulo. */

void graficar ( void );
int  inicializar(void);
int  pausa(void);
void ventana_principal( char *encabezado );
void linea_de_estado( char *msj );
void dibujar_borde( int ancho_de_linea);
void graficar_parametro( char *msj, float *parametro );
int  gprintf ( float x, float y, char *form, ... );

/*
** DECLARACION DE VARIABLES GLOBALES.
*/
```

```
int     MaxX, MaxY;        /* La máxima resolución de la pantalla.           */
int     maxx_v, maxy_v;  /* Esquina inferior derecha de la ventana principal.*/
int     minx_v, miny_v;  /* Esquina superior izquierda de la misma ventana.  */
int     medx_v, medy_v;  /* Coordenadas del punto medio de dicha ventana.    */
int     alto;            /* Alto en pixels de una letra con el font actual.  */


/*
** CODIGO DE LAS FUNCIONES.
*/


/*                                                                           */
/* graficar ( ) : Función principal de este módulo, utilizada para visuali- */
/*                zar, de manera gráfica, la manera en la cual va evolucio- */
/*                nando, cada uno de los parámetros del circuito equivalente*/
/*                de la máquina de inducción.                                */

void graficar( void )
{
  /* Declaración de variables. */

  extern DATOS_ENTRADA valor;          /* Estructura declarada en AG.C .    */
  extern RESULTADOS resultado[MAX_GEN];/* Estructura declarada en REPORTES.C*/
  float r2[MAX_GEN], x1[MAX_GEN];      /* Arreglos utilizados para almacenar*/
  float xm[MAX_GEN], rfe[MAX_GEN];     /* el grupo de parámetros óptimos de */
  float aptitud_maxima[MAX_GEN];       /* cada una de las generaciones.     */
  int error_grafico; /* Indicador de la ocurrencia de un error durante la   */
                     /* inicialización del modo gráfico.                    */
  int ancho;         /* Ancho de una letra en pixels, para el font actual.  */
  int opcion;        /* Opción escogida durante la pausa en la graficación. */
  int j;             /* Variable auxiliar.                                  */


  /* Establece a la pantalla en el modo gráfico, si existe algún error du-  */
  /* rante la inicialización, se regresa al menú desde el cual se llamó a   */
  /* función.                                                               */
  error_grafico = inicializar();
  if ( error_grafico == -1 ) return;

  /* Almacenamiento de los valores de los parámetros óptimos de cada gene-  */
  /* ración, en el arreglo correspondiente.                                 */
  for ( j = 0; j < ( valor.ultima_generacion + 1 ); j++ )
  {
    r2[j]  = resultado[j].parametro.r2;
    x1[j]  = resultado[j].parametro.x1;
    xm[j]  = resultado[j].parametro.xm;
    rfe[j] = resultado[j].parametro.rfe;
    aptitud_maxima[j] = log( resultado[j].aptitud.maximo );
  }

  /* Determinación del alto y ancho de una letra, en pixels, para el font   */
  /* actual.                                                                */
  alto  = textheight("H");
  ancho = alto;

  /* Lazo de graficación.  */
```

```
while ( 1 )
{
  cleardevice();   /* Borra la pantalla. */

  /* Visualiza encabezado de la pantalla.  */

  ventana_principal ( "PROCESO DE EVOLUCION ARTIFICIAL" );

  /* Grafica R2. */
  setviewport ( minx_v + ancho, miny_v + alto,
                medx_v - ancho, medy_v - alto, 1 );
  dibujar_borde ( NORM_WIDTH );
  graficar_parametro( "R2 vs. # de generacion", &r2[0]);

  /* Grafica X1 ( = X2 ). */
  setviewport ( medx_v + ancho, miny_v + alto,
                maxx_v - ancho, medy_v - alto, 1 );
  dibujar_borde ( NORM_WIDTH );
  graficar_parametro( "X1 = X2  vs. # de generacion", &x1[0]);

