15
2 ej.

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
Facultad de Ingeniería

PAQUETE PARA EL ANÁLISIS, DISEÑO, Y SIMULACIÓN DE SISTEMAS DINÁMICOS

TESIS
Que para obtener el título de Ingeniero en Computación
P r e s e n t a :

Alvaro Castiello de la Hidalga

Director de Tesis:
Ing. Francisco Rodríguez Ramírez

MEXICO, D.F. 1991.

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

PRÓLOGO

¿Quién en estos días no se ve de alguna u otra manera influenciado por la computación?. Tenemos que reconocer que la computación es el monstruo tecnológico de este siglo y de venideros.

Pero, ¿Qué se puede hacer con una computadora?. La respuesta a esta quizá demasiado ambiciosa pregunta, se me antoja muy sencilla : Todo y nada.

¿Por qué todo y por qué nada?. Al decir todo, me refiero al hecho de que infinidad de campos en los que se pensaba que la computadora no tenía nada que hacer, han sido atrapados en sus redes : Médicos, abogados, historiadores y hasta adivinos[1], han empezado a usar una computadora.

Ahora bien, con una computadora no se puede hacer nada mientras no se le suministren instrucciones y datos para realizar una determinada labor. Punto. Quizá esta realidad destruya el sueño de muchos, de que con conectar una computadora a la toma de corriente, ésta predecirá quién será el próximo presidente de México, o podrá decir exactamente de que materiales está hecha la mesa en que fue colocada.

El objetivo final de esta tesis no es el de filosofar acerca de la computación. Como ingeniero que pretendo llegar a ser me centraré en el aspecto práctico de estas máquinas de rara apariencia, (viéndolo bien son bastante feas), que pueden ahorrarnos horas, días, meses y hasta años de trabajo.

Existen dos ramas principales dentro de la computación : el *software*, (o programática como algunos le llaman), que es la rama que se centra en las tareas que realiza una computadora, es decir, en su programación, y el *hardware*, (quizá se le pueda llamar electrónica computacional), que se dedica a estudiar el funcionamiento interno de la computadora y sus múltiples dispositivos.

Dentro del *software* existen infinidad de nuevas ramas y temas de estudio, como por ejemplo la inteligencia aritificial, cuyo objetivo es el de enseñar a las computadoras los métodos de razonamiento del ser humano. Existen asimismo gran cantidad de disciplinas de programación[2] : Programación estructurada, programación orientada a objetos (OOPS), algoritmos genéticos, diseño estructurado, etc.

[1] No es broma. Yo llegué a ver puestos en los que se hacían horóscopos por computadora.

[2] Se reconoce que gran parte de la tecnología computacional se ha desarrollado en Estados Unidos, y que los estadounidenses han tratado de buscar nombres chuscos a sus nuevos inventos o descubrimientos. OOPS es una expresión muy usada por ellos y es equivalente a nuestro "En la torre". La expresión OOPS son las siglas de "Object oriented Programming Systems". Otro ejemplo es el de WORM, (gusano), que quiere decir "Write Once Read Many".

Cabe decir que el hecho de llamarlas disciplinas se debe a que no es necesario tener una herramienta que las soporte para poder implementarlas. Se puede programar estructuradamente en un lenguaje no estructurado como lo es el BASIC. Igualmente se puede hacer programación orientada a objetos sin necesidad de usar un compilador de C++, aunque se debe procurar como en todo, usar la herramienta adecuada para la tarea adecuada.

Pues bien, el objetivo de esta tesis es el de mostrar algunas de estas disciplinas del *software* a través de una aplicación práctica. Aplicaciones de este tipo hay muchas, pero recuerdo que un día el Ing. Francisco Rodríguez me dijo, quizá a modo de queja, ¿Dónde está la ingeniería?, y yo le doy la razón. Si pretendemos ser ingenieros, hay que buscar aplicaciones ingenieriles a la computación.

Así pues, el tema que se ha escogido es el de los algoritmos y métodos de control de plantas. Un proyecto bastante ambicioso y bastante extenso. Cuenta con numerosos módulos, de los cuales yo me abocaré al módulo principal, a los módulos de captura de información, al intercambio de dicha información dentro de los diferentes módulos y al módulo de creación de lugares geométricos.

La tesis está dividida en cuatro capítulos : la introducción y un capítulo para cada módulo. Al final de cada capítulo se muestran conclusiones del mismo, así como comentarios para que el lector pueda obtener sus propias conclusiones. Se termina con las conclusiones generales de este trabajo y los listados de los programas diseñados y realizados.

Entremos pues al mundo de la computación y el control. Trataré en lo posible de no ser rigurosamente técnico, sino de dar un enfoque ameno, práctico y comprensible. Después de todo, no se trata de dormir al lector, sino de despertar en él el interés por las diferentes técnicas de computación aplicadas al control.

INDICE

En este capítulo mostraré un panorama general del proyecto de control. Asimismo se hablará de los métodos de diseño de *software*, un poco de historia del mismo y del lenguaje elegido para la implementación del proyecto.

El problema de control.

Muchas personas mencionan el control, pero, ¿Qué es en realidad el control?. Esta pregunta tiene muchas respuestas. Desde el punto de vista ingenieril, podríase decir que el control es la disciplina que estudia la manera de que los sistemas, cualesquiera que éstos sean, se comporten dentro de las especificaciones y funcionamiento para los que fueron diseñados, aun ante la presencia de factores externos, ajenos al sistema.

Hablar de control, necesariamente implica hablar de matemáticas, de álgebra, de cálculo diferencial e integral, de transformadas de Laplace, de Fourier, de transformada Z, de solución de polinomios, generalmente de grados mayores a cinco, etc. Todos estos cálculos pueden requerir bastante tiempo de trabajo humano, de tal manera que lo mejor es delegar este trabajo a las computadoras.

Los humanos son lentos para calcular y las computadoras muy rápidas, pero las computadoras requieren de que se les suministren programas y datos para poder realizar dichos cálculos. Estos datos y programas serán realizados por humanos, los cuales pueden comprender el problema de control, mientras que una computadora no. De tal manera, una computadora y un humano, forman un binomio terriblemente eficaz.

Panorama general del proyecto de control.

La idea central de este proyecto de control, es la desligar al diseñador o ingeniero de control de la tediosa labor de cálculo numérico, de tal manera que pueda centrarse primordialmente en la labor de diseño. El programa también será capaz de auxiliarlo en la etapa de análisis de sistemas.

El programa contará con los siguientes tópicos :

1.- Técnicas de análisis :

1.1.- Respuesta en el tiempo :

- Variables de estado
- Función de transferencia

1.2.- Dominio de la frecuencia :

- Diagramas de Bode
- Diagramas de Nyquist
- Diagramas de Nichols
- Trazado de lugares geométricos de las raíces

2.- Técnicas de diseño :

2.1 .- Controlador proporcional
2.2 .- Controlador integral
2.3 .- Controlador derivativo
2.4 .- Controlador integral derivativo (PID)
2.5 .- Controlador pseudoderivativo
2.6 .- Estructuras PID
2.7 .- Sintonización de controladores
2.8 .- Desempeño específico
2.9 .- Red de adelanto
2.10.- Red de atraso
2.11.- Red de adelanto y atraso

3.- Técnicas de simulación :

3.1.- Caso lineal
3.2.- Caso no lineal
3.3.- Sistemas con tiempo de retardo

4.- Utilerías de diseño y análisis

4.1 .- Regiones de trabajo

- Continuo
- Discreto

4.2 .- Discretización

4.2.1.- Integración numérica :

- Aproximación rectangular hacia adelante
- Aproximación rectangular hacia atrás
- Aproximación trapezoidal

4.2.2.- Mapeo de polos y ceros :

- Ceros no finitos en -1
- Ceros no finitos en 0

4.2.3.- Muestreador y retén

4.3 .- Variables de estado a función de transferencia
4.4 .- Función de transferencia a variables de estado

4.5 .- Formas canónicas :

- Caso escalar
- Caso multivariable

4.6 .- Patrón de polos y ceros
4.7 .- Expansión en fracciones parciales
4.8 .- Transformación a plano W
4.9 .- Identificación (tiempo-frecuencia)
4.10.- Criterio de estabilidad de Routh

4.11.- Caracterización :

- Primer orden con tiempo muerto
- Segundo orden con tiempo muerto

4.12.- Parámetros de diseño

Historia del software.

Al hablar de la historia del *software* no me refiero a dar fechas, lugares y nombres ni de personas ni de programas. Me refiero a recordar la época en que cada quien hacía el *software* como mejor le parecía.

No había metodologías de diseño ni de implementación. Los programas se

hacían "durante el vuelo"[1]. Se concebía un programa y se empezaba a teclearlo, (o a perforarlo en tarjetas), sin ninguna etapa de análisis y diseño.

Fue hasta que se concebió la necesidad de estas etapas cuando nace la ingeniería de programación, una nueva rama que aplica los métodos ingenieriles al desarrollo de programas.

Los programas son estudiados y analizados. Se hacen estudios de viabilidad, factibilidad y de mercado. Surgen autores que muestran diversos métodos de estudio y diseño. Surge el CASE[2] y en general se toma al desarrollo de *software* como una labor seria, en algunos casos aplicándose el método científico.

Esquemas de diseño de software.

El *software* es un sistema que debe diseñarse, (aunque aquí no hay que hablar de matemáticas tan rudas como las de diseño de sistema físicos, tan sólo abarca las mismas matemáticas que la tarea para la que el programa será realizado).

Al igual que los sistemas físicos, existen un gran número de metodologías de diseño de *software*. Un *software* como un sistema físico, cuenta con varias partes o módulos. De qué manera estos módulos se funcionan internamente, es el tema de los métodos de diseño de *software*. Hay que aclarar que estos métodos, más que métodos son una disciplina, ya que tan sólo son herramientas que se pueden usar o no.

Un esquema de diseño es la forma en la que los diferentes módulos se acoplan entre sí. Es decir, la forma en que interactúan, entendiéndose por interacción el que un módulo requiera de una subrutina que está en otro, o el intercambio de datos entre ellos.

[1] Término traducido de literatura estadounidense donde hablan de cosas hechas "on the fly".

[2] CASE. Computer Aided Software Engineering.

En todos los casos, existe un módulo principal que es el que regula la interacción entre los demás módulos.

Esquema de estrella.

El esquema de estrella supone que el módulo principal está rodeado por los demás módulos, y que él interactúa con ellos, no habiendo interacción entre los demás módulos. Si dos módulos requieren interacción entre ellos, ésta debe realizarse a través del módulo principal. La siguiente figura muestra este esquema :

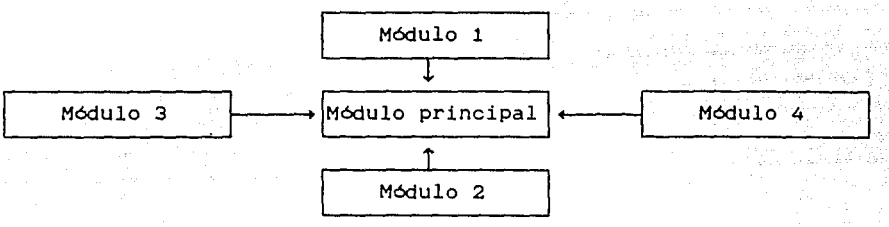


FIGURA 1.1. ESQUEMA DE ESTRELLA.

Este esquema tiene ventajas y desventajas. Su principal ventaja radica en que los diversos módulos pueden programarse concurrentemente, con lo que se reduce el tiempo de implementación. Si un módulo presenta fallas, puede fácilmente retirarse del sistema. Sin embargo, si existen interacciones entre los módulos, el módulo principal debe estarse cambiando continuamente, y en general cualquier interacción entre módulos, deberá analizarse con cuidado.

Si existen demasiadas interacciones entre dos o más módulos, es preferible fusionarlos en uno solo, pero en algunos casos prácticamente es imposible retirar las interacciones entre módulos, ya que algunos de ellos serán módulos que contengan librerías de uso general para realizar menús o ventanas. Lo más recomendable es que estas funciones de interfaz sean

realizadas por el módulo principal, pero si esto no es posible y existirán dichas interacciones, entonces se tiene un esquema llamado de red. Dicho de una manera más clara, el esquema de red es un esquema de "todos contra todos". En general es más complicado mantener un sistema con esquema de red.

Esquema lineal.

El esquema lineal puede simplificar un esquema de red, sin que por ello dejen de existir interacciones entre módulos. A cada módulo se le asigna una prioridad, los módulos que realizan las funciones de bajo nivel y los de librería tienen la prioridad más baja, y los módulos que se desarrollan para el sistema la más alta.

Un módulo de prioridad 'n' puede interactuar libremente con otro de prioridad 'm', siempre y cuando m < n, es decir que el primero tenga mayor prioridad que el otro. El módulo principal tiene la mayor prioridad y por lo tanto puede interactuar libremente con los demás.

Cabe resaltar que en todos estos esquemas, siempre es el módulo principal el que tiene prioridad sobre todos los módulos, aunque éstos pueden contener información privada, no visible al módulo principal.

La desventaja del esquema lineal es que no se pueden programar los módulos de manera concurrente. Antes de proceder con un módulo, todos los que tienen menor prioridad deben estar funcionando, y cuando se descubre una falla en un módulo, no puede retirarse de manera fácil, además de que el error se propaga a todos los demás módulos.

Interacción de módulos.

El problema de la interacción se ha tratado de manera parcial. Algunos lenguajes de programación permiten el diseño en base a más de un esquema, por ejemplo, el lenguaje C soporta tanto el esquema de estrella, el de red

y el lineal.

Sea cual sea el esquema a seguir, (principalmente en el de red), un módulo debe informar a los demás acerca de qué funciones y variables contiene. De la misma manera, debe esconder la información que considera privada. Quizá se piense que esto no es necesario en el esquema de estrella, pero en realidad persiste, ya que se debe recordar que todos los módulos deben informar por lo menos al principal acerca de su contenido.

Este acto de informar acerca del contenido de un módulo, se realiza en muchas ocasiones por medio de lo que se conoce como un "archivo de encabezado". Este archivo de texto contiene las declaraciones de las funciones y variables que otros módulos pueden accesar. Es responsabilidad del programador que el contenido de dicho archivo coincida con el contenido del módulo.

Módulos, archivos y subrutinas.

¿Qué es un módulo?, ¿Cuál es la diferencia entre módulo y subrutina?, ¿Qué es un archivo?. Estos tres términos tienden a causar confusión. Aclaremos pues en qué consisten.

Un archivo es el medio físico en el que se almacenan las funciones de un módulo o subrutina. Punto. De tal manera que un archivo puede contener varios módulos, o una subrutina puede abarcar varios archivos. También un módulo puede residir en varios archivos. Tómese como ejemplo los "archivos de encabezado", el módulo como tal reside en dos archivos, uno de implementación y otro de declaraciones.

Ahora, ¿Cuál es la diferencia entre módulo y subrutina?. Un módulo es una entidad lógica que realiza una o varias funciones. Cada función puede realizarse por medio de varias subrutinas. Por ejemplo, un módulo para editar archivos, realiza esta función por medio de la subrutina de lectura del archivo, otra que lo edita, etc.

En este punto quizá piense el lector que me he alejado del tema del

proyecto de control y pensará que cuál es la necesidad de mencionar todos estos aspectos de programación.

La idea es colocarlo en el ambiente de trabajo del proyecto, el cual será realizado por varias personas, por lo que hay que buscar la manera óptima de que puedan trabajar.

Esquema de ranuras.

¿Por qué no dejar libres las interacciones con los módulos de librería y limitarlas en cuanto a los módulos propios del sistema?.

Contestando esta pregunta, el Ing. Francisco Rodríguez y yo, acordamos crear un esquema que llamaremos de ranuras. Este esquema será una combinación del de red, (y por lo tanto del de estrella), y del lineal.

A cada módulo, aparte de una prioridad, se le asignará una clase. Para no darles nombres rebuscados, los módulos tendrán tres clases : sistema, librería y propio. Los módulos propios serán los que el equipo de trabajo vaya desarrollando y podrán interactuar libremente con los módulos de sistema y de librería, pero no deberán llamar a otro de clase propio, aun cuando puedan hacerlo, (es decir cuando tenga menor prioridad).

De esta manera, se puede programar concurrentemente y libertad de interacción con los módulos de librería.

Si esto se realiza de dicha manera, el agregarle un nuevo módulo al módulo principal requerirá un mantenimiento mínimo. Sería algo así como insertar una tarjeta en una ranura de una computadora y darla de alta en la configuración. De ahí que este esquema se llame de ranuras. En la siguiente figura se muestra:

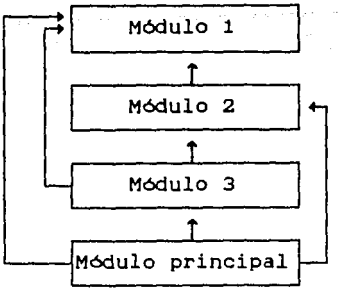


FIGURA 1.2. ESQUEMA DE RANURAS.

Selección del lenguaje.

Si se va a seguir el esquema de ranuras, se debe escoger un lenguaje de programación que lo soporte. El lenguaje C es muy flexible y podría soportarlo, pero tiene un gran inconveniente : los módulos requieren del "archivo de encabezado" para la información del contenido de un módulo y el mantenimiento de dichos archivos es tedioso.

El lenguaje seleccionado es Pascal y específicamente Turbo Pascal, versiones 4.0 en adelante. El desarrollo se hará desde la plataforma IBM PC.

Para la selección del Turbo Pascal se tomaron en cuenta las siguientes consideraciones:

1) Es el lenguaje en que la mayoría de los participantes del proyecto sabe programar.

2) Es un lenguaje de alto nivel y estructurado que no obstante permite realizar operaciones de bajo nivel y presenta una gran facilidad para combinarlo con otros lenguajes como el C y ensamblador.

3) Elimina la necesidad de los "archivos de encabezado" al presentar el módulo final en un modo compacto llamado "Turbo Pascal Unit (TPU)".

Cabe aclarar que se seleccionó la versión 4.0 o posterior debido a que los TPU sólo existen en dichas versiones. En las anteriores no existían ni TPU ni módulos ni nada. De hecho cada vez que se hacía una modificación a cualquier módulo, se tenía que recompilar todo el sistema.

Comentarios.

Se han presentado algunos esquemas de programación. Estos son los principales, aunque existen otros que no son sino variantes de ellos.

Esta introuducción se centró en las metodologías de diseño de *software* sin profundizar en el problema de control. En los siguientes capítulos se irá viendo la manera de enlazar estos conceptos al proyecto de control.

Se mencionó asimismo la elección de Turbo Pascal para el desarrollo del sistema. Al seleccionar Pascal, no quedan fuera de ámbito otros lenguajes como el C y el ensamblador, ya que hoy en día, es raro el sistema que se desarrolla enteramente en un solo lenguaje.

Conclusiones.

El proyecto de control es ambicioso pero realizable, si se sigue el esquema de programación propuesto.

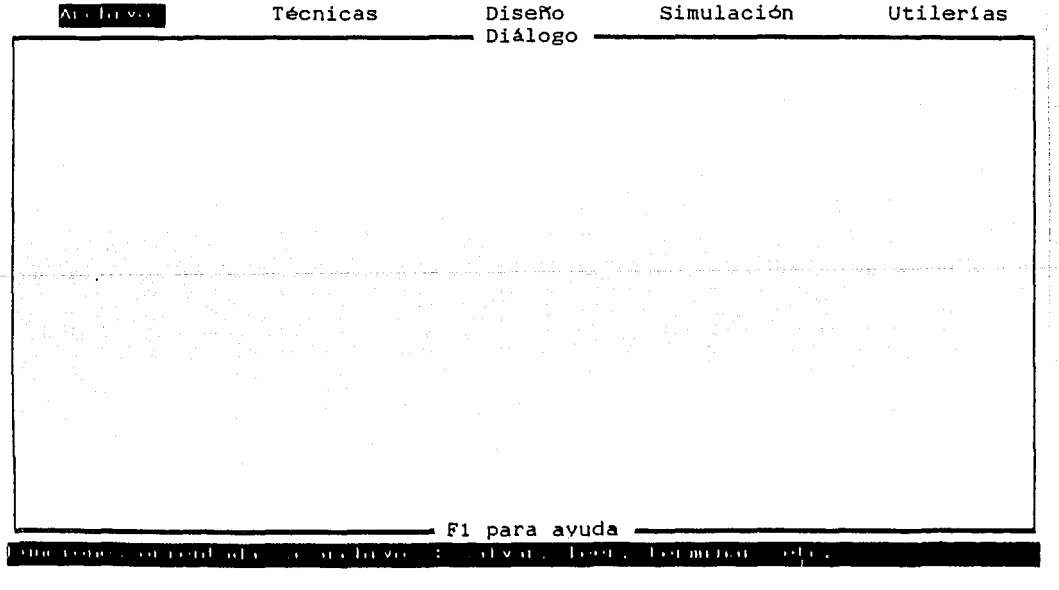
El esquema de ranuras ofrece las ventajas del esquema de red y del lineal, con mínimas desventajas. Asimismo el Turbo Pascal facilita enormemente el mantenimiento de los módulos a través de los TPU.

El módulo principal es como su nombre lo indica, el corazón de todo el sistema. Será este módulo el que ofrezca las ranuras para la inserción de los otros módulos.

El módulo principal consiste en la pantalla de presentación del proyecto y el menú principal del sistema. Este menú es realizado por medio de una llamada a un módulo de librería.

Menú principal.

El menú principal fue creado lo más apegado posible al panorama del proyecto presentado en el capítulo uno. A continuación se muestra la pantalla principal del proyecto:



La línea superior de la pantalla, contiene las opciones del menú, agrupadas por tópicos. La opción seleccionada se muestra con una barra de color. El menú es de persiana, es decir, al seleccionar algún tópico, se "descolgará" la persiana que contiene los temas relacionados al tópico.

La ventana central es la ventana de diálogo. Por medio de ella se pedirán datos y se emitirán respuestas.

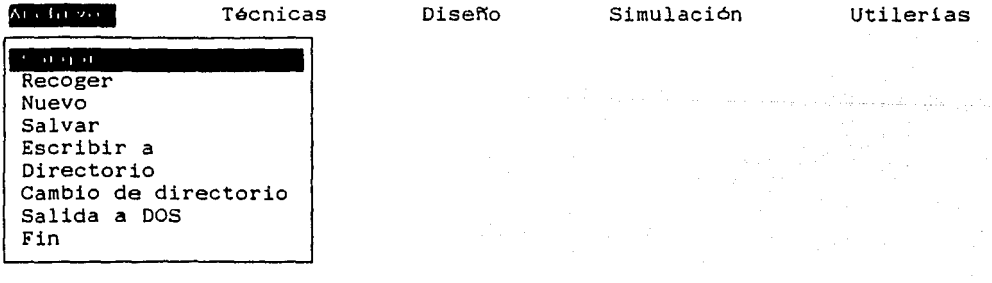
La última línea de la pantalla contendrá información acerca del tópico seleccionado.

Opciones del menú principal.

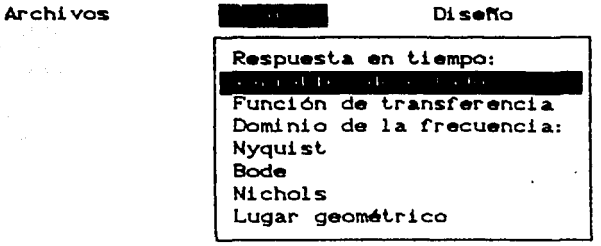
A continuación se mostrarán las diferentes opciones del menú principal, junto con los tópicos que contiene.

Archivos : Por medio de esta opción se leerán los diferentes archivos del sistema, se grabarán en disco, se podrá mostrar el directorio, y cambiar de directorio. Por medio de esta opción se podrá termina la ejecución del programa.

