UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

FACULTAD DE INGENIERÍA CIVIL

TESINA:

"Programación de los métodos de Bolzano, Aproximaciones Sucesivas, y Newton-Raphson para la solución de ecuaciones polinomiales"

QUE PARA OBTENER EL TITULO DE INGENIERO CIVIL
PRESENTA:

HECTOR YEPES BARRIGA

Asesor:

M.I. ENRIQUE OMAR NAVARRO CABALLERO

MORELIA, MICH. JULIO DEL 2009

ÍNDICE

INTRODUCCIÓN

Son las matemáticas, la herramienta más importante para los ingenieros de cualquier especialidad. Hoy en día, una forma muy eficiente de aplicar las matemáticas es con el uso de la computadora.

Como ejemplo de lo anterior se presenta la aplicación del lenguaje de programación Fortran 90 en la solución de ecuaciones polinomiales. En el presente trabajo se describen, primero los métodos numéricos que fueron programados, seguido de una explicación de los lenguajes de programación, teniendo Fortrand 90 un apartado especial; y por último la forma en que métodos numéricos y programación se pueden combinar, para crear una nueva herramienta que facilita la solución de problemas.

Las limitantes que pueden tener un programa pueden ser numerosas, dependiendo de la aplicación deseada; sin embargo, la diversidad de formas de programación, así como de los lenguajes de programación nos permiten una infinidad de posibilidades para llegar a elaborar los programas que solucionen nuestros problemas, de manera muy precisa. Es por ello que los programas que aquí se proponen pueden ser vistos como el inicio de proyectos de mayor complejidad, o como la base que da la idea para programas con objetivos completamente distintos.

Debe destacarse, que el lenguaje de programación utilizado en esta ocasión, está enfocado para el desarrollo de programas que ayuden a científicos e ingenieros en sus tareas diarias, lo que proporciona gran versatilidad, así como simplicidad en su desarrollo; y no será de extrañarse haber usado Fortran 90, aún cuando existan lenguajes de mejores características, pero que no cubren la necesidad de aplicación y solución de problemas casi inmediatas.

OBJETIVO

El objetivo de este trabajo es programar en Fortran 90 los métodos numéricos de Bolzano, Aproximaciones Sucesivas y Newton-Raphson para la solución de ecuaciones polinomiales, para así contar con una herramienta que permita eficientar la obtención de raíces de este tipo de ecuaciones.

CAPÍTULO 1

BÚSQUEDA DE CEROS DE FUNCIONES

Muchos problemas pueden modelarse matemáticamente como una ecuación $f(x)=0$, donde f es una cierta función de una variable x. Se trata pues, de hallar los valores de x que satisfacen la ecuación. Estos valores se llaman ceros de la función f o raíces de la ecuación $f(x)=0$, y se denotan por x. Gráficamente, los ceros de una función son los puntos de intersección de la gráfica $y=f(x)$ con el eje de las x.

Para algunos casos sencillos la ecuación puede resolverse analíticamente; sin embargo para otros casos ni siquiera se puede saber a priori la cantidad de ceros que tiene f en el campo real, es entonces, cuando se necesita utilizar una técnica numérica iterativa: a partir de una aproximación inicial a un cero se construye, iterativamente, una sucesión de aproximaciones que se detiene cuando se encuentra una suficientemente buena (para fines prácticos).

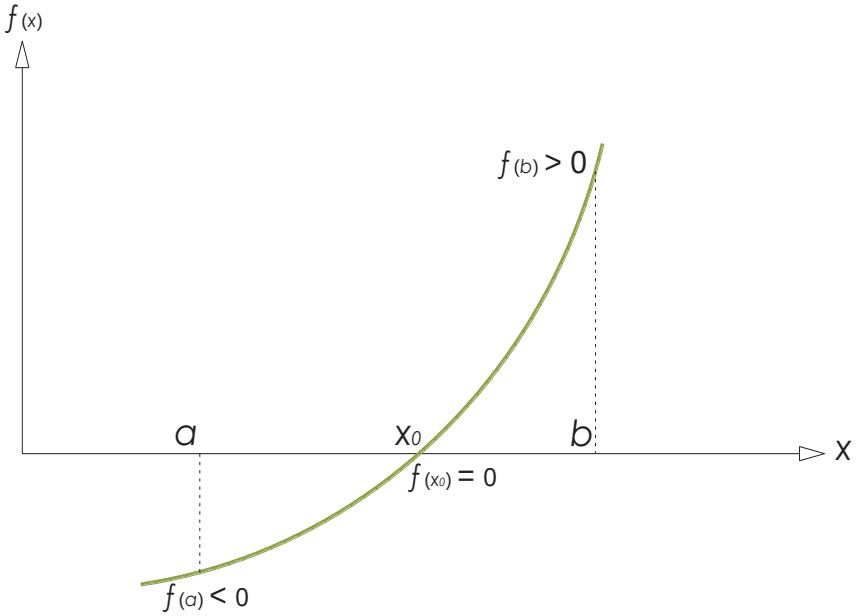
Hoy en día dichos métodos han tomado cierta importancia, por el uso de las computadoras, ya que pueden ser programados para facilitar su solución.

Método de la bisección.

Llamado también Método de Bolzano, está basado en el Teorema del valor intermedio, el cual afirma: "Toda función continua en un intervalo cerrado, una vez que alcanzó ciertos valores en los extremos del intervalo, entonces debe alcanzar todos los valores intermedios".

En otras palabras, sea $f(x)$ continua en un intervalo $[a,b]$ y supongamos que $f(a)< f(b)$. Entonces para cada z tal que $f(a) < z < f(b)$, existe un x_0 entre a y b tal que $f(x_0) = z$. La misma conclusión se obtiene para el caso de que $f(a) > f(b)$.

En particular, si $f(a)$ y $f(b)$ tienen signos opuestos, entonces un valor intermedio es precisamente $z=0$, y por lo tanto, el Teorema del Valor Intermedio nos asegura que debe existir x_0 tal que $f(x_0) = 0$, es decir, debe haber por lo menos una raíz de $f(x)$ en el intervalo $[a,b]$. Se puede ilustrar como sigue:



El método de bisección tiene los siguientes pasos:

Sea $f(x)$ continua,

1. Encontrar valores iníciales x_a, x_b, tales que $f(x_a)$ y $f(x_b)$ tienen signos opuestos, es decir:
$$f(x_a) \cdot f(x_b) < 0$$
2. La primera aproximación a la raíz se toma igual al punto medio entre x_a y x_b:
$$x_r = \frac{x_a + x_b}{2}$$
3. Evaluar $f(x_r)$. Forzosamente debemos caer en uno de los siguientes casos:
   - $f(x_a)$ y $f(x_r)$ tienen signos opuestos, y por lo tanto la raíz se encuentra en el intervalo $[x_a, x_r]$.
   - $f(x_a)$ y $f(x_r)$ tienen el mismo signo, entonces $f(x_r)$ y $f(x_b)$ tienen signos opuestos; por lo tanto la raíz se encuentra en el intervalo $[x_r, x_b]$.
   - $f(x_r) = 0$, la raíz está localizada

El proceso se vuelve a repetir hasta que la aproximación a cero sea satisfactoria.

El método de bisección es poco eficiente por la lentitud en el avance de las iteraciones, pero es casi segura la convergencia.

Si f es una función continua en el intervalo $[a,b]$ y $f(a)\cdot f(b) < 0$, entonces este método converge a la raíz de f. En otras palabras, se garantiza la convergencia si $f(a)$ y $f(b)$ tienen distinto signo.

Para ejemplificar este método considérese la ecuación: $f(x)= x^2 + 2x - 5$

Se trata de una ecuación de segundo grado, por lo que el número de raíces buscadas es dos.

Se comienza con la evaluación de la ecuación en varios números, preferentemente enteros, para encontrar los valore iníciales:

x	$f(x)$	
-5	10	
-4	3	} raíz
-3	-2	
-2	-5	
-1	-6	
0	-5	
1	-2	} raíz
2	3	

De la tabulación anterior se tiene que hay una raíz en el intervalo [-4,-3], y otra en el intervalo [1,2].

Aplicando el Teorema del valor intermedio iterativamente en el primer intervalo y mostrándolo a manera de tabla para facilitar su comprensión se tiene:

a	b	f(a)	f(b)	f(b) · f(b)	z=(a+b)/2	f(z)
-4	-3	3.000	-2.000	< 0	-3.5	0.250
-3.5	-3	0.250	-2.000	< 0	-3.25	-0.938
-3.5	-3.25	0.250	-0.938	< 0	-3.375	-0.359
-3.5	-3.375	0.250	-0.359	< 0	-3.438	-0.056
-3.5	-3.438	0.250	-0.056	< 0	-3.469	0.096
-3.469	-3.438	0.096	-0.056	< 0	-3.454	0.022
-3.454	-3.438	0.022	-0.056	< 0	-3.446	-0.017
-3.454	-3.446	0.022	-0.017	< 0	-3.45	0.003
-3.45	-3.446	0.003	-0.017	< 0	-3.448	-0.007
-3.45	-3.448	0.003	-0.007	< 0	-3.449	-0.002
-3.45	-3.449	0.003	-0.002	< 0	**-3.45**	0.003

Con el uso de tres decimales la mejor aproximación de la raíz es **-3.45**

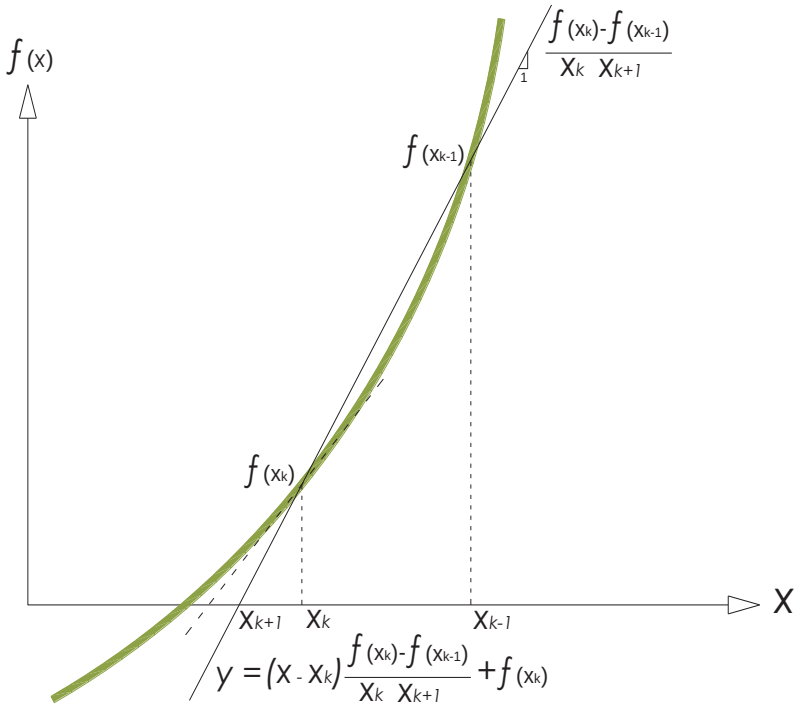
De manera análoga para el segundo intervalo:

a	b	f(a)	f(b)	f(b) · f(b)	z=(a+b)/2	f(z)
1	2	-2.000	3.000	< 0	1.5	0.250
1	1.5	-2.000	0.250	< 0	1.25	-0.938
1.25	1.5	-0.938	0.250	< 0	1.375	-0.359
1.375	1.5	-0.359	0.250	< 0	1.438	-0.056
1.438	1.5	-0.056	0.250	< 0	1.469	0.096
1.438	1.469	-0.056	0.096	< 0	1.454	0.022
1.438	1.454	-0.056	0.022	< 0	1.446	-0.017
1.446	1.454	-0.017	0.022	< 0	1.45	0.002
1.446	1.45	-0.017	0.002	< 0	1.448	-0.007
1.448	1.45	-0.007	0.002	< 0	1.449	-0.002
1.449	1.45	-0.002	0.002	< 0	**1.45**	0.002

Para este intervalo la mejor aproximación es **1.45** .

Método de Aproximaciones Sucesivas.

También conocido como el método de la secante, está basado en que la recta tangente de la curva $y = f(x)$ en el punto $(x_k, f(x_k))$ se aproxima mediante la recta secante que pasa por este punto, y el punto $(x_{k-1}, f(x_{k-1}))$ obtenido en la iteración anterior. La intersección de esta recta secante con el eje de las x se toma como siguiente aproximación x_{k+1}. Este método necesita dos aproximaciones iníciales (x_0 y x_1) para poder trazar la primera recta secante. Se ilustra a continuación:



En otras palabras, la derivada $f'(x)$ no se calcula, sino que se aproxima por:

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

No todas las ecuaciones pueden resolverse por este método, solamente si el valor absoluto de la derivada de la función $f(x)$ en vecindad de la raíz es menor que la unidad.

Una secuencia útil para la aplicación del método puede ser:

1. Sea $f(x)$ continua, y $x=0$ una raíz de $f(x)$. Se suma x a ambos miembros de la ecuación:

$$x = x+f(x)$$

2. Se evalúa un valor inicial en la ecuación anterior, de donde se obtiene el valor para la iteración siguiente:

$$x_{k+1}=x_k+f(x_k)$$

3. Se repite el proceso hasta que la diferencia de las dos últimas iteraciones sea convenientemente aproximada a cero. Si la diferencia aumenta cada vez, el método diverge.

Por ejemplo, si $f(x) = x^2 - 0.9$; es una ecuación de segundo grado, por lo que se buscan dos raíces que la satisfagan.

Sumando x a ambos lados de la ecuación,

$$x = x^2 + x - 0.9$$

y tabulando los resultados:

k	x_k	x_{k+1}
1	-1	-0.9
2	-0.9	-0.99
3	-0.99	-0.91
4	-0.91	-0.982
5	-0.982	-0.918
29	-0.951	-0.947
30	-0.947	-0.95
31	-0.95	-0.948
32	-0.948	-0.949
33	-0.949	**-0.948**

Una raíz con aproximación a tres decimales es **-0.948** .

