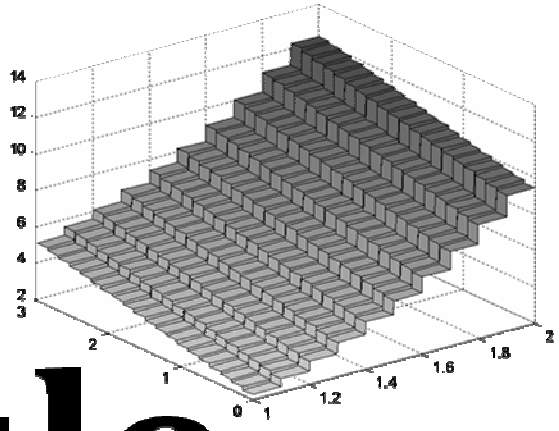
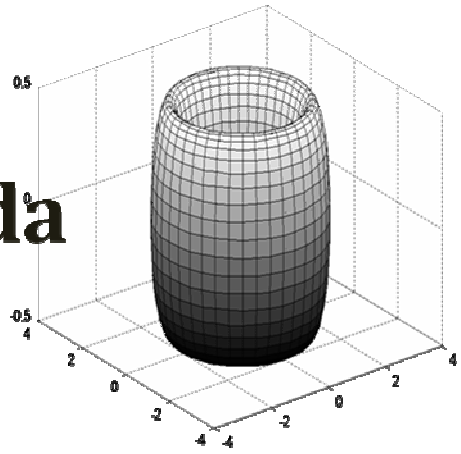


Cálculo
Con MATLAB



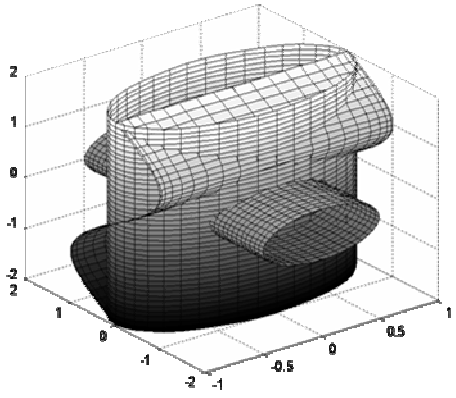
Mario E. Matiauda

$$\iint f(x,y)dxdy$$

Año 2010

$$\frac{1}{2\pi}\int_0^{\pi}\frac{sen\,(t)}{t}\,dt\,-1\approx 0.0895$$

$$\bar{y}=\frac{1}{m}\int_C y\rho(s)ds$$



$$\iiint_G \rho(x,y,z)dV(x,y,z)=\int_o^{\pi}\int_0^{2\pi}\int_2^4[1+\cos(r\cos\theta sen\varphi)]r^2 sen\varphi dr d\theta d\varphi$$

$$b_k=\frac{10}{4\pi^2}\left[-wt.\frac{\cos(kwt)}{k}+\int_0^{2\pi}\frac{\cos(kwt)}{k}d(wt)\right]_0^{2\pi}=$$

$$b_k-\frac{10}{4\pi^2}\left[wt.\frac{\cos(kwt)}{k}-\frac{sen(kwt)}{k^2}\right]_0^{2\pi}=$$

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 978-950-579-150-7

Impreso en Argentina

Matiauda, Mario Eugenio
Cálculo con Matlab. - 1a ed. - Posadas : EDUNAM - Editorial Universitaria de la Universidad Nacional de Misiones, 2010.
196 p. ; 30x21 cm.
ISBN 978-950-579-150-7
1. Cálculos Matemáticos. 2. Enseñanza Superior. I. Título
CDD 513.071 1

Fecha de catalogación: 22/02/2010.

EL AUTOR

MATIAUDA, Mario Eugenio
Título grado: Ingeniero Químico. FCEQyN-UNaM
Tititulo 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 inició 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.

ÍNDICE

PRÓLOGO

No sería honesto y justo denominar libro, obvio al ver el desarrollo de los ítems, al sucesivo de estas líneas, tampoco cuaderno de Cátedra resultó apropiado pues difícilmente responda a un analítico determinado, de allí la tentación y el facto de mencionarlo "impreso".

Quizá la misma disquisición apareció sobre el título y con disculpas si suena pomposo o atrevido, quedó Cálculo y Matlab.

La idea, al igual que Matlab, era generar un espacio abierto al potencial sin cotas de los temas centrales de cualquier curso sencillo de Cálculo, en el ámbito que ofrecen las herramientas de Matlab.

Lejos estamos de apuntar a la programación como práctica de de diseño y sobre los cimientos de las teorías del diseño axiomático, de hipótesis centrales como menor información e independencia extrema.

Simplemente, alejándonos de la discusión central del empleo de soft en los cursos básicos, se estima un complemento para ellos, sin explayarnos sobre los aspectos teóricos salvo en los dos últimos capítulos donde se incluye una breve discusión.

CAPITULO 1 - INTRODUCCION

1.1 QUÉ ES MATLAB?

*MATLAB = '**MAT**rix **LAB**oratory' (LABORATORIO DE MATRICES).*

MATLAB es un medio computacional técnico, con un gran *desempeño* para el *cálculo* numérico computacional y de visualización.

MATLAB integra *análisis* numérico, matrices, procesamiento de señales y *gráficas*, todo esto en un *ambiente* donde los *problemas* y *soluciones* son expresados tal como se escriben matemáticamente.

Escrito inicialmente como auxiliar en la *programación* de cálculo con matrices. *MATLAB* fue escrito originalmente en *Fortran*, actualmente está escrito en *lenguaje* C.

MATLAB es un *lenguaje de programación* amigable al usuario con *características* más avanzadas y mucho más fáciles de usar que los *lenguajes de programación* como *Basic*, *Pascal* o C.

MATLAB cuenta con paquetes de *funciones* especializadas llamadas toolboxes.

1.2 INTERFAZ DE MATLAB

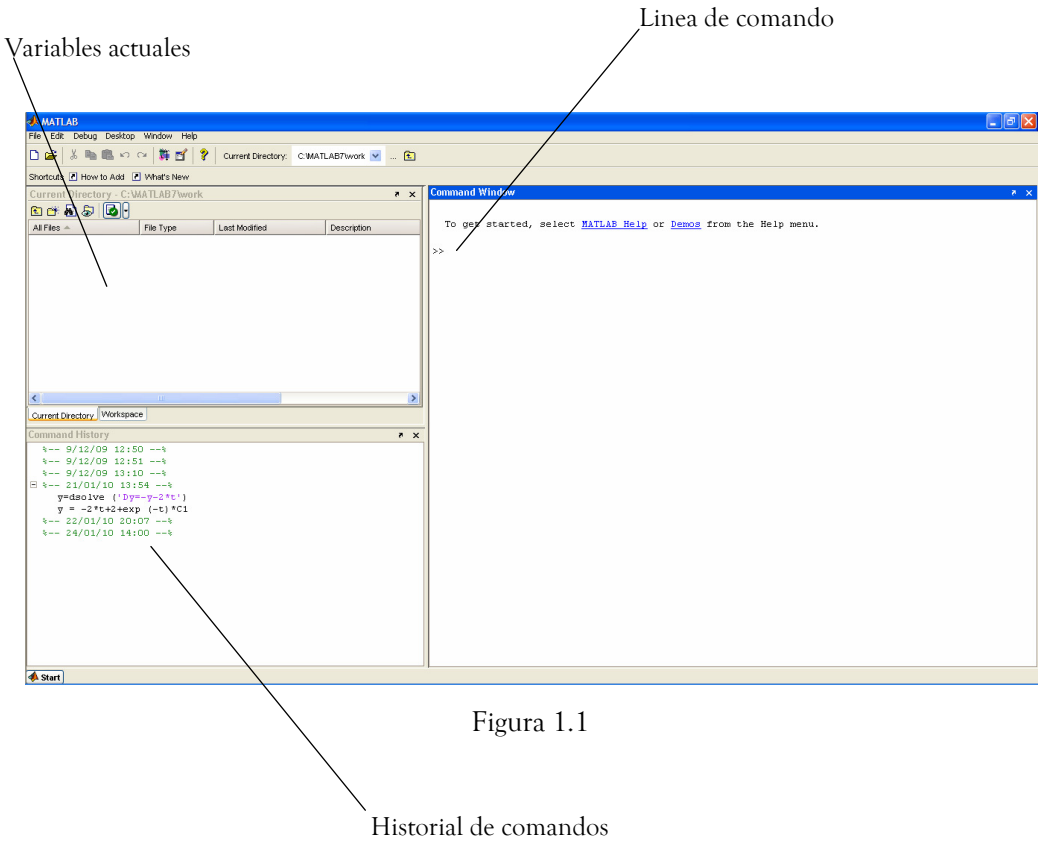


Figura 1.1

Al abrir el programa, se puede encontrar una interfaz como la de la figura1.1, 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 "*MATLAB help*" (Figura 1.2). Este segundo modo de ayuda resulta bastante más potente y eficaz que la primera añadiendo en muchos casos ejemplos de utilización.

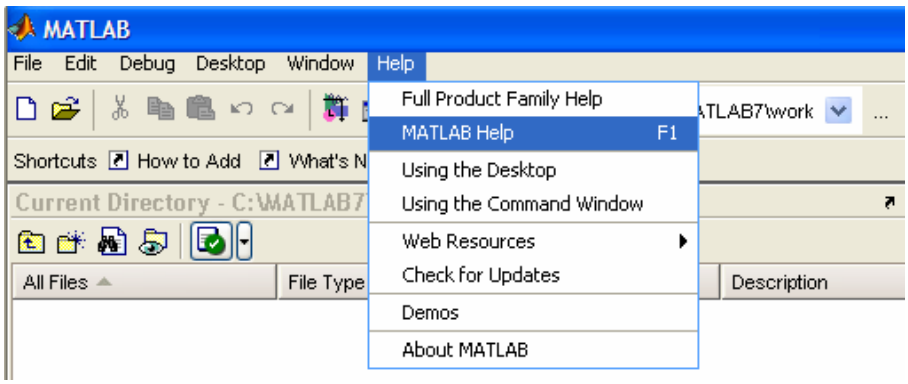


Figura 1.2

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 1.3.

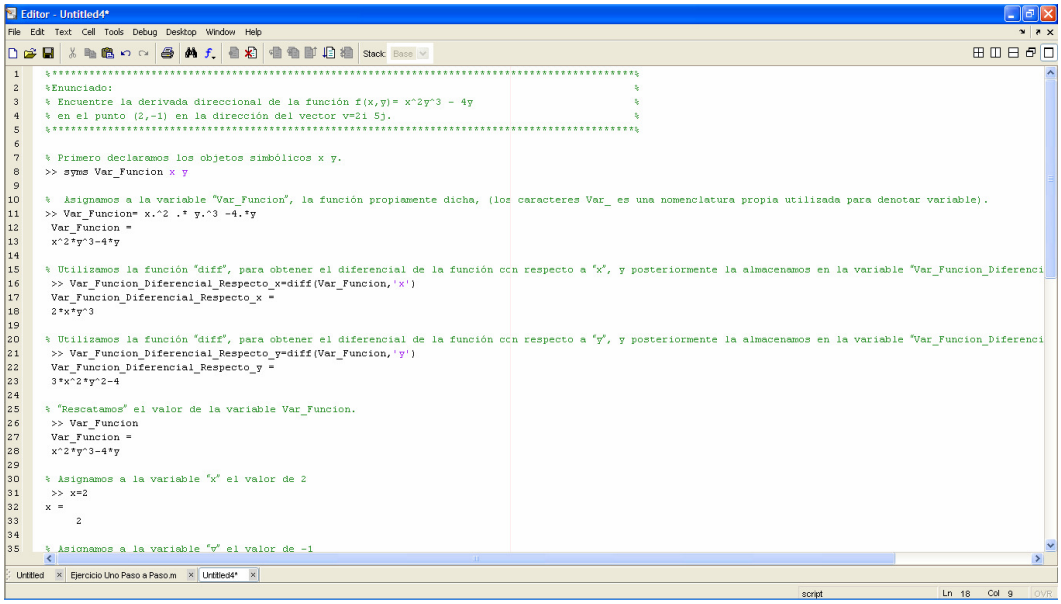


Figura 1.3

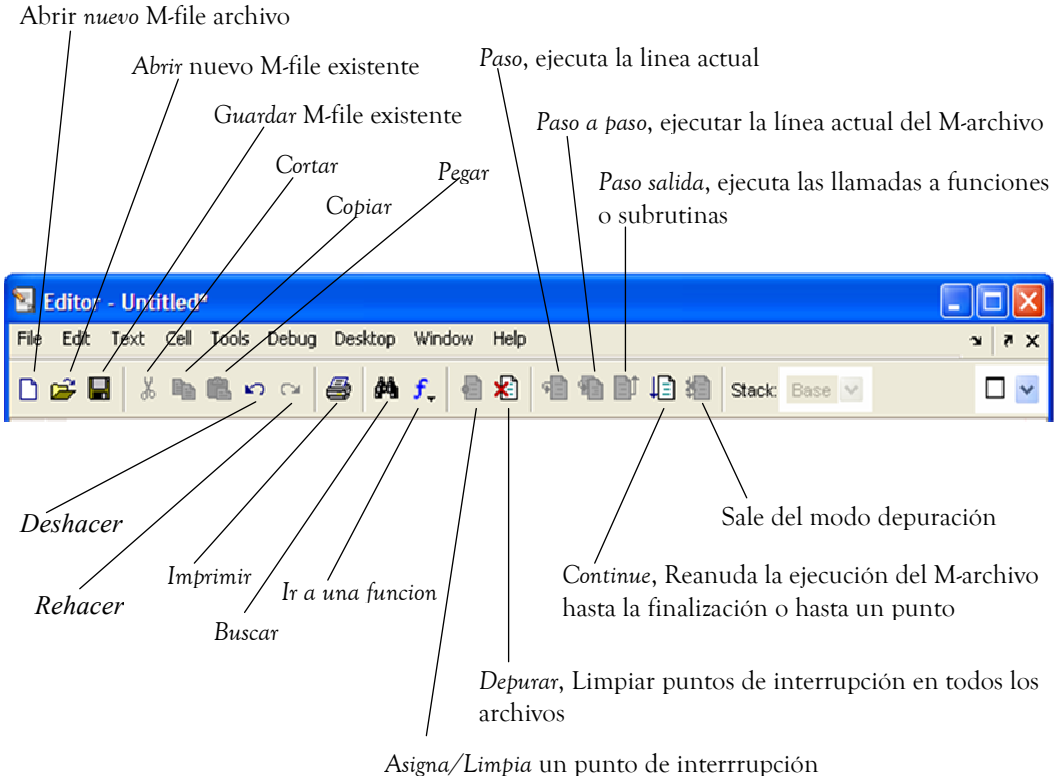


Figura 1.4

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.

1.3 INSTRUCCIONES BASICAS

Las operaciones se evalúan por orden de prioridad: primero las potencias, después las multiplicaciones y divisiones y, finalmente, las sumas y restas. Las operaciones de igual prioridad se evalúan de izquierda a derecha:

Los paréntesis modifican la jerarquía., o sea.

```
>>2+3/5+7*6+2^7          >>2+3/(5+7*6)+2^7
ans=                     ans=
  172.6000                 130.0638
```

Para escribir o manejar números (formatos)

```
>> format short e
>> 230/6
ans =
  3.8333e+001
```

Como escribimos según la presentación científica:

sea $D=1.10^{-5:}$
$^{>>}$1.0000*e*-005
En cambio
$3.25e^5$ será:

```
>>3.25*exp(5)
ans =
  4.8234e+002
```

1.4. FUNCIONES

A) Trigonométricas

sin(0.5): Seno de (0.5)

Así mismo

cos(x) tan(x)
asin(x) acos(x) atan(x) inversa
sinh(x) cosh(x) tanh(x) hiperbólica
asinh(x) acosh(x) atanh(x) Inversa- Hiperbólica

atan 2 (x,y) Inversa de la tangente en los cuatro cuadrantes.

B) Logaritmos

log(0.5): Logaritmo natural

log_{10} *(x)*: Logaritmo decimal.

C) Funciones matemáticas especiales

abs(-3) Valor absoluto o magnitud de un número complejo

ceil(123.123123*)* Redondea hacia más infinito

floor(x) Redondea hacia menos infinito

fix(x) Redondea hacia cero

round(x) Redondea hacia el entero más próximo

imag(30 - 5*j)* Parte imaginaria de un número complejo

real(x) Parte real de un número complejo

angle(x) Angulo de un número complejo

conj(x) Complejo conjugado

sign(-5) Función signo : Devuelve el signo del argumento

(1 si es positivo, -1 si es negativo)

exp(1 *)* Exponencial : $e(x)$

```
>>exp(2.4)
ans =
   11.0232
```

Si se quiere la exponencial de una función:

```
>>syms y
>>exp(2*y)
```

>>*sqrt(2)* Raíz cuadrada

1.5. CREACION DE ARCHIVOS DE EXTENSION .M. TIPOS DE FUNCIONES

Son archivos con la terminación .m que *MATLAB* utiliza para trabajar con funciones y scripts. Un script es una secuencia de comandos que se pueden ejecutar a menudo y que se pueden guardar en un archivo de extensión .m para no tener que escribirlos de nuevo. La mayoría de las funciones de *MATLAB* están en realidad en archivos .m, y se pueden visualizar escribiendo *type xxx*, donde *xxx* es el nombre de la función.

Para elaborar propios scripts o funciones, deberán generarse un nuevo archivo de texto con el nombre que se quiera, siempre y cuando termine en *.m*, para que *MATLAB* lo reconozca. Este tipo de archivos se pueden crear, editar y guardar con cualquier editor de textos, como *emacs, EZ,* o *vi*. Un archivo de script es

simplemente una lista de comandos de *MATLAB*. Cuando se escribe el nombre del archivo en el *prompt* de *MATLAB*, su contenido se ejecuta. Para que un archivo *.m* sea una función, tiene que empezar por la palabra *function* seguida de las variables de salida entre paréntesis, el nombre de la función y las variables de entrada.

Ejemplo

```
function [x1,x2]=raices(a,b,c)
x1=-b+sqrt(b^2-4*a*c)/2*a;
x2=b+sqrt(b^2-4*a*c)/2*a;
end
```

la guardamos como *raices.m*

Desde el prompt:

```
raices(0,2,4)
ans =
    -2
function [V, D, r]=properties(A)
% Esta función calcula el rango, autovalores y autovectores de A
[m, n]=size(A);
if m==n
[V, D]=eig(A); r=rank(A); else
disp('Error: La matriz debe ser cuadrada');
end
```

Aquí, la función toma la matriz *A* como entrada y sólo muestra dos matrices y el rango como salida. El % se utiliza para marcar un comentario. La función comprueba si la matriz de entrada es cuadrada y luego calcula el rango, los autovalores y autovectores de la matriz *A*. Al escribir *properties(A)* sólo se obtendrá la primera salida, *V*, la matriz de autovectores. Es necesario escribir *[V, D, r]=properties(A)* para obtener las tres salidas.
Nos detendremos en las posibilidades de definición de funciones por el usuario.
Una función, en general, tiene argumentos de entrada (o parámetros) y las variables de salida (o parámetros) que pueden ser escalares, vectores, o matrices de cualquier tamaño. Pueden ser cualquier número de entrada y salida de parámetros, incluyendo el cero. Los cálculos realizados dentro de una función suelen hacer uso de los parámetros de entrada, y los resultados de la los cálculos son transferidos fuera de la función por los parámetros de salida.
Para escribirla function *[OUT1, OUT2,...] = function (IN_1, IN2,...)* donde "function" es el nombre de la función definida por el usuario, *IN1, IN2,...* son los parámetros de entrada, y *OUT1, OUT2,...* son los parámetros de salida.

Los paréntesis son necesarios, incluso si la función no tiene entrada de parámetros:
función *[OUT1, OUT2,...] = function()*

Si sólo hay un parámetro de salida, los corchetes pueden ser omitidos:
function *out = function(IN1, IN2,...)*

Si no hay un parámetro de salida en todos, entonces se les llama vacío
funciones. La función de línea de definición se escribe como:
function *función(IN1, IN2,...)*

Para que la función trabaje, los argumentos de salida deben ser asignados con valores en el cuerpo de la función.
Una función *anónima* es un simple, típico de una sola línea, definida por el usuario, función que se define y por escrito en el código de computadora (no en el un archivo aparte) y luego se utiliza en el código. Las funciones anónimas pueden ser definidas en cualquier parte de *MATLAB* (en la ventana de comandos, en los *scripts*, y dentro de funciones definidas por el usuario).
Las funciones anónimas se han introducido en *MATLAB* 7. Ellas tienen varias ventajas sobre la función en línea (*'inline'*).
En este momento las funciones anónimas y en línea pueden ser utilizadas, pero función en línea se irá reduciendo progresivamente.
Una función *anónima* se crea escribiendo la siguiente declaración:
function = @(var1, var2,...) expresión donde '*function*' es el nombre de la función anónima, *'var1', 'var2'*, etc son una lista separada por comas de los argumentos de la función, y "*expresión*" es una sola expresión matemática involucrando las variables. La expresión puede incluir variables predefinidas que ya están definidos cuando se define la función anónima. Por ejemplo, si tres variables *a, b, c,* y los valores han sido asignados, a continuación, pueden ser utilizados en la expresión de la función anónima:

```
a = 3; b = 4; c = 5;
parabola = @(x) (a * x + b) * x + c
```

Es importante señalar que *MATLAB* captura los valores de las variables predefinidas cuando se define la función anónima. Esto significa que si posteriormente los nuevos valores son asignados a la predefinida variable, la función anónima no se cambia.

Así, por ejemplo,

```
>> F = @ (x) exp(x.^2)./sqrt(x.^2+5)
F =
@(x) exp(x.^2). / sqrt(x.^2+5)
>> p = F(2)
p = 18.1994
>> Pvec = F([1 0.5 2]))
Pvec = 1.1097 0.5604 18.1994
```

Similar a la función anónima, función en *línea*(*inline*) es una simple función definida por el usuario, sin crear una función separada en archivo M.
Una función en línea se ha creado con el comando en línea de acuerdo para el siguiente formato:
name = inline('expresión matemática escribe como una cadena')
Un ejemplo simple, es:
CUBO = inline('X^3')

que calcula la potencia cúbica de una entrada de escalar, vector o matriz.
Para una función en *línea*,

i. La expresión matemática puede tener una o varias variables independientes
ii. Ya que los argumentos no se especifican en el formulario de arriba de la función inline, *MATLAB* dispone de los argumentos en orden alfabético. Hay una forma alternativa que permite a los argumentos que se especifiquen de forma explícita.
iii. Cualquier nombre, excepto *i* y *j* se puede utilizar para los nombres de los variables independientes en la expresión.
iv. La expresión no puede incluir ninguna variable asignada previamente
v. La expresión puede incluir cualquier incorporado o funciones definidas por el usuario.
vi. Una vez que la función inline se define, puede ser utilizada escribiendo su nombre y una lista de valores para sus argumentos separado por comas con paréntesis.
vii. Una función en línea puede ser utilizado como un argumento en otras funciones.

```
>> F = inline('exp(x.^2). /sqrt(x.^2+5)')
F =
Inline function:
F(x) = exp(x.^2)./sqrt(x.^2+5)
>> F(2)
ans =
   18.1994
>> FA([1 0.5 2]))
ans =
  1.1097 0.5604 18.1994
Con dos argumentos:
>> H = inline('y^2+2*x*y+3*x^2')
H =
Inline function:
H(x,y) = y^2+2*x*y+3*x^2
```

1.5.1 Llevar un diario de trabajo

El comando *diary('file')* ordena a *MATLAB* que grabe todas la operaciones que se realizan en su ventana y que guarde los resultados en el archivo de texto de nombre *'file'*. Al escribir *diary on* y *diary off* activa y desactiva la grabación. Los archivos del diario se pueden visualizar mediante un editor de textos, o se pueden imprimir con *lpr* en unix. En *MATLAB* se pueden visualizar utilizando el comando type *file*.

1.6. ARREGLOS (ARRAYS) O VECTORES

Si se desea calcular el *seno* de "0 *a* 1" con incrementos de 0.25, se pueden capturar *los valores* y después llamar el seno de la *función*.
Seno de 0 *a* 1 con incrementos de 0.25
>>*x = [*0, 0.25, 0.5, 0.75,1*]*
Se pueden omitir las comas cuando se capturan los números.
Con los números capturados, se obtiene el seno de la variable *x* escribiendo simplemente:
>>*sin(x)*
MATLAB opera en radianes, donde 2π = 360 grados.
Ahora se requiere obtener el coseno de cero a uno con incrementos de 0.01; lo que equivale a capturar 101 elementos.
Para evitar capturarlos a mano, *MATLAB* nos permite crear un vector de la siguiente manera:
Variable =(Valor inicial: Con incrementos de: Valor final)
x =(0: 0.01: 1*)*
cos(x) Ahora se puede obtener el coseno de la variable R.

Hagamos el siguiente vector:

>>*y* = (0: 1: 10*)*

Si queremos saber cuál es el cuarto elemento del vector ponemos:

>>*y*(4*)*

Si nos interesan los elementos 5 al 10:

>>*y*(5: 10*)*

Otras opciones son:

>>*y*(1: 2: 9*)* Toma los elementos del 1 al 9 con incrementos de 2
>>*y*(*[*1, 3, 7,10]*)* Toma los elementos 1, 3, 7 *y* 10 del array

Para introducir una matriz, se define por filas separándolas por *intro o punto y coma*. Por ejemplo, el comando siguiente define la matriz A de dimensión 2x3

>> A = *[*3 4 1; 5 -3 0*]*
A =
  3 4 1
  5 -3 0

1.6.1 Vectores y sus operaciones

Las operaciones de adición, diferencia y producto por escalares, se realizan mediante una sintaxis simple.
Consideremos los vectores ***x*** =(-1,3 -2), ***y*** =(0, -2, -1) y el escalar 2.
>> *z = x + y*
z =
  -1 1 -3

```
>> w = x - y
w =
    -1 5 -1
>> u = 2*x
u =
   -2 6 -4
```

Geométricamente:

Sean los vectores **x** =(1, 2, 0), **y** =(0, 3, 2) y $\boldsymbol{x} + \boldsymbol{y}$ =(1, 5, 2)

```
>> x = [1 2 0]; y = [0 3 2]; u= [1 5 2]; %u=x+y
>> plot3(x,y, u)
```

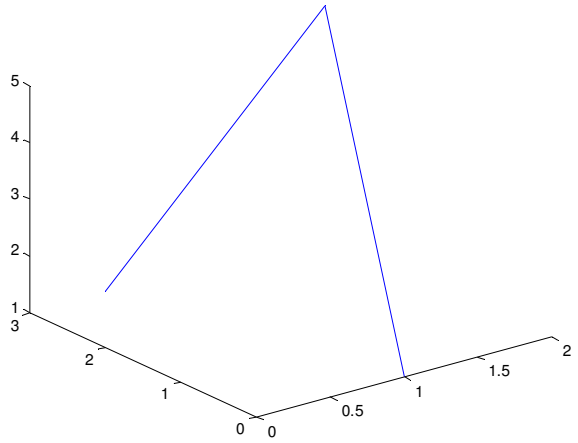


Figura 1.5

Para la norma

```
>> norm(x)
ans =
   2.2361
```

Para el producto punto, el producto vectorial y el ángulo entre vectores, se procede de la forma siguiente:

Ejemplo

```
>> dot(x,y)
ans =
  6
>> cross([1,2,0],[0,3,2])
ans =
  4 -2 3
>> theta = acos(dot(x,y)/(norm(x)*norm(y)))
theta =
   0.7314
>> 360*theta/(2*pi)
ans =
   41.9088
```

Para obtener el área del paralelogramo cuyos lados adyacentes son los vectores ***x***

y ***y***.

```
>> area = norm(x)*norm(y)*(sin(theta))^2
area =
    3.5970
```

1.7 OPERACIONES CON MATRICES

Para multiplicar una matriz por un vector y para el producto de matrices, se procede de la forma siguiente:

$$A=\begin{pmatrix} 0 & 4 & 6 & 1 \\ 2 & -2 & 0 & 1 \\ -2 & 1 & 3 & -4 \\ 3 & -1 & -3 & 6 \end{pmatrix} \quad \text{y} \quad x=\begin{pmatrix} 3 \\ 2 \\ 1 \\ -1 \end{pmatrix}$$

```
    1
    3
   -2
```

Consideremos las matrices

```
A = [5 -2 0; 1 -1 4; 3 -3 2; 0 -5 -1] y B = [-2 3 4 6 7 -1; 0 9 -2 1 0 3; 1 -1 0 3 2 7]
>> C = A*B
C =
    -10 -3 24 28 35 -11
     2 -10 6 17 15 24
    -4 -20 18 21 25 2
    -1 -44 10 -8 -2 -22
>> D = B*A
??? Error using ==> *
Inner matrix dimensions must agree
     200            -0.0000 - 0.0000i
```

CAPITULO 2 - GRÁFICAS CON MATLAB

A costo de ser reiterativos, incluimos en este apartado nociones generales sobre las herramientas de graficación de *MATLAB*, pese a que la misma será de uso recursivo en las distintas aplicaciones que se irán desarrollando.
El comando más simple es *plot(x,y)*, que utiliza dos vectores, *x* e *y*, de la misma longitud. Éste dibujará los puntos *(xi, yi)* y los unirá mediante rectas continuas.
Si no se le da ningún vector *x*, *MATLAB* asume que *x(i) = i*. A continuación *plot(y)* recibe el mismo espacio en el eje de las *x*: los puntos son *(i, y (i))*.
Se pueden cambiar el tipo y color de la línea que une los puntos mediante un tercer argumento. Si este argumento no existe, *MATLAB* dibuja por defecto una línea continua de color negro "-". Introduciendo *help plot* se obtienen muchas opciones, aquí sólo indicamos unas pocas:
MATLAB 5: *plot(x,y,'r+ :')* dibuja *r* en rojo, los puntos en forma de + y unidos por línea de puntos.
MATLAB 4: *plot(x,y,' -')* dibuja una línea discontinua y *plot(x,y,' .')*, una línea de puntos.
Se pueden omitir las líneas y representar sólo los puntos discretos de distintas formas:
plot(x,y,' o') dibuja círculos. Otras opciones son *'+', 'x' o '*'*.
Para obtener dos gráficas en los mismos ejes, utilizar *plot(x,y,X,Y)*. Sustituyendo *plot* por *loglog*, *semilogy* o *semilogx*, se cambian uno o ambos ejes a la escala logarítimica. El comando *axis([a b c d])* ajusta el tamaño del gráfico al del rectángulo $a \le x \le b$, $c \le y \le d$. Para dar título al gráfico o marcar los ejes de las *x* o de las *y*, se escribe entre comillas la etiqueta deseada, como en los ejemplos siguientes:
title('altura del satélite') *xlabel*('tiempo en segundos') *ylabel*('altura en metros')
El comando *hold* conserva el gráfico anterior mientras se dibuja uno nuevo. Al repetir *hold*, se borra la pantalla. Para imprimir o guardar la pantalla de gráficos en un archivo, véase *help print* o ejecútese *print –Pnombre* de la impresora *print –d* nombre del archivo.

2.1. GRÁFICAS

En *MATLAB* se pueden crear gráficas tan simples como:
D = [1 2 3 5 4 7 6 8 9 8 6 3 1 3]; plot(D)

o se pueden crear gráficas tan complejas como :

>>*cplxroot*(3,10) Superficie de una raíz cúbica.

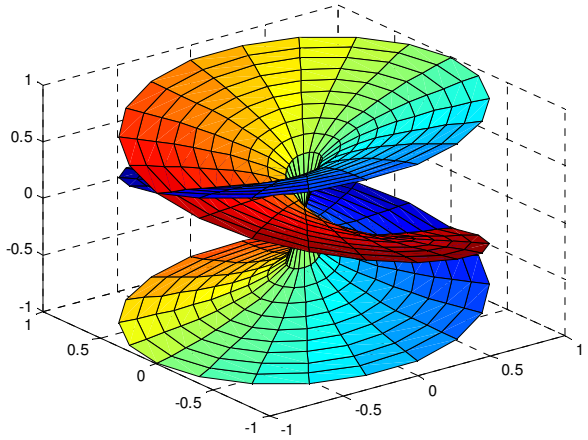


Figura 2.1

Como se vio en el primer ejemplo es posible graficar una serie de puntos y *MATLAB* automáticamente ajusta los ejes donde se grafica.

Por ejemplo, para graficar la función seno se pueden crear un rango de valores

>>*x = 0: 0.1: 20*; *x = vector* de cero a veinte con incrementos de 0.1

>>*y = sin(x)*; Seno del vector (x)

>>*plot(x,y)* Gráfica del seno

>>*z = cos(x)*; Coseno del vector anterior

>>*plot(x,z)* Gráfica del coseno de x.

>>*plot(x,y,x,z)* Gráfica del seno y coseno en la misma pantalla

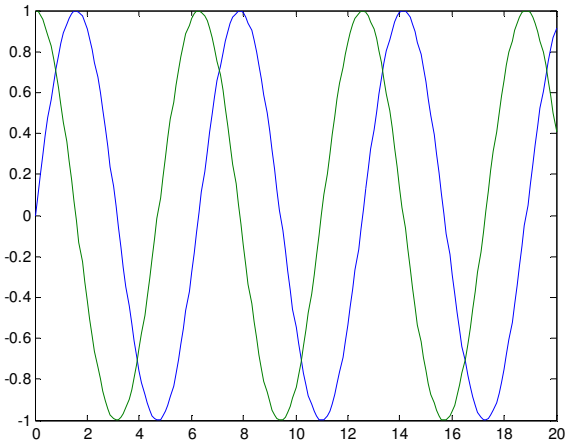


Figura 2.2

>>*plot(x,z,'*')* Gráfica del coseno con los signos ' * '

Hace la gráfica en azul, y los signos ' + ', intercambiando los ejes.

plot(z, x,'b+')

Como se ve es posible graficar en *MATLAB* con símbolos y además escoger el color, tal como se muestra en la tabla inferior.

Símbolo	*Color*	*Símbolo*	*Estilo de línea*
y	amarillo	.	punto
m	magenta	o	circulo
c	cían	x	equis
r	rojo	+	más
g	verde	*	asterisco
b	azul	-	menos
w	blanco	:	dos puntos
k	negro	-.	menos punto
		--	menos menos

La función *plot* nos permite otras opciones como superponer gráficas sobre los mismos ejes:
>> *x = [-2 -1 0 1 2 3]; y = [4 1 0 1 4 9]; z = [6 5 3 7 5 2];*
>> *plot(x,y,x,z)*

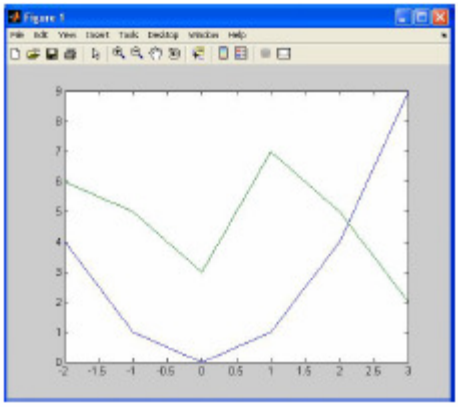


Figura 2.3

Además podemos colocar etiquetas o manipular la gráfica:
etiqueta sobre el eje X de la gráfica actual: >> ***xlabel('texto')***
etiqueta sobre el eje Y de la gráfica actual: >> ***ylabel('texto')***
título en la cabecera de la gráfica actual: >> ***title('texto')***
texto en el lugar especificado por las coordenadas: >> ***text(x,y, 'texto')***
texto, el lugar lo indicamos después con el ratón: >> ***gtext('texto')***
dibujar una rejilla: >> ***grid***
fija valores máximo y mínimo de los ejes: >> ***axis([xmin xmax ymin ymax])***
fija que la escala en los ejes sea igual: >> ***axis equal***
fija que la gráfica sea un cuadrado: >> ***axis square***
desactiva ***axis equal*** y ***axis square:*** >> ***axis normal***
abre una ventana de gráfico: >> ***hold on***
borra lo que hay en la ventana de gráfico: >> ***hold off***

Otros comandos relacionados con las gráficas son los siguientes:

Orden	¿Qué hace?	Imagen
area	colorea el area bajo la gráfica	
bar	diagrama de barras (verticales)	
barh	diagrama de barras (horizontales)	
hist	histograma	
pie	sectores	
rose	histograma polar	
stairs	gráfico de escalera	
stem	secuencia de datos discretos	
loglog	como plot pero con escala logarítmica en ambos ejes	
semilogx	como plot pero escala logarítmica en el eje x	
semilogy	como plot pero escala logarítmica en el eje y	

Subplot

El comando subplot nos permite desplegar en pantalla varias gráficas.

