Apéndice A

Código Fuente del fichero app.c

A.1 La Función app_build_function_set

```
/* app_build_function_sets()
 *
 * Esta función define todas las estructuras de datos necesarias para
 * determinar los conjuntos de terminales y funciones que van a formar
 * parte de los individuos.
 *
 * Consultar el manual para más información.
 */

int app_build_function_sets ( void )
{
     function_set fset;
     int tree_map;
     char *tree_name;
     function sets[11] =
     { { f_mult,  NULL,      NULL,      2, "*",    FUNC_DATA, -1, 0 },
       { f_div,   NULL,      NULL,      2, "/",    FUNC_DATA, -1, 0 },
       { f_suma,  NULL,      NULL,      2, "+",    FUNC_DATA, -1, 0 },
       { f_resta, NULL,      NULL,      2, "-",    FUNC_DATA, -1, 0 },
       { f_raiz,  NULL,      NULL,      1, "raiz", FUNC_DATA, -1, 0 },
       { f_cuad,  NULL,      NULL,      1, "^2",   FUNC_DATA, -1, 0 },
       { f_sen,   NULL,      NULL,      1, "sen",  FUNC_DATA, -1, 0 },
       { f_cos,   NULL,      NULL,      1, "cos",  FUNC_DATA, -1, 0 },
       { f_atan,  NULL,      NULL,      1, "atan", FUNC_DATA, -1, 0 },
       { f_est1,  NULL,      NULL,      0, "f1",   TERM_NORM, -1, 0 },
       { f_est2,  NULL,      NULL,      0, "f2",   TERM_NORM, -1, 0 },
       { f_x1,    NULL,      NULL,      0, "X1",   TERM_NORM, -1, 0 },
       { f_x2,    NULL,      NULL,      0, "X2",   TERM_NORM, -1, 0 },
       { NULL,  f_erc_gen, f_erc_print,  0, "R", TERM_ERC,  -1, 0 }
};

/*   el parametro app.use_ercs indica si vamos a usar ercs o no en
 *   nuestra aplicación. En función de ello, el tamaño del conjunto
 *   del conjunto de funciones definidas, fset.size, variará
 */

     binary_parameter ( "app.use_ercs", 1 );
     if ( atoi ( get_parameter ( "app.use_ercs" ) ) )
          fset.size = 11;
     else
          fset.size = 10;

/*   inicialización de variables asociadas a las estructuras de datos
 *   necesarias para representar los nodos y los árboles.
 */

     fset.cset = sets;

     tree_map = 0;
     tree_name = "TREE";

     return function_sets_init ( &fset, 1, &tree_map, &tree_name, 1 );

}
```

A.2 La Función app_eval_fitness

```
/* app_eval_fitness()
 *
 * esta funcion debería evaluar el fitness de un individuo.