Se repite la secuencia para la segunda raíz:

k	x_k	x_k+1
1	1	1.1
2	1.1	1.41
3	1.41	2.498
4	2.498	7.838
5	7.838	68.372

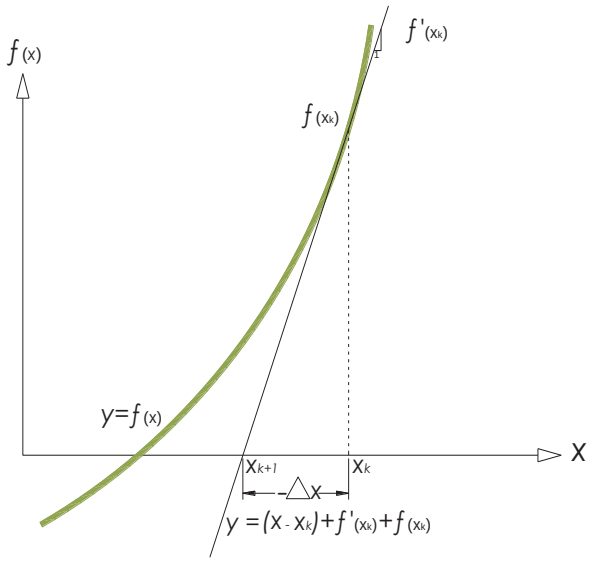
} Diverge

Para la segunda raíz el método diverge.

Método de Newton – Raphson

El método de Newton – Raphson asume que la función $f(x)$ es derivable sobre un intervalo cerrado $[a,b]$. Entonces $f(x)$ tiene una pendiente definida y una única línea tangente en cada punto en $[a,b]$. La tangente en $(x_0, f(x_0))$ es una aproximación a la curva de $f(x)$ cerca del punto $(x_0, f(x_0))$. En consecuencia, el cero de la línea tangente es una aproximación del cero de $f(x)$.

Este método se puede deducir gráficamente de la siguiente manera: dada una cierta x_k, se aproxima la función f por la recta tangente a la curva $y=f(x)$ en el punto $(x_k, f(x_k))$. La pendiente de esta recta es justamente la derivada de f en x_k. A continuación se toma la intersección de esta recta con el eje de las x como siguiente aproximación x_{k+1}. En la figura se puede observar que $f(x_k)$, Δx_{k+1} y la pendiente $f'(x_k)$ están relacionados según:



En resumen se puede decir que para la ejecución del método se aplican los siguientes pasos:

1. Si es $f(x)$ una curva continúa, se obtiene $f'(x)$.
2. Se propone o se calcula un valor inicial x_k, preferentemente cerca de la raíz.
3. Se calcula el valor de x_k para la siguiente iteración con la formula:

$$x_{k+1} = x_k - \frac{f(x)}{f'(x)}$$

4. Se continúa el proceso hasta que la diferencia entre las iteraciones sea aceptablemente pequeña.

Un ejemplo de aplicación se puede hacer con $f(x) = 200x^2 - 43x - 12$,

donde $f'(x) = 400x - 43$

Se obtienen los valores iníciales evaluando $f(x)$ con varios números:

x	*f (x)*	
-5	5203	
-4	3360	
-3	1917	
-2	874	
-1	231	} raíz
0	-12	} raíz
1	145	
2	702	
3	1659	

Tabulando los valores para la primera raíz:

k	xk	*f* (xk)	*f*'(xk)	xk+1
0	-1	231	-443	-0.479
1	-0.479	54.485	-234.6	-0.247
2	-0.247	10.823	-141.8	-0.171
3	-0.171	1.2012	-111.4	-0.16
4	-0.16	0	-107	**-0.16**

Para la primera raíz el valor correspondiente es **-0.16**, con aproximación de tres decimales.

Se obtiene la segunda raíz de la misma manera:

k	x_k	$f(x_k)$	$f'(x_k)$	x_k+1
0	1	145	357	0.594
1	0.594	33.025	194.6	0.424
2	0.424	5.7232	126.6	0.379
3	0.379	0.4312	108.6	0.375
4	0.375	0	107	**0.375**

El valor para la segunda raíz es **0.375**, con la misma aproximación que para la primera.

CAPÍTULO 2

GENERALIDADES DE LA PROGRAMACIÓN

Conceptos Básicos

Se le llama programación al instrumento operativo que ordena y vincula cronológica, espacial y técnicamente las actividades y recursos necesarios para alcanzar, en un tiempo dado, determinadas metas y objetivos. Implica el para qué, cómo y cuándo.

Dentro de la informática, los programas son los elementos que forman el software, que es el conjunto de las instrucciones que ejecuta el hardware de una computadora para realizar una tarea determinada, sin embargo pueden estar también integrados al propio hardware, y en computadoras especializadas las instrucciones operativas están incorporadas en el sistema de circuitos.

Para la elaboración de programas de cierta envergadura o complejos, con ciertas garantías de calidad, es conveniente seguir alguno de los modelos ya existentes, en donde la programación es sólo una de las etapas del proceso. Los modelos de desarrollo de software los aborda una disciplina específica del campo de la informática: la ingeniería del software.

Existe una tendencia a identificar el proceso de creación de un programa informático con la programación que es cierta cuando se trata de programas pequeños para uso personal y que dista de la realidad cuando se trata de grandes proyectos.

El proceso de creación de software, desde el punto de vista de la ingeniería, incluye los siguientes pasos:

1. Reconocer la necesidad de un programa para solucionar un problema o identificar la posibilidad de automatización de una tarea.
2. Recoger los requisitos del programa. Clarificar las tareas que el programa realizará.
3. Realizar el análisis de los requisitos del programa. Debe quedar claro cómo debe realizar el programa las cosas que debe hacer. Las pruebas que comprueben la validez del programa se pueden especificar en esta fase.
4. Diseñar la arquitectura del programa. Descomponer el programa en partes de complejidad abordable.
5. Implementar el programa. Consiste en realizar un diseño detallado, especificando completamente todo el funcionamiento del programa, tras lo cual la codificación debería resultar inmediata.

6. Implantar (instalar) el programa. Consiste en poner el programa en funcionamiento junto con los componentes que pueda necesitar (bases de datos, redes de comunicaciones, etc.).

Tipos de programación

Se han propuesto diversas técnicas de programación cuyo objetivo es mejorar tanto el proceso de creación de software como su mantenimiento. Entre ellas, se pueden mencionar las siguientes:

- Programación estructurada.

  La programación estructurada es una teoría de programación que consiste en construir programas de fácil comprensión. Es especialmente útil, cuando se necesitan realizar correcciones o modificaciones después de haber concluido un programa o aplicación. Al haberse utilizado la programación estructurada, es mucho más sencillo entender la codificación del programa que se habrá hecho en diferentes secciones. Ellas están disponibles en todos los lenguajes modernos de programación imperativa en forma de sentencias.

  Está basada en una metodología de desarrollo de programas llamada refinamientos sucesivos: Se plantea una operación como un todo y se divide en segmentos más sencillos o de menor complejidad. Una vez terminados todos los segmentos del programa, se procede a unificar las aplicaciones realizadas por el equipo de programadores. Si se ha utilizado adecuadamente la programación estructurada, esta integración debe ser sencilla y no presentar problemas al realizar la misma, y de haber algún problema, será rápidamente detectable para su corrección.

  Así combinando esquemas sencillos, se pueden llegar a construir sistemas amplios y complejos pero de fácil entendimiento.

  La representación gráfica de la programación estructurada se realiza a través de diagramas de flujo, los cuales representan el programa con sus entradas, procesos y salidas.

- Programación modulada.

  La programación modular consta de varias secciones, llamadas modulos, que interactúan a través de llamadas a procedimientos, integrando el programa en su totalidad.

  En la programación modular, el programador principal coordina las llamadas a los módulos secundarios y pasa los datos necesarios en forma de parámetros.

  A su vez cada módulo puede contener sus propios datos y llamar a otros módulos o funciones.

- Programación orientada a objetos.

  La programación orientada a objetos, intenta simular el mundo real a través del significado de objetos que contiene características y funciones. Los lenguajes orientados a objetos se clasifican como lenguajes de quinta generación.

  Como su mismo nombre indica, la programación orientada a objetos se basa en la idea de un objeto, que es una combinación de variables locales y procedimientos llamados métodos que juntos conforman una entidad de programación.

  El termino encapsulación se usa para describir la combinación de estructuras de datos y de métodos que son manipulados por el objeto. La llamada a un objeto es lo que se denomina pasar un "aviso" a un objeto.

  Dicho de otra manera, encapsular significa, reunir y controlar el grupo resultante como un todo y no individualmente. Por otro lado, la abstracción es un término externo al objeto, que controla la forma en que es visto por los demás.

  En este tipo de programación la modularidad se considera de la siguiente manera: Un programa grande siempre será más complicado que la suma de varios programas pequeños, con lo que se considera ventajoso dividir un gran sistema en diversos módulos; en general, para este concepto, se puede considerar la misma definición de programación modulada.

Como un término más, en la programación orientada a objetos tenemos la jerarquía, la cual consiste en la clasificación y organización de las abstracciones según su naturaleza. El más claro ejemplo de jerarquía es la herencia, la cual se define como una jerarquía de extracciones, y la relación entre clases, donde se comparte la estructura y el comportamiento de una o más clase considerada como clases superiores o una superclase, con lo cual se resume que la herencia es una unidad independiente por si misma heredada de una abstracción o superclase. Un ejemplo cotidiano lo encontramos en las aplicaciones que existen actualmente en el mercado, donde un formulario cualquiera hereda las características de una ventana del sistema operativo Windows (Maximizar, Minimizar, Cerrar)

- Programación declarativa.

  La programación declarativa es una forma de programación que implica la descripción de un problema dado en lugar de proveer una solución para dicho problema, dejando la interpretación de los pasos específicos para llegar a dicha solución a un intérprete no especificado. La programación declarativa adopta, por lo tanto, un enfoque diferente al de la programación imperativa tradicional, como por ejemplo el caso de C, la que requiere que el programador especifique una lista completa de instrucciones de modo de ejecutar una tarea determinada.

  En otras palabras, la programación declarativa provee el "que", pero deja el "como" liberado a la implementación particular del intérprete. Por lo tanto se puede ver que la programación declarativa tiene dos fases bien diferenciadas, la declaración y la interpretación.

  Es importante señalar que a pesar de referirnos a interprete no estamos limitados a "lenguajes interpretados" en el sentido habitual del término, sino que también podemos estar trabajando con "lenguajes compilados".

  Los lenguajes declarativos describen la relación entre variables en términos de funciones y reglas de inferencia o transformación. El intérprete produce, mediante algoritmo de procesado de estas declaraciones, un resultado.

La programación declarativa incluye tanto aspectos de programación lógica como de programación funcional, fue también conocida como “programación orientada al valor”, pero este término ha dejado de usarse últimamente.

La principal desventaja de la programación declarativa es que no puede resolver cualquier problema ya que está restringida al subconjunto de problemas para los que el intérprete fue diseñado.

Otra desventaja de la programación declarativa es que está relacionada con la eficiencia. Dado que es necesaria una fase de interpretación extra, en la cual se deben evaluar todas las consecuencias de todas las declaraciones realizadas, el proceso es relativamente más lento que en la programación imperativa, en que los cambios de estado del sistema están dados por instrucciones particulares y no por un conjunto de condiciones arbitrariamente grande.

A pesar de lo anterior existen algunas ventajas en el uso de la programación declarativa. Entre las cuales se destaca que la solución de un problema se puede realizar con un nivel de abstracción considerablemente alto, sin entrar en detalles de implementación irrelevantes, lo que hace a las soluciones más fácil de entender por las personas. La resolución de problemas complejos es resuelta por el intérprete a partir de la declaración de las condiciones dadas.

La programación declarativa es muy usada en la resolución de problemas relacionados con inteligencia artificial, problemas de condiciones de borde, bases de datos, configuración, y comunicación entre procesos, sin embargo ningún leguaje declarativo se aproxima en popularidad a los lenguajes imperativos.

Calidad de un programa

La programación debe perseguir la obtención de programas de calidad. Para ello se establece una serie de factores que determinan la calidad de un programa. Algunos de los factores de calidad más importantes son los siguientes:

- *Delimitación*. Un programa es correcto si hace lo que debe hacer tal y como se estableció en las fases previas a su desarrollo. Para determinar si un programa hace lo que debe, es muy importante especificar claramente la tarea que hará el programa antes de desarrollarlo y, una vez acabado, compararlo con lo que realmente hace.

- *Claridad*. Es muy importante que el programa sea lo más claro y legible posible, para facilitar así su desarrollo y posterior mantenimiento. Al elaborar un programa se debe intentar que su estructura sea sencilla y coherente, así como cuidar el estilo en la edición; de esta forma se ve facilitado el trabajo del programador, tanto en la fase de creación como en las fases posteriores de corrección de errores, ampliaciones, modificaciones, etc. Fases que pueden ser realizadas incluso por otro programador, con lo cual la claridad es aún más necesaria para que puedan continuar el trabajo fácilmente.

- *Eficiencia*. Se trata de que el programa, además de realizar aquello para lo que fue creado (es decir, que este bien delimitado), lo haga gestionando de la mejor forma posible los recursos que utiliza. Normalmente, al hablar de eficiencia de un programa, se suele hacer referencia al tiempo que tarda en realizar la tarea para la que ha sido creado y a la cantidad de memoria que necesita, pero hay otros recursos que también pueden ser de consideración al obtener la eficiencia de un programa, dependiendo de su naturaleza (espacio en disco que utiliza, tráfico de red que genera, etc.).

- *Portabilidad*. Un programa es portable cuando tiene la capacidad de poder ejecutarse en una plataforma, ya sea hardware o software, diferente a aquélla en la que se elaboró. La portabilidad es una característica muy deseable para un programa, ya que permite que pueda llegar a mayor cantidad de usuarios más fácilmente.

Secuencia de programación

Para garantizar que un programa cubra las anteriores características, es importante el diseño de un buen algoritmo, este debe ser una secuencia no ambigua, finita y ordenada de instrucciones que han de seguirse para resolver un problema. Nótese que la secuencia de instrucciones en sí (la ejecución) es la que debe ser finita, no el número de pasos realizados.

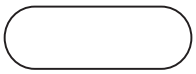
La descripción de un problema puede hacerse de varias maneras, pero es necesario desarrollar la habilidad de hacerlo lo más claro y especifico posible, para entonces, encontrar la manera de solucionarlo.