>>*subplot(m,n,a)*

'*m*' y '*n*' es una matriz que representa las cantidades de gráficas que se van desplegar; *'a'* indicaría el lugar que ocuparía la gráfica en el subplot.

Hagamos la gráfica de los siguientes puntos. La desplegaremos en cuatro puntos diferentes en pantalla para ver las características de subplot.

>>*a=[* 1 ,2 ,3 9 ,8 ,7 ,4, 5, 6, 8, 7, 5*];*

>>*plot(a)*

Una ventana gráfica se puede dividir en *m* particiones horizontales y en *n* verticales, de modo que cada subventana tiene sus propios ejes, y para hacer esto vamos a usar *subplot (m, n, p)* donde *p* indica la subdivisión que se convierte en activa.

>> *x* = 1:360; *y*1 = *sind(x)*; *y*2 = *cosd(x)*; *y*3 = *exp(x)*; *y*4 = *exp(-x)*;

>> *subplot*(2,2,1), *plot*(*x*,*y*1), *title*('seno')

>> *subplot*(2,2,2), *plot*(*x*,*y*2), *title*('coseno')

>> *subplot*(2,2,3), *plot*(*x*,*y*3), *title*('exponencial')

>> *subplot*(2,2,4), *plot*(*x*,*y*4), *title*('-exponencial')

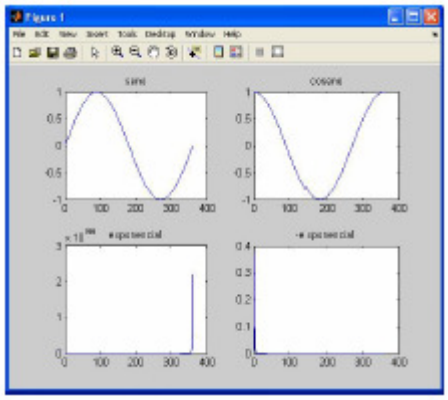


Figura 2.4

CLF borra todos los objetos de la gráfica.

CLF RESET Borra todo lo que hay en la gráfica y resetea todas las propiedades de la figura.
clf
Para dibujar polígonos podemos usar la función *plot* pero teniendo en cuenta que el último punto de ambos vectores deben coincidir para que la gráfica quede cerrada. Pero si lo que queremos es que quede coloreado todo el interior del polígono debemos usar mejor la función *fill*, tiene tres argumentos, los dos vectores que forman los puntos y un tercer argumento para indicar el color.
>> *x = [-2 0 2 0 -2]; y = [4 8 4 0 4];*
>> *plot(x,y)*

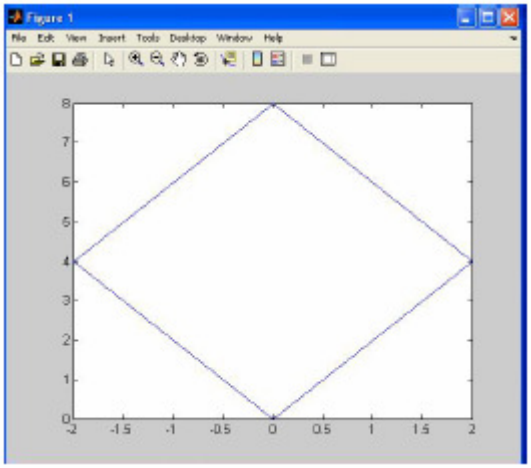


Figura 2.5

>> *x = [-2 0 2 0 -2]; y = [4 8 4 0 4];*
>> *fill(x,y,'r')* % dibuja el polígono, *'r'* indica el color rojo

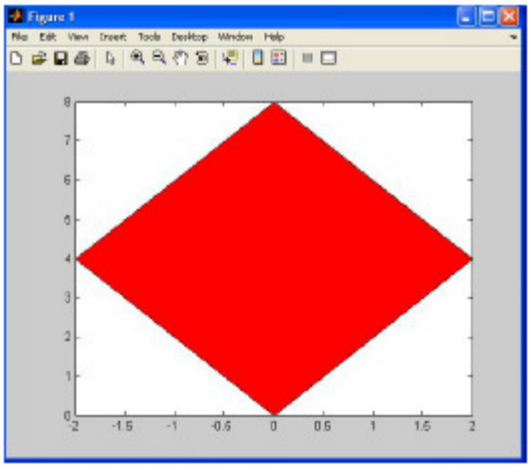


Figura 2.6

2.2 GRÁFICAS EN TRES DIMENSIONES

El comando plot se puede extender a 3 dimensiones con el comando *plot3*.

El siguiente ejemplo hace una gráfica de una espiral en tres dimensiones.
>>*t*=0:*pi*/50:10**pi*;
>>*plot3*(*sin*(*t*),*cos*(*t*),*t*)

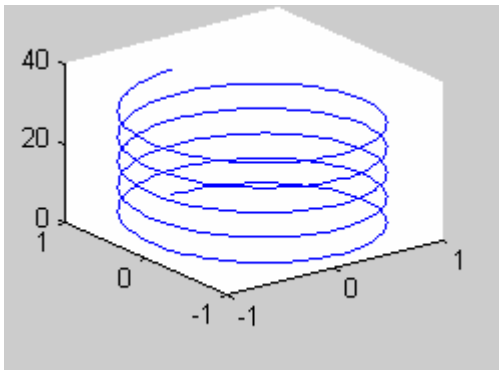


Figura 2.7

zlabel('etiqueta')

Se utiliza para dar etiquetas al eje *z*, en las gráficas en tres dimensiones.

2.2.1 Gráficos de malla y superficie.

>>*z* = *peaks*(10)
El comando *peaks* crea un conjunto de valores que al ser graficados, se ven de la siguiente manera.
>>*plot*(*z*)
Se tomará como base la gráfica anterior para demostrar algunas funciones de graficación en tres dimensiones.
>>*mesh*(*z*)
>>*contour*(*z*,10)
>>*surf*(*z*)
Es posible cambiar el sentido de orientación de las gráficas con el comando *view*(*x*,*y*)
>>*view*(0,0)
>>*view*(90,0)
Si queremos representar un polígono en 3 dimensiones lo haremos con la función *fill3* de forma similar a *fill* pero ahora con 4 argumentos, siendo el cuarto el que indica el color.
>> *x* = *[-2 0 2 0 -2]*;
>> *y* = *[4 8 4 0 4]*;
>> *z* = *[3 5 10 5 3]*;
>> *fill3* (*x*,*y*,*z*,*'b'*) % dibuja en *3-D*, *'b'* indica el color azul

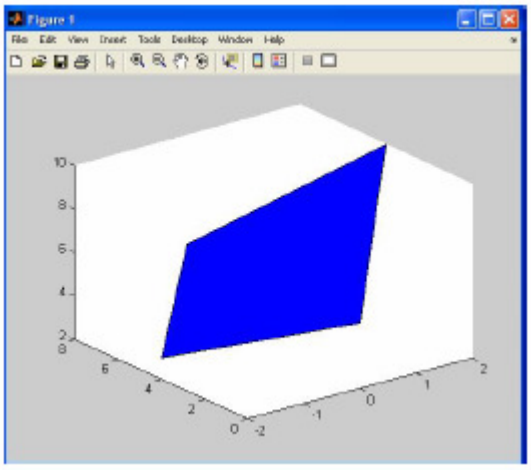


Figura 2.8

Superficie de malla:

La orden *[X,Y]=meshgrid(x,y)* crea una matriz *X* cuyas filas son copias del vector *x* y una matriz *Y* cuyas columnas son copias del vector *y*. Para generar la gráfica de malla se usa la orden *mesh(X,Y,Z)*, *mesh* acepta un argumento opcional para controlar los colores. También puede tomar una matriz simple como argumento: *mesh(Z)*.

Ejemplo:

```
>> x = -10:0.5:10; y = -10:0.5:10;
>> [X,Y] = meshgrid(x,y); % crea matrices para hacer la malla
>> Z = sin(sqrt(X.^2 + Y.^2)). / sqrt(X .^ 2 + Y .^ 2 + 0.1);
>> mesh(X,Y,Z) % dibuja la gráfica
```

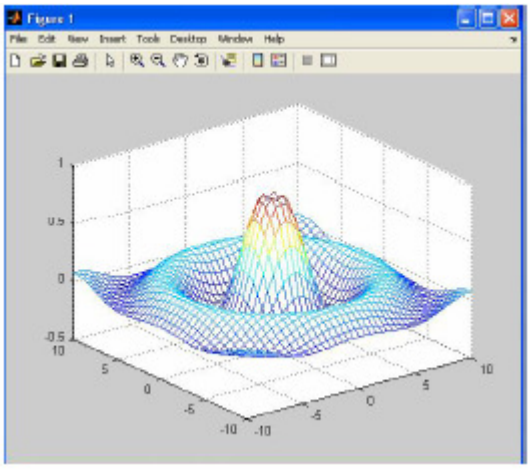


Figura 2.9

Hubiera dado igual si hubiéramos escrito:

```
>> [X,Y] = meshgrid(-10:0.5:10);
>> Z = sin(sqrt(X.^2 + Y.^ 2)). / sqrt(X .^ 2 + Y .^ 2 + 0.1);
```

>> *mesh*(X,Y,Z)

Gráfica de superficie:

Es similar a la gráfica de malla, pero aquí se rellenan los espacios entre líneas. La orden que usamos es *surf* con los mismos argumentos que para *mesh*.

Ejemplo:

>> *surf*(X,Y,Z)

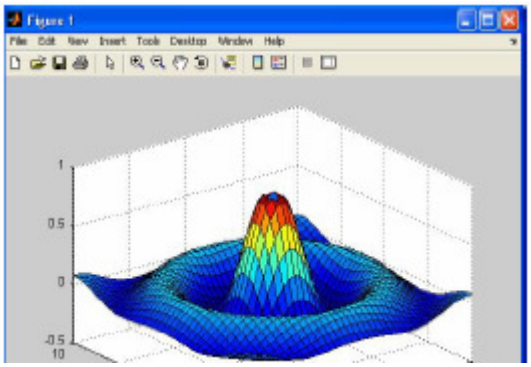


Figura 2.10

Las gráficas de contorno en *2-D* y *3-D* se generan usando respectivamente las funciones *contour* y *contour3*.

Ejemplo:

>> *contour*(X,Y,Z) % dibuja las líneas de contorno

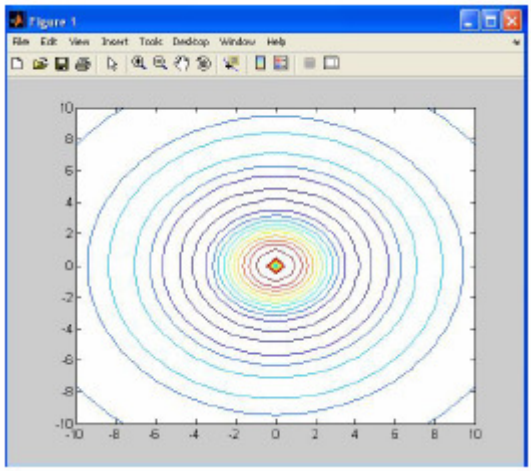


Figura 2.11

La función *pcolor* transforma la altura a un conjunto de colores.

Ejemplo:

>> *pcolor*(X,Y,Z)

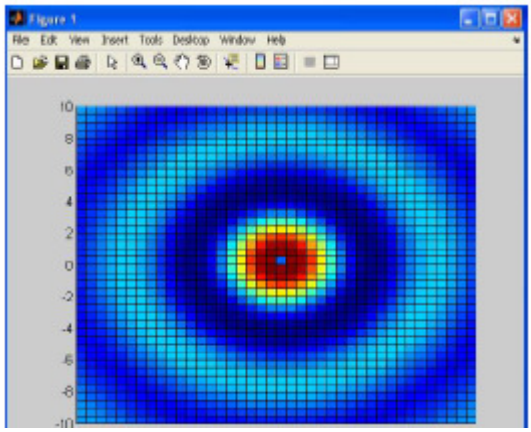


Figura 2.12

Manipulación de gráficos:

Fijar el ángulo de visión especificando el *azimut* y la elevación: >> *view(az,el)*
colocar su vista en un vector de coordenada cartesiana *(x,y,z)* en el espacio 3-D: >> *view([x,y,z])*
almacenar en *az* y *el* los valores del *azimut* y de la elevación de la vista actual: >> *[az,el]=view*
añadir etiquetas de altura a los gráficos de contorno: >> *clabel(C,h)*
añadir una barra de color vertical mostrando las transformaciones: >> *colorbar*

Ejemplos:

>> *surf(X,Y,Z); view(10,70)*

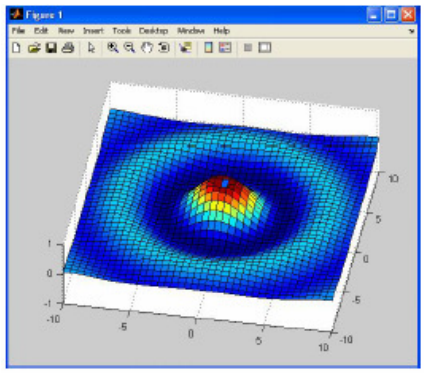


Figura 2.13

Colorbar %añade la barra de color a la figura actual

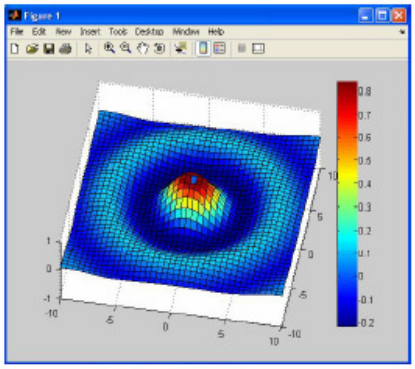


Figura 2.14

```
>> surf(X,Y,Z)
>> view([10,-12, 2])
```

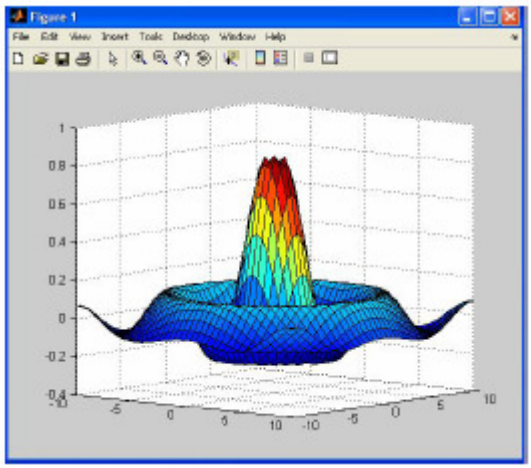


Figura 2.15

```
>> surf(X,Y,Z)
>> [az,el] = view
az =
    -37.5000
el =
     30
>> [C,h] =contour(X,Y,Z);
>> clabel(C,h)
```

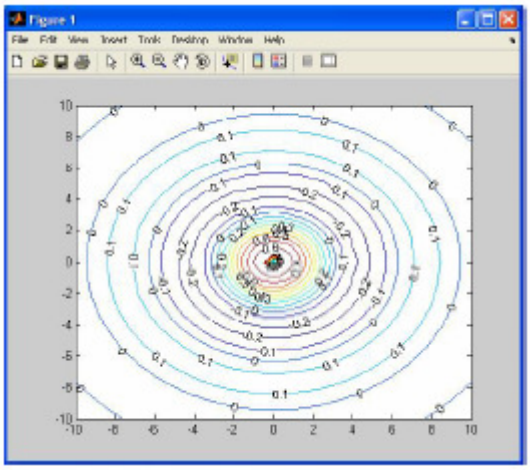


Figura 2.16

Compresión de los mapas de color:

Color	*Nombre corto*	*Rojo/Verde/Azul*
Negro	k	[000]
Blanco	w	[111]
Rojo	r	[100]
Verde	g	[010]
Azul	b	[001]
Amarillo	y	[110]
Magenta	m	[101]

La sentencia *colormap(M)* instala al matriz M como el mapa de color a utilizar por la figura actual.

Función	Colores
Jet	
HSV	
Hot	
Cool	
Spring	
Summer	
Autumn	
Winter	
Gray	
Bone	
Copper	
Pink	
Lines	

>> M = *[0 0 0; 1 0 0; 0 1 0; 0 0 1; 1 1 0];* % creamos una matriz de colores
>> *colormap(M)*

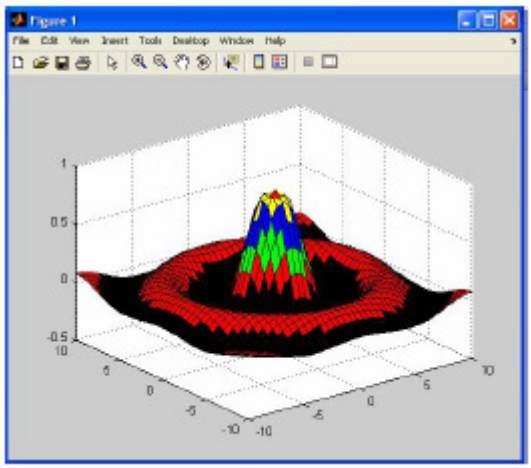


Figura 2.17

2.2.3 Gráficas en el plano complejo

Ahora vamos a crear un conjunto de valores para graficar en el plano complejo, en tres dimensiones.
>>*z= cplxgrid*(5)
>>*cplxmap(z,z)*
>>*cplxmap(z,z.^z)*
>>*cplxroot*(2,10) Raíz cuadrada
Se pueden crear gráficos en coordenadas polares con el comando *Polar(t, r, s)* donde *t* es el vector en ángulos en radianes, *r* es el radio del vector y s es la cadena de caracteres que describe, color, símbolo del estilo de la línea.
>>*t=0:0.1:2*pi;*
>>*r = sin(2*t).*cos(2*t);*
>>polar(t,r)
gtext(' texto ')
Se utiliza para colocar texto en una gráfica, con la ayuda del ratón. Simplemente se ejecuta el comando y con el ratón se selecciona la coordenada deseada y se presiona el botón derecho del ratón, quedando fijo el texto en la pantalla.

Copiar una gráfica

Cuando se quiera realizar algún reporte formal en un procesador de palabras como en este caso Word, es posible copiar las gráficas hechas en *MATLAB* por medio de la orden copy to bitmap. El procedimiento sería:

- En *MATLAB*, en el menú de la ventana principal de la gráfica, se escoge el menú '*Edit*' y de este se escoge *Copy to 'bitmap'*;
- Se minimiza *MATLAB* y se pasa al procesador de palabras escogido
- Se localiza la posición en la cual estará la gráfica, y del menú *Edit* se escoge '*Paste o Pegar*'.

La gráfica aparecerá en el procesador de palabras.
Existe un pequeño inconveniente ya que la gráfica aparecerá sobre un fondo de color negro que *MATLAB* tiene por *default*, si se imprime este documento obviamente la gráfica aparecerá sobre un fondo negro lo cual hará que la impresora gaste tinta en exceso.
Para remediar esto se puede cambiar el color de fondo de las gráficas a blanco con el comando.

Whitebg

Después se hace procedimiento mencionado anteriormente.
Imprimir una gráfica. Se puede imprimir una gráfica directamente desde el menú de la ventana de la gráfica, seleccionando la opción.

CAPITULO 3 - FUNCIONES. LÍMITES Y DERIVADA

3.1 FUNCIONES

Una vez definida explícitamente simbólicamente la función, se pueden reemplazar valores de la función, con los comandos

Subs(f,a): aplica *f* en el punto a
Subs(f,a,b): sustituye en *f* el valor de a por el valor de *b*
Subs(f,[x,y,...],[a,b,...]: sustituye en la expresión de las *variables[x,y,...]* por los *valores[a,b,...]*
Sea $f(x)= x^2$

```
>>syms b;
>>f='x^2';
>>subs(f,b+2)
ans =
   (b+2)^2
```

Veamos una función suma *f=a+b*;

```
>> syms a b;
>> subs(a+b,a,4)
ans =
   4+b
```

Ahora

```
>> subs(a+b,[a b],[3 5])


ans =
   8
```

Sea sumar funciones o dividirlas

```
>> syms x;
>> f=x^2;g=x^5-1;h=cos(x);
>> f+g+h


ans =
   x^2+x^5-1+cos(x)
>> f/g
 ans =
    x^2/(x^5-1)
```

Para componerlas

```
>> compose(f,g)
ans =
   (x^5-1)^2
```

Si deseamos la inversa de esta, recordemos que es cuadrática:

```
>> h=finverse((x^5-1)^2)
Warning: finverse((x^5-1)^2) is not unique.
> In sym.finverse at 48
h =
(1+x^(1/2))^ (1/5)
```

MATLAB permite también la transformación de coordenadas, así:

[THETA, RHO, Z]=cart2pol(X,Y,Z) cartesianas a cilíndricas
[THETA, RHO]=cart2pol(X,Y) cartesianas a polares
[THETA, PHI, R]=cart2sph(X,Y,Z) cartesianas a esféricas
[X,Y,Z]=pol2cart[THETA,RHO,Z] cilíndricas a cartesianas
[X,Y]=pol2cart[THETA,RHO] polares a cartesianas
[X,Y,Z]=sph2cart[THETA,PHI,R] esféricas a cartesianas

Por ejemplo, sea el punto *(1, 1 ,1)* en cartesianas pasar a esféricas y cilíndricas

```
>> >> [X,Y,Z]=cart2sph(1, 1, 1)
X =
   0.7854
Y =
  0.6155
Z =
  1.7321
>>[X,Y,Z]=cart2pol(1, 1, 1)
X =
   0.7854
Y =
   1.4142
Z =
  1
```

3.2 LÍMITE

Haremos hincapié en las posibilidades ofrecidas por *Symbolic Math Toolbox.*
Con *limit* se calcula límite de la expresión simbólica.
(*límit expr, x, a*) calcula límite bidireccional de la expresión expr simbólica cuando *x* tiende a *a*.
límite(expr, a) calcula el límite bidireccional de la expresión *expr* simbólica cuando variable por defecto tiende a *a*.
límit(expr) calcula límite bidireccional de la expresión simbólica expr cuando la variable se aproxima a 0 por defecto.
límit / expr, x, a, 'left') calcula el límite de la expresión expr simbólica cuando x tiende a *a* por la izquierda.
límit(expr, x, a, 'right') calcula el límite de la expresión expr simbólica cuando x tiende a *a* por la derecha.

Si establecemos la expresión matemático-comando *MATLAB* sería:

Operacón matemática	*Comando de MATLAB*
$\lim_{x \to 0} f(x)$	limit(f)
$\lim_{x \to a} f(x)$	limit(f,x,a) or limit(f,a)
$\lim_{x \to a-} f(x)$	limit(f,x,a,'left')
$\lim_{x \to a+} f(x)$	limit(f,x,a,'right')

Extendemos el concepto para sucesiones

limit(sucesion,n,inf): calcula el limite de la sucesión escrita en su término general(para n tendiendo a infinito)

```
>> limit(((2*n-3)/(3*n-7))^4,inf)
ans =
   16/81
>> syms x;
>> limit(sin(x)/x)  % límite notable
ans =
   1
```

Si incluimos un parámetro *a*

```
>> limit(sin(a*x)^2/x^2,x,0)

ans =
   a^2

>> limit(cos(a*x)^2/x^2,x,pi/2)
ans =
   4*cos(1/2*pi*a)^2/pi^2
```

Tomemos la función límite de la sucesión $h_n(x)=(x^2+nx)/n$, para x real

```
>> limit((x^2+n*x)/n,n,inf)
ans =
    x
```

Nos dio la recta identidad, probemos en un intervalo, para ciertos n:

```
><fplot('[(x^2+x),(x^2+2*x)/2,(x^2+3*x)/3,(x^2+4*x)/4,(x^2+5*x)/5,(x^2+6*x)/6]',[-
2,2,-2.2])
```

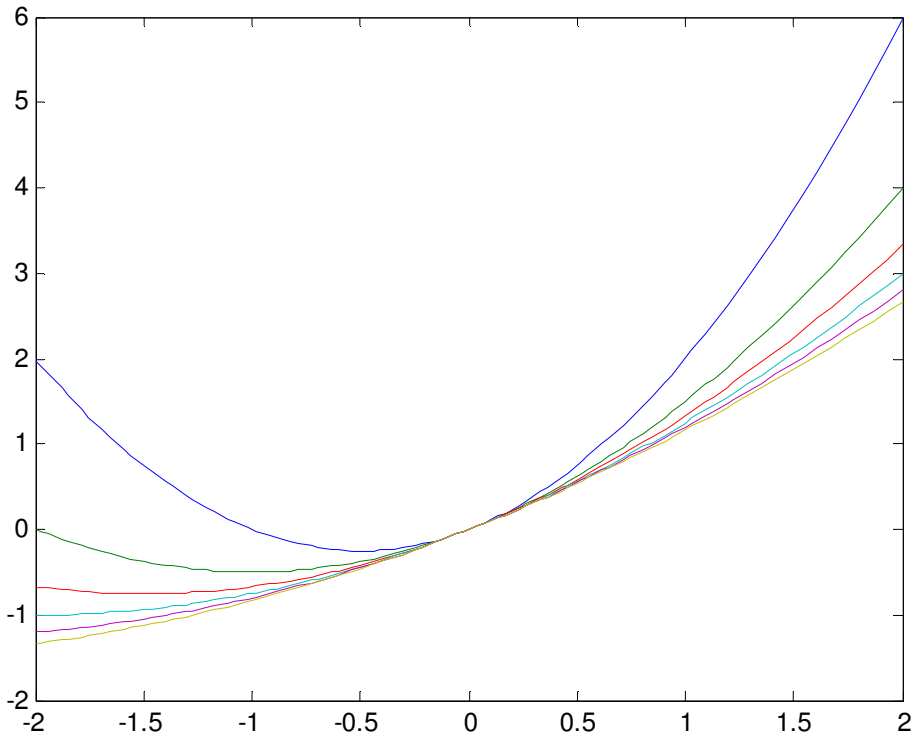


Figura 3.1

3.2.1 Límite y continuidad

Apliquemos para la función *sin(x)/x*, en dominio *R-{0}*:

```
>> syms x a;
>> limit(sin(x)/x,x,a)
 ans =sin(a)/a
```

que vemos es el valor de *f(a)*

De la misma manera estudiemos para $f(x) = 1/(1+\sqrt[x]{e})$ *si* $x \neq 0$ *y* $=1$ *si* $x=0$. En $x=0$ habría una discontinuidad.

```
>> syms x;
>> limit(1/(1+exp(1/x)),x,0,'right')
ans =
   0


>> limit(1/(1+exp(1/x)),x,0,'left')
 ans =
    1
```

Vemos que no coinciden los límites laterales pero son finitos, sería una discontinuidad de primera especie, gráficamente:

```
>>fplot('1/(1+exp(1/x))',[-3 3])
```

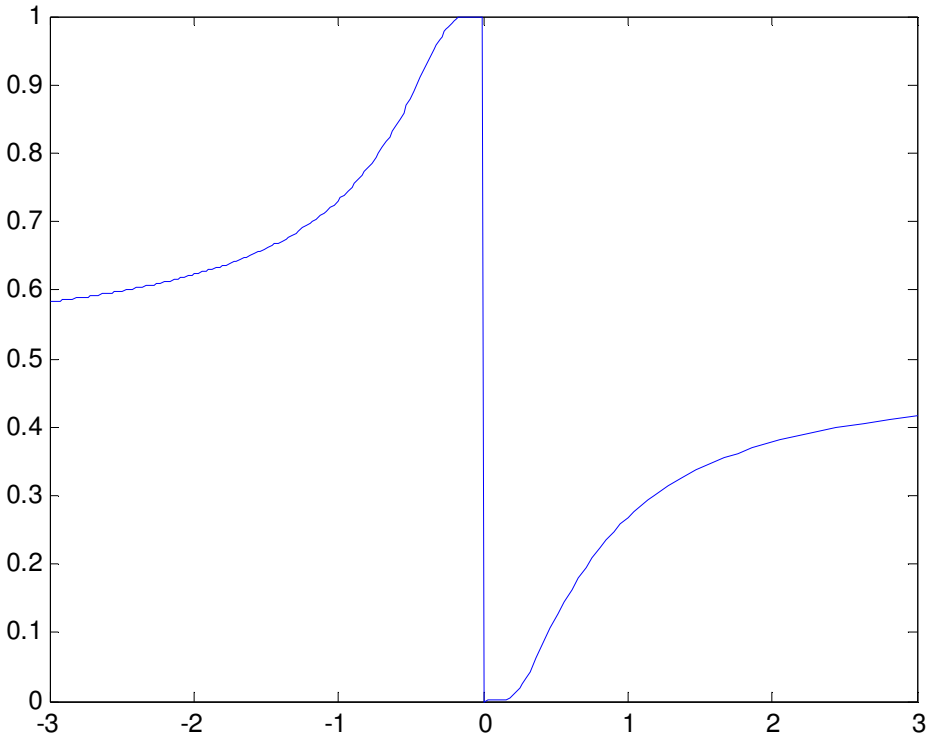


Figura 3.2

También podemos estudiar la convergencia de una sucesión numérica: $a_n = \sum_{1}^{\infty} n/2^n$

Por el criterio del *cociente*($\lim_{\infty} a(n+1)/a(n)$ menor que 1), se halla la suma

```
>> syms n
>> f=n/2^n;
f =
n/(2^n)
```

y el límite

```
>> limit(subs(f,n,n+1)/f,n,inf)
ans =
   ½
```

que vemos es menor que 1, siendo convergente la serie, su suma valdrá:

```
>> symsum(f,n,1,inf)
ans =
  2
```

3. 3 DERIVADA

La funciones de *MATLAB* para la derivada son:

diff('f','x'): halla la derivada de *f* respecto a *x*
syms x,diff(f,x): halla la derivada de *f* respecto a *x*

diff('f','x,',n): halla la derivada n-ésima de *f* respecto a *x*
syms x,diff(f,x,n): halla la derivada n-esima de *f* respecto a *x*
Ejemplos.
i)f(x)=log(cos2x)

```
>> pretty(diff('log(cos(2*x))','x')) % version arreglada
-2sin(2 x)/cos(2x)
>> pretty(simple(diff('log(cos(2*x))','x'))) %versión simplificada

                -2 tan(2 x)
```

ii)f(x)=cos(x), hallar las tres primeras derivadas

```
>> [diff(f),diff(f,2),diff(f,3)]
ans =
  [ -sin(x), -cos(x),  sin(x)]
```

Si recordamos la derivada como el paso al límite de la variación incremental de la función, sea por ejemplo *a)sen(x) b)log(x),* se podría plantear desde la función límite:

```
>> syms x h;
    a)  df=limit((sin(x + h) - sin(x))/h, h, 0)
df = cos(x)
b) df=limit((log(x + h) - log(x))/h, h, 0)
df = 1/x
c) >>syms h n x;
>>limit((1 + x/n)^n,n,inf ),
ans =exp(x)
```

3.4 DESARROLLOS DE TAYLOR Y MAC LAURIN

Las funciones de *MATLAB* para el desarrollo de Taylor son:
t=taylor(f,n,v,a) :encuentra el desarrollo de Taylor de orden *n-1* para *f* en la variable *v* cerca del punto *a*
t=taylor(f,n,v) :encuentra el desarrollo de Mac Laurin de orden *n-1* para *f* en la variable *v* .
t=taylor(f) :encuentra el desarrollo de Mac Laurin de orden 5 para *f* en la variable por defecto.
Así, para $f(x)=(1-x)^{-1}$, cerca de *x*=2, para grado 8 de la serie de Taylor

```
>> syms x;
>> f=1/(1-x);
>> pretty(taylor(f,9,x,2))
```

$$-3 + x - (x - 2)^2 + (x - 2)^3 - (x - 2)^4 + (x - 2)^5 - (x - 2)^6 + (x - 2)^7 - (x-2)^8$$

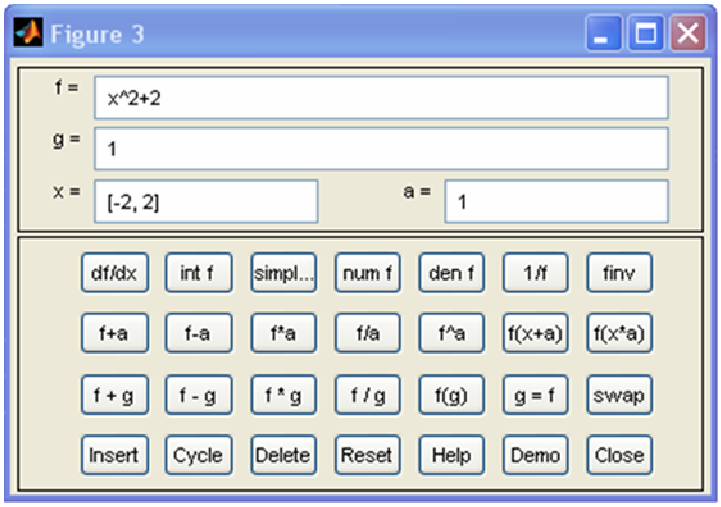
La correspondiente de Mac Laurin:

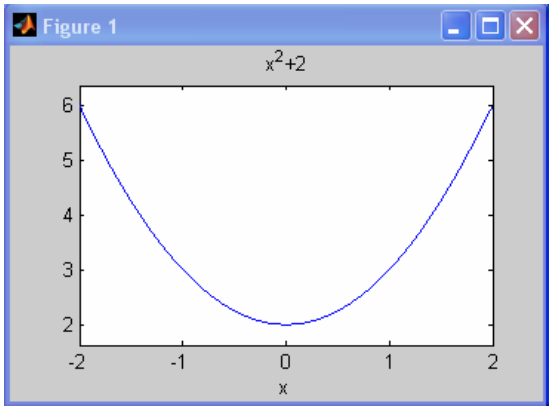
```
>>pretty(taylor(f,8,x))
```

$$1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7$$

Además en las posibilidades de *MATLAB* se dispone de:
**funtool*

Se trata de una herramienta interactiva dentro del Toolbox de Matemática Simbólica.
Abre una "calculadora gráfica interactiva" que, para una función real de una variable, *f(x)*, permite calcular de forma analítica y, al mismo tiempo, representarla gráficamente, incluso las operaciones con otra función *g(x)*, derivada, integral, inversa, etc
* *taylortool*
Abre una ventana de dibujo interactiva que grafica una función y su desarrollo de *Taylor* en torno a un punto. Esta ventana permite cambiar: la función, el intervalo a representar, el grado y el punto en torno al cual se calcula el polinomio.
Ejemplo: *f(x)= x^2+1*
>>*funtool*
Aparecerán las tres figuras, trabajamos sobre la fig.3.3 para cargar los datos





