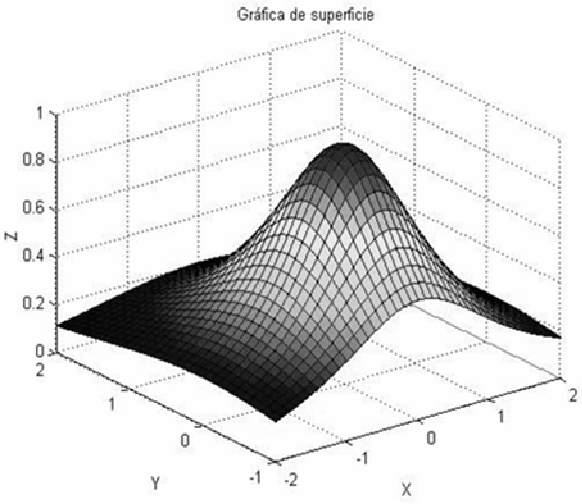
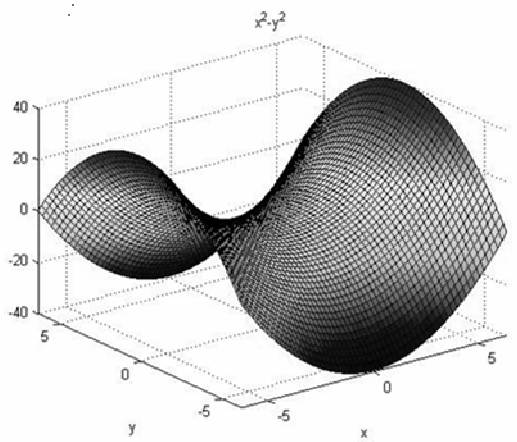
Algebra
Y MATLAB

Mario E. Matiauda

Año 2010

$$\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n^2}, \quad \sum_{n=1}^{\infty} \frac{\operatorname{sen}(4/(n+2))}{n+5}$$





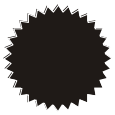
UNIVERSIDAD NACIONAL DE MISIONES

Algebra

Y MATLAB

Mario E. Matiauda

Año 2010



EDITORIAL UNIVERSITARIA DE MISIONES

San Luis 1870

Posadas - Misiones – Tel-Fax: (03752) 428601

Correos electrónicos:

edunam-admini@arnetbiz.com.ar

edunam-direccion@arnetbiz.com.ar

edunam-produccion@arnetbiz.com.ar

edunam-ventas@arnetbiz.com.ar

Coordinación de la edición: Claudio Oscar Zalazar

ISBN 978-950-579-151-4

Impreso en Argentina

Matiauda, Mario Eugenio
Algebra y Matlab. - 1a ed. - Posadas : EDUNaM - Editorial Universitaria de la Universidad Nacional de Misiones, 2010.
88 p.; 30x21 cm.
ISBN 978-950-579-151-4
1. Algebra. 2. Enseñanza Universitaria . I. Título
CDD 511.071 1

Fecha de catalogación: 11/03/2010.

EL AUTOR

MATIAUDA, Mario Eugenio
Título grado: Ingeniero Químico. FCEQ y N-UNaM
Titulo posgrado:
-Especialista en Celulosa y Papel (FCEQyN)-UNaM.
-Especialista en Vinculación Tecnológica (UNL).
-Magister en Ciencias de la Madera, Celulosa y Papel (FCEQyN)-UNaM.
Como docente se inicio en el año 1986 pasando por diversos cargos de la carrera docente.
Actualmente es Prof. Adjunto exclusiva (FCEQyN), a cargo de las asignaturas Matemática 1, Matemática II, Matemática Aplicada (Analista en Sistemas-FCEQyN-UNaM), y Matemática I, II, III, IV e Investigación Operativa (Licenciatura en sistemas, FCEQyN-UNaM).
Investigación, en la actualidad: integrante del proyecto "Gasificación de aserrín para producción de gas enriquecido en hidrógeno", FCEQyN.
Desde el año 1986 trabaja en investigación siempre relacionado a Ciencia y Tecnología de la madera: Secado de madera, Optimización energética Secadero de madera, Tensiones y Deformaciones en el secado de madera, Combustión.
Publicaciones realizadas (independiente):
Cálculo diferencial e *integral,* 120 páginas, año 1999, expte 66754 de solicitud de depósito en custodia de obra inédita, Dirección Nacional del derecho de autor, 23/06/00.Único titular.
Acerca de optimización, 106 páginas, -año 1999, expte 66753 de solicitud en depósito en custodia de obra inédita, Dirección Nacional del derecho de autor, 23/06/00. Único titular.
Algebraicas 1.0, 109 páginas, af'ío2000, expte 82747 de de solicitud de depósito en custodia de obra inédita, Dirección Nacional del derecho de autor, 05/09/00. Único titular.
Programación No Lineal, l~ Parte, año 2002, 30 páginas, producción independiente.
Programación No Lineal. 2~ Parte. año 2002, 30 páginas, producción independiente.
*Algebra lineal (Nociones teóricas)' elementales y ejercitación). Año 2005.*73 páginas, producción independiente.
La solución Numérica Elemental-Editorial universitaria de Misiones-ISBN 978-950-579-126-2, 2009.

CONTENIDOS

PRÓLOGO

Muchos temas pueden tratarse más eficientemente, que usando métodos de enseñanza tradicionales. Cantidad de problemas que requieren cálculos extensos y laboriosos pueden resolverse en forma muy simple. En lugar de enseñar y aprender habilidades de cálculo, los profesores y los estudiantes, pueden centrarse en los aspectos más esenciales de las técnicas de resolución de problemas. Esto facilita la comprensión y el desarrollo de los conceptos matemáticos y sus aplicaciones. Dejando claro que no es principio excluyente, se aborda el contexto apuntando al entorno de *MATLAB*.

MATLAB es una herramienta usada para hacer y aplicar las matemáticas, la cual permite documentar el trabajo, aprender y enseñar matemáticas.

Para el docente y para el estudiante, *MATLAB* es una herramienta importante que permite apoyar el aprendizaje y la enseñanza de las matemáticas. Gracias a sus capacidades gráficas, numéricas y algebraicas, aporta nuevos enfoques en la enseñanza, en el aprendizaje y en la comprensión de las matemáticas.

Los estudiantes pueden concentrarse en el significado de los conceptos matemáticos, dejando a *MATLAB* los procedimientos mecánicos y los algoritmos de la resolución de problemas.

El objetivo de este impreso es incursionar en esta dirección en forma elemental, a través del uso de *MATLAB*, que no significa de ningún modo establecer como sinónimo único de herramienta de hacer y aplicar matemática, en este caso a áreas introductoria de álgebra y funciones.

CAPITULO 1: SOBRE MATLAB

El uso de nuevas tecnologías integradas a la enseñanza de las matemáticas, permite los siguientes logros:

- El desarrollo de habilidades creativas en todos los cursos, mediante el acceso a los computadores y a software específicos como el *MATLAB*.
- Mejoras en actitud hacia las matemáticas por parte de los estudiantes.
- El mejoramiento académico, a partir del desarrollo mental de los estudiantes, lo cual les permite alcanzar los propósitos académicos.

En este impreso se pretende un acercamiento a partir de la integración del software *MATLAB* y el microcurrículo de Álgebra, esbozos de lógica y trigonometría, sin que se consideren problemas aislados, sino formando parte de una estrategia de análisis, donde la herramienta cumpla un papel importante, que permita hacer comparaciones.

Lógicamente, es necesario hacer una aplicación reflexiva del instrumento y analizar las posibles formas de uso, en donde la reflexión por parte del estudiante sea el corazón de la actividad.

Asimismo debe incluirse la autoevaluación; siendo esencial que el alumno conozca lo que ha aprendido e identifique cómo debe recuperar los objetivos didácticos no alcanzados.

1.1. QUÉ ES MATLAB

MATLAB es un lenguaje de programación específicamente diseñado para el Cálculo Numérico, representando en la práctica un conjunto de herramientas que Mathworks distribuye para programar en *MATLAB*.

A diferencia de la mayoría de lenguajes de programación no existe ningún estándar que lo rija, técnicamente el lenguaje *MATLAB* es tal como se distribuye en su última versión.

Pero lo realmente valioso de *MATLAB* no son sus capacidades como lenguaje sino las siguientes:

- Existe un uso generalizado de *MATLAB* en Ingeniería, es una herramienta de gran popularidad y es útil para una carrera profesional. Esto lo ha convertido en un estándar de-facto para la escritura de pequeños programas de simulación.
- *MATLAB* cuenta con una extensa biblioteca de funciones que cubren casi todas las disciplinas de la Ciencia y la Ingeniería extensamente documentada y de fácil uso.

El producto que recibe el nombre de *MATLAB* consta esencialmente de dos partes: un intérprete y un editor.

1.2. EL INTÉRPRETE

Antes de hablar del intérprete del lenguaje *MATLAB* es imprescindible entender algunas de las características de los lenguajes de programación.

Los lenguajes de programación, como los lenguajes naturales escritos, no son más que una serie de normas para transmitir conceptos. Mientras el lenguaje escrito sirve para que los seres humanos se comuniquen entre ellos los lenguajes de programación se crearon para comunicarse con los ordenadores mediante una serie finita de claves.

Los lenguajes de programación también tienen gramática y léxico pero son mucho más simples que, por ejemplo, los de la lengua castellana. Los seres humanos estamos educados para convertir palabras y frases en sonidos. Hay que dotar a los ordenadores de un método para convertir el código implementado en un lenguaje de programación en órdenes que sea capaz de cumplir. Hay casi una infinidad de maneras de lograr este objetivo.

Cuando apareció el ordenador programable la única manera de comunicarse con él era describir sin ambigüedad qué sucedía con cada posición de memoria. Este código de bajo nivel, llamado comúnmente *ensamblador*, es traducido a lenguaje máquina que ya un ordenador es capaz de entender. Aunque hoy este método de programación pueda parecer inverosímil es la mejor manera de mover máquinas lentas y con poca memoria como las de entonces.

El paso siguiente llegó con la aparición de los compiladores. A medida que los ordenadores se hacían más potentes escribir los programas en ensamblador empezó a hacerse una tarea muy laboriosa. El número de direcciones de memoria crecía exponencialmente y las arquitecturas, aunque seguían el modelo de Von Neumann, se hacían más complejas. El siguiente paso fue utilizar el mismo ordenador para traducir desde un lenguaje más humano, de alto nivel, a ensamblador. El ensamblador pasó de ser un lenguaje de uso a un léxico intermedio. El programa que convierte este código de alto nivel se llama *compilador*.

Este planteamiento tiene una ventaja adicional. El código ensamblador no es el mismo para todas las arquitecturas. Un programa compilado para x86 no puede ejecutarse en SPARC o POWER pero el código es el mismo

El paso siguiente es poder utilizar un ensamblador independiente de cada arquitectura mediante un traductor de código propio a código máquina. Esta aplicación se llama *máquina virtual*. Una máquina virtual es tan hábil como se desee (mucho más hábil que un procesador) y realizará tareas como la declaración de variables, la liberación de memoria o la gestión del flujo de ejecución. El conjunto compilador y máquina virtual se denomina intérprete y los lenguajes que soportan este funcionamiento se llaman *lenguajes interpretados*. Que el código sea ejecutado por un programa y no por el propio ordenador es mucho más lento, por este motivo las máquinas virtuales no se popularizaron hasta finales de los noventa.

El paso siguiente es hacer desaparecer incluso este ensamblador intermedio y con él el compilador. Ya no existe un compilador y una máquina virtual sino que sólo un programa, el intérprete, realiza todo el trabajo. Este último planteamiento no es necesariamente superior en eficacia o rendimiento a una máquina virtual, simplemente es más fácil de diseñar e implementar. *MATLAB* pertenece a este último grupo.

1.2.1. Lenguajes estáticos y dinámicos

En muchos lenguajes de programación como C o *Fortran* es imprescindible declarar cada variable. La definición estricta de declaración es la de identificar un determinado espacio en la memoria del ordenador con un nombre. Volviendo otra vez a un C que cualquiera pueda entender la declaración *int a*; significa que un espacio en la memoria física lo suficientemente grande como para almacenar un entero va a recibir el nombre de a. Estos lenguajes, los que asocian variables a memoria, se llaman *estáticos.*

La llegada de los lenguajes interpretados permitió manejar la memoria de una manera mucho más versátil. *Java*, que aunque es interpretado es también estático, incluye un recolector de basura que descarga al programador de la tarea de limpiar la memoria. Pero la mayoría de los lenguajes interpretados modernos como *Python* o *Ruby* son además *dinámicos*. En un lenguaje dinámico no existen declaraciones porque el concepto de variable es distinto, *ya no es el nombre que se asocia a un espacio en la memoria, es el nombre de un valor*. De esta manera la variable tiene un sentido mucho más natural, más matemático. *MATLAB* es un lenguaje dinámico aunque no puede considerarse moderno.

Desde el punto de vista del intérprete cualquier variable o estructuras de variables son mutables en tiempo de ejecución complicando significativamente el manejo de memoria.

Programar con un *lenguaje dinámico es completamente distinto hacerlo con uno estático*. La mayor versatilidad suele venir acompañada de mayor coste computacional o de nuevos errores de programación. No debemos perder nuca de vista que la programación es la manipulación de datos almacenados en la memoria de un ordenador y con un lenguaje dinámico estamos más lejos de los mismos.

MATLAB es sólo un lenguaje, nada liga un lenguaje a un intérprete en concreto. El lenguaje de programación C++ cuenta con casi medio centenar de compiladores distintos al igual que Java puede ejecutarse en más de una máquina virtual, incluso Microsoft creó una propia. Nada impide programar un intérprete de lenguaje *MATLAB* distinto del distribuido por *Mathworks*.

Octave, como *Freemat* o *Scilab*, nació como un producto para cálculo numérico con un lenguaje de programación propio pero con un funcionamiento idéntico al de *MATLAB*. Los objetivos del proyecto fueron cambiando progresivamente hasta convertirse en un intérprete capaz de evaluar casi la totalidad del código escrito en *MATLAB*. Incluso soporta ciertas extensiones muy cómodas que podrían perfectamente ser incorporadas en el intérprete *oficial*.

MATLAB, como su clon *Octave*, soporta sin grandes diferencias los tres sistemas operativos más comunes: *Windows*, *Linux* y *MacOS*.

1.3. EL EDITOR

Un editor es un programa diseñado especialmente para escribir código. Escribir código fuente no se parece en nada a redactar un informe o escribir una novela, no parece lógico utilizar la misma herramienta para ambas tareas. Un buen editor nos

ayudará a sangrar bloques, coloreará las palabras clave, nos avisará si cometemos algún error evidente.

Una parte muy importante del entorno de desarrollo *MATLAB* es el editor diseñado específicamente para el lenguaje. Esto no significa que sea la única posibilidad, casi la totalidad de editores de texto disponen de macros o modos dedicados a la programación en *MATLAB*.

Integrated Development Environment(IDE):Cuando en una misma aplicación gráfica se incluyen intérprete, consola interactiva, editor, cronología de comandos, visor de documentación, gestor de archivos y debugger se hace con la intención que no se tenga que utilizar ninguna aplicación externa. El producto *MATLAB* es, si se busca una definición estricta, un IDE para la programación en lenguaje *MATLAB*.

CAPITULO 2: PROPOSICIONES - FUNCIONES

2.1. LA INTERFAZ DE MATLAB

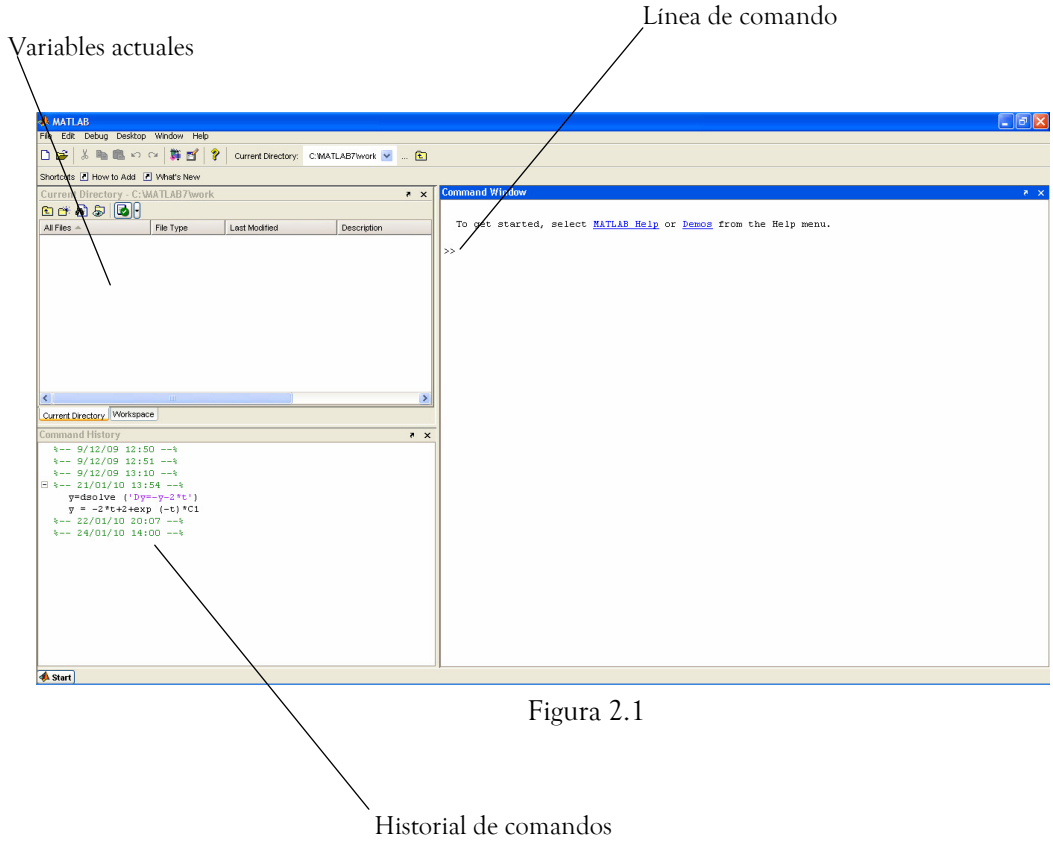


Figura 2.1

Al abrir el programa, se puede encontrar una interfaz como la de la figura, donde aparece el cuadro de variables actuales (*current directory*), el historial de comandos (*command history*) y la línea de comandos (*command window*), que será la configuración por default (de la pestaña desktop, se elige desktop *layout*, apareciendo las opciones para modificarla, si se las quiere mantener conviene *guardarla*, con *save*).

Básicamente, existen dos formas de utilizar la ayuda de *MATLAB*: a través de la ayuda en línea; o bien, a través del navegador de ayuda.

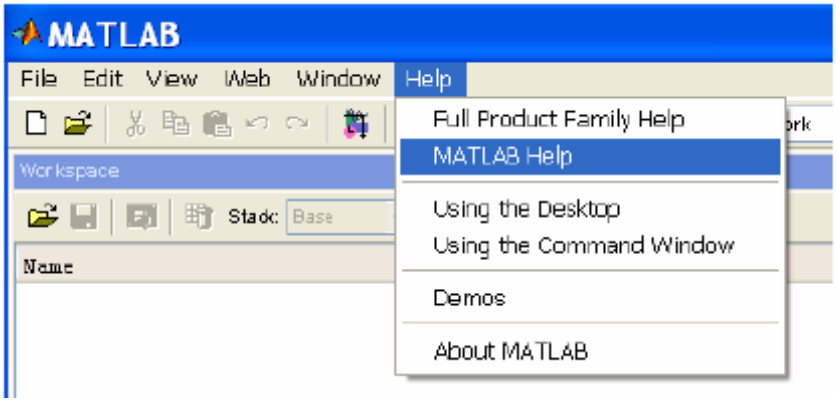


Figura 2.2

Al abrir el programa, se puede encontrar una interfaz como la de la figura, donde aparece el cuadro de variables actuales (*current directory*), el historial de comandos (*command history*) y la línea de comandos (*command window*), que será la configuración por default (de la pestaña desktop, se elige desktop *layout*, apareciendo las opciones para modificarla, si se las quiere mantener conviene *guardarla*, con *save*).