Sin embargo, no todos los problemas pueden considerarse solucionables, ya sea por el planteamiento propio del problema o porque el algoritmo no sea ejecutable, esto último puede ser incluso por el tiempo que tardaría en resolverse.

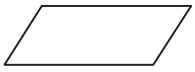
Se recomienda, entonces, antes de empezar a escribir el programa; hacer un análisis de las tareas que el programa debe realizar y como deben programarse. El hecho de invertir cierto tiempo en el diseño del programa conlleva generalmente a un ahorro tanto en el tiempo de programación como en el de ejecución.

Una herramienta útil para facilitar la compresión de algoritmos es su representación, usada principalmente de dos maneras: diagramas de flujo y pseudocódigos.

En la representación de algoritmos mediante diagramas de flujo se utilizan una serie de símbolos con un significado predeterminado, que se unen mediante flechas que indican el orden en que deben ejecutarse las instrucciones. Aunque estos símbolos no están totalmente estandarizados, algunos de ellos están ampliamente aceptados:

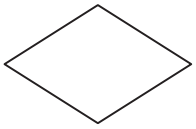


Terminal, representa el inicio y el fin de un programa. También puede representar una parada o una interrupción.

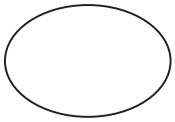


Entrada/Salida, cualquier tipo de introducción de datos en la memoria desde los periféricos o registro de información procesada en un periférico.

Proceso, cualquier tipo de operación que pueda originar cambio de valor, formato de posición de la información almacenada en memoria, operaciones aritméticas, de transformaciones etc.



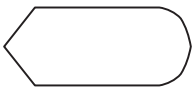
Decisión, indica operaciones lógicas o de comparación entre datos y en función del resultado de la misma determina cual de los distintos caminos alternativos del programa se debe seguir.



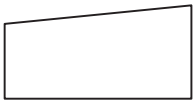
Conector misma página, sirve para enlazar dos partes cualesquiera de un diagrama a través de un conector en la salida y otro conector en la entrada. Se refiere a la conexión en la misma página del diagrama.

Conector diferente página, conexión entre dos puntos del organigrama situado en páginas diferentes.

Impresora, se utiliza en lugar del símbolo de salida.



Pantalla, utilizado en lugar del símbolo de salida.



Teclado, utilizado en lugar del símbolo de entrada.

Línea de flujo, indica el sentido de la ejecución de las operaciones.

En la representación mediante pseudocódigo, las instrucciones se especifican de forma similar a como se programan en un lenguaje de programación. Es importante resaltar que no existe ningún convenio acerca de cómo representar las instrucciones. En este sentido, la forma en que cada persona las detalla depende mucho del lenguaje de programación que acostumbra a utilizar.

Por otro lado todos los ordenadores funcionan en el sistema binario o lenguaje máquina. En consecuencia, todos los programas deben ser traducidos a dicho sistema, independientemente del lenguaje de programación en que hayan sido diseñados; por tal motivo el código fuente debe ser sometido a un proceso de traducción para convertirse en un lenguaje interpretable por el procesador.

Normalmente para la creación de un archivo ejecutable (un típico .exe para Microsoft Windows o Dos) conlleva dos pasos. El primer paso se llama compilación y traduce el código fuente escrito en un lenguaje de programación almacenado en un archivo a código en bajo nivel (normalmente en código objeto, no directamente a lenguaje máquina). El segundo paso se llama *enlazado* (del inglés *link* o *linker*), en el cual se enlaza el código de bajo nivel generado de todos los ficheros y subprogramas que se han mandado compilar y se añade el código de las funciones que hay en las bibliotecas del compilador para que el ejecutable pueda comunicarse directamente con el sistema operativo, traduciendo así finalmente el código objeto a código máquina, y generando un módulo ejecutable.

Estos dos pasos se pueden hacer por separado, almacenando el resultado de la fase de compilación en archivos objetos (un típico.obj para Microsoft Windows, DOS o para Unix); para enlazarlos en fases posteriores, o crear directamente el ejecutable; con lo que la fase de compilación se almacena sólo temporalmente. Un programa podría tener partes escritas en varios lenguajes, que se podrían compilar de forma independiente y luego enlazar juntas para formar un único módulo ejecutable.

Finalmente es posible ejecutar el programa, naturalmente, siempre se ejecuta el archivo ejecutable (exe). Los resultados obtenidos deben ser analizados.

Es importante resaltar que si en alguna de las anteriores fases se produce un error, su corrección debe realizarse sobre el programa fuente; por tanto, es preciso volver a realizar las fases de compilación, ensamblado y ejecución.

Lenguajes de programación

Los lenguajes de programación son una invención relativamente reciente. Han sido desarrollados por un periodo corto de años y su mejoramiento e innovación son constantes.

Un lenguaje de programación provee a la computadora digital las herramientas para procesar datos y obtener resultados en la solución de problemas. Así se han desarrollado diferentes tipos de lenguajes de programación con características particulares que los hacen adecuados para tareas específicas.

Aunque en esencia los primeros lenguajes desarrollados realizan las mismas tareas que los más recientes, estos últimos son preferidos por sus avances que facilitan tanto su comprensión como su utilización.

Una constante en los lenguajes de programación, ha sido la capacidad de las computadoras, ya que en ésta se basa la optimización del lenguaje, de tal manera que en un principio sólo se abordaban dos conceptos principales: los lenguajes maquina y los ensambladores.

Según su nivel de abstracción los lenguajes de programación pueden clasificarse en Lenguajes máquina, Lenguajes de bajo nivel y Lenguajes de alto nivel.

Lenguajes máquina

El lenguaje máquina de una computadora consta de cadenas de números en sistema binario (ceros y unos) y es el único que entienden directamente los procesadores. Todas las instrucciones preparadas en cualquier lenguaje de máquina tienen por lo menos dos partes. La primera es el comando u operación, que dice a la computadora cuál es la función que va a realizar. Todas las computadoras tiene un código de operación para cada una de sus funciones. La segunda parte de la instrucción es el operando, que indica a la computadora donde hallar o almacenar los datos y otras instrucciones que se van a manipular; el número de operandos de una instrucción varía en las distintas computadoras.

En el principio de la computación éste era el lenguaje que tenía que "hablar" el ser humano con la computadora y consistía en insertar en un tablero miles de conexiones y alambres y encender y apagar interruptores.

Aunque en la actualidad ya no se emplea, es importante reconocer que ya no es necesario que nos comuniquemos en este lenguaje de "unos" y "ceros", pero es el que internamente una computadora reconoce o "habla".

Lenguajes de bajo nivel.

Son lenguajes con poca abstracción existente entre el lenguaje y el hardware, su principal característica es que se desarrolla a nivel de instrucciones, lo que genera una programación muy detallada, además de ser muy adaptables al equipo y obtener una alta velocidad con un mínimo uso de memoria.

Este tipo de lenguajes debe ser usado por profesionales del software, ya que presentan gran dificultad en su utilización puesto que es necesario conocer un centenar de instrucciones así como la arquitectura de la maquina con todo detalle.

Por algunos, el lenguaje máquina es considerado como de bajo nivel de primera generación por su propio desuso.

Lenguajes de alto nivel.

Este tipo de lenguaje logra independencia de la máquina, es decir que es posible utilizar un mismo programa en diferentes equipos con la única condición de disponer de un programa traductor o compilador, que lo suministra el fabricante, para obtener el programa ejecutable en lenguaje binario de la máquina que se trate sin ser necesario conocer el hardware específico de dicha máquina.

Su uso se facilita con su capacidad para aproximarse al lenguaje natural, eliminando muchas de las posibilidades de cometer errores, ya que utilizan palabras en lugar de cadenas de símbolos.

Además se incluyen rutinas de uso frecuente como son las de entrada/salida, funciones matemáticas, manejo de tablas, etc., que figuran en una especie de librería del lenguaje, de tal manera que se pueden utilizar siempre que se quiera sin necesidad de programarlas cada vez.

La trascendencia del lenguaje Fortran

Considerado el lenguaje de programación de la ciencia y la ingeniería, Fortran ha permanecido en uso durante largo tiempo, además ha sido uno de los lenguajes de alto nivel más ampliamente usado, así como estandarizado.

Fue desarrollado en un periodo de tres años, de 1954 a 1957, por un grupo de la IBM a cargo de John Backus. El objetivo era crear un sistema que permitiera convertir programas escritos en notación matemática en instrucciones máquina para la computadora IBM 704. Recibe su nombre de la unión de las palabras en inglés: *Formula Translation*, y fue rápidamente acogido por la comunidad de científicos que necesitaban la solución de problemas con altos contenidos aritméticos; es esta última característica la que ha logrado que continúe vigente.

Sin embargo, mucho se ha sugerido el abandono de Fortran, y la simple sustitución por otro lenguaje moderno, fundamentando principalmente la falta de características, comparado frente a muchos otros lenguajes de programación modernos. No obstante hay fuertes razones para no hacerlo. Las pocas ventajas que presenta Fortran hoy en día siempre han estado ahí, y un cambio causaría una incompatibilidad con los estándares previos y el código existente. Así mismo existen otras tres grandes razones para no cambiar a otro lenguaje de programación:

1. Es, en muchos casos, la mejor opción para la solución de problemas por la forma en que opera, de hecho es reconocido ser tan avanzado como muchos otros lenguajes modernos, a pesar de su carencia de propiedades.
2. Existe una gran inversión en programas en Fortran, y un cambio a otro lenguaje implicaría la reescritura a un alto costo de muchos programas.
3. La última y la más sutil de las razones, es que cambiar a otro lenguaje de programación estaría ligada al reentrenamiento de muchos programadores, esto tendría un particular impacto en aquellos usuarios que lo utilizan como una herramienta más para la solución de sus propios problemas.

CAPÍTULO 3

ESPECIFICACIONES BÁSICAS DE FORTRAN 90

Formato Libre de Fortran 90

Para llevar a cabo un proyecto de programación es importante, primero saber las normas de escritura del programa según el lenguaje de programación a utilizar, en el caso de Fortran 90, el formato es “libre”, lo que facilita su utilización.

Dicho formato, permite que cada línea pueda tener como máximo 132 caracteres, donde las instrucciones simplemente se escriben, y los comentarios van precedidos por un signo de admiración (“!”). Todos los espacios en blanco son ignorados, de tal manera que se pueden usar todos los necesarios para incrementar la legibilidad del programa.

Estructura de un programa en Fortran 90

La estructura de un programa puede ser variable, según el tipo de programación que se usa (estructurada, modulada, etc.), pero en general se pueden identificar cuatro zonas principales:

1. Nombre del programa. Formada por la sentencia PROGRAM seguida del nombre del programa.
2. Declaración de variables, constantes y estructuras del programa, es de definición y no se pueden situar sentencias de otro tipo.
3. Cuerpo del programa, donde se define el proceso de los datos.
4. Final del programa, constituida por la sentencia END PROGRAM, sirve para separar el programa principal de los sub-programas, es una línea única e imprescindible.

Tipo de Datos

Fortran fue diseñado para que científicos e ingenieros tuvieran una herramienta para la fácil solución de problemas mediante el uso de las computadoras. Así las declaraciones pueden ser de varios tipos:

- Operaciones numéricas simples.
- Operaciones con números complejos.
- Contadores de veces a realizar cierta operación.
- Operaciones basadas en decisiones lógicas.
- Expresión de resultados.

Entonces los datos a procesar deben ser, a su vez, diferentes unos de otros según la tarea que desarrolle cada uno, tal como se muestra en el siguiente recuadro.

Tipo de Dato	Descripción
Real	Para operaciones simples
Complejo	Para operaciones con números complejos
Entero	Contadores
Lógico	Para toma de decisiones
De caracter	Expresión de resultados

Estos son los cinco tipos de datos más frecuentes, y son especificados en el lenguaje Fortran como intrínsecos. Cualquier otro tipo de dato puede ser especificado por el programador como datos del tipo “derivado”.

El tipo de dato determina la manera en que estos pueden ser procesados. Por ejemplo, con los datos reales, complejos y enteros, las operaciones que se pueden realizar son la suma, la resta, la multiplicación, la división, la exponenciación, la negación y la comparación; con los datos lógicos las operaciones son la negación, la equivalencia, la conjunción y la disyunción.

Es decir que los datos intrínsecos tienen operadores también intrínsecos. Por el contrario con los datos derivados los operadores son definidos por el usuario.

Declaración de variables.

Las declaraciones son usadas para precisar el tipo y otros atributos de las variables del programa. El tipo es la característica más importante por especificar, sin embargo existen otras propiedades que las variables poseen de hecho, existen variables como las subrutinas que no necesitan que se especifique el tipo de variable directamente. Entonces la declaración determina la forma en que dichas variables podrán ser utilizadas en el programa.

Fortran tiene palabras clave, que se usan para la declaración de variables, se enlistan a continuación:

Declaración	Palabra clave
Tipo	INTEGER, entero REAL (doble precisión), Real COMPLEX, compleja LOGICAL, lógica CHARACTER, caracter TYPE, tipo definido por el usuario.
Propiedades de Matriz	DIMENSION , dimensión ALLOCATABLE, asignable
Propiedades de Indicador	POINTER, indicador TARGET, objetivo
Especificación de valores	DATA, datos PARAMETER, prametros
Accesibilidad y uso	PUBLIC, público PRIVATE, privado INTENT OPTIONAL, opcional SAVE, seguro
Propiedades de procedimiento	EXTERNAL, externo INTRINSIC, intrínseco

De la declaración que demos a las variables depende el tipo de procesos que les podemos dar, y por consiguiente el tipo de operadores que podemos utilizar, así como el resultado que obtengamos de la aplicación de dichos operadores, por ejemplo si operamos con números enteros, el resultado será otro número entero.

Operadores Aritméticos

La lista completa de operadores aritméticos de Fortran es la siguiente:

Operador	Descripción
**	Exponenciación
/ y *	División y multiplicación
+ y -	Suma y resta

Los anteriores son operadores “binarios” en el sentido de que se usan siempre entre dos números y generan un resultado que depende de ambos números. (Sin embargo los signos de suma y resta pueden ser aplicados a un solo número para determinar su posición en la recta numérica).

No obstante lo anterior pueden estos operadores pueden combinarse para realizar operaciones más complejas que la simple operación de dos números, aunque las operaciones se van haciendo entre dos números aunque haya más signos.