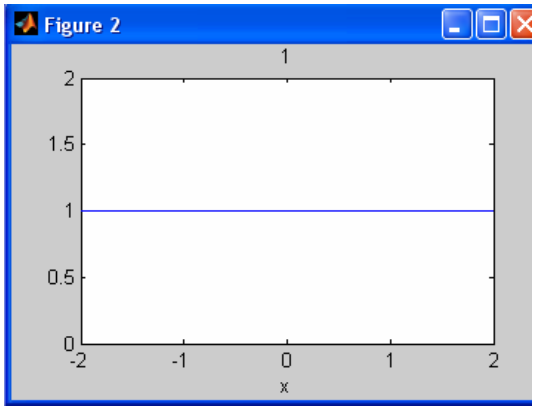


Figura 3.3

En la calculadora se ingresa por ejemplo *df/dx*, resultando:

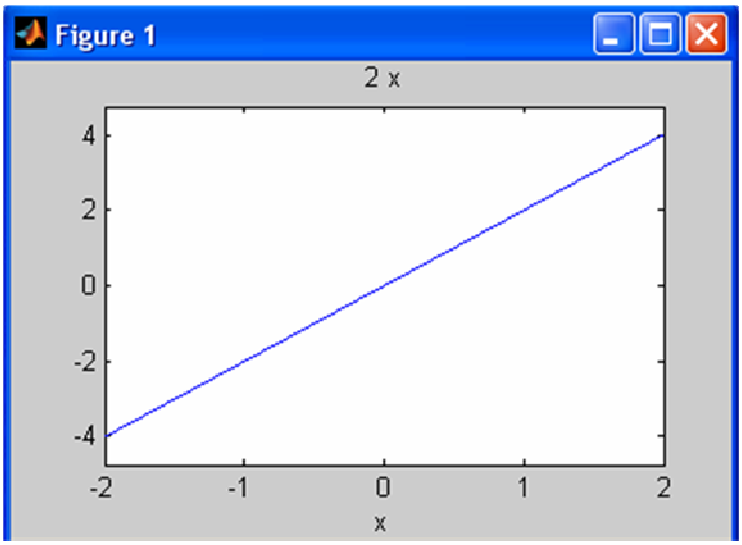
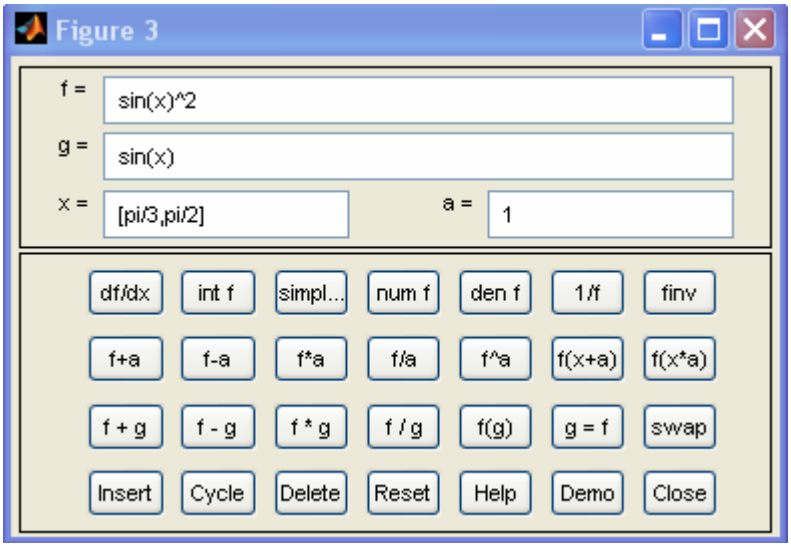


Figura 3.4

Si *g(x)=sen(x)*, entre *pi/3* y *p/2*; buscando *f(g)*, se obtendrá:



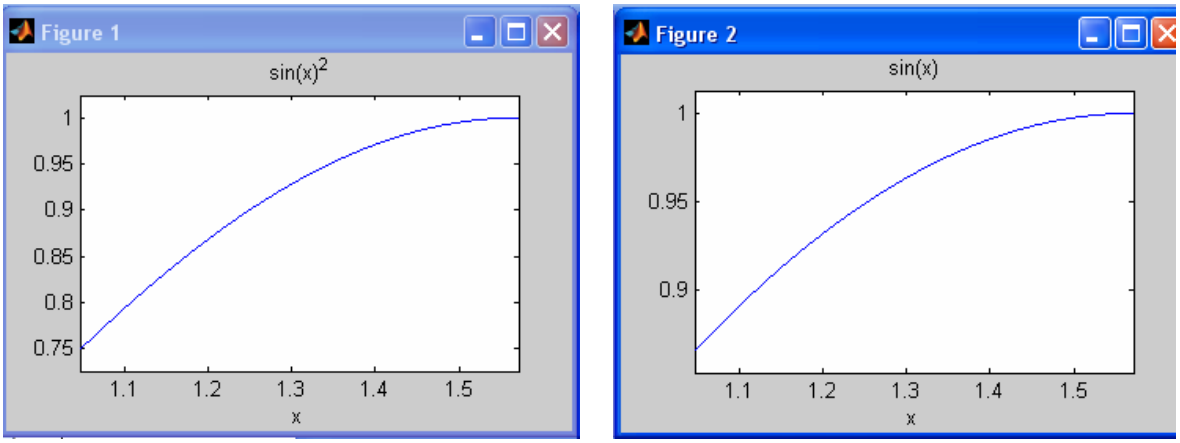


Figura 3.5

>>*taylortool* *%abre* la ventana gráfica
Sea la función *xsen(x)*, de grado 8 en un entorno de *pi/*8:

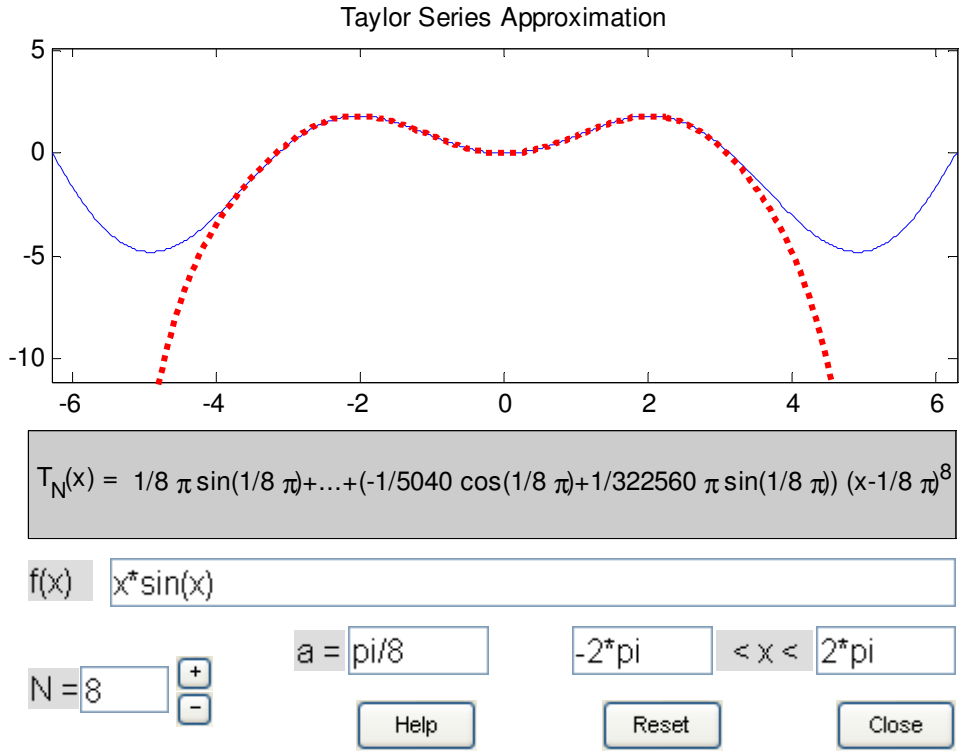


Figura 3.6

Aplicación del teorema del valor medio

Sea *f(x)=cos(x-1)* en el compacto *[1 2.5]*, cuál será el valor de abscisa donde la función valor 0.8
>> >> *c=fzero(inline('cos(x-1)-.8'), [1 2.5])*
c =
1.6435

Ejemplo de análisis de una función

Sea la función $f(x)=\dfrac{x^2-2}{x^2+x}$

a) Crear la función
>>*syms x*
>>*num = x^2 -2;*
>>*denom = x^2 + x;*
>>*f = num/denom*
f =(x^2-2)/(x^2+x)
si se grafica:
>>*ezplot(f)*

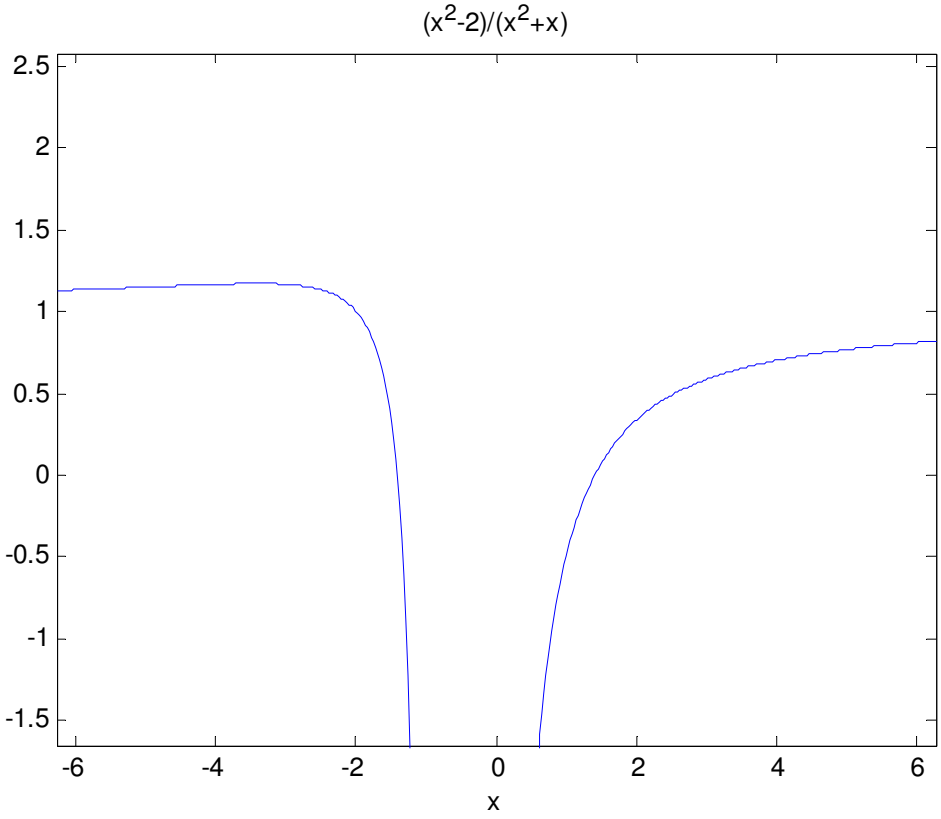


Figura 3.7

b) búsqueda de asíntotas

```
>>limit(f, inf)
ans =1,    (y=1)
```

para las verticales hacer denominador hacia 0:

```
>>roots = solve(denom)
roots =  0 –1
```

también se pueden graficar

c) Máximos y mínimos

```
>>f1 = diff(f)
f1 =2*x/(x^2+x)-(x^2-2)/(x^2+x)^2*(2*x+1)
>>pretty(f1)
```

$$2\,\frac{x}{x^2+x}-\frac{(x^2-2)\,(2x+1)}{(x^2+x)^2}$$

poniendo la derivada igual a cero y resolver para puntos críticos

```
>>crit_pts = solve(f1)
crit_pts =
-2+2^ (1/2)
-2-2^ (1/2)
```

del gráfico se ve que la primera raíz es máximo, la segunda mínimo

```
>>ezplot(f)
>>hold on
>>plot(double(crit_pts), double(subs(f, crit_pts)), 'ro')
>>title('Maximo y  Minimo de  f')
```

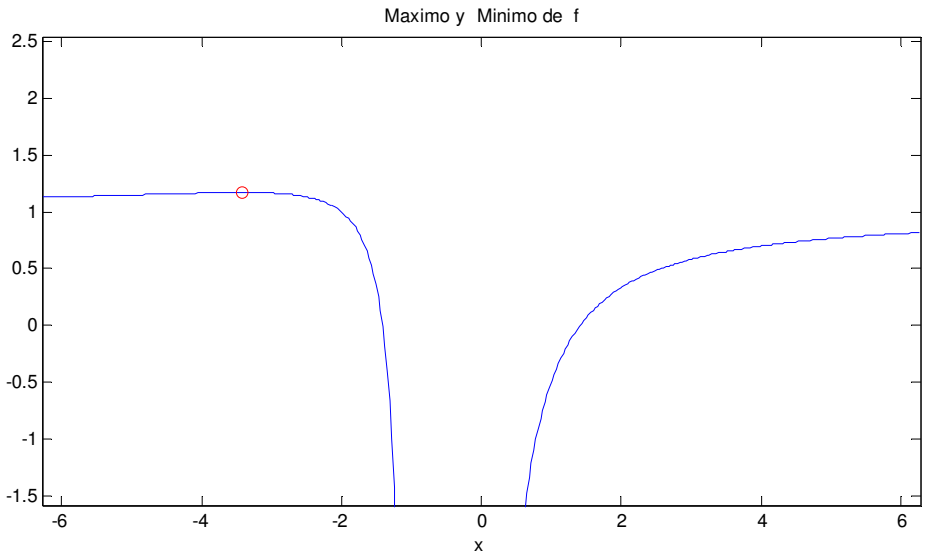


Figura 3.8

Punto de inflexión (anular segunda derivada)

```
>>f2 = diff(f1);
>>inflec_pt = solve(f2);
>>double(inflec_pt)
ans =
   -4.8473
   -0.5763 - 0.2836i
   -0.5763 + 0.2836i
```

sólo el primero es real, se pueden descartar los otros con

```
>>inflec_pt = inflec_pt(1)
>>inflec_pt = -2^(2/3)-2^(1/3)-2
```

Otro ejemplo:

$f(x) = \frac{1}{5 + 4cox}$ en reales

Creando la función

```
>>syms x
>>f = 1/ (5+4*cos(x));
```

Búsqueda de max-min

```
>> f2 = diff(f, 2)
f2 =
```

32/(5+4*cos(x))^3*sin(x)^2+4/(5+4*cos(x))^2*cos(x)

graficación

```
>>ezplot(f2);
>>axis([-2*pi 2*pi -5 2]);
>>title('Grafico de f2')
```

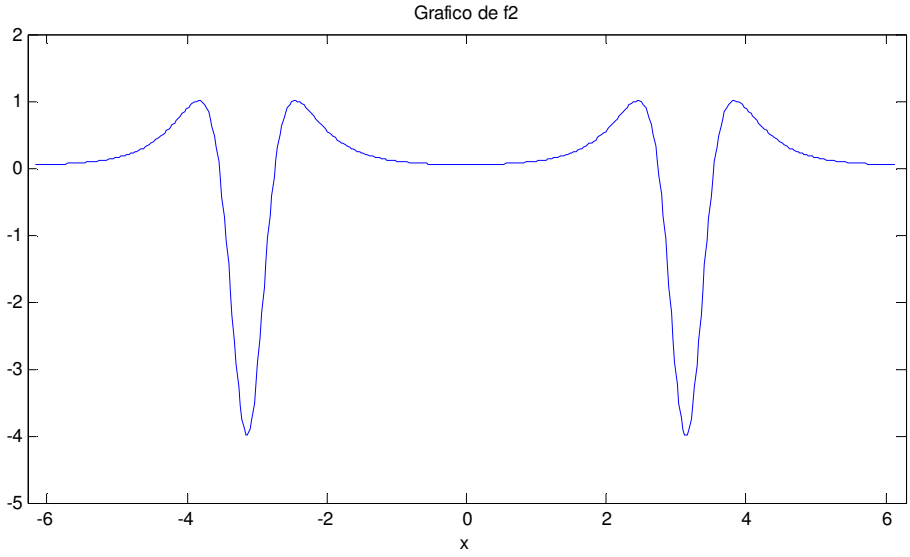


Figura 3.9

Búsqueda de ceros(max y min de *f2* ocurren en los ceros *f3*)

```
>>f3 = diff(f2);
>>pretty(f3)
```

$$384\,\frac{\sin(x)^3}{(5+4\cos(x))^4} + 96\,\frac{\sin(x)\cos(x)}{(5+4\cos(x))^3} - 4\,\frac{\sin(x)}{(5+4\cos(x))^2}$$

O en la forma

```
>>f3 = simple(f3);
>>pretty(f3)
```

$$4\,\frac{\sin(x)\,(96\sin(x)^2 + 80\cos(x) + 80\cos(x)^2 - 25)}{(5+4\cos(x))^4}$$

Para los ceros de la tercera derivada:

```
>> ceros = solve(f3)

ceros =
                                                    0
atan((-255-60*19^(1/2))^(1/2),10+3*19^(1/2))
atan(-(-255-60*19^(1/2))^(1/2),10+3*19^(1/2))
```

*atan((-255+60*19^(1/2))^(1/2)/(10-3*19^(1/2)))+pi*
*-atan((-255+60*19^(1/2))^(1/2)/(10-3*19^(1/2)))-pi*
(matriz 5x1 simbólica, una de sus entradas es un cero de la tercera derivada), con los comandos

```
>>format; % Default format of 5 digits
>>cerosd = double(ceros)
```

se convierten los ceros a la forma double

```
cerosd =
  0
        0 + 2.4381i
        0 - 2.4381i
        2.4483
       -2.4483
```

Apareciendo tres reales y dos complejos, no todos, en realidad son ceros, pues

```
>>ezplot(f3)
>>hold on;
>>plot(cerosd,0*cerosd,'ro') % Plot ceros
>>plot([-2*pi,2*pi], [0,0],'g.'); % Plot  eje x
>>title('Grafico de f3')
```

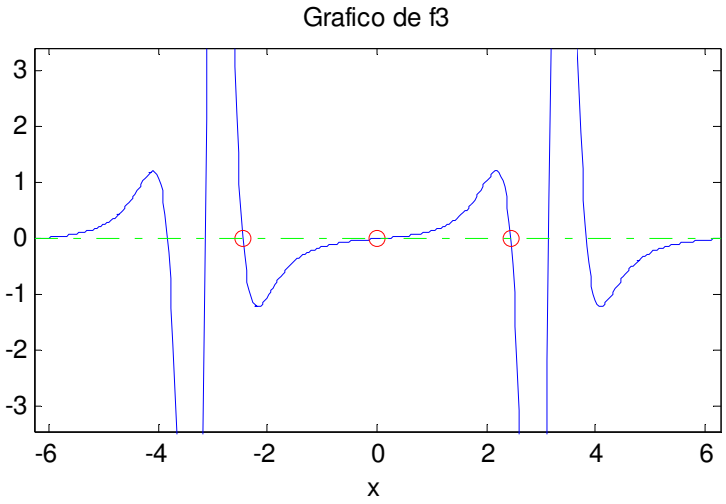


Figura 3.10

Una lista completa de ceros de la tercera derivada entre *–pi* y *pi*, será:

>>cerosd = [cerosd(1) cerosd(4) cerosd(5) pi]; si se quiere ver en la gráfica:

```
>>ezplot(f3)
>>hold on;
>>plot(cerosd,0*cerosd,'ro')
>>plot([-2*pi,2*pi], [0,0],'g.'); % Plot eje x
>>title('ceros de f3')
>>hold off;
```

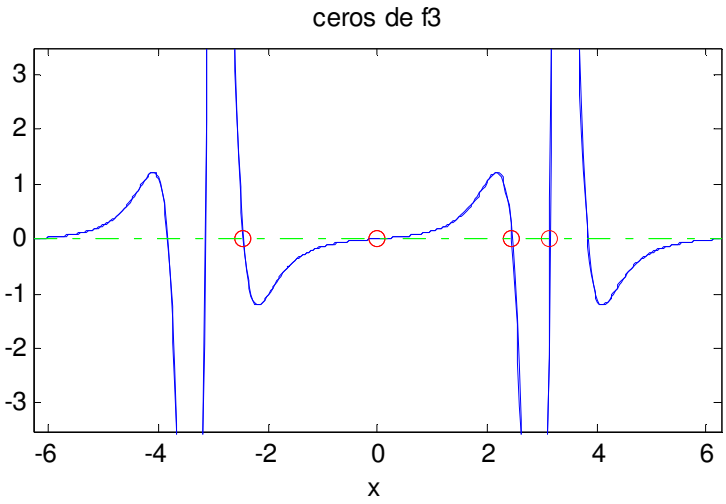


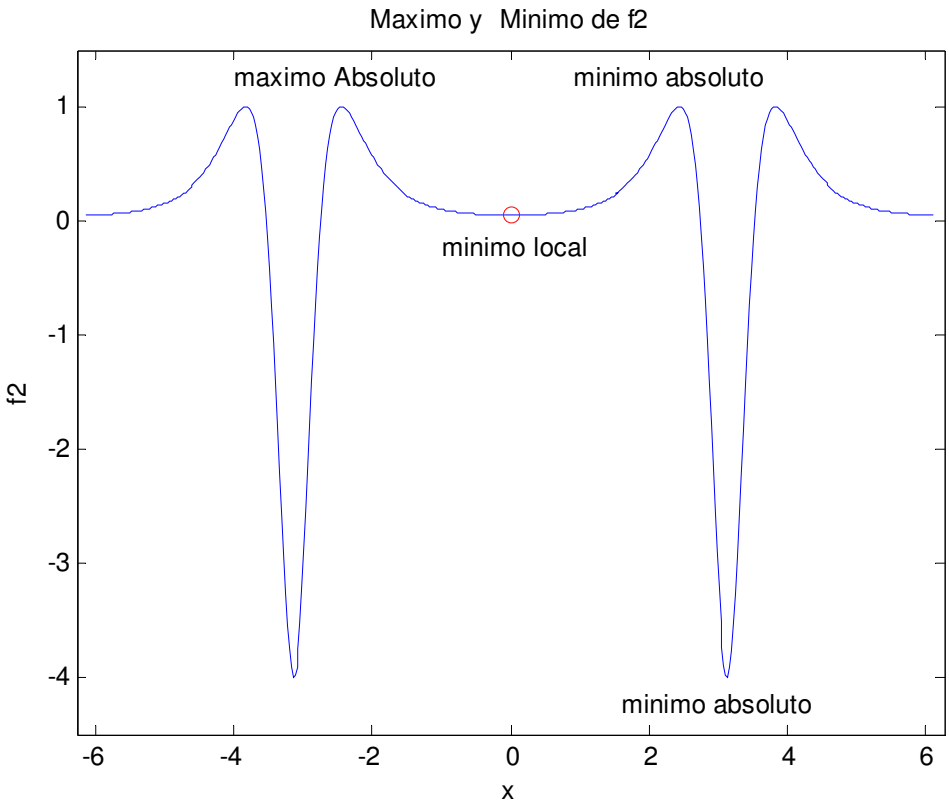
Figura 3.11

Calculando los valores de *f*'' en los ceros de *f*'''.

```
>>[cerosd; subs(f2,cerosd)]
ans =
  0    2.4483   -2.4483    3.1416
  0.0494    1.0051    1.0051   -4.0000
```

Se pueden mostrar los máximos y mínimos con

```
>>clf
>>ezplot(f2)
>>axis([-2*pi 2*pi -4.5 1.5])
>>ylabel('f2');
>>title('Maximo y  Minimo de f2')
>>hold on
>>plot(zeros, subs(f2,zeros), 'ro')
>>text(-4, 1.25, 'maximo Absoluto')
>>text(-1,-0.25,'minimo local')
>>text(.9, 1.25, 'minimo absoluto')
>>text(1.6, -4.25, 'minimo absoluto')
>>hold off;
```


Maximo y Minimo de f2
maximo Absoluto
minimo absoluto
minimo local
minimo absoluto
f2
x
1
0
-1
-2
-3
-4
-6
-4
-2
0
2
4
6
Figura 3.12

CAPITULO 4- INTEGRACION

4.1 LAS FUNCIONES VINCULADAS CON LA INTEGRACION EN MATLAB

syms *x*,int(*f*(*x*),*x*) o int('*f*(*x*)','*x*')	Halla $\int f(x)dx$ indefinida
int(int('*f*(*x*,*y*)','*x*'),'*y*')	Halla la doble $\iint f(x,y)dxdy$
syms *x y*,int(int(*f*(*x*,*y*),*x*),*y*))	Halla la doble $\iint f(x,y)dxdy$
syms *x a b*,int(*f*(*x*),*a*,*b*)	halla $\int_a^b f(x)dx$
int('*f*(*x*)','*x*','*a*','*b*')	halla $\int_a^b f(x)dx$
int(int('*f*(*x*,*y*)','*x*','*a*','*b*'),'*y*','*c*','*d*'))	Halla $\int_a^b \int_c^d f(x,y)dxdy$
syms *x y a b c d*, Int(int(*f*(*x*,*y*),*x*,*a*,*b*),*y*,*c*,*d*))	Halla $\int_a^b \int_c^d f(x,y)dxdy$

En contraste con la diferenciación, la integración simbólica es una tarea más complicada. Una serie de dificultades pueden surgir en el cálculo de la integral:

- La primitiva puede definir una función desconocida.
- La primitiva puede existir, pero el software no puede encontrarla.
- El software puede encontrar la primitiva en una computadora más grande, pero se correr fuera del tiempo o de la memoria de la máquina disponible

Ejemplos : sea la función $f(x)=a/(x^2-1)$

```
>> syms  x;
>> int('a/(x^2-1)','x')
ans =
   -a*atanh(x)
```

Si *f(x)=bln(2-ax^2)*

```
>> syms x a b;
>> pretty(simple(int(b*log(2-b*x^2),x)))
```

$$b\,x\,\log(2 - b\,x^2) - 2\,b\,x + 2\,b^{1/2}\quad 2^{1/2}\quad \operatorname{atanh}(1/2\,b^{1/2}x\,2^{1/2})$$

si la hacemos definida en el intervalo *[0,1]*

```
>>pretty(simple(int(b*log(2-b*x^2), x, 0, 1)))

b log(2 - b) - 2 b + 2 b^1/2    2^1/2   atanh(1/2 b^1/2    2^1/2   )
```

para la función $f(x)= a\ln(1-bx)$su integral doble en *[0,1] , [2 3]* será:

```
>> pretty(simple(int(int(a*log(1+b*x),x,0,1),b,2,3)))
              -2 a + a dilog(3) + a log(256/27) - a dilog(4)
```

Integración con parámetros:

Sea e^{-bx^2} , si se restringe b a reales positivos, se escribe

```
>>syms x
>> syms b positive
>> f = exp(-b*x^2);
>> int(f,x,-inf,inf)

ans =1/b^(1/2)*pi^(1/2)
```

si no hay limitaciones para b real.

```
>>syms b real
>>f=exp(-b*x^2);
>>F = int(f, x, -inf, inf)

F =piecewise([1/b^(1/2)*pi^(1/2), signum(b) = 1], [Inf, otherwise]
```

Con *ezplot(f)* se puede graficar

```
>>syms x
>>a = sym(1/2);
>>f = exp(-a*x^2);
>>ezplot(f)
```

Supongamos tenemos la función $h= x^2\text{-}sin(x^4)$, queriendo hallar la integral en *[0,1]*

```
>> syms x;h=sqrt(x^2-sin(x^4)),int(h,0,2)

h = (x^2-sin(x^4))^(1/2)

 Warning: Explicit integral could not be found.
> In sym.int at 58
ans =
   int((x^2-sin(x^4))^(1/2),x = 0 .. 2)
```

Sin embargo, si escribimos *double*(indicando números de doble precisión) antes de la expresión integral, *MATLAB* retornará el resultado de una integración numérica.

```
>> double(int(h,0,2))
Warning: Explicit integral could not be found.
```

> *In sym.int at* 58
ans =
1.7196

La integración numérica invocada por la combinación de *double* e *int* no es propia de *MATLAB* sino de *MAPLE* del cual han sido tomadas las rutinas de cálculo simbólico de *MATLAB*.

MATLAB también dispone de un integrador numérico denominado *quadl*. Las rutinas *double (int (...))* y *quadl (...)* proporcionan respuestas ligeramente distintas. Como todas las herramientas del cálculo numérico no serán motivos de tratamiento en este impreso, más allá de la necesidad de mencionarlas informativamente y/o el tratamiento de archivos especiales para algunas aplicación del cálculo y/o Análisis.

4.2 ECUACIONES DIFERENCIALES ORDINARIAS (EDOs)

Una ecuación diferencial lineal de orden *n*, se escribe en general como
$a_n(t)y^{n)} + \_\_\_ + a_{1(t)}y^{1)} + a_{0(t)}y = f(t)$:
En particular nos interesa hasta el caso *n* = 2.
La función *dsolve* obtiene las soluciones simbólicas a las EDO. La letra ***D*** denota diferenciación, así *D2, D3,..., N* expresan el orden de la derivada, entonces d^2y/dt^2 se escribe *D2y*.
Los nombres de las variables simbólicas no deben contener *D*, las variables dependientes son todas las precedidas por *D* y por default la independiente es *t*.
Las *CI* (condiciones iniciales) se especifican por ecuaciones adicionales, sino aparecerán *C1, C2, etc*.

Ejemplo

sea *dy/dx=xy*, se escribirá,

*>>dsolve('Dy=x*y','x'),*
*ans =exp(1/2*x^2)*C1*

Ejemplo

Con *CI*: $y'= 1+y^2$, *y(0)=1*, se tiene
y = dsolve('Dy=1+y^2','y(0)=1'),
*y =tan(t+1/4*pi), enter*
*ans =tan(t+1/4*pi)*

Las no lineales tendrán diferentes soluciones, incluso con *CI* dadas**.**

Para una E DOde segundo orden *u''=u', u(0)=1, u'(0)=1*

>>u = dsolve('D2u=u','u(0)=1','Du(0)=-1','x'),
>>u =exp(-x).

4.2.1 Sobre las lineales

Las *ED* lineales de orden superior pueden ser tratadas con el comando *DSOLVE* del *MATLAB*, aunque son relativamente pocas las ecuaciones de coeficientes variables para las que se obtiene información útil.

Ejemplo 1. Resolver, utilizando herramientas *MATLAB*, la ecuación diferencial y homogénea de orden 2 y coefientes constantes.

y'' + 6y' + 12y = 0: (1)

Resolución del ejemplo con *DSOLVE*.

```
>>S=dsolve('D2y+6*Dy+12*y=0')
S =C1*exp(-3*t)*cos(3^(1/2)*t)+C2*exp(-3*t)*sin(3^(1/2)*t)
```

Resolución del ejemplo 1 por la vía de la ecuación característica.

La ecuación característica de (1) es $\gamma^2+4\gamma+\gamma=0$.

```
>>solve('x^2+6*x+12=0','x')
ans = [ -3+i*3^(1/2)]
[-3-i*3^(1/2)]
```

De modo que las raíces son complejas conjugadas. Se concluye que el sistema fundamental de soluciones es $e^{-3t}sen(\sqrt{3}), e^{-3t}\cos(\sqrt{3})$.

Ecuación diferencial lineal completa de orden 2 y coeficientes constantes.

Resolver la ecuación diferencial *y'' + 18y' + 90y = t:*

```
>>S=dsolve('D2y+18*Dy+90*y=t')
S =-1/450+1/90*t+C1*exp(-9*t)*sin(3*t)+C2*exp(-9*t)*cos(3*t)
```

El polinomio característico tiene raíces complejas conjugadas $-9\pm 3i$.

Ecuación diferencial lineal completa de orden 3 y coeficientes constantes.

. Resolver

$t^2y''' - 6ty'' + 5y' = 0$

y(1) = 0; y0(1) = 1; y00(1) = 2:

```
>>y=dsolve('D3y-6*D2y+5*Dy=0','y(1)=0','Dy(1)=1','D2y(1)=2','x')
y=-1/20*(15*exp(1)^5+exp(5))/exp(1)^5+3/4/exp(1)*exp(x)+...
1/20/exp(1)^5*exp(5*x)
```

4.2.2 En general, el comando DSOLVE. Resolución de P.V.I.

El comando *DSOLVE* permite obtener la formulación exacta de la solución general de algunas *E.D.*, sin necesidad de declarar simbólicas a las variables involucradas.