 * Iterará sobre el conjunto de casos de fitness. Los siguientes
 * campos en la variable (individual *) deben tomar un valor:
 *    r_fitness    (raw fitness)
 *    s_fitness    (standardized fitness)
 *    a_fitness    (adjusted fitness)
 *    hits         (hits)
 *    evald        (siempre igualado a EVAL_CACHE_VALID)

 * en las tablas equis tenemos los casos de fitness para x1 y x2
 * y en las tablas app_haches tenemos x1+h y x2+h
 */

void app_eval_fitness ( individual *ind )
{

     int i;              /* contador para el bucle */
     int cond1;          /* banderas para las condiciones
     int cond2;           * de Lyapunov */
     int local_flag;

     set_current_individual ( ind );

/*   ponemos inicialmente el fitness y los hits a cero  */

     ind->r_fitness = 0.0;
     ind->hits = 0;

/*   banderas para detectar operaciones indebidas como división por
 *   cero o raiz de un número negativo  */

     div_cero_flag=0;
     local_flag=0;

 /* evaluamos V(0,0). */

     g.x1 = 0.0;
     g.x2 = 0.0;
     vorig = evaluate_tree ( ind->tr[0].data, 0 );

 /*si divido por cero en el individuo le doy fitness nulo*/

     if ( div_cero_flag == 1 )
        local_flag=1;

     div_cero_flag=0;


     /* bucle para evaluar el fitness de las muestras. */

     for ( i = 0; i < no_casos; ++i )
     {
         cond1 = 0;
         cond2 = 0;


     /* evaluamos V(x1, x2). */

         g.x1 = app_equis[0][i];
         g.x2 = app_equis[1][i];
         vobj = evaluate_tree ( ind->tr[0].data, 0 );

     /* si divido por cero en el individuo le doy fitness nulo */

         if ( div_cero_flag == 1 )
            local_flag=1;

         div_cero_flag=0;

     /* evaluamos V(x1+h, x2). */
```

```
            g.x1 = app_haches[0][i];
            vp1 = evaluate_tree ( ind->tr[0].data, 0 );

        /* evaluamos V(x1, x2+h). */

            g.x1 = app_equis[0][i];
            g.x2 = app_haches[1][i];
            vp2 = evaluate_tree ( ind->tr[0].data, 0 );

        /* calculamos derivadas parciales de V. */

            vparc1 = (vp1-vobj)/haux;
            vparc2 = (vp2-vobj)/haux;
            g.x2 = app_equis[1][i];

        /* calculamos V prima. */

            vprima = (vparc1*fun1())+(vparc2*fun2());


        /* comprobamos la primera condicion. */

            if ( vobj > vorig )
            {
               ind->r_fitness += 1.0;
               cond1=1;
            }


        /* evaluamos la segunda condicion. */

            if ( vprima <= 0.0 )
            {
               ind->r_fitness += 1.0;
               cond2 = 1;
            }

        /* si se cumplen ambas: un triunfo. */

            if ( (cond1==1)&&(cond2==1) )
               ++ind->hits;

    }

    /* si ha habido alguna div por cero, le doy a ese individuo
     * fitness cero */

    if ( local_flag == 1)
    {
       ind->r_fitness=0.0;
       ind->hits=0;
    }

    /* calculo los distintos tipos de fitness. */

    ind->s_fitness = (double)((2*no_casos)-(ind->r_fitness));
    ind->a_fitness = 1/(1+ind->s_fitness);

    /* esto siempre es asi. */

    ind->evald = EVAL_CACHE_VALID;
}
```

A.3 La Función app_end_of_evaluation

```
/* app_end_of_evaluation()
 *
 * esta función se llama después de que la población de una generacion
 * haya sido evaluada.
 *
 * run_stats[0].best[0]->ind es un puntero al mejor individuo de toda la
 * ejecución.  Si la bandera "newbest" vale 1, indica que ha aparecido un
 * nuevo mejor individuo.
 *
 * igualmente, gen_stats[0].best[0]->ind apunta al mejor individuo de
 * cada generación.
 *
 * NO MODIFICAR NINGUNA DE LAS ESTRUCTURAS QUE SE PASAN COMO PUNTEROS.
 *
 * Debería devolver un 1 si se alcanza el criterio de terminación (y la
 * ejecución del programa se detendría) y 0 en otro caso. No es necesario
 * comprobar si se ha llegado a la máxima generación  permitida – esto lo
 * hace el kernel.