El orden descendente de la lista anterior de operadores representa también la jerarquía con que las operaciones se realizan, y para cambiar el orden en que las operaciones se realizan es necesario el uso de paréntesis.

Si juntos están dos operadores de la misma jerarquía, las operaciones se realizan de izquierda a derecha, excepto en el caso de la exponenciación que las operaciones se realizan de derecha a izquierda. De cualquier manera, es siempre recomendable el uso de paréntesis y el previo análisis de las operaciones para evitar errores en el programa, tales como el uso de cero o los resultados de raíces negativas, para lo cual se usarán números complejos.

Otros operadores

En Fortran existe una librería de funciones para realizar ciertos cálculos que se emplean con frecuencia, las más comunes son las siguientes:

Función	Descripción	Argumento	Valor de retorno
ABS (X)	Valor absoluto de x	Entero o real	Igual al argumento
ACOS(X)	Arco coseno de x(rad)	Real	Real
ASIN(X)	Arco seno de x(rad)	Real	Real
ATAN(X)	Arco tangente de x (rad)	Real	Real
COS(X)	Coseno de x (rad)	Real	Real
COSH(X)	Coseno hiperbólico de x (x)	Real	Real
EXP(X)	Función exponencial	Real	Real
INT(X)	Parte entera de x	Real	Entero
FRACTION(X)	Parte fraccionaria de x en su representación de número real.	Real	Real
LOG(X)	Logaritmo natural de x	Real	Real
LOG10(X)	Logaritmo en base 10 de x	Real	Real
REAL(X)	Conversión de x al tipo REAL	Entero	Real
SIN(X)	Seno de x (rad)	Real	Real
SQRT(X)	Raíz cuadrada de x	Real	Real
TAN(X)	Tangente de x (rad)	Real	Real
TANH(X)	Tangente hiperbólica de x (rad)	Real	Real

Transferencia de datos.

Para que un programa sea útil, debe existir una comunicación usuario-maquina. Es así como la computadora recibe los datos correctos, y arroja un resultado para que el usuario pueda recogerlo y utilizarlo a sus propios intereses.

Entrada simple de datos.

En un programa a las variables se les debe asignar cierto valor, y procurando el principio de portabilidad de programas, estos deben poder ser aplicados para la solución de diferentes problemas similares; entonces las variables pueden tomar valores diferentes dependiendo del problema para el que el programa se aplique.

La forma más simple de darle a la computadora la instrucción de obtener el valor de determinada variable, es con el comando READ*; el cual permite leerlo mientras el programa corre.

Siguiendo prácticamente el mismo principio, es también posible, obtener el valor de datos desde un archivo de texto, donde estén escritos los valores necesarios. Se hace con el comando OPEN seguido del nombre del archivo. Esta herramienta es significativamente útil, sobre todo cuando el programa se encuentra en fase de prueba, ya que rara vez un programa funciona correctamente la primera vez que se corre, entonces no será necesario escribir el valor de cada variable cada vez que se pruebe.

Salida simple de datos.

La misma importancia que tiene el hecho de introducir el valor de los datos, tiene la acción de que se expresen los resultados obtenidos del proceso al que fueron sometidos dichos datos.

Para ello se usa la sentencia PRINT*, seguida del nombre de la variable de la cual se dará a conocer su valor. Se puede considerar como el comando inverso a READ*, y sintácticamente se usan igual.

El comando WRITE es también usado en la salida de datos, con la variante en la especificación de la unidad donde se almacenara el valor. Esta última característica permite la posibilidad de usar el comando OPEN para guardar en un archivo de disco, o incluso imprimir el valor usando el numero de la unidad de la impresora.

Entrada y salida de datos con formato.

No obstante que los comandos READ, PRINT y WRITE son por si mismo muy útiles, existe una manera de crear un área de programa más limpia y legible; entonces se tiene la posibilidad de usar estos comandos con cierto formato.

Los especificadores de formato, pueden ser usados una sola vez cuando se determinan directamente después de escribir el comando, o pueden ser usados varias veces usando etiquetas y el comando FORMAT.

Los especificadores de formato más comunes son los siguientes:

FORMATO		VALOR DE RETORNO
nIw	nIw.m	Datos enteros.
nFw.d		Datos de punto flotante en notación decimal.
nEw.d	nEw.dEe	Datos de punto flotante en notación exponencial.
nESw.d	nESw.dEe	Datos de punto flotante en notación científica.
nENw.d	Nenw.dEe	Datos de punto flotante en notación de ingeniería.
nGw.d	nGw.dEe	Datos de punto flotante en notación general.
nA	nAw	Datos de caracter.
nX		Espacio horizontal en blanco.
/		Espacio vertical (nueva línea)
\		Adjuntar a la línea anterior *

*Deshabilita la nueva línea que por defecto se envía con cada instrucción PRINT o WRITE

Donde:

w: Número entero positivo que especifica el ancho de campo.

m: Número entero positivo que especifica el número mínimo de dígitos a ser leídos / desplegados.

d: Número entero positivo que especifica el número de dígitos a la derecha del punto decimal.

e: Número entero positivo que especifica el número de dígitos en el exponente.

n: Número entero positivo que especifica el número de veces que se repetirá el formato.

La toma de decisiones y los ciclos.

Una propiedad importante de las computadoras es la habilidad para toma decisiones, esto aunado con la capacidad para repetir un mismo proceso infinidad de veces, hacen de la computadora una potente herramienta en la solución de problemas.

En Fortran la manera de desarrollar dichas características se cuenta con los comandos IF y DO, y ambos necesitan de relaciones lógicas en las cuales basar la función a realizar.

Operadores lógicos.

La relación lógica más común podría ser aquella que tiene solo dos constantes literales, es decir, cierto (.TRUE.) y falso (.FALSE.).

Sin embargo existen expresiones lógicas que amplían las posibilidades, y se pueden formar de dos maneras: combinando las expresiones numéricas con los seis operadores lógicos simples, o combinando los operadores lógicos simples con variables y los cinco operadores lógicos compuestos.

Operador lógico simple	Significado
.LT. o <	Menor que
.GT. o >	Mayor que
.EQ. o ==	Igual que
.LE. o <=	Menor o igual que
.GE. o >=	Mayor o igual que
.NE. o /=	Diferente a

Operador Lógico compuesto	Significado
.NOT.	Negación
.AND.	Intersección
.OR.	Unión
.EQV.	Equivalencia
.NEQV.	No equivalencia.

Comando IF

En general y en su forma más simple el comando IF condiciona a la computadora a realizar cierta acción, en caso de que se cumpla un operador lógico.

Pero existen más posibilidades para decisiones más complejas, ya que es viable ejecutar más de una acción si la condición se cumple, se hace con el comando THEN. Es importante resaltar que para este caso es necesario "cerrar" el condicional, es decir determinar las acciones que se van a realizar si la condición se cumple, se hace con la escritura de END IF después de la última acción.

Otra opción es el hecho de tener que evaluar varias operaciones lógicas, lo que se puede hacer con el uso de la sentencia ELSE, en combinación con THEN, y es también necesario finalizar con END IF.

Además, es posible anidar la sentencia IF, lo que, aunado con los operadores lógicos, incrementa aún más las combinaciones posibles entre las condiciones y las Instrucciones.

Comando DO

Para completar el desarrollo de las características deseables en un programa, debe ser posible el desarrollo de ciclos, herramienta muy poderosa en cualquier lenguaje de programación. Para el caso particular de Fortran, los ciclos se hacen con la sentencia DO.

El uso de DO es necesario cuando un conjunto de instrucciones tiene que ser repetida hasta cumplir cierta condición, llamada "Sentencia de control". Sera necesario, entonces delimitar el numero de instrucciones que entraran en ciclo con las palabras clave END DO.

Las utilidades de un ciclo dentro de la programación pueden son variadas, por eso, la sentencia DO se puede utilizar también de varias formas, dependiendo del tipo que sea la sentencia de control:

- Si toma una progresión de valores hasta alcanzarun valor predeterminado se usa DO-CONTADOR.
- Si es una expresión lógica, que determina que el ciclo continuará mientras esta sea cierta, entonces se usa DO-WHILE.
- Si se encuentra dentro del bloque de instrucciones, como restricción al ciclo, entonces se usa DO-SEMI INFINITO.

El ciclo DO, también puede ser anidado, de tal manera que sea posible varios contadores variando simultáneamente. Y, así mismo, se pueden también anidar ciclos con condicionales, es decir hacer combinaciones con las sentencias IF y DO.

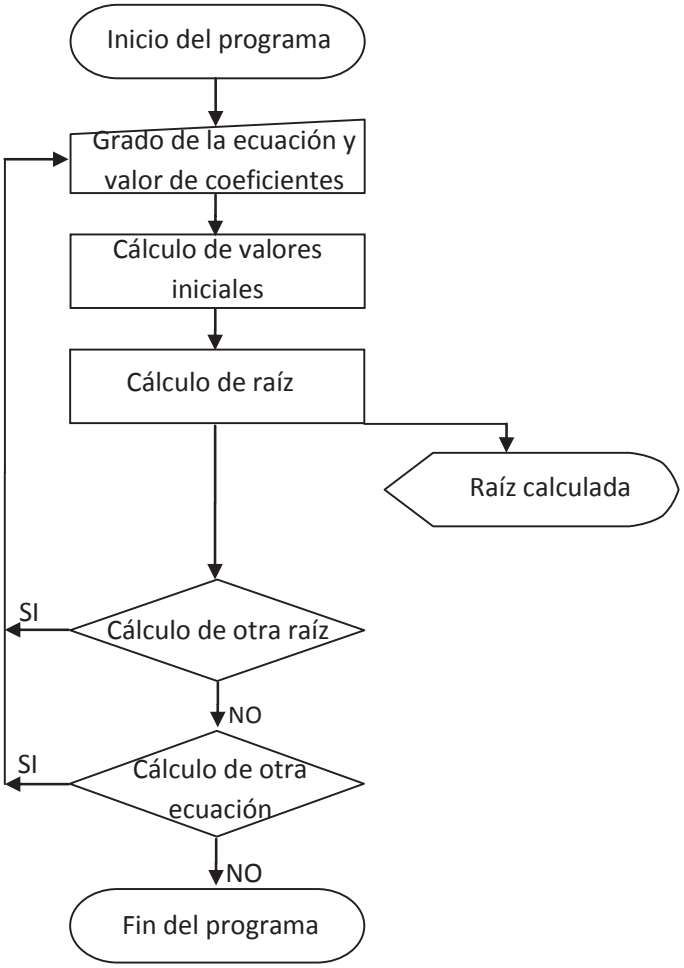
CAPÍTULO 4

DESARROLLO DE LOS PROGRAMAS

Descripción general.

Se han programado los tres métodos, para la solución de raíces de ecuaciones, antes descritos: Bisección, aproximaciones sucesivas y Newton-Raphson; un programa diferente para cada método. De tal manera que es posible aplicarlos en la búsqueda de las raíces reales de ecuaciones polinomiales.

En general los tres programas siguen la misma secuencia de cálculo, expresada en el siguiente diagrama de flujo:



Datos de entrada.

Los programas, realizan el cálculo de ecuaciones de *n* grado, con números de coeficientes de cualquier rango, entonces es necesario que el usuario entre los datos particulares de la ecuación a resolver.

Así, primero los programas piden se especifique el grado de la ecuación, mediante la sentencia READ* y una variable INTEGER. En seguida esta misma variable, de la cual se determina también el número de coeficientes, será el punto de control para un ciclo DO, que servirá para que los programas "lean" uno a uno los coeficientes de la ecuación, guardados en una misma variable del tipo REAL, con declaración DIMENSION, es decir los valores de los coeficientes serán considerados en lo sucesivo como una matriz.

Para ilustrar lo anterior, se presenta esta parte del código fuente del programa del método de bisección:

```
PROGRAM Bolzano_3000

implicit none

INTEGER::g,i,k,t,u,opc
REAL(8)::x,fd,fh,d,h,a,b,m,fa,fb,fm,da,db,dm
REAL(8),dimensi□n(10000)::coef


PRINT "(2/,1x,a)","----------   Para calcular las raices de una ecuacion:"
PRINT "(2/,14x,a,12x,a,\)","Escribe el grado de la ecuacion","--  "
READ*,g
PRINT "(2/,1x,a)","----------   Ahora escribe los coeficientes en orden:"
PRINT "(22x,a)","Si es negativo incluye el signo"
DO i=1,(g+1),1
        PRINT "(2/,14x,a,1x,i100,12x,a,\)","coeficiente",i,"--  "
        READ*,coef(i)
END DO
```

Cálculo de los valores inciales.

En el desarrollo de los tres métodos es necesario tener primero valores iniciales que indiquen cierto intervalo en el que se deben buscar las raíces, para poder garantizar una convergencia segura y rápida, dicho intervalo debe ser lo más corto posible.

La determinación de estos intervalos, en los programas, se realiza con un ciclo DO, iniciado en un número considerablemente pequeño, y con un avance lo más aproximado a cero posible. En cada ciclo la ecuación es evaluada, y el resultado es comparado con el anterior, cuando la multiplicación de estos es negativa, se tiene un intervalo donde hay una raíz.

A continuación la forma en que lo anterior se escribe en el código fuente del programa del método de Aproximaciones sucesivas.

```
x=-1000000.05
c=0
do
        fx=0
        fk=0
        do u=1,(g+1),1
                w=coef(u)*(x**((g+1)-u))
                z=coef(u)*((x+0.1)**((g+1)-u))
                fx=fx+w
                fk=fk+z
        end do
        if ((fx*fk)<=0) exit
        x=x+0.1
end do
```

Cálculo de las raíces.

En este paso, los tres programas difieren, por ser este el momento en que cada programa usa método propio de cálculo.

Programa del método de la bisección.

Con el intervalo definido, se definen tres variables reales, la primera toma el valor del primer número del intervalo, la segunda el valor del segundo número, y la tercera variable, toma el valor del promedio de las primeras dos, es decir, el valor intermedio. Con un ciclo DO, se evalúa la ecuación en las tres variables, obteniendo así, tres resultados más, definidas por otras tres variables reales, seguido por una sentencia IF-ELSE, con la cual se define el valor de los números del nuevo intervalo, o define la salida del ciclo DO cuando el valor del valor intermedio sea igual o menor a un numero aproximadamente igual a cero.