PASO 1) Resolver la *E.D.* $y' = 1 + y^2$

```
>> y=dsolve('Dy=1+y^2')
y =tan(t+C1)
```

PASO 2) Obtener la solución local con la C.I. (0; 1)

```
>> y=dsolve('Dy=1+y^2','y(0)=1')
y =tan(t+1/4*pi)
```

Representar dichas soluciones locales involucradas.

```
>>ezplot('tan(t+pi/4)', [-pi/4, pi/4]);
```

```
>> hold on
>> plot(0,1,'*r')
```

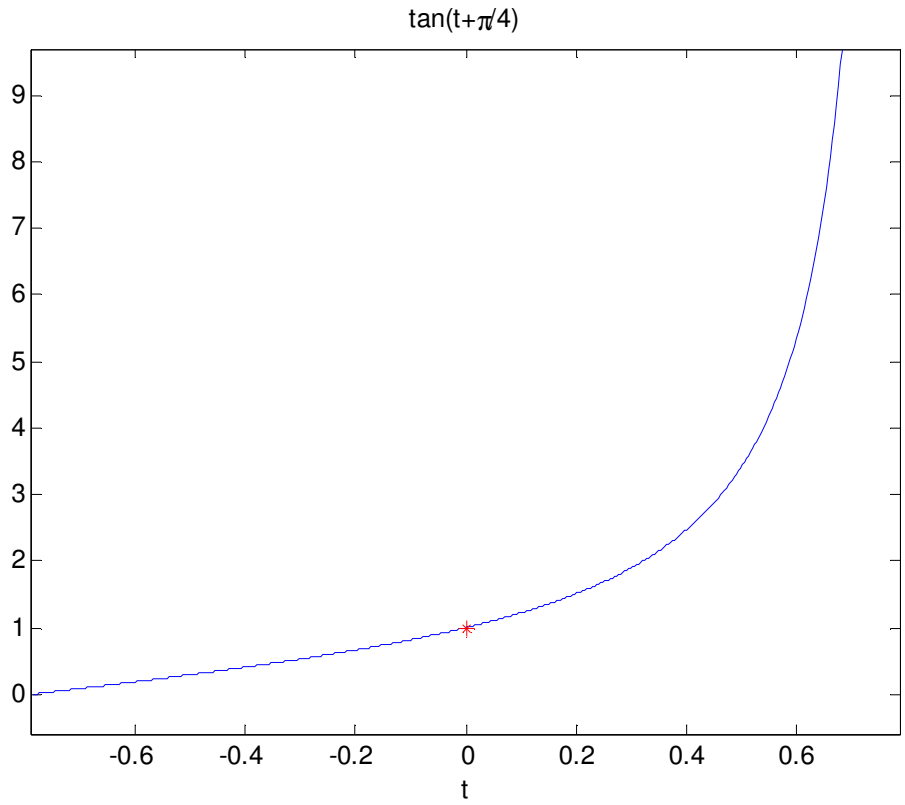


Figura 4.1

Resolver la *E.D.* $y' = -y - 2t$

```
>> y=dsolve('Dy=-y-2*t')
y = -2*t+2+exp(-t)*C1
```

Obtener la solución local con la *C.I.(-2; 1)*

```
>> y=dsolve('Dy=-y- 2*t','y(-2)=1')
y =
   -2*t+2-5*exp(-t)/exp(2)
```

Representar dicha solución local

```
>> ezplot('-2*t+2-5*exp(-t)/exp(2)', [-4,3])
>> hold on
>> plot(-2,1,'*g')
```

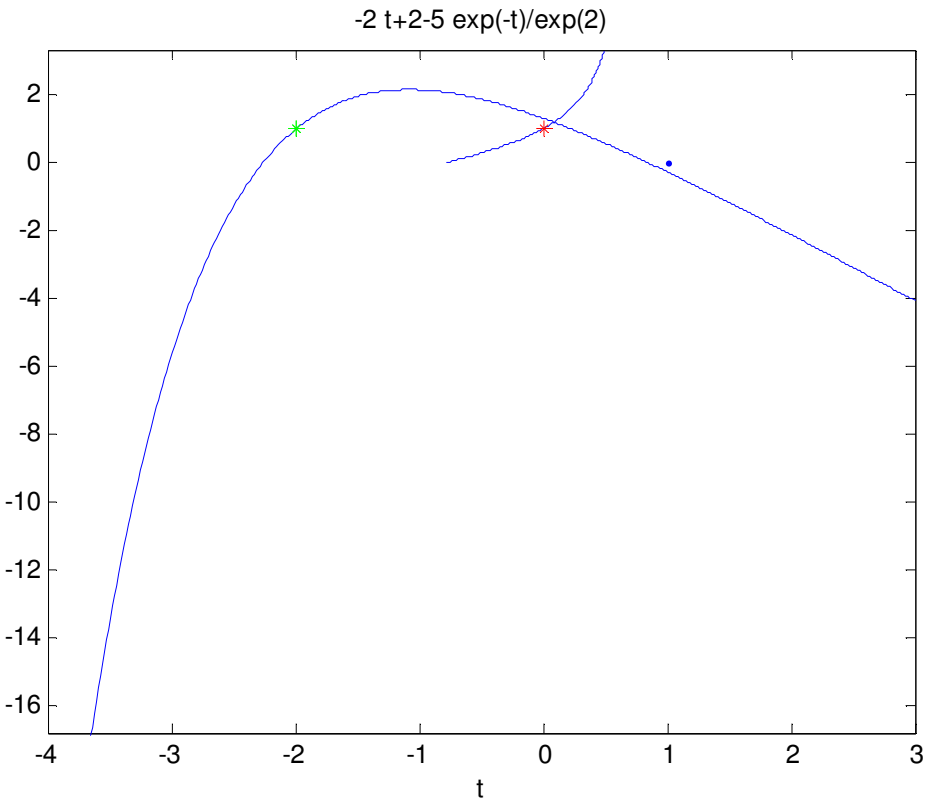


Figura 4.2

Algunas *ED* con solución no pueden ser resueltas por *MATLAB*
y' = -2t - y + cos(y)
*>> dsolve('Dy=-2*t-y+cos(y)')*

Warning:

Explicit solution could not be found.
>In C:\MATLABR11\toolbox\symbolic\dsolve.m at line 326
ans = [empty sym]

En versiones anteriores de *MATLAB*, Si el comando *DSOLVE* no puede obtener explícitamente la solución, entonces el propio *DSOLVE* ofrece la solución en la forma implícita.
Así: $y' = y^2(1 - y)$
>> dsolve('Dy=y^2(1-y)')*
Warning: Explicit solution could not be found;
implicit solution returned.
> In C:\MATLABR11\toolbox\symbolic\dsolve.m
at line 292
t+1/y-log(y)+log(-1+y)+C1=0

Para la que utilizamos, el problema ofrece la solución

>> dsolve('Dy=y^2(1-y)')*

```
ans =
   1/ (lambertw(-1/C1*exp(-t-1)) +1)
```

Donde es $w(z) = Lambertw(z)$ definida implícitamente por l

$lambertw(z)* exp(lambertw(z)) =z$, es decir $w = LAMBERTW(z)$ es la solución a $w*exp(w) = z.z)) = z$

4.3 ORDEN SUPERIOR

Resolver la EDO:
$y'' = -y + cos(2x)$
con la C.I. $y(0) = 1$, $y'(0) = 0$

```
>> y=dsolve('D2y=-y+ cos(2*x)','y(0)=1','Dy(0)=0','x')
y =
  4/3*cos(x)-1/3*cos(2*x)
```

Campo de direcciones de una *ED*.

Dada la *EDO*: $y' = f(x; y)$, si $y = y(x)$ es solución $y(x_0; y_0)$ es un punto de la gráfica de $y(x)$, entonces la pendiente de $y = y(x)$ en $(x_0; y_0)$ es $y'(x_0)$ que es igual a $f(x_0; y_0)$.

Así pues, para conocer la pendiente de la solución que pasa por $(x_0; y_0)$ no hace falta conocer dicha solución; basta con calcular $f(x_0; y_0)$.

Si esto se hace con todos los puntos del plano, tendremos las pendientes de las soluciones que pasan por cada punto del plano. Naturalmente, en la práctica es imposible hacer esto con todos los puntos del plano, pero nada nos impide hacerlo con tantos puntos como querramos, configurando un dibujo que se llama campo de direcciones de la *ED*.

Se emplean herramientas que volvemos a ver en funciones vectoriales

Sea $n = 1$ y sea $y'(x) = f(x; y(x))$. Para construir el *CAMPO DE DIRECCIONES* de la *ED*, por cada punto $(x; y)$ de una red de puntos de R^2 se dibuja un segmento o vector de pendiente $f(x; y)$.

Para el estudio gráfico de las soluciones de una *ED* sin haberla resuelto, usaremos un *script* en el que intervienen los comandos *meshgrid* y *quiver*, que permiten dibujar el campo de direcciones.

Los vectores a considerar para obtener el campo de direcciones serán $(1; y0) = (1; f(x; y))$.

```
>> f=inline('x','x','y');
>> paso=0.5;iz=-3;der=3;
>> [x,y]=meshgrid(iz:paso:der,iz:paso:der);
>> [n, m] =size(x);
>> dx=ones(n,m);
>> z=f(x,y);dy=z;
>> quiver(x,y,dx,dy)
```

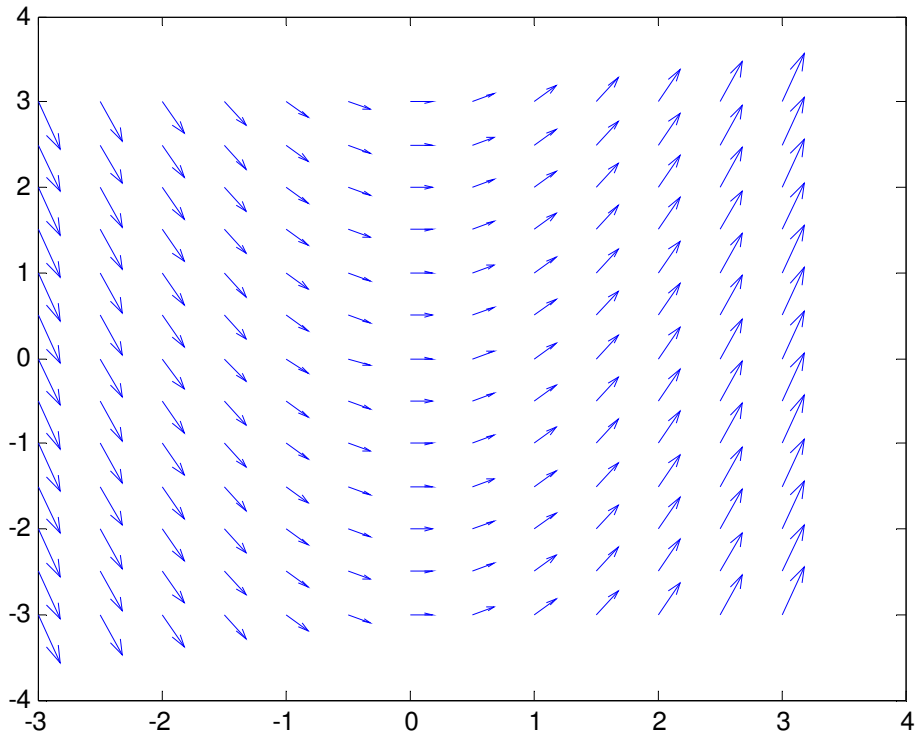


Figura 4.3

se desea representar la solución particular que verifica la CI y (0)=0:

>> *dsolve('Dy=x','y(0)=0', 'x')*
>>*ezplot*('1/2*x^2', *[-2.5, 2.5])*,
>>*hold on* % *hold* on permite superponer los gráficos
>>*plot*(0, 0,'*g')
ans =
*1/2*x^2*

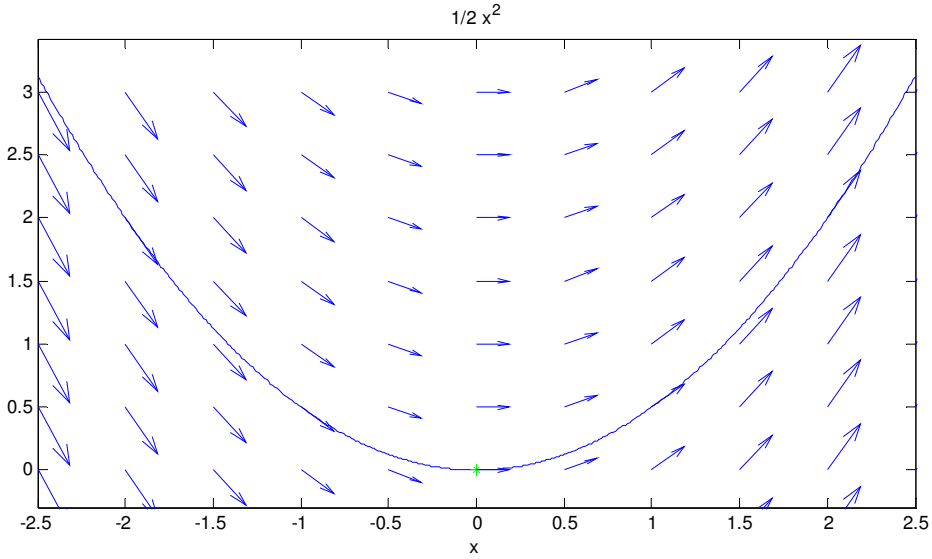


Figura 4.4

Dada la *EDO* $y' = f(x; y)$, se llaman curvas de nivel o isoclinas a las obtenidas al imponer la condición $y' = k$.

4.3.1 Método de las isoclinas

Es una variante de las ideas antes descritas. Los puntos del plano por los que pasa una solución con pendiente k, son los puntos de la curva de ecuación $f(x; y) = k$(isoclina de pendiente k).
Dibujando las distintas isoclinas se obtiene una representación similar a la del campo de direcciones. Puede tener interés identificar la isoclina para la pendiente 0 pues las soluciones tendrán generalmente un máximo o un mínimo al pasar por esta isoclina.
Representar las isoclinas de la *ED* $y' = x + y^2$.
Representar el gráfico de contorno (curvas de nivel o isoclinas) de la superficie $z = x + y^2$.

Desarrollo de la solución:

```
>>[x,y]=meshgrid(0:0.05:3,-2:0.05:2);
>> z=x+y.^2;
>> isoclinas= contour(x,y,z,20)
```

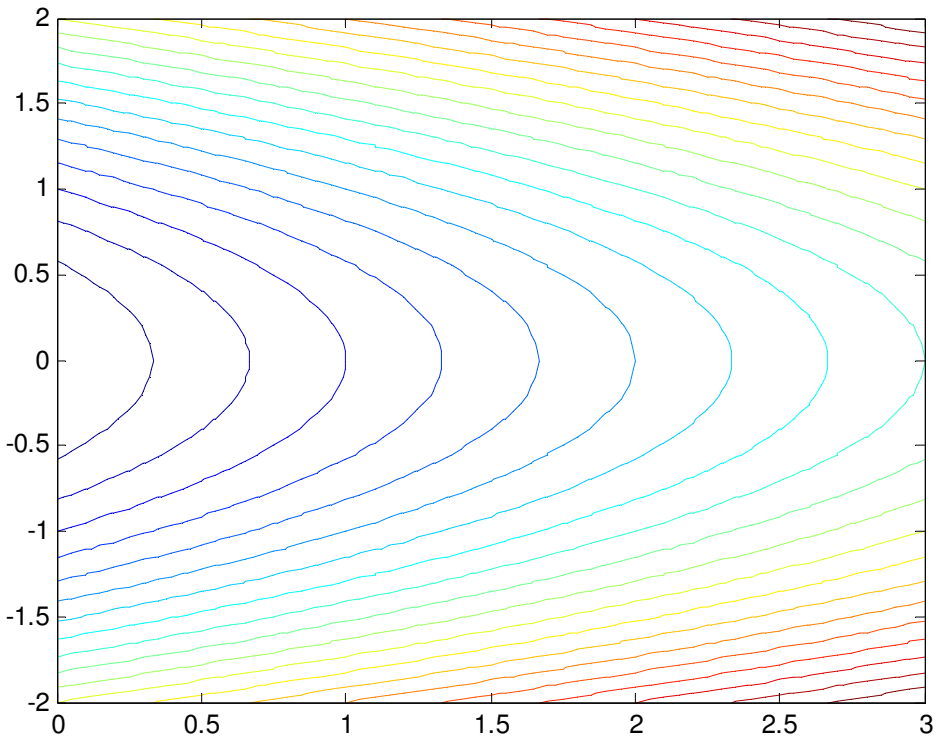


Figura 4.5

Dibujar las curvas de nivel de $y'=x^2+y^2$ (prestar atención a la función muy familiar)

```
>>[x,y]=meshgrid(-4:0.05:4);
>>z=x.^2+y.^2;
>>isoclinas=contour(x,y,z,20)
```

En la línea de comandos aparecen los pares que dieron origen a las curvas

Construir el campo de direcciones y las curvas de nivel de la *ED* y0 = *sen(x)* + y

```
>> f=inline('sin(x)+y', 'x','y');
>> paso= 0.5;iz=-3;der=3;
>> [x,y]=meshgrid(iz:paso:der,iz:paso:der);
>> [n,m] =size(x); dx=ones(n,m); z=f(x,y); dy =z;
>> hold on, contour(x,y,z,20); quiver(x,y,dx,dy);
```

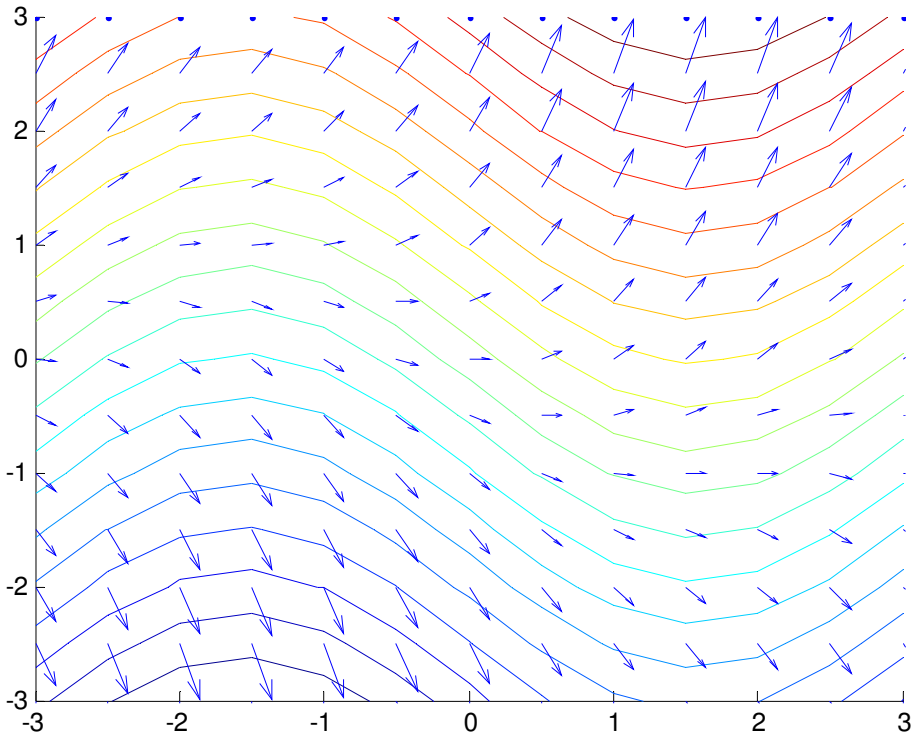


Figura 4.6

4.4 SISTEMAS DE EDOS

Suponiendo un par de ecuaciones lineales de primer orden, dadas por:

```
>>S = dsolve('Df = 3*f+4*g', 'Dg = -4*f+3*g')
S = f: [1x1 sym]
g: [1x1 sym]
```

Las soluciones se retornan en la estructura S, para hallar *f* y *g* se escribe:

```
>>f = S.f
f =exp(3*t)*(C1*sin(4*t)+C2*cos(4*t))
```

igual para g:

```
>>g=S.g, enter
g =-exp(3*t)*(-C1*cos(4*t)+C2*sin(4*t))
```

El empleo de la herramienta numérica en *MATLAB*, lleva al uso de los archivos *ODEss*

4.5 ARCHIVOS ODE

El archivo *ode* no es un comando ni función, es una entrada (ayuda) que señala como crear un archivo M definiendo el sistema de ecuaciones a resolver.
En la documentación de *MATLAB*, este archivo M está referenciado como un archivo *ODE* aunque se lo puede nombrar como se quiera.
Se puede usar el archivo *ode* M para definir un sistema de ecuaciones diferenciales como:
$y` = f(t,y)$ *o* $M(t,y)y' = f(t,y)$, donde:
t es una variable escalar independiente (tpo)
y es el vector de variable dependiente
f es una función de *t* e *y* que retorna un vector columna de la misma longitud de *y*
M(*t*,*y*) matriz dependiente del estado y el tiempo

El archivo *ODE* debe aceptar los argumentos *t* e *y*, aunque no tenga que usarlos.
Por default *ODE* debe retornar un vector columna de igual longitud que *y* todos los resolvedores de *ODE* pueden resolver $M(t, y)\, y' = f(t, y)$, menos ***ode23s***, que sólo resuelve problemas con matrices de masa constante.
Los resolvedores *ode15s* y *ode23t* pueden resolver algunas ecuaciones diferenciales algebraicas del tipo $M(t)\, y' = f(t, y)$, (*DAEs*).

CAPITULO 5 - FUNCIONES VECTORIALES

5.1 CURVAS EN EL ESPACIO

Representación paramétrica

Por ejemplo para graficar un círculo con centro en (1,1) y radio *r*=2:

```
>>t=linspace (0,2*pi, 101);
>>x=1+2*cos(t);
>>y=1+2*sin(t);
>>plot(x,y)
```

También se puede usar la función *inline*:

```
>>x=inline('1+2*cos(t')';
>>y=inline('1+2*sin(t)');
>>t=linspace(0,2*pi, 101);

>>plot(x(t), y (t))
```

Incluso declarando con *syms* la variable *t*, usando *ezplot (su rango por default es* 0-2π*)*

Para una hélice circular, en 3D:

```
>>t=linspace(0,4*pi, 201);
>>plot3 (cos(t), sin(t), t/ (2*pi))
```

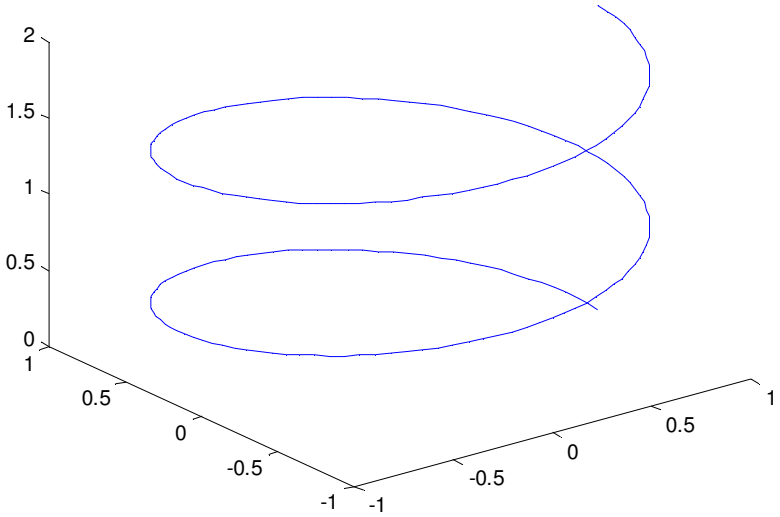


Figura 5.1

Ahora, si las funciones son más complicadas es preferible la entrada *inline* o por archivos *m*.

Si queremos agregar vectores tangentes a lo largo de varios puntos de un gráfico, recordando que *v(t)= [x'(to), y'(to), z'(to)]*, tomando por ejemplo los tiempos: 0, *pi*/4, *pi*/2...4*pi*, para la hélice, se escribe el script *animation*:

```
>>animation
```

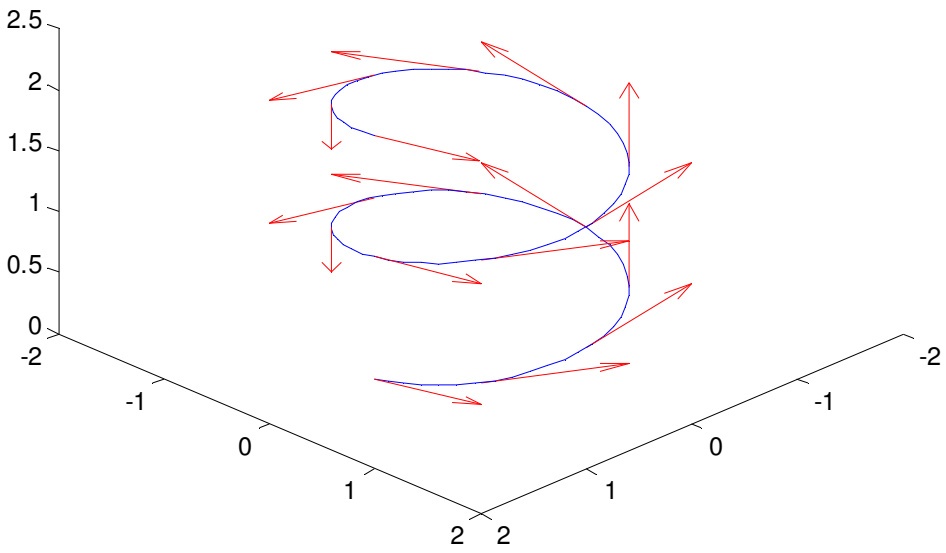


Figura 5.2

Para mostrar desde el inicio hasta final de la curva

>>*ezplot3(x,y,z,[0,4*pi],'animation')*

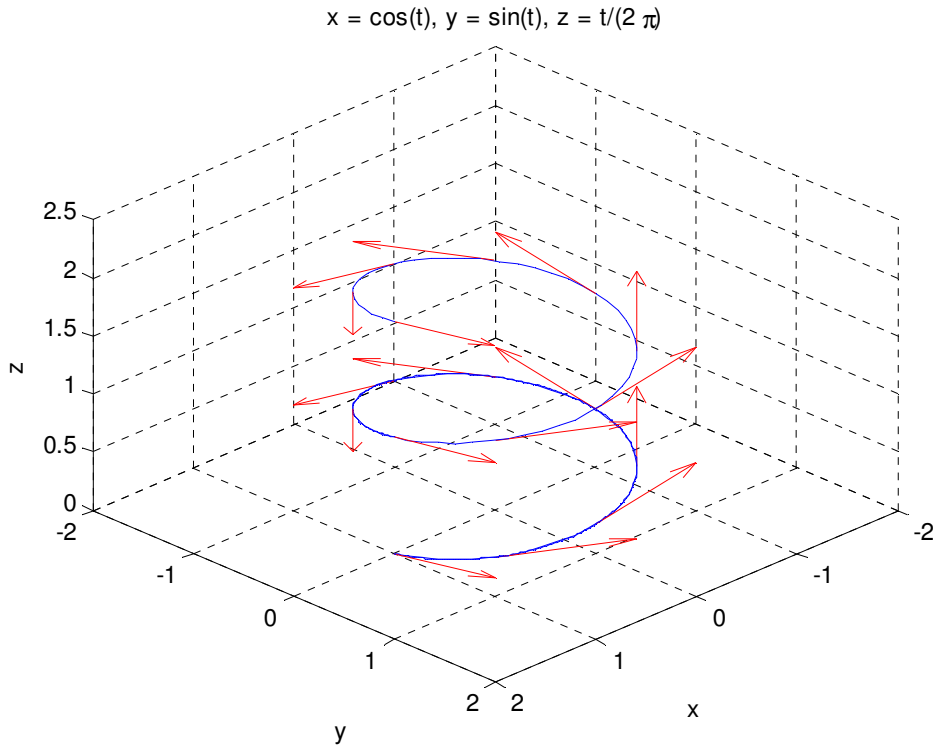


Figura 5.3

5.2 LONGITUD DE ARCO

Tomando una curva parametrizada por *r(t) = [2t, t2, ln(t)]* para *t* en *[0,1]*, se genera un *script* para la fórmula de las sumas o por la variante simbólica:

```
>> syms t b
>> r=
   [2*t, t^2, log(t)]

r =
   [2*t, t^2, log(t)]

>> v=diff(r)

v =
   [2, 2*t, 1/t]

>> velocidad= sqrt(v(1)^2+v(2)^2+v(3)^2)

velocidad =
    (4+4*t^2+1/t^2) ^ (1/2)
>>int(velocidad, t, 1, 2)
ans =
   3+log(2)=3.6931
```

5.3 VECTOR TANGENTE, BINORMAL

Con el empleo *frenet.m* , se muestran los *vectoresT(tangente),N(normal principal)* y *B(binormal)* en un punto elegido de la curva y en la pantalla se muestra la matriz cuya primera columna es *T*, la segunda es *N* y la tercera es *B*: deben definirse las coordenadas de la función *x(t),y(t),z(t)* como archivo *m* o *inline*.

```
>> x=inline('2*t');
>> y=inline('t.^2');
>> z=inline('t.^3/2');
>> t=0:0.1:2;
>> plot3(x(t), y(t), z(t));
>> axis equal
>> hold on
>> frenet(x,y,z)
enter a value of  t   0

t =
     0
     T          N          B

frame =
       1.0000         0        -0.0000
```

```
         0    1.0000         0
    0.0000         0    1.0000
```

```
kappa       a_T          a_N
ans =
    0.5000         0    2.0000
enter a value of  t   .8
t =
    0.8000

      T           N           B

frame =

    0.7312   -0.6157    0.2937
    0.5850    0.3444   -0.7343
    0.3510    0.7088    0.6119
```

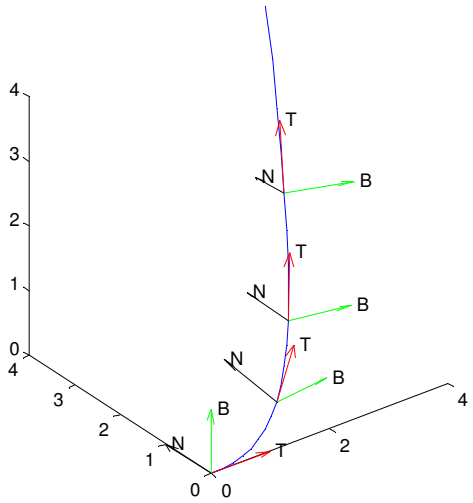


Figura 5.4

5.4 ROTACIONES EN EL PLANO

Sea una elipse $(x^2/9) + (y^2/4)=1$, parametrizada con $r(s) = [3\cos s,\ 2\ \sin s]$, entre $0\text{-}2\pi$; rotaremos un ángulo θ, antihorario:

```
>> theta= pi/4;
>> s=linspace(0, 2*pi, 101);
>> x0=3*cos(s);y0=2*sin(s);
>> % elipse sin rotar
>> plot(x0,y0)
>> hold on
>> % rotamos y dibujamos de nuevo
>> x=cos(theta)*x0-sin(theta)*y0;
```

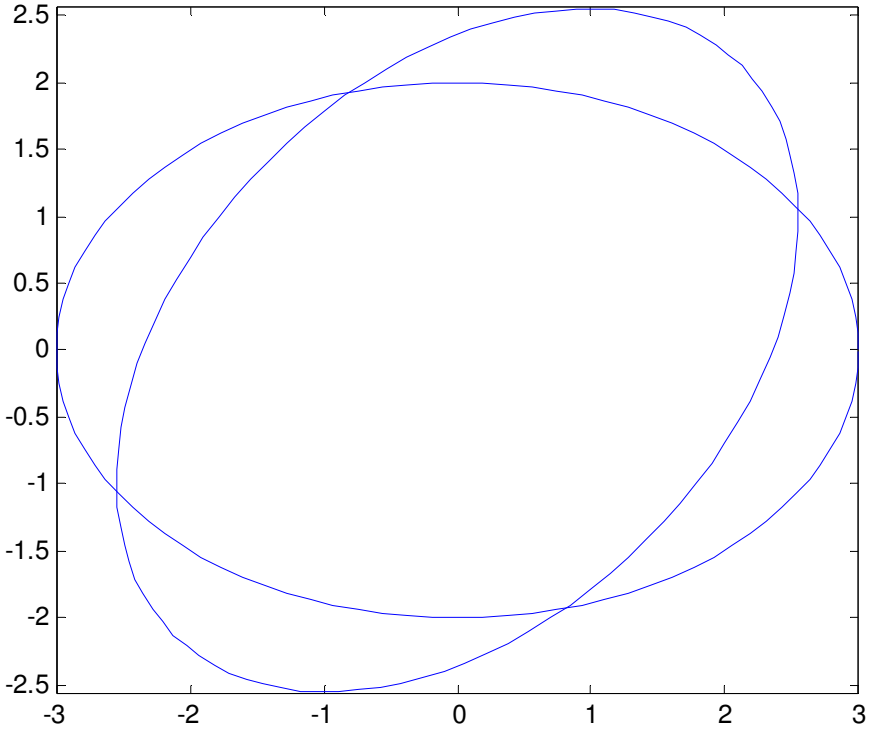


Figura 5.5

5.5 FUNCIONES NUMERICAS DE DOS VARIABLES

Sea $f(x,y) = x + ye^{x^2} + y^2$, se puede usar un archivo m

```
function z=f(x,y)
z=x+y.*exp(x.^2+y.^2); % se usa array con,* y.^, para operar con matrices
```

como inline:

```
f=inline('x+y.*exp(x.^2+y.^2)','x','y')
```

Se prefiere el archivo *m* para funciones más complicadas

5.6 GRAFICACION DE FUNCIONES DE DOS VARIABLES

Si vamos a graficar sobre dominios rectangulares, es necesario crear las matrices

```
[X, Y]=meshgrid(x,y)
```

Entones, si usamos el modo *inline*:

```
>>x=-1:.4:1; y=0:.4:4;
>> [X,Y]=meshgrid(x,y);
>> f=inline('10*x.^ 2+y.^2','x','y');
>> Z=f(X,Y)

Z =

  Columns 1 through 4

   10.0000   10.0000   10.0000   10.0000
```

10.1600 10.1600 10.1600 10.1600
10.6400 10.6400 10.6400 10.6400

.....