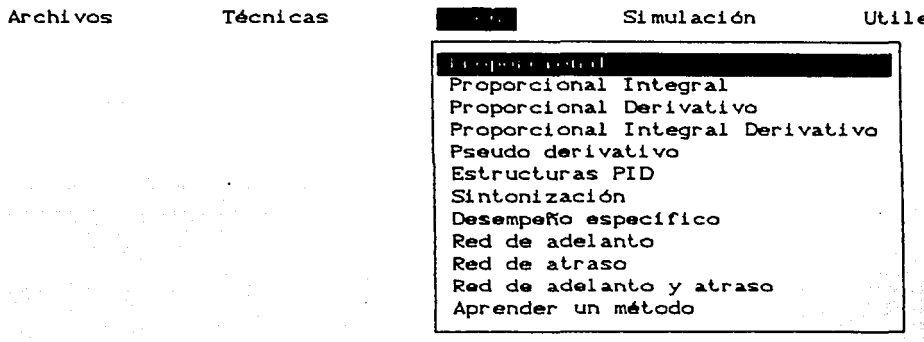
Al seleccionar esta opción aparecerá la siguiente pantalla :



Técnicas : Por medio de esta opción se seleccionan las técnicas de análisis. Contiene opciones para la respuesta en tiempo y en frecuencia. La pantalla que aparecerá será la siguiente :

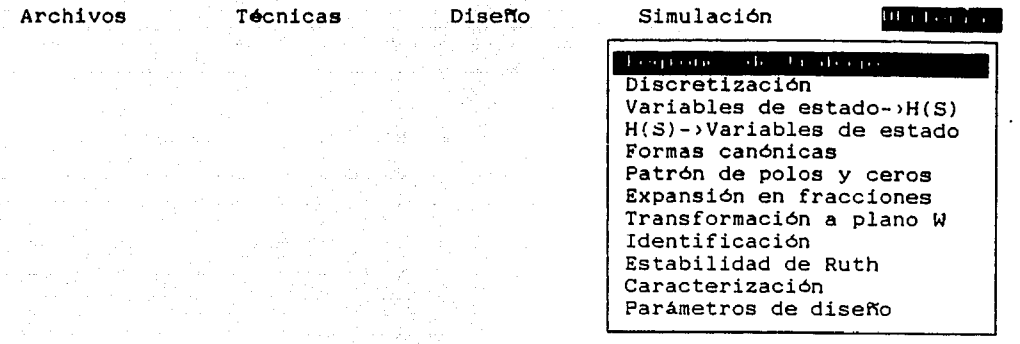


Diseño : Por medio de esta opción se seleccionan los tópicos referentes a las técnicas de diseño de sistemas. La pantalla en este caso es :

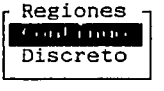


Simulación : Esta opción muestra las técnicas de simulación :

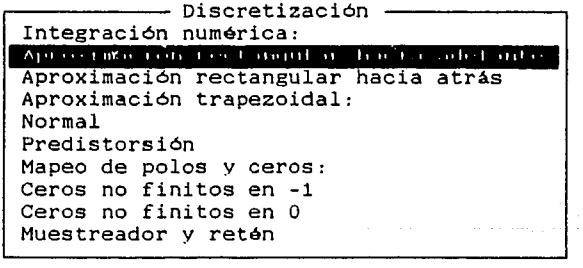
Utilerías : Esta opción es la más extensa pues contiene todas las utilerías del sistema. Además es la única opción que contiene un nivel más de menú, para las utilerías que tienen otras opciones. La primera pantalla que aparece es :



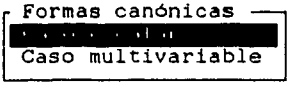
Cuando se seleccione la opción de "Regiones de trabajo" aparecerá este menú :



A su vez, la opción de discretización tiene estas opciones :



Cuando se selecciona "Formas canónicas", se puede escoger entre estas otras opciones :



Finalmente, la opción de "Caracterización" tiene las opciones :

El programa principal de este módulo también principal, almacena la opción seleccionada en el menú de persiana. Esta opción es llamada horizontal. De la misma manera almacena la opción escogida en la persiana, que es la opción vertical. Cuando se requiere, almacena la opción del menú de utilerías.

Con todo lo anterior, insertar una ranura no es más que hacer una llamada al procedimiento principal de otro módulo. Esto se logra por medio de una estructura de selección múltiple.

En el listado del módulo principal puede verse que dentro de la mencionada estructura de selección, cuando una opción es elegida, lo único que contiene es la instrucción nula de Pascal (;). La instrucción nula hace las funciones de la ranura, y el acto de dar de "alta" el nuevo módulo, es equivalente a modificar la sentencia **Uses** del módulo principal.

Por ejemplo, supóngase que una persona ha terminado de programar dentro de las técnicas de diseño, la parte de controladores proporcionales. El agregar esta parte al programa implica modificar la sentencia **Uses** para que se vea :

```
Uses
   Dos,
   Crt,
   .
   .
   .
   ContProp;
```

Ahora bien, la opción horizontal de diseño es la tres, y la vertical de controladores proporcionales es la uno, de tal manera que la inserción se hace cambiando el código que está así (se han omitido las otras posibles inserciones para mayor claridad) :

```
Case OpcionH Of
   1   : ;
   2   : ;
   3   : ;
   4   : Case OpcionV Of
             1 : ;
             2 : ;
             .
             .
             .
             End;
   .
   .
   .
   End;
```

De tal manera que quede así, (suponiendo que el procedimiento principal se llama "ControladorP") :

```
Case OpcionH Of
   1   : ;
   2   : ;
   3   : ;
   4   : Case OpcionV Of
             1 : ControladorP ;
             2 : ;
             .
             .
             .
             End;
   .
   .
   .
   End;
```

Si más adelante el módulo de controladores proporcionales muestra fallas, se podrá retirar fácilmente, quitando el nombre del módulo ("ContProp"), de la sentencia **Uses** y quitando la llamada a "ControladorP" del programa principal. Más fácil aun, se pueden tan sólo colocar como comentarios :[1]

[1] Esta técnica de comentar código ejecutable, es ampliamente usada hoy en día por la mayoría de los programadores.

```
Case OpcionH Of
   1    : ;
   2    : ;
   3    : ;
   4    : Case OpcionV Of
             1 : { ControladorP } ;
             2 : ;
             .
             .
             .
             End;
   .
   .
   .
   End;
```

Comentarios.

En este capítulo no se pretendió dar una explicación completa de cada una de las opciones del menú principal, sino mostrar todas esas opciones y mostrar también cómo el menú se organizó lo más apegado al panorama del proyecto.

Tampoco se explicaron los pasos que hay que seguir para crear el menú. En realidad son muy pocos y bastante sencillos, pero el objetivo del capítulo no era el de adentrarse en el "cómo hacer", sino en el "qué se hizo". Si se desea consultar acerca de la construcción del menú, obsérvense los listados.

El diseño de un menú puede parecer una tarea trivial, pero hay que recordar que en muchas ocasiones el menú, es el elemento principal de una interfaz con el usuario, la cual definitivamente no le da calidad a un programa, pero sí confianza al usuario.

La librería de menús requiere de la creación de varias cadenas para el mismo. Obsérvese en los listados que estas cadenas se crearon estáticamente por medio de los "Typed constants"[2] de Turbo Pascal. Esto hace el código

[2] Las "Typed constants", (constantes con tipo), no son otra cosa que variables inicializadas. Es extraño que Borland haya decidido que estas variables, (sí son variables ya que se les puede cambiar su valor), se

más claro y mucho más eficiente.

También se usaron numerosas constantes para las diferentes cadenas. Esta técnica llamada de "propagación de constantes" es muy utilizada hoy en día y es muy recomendable. Por ejemplo, supóngase que una variable puede contener el valor uno o dos. Si es uno, se desea impresión a pantalla y si es dos, se usará impresión a impresora. El código para decidir podría ser :

```
If Opcion = 1 Then
   Terminal ;
```

Este código se vuelve más claro y genera exactamente el mismo código de máquina, si se utiliza :

```
Const
   Pantalla = 1;

If Opcion = Pantalla Then
   Terminal ;
```

Aparte de lo anterior, sería desastrozo si ahora la variable toma el valor de cuatro cuando es a impresora y siete cuando es a pantalla. Con la "propagación de constantes", sólo se cambia la constante de uno a cuatro y problema resuelto. Incluso hay lenguajes como el ALGOL que no permiten la referencia a ningún número que no sea cero, uno o dos, sino es por medio de una constante. El ejemplo mostrado es sencillo, pero cuando en un módulo, éste se refiere a un uno, ¿Qué pasará cuando hay treinta dos unos y se va a cambiar por diecisiete?, ¿Cuáles hay que cambiar?, ¿Cuáles no hay que cambiar?.

Conclusiones.

Al mostrar un sencillo ejemplo de la aplicación del esquema de ranuras al módulo principal, se ve claramente cual es su potencial. Quizá parezca que es demasiado sencillo, pero eso nos lleva a una conclusión muy importante. Que muchas veces la solución más apropiada no necesariamente es

declaren en la sección de constantes y no en la de variables. De ahí que surgiera la necesidad de buscarles un nuevo nombre.

la más complicada, y en muchos casos, es la más sencilla también.

Otra de las razones para seleccionar Turbo Pascal está reflejado en el uso de constantes. Un TPU no sólo contiene las declaraciones de variables y funciones, sino también de las constantes que declare el módulo. Esta aparición "mágica", es apreciada por muchos programadores de antaño, cuando ni siquiera había modo de declarar constantes mnemotécnicas.

Por captura se entiende la acción de ingresar datos a la computadora. Esta labor puede ser bastante tediosa si no se cuenta con una interfaz adecuada, es por eso que este módulo, a pesar de no relacionarse en sí con el problema de control, es de suma importancia. El módulo de captura será un módulo de clase librería, por lo que es necesario que funciones perfectamente, ya que otros módulos lo usarán.

Captura de matrices.

La teoría de control abarca matrices y polinomios. El ingresar sus componentes es una labor pesada. Coloquémonos en una situación real.

Imagínese el lector que se debe capturar una matriz de 10 x 10. Sería bastante incómodo que el sistema empezara a preguntar uno por uno de los coeficientes, (que en este caso son cien). La gran mayoría de ellos son cero, pero aun así se debe a informar a la computadora de que es cero. Cuando finalmente ingresa el penúltimo, se da cuenta de que el treinta y ocho se ingresó incorrectamente y el programa no cuenta con mecanismos para editar la matriz. De tal manera que demostrando tener nervios de acero, se vuelven a ingresar los cien. Esta vez en el coeficiente sesenta y tres, se ingresa una cosa como "123.6y45" y el programa aborta, al no reconocer un número real.

¿Qué pasaría en una matriz de 20 x 20?. Seguramente nunca se lograría ingresar los coeficientes, y aunque así fuera, ¿Cómo se podría revisar visualmente si los datos fueron ingresados correctamente?.

La captura a través de una interfaz.

Para solucionar todas las pesadillas mencionadas en la sección

anterior, se diseño una interfaz de captura de matrices, inspirada en la interfaz de Lotus 123.[1]

Quizá se pudiera pensar que el realizar una interfaz de un programa que tiene aplicaciones primordialmente administrativas, a uno de control no es un acierto, pero hay que recalcar que en algunos sistemas, el usuario está usando la interfaz cerca del 90% del tiempo.

Antaño no se pensaba en la comodidad del usuario. El usuario se ajustaba a la lógica del programador en vez de que fuera al revés. Como resultado usar una computadora era un verdadero suplicio. Si podemos hacer de la computadora una herramienta poderosa y fácil de usar, ¿porqué no hacerlo?. Aparte de que en general, los programas de control actuales, cuentan con una interfaz muy pobre y muy incómoda de usar.[2]

En el capítulo anterior se presentó el menú principal del proyecto. Si además se cuenta con una interfaz adecuada, el programa tendrá mayor aceptación, ya que en algunos casos extremos, el usuario compra el programa por su interfaz. Ahí tenemos el caso del Turbo C[3], que siendo un compilador bastante mediocre y pobremente documentado, es aceptado por su comodísima interfaz.

Captura de matrices.

Regresando al tema de la captura de matrices, se diseño un módulo para dicha captura. Este módulo tiene las siguientes características :

- Muestra hasta doscientos coeficientes en la pantalla.
- Permite ingresar la matriz y editarla al mismo tiempo.
- Permite captura matrices de hasta 50 x 50 elementos.
- Rechaza ingresos no numéricos, (como el mencionado "123.6y45").
- Confirma el fin de captura.

[1] LOTUS 123 es una marca registrada de Lotus Development Corporation.

[2] Por ejemplo, véase la interfaz del programa CC.

[3] Turbo C es una marca registrada de Borland International.

- Permite inicializar la matriz a ceros.
- El usuario sabe exactamente que coeficiente está ingresando.
- Permite la reedición de la matriz una vez capturada.

Para ingresar un coeficiente, el usuario coloca la celda de captura, (una barra de color), sobre el elemento que desea ingresar y lo teclea. El módulo lo muestra entonces con dos decimales, pero internamente mantiene todos los que se ingresaron. Si el usuario desea observar el coeficiente, de nuevo coloca la celda sobre el elemento, y éste aparecerá en la línea superior de la pantalla. En esta línea también aparecerá siempre las coordenadas del elemento que está capturando en ese momento.

Si un número es demasiado grande para caber en la celda, ésta se mostrará llena de asteriscos[4], (por ejemplo, '********'), pero al colocar la celda en el elemento, se mostrará el número completo.

Con todo esto en mente, se muestra una pantalla de captura :

```
[illegible]

 1.24    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
 0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
 0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
 0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
 0.00    1.23    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
 0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
 0.00    0.00    0.00    0.00  *******   0.00    0.00    0.00    0.00    0.00
 0.00    0.00  [illegible] 0.00    0.00    0.00    0.00    0.00    0.00    0.00
 0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
 0.00    0.00    0.00    0.00    0.00    1.00    0.00    0.00    0.00    0.00
 0.00    0.00    0.00    0.00    0.00    0.00   23.00    0.00    0.00    0.00
 0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
 0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
 0.00    0.00    0.00    0.00    0.00  -34.00    0.00    0.00    0.00    0.00
 0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
 0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
 0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
 0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
 0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
 0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00

[illegible]
```

FIGURA 2.1. PANTALLA DE CAPTURA DE MATRICES.

---

[4] Al igual que en Lotus 123.

Como se puede observar, el capturar matrices a través de este módulo facilitará enormemente dicha captura.

Captura de polinomios.

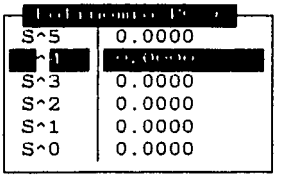
El control siempre requiere del uso de polinomios, por lo que el módulo de captura, también debe proveer los mecanismos para ingresar dichos polinomios.

Los polinomios son capturados siguiendo la misma filosofía de los menús, ya que la captura de polinomios puede verse como un menú en el que cada opción es un coeficiente del polinomio, pero al escoger dicha opción, se debe proceder a la lectura del coeficiente escogido.

Supóngase que se tiene el siguiente polinomio :[5]

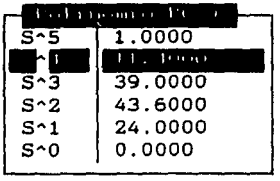
$$I(S) = S^5 + 11.4S^4 + 39S^3 + (43.6)S^2 + (24 + 2K)S + 4K = 0$$

Se desea ingresar este polinomio como P(S) + KQ(S). Entonces el módulo de captura, en la parte de captura de polinomios, presentará :



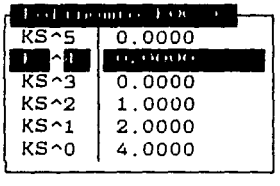
Se podrá elegir cualquiera de los coeficientes. El elegido será ingresado dentro de esta "caja" de coeficientes. Una vez terminada la captura, la "caja" mostrará :

[5] Este polinomio será usado en el capítulo cuatro, el de módulo de trazado de lugares geométricos y corresponde al ejemplo de la página 376 del libro "Ingeniería de Control Moderna", de Katsuhiko Ogata, Prentice Hall.



De nuevo se tienen todas las ventajas de la captura de matrices. Es decir, una inspección global de los datos antes de su aceptación. Aparte de que el captor de polinomios, ofrece un completo sistema de edición del número que se está ingresando.

Para el polinomio KQ(S) mostrado, el captor mostraría una vez hecha la captura, la siguiente pantalla :



Puede observarse que este captor de polinomios presenta una interfaz más apegada al lenguaje matemático, que si se pidiera "Ingrese el coeficiente de S^5 -> ", etc.

Comentarios.

En este capítulo se mostró uno de los módulos más sencillos, pero que hará más accesible el proyecto de control.

Al igual que en el capítulo del módulo principal, no se hizo hincapié en la programación del módulo, sino en lo que hace; ya que adentrarse en los detalles de dicha programación, haría esta tesis innecesariamente extensa. Aparte de que dichos detalles están íntimamente relacionados con la arquitectura interna de la IBM PC.

Al igual que los otros programas, los listados se muestran al final de la tesis.

Conclusiones.

La función primordial de una interfaz es la de facilitar el uso de un sistema. Incluso existen sistemas en los que se dedica gran parte del diseño del mismo a la creación de la interfaz.

Cumple también con la función de ofrecer al usuario un panorama general del contenido de un sistema, de los datos ingresados y de los resultados que se obtendrán a partir de dichos datos.

Dicho en palabras más llanas, podemos decir que una buena interfaz hace que el usuario pase el menor tiempo ingresando datos, (tecleando), y el mayor tiempo posible obteniendo resultados de esos datos.

Con los módulos mostrados en éste y los anteriores capítulos, pasaré al último capítulo de esta tésis, en el que se muestra una aplicación de control : la del trazado de lugares geométricos de raíces, en el que ya se usan algunos de los módulos mostrados.

Obviamente una tesis que trata sobre el problema de control no podría quedar completa sin tocar algún tema de control.

Así, este capítulo muestra un módulo en funcionamiento sobre una herramienta usada en control : el lugar geométrico de las raíces. Este módulo ya utiliza las técnicas de programación y los módulos expuestos anteriormente. Es decir, los polinomios se capturan por medio del captor de polinomios del módulo de captura y el menú principal del módulo es realizado con el mismo procedimiento que realiza el menú en el módulo principal.

En este capítulo sí es conveniente hablar un poco de la teoría relacionada con el lugar geométrico de las raíces (LGR), ya que el funcionamiento del módulo no podría entenderse si no se conoce la utilidad del método de lugar geométrico.

Introducción al método del lugar geométrico de las raíces.

Sea el sistema mostrado en la siguiente figura :

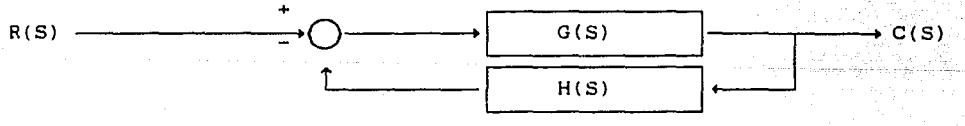


FIGURA 4.1. SISTEMA DE CONTROL.

La función de transferencia de lazo cerrado de este sistema es :

$$\frac{C(S)}{H(S)} = \frac{G(S)}{1 + G(S)H(S)}$$

La ecuación caracterisitca del sistema se obtiene al igualar el denominador de la función de transferencia a 0 :

$$1 + G(S)H(S) = 0$$

En general, G(S) a su vez es igual a :

$$G(S) = \frac{KA(S)}{I(S)}$$

Donde K es la ganacia de la planta.

Al multiplicar G(S) por H(S), sumarle uno e igualar a cero, se tendrá un polinomio con la siguiente forma general :

$$P(S) + KQ(S) = 0 \quad ...(1)$$

Pues bien, el método del lugar geométrico de las raíces consiste en encontrar el lugar que forman todas las raíces al variar K desde cero hasta infinito.

La disposición de estas raíces en el plano complejo posee mucha información acerca de la planta. Información referente a su estabilidad, a su comportamiento (amortiguado, críticamente amortiguado, oscilante, etc.), entre otras cosas. De hecho el lugar geométrico puede usarse para diseñar un sistema en base a los parámetros de diseño.

A las raíces de la ecuación :

$$KQ(S) = 0$$

Se les llama "ceros" del sistema y se representan por un pequeño círculo en la gráfica. A su vez las raíces de :

$$P(S) = 0$$

Se llaman "polos" y se representan como una cruz. Es importante que sean graficados, ya que los "polos" se "mueven" a lo largo del lugar geométrico, mientras que los ceros permanecen estáticos.

Obtención del lugar geométrico por el método analítico.

Idealmente, el lugar geométrico puede obtenerse al encontrar la solución general de la ecuación (1), es decir :

$$S = F(K)$$

Desgraciadamente, esto sólo es posible en algunos casos sencillos, como en los que el polinomio es de segundo orden, por ejemplo, si se tiene la ecuación característica :

$$s^2 + s + K = 0$$

La solución para este caso es :

$$s = -\frac{1}{2} \pm \frac{1}{2}\sqrt{1 - 4K}$$

Con esta solución, es fácil variar K desde 0 hasta infinito e ir graficando las diversas soluciones de s. Sin embargo, para sistemas de orden mayor al dos, la obtención de esta solución no es fácil de obtener.[1]

Cuando se tiene este caso, se recurre a un conjunto de reglas para trazar el lugar. Estas reglas facilitan la obtención del lugar, pero implican nuevos cálculos tediosos tales como obtención de la derivada de s con respecto a K, obtención de asíntotas, etc.

Formalización del método del lugar geométrico y el problema de control.

Con lo expuesto en las secciones anteriores, se puede dar una formalización del método del lugar geométrico, enfocado al problema de control.[2]

[1] Aparte de que existe un teorema de que es imposible obtener soluciones generales para polinomios de orden cinco o mayor.

[2] Gómez de Silva, F. Andrés. Tesis profesional.

"En este capítulo se describe en qué consiste la técnica del lugar geométrico de las raíces y para qué se utiliza. Para entender algunos de los conceptos presentados aquí se necesitan tener conocimientos básicos de teoría de control y de álgebra polinomial.

Usos de la técnica del lugar geométrico de las raíces

En la teoría de control, el propósito principal es poder estabilizar la respuesta de los sistemas dinámicos. Si se tiene un calentador, por ejemplo, el objetivo es que éste pueda ser controlado para que la temperatura del ambiente sea aproximadamente constante. Para esto, el calentador se debe prender únicamente si la temperatura desciende de un cierto valor, y se debe apagar sólo si la temperatura sobrepasa el límite superior del umbral que se haya fijado.

La idea en la teoría de control es que el calentador se prenda y se apague automáticamente, sin que ningún supervisor humano tenga que decidir cuándo hacerlo o tenga que realizar la acción. Lo que se hace para poder automatizar el procedimiento es conectar el sistema dinámico, en este caso un calentador, a otro sistema, llamado "controlador". Con la unión de estos dos sistemas, y con algunos circuitos de apoyo como sensores, transductores y actuadores, se forma lo que se llama una "malla de control".

El controlador debe encargarse de detectar la diferencia entre el valor deseado para una cantidad, que en el caso del calentador es la temperatura, y su valor real. Dicha diferencia entre valores se conoce como la "señal de error". Si existe un error lo suficientemente grande, en valor absoluto, el controlador debe aplicar alguna acción al sistema que se desea controlar, que permita disminuirlo.

La acción que toma el controlador se denomina "señal de control", y es la que entra directamente al sistema dinámico dentro de la malla de control. El objetivo es hacer que la cantidad que se desea controlar, que es la salida del sistema dinámico, se mantenga dentro de los límites aceptables. Para ello se necesita que el controlador utilice un "algoritmo de control" para determinar el valor que debe proporcionar en su salida, que es la señal de control, en cada instante.

Hay controladores de distintos tipos, pero todos tienen una componente proporcional, que multiplica la señal de error por una constante para determinar parte de la señal de control. La técnica del lugar geométrico de las raíces consiste en estudiar los efectos, en la respuesta del sistema, de tener distintos valores de dicha constante K en la señal de control. Para esto hay que tomar en cuenta el hecho de que se tiene un sistema dinámico distinto al inicial, puesto que ya se ha incluido el controlador proporcional dentro de una malla de control junto con el sistema original.