Es decir:

```
a=x
b=x+0.0001
do
        m=((a+b)/2)
        fa=0
        fb=0
        fm=0
        do u=1,(g+1),1
                da=coef(u)*(a**((g+1)-u))
                db=coef(u)*(b**((g+1)-u))
                dm=coef(u)*(m**((g+1)-u))
                fa=fa+da
                fb=fb+db
                fm=fm+dm
        end do
        if(fm<=0.0000001)exit
        if((fa*fm)<0)then
                a=a
                b=m
        else
                a=m
                b=b
        end if
end do
```

Programa del método de las aproximaciones sucesivas.

Para el cálculo por este método, lo primero es definir una variable de inicio, que se toma igual al valor más chico del intervalo que se ha encontrado, con esta nueva variable se evalúa la ecuación; el nuevo valor obtenido se multiplica por un número que garantice que la derivada de la ecuación evaluada sea menor que cero, y se le suma nuevamente la variable, este será el número para la nueva iteración. El ciclo termina con una sentencia IF, cuando la diferencia entre las últimas dos iteraciones es aproximadamente igual a cero.

En código fuente, queda escrito como sigue:

```
do
        t=0
        do o=1,(g+1),1
                fs=coef(o)*(s**((g+1)-o))
                t=t+fs
        end do
        p=s+(0.00001*t)
        if ((p-s)<0.00001)exit
        s=p
end do
```

Programa del método del Newton-Raphson.

Para este último caso, el valor inferior del intervalo es evaluado en la ecuación y en la derivada de la ecuación, en un ciclo DO que al valor primero le resta el resultado de la división de las evaluaciones realizadas, obteniendo así el valor de la siguiente iteración. El programa sale del ciclo cuando la diferencia de las últimas dos iteraciones es aproximadamente igual a cero, definido con la sentencia IF. De la manera siguiente:

```
do
        ll=0
        do r=1,(h+1),1
                e=hctr(r)*(o**((h+1)-r))
                ll=ll+e
        end do
        g=0
        do a=1,(h+1),1
                m=(hctr(a)*((h+1)-a))*(o**((h+1)-(a+1)))
                g=g+m
        end do
rr=o-(ll/g)
if((rr-o)<=0.0000001) exit
o=rr
end do
```

Siguientes raíces y finalización del programa.

Todos los procedimientos anteriores, son parte de otro ciclo DO, en el que nuevamente el valor del grado de la ecuación introducido por el usuario en un principio es el valor del control, ya que una vez que el programa ha calculado el valor de raíces el cual es igual al valor del grado de la ecuación, entonces el programa termina el cálculo para esa ecuación.

Una vez que el programa termino de calcular todas las raíces, entonces aparecerá en pantalla un menú, diseñado para dar al usuario la oportunidad de realizar otro cálculo sin tener la necesidad de reiniciar el programa. Esta hecho con la sentencia GOTO y dos etiquetas, una que envía al usuario al inicio del programa y otra al final, según la opción elegida antes en el menú.

Recomendaciones y restricciones de los programas.

Los métodos usados, no convergen para todas las ecuaciones, por lo que se recomienda usar los tres métodos para una misma ecuación y poder estar seguros de la solución, ya que los programas pueden calcular raíces erróneas.

Se enlistan a continuación las restricciones para los programas:

- El valor de inicio se propone por el programador, por lo que todas las raíces que se encuentren antes de dicho número no serán calculadas, por las características propias de cada método, el valor es diferente en los tres casos, para el programa del método de bisección el rango de búsqueda de raíces es a partir de 1,000.00005, para el método de aproximaciones sucesivas 1,000,000.05, y para el de Newton-Raphson es 100,000.05.
- Los programas presentaban un conflicto al utilizar arreglos con memoria dinámica, por lo tanto se elaboraron con memoria estática, aprovechando la velocidad de las computadoras de hoy; pero esto conlleva a una restricción más: los programas son útiles para ecuaciones de hasta grado 99, porque están diseñados con memoria hasta para 100 coeficientes.

Para cualquier modificación necesaria se anexa el código fuente de los tres programas, y se podrán hacer de manera particularmente conveniente, sin tener que escribir todo el proceso.

Ejemplo de aplicación.

Considérese la búsqueda de las raíces de la siguiente ecuación:

$$3x^4 - 9x^3 - 2x^2 + 15x - 5 = 0$$

Resolviendo la ecuación por el método de la bisección, para lo cual se busca primero el intervalo donde se encuentren las raíces:

Para x= (-3): $3(-3)^4 - 9(-3)^3 - 2(-3)^2 + 15(-3) - 5 = 418$

Para x= (-2): $3(-2)^4 - 9(-2)^3 - 2(-2)^2 + 15(-2) - 5 = 77$

Para x= (-1): $3(-1)^4 - 9(-1)^3 - 2(-1)^2 + 15(-1) - 5 = -10$

Para x= (0) : $3(0)^4 - 9(0)^3 - 2(0)^2 + 15(0) - 5 = -5$

Para x= (1) : $3(1)^4 - 9(1)^3 - 2(1)^2 + 15(1) - 5 = 2$

Para x= (2) : $3(2)^4 - 9(2)^3 - 2(2)^2 + 15(2) - 5 = -7$

Para x= (3) : $3(3)^4 - 9(3)^3 - 2(3)^2 + 15(3) - 5 = 22$

Por los cambios de signo se identifican los siguientes intervalos:

(-2) < (x) < (-1)
(0) < (x) <(1)
(1) < (x) < (2)
(2) < (x) < (3)

Se busca la raíz para el primer intervalo:

1ª iteración

$a = (-2)$

$b = (-1)$

$$m = \frac{a+b}{2} = \frac{(-2)+(-1)}{2} = -1.5$$

$$f(-1.5) = 3(-1.5)^4 - 9(-1.5)^3 - 2(-1.5)^2 + 15(-1.5) - 5 = 13.563$$

2ª iteración

$a = (-1.5)$

$b = (-1)$

$$m = \frac{a+b}{2} = \frac{(-1.5)+(-1)}{2} = -1.25$$

$$f(-1.25) = 3(-1.25)^4 - 9(-1.25)^3 - 2(-1.25)^2 + 15(-1.25) - 5 = -1.973$$

3ª iteración

$a = (-1.5)$

$b = (-1.25)$

$$m = \frac{a+b}{2} = \frac{(-1.5)+(-1.25)}{2} = -1.375$$

$$f(-1.375) = 3(-1.375)^4 - 9(-1.375)^3 - 2(-1.375)^2 + 15(-1.375) - 5 = 4.714$$

4ª iteración

$a = (-1.375)$

$b = (-1.25)$

$$m = \frac{a+b}{2} = \frac{(-1.375)+(-1.25)}{2} = -1.313$$

$$f(-1.313) = 3(-1.313)^4 - 9(-1.313)^3 - 2(-1.313)^2 + 15(-1.313) - 5 = 1.145$$

5ª iteración

$a = (-1.313)$

$b = (-1.25)$

$$m = \frac{a+b}{2} = \frac{(-1.313)+(-1.25)}{2} = -1.282$$

$$f(-1.282) = 3(-1.282)^4 - 9(-1.282)^3 - 2(-1.282)^2 + 15(-1.282) - 5 = -0.451$$

6ª iteración

$a = (-1.313)$

$b = (-1.282)$

$$m = \frac{a+b}{2} = \frac{(-1.313)+(-1.282)}{2} = -1.298$$

$$f(-1.298) = 3(-1.298)^4 - 9(-1.298)^3 - 2(-1.298)^2 + 15(-1.298) - 5 = 0.358$$

7ª iteración

$a = (-1.298)$

$b = (-1.282)$

$$m = \frac{a+b}{2} = \frac{(-1.298)+(-1.282)}{2} = -1.29$$

$$f(-1.29) = 3(-1.29)^4 - 9(-1.29)^3 - 2(-1.29)^2 + 15(-1.29) - 5 = -0.05$$

8ª iteración

$a = (-1.298)$

$b = (-1.29)$

$$m = \frac{a+b}{2} = \frac{(-1.298)+(-1.29)}{2} = -1.294$$

$$f(-1.294) = 3(-1.294)^4 - 9(-1.294)^3 - 2(-1.294)^2 + 15(-1.294) - 5 = 0.153$$

9ª iteración

$a = (-1.294)$

$b = (-1.29)$

$$m = \frac{a+b}{2} = \frac{(-1.294)+(-1.29)}{2} = -1.292$$

$$f(-1.292) = 3(-1.292)^4 - 9(-1.292)^3 - 2(-1.292)^2 + 15(-1.292) - 5 = 0.051$$

10ª iteración

$a = (-1.292)$

$b = (-1.29)$

$$m = \frac{a+b}{2} = \frac{(-1.292)+(-1.29)}{2} = -1.291$$

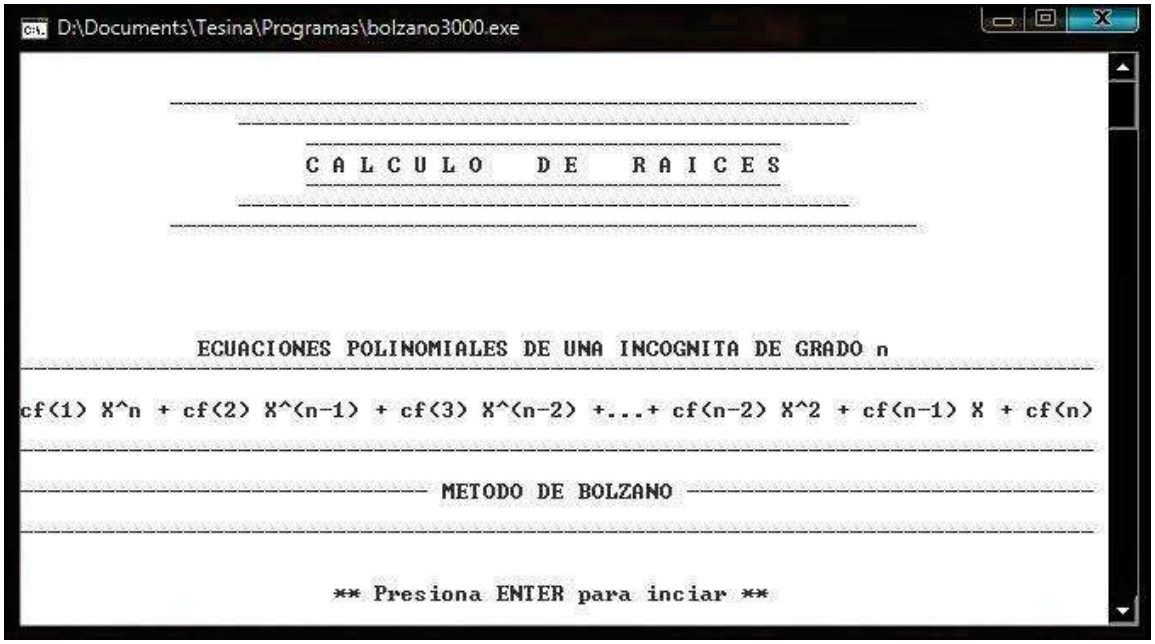
$$f(-1.291) = 3(-1.291)^4 - 9(-1.291)^3 - 2(-1.291)^2 + 15(-1.291) - 5 = 0.$$

Así, la primera raíz encontrada es **x=(-1.291)**.

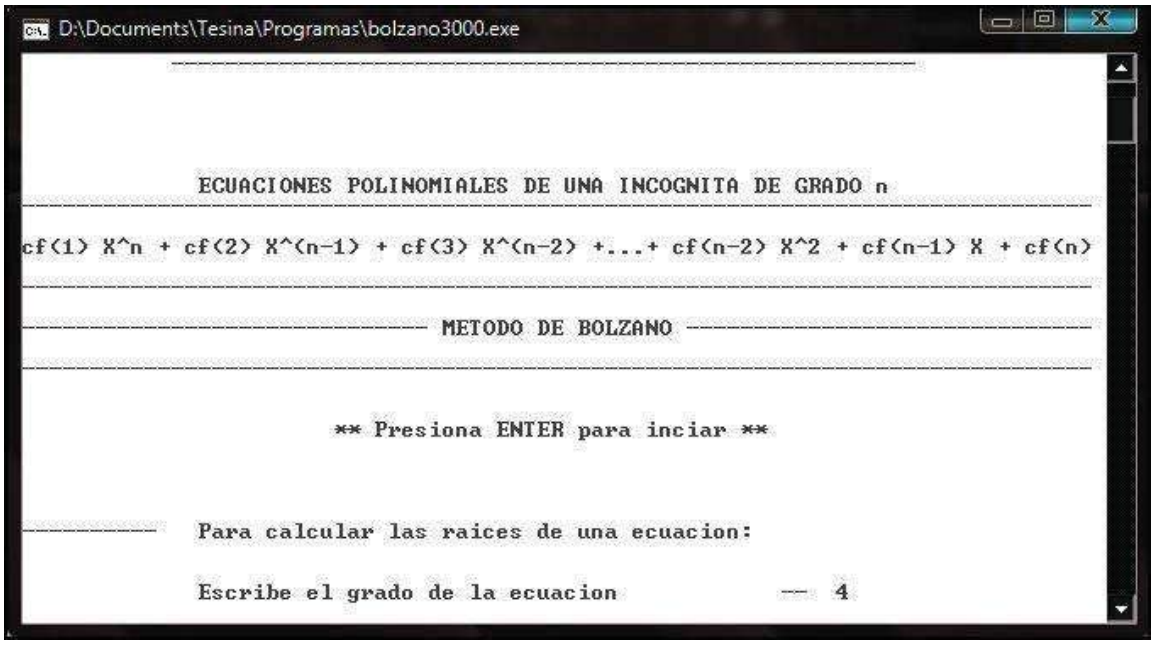
De manera análoga, para los otros intervalos se obtiene:

La segunda raíz, **x=(0.382)**,
La tercera raíz, **x=(1.291)**,
La cuarta raíz, **x=(2.618).**