La ventana de comandos constituye el principal mecanismo para comunicarse con *MATLAB*. Las funciones introducidas (o "las entradas") se ejecutan pulsando la tecla Enter. Al escribir los nombres de las funciones o de los comandos, es importante recordar que *MATLAB* distingue entre mayúsculas y minúsculas (habitualmente, las funciones se escriben en minúsculas).

Asimismo, tras seleccionar una zona de esta ventana, el botón derecho del ratón despliega un menú emergente que permite, entre otras opciones, evaluar dicha selección, e igualmente, abrirla con el *Editor/Debugger* como un *M-fichero*.

Seleccionar *File* → *Preferences...* permite especificar el formato numérico a emplear y otras opciones de presentación en pantalla. Asimismo, es posible seleccionar el tipo y el color de las fuentes de texto.

Básicamente, existen dos formas de utilizar la ayuda de *MATLAB*: a través de la ayuda en línea; o bien, a través del navegador de ayuda.

Para acceder a la ayuda en línea basta con teclear en la línea de comandos:

>> *Help funcion*

donde "*funcion*" sería el nombre de la función sobre la que necesitamos la ayuda.

Por otro lado, para acceder a la ayuda a través del navegador, es necesario seleccionar la opción "*Matlabhelp*" (Figura 2.3). Este segundo modo de ayuda resulta bastante más potente y eficaz que la primera añadiendo en muchos casos ejemplos de utilización.

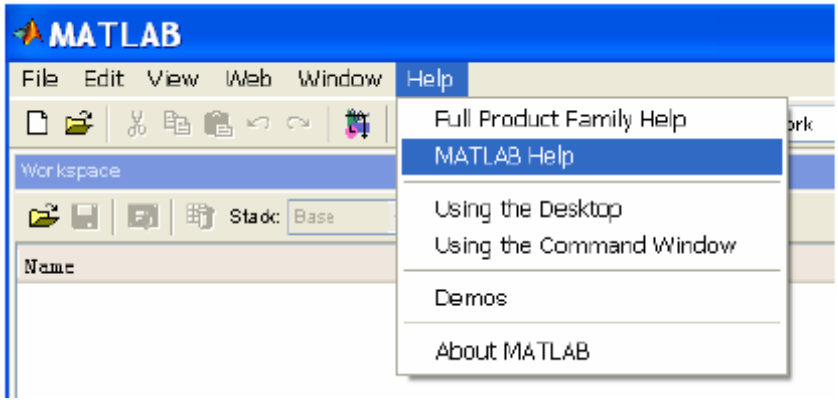


Figura 2.3

Se enumeran a continuación los principales comandos que se emplean en la ventana de comandos.

» *Load* Lee todas o algunas de las variables de un fichero.

» *Open* Abre, entre otros, los ficheros *.mat*, M-ficheros o ficheros *.fig* de gráficos.

Un fichero también puede abrirse seleccionando *File* → *Open...*

De modo equivalente, las variables pueden importarse eligiendo *File* → *Import Data...*

» *Clear* Elimina algunas o todas las variables del espacio de trabajo.

Igualmente, *Edit* → *Clear Workspace* elimina todas las variables del espacio de trabajo.

» *Clc* Borra la ventana de comandos (no elimina las variables).

Este comando equivale a seleccionar *Edit* → *Clear Workspace*.

» *Format* modo Determina el formato de salida en la ventana de comandos. Entre los distintos "*modos*", pueden destacarse: *short* (muestra hasta 5 dígitos) *long* (muestra hasta 15 dígitos) y *rat* (formato racional).

» *Cd* Permite conocer y cambiar el directorio actual.

» *Cd...* Disminuye un nivel en el árbol de carpetas.

El directorio puede ser igualmente modificado en la ventana de directorio actual.

» *Who* Muestra un listado con las variables del espacio de trabajo.

Estas variables aparecen, igualmente, en la ventana de espacio de trabajo.

» *Dir* Muestra un listado con los archivos del directorio actual.

Esta información también es asequible a través de la ventana de directorio actual.

» *Edit* M-fichero Abre una ventana de edición con un *M-fichero*.

Si no se especifica un M-fichero la ventana de edición se abre en blanco. Igualmente puede seleccionarse.

File → *New* → *M-file*, o hacer clic en el botón de la barra de herramientas.

» *Save* Guarda todas o algunas de las variables del espacio de trabajo.

Análogamente, puede seleccionarse *File* → *Save Workspace As...*

» *Exit/Quit* Cierra el programa *MATLAB*.

Igualmente es posible cerrar el programa mediante *File* → *Exit MATLAB*.

Algunas teclas o combinaciones de teclas resultan especialmente interesantes en la ventana de comandos:

- Las teclas ↑ y ↓ permiten recuperar comandos escritos con anterioridad.
- La tecla *Esc* elimina todo el texto escrito en una línea.
- La combinación de teclas *Control* + *c* aborta la ejecución de una sentencia

Editor/Debugger... de M-ficheros.

La interacción con *MATLAB* puede llevarse a cabo directamente a través de la ventana de comandos.

Alternativamente, es posible escribir, en primer lugar, todo un conjunto de funciones o entradas en un M-fichero y ejecutarlas posteriormente. La creación de este tipo de M-ficheros se lleva a cabo en el *Editor/Debugger...* que se muestra en la Figura 2.4.

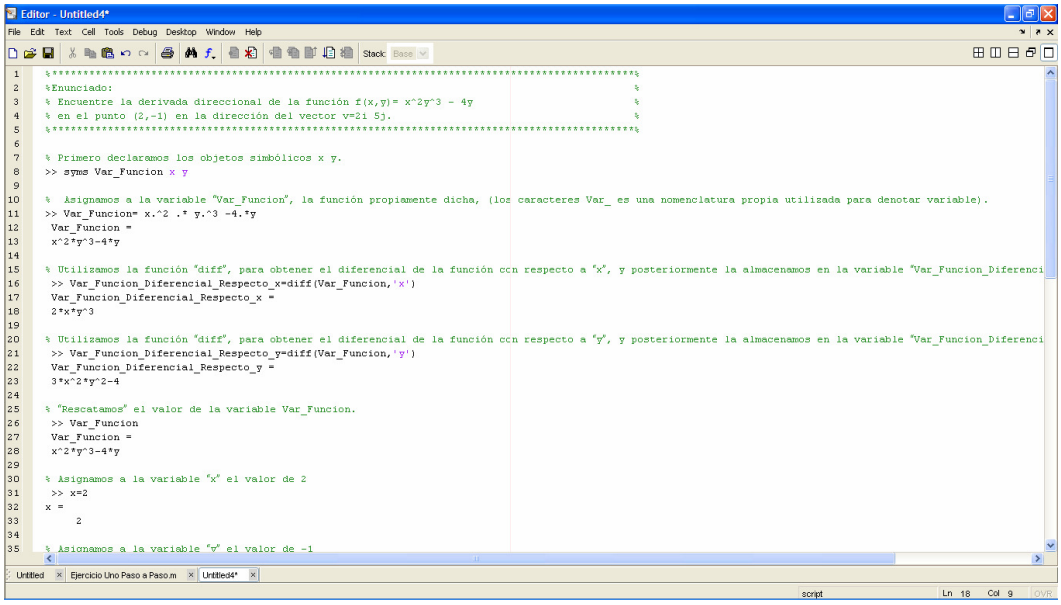


Figura 2.4

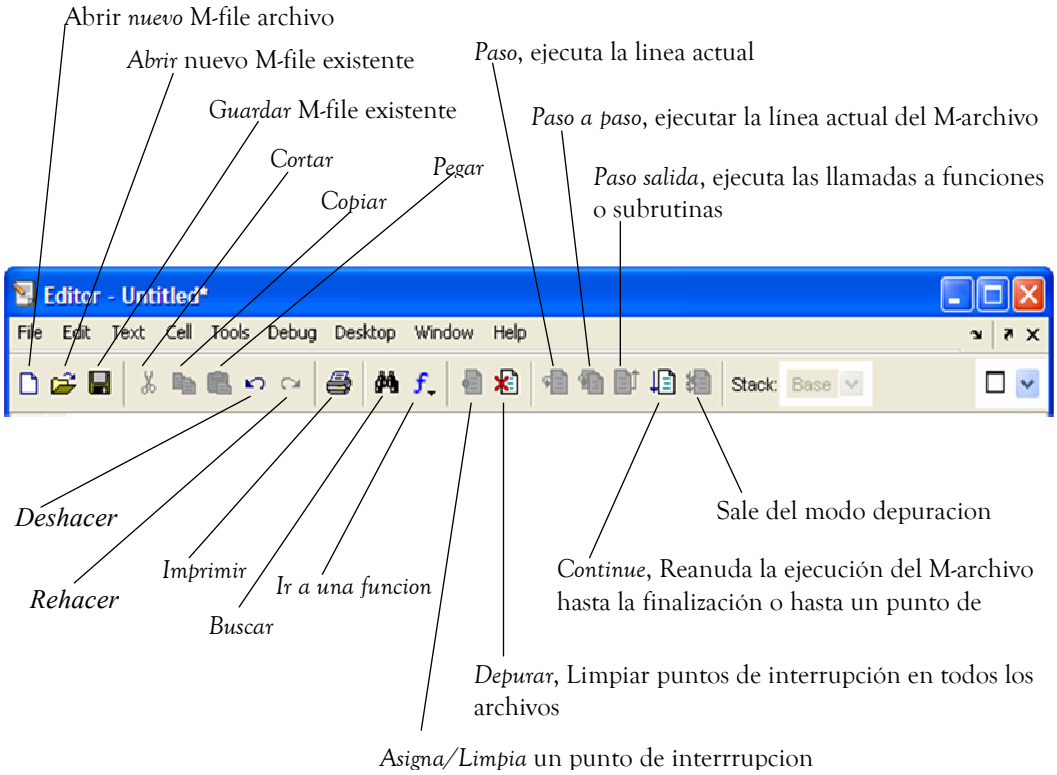


Figura 2.5

Este editor puede abrirse haciendo clic en el botón de la barra de herramientas, escribiendo *edit* en la ventana de comandos, o bien seleccionando *File* → *New* → *M-file*.

Un M-fichero, ya existente, se abre utilizando *File* → *Open...* Igualmente, es posible seleccionar bien un M-fichero, o bien una o varias sentencias, y editarlos empleando el botón derecho del ratón.

En este tipo de ficheros, resulta útil introducir comentarios aclaratorios. Para que *MATLAB* pueda distinguir entre comentarios y entradas, los primeros irán precedidos de un %.

También es importante resaltar que cuando una expresión termina en punto y coma (;) se calcula su resultado pero no se muestra en pantalla. Al no mostrar los resultados intermedios que no sean de interés, se consigue agilizar el cálculo.

2.2. OPERADORES LOGICOS

En *MATLAB*, hay cuatro operadores lógicos:

Operador booleano	**significado:**
&	Lógico y(AND)
|	lógico o(OR)
~	Lógico no(NOT,complemento)
xor	Excluyente o(OR)

Estos operadores producen vectores o matrices del mismo tamaño que los operandos y 1 cuando la condición es verdadera, y 0 cuando la condición es falsa

Sean los arreglos ***x*** = [0 7 3 5], ***y*** = [2 8 7 0], las posibles operaciones son:

Operación:	**Resultado:**
n = x & y	*n* = [0 1 1 0]
m = ~(y | x)	*m* = [0 0 0 0]
p = xor(x, y)	*p* = [1 0 0 1]

Ya que la salida de las operaciones de lógica o booleana es un vector o matriz con valores sólo 0 o 1, la salida se puede utilizar como índice de una matriz para extraer los elementos apropiados. Por ejemplo, para ver los elementos de *x* que cumplen las dos condiciones (*x* <*y*) y (*x* <4), puede escribir *x* ((*x* <*y*) y (*x* <4)).

Operación:	**Resultado:**
x<*y*	*ans* = [1 1 1 0]
x<*4*	*ans* = [1 0 0 0]
q = x((x<y)&(x<4))	*q* = [0 3]

hay varias útiles funciones integradoras lógicas, tales como

any	Verdadera si ningún elemento de un vector es verdadero
all	Verdadera si todos los elementos de un vector son verdaderos
exist	Verdaderas si los argumentos existen
isempty	Verdadera para una matriz vacía
isinf	Verdadera para todos los infinitos elementos de una matriz
isnan	Verdadera para todos los elementos de una matriz no números
find	Halla los índices de elementos no ceros de una matriz

Operadores de relación: son seis en *MATLAB*

Operador de relación:	**significado:**
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que
==	Igual (posibilidad, no es asignación)
~=	No igual

Estas operaciones resultan en un vector de la matriz del mismo tamaño que los operandos, con 1 cuando la relación es verdadera, y 0 cuando es falso
Para ***x*** = [0 7 3 5], ***y*** = [2 8 7 0], las posibles operaciones de relación son:

Operación:	**Resultado:**
k = x<y	*k* = [1 1 1 0]
k = x <= y	*k* = [1 1 1 0]
k = x == y	*k* = [0 0 0 0]

Aunque estas operaciones se suelen utilizar en sentencias condicionales, como si-más(if-else) que se ramifican a los diferentes casos, pueden ser utilizados para hacer una manipulación de matriz muy compleja. Por ejemplo, *x = y* (*y*> 0,45) encuentra

todos los elementos del vector y tal que *yi*> 0,45 y las almacena en el vector X. Estas operaciones se pueden también combinar con operadores boléanos.

2.3. EL ÁLGEBRA DE BOOLE ES EL ÁLGEBRA DE LAS PROPOSICIONES.

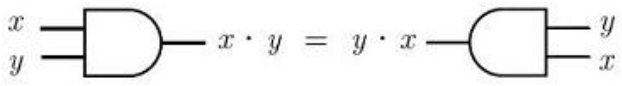
Las proposiciones se denotan por letras, como *A*, *B*, X o *Y*, etc. En los siguientes axiomas y teoremas (leyes de álgebra de Boole), los signos '+' o

'*V*' representa un operador lógico *OR* (o conjunción), el'. 'o '^' representan un operador lógico *AND* (o disjunción), y '¬' o '˜' representan un NO lógico (o la negación).

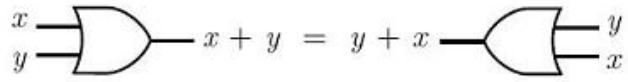
Cada proposición tiene dos valores posibles: 1 (o T), cuando la proposición es verdadera y 0 (o F), cuando la proposición es falsa.
La negación de A se escribe **¬A** (o **˜A)** , se lee 'no A'. Si A es verdadera entonces ¬A es falsa, similarmente si A es falsa entonces ¬A es verdadera.

Descripción	**OR forma**	**AND forma**	**Otra manera de expresarla:**
Axioma	$x+0 = x$	$x.1 = x$	$A \ V \ F = A$ $A \wedge T = A$
Conmutativa	$x+y = y+x$	$x.y = y.x$	$A \ V \ B = B \ V \ A$ $A \wedge B = B \wedge A$
Distributiva	$x.(y+z) = (x.y)+(x.z)$	$x+y.z = (x+y).(x+z)$	$A \wedge (B \ V \ C) = (A \wedge B) \ V \ (A \wedge C)$ $A \ V \ B \wedge C = (A \ V \ B) \wedge (A \ V \ C)$
Axioma	$x+\neg x = 1$	$x.\neg x = 0$	$A \ V \ \neg A = T$ $A \wedge \neg A = F$
Teorema	$x+x = x$	$x.x = x$	$A \ V \ A = A$ $A \wedge A = A$
Teorema	$x+1 = 1$	$x.0 = 0$	$A \ V \ T = T$ $A \wedge F = F$
Teorema	$\neg\neg x = x$		$\neg(\neg A) = A$
Asociatividad	$x+(y+z) = (x+y)+z$	$x.(y.z) = (x.y).z$	$A \ V \ (B \ V \ C) = (A \ V \ B) \ V \ C$ $A \wedge (B \wedge C) = (A \wedge B) \wedge C$
Absorción	$x+x.y = x$	$x.(x+y) = x$	$A \ V \ A \wedge B = A$ $A \wedge (A \ V \ B) = A$
Leyes deMorgan	$x+y = \neg(\neg x.\neg y)$	$x.y = \neg(\neg x+\neg y)$	$A \ V \ B = \neg(\neg A \wedge \neg B)$ $A \wedge B = \neg(\neg A \ V \ \neg B)$

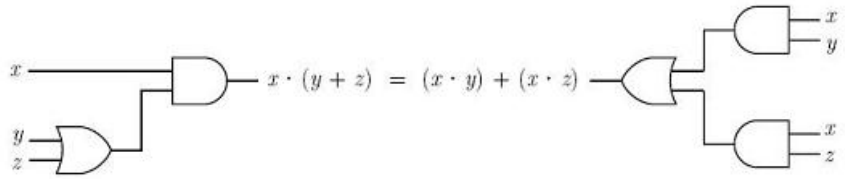
Con circuitos lógicos, la conmutativa para *And* lógico será



La conmutativa para un *or* lógico:

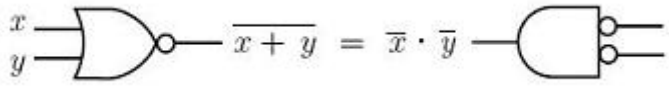


La distributiva

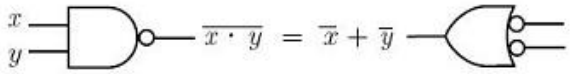


Leyes de Morgan

Estas leyes nos enseñan a intercambiar con los operadores *AND* y *OR* lógico. Uso de puertas (de uso común en Electrónica Digital), que puede expresarse de dos formas: la *ON*



Y la AN



Veamos un script *morgan .m*

```
% sean x e y los vectores columnas
x = [0 0 1 1]'; y = [0 1 0 1]';
x_or_y = x|y
DeMorg1 = not(not(x)& not(y))
% se puede demostrar la forma AND de la ley con estas líneas
x_and_y = x&y
DeMorg2 = not(not(x) | not(y))
Al correr en la línea de comandos
>> morgan

x_or_y =
    0
    1
    1
    1

DeMorg1 =
    0
    1
    1
    1
```

x_and_y =

0
0
0
1

DeMorg2 =

0
0
0
1

Que demuestra la ley de Morgan. Veamos los operadores

- Lógico *AND*

A	*B*	*A & B*
0	0	0
0	1	0
1	0	0
1	1	1

A & *B* realiza un *AND* lógico de las matrices *A* y *B* y devuelve una matriz que contiene los elementos establecidos ya sea lógico 1 (*TRUE*) o lógico 0 (*FALSO*). Un elemento de la matriz de salida es 1 si ambas matrices de entrada contiene un elemento distinto de cero en ese punto misma matriz. De lo contrario, ese elemento se establece en 0. *A* y *B* deben tener las mismas dimensiones que el que no un escalar.
ejemplo. Dada *A*, *B*

```
>> A=[0 0 1 1;1 1 0 0;0 0 0 0;1 1 1 1];
>> B=[0 0 0 0;1 1 1 1;0 1 0 1;1 0 1 0];
```

Entonces. La operación ***AND*** entre *A* y *B* es:

```
>> A & B
ans =
    0    0    0    0
    1    1    0    0
    0    0    0    0
```

Ejemplo si los vectores *x* e *y* son:

```
>>x =
    0    1    2    3    0
>>y =
    1    2    3    0    0
```

la operación AND entre *x* e *y* es:

>> *x & y*

ans =

0 1 1 0 0

*Lógico **OR**

A	*B*	*A* | *B*
0	0	0
0	1	1
1	0	1
1	1	1

A | *B* realiza un *OR* lógico de las matrices *A* y *B* y devuelve una matriz que contiene los elementos establecidos ya sea lógico 1 (*TRUE*) o lógico 0 (*FALSO*). Un elemento de la matriz de salida está fijado en 1 si alguno de matriz de entrada contiene un elemento distinto de cero en ese punto misma matriz. De lo contrario, ese elemento se establece en 0. *A* y *B* deben tener las mismas dimensiones que el que no un escalar

Ejemplo**:** para *A* y *B*

>>A=[0 0 1 1;1 1 0 0;0 0 0 0;1 1 1 1]; *B*=[0 0 0 0;1 1 1 1;0 1 0 1;1 0 1 0]; la operación ***OR*** entre *A* y *B* es:

>> *A* | *B*

ans =

0 0 1 1
1 1 1 1
0 1 0 1
1 1 1 1

de la misma manera, para

>>*x* = [0 1 2 3 0]

>>*y* = [1 2 3 0 0]

>> *x* | *y*

ans = 1 1 1 1 0

*****XOR*** – Lógico excluyente *OR*

A	*B*	*x* o *r* (*A*, *B*)
0	0	0
0	1	1
1	0	1
1	1	0

Para el *OR* exclusivo lógico, *XOR* (*A*, *B*), el resultado es lógico 1 (*TRUE*), donde *A* o *B*, pero no ambas, es distinto de cero. El resultado es lógico 0 (*FALSO*), donde *A* y *B* son ceros o distinto de cero.