 *
 */

int app_end_of_evaluation ( int gen, multipop *mpop, int newbest,
                            popstats *gen_stats, popstats *run_stats )
{
     int i, j;              /* contadores para bucles */
     double v, v2;          /* variables para almacenar el valor
                             * de un árbol */
     double rad, angulo;    /* radio y angulo de los casos de fitness */

     /* se crea un nuevo fichero de salida de datos y para ello es
      * necesario un puntero a fichero */

     FILE *res;
     int gener=gen;
     res = output_filehandle ( OUT_USER+1 )

     /* después de cada generación cambiamos el
      * set de puntos de muestreo */

     for ( j = 0; j < no_casos; ++j )
     {
         teta = random_double()*2*PI;
         radio = random_double()*erre;
         uno = radio*sin(teta);
         dos = radio*cos(teta);
         while((uno==0.0)&&(dos==0.0))         /* no considero el origen*/
        {
               teta = random_double()*2*PI;
               radio = random_double()*erre;
               uno = radio*sin(teta);
               dos = radio*cos(teta);
        }

        haux=h;
        unoh = uno+haux;
        dosh = dos+haux;
        while((unoh==uno)||(dosh==dos))    /* si tomo la h demasiado
        {                                   * pequeña como para que se
             haux = haux*10.0;              * confundan los puntos, la
             unoh = uno+haux;               * aumento automáticamente */
             dosh = dos+haux;
        }

        app_equis[0][j] = uno;
        app_equis[1][j] = dos;
        app_haches[0][j] = unoh;
        app_haches[1][j] = dosh;
     }
```

```
    /* si alcanzo el dominio completo, hemos terminado.
     * erre es una variable global que contiene el máximo
     * radio permitido para el dominio */

     if ( erre >= PI )  return 1;

    /* en este if, cada vez que aparece un individuo exitoso,
     * se imprime en un fichero de salida definido por el usuario
     * dicho indivuduo así como un conjunto de puntos en el que es
     * evaluado y su valor correspondiente */

     if ( newbest )
     {
        output_stream_open ( OUT_USER );
        oprintf ( OUT_USER, 50, "\n\n== GENERACIÓN: %d ==\n", gener );
        oprintf ( OUT_USER, 50, "\n\n== DOMINIO: %lf ==\n", erre );
        for ( i = 0; i <= 200; ++i )
        {
           rad = (double)i*(erre/200.0);
           angulo = (double)i*(PI/100.0);
           g.x1 = rad*sin(angulo);
           g.x2 = rad*cos(angulo);
           v = evaluate_tree ( run_stats[0].best[0]->ind->tr[0].data, 0);
           oprintf ( OUT_USER, 50, "%lf %lf %lf\n", g.x1, g.x2, v );
        }

        output_stream_close ( OUT_USER );
/*     ahora no terminamos cuando encuentro un individuo exitoso  */
/*     if ( run_stats[0].best[0]->ind->hits == no_casos )
          return 1 ; */

    }


/* después de cada generación imprimo en el fichero de resultados
 * el mejor individuo e información de interés como profundidad,
 * numero de nodos, fitness, radio del dominio, hits y algunos puntos
 * donde se evalúa y se muestra su valor */

    oprintf ( OUT_USER+1, 50, "\n\n=== BEST-OF-GEN  %d  ===\n", gener );
    oprintf ( OUT_USER+1, 50, "  nodes: %d\n", gen_stats[0].bestnodes );
    oprintf ( OUT_USER+1, 50, "  depth: %d\n", gen_stats[0].bestdepth );
    oprintf ( OUT_USER+1, 50, "   hits: %d\n",
              gen_stats[0].best[0]->ind->hits );
    oprintf ( OUT_USER+1, 50, "      Radio del dominio: %lf\n", erre );
    oprintf ( OUT_USER+1, 50, "            raw fitness: %.4lf\n",
              gen_stats[0].best[0]->ind->r_fitness );
    oprintf ( OUT_USER+1, 50, "    standardized fitness: %.4lf\n",
              gen_stats[0].best[0]->ind->s_fitness );
    oprintf ( OUT_USER+1, 50, "        adjusted fitness: %.4lf\n",
              gen_stats[0].best[0]->ind->a_fitness );
    oprintf ( OUT_USER+1, 50, "\n\nTOP INDIVIDUAL:\n\n" );

    pretty_print_individual ( gen_stats[0].best[0]->ind, res );

    oprintf ( OUT_USER+1, 50, "\n\nALGUNAS MUESTRAS DEL INDIVIDUO\n\n");
    oprintf ( OUT_USER+1, 50, "\n\nx1\t        x2\t
         V(x1,x2)\n\n");

    for ( j = 0; j <= 20; ++j )
    {
        rad = (double)j*(erre/20.0);
        angulo = (double)j*(PI/10.0);
        g.x1 = rad*sin(angulo);
        g.x2 = rad*cos(angulo);
        v2 = evaluate_tree ( gen_stats[0].best[0]->ind->tr[0].data, 0 );
        oprintf ( OUT_USER+1, 50, "%lf\t%lf\t%lf\n", g.x1, g.x2, v2 );
    }

    /* si hay un éxito en la generación aumento el dominio */

    if ( gen_stats[0].best[0]->ind->hits == no_casos )
        erre = erre + 0.05*PI;

    return 0;
}
```

A.4 La Función app_initialize y app_uninitialize

```
/* app_initialize()
 *
 * se realizan por parte del usuario, todas aquellas inicializaciones
 * que sean necesarias.