  /* Grafica Xm.  */
  setviewport ( minx_v + ancho, medy_v + alto,
                medx_v - ancho, maxy_v - alto, 1 );
  dibujar_borde ( NORM_WIDTH );
  graficar_parametro( "Xm vs. # de generacion", &xm[0]);

  /* Grafica Rfe. */
  setviewport ( medx_v + ancho, medy_v + alto,
                maxx_v - ancho, maxy_v - alto, 1 );
  dibujar_borde ( NORM_WIDTH );
  graficar_parametro( "Rfe vs. # de generacion", &rfe[0]);

  /* Visualiza la linea de estado. */
  linea_de_estado(
  "ESC para terminar o presione una tecla para continuar...");

  /* Hace una pausa, si durante ella se presiona la tecla ESC, se regresa */
  /* al menu anterior, de lo contrario sigue con la graficación.          */
  opcion = pausa();
  if ( opcion == 1 )  return;

  cleardevice();   /* Borra la pantalla. */

  /* Visualiza encabezado de la pantalla.  */

  ventana_principal ( "PROCESO DE EVOLUCION ARTIFICIAL" );

  /* Grafica el logaritmo natural de la aptitud. */
  setviewport ( minx_v + 10*ancho, miny_v + 10*alto,
                maxx_v - 10*ancho, maxy_v - 10*alto, 1 );
  dibujar_borde ( NORM_WIDTH );
  graficar_parametro( "Ln ( Aptitud )  vs. # de generacion",
                      &aptitud_maxima[0] );

  /* Visualiza la linea de estado. */
  linea_de_estado(
  "ESC para terminar o presione una tecla para ver pantalla anterior...");
```

```
    /* Hace una pausa, si durante ella se presiona la tecla ESC, se regresa */
    /* al menu anterior, de lo contrario sigue con la graficación.          */
    opcion = pausa();
    if ( opcion == 1 )  return;

  }
} /*  Fin de la Función graficar () . */


/*                                                                          */
/* inicializar () : Función utilizada para inicializar la pantalla en el    */
/*                  modo gráfico.  Retorna -1, si existió un error en dicha */
/*                  incialización.                                          */

int inicializar ( void )
{
  /* Declaración de variables. */

  int GraphDriver;          /* El nombre del manejador gráfico.   */
  int GraphMode;            /* El valor del modo gráfico.         */
  int ErrorCode;            /* Reporta si ha ocurrido algún error. */

  /* Detecta automaticamente el manejador gráfico.  */
  GraphDriver = DETECT;

  /* Inicializa el modo gráfico de la pantalla. */
  initgraph( &GraphDriver, &GraphMode, "" );

  /* Lee el resultado de la inicialización. */
  ErrorCode = graphresult();

  /* Si ocurre algún tipo de error, visualiza un mensaje en la pantalla, y  */
  /* retorna un -1, a la función que llamó a esta sub-función.              */
  if( ErrorCode != grOk )
  {
    putch ('\a');
    ventana_proceso ( COLOR_ERROR );
    gotoxy ( 2, 2 );
    cprintf("Error en el Sistema Gráfico :");
    gotoxy ( 2, 3 ); cprintf("%s", grapherrormsg( ErrorCode ) );
    gotoxy ( 2, 4 ); cprintf("Presione una tecla para continuar....");
    getch();
    return -1;
  }


  /* Obtiene la máxima resolución de la pantalla. */
  MaxX = getmaxx(); MaxY = getmaxy();

  /* Borra la pantalla. */
  cleardevice();

  return 0;
}          /* Fin de la Función inicializar () .  */
```

```
/*                                                                    */
/* pausa () : Detiene la ejecución del programa hasta que se haya escogido  */
/*            una de las opciones indicadas en la línea de estado.  Si la   */
/*            tecla presionada es ESC, se retorna 1, sino se retorna 0.     */

int pausa ( void )
{

  int c;            /* Variable auxiliar para leer el teclado. */

  c = getch();      /* Lee un caracter desde el teclado.       */

  if( ESC == c )
  {                 /* Si el usuario quiere salir, borra la pantalla, regre-*/
    cleardevice();  /* sa al modo de texto, y retorna un 1.                 */
    closegraph();
    return 1;
  }

  if( 0 == c )
  {                 /* Si se ha leído un código que no es ASCII, se vuelve a*/
    c = getch();    /* leer un caracter.                                    */
  }
  return 0;
}          /* Fin de la Función pausa ().  */