A y *B* siempre deben tener la misma dimensión (uno puede ser escalar) Ejemplo: para las mismas *A* y *B*, el lógico excluyente *EXCLUSIVE OR* entre *A* y *B* es:

```
>> xor(A,B)
ans =
    0    0    1    1
    0    0    1    1
    0    1    0    1
    0    1    0    1
```

para los mismos vectores *x* e *y*:

```
>> xor(x,y)

ans =
    1    0    0    1    0
```

*Lógico *NOT*

A	~A
0	1
1	0

En *MATLAB*, ~ A *NO* realiza una lógica de una matriz de entrada y devuelve una matriz que contiene los elementos establecidos ya sea lógico 1 (*TRUE*) o lógico 0 (*FALSO*).

Un elemento de la matriz de salida está fijado en 1, si *A* contiene un elemento de valor cero en la ubicación misma matriz. De lo contrario, ese elemento se establece en 0.

Para nuestro caso

```
>> ~A
ans =
    1    1    0    0
    0    0    1    1
    1    1    1    1
    0    0    0    0
>> ~x
ans =

    1    0    0    0    1
```

2.4. INTRODUCTORIAS

La jerarquía de los operadores aritméticos en *MATLAB* es la misma que en el lenguaje de alto nivel C y se resume en la siguiente tabla:

Prioridad	Nombre	Operador aritmético	*MATLAB*
1º	Potencia	^	*x* ^ *y*
2º	Producto	*	*x* * *y*
	Cociente	/	*x* / *y*
3º	Adicción	+	*x* + *y*
	Sustracción	–	*x* – *y*
4º	Asignación	=	*x = 2 + 3*

Tabla 1: Jerarquía de los operadores aritméticos de MATLAB

Como es de esperarse, el uso de paréntesis puede modificar la jerarquía de operadores aritméticos.

Ejemplo. Al introducir 2+3 y *ENTER* en la ventana principal de *MATLAB* obtenemos:

```
>> 2+3
ans =
    5
```

Ejemplo. En cambio, si introducimos 2+3**;** y *ENTER* tendremos solamente:

```
>> 2+3;
```

Es decir, *MATLAB* hará la evaluación directa y no arrojará el valor o resultado de la misma.

MATLAB también realiza evaluaciones por asignación:

Ejemplo. Si tecleamos *x*=2+3 y *ENTER* tendremos:

```
>> x=2+3
x =
    5
```

*Si deseamos limpiar la pantalla de los cálculos que hemos hecho, escribiremos *clc*
Si queremos disponer del ícono, acudimos a *shorcuts*.

2.5. FUNCIONES MATEMÁTICAS COMUNES

- *abs(x)* Valor absoluto de x, $|x|$.
- *sqrt(x)* Raíz cuadrada de *x*, $\sqrt{x}$.
- *round(x)* Redondeo de *x* al entero más cercano.

Cada vez que tecleemos ENTER aparecerá >>.

Ejemplo.

```
>> round(3.4)
ans =
    3
```

- *fix*(*x*) Redondea (o trunca) *x* al entero más cercano a cero.

Ejemplo.

```
>> fix(3.8)
ans =
    3
```

- *floor*(x) Redondea *x* al entero más cercano a -∞.
- *ceil*(x) Redondea *x* al entero más cercano a +∞.
- *sign*(*x*) Devuelve el valor de –1 si *x* es menor que 0, un valor de 0 si *x* es igual a

0 y un valor de 1, si *x* es mayor que 0.

Ejemplo.

>> *floor*(-3.5)

ans =

-4

>> *ceil*(-3.6)

ans = -3

>> *sign*(3.6)

ans =

1.

rem(*x*1,*x*2) División módulo, donde x_1 es el dividendo y x_2 es el divisor.

Ejemplo.

>> *rem*(7,3)

ans =1

Operaciones con números enteros: para calcular una operación entre números enteros con dado número de cifras exactas se utiliza el comando *vpa*(*variable precision arithmetic*), así 5 elevado a la 410 con 400 cifras exactas sería.

>>*vpa*(5^410,400)

ans =

3781828041845037190415746198017649519873965552668505367588673357670432345415338643101522906821433003237873387844559048109214286140667060745377456547389153689872320114101748635030117576253769335079931707798324180383317301362394089926439751133197361459582245439909318465229881454602118758 4.

Números enteros y divisibilidad

>> *rem*(14,3)

ans =2

>>*max*(2,14)

ns=14

máximo común divisor de 2000, 400 y 650

>>*gcd*(2000,gcd(400,650))

ans =

50

Mínimo común múltiplo de2000, 400 y 650

lcm(2000,lcm(400,650))

ans =

26000

Descomposición en factores primos.

>>*factor*(46)

ans =

2 23

Resto:

- *exp*(*x*) Calcula e^x donde *e*, es la base de los logaritmos naturales (2.718282...)
- *log*(*x*) Calcula *ln*(*x*), el logaritmo natural de *x*.
- *log*10(*x*)Calcula $\log_{10}(x)$, el logaritmo común de *x* con base 10

2.6. OPERACIONES ARITMÉTICAS DE ARREGLOS ELEMENTO POR ELEMENTO

En *MATLAB*, para definir un vector pueden emplearse un espacio en blanco o bien una coma como separadores.

Ejemplo. Para introducir los vectores *A* y *B* donde *A*=(2,5,6) y *B*=(2,3,5), escribimos

>> *A* = [2 5 6];

>> *B* = [2, 3, 5];

Y las operaciones aritméticas elemento por elemento, para las cuales es necesario escribir punto antes del operador aritmético en cuestión.

Ejemplo. Evalúe las siguientes operaciones en *MATLAB*, donde *A*=(2,5,6) y *B*=(2,3,5).

a). *A***B*=[2(2),5(3),6(5)]

>> *A.*B*

ans =

4 15 30

b). A^2=[2^2,5^2,6^2]

>> *A.^2*

ans =

4 25 36

c). $\frac{1}{A} = \left(\frac{1}{2}, \frac{1}{5}, \frac{1}{6}\right)$

>> *1./A*

ans =

0.5000 0.2000 0.1667

d). $\frac{A}{B} = \left(\frac{2}{2}, \frac{5}{3}, \frac{6}{5}\right)$

>> *A./B*

ans =

1.0000 1.6667 1.2000

e). $\frac{A}{2} = \left(\frac{2}{2}, \frac{5}{2}, \frac{6}{2}\right)$

>> *A/2*

ans =

1.0000 2.5000 3.0000

2.7. FUNCIONES TRIGONOMÉTRICAS

Las siguientes funciones operan sobre ángulos expresados en radianes:

ángulo_grados = *ángulo_radianes* * (180/*pi*);
ángulo_radianes = *ángulo_grados* * (*pi*/180);

- *sin(x)* Seno de x.
- *cos(x)* Coseno de x.
- *tan(x)* Tangente de x.
- *asin(x)* Arco seno de x, -1 ≤ x ≤ 1, devuelve un ángulo en radianes entre -π/2 y π/2.
- *acos(x)* Arco coseno de x. Devuelve un ángulo en radianes entre 0 y π.
- *atan(x)* Arco tangente de x. Devuelve un ángulo en radianes entre -π/2 y π/2.

Las demás funciones trigonométricas pueden calcularse empleando las siguientes ecuaciones:

$$\sec(x) = \frac{1}{\cos(x)} \qquad \csc(x) = \frac{1}{sen(x)} \qquad \cot(x) = \frac{1}{\tan(x)}$$

2.8. FUNCIONES HIPERBÓLICAS

Las *funciones hiperbólicas* son funciones de la función exponencial natural, e^x ; las funciones hiperbólicas inversas son funciones de la función de logaritmo natural. A continuación se describen brevemente las funciones hiperbólicas de *MATLAB*.

- *sinh(x)* Seno hiperbólico de x, su fórmula es:
  $$\frac{e^x - e^{-x}}{2}$$
- *cosh(x)* Coseno hiperbólico de x, su fórmula es:
  $$\frac{e^x + e^{-x}}{2}$$
- *tanh(x)* Tangente hiperbólica de x, su fórmula es:
  $$\frac{senh(x)}{\cosh(x)}$$
- *asin(x)* Arco seno hiperbólico de x, (seno hiperbólico inverso), igual a:
  $$\ln(x + \sqrt{x^2 + 1})$$
- *acosh(x)* Arco coseno hiperbólico de x (coseno hiperbólico inverso), igual a:

$\ln(x + \sqrt{x^2 - 1})$

- *atanh(x)* Arco tangente hiperbólico de x (tangente hiperbólico inverso),

igual a:

$\ln\left(\sqrt{\frac{1+x}{1-x}}\right)$ para $|x| \leq 1$

2.9. OPERACIONES CON NÚMEROS COMPLEJOS

Un número complejo puede expresarse de la forma $z = a + ib$, donde i es $\sqrt{-1}$, a es la parte real del número y b es la parte imaginaria. Sean z_1 y z_1 dos números complejas, es decir, $z_1 = a_1 + ib_1$ y $z_2 = a_2 + ib_2$. Sobre los complejos se definen las siguientes operaciones:

Suma: $z_1 + z_2 = (a_1 + a_2) + i(b_1 + b_2)$

Resta: $z_1 - z_2 = (a_1 - a_2) + i(b_1 - b_2)$

Producto: $z_1 \times z_2 = (a_1a_2 - b_1b_2) + i(a_1b_2 + a_2b_1)$

División: $\frac{z_1}{z_2} = \left(\frac{a_1a_2 + b_1b_2}{a_2^2 + b_2^2}\right) + i\left(\frac{a_2b_1 - b_2a_1}{a_2^2 + b_2^2}\right)$

Magnitud o valor absoluto de z_1: $|z_1| = \sqrt{a_1^2 + b_1^2}$

Conjugado de z_1: $z_1^* = a_1 - ib_1$

En *MATLAB* un número complejo se almacena en dos variables (la real y la imaginaria), y los comandos de *MATLAB* suponen que *i* representa $\sqrt{-1}$ a menos que se haya dado a *i* un valor distinto.

Ejemplo. La variable compleja puede definirse a través del siguiente comando:

>> *z* = 1 - *i* * 0.4;

Las operaciones aritméticas con números complejos pueden realizarse a través de los siguientes comandos:

- *conj(j)* Calcula el conjugado de *z*.
- *real(z)* Calcula la parte real de *z*.

imag(z),abs(z) Calcula la parte imaginaria de *z* y el valor absoluto o magnitud de *z*, respectivamente. La función *isreal* retorna 0 en caso de que el complejo al cual es aplicada posea parte imaginaria no nula, y 1 en caso contrario.

Ejemplo: sean los complejos

>> *a* = 1 + i;

>> *b* = 1;

>> *c* = i

>> *isreal(a) ans* = 0

>> *isreal(b) ans* = 1

Apliquemos ahora la función ***abs***:

>> *abs(a) ans* = 1.4142

>> *abs(b) ans* = 1

>> *abs(c) ans* = 1

Puede observarse que esta función retorna el módulo de un complejo.

Para conseguir el argumento se aplica ***angle,*** el resultado es el ángulo en radianes.

>> *angle(a)*

ans = 0.7854

Note que es lo mismo

>> *atan(imag(a)/real(a))*

ans = 0.7854

Si queremos el ángulo expresado en grados debemos utilizar la función *rad2deg* (también existe la función inversa *deg2rad)*:

>> *rad2deg(ans)*

ans = 45

Entonces el ejercicio de convertir un número complejo a forma polar puede llevarse a cabo en dos pasos:

>> *d = -sqrt*(3) + *i*

d = -1.7321 + 1.0000i

>> *modulo = abs(d)*

modulo = 2

>> *argumento = rad2deg(angle(d))*

argumento = 150.0000

Es interesante ver la representación gráfica de los complejos en el plano; mediante la función ***polar*** podemos hacerlo.

>> *polar(angle(a), abs(a))*; % no se ve punto en la figura

Esta función recibe dos argumentos, el primero es el argumento del número complejo a representar, y el segundo su módulo.

>>*polar(angle(a), abs(a), '*r')*

Ahora la función recibe tres argumentos, los dos que ya conocemos y un tercero que le indica a *MATLAB* que aspecto tendrá el complejo,

- el asterisco, *, significa que en lugar de aparecer un diminuto punto aparecerá un *
- la letra ***r*** indica el color del asterisco, en este caso **r**ed (rojo).

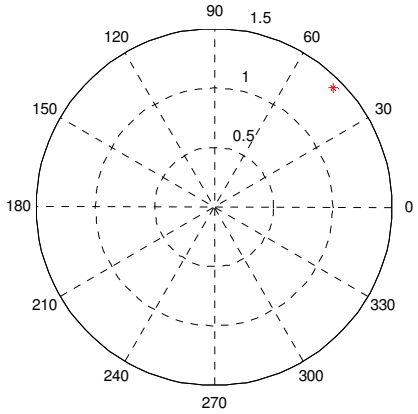


Figura 2.6

2.9.1. Operaciones básicas

Sean
>> *g* = 3 + 2*i*;
>> *h* = 4 - *i*;
Suma: Sencillamente se utiliza el símbolo +.
>> *g* + *h*
ans = 7.0000 + 1.0000i
Resta: Se utiliza el símbolo -.
>> *g* - *h*
ans = -1.0000 + 3.0000i
Producto: Se utiliza el símbolo *.
>> *g* * *h*
ans = 14.0000 + 5.0000i
División: Se utiliza el símbolo /.
>> *g* / *h*
ans = 0.5882 + 0.6471i

2.9.2. Potencias de complejos

Sea el complejo Z=1+*i*, de módulo raíz de 2 y argumento *pi*/4; las sucesivas potencias , para *k*=1..7 serán (raíz de 2)^k y argumentos *k*(*pi*/4), representando en *MATLAB*
> *z*=1 +*i*; *n*=1:8;
>> *compass*(*z*.^*n*,'*r*')

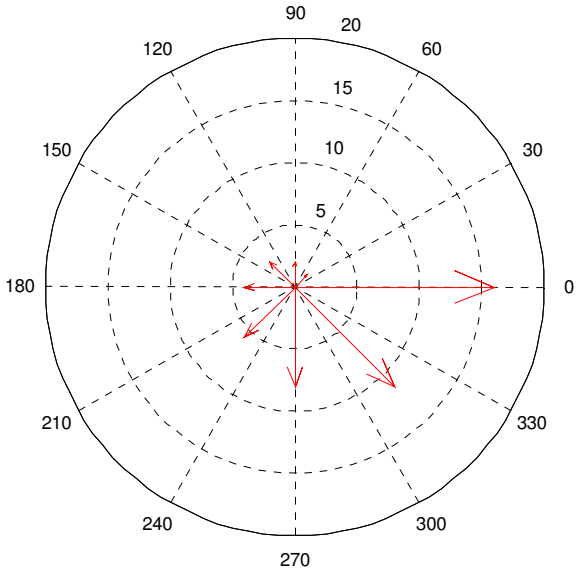


Figura 2.7

Las raíces de un complejo, $z=|z|(\cos\theta+i\,\mathrm{sen}\,\theta)$ las obtenemos como
$w_k=|z|^{1/n}(\cos\varphi_k+i\,\mathrm{sen}\,\varphi_k)$ con $\varphi_k=(\theta/n)+k(2\pi/n)$, $k=0.1.,\ldots,n-1$
Sea $z=-1$, cuyo módulo es 1 y argumento $3\pi/2$; las raíces cúbicas son de módulo 1 argumento $(\pi/2)+k(2\pi/3)$, con $k=0,1,2$.

Con *MATLAB*, su representación será:

```
>> z=-i;
>> compass(z,'-.')
>> hold
Current plot held
for k=0:2
fi=pi/2+k*2*pi/3;
compass(cos(fi)+i*sin(fi),'r')
end
```

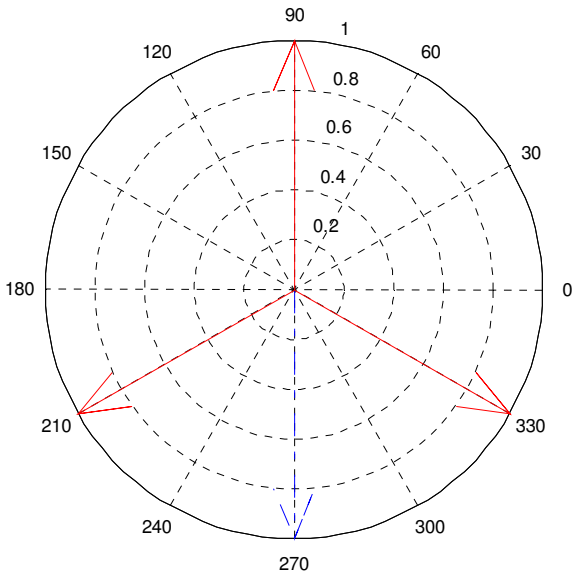


Figura 2.8

CAPITULO 3: POLINOMIOS

3.1. FUNCIONES POLINÓMICAS

Un polinomio puede expresarse de manera general como sigue:

$f(x) = a_0x^n + a_1x^{n-1} + \ldots + a_{n-2}x^2 + a_{n-1}x + a_n$

En *MATLAB*, el polinomio $f(x) = 3x^4 - 7x^3 + x - 9$ se define de la siguiente manera:

>>*f* = 3 *x ^ 4 – 7 *x ^ 3 + x -9;

3.1.1. Evaluación de polinomios

Los polinomios pueden evaluarse con la función *polyval(a, x)*, donde *a* es la lista de coeficientes de polinomio *x*.

Ejemplo. Defina el vector de coeficientes del polinomio $f(x) = 3x^4 - 7x^3 + x - 9$.