Cada sistema dinámico lineal tiene una función de transferencia, que es un modelo matemático que especifica el tipo de respuesta que proporciona el sistema dada cualquier señal de entrada. La función de transferencia de un sistema dinámico aislado es su "función de transferencia de malla abierta". Una vez incluido el sistema dentro de una malla de control, el conjunto de sistemas que forman la malla tiene una función de transferencia

distinta, llamada "función de transferencia de malla cerrada".

La técnica del lugar geométrico de las raíces se basa en una serie de reglas, que usan la función de transferencia de malla abierta de un sistema para determinar su estabilidad una vez que se junte con un controlador proporcional para formar una malla cerrada. La estabilidad es una característica que determina si la respuesta de un sistema va a tender hacia un valor finito o no. Si un sistema es inestable, su respuesta no puede ser controlable. Por lo tanto, el LGR de un sistema sirve para determinar para cuáles valores de la constante *K* el sistema es estable y para cuáles e inestable, y por lo tanto cuáles valores no se le deben dar a dicha constante del controlador.

Cualquier función de transferencia tiene la forma del cociente de dos polinomios, en los cuales normalmente se usa la variable independiente *s*. Las raíces del polinomio numerador de la función de transferencia se llaman los "ceros" del sistema, y las raíces del denominador se llaman "polos". El conjunto de los ceros y polos de un sistema dinámico son sus "singularidades".

Para hacer el análisis de los valores de *K* permitidos, se genera una gráfica de los polos de malla cerrada del sistema conforme varía el valor de *K*. Cuando *K* vale 0, los polos de malla cerrada son los mismos que los de malla abierta. Cuando *K* aumenta, los polos de malla cerrada se trasladan sobre el plano de Argand, formando trayectorias de distintos tipos.

Estas trayectorias son las que se conocen como el "lugar geométrico de las raíces positivo", que de ahora en adelante se le llama simplemente "lugar geométrico de las raíces" o "LGR". Los controladores generalmente sólo pueden proporcionar constantes mayores o iguales a 0, por lo que los valores negativos de *K* normalmente no se analizan. Si se fuera a hacer dicho análisis, sin embargo, se obtendrían trayectorias que formarían el "lugar geométrico de las raíces negativo" del sistema.

La estabilidad del sistema está en relación directa con los cruces de las trayectorias de los polos con ciertos límites, que difieren si el sistema que se está analizando es continuo o discreto. La interpretación de la gráfica, sin embargo, siempre es en base a dichas trayectorias.

La gráfica del lugar geométrico de las raíces se genera conforme unas reglas que se han establecido, generalmente conocidas como las "reglas de Evans". Estas reglas permiten obtener las características principales de las trayectorias de los polos de malla cerrada de un sistema específico. Las reglas de Evans son citadas de distintas formas o con distintos grados de explicitez por cada autor, pero el resultado de seguirlas siempre es la misma gráfica. La forma en que se usaron para elaborar esta tesis se presenta en la siguiente sección. Algunas de ellas, como se puede ver en el capítulo IV.5, en realidad no cubren todos los posibles casos, y tendrían que ser modificadas cuando se encuentre un algoritmo apropiado para ellas.

Las reglas de Evans

Regla 1. Principio y terminación de las trayectorias del LGR.

Conforme aumenta el valor de la constante *K*, una trayectoria del LGR

sale de cada polo de malla abierta del sistema. Cuando el valor de K se haya hecho lo suficientemente grande, las trayectorias terminan en los ceros del sistema, o siguen asintóticamente hacia el infinito. Cada polo del sistema termina siguiendo una trayectoria asintótica diferente, o acaba teniendo el mismo valor que un cero distinto. Si una función de transferencia tiene numerador de grado m y denominador de grado n, el sistema al que pertenece tiene m ceros y n polos. Dadas estas características, m polos terminarán sus trayectorias en algún cero del sistema, y $n-m$ polos seguirán trayectorias asintóticas.

Si la función de transferencia de malla abierta es

$$G(s) = q(s)/p(s)$$

y luego se inserta este sistema en una malla con un controlador proporcional, la función de transferencia de malla cerrada resultante sería la siguiente:

$$H(s) = \frac{KG(s)}{1+KG(s)} = \frac{Kq(s)/p(s)}{1+Kq(s)/p(s)} = K\frac{q(s)}{p(s)+Kq(s)}.$$

De esta última ecuación, se ve que los ceros de malla abierta son iguales a los de malla cerrada, y que por lo tanto no son afectados por el valor de la constante K. Los polos de malla cerrada, sin embargo, sólo son iguales a los de malla abierta cuando $K=0$. Cada vez que K tome otro valor los n polos de malla cerrada tienen valores diferentes. Debido a esto, el lugar geométrico de las raíces son las trayectorias de los polos de malla cerrada, que son las raíces del denominador del conjunto del sistema dinámico y el controlador.

Regla 2. Tramos del LGR que están sobre el eje real.

Todos los puntos del eje real que tienen a su derecha un número impar de singularidades forman parte de las trayectorias de los polos de malla cerrada. Para contar las singularidades que tienen a su derecha sólo es necesario tomar en cuenta las que están sobre el eje real, puesto que las que son complejas vienen en parejas y no modifican la paridad.

Regla 3. Asíntotas del LGR.

La cantidad de asíntotas que tiene el LGR de un sistema dinámico es igual a $n-m$. Los ángulos $\alpha_1 \ \ldots \ \alpha_{n-m}$ que forman estas asíntotas con respecto al eje real se calculan mediante la siguiente fórmula, que los proporciona en radianes:

$$a_{k+1} = \frac{\pi(2k+1)}{n-m} \qquad \text{para } k = 0, 1, \ldots, n-m-1.$$

Si hay más de una asíntota en el LGR del sistema, las que haya se intersectan en un punto σ, que está sobre el eje real, llamado su "centroide". La coordenada real del centroide de las asíntotas se calcula de la siguiente manera:

$$\sigma = \frac{\sum_{i=1}^{n} p_i - \sum_{i=1}^{m}}{n-m} .$$

Regla 4. Eje vertical de simetría.

El LGR de cualquier sistema dinámico es simétrico con respecto al eje real. Sin embargo, algunos sistemas también presentan simetría con respecto a un eje vertical, que no necesariamente tiene que ser el eje imaginario. Cuando esto sucede, y al continuar con la traza del LGR dentro de la regla 7, hay que considerar a dicho eje como si fuera una asíntota. Para que un sistema tenga un eje vertical de simetría se deben cumplir tres condiciones:

a) El sistema no debe tener una asíntota con ángulo de 90°, puesto que en ese caso el eje vertical de simetría ya estaría considerado dentro de las asíntotas del sistema y no habría que agregar sus datos antes de analizarlas.

b) El centroide de las asíntotas del sistema debe coincidir con el centro geométrico de la gráfica, el cual se calcula sumando el valor real más pequeño que aparece dentro de sus singularidades con el más grande, y dividiendo el resultado entre 2.

c) El eje vertical de simetría encontrado mediante el inciso anterior debe contener singularidades complejas para que se tenga que tratar como una asíntota.

Regla 5. Valor de la constante K en cualquier punto del LGR.

El polinomio cuyas raíces se tienen que encontrar para obtener los polos de malla cerrada de un sistema es el siguiente:

$$p(s)+Kq(s) = 0.$$

En esta ecuación se puede despejar el valor de K de la siguiente manera:

$$K = \frac{-p(s)}{q(s)} .$$

Si se quiere obtener el valor de K en algún punto del LGR del sistema, se deben sustituir las coordenadas de ese punto dentro de los polinomios $p(s)$ y $q(s)$, dividir los dos valores resultantes, e invertir el signo del resultado. Al hacer esto hay que utilizar álgebra compleja, puesto que normalmente se va a aplicar el procedimiento en puntos que están fuera del eje real y que por lo tanto tienen componente imaginaria. El hecho de poder calcular el valor de K en un punto del plano de Argand permite obtener adecuadamente los puntos silla del sistema dinámico, como se verá en la regla 6.

Regla 6. Puntos silla del LGR.

Los puntos silla de un sistema son puntos sobre el eje real, sobre el posible eje vertical de simetría, o sobre alguna asíntota, en donde las trayectorias de dos polos de malla cerrada se intersectan. No todos los

puntos silla que estén sobre una asíntota o eje deben pertenecer al LGR del sistema, sin embargo. Además, los sistemas que tienen un numerador de grado 0 y un denominador de grado 1, o cuyos dos polinomios son de grado 1, no tienen puntos silla. Para obtener los puntos silla de un sistema se deben encontrar las raíces de la siguiente ecuación:

$$\frac{d}{ds} G(s) = \frac{d}{ds} \frac{q(s)}{p(s)} = \frac{p(s) \frac{d}{ds} q(s) - q(s) \frac{d}{ds} p(s)}{p(s)^2} = 0 .$$

Los valores que hacen que la última ecuación sea igual a cero son las raíces de su numerador, por lo que se puede descartar el denominador para la obtención de los puntos silla. Una vez encontradas las raíces del numerador de la fórmula anterior se tendrán todos los puntos silla de un sistema. Antes de continuar, sin embargo, se tiene que verificar cuáles de ellos realmente forman parte del LGR del sistema al que pertenecen.

Para ello hay que calcular el valor de K en cada uno de los puntos obtenidos. Si la componente imaginaria de K es nula y la componente real es mayor o igual a 0, el punto silla pertenece al lugar geométrico de las raíces positivo. Si la parte imaginaria es nula y la parte real es menor o igual a cero, el punto silla pertenece al lugar geométrico[3] de las raíces negativo, el cual no fue incluido en el sistema experto de la tesis. Cualquier valor de K que tenga componente imaginaria no nula implica que el "punto silla" al que pertenece dicho valor en realidad no forma parte de ningún lugar geométrico del sistema.

Las raíces de la ecuación anterior que estén sobre el eje real y que formen parte del LGR positivo del sistema se pueden detectar de la misma forma, aunque hay otro método que probablemente es más rápido. Lo único que se tiene que hacer es detectar si pertenecen a algún tramo de eje en el que ya se haya dibujado una trayectoria de LGR en el paso 2. En caso afirmativo, el punto silla es parte del lugar geométrico de las raíces positivo, y en caso contrario es parte del lugar geométrico negativo. El valor de K para un punto silla que esté sobre el eje real no puede tener componente imaginaria, y por lo tanto éste siempre pertenece a alguno de los dos lugares geométricos.

Regla 7. LGR sobre las asíntotas.

En todas las asíntotas que no coinciden con el eje real, incluyendo el posible eje vertical de simetría, también pueden haber tramos rectos de LGR que coinciden con partes de las asíntotas, al igual que los hay sobre el eje real. Para determinar cuáles son dichos tramos, hay que separar la asíntota en sectores cuyos límites sean las singularidades que haya sobre ella. Si una asíntota no contiene singularidades, sólo tiene una parte, que va desde el centroide hasta el infinito.

Cualquiera de los tramos de la asíntota que se obtengan y que contenga un punto silla es parte del LGR del sistema. El tipo del punto silla se puede detectar identificando la singularidad más cercana a él. Si dicha singularidad es un cero, el punto silla es de llegada, mientras que si es un polo, el punto silla es de partida.

[3] El Ing. Andrés G. Silva, se refiere al tema de sus tesis.

Regla 8. Parejas de puntos silla.

Lo que se ha visto implícitamente hasta ahora es que los puntos silla indican las coordenadas en las que dos trayectorias de LGR se separan después de haber formado un tramo recto, o se juntan para luego formar un tramo recto. Los tramos rectos sólo ocurren sobre los ejes de simetría o sobre las asíntotas de un sistema dinámico. Fuera de los ejes o de las asíntotas, el LGR puede tener formas de trayectorias muy diversas, desde circulares y elípticas, hasta hiperbólicas y espirales.

Las singularidades de un sistema que están sobre algún eje o asíntota y que han sido tocadas con trayectorias de LGR pueden ser descartadas de los análisis posteriores, puesto que el dibujo del resto del LGR ya no las toma en cuenta, como se verá más adelante. Los polos y ceros complejos que no hayan sido tocados todavía por tramos, sin embargo, todavía tienen que ser consideradas en el análisis. De hecho, los pares conjugados de polos que queden todavía sin tocar alguna parte del LGR actúan como si fueran puntos silla de partida.

Algunos puntos silla de partida indican el lugar en el que dos trayectorias comienzan a acercarse a algún cero o a avanzar asintóticamente hacia el infinito. Sin embargo, cada uno de los demás puntos silla de partida es el lugar en el que dos trayectorias avanzan hacia sus respectivos puntos silla de llegada. El siguiente paso en la obtención de un LGR, entonces, es determinar qué puntos silla de llegada forman pareja con cuáles puntos silla de inicio. Esto no determina la forma que tiene la trayectoria que los une, pero permite tener una idea general de lo que falta por dibujar en un LGR y de la región aproximada que abarca.

Para cualquier punto silla de llegada, el punto silla de partida que esté más cercano en el plano de Argand es donde va a comenzar una trayectoria que los va a unir. Todos los puntos de llegada deben tener su pareja. Los puntos de partida que sobren son los lugares donde comienzan las trayectorias que van hacia los ceros y las asintóticas.

Hasta aquí llegan las reglas de Evans como son presentadas normalmente y en las versiones que se utilizaron para programar el sistema experto[4], aunque de hecho aquí se incluyeron más detalles de los que se mencionan comúnmente. Algunas veces se agregan reglas para obtener el ángulo que tiene el siguiente punto de una trayectoria a partir de un punto inicial que no sea punto silla, o el ángulo de llegada a los ceros complejos restantes. Sin embargo[5], estas reglas no se utilizaron al programar el sistema experto descrito en esta tesis.

Como se puede ver, ninguna de las reglas presentadas incluye información que permite conocer las formas de las trayectorias no rectas del LGR de un sistema. A pesar de esto, con la información que se tiene los ingenieros de control normalmente pueden inferir los tramos faltantes con la suficiente precisión como para que no haya problemas al usar la traza resultante para el análisis de la estabilidad del sistema".

[4] Ibidem.
[5] Ibidem.

Obtención del lugar geométrico por el método de solución.

Las reglas mencionadas en la sección anterior hacen un análisis del comportamiento del lugar geométrico. El método de solución no hace ningún tipo de análisis y es bastante simple, sin embargo, sólo es conveniente su implementación en una computadora.

En lo único que consiste este método es en sustituir un valor de K en la ecuación (1), resolver el polinomio así formado y graficar las soluciones obtenidas. Aquí se puede observar el porque de una computadora. Una persona tardaría mucho en obtener la solución de un polinomio, digamos de quinto orden, y si lo lograra, seguramente estaría harto como para emprender la solución de otro polinomio.

Obviamente la computadora debe calcular las raíces del polinomio lo más rápido posible, de lo contrario el trazado tardaría mucho tiempo. Hay muchos métodos para encontrar soluciones de polinomios en forma numérica, pero el mejor a mi criterio es el método de Bairnstow.

Operación general del método de Bairnstow.

No pretendo dar todos los detalles de este método sino sólo plantear su operación más general. Este método supone un polinomio en su forma general :

$$a_n s^n + a_{n-1} s^{n-1} + a_{n-2} s^{n-2} + \ldots + a_1 s^1 + a_0 = 0$$

El método por medio de aproximaciones sucesivas, encuentra un factor cuadrático de la forma :

$$s^2 + ps + q = 0$$

Este polinomio es resuelto por la fórmula general, el grado del polinomio original es disminuido en dos y se repite el proceso hasta que el grado del polinomio restante sea dos o uno.

El método de Bairnstow converge muy rápidamente, (en cinco o seis iteraciones), o no converge. Hay otros algoritmos que convergen pero muy lentamente, como este método lo hace de manera rápida, casi se puede asegurar que si en cincuenta iteraciones no ha convergido, no lo hará en más.

Esto disminuye aun más el tiempo de trazado, ya que el algoritmo no requiere continuar iterando si después de cincuenta iteraciones no ha convergido, por lo que se seleccionó cincuenta como el máximo número de iteraciones.

Formalización del método de Bairnstow.

En esta sección se muestra la formalización matemática del método de Bairnstow.[6]

"En varias partes del sistema experto[7] se necesitó contar con una rutina que encontrara las raíces de polinomios. El método numérico que se escogió para realizar esta tarea es el de Lin-Bairstow. En este capítulo se describe la manera en que dicho método obtiene las raíces de polinomios. Los cálculos que requiere el algoritmo se van desarrollando paso a paso. Al final de la discusión se encuentra un resumen de los cómputos necesarios para realizar el algoritmo, y también un diagrama de flujo. Para seguir la siguiente descripción se necesitan tener conocimientos básicos acerca de polinomios y sus raíces.

Descripción general

El método numérico de Lin-Bairstow permite calcular las raíces reales y complejas de polinomios de coeficientes reales. Las raíces de los polinomios pueden ser múltiples, porque el algoritmo las encuentra todas las veces que ocurran. Los cálculos involucrados en el método se hacen recursiva e iterativamente, y requieren manipular sólo números reales.

El método se basa en la obtención sucesiva de varios polinomios cuadráticos que sean factores del polinomio original. Las raíces de dichos factores cuadráticos se pueden obtener fácil y directamente mediante la ecuación cuadrática, y son también raíces del polinomio original.

Para calcular los coeficientes de cada uno de los factores cuadráticos se requiere aplicar un algoritmo repetitivamente. En cada una de las iteraciones, la rutina usa los resultados parciales calculados en la repetición anerior para obtener nuevos resultados, más cercanos a los verdaderos. Este procedimiento iterativo sigue hasta que converge el

[6] Gómez de Silva, F. Andrés. Tesis profesional.

[7] El autor se refiere de nuevo al tema de su tesis.

algoritmo a una solución.

Los polinomios a los cuales se aplica el método deben estar normalizados. Es decir, si se tiene un polinomio de grado N

$$P_N(x) = c_0x^N + c_1x^{N-1} + c_2x^{N-2} + \ldots + c_{N-1}x + c_N$$

entonces se tiene que dividir cada coeficiente dentre c_0, para alimentar al algoritmo de Lin-Bairstow con el siguiente polinomio:

$$P_N(x) = x^N + a_1x^{N-1} + a_2x^{N-2} + \ldots + a_{N-1}x + a_N$$

donde cada a es igual a c_i/c_0. Este último polinomio también se puede expresar como:

$$P_N(x) = (x - r_1)(x - r_2) \ldots (x - r_N)$$

donde las r_i, que pueden ser reales o complejas y no necesariamente tienen que ser distintas, son las raíces del polinomio, es decir, los valores de x que hacen que $P_N(x) = 0$.

Si el grado del polinomio es un número par $N = 2k$, donde k es un número entero positivo, entonces $P_N(x)$ se puede expresar como el producto de k términos cuadráticos si se agrupan por pares los términos lineales. De esta forma el polinomio queda como:

$$P_N(x) = \prod_{m=1}^{k} F_m(x)$$

donde $F_m(x) = x^2 + r_mx + s_m$.

En caso de que el grado del polinomio sea un número impar $N = 2k + 1$, entonces $P_N(x)$ se podría expresar de la misma forma, pero multiplicado por un sólo término lineal:

$$P_N(x) = (x - r_{2k+1}) \prod_{m=1}^{k} F_m(x).$$

Cuando el polinomio tiene raíces complejas, éstas deben agruparse por pares conjugados para que los factores cuadráticos $F_m(x)$ del polinomio que se encuentren únicamente tengan coeficientes reales. La raíz r_{2k+1}, que ocurre cuando el polinomio es de grado impar, siempre es real puesto que no puede tener un complejo conjugado.

El método de Lin-Bairstow permite, cuando converge el algoritmo, calcular las raíces de cualquier polinomio de coeficientes reales, al extraer sucesivamente todos sus factores cuadráticos. Las raíces de cada factor cuadrático que se obtiene, y las del término lineal que queda cuando el polinomio es de grado impar, se pueden obtener directamente. Estas raíces son también raíces del polinomio original.

Algunos ejemplos de polinomios con distintas características, y de cómo extraería sus factores cuadráticos el algoritmo, son los que siguen:

Grado **N** par, polinomio con todas sus raíces complejas:

$$P_4(x) = [x-(a_1+b_1i)]\,[x-(a_1-b_1i)]\,[x-(a_2+b_2i)]\,[x-(a_2-b_2i)]$$

$$= [x^2 - 2a_1x + (a_1^2+b_1^2)]\,[x^2 - 2a_2x + (a_2^2+b_2^2)].$$

Grado **N** par, polinomio con raíces reales y complejas:

$$P_4(x) = [x-r_1]\,[x-r_2]\,[x-(a+bi)]\,[x-(a-bi)]$$

$$= [x^2 - (r_1+r_2)x + r_1r_2]\,[x^2 - 2ax + (a^2+b^2)].$$

Grado **N** impar, polinomio con raíces reales y complejas:

$$P_5(x) = [x-(a+bi)]\,[x-(a-bi)]\,[x-r_1]\,[x-r_2]\,[x-r_3]$$

$$= [x^2 - 2ax + (a^2+b^2)]\,[x^2 - (r_1+r_2)x + r_1r_2]\,[x - r_3].$$

Grado **N** impar, polinomio con todas sus raíces reales:

$$P_5(x) = [x-r_1]\,[x-r_2]\,[x-r_3]\,[x-r_4]\,[x-r_5]$$

$$= [x^2 - (r_1+r_2)x + r_1r_2]\,[x^2 - (r_3+r_4)x + r_3r_4]\,[x - r_5].$$

Determinación de un factor cuadrático de un polinomio

En cada etapa del algoritmo de Lin-Bairstow se extrae un factor cuadrático del polinomio que se tenga como entrada a esa etapa. Si ya se han obtenido **k** factores cuadráticos, el grado del polinomio de entrada a la siguiente repetición del algoritmo es **n**, donde **n = N - 2k**. Durante la ejecución de las etapas el valor de **k** va variando desde 0 hasta el valor del cociente entero de **N/2**.

Se puede dividir el polinomio de entrada $P_n(x)$ entre un factor cuadrático de prueba, $F(x) = x^2 + rx + s$, donde **s** y **r** son constantes reales arbitrarias, para obtener los siguientes resultados:

$$P_n(x)/F(x) = P_{n-2}(x) + Rx + S \Rightarrow P_n(x) = F(x)P_{n-2}(x) + Rx + S$$

donde $P_{n-2}(x)$ es el cociente de la división del polinomio $P_n(x)$ entre $F(x)$, y $Rx + S$ es el residuo de dicha división.

Se puede escribir la última ecuación en forma expandida para obtener lo siguiente:

$$x^n + a_1x^{n-1} + a_2x^{n-2} + \ldots + a_{n-1}x + a_n$$

$$= (x^2 + rx + s)\,(x^{n-2} + b_1x^{n-3} + b_2x^{n-4} + \ldots + b_{n-3}x + b_{n-2}) + Rx + S.$$

De aquí se ve que un cambio en el valor de **r** o **s** causa que cambien los valores de los coeficientes b_k del polinomio cociente $P_{n-2}(x)$, y también los valores de los coeficientes **R** y **S** del residuo. Debido a esto, se puede considerar que **r** y **s** son variables independientes y que **R**, **S** y las b_k son funciones que dependen de dichas "variables". Estas funciones, se pueden

escribir como R(r,s), S(r,s) y bk(r,s), respectivamente.

Para que F(x) sea un factor de Pn(x), el residuo de la división de Pn(x) entre F(x) debe ser nulo, y por lo tanto se deben cumplir las siguientes restricciones:

$$R(r,s) = 0 \text{ y} \qquad (1)$$
$$S(r,s) = 0.$$

Entonces, para extraer un factor cuadrático de Pn(x) se deben calcular las raíces de estas dos últimas ecuaciones restrictivas.

Cálculo de las raíces de las dos restricciones

Se debe comenzar con valores estimados iniciales ro y so para las raíces de las dos ecuaciones del sistema (1), es decir, para los coeficientes del factor cuadrático Fm(x). Estos dos valores iniciales se tendrán que ir incrementando en valores pequeños δr y δs hasta que se obtengan valores lo suficientemente correctos para r y s. Al efectuar los incrementos, los valores de las funciones R y S cambian de la siguiente manera:

$$\delta R = \frac{\delta R}{\delta r}\delta r + \frac{\delta R}{\delta s}\delta s + \text{todas las diferenciales de orden superior y}$$

$$\delta S = \frac{\delta S}{\delta r}\delta r + \frac{\delta S}{\delta s}\delta s + \text{las diferenciales de orden superior.} \qquad (2)$$

Estas diferenciales totales son aproximaciones de primer orden debido a que se despreciaron las diferenciales de orden superior. Las diferenciales parciales de R y S con respecto a r y s de ahora en adelante se abreviarán como Rr, Rs, Sr y Ss.