/*                                                                    */
/* ventana_principal () : Define la pantalla principal de graficación, dibu-*/
/*                        ja un marco y visualiza un encabezado.      */
/*                        Parámetro:                                  */
/*                      - *encabezado --> puntero al string a imprimirse en */
/*                                        pantalla.                   */

void ventana_principal ( char *encabezado )
{
  struct viewporttype vp; /* Estructura empleada para obtener las condicio- */
                          /* nes actuales del pórtico de visualización.     */
  cleardevice();          /* Borra la pantalla.  */
  setbkcolor(1);          /* Establece el color de fondo como azul. */
  setcolor( 14 );         /* Establece el color del texto como amarillo. */

  /* Abre el pórtico de visualización como pantalla completa. */
  setviewport( 0, 0, MaxX, MaxY, 1 );

  /* Define el estilo y la justificación del texto en pantalla. */
  settextstyle( DEFAULT_FONT, HORIZ_DIR, 1 );
  settextjustify( CENTER_TEXT, TOP_TEXT );

  /* Visualiza el encabezado. */
  outtextxy( MaxX/2, alto/2, encabezado );


  /* Define el pórtico de graficación y dibuja un marco a su alrededor.     */
  setviewport( 0, 2*alto, MaxX, MaxY-(2*alto+4), 1 );
  dibujar_borde( THICK_WIDTH );
  setviewport( 4, 2*alto+2, MaxX-4, MaxY-(2*alto+6), 1 );
```

```
  /* Lee las características del pórtico actual de visualización.  */
  getviewsettings( &vp );
  minx_v = vp.left;       /* Obtiene las coordenadas de la esquina superior */
  miny_v = vp.top;        /* izquierda de la actual ventana.                */
  maxx_v = vp.right;      /* Obtiene las coordenadas de la esquina inferior */
  maxy_v = vp.bottom;     /* derecha de la actual ventana.                  */
  medx_v = vp.left + ( vp.right - vp.left )/2.0; /* Obtiene las coordenadas */
  medy_v = vp.top + ( vp.bottom - vp.top )/2.0;  /* del centro de ventana.  */

  return;
}        /* Fin de la Función ventana_principal (). */



/*                                                                          */
/* linea_de_estado () : Define la línea de estado y visualiza un mensaje.   */
/*                      Parámetro:                                          */
/*                    - *msj --> puntero al mensaje a visualizarse.         */

void linea_de_estado ( char *msj )
{
  /* Abre al pórtico de visualización como pantalla completa.    */
  setviewport( 0, 0, MaxX, MaxY, 1 );
  setcolor( 14 );    /* Define el color del texto como amarillo. */

  /* Define el estilo y la justificación del texto en pantalla.  */
  settextstyle( DEFAULT_FONT, HORIZ_DIR, 1 );
  settextjustify( CENTER_TEXT, TOP_TEXT );

  /* Define el estilo y el tipo de relleno de las líneas en pantalla. */
  setlinestyle( SOLID_LINE, 0, THICK_WIDTH );
  setfillstyle( EMPTY_FILL, 0 );

  /* Dibuja el marco de la línea de estado.  */
  bar( 2, MaxY-(2*alto+4), MaxX-2, MaxY );
  rectangle( 0, MaxY-(2*alto+4), MaxX, MaxY );
  setcolor( 15 );   /* Define el color del texto como blanco.   */

  /* Visualiza el mensaje de la línea de estado.   */
  outtextxy( MaxX/2, MaxY-(alto+5), msj );

  /* Define un nuevo pórtico de visualización. */
  setviewport( 4, 2*alto+4+2, MaxX-4, MaxY-(2*alto+4+2), 1 );
  setcolor( 14 );   /* Define el color del texto como amarillo. */
  return;
}        /* Fin de la Función linea_de_estado ().  */