>>*a* = [3, -7, 0, 1, -9]

Para evaluar polinomios con la función *polyval(a,x)* es necesario indicar además de la lista de coeficientes, el dominio de *x* y los incrementos en el intervalo sobre los que se evaluará el polinomio. Para ello, escribimos:

x= inicio_intervalo:incremento:fin_intervalo;

Ejemplo. Evaluar el polinomio $g(x) = -x^4 + 3x^3 - 2.5x^2 - 2.5$ en el intervalo [0,5] con incrementos de 0.1

>> *a* = [-1, 3, -2.5, 0, -2.5];
>> *x* =0:0.1:5;
>> *polyval(a,x)*
ans =
Columns 1 through 13
-2.5000 -2.5221 -2.5776 -2.6521 -2.7336 -2.8125 -2.8816 -2.9361
-2.9736 -2.9941 -3.0000 -2.9961 -2.9896
Columns 14 through 26
-2.9901 -3.0096 -3.0625 -3.1656 -3.3381 -3.6016 -3.9801 -4.5000
-5.1901 -6.0816 -7.2081 -8.6056 -10.3125
Columns 27 through 39
-12.3696 -14.8201 -17.7096 -21.0861 -25.0000 -29.5041 -34.6536 -40.5061
-47.1216 -54.5625 -62.8936 -72.1821 -82.4976
Columns 40 through 51
-93.9121 -106.5000 -120.3381 -135.5056 -152.0841 -170.1576 -189.8125 -211.1376
-234.2241 -259.1656 -286.0581 -315.0000

MATLAB arroja 51 evaluaciones debido a que el intervalo incluye al cero.

Si hacemos la asignación *g=polyval(a,x)*, *g* guardará la evaluación de cada *x* y, por supuesto, *x*=0:0.1:5 almacena en *x* los valores en el dominio sobre los cuales se evaluó el polinomio. En otras palabras, *g* almacena el *codominio* y *x* el *dominio*. Conviene recordar esto para graficar polinomios.

Las operaciones con polinomios se realizan con la misma lógica que hemos seguido hasta ahora:

Ejemplo. Sean $g(x) = x^4 - 3x^2 - x + 2.4$ y $h(x) = 4x^3 - 2x^2 + 5x - 16$. Realice las operaciones $s(x) = g(x) + h(x)$, $t(x) = g(x) - h(x)$ y $u(x) = 3g(x)$.

```
>>g = [ 1, 0, -3, -1, 2.4];
>>h = [ 0, 4, -2, 5, -16];
>> g+h
ans =
   1.0000    4.0000   -5.0000    4.0000  -13.6000
>> g-h
ans =
   1.0000   -4.0000   -1.0000   -6.0000   18.4000
>> 3*g
ans =
   3.0000         0   -9.0000   -3.0000    7.2000
```

A su vez, *s*, *t*, y *u* son vectores que pueden ser evaluados para *x*.

Para el producto de polinomios se emplea el comando *conv(a, b)* donde *a* y *b* son vectores de coeficientes.

Ejemplo. Sean $g(x) = x^4 - 3x^2 - x + 2.4$ y $h(x) = 4x^3 - 2x^2 + 5x - 16$. Calcule $p(x) = g(x)h(x)$.

```
>> p=conv(g,h)
p =
  0    4.0000   -2.0000   -7.0000  -14.0000   -3.4000   38.2000   28.0000
 -38.4000
```

Para el cociente se usa la función o comando *[q,r]=deconv(n,d)*, la cual devuelve dos vectores de coeficientes, el primero contiene los coeficientes del cociente y el segundo los coeficientes del residuo.

Ejemplo. Sean $g(x) = x^4 - 3x^2 - x + 2.4$ y $h(x) = 4x^3 - 2x^2 + 5x - 16$. Calcule $q(x) = \dfrac{h(x)}{g(x)}$.

```
>> [q,r]=deconv(h,g)
q =
   0
r =
    0    4   -2    5  -16
```

Si *f* es un polinomio de coeficientes en Q, el comando *factor(f)* nos dará un producto de polinomios de menor grado con coeficientes racionales

```
>> syms x;
>> f=x^3-6*x^2+11*x-6;factor(f)
 ans =
  (x-1)*(x-2)*(x-3)
```

Ahora si el polinomio de grado *n*, factores de tipo x^n+1, con el desarrollo:

```
>>      syms x;
>>      n = (1:9)';
>>      p = x.^n + 1;
>> f=factor(p);[p f]
```

ans =

```
[          x+1,                    x+1                ]
[        x^2+1,                    x^2+1              ]
[        x^3+1,              (x+1)*(x^2-x+1)          ]
[        x^4+1,                    x^4+1              ]
[        x^5+1,        (x+1)*(x^4-x^3+x^2-x+1)        ]
[        x^6+1,          (x^2+1)*(x^4-x^2+1)          ]
[        x^7+1, (x+1)*(1-x+x^2-x^3+x^4-x^5+x^6)       ]
[        x^8+1,                    x^8+1              ]
[        x^9+1,    (x+1)*(x^2-x+1)*(x^ 6-x^3+1)       ]
```

Nos da una matriz con los polinomios en la primera columna y sus factorizaciones en la segunda.

La función *simplify* es una poderosa herramienta de propósito general que se aplica una serie de identidades algebraicas de sumas, potencias integrales, raíces cuadradas, etc

```
>> syms x;f=x*(x*(x-6)+11)-6;simplify(f)
 ans =
   x^3-6*x^2+11*x-6
>>f=(1-x^2)/(1-x);
>>simplify(f)
ans= x+1
>> syms x y positive;
>> f=log(x*y) ;
>> simplify(f)
 ans =
  log(x)+log(y)
```

La función *simple* logra su objetivo de simplificar independiente de aplicar *simplify*, *collect*, factor, y otras funciones de simplificación a una expresión resultados. devuelve el menor resultado

```
>> simple(cos(x)^2 + sin(x)^2)
```

Dará las siguientes simplificaciones alternativas en la línea de comandos

```
simplify:
1
radsimp:
cos(x)^2+sin(x)^2
combine(trig):
1
factor:
cos(x)^2+sin(x)^2
expand:
cos(x)^2+sin(x)^2
combine:
```

```
1

convert(exp):
(1/2*exp(i*x)+1/2/exp(i*x))^2-1/4*(exp(i*x)-1/exp(i*x))^2

convert(sincos):
cos(x)^2+sin(x)^2

convert(tan):
(1-tan(1/2*x)^2)^2/(1+tan(1/2*x)^2)^2+
4*tan(1/2*x)^2/(1+tan(1/2*x)^2)^2

collect(x):
cos(x)^2+sin(x)^2
devolviendo
ans =
   1
Si escribimos de la forma:
>> f = simple(cos(x)^2+sin(x)^2)
f=1
La función simple a veces mejora el resultado de simplify
>> syms a;
>> f=(1/a^3+6/a^2+12/a+8)^(1/3);
simplify(f)
 ans =
  ((2*a+1)^3/a^3)^(1/3)
>> g=simple(f)
 g =
  (2*a+1)/a
Es muy efectiva en las expresiones trigonométricas:
> syms x,
>> f=cos(x)^2-sin(x)^2;g=simple(f)
 g =
   cos(2*x)
>> cos(3*acos(x));h=simple(cos(3*acos(x)))
 h =
   4*x^3-3*x
```

3.1.2. Raíces de polinomios

Sea $y=f(x)$ los valores de x que hacen $f(x)=0$ se denominan raíces. Las raíces de un polinomio pueden ser reales o complejas. Si un polinomio tiene coeficientes $a_0, a_1, a_2, .. a_{n-1}, a_n$ reales, entonces todas las raíces complejas siempre ocurrirán en

pares conjugados complejos. Por ejemplo, un polinomio cúbico tiene la siguiente forma general:

$$f(x) = a_0x^3 + a_1x^2 + a_2x + a_3$$

El teorema fundamental del álgebra indica que un polinomio de grado n, tiene n raíces. En el caso del polinomio cúbico pueden darse los siguientes casos:

- Tres raíces reales distintas.
- Una raíz real con multiplicidad 3.
- Una raíz real simple y una raíz real con multiplicidad 2.
- Una raíz real y un par conjugado complejo.

Ejemplo. Las raíces de los siguientes polinomios se resumen a continuación.

a) Tres raíces reales distintas:

$$f_1(x) = x^3 - 3x^2 - x + 3$$
$$= (x - 3)(x + 1)(x - 1)$$
$$x_1 = 3, x_2 = -1, x_3 = 1$$

b) Una raíz real con multiplicidad 3:

$$f_2(x) = x^3 - 6x^2 + 12x - 8$$
$$= (x - 2)^3$$
$$x_1 = 2, x_2 = 2, x_3 = 2$$

c) Una raíz real simple y una raíz real con multiplicidad dos:

$$f_3(x) = x^3 - 12x + 16$$
$$= (x + 4)(x - 2)^2$$
$$x_1 = -4, x_2 = 2, x_3 = 2$$

d) Una raíz real y un par conjugado complejo:

$$f_4(x) = x^3 - 2x^2 - 3x + 10$$
$$= (x + 2)(x - (2 + i))(x - (2 - i))$$
$$x_1 = -2, x_2 = 2 + i, x_3 = 2 - i$$

La función de *MATLAB* para determinar las raíces de un polinomio es *roots(a)* donde *a*, es el vector de coeficientes del polinomio.

Ejemplo. Sea $f(x) = x^3 - 2x^2 - 3x + 10$. Obtenga sus raíces.

Puede hacerse:

```
>> p=[ 1, -2, -3, 10 ];
>> roots(p)
ans =
   -2.0000 2.0000 + 1.0000i
    2.0000 - 1.0000i
```

O directamente:

```
>> roots([ 1, -2, -3, 10 ])
ans =
   -2.0000
    2.0000 + 1.0000i
    2.0000 - 1.0000i
```

Para comprobar que estas son las raíces del polinomio, se sustituyen en el comando *polyval(p,r)*.

Otros ejercicios para insertar de polinomios

```
>> syms a b p x
>> a = x^3+3*x^2-2*x+7;
>> b = x^2+x+3;
>> p = a * b
p =
  (x^3+3*x^2-2*x+7)*(x^2+x+3)
```

obtenemos el polinomio producto en forma factorizada. Si lo queremos ver forma expandida, pondremos

```
>> expand(p)
ans =
  x^5+4*x^4+4*x^3+14*x^2+x+21
```

Supongamos que en el polinomio $f = ax^2 + bx + c$ queremos sustituir x por -1. Para ello, si escribimos

```
>> syms x a b c
>> f = a*x^2 + b*x + c;
>> g = subs(f,x,-1)
```

entonces sale

```
g =
  a - b + c
```

Potencia de un binomio

```
>> syms s t;expand((s+t)^3)
ans =
  s^3+3*s^2*t+3*s*t^2+t^3
```

<u>Numerador y denominador</u>

En una expresión simbólica racional suele interesar conocer el numerador y el denominador de la misma. Para esto tenemos la orden numden. Si, por ejemplo, nos dan la expresión

$h = x^2 + 1/2x - 1 + x/x - 1$

```
>> syms x;
>> h= (x^2+1)/(2*x-1) + x/(x-1);
>> [n,d] = numden(h)
 n =
   x^3+x^2-1
d =
  (2*x-1)*(x-1)
```

<u>Conversión de polinomios</u>

Las órdenes *poly2sym* y *sym2poly* sirven, respectivamente, para convertir un polinomio expresado en forma numérica (vector de coeficientes) en su expresión simbólica, y viceversa.

```
>> p = [1 2 3 4 5];
>> px = poly2sym(p,x)
```

```
px =
  x^4+2*x^3+3*x^2+4*x+5

>> sym2poly(px)
ans =
  1    2    3    4    5

>> syms a b p x;
>> a = x^3+3*x^2-2*x+7;
>> b = x^2+x+3;
>> p = a * b

p =
   (x^3+3*x^2-2*x+7)*(x^2+x+3)
```

3.1.3. Gráficas de polinomios

En general, para graficar un polinomio es necesario realizar los siguientes pasos:

1. Definir el polinomio a través de un vector de coeficientes.
2. Generar en un arreglo los valores del dominio.

Nos detenemos aquí para definir un vector de términos crecientes o decrecientes, se utiliza la siguiente nomenclatura:

vector = [inicio_vector:incremento:fin_vector]

Ejemplo:

```
>> t = [0:0.1:10] % De esta forma definimos un vector t que va desde 0 hasta 10 con un
 incremento de 0.1
t =
Columns 1 through 8
    0 0.1000 0.2000 0.3000 0.4000 0.5000 0.6000 0.7000
```

3. Generar el contradominio y guardarlo en un arreglo.
4. Graficar el dominio x contra el contradominio y usando el comando *plot(x,y).*

Ejemplo. Graficar $f(x) = -x^4 + 7x^3 + 6x - 12$.

```
>> a=[ -1, 7, 0, 6,-12 ];
>> x=-5:0.1:5;
>> y=polyval(a,x);
>> plot(x,y);
>> title('Gráfica de un polinomio')
>> xlabel ('Dominio')
>> ylabel ('Codominio')
>> grid
```

Los comandos *title('Gráfica de un polinomio'), xlabel('Dominio'), ylabel('Codominio')* arrojan el título, la etiqueta del eje *x* y la etiqueta del eje *y*, respectivamente. La instrucción *grid* genera las líneas punteadas que se aprecian en la gráfica.

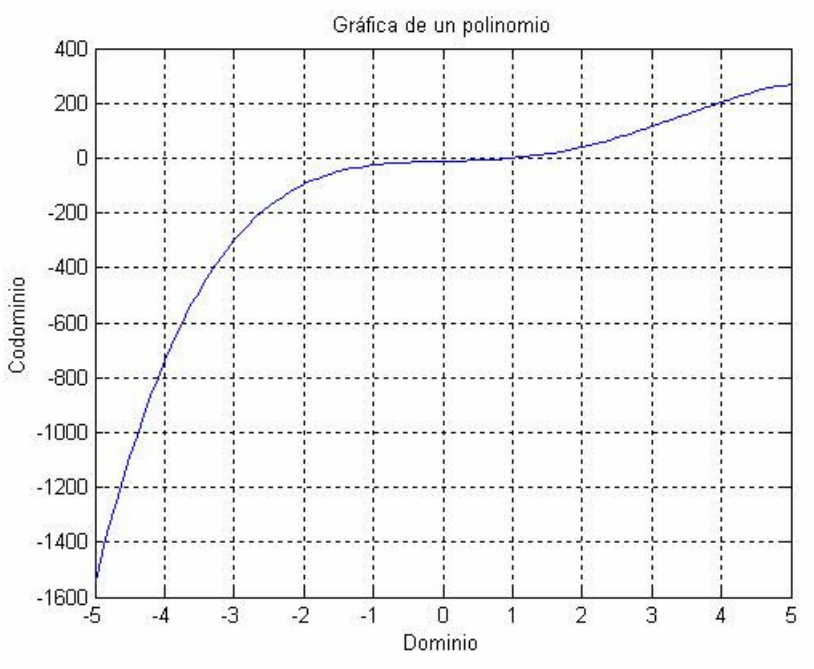


Figura 3.1

3.1.4. Fracciones parciales

Sea G el cociente de dos polinomios de la variable v

Esta última puede expresarse como el cociente de dos polinomios en v. El polinomio del denominador se puede escribir como un producto de factores lineales con raíces p_1, p_2, p_3 (las raíces del numerador se denominan *ceros* y las del denominador *polos* de la función). Cada raíz del denominador puede representar una raíz única o una raíz múltiple, como el caso

$$\frac{N(v)}{D(v)} = \frac{b_1 v^{n-1} + b_2 v^{n-2} + \ldots + b_{n-1} v + b_n}{(v - p_1)^{n1} (v - p_2)^{n2} \ldots (v - p_r)^{nr}}$$

Esta fracción propia puede escribirse como una *suma de fracciones parciales*. Las raíces únicas corresponden a un término de la fracción parcial; una raíz con multiplicidad k corresponderá a k términos de la expansión de fracciones parciales, la expansión puede denotarse por:

$$\begin{aligned}\frac{N(v)}{D(v)} = \quad & \frac{C_{11}}{v - p_1} + \frac{C_{12}}{(v - p_1)^2} + \ldots + \frac{C_{1m1}}{(v - p_1)^{m1}} \\ & + \frac{C_{21}}{v - p_2} + \frac{C_{22}}{(v - p_2)^2} + \ldots + \frac{C_{2m2}}{(v - p_2)^{m2}} \\ & + \ldots \\ & + \frac{C_{r1}}{v - p_r} + \frac{C_{r2}}{(v - p_r)^2} + \ldots + \frac{C_{rmr}}{(v - p_r)^{nr}}\end{aligned}$$

La función *residue(B,A)* realiza una expansión en fracciones parciales:

- *[r, p, k] = residue (B, A)*

Realiza una expansión en fracciones parciales de un cociente de dos polinomios, B / A . Los vectores B y A contienen los coeficientes de los polinomios B y A, respectivamente. El vector r contiene los coeficientes C_{ij} , el vector p contiene los valores de los polos p_n y el vector k contiene los valores de k_n .

Ejemplo. Realice la expansión en fracciones parciales de $f(z) = \dfrac{z^2}{z^2 - 1.5z + 0.5}$

```
>> B=[1,0,0];
>> A=[1,-1.5,0.5];
>> [r,p,k]=residue(B,A)
r =
    2.0000
   -0.5000
p =
    1.0000
    0.5000
k =
     1
```

Esto es:

$$f(z) = \frac{z^2}{z^2 - 1.5z + 0.5} = 1 + \frac{2}{z - 1.0} - \frac{0.5}{z - 0.5}$$

CAPITULO 4: GRAFICACIÓN

4.1. GRÁFICAS DE FUNCIONES

Anteriormente a un polinomio se le ha definido a través de un valor de coeficientes. Por ejemplo, en el polinomio $f(x) = 2x^2 - 3x$ el vector de coeficientes es $a = [2, -3, 0]$. Para graficar este polinomio y tomando considerando $x \hat{I} [0,5]$ con incrementos de 0.1, escribimos en *MATLAB* los siguientes comandos:

```
>> a = [2, -3, 0];
>> x = 0: 0.1: 5;
>> y=polyval(a,x);
>> plot(x,y);
>> title('Gráfica de función una variable')
>> xlabel('Dominio')
>> ylabel('Codominio')
>> grid
```

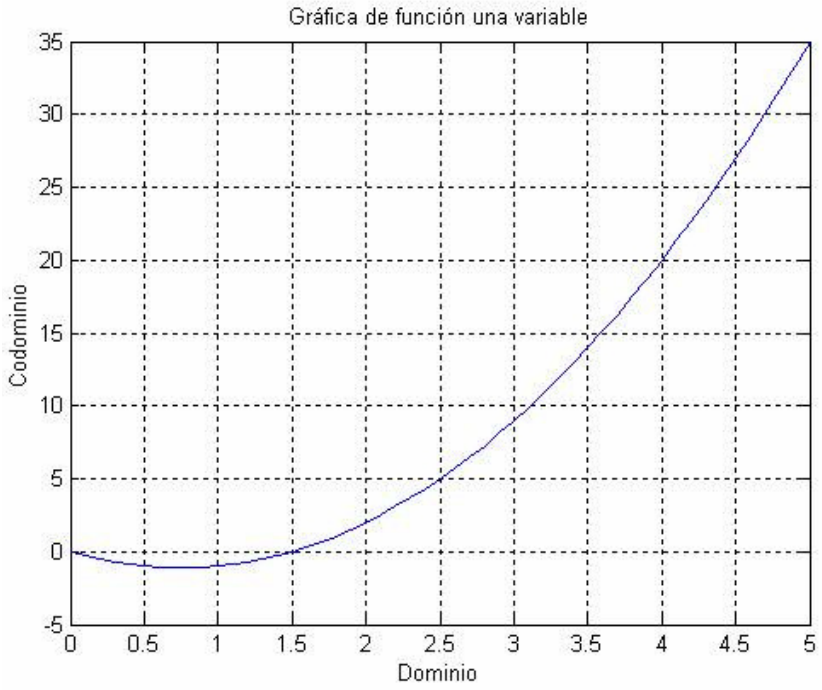


Figura 4.1

Otra manera es introducir directamente la función, lo cual se muestra a continuación:

```
>> x = 0:0.1: 5 ;
>> f=2*x.^2-3*x;
>> plot(x,f);
>> title('Gráfica de función de una variable')
>> xlabel('Dominio')
```