El gráfico lo hacemos con

>>*surf(X,Y,Z)*

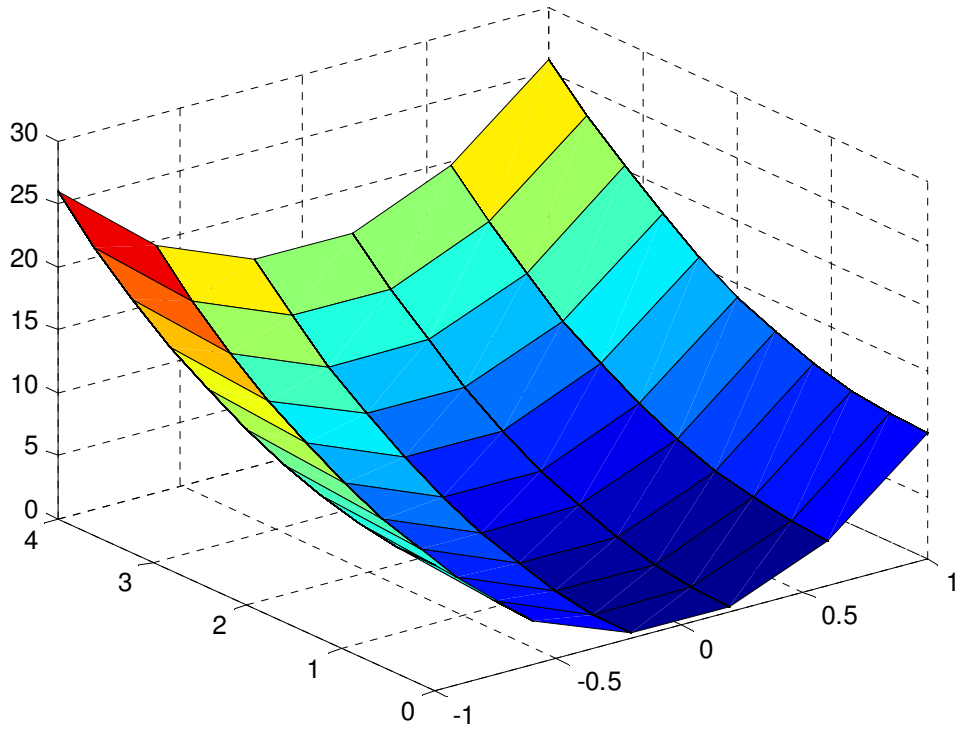


Figura 5.6

Podemos afinar, cambiando *x* e *y*:

>>*x=-1:.2:1, y=0:.2:4, [X,Y]=meshgrid(x,y); surf(X,Y, f(X,Y))*

Con el archivo qsurf, es más rápido este tipo de graficación, llamándolo:

qsurf(f,esquinas,n), f es un archivo *m*, esquinas es el vector de extremos, *n* el número de subdivisiones)

5.6.1 Uso de coordenadas polares

Tomemos la *función*= x-1 +y^2 sobre el disco $(x-1)^2+ (y-3)^2$=4.Generando el script

```
r=linspace(0,2 ,21);
theta =linspace(0,2*pi,41);
[R, TH]=meshgrid(r, theta);% la malla sobre r y las coordenadas en theta
% conversión en coordenadas curvilíneas
X=1+R.*cos(TH);
Y=3+R.*sin(TH);
Z=X-1+Y.^ 2;
surf(X,Y,Z)
hold on
%agregamos un plano z=5 con la matriz curvilínea
surf(X,Y,-4.5+0*Z)
```

hold off

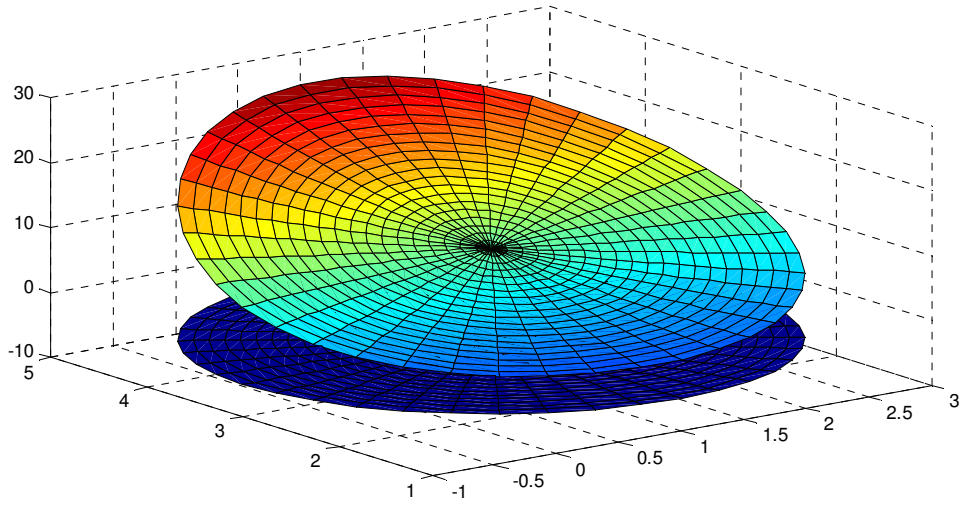


Figura 5.7

5.6.2 Curvas de nivel (líneas de contorno)

En *MATLAB* haremos uso de *meshgrid(x,y)* y *contour(X,Y,f(X,Y))* para el graficado de las líneas de contorno sobre el dominio rectangular. Por ejemplo

Sea $f(x,y) = e^{-x^2-y^2} + x + y$ sobre *[-2, 2] x [-2, 2]*

```
>>x=-2:.05:2; y=-2:0.05:2;
>>[X,Y]=meshgrid(x,y);
>>f=inline('exp(-x.^2-y.^2) +x+y','x','y') ;
Z=f(X,Y) ;
Pcolor(X,Y,Z)
hold on
contour(X,Y,Z, 10,'k')
xlabel(' eje x')
ylabel('eje y');hold off
```

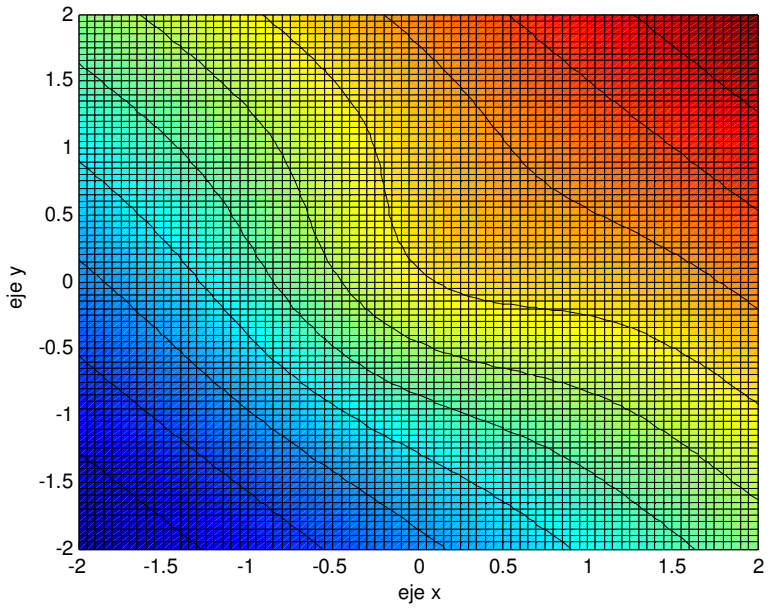


Figura 5.8

5.6.3 Graficación para funciones definidas simbólicamente

Las funciones para tal fin son: *ezsurf*, *ezmesh* y *ezcontour*, Así por ejemplo:

>>*syms x y*

>> *f= sin(x)*exp(x-y)*

>> *ezsurf(f)*

Ahora si damos el dominio *[a, b, c, d]*, se ingresa como parámetro de entrada

>> *ezsurf(f,[4,6,0,2])*

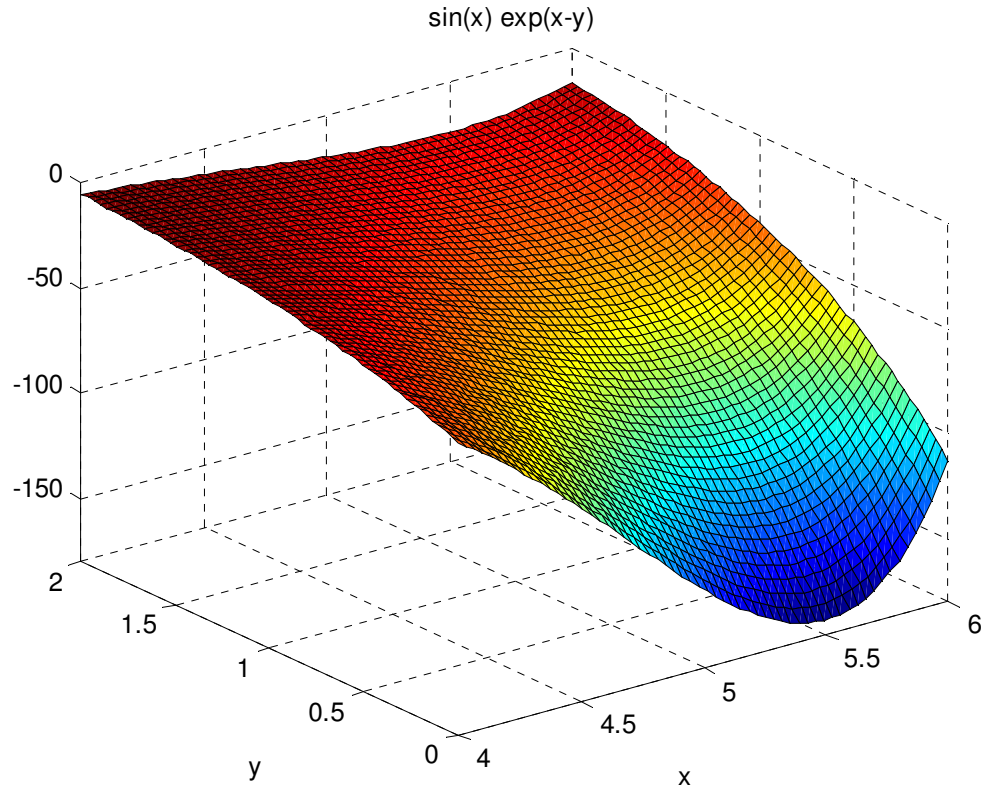


Figura 5.9

5.7 DERIVADAS PARCIALES Y DERIVADAS DIRECCIONALES

Primeramente hacemos uso de las instancias simbólicas de *MATLAB* para funciones de variable vectorial

Sea la función $f(t, v)=cos(tv)+sen(tv^2)$, hallar las derivadas parciales respecto a t y v

```
>>syms t v;
>>f=cos(t*v) +sin(t*v^2) ;
>>diff(f, t)
ans =
   -sin(t*v)*v+cos(t*v^2)*v^2
>>diff(f, v)
ans =
    -sin(t*v)*t+2*cos(t*v^2)*t*v
```

Para la segunda respecto a t

```
>>diff(diff(f, t), t)
ans =
   -cos(t*v)*v^2-sin(t*v^2)*v^4
```

Una mixta, primera respecto *at* y luego a $v(\delta f/\delta t\delta v)$

```
>>diff(diff(f,t),v)
ans =
```

```
-cos(t*v)*t*v-sin(t*v)-2*sin(t*v^2)*t*v^3+2*cos(t*v^2)*v
```

Dada la función vectorial

$$u(x,y)=\frac{x^4+y^4}{x}, v(x,y)=sen(x)+\cos(y)$$

Hallar la matriz derivada de la función

```
>>syms x y;
>> J=simple((jacobian([(x^4+y^4)/x,sin(x)+cos(y)],[x y])))

J =

[ (3*x^4-y^4)/x^2,          4*y^3/x]
[          cos(x),          -sin(y)]
>>  pretty(det(J))

                  3 sin(y) x^4 - sin(y) y^4  + 4 y^3  cos(x) x
                 ---------------------------------------------
                                      x^2
```

Si nos interesa saber si tendrá inversa, el determinante deberá no anularse

Cuál será la derivada de la función inversa? Será la matriz inversa de la jacobiana inicial

```
>>I=simple(inv(J))
I=simple(inv(J))

I =

[ sin(y)/(3*sin(y)*x^4-sin(y)*y^4+4*y^3*cos(x)*x)*x^2, 4*y^3*x/(3*sin(y)*x^4-
sin(y)*y^4+4*y^3*cos(x)*x)]
[ cos(x)/(3*sin(y)*x^4-sin(y)*y^4+4*y^3*cos(x)*x)*x^2, -(3*x^4-y^4)/(3*sin(y)*x^4-
sin(y)*y^4+4*y^3*cos(x)*x)]
```

Se muestra el uso de los archivos *xslice, yslice*.

Sea $f(x,y)=1-(x^2+y^2)/2$ para $x\geq 0$, $y\leq 2$, con el plano $y=1$

```
>> f=inline('1-0.5*(x.^2+y.^2)','x','y');
>> qsurf(f,[0 2 0 2]);shading flat
>> hold on
>> x=linspace(0,2,51)
>>hold off
```

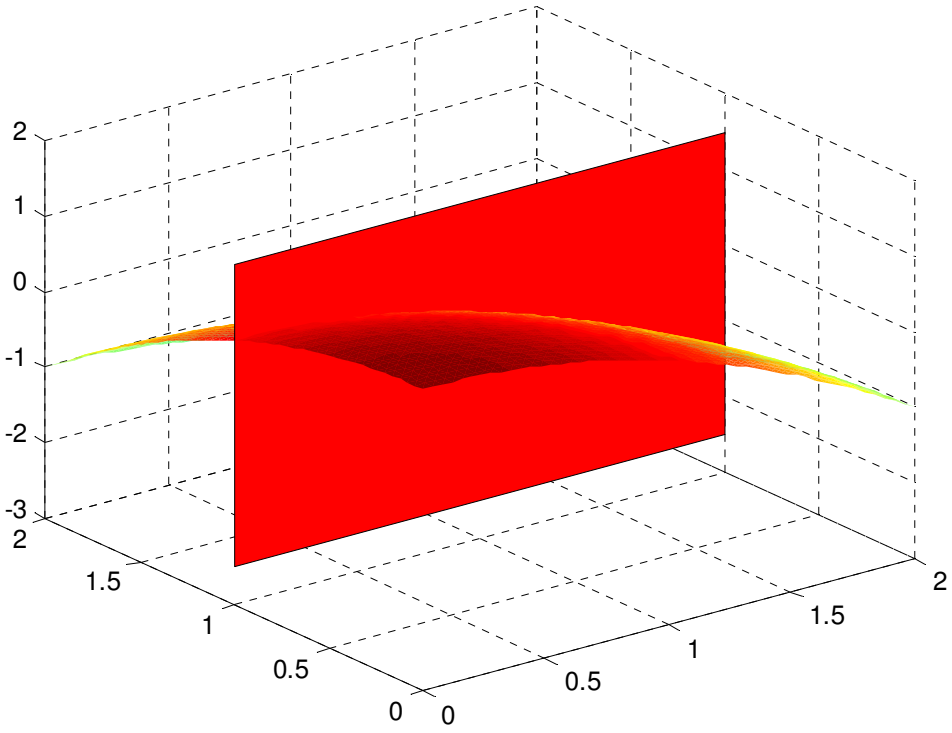


Figura 5.10

El archivo *mslice* muestra el comportamiento de la función *f* en la dirección ***u*** intersectando el gráfico de *f* con una parte del plano *vertical(paralelo a z)*
$(x\text{-}x_0)\ u_2 = (y\text{-}y_0)\ u1$. al llamar *('f',P)*, *P* es el punto donde se calcula la derivada direccional, Sea el ejemplo $f=x^2+y$ en el punto *P*=(1,2)

```
>>f=inline('x.^2+y','x','y')
>>P= [1, 2]
>>mslice(f, P)
```

Aparece: This is a chance to enlarge the window, and rotate the figure (nos situamos en un punto)

When you are finished, enter return

```
u1        u2        Duf
ans =
   -0.2172   -0.9761   -1.4105
ans =
   0.9481    0.3180    2.2142
```

Nos fue mostrando las figuras

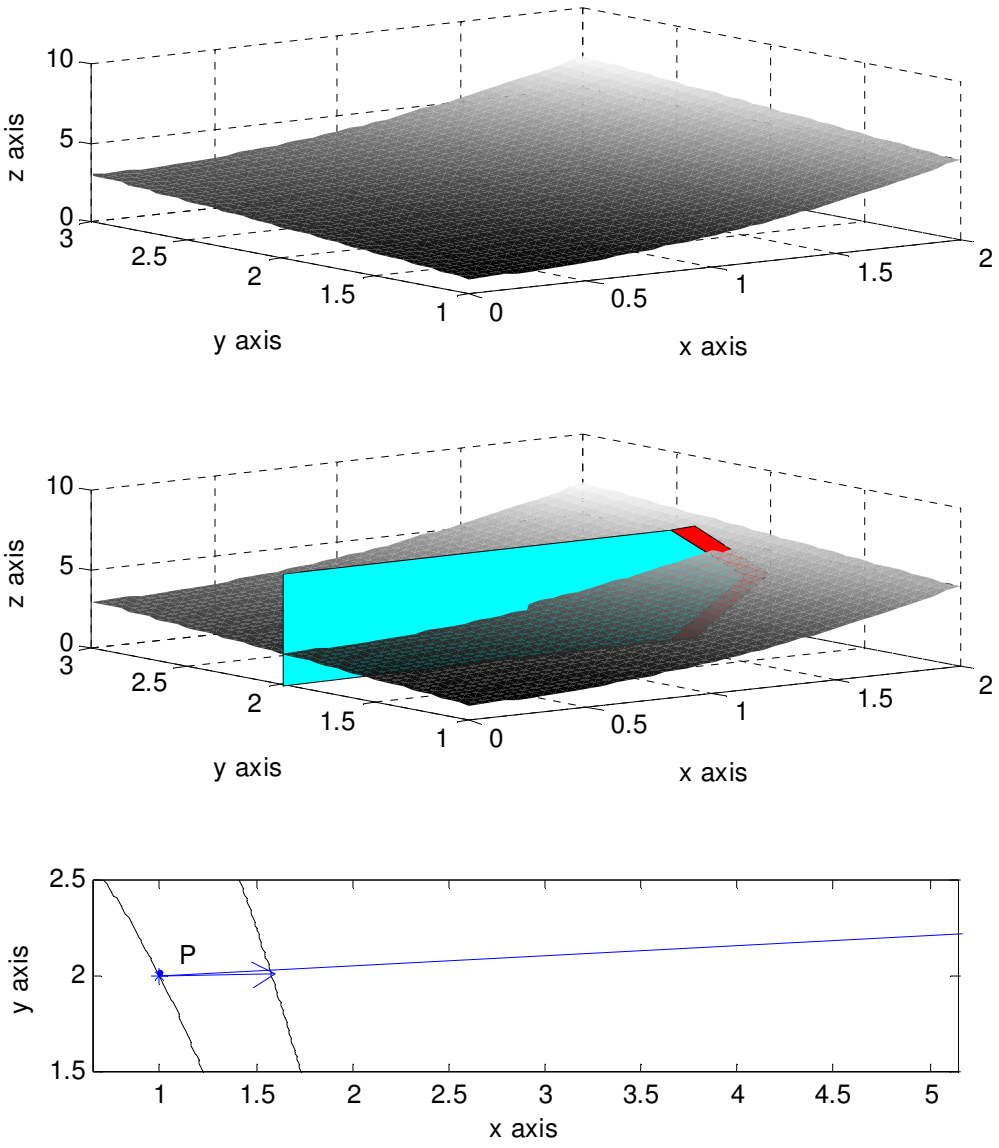


Figura 5.11

En la parte superior nos muestra un plano vertical que esta sobre la flecha del punto marcado desde *P(Q)*; desde la curva de intersección del gráfico y el plano *xy* se puede ver si la función crece o decrece en la dirección, lo marca la flecha roja sobre el plano, se aproxima en diferencias finitas la $D_u f$ en *P*

5.8 VECTOR GRADIENTE Y CURVAS DE NIVEL

Usaremos el comando quiver de *MATLAB*, que ubicará una flecha en cada punto de la malla que definamos, si hay muchos puntos la malla *x,y* las flechas se solapan. Generemos un script para una función y un dominio

```
>> f=inline('x.*y-(x.^3)/3','x','y')
f =
     Inline function:
     F(x,y) = x.*y-(x.^3)/3
>> fx= inline('y-x.^2','x','y')%derivadas parciales
```

```
fx =
    Inline function:
     fx(x,y) = y-x.^2
>> fy= inline('x','x','y')
fy =
    Inline function:
     fy(x,y) = x
>> x=-2:.05:2; y=x;
>> [X,Y]=meshgrid(x,y);
>> Z=f(X,Y); %elegimos las curvas de nivel
>> levels= [-6:.5:6];
>> contour(X,Y,Z,levels)
>> hold on
>> xx=-2:.2:2; yy=xx; %la malla gruesa para las flechas
>> [XX,YY]=meshgrid(xx,yy);
>> U=fx(XX,YY); V=fy(XX,YY);
>> quiver(XX,YY,U,V)
>> axis equal
```

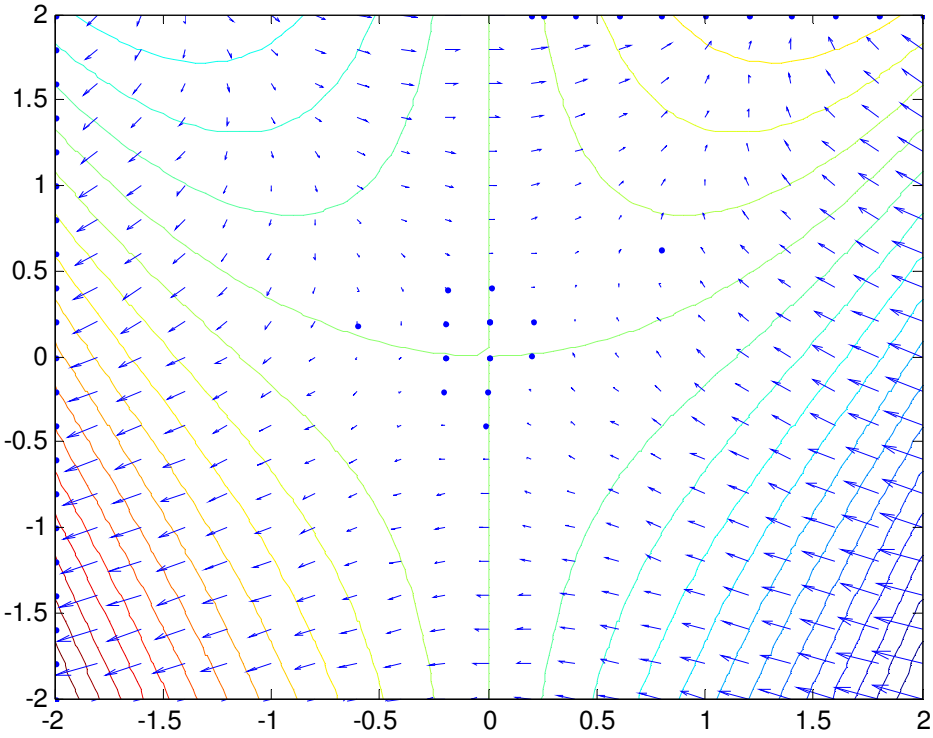


Figura 5.12

Empleando la función gradient de *MATLAB*
[FX, FY, FZ,...] = gradient(F) donde *F* tiene *N* dimensiones, devolviendo las *N* componentes del gradiente de *F*. Para controlar el espacio entre los valores de *F*, se puede hacer:

* Un valor de interlineado sencillo, *h*, especifica la distancia entre los puntos en cada dirección.

* valores de *N* espaciamiento *(h1, h2,...)* especifica el espacio para cada dimensión de *F*. Los parámetros de espacio escalar especifican una distancia constante para cada dimensión. Parámetro vectorial especifica las coordenadas de los valores a lo largo de las dimensiones correspondientes de *F*. En este caso, la longitud del vector debe coincidir con el tamaño de la dimensión correspondiente
Ver que la primera salida *FX* es siempre el gradiente a lo largo de la segunda dimensión de *F*, yendo a través de las columnas. La segunda salida *FY* es siempre el gradiente a lo largo de la primera dimensión de *F*, pasando a través de las filas. Para la tercera salida *FZ* y los resultados que siguen, el resultado es el enésimo gradiente a lo largo de la dimensión n de *F*
[...] = gradient(F, h) con *h* un scalar que usa h como el espaciado entre los puntos en cada dirección.
[...] = gradient(F, h1, h2,...) con *N* parámetro de espaciamiento que especifica el espaciamiento para cada dimensión de *F*

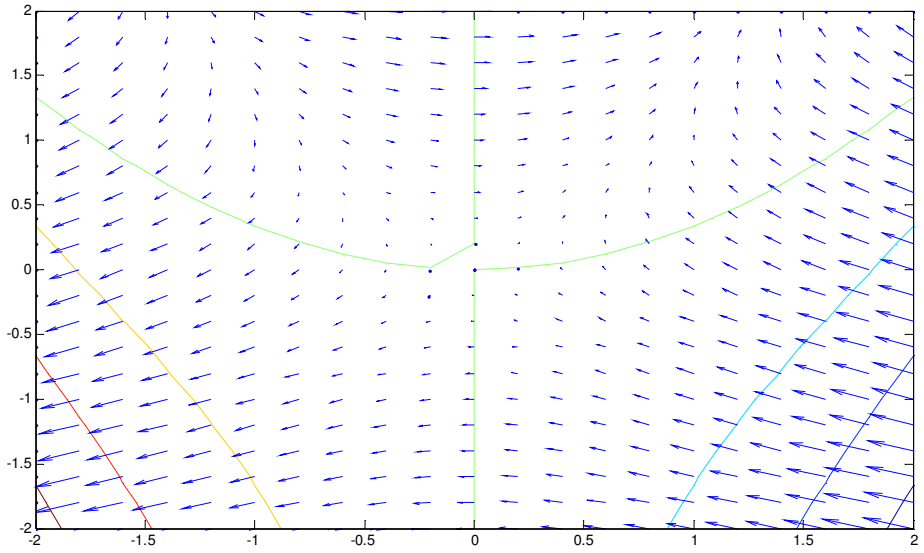


Figura 5.13

Desde *MATLAB*, para la derivada direccional
df = fndir(f,y) es la *ppform* de la derivada direccional, de la función en *f*, en la dirección del vector columna *(-)* y. Esto significa que *df* describe la función
Si *y* es una matriz con *n* columnas, y es *d-valuada*, entonces la función en *df* es *prod(d) * n-valor*. Su valor en, adecuado al tamaño *[d, n]*, tiene en su *th* columna 'la derivada direccional en la dirección de la *th* columna de *y*. O si se desea que *dfF* refleje explícitamente el tamaño real usar en su lugar

df = fnchg(fndir(f,y), 'dim',[fnbrk(f,'dim'),size(y,2)]);

Como *fndir* se basa en la *ppform* de la función de *f*, no funciona para las funciones racionales, ni para las funciones en *stform*
Ejemplo:
Por ejemplo, si *f* describe un *m-d-variate* función vectorial y *x* es un punto en su dominio, entonces, por ejemplo, con este particular, *ppform f* que describe un polinomio *bilineal* de valores escalares

```
>>f = ppmak({0:1,0:1},[1 0;0 1]); x = [0;0];
>>[d,m] = fnbrk(f,'dim','var');
>>jacobian = reshape(fnval(fndir(f,eye(m)),x),d,m)
```

es el jacobiano de esa función en ese punto (que, para esta particular función escalar valor, es su gradiente, y es igual a cero en el origen).

Como un ejemplo relacionado, la sentencia grafican los gradientes (una buena aproximación a la función) de la función Franke en una malla regular

```
>>xx = linspace(-.1, 1.1, 13); yy = linspace(0, 1, 11);
>>[x,y] = ndgrid(xx,yy); z = franke(x,y);
>>pp2dir = fndir(csapi({xx,yy}, z), eye(2));
>>grads = reshape(fnval(pp2dir, [x(:) y(:)].'),...
[2, length(xx), length(yy)]);
>>quiver(x,y,squeeze(grads(1,:,:)),squeeze(grads(2,:,:)))
```

Resultando:

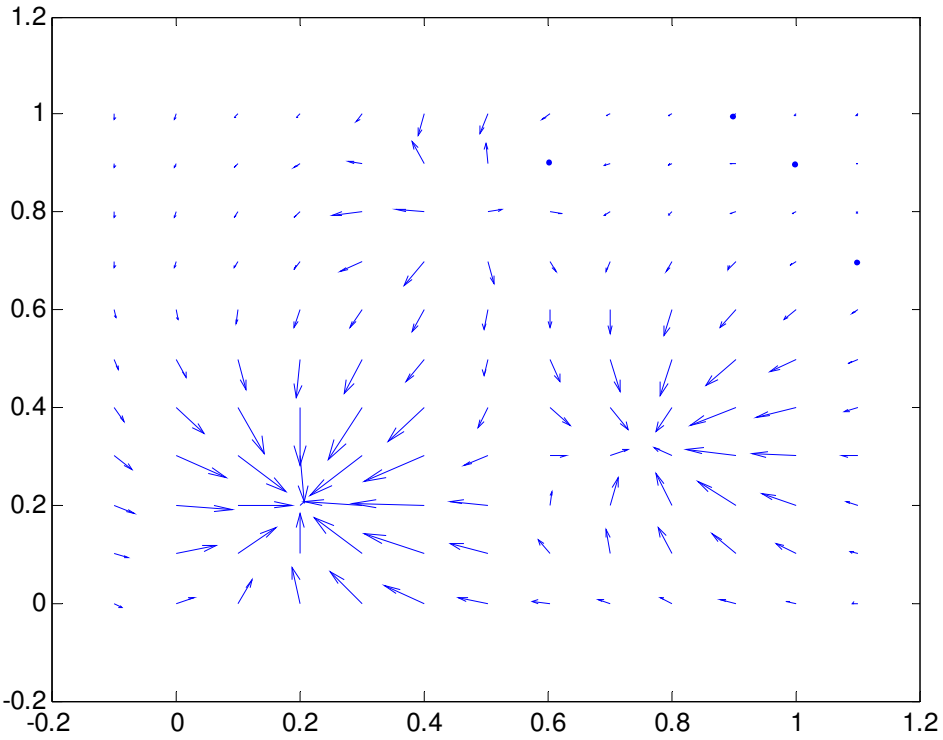


Figura 5.14

5.9 LA APROXIMACION DEL PLANO TANGENTE

Permite a veces aproximar complicadas funciones no lineales por funciones lineales más simples, por lo menos localmente

Ejemplo: encontrar la aproximación del plano tangente a $f(x,y)=(1-y^2)\cos x$ en el punto $(x_0,y_0)(.2,-.4)$, sus derivadas parciales son $f_x(x,y)=-(1-y^2)senx$, $f_y(x,y_)=2y\cos x$.

El plano tangente a f en $P_0=(.2,-.4,f(.2,-.4))$ es

$Z= f(.2,-.4)+ f_x(.2,-.4)(x-.2)+ f_y(.2,-.4)(y+.4)=.8233-.1699(x-.2)+.7481(y+.4)$

Con vector normal $N=[- f_x(x_0y_0),f_y(x_0y_0) ,1]=[.1699,-.7841,1]$

Graficando en el dominio *[-1≤x,y≤1]* y adjuntamos el plano tangente;que lo graficamos sobre el menor cuadrado*{ | x-.2 | , | y+.4 | ≤.5}*

```
>> f=inline('(1-y.^2).*cos(x)','x','y')
f =Inline function:
     f(x,y) = (1-y.^2).*cos(x)
>> l=inline('.8233-.1699*(x-.2)+.7481*(y+4)','x','y')
1= Inline function:
     l(x,y) = .8233-.1699*(x-.2)+.7481*(y+4)
>> qsurf(f,[-1,1 -1,1])
>> hold on
```

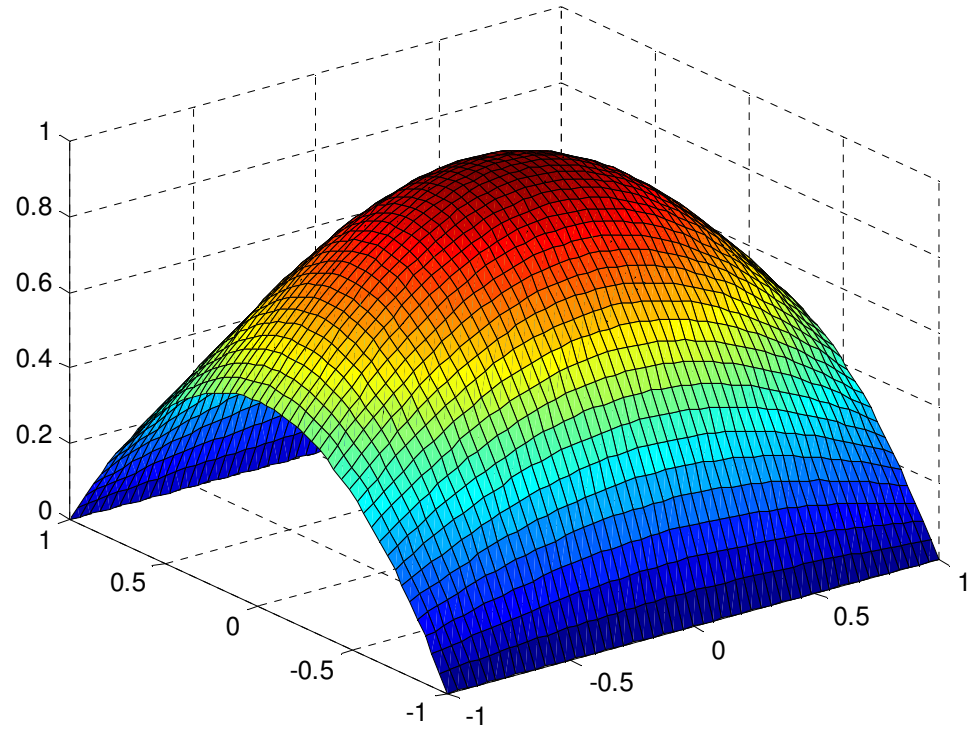


Figura 5.15

```
>> qsurf(l,[-.3,-.7 -.9,.1],10)
>> hold off
```

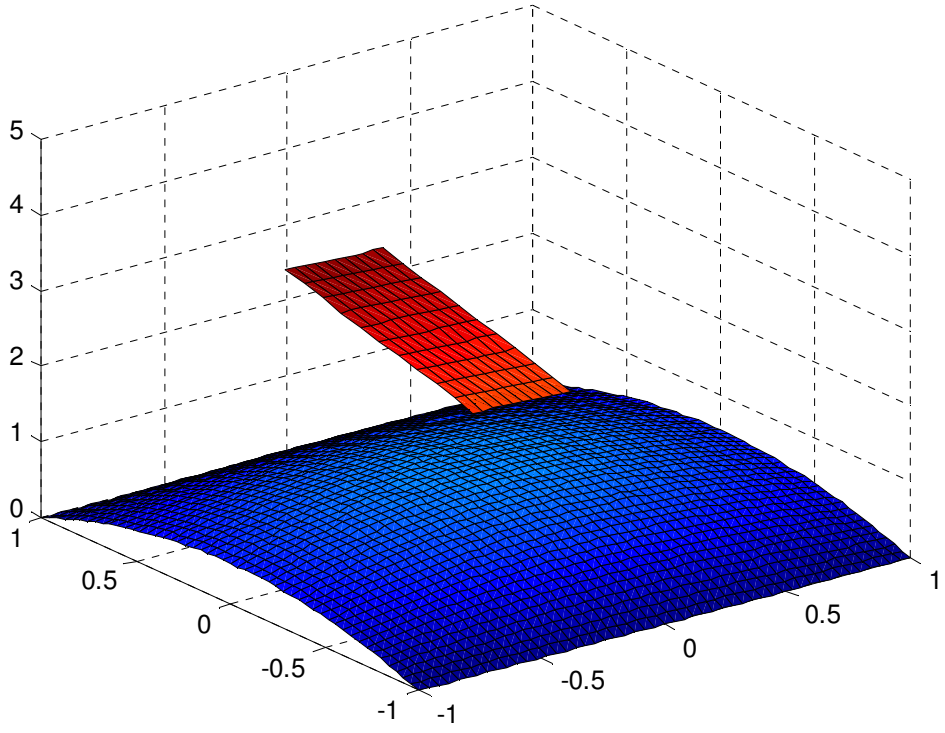


Figura 5.16

```
>> % vamos a estimar el error en la aproximación
>> x=linspace(0,.4,101);
>> y=linspace(-.6,-.2,101);
>> [X,Y]=meshgrid(x,y);
>> E=f(X,Y)-l(X,Y);
>> max(max(abs(E)))
ans =
   2.7608
```

Para h=.2 de $\{|x-.2|, |y+.4| \le .h\}$, el cuadrado centrado en (x_0, y_0)

Corte de un gráfico

Cuando se quiere ver con mayor detalle el comportamiento de una función (*2D* o *3D*), en una región más reducida. Se emplea el comando *axis*
Comando *subplot* (para distintas gráficas en distintos cuadros)
Sea la función $f(x,y) = ce^{(x-1)^2-2y^2} + (1-c)e^{-(x+1)^2-y^2}$

```
f=inline('c*exp(-(x-1).^2-2*y.^2)+(1-c)*exp(-(1+x).^2-y.^2)','x','y','c')

f =
     Inline function:
     f(x,y,c) = c*exp(-(x-1).^2-2*y.^2)+(1-c)*exp(-(1+x).^2-y.^2)

>> [X,Y]=meshgrid(-2:.2:2);
>> subplot(3,2,1)
>> surf(X,Y,f(X,Y,1))
```

```
>> title('c=1')
>> subplot(3,2,2)
>> surf(X,Y,f(X,Y,.4))
>> title('c=.4')
>> subplot(3,2,3)
>> surf(X,Y,f(X,Y,.8))
>> title('c=.8')
>> subplot(3,2,4)
>> surf(X,Y,f(X,Y,.2))
>> title('c=.2')
>> subplot(3,2,5)
>> surf(X,Y,f(X,Y,.6))
>> title('c=.6')
>> subplot(3,2,6)
>> surf(X,Y,f(X,Y,.0))
>> title('c=0')
>> colormap(cool)
```