/*                                                                          */
/* dibujar_borde () : Dibuja un marco alrededor de la pórtico actual de vi- */
/*                    sualización, con el tipo de línea especificado.       */
/*                    Parámetro:                                            */
/*                  - ancho_de_linea --> tipo de línea del marco.           */

void dibujar_borde ( int ancho_de_linea )
{
  struct viewporttype vp; /* Estructura empleada para obtener las condicio- */
                          /* nes actuales del pórtico de visualización.     */
```

```
  setcolor( 14 );          /* Define el color del texto como amarillo.  */

  /* Selecciona el tipo de línea con el cual dibujar el borde. */
  setlinestyle( SOLID_LINE, 0, ancho_de_linea );

  /* Lee las características del pórtico actual de visualización. */
  getviewsettings( &vp );

  /* Dibuja el borde de la ventana actual.  */
  rectangle( 0, 0, vp.right - vp.left, vp.bottom - vp.top );

  return;
}        /* Fin de la Función dibujar_borde (). */



/*                                                                      */
/* graficar_parametro () : Función utilizada para graficar la forma en la  */
/*                         cual ha ido evolucionando un parámetro dado.    */
/*                         Parámetros:                                  */
/*                       - *msj       --> Puntero al mensaje a visualizarse.*/
/*                       - *parametro --> Puntero al parámetro a graficarse.*/

void graficar_parametro ( char *msj, float *parametro )
{
  /* Declaración de variables. */

  extern DATOS_ENTRADA valor;            /* Estructura declarada en AG.C . */
  struct viewporttype vp; /* Estructura empleada para obtener las condicio- */
                          /* nes actuales del pórtico de visualización.     */
  int xmax, ymax;         /* Tamaño de la actual ventana de visualización.  */
  int xo, yo;             /* Origen de coordenadas.                         */
  int xf;                 /* Coordenada máxima permisible en el eje X.      */
  int yfn, yfp;           /* Coordenadas superior e inferior, máximas en Y. */
  float dx, dy;           /* Escalas utilizadas en cada eje.                */
  float x, y;             /* Coordenadas para visualizar los valores en ejes*/
  float maximo, minimo;   /* Valor máximo y mínimo del parámetro a graficar.*/
  int generaciones;       /* Número de generaciones a graficarse.           */
  int j;                  /* Variable auxiliar.                             */
  float limx;             /* Número de divisiones en el eje X.              */

  /* Lectura del número de valores a graficarse, según el número de gener.  */
  generaciones = valor.ultima_generacion;

  setcolor ( 15 );               /* Define el color del texto como blanco. */

  /* Seleciona el estilo y la justificación del texto en pantalla.          */
  settextstyle( DEFAULT_FONT, HORIZ_DIR, 1 );
  settextjustify( CENTER_TEXT, TOP_TEXT );

  /* Selecciona el tipo de línea con el cual hacer los gráficos.            */
  setlinestyle( SOLID_LINE, 0, NORM_WIDTH );

  /* Lee las características del pórtico actual de visualización. */
  getviewsettings( &vp );
  xmax = vp.right - vp.left; ymax = vp.bottom - vp.top;
```

```
/* Visualiza el mensaje correspondiente. */
outtextxy( xmax/2, ymax-(alto+2), msj );

/* Determina el valor máximo del parámetro. */
maximo = *parametro;
for ( j = 1; j < ( generaciones + 1 ); j++ )
{
  if ( *(parametro + j) > maximo )
    maximo = *(parametro + j );
}
maximo = ceil( maximo );

/* Determina el valor mínimo del parámetro. */
minimo = *parametro;
for ( j = 1; j < ( generaciones + 1 ); j++ )
{
  if ( *(parametro + j) < minimo )
    minimo = *(parametro + j );
}
minimo = floor( minimo );

if ( minimo > 0.0 )  minimo = 0.0;