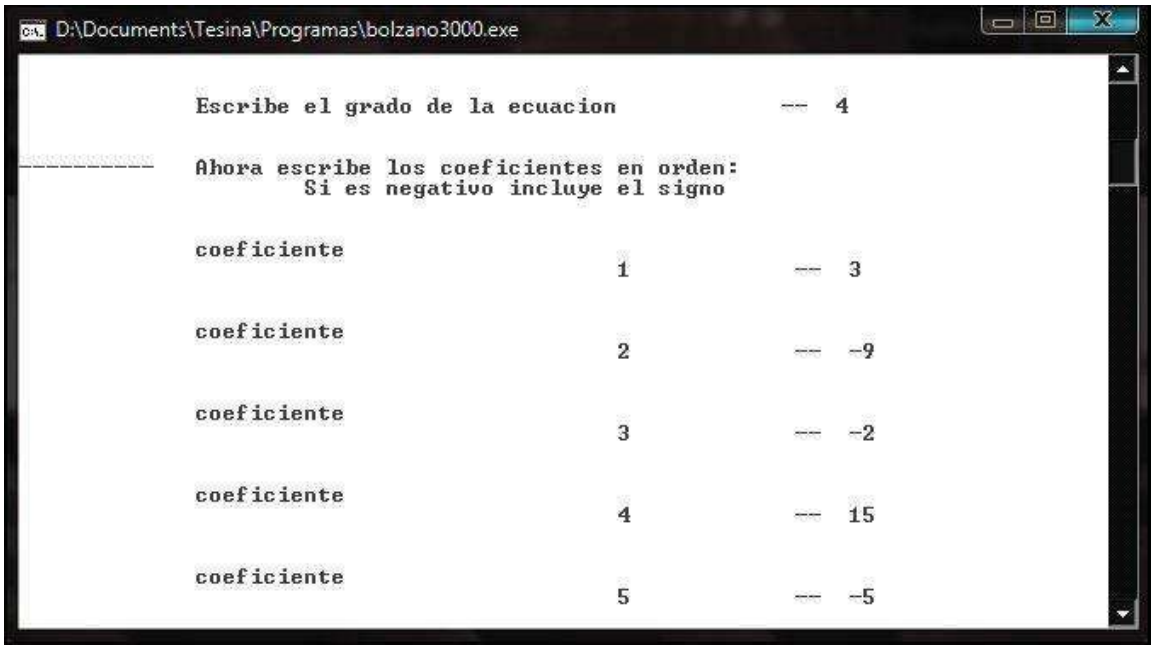
Entonces, si se utiliza el programa para resolver esta ecuación aparece en pantalla la bienvenida al programa, donde se describe la forma de general de las ecuaciones polinomiales seguido del nombre del método a que usa el programa, que para este caso es el método de Bolzano:



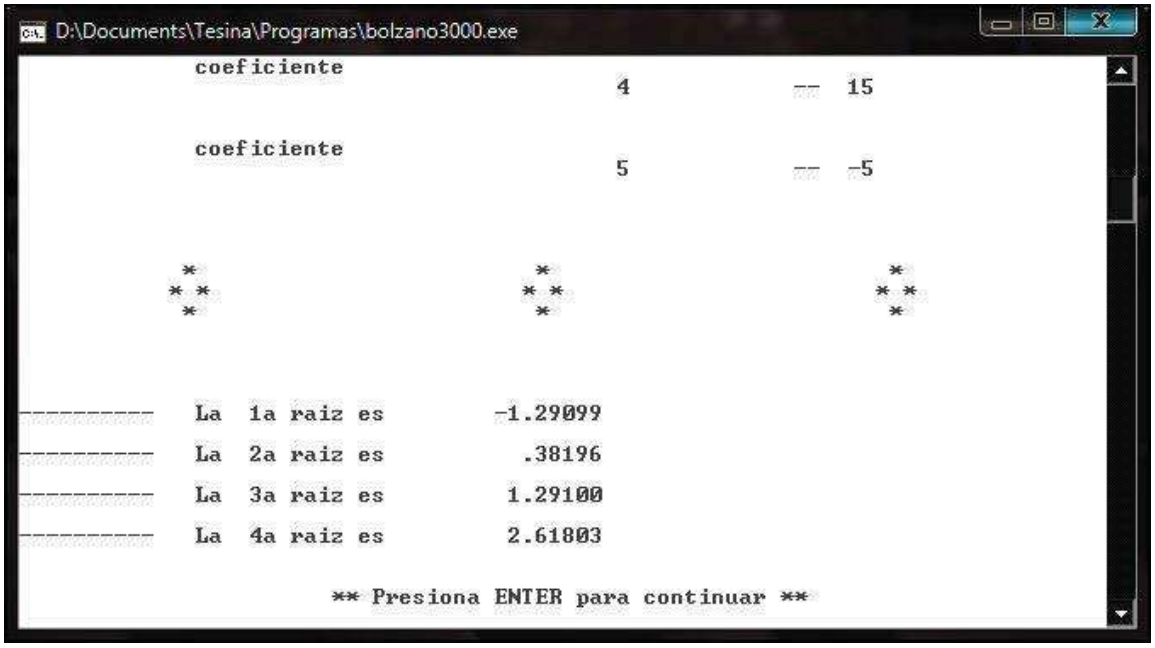
Para iniciar el programa es necesario presionar la tecla ENTER, y se inicia la recopilación de los datos con el grado de la ecuación:



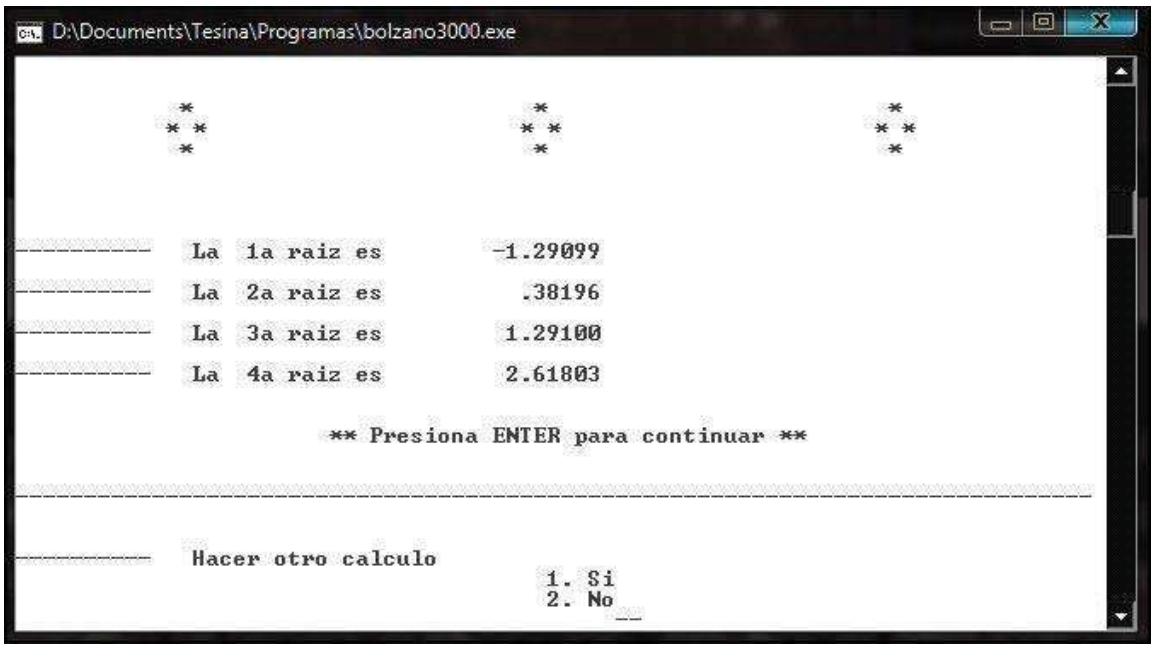
Para el ejemplo que se desarrolla el grado es 4; el siguiente dato a recopilar son los coeficientes de la ecuación, los cuales se piden uno a uno, especificando el signo en caso de ser negativo:



Seguido el programa realiza el cálculo de las raíces y las imprime en pantalla.



Finalmente el programa ofrece la posibilidad de realizar otro calculo sin tener que reiniciarlo, para ello, se presiona ENTER después de obtener los resultados y se elige en el menú la opción “si, hacer otro cálculo” para que el proceso se repita; o la opción “no, hacer otro calculo” si es el caso:



Si la misma ecuación se resuelve por aproximaciones sucesivas, se suma entonces x a $f(x)$:

$$(3x^4 - 9x^3 - 2x^2 + 15x - 5) + x = 3x^4 - 9x^3 - 2x^2 + 16x - 5$$

Se inician las iteraciones considerando los mismos intervalos obtenidos para el método anterior:

K=1, x=(-2)

$$f(-2) = 3(-2)^4 - 9(-2)^3 - 2(-2)^2 + 16(-2) - 5 = 75$$

K=2, x=(75)

$$f(75) = 3(75)^4 - 9(75)^3 - 2(75)^2 + 16(75) - 5 = 91114945$$

Para esta raíz el método converge, por lo que se hace necesario multiplicar la ecuación por un número tal que genere una derivada de la ecuación menor a uno en valor absoluto,:

$$0.01(f(x)) = 0.03x^4 - 0.09x^3 - 0.02x^2 + 0.15x - 0.05$$

Ahora se suma x y se inician las iteraciones nuevamente:

$$(0.03x^4 - 0.09x^3 - 0.02x^2 + 0.15x - 0.05) + x = 0.03x^4 - 0.09x^3 - 0.02x^2 + 1.15x - 0.05$$

K=1, x=(-2)

$$f(-2) = 0.03(-2)^4 - 0.09(-2)^3 - 0.02(-2)^2 + 1.15(-2) - 0.05 = -1.23$$

K=2, x=(-1.23)

$$f(-1.23) = 0.03(-1.23)^4 - 0.09(-1.23)^3 - 0.02(-1.23)^2 + 1.15(-1.23) - 0.05 = -1.259$$

K=3, x=(-1.259)

$$f(-1.259) = 0.03(-1.259)^4 - 0.09(-1.259)^3 - 0.02(-1.259)^2 + 1.15(-1.259) - 0.05 = -1.275$$

K=4, x=(-1.275)

$$f(-1.275) = 0.03(-1.275)^4 - 0.09(-1.275)^3 - 0.02(-1.275)^2 + 1.15(-1.275) - 0.05 = -1.283$$

K=5, x=(-1.283)

$$f(-1.283) = 0.03(-1.283)^4 - 0.09(-1.283)^3 - 0.02(-1.283)^2 + 1.15(-1.283) - 0.05 = -1.287$$

K=6, x=(-1.287)

$$f(-1.287) = 0.03(-1.287)^4 - 0.09(-1.287)^3 - 0.02(-1.287)^2 + 1.15(-1.287) - 0.05 = -1.289$$

K=7, x=(-1.289)

$$f(-1.289) = 0.03(-1.289)^4 - 0.09(-1.289)^3 - 0.02(-1.289)^2 + 1.15(-1.289) - 0.05 = -1.29$$

K=8, x=(-1.29)

$$f(-1.29) = 0.03(-1.29)^4 - 0.09(-1.29)^3 - 0.02(-1.29)^2 + 1.15(-1.29) - 0.05 = -1.291$$

K=9, x=(-1.291)

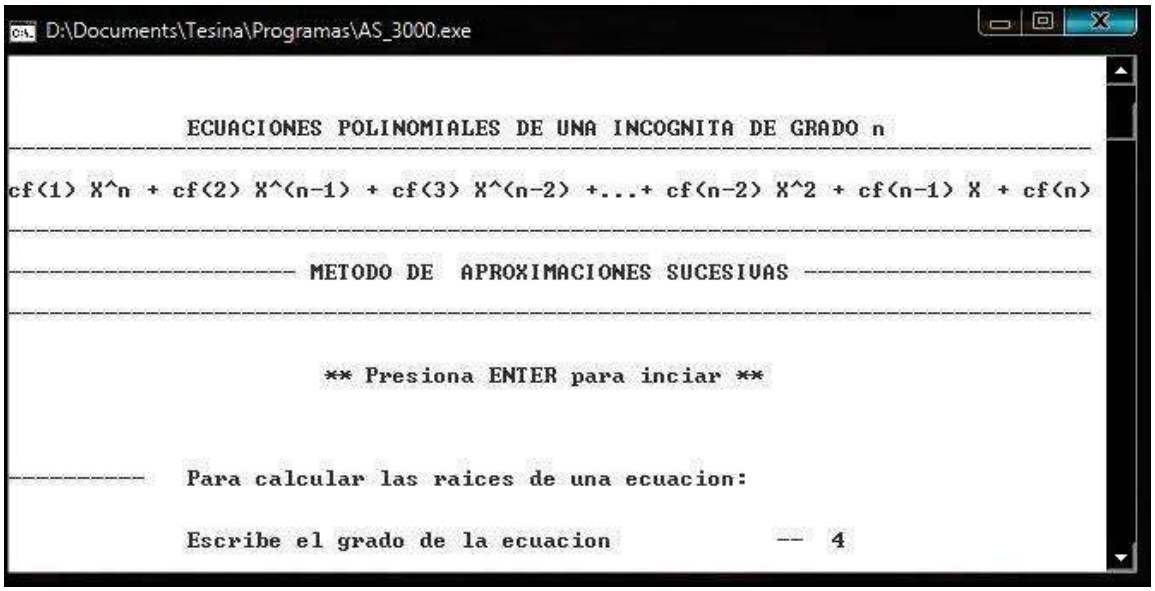
$$f(-1.291) = 0.03(-1.291)^4 - 0.09(-1.291)^3 - 0.02(-1.291)^2 + 1.15(-1.291) - 0.05 = -1.291$$

La primera raíz encontrada es **x=(-1.291)**.