Dados los valores estimados iniciales ro y so, se pueden calcular los valores de R(ro,so) y S(ro,so) al dividir Pn(x) entre $x^2 + rox + so$. Los coeficientes R y S del residuo serán:

$$R = R(ro,so) \neq 0 \quad y \quad S = S(ro,so) \neq 0.$$

Como se quiere hacer que el residuo sea nulo para que el polinomio cuadrático sea un factor del polinomio Pn(x), se deben encontrar los valores de δr y δs para que las diferenciales totales δR y δS cumplan que:

$$R(ro,so) + \delta R = 0 \quad y \quad S(ro,so) + \delta S = 0.$$

Es decir, se deben cumplir las restricciones de que:

$$\delta R = -R(ro,so) \quad y \quad \delta S = -S(ro,so). \qquad (3)$$

De las ecuaciones (2) y (3) se tiene el siguiente sistema de ecuaciones lineales:

$$Rr\delta r + Rs\delta s = -R(ro,so) \quad y \quad Sr\delta r + Ss\delta s = -S(ro,so). \qquad (4)$$

Después de calcular los valores de Rr, Rs, Sr, Ss, R(ro,so) y S(ro,so), se podrá resolver el sistema (4) de dos ecuaciones lineales, que

contiene dos incógnitas, δr y δs.

Una vez que se conozcan δr y δs se puede hacer $r_1 = r_0 + \delta r$ y $s_1 = s_0 + \delta s$, para calcular:

$$R(r_1,s_1) = R(r_0,s_0) + \delta R = R(r_0,s_0) + Rr\delta r + Rs\delta s \quad y \qquad (5)$$
$$S(r_1,s_1) = S(r_0,s_0) + \delta S = S(r_0,s_0) + Sr\delta r + Ss\delta s.$$

De las ecuacines (sic) (4) se puede ver que el lado derecho de las ecuaciones (5) es igual a cero, y por lo tanto se tiene que:

$$R(r_1,s_1) = 0 \quad y \quad S(r_1,s_1) = 0.$$

Esto normalmente implicaría que ya se hubiera encontrado el primer factor cuadrático $F_m(x)$, cuyos coeficientes serían r_1 y s_1. Sin embargo, en el sistema de ecuaciones (2) se obtuvieron únicamente aproximaciones de primer orden de δR y δS. Por lo tanto, el lado derecho de las ecuaciones (5) no vale exactamente cero, y debido a esto se tiene que continuar iterando con el algoritmo hasta realmente encontrar el factor cuadrático.

Para esto se tienen que calcular nuevamente δr y δs, a partir de nuevos valores de R, S, Rr, Rs, Sr y Ss, con ellas calcular $r_2 = r_1+\delta r$ y $s_2 = s_1+\delta s$, y seguir repitiendo el procedimiento hasta que se tengan valores para r y s lo suficientemente exactos. Una manera de detectar esta convergencia y de utilizarla para detener del algiritmo, por ejemplo, sería de continuar las repeticiones hasta que los valores de δr y δs sean más pequeños que un valor máximo que se haya fijado previamente.

Cálculo de los coeficientes R y S del residuo

Los valores de R y S, que se necesitan calcular en cada iteración del algoritmo de Lin-Bairstow, se podrían obtener al dividir el polinomio de entrada $P_n(x)$ entre el de prueba $F(x)$ directamente. Sin embargo, también se pueden obtener fórmulas recursivas que permiten hacer los cálculos más rápidamente si se programan en una comuptadora.

Para calcular R y S, primero se tienen que obtener los coeficientes b_k del polinomio cociente $P_{n-2}(x)$. Las fórmulas que nos permiten obtener los valores de dichos coeficientes son recursivas porque permiten calcular a cada b_k a partir de las b_k calculadas anteriormente. En el resto del capítulo se describe la obtención de las ecuaciones recursivas.

Si se divide $P_n(x)$ entre $F(x)$, se obtiene:

$$x^n + a_1x^{n-1} + a_2x^{n-2} + \ldots + a_{n-1}x + a_n$$
$$= (x^2 + rx + s)(x^{n-2} + b_1x^{n-3} + b_2x^{n-4} + \ldots + b_{n-3}x + b_{n-2}) + Rx + S.$$

Si se multiplican los dos polinomios del lado derecho para obtener los coeficientes de los términos del lado izquierdo de la igualdad, se puede ver que:

$$
\begin{aligned}
a_1 &= b_1 + r \\
a_2 &= b_2 + rb_1 + s \\
a_3 &= b_3 + rb_2 + sb_1 \\
&\vdots \\
a_k &= b_k + rb_{k-1} + sb_{k-2} \\
&\vdots \\
a_{n-1} &= R + rb_{n-2} + sb_{n-3} \\
a_n &= S + sb_{n-2}.
\end{aligned}
\qquad (6)
$$

Como se conocen las a_k y los valores de r y s, se puede despejar b_k en la ecuación intermedia, de donde se obtiene la fórmula recursiva:

$$b_k = a_k - rb_{k-1} - sb_{k-2} \qquad (\text{para } k = 3, 4, \ldots, n).$$

Adicionalmente, antes de poder empezar la recursión, se necesitan calcular los valores de b_1 y b_2, usando las ecuaciones:

$$
\begin{aligned}
b_1 &= a_1 - r \\
b_2 &= a_2 - rb_1 - s.
\end{aligned}
$$

Al final de cuentas se obtienen las siguientes ecuaciones, mediante las cuales se calculan los valores de las b_k:

$$
\begin{aligned}
b_1 &= a_1 - r \\
b_2 &= a_2 - rb_1 - s \\
b_3 &= a_3 - rb_2 - sb_1 \\
&\vdots \\
b_k &= a_k - rb_{k-1} - sb_{k-2} \\
&\vdots \\
b_{n-1} &= a_{n-1} - rb_{n-2} - sb_{n-3} \\
b_n &= a_n - rb_{n-1} - sb_{n-2}.
\end{aligned}
\qquad (7)
$$

Los valores de b_{n-1} y b_n en realidad no son coeficientes del cociente de la división, debido a que éste sólo tiene $n-2$ coeficientes, pero son cantidades matemáticas que permiten calcular los valores de R y S. Si se comparan las últimas dos ecuaciones del sistema (6) con las del sistema (7), se ve que se pueden despejar R y S, de donde se obtiene que:

$$
\begin{aligned}
R &= b_{n-1} \\
S &= b_n + rb_{n-1}.
\end{aligned}
$$

Estos son los valores de las funciones $R(r_0, s_0)$ y $S(r_0, s_0)$, requeridas por el algoritmo para resolver el sistema de ecuaciones (5). Los cálculos de las derivadas parciales R_r, R_s, S_r, y S_s, que también son necesarias para la resolución del sistema de ecuaciones, se desarrollan en la siguiente sección :

Cálculo de las derivadas parciales Rr, Rs, Sr y Ss

Se pueden calcular los valores de **Rr**, **Rs**, **Sr**, y **Ss** mediante las siguientes ecuaciones:

$$Rr = \frac{\delta b_{n-1}}{\delta r} = \frac{\delta\,[a_{n-1} - r b_{n-2} - s b_{n-3}]}{\delta r} = -b_{n-2} \qquad (8$$

$$Rs = \frac{\delta b_{n-1}}{\delta s} = \ldots$$

$$Sr = \frac{\delta b_{n}}{\delta r} + r\,\frac{\delta b_{n-1}}{\delta r} + b_{n-1} = \ldots$$

$$Ss = \frac{\delta b_{n}}{\delta s} + r\,\frac{\delta b_{n-1}}{\delta s} = \ldots.$$

Sin embargo, también se pueden obtener ecuaciones recursivas para el cálculo de estos valores, lo cual hace que se puedan realizar más fácil y rápidamente dentro de una computadora. Para simplificar la notación, de ahora en adelante se usarán las letras p_k y q_k para significar la derivada parcial de b_k con respecto a r y con respecto a s, respectivamente. Es decir,

$$p_k = \frac{\delta b_k}{\delta r} \quad y$$

$$q_k = \frac{\delta b_k}{\delta s}\,.$$

Si se derivan las b_k del sistema (7) con respecto a r y s, respectivamente, y se sustituyen en los resultados parciales las igualdades apropiadas de las que se muestran en el sistema (6), se obtienen las siguientes relaciones:

$$p_1 = \delta[a_1 - r]/\delta r = 0 - 1 = -1$$

$$p_2 = \delta[a_2 - r b_1 - s]/\delta r = 0 - \delta[r b_1]/\delta r - 0 = -\delta[r a_1 - r^2]/\delta r = -[a_1 - 2r] = -[b_1 + r - 2r] = -[b_1 - r] = r - b_1$$

$$p_3 = \delta[a_3 - r b_2 - s b_1]/\delta r = \ldots = -b_2 - r p_2 - s p_1$$

$$\vdots$$

$$p_k = \ldots = -b_{k-1} - r p_{k-1} - s p_{k-2}$$

$$\vdots$$

$$p_{n-1} = \ldots = -b_{n-2} - r p_{n-2} - s p_{n-3}$$

$$p_n = \ldots = -b_{n-1} - r p_{n-1} - s p_{n-2}$$

y también:

$$
\begin{aligned}
q_1 &= 0 \\
q_2 &= -1 \\
q_3 &= -b_1 - rq_2 - sq_1 \\
&\vdots \\
q_k &= -b_{k-2} - rq_{k-1} - sq_{k-2} \\
&\vdots \\
q_{n-1} &= -b_{n-3} - rq_{n-2} - sq_{n-3} \\
q_n &= -b_{n-2} - rq_{n-1} - sq_{n-2}.
\end{aligned}
$$

Después de calcular las p_k y q_k, se pueden calcular las derivadas parciales de la siguiente forma, usando la notación simplificada presentada anteriormente dentro de las ecuaciones del sistema (8):

$$
\begin{aligned}
Rr &= p_{n-1} \\
Rs &= q_{n-1} \\
Sr &= p_n + rp_{n-1} + b_{n-1} \\
Ss &= q_n + rq_{n-1}.
\end{aligned}
$$

Con estos valores y con los de **R** y **S** se pueden encontrar δr y δs, para luego obtener r_1 y s_1. Después, con estos nuevos valores de r y s, se efectúan todos los cálculos otra vez, para obtener r_2 y s_2, y así sucesivamente, hasta que se cumpla alguna condición de convergencia. Luego se extrae el factor cuadrático que se haya encontrado, y se empieza a ejecutar el algoritmo otra vez para el polinomio cociente, que es el polinomio cuyos coeficientes son las últimas b_k calculadas. Este procedimiento sigue hasta que ya se haya descompuesto el polinomio original en todos sus factores cuadráticos, y en uno lineal adicional en caso de que haya sido de grado impar, que es cuando ya se pueden calcular todas sus raíces.

Resumen del cómputo de las raíces de polinomios mediante el método de Lin-Bairstow

Inicio. Pedir datos e inicializar variables: leer, o asignar directamente, los valores del grado **N**, el criterio de convergencia ε, el número máximo de iteraciones para establecer la existencia de convergencia **LMAX**, y los valores iniciales estimados **ro** y **so** para los coeficientes del primer factor cuadrático a estimar. Leer los coeficientes a_k ($k = 1, 2, \ldots, N$) del polinomio $P_N(k)$. Fijar **m = 0** (**m** = número de factores cuadráticos extraidos), **j = 1** (**j** = número de pares de raíces obtenidas o en proceso de obtención hasta el momento).

Paso 1. Calcular el grado **n** del polinomio de entrada actual mediante **n = N - 2m**. Inicializar el contador de iteraciones **L = 0**. Inicializar **(r,s)** con los valores **(ro,so)**.

Paso 2. Revisar el valor de n. Si n>2, saltar al paso 3. En caso contrario, quiere decir que ya se obtuvieron todos los factores cuadráticos, y posiblemente uno lineal en caso de que N sea un número impar. En este caso, si n=2, saltar al paso 2b y si n < 2, saltar al paso 2a.

(a) Calcular la raíz rj del factor lineal que queda, $x + a_1$. Saltar al paso 9.

(b) Calcular las raíces rj y rj+1 del factor cuadrático que queda, $x^2 + a_1x + a_2$. Saltar al paso 9.

Paso 3. Dividir $P_N(x)$ entre $x^2 + rx + s$ y calcular los coeficientes del residuo R y S. Para ello, hacer:

$$b_1 = a_1 - r$$
$$b_2 = a_2 - rb_1 - s$$
$$b_k = a_k - rb_{k-1} - sb_{k-2} \qquad (k = 3, 4, \ldots, n).$$

Con estos valores, calcular:

$$R = b_{n-1}$$
$$S = b_n + rb_{n-1}.$$

Paso 4. Calcular las derivadas parciales Rr, Rs, Sr y Ss. Para ello, hacer:

$$P_1 = -1$$
$$P_2 = -b_1 + r$$
$$P_k = -b_{k-1} - rp_{k-1} - sp_{k-2} \qquad (k = 3, 4, \ldots, n) \quad y$$

$$q_1 = 0$$
$$q_2 = -1$$
$$q_k = -b_{k-2} - rq_{k-1} - sq_{k-2} \qquad (k = 3, 4, \ldots, n).$$

Usando estos valores, y los que se obtuvieron en la primera parte del paso 3, obtener:

$$Rr = p_{n-1}$$
$$Rs = q_{n-1}$$
$$Sr = p_n + rp_{n-1} + b_{n-1}$$
$$Ss = q_n + rq_{n-1}.$$

Paso 5. Encontrar δr y δs usando el sistema de ecuaciones:

$$Rr\delta r + Rs\delta s = -R$$
$$Sr\delta r + Ss\delta s = -S$$

de donde se pueden despejar δr y δs para finalmente usar:

$$\delta r = (-R*Ss+S*Rs) \div (Rr*Ss-Rs*Sr)$$
$$\delta s = (-Rr*S+Sr*R) \div (Rr*Ss-Rs*Sr).$$

Paso 6. Calcular los valores mejorados para r y s:

$$r = r + \delta r$$
$$s = s + \delta s.$$

Paso 7. Hacer prueba de convergencia:[8]

(a) Si $|\delta r| \leq \varepsilon$ y $|\delta s| \leq \varepsilon$, calcular las dos raíces r_j y r_{j+1} del cuadrático x^2+rx+s. Saltar al **paso 8**.

(b) En caso contrario, no ha habido convergencia todavía. Si **L** $\leq$ **LMAX**, incrementar **L** en 1 y saltar al **paso 3**. Si **L** $>$ **LMAX**, el algoritmo no converge bajo los límites fijados. Salir del programa y desplegar mensaje de falla de convergencia.

Paso 8. Ya hubo convergencia. Por lo tanto hay que reemplazar el polinomio $P_n(x)$ por $P_{n-2}(x)$, es decir, reemplazar las a_k por las b_k $(k=1,2,\ldots,n-2)$. Luego hay que incrementar el contador de factores cuadráticos **m** en 1 y el contador de pares de raíces **j** en 2, y regresar al **paso 1**.

Paso 9. Terminación exitosa del algoritmo. Desplegar los valores de las raíces.

Algoritmo de trazado del lugar geométrico y problemas de graficación.

Con todo lo que se ha expuesto, podemos mostrar el algoritmo de trazado del lugar geométrico.

Los coeficientes de P(S) serán almacenados en un arreglo P, mientras que los de KQ(S) serán almacenados en Q. Se usa un tercer arreglo I, para formar el polinomio intermedio. Suponemos también que los coeficientes que no existen tanto en P(S) como en KQ(S) se han ingresado como 0. Las raíces son almacenadas en un arreglo PR para la parte real y otro PIM para la parte imaginaria :

[8] En la implementación del módulo del lugar gométrico, se usó otro criterio de convergencia, basado en la suma de los valores absolutos de los diferenciales de r y s.

```
Procedimiento LGR (Orden : Entero; P,Q : Arreglo;
                   KMin, KMax, DeltaK : Entero);

J,K         : Entero;
Diverge     : Lógica;
I,PR,PIM    : Arreglo;

Comienzo

   EjesXY ;

{ Graficación de los polos del sistema                              }

   Bairnstow (Orden,P,PR,PIM,Diverge);
   Si No Diverge
      Para J = 1 Hasta Orden
         Polo (PR[J],PIM[J]);

{ Graficación de los ceros del sistema                              }

   Bairnstow (Orden,Q,PR,PIM,Diverge);
   Si No Diverge
      Para J = 1 Hasta Orden
         Cero (PR[J],PIM[J]);

   K = KMin

   Mientras K.<= KMax

      Comienzo
      Para J = 1 Hasta Orden
         I[J] = P[J] + K*Q[J];

      Bairnstow (Orden,I,PR,PIM,Diverge);

      Si No Diverge
         Para J = 1 Hasta Orden
            Punto (PR[J],PIM[J]);
      K = K + DeltaK

      Fin

Fin
```

Al final de este capítulo de puede observar el lugar geométrico obtenido por este método y con el polinomio mostrado en el capítulo anterior.

Se tiene tan sólo un problema que resolver. Al graficar los ejes en la pantalla, si una solución cae sobre uno de ellos, el punto graficado no se verá. Además al obtener una solución para una K, ¿Cómo se va a saber con

que puntos hay que unir para formar el LGR?.

Por ejemplo, supóngase un polinomio de cuarto orden. Se obtiene entonces su primer conjunto de soluciones. Al obtener el segundo conjunto, que también contiene cuatro soluciones, no hay manera de saber si una raíz A, debe unirse con la A del conjunto anterior o con la B.

Se analizaron algunos métodos para dar solución a este problema, ya que el LGR se veía como un conjunto de puntos y no como una línea continua, sin embargo dichas soluciones mostraron ser más complicadas que eficaces.

La solución más adecuada, (y más sencilla), fue el simplemente graficar un "puntote" en vez de un solo "pixel".[9] Con esto el lugar geométrico se ve continuo y no se tiene necesidad de hacer ningún cálculo para analizar la vecindad del punto graficado.

Regiones de trabajo.

El LGR es trazado en el plano complejo como se muestra :

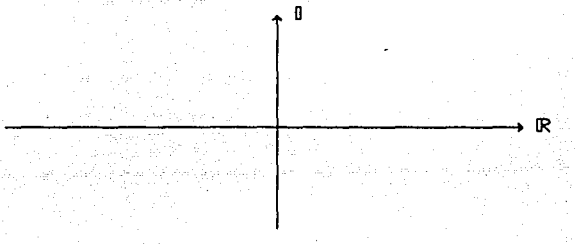


FIGURA 4.2. PLANO COMPLEJO.

Hay ciertas regiones del plano complejo, que cuando tiene trazado un LGR representan diversos parámetros del sistema.

Todo el semiplano izquierdo del plano, representa la zona de

[9] PIXEL es una contracción de "picture element" y se refiere al punto más pequeño que puede direccionarse dentro del video de una computadora.

estabilidad del sistema y el eje imaginario, la zona de oscilación.

Una circunferencia de radio ωn, representa la frecuencia de amortiguamiento del sistema, y las rectas $y = \pm \tan\lambda x$ representan el coeficiente de amortiguamiento relativo (λ) del sistema. A su vez la recta $x = -\beta$ representa el tiempo de asentamiento del sistema.

Una zona de trabajo está determinada por los parámetros de diseño del sistema. Podría especificarse un cierto tiempo de asentamiento, una frecuencia de amortiguamiento y un coeficiente de amortiguamiento. Con todo lo anterior, se debe asegurar que el LGR pase dentro de esta región de trabajo, de lo contrario el sistema no cumplirá con las especificaciones de diseño.

Regiones de trabajo dentro del módulo de LGR.

El módulo presentado cuenta con el soporte a las regiones de trabajo. Una vez que el LGR ha sido trazado, el módulo pide una frecuencia y coeficiente de amortiguamiento seguido de un tiempo de asentamiento.

Seguido de esos datos, se dibujará la región de trabajo formada por dichos parámetros. Estos parámetros no son estáticos ya que el módulo permite su modificación de manera visual.

Al decir visual, se entiende que si por ejemplo se ha elegido a la frecuencia de amortiguamiento como el parámetro que se desea alterar, al presionar una tecla, la circunferencia asociada a dicha frecuencia se hará de mayor o menor tamaño según desee el usuario. Lo mismo puede hacerse con el coeficiente de amortiguamiento relativo y el tiempo de asentamiento.

El módulo cuenta con ciertas características muy útiles :

- Conforme avanza el trazado del LGR, el procedimiento imprime en la pantalla el valor de K en ese momento del cálculo, por lo que el usuario puede observar a que valores de K le corresponden los puntos dentro del LGR.

- Si existe una K de particular interés, el usuario con sólo presionar una tecla, detiene el trazado del LGR, con lo que puede observar el valor de K y su influencia en el LGR. Al oprimir otra tecla se continua el trazado si no fue la tecla ESC.[10] Si fue dicha tecla se procede al ingreso de los parámetros de diseño.

- Aparecéra un cursor en forma de flecha ( ), una vez terminado el trazado. El usuario puede mover este cursor y cuando lo coloque sobre algún punto perteneciente al LGR, el procedimiento mostrará el valor de K correspondiente a dicho punto. Si el usuario desea calcular el valor de K, aun cuando el LGR no pase por el punto en el que se encuentra el cursor, lo puede hacer.

Con todo lo anterior se puede ver que el diseño de sistemas puede facilitarse enormente. Si el usuario ha trazado una región de trabajo, sólo para descubrir que el LGR no pasa por dicha región, puede crear una visualmente por la que sí pase, y evaluar de que manera influye la nueva región en el diseño original.

Puede asimismo ver los valores de K que hacen el sistema inestable u oscilante, por medio del cursor. O puede combinar el uso del cursor y el de la región de trabajo para mejorar un diseño o simplemente para analizarlo.

Comentarios.

Se ha visto que el método del LGR proporciona mucha información acerca de la dinámica del sistema. En el capítulo se cubrió la de más uso y bajo la suposición de que el parámetro de interés es la ganancia de la planta. En la mayoría de los casos es así, pero hay algunos otros en los que busca analizar otros parámetros del sistema.

Se mostró el método analítico y se mencionó la existencia de reglas de

[10] ESC. Escape. Es prácticamente un estándar en computación que esta tecla termine la ejecución de la labor que se está realizando.

trazado cuando no es posible obtener la solución analítica. En ambos casos se insistió en la necesidad de cálculos para la obtención del LGR.

Finalmente se dio un bosquejo del método de Bairnstow y de su aplicación del método de solución, y se explicó el porque de implementarlo en una computadora.

Finalmente se habló de las regiones de trabajo y sus aplicaciones en el diseño de sistemas, así como de las facilidades que otorga el módulo para el análisis del lugar geométrico y de la región de trabajo.

Conclusiones.

El método del lugar geométrico de las raíces es usada hoy en día para el diseño de sistemas. Este diseño es apoyado por las regiones de trabajo : zonas dentro del plano complejo que reflejan un parámetro de diseño.

El trazado del LGR es una labor pesada, sin embargo al apoyarse en el método de Barinstow y en el método de solución, se vuelve una tarea rutinaria, que deja al diseñador más tiempo para su trabajo.

El módulo de LGR puede insertarse en el módulo principal del sistema o usarse como un programa por sí mismo.

Conclusiones finales.

Se puede afirmar que el objetivo primordial de esta tesis fue alcanzado.

Se mostraron técnicas de desarrollo de *software* y se implementaron módulos apegándose a dichas técnicas; obteniéndose en todos los casos un desempeño satisfactorio del módulo.

Asimismo, el esquema de programación creado por el Ing. Francisco Rodríguez y yo, también cumplió sus objetivos, al permitir el desarrollo de

todo el proyecto de control, de manera concurrente.

Una conclusión muy importante, es que el método de solución para trazado de lugares geométricos, es aceptable siempre y cuando el algoritmo de solución de polinomios, sea lo suficientemente rápido.