/* Determina las coordenadas máximas y mínimas del eje X, así como la es- */
/* cala respectiva.                                                        */
xo = 6*alto;
xf = xmax - 2*alto;

limx = 5.0*(ceil(generaciones/5.0));

dx = (float)(xf - xo)/limx;

/* Determina las coordenadas máximas y mínimas del eje Y, así como la es- */
/* cala respectiva.                                                        */
yfn = ymax - 4*alto;
yfp = 2*alto;
dy = (float)(yfn - yfp)/(maximo-minimo);/* dividido para el valor maximo */
yo = yfp + dy*maximo;

/* Dibuja el eje de coordenadas.   */
line ( xo, yfp - alto, xo, yfn + alto );
line ( xo, yo, xf + alto, yo );

/* Presenta en pantalla la escala del eje X. */
for ( j = 1; j < 6; j++ )
{
  x = xo + j*(xf-xo)/5.0;
  moveto ( x, yo - 2 ); lineto ( x, yo + 2 );
  gprintf ( x, yo + alto,"%d", (int)(j*limx/5.0) );
}

/* Presenta en pantalla la escala del eje Y. */

settextjustify( RIGHT_TEXT, CENTER_TEXT );

for ( j = 0; j < 6; j++ )
{
  y = yo - j*(yo-yfp)/5.0;
```

```
    moveto ( xo - 2, y ); lineto ( xo + 2, y );
    gprintf ( xo - 4, y , "%5.1f", j*maximo/5.0 );
  }

  if ( minimo < 0.0 )
  {
    for ( j = 1; j < 3; j++ )
    {
      y = yo - j*(yo-yfn)/2.0;
      moveto ( xo - 2, y ); lineto ( xo + 2, y );
      gprintf ( xo - 4, y , "%5.1f", j*minimo/2.0 );
    }
  }

  /* Grafica la curva correspondiente al parámetro especificado. */
  moveto ( xo, yo - (*parametro)*dy );
  for ( j = 1; j < (generaciones + 1); j++ )
  {
    lineto ( xo + dx*j, yo - (*(parametro + j))*dy );
  }

  return;
}        /* Fin de la Función graficar_parametro () . */


/*                                                                    */
/* gprintf () :  Usada como PRINTF, excepto que la salida es enviada a la  */
/*               pantalla en el modo gráfico en las coordenadas especifica- */
/*               das.                                                 */
/*               Parámetros:                                          */
/*             - x, y  --> coordenadas.                               */
/*             - *form --> string a imprimirse en pantalla.           */

int gprintf ( float x, float y, char *form, ... )
{
  va_list argptr;                 /* Puntero a la lista de argumentos.      */
  char str[10];                   /* Buffer en el cual construir el string. */
  int cnt;                        /* Resultado de SPRINTF para el retorno.  */

  va_start ( argptr, form );      /* Inicializa funciones va_ .             */

  cnt = vsprintf ( str, form, argptr ); /* Imprime el string en el buffer.  */
  outtextxy ( x, y, str );              /* Envía el string en modo gráfico. */

  va_end ( argptr );                    /* Cierra las funciones va_ .       */

  return ( cnt );
}                /* Fin de la Función gprintf () .  */
```

C.10. MODULO AGINCLUD.H.

```
/*                                                                    */
/* ESCUELA POLITECNICA NACIONAL.                                      */
/* FACULTAD DE INGENIERIA ELECTRICA.                                  */
/* DEPARTAMENTO DE ELECTRONICA Y CONTROL.                             */
/*                                                                    */
/* TESIS DE GRADO: Identificación de Parámetros utilizando un Algoritmo  */
/*                 Genético de Optimización, Aplicación al Circuito   */
/*                 Equivalente de la Máquina de Inducción.            */
/*                                                                    */
/* AUTOR    : A. Javier Quintana M.                                   */
/* DIRECTOR : Dr. Hugo Banda Gamboa                                   */
/*                                                                    */
/* AGINCLUD.H :  Archivo de inclusión para los Algoritmos Genéticos, en el  */
/*               que se definen todas las estructuras empleadas por el pro- */
/*               grama, asi como todas las constantes.                */
/*                                                                    */
/* CREACION : 2 - XII - 1993.                                         */