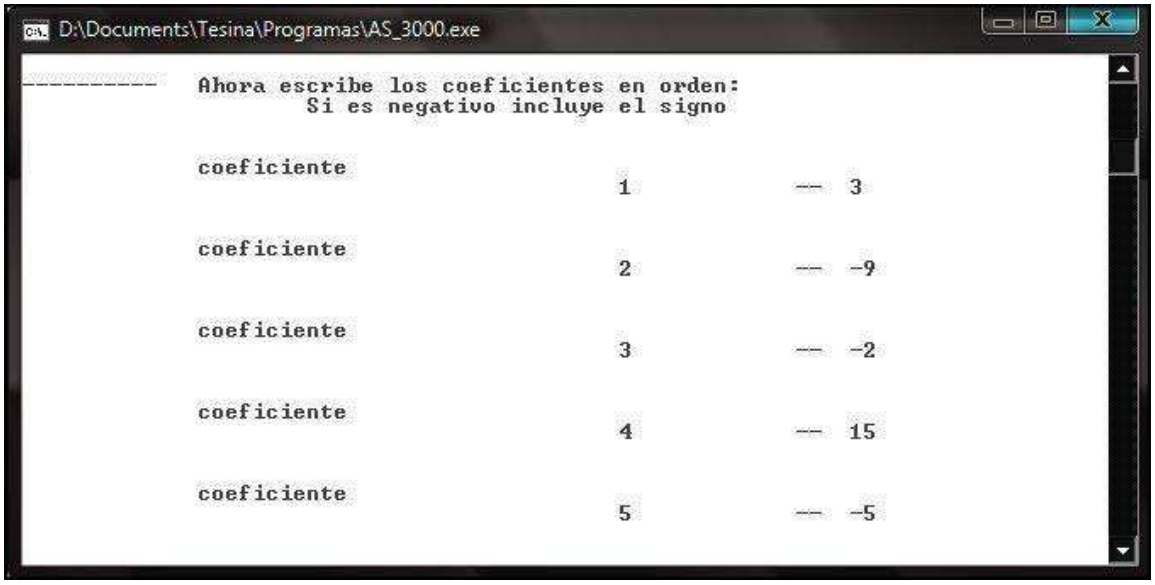
Se repite el proceso para los otros tres intervalos, y se obtienen las tres raíces faltantes:

La segunda raíz, x=1.291
La tercera raíz, no se encontró.
La cuarta raíz, no se encontró.

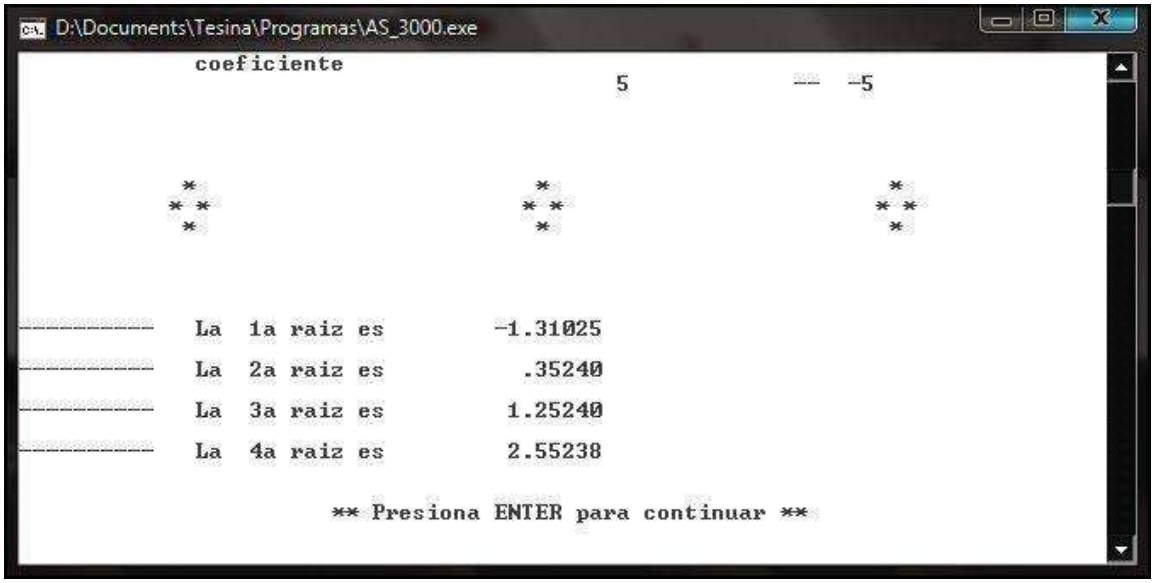
Si se utiliza el programa para solucionar la ecuación por este método, el primer dato a escribir de la ecuación es el grado,



Seguido de los coeficientes:



y el programa calcula y muestra los resultados:



Nótese que los resultados obtenidos no son iguales a los obtenidos con el método de Bolzano, esto es ocasionado por los problemas de convergencia que presenta la ecuación al aplicar este método.

Y al igual que en el programa anterior a continuación se tiene la opción de realizar otro cálculo o salir de programa según se requiera.

Por último, se resuelve la ecuación por el método de Newton-Raphson.

$$f'(x) = 12x^3 - 27x^2 - 4x + 15 = 0$$

Haciendo iteraciones:

K=1, x_k=(-2)

$$f(-2) = 3(-2)^4 - 9(-2)^3 - 2(-2)^2 + 15(-2) - 5 = 77$$

$$f'(x) = 12x^3 - 27x^2 - 4x + 15 = -181$$

$$x_{k+1} = x_k - \frac{f(x)}{f'(x)} = (-2) - \frac{77}{(-181)} = (-1.575)$$

K=2, x_k=(-1.575)

$$f(-1.575) = 3(-1.575)^4 - 9(-1.575)^3 - 2(-1.575)^2 + 15(-1.575) - 5 = 20.037$$

$$f'(-1.575) = 12(-1.575)^3 - 27(-1.575)^2 - 4(-1.575) + 15 = -92.561$$

$$x_{k+1} = x_k - \frac{f(x)}{f'(x)} = (-1.575) - \frac{20.037}{(-92.561)} = (-1.359)$$

K=3, x_k=(-1.359)

$$f(-1.359) = 3(-1.359)^4 - 9(-1.359)^3 - 2(-1.359)^2 + 15(-1.359) - 5 = 3.743$$

$$f'(-1.359) = 12(-1.359)^3 - 27(-1.359)^2 - 4(-1.359) + 15 = (-59.549)$$

$$x_{k+1} = x_k - \frac{f(x)}{f'(x)} = (-1.359) - \frac{3.743}{(-59.549)} = (-1.296)$$

K=4, x_k=(-1.296)

$$f(-1.296) = 3(-1.296)^4 - 9(-1.296)^3 - 2(-1.296)^2 + 15(-1.296) - 5 = 0.255$$

$$f'(-1.296) = 12(-1.296)^3 - 27(-1.296)^2 - 4(-1.296) + 15 = (-51.287)$$

$$x_{k+1} = x_k - \frac{f(x)}{f'(x)} = (-1.296) - \frac{0.255}{(51.287)} = (-1.291)$$

K=5, x_K=(-1.291)

$$f(-1.291) = 3(-1.291)^4 - 9(-1.291)^3 - 2(-1.291)^2 + 15(-1.291) - 5 = 0$$

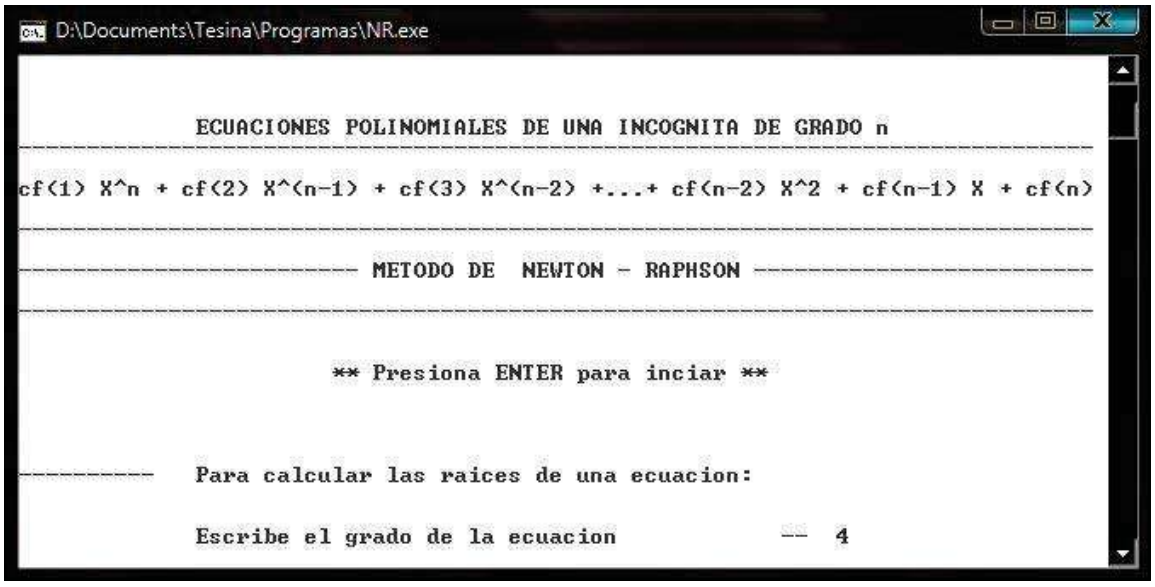
Se encontró la primera raíz, **x=(-1.921)**

Segunda raíz, **x=0.382**

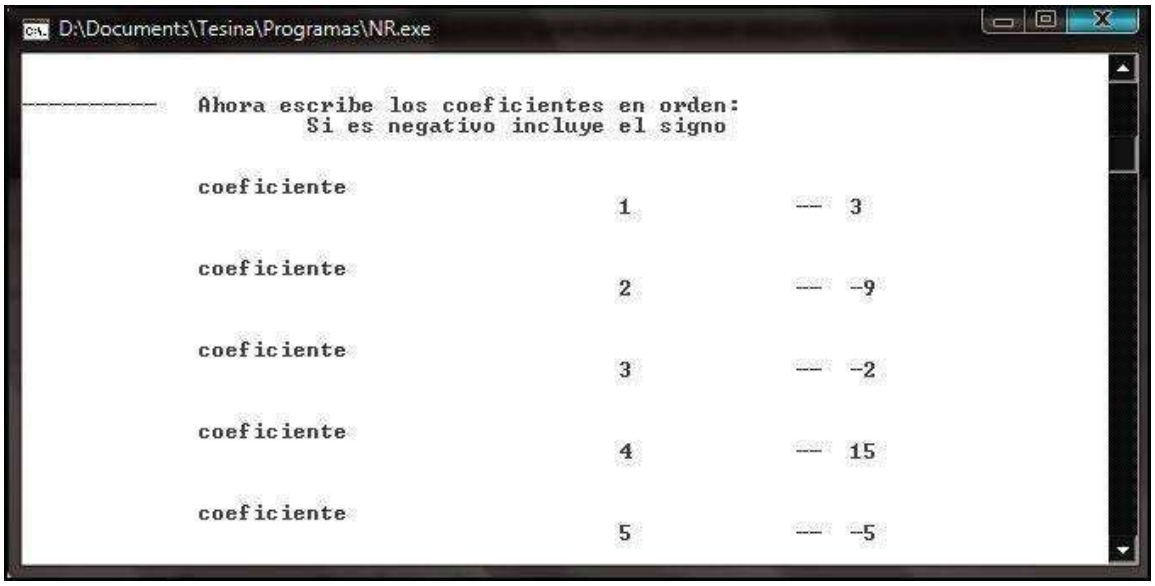
Tercera raíz, **x=1.291**

Cuarta raíz, **x=2.618**

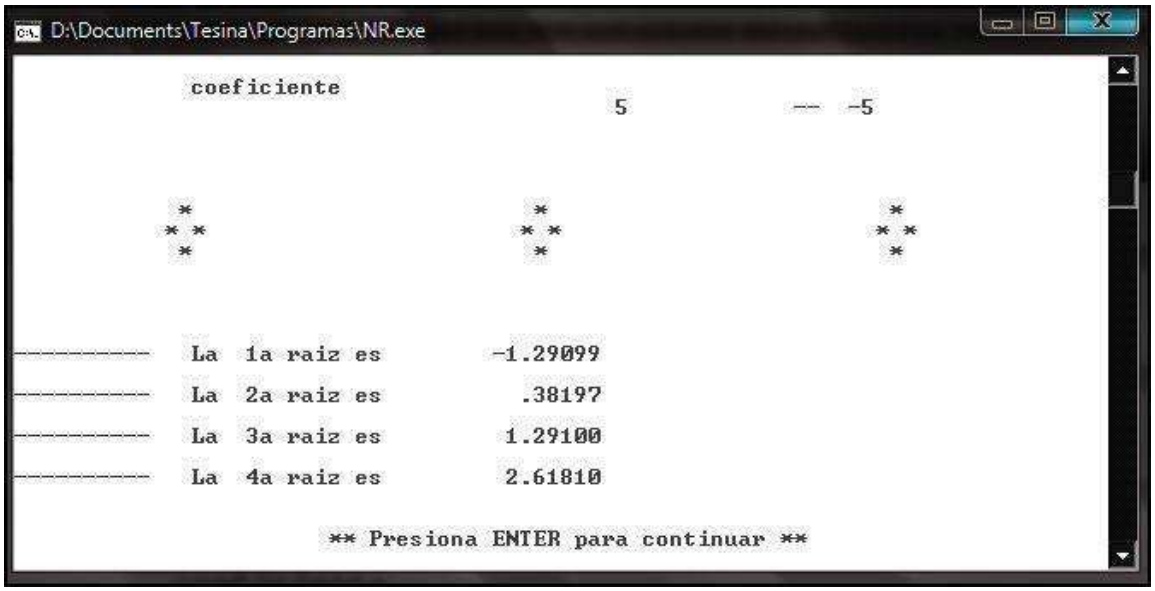
Al igual que en los dos métodos anteriores, si se usa el programa de Newton-Rapshon para la solución de esta ecuación, el primer dato a introducir es el grado de la ecuación que se pide justo después de una pantalla de bienvenida:



Y también, se escriben después los coeficientes de la ecuación:



El programa, entonces muestra los resultados:



Enseguida se tiene el menú donde las opciones “Si, hacer otro cálculo”, y “No, hacer otro cálculo” están disponibles.

CONCLUSIÓN

Los tres métodos aplicados resultan útiles en la solución de ecuaciones polinomiales, aunque si se realiza una comparativa entre ellos, se destaca que podría considerarse el método de Newton-Raphson como el más eficiente, ya que converge rápido como el de Aproximaciones Sucesivas, pero tiene una convergencia casi tan segura como el método de Bolzano.