Tenemos pues, un programa de control, robusto, y con una buena interfaz de usuario. El programa cumple con dos tareas importantes : una a nivel didáctico, al ofrecer a los alumnos de la Facultad de Ingeniería una herramienta lo suficientemente poderosa como para poder afrontar problemas de control más apegados a la realidad, sin la necesidad de realizar laboriosos cálculos. Aparte de que los diferentes módulos de diseño, le permitirán a los alumnos analizar y mejorar diseños hechos en clase, dándoles así, un mejor panorama del problema de control.

De la misma manera que los estudiantes, el profesorado de control podrá exigir trabajos de mayor calidad al contar con este programa. En lo que respecta al módulo de trazado de lugares geométricos junto con las regiones de trabajo, facilitará la tarea de elegir entre varios controladores para una planta, al permitir la obtención de los lugares geométricos rápidamente.

La otra tarea es a nivel de trabajo profesional. El ingeniero que ya domine las técnicas de control, encontrará en este programa una herramienta igualmente poderosa que en el caso de un estudiante, pero con la diferencia de que podrá explotar más sus capacidades.

Una experiencia importante que muestra el desarrollo de este programa, es que no necesariamente se deben usar los últimos lenguajes de programación para desarrollar un buen programa. El lenguaje Pascal es bastante antiguo. Prácticamente el instituto ANSI[11], sólo se ha preocupado por el desarrollo del lenguaje C, y a pesar de que existe una definición ANSI de Pascal, ésta ha sufrido mínimas modificaciones. Los que parecen haber tomado el estándar de programación en Pascal, son la empresa Borland. De hecho, en aquellos tiempos de Turbo Pascal 1.0, ellos se anunciaban como la empresa que "le dió vida a Pascal en los sistemas PC", y yo creo que

[11] ANSI. American National Standards Institute.

tienen razón.

El Pascal es un lenguaje del que muchos tienen la idea de que sirve para aprender la programación estructurada y nada más. Una vez entendida ésta, se olvidan de Pascal y buscan otros lenguajes. También se ha dicho que la programación estructurada fue el estándar de programación de los 80, y que la programación orientada a objetos será, (o es), la de los 90. De ahí muchos concluyen que Pascal es obsoleto y que no se puede hacer ya nada con él. El programa de control ha demostrado que lo anterior es falso. Un buen programador hará excelentes programas en BASIC[12], y un mal programador, ni siquiera en Oberon[13] hará un buen programa. Además hay que recordar que la programación estructurada y la programación orientada a objetos, son disciplinas y no técnicas.

Esta tesis quizá persiguió objetivos mixtos : uno de ellos fue el de mostrar de manera sencilla, algunas de las técnicas de diseño de *software* que existen en la actualidad. El otro objetivo fue el de aplicar esas técnicas a un problema de ingeniería como lo es el problema de control.

Ambos objetivos fueron alcanzados, pero aun queda mucho por hacer. En el campo de diseño de *software*, todavía quedan muchas nuevas metodologías que analizar y usar, y en el campo de control, hay muchos algoritmos de control que no han sido implementados en computadora, los cuales quizá requieran de nuevos esfuerzos por parte del *software*.

Llegamos así al final de esta tesis. Espero que el tiempo que dediqué a la realización de la misma, sea útil tanto a alumnos y a profesores de la Facultad de Ingeniería, a la que le quedo eternamente agradecido por la formación profesional que me regaló, y a la que espero poder corresponder de alguna manera algún día.

Agradezco también al Ing. Francisco Rodríguez, el haberme soportado durante el tiempo en que elaboramos esta tesis.

[12] El programa de control CC, que es de los mejores (o quizá el mejor), en la actualidad, está hecho en BASIC.

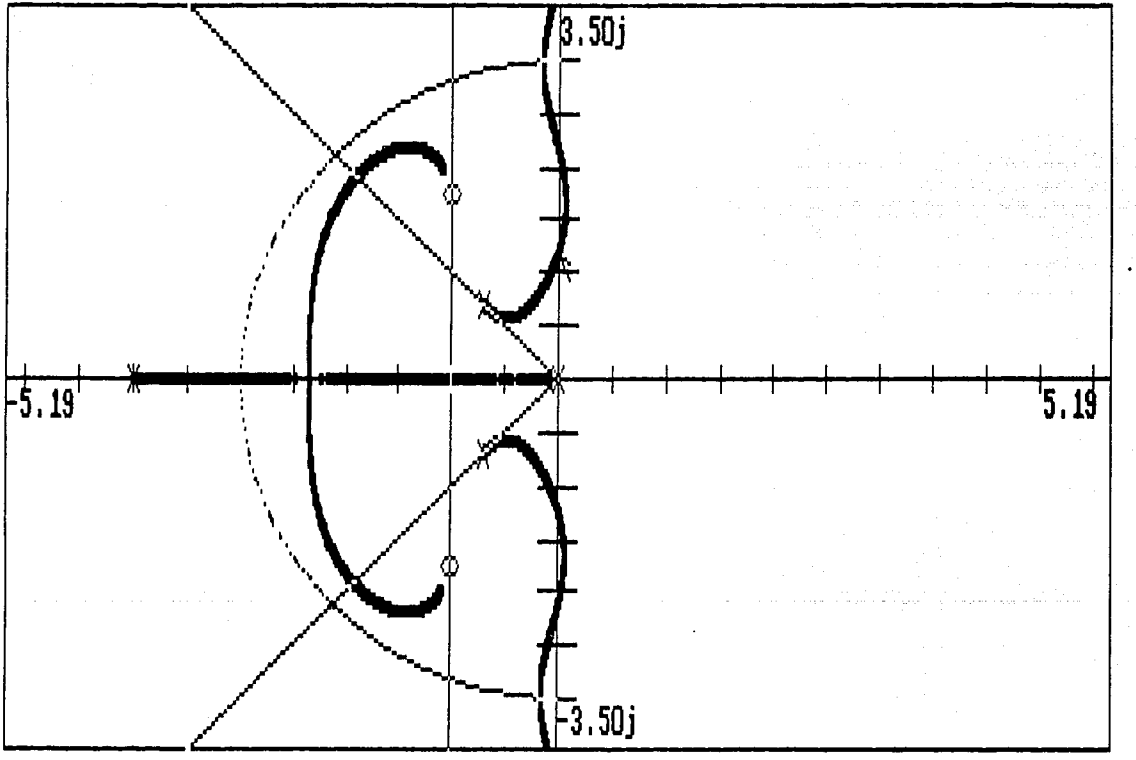
[13] Oberon es el último lenguaje de Niklaus Wirth y es el sucesor de Modula-2, que a su vez es el sucesor de Pascal. Wirth clama que Oberon es la respuesta a un dialecto de Modula-2, llamado Modula-R.

Agradezco también al Ing. Francisco Rodríguez, el haberme soportado durante el tiempo en que elaboramos este trabajo.

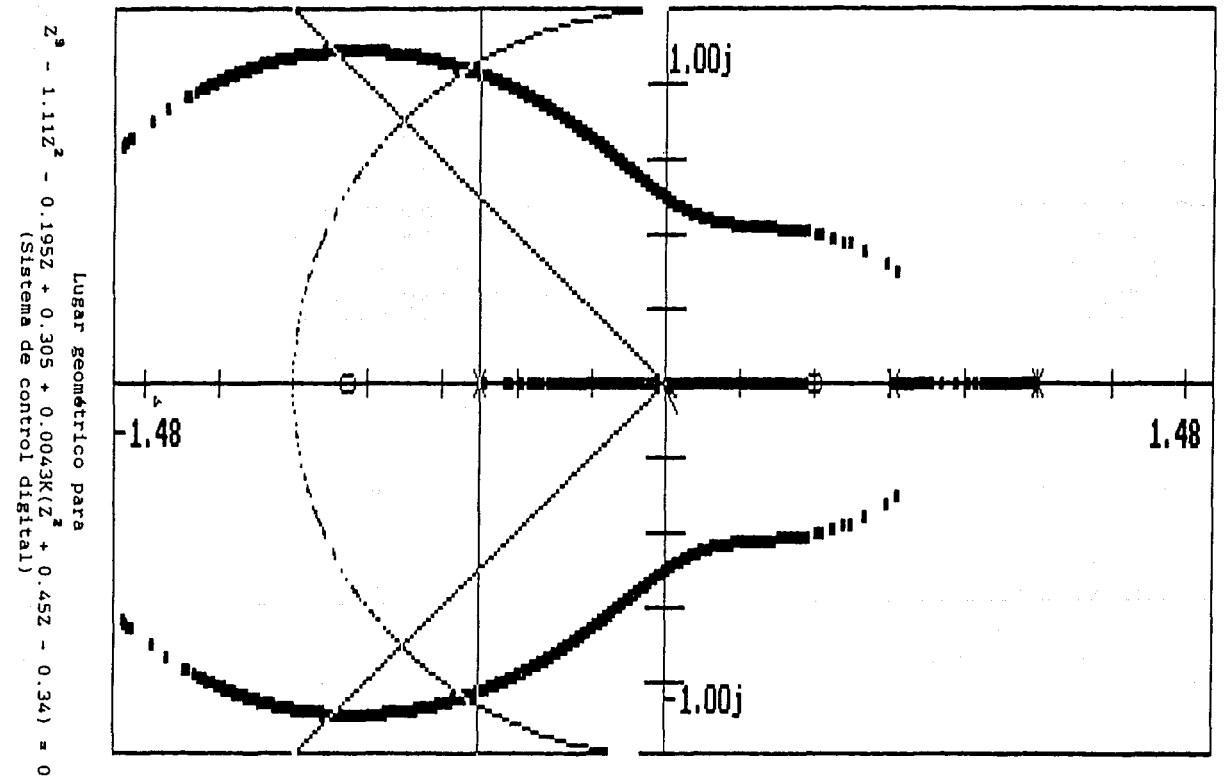
Al realizar esta tesis ya he estado sumergido en el campo profesional por más de un año, y al convivir y competir con y contra ingenieros de otras universidades, he visto que nuestra facultad está en un muy buen nivel académico, por eso, no me cansaré de decir, ¡Gracias Facultad de Ingeniería!, ¡Gracias UNAM!, y sobre todo ¡Gracias México!.

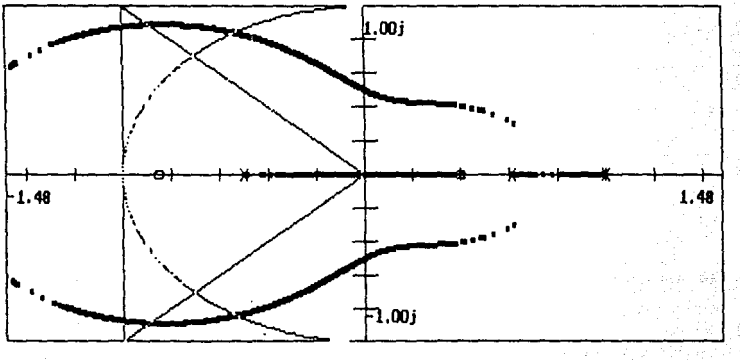
Alvaro Castiello de la Hidalga.
México, D.F, 1991.

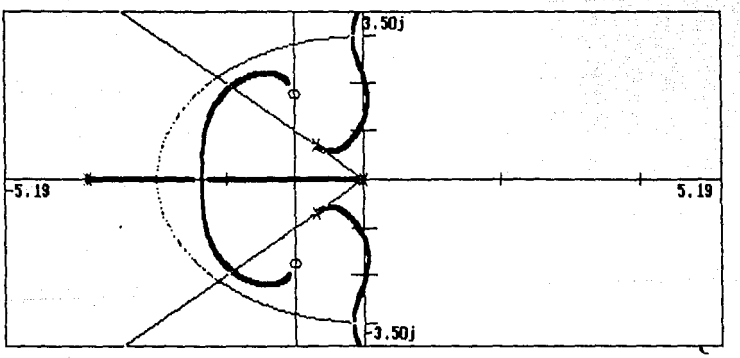
Lugar geometrico para K = [0;1500]. Cota del eje real = 0.50, Imaginario = 0.50
K = 14.103468 - 0.431594j Moviendo K



Lugar geométrico para
$S^3 + 11.4S^4 + 39S^3 + (43.6)S^2 + (24 + 2K)S + 4K = 0$


Lugar geometrico para K = [0;950]. Cota del eje real = 0.20, Imaginario = 0.20
K = 0.897036 - 0.006813j
Moviendo K
1.00j
-1.48
1.48
-1.00j
Lugar geométrico para
$z^3 - 1.11z^2 - 0.195z + 0.305 + 0.0043K(z^2 + 0.45z - 0.34) = 0$
(Sistema de control digital)


1.00j
1.48
1.48
-1.00j


3.50j
5.19
5.19
-3.50j

Los diferentes módulos del sistema tendrán que intercambiar información entre ellos. La mayoría de las veces, esta información será una función tabulada, es decir, una variable independiente, y una o más variables dependientes.

El intercambiar estas funciones a través de variables en memoria, complica el esquema de ranuras expuesto en el capítulo uno, ya un módulo debe conocer la estructura y la dirección, (identificador o nombre), de la variable para poder realizar dicho intercambio.

La manera más sencilla de hacer esto, es a través de un archivo que contenga toda la función tabulada.

Archivos binarios y archivos de texto.

En computación existen básicamente dos tipos de archivos : los binarios y los de texto.

Los binarios, como su nombre lo indica, contienen información codificada en binario, por lo que dichos archivos sólo pueden ser interpretados, leídos y escritos por el programa en cuestión.

Los archivos de texto por otra parte, contienen la información en modo textual, por lo que la información es interpretable tanto por la computadora como por una persona.

La principal desventaja de los archivos de texto, es que sólo son accesables en forma secuencial : no se puede releer una línea de texto que ha sido leída, sino que se debe volver a leer desde el inicio del archivo, todas las líneas que estén antes de ella para volverla a leer. Sin embargo, en el proyecto, no se requerirá de accesos no secuenciales, por lo que esta

desventaja no lo afecta.

Asimismo, estos archivos tienen la grandísima ventaja de que pueden ser modificados sin la necesidad el programa que los creó, ya que cualquier editor de texto estándar hará dicha labor sin ningún problema.

Formato de archivos.

Así pues, el intercambio de información se hará a través de archivos de texto estándar o archivos ASCII, como se les conoce comúnmente.

Este archivo tendrá el siguiente formato : Las primeras líneas contendrán cualquier información que el módulo desee colocar. Dicha información puede ser la fecha y hora de creación, una explicación del contenido del archivo, el nombre del módulo, etc. A esto se le llamará el encabezado del archivo.

Seguido el encabezado, seguirá una línea que marque el inicio de los datos. Esta línea de inicio, deberá contener la línea "INICIO:". Seguida de esta marca, la siguiente línea indicará el número de variables dependientes para cada valor de la variable independiente.

Por ejemplo, un archivo que contuviera la tabulación de la función $f(x) = x^2$, podría contener los siguiente :

```
Archivo : PARABOLA.FX
Módulo de evaluación de funciones especiales.
2-Junio-1991. 7:28 PM.
INICIO:
1
1     1
2     4
3     9
4     16
5     25
  .
  .
  .
```

Si se creara un archivo que contuviera la señal de salida y la señal

de error de un sistema, el archivo de intercambio podría ser :

```
Fecha : 18-Mayo-91        Hora : 19:30 PM.
Evaluación de señal de error y de salida para el controlador
de la planta de vapor.
Módulo de análisis de error en el tiempo.
INICIO:
2
0.0        10.2        0.2
0.1        10.4        0.4
0.2        10.6        0.6
0.3         9.8       -0.2
0.4         9.6       -0.4
0.5         9.2       -0.8
0.6        10.0        0.0
.
.
.
```

Como puede verse, el mantenimiento de estos archivos se realizará con mucha facilidad, incluso pueden hacerse modificaciones a los archivos, sin la necesidad de volver a ejecutar el módulo que los creó.

Se consultaron los siguientes libros para la elaboración de esta tesis :

Andrés. G. de Silva F.
Tesis Profesional.
Facultad de Ingeniería, UNAM.
México, D.F., México, 1991.

Ogata Katsuhiko
Ingeniería de control moderna.
Prentice Hall 1970.
ISBN 968-880-018-X.
p.p. 343-345, 376.