>> *ylabel('Codominio')*
>> *grid*
con lo cual se obtiene la misma gráfica

4.2. GRÁFICAS EN COORDENADAS POLARES

El comando *MATLAB* que genera una gráfica polar partiendo de los vectores *theta* y *r* es *polar(theta,r)*, el cual genera una gráfica usando los índices del vector *r* como los valores de q.
Ejemplo. Genere la gráfica polar donde los valores del ángulo van de 0 a $2p$ y el radio aumenta de 0 a 1.
>> *theta*=0:2*pi/100:2*pi;
>> *r=theta/(2*pi)*;
>> *polar(theta,r)*;
>>*title('Gráficapolar')*

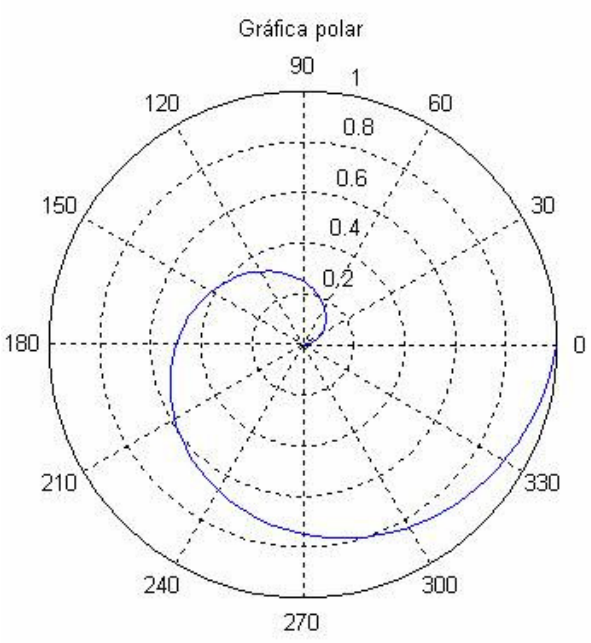


Figura 4.2

4.3. GRÁFICAS EN TRES DIMENSIONES

Para evaluar una función de dos variables *f(x,y)*, es necesario definir primero una *retícula bidimensional* en el plano *xy*. A continuación se evalúa la función en los puntos de la retícula para determinar los puntos de una superficie tridimensional.
Para definir una retícula bidimensional en el plano *xy* en *MATLAB* se utilizan dos matrices, una con las coordenadas *x* de todos los puntos de la retícula y otra con todas las coordenadas de *y*.
Por ejemplo, supongamos que queremos definir una retícula bidimensional en donde *x* varía de -2 a 2 en incrementos de 1 e *y* varía de -1 a 2 en incrementos de 1.
La matriz correspondiente de valores *x* de la retícula es la siguiente:

-2	-1	0	1	2
-2	-1	0	1	2
-2	-1	0	1	2
-2	-1	0	1	2

y para *y* se tiene:

-1	-1	-1	-1	-1
0	0	0	0	0
1	1	1	1	1
2	2	2	2	2

La combinación de ambas matrices genera los puntos de la retícula *xy*:

(-2,-1)	(2,-1)
(-2, 2)	(2, 2)

que son las esquinas de la retícula.

La instrucción [*x_grid,y_grid*]=*meshgrid(x,y)* genera dos matrices *x*, *y* para conformar la retícula bidimensional. Las matrices son de tamaño *m x n*, con base en los valores de los vectores *x* e *y* que contienen *m*-valores y *n*-valores respectivamente. La matriz *x_grid* contiene los valores de *x* repetidos en cada fila, y la matriz *y_grid* contiene los valores de *y* repetidos en cada columna.

Los comandos para generar gráficas en tercera dimension de *MATLAB* son los siguientes:

- *mesh(x,y,z)* Gráfica de malla.
- *surf(x,y,z)* Gráfica de superficie.
- *contour(x,y,z)* Gráfica de contorno. Es aquella que contiene un grupo de líneas que conectan elevaciones iguales, de manera tal que podemos identificar dónde están las montañas y los valles.
- *meshc(x,y,z)* Gráfica de malla/contorno.

Ejemplo. Elabore las correspondientes gráficas de la función de dos variables $z = f(x,y) = \frac{1}{1 + x^2 + y^2}$ considerando los intervalos $x \in [-2,2]$ y $y \in [-1,2]$ en incrementos de 0.1

A- Gráfica de malla.

```
>> x=-2:0.1:2;
>> y=-1:0.1:2;
>> [x_grid,y_grid]=meshgrid(x,y);
>> z = 1./(1 + x_grid.^2 + y_grid.^2);
>> mesh(x,y,z);
>> title('Gráfica de malla');
>> xlabel('X');
>> ylabel('Y');
>> zlabel('Z');
```

B- Gráfica de superficie.

```
>> x=-2:0.1:2;
>> y=-1:0.1:2;
>> [x_grid,y_grid]=meshgrid(x,y);
```

```
>> z = 1./(1 + x_grid.^2 + y_grid.^2);
>> surf(x,y,z);
>> title('Gráfica de superficie');
>> xlabel('X');
>> ylabel('Y');
>> zlabel('Z');
```

C- Gráfica de contorno.

```
>> x=-2:0.1:2;
>> y=-1:0.1:2;
>> [x_grid,y_grid]=meshgrid(x,y);
>> z = 1./(1 + x_grid.^2 + y_grid.^2);
>> contour(x,y,z);
>> title('Gráfica de contorno');
>> xlabel('X');
>> ylabel('Y');
```

D- Gráfica de malla/contorno.

```
>> x=-2:0.1:2;
>> y=-1:0.1:2;
>> [x_grid,y_grid]=meshgrid(x,y);
>> z = 1./(1 + x_grid.^2 + y_grid.^2);
>> meshc(x,y,z);
>> title('Gráfica de malla/contorno');
>> xlabel('X');
>> ylabel('Y');
>>zlabel('Z');
```

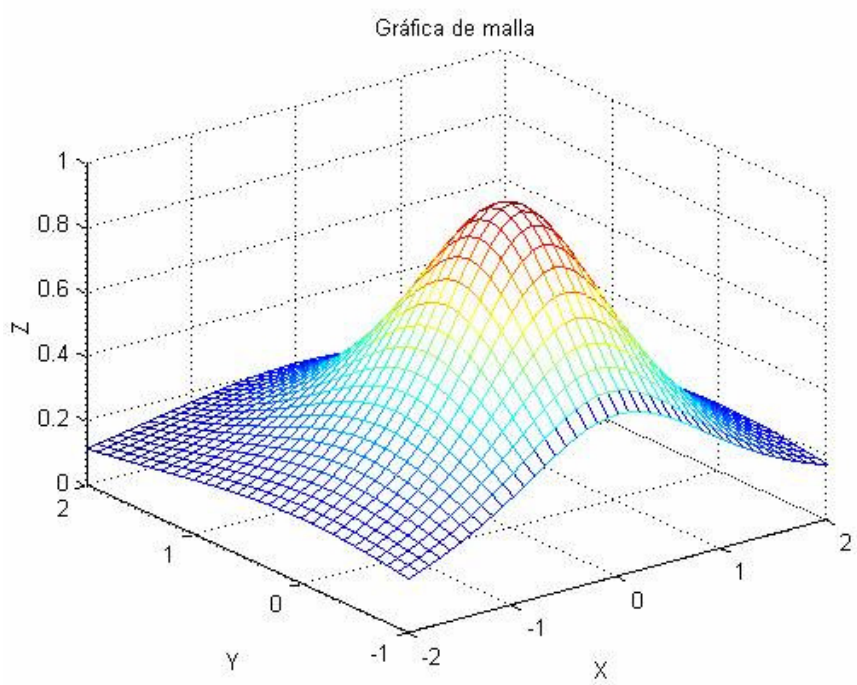


Figura 4.3

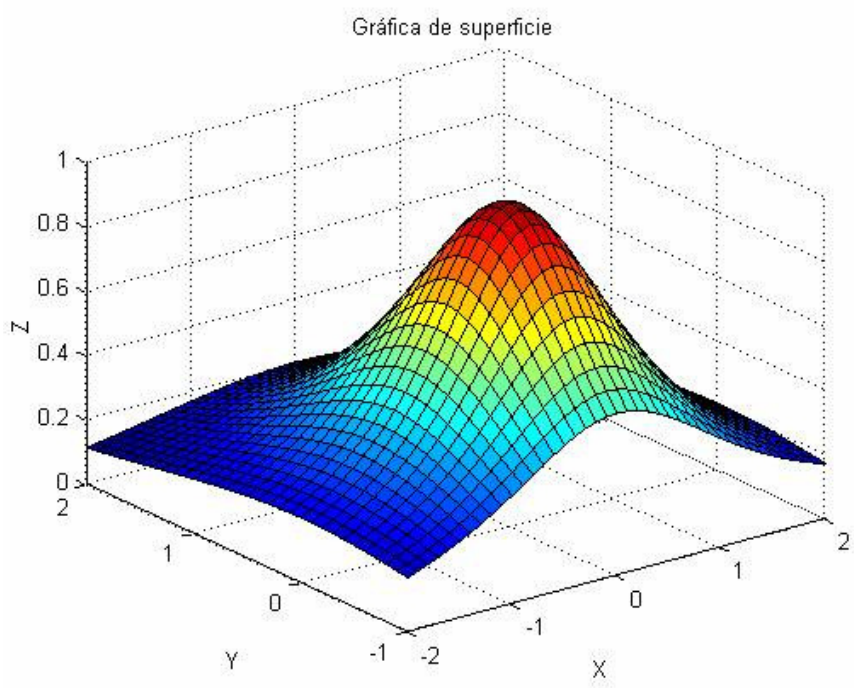


Figura 4.4

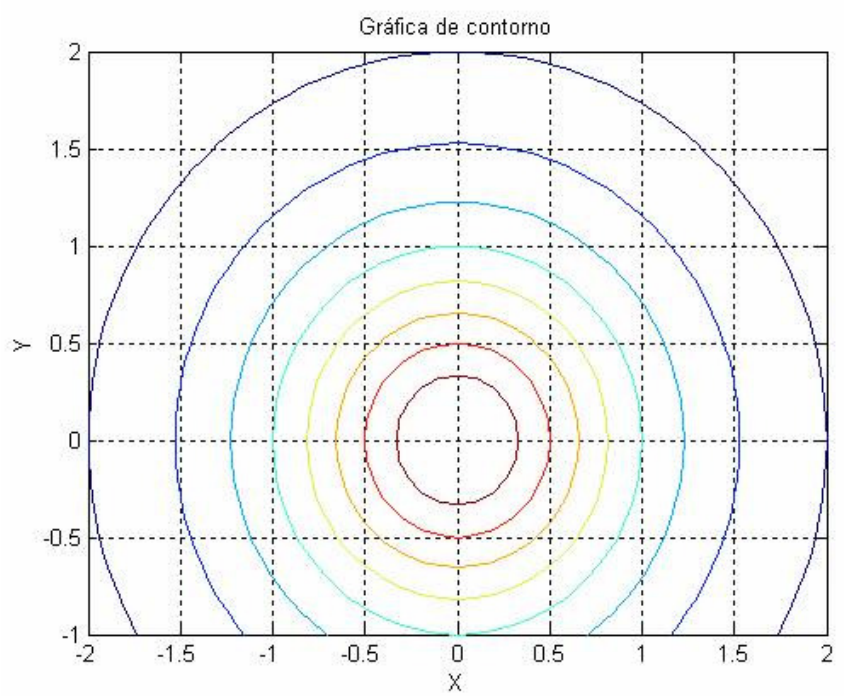


Figura 4.5

4.4. FUNCIONES DE ANÁLISIS DE DATOS E HISTOGRAMAS

Sea *x* un arreglo tipo vector. Para evaluar un conjunto de datos recabados de un experimento, se tienen las siguientes funciones en *MATLAB*.

- *sort(x)* Ordena ascendentemente los elementos del arreglo *x*.
- *max(x)* Determina el valor más grande contenido en *x*.
- *min(x)* Determina el valor menor contenido en *x*.
- *sum(x)* Determina la suma de los elementos de *x*:

$$\sum_{i=1}^{n} X_i$$

- *prod(x)* Determina el producto de los elementos de *x*:

$$\prod_{i=1}^{n} X_i$$

- *mean(x)* Determina la media del arreglo *x*:

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

- *median(x)* Determina la mediana del arreglo *x*.
- *std(x)* Determina la desviación estándar de *x*:

$$\sqrt{\frac{\sum_{i=1}^{n} (X_i - \bar{X})^2}{n - 1}}$$

La varianza de un arreglo puede calcularse como *std(x)^2*.

Ejemplo. Sea $x = (2.2, 5, 8.8, 4.3, 7.3)$.

a) Ordene ascendentemente los elementos del arreglo y determine los valores máximo y mínimo.

```
>> x=[2.2,5,8.8,4.3,7.3] ;
>> sort(x)
ans =
   2.2000    4.3000    5.0000    7.3000    8.8000
>> max(x)
ans =
   8.8000
>> min(x)
ans =
   2.2000
>> sum(x)
ans =
  27.6000
>> prod(x)
ans =
  3.0386e+003
```

b) Determine la suma y el producto de los elementos del vector.

```
>> x=[2.2,5,8.8,4.3,7.3];
>> sum(x)
ans =
   27.6000
>> prod(x)
```

```
ans =
   3.0386e+003
```

Ejemplo. Determine la media, la mediana, la desviación estándar y la varianza del *vector* $x = (3.5, 6.1, 9.8, 4.6, 3.5, 8.9, 3.1, 3.5)$.

```
>> x=[3.5, 6.1, 9.8, 4.6, 3.5, 8.9, 3.1, 3.5];
>> mean(x)
ans =
   5.3750
>> median(x)
ans =
   4.0500
>> std(x)
ans =
   2.6418
>> std(x)^2
ans =
   6.9793
```

4.5. HISTOGRAMAS

En *MATLAB*, el histograma calcula el número de valores que caen en 10 intervalos espaciados equitativamente entre los valores mínimo y máximo de un conjunto de valores, a menos que se le indique otra cosa. El histograma muestra no sólo los valores mismos sino además permite observar la forma en que están distribuidos. Los comandos para generar histogramas en *MATLAB* son:

- *hist(x)* Genera un histograma de los valores de *x* usando 10 intervalos.
- *hist(x,n)* Genera un histograma de los valores de *x* usando *n* intervalos.

Ejemplo. Obtenga el histograma en 10 y en 8 intervalos del siguiente conjunto de valores:

1.85	1.86	2.02	2.10	1.96	1.90	1.96	1.97	1.87	1.90
1.95	1.96	1.85	2.13	2.01	2.17	2.06	2.09	1.98	1.89
1.90	1.86	1.96	2.08	2.02	2.13	2.15	2.20	2.09	2.04
2.07	1.87	1.90	1.93	1.86	2.15	2.17	2.01	2.04	2.06
1.89	1.87	1.90	1.97	1.94	1.97	2.00	2.09	2.08	1.87

Para el histograma de 10 intervalos:

```
>> x=[ 1.85 1.86 2.02 2.10 1.96 1.90 1.96 1.97 1.87 1.90 1.95 1.96 1.85 2.13 2.01
2.17 2.06 2.09 1.98 1.89 1.90 1.86 1.96 2.08 2.02 2.13 2.15 2.20 2.09 2.04 2.07
1.87 1.90 1.93 1.86 2.15 2.17 2.01 2.04 2.06 1.89 1.87 1.90 1.97 1.94 1.97 2.00
2.09 2.08 1.87];
>> hist(x)
>> title('Histograma con 10 intervalos')
>> xlabel('Valor')
>> ylabel('Frecuencia')
```

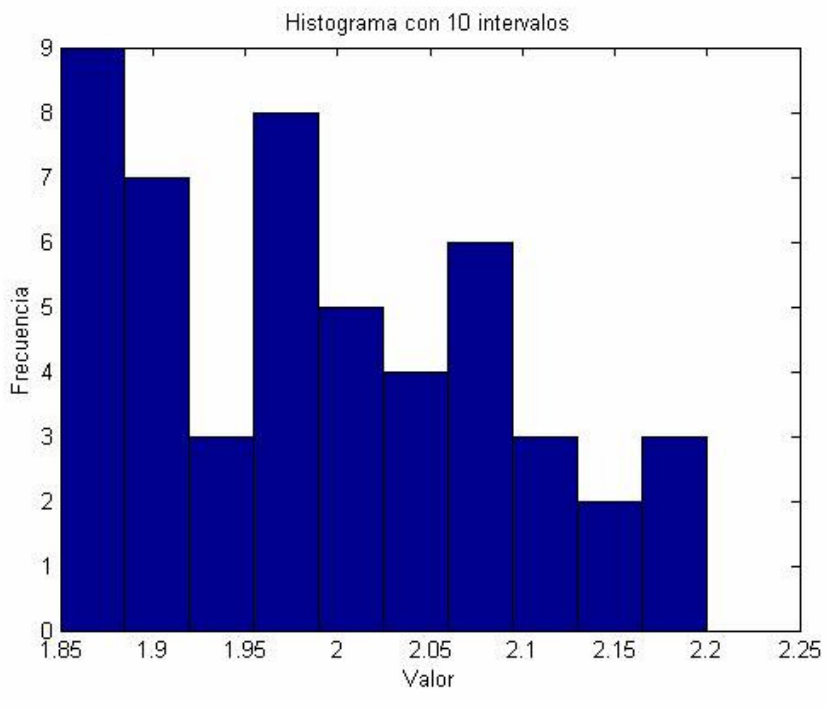


Figura 4.6

Dentro de las múltiples herramientas de *MATLAB*, se dispone de *funtool*, que se abre tipeando desde la línea de comandos.

>>*funtool*

Supongamos que $f=x^2$ y $g=x$, en el dominio (-4,4), se muestran los cuadros de diálogo para el cargado y las posibilidades de operación en este caso realizamos la suma de *f* y *g*

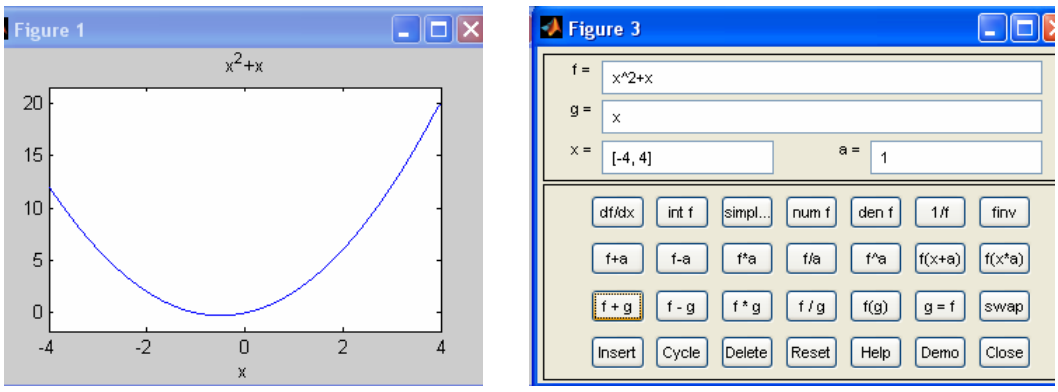


Figura 4.7

CAPITULO 5: SIMBÓLICA

5.1. CÁLCULO SIMBÓLICO

Una expresión simbólica se almacena en *MATLAB* como una cadena de caracteres, por lo que se emplean apóstrofos para definirlas. Las variables simbólicas de una expresión deben definirse con la instrucción *syms*:

```
>> syms x, y;
>> y=x^2;
```

o a través de apóstrofes:

```
>> y='x^2';
```

MATLAB también puede generar la gráfica de una expresión simbólica, donde la variable independiente generada adopta valores dentro del intervalo $[-2p, 2p]$, a menos que el intervalo contenga un punto para el cual la expresión no esté definida.