Por otro lado tenemos que la accesibilidad con que se puede contar al usar Fortran 90, resulta de gran valor cuando la optimización de recursos es cada día más imperativa. Se puede programar casi cualquier cosa de forma rápida, sencilla, y eficaz; además de que no existe mayor dificultad en el aprendizaje del manejo del lenguaje.

Sin embargo, debe existir apertura para la introducción de un nuevo lenguaje a nuestra rutina de trabajo, ya que, si bien, las herramientas existentes parecen eficientes, la tecnología avanza a pasos agigantados y las mejoras en la misma son cada vez mayores, entonces siempre estará presente la posibilidad de mejores formas de realizar nuestras tareas.

De cualquier forma, siempre podremos contar con las matemáticas, y no importa la herramienta que usemos para su aplicación, el buen uso de todos los conocimientos de ingeniería siempre dependerá de las matemáticas que seamos capaces de desarrollar.

ANEXOS

Código fuente del programa de bisección:

```
program Bolzano_3000
implicit none
integer::g,i,k,t,u,opc
real(8)::x,fd,fh,d,h,a,b,m,fa,fb,fm,da,db,dm
real(8),dimension(1000)::coef
!-----------------------------------------
!---------- PRESENTACION ----------
print "(2/,12x,a)","-------------------------------------------------"
print "(17x,a)","-----------------------------------------"
print "(22x,a)","---------------------------------"
print "(22x,a)","C A L C U L O  D E  R A I C E S"
print "(22x,a)","---------------------------------"
print "(17x,a)","-----------------------------------------"
print "(12x,a)","-------------------------------------------------"
print "(5/,14x,a)","ECUACIONES POLINOMIALES DE UNA INCOGNITA DE GRADO n"
print*,"-------------------------------------------------------------------------------"
print*,"cf(1) X^n + cf(2) X^(n-1) + cf(3) X^(n-2) +...+ cf(n-2) X^2 + cf(n-1) X + cf(n)"
print*,"-------------------------------------------------------------------------------"
print*,"---------------------------- METODO DE BOLZANO ----------------------------"
print*,"-------------------------------------------------------------------------------"
print*
pause "                    ** Presiona ENTER para inciar **"
!-----------------------------------------
!-----------------------------------------
!-----------------------------------------
!---------- OBTENCION DE DATOS ----------
```

```
print "(2/,1x,a)","----------   Para calcular las raices de una ecuacion:"
100 print "(2/,14x,a,12x,a,\)","Escribe el grado de la ecuacion","--  "
read*,g
print "(2/,1x,a)","----------   Ahora escribe los coeficientes en orden:"
print "(22x,a)","Si es negativo incluye el signo"
do i=1,(g+1),1
        print "(2/,14x,a,1x,i100,12x,a,\)","coeficiente",i,"--  "
        read*,coef(i)
end do
print "(4/,13x,a,25x,a,25x,a)","*","*","*"
print "(12x,a,23x,a,23x,a)","* *","* *","* *"
print "(13x,a,25x,a,25x,a)","*","*","*"
print*
print*
print*
!-----------------------------------------
!---------- CALCULO DEL INTERVALO ----------
x=-1000.00005
t=0
do
do
fd=0
fh=0
do k=1,(g+1),1
d=coef(k)*(x**((g+1)-k))
h=coef(k)*((x+0.0001)**((g+1)-k))
fd=fd+d
fh=fh+h
end do
if ((fd*fh)<0)exit
x=x+0.0001
end do
!print "(a,f16.2,a,f16.2)","raiz en el intervalo",x," - ",(x+0.0001)
!print*
!-----------------------------------------
!------------ CALCULO DE LA RAIZ -------------
a=x
b=x+0.0001
```

```
do
        m=((a+b)/2)
        fa=0
        fb=0
        fm=0
        do u=1,(g+1),1
                da=coef(u)*(a**((g+1)-u))
                db=coef(u)*(b**((g+1)-u))
                dm=coef(u)*(m**((g+1)-u))
                fa=fa+da
                fb=fb+db
                fm=fm+dm
        end do
        if(fm<=0.0000001)exit
        if((fa*fm)<0)then
                a=a
                b=m
        else
                a=m
                b=b
        end if
end do
!-------------------------------------------------
print "(1/,1x,a,i3,a,f16.5)","----------   La",(t+1),"a raiz es",m
x=x+0.0001
t=t+1
if(t>=g)exit
end do
print*
print*
pause "                    ** Presiona ENTER para continuar **"
print*,"--------------------------------------------------------------------------------"
print "(1/,1x,a)","----------   Hacer otro calculo"
print "(40x,a)","1. Si"
print "(40x,a)","2. No"
print "(45x,a,\)","--  "
read*,opc
goto (100,200),opc
```

```
200 pause "                    ** Presiona ENTER para salir **"
end program
```

Código fuente para el método de aproximaciones sucesivas:

```
program Aprox_Sucesivas_2000
implicit none
integer::g,i,c,o,u,opc
real(8)::x,fx,fk,w,z,s,t,p,fs
real(8),dimension(1000)::coef
!-----------------------------------------
!-------------- PRESENTACION ---------------
print "(2/,12x,a)","-------------------------------------------------------"
print "(17x,a)","-----------------------------------------"
print "(22x,a)","-----------------------------------"
print "(22x,a)","C A L C U L O   D E   R A I C E S"
print "(22x,a)","-----------------------------------"
print "(17x,a)","-----------------------------------------"
print "(12x,a)","-------------------------------------------------------"
print  "(5/,14x,a)","ECUACIONES POLINOMIALES DE UNA INCOGNITA DE
GRADO n"
print*,"-------------------------------------------------------------------------------"
print*,"cf(1) X^n + cf(2) X^(n-1) + cf(3) X^(n-2) +...+ cf(n-2) X^2 + cf(n-1) X +
cf(n)"
print*,"-------------------------------------------------------------------------------"
print*,"--------------------- METODO DE  APROXIMACIONES SUCESIVAS ----------------
------"
print*,"-------------------------------------------------------------------------------"
print*
pause "                    ** Presiona ENTER para inciar **"
!-----------------------------------------
!------------ OBTENCION DE DATOS -----------
print "(2/,1x,a)","----------   Para calcular las raices de una ecuacion:"
100 print "(2/,14x,a,12x,a,\)","Escribe el grado de la ecuacion","--  "
read*,g
print "(2/,1x,a)","----------   Ahora escribe los coeficientes en orden:"
print "(22x,a)","Si es negativo incluye el signo"
```

```
do i=1,(g+1),1
        print "(2/,14x,a,1x,i100,12x,a,\)","coeficiente",i,"--  "
        read*,coef(i)
end do
print "(4/,13x,a,25x,a,25x,a)","*","*","*"
print "(12x,a,23x,a,23x,a)","* *","* *","* *"
print "(13x,a,25x,a,25x,a)","*","*","*"
print*
print*
print*
!-----------------------------------------
!---------- CALCULO DEL INTERVALO ----------
x=-1000000.05
c=0
do
do
        fx=0
        fk=0
        do u=1,(g+1),1
                w=coef(u)*(x**((g+1)-u))
                z=coef(u)*((x+0.1)**((g+1)-u))
                fx=fx+w
                fk=fk+z
        end do
        if ((fx*fk)<=0) exit
        x=x+0.1
end do
!print*,"el intervalo es",x," - ",(x+0.1)
!-----------------------------------------
!----------- CALCULO DE LA RAIZ --------------
s=x
do
        t=0
        do o=1,(g+1),1
                fs=coef(o)*(s**((g+1)-o))
                t=t+fs
        end do
        p=s+(0.00001*t)
```

```
        if ((p-s)<0.00001)exit
        s=p
end do
!-------------------------------------------------
print "(1/,1x,a,i3,a,f16.5)","----------   La",(c+1),"a raiz es",p
if ((c+1)>=g) exit
x=x+0.1
c=c+1
end do
print*
print*
pause "                    ** Presiona ENTER para continuar **"
print*,"--------------------------------------------------------------------------------"
print "(1/,1x,a)","----------   Hacer otro calculo"
print "(40x,a)","1. Si"
print "(40x,a)","2. No"
print "(45x,a,\)","--  "
read*,opc
goto (100,200),opc
200 pause "                    ** Presiona ENTER para salir **"
end program
```

Código fuente para el método de Newton-Raphson

```
program Newton_Raphson
implicit none
integer::h,i,l,a,c,opc
real(8)::y,p,s,b,rr,g,m,o,r,e,ll,d
real (8),dimension(1000)::hctr
!-----------------------------------------
!--------------- PRESENTACION ---------------
print "(2/,12x,a)","-------------------------------------------------"
print "(17x,a)","-----------------------------------------"
print "(22x,a)","---------------------------------"
print "(22x,a)","C A L C U L O   D E   R A I C E S"
print "(22x,a)","---------------------------------"
```

```
print "(17x,a)","-----------------------------------------"
print "(12x,a)","-----------------------------------------------------"
print  "(5/,14x,a)","ECUACIONES POLINOMIALES DE UNA INCOGNITA DE
GRADO n"
print*,"-------------------------------------------------------------------------"
print*,"cf(1) X^n + cf(2) X^(n-1) + cf(3) X^(n-2) +...+ cf(n-2) X^2 + cf(n-1) X +
cf(n)"
print*,"-------------------------------------------------------------------------"
print*,"------------------------ METODO DE  NEWTON - RAPHSON ------------------------
"
print*,"-------------------------------------------------------------------------"
print*
pause "                    ** Presiona ENTER para inciar **"
!-----------------------------------------
!-----------------------------------------
!-----------------------------------------
!------------ OBTENCION DE DATOS -----------
print "(2/,1x,a)","----------   Para calcular las raices de una ecuacion:"
100 print "(2/,14x,a,12x,a,\)","Escribe el grado de la ecuacion","--  "
read*,h
print "(2/,1x,a)","----------   Ahora escribe los coeficientes en orden:"
print "(22x,a)","Si es negativo incluye el signo"
do i=1,(h+1)
        print "(2/,14x,a,1x,i100,12x,a,\)","coeficiente",i,"--  "
        read*,hctr(i)
end do
print "(4/,13x,a,25x,a,25x,a)","*","*","*"
print "(12x,a,23x,a,23x,a)","* *","* *","* *"
print "(13x,a,25x,a,25x,a)","*","*","*"
print*
print*
print*
!-----------------------------------------
!---------- CALCULO DEL INTERVALO ----------
d=-100000.05
c=1
do
       do
```

```
        p=0
        s=0
        do l=1,(h+1),1
                y=hctr(l)*(d**((h+1)-l))
                b=hctr(l)*((d+0.1)**((h+1)-l))
                p=p+y
                s=s+b
        end do
        if((p*s)<=0)exit
        d=d+0.1
        end do
        !print*,"***"
   !print*,d
!-------------------------------------------
!----------- CALCULO DE LA RAIZ --------------
o=d
do
        ll=0
        do r=1,(h+1),1
                e=hctr(r)*(o**((h+1)-r))
                ll=ll+e
        end do
        g=0
        do a=1,(h+1),1
                m=(hctr(a)*((h+1)-a))*(o**((h+1)-(a+1)))
                g=g+m
        end do

rr=o-(ll/g)
if((rr-o)<=0.0000001) exit
o=rr
end do
!-------------------------------------------------
print "(1/,1x,a,i3,a,f16.5)","----------   La",c,"a raiz es",rr
if (c>=h)exit
d=d+0.1
c=c+1
end do
```

```
print*
print*
pause "                    ** Presiona ENTER para continuar **"
print*,"--------------------------------------------------------------------------------"
print "(1/,1x,a)","----------   Hacer otro calculo"
print "(40x,a)","1. Si"
print "(40x,a)","2. No"
print "(45x,a,\)","--  "
read*,opc
goto (100,200),opc
200 pause "                    ** Presiona ENTER para salir **"
end program
```

BIBLIOGRAFÍA

Adams Jeane C., Brainerd Walter S., Martin Jeanne T., Smith Brian T., Wagener Jerrold L.
Fortran 90, Handbook.
Editorial, McGraw-Hill

Chivers Ian D. y Sleightholme Jane
Introduction to programming with Fortran.
Editorial, Springer.

Counihan Martin
Fortand 95
Editorial, UCL press

Hahn Brian
Introduction to Fortran 90 for Scientist and Engineers.
Editorial, Department of Applied Mathematics, University of Cape Town

Hayek S. I.
Advanced mathematical methods in science and engineering.
Editorial, Marcel Dekker, Inc.

Huerta Cerezuelo A., Sarrate-Ramos J.,Rodríguez-Ferran A.
Métodos Numéricos. Introducción, aplicaciones y propagación.
Editorial, Universidad Politécnica de Catalunya.

Louden Kenneth C.
Lenguajes de programación, principios y práctica
Editorial, Thomson 2004

McCracken Daniel D., Doron William S.
Métodos Numéricos y Programación en Fortran
Editorial Limusa 1982

Rojas Rojas Rafael
Matemáticas, propedéutico de Estructuras.
Editorial, División de estudios de postgrado, facultad de Ingeniería Civil, UMSNH

Sánchez Ibarra Alma Rosa
Programación en Fortran
Editorial, División de estudios de postgrado, facultad de Ingeniería Civil, UMSNH

Páginas Web:

www.mercosur.int/MSWEB/CCCP/Comun/Documentos/actas/Actas/2006/012006/Word/Anexo%20VI%20Conceptos%20que%20se%20considera%20necesario%20agregar%20al%20glosario.doc

http://docentes.uacj.mx/gtapia/AN/Unidad2/biseccion.htm

http://www.scribd.com/doc/9926842/Metodo-de-Biseccion-Lab1

http://www.dgf.uchile.cl/~ma33a/Jaime/newton_raphson_method.htm

http://stark.udg.es/~emili/docent/qtc/pdf/01_raices.pdf

http://www.sc.ehu.es/sbweb/fisica/cursoJava/numerico/raices/aproximaciones/aproximaciones.htm

http://www.dgf.uchile.cl/~ma33a/Jaime/newton_raphson_method.htm

http://docentes.uacj.mx/gtapia/AN/Unidad2/Newton.htm

http://www.desarrolloweb.com/articulos/2477.php

http://www.lenguajes-de-programacion.com/programacion-orientada-a-objetos.shtml

http://pviojo.net/posts/programacion-declarativa/

http://fcqi.tij.uabc.mx/docentes/mgarduno/Program1/Unidad1/u1_1.htm