En este apéndice se muestran los listados de los programas creados para esta tesis.

```
{ $A-,B-,D-,E+,F-,I-,L-,N-,O-,R-,S-,V-}
{ $M 16384,30000,30000}
Program LGR ;

Uses
  Crt,
  TIDef,
  Menus,
  Math,
  LGRInt,
  Graph;

{$Define DEMO }

Const
  Epsilon        =  0.0001;
  Bottom         =       1;
  Space80        =                                                    '+'M;
  AxisStyle      = SolidLn;
  AxisPat        =   $0000;
  ScaleStyle     = SolidLn;
  Polo           =       1;
  Cero           =       2;
  MaxN           =      30;

Type
  Roots        = Record
                   C,K   : Coeff
                 End;
  ProgRealType = MathRealType;
  CProc        = Procedure (X,Y,Yc   : Integer);
  CompareFunc  = Function  (Var X,Y) : ShortInt;

Const

{- Variables de graficación ---------------------------------------------}

  KMax           : Integer      =   10;
  KMin           : Integer      =    1;
  DK             : Integer      =    1;
  XMin           : ProgRealType = -1.0;
  Xmax           : ProgRealType =  1.0;
  YMax           : ProgRealType =  1.0;
  YMin           : ProgRealType = -1.0;

Var
  Fin                    : Boolean;
  Poly                   : Roots;
  A,PolyReal,PolyImag    : Coeff;
  UA,N,BottomLine        : Byte;
  PointColor,ARx,ARy     : Word;
{  TenOfW,}
  AR,P,K,Coeff           : Real;
  Code,
  GraphDriver,
  GraphMode              : Integer;
  AMX,AMY,
  Px,Py,
  DeltaX,DeltaY          : ProgRealType;
```

```
  Cadena                : TString;
{ Last                  : Array [1..MaxPolyDegree] Of Record
                                                     R,C   : Integer
                                                     End;}

{+-------------------------------------------------------------------------+}
{! Función para calcular la X en pantalla                                  !}
{+-------------------------------------------------------------------------+}
Function XP (X : ProgRealType) : Integer;

Begin
  XP := Round((X - XMin)/Px)
End;


{+-------------------------------------------------------------------------+}
{! Función para calcular la X real                                         !}
{+-------------------------------------------------------------------------+}
Function XR (X : Integer) : ProgRealType;

Begin
  XR := Px*X + XMin
End;


{+-------------------------------------------------------------------------+}
{! Función para calcular la Y en pantalla                                  !}
{+-------------------------------------------------------------------------+}
Function YP (Y : ProgRealType) : Integer;

Begin
  YP := Round((YMax - Y)/Py)
End;

{+-------------------------------------------------------------------------+}
{! Función para calcular la Y en pantalla                                  !}
{+-------------------------------------------------------------------------+}
Function YR (Y : Integer) : ProgRealType;

Begin
  YR := YMax - Py*Y
End;


{+-------------------------------------------------------------------------+}
{! Procedimiento para averiguar las dimensiones del puerto de visión       !}
{+-------------------------------------------------------------------------+}
Procedure GetViewDimensions (Var Width, Large : Word);

Var
  VP   : ViewPortType;

Begin
  GetViewSettings(VP);
  With VP Do
     Begin
     Width := X2 - X1 + 1;
     Large := Y2 - Y1 + 1
     End
End;
```

```
120:
121: {+-------------------------------------------------------------------+}
122: {| Procedimiento para borrar la segunda linea                        |}
123: {+-------------------------------------------------------------------+}
124: Procedure EraseTop ;
125:
126: Begin
127:   GotoXY(1,Bottom+1);
128:   Write(Space80)
129: End;
130:
131: {+-------------------------------------------------------------------+}
132: {| Procedimiento para dibujar polos y ceros                          |}
133: {+-------------------------------------------------------------------+}
134: Procedure PlotXC (X,Y,Mag,What : Integer);
135:
136: Begin
137:   If What = Polo Then
138:      Begin
139:      Line (X - Mag,Y - Mag,X + Mag,Y + Mag);
140:      Line (X - Mag,Y + Mag,X + Mag,Y - Mag)
141:      End
142:   Else
143:      Circle (X,Y,Mag)
144:
145: End;
146:
147: {$F+,L Circle}
148: {+-------------------------------------------------------------------+}
149: {| Procedimiento de circunferencia de Bresenham                      |}
150: {+-------------------------------------------------------------------+}
151: Procedure BCircle (X,Y,Radius : Integer; CP : CProc);          External;
152:
153: {+-------------------------------------------------------------------+}
154: {| Procedimiento para prender los pixeles                            |}
155: {+-------------------------------------------------------------------+}
156: Procedure PutPixels (X,Y,Yc : Integer) ;                       External;
157: {$F-}
158:
159: {+-------------------------------------------------------------------+}
160: {| Procedimiento para analizar el lugar geomtrico                    |}
161: {+-------------------------------------------------------------------+}
162: Procedure Atiende ;
163:
164: Const
165:   DeltaX = 10;
166:   DeltaY = 10;
167:   MoveK  =  0;
168:   MoveW  =  1;
169:   MoveA  =  2;
170:   MoveTs =  3;
171:
172: Var
173:   C          : Tecla;
174:   W,L,
175:   X1,Y1,
176:   X,Y        : Word;
177:   InMove     : Byte;
178:   R,C1,C2    : Complex;
179:   Buff       : String;
```

```
180:   WAbs,
181:   Ang,
182:   Alfa,
183:   Wm,Amort  : ProgRealType;
184:   Proc      : Procedure;
185:   V         : ViewPortType;
186:   X0,Y0     : Integer;
187:
188:    {+-------------------------------------------------------------------+}
189:    {| Procedimiento para borrar la flecha                               |}
190:    {+-------------------------------------------------------------------+}
191:   Procedure EraseArrow ;
192:
193:   Begin
194:     PutImage(X,Y,Buff,NormalPut)
195:   End;
196:
197:    {+-------------------------------------------------------------------+}
198:    {| Procedimiento para dibujar la flecha                              |}
199:    {+-------------------------------------------------------------------+}
200:   Procedure DrawArrow ;
201:
202:   Begin
203:
204:     GetImage(X,Y,X+6,Y+6,Buff);
205:     Line(X,Y,X+6,Y-2);
206:     Line(X,Y,X,Y+5);
207:     Line(X,Y,X+6,Y+6);
208:
209:     Repeat Until KeyPressed;
210:     EraseArrow
211:
212:   End;
213:
214:    {+-------------------------------------------------------------------+}
215:    {| Procedimiento para dibujar el circulo de Wn                       |}
216:    {+-------------------------------------------------------------------+}
217:   Procedure DrawWn (Wn : PrograalType);
218:
219:   Begin
220:     Circle (X0,Y0,Xp (XMin + Wm),PutPixels)
221:   End;
222:
223:    {+-------------------------------------------------------------------+}
224:    {| Procedimiento para dibujar las rectas de amortiguamiento          |}
225:    {+-------------------------------------------------------------------+}
226:   Procedure DrawAmort (Amort : PrograalType);
227:
228:   Var
229:     X     : Integer;
230:     Ang   : ProgRealType;
231:
232:   Begin
233:     Amort := Amort*Pi/180;
234:     X     := Xp (XMin);
235:     Ang   := XMin*Sin(Amort)/Cos(Amort);
236:
237:     Line (X0,Y0,X, Yp ( Ang));
238:     Line (X, Yp (-Ang),X0,Y0)
239:   End;
```

```
240:
241:    {+--------------------------------------------------------------------+}
242:    {! Procedimiento para dibujar la recta de ts                         !}
243:    {+--------------------------------------------------------------------+}
244:    Procedure DrawTs (Alfa : ProgRealType);
245:
246:    Var
247:      X   : Integer;
248:
249:    Begin
250:      X := Xp (Alfa);
251:      Line (X,Yp (YMin),X,Yp (YMax))
252:    End;
253:
254:    {+--------------------------------------------------------------------+}
255:    {! Procedimiento para mover K                                         !}
256:    {+--------------------------------------------------------------------+}
257:    {$F+}
258:    Procedure _MoveK ;
259:    {$F-}
260:
261:    Var
262:      Flag   : Boolean;
263:
264:    Begin
265:
266:      Flag := False;
267:
268:      Case C.Codigo Of
269:         UARR      : If Y <> 0 Then
270:                        Dec(Y);
271:         DARR      : If Y <  L Then
272:                        Inc(Y);
273:         RARR      : If X <  W Then
274:                        Inc(X);
275:         LARR      : If X <> 0 Then
276:                        Dec(X);
277:         ALTX      : X := W Div 2;
278:         ALTY      : Y := L Div 2;
279:         HOME      : Y := 0;
280:          END      : Y := L;
281:         CTRLHOME  : X := 0;
282:         CTRLEND   : X := W;
283:         PGUP      : If INTEGER(Y-DeltaY) > 0 Then
284:                        Dec(Y,DeltaY);
285:         PGDN      : If Y+DeltaY <  L Then
286:                        Inc(Y,DeltaY);
287:         CTRLRARR  : If X+DeltaX <  W Then
288:                        Inc(X,DeltaX);
289:         CTRLLARR  : If INTEGER(X-DeltaX) > 0 Then
290:                        Dec(X,DeltaX);
291:         ALTK      : Flag := True
292:         End;
293:
294:      EraseToo ;
295:
296:      If (GetPixel(X,Y) = PointColor) Or Flag Then
297:         Begin
298:
299:         R.Pr  := (R (X);
```

```
300:        If Abs(R.Pr) < Epsilon Then
301:           R.Pr := 0.0;
302:
303:        R.Pim := YR (Y);
304:        If Abs(R.Pim) < Epsilon Then
305:           R.Pim := 0.0;
306:
307:        PolyEval  (N,Poly.C,R,C1);
308:        PolyEval  (N,Poly.K,R,C2);
309:        Conjugado (C2,C2);
310:        RMul      (C2,1.0/Sqr(Mag(C2)));
311:        CMul      (C1,C2,C1);
312:        RMul      (C1,-1.0);
313:
314:        Write('K = ',C1.Pr:0:6,' ');
315:
316:        If C1.Pim > 0 Then
317:           Write('+ ',C1.Pim:0:6,'j')
318:        Else
319:           If C1.Pim < 0 Then
320:              Write('- ',-C1.Pim:0:6,'j')
321:        End
322:
323:     Else
324:        Writeln('Coloca el cursor sobre el LGR')
325:
326:  End;
327:
328:  {+-------------------------------------------------------------------------+}
329:  {| Procedimiento para mover Wn                                             |}
330:  {+-------------------------------------------------------------------------+}
331:  {$F+}
332:  Procedure _MoveW ;
333:  {$F-}
334:
335:  Begin
336:
337:    DrawWN (Wm);
338:
339:    Case C.Codigo Of
340:       LARR,
341:       UARR      : If Wm + 1.0 <= WAbs Then
342:                      Wm := Wm + 1.0;
343:       RARR,
344:       DARR      : If Wm - 1.0 >  0.0 Then
345:                      Wm := Wm - 1.0;
346:
347:       PGUP      : If Wm + 0.1 <= WAbs Then
348:                      Wm := Wm + 0.1;
349:
350:       PGDN      : If Wm - 0.1 >  0.0 Then
351:                      Wm := Wm - 0.1
352:       End;
353:
354:    DrawWN (Wm)
355:
356:  End;
357:
358:  {+-------------------------------------------------------------------------+}
359:  {| Procedimiento para mover el angulo                                      |}
```

```
360:    {+-------------------------------------------------------------------+}
361:    {$F+}
362:    Procedure _MoveA ;
363:    {$F-}
364:
365:    Begin
366:
367:      DrawAmort (Amort);
368:
369:      Case C.Codigo Of
370:         LARR,
371:         DARR      : If Amort - 5.0 >=  0.0 Then
372:                        Amort := Amort - 5.0;
373:         RARR,
374:         UARR      : If Amort + 5.0 < 90.0 Then
375:                        Amort := Amort + 5.0;
376:         PGDN      : If Amort - 1.0 >=  0.0 Then
377:                        Amort := Amort - 1.0;
378:         PGUP      : If Amort + 1.0 < 90.0 Then
379:                        Amort := Amort + 1.0
380:         End;
381:
382:      DrawAmort (Amort)
383:
384:    End;
385:
386:    {+-------------------------------------------------------------------+}
387:    {| Procedimiento para mover alfa                                     |}
388:    {+-------------------------------------------------------------------+}
389:    {$F+}
390:    Procedure _MoveTs ;
391:    {$F-}
392:
393:    Begin
394:
395:      DrawTs (Alfa);
396:
397:      Case C.Codigo Of
398:         RARR      : If Alfa + 1.0 <= 0 Then
399:                        Alfa := Alfa + 1.0;
400:         LARR      : Alfa := Alfa - 1.0;
401:         CTRLRARR  : If Alfa + 0.1 <= 0 Then
402:                        Alfa := Alfa + 0.1;
403:         CTRLLARR  : Alfa := Alfa - 0.1
404:         End;
405:
406:      DrawTs (Alfa)
407:
408:    End;
409:
410:
411: Begin
412:
413: {- Leer las coordenadas de la ventana ------------------------------------}
414:
415:   GetViewSettings(V);
416:
417: {- Obtener las dimensiones "pixelianas" de la ventana ---------------------}
418:
419:   GetViewDimensions (W,L);
```

```
420:
421: (- Empezar moviendo a K ------------------------------------------------)
422:
423:    InMove :=   MoveK;
424:   @Proc   := @_MoveK;
425:   X       := Xp (0.0);
426:   Y       := Yp (0.0);
427:   X0      := X;
428:   Y0      := Y;
429:   SetWriteMode(XorPut);
430:
431: (- Dibujar el círculo de Wm -----------------------------------------------)
432:
433:   EraseTop ;
434:   Write('Wm-> ');
435:   Read(Wm);
436:   DrawWn (Wm);
437:   WAbs := Abs(XMin);
438:
439: (- Dibujar las rectas del angulo ------------------------------------------)
440:
441:   EraseTop ;
442:   Write('Amortiguamiento (angulo) -> ');
443:   Read(Amort);
444:   DrawAmort (Amort);
445:
446: (- Dibujar la línea de ts -------------------------------------------------)
447:
448:   Alfa := -1.0;
449:   DrawTs (Alfa);
450:
451: (- Iterar mientras no se presione ESC -------------------------------------)
452:
453:   Repeat
454:
455:      GotoXY(40,Bottom+1);
456:      Write('Moviendo ');
457:      Case InMove Of
458:         MoveK   : Writeln('K             ');
459:         MoveW   : Writeln('Wn            ');
460:         MoveA   : Writeln('Amortiguamiento ');
461:         MoveTs  : Writeln('ts            ')
462:         End;
463:
464:      DrawArrow ;
465:
466:      LeeT (C);
467:
468:      If C.Codigo = ALTM Then
469:         Case InMove Of
470:            MoveK   : Begin
471:                        InMove :=   MoveW;
472:                      @Proc    := @_MoveW;
473:                      End;
474:            MoveW   : Begin
475:                        InMove :=   MoveA;
476:                      @Proc    := @_MoveA;
477:                      End;
478:            MoveA   : Begin
479:                        InMove :=   MoveTs;
```

```
480:                    @Proc   := @_MoveTs
481:                    End;
482:          MoveTs  ; Begin
483:                      InMove :=   MoveK;
484:                    @Proc   := @_MoveK
485:                    End
486:          End;
487:
488:     InLine($55);
489:     Proc (;
490:
491:     If InMove <> MoveK Then
492:        Begin
493:        DrawWn    (Wm);
494:        DrawAmort (Amort);
495:        DrawTs    (Alfa)
496:        End
497:
498:  Until C.Ascii = ESC
499:
500: End;
501:
502:
503: {+-------------------------------------------------------------------+}
504: {! Este procedimiento inicializa las variables                      !}
505: {+-------------------------------------------------------------------+}
506: Procedure InitValues ;
507:
508: Var
509:   Xw,Yw   : Word;
510:
511: Begin
512:   AMX := XMax - XMin;
513:   AMY := YMax - YMin;
514:   GetViewDimensions (Xw,Yw);
515:   Px  := AMX/Xw;
516:   Py  := AMY/Yw
517: End;
518:
519: (*
520: {+-------------------------------------------------------------------+}
521: {! Este procedimiento grafica un punto, con el color pasado          !}
522: {+-------------------------------------------------------------------+}
523: Procedure PlotPoint (X,Y : ProgRealType; Kolor: Word) ;
524:
525: Begin
526:   PutPixel(XP (X),YP (Y),Kolor)
527: End;
528: *)
529:
530: {+-------------------------------------------------------------------+}
531: {! Procedimiento para dibujar una ventana                            !}
532: {+-------------------------------------------------------------------+}
533: Procedure ViewPort (X1,Y1,X2,Y2 : Word) ;
534:
535: Var
536:   Xw,Yw   : Word;
537:
538: Begin
539:   SetViewPort(X1,Y1,X2,Y2,ClipOn);
```

```
540:   GetViewDimensions (Xw,Yw);
541: {  TenOfW := Xw*0.05;}
542: {  P      := Sqrt(2)*(GetMaxY+1)/(Y2 - Y1);}
543:   Rectangle(0,0,Xw-1,Yw-1)
544: End;
545:
546:
547: {+-------------------------------------------------------------------+}
548: {! Procedimiento para dibujar una línea con el estilo pasado        !}
549: {+-------------------------------------------------------------------+}
550: Procedure LineStyle (X1,Y1,X2,Y2 : Integer; Style : Word);
551:
552: Var
553:   L   : LineSettingsType;
554:
555: Begin
556:
557: {- Guardar el estilo de línea ---------------------------------------}
558:
559:   GetLineSettings(L);
560:   SetLineStyle(Style,L.Pattern,L.Thickness);
561:
562: {- Dibujar la línea -------------------------------------------------}
563:
564:   Line(X1,Y1,X2,Y2);
565:
566: {- Restaurar el estilo de línea -------------------------------------}
567:
568:   SetLineStyle(L.LineStyle,L.Pattern,L.Thickness)
569:
570: End;
571:
572: {+-------------------------------------------------------------------+}
573: {! Este procedimiento dibuja los ejes                               !}
574: {+-------------------------------------------------------------------+}
575: Procedure DrawAxis (Style : Word) ;
576:
577: Begin
578:
579: {- Dibujar los ejes -------------------------------------------------}
580:
581:   LineStyle (XP (XMin), YP (0.0),  XP (XMax), YP (0.0),  Style);
582:   LineStyle (XP (0.0),  YP (YMin), XP (0.0) , YP (YMax), Style)
583:
584: End;
585:
586:
587: {+-------------------------------------------------------------------+}
588: {! Este procedimiento cuadricula la pantalla de acuerdo a 'DeltaX' y 'DeltaY'!}
589: {+-------------------------------------------------------------------+}
590: Procedure ScaleScreen (DeltaX,DeltaY: ProgRealType; Style : Word) ;
591:
592: Var
593:   NextX,R    : ProgRealType;
594:   A1,A2,A3   : Integer;
595:
596: Begin
597:   NextX := DeltaX*Int(XMin/DeltaX);
598:   R     := YMax/30.0;
599:   A1    := YP ( R);
```

```
600:   A2    := YP (-R);
601:
602:   Repeat
603:      A3 := XP (NextX);
604:      LineStyle (A3,A1,A3,A2,Style);
605:      NextX := NextX + DeltaX
606:   Until NextX > Xmax;
607:
608:   NextX := DeltaY*Int(YMin/DeltaY);
609:   R     := XMax/30.0;
610:   A1    := YP ( R);
611:   A2    := XP (-R);
612:
613:   Repeat
614:      A3 := YP (NextX);
615:      LineStyle (A1,A3,A2,A3,Style);
616:      NextX := NextX + DeltaY
617:   Until NextX > YMax
618:
619: End;
620:
621: {+-------------------------------------------------------------------------+}
622: {! Función para seleccionar                                                !}
623: {+-------------------------------------------------------------------------+}
624: {$F+}
625: Function SelectInteger (Opcion : Integer) : Byte;
626: {$F-}
627:
628: Begin
629:    If Opcion <> 4 Then
630:       SelectInteger := ReadInteger
631:    Else
632:       SelectInteger := ReadReal
633: End;
634:
635: {+-------------------------------------------------------------------------+}
636: {! Procedimiento para leer k                                               !}
637: {+-------------------------------------------------------------------------+}
638: Procedure ReadK ;
639:
640: Const
641:   X      =          10;
642:   Y      =           5;
643:   Opcion : Integer = 1;
644:
645: Var
646:   Code   : Integer;
647:
648: Begin
649:
650:   Str(kMin      ,LGRStrings[1]);
651:   Str(KMax      ,LGRStrings[2]);
652:   Str(DK        ,LGRStrings[3]);
653:   Str(KCoeff:0:4,LGRStrings[4]);
654:
655:   ConstructMenu (' Valores de K ',
656:                  'K minima K maxima Incremento de K Coeficiente de K',
657:                  !S,18,7,X,Y,WTypeV,AtrFrame,AtrTitle,AtrBox,AtrNotSelected,
658:                  AtrDisabled,AtrFirst,AcceptWithCR,SelectInteger,
659:                  Opcion,
```

```
660:                    LGRStrings);
661:
662:   Val(LGRSTrings[1],XMin  ,Code);
663:   Val(LGRSTrings[2],XMax  ,Code);
664:   Val(LGRSTrings[3],DX    ,Code);
665:   Val(LGRSTrings[4],XCoeff,Code)
666:
667: End;
668:
669: {+-------------------------------------------------------------------------+}
670: {| Función para seleccionar                                                |}
671: {+-------------------------------------------------------------------------+}
672: {$F+}
673: Function SelectReadData (Opcion : Integer) : Byte;
674: {$F-}
675:
676: Begin
677:   SelectReadData := ReadReal
678: End;
679:
680: {+-------------------------------------------------------------------------+}
681: {| Leer datos de graficación                                               |}
682: {+-------------------------------------------------------------------------+}
683: Procedure ReadData;
684:
685: Const
686:   W      =          0;
687:   D      =          2;
688:   X      =         10;
689:   Y      =          5;
690:   Opcion : Integer = 1;
691:
692: Var
693:   Code   : Integer;
694:
695: Begin
696:
697:   Str(XMin  :W:D,LGRStrings[1]);
698:   Str(XMax  :W:D,LGRStrings[2]);
699:   Str(YMin  :W:D,LGRStrings[3]);
700:   Str(YMax  :W:D,LGRStrings[4]);
701:   Str(DeltaX:W:D,LGRStrings[5]);
702:   Str(DeltaY:W:D,LGRStrings[6]);
703:
704:   ConstructMenu ('Parámetros de graficación',
705:                  'R mínima R máxima I mínima I máxima Cota real Cota imaginaria',
706:                  IS,5,6,X,Y,WTypeV,AtrFrame,AtrTitle,AtrBox,AtrNotSelected,
707:                  AtrDisabled,AtrFirst,AcceptWithCR,SelectReadData,
708:                  Opcion,
709:                  LGRStrings);
710:
711:   Val(LGRSTrings[1],XMin  ,Code);
712:   Val(LGRSTrings[2],XMax  ,Code);
713:   Val(LGRSTrings[3],YMin  ,Code);
714:   Val(LGRSTrings[4],YMax  ,Code);
715:   Val(LGRSTrings[5],DeltaX,Code);
716:   Val(LGRSTrings[6],DeltaY,Code);
717:
718:   Writeln;
719:
```

```
720: {  If AskUser ('¿Acotar el eje imaginario automáticamente') Then
721:     Begin
722:     P     := Sqrt(2)*(GetMaxY+1)/(GetMaxY + 1 - BottomLine);
723:     YMin  := XMin/P;
724:     YMax  := XMax/P
725:     End
726:   Else
727:     Begin
728:     Str(YMin  :W:D,LGRStrings[1]);
729:     Str(YMax  :W:D,LGRStrings[2]);
730:
731:     ConstructMenu (' Parámetros de graficación ',
732:                    'I mínima I máxima',
733:                    IS,24,10,X,Y,WTypeV,AtrFrame,AtrTitle,AtrBox,AtrNotSelected,
734:                    AtrDisabled,AtrFirst,AcceptWithCR,SelectReadData,
735:                    Opcion,
736:                    LGRStrings);
737:
738:     Val(LGRStrings[1],YMin,Code);
739:     Val(LGRStrings[2],YMax,Code)
740:     End;
741:
742:   Writeln)
743:
744: End;
745:
746: {+------------------------------------------------------------------------+}
747: {! Función para construir la cadena de un polinomio                       !}
748: {+------------------------------------------------------------------------+}
749: Function ConstructString (P1 : ShortString; N : Byte) : MString;
750:
751: Var
752:   Result   : MString;
753:   X        : ShortString;
754:   I        : Byte;
755:
756: Begin
757:
758:   Result := '';
759:
760:   For I := N DownTo 0 Do
761:     Begin
762:     Str(I,X);
763:     Result := Result + P1 + X + ' '
764:     End;
765:
766:   ConstructString := Result
767:
768: End;
769:
770:
771: {+------------------------------------------------------------------------+}
772: {! Leer el polinomio                                                      !}
773: {+------------------------------------------------------------------------+}
774: Procedure ReadPoly (Opcion : Integer);
775:
776: Const
777:   X       =           2;
778:   Y       =           5;
779:   MOpcion : Integer = 1;
```

```
780:
781: Var
782:   T,
783:   M   : MString;
784:   P   : ^Coeff;
785:   I   : Byte;
786:   C   : Integer;
787:
788: Begin
789:
790:   If Opcion = 2 Then
791:      Begin
792:      M := ConstructString ('S^',N);
793:      P := @Poly.C;
794:      T := ' Polinomio P(S) '
795:      End
796:   Else
797:      Begin
798:      M := ConstructString ('KS^',N);
799:      P := @Poly.K;
800:      T := ' Polinomio KQ(S) '
801:      End;
802:
803:
804:   For I := N DownTo 0 Do
805:       Str(P^[I]:0:4,LGRStrings[I+1]);
806:
807:   ConstructMenu (T,M,IS,5,10,X,Y,WTypeV,AtrFrame,AtrTitle,AtrBox,AtrNotSelected,
808:                  AtrDisabled,AtrFirst,AcceptWithCR,SelectReadData,
809:                  MOpcion,
810:                  LGRStrings);
811:
812:   For I := N DownTo 0 Do
813:       Val(LGRStrings[I+1],P^[I],Code)
814:
815: End;
816:
817:
818: {+-------------------------------------------------------------------------+}
819: {| Procedimiento para etiquetar los ejes                                   |}
820: {+-------------------------------------------------------------------------+}
821: Procedure LabelAxis ;
822:
823: Var
824:   S   : String[20];
825:
826: Begin
827:
828:   Str(XMin:0:2,S);
829:   OutTextXY( XP (XMin), YP (-0.10), S);
830:
831:   Str(XMax:0:2,S);
832:   S := S + ' ';
833:   OutTextXY( XP (XMax) - TextWidth (S), YP (-0.10), S);
834:
835:
836:   Str(YMax:0:2,S);
837:   S := S + 'j';
838:   OutTextXY( XP (0.0), YP (YMax - 0.10), S);
839:
```

```
840:   Str(YMin:0:2,S);
841:   S := S + 'j';
842:   OutTextXY( XP (0.0), YP (YMin + 0.10) - TextHeight (S), S)
843:
844: End;
845:
846:
847: {+-------------------------------------------------------------------+}
848: {| Inicializar las gráficas                                          |}
849: {+-------------------------------------------------------------------+}
850: Procedure InitGraphics ;
851:
852: Var
853:   Gr      : Integer;
854:   M       : Word;