- *ezplot(s)* Genera una gráfica de *s*, donde *s* es una función de una variable. Por omisión la variable independiente corre en el intervalo $[-2p, 2p]$.
- *ezplot(s,[xmin,xmax])* Genera una gráfica de *s*, donde *s* es una función de una variable. La variable independiente corre desde *xmin* hasta *xmax*.
- *ezpolar(s)* Genera una gráfica de *s*, donde *s* es una función expresada en coordenadas polares.
- *ezsurf(s)* Genera una traza de superficies de *s*, donde *s* es una función de dos variables.

Ejemplo. Grafique

i) $y = x^3 - 6x + 1$ *en* $[-3, 2]$.

```
>> y='x^3-6*x+1';
>> ezplot(y,[-3,2]);
>> ylabel('y');
>> grid
```

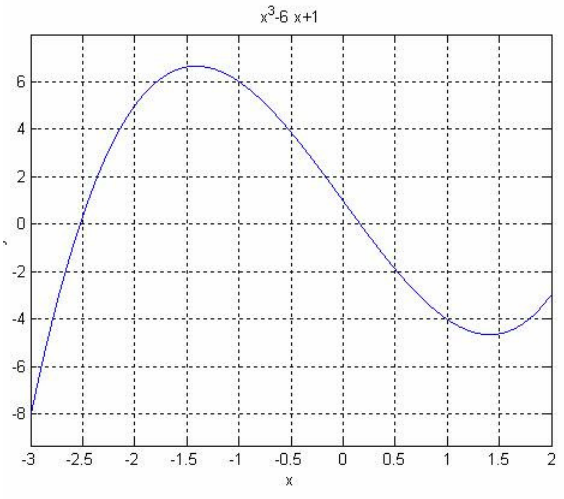


Figura 5.1

ii) $y = 3sen(x)\cos(x)$ *en* $[-3p, p]$.

```
>> y='3*sin(x)*cos(x)';
>> ezplot(y,[-3*pi,pi]);
>> ylabel('y');
>> grid
```

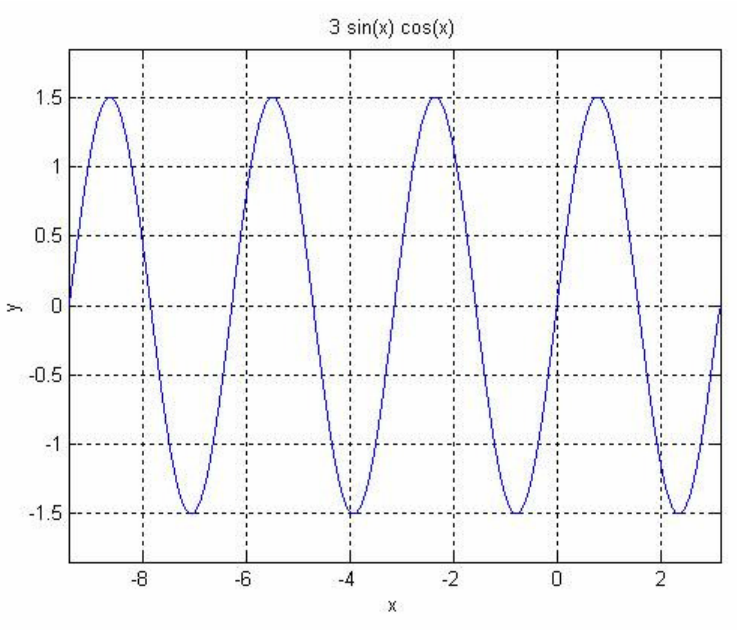


Figura 5.2

iii) $r = 1 + \cos(t)$ *en* $[0, 2p]$.

```
>> ezpolar('1+cos(t)')
```

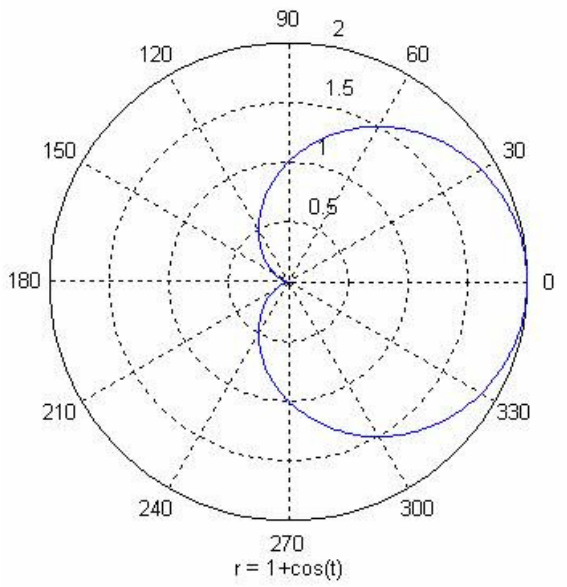


Figura 5.3

iv) $z = x^2 - y^2$ *.>>ezsurf('x^2-y^2')*

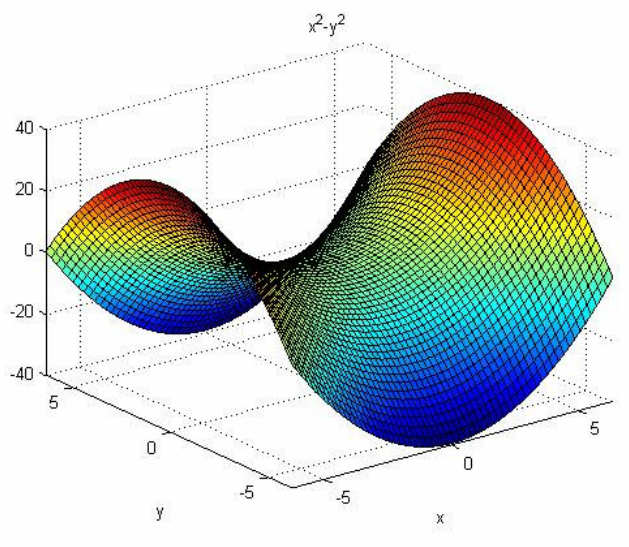


Figura 5.4

5.2. SIMPLIFICACIÓN DE EXPRESIONES SIMBÓLICAS

Al introducir expresiones como: *syms x*

a) $x * x * x$ *b)* $2 * x - 3 * x$ *c)* $x * x + x + 3 * x * x$

MATLAB realiza automáticamente su simplificación:
a) $x \wedge 3$ b) $-x$ c) $4 * x \wedge 2 + x$

Adicionalmente, se dispone de las siguientes operaciones algebraicas:

- *collect(s)* Agrupa términos semejantes de *s*.
- *collect(s,'v')* Agrupa términos semejantes de *s* respecto a la variable independiente *v*.
- *expand(s)* Realiza una expansión (desarrollo) de *s*.
- *factor(s)* Intenta factorizar *s* (sólo si los factores son racionales).
- *simple(s)* Simplifica la forma de *s* a una forma más corta si es posible.

Ejemplo. Considere que se ha definido en *MATLAB*:

```
>> syms x y a b;
>> s1=x^3-1;
>> s2=(x-3)^2+(y-4)^2;
>> s3=sqrt(a^4*b^7);
>> s4=14*x^2/(22*x*y);
```

OPERACIÓN	RESULTADO
factor(s1)	*(x-1)*(x^2+x+1)*

expand(s2)	*x^2-6*x+25+y^2-8*y*
collect(s2)	*x^2-6*x+9+(y-4)^2*
collect(s2,'y')	*y^2-8*y+(x-3)^2+16*
simple(s3)	*a^2*b^(7/2)*
simple(s4)	*7/11*x/y*

5.3. OPERACIONES SIMBÓLICAS

MATLAB puede realizar las siguientes operaciones simbólicas. Para *A* y *B* expresiones simbólicas:

- *symadd (A,B)* Realiza una suma simbólica, *A+B*.
- *symdiv (A,B)* Realiza una división simbólica *A/B*.
- *symmul (A,B)* Realiza una multiplicación simbólica *A*B*.
- *symsub (A,B)* Realiza una resta simbólica *A-B*.
- *sympow (S,p)* Realiza una elevación a potencia simbólica *S^p*.

Ejemplo. Considere que hemos definido en *MATLAB* las siguientes expresiones:

```
>> p1='1/(y-3)';
>> p2='3*y/(y+2)';
>> p3='(y+4)+(y-3)*y';
```

OPERACIÓN	RESULTADO
	symmul (p1, p3) *1/(y–3)*(y+4+(y-3)*y)* sólo la representa, *no* la evalúa. (para evaluarla se usa simplemente p1* p3).
sympow (p2,p3)	*27*y^3/(y+2)^3*. *MATLAB* simplifica

expresiones como *x^3/x^2* → x. En este caso el numerador es un monomio y se simplifica automáticamente,
no así el denominador que sólo se eleva simbólicamente.

symadd (p1, p2) *1/(y–3)+3*y/(y+2)* sólo representa simbólicamente la expresión, *n*o la evalúa.

A partir de la versión 7 de *MATLAB* los comandos se sustituyen por el símbolo que se muestra a su derecha:

- *symadd (A,B)* +
- *symdiv (A,B)* /
- *symmul (A,B)* *
- *symsub (A,B)* -
- *sympow (S,p)* ^

CAPITULO 6: LOS M

6.1. DECLARACIÓN DE UNA FUNCIÓN EN MATLAB

function variable = nombre (parámetros)
instrucciones
variable contendrá el resultado que entrega la función
parámetros son variable que reciben los datos que entran a la función
nombre identifica a la función
instrucciones se incluyen en la función según la tarea especificada

Las funciones se escriben en la ventana de edición de *MATLAB* y se las almacena en alguna carpeta, con extensión m. Es conveniente que el nombre asignado sea igual al nombre usado en la declaración de la función.

El uso de una función es similar al uso de las funciones comunes de *MATLAB*. El nombre debe coincidir con el nombre asignado, aunque los parámetros pueden tener nombres diferentes, pero su uso debe ser coherente.

Es decir que esquemáticamente:
La estructura a seguir a la hora de implementar una función es la siguiente:

```
~~~~~~~~~~~~~~~~~~~~ Identificador de "function".
|
| ~~~~~~~~~~~~~~~~ Argumento de salida.
| |
| | ~~~~~~~ Nombre de la función.
| | |
| | | ~ Argumento(s) de entrada.
| | | |
V V V V
function [o1,o2,...] = nombre_fun(i1,i2...)  Definición
% Aquí se escribiría la ayuda que queremos que aparezca cuando
% el usuario escriba "help nombre_fun"
% ...
% ...
```

Cuerpo de la función (Aquí estaría la parte del código).
Ej. Escriba una función para elegir el mayor entre dos números
Abra un documento nuevo en la ventana de edición y escriba:

```
function m = mayor(a, b)
if a>b
m = a;
else
m = b;
end
```

m es la variable que entrega el resultado
mayor es el nombre de la función
a, b son los parámetros que ingresan los datos a la función

Almacene esta función en el directorio con el nombre *mayor*
Suponer que quiere escoger el mayor entre e^{π} y π^{e}.
Escriba en la ventana de comandos:

```
>> a = exp(pi);
>> b = pi^exp(1);
>> m = mayor(a, b)
```

23.1407 (respuesta que muestra MATLAB)
***Los nombres de las variables pueden ser diferentes:

```
>> x = exp(pi);
>> y = pi^exp(1);
>> t = mayor(x, y)
```

1407. (respuesta que muestra *MATLAB*)

Ej. Escriba una función que reciba un número y determine si es un número primo. El resultado que entrega la función será 1 o 0 según corresponda;

```
function p = primo( x )
c = 0;
for d = 1: x
if mod(x, d) == 0
c = c + 1;
end
end
if c > 2
p = 0;
else
p = 1;
end
```

Guarde la función en el disco con el nombre **primo**
Pruebe la función desde la ventana de comandos

```
>> x = 25;
>> p = primo(x)
```

1. (resultado que muestra *MATLAB*)

```
>> x = 43;
>> p = primo(x)
```

2. (resultado que muestra *MATLAB*)

Escriba en una nueva ventana de edición un programa que use la función *primo* para encontrar todos los números primos menores a 20:

```
for x = 1: 20
if primo(x) == 1
disp(x);
end
end
```

Almacene su programa en el disco con el nombre *prove*
En la ventana de comandos pruebe su programa:

```
>> prove
```

1 (resultados mostrados por *MATLAB*)
2
3
5
7
11
13
17
19

**** *Una función puede entregar más de un resultado*

Las variables que entregan los resultados deben definirse entre [ ]

Ej. Escriba una función que entregue el área y el volumen de un cilindro dados su radio (*r*) y su altura (*h*)

```
function [area, vol] = cilindro(r, h)
area = 2*pi*r*h + 2*pi*r^2;
vol = pi*r^2*h;
```

Escriba y almacene la función con el nombre cilindro.

Use la función para calcular el área y el volumen de una lata de cilíndrica que tiene un diámetro de 10cm y una altura de 12cm

Escriba en la ventana de comandos:

```
>> r = 5;
>> h = 12;
>> [a, v] = cilindro(r,h);
>> a
>> v
```

MATLAB mostrará los resultados almacenados en a y en v

***** *Las variables definidas dentro de una función son locales, es decir que a diferencia de los programas, no son visibles fuera de la función*

Ej. Escriba la función:

```
function x=fn(a, b)
c = a + b;
x = 2*c;
```

Almacene con el nombre *fn* y úsela desde la ventana de comandos:

```
>> a = 3;
>> b = 5;
>> t = fn(a, b)
t = 16 (resultado que muestra MATLAB)
>> c (intentamos conocer el valor de c en la función)
??? Undefined function or variable 'c'. (mensaje de error de MATLAB)
```

Compare con lo que ocurre si escribe un programa en vez de la función;

```
a = input('ingrese dato ');
b = input('ingrese dato ');
c = a + b;
x = 2*c;
disp(x);
```

Almacene con el nombre prueba y active el programa:

>> *prueba*

ingrese dato 3 *(interacción para ingreso de datos)*

ingrese dato 5

16 *(resultado que muestra MATLAB)*

>> *c*

c = 8 (la variable c puede ser utilizada)

********Es posible hacer que las variables de una función sean visibles fuera de su ámbito, mediante la declaración global*

Ej. Modifique la función *fn* para que la variable c sea visible:

```
function x=fn(a, b)
global c;
c = a + b;
x = 2*c;
```

Almacene con el nombre fn y use la función:

```
>> a = 3;
>> b = 5;
>> t = fn(a, b)
```

t = 16 (resultado que muestra MATLAB)

>> *c (intentamos conocer el valor de c en la función)*

c= 8 (la variable c está disponible ahora)

********Una función puede no necesitar parámetros*

Ej. Escriba una función que lea y valide un entero entre 1 y 5

```
function n=entero
x=0;
while x==0
n=input('ingrese un entero entre 1 y 5 ');
if n>0 & n<6
x=1;
end
end
```

*********Una función puede no entregar resultados ni usar parámetros*

Ej. Escriba una función que imprima un menú

```
function menú
disp('1) ingresar');
disp('2) borrar');
disp('3) salir');
```

para usa esta función escriba

>> *menu*

**********Una función puede recibir como parámetros vectores o matrices.*

Ej. Escriba una función que reciba un vector y entregue el promedio del valor de sus elementos.

```
function p=prom(x)
n=length(x);
s=0;
for i=1:n
s=s+x(i);
```

```
end
p=s/n;
```
Para usar esta función debe definir el vector antes de llamar a la función.
La función determina la longitud del vector con la función *length*
```
>> x=[2 7 3 5 4 7 6];
>> t=prom(x)
```
t = 4.8571 (es el resultado que muestra *MATLAB*)

Un programa puede llamar a funciones

Ej. Escriba una función para eliminar espacios intermedios de una frase:

```
function x=compactar(f)
n=length(f);
x='';
for i=1:n
if f(i) ~= ' '
x = strcat(x, f(i));
end
end
```

Ahora escriba un programa que lea una frase, use la función *compactar* para eliminar los espacios intermedios, y luego muestre un mensaje en caso de que sea simétrica: sus caracteres opuestos son iguales

```
f=input('ingrese una frase ');
f=compactar(f);
n=length(f);
sim=1;
for i=1:n/2
if f(i) ~= f(n-i+1)
sim=0;
end
end
if sim == 1
disp('la frase es simetrica');
else
disp('la frase no es simetrica');
end
```

Probamos este programa suponiendo que lo hemos almacenado con el nombre *prove1*:

```
>> prove1
ingrese una frase 'carlos tiene sed'; (dato que ingresamos)
```

la frase es simétrica (resultado de *MATLAB*)

**********Una función puede llamarse a si misma

Estas funciones de denominan recursivas

Ej. Use la siguiente definición recursiva para calcular el máximo común divisor entre dos números enteros:

$$mcd(a,b) = \begin{cases} mcd(a-b,b), & a > b \\ mcd(a,b-a), & b > a \\ a, & a = b \end{cases}$$

Escriba y almacene una función para instrumentar esta definición:

```
function c=mcd(a, b)
if a>b
c=mcd(a-b, b);
else
if b>a
c=mcd(a, b-a);
else
c=a;
end
end
```

Use la función:

```
>> x=mcd(36, 48)
```

Para desplegar un mensaje de error y terminar la ejecución use error

Ej.

```
if d<0
error('valor incorrecto');
end
```

CAPITULO 7: SUCESIONES Y SERIES

7.1. SUCESIONES

Una sucesión en R^m es una función
$f : N \longrightarrow Rm$
$n \longrightarrow x_n = f(n)$
Notación standard: (x_n).
Sucesión acotada
(x_n) está acotada si existe M > 0 tal que
$\| x_n \| \leq M;\ n\ \varepsilon\ N$
Límite de una sucesión.
Sea (x_n) una sucesión de R^m. Decimos que
$lim_n\, xn = L\ si\ \ \forall \varepsilon > 0 \exists\ N$, tal que
$\|x_n - L\| < \varepsilon, n \geq N$
Resultados de importancia metodológica
Si $x_n = [x_{1;n};\ \ldots\ ;\ xm_{;n}]\ \varepsilon R^m$, entonces el estudio de la acotación o el límite de (x_n) se reduce al estudio de la acotación o el límite de cada sucesión componente
$(x_{k;n}),\ k = 1;\ \ldots;m.$
Hacer el gráfico de la sucesión
$x_n = 3n/n + 5;\ n = 1;\ 2;\ldots$ para opinar acerca de la existencia del límite de (x_n), considerando una cantidad razonable de valores de n. Estimar el valor del límite en caso de que se sospeche su existencia.