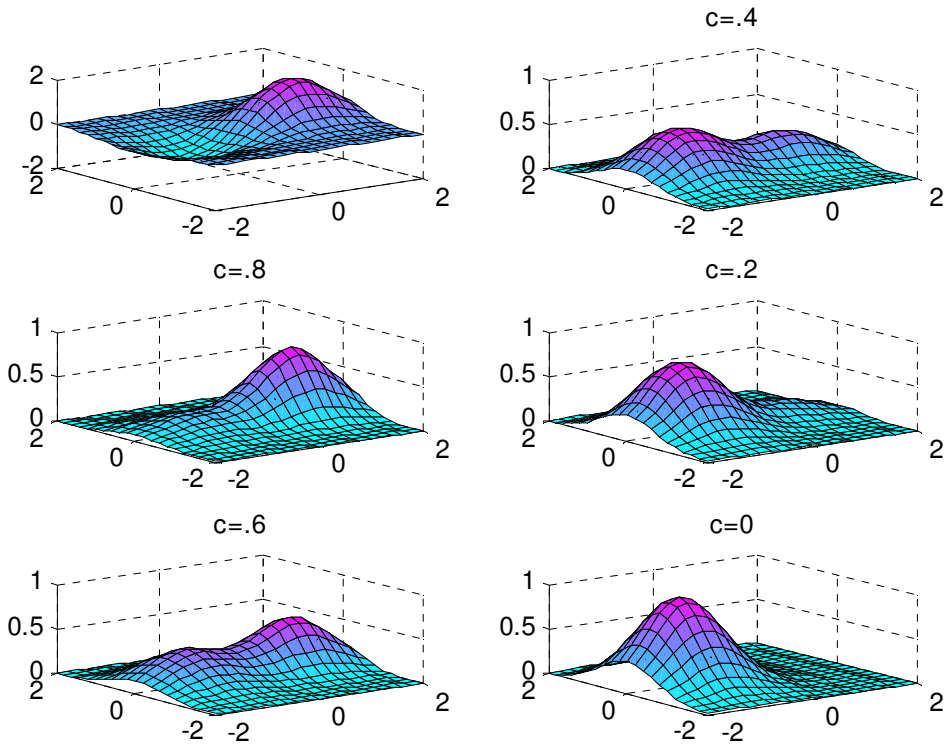


Figura 5.17

Graficación en 3-D

Superficies y conjuntos de nivel

Si f es tres variables $f(x,y,z)$, los conjuntos de nivel $S_c=\{(x,y,z)/f(x,y,z)=c)\}$ consisten de una más superficies bidimensionales llamadas superficies de nivel, entre ellas las superficies cuadráticas, como ser:

Paraboloide: $f(x,y,z)=z-x^2-y^2$, cuya $S_c=\{(x,y,z)/z-x^2-y^2=c)\}$, grafica de $z=x^2+y^2+c$

Paraboloide hiperbólico: $f(x,y,z)=z-x^{2+}y^2$, grafica de $z=x^2-y^2+c$
Hiperboloide: es una superficie de nivel de $f(x,y,z)=x^{2+}y^2-z^2$, para $c>0$ $S_c=\{f(x,y,z)=c)\}$es un hiperboloide de una hoja, $c<0$ de dos hojas
Una elipsoide es una superficie de nivel de $f(x,y,z)=(x^2/a^2)+(y^2/b^2)+(z^2/c^2)$, esfera si $a=b=c$
Ejemplo. Sea $f(x,y,z)= x^{2+}y^2-z^2$(hiperboloides), se usará el archivo *impl.m*:

```
f=inline('x.^2+y.^2-z.^2','x','y','z')

f =
    Inline function:
    f(x,y,z) = x.^2+y.^2-z.^2

>> subplot(2,2,1)
>> impl(f,esquinas,1) % c=1

ans =

The max over this domain is 8.00000

ans =

The min over this domain is -4.00000

>> subplot(2,2,2)
>> impl(f,esquinas,.1) % c=.1
ans =
The max over this domain is 8.00000
ans =
The min over this domain is -4.00000
>> subplot(2,2,3)
>>
>> impl(f,esquinas,0) % c=0
ans =
The max over this domain is 8.00000
ans =
The min over this domain is -4.00000
>> subplot(2,2,4)
>> impl(f,esquinas,-.5) % c=-0.5
ans =
The max over this domain is 8.00000
ans =
The min over this domain is -4.00000
```

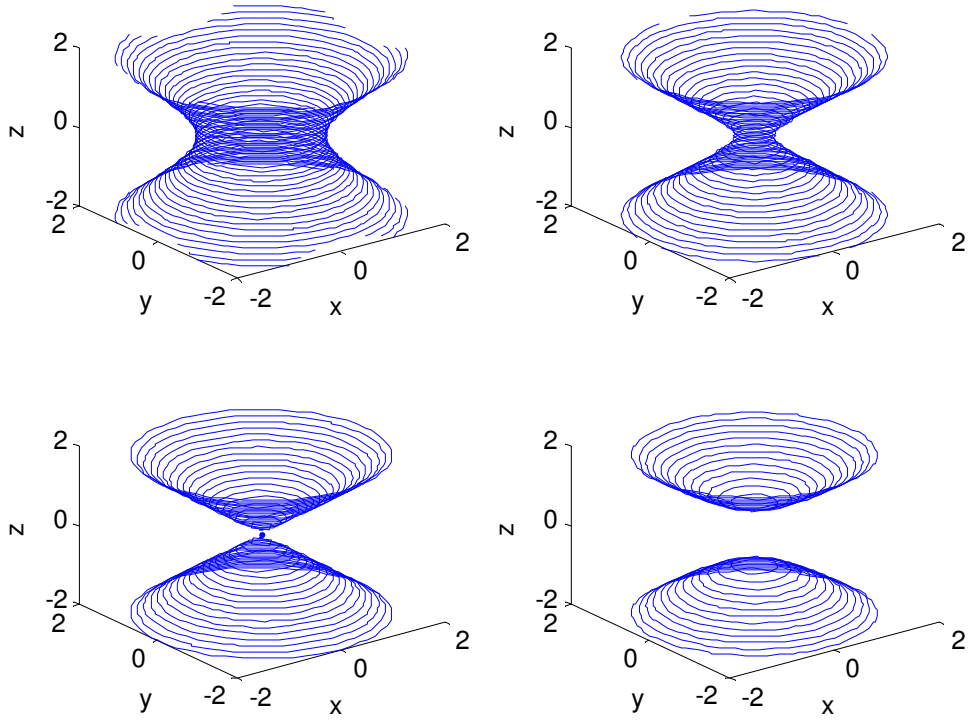


Figura 5.18

Cortes de un sólido

Sea la region 3D ,$R=\{(x,y,z):-2\le x,y\le 2, 0\le z\le 4\}$ y $f(x,y,z)= 20(z+e^{-x^2-y^2})$

Ponemos un array3D de puntos de malla de r como. Tildamos los valores de la coordenada x_i=0-2,-1.8,...1.8,2, los y_i=-2,-1.8,...,1.8,2 y los z_k=0,.2,.4...,3.8,4, así la malla contendrá puntos (x_i,y_j,z_k), los valores de la función serán $f(x_i,y_j,z_k)=w_{i,j,k}$

```
f=inline('20*(z+exp(-x.^2-y.^2))','x','y','z')

f =
    Inline function:
    f(x,y,z) = 20*(z+exp(-x.^2-y.^2))

>> x=-2:.2:2;y=x;z=0:.2:4;
>> [X,Y,Z]=meshgrid(x,y,z);
>> W=f(X,Y,Z);
>> % cortamos la región con planos paralelos a los planos coordenadas,por ej
>> %x=0, y=0, z=2
>> slice(X,Y,Z,W,0,0,2)
>> colormap(hot),colorbar
```

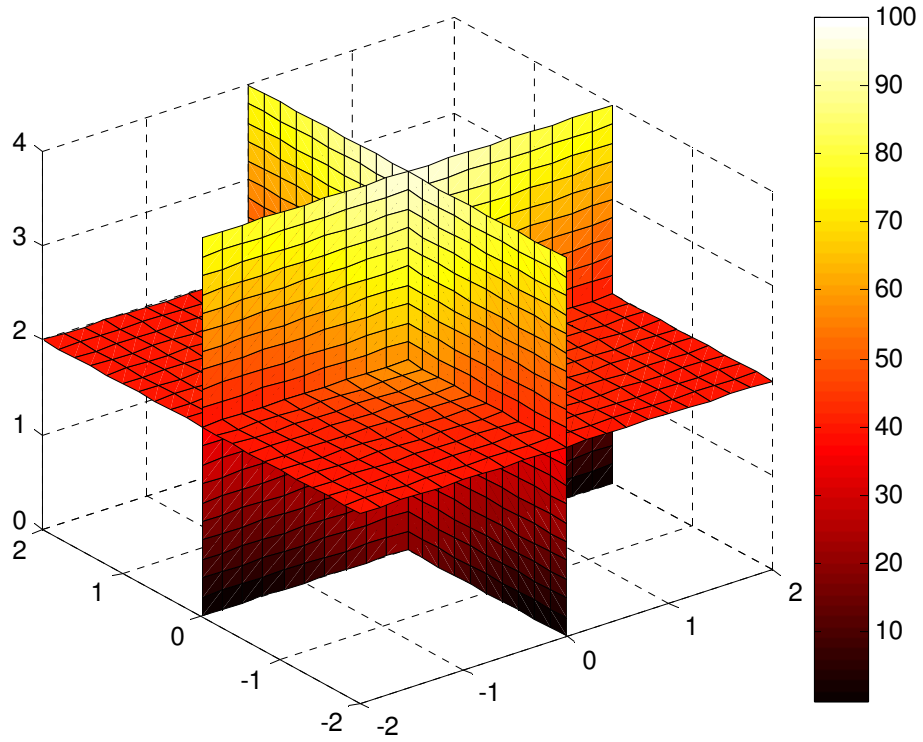


Figura 5.19

Vemos que, si los colores representan el incremento de la temperatura en la secuencia negro, marrón...blanco, crece con z, alcanzando su valor de 100 en (0,0,4)

Campo del vector gradiente

Sea $f(x,y,z)=z+(y^2-x^2)/4$, paraboloide hiperbólico, su gradiente es $[-x/2,y/2,1]$, lo hacemos en dos formas, primero adjuntamos los vectores gradientes en los punyos ena familia de planos paralelos a x,z a una altura z=-1,0,1 sobre el cuadrado $x\geq 0, y\leq 2$

```
>> [X,Y]=meshgrid(0:.4:2);
>> U=-X/2;
>> V=Y/2;
>> W=1+0*X;
>> subplot(1,2,1)
>> for z=[-1,0,1]
Z=z+0*X;
quiver3(X,Y,Z,U,V,W)
hold on
end
>> axis image
```

Se eligieron sólo tres niveles de z, por cuestión de claridad

En la segunda forma se muestra sobre la superficie de nivel f=0, a través de:

```
% graficar la superficie con malla más fina
```

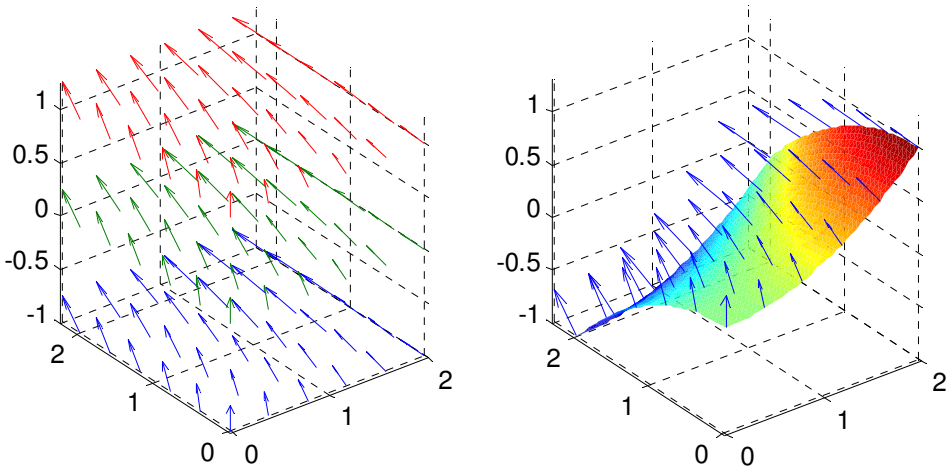


Figura 5.20

el plano tangente a la superficie S en el punto $P_0 \varepsilon S$ es la colección de todos los vectores juntos a P_0 y tangente a S_0 en P_0, o sea la reunión de todos los vectores juntos a P_0 que son ortogonales a un vector normal a S en ese punto.

Parametrización de superficies

Sea el cilindro de radio a >0, centrado sobre el eje z y se extiende de $-pi/2$ a pi/2, iniciamos ahora con un dominio rectangular: $0 \leq u \leq 2pi, -pi/2 \leq v \leq pi/2$; buscamos doblar estas partes planas para generar el cilindro, con línea de segmento paralela al eje u en círculos de radio y la línea de segmento paralela al eje v en líneas verticales sobre la superficie del cilindro. Las funciones coordenadas que lo harán:

$x(u,v)=a\cos(u)$, $y(u,v)=a\sin(u)$, $z(u,v)=v$

Si a=1

```
>> %definir coordenadas en el espacio u,v
>> u=linspace(0,2*pi,21);
>> v=linspace(-pi/2,pi/2,21);
>> [U,V]=meshgrid(u,v);
>> a=1;

>> %definir las funciones coordenadas
>> X=a*cos(U);
>> Y=a*sin(U);
>> Z=V;
>> surf(X,Y,Z)
```

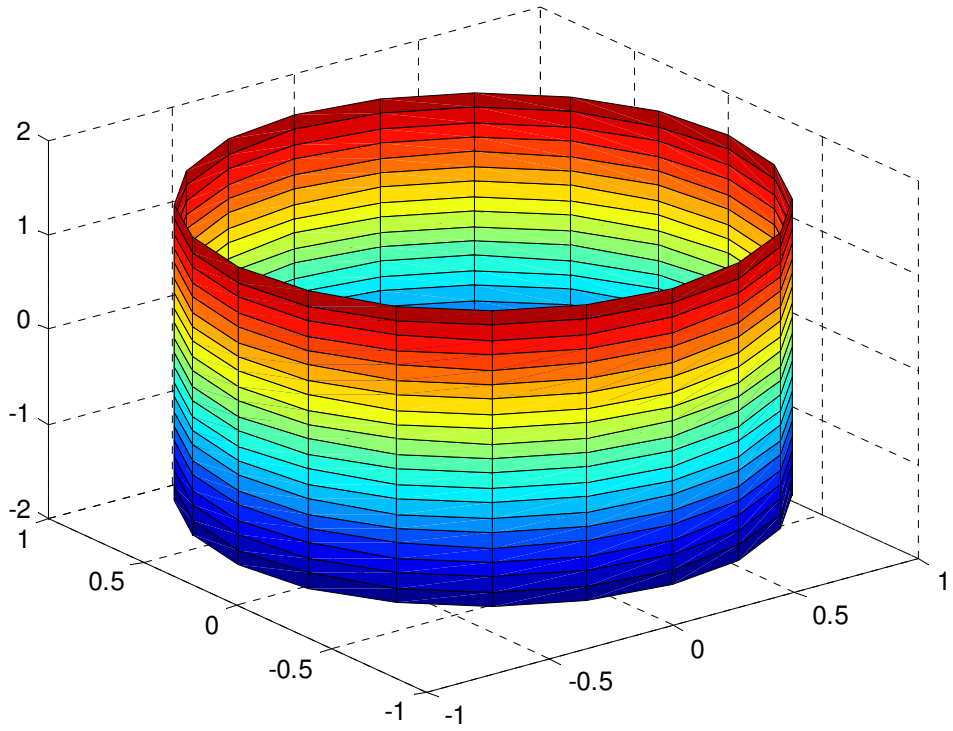


Figura 5.21

```
>> a=2;
>> u=linspace(0,2*pi,41);
>> v=linspace(-pi/2,pi/2,31);
>> [U,V]=meshgrid(u,v);
>> X=a*cos(V).*cos(U);
>> Y=a*cos(V).*sin(U);
>> Z=a*sin(V);
>> surf(X,Y,Z);colormap(gray);shading flat
```

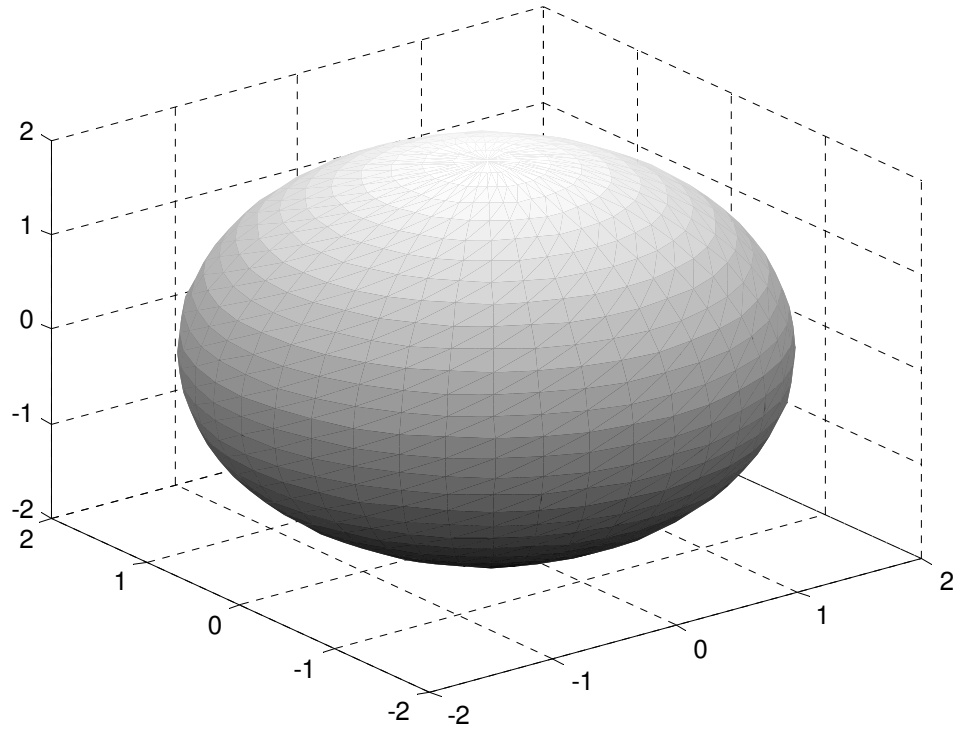


Figura 5.22

Ahora si desea una esfera unidad simplemente con:
>>*sphere*

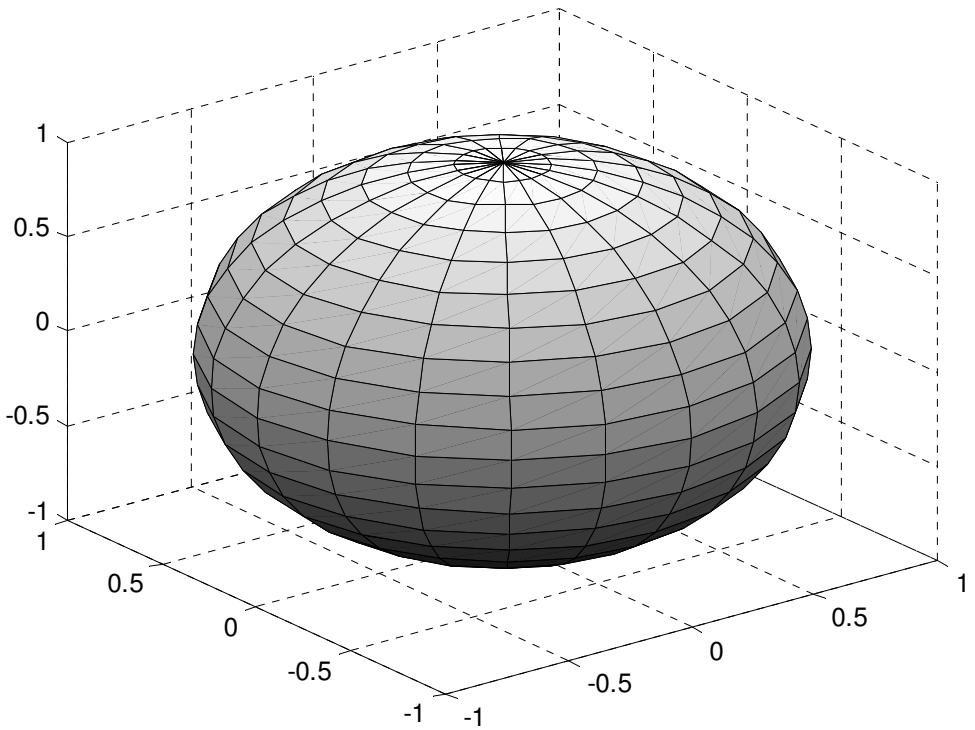


Figura 5.23

Superficies de revolución (generadas por el giro de una curva en el plano *x,z* sobre el eje *z*).

Sea un círculo de radio a en el plano *x,z* con centro en (*r*,0,0), con *r*>*a*, el toro es la superficie barrida por el círculo al girar el plano del círculo sobre el eje z

```
>> a=.5;r=2;
>> u=linspace(0,2*pi,41);v=u;
>> [U,V]=meshgrid(u,v);
>> X=cos(U).*(r+a*cos(V));%paramétricas del circulo en x,z
>> Y=sin(U).*(r+a*cos(V));
>> Z=a*sin(V);
>> surf(X,Y,Z);
```

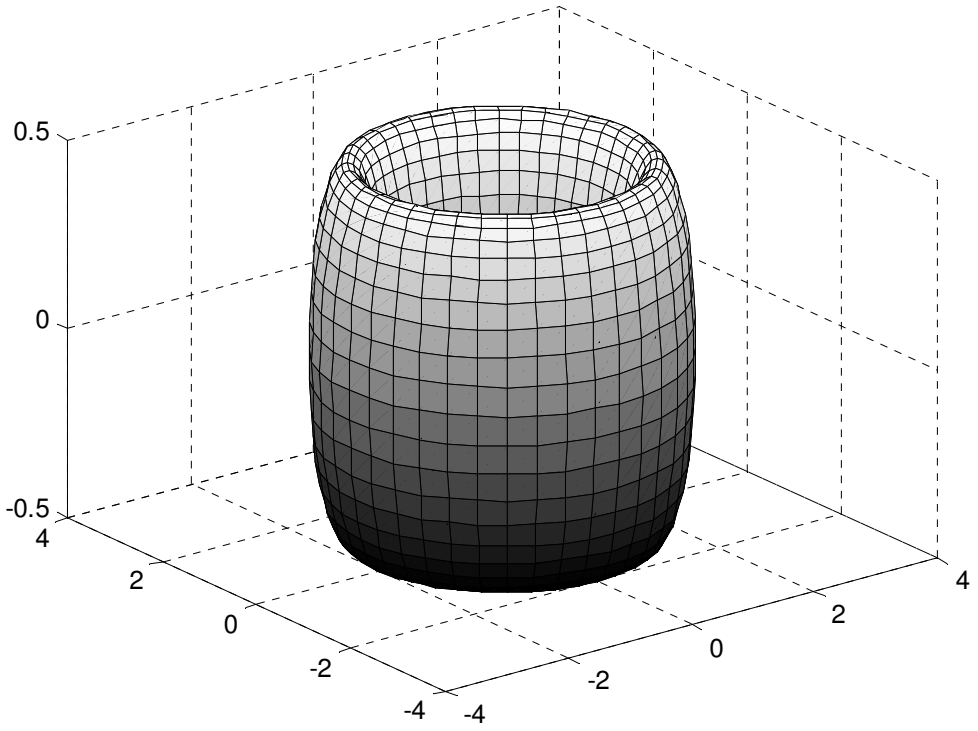


Figura 5.24

Función propia de *MATLAB* para superficie de revolución. Si *f(x,y)* está dada por *x=f(z)*, para *z en [0,1], si* $x=f(z)=3(z-1/3)^2$:

```
>> z=linspace(0,1,41);
>> cylinder(3*(z-1/3).^2)
```

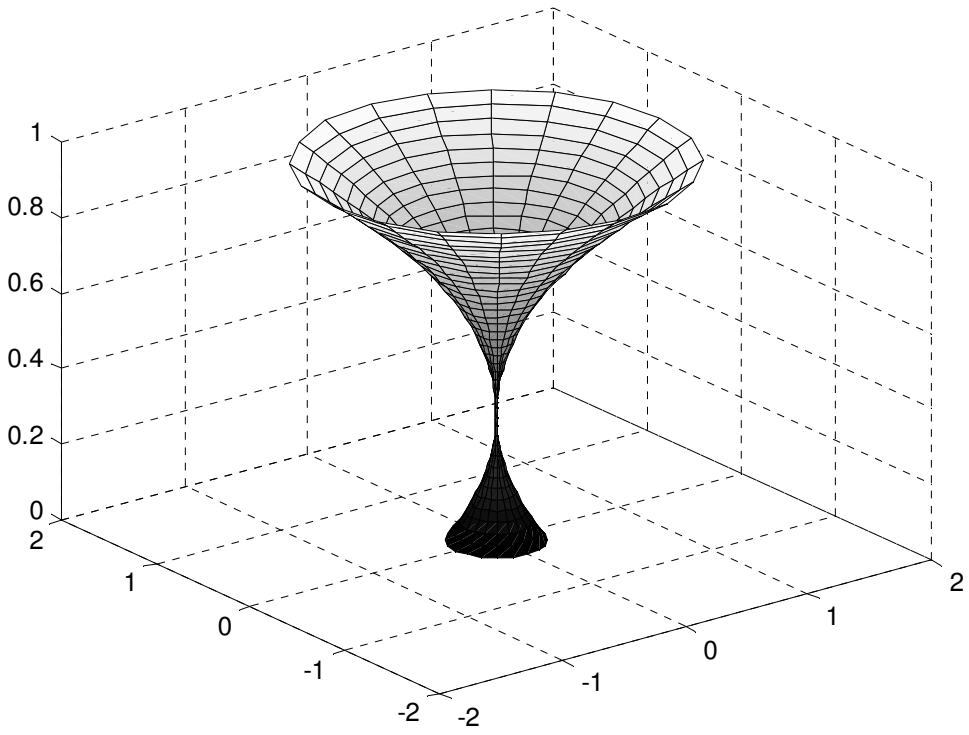


Figura 5.25

intersección de cilindros

```
> u=linspace(0,2*pi,41);v=linspace(-2,2,41);[U,V]=meshgrid(u,v);
>> %cilindro vertical de radio 1
>> surf(cos(U),sin(U),V);hold on
>> %cilindro horizontal de radio 0.5
>> surf(.5*cos(U),V,.5*sin(U));hold off
```

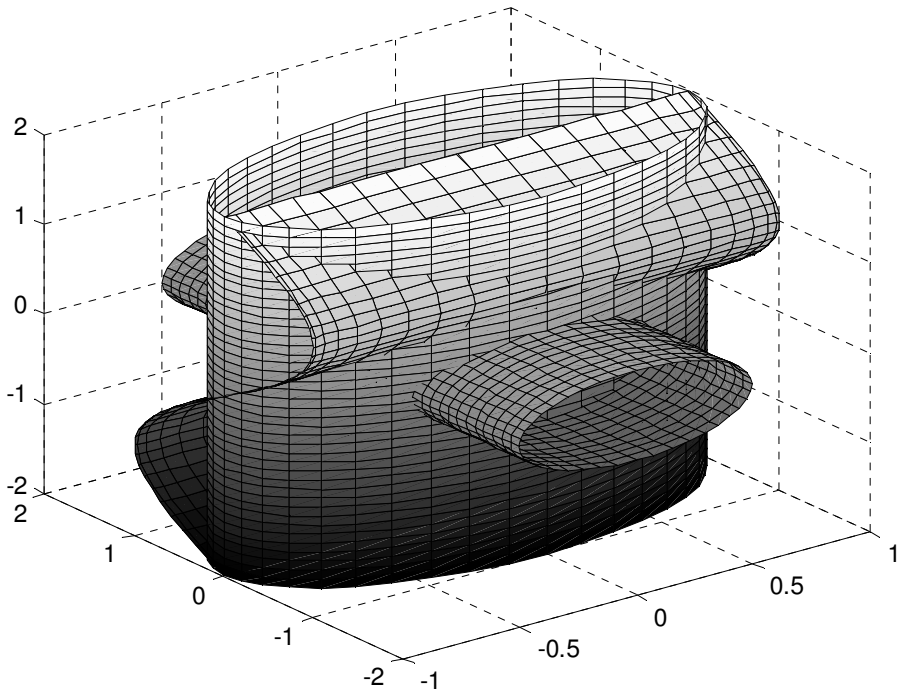


Figura 5.26

El comando ezsurf

Se puede emplear para graficar superficies dadas paramétricamente en términos de expresiones simbólicas, con la sintaxis *ezsurf(x,y,z,[a b c d])*, parámetros *s* y *t*(definen el dominio [*a b c d*], con *x,y,z* expresiones simbólicas dependientes de *s,t*.

```
>> syms s t
>> %cilindro vertical de radio 1
>> x=cos(s);
>> y=sin(s),
 >> z=t;ezsurf(x,y,z,[0 2*pi -2 2])
>> hold on
```

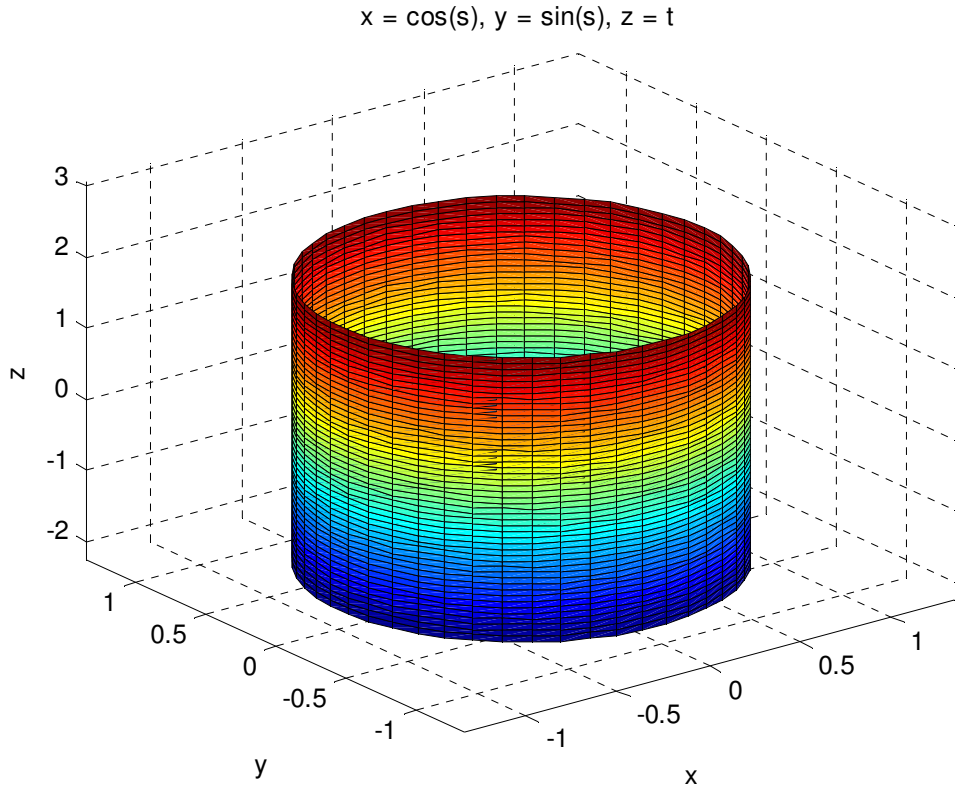


Figura 5.27

```
% función formada por la mitad de un toro de r=1, a=.25, centrada en (1,0,.5)
>> xhandle=1+cos(s)*(1+.25*cos(t));
>> yhandle=.25*sin(t);
>> zhandle=.5+sin(s)*(1+.25*cos(t));
ezsurf(xhandle,yhandle,zhandle,[-pi/2 pi/2 0 2*pi])
>> hold off
>>axis([-2 3 -2 2 -2 2])
```

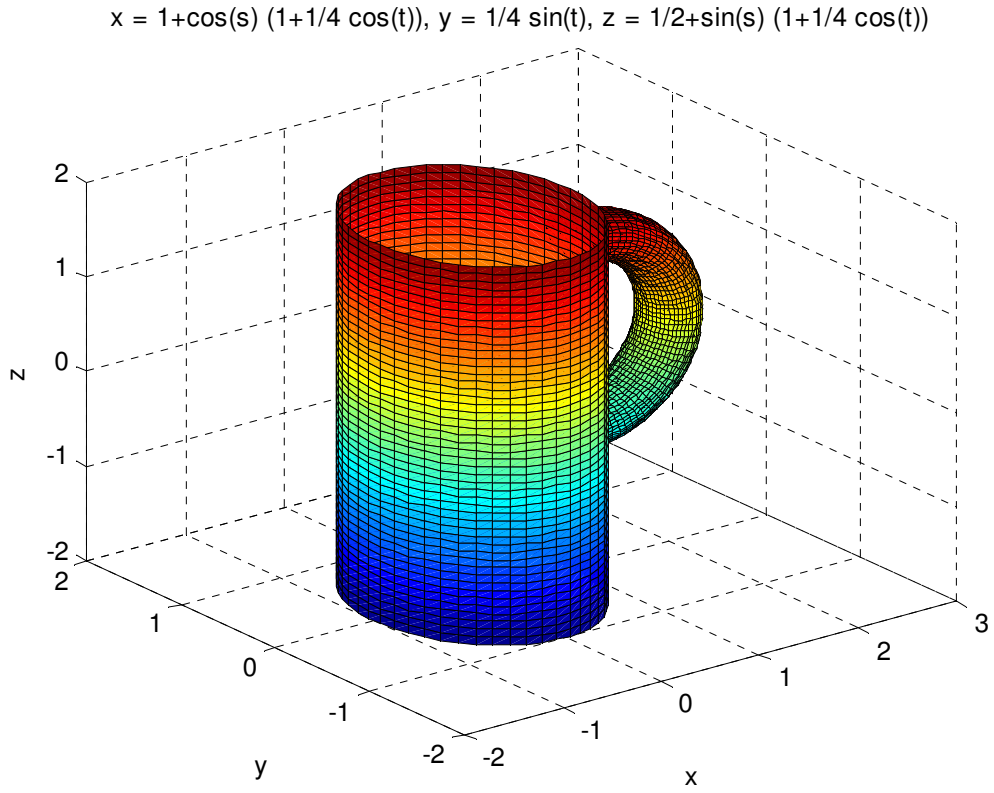


Figura 5.28

5.10 PUNTOS CRITICOS Y LA SEGUNDA DERIVADA

Si $f(x,y)$ tiene un mínimo local en el punto(x_0,y_0) la intersección del gráfico de f con cualquier plano vertical $a(x-x_0)+b(y-y_0)=0$ debe ser una curva que es cóncava hacia arriba cerca del punto referido, si (x_0,y_0) es un máximo esta curva debe ser cóncava hacia abajo cercana a (x_0,y_0). Sea el discriminante

$D(x,y)=f_{xx}(x,y)f_{yy}(x,y)-f^2_{xy}(x,y)$ determinante 2x2 de la matriz hessiana

-Si $D(x_0,y_0)>0$ y $f_{xx}(x_0,y_0)>0$, entonces (x_0,y_0) es un mínimo local

- Si $D(x_0,y_0)>0$ y $f_{xx}(x_0,y_0)<0$, entonces (x_0,y_0) es un máximo local

- Si $D(x_0,y_0)<0$, entonces (x_0,y_0) es un punto silla

- Si $D(x_0,y_0)=0$, no hay decisión

Si aproximamos una función f en un punto crítico por una función cuadrática, usando el desarrollo de Taylor en (x_0,y_0):

$f(x,y)=f(x_0,y_0)+f_x(x_0)(x_0y_0)+f_y(x_0,y_0)(y-y_0)+q(x,y)+E(1)$

Con $q(x,y)$ la función cuadrática:

$q(x,y)= 0.5[f_{xx}(x_0,y_0)\ )(x_0y_0)^2+f_{yy}(x_0,y_0)(y-y_0)^2]+2f_{xy}(x_0,y_0)\ )(x_0y_0)(y-y_0)$ (2)

Si f es de clase 3, el error $E\leq C max\ |x-x_0|^3, |y-y_0|^3$ (3)

Los primeros tres términos de (1) representan la aproximación al plano tangente a f en (x_0,y_0).En el punto crítico, $f_x(x_0,y_0)=f_y(x_0,y_0)$, reduciéndose la expansión de Taylor a

$f(x,y)=f(x_0,y_0)+q(x,y)+E(4)$

Cuando E tiende a cero, f es aproximadamente q cerca de un punto crítico, o sea que(x,y) cercano $a(x_0,y_0)$, las líneas de nivel de f deben aproximarse por las de q, ahora q en términos de una cuadrática sería:

$2a=f_{xx}(x_0,y_0), 2b=f_{xy}(x_0,y_0); 2c=f_{yy}(x_0,y_0)$ con x reemplazada por $x-x_0$,y por $y-y_0$

En los dos primeros casos de la prueba d la segunda derivada, cuando $D(x_0,y_0)>0$, las curvas de nivel de q son elipses, centradas en (x_0,y_0), en el tercer caso$(D<0)$ puede ocurrir cuando f_{xx} y f_{yy} son diferentes de cero y signo diferentes o si $f_{xx} = f_{yy} =0$ en (x_0,y_0) y $f_{xy} \neq 0$ en (x_0,y_0).El gráfico de f cerca del punto crítico es una superficie silla

Ejemplo. Sea $f(x,y)=x^3-3xy^2$, el origen es punto crítico para f y $D(0,0)=0$, dibujamos el gráfico y las curvas de nivel cerca de (0,0)

```
>> f=inline('x.^3-3*x.*y.^2','x','y')%se dibuja el gráfico de f del lado izquierdo

f =
     inline function:
     f(x,y) = x.^3-3*x.*y.^2

>> [X,Y]=meshgrid(-1:.1:1);
>> subplot(1,2,1)
>> surf(X,Y,f(X,Y))%se dibujan las curvas de nivel a la derecha
>> [XX,YY]=meshgrid(-1:.025:1);
>> subplot(1,2,2)
>> contour(XX,YY,f(XX,YY),'k')
```

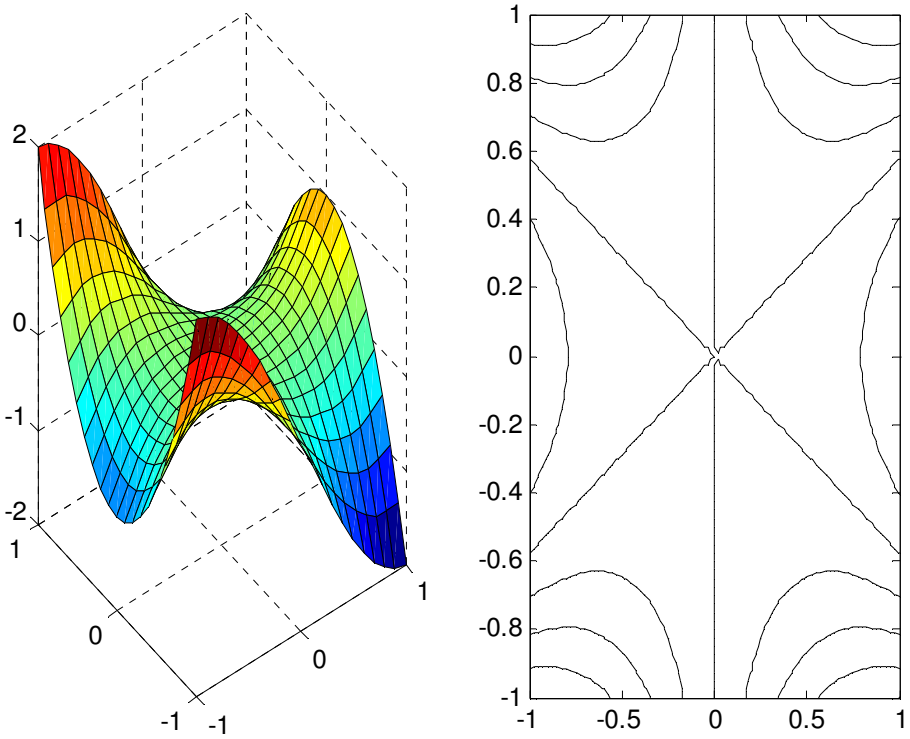


Figura 5.29

5.11 BUSCANDO MÁXIMO Y MÍNIMO

Si $v=[v_1,...,v_n]$, columna o fila, *max(v)* devuelve $\max v_i$, igual *min(v)*; con el agregado *[v,i]* da el el primer indice *i* donde se alcanza el máximo. empleando para $f=x^2(1-x^2)$ en [-1,1]:

```
>> f=inline('x.^2.*(1-x.^2)');
>> x=-1:0.2:1;
>> [M,i]=max(f(x))

M =
    0.2304
i =
  3
>> x0=x(i)
x0 =
   -0.6000
```

Se halló sobre una malla, no sobre el *intervalo*[-1,1], ya que el máximo verdadero de la función es 0.25, en $x=\pm 1/x^{0.5}$

Para funciones de dos variables, debemos usar matrices, *max(A)* halla el máximo en cada columna, grabando los resultados como vector fila, entonces con *max(max(A))* se halla el máximo del vector fila *max(A)*, o sea sobre los elementos $a_{i,j}$, igual para mínimo.

Entonces, para $f=x/2+exp(-x^2-y^2)$ en $G=\{0\le x\le 1\}X\{-1/2\le y\le 1/2\}$:

```
>> f=inline('.5*x+exp(-x.^2-y.^2)','x','y');
>> x=0:.02:1;
>> y=-.5:.02:.5;
>> [X,Y]=meshgrid(x,y);
>> [row,I]=max(f(X,Y));
>> [maxf,j]=max(row)

maxf =
    1.0646

j =
 14
>> x0=x(j)
x0 =
    0.2600
>> y0=y(I(j))
y0 =
    0
```

Así $f(x_j,y_i)\le f(.26,0)=1.0646$ para todos los puntos de la grilla

Con el archivo *findcrit* se emplean los mapas de líneas que sumado a max y min, sirven para estimar máximo-mínimo sobre un rectángulo de *esquinas(corners)*

(a,c),(b,c),(b,d) y *(a,d),* permitiendo moverse sobre las esquinas superiores e inferiores donde se piensa que cae el punto crítico, rellenándose el rectángulo azul, conteniendo un mapa de líneas. El proceso puede repetirse cuatro veces, la función en archivo m.

Sea $f(x,y)=sen(x+y)+1-x^2/40+exp(-y^2)$ en el rectángulo [-6,9]x[-4,4].Primero se grafica la función:

```
>> f=inline('sin(x+y)+1-(1/40)*x.^2+exp(-y.^2)','x','y');
>> [X,Y]=meshgrid(-6:.2:9,-4:.2:4);
>> [X,Y]=meshgrid(-6:.2:9,-4:.2:4);
>> surf(X,Y,f(X,Y))
```

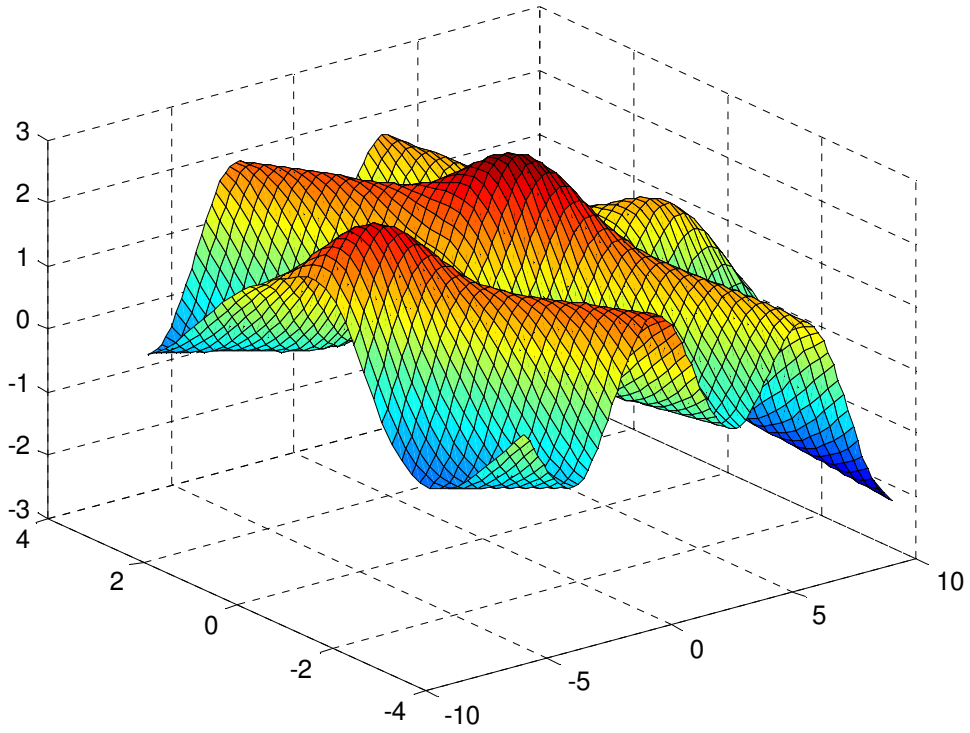


Figura 5.30

Usamos un mapa de líneas en el rectángulo para una idea gruesa de los puntos críticos:

```
>>contour(X,Y,f(X,Y),20)
```

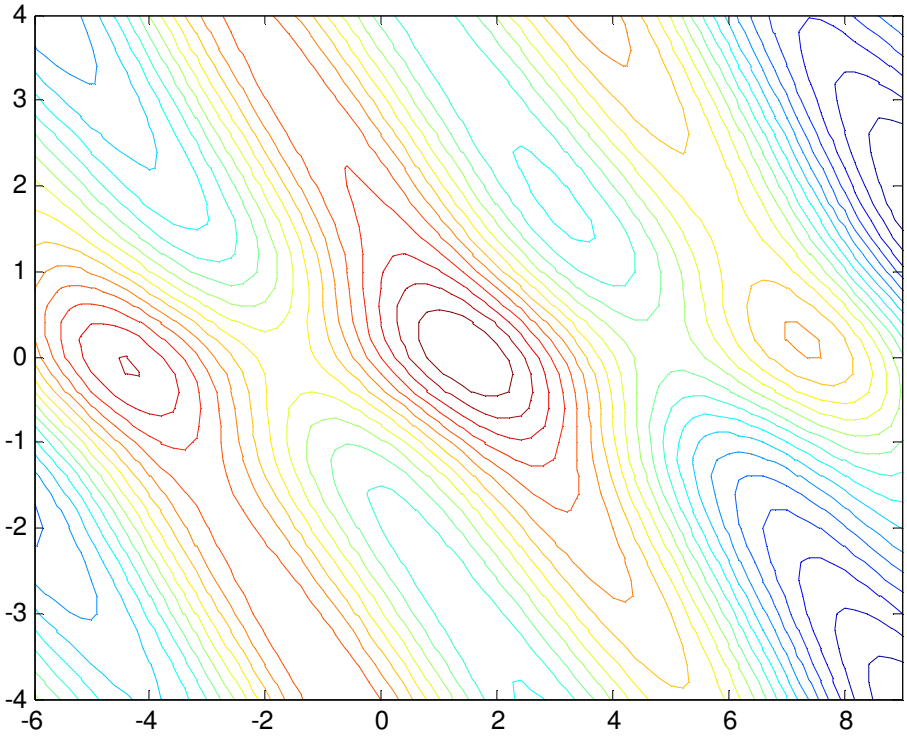


Figura 5.31

De esta gráfica se nota alternancia de máximos con puntos silla; para hallar las coordenadas de estos puntos críticos, se deben resolver las primeras derivadas parciales o emplear técnicas numéricas, intentamos la resolución simbólica

```
>> syms x y
>> fx=cos(x+y)-x/20;
>> fy=cos(x+y)-2*y*exp(-y^2);
>> [a,b]=solve(fx,fy),double([a,b]);

a =
   1.461099801908717663838449179406 9
b =
   .365763954281042082989609968871 46e-1
```

Se encontró sólo un punto crítico y en forma numérica. Aplicamos ahora la prueba de la segunda derivada, hallando el discriminante y evaluarlo en el punto crítico

```
> fxx=diff(fx,x);fyy=diff(fy,y);fxy=diff(fx,y);
>> D=fxx*fyy-fxy^2;
>> subs(D,[x y],[a b])
ans =
  2.136124693555305084605426465019 7

>> subs(fxx,[x y],[a b])
ans =
   -1.047327914189789317864424017506 4
```

Como $D>0$ y $f_{xx}<0$ en el punto crítico, se concluye que es un máximo local, de acuerdo con las gráficas.
Para hallar los otros puntos críticos, usaremos el findcrit, de la gráfica de líneas parece haber un punto silla cerca de (-1,0), lo encerramos con esquinas [-2,0,-1,1], clickeando varias veces se obtiene la figura, apareciendo localizarse en (-1,61,-.040), que se confirma analíticamente

```
>> a=-1.61,b=-0.04;

a =
  -1.6100

>> double(subs(fx,[x y],[a b]))

ans =
   0.0014

>> double(subs(fy,[x y],[a b]))

ans =
   7.5121e-004

>> double(subs(D,[x y],[a b]))

ans =
   -1.9345
```

Siendo resultados aproximados, con el *findcrit* se puede trabajar sobre rectángulos más chicos.
También se puede usar el método de Newton para resolver el sistema de las condiciones suficientes y necesarias, necesitando archivos m para f_x,f_y, la matriz jacobiana, para un punto de salida, se podría la ubicación aproximada de los puntos críticos hallados con *findcrit*.

5.11.1 Problemas de máximos y mínimos restringidos

El problema sería hallar max-mín de f sujeto a la restricción $g(x,y)\leq 0$
El máx o mín puede darse en el interior de K o en la frontera de K, con K descripto por:
$K=\{(x,y):g(x,y) \leq 0\}$ y la frontera del conjunto K es la curva de nivel $(x,y):g(x,y) =0$.
Los multiplicadores de Lagrange se emplean para hallar los extremos de funciones sobre la curva frontera de un conjunto; buscando los puntos sobre esta frontera donde la curva de nivel de f es tangente a la frontera de K, cómo el vector gradiente es ortogonal a las curvas de nivel, en cada punto f y g tendrán igual dirección y sentido opuesto, lo que se establece como: $\nabla f(x_0,y_0)=\lambda \nabla g(x_0,y_0)$, para algún escalar λ.
Se conformará un sistema de tres ecuaciones en tres variables:

$f_x(x,y)=\lambda g_x(x,y)$, (1)

$f_y(x,y)=\lambda g_y(x,y)$ (2)

$g(x,y)=0$ (3)

El archivo *lagrange.m* llama a *lagrange(f,g,esquinas)*, *esquinas* es *[a b c d]*, con *(a,c),(b,c),(b,d)* y *(a,d)* son las esquinas del rectángulo, pudiendo clickear en cualquier punto del rectángulo, graficándose la curva de *f* a través ese punto, se puede efectuar hasta cinco veces.

Sea $f(x,y)=x^3+x^2+y^2/3$, con *K* el disco de radio 6 y el centro en el origen:$x^2+y^2\leq 36$, es decir que $g(x,y)= x^2+y^2 =36$, resultando el sistema

$3x^2+2x=2\lambda x$

$2y/3=2\lambda y$

$x^2+y^2 =36$

```
>> f=inline('x.^3+x.^2+(y.^2)/3','x','y');
>> g=inline('x.^2+y.^2-36','x','y');
>> esquinas=[-8,9.5,-7,7];
>> lagrange(f,g,esquinas)
Varios clicks % se muestran algunos clicks
x          y          f(x,y)

ans =
   -0.0339    6.0049   12.0207
ans =
    4.0838   -0.9951   85.1133
ans =
    5.9367   -0.2402  244.5011
ans =
   -3.5339   -6.2794  -18.4999
ans =
   -5.7986   -6.0049 -149.3253
```

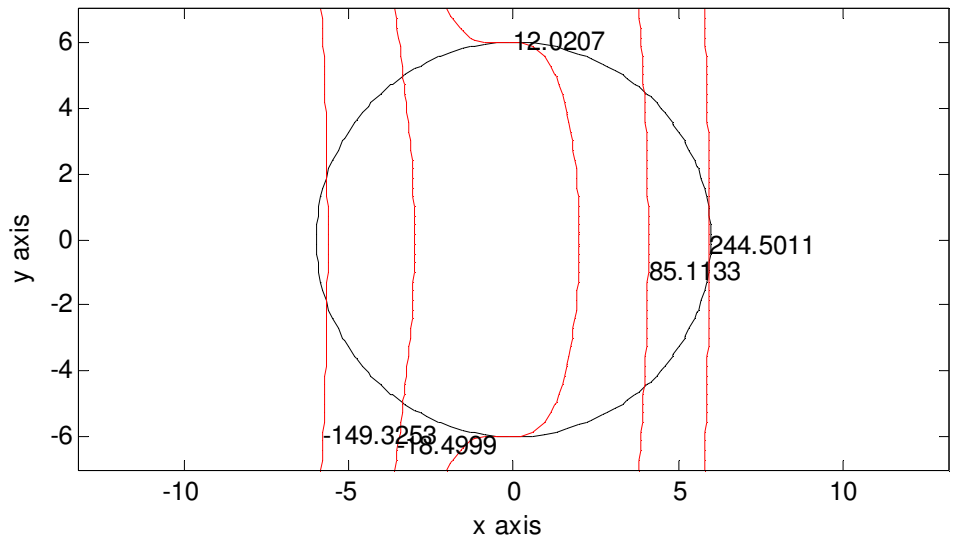


Figura 5.32

```
>> % se empieza los cálculos simbólicos de las raíces
>> syms x y lambda
>> ec1=3*x^2+2*x-2*lambda*x;
>> ec2=(2/3)*y-2*lambda*x;
>> g=x^2+y^2-36;
>> [lambda x y]=solve(ec1,ec2,g);
>> [lambda x y]
[      10,      6,              0]
[      -8,     -6,              0]
[     1/3,      0,              6]
[     1/3,      0,             -6]
[     1/3,    4/9,  0/9*29^(1/2)]
[     1/3,    4/9,  0/9*29^(1/2)]
```

Coincidiendo con el gráfico de las curvas de nivel, el máximo se alcanza en $(x,y)=(6,0)$ y el mínimo en $(x,y)=(-6,0)$, las cuatro puntos restantes son extremos locales de f sobre la curva $g=0$, se ve que las curvas de nivel de f son tangentes a la curva $g=0$ en esos puntos.

5.12 FUNCIONES DE TRES VARIABLES

Ahora se necesitan el cumplimiento de

$f_x(x,y,z)=0$ *(1)*

$f_y(x,y,z)=0$

$fz(x,y,z)=0$

resuelta simbólicamente, aunque con un punto de arranque es común el uso de técnicas numéricas, con los puntos críticos localizados su graficación no es sencilla, debe usarse toda la información disponible

Ejemplo. Sea $f(x,y,z)=1.5e^{-(x-1)^2-y^2-z^2}+e^{-(x+1)^2-z^2}$, se esperaría dos máximos locales, uno cerca de $(x,y,z)=(1,0,0)$ y el otro cerca de $(-1,1,0)$; escribiendo un m para f:

```
function w=f(x,y,z)
w1=1.5*exp(-(x-1).^2-y.^2-z.^2);
w2=exp(-(x+1).^2-(y-1).^2-z.^2);
w=w1+w2;
```

se genera el arreglo en 3D, empleando slice, en este caso ingresamos la función inline

```
>>f=inline('1.5*exp(-(x-1).^2-y.^2-z.^2)+exp(-(x+1).^2-(y-1).^2-z.^2)','x','z','y');
>> x=linspace(-1.5,1.5,41);
>> y=linspace(-.5,1.5,41);
>> z=linspace(-1,1,41);
>> [X,Y,Z]=meshgrid(x,y,z);W=f(X,Y,Z);
>> slice(X,Y,Z,W,[-1,0,1],0,0);colorbar
>> x=linspace(-1.5,1.5,41);
>> y=linspace(-.5,1.5,41);
```

```
>> z=linspace(-1,1,41);
>> [X,Y,Z]=meshgrid(x,y,z);W=f(X,Y,Z);
>> slice(X,Y,Z,W,[-1,0,1],0,0];colorbar
```

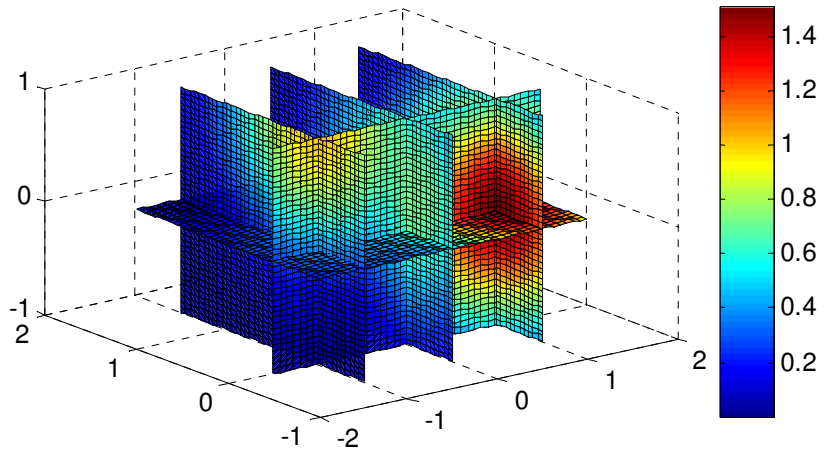


Figura 5.33

Se ha sustituído un *vector*[-1,0,1] para las coordenadas de los planos paralelos al eje *x*, dándonos cortes en los tres planos paralelos *x*=-1,0 y 1, viéndose en la figura.
La barra de colores muestra que los valores más grandes de f corresponden al tono gris más ligero, aparece que hay máximos locales cerca de (1,0,0) y (-1,1,0),
Hallado el punto crítico, se pasa al estudio de la segunda derivada, siendo CS para mínimo local en (x_0,y_0,z_0).

$f_{xx}>0$

$f_{xx}f_{yy}-f^2_{xy}>0$

$det\boldsymbol{H}>0$ evaluados en (x_0,y_0,z_0)

Siguiendo con el estudio de la función, de la forma de la función se espera que máximo local cercano a (1,0,0) sea 1.5, el cercano a (-1,1,0) sea 1, entonces las superficies de nivel entre $1<c<1.5$ consistirían de una superficie cerrada simple pareciendo una esfera que contiene al punto (1,0,0), para $c<1$, la superficie de nivel de dos superficies cerradas, una contiene a (1,0,0) y la otra a (-1,0,0); para $c>=$ suficientemente pequeño, la superficie de nivel engloban ambos puntos críticos

```
>> f=inline('1.5*exp(-(x-1).^2-y.^2-z.^2)+exp(-(x+1).^2-(y-1).^2-z.^2)','x','z','y');
>> esquinas=[-2,2,-2,2,-2,2];subplot(2,2,1)
>> impl(f,esquinas,.5)
ans =
The max over this domain is 1.50674
ans =
The min over this domain is 0.00000
>> title('c=.5')
>> subplot(2,2,2)
>> impl(f,esquinas,.9)
ans =
The max over this domain is 1.50674
```

```
ans =
The min over this domain is 0.00000
>> title('c=.9')
>> subplot(2,2,3)
>> impl(f,esquinas,.7)
ans =
The max over this domain is 1.50674
ans =
The min over this domain is 0.00000
>> title('c=.7')
>> subplot(2,2,4)
>> impl(f,esquinas,1.2)
ans =
The max over this domain is 1.50674
ans =
The min over this domain is 0.00000
>> title('c=1.2')
```

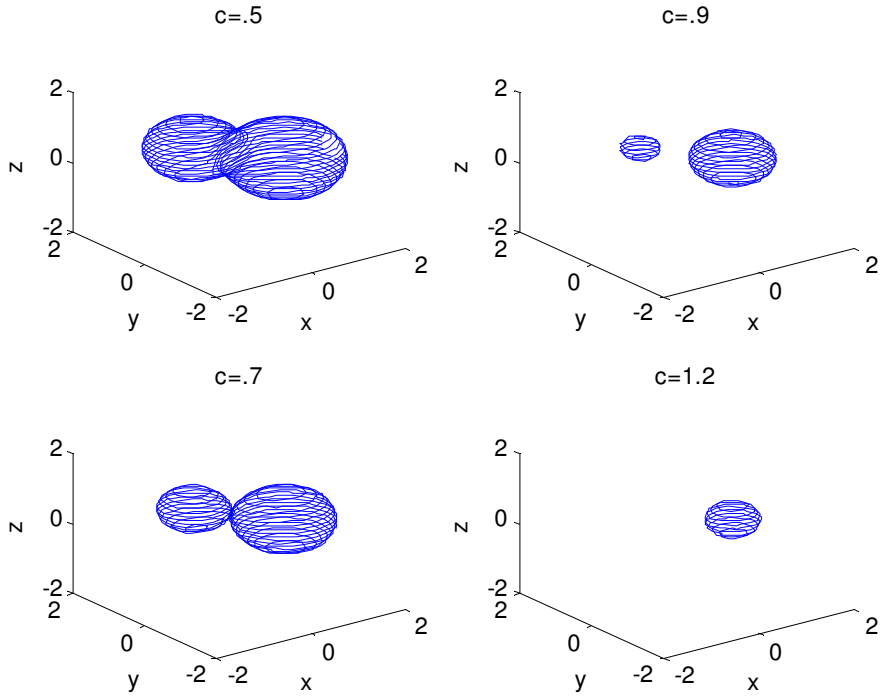


Figura 5.34

5.13 INTEGRACION

Integrales dobles sobre rectángulos

El archivo *riemann* calcula las sumas de Riemann y da visualización gráfica de la función de aproximación constante a trozos, se llama *riemann*(*f*,esquinas), con *f* en dos variables como *inline*, si se da en archivo *m* se llama riemann('*f*',esquinas), eligiéndose la cantidad *n* y *m* de subintervalos de la región rectangular, empleándose una elección para $p_{i,j}$ el centro de cada subrectángulo.

Sea $f(x,y)=x^3+x+y$ *en* $R=\{1\leq x\leq 2, 0\leq y\leq 3\}$

```
>> f=inline('x.^3+x+y','x','y');
>> esquinas=[1 2 0 3];
>> riemann(f,esquinas,1)
enter the number of subdivisions in x and y  direction  as [n m] [10 20]

subdiv =

   10    20

 Approximate value of the integral

ans =
   20.2387
```

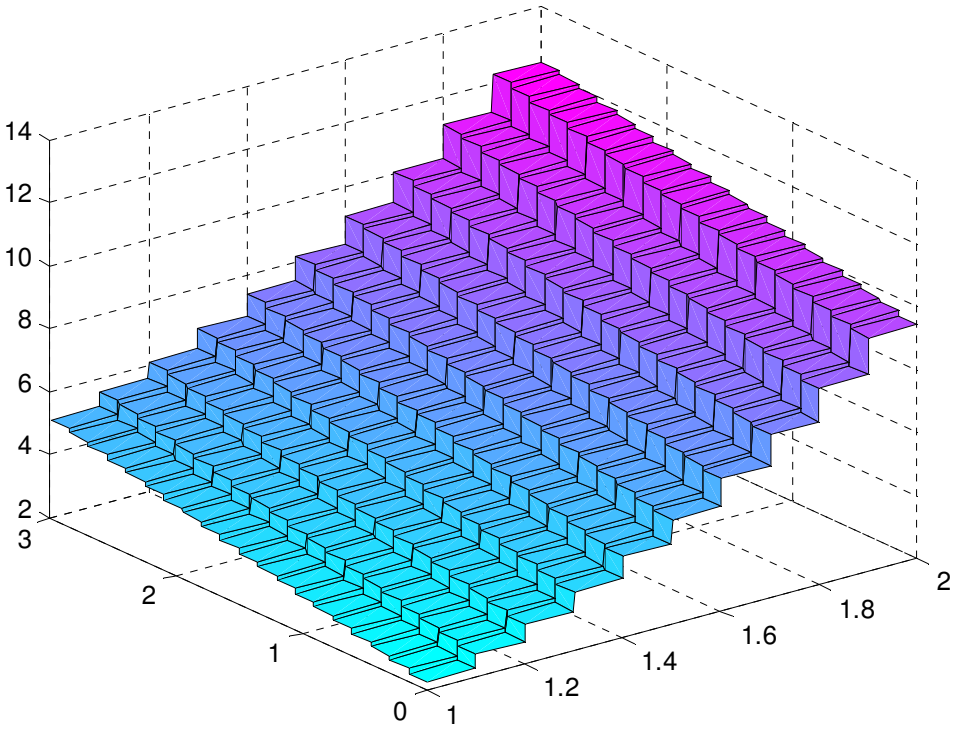


Figura 5.35

5.13.1 Integración simbólica

Recurriendo a la integración iterada, sería:

```
>> syms x y
>> f=x.^3+x+y;
>> int(int(f,y,0,3),1,2)
ans =
   81/4
```

Se pueden trabajar los límites como parámetros:

```
>> syms x y a b c d;
```

```
>> int(int(f,y,c,d),a,b)

ans =
   1/4*(d-c)*(b^4-a^4)+1/2*(d-c)*(b^2-a^2)+1/2*d^2*(b-a)-1/2*c^2*(b-a)
```

5.13.2 Alternativa numérica

Se pueden utilizar los métodos de simpson, trapeciales, adaptativos, etc

5.14 INTEGRADORES DE MATLAB

Las rutinas propias en una dimensión son *quad* y *quads(emplean la cuadratura adaptativa)*, que refinan la malla cuando la función cambia más rápido, llamando con *(f,a,b)* como *inline* o desde archivo *m(con')*, para integración doble *dblquad(f,a,b,c,d)* con primera integración en *x*, si se desea primero en el sentido de *y*, *dblquad(g,c,d,a,b).*

5.14.1 Regiones no rectangulares de integración

Sea $G=\{(x,y): x(3-x)\leq y\leq \sin(x), 0\leq x\leq 2.4\}$, *con* $f(x,y)=xy$, *se computa* $\int\int_G f(x,y)dA$

```
>> syms x y;
>> f=x*y;
>> F=int(f,y,x*(x-3),sin(x))

F =
  1/2*x*(sin(x)^2-x^2*(x-3)^2)
>>int(F,0,2.4)

ans =
   -74286/15625-3/5*sin(12/5)*cos(12/5)+1/8*sin(12/5)^2
>>double(ans)

ans =
   -4.3984
```

Los métodos numéricos, muy útiles en muchos casos, requieren regiones rectangulares que deben generarse empleando cambio de variables.

5.14.2 Cambio de variables en integrales dobles

Transformaciones afin del plano

Las transformaciones que permiten mapear las coordenadas de un plano en otras coordenadas del plano, de la forma:

$x=f(u,v)=Au+Bv+w_1$

$y=g(u,v)=Cu+Dv+w_2$ (1) o más compacta $(x,y)=T(u,v)$ representa una transformación afin; o sea se ve que $T(0,0)=w=(w_1,w_2)$, si $(w_1,w_2)=0$, t toma (0,0) en (0,0) es particularmente una transformación lineal, el caso más sencillo es que un paralelogramo lleva a un paralelogramo por una transformación afín.

Transformaciones generales del plano

Tomando el rectángulo $R=\{a\le x\le b, c\le y\le d\}$, la transformación $T(u,v)=(f(u,v),g(u,v))$ mapeará R en un conjunto G con fronteras posiblemente curvilíneas, por ejemplo si $T(u,v)=(u^2-v^2,2uv)$.