855:
856: Begin
857:
858: {- Detectar tarjeta gráfica ---------------------------------------------}
859:
860:   DetectGraph(GraphDriver,GraphMode);
861:   Case GraphDriver Of
862:      CGA         : GraphMode := CGAHi;
863:      MCGA        : GraphMode := MCGAHi;
864:      EGA         : GraphMode := EGAHi;
865:      EGA64       : GraphMode := EGA64Hi;
866:      EGAMono     : GraphMode := EGAMonoHi;
867:      IBM8514     : GraphMode := IBM8514Hi;
868:      HercMono    : GraphMode := HercMonoHi;
869:      ATT400      : GraphMode := ATT400Hi;
870:      VGA         : GraphMode := VGAHi;
871:      PC3270      : GraphMode := PC3270Hi
872:      Else
873:                    GraphDriver := Detect;
874:      End;
875:
876: {- Inicializar el sistema gráfico BGI -----------------------------------}
877:
878:   InitGraph(GraphDriver,GraphMode,'');
879:   Gr := GraphResult;
880:
881:   If Gr <> GrOk Then
882:      Begin
883:      Writeln('Error de inicialización : ',Gr);
884:      Halt($FF)
885:      End;
886:
887:
888: {- Buscar colores adecuados ---------------------------------------------}
889:
890:   PointColor  := GetMaxColor ;
891:   DirectVideo := False;
892:
893: {- Calcular el aspecto --------------------------------------------------}
894:
895:   GetAspectRatio(ARx,ARy);
896:   AR := ARx/ARy;
897:
898:   M := GetMaxY + 1;
899:   BottomLine := M Div 10;
```

```
900:   P          := 4*M/(3*(M - BottomLine));
901:
902: (- Restaurar el video ------------------------------------------------)
903:
904:   RestoreCRTMode;
905:
906: (- Inicializar las variables ---------------------------------------------)
907:
908:   kMax   :=   150;
909:   KMin   :=     0;
910:   DK     :=     1;
911:   KCoeff :=   1.0;
912:   XMin   :=  -1.0;
913:   Xmax   :=   1.0;
914:   DeltaX :=   0.5;
915:   DeltaY :=   0.5;
916:   Fin    := False;
917:   N      :=     1;
918:   FillChar(Poly,SizeOf(Poly),0);
919:
920: {$IfDef DEMO }
921: { Página del Dob }
922:
923:   N := 3;
924:   Poly.C[0] :=  1.0;
925:   Poly.C[1] := -1.11;
926:   Poly.C[2] := -0.195;
927:   Poly.C[3] :=  0.305;
928:   Poly.K[0] :=  0.0;
929:   Poly.K[1] :=  1.0;
930:   Poly.K[2] :=  0.45;
931:   Poly.K[3] := -0.34;
932:   XMin      := -5.0;
933:   XMax      :=  1.0;
934:   YMin      := -1.0;
935:   YMax      :=  1.0;
936:   DeltaX    :=  0.2;
937:   DeltaY    :=  0.2;
938:   DK        :=    1;
939:   KCoeff    := 0.0043
940:
941: {$EndIf }
942:
943: End;
944:
945: {+-------------------------------------------------------------------+}
946: {| Procedimiento para poner un "pixelote" en la pantalla             |}
947: {+-------------------------------------------------------------------+}
948: Procedure PutBigPixel (X,Y : Integer) ;
949:
950: Begin
951:   Rectangle(X-1,Y-1,X+1,Y+1);
952:   PutPixel(X,Y,1)
953: End;
954:
955: {+-------------------------------------------------------------------+}
956: {| Procedimiento para dibujar el LGR                                 |}
957: {+-------------------------------------------------------------------+}
958: Procedure ConstructLGR ;
959:
```

```
960: Var
961:   I         : Byte;
962:   K         : Integer;
963:   C         : Char;
964:   Diverge   : Boolean;
965:
966: Begin
967:
968: {- Preparar la pantalla ------------------------------------------------}
969:
970:   SetGraphMode(GraphMode);
971:   ViewPort (0,BottomLine,GetMaxX,GetMaxY);
972:   InitValues ;
973:   ScaleScreen (DeltaX,DeltaY,ScaleStyle);
974:   DrawAxis (AxisStyle);
975:   LabelAxis ;
976:
977: {- Evaluar los polos ------------------------------------------------}
978:
979:   A := Poly.C;
980:   PolyRoots (N,A,PolyReal,PolyImag,Epsilon,MaxN,Diverge);
981:
982:   If Not Diverge Then
983:      For I := 1 To N Do
984:         PlotXC (Xp (PolyReal[I]), Yp (PolyImag[I]),3,Polo);
985:
986: {- Evaluar los ceros ------------------------------------------------}
987:
988:   A := Poly.K;
989:   I :=      1;
990:   K :=      N;
991:
992:   While A[0] = 0.0 Do
993:      Begin
994:      For I := 0 To K-1 Do
995:         A[I] := A[I+1];
996:      Dec(K)
997:      End;
998:
999:   PolyRoots (K,A,PolyReal,PolyImag,Epsilon,MaxN,Diverge);
1000:
1001:   If Not Diverge Then
1002:      For I := 1 To K Do
1003:         PlotXC (Xp (PolyReal[I]), Yp(PolyImag[I]),5,Cero);
1004:
1005:   GotoXY(1,Bottom);
1006:   Write('Lugar geometrico para K = (',KMin,';',KMax,'). Cota del eje real = ',DeltaX:0:2,', Imaginario = ',DeltaY:0:2);
1007:
1008: {- Iterar para cada K -----------------------------------------------}
1009:
1010:   EraseTop ;
1011:   C := 'I';
1012:   K := KMin;
1013:
1014:   While (K <= KMax) And (C <> CHAR(ESC)) Do
1015:      Begin
1016:
1017: {- Construir el polinomio para esta K ------------------------------}
1018:
1019:      For I := 0 To N Do
```

```
1020:         A[I] := Poly.C[I] + KCoeff*K*Poly.K[I];
1021:
1022:      PolyRoots (N,A,PolyReal,PolyImag,Epsilon,MaxN,Diverge);
1023:
1024:      Write(' K = ',K:4,^M);
1025:
1026:      If Not Diverge Then
1027:         For I := 1 To N Do
1028:            PutBigPixel (Xp (PolyReal[I]), Yp (PolyImag[I]))
1029:      Else
1030:         Write('(K)',^M);
1031:
1032:      If KeyPressed Then
1033:         Begin
1034:         EraseTop ;
1035:         C := ReadKey;
1036:         Write('Pausa, K = ',K,'. Presiona cualquier tecla para continuar, ESC para terminar...',^M);
1037:         C := ReadKey;
1038:         EraseTop
1039:         End;
1040:
1041:      Inc(K,DK)
1042:      End;
1043:
1044:   Atiende ;
1045:   RestoreCRTMode
1046:
1047: End;
1048:
1049:
1050: {+-------------------------------------------------------------------+}
1051: {| P R O G R A M A   P R I N C I P A L                               |}
1052: {+-------------------------------------------------------------------+}
1053: Begin
1054:
1055: {- Inicializar el sistema gráfico ---------------------------------------}
1056:
1057:   InitGraphics ;
1058:   SysWindow    ;
1059:   ClrScr;
1060:
1061: {- Iterar en el menú ----------------------------------------------------}
1062:
1063:   Repeat
1064:
1065:      DoMenu ;
1066:
1067:      Case OpcionH Of
1068:         1   : Case OpcionV Of
1069:                  1 : Begin
1070:                      Str(N,Cadena);
1071:                      Writeln;
1072:                      ReadNum (Cadena,Cadena,'Grado del polinomio-> ',2,ReadInteger,UA);
1073:                      Writeln;
1074:
1075:                      If UA = UserAccepted Then
1076:                         Val(Cadena,N,Code)
1077:
1078:                      End;
1079:
```

```
1080:                   2,
1081:                   3 : ReadPoly (OpcionV)
1082:
1083:                   End;
1084:
1085:           2   : Case OpcionV Of
1086:                   1   : ReadData ;
1087:                   2   : ReadK    ;
1088:                   3   : Begin
1089:                           YMin := XMin/P;
1090:                           YMax := XMax/P
1091:                           End
1092:                   Else
1093:                           Begin
1094:                           XMin := YMin*P;
1095:                           XMax := YMax*P
1096:                           End
1097:                   End;
1098:
1099:           3   : Begin
1100:                 Window(1,1,80,25);
1101:                 ConstructLGR ;
1102:                 SysWindow
1103:                 End;
1104:
1105:           4   : Begin
1106:                 Writeln;
1107:                 Fin := True; {AskUser ('¿Terminar ejeucidn');}
1108:                 Writeln  .
1109:                 End
1110:
1111:           End
1112:
1113:   Until Fin;
1114:
1115:   Window(1,1,80,25);
1116:   TextAttr :=  15;
1117:   ClrScr;
1118:   CloseGraph ;
1119:   Writeln('*** Fin de ejecucidn LGR 1.0')
1120:
1121: End.
```

```
 1: {                                                                    }
 2: {                          LGRInt.Pas                        V1.0    }
 3: {                                                                    }
 4: {     Unidad para hacer la interfaz del programa LGR.                }
 5: {                                                                    }
 6: {                                                                    }
 7: {                                                                    }
 8: {                          Mxico. Marzo. 1991.                     }
 9: {                                                                    }
10:
11: (+------------------------------------------------------------------+)
12: (! Para probar la unidad, definir PRUEBA                           !)
13: (+------------------------------------------------------------------+)
14:
15: ($Define  PRUEBAx )
16:
17: ($IfDef  PSUEBA )
18: ($Define UINIT  )
19: ($EndIf )
20:
21: (+------------------------------------------------------------------+)
22: (! Si la unidad tendrá inicialización, definir UINIT. Si de define PRUEBA,  !)
23: (! automáticamente se define UINIT                                  !)
24: (+------------------------------------------------------------------+)
25:
26: ($Define UINIT )
27:
28: ($IfNDef PRUEBA )
29: Unit
30:   LGRInt;
31:
32: Interface
33:
34: ($EndIf )
35:
36: Uses
37:   Crt,
38:   FlDef,
39:   Menus;
40:
41: (+------------------------------------------------------------------+)
42: (! Variables, tipos y constantes públicas                           !)
43: (+------------------------------------------------------------------+)
44:
45: ($I KeyCodes )
46:   ShortStringSize =   5;
47:   TStringSize     =  80;
48:   UserPressedESC  =   1;
49:   UserWantsHelp   =   2;
50:   UserAccepted    =   3;
51:   ReadInteger     =   1;
52:   ReadReal        =   2;
53:
54: (- Para el menú ---------------------------------------------------)
55:
56:   IS              =     '|_';
57:   WTypeH          = Invisible;
58:   WTypeV          =    Simple;
59:   AtrFrame        =        15;
```

```
  AtrTitle        =      112;
  AtrBox          =       94;
  AtrNotSelected  =       30;
  AtrDisabled     = BYTE('x');
  AtrFirst        =       27;

Type
  ShortString     = String[ShortStringSize];
  TString         = String[TStringSize];
  SelectF         = Function (O : Integer) : Byte;

Var
  OpcionH,OpcionV   : Integer;
  LGRStrings        : Strings;

{+-------------------------------------------------------------------------+}
{! Procedimientos INLINE                                                   !}
{+-------------------------------------------------------------------------+}

{+-------------------------------------------------------------------------+}
{! Encabezados de funciones y procedimientos públicos, debajo del IfDef     !}
{+-------------------------------------------------------------------------+}

{$IfNDef PRUEBA }

Procedure DoMenu    ;
Procedure SysWindow ;
Procedure ReadNum       (Var Cadena : TString ; Inicial,Mensaje : TString;
                         Max,NumType : Byte; Var UserAction : Byte) ;
Procedure ConstructMenu (Title : TString; Options : MString;
                         InsertString : ShortString;
                         TextWidth,DataWidth,
                         X,Y,WType,
                         AtrFrame,
                         AtrTitle,
                         AtrBox,
                         AtrNotSelected,
                         AtrDisabled,
                         AtrFirst      : Byte;
                         AcceptCROnly  : Boolean;
                         Select        : SelectF;
                         Var Opcion    : Integer;
                         Var Nums      : Strings);

Implementation

{$EndIf}

{+-------------------------------------------------------------------------+}
{! Implementación de procedimientos y funciones, tanto públicas como privadas!}
{! Constantes, tipos y variables privadas                                  !}
{+-------------------------------------------------------------------------+}

Const
  Title         = ' Trazado de lugares geometricos ';
  Opciones      = '  Polinomios       Parámetros de graficación       Graficar        Salida ';
  X             =       1;
  Y             =       1;

  Exp           : Strings = ('Especificar los polinomios P(S) y KQ(S) del sistema',
```

```
120:                            'Ingresar y/o modificar los parámetros y el intervalo de graficación',
121:                            'Graficar el lugar geometrico con los parámetros actuales',
122:                            'Salir a MS-DOS',
123:                            '','','','','','');
124:
125:   OtrasOpciones  : Strings = ('Grado del polinomio Ingresar_P(S) Ingresar_KQ(S)',
126:                            'Cotas de la gráfica Valores de K '+
127:                            'Acotar el eje imaginario Acotar el eje real',
128:                            '','','','','','','','');
129:
130:
131: {+-------------------------------------------------------------------------+}
132: {| Procedimiento para hacer el menú                                        |}
133: {+-------------------------------------------------------------------------+}
134: Procedure DoMenu ;
135:
136: Begin
137:   BothMenus  (Title,Opciones,OtrasOpciones,X,Y,WTypeH,WTypeV,
138:               AtrFrame,AtrFrame,AtrTitle,AtrBox,AtrNotSelected,
139:               AtrDisabled,AtrFirst,MenuNotInWindow,
140:               AcceptWithCR,AcceptWithCR,OpcionH,OpcionV)
141: End;
142:
143: {+-------------------------------------------------------------------------+}
144: {| Procedimiento para la ventana del sistema                               |}
145: {+-------------------------------------------------------------------------+}
146: Procedure SysWindow ;
147:
148: Begin
149:   DrawWindowWithTitle (' Diálogo ','',2,3,79,23,Simple,15,48,Activa)
150: End;
151:
152: {+-------------------------------------------------------------------------+}
153: {| Procedimiento para leer un número como cadena                           |}
154: {+-------------------------------------------------------------------------+}
155: Procedure ReadNum (Var Cadena : TString ; Inicial,Mensaje : TString;
156:                    Max,NumType : Byte; Var UserAction : Byte) ;
157:
158: Label
159:   break,
160:   continue;        { Para simular las instrucciones 'break' y 'continue' de C }
161:
162: Const
163:   Inserting : Boolean = True;
164:
165: Var
166:   N,I,X,Y                : Byte;
167:   Punto,Expn,Terminado   : Boolean;
168:   Blank                  : String[30];
169:   ValidChars             : Set Of Char;
170:   C                      : Tecla;
171:
172:   {+-------------------------------------------------------------------------+}
173:   {|                                                                         |}
174:   {+-------------------------------------------------------------------------+}
175:   Procedure InitValues ;
176:
177:   Begin
178:     I      :=                1;
179:     N      :=      Length(Cadena);
```

```
180:      Punto  := Pos('.',Cadena) > 0;
181:      Expn   := Pos('e',Cadena) > 0;
182:      PokeMsgInWindow (Blank ,X,Y,TextAttr);
183:      PokeMsgInWindow (Cadena,X,Y,TextAttr);
184:      GotoXY(X,Y)
185:   End;
186:
187:
188: Begin
189:
190: {- Inicializar variables ------------------------------------------------}
191:
192:   ValidChars := ['0'..'9','-','.'];
193:
194:   If NumType = ReadInteger Then
195:      ValidChars := ValidChars - ['.'];
196:
197:   Terminado  := False;
198:   I          :=    1;
199:   Write(Mensaje);
200:   X          := WhereX;
201:   Y          := WhereY;
202:   Blank      := StrOf (' ',Max);
203:
204: { -- Poner el mensaje y almacenar la posición del cursor. Inicializar ----- }
205:
206:   Cadena := Inicial;
207:   InitValues ;
208:
209: { -- Iniciar la lectura de la Cadena ------------------------------------- }
210:
211:   While Not Terminado Do
212:
213:      Begin
214:      Cadena[0] := Char(N);
215:      If Inserting Then
216:         ShowCursor (7,7)
217:      Else
218:         ShowCursor (0,7);
219:
220:      LeeT (C);
221:
222: {- Se ha recibido un caracter ASCII? ------------------------------------}
223:
224:      If C.Ascii <> 0 Then
225:
226:         Case C.Ascii Of
227:
228: {- Ctrl-Y ----------------------------------------------------------------}
229:
230:            CTRLY : Begin
231:                    Cadena := '';
232:                    InitValues
233:                    End;
234:
235: {- Ctrl-R ----------------------------------------------------------------}
236:
237:            CTRLR : Begin
238:                    Cadena := Inicial;
239:                    InitValues
```

```
240:                    End;
241:
242: {- BackSpace ------------------------------------------------------------}
243:
244:            BS    : If (WhereX <> X) And (N <> 0) Then
245:                      Begin
246:
247:             .        Write(^H);
248:                      Dec(N);
249:                      ScrollLine (Y,WhereX,X + N,Left,' ');
250:
251:                      If Cadena[I - 1] = '.' Then   { Si había punto    }
252:                        Punto := False;
253:
254:                      If Cadena[I - 1] = 'e' Then   { Si era exponente  }
255:                        Begin
256:                        Expn := False;
257:
258:                        If (Pos ('-',Cadena) > 0 ) And (Cadena[I] = '-') Then { Si era exponente negativo }
259:                          Begin
260:                          Delete(Cadena,I,1);
261:                          Dec(N);
262:                          ScrollLine (Y,WhereX,X + N,Left,' ')
263:                          End
264:
265:                        End;
266:
267:                      Dec(I);
268:                      Delete(Cadena,I,1)
269:
270:                      End;
271:
272: {- Se termina con CR o ESC -------------------------------------------------}
273:
274:            ESC,
275:            CR    : Begin
276:                    Cadena[0] := Char(N);
277:                    Terminado := True;
278:
279:                    If C.Ascii = ESC Then
280:                      UserAction := UserPressedESC
281:                    Else
282:                      UserAction := UserAccepted
283:
284:                    End;
285:
286: {- Signo - en el teclado numerico -------------------------------------------}
287:
288:       ( - )  45    : If ((I <> 1) And (Cadena[I-1] <> 'e')) Or (Cadena[I] = '-') Then
289:                        Goto break
290:                    Else
291:                        Goto continue;
292:
293: {- Cualquier otra cosa -------------------------------------------------------}
294:
295:            Else    Begin
296:
297: continue :         If Not (C.Chr In ValidChars) Then
298:                      Goto break;
299:
```

```
300:                  If (Cadena[I] = '-') And (I = 1) And Inserting And (Length(Cadena) > 0) Then
301:                     Goto break;
302:
303:                  If (C.Chr = '.') And Punto Then
304:                     Goto break;
305:
306:                  If ((C.Chr = ',') And (I > Pos('e',Cadena)) And (Pos('e',Cadena) <> 0)) And
307:                     (NumType = ReadInteger)                                              Then
308:                     Goto break;
309:
310:                  If (N >= Max) And (Inserting Or (I > N)) Then
311:                     Goto break;
312:
313:                  If C.Chr = '.' Then
314:                     Punto := True;
315:
316:                  If Inserting Then
317:                     Begin              { Si hay estado de inserción       }
319:                     Insert(' ',Cadena,I);
319:                     Inc(N);
320:                     ScrollLine (Y,WhereX,X + N,Right,' ')
321:                     End;
322:
323:                  If Cadena[I] = '.' Then
324:                     Punto := False;
325:
326:                  If Cadena[I] = 'e' Then
327:                     Begin
328:                     Expn := False;
329:
330:                     If Cadena[I+1] = '-' Then
331:                        Begin
332:                        Delete(Cadena,I+1,1);
333:                        Dec(N);
334:                        ScrollLine (Y,X + I,X + N,Left,' ')
335:                        End
336:
337:                     End;
338:
339:                  Write(CHAR(C.Ascii));
340:                  Cadena[I] := CHAR(C.Ascii);
341:                  Inc(I);
342:
343:                  If I > N Then
344:                     N := I-1
345:                  End
346:
347:            End
348:      Else
349:
350:         Case C.Codigo Of
351:            F1   : Begin
352:                   Cadena[0] := Char(N);
353:                   Terminado := True;
354:                   UserAction := UserWantsHelp
355:                   End;
356:
357:            AltE : Begin
358:
359:                   If NumType = ReadInteger Then
```

```
360:                    Goto break;
361:
362:                 If Expn Or (Pos ('.',Cadena) >= I) Then
363:                    Goto break;
364:
365:                 If (N >= Max) And (Inserting Or (I >= N)) Then
366:                    Goto break;
367:
368:                 Inc(N);
369:                 ScrollLine (Y,WhereX,X + N,Right,' ');
370:                 Write('e');
371:                 Insert('e',Cadena,I);
372:                 Inc(I);
373:                 Expn := True
374:                 End;
375:
376:           HOME : Begin
377:                 GotoXY (X,Y);
378:                 I := 1
379:                 End;
380:
381:           _END : Begin
382:                 GotoXY (X + N,y);
383:                 I := N+1
384:                 End;
385:
386:           LARR : If I <> 1 Then
387:                    Begin
388:                    GotoXY (WhereX - 1,Y);
389:                    Dec(I)
390:                    End;
391:
392:           RARR : If I < N Then
393:                    Begin
394:                    GotoXY (WhereX + 1,Y);
395:                    Inc(I)
396:                    End;
397:
398:           INS  : Inserting := Not Inserting;
399:
400:           DEL  : If (N <> 0) And (I <= N) Then
401:                    Begin
402:                    Dec(N);
403:                    ScrollLine (Y,WhereX,X + N,Left,' ');
404:
405:                    If Cadena[I] = '.' Then
406:                       Punto := False;
407:
408:                    If Cadena[I] = 'e' Then
409:                       Begin
410:                       Expn := False;
411:
412:                       If Cadena[I+1] = '-' Then
413:                          Begin
414:                          Dec(N);
415:                          ScrollLine (Y,WhereX,X + N,Left,' ');
416:                          Delete(Cadena,I+1,1);
417:                          End
418:
419:                       End;
```

```
420:
421:                        Delete(Cadena,I,1)
422:                        End
423:
424:             End;
425:
426: break :  { Simulación de la instrucción 'break' de C                        }
427:
428:     End
429:
430: End;
431:
432: {+-------------------------------------------------------------------------+}
433: {! Procedimiento para construir el menú                                    !}
434: {+-------------------------------------------------------------------------+}
435: Procedure ConstructMenu (Title : TString; Options : MString;
436:                          InsertString : ShortString;
437:                          TextWidth,DataWidth,
438:                          X,Y,WType,
439:                          AtrFrame,
440:                          AtrTitle,
441:                          AtrBox,
442:                          AtrNotSelected,
443:                          AtrDisabled,
444:                          AtrFirst       : Byte;
445:                          AcceptCROnly   : Boolean;
446:                          Select         : SelectF;
447:                          Var Opcion     : Integer;
448:                          Var Nums       : StringS);
449:
450: Var
451:   Xw,Yw,
452:   Sx,Sy,Ta,
453:   I,UA,
454:   X2,Y2    : Byte;
455:   P        : Pointer;
456:   Size     : Word;
457:   M        : MString;
458:   T        : TString;
459:   SaveExp  : Boolean;
460:   Op2      : Integer;
461:
462:   {+-----------------------------------------------------------------------+}
463:   {! Función para completar una cadena                                     !}
464:   {+-----------------------------------------------------------------------+}
465:   Function Complete (S : String; C : Char; L : Byte) : String;
466:
467:   Begin
468:      If Length(S) < L Then
469:         S := S + StrOf (C,L - Length(S));
470:
471:      Complete := S
472:
473:   End;
474:
475: Begin
476:
477:   SaveExp := Explain;
478:   Explain := False;
479:   Sx      := WhereX;
```

```
480:   Sy      := WhereY;
481:   Xw      := XWWin - 1;
482:   Yw      := YWWin - 1;
483:   Ta      := TextAttr;
484:
485:   WindowToScreen (X,Y,X,Y);
486:
487: {- Construir la cadena del menú ------------------------------------------}
488:
489:   I       := 0;
490:   M       := '';
491:   T       := StrTOK (Options,[' ']);
492:
493:   While T <> '' Do
494:     Begin
495:
496:     Inc(I);
497:
498: {- Truncar en caso necesario ---------------------------------------------}
499:
500:     If Length(T) > TextWidth Then
501:        T[0] := CHAR(TextWidth)
502:     Else
503:        T    := T + StrOf ('_',TextWidth - Length(T));
504:
505:     M := M + T + InsertString + Complete(Nums[I],'_',DataWidth) + ' ';
506:
507:     T := StrTOK ('',[' '])
508:
509:     End;
510:
511: {- Analizar el menú ------------------------------------------------------}
512:
513:   MenuSize  (M,X,Y,Shadow,Vertical,X2,Y2,Size);
514:   GetMem(P,Size);
515:   GetScreen (X,Y,X2,Y2,P^);
516:
517: {  Opcion := 0;}
518:
519: {- Hacer el menú ---------------------------------------------------------}
520:
521:   Repeat
522:     VPullDownMenu (Title,M,X,Y,WType,AtrFrame,AtrTitle,AtrBox,AtrNotSelected,