```
>> n=1:50;
>> s1=3*n./(n+5);
>> plot(s1,0,'b*')
```

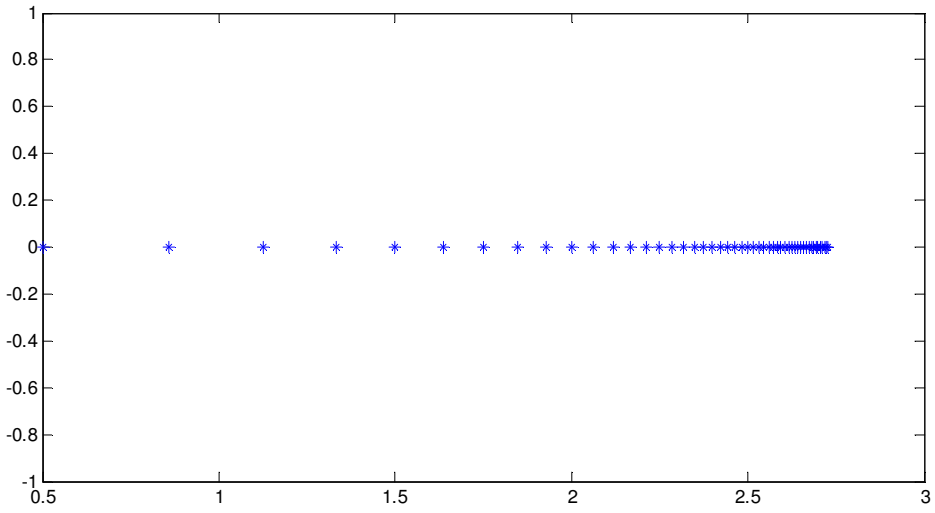


Figura 7.1

Se puede comparar para n creciendo la aproximación al límite

```
>> abs(s1(50)-3)
```

ans =

0.2727

> *abs(s1(100)-3)*

ans =

0.1429

Sea la sucesión x_n = *[2n sen(1/n); 3n/(n + 1)]*; en R^2

>>*n=1:50;s1=2*n.*sin(1./n);s2=3*n./(n+1);plot(s1,s2,'b.')*

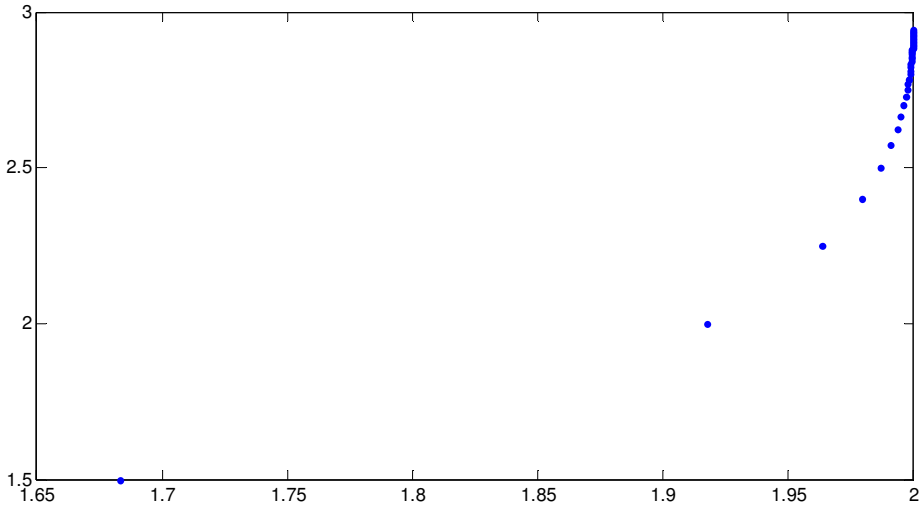


Figura 7.2

Debemos tener presente que en la Informática existen dos grandes ámbitos o vertientes tecnológicas: la simbólica y la numérica.

Ambas interactúan y se complementan. La primera permite simular los mecanismos mentales del hombre, por ejemplo, aquellos que conducen al cálculo de las primitivas de una función dada.

Mediante herramientas simbólicas el usuario puede simular en el ordenador una aritmética de precisión infinita, es decir, aparentemente trabaja con todos los números reales.

>>*syms n% Declarar simbolica la variable n*

>>*L1=limit(((6*n-3)/(3*n-7))^7, n, inf)*

L1 =

128

De la misma forma

>> *syms a n*

>> *limit((1+1/n)^n, n,inf) % limite cuando n tiende a infinito*

ans =

exp(1)

>> *limit(((n-a)/(n+3))^(n/2), n,inf)*

ans =

exp(-a-3)^(1/2)

7.2. GRAFICACIÓN

si x es un vector de números enteros
>>*m*=1:20; *x*=1./*m*; *plot*(*x*)

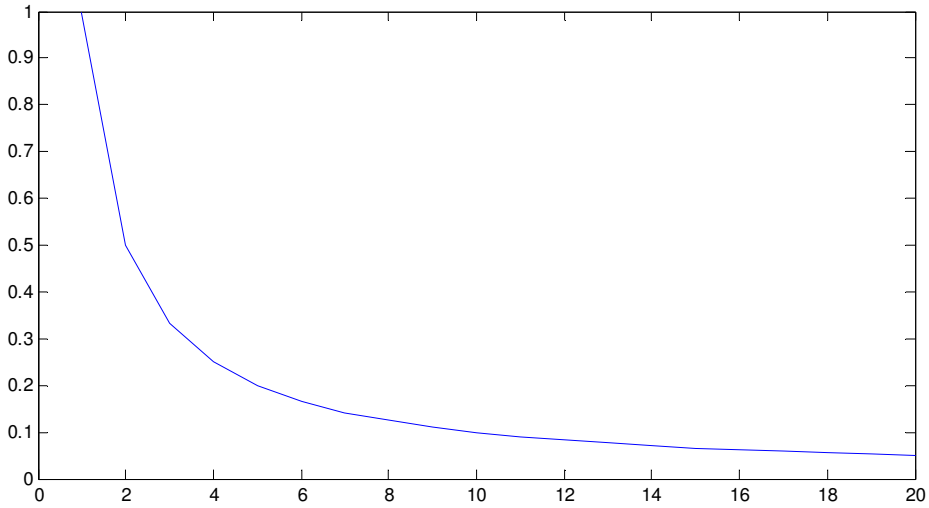


Figura 7.3

El aspecto final de la gráfica se puede modificar con un argumento opcional. Así >>*plot*(*x*,'*o*') dibuja los puntos sin unirlos con segmentos, y cada punto se señala con un círculo.

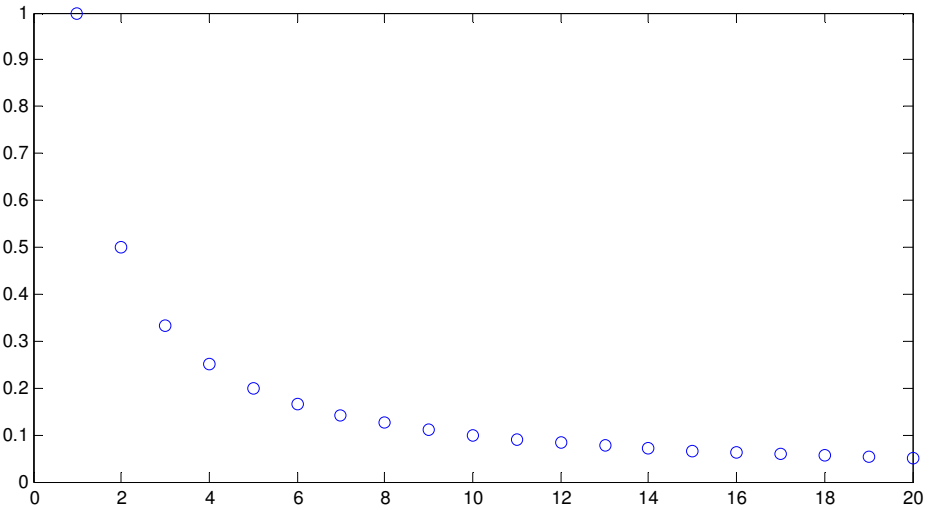


Figura 7.4

>>*plot*(*x*,'*ro-.*') los puntos se unen mediante una línea punteada de color rojo;). En general, *plot*(*x*,*S*) *e*,*l* comando dibuja *x* con *S* una cadena de caracteres (entre ') que especifica su aspecto final.

7.3. UNA APLICACIÓN

El número áureo o de oro (también llamado número dorado, razón áurea, razón dorada, media áurea, proporción áurea y divina proporción) representado por la Letra griega *φ(fi)*, es el número irracional $\varphi = \frac{1+\sqrt{5}}{2}$ =1,618033988749894848..., el ángulo dorado es $\psi = 137{,}5^{\circ}$

La mayor parte de las plantas hacen crecer sus pétalos siguiendo una secuencia en la cual cada uno de los siguientes pétalos se encuentra separado del inmediatamente anterior por el llamado *ángulo áureo*, el cual desde la época de los griegos era considerado como la razón o proporción "perfecta".

El ángulo áureo no puede ser escrito como fracción simple de una vuelta, y su valor es aproximadamente 137.5°.

La ventaja de esta forma de crecimiento con el ángulo áureo es su eficiencia de espacio y de captación de la luz solar, ya que *únicamente* con éste ángulo se asegura que no haya pérdida de espacio y que no ningún pétalo obstaculice a otro en la recolección de luz solar

Mediante el *script* en *MATLAB*(aureo)

```
function aureo
n=input('Número de puntos: ');
x=0;
y=0;
r=0;
hold on;
plot(x,y,'*');
for i=1:n;
    r=r+0.1;
    x=r*cos(pi*i*137.5*2*pi/360);
    y=r*sin(pi*i*137.5*2*pi/360);
    hold on;
    plot(x,y,'*');
end
```

desde la linea de comandos, se llama

```
>>aureo
Número de

puntos: 5000
```

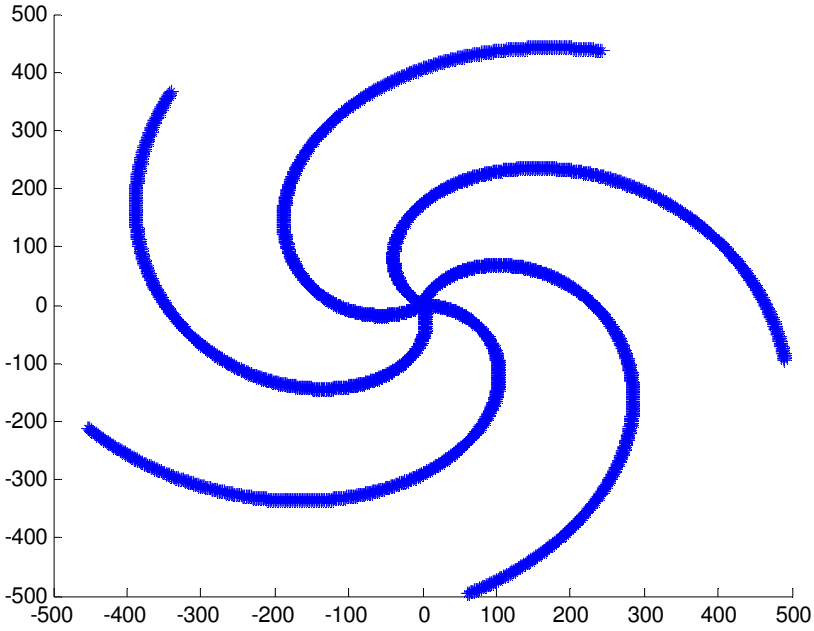


Figura 7.5

7.4. SERIES

Sea $(x_n)_{n=1}^{\infty}$ una sucesión numérica y sea S_n la sucesión definida por

$S_1 = x_1$;

$S_2 = x_1 + x_2$;

.....

$S_n = x_1 + ... + x_n$; $n = 1; 2; 3;...$

Al par de sucesiones $((x_n); (S_n))$ se le llama serie.

La sucesión (Sn) se llama sucesión de sumas parciales de la serie. A S_n se le llama n-esima suma parcial de la serie. A x_n se le llama término n-esimo de la serie.

Como notación para las series usaremos el símbolo

$\sum_{n=1}^{\infty} x_n$,o simplemente $\sum x_n$, cuando no interese saber cuál es el primer término de la serie.

Una serie se dice "convergente" si la correspondiente sucesión de sumas parciales (S_n), $S_n = \sum_{k=1}^{n} x_k, n \varepsilon N$ convergente. En caso contrario decimos que la serie no converge.

Recordar cuando se dice que una serie diverge al infinito y cuando una serie es oscilante.

En símbolos, la serie $\sum_{n=1}^{\infty} x_n$ converge si existe $\lim_n S_n = S$

Si una serie no converge, el símbolo $\sum x_n$ simplemente representa a la sucesión de sumas parciales.
Si converge, entonces también representa al número S que es el límite o suma de la serie. En tal caso debe aclararse en que término n_0 comienza la suma, es decir, escribimos

. $$\sum_{n=n_0}^{\infty} x_n = S$$

Mediante herramientas numéricas *MATLAB* se calculará APROXIMADAMENTE el valor de la suma de algunas series, previa comprobación matemática de su convergencia.
Utilizando comandos simbólicos se intentará calcular EXACTAMENTE el valor de algunas series convergentes.
Podemos sumar parcialmente una serie de forma breve y directa en la línea de comandos, utilizando el comando numérico (y simbólico) SUM.
A continuación aparecen dos variantes de un mismo procedimiento INLINE. La variante 2 es una versión más breve que la 1, y sólo consiste en fundir las líneas a, b y c de la variante 1, en la línea a de la variante 2.
Variante 1

```
>> suce=inline('1./n.^2');
>> vector=1:1000;%a
>> Sn=suce(vector);%b
>> S=sum(Sn)%c

S =
   1.6439
```

Variante 2

```
>> suce=inline('1./n.^2');
>> S=sum(suce(1:1000))%a

S =
   1.6439
```

Sean ahora las series

$$\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n^2}, \quad \sum_{n=1}^{\infty} \frac{\operatorname{sen}(4/(n+2))}{n+5}.$$

Lo resolveremos de dos maneras, una creando la función
1)sucesión término general (x_n), sucesion.m

```
function xn=sucesion(n)
xn=(-1)^(n+1)./n.^2;
desde el prompt
>> sucesion(50)
ans =
   -4.0000e-004
```

como sucesión de sumas parciales (*serie.m*)

```
function sn=serie(n,n0)
sn=0;
for k=n:-1:n0
sn=sn+(-1)^(k+1)./k.^2;
```

Desde el prompt

```
>> serie(50,1)
ans =
   0.8223
```

Los programas tipo FUNCTION constituyen la forma más potente que tiene el usuario para definir funciones en *MATLAB*. El INPUT puede consistir de matrices, otras funciones, etc. Se conservan guardadas como un archivo *m* con el mismo nombre que identifica a la función-sucesión

La alternativa sería usando funciones en *línea(inline)*

Sucesión x_n

```
>>xn=inline('sin(4./(n+2))./(n+5)','n');
```

Serie (S_n)

```
>>sn=inline('sum(sin(4./((1:n)+2))./((1:n)+5))','n')
```

Esta modalidad, en principio, debe limitarse a las funciones elementales.

Sin embargo, la arquitectura matricial del *MATLAB* permite estructurar en una sola línea la definición de S_n. Notar que la cláusula 1: n, que se debe encerrar entre paréntesis, también puede emplearse en el código FUNCTION anterior.

Sería entonces, desde la línea de comandos

```
>>sn=inline('sum(sin(4./((1:n)+2))./((1:n)+5))','n')
sn =
     Inline function:
     sn(n) = sum(sin(4./((1:n)+2))./((1:n)+5))
>> sn(50)
ans =
   0.8623
```

Además, Las sucesiones (funciones) en *MATLAB* también pueden crearse con identificador propio, como cadenas de caracteres (*string*) o expresiones formadas con variables declaradas como simbólicas (*sym*).

```
>> sucstr='n/(n^3+3*n+2)'; % xn STRING
>> n=200; valor=eval(sucstr)
valor =
   2.4998e-005

>> syms n, sucsym=n/(n^3+3*n+2); % xn SIMBOLICA
>> h=subs(sucsym,200,n)

h =
   2.4998e-005

>>serie_n='sum(sin(4./((1:n)+2))./((1:n)+5))';%Sn STRING
```

Tener en cuenta que, a diferencia del caso STRING, con variables SYM no

podemos usar la arquitectura matricial del *MATLAB* para estudiar el error de las aproximaciones, por ejemplo el criterio del error relativo es pobre $\left|x_{n+1}/S_n\right|$, sea el caso de 1/n, serie armónica que sabemos diverge

```
>> error=abs((1/1000001)/sum(1./(1:1000000)))
error =
   6.9479e-008( muy pequeño y la serie es divergente)
```

Acudimos a herramientas simbólicas

Comando *MATLAB* para calcular sumas finitas e infinitas con exactitud absoluta: *symsum*.

Sea la suma de las m primeros números naturales

```
>> syms m % esta declaracion es necesaria
>> symsum(m,1,m)

ans =
   1/2*(m+1)^2-1/2*m-1/2
```

Esta expresión es igual a *m(m + 1)=2*

Tomemos la suma de los cuadrados de los k primeros naturales

```
> syms k;
>> symsum(k^2,1,k)
 ans =
   1/3*(k+1)^3-1/2*(k+1)^2+1/6*k+1/6
```

Si nos interesa alguno en particular

```
>> symsum(k^2,1,70)
ans =
   116795
```

Para sumas infinitas:

```
>> syms k;symsum(1/k^2,1,Inf)
ans =
   1/6*pi^2
>> symsum(1/k,1,Inf)
ans =
   Inf
```

Nota : si la serie contiene n ! debe llamarse al núcleo maple 8sintaxis distinta a matlab),ej:

```
>> maple('sum(2^n/factorial(n),n=0..inf)')
ans =
   exp(2)*gamma(inf+1,2)/gamma(inf+1)
```

otros ejemplos

```
>> syms n;symsum((n+5)/(n*(n+3)*(n+7)),1,inf)
ans =
   8657/17640
>> syms n;symsum((3*n-10)/(2*n^2*(n+4)^3),1,inf)
ans =
   18475/13824-11/16*zeta(3)-7/128*pi^2
```

```
>> maple('evalf(zeta(3))')
ans =
  1.20205690315959428539973816151115
```

Desde la línea de comandos tomemos la serie que sabemos está constituída por (4-4/3+4/5-4/7+4/9...), expresando su regla de correspondencia y calculando hasta el valor 20, como generaríamos esta suma:
Lo expresamos en dos series , *t* y *r*

```
>>t=0;for n=1:70;t=t+1/(4*n+1);end;t
t =
   1.1217
>>r=0;for n=1:70;r=r+1/(4*n-1);end;r
r =
   1.3345
```

La expresión general es 4*(1-(1/(4*n-1))+(1/(4*n+1)), *quedando*
4*(1-1.3345+1.1217)

```
ans =
   3.1488
```

CAPITULO 8:VECTORES-RECTAS

8.1. BREVIARIO DE GEOMETRÍA ANALÍTICA

Ya se ha visto como introducir vectores en *MATLAB*, por ejemplo en el espacio
V=[v1,v2,v3,...,vn] **o** *V=[v1 v2 v3 ...vn]*
Así >>*vector1*=[1,4,9,3,1/2]

```
vector1 =
   1.0000    4.0000    9.0000    3.0000    0.5000
>>sqrt(vector1)
ans =
   1.0000    2.0000    3.0000    1.7321    0.7071
```

formas de definir una variable vectorial en forma comprensiva:

Variable=[a,b]	Define el vector cuyos primeros y últimos elementos son *a* y *b*, los intermedios se diferencian en una unidad
Variable=[a.s:b]	Primer y último elementos *a* y *b*, los intermedios se diferencian en *s*
Variable=linespace[a,b,n]	Primer y último elementos *a* y *b*, y tiene en total n elementos igualmente espaciados entre sí
Variable=logespace[a,b,n]	Primer y último elementos *a* y *b*, y tiene en total n elementos en escala logarítmica igualmente espaciados entre sí

>>*vector3*=[10:30]

```
vector3 =
 Columns 1 through 19
   10   11   12   13   14   15   16   17   18   19   20   21   22   23   24   25
26   27   28
 Columns 20 through 21
        29         30
```

–representar vector columna: separar sus elementos por punto y coma,
>>*a*=[10;14;21;15]

```
a =
    10
    14
    21
    15
```

o >>*a*=(10:14);*b*=*a*'