/*
** DEFINICION DE CONSTANTES.
*/

#define NUM_DATOS_PRUEBA_BASICO   6
#define NUM_DATOS_PRUEBA_IEEE    10
#define NUM_PARAMETROS            4
#define TAMANIO_POB  100  /* Dimensión de las estructuras POBLACION.        */
#define MAX_GEN      301  /* Máximo número de generaciones permitidas.      */
#define L_PARAMETRO   10  /* Longitud del código binario de cada parámetro. */
#define L_CROMOSOMA   40  /* Longitud total del cromosoma.                  */
#define FALSO          0  /* Definición de constantes booleanas.            */
#define VERDAD         1
#define M_IEEE         0  /* Definición de los códigos del método empleado. */
#define M_BASICO       1
#define PI             3.141516  /* Define el valor de la constante PI      */
#define ESC         0x1b  /* Definición de teclas utilizadas en los menús.  */
#define ENTER       0x0d
#define TAB         0X09
#define DOBLE          2  /* Tipos de borde en los cuadros de presentación. */
#define SIMPLE         1
                          /* Colores en pantalla. */
#define COLOR_NORMAL  31  /* Azul  + Blanco.      */
#define BORDE_NORMAL  30  /* Azul  + Amarillo.    */
#define DOS_COLOR      7  /* Negro + Blanco.      */
#define COLOR_MENU   116  /* Gris  + Rojo.        */
#define REVERSO_MENU  79  /* Rojo  + Blanco.      */
#define COLOR_PROCESO 95  /* Magenta + Blanco     */
#define COLOR_ERROR   78  /* Rojo  + Amarillo.    */


/*
** DEFINICION DE LAS ESTRUCTURAS DE DATOS.
*/
```

```
/*                                                                    */
/* POBLACION :  Estructura que guarda las características principales de  */
/*              todos los grupos de parámetros de una población.      */

typedef struct
{
  int   crom [L_CROMOSOMA];
  float r1, r2, x1, xm, rfe;
  float aptitud;
  int   pareja;
  int   ruleta;
  int   cruce;
  float z_real, z_imag, torque;

} POBLACION;


/*                                                                    */
/* RESULTADOS : Estructura en que se guardan las características principa- */
/*              les del mejor grupo de parámetros de una población, en cada */
/*              generación.                                           */

typedef struct
{
  int generacion;

  struct
  {
    float minimo, promedio, maximo;
    float sumatorio;
  } aptitud;

  float error_minimo;

  struct
  {
    float r1, r2, x1, xm, rfe ;
  } parametro;

  struct
  {
    float real, imag;
  } impedancia;

  float torque;
  long n_mutaciones, n_cruces;

} RESULTADOS;
```

```
/*                                                                    */
/* DATOS_ENTRADA : Estructura en la cual se almacenan los valores ingresados*/
/*                 para las variables que se utilizan en el programa.  */

typedef struct
{
  int metodo;
  int num_pruebas;
  char archivo[21];
  float r1;
  int max_generaciones;
  int ultima_generacion;
  float p_cruce, p_mutacion;
  float torque, deslizamiento;
  float voltaje, corriente, potencia;
  float z_real, z_imag;
  float r2_max, r2_min;
  float x1_max, x1_min;
  float xm_max, xm_min;
  float rfe_max, rfe_min;
  float k1, k2, k3;
  int m;

} DATOS_ENTRADA;




/*                                                                    */
/* ESTADISTICAS : Estructura en la cual se almacenan los índices estadísti- */
/*                cos de una población.                               */

typedef struct
{
  float aptitud_minima;
  float aptitud_promedio;
  float aptitud_maxima;
  float aptitud_sumatoria;
  long  n_mutaciones;
  long  n_cruces;

} ESTADISTICAS;




/*                                                                    */
/* TEXEL : Estructura en la cual se almacenan las características de una    */
/*         celda de texto.                                            */

typedef struct
{
  unsigned char ch;
  unsigned char attr;

}TEXEL;
```