523:                    AtrDisabled,AtrFirst,MenuNotInWindow,DontStoreScreen,
524:                    AcceptCRonly,Opcion);
525:
526:     If Opcion > 0 Then
527:        Begin
528:
529:        Op2 := Opcion;
530:        GotoXY(X + TextWidth + Length(InsertString) + 2 - Xw,Y + Opcion - Yw);
531:        TextAttr := AtrBox;
532:        I        := Pos('_',Nums[Opcion]);
533:
534:        If I > 0 Then
535:           Delete(Nums[Opcion],I,255);
536:
537:        M := Nums[Opcion];
538:        ReadNum (Nums[Opcion],M,'',DataWidth,Select (Opcion),UA);
539:
```

```
540:        If UA <> UserAccepted Then
541:           Nums[Opcion] := M;
542:
543: (- Construir la nueva cadena ------------------------------------------)
544:
545:        M     := '';
546:        I     := 1;
547:        T     := StrTOK (Options,[' ']);
548:
549:        While T <> '' Do
550:           Begin
551:
552: (- Truncar en caso necesario --------------------------------------------)
553:
554:           If Length(T) > TextWidth Then
555:              T[0] := CHAR(TextWidth)
556:           Else
557:              T    := T + StrOf ('_',TextWidth - Length(T));
558:
559:           M := M + T + InsertString + Complete(Nums[I],'_',DataWidth) + ' ';
560:
561:           T := StrTOK ('',[' ']);
562:           Inc(I)
563:
564:           End
565:
566:        End
567:
568:  Until Opcion = 0;
569:
570:  Opcion := Op2;
571:  GotoXY(Sx,Sy);
572:  Explain  := SaveExp;
573:  TextAttr :=     Ta;
574:
575:  PutScreen (X,Y,X2,Y2,P^);
576:  FreeMem(P,Size)
577:
578: End;
579:
590:
581: {+-------------------------------------------------------------------+}
582: {! Variables y tipos que sólo se usan para probar                    !}
583: {+-------------------------------------------------------------------+}
584:
585: {$IfDef PRUEBA }
586:
587: Var
588:   CualquierVariable   : Real;
589:
590: {$EndIf}
591:
592:
593: {+-------------------------------------------------------------------+}
594: {! I N I C I A L I Z A C I O N   Y / O   P R U E B A                 !}
595: {+-------------------------------------------------------------------+}
596: {$IfDef UINIT }
597:
598: Begin
599:
```

```
600: {$EndIf }
601:
602: {------------------ Código para inicializar la unidad ----------------------}
603:
604: (- Inicializar las variables de los menús ------------------------------------)
605:
606:   Explain    :=  True;
607:   ExpPtr     :=  @Exp;
608:   AtrExplain :=  113;
609:   XExp       :=    1;
610:   YExp       :=   25;
611:   AtrShadow  :=  112;
612:   Shadow     :=  True;
613:
614: {------------------ Código para probar la unidad ----------------------------}
615:
616: {$IfDef PRUEBA }
617:   ;
618:
619: {$EndIf}
620:
621: End.
```

```
{                                                                          }
{                                  Math.Pas                          V1.0  }
{                                                                          }
{  Unidad miscelánea matemática. Resuelve y evalua polinomios. Cal-        }
{ cula transformadas de Fourier. Realiza operaciones complejas.            }
{                                                                          }
{                                                                          }
{                                 Mxico. Febrero. 1991.                    }
{                                                                          }

{+-------------------------------------------------------------------------+}
{! Para probar la unidad, definir PRUEBA                                   !}
{+-------------------------------------------------------------------------+}

{$Define  PRUEBA_ }

{$IfDef  PRUEBA }
{$Define UINIT  }
{$EndIf }

{+-------------------------------------------------------------------------+}
{! Si la unidad tendrá inicialización, definir UINIT. Si de define PRUEBA,  !}
{! automáticamente se define UINIT                                          !}
{+-------------------------------------------------------------------------+}

{$Define UINIT_ }

{$IfNDef PRUEBA }
Unit
  Math;

Interface

{$EndIf }

Uses
  Dos;

{+-------------------------------------------------------------------------+}
{! Variables, tipos y constantes públicas                                  !}
{+-------------------------------------------------------------------------+}
Const
  MaxPolyDegree   = 30;

Type

{$IfOpt F-}
  MathRealType    = Real;
{$Else }
  MathRealType    = Extended;
{$EndIf}
  Coeff           = Array [0..MaxPolyDegree] Of MathRealType;
  Complex         = Record
                      Case X : Byte Of
                        0   : (Pr,Pim   : MathRealType);
                        1   : (R,Ang    : MathRealType)
                    End;
```

```
60: {+--------------------------------------------------------------------+}
61: {! Procedimientos INLINE                                              !}
62: {+--------------------------------------------------------------------+}
63:
64: {+--------------------------------------------------------------------+}
65: {! Encabezados de funciones y procedimientos públicos                  !}
66: {+--------------------------------------------------------------------+}
67:
68: {$IfNDef PRUEBA }
69:
70: Procedure PolyRoots  (Degree  : Byte; Var Poly; Var PolyReal,PolyImag : Coeff;
71:                       Epsilon : Real; MIter : Word; Var Diverge : Boolean);
72: Procedure PolyEval   (Degree  : Byte; Var Poly; X : Complex; Var Result : Complex);
73: Procedure CAdd       (C1,C2 : Complex; Var Sum  : Complex) ;
74: Procedure CSub       (C1,C2 : Complex; Var Sub  : Complex) ;
75: Procedure CMul       (C1,C2 : Complex; Var Prod : Complex) ;
76: Procedure Conjugado  (C     : Complex; Var CC   : Complex) ;
77: Procedure Polar      (Var C : Complex)                     ;
78: Procedure Cartesiano (Var C : Complex)                     ;
79: Function  Mag        (C     : Complex)         : MathRealType ;
80: Function  Sign       (X     : MathRealType)    : MathRealType ;
81: Procedure RMul       (Var C : Complex;       R : MathRealType);
82: Procedure CPow       (C     : Complex;       N : Integer; Var Pow : Complex) ;
83:
84: Implementation
85:
86: {$EndIf}
87:
88: {+--------------------------------------------------------------------+}
89: {! Implementación de procedimientos y funciones, tanto públicas como privadas!}
90: {! Constantes, tipos y variables privadas                             !}
91: {+--------------------------------------------------------------------+}
92:
93: {+--------------------------------------------------------------------+}
94: {! Procedimiento para resolver polinomios por el método de Bairnstow   !}
95: {+--------------------------------------------------------------------+}
96: Procedure PolyRoots (Degree  : Byte; Var Poly; Var PolyReal,PolyImag : Coeff;
97:                      Epsilon : Real; MIter : Word; Var Diverge : Boolean);
98:
99: Var
100:   I             : Byte;
101:   A,B,C         : Coeff;
102:   Poly2         : Coeff Absolute Poly;
103:   D,P,Q,Dp,Dq,T  : MathRealType;
104:
105:
106:   {+--------------------------------------------------------------------+}
107:   {! Procedimiento para resolver polinomios de segundo grado            !}
108:   {+--------------------------------------------------------------------+}
109:   Procedure Pol2 (Var P,Q,Pr1,Pi1,Pr2,Pi2 : MathRealType);
110:
111:   Var
112:      Delta   : MathRealType;
113:
114:   Begin
115:
116:      Delta:= Sqr(P)-4.0*Q;
117:
118:      If Delta < 0 Then
119:         Begin
```

```
120:        Pr1 := -P/2.0;
121:        Pr2 :=  Pr1;
122:        Pi1 := (Sqrt(-Delta))/2.0;
123:        Pi2 := -Pi1
124:        End
125:     Else
126:        Begin
127:        Delta := Sqrt(Delta);
128:        Pr1   := (-P - Delta)/2.0;
129:        Pr2   := (-P + Delta)/2.0;
130:        Pi1   := 0.0;
131:        Pi2   := 0.0
132:        End
133:  End;
134:
135:
136: Begin
137:
138: (- Almacenar poly en a ------------------------------------------------------)
139:
140:   For I := 0 To Degree Do
141:      A[I] := Poly2[I];
142:
143: (- Quitar los coeficientes en cero ----------------------------------------)
144:
145:   While A[0] = 0.0 Do
146:     Begin
147:     For I := 1 To Degree do
148:        A[I-1] := A[I];
149:     PolyReal[Degree]:=0.0;
150:     PolyImag[Degree]:=0.0;
151:     Dec(Degree)
152:     End;
153:
154:  Diverge := False;
155:
156: (- Iterar mientras se tenga grado superior al segundo ----------------------)
157:
158:  While (Degree > 2) And (Not Diverge) Do
159:     Begin
160:
161:     P := 1.0;
162:     Q := 1.0;
163:
164: (- Construir un polinomio de segundo grado ---------------------------------)
165:
166:     Repeat
167:        B[I] := 0.0;
168:        C[1] := 0.0;
169:        B[0] := 0.0;
170:        C[0] := 0.0;
171:
172:        For I := 2 To Degree+2 Do
173:           Begin
174:           B[I] :=  A[I-2] - P*B[I-1] - Q*B[I-2];
175:           C[I] := -B[I-1] - P*C[I-1] - Q*C[I-2]
176:           End;
177:
178:        D  :=   Sqr(C[Degree+1])        + C[Degree]*(P*C[Degree+1] + Q*C[Degree]);
179:        Dp :=   C[Degree]*B[Degree+2)   - C[Degree+1]*B[Degree+1];
```

```
180:         Dq := -(CC[Degree+1]+B[Degree+2] + B[Degree+1]*(Q*CC[Degree] + P*CC[Degree+1]));
181:
182:         If D = 0.0 Then
183:            Begin
184:            Dp := Cos(P);
185:            Dq := Cos(Dq)
186:            End
187:         Else
188:            Begin
189:            Dp := Dp/D;
190:            Dq := Dq/D
191:            End;
192:
193:         Dec(NIter);
194:         P := P + 0.99*Dp;
195:         Q := Q + 0.98*Dq;
196:         T := Abs(P) + Abs(Q);
197:
198:         If (T <> 0.0) Then
199:            T := (Abs(Dp) + Abs(Dq))/T
200:         Else
201:            T := 1.0;
202:
203:      Until (T <= Epsilon) Or (NIter = 0);
204:
205:      Diverge := T > Epsilon;
206:
207: {- Resolver el polinomio de segundo grado formado... ------------------------}
208:
209:      If Not Diverge Then
210:         Begin
211:         Pol2 (P,Q,PolyReal[Degree-1],PolyImag[Degree-1],PolyReal[Degree],PolyImag[Degree]);
212:
213:         Dec(Degree,2);
214:
215: {- Formar un nuevo polinomio de grado n-2 -------------------------------------}
216:
217:         For I := 0 To Degree Do
218:            A[I] := B[I+2]
219:
220:         End
221:
222:   End;
223:
224:   P := A[1]/A[0];
225:   Q := A[2]/A[0];
226:
227: {- Si quedó un polinomio de segundo grado, resolverlo ------------------------}
228:
229:   If Not Diverge Then
230:      If Degree = 2 Then
231:         Pol2 (P,Q,PolyReal[1],PolyImag[1],PolyReal[2],PolyImag[2])
232:      Else
233:         Begin
234:         PolyReal[1] := -P;
235:         PolyImag[1] :=  0.0
236:         End
237:
238: End;
239:
```

```
240:
241: {+------------------------------------------------------------------+}
242: {! Suma compleja                                                   !}
243: {+------------------------------------------------------------------+}
244: Procedure CAdd (C1,C2 : Complex; Var Sum : Complex) ;
245:
246: Begin
247:   Sum.Pr  := C1.Pr  + C2.Pr;
248:   Sum.Pim := C1.Pim + C2.Pim
249: End;
250:
251: {+------------------------------------------------------------------+}
252: {! Resta compleja                                                  !}
253: {+------------------------------------------------------------------+}
254: Procedure CSub (C1,C2 : Complex; Var Sub : Complex) ;
255:
256: Begin
257:   Sub.Pr  := C1.Pr  - C2.Pr;
258:   Sub.Pim := C1.Pim - C2.Pim
259: End;
260:
261: {+------------------------------------------------------------------+}
262: {! Producto complejo                                               !}
263: {+------------------------------------------------------------------+}
264: Procedure CMul (C1,C2 : Complex; Var Prod : Complex) ;
265:
266: Begin
267:   Prod.Pr  := C1.Pr*C2.Pr  - C1.Pim*C2.Pim;
268:   Prod.Pim := C1.Pr*C2.Pim + C1.Pim*C2.Pr
269: End;
270:
271: {+------------------------------------------------------------------+}
272: {! Conjugado de un complejo                                        !}
273: {+------------------------------------------------------------------+}
274: Procedure Conjugado (C : Complex; Var CC : Complex) ;
275:
276: Begin
277:   CC.Pr  :=  C.Pr;
278:   CC.Pim := -C.Pim
279: End;
280:
281: {+------------------------------------------------------------------+}
282: {! Magnitud de un complejo                                         !}
283: {+------------------------------------------------------------------+}
284: Function Mag (C : Complex) : MathRealType;
285:
286: Begin
287:   Mag := Sqrt(Sqr(C.Pr) + Sqr(C.Pim))
288: End;
289:
290: {+------------------------------------------------------------------+}
291: {! Función signo                                                   !}
292: {+------------------------------------------------------------------+}
293: Function Sign (X : MathRealType) : MathRealType ;
294:
295: Begin
296:   If X > 0 Then
297:     Sign := 1.0
298:   Else
299:     If X < 0 Then
```

```
300:        Sign := -1.0
301:     Else
302:        Sign :=  0.0
303: End;
304:
305: {+-------------------------------------------------------------------+}
306: {| Procedimiento para cambiar de forma cartesiana a polar            |}
307: {+-------------------------------------------------------------------+}
308: Procedure Polar (Var C : Complex) ;
309:
310: Var
311:   X   : MathRealType;
312:
313: Begin
314:
315:   If C.Pr <> 0 Then
316:      Begin
317:      X      := ArcTan(C.Pim/C.Pr);
319:      If C.Pr < 0 Then
319:         X :=  X + Pi
320:      End
321:   Else
322:      X      := Sign(C.Pim)*Pi/2;
323:
324:   C.R   := Mag(C);
325:   C.Ang := X
326:
327: End;
328:
329: {+-------------------------------------------------------------------+}
330: {| Procedimiento para forma cartesiana                               |}
331: {+-------------------------------------------------------------------+}
332: Procedure Cartesiano (Var C : Complex) ;
333:
334: Var
335:   X,Y   : MathRealType;
336:
337: Begin
338:   X     := C.R*Cos(C.Ang);
339:   Y     := C.R*Sin(C.Ang);
340:   C.Pr  := X;
341:   C.Pim := Y
342: End;
343:
344:
345: {+-------------------------------------------------------------------+}
346: {| Potencia real                                                     |}
347: {+-------------------------------------------------------------------+}
348: Function RPow (X : MathrealType; N : Integer) : MathRealType ;
349:
350: Var
351:   P   : MathRealType;
352:   F   : Boolean;
353:
354: Begin
355:
356:   P := 1.0;
357:   F := N < 0;
358:
359:   If F Then
```

```
360:     N := - N;
361:
362:   While (N > 0) Do
363:     Begin
364:     P := P*X;
365:     Dec(N)
366:     End;
367:
368:   If F Then
369:     P := 1.0/P;
370:
371:   RPow := P
372:
373: End;
374:
375: {+---------------------------------------------------------------------+}
376: {| Potencia compleja                                                   |}
377: {+---------------------------------------------------------------------+}
378: Procedure CPow (C : Complex; N : Integer; Var Pow : Complex) ;
379:
380: Begin
381:   Polar (C);
382:   Pow.R   := RPow(C.R,N);
383:   Pow.Ang := C.Ang*N;
384:   Cartesiano (Pow)
385: End;
386:
387: {+---------------------------------------------------------------------+}
388: {| Multiplicación por un real                                          |}
389: {+---------------------------------------------------------------------+}
390: Procedure RMul (Var C : Complex; R : MathRealType) ;
391:
392: Begin
393:   C.Fr  := R*C.Fr;
394:   C.Fim := R*C.Fim
395: End;
396:
397: {+---------------------------------------------------------------------+}
398: {| División compleja                                                   |}
399: {+---------------------------------------------------------------------+}
400: Procedure CDiv (C1,C2 : Complex; Var CCDiv : Complex) ;
401:
402: Begin
403:   Conjugado (C2,C2);
404:   CMul (C1,C2,CCDiv);
405:   RMul (CCDiv,1/Sqr(Mag(C2)))
406: End;
407:
408:
409: {+---------------------------------------------------------------------+}
410: {| Procedimiento para evaluar un polinomio                             |}
411: {+---------------------------------------------------------------------+}
412: Procedure PolyEval (Degree : Byte; Var Poly; X : Complex; Var Result : Complex);
413:
414: Var
415:   Poly2   : Coeff Absolute Poly;
416:   C1      : Complex;
417:   I       : Byte;
418:
419: Begin
```

```
420:
421:   CPow (X,Degree,Result);
422:   RMul (Result,Poly2C0]);
423:
424:   For I := 1 To Degree-2 Do
425:      Begin
426:      CPow (X,Degree-I,C1);
427:      RMul (C1,Poly2CI]);
428:      CAdd (Result,C1,Result)
429:      End;
430:
431:   RMul (X,Poly2CDegree-1]);
432:   CAdd (Result,X,Result);
433:
434:   Result.Pr := Result.Pr + Poly2CDegree]
435:
436: End;
437:
438:
439: {+-------------------------------------------------------------------+}
440: {! Variables y tipos que sólo se usan para probar                    !}
441: {+-------------------------------------------------------------------+}
442:
443: {$IfDef PRUEBA }
444:
445: Var
446:   X                      : MathRealType;
447:   Poly,PolyReal,PolyImag  : Coeff;
448:   I,Degree               : Byte;
449:   C,Result               : Complex;
450:
451: {$EndIf}
452:
453:
454: {+-------------------------------------------------------------------+}
455: {! I N I C I A L I Z A C I O N   Y / O   P R U E B A                 !}
456: {+-------------------------------------------------------------------+}
457: {$IfDef UINIT }
458:
459: Begin
460:
461: {$EndIf }
462:
463: {------------------ Código para inicializar la unidad ----------------}
464:
465:
466: {------------------ Código para probar la unidad ---------------------}
467:
468: {$IfDef PRUEBA }
469:
470: {  Write('Grado del polinomio -> ');
471:   Readln(Degree);
472:
473:   For I := 0 To Degree Do
474:      Begin
475:      Write('Coeficiente ',Degree-I,'-> ');
476:      Readln(Poly[I])
477:      End;}
478:
479:   Degree  :=  5;
```

```
480:   Poly[0] :=  1.0;
481:   Poly[1] :=  5.0;
482:   Poly[2] :=  0.0;
483:   Poly[3] := -3.0;
484:   Poly[4] := -7.0;
485:   Poly[5] := 11.0;
486:
487:   Repeat
488:      Write('Parte real -> ');
489:      Readln(C.Pr);
490:      Write('Parte imaginaria -> ');
491:      Readln(C.Pim);
492:
493:      PolyEval (Degree,Poly,C,Result);
494:
495:      Writeln('Parte real = ',Result.Pr:0:6,'. Parte imaginaria = ',Result.Pim:0:6)
496:
497:   Until C.Pr = 0.0;
498:
499:   PolyRoots (Degree,Poly,PolyReal,PolyImag,0.0001);
500:   I := 1;
501:
502:   While I <= Degree Do
503:      Begin
504:      Write('Raiz ',I:2,': ',PolyReal[I]:10:4,' ');
505:      X := PolyImag[I];
506:      If X <> 0.0 Then
507:         Begin
508:         If X > 0 Then
509:            Write('+',X:10:4)
510:         Else
511:            Write('-',-X:10:4);
512:         Writeln('i')
513:         End
514:      Else
515:         Writeln;
516:      Inc(I);
517:      End
518:
519: {$EndIf }
520:
521: End.
```

```
1: {                                                                    }
2: {                          General2.Pas                      V1.0    }
3: {                                                                    }
4: {     Módulo principal del proyecto de Control.                      }
5: {                                                                    }
6: {                                                                    }
7: {                          Mxico. Abril. 1991.                     }
8: {                                                                    }
9:
10: Uses
11:   Crt,
12:   Dos,
13:   TIDef,
14:   Menus,
15:   Poly2;
16:
17: Const
19:
19: {- Constantes de la interfaz y del menú ------------------------------------}
20:
21:   MTypeH  =       Invisible;
22:   MTypeV  =          Simple;
23:   AtrH    =              48;
24:   AtrV    =              31;
25: {  AtrT    =       BYTE('+');}
26:   AtrT    =             AtrV;
27:   AtrBox  =              94;
28:   AtrMS   =              30;
29:   AtrD    =              48;
30:   AtrF    =              19;
31:   HAcp    = AcceptWithLetter;
32:   VAcp    =     AcceptWithCR;
33:   Title   = '';
34:   Menu    = '    Archivos       Tcnicas       Diseño       Simulación       Utilerías   ';
35:   Op1     = 'Cargar Recoger Nuevo Salvar Escribir a Directorio Cambio_de_directorio Salida_a_DOS Fin';
36:   Op2     = '?Respuesta_en_tiempo: Variables_de_estado Función_de_transferencia '+
37:             '?Dominio_de_la_frecuencia: Nyquist Bode Nichols Lugar_geométrico';
38:   Op3     = 'Proporcional Proporcional_Integral Proporcional_Derivativo Proporcional_Integral_Derivativo '+
39:             'Pseudo_derivativo Estructuras_PID Sintonización Desempeño_específico '+
40:             'Red_de_adelanto Red_de_atraso Red_de_adelanto_y_atraso ?Aprender_un_método';
41:   Op4     = 'Caso_lineal Caso_no_lineal Sistemas_con_tiempo_de_retardo';
42:   Op5     = 'Regiones_de_trabajo Discretización Variables_de_estado->H(S) H(S)->Variables_de_estado '+
43:             'Formas_canónicas Patrón_de_polos_y_ceros Expansión_en_fracciones Transformación_a_plano_W '+
44:             'Identificación Estabilidad_de_Routh Caracterización Parámetros_de_diseño';
45:   Op6     = '';
46:   Op7     = '';
47:   Op8     = '';
48:   Op9     = '';
49:   Op10    = '';
50:   Opcions : Strings = (Op1,Op2,Op3,Op4,Op5,Op6,Op7,Op8,Op9,Op10);
51:
52:   Exp1    = 'Funciones orientadas a archivos : salvar, leer, terminar, etc.';
53:   Exp2    = 'Tcnicas de análisis en el tiempo y la frecuencia';
54:   Exp3    = 'Diseñar un sistema por varios algoritmos de control y redes, aprender un método de diseño';
55:   Exp4    = 'Tcnicas de simulación de sistemas lineales, no lineales y con retardo';
56:   Exp5    = 'Utilerías para el diseño';
57:   Exp6    = '';
58:   Exp7    = '';
59:   Exp8    = '';
```

```
  Exp9    = '';
  Exp10   = '';
  Exp     : Strings = (Exp1,Exp2,Exp3,Exp4,Exp5,Exp6,Exp7,Exp8,Exp9,Exp10);
  AtrEx   = 121;


{- Constantes del menú de utilerias ----------------------------------------}

  XAux            = 60;
  YAux            =  4;
  Regiones        = 'Continuo Discreto';
  Discretizacion  = '?Integración numerica: Aproximación rectangular hacia adelante Aproximación rectangular hacia atrás '+
                    '?Aproximación trapezoidal: Normal Predistorsión ?Mapeo de polos y ceros: '+
                    'Ceros no finitos en -1 Ceros no finitos en 0 Muestreador y retn';
  Canonicas       = 'Caso escalar Caso multivariable';
  Caracterizacion = 'Primer orden con tiempo muerto Srgundo orden con tiempo muerto';

  WTitle          =      ' Diálogo ';
  WBottom         = ' F1 para ayuda ';
  WType           =       DobleTecho;
  WAtrF           =                7;
  WAtrT           =              120;
  DAttr           =               15;
  AS              =              112;

Var
  OpcionH,OpcionV,OpUtil   : Integer;
  Buff                     : Array [1..2000] Of Char;


{+--------------------------------------------------------------------------+}
{| Procedimiento para inicializar las variables de la unidad MENUS          |}
{+--------------------------------------------------------------------------+}
Procedure InitMenus ;


Begin

{- Variables de la explicación del menú ------------------------------------}

  Explain    :=  True;
  AtrExplain :=  AtrEx;
  XExp       :=     1;
  YExp       :=    25;
  BuffPtr    :=  @Buff;
  ExpPtr     :=  @Exp;

{- Sombra transparente -----------------------------------------------------}

  Shadow     :=  True;
  AtrShadow  :=    AS;
  CharShadow :=   #0;

End;


{+--------------------------------------------------------------------------+}
{| Procedimiento para hacer el menú de utilerías                            |}
{+--------------------------------------------------------------------------+}
Procedure UtilMenu (Title : TString; T : MString; X : Byte) ;
```

```
120: Var
121:   SaveExp   : Boolean;
122:
123: Begin
124:
125:   SaveExp := Explain;
126:   Explain := False;
127:
128:   VFullDownMenu (Title,I,XAux - X,YAux,MTypeV,AtrV,AtrI,AtrBox,AtrNS,AtrD,AtrF,
129:                  MenuNotInWindow,StoreScreen,AcceptWithLetter,OpUtil);
130:
131:   Explain := SaveExp
132:
133: End;
134:
135: {+-------------------------------------------------------------------+}
136: {! P R O G R A M A   P R I N C I P A L                               !}
137: {+-------------------------------------------------------------------+}
138: Begin
139:
140:   TextAttr := 15;
141:   ClrScr;
142:   InitMenus  ;
143:   HideCursor ;
144:
145:   DrawWindowWithTitle (WTitle,WBottom,2,3,79,23,WType,WAtrF,WAtrT,Activa);
146:
147:   TextAttr := DAttr;
148:   ClrScr;
149:   Writeln('¡Esta es una muestra del color de la ventana de diálogo!');
150:   Writeln;
151:   Writeln('Se muestra este texto de prueba para observar las sombras del menú principal');
152:   Writeln('Este texto debe aparecer sombreado al escoger alguna de las opciones del menú');
153:
154:   Repeat
155:     BothMenus (Title,Menu,Opcions,1,1,MTypeH,MTypeV,
156:                AtrH,AtrV,AtrI,AtrBox,AtrNS,AtrD,AtrF,
157:                MenuNotInWindow,HAcp,VAcp,OpcionH,OpcionV);
158:
159:     Case OpcionH Of
160:       1   : AlvaroMain ;
161:       2   : ;
162:       3   : ;
163:       4   : ;
164:       5   : Case OpcionV Of
165:               1    : UtilMenu (' Regiones '        ,Regiones      , 0);
166:               2    : UtilMenu (' Discretizacion '  ,Discretizacion ,25);
167:               5    : UtilMenu (' Formas canónicas ',Canonicas      , 5);
168:               11   : UtilMenu (' Caracterización ' ,Caracterizacion,18)
169:               End;
170:       End;
171:
172:   Until (OpcionH = 1) And (OpcionV = 9);
173:
174:   Window(1,1,80,25);
175:   TextAttr := 15;
176:   ClrScr;
177:   ShowCursor(7,7)
178:
179: End.
```