```
b =
    10
```

11
12
13
14

Cómo seleccionar un elemento de un vector o un subconjunto de elementos?

X(n)	Da el enésimo elemento de *x*
X(a:b)	Da los elementos ubicados entre el a-esimo y el be-simo, incluyendo ambos
X(a:p:b)	Da los elementos ubicados entre el a-esimo y el be-simo, incluyendo ambos, separados de *p* en *p* unidades
X(b:-p:a)	Da los elementos ubicados entre el b-esimo y el a-esimo, incluyéndolos, separados de *p* en *p* empezando por el *b-esimo(b>a)*

>>*vector1*=(2:3:9)

vector1 =

2 5 8

(puede ir o no el paréntesis)

Equivalentemente, si lo que conocemos del vector es que la primera coordenada vale 0, la última 20 y que tiene 11 en total, se escribe:

>>*vect2=linspace*(0,20,11)

vect2 =

0 2 4 6 8 10 12 14 16 18 20

A las coordenadas de un vector se accede sin más que escribir el nombre del vector y, entre paréntesis, su índice:

>>*vect2*(3)

ans =

4

y se pueden extraer subvectores, por ejemplo:

>>*vect2*(2:5)

ans=

2 4 6 8

- Productos

Un vector fila y un vector columna de igual dimensión se pueden multiplicar en cualquier orden dando un escalar (producto interno) o una matriz (exterior), así:

>>*u* = [3; 1; 4];

>>*v* = [2 0 -1];

>>*x* = *v***u*

x = 2

o

>>X = *u***v*

>>X =

6 0 -3

2 0 -1
8 0 -4

El producto escalar también se puede efectuar con:

a = [1 2 3]; *b* = [4 5 6];

>> *dot(a,b)*

ans =

32

El producto cruz se obtiene de la forma:

>>*a* = [1 2 3];

>>*b* = [4 5 6];

>>*c* = cross(a,b)

c =

-3 6 -3

Con ambos productos se puede calcular el producto mixto

Para vectores complejos, los dos productos escalares *x'*y* and *y'*x* son conjugados complejos de cada uno y el producto *x'*x* es un real (*x'* es el transpuesto).

Para complejos:

>>*z* = [1+2*i* 3+4*i*]

>>*z'*

1-2*i*

3-4*i*

y >>*z.' es*

1+2*i*

3+4*i*

8.2. RECTAS Y PLANOS

En el caso de una recta expresada en forma paramétrica, se debe declarar el rango del parámetro.

Ejemplo:

(x ; y ; z) = (1; 1; -2)+ *t.*(1; 0 ; 2)

Se toma el parámetro ***t*** *comprendido entre* –3 *y* 9:

>>t=(-3:0.1:9); *%El* ***0.1*** *indica el incremento de* ***t*** *a partir del* –3 *hasta el* 9.

>>*x*=1+1***t;

>>*y*=1***0***t;

>>*z*=-2+2***t;

>>*plot3*(*x,y,z*)

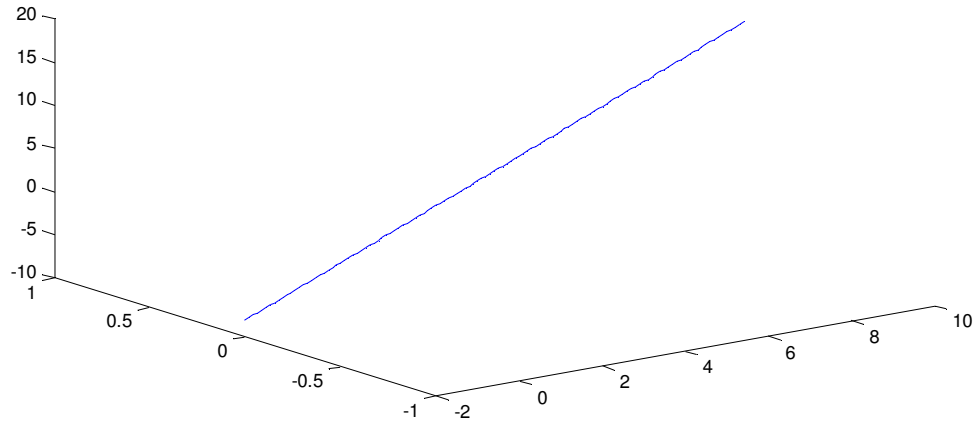


Figura 8.1

Para graficar un plano se despeja una de las variables.

Ejemplos:

a) $x+2y-z+2=0$, si se despeja $z = x +2y +2$. Se indica el rango de variación de las variables ***x*** e ***y*** empleando el comando ***meshgrid*:**

>>[x,y]=meshgrid(x min:x:x ⊿máx,y min:⊿y:y máx);

*>>z=1*x+2*y+2;*

>>plot3(x,y,z)

Entonces

>>[x,y]=meshgrid(0:1:30,0:1:30)

*>>z=1*x+2*y+2;*

>>plot3(x,y,z)

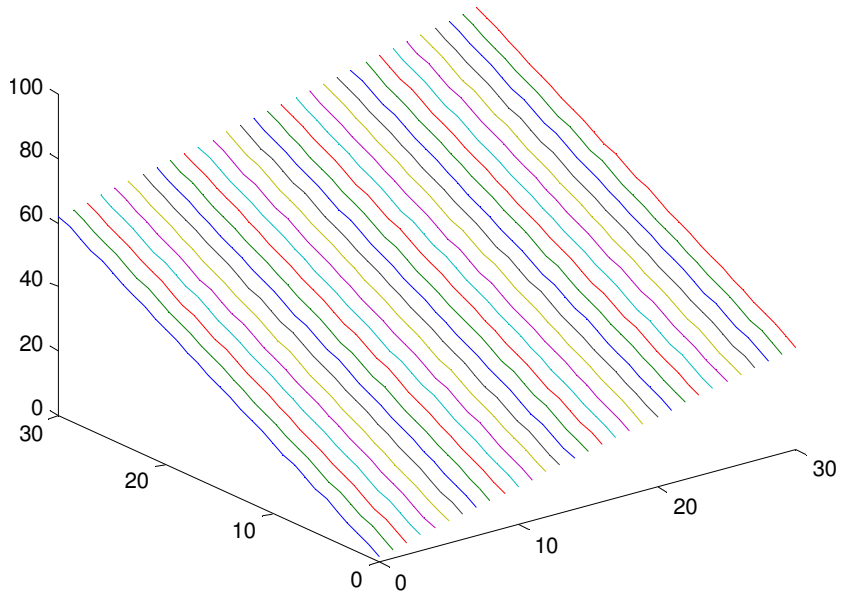


Figura 8.2

Podemos usar uso de un archivo sencillo, *planosyrectas.m*

```
==================================================
 | QUE DESEA HACER? |
 | |
 | |
 | 1 - UNA RECTA |
 | 2 - UN PLANO |
 | |
 | |
 ==================================================
INGRESE UNA OPCION: 1
OP =
     1
 ==================================================
 | SE DIBUJARA UNA RECTA DE LA FORMA Y=aX+b |
 ==================================================
 Ingrese el valor del coeficiente "a": 2
 Ingrese el valor del coeficiente "b": 4
```

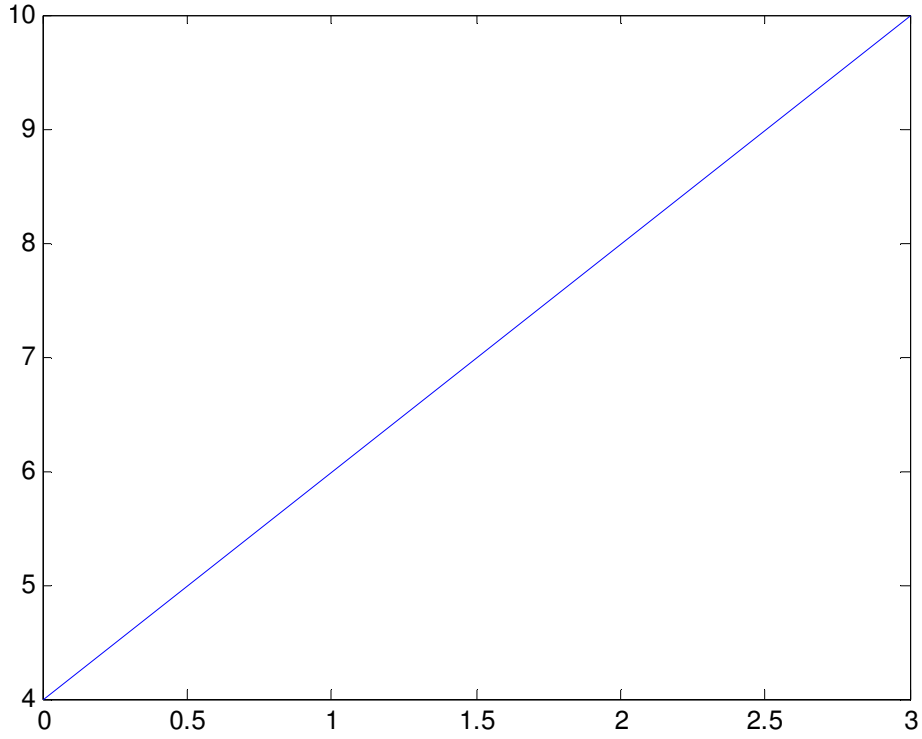


Figura 8.3

8.3. CÓNICAS

si tenemos que representar una cónica cuya ecuación tiene todos los términos, como por ejemplo $4x^2 + 3y^2 - 5xy + 4x + 4y - 3 = 0$,

```
>>ezplot('4*x^2+3*y^2-5*x*y+4*x+4*y-3')
```

Si queremos indicar nosotros el recuadro del plano donde nos interesa la gráfica, lo añadimos como argumento:

>> *ezplot('4*x^2+3*y^2-5*x*y+4*x+4*y-3',[-6 2 -1 1])*

dibuja los puntos de la gráfica que están contenidos en el rectángulo [−6, 2]× [−1, 1].

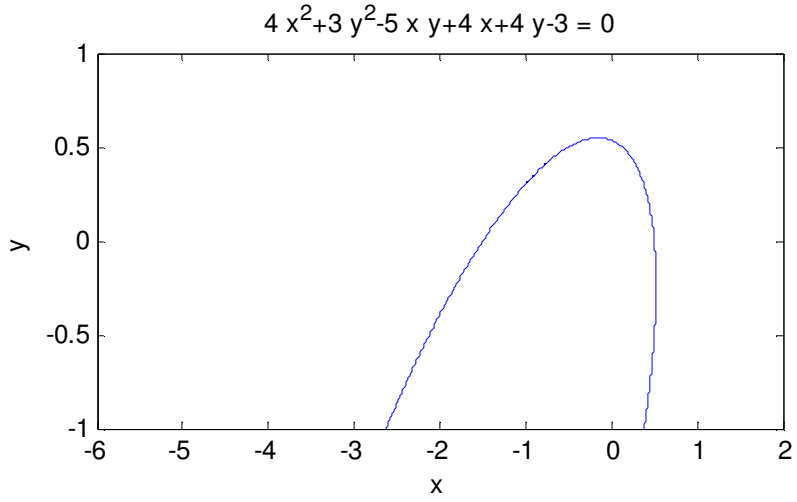


Figura 8.4

CAPITULO 9: ENUMERAMIENTOS

Para el cálculo combinatorio clásico el paquete de funciones se llama *combinat* y pertenece al Extended *Symbolic Toolbox*.
Este paquete se carga con
>>maple('with(combinat)')
Warning: Warning, the name fibonacci has been redefined
Warning: Warning, the protected name Chi has been redefined and unprotected
ans=
[Chi, bell, binomial, ..., stirling2, subsets, vectoint]
La sintaxis general será::
>>maple('funcion')

9.1. PERMUTACIONES

Sabemos que el número de permutaciones de un conjunto de n elementos son todas las ordenaciones de los elementos del *conjunto:*$P_n=n!$
Así, para hallar los ordenaciones:
>> maple('permute(2)')

ans =
[[1, 2], [2, 1]]
Para el número de permutaciones, se tiene
>> maple('numbperm(5)')

ans =
120
Para construir las ordenaciones de un conjunto arbitrario
>> maple('permute([x1,x2,x3])')

ans =
[[x1, x2, x3], [x1, x3, x2], [x2, x1, x3], [x2, x3, x1], [x3, x1, x2], [x3, x2, x1]]
Si se desea hallar las formas de disponer 3 personas en tres lugares
maple('permute([al1,al2,al3])')

ans =
[[al1, al2, al3], [al1, al3, al2], [al2, al1, al3], [al2, al3, al1], [al3, al1, al2], [al3, al2, al1]]
Tenemos 10 cajas distintas, de los cuales 3 son rojas, 2 azules y 5 verdes.
¿De cuántas formas podemos colocarlos en una estantería con un
espacio vacío para 10 cajas?
>>maple('numbperm(10)')
ans=
3628800
Y si queremos que las dos más grandes estén en los extremos?

>>*maple('2*numbperm(8)')*

ans=

80640

Y si queremos que esté agrupados por colores?

>>*maple('numbperm(3)*numbperm(3)*numbperm(2)*numbperm(5)')*

ans=

8640

9.1.1. Permutaciones con repetición

Las permutaciones con repetición son las disposiciones ordenadas de n elementos (no todos iguales) de modo que del primer tipo hay n_1, del segundo tipo n_2, ... y, del *k-esimo* tipo hay n_k. El número de permutaciones con repetición de un conjunto de *n* elementos de modo que del primer tipo hay n_1, del segundo tipo n_2, ... y, del *k-esimo* tipo hay n_k es:

$PR^{n1n2...nk}_{n} = n!/n1!\ n2!\ ...\ nk!$

permute([*a1, a1, .n.1., a1, a2, a2, .n.1., a2, . . . , ak, ak, .n.k., ak*]) calcula las permutaciones con repetición de los elementos $a_1, a_1, \ldots, a_k$ de modo que a_1 se repite

n_1 veces, a_2 se repite n_2, ... con $n_1 + n_2 + \ldots n_k = n$

>> *maple('permute*([1,1,2])')

ans =

[[1, 1, 2], [1, 2, 1], [2, 1, 1]]

Ahora para hallar el número PR

>> *maple('multinomial*(3,2,1)')

ans =

3

Ejemplo: En una oficina hay que distribuir 12 administrativos en 3 despachos , no más de 4 por despacho. ¿De cuántas formas puede hacerse? Lamando *di* al despacho número *i*, para todo *i = 1, 2, 3, 4*. Suponiendo los 12 agentes ordenados, una distribución de los despachos es una cadena de 12 caracteres dónde cada *di* aparece 4 veces.

>>*maple('multinomial*(12,4,4,4)')

ans=

34650

9.2. VARIACIONES SIN REPETICIÓN

Las variaciones sin repetición de orden *r* de un conjunto de n elementos son todas las ordenaciones de tamaño *r* de elementos del conjunto, sin que existan repeticiones. El número de variaciones sin repetición de n elementos de orden *r* es:

$V_{n,r} = n!/(n - r)!$

permute(n,r) construye las variaciones sin repetición de tamaño *r* del conjunto {1, 2, . . . , n}

>>*maple('permute*(4,2)')

ans=

[[1, 2], [1, 3], [1, 4], [2, 1], [2, 3], [2, 4], [3, 1], [3, 2],
[3, 4], [4, 1], [4, 2], [4, 3]]

numbperm(n,r) da el número de variaciones sin repetición de tamaño *r* de un conjunto de n elementos.

>> *maple('numbperm*(4,2)')

ans=

12

permute([a1,a2,. . . ,an],r) construye las variaciones sin repetición de tamaño r del conjunto {a1, a2, . . . , an}

>> *maple('permute([a1,a2,a3],2)')*

ans =

[[a1, a2], [a1, a3], [a2, a1], [a2, a3], [a3, a1], [a3, a2]]

Dado el conjunto de númeross A = {0,1, 2, 3, 4, 5, 6, 7, 8, 9}.

(1) ¿Cuántos números distintos de 6 números distintos pueden generarse con los números del conjunto ?

>>*maple('numbperm*(10,6)')

ans=

151200

(2) ¿Cuántos de los números del inciso anterior tienen al 3? >>

*maple('*6*numbperm*(9,5)')

ans =

90720

9.2.1. Variaciones con repetición

Una variación con repetición de orden *r* de un conjunto de *n* elementos es una sucesión ordenada de tamaño *r* de elementos del conjunto, en donde los elementos pueden repetirse. El número de variaciones con repetición de orden *r* de un conjunto de *n* elementos es:

$VR_{n,r} = n^r$

permute([a1, a1, . r. ., a1, a2, a2, . r. ., a2, . . . , an, an, . r. ., an], r) calcula las variaciones

con repetición de orden *r* del conjunto $\{a_1, a_2, \ldots, a_n\}$

>> >> *maple('permute(*[1,1,2,2],2)')

ans =

[[1, 1], [1, 2], [2, 1], [2, 2]

n ^ r calcula $VR_{n,r} = nr$

Por ejemplo, ¿cuántos números hexadecimales hay de tres cifras, ninguna de ellas nulas? Hay que calcular $VR_{15,3}$:

>>15^3

ans=

3375

9.3. COMBINACIONES

Las permutaciones y variaciones de un conjunto se diferencian por el orden de los elementos. Sin embargo, en las combinaciones no importa el orden: dos combinaciones son diferentes si, y sólo si, tienen algún elemento distinto.

Combinaciones sin repetición.

Una combinación de orden r de un conjunto de n elementos es un subconjunto de tamaño r del conjunto.
El número de combinaciones sin repetición de tamaño *r* de un conjunto de n elementos es:

$$C_{n,r} = \binom{n}{r}$$

choose(n,r) construye las combinaciones sin repetición de tamaño r del conjunto {1, 2, . . . , n}

```
 >> maple('choose(4,3)')
ans =
   [[1, 2, 3], [1, 2, 4], [1, 3, 4], [2, 3,4])
```

binomial(n,r) calcula $C_{n,r}$

```
>>maple('binomial(4,3)')
ans=
   4
```

choose([a1,a2,. . . ,an],r) construye las combinaciones sin repetición de tamaño *r* del conjunto *{a1, a2, . . . , an}*

```
>>maple('choose([a1,a2,a3],2)')
ans=
   [[a1, a2], [a1, a3], [a2, a3]]
```

Ejemplo, determinar el número de manos de póker distintas (5cartas) que pueden formarse con una baraja de 52 naipes.

```
>>maple('binomial(52,5)')
ans=
   2598960
```

¿Cuántas manos contienen exactamente 3 ases?

```
>>maple('binomial(4,3)*binomial(48,2)')
ans=
   4512
```

9.3.1. Combinaciones con repetición

Una combinación con repetición de orden r de un conjunto de n elementos es una lista de tamaño r de elementos del conjunto que pueden repetirse. El número de combinaciones con repetición de orden r de un conjunto de n elementos es:

$$CR_{n,r} = \binom{n+r-1}{r}$$

choose([a1, a1, . r. ., a1, a2, a2, . r. ., a2, . . . , an, an, . r. ., an], r) construye las combinaciones con repetición de tamaño *r* del conjunto $\{a_1, a_2, \ldots, a_n\}$

```
>>maple('choose([a1,a1,a2,a2,a3,a3],2)')
ans=
  [[a1, a1], [a1, a2], [a1, a3], [a2, a2], [a2, a3], [a3, a3]]
```

binomial(n+r-1,r) calcula $CR_{n,r}$

```
>>maple('binomial(4,2)')
ans=
  6
```

Ejemplo , en un comercio hay microondas de cuatro marcas diferentes, de cuántas formas se pueden elegir 5 microondas?

Debemos hallar $CR_{4,5}$, donde *n*=4,*r*=5

```
>> maple('binomial(8,5)')
ans =
   56
```

Alternativamente se puede manejar con el empleo de la función factorial para las expresiones combinatorias, así si se quieren hallar las combinaciones sin repetición de 26 elementos tomados de a 12:

```
>>factorial(26)/(factorial(12)*factorial(26-12))
ans =
   9657700
```

$$\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n^2}, \quad \sum_{n=1}^{\infty} \frac{\operatorname{sen}(4/(n+2))}{n+5}.$$