 *
 * devuelve 0 si todo va bien, o 1 para abortar la ejecución.
 */

int app_initialize ( int startfromcheckpoint )
{
   char param;        /* puntero a char para leer un parámetro */
   int j;             /* contador para un bucle*/

/* le doy el valor inicial a erre */

   erre = 0.1*PI ;

/*  startfromcheckpoint es una bandera que indica si la ejecución del
programa comienza desde el principio o desde un punto intermedio donde
quedó con anterioridad */

   if ( !startfromcheckpoint )
   {
      oprintf ( OUT_PRG, 50, "no se empieza de un fichero
      checkpoint.\n");

      /* si no le damos ningún valor al número de casos de fitness en el
       * fichero de entrada de parametros, se le asigna 200 por defecto
      */

      param = get_parameter ( "app.no_casos" );
      if ( param == NULL )
         no_casos = 200;
      else
      {
         no_casos = atoi ( param );
         if ( no_casos < 0 )
         error ( E_FATAL_ERROR, "valor invalido para \"app.no_casos\".");
      }

/* se reserva espacio en memoria para los casos de fitness*/

      app_equis[0] = (double *)MALLOC ( no_casos * sizeof ( double ) );

      app_equis[1] = (double *)MALLOC ( no_casos * sizeof ( double ) );

      app_haches[0] = (double *)MALLOC ( no_casos * sizeof ( double ) );

      app_haches[1] = (double *)MALLOC ( no_casos * sizeof ( double ) );

      /*le damos un valor inicial a los puntos de testeo*/

      for ( j = 0; j < no_casos; ++j )
      {
          teta = random_double()*2*PI;
           radio = random_double()*erre;
          uno = radio*sin(teta);
          dos = radio*cos(teta);
          while((uno==0.0)&&(dos==0.0))
          {
               teta = random_double()*2*PI;
                radio = random_double()*erre;
               uno = radio*sin(teta);
               dos = radio*cos(teta);
          }

          haux=h;
          unoh = uno+haux;
          dosh = dos+haux;
```

```
            while((unoh==uno)||(dosh==dos))
            {
                haux = haux*10.0;
                unoh = uno+haux;
               dosh = dos+haux;
            }

            app_equis[0][j] = uno;
            app_equis[1][j] = dos;

            app_haches[0][j] = unoh;
            app_haches[1][j] = dosh;


        }

    }

    else
    {
       oprintf ( OUT_PRG, 50, "comienzo desde un fichero checkpoint.\n" );
    }



    return 0;

}



/* app_uninitialize()
 *
 * se libera espacio de memoia reservado
 */

void app_uninitialize ( void )
{

     FREE ( app_equis[0] );
     FREE ( app_equis[1] );
     FREE ( app_haches[0] );
     FREE ( app_haches[1] );

}
```