Se puede aproximar el conjunto imagen G por un paralelogramo P que es imagen de R bajo una transformación afín $\breve{T}$, obtenido las aproximaciones del plano tangente para f y g en un punto $(u_0,v_0)\varepsilon R$:

$$f(u,v)\tilde{} f(u_0,v_0)+f_u(u_0,v_0)(u-u_0)+f_v(u_0,v_0)(u-u_0) \quad (2)$$

$$g(u,v)\tilde{} g(u_0,v_0)+g_u(u_0,v_0)(u-u_0)+g_v(u_0,v_0)(u-u_0) \quad (3)$$

empleando el jacobiano para f y g:

$T(u,v)=)=(f(u,v),g(u,v)))\tilde{} T(u_0,v_0)+J(u_0,v_0)[u-u_0;v-v_0]$, con la matriz

$J(u_0,v_0)[u-u_0;v-v_0]=[\ f_u(u_0,v_0)(u-u_0)+f_v(u_0,v_0)(u-u_0);\ g_u(u_0,v_0)(u-u_0)+g_v(u_0,v_0)(u-u_0)]$

La aproximación de la transformación afín, con $A= f_u(u_0,v_0), B= f_v(u_0,v_0), C= g_u(u_0,v_0)$ y $D= g_v(u_0,v_0)$, será:

$$\breve{T}(u,v)= T(u_0,v_0)+J(u_0,v_0)[u-u_0;v-v_0], \quad (4)$$

el punto (u_0,v_0) puede ser cualquiera de R, se elige el punto medio$(u_0,v_{0)}=((a+b)/2,(c+d/2))$, el factor de multiplicación del área de la aproximación afin es

$$\left| AD-BC \right| = \left| detJ((a+b)/2,(c+d/2)) \right|$$

El area del paralelogramo P, imagen de R bajo $\breve{T}$ será:

$$\text{area}(P)= \left| J((a+b)/2,(c+d/2)) \right| (b-a)(d-c)$$

Estas aproximaciones se pueden efectuar para cada elemento i,j de R, a medida que crece este número, la unión de los paralelogramos $P_{i,j}$ da una mejor aproximación del

conjunto imagen G, como su suma el área de G, así:

$\iint_G dA(x,y) = A(G) \approx \sum_{i,j} A(P_{i,j}) = \sum_{i,j} J(u_i,v_j) A(R_{i,j})$ con (u_i,v_j) puntos medios de $R_{i,j}$

En el paso al límite, con los diámetros de $R_{i,j}$ tendiendo a cero, se obtiene:

$$Area(G)= \iint_G dA(x,y) = \iint_R \left|J(u,v)\right| dA(u,v) \quad (5)$$

El archivo trf muestra el proceso de esta aproximación, transformando un rectángulo en un conjunto curvilíneo G, haciendo una aproximación del área de G por la suma de los paralelogramos de aproximación

Las funciones $f(u,v)$ y $g(u,v)$ desde archivos m o *inline*, el rectángulo por esquinas, se lo llama *trf(f,g,esquinas)* o *trf('f','g',esquinas)*, sea *inline* o m respectivamente; se debe ingresar el número de subdivisiones en cada dirección, el rectángulo R aparece a la izquierda y G a la derecha, en pantalla la suma de las áreas de los rectángulos

Ejemplo: sea $T(u,v)=(u^2-v^2,2uv)$ y $R=[1,2]x[0,1]$

```
>> f=inline('u.^2-v.^2','u','v');
```

```
>> g=inline('2*u*v','u','v');
>> esquinas=[1 2 0 1];
>> trf(f,g,esquinas)
enter the number of subdivisions [n m]  [6 6]

subdiv =
     6     6
```

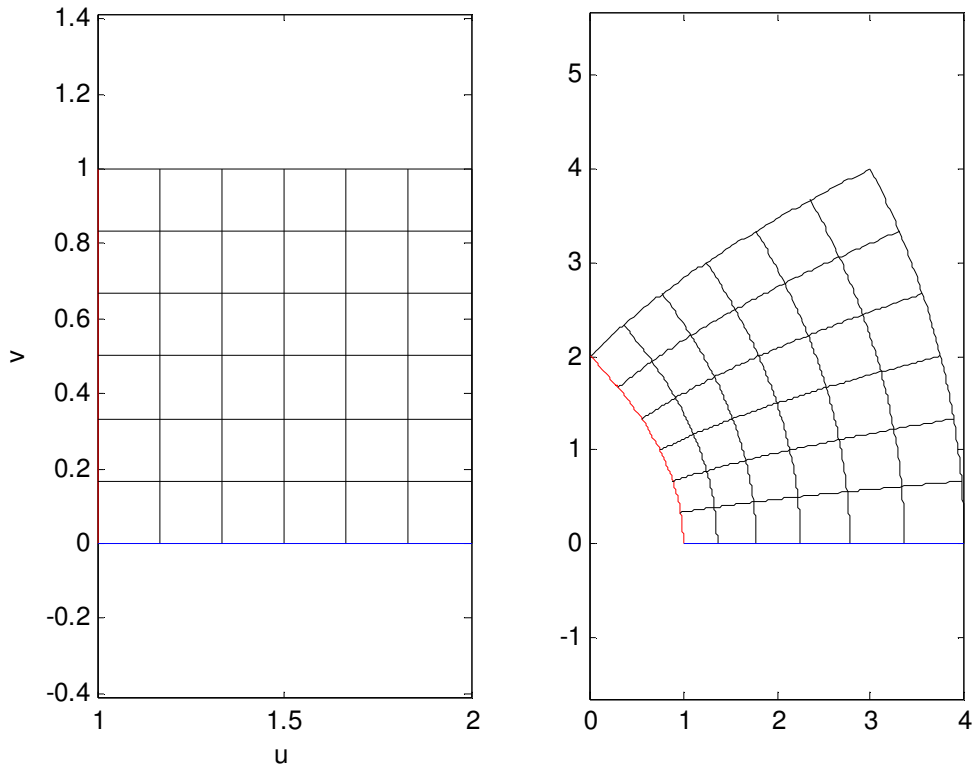


Figura 5.36

Hit return to continue % para ver los paralelogramos aproximados sobreimpuestos sobre G

```
A =
   10.6481
```

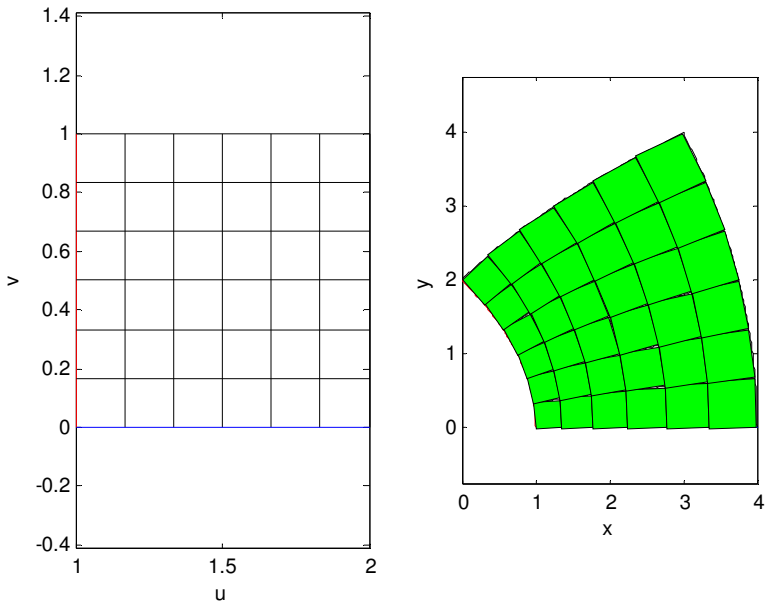


Figura 5.37

La expresión (5) se extiende a la integral de una función *h (x,y)* sobre el conjunto G, siendo la fórmula general

$$\iint_G h(x,y)dA(x,y) = \iint_R \bar{h}(u,v)\left|J(u,v)\right|dA(u,v) \qquad (6)$$

La función $\bar{h}(u,v) = h(f(u,v),g(u,v))$ es la composición de la transformación *T* y la función h, llamando $\bar{h}$ el retorno de *h* al rectángulo *R*.
El cambio de variable de (6) sirve para situaciones más generales donde g sea la imagen bajo una transformación *T* de otro conjunto curvilíneo $\overline{G}$, en esta situación se dirá que $\overline{G}$ es el retorno de G bajo la transformación *T*; esta técnica permite que se apliquen técnicas numéricas de integración sobre rectángulos, para integrar $\bar{h}$ sobre *R*.

5.14.3 Regiones horizontal y verticalmente simples G

Si G es verticalmente simple región: G={*(x,y):a≤x≤b,c(x)≤y≤d(x)*}, sea el rectángulo *R* dado por *R={(u,v):a≤u≤b,0≤v≤1}*, la transformación simple
T(u,v)=(f(u,v),g(u,v))=(u,(1-v)c(u)+vd(u)) mapea *R* sobre G, *v*=0 en la curva *c(x)=y*,*v*=1 en *y=d(x)*; el jacobiano de esta transformación es:

J(u,v)=(1 0;(1-v)c'(u)+vd'(u) d(u)-c(u)) y $\left|J(u,v)\right| = \left|d(u)-c(u)\right|$, la integral sobre G será

$$\iint_G h(x,y)dA(x,y) = \iint_R \bar{h}(u,v)\left|d(u)-cd(u)\right|dvdu \text{ , con}$$

$$\bar{h}(u,v) = h(u,(1-v)c(u)+vd(u))$$

Ejemplo. Si h(x,y)=$\cos^2$(x)exp(-xy) con G={*(x,y):x(x-3)≤y≤sen(x), 0≤x≤π/2*}

$$\iint_G h(x,y)dA(x,y) = \int_0^{\pi/2}\int_0^1 \bar{h}(u,v)\left|senu-u(u-3)\right|dvdu \text{ , con}$$

$$\bar{h}(u,v) = h(u,(1-v)c(u)+vd(u)) = \cos^2(u)e^{-u(1-v)u(u-3)+vsen(u))}$$

Se empleará *simp2*, el integrando *q(u,v)*= $\bar{h}(u,v)(d)u)-c(u))$
Debido a la complejidad de la función la volcamos en un m

```
function z=q(u,v)
x=u;
y=(1-v).*u.*(u-3)+v.*sin(u);
z=cos(x).^2.*exp(-x.*y).*abs(sin(u)-u.*(u-3));
.....
```

Desde el prompt

```
>>format long
>> esquinas=[0 pi/2 0 1];
>> simp2('q',esquinas)

the number of subivisions n and m in each direction must be even
enter the number of subdivisions in x and y direction as [n m] [30 20]

subdiv =
   30    20

Approximate value of the integral using Simpsons rule

ans =
   2.032203488537526
```

Aumentando el número de subdivisiones

```
>> simp2('q',esquinas)

the number of subivisions n and m in each direction must be even
enter the number of subdivisions in x and y direction as [n m] [30 20]

subdiv =
   30    20

Approximate value of the integral using Simpsons rule

ans =
   2.032203488537526
```

La diferencia está en el orden de 0,00001, o sea que el error absoluto en el primer resultado es de ese orden.

5.15 INTEGRALES TRIPLES

Suponiendo un sólido R ={$(x,y,z): a_1 \le x \le a_2, b_1 \le y \le b_2, c_1 \le z \le c_2$},se tiene:

$\iiint_R f(x,y,z)dV = \int_a^{a_2}\int_{b_1}^{b_2}\int_{c_1}^{c_2} f(x,y,z)dzdydx$, pudiendo variarse en 6 diferentes ordenes para la integración, se puede resolver con operador simbólico ***int*** tres veces, o usando *simp3.m*

Ejemplo: sea el sólido cubo unidad, con esquinas en el origen; entonces *R=[0 1 0 1 0 1]*, con $f(x,y,z)=(y+z)sen(\pi(x^2+y^2))$; efectuamos primero la integración simbólica tres veces,luego se sigue con el *simp3.m*:

```
>> syms x y z
>> ff=(y+z)*sin(pi*(x^2+y^2))

ff =
  (y+z)*sin(pi*(x^2+y^2))

>> int(int(int(ff,x,0,1),y,0,1),0,1)

ans =
   1/2*FresnelS(2^(1/2))*FresnelC(2^(1/2))+1/2*FresnelC(2^(1/2))*2^(1/2)/pi
>> double(ans)

ans =
   0.307849384708839

>> f=inline('(y+z).*sin(pi*(x.^2+y.^2))','x','y','z')

f =
     Inline function:
     f(x,y,z) = (y+z).*sin(pi*(x.^2+y.^2))

>> esquinas=[0 1 0 1 0 1]

esquinas =
     0     1     0     1     0     1

>> simp3(f,esquinas)

the number of subdivisions m,n and p in each direction must be even
enter the number of subdivisions [n m p]  [20 20 20]

subdiv =
   20    20    20

ans =
    0.307863142943807

>> simp3(f,esquinas)
```

the number of subdivisions m,n and p in each direction must be even
enter the number of subdivisions [n m p] [40 40 40]

subdiv =
40 40 40

ans =
0.307850232003580

las funciones de Fresnel son:

$$FresnelC(x) = \sqrt{\frac{2}{\pi}} \int_0^{\sqrt{x}} \cos(t^2)dt$$

$$FresnelS(x) = \sqrt{\frac{2}{\pi}} \int_0^{\sqrt{x}} sen(t^2)dt$$

Las tres respuestas difieren en el orden de 10^{-6}
Similarmente a los casos bidimensionales, la integración sobre regiones no rectangulares debe usarse una transformación *T(u,v,w)*.
También se puede utilizar el cambio de variables, a coordenadas esféricas.
Ejemplo: sea g la región entre las esferas de radio *r*=2 y *r*=4, centradas en el origen, suponiendo rellena con un material de densidad variable $\rho(x,y,z)=1+cos(x)$, se desea averiguar la masa y la densidad media de la región.

La masa viene dada por $\iiint_G \rho(x,y,z)dV(x,y,z)$

La región entre las esferas está dada por $2 \le r \le 4$, $0 \le \theta \le 2\pi$ y $0 \le \varphi \le \pi$, empleando el cambio de variable.

$\iiint_G \rho(x,y,z)dV(x,y,z) = \int_o^{\pi} \int_0^{2\pi} \int_2^4 [1+\cos(r\cos\theta sen\varphi)]r^2 sen\varphi dr d\theta d\varphi$, empleamos simp3:

Escribimos el archivo m de la función rrho:

```
function out=rrho(r,theta,phi)
x=r.*cos(theta).*sin(phi);
out=(1+cos(x)).*r.^2.*sin(phi);
.....
```

Desde el prompt

```
>> esquinas=[2 4 0 2*pi 0 pi];
>> simp3('rrho',esquinas)
```

the number of subdivisions m,n and p in each direction must be even
enter the number of subdivisions [n m p] [20 60 30]
subdiv =

```
   20    60    30

ans =
   2.360351669395974e+002

>> simp3('rrho',esquinas)

the number of subdivisions m,n and p in each direction must be even
enter the number of subdivisions [n m p]  [40 120 60]

subdiv =
   40   120    60
ans =
   2.360324035495266e+002
```

la diferencia entre las respuestas es del orden de 3.10^{-3}, *será el error absoluto en la primera respuesta, el relativo será del orden de* $1.5.10^{-5}$; *la densidad media será igual a la masa dividida el volumen* $(4/3)(4^3-2^3)$

5.16 INTEGRALES ESCALARES SOBRE CURVAS Y SUPERFICIES

Sobre curvas

Tienen la forma $\int_C fds$, con *s* la longitud de arco y *f* una función definida sobre la curva(en términos de s o en coordenadas de un punto sobre C)

Ejemplo. Suponiendo que la densidad de un alambre curvado depende de la longitud de arco entre los extremos del alambre, la masa vendrá dada por:

$m = \int_C \rho(s)ds = \int_0^L \rho(s)ds$ con $\rho(s)$ la densidad lineal y *l* la longitud de arco.

Con las integrales siguientes, se hallan las coordenadas del centro de masa del alambre:

$$\bar{x} = \frac{1}{m}\int_C x\rho(s)ds\ ,\ \bar{y} = \frac{1}{m}\int_C y\rho(s)ds\ ,\ \bar{z} = \frac{1}{m}\int_C z\rho(s)ds$$

Tomando c el semicírculo de radio 2 en la plano superior, suponiendo que $\rho(s) = \sqrt{1+s^2}$, con el arco medido desde el punto (2,0), *L*=2π, entonces la masa es.

$m = \int_0^{2\pi} \sqrt{1+s^2}\,ds$, ejecutando simbólicamente:

```
>> syms s
>> densidad=sqrt(1+s^2);
>> int(densidad,0,2*pi)
ans =
   pi*(1+4*pi^2)^(1/2)-1/2*log(-2*pi+(1+4*pi^2)^(1/2))

>> masa=double(ans)
```

masa =

21.2563

Para calcular las coordenadas del centro de masa parametrizamos C por

$\mathbf{r}(t)=[2\cos(t),2\operatorname{sen}(t),0]$, $0\le t\le\pi$

$ds=\left|\left|\mathbf{r}'(t)\right|\right|dt=2dt$

como $s(t)=2t$ y $\rho(s)=\sqrt{1+4t^2}$, al ser $x=2\cos(t)$ sobre la curva, se tendrá:

$\bar{x}=\frac{1}{m}\int_0^{2\pi}2\cos(t)\sqrt{1+4t^2}\,dt$, empleando quad8:

```
>> f=inline('4*sqrt(1+4*t.^2).*cos(2*t)');
>> xbar=(1/masa)*quad(f,0,pi)
```

xbar =

0.0364

Sobre superficies

Si la superficie S está dada como el grafico de $z=f(x,y)$ continuamente difrenciable sobre un conjunto G, la fórmula del área de S:

$Area(S)=\iint_G\sqrt{1+f_x^2+f_y^2}\,dA(x,y)$

Para $g(x,y,z)$

$\iint_G g(x,y,f(x,y))\sqrt{1+f_x^2+f_y^2}\,dA(x,y)$ con $dS=\sqrt{1+f_x^2+f_y^2}\,dA(x,y)$, entonces la integral escalar de superficie será

$\iint_G g(x,y,z)dS$

Ejemplo. Sea la superficie dada por $z=f(x,y)=\operatorname{sen}(x^2+y^2)$ sobre $R=[0,2]x[0,2]$, s área viene dada

por: $\iint_R\sqrt{1+\cos^2(x+y^2)(1+4y^2)}dA(x,y)$, estimamos por la regla de simpson

```
>> h=inline('sqrt(1+cos(x+y.^2).^2.*(1+4*y.^2))','x','y');
>> simp2(h,[0 2 0 2])
```

the number of subivisions n and m in each direction must be even
enter the number of subdivisions in x and y direction as [n m] [20 20]

subdiv =

20 20

Approximate value of the integral using Simpsons rule

ans =

7.2589

Si S es una superficie de metal con densidad $\rho(x,y,z) = x + y + z$, la masa será:

$m = \iint_R (x + y + sen(x + y^2))\sqrt{1 + \cos^2(x + y^2)(1 + 4y^2)}\,dA(x,y)$, la coordenada z del centro de masa es

$1/m \iint_R (x + y + sen(x + y^2))sen(x + y^2)\sqrt{1 + \cos^2(x + y^2)(1 + 4y^2)}\,dA(x,y)$,

usaremos *dblquad*, escribiendo los m de las funciones

```
function integrando=h(x,y)
z=sin(x+y.^2);
densidad=x+y+z;
elementoarea=sqrt(1+cos(x+y.^2).^2*(1+4*y.^2));
integrando=densidad.*elementoarea;
.....
function integrando=h1(x,y)
z=sin(x+y.^2);
integrando=z.*h(x,y);
.....
```

Desde el prompt

```
>>masa=dblquad('h',0,2,0,2)
masa =
   17.0009
>>zbar=(1/masa)*dblquad('h1',0,2,0,2)
zbar =
   0.2285
```

Superficies dadas paramétricamente

El elemento de área de superficie para S en forma paramétrica es:

$$dS = \|n(u,v)\|dA(u,v) = ((y_u z_v - y_v z_u)^2 + (x_v z_u - x_u z_v)^2 + (x_u y_v - x_v y_u)^2)^{0.5}\,dA(u,v)$$

El area de S= $\iint_R \|n(u,v)\|dA(u,v)$ (7) y la integral de *f(x,y,z)* es

$$\iint_S f(x,y,z)dS = \iint_R f(x(u,v),y(u,v),z(u,v))\|n(u,v)\|dA(u,v) \quad (8)$$

Para una superficie de revolución, el elemento de área de la superficie es:

$$dS = \|n(u,v)\|dA(u,v) = |x_0(v)|\sqrt{[x_0'(v)]^2 + [z_0'(v)]^2}\,dudv$$

Así el area de una superficie de revolución se reduce a una integral en *v*

$$Area(S) = 2\pi \int_a^b |x_0(v)|\sqrt{[x_0'(v)]^2 + [z_0'(v)]^2}\,dv \qquad (9)$$

Ejemplo: se quiere hallar área de una banana, empezamos con una elipsoide, que es una superficie de revolución

$x(u,v)=\cos u \cos v$

$y(u,v)=\text{senu}\cos v$

$z(u,v)=3\text{sen}v, \quad 0\le u\le 2\pi, \quad -\pi/2\le v\le \pi/2$

curvando en la dirección x agregando una perturbación sobre la coordenada x :

$x(u,v)=\cos u \cos v+2\text{sen}^2 v$

la elipsoide y la banana se muestran, el vector normal es

$\boldsymbol{n}(u,v)=[3\cos u \cos^2 v,\ 3\text{sen}u \cos^2 v,\ \cos v\text{sen}v -4\cos u\ \text{sen}v\cos^2 v]$

y el área dada por la integral $\int_0^{2\pi}\int_{-\pi/2}^{\pi/2} \|n(u,v)\| dA(u,v)$

donde $\| \boldsymbol{n}(u,v) \| = 9\cos^4 v + (\cos v\text{sen}v -4\cos u\ \text{sen}v\cos^2 v)^2$

se escribe un m para **n**

```
function out=da(u,v)
n1=3*cos(u).*cos(v).^2;
n2=3*sin(u).*cos(v).^2;
n3=cos(v).*sin(v)-4*cos(u).*sin(v).*cos(v).^2;
out=sqrt(n1.^2+n2.^2+n3.^2);
.....
```

Desde el prompt**:**

```
>> %grafico de la banana
>> u=linspace(0,2*pi,41);
>> v=linspace(-.5*pi,.5*pi,41);
>> [U,V]=meshgrid(u,v);
>> X=cos(U).*cos(V)+2*sin(V).^2;
>> Y=sin(U).*cos(V);
>> Z=3*sin(V);
>> surf(X,Y,Z);
```

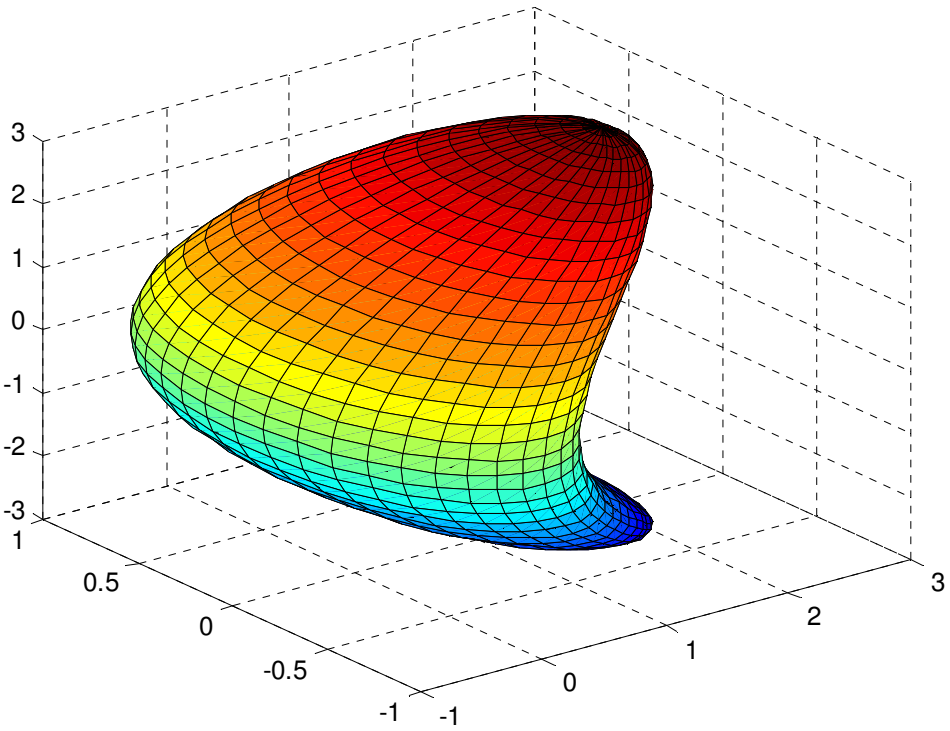


Figura 5.38

```
>> %calculamos el área de la superficie
>> simp2('da',[0 2*pi -.5*pi .5*pi])

the number of subivisions n and m in each direction must be even
enter the number of subdivisions in x and y direction as [n m] [100 100]

subdiv =
   100   100

Approximate value of the integral using Simpsons rule

ans =
   33.3702
```

5.17 SUPERFICIES COMPUESTAS DE TRIÁNGULOS

MATLAB emplea una representación de bloques triangulares con el comando *surf*, se sabe que en general no se ajusta un plano por cuatro puntos en el espacio, así lo que puede parecer un paralelogramo sería, en realidad dos triángulos.

5.17.1 Domos geodésicos

En la construcción de estructuras, empleando triángulos, se prefieren que estos sean equiláteros o isósceles, aparecen los domos *geodésicos*(fuertes y livianos) sin necesidad de columnas interiores para soportarlos. La más simple es un icosaedro, constituído de poliedros con sus caras iguales al mismo triángulo equilátero.

>>gdome

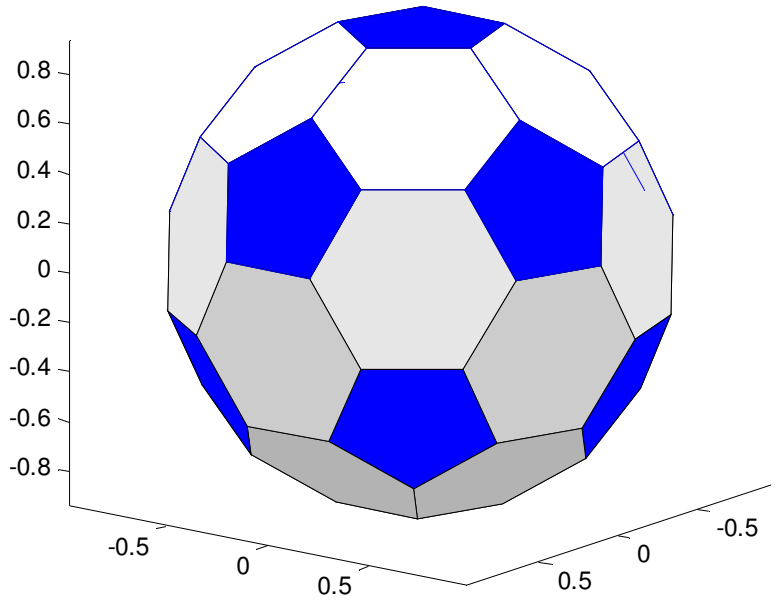


Figura 5.39

hit return to continue
>> return

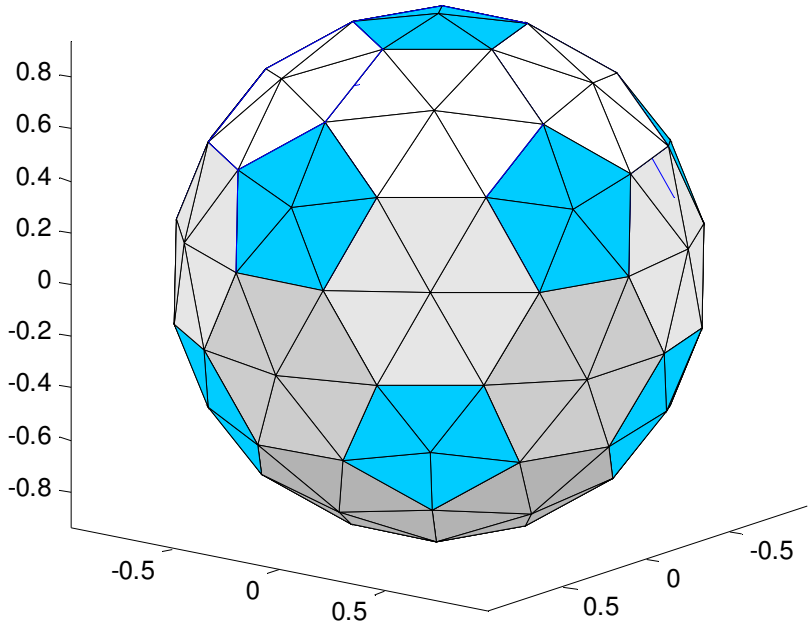


Figura 5.40

También se puede aproximar una figura suave con parches triangulares, el área de la superficie suave se aproxima las areas de estos parches, al menos como una aproximación, pero evita conocer derivadas como en el caso de integración de las funciones.

Ejemplo: sea $f(x,y)=xy$ sobre $R=\{0\le x,y\le 1\}, z_1=z_2=0, z_4=0$ y $z_3=1$(rectángulo), la triangulación da un área de la superficie de $1+\sqrt{3}/2\approx 1.366$, el área verdadera de la superficie es:

$$\int_0^1\int_0^1 \sqrt{1+f_x^2+f_y^2}\,dxdy = \int_0^1\int_0^1 \sqrt{1+x^2+y^2}\,dxdy = 1.2808$$

Generalizando, si se da *S* como el gráfico de una $z=f(x,y)$ sobre R=[*a*,*b*]x[*c*,*d*], se calcula el area de *S* a partir de los parches triangulares del siguiente modo: se introduce una malla en *R* con puntos (x_j,y_i).
$a=x_1<\ldots<x_{n+1}=b$, $c=y_1<\ldots<y_{n+1}=d$ y valores $z_{i,j}=f(x_j,y_i)$ en esos puntos; dividiendo cada rectángulo $R_{i,j}$ en triángulos $\sigma_{i,j}$ y $\tau_{i,j}$, el area del rectángulo será:

$$A=\sum_{i=1}^{m}\sum_{j=1}^{n}[area(\sigma_{i,j})+area(\tau_{i,j})] \quad (9) \text{ donde}$$

$$area(\sigma_{i,j})=0.5\sqrt{\Delta x^2\Delta y^2+\Delta x^2(z_{i+1,j}-z_{i,j})^2+\Delta y^2(z_{i,j+1}-z_{i,j})^2}$$

$$area(\tau_{i,j})=0.5\sqrt{\Delta x^2\Delta y^2+\Delta x^2(z_{i+1,j}-z_{i+1,j+1})^2+\Delta y^2(z_{i+1,j+1}-z_{i+1,j+1})^2}$$

El archivo *tsurf* produce la aproximación de los parches triangulares a una superficie y calcula la suma de las áreas de los triángulos.

Ejemplo,si $f(x,y)=x^2+y^2$ en el cuadrado[0,2]x[0,2]

```
>> f=inline('x.^2+y.^2','x','y');
>> %calculamos el area de los parches triangulares
>> [X,Y]=meshgrid(0:2);
>> Z=f(X,Y);
>> tsurf(X,Y,Z)

ans =
   12.7242
```

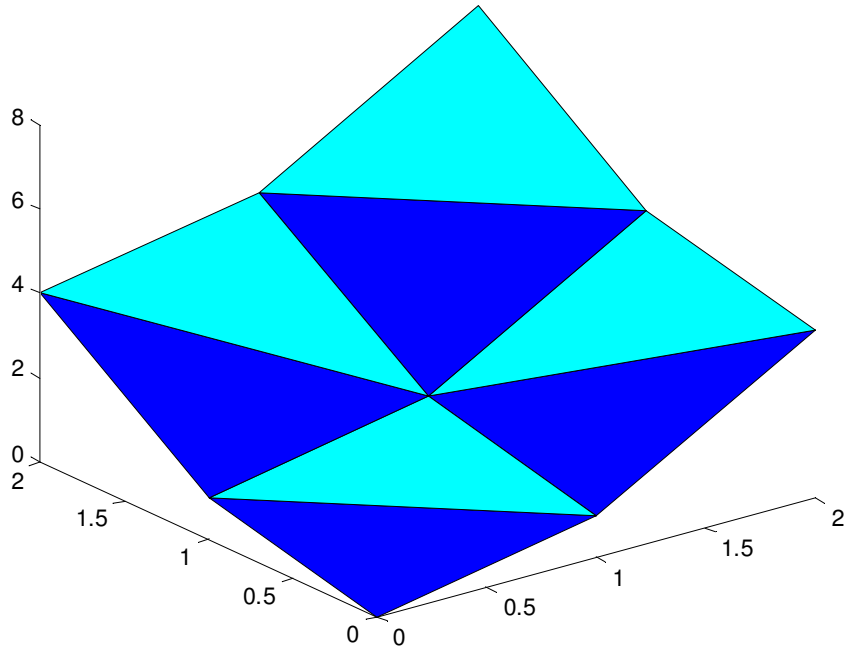


Figura 5.41

```
>>%graficamos la supefcie con una malla más refinada
>>[XX,YY]=meshgrid(0:.1:2);
>>ZZ=f(XX,YY);
>>surf(XX,YY,ZZ);colormap(gray);shading interp;
```

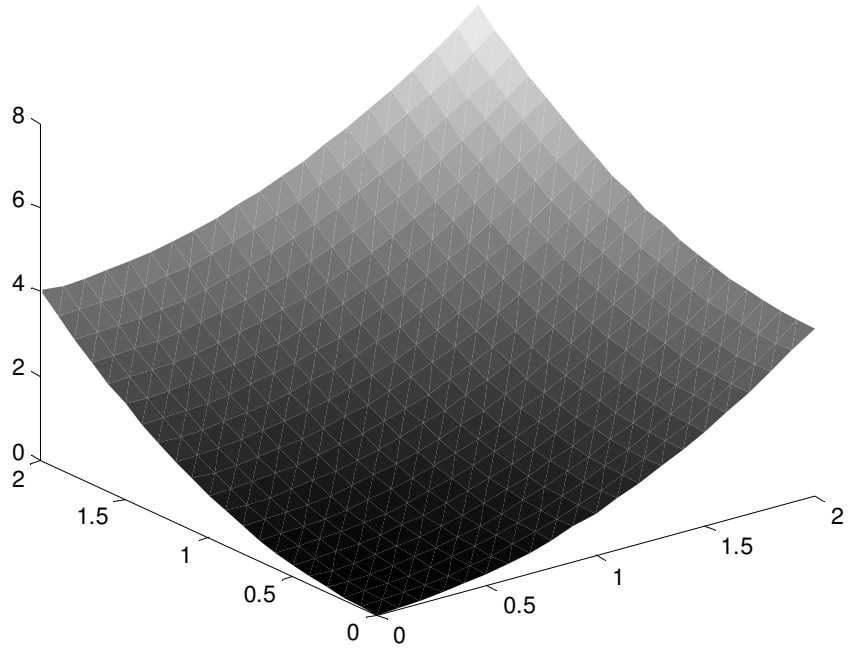


Figura 5.42

```
>> %calculamos el area de la superficie numéricamente
>> g=inline('sqrt(1+4*(x.^2+y.^2))','x','y');
>> esquinas=[0 2 0 2];
>> simp2(g,esquinas)

the number of subivisions n and m in each direction must be even
enter the number of subdivisions in x and y direction as [n m] [40 40]

subdiv =

    40    40
Approximate value of the integral using Simpsons rule

ans =
   13.0046
```

Se puede incrementar el número de triángulos, buscando convergencia al área de la superficie.

5.18 PROBLEMA DE LA SUPERFICIE MÍNIMA

Un problema básico geométrico es hallar la superficie de menor área que cubre un marco de alambre, ej la *catenoide*(superficie de revolución formada girando *ccosh(c/x)* sobre el eje *x*.

Problema de la meseta

Dado un conjunto G en el espacio *x,y* y una función f definida sobre la frontera de G, sea *A* el conjunto de todas las funciones diferenciables *u(x,y)* sobre G tal que *u=f* sobre la frontera G; buscamos esa función *u* que minimice el área de la superficie

$$\iint_G \sqrt{1+u_x^2+u_y^2}\,dA$$

Si esta función minimizadora *f* tiene segundas derivadas parciales continuas, *u* satisface la *EDP* no lineal

$$\frac{\partial}{\partial x}\left(\frac{u_x}{\sqrt{1+u_x^2+u_y^2}}\right)+\frac{\partial}{\partial y}\left(\frac{u_y}{\sqrt{1+u_x^2+u_y^2}}\right)=0 \quad (10)$$ ecuación de superficie mínima, usándose los retazos triangulares para su solución(elementos finitos)

Sea $R=[a,b]x[c,d]$, se considera una función $f(x,y)$ definida en *R*, usando sus restricciones a cada lado de *R* como los valores de frontera para la superficie. Reconstruye una superficie S, de triángulos, que coincide con *f* sobre los lados de *R*, buscando también ajustar los triángulos dentro de *R* para hacer el área de s lo menor posible. Con puntos (x_j,y_i) en *R*, calculando el área con triángulos según(6), los valores $z_{i,j}=f(x_j,y_i)$, se dan para cada (x_j,y_i) en la frontera de *R*, para el interior de *R* se eligen los valores, entonces el área de (6) es una función de $(n-1)(m-1)$ variables $z_{i,j}$: $A=A(z_{2,2},z_{2,3},\ldots,z_{2,n},z_{3,2},\ldots,z_{3,n},\ldots,z_{m,2},\ldots,z_{m.n})$

ejemplo: sea $R=[1,4]x[1,4]$, lo dividimos en cuadrados de lados 1 y cada uno de ellos en dos triángulos,m=n=3, quedando 18 triángulos y cuatro puntos interiores

$f(x,y)=x^2+y^4/20$ da los valores de frontera la suma para el área de (6) tendrá 18 términos, siendo la sima función de las alturas $z_{2,2},z_{2,3},z_{3,2},z_{33}$,de la superficie en los cuatro puntos interiores, adoptamos la notación:

$a=z_{2,2}$, $b=z_{2,3}$, $c=z_{3,2}$, $d=z_{33}$, así $A=A(a,b,c,d)$;creamos el *m* para esta función:

```
function out=A(a,b,c,d)
f=inline('x.^2+(y.^4)/20','x','y');
x=[1 2 3 4];y=[1 2 3 4];
[X,Y]=meshgrid(x,y);
Z=f(x,y);
Z(2,2)=a;Z(3,2)=b;Z(3,2)=c;Z(3,3)=d;
tsurf(X,Y,Z)
```

se ve que el archivo reemplaza los valores de $f(2,2),f(2,39),f(3,2),f(3,3)$ en la matriz Z con los valores de a,b,c,d y calcula el área de la superficie de los retazos triangulares. Para $a=8.3,b=13.05,c=12.55,d=17.05$, se genera desde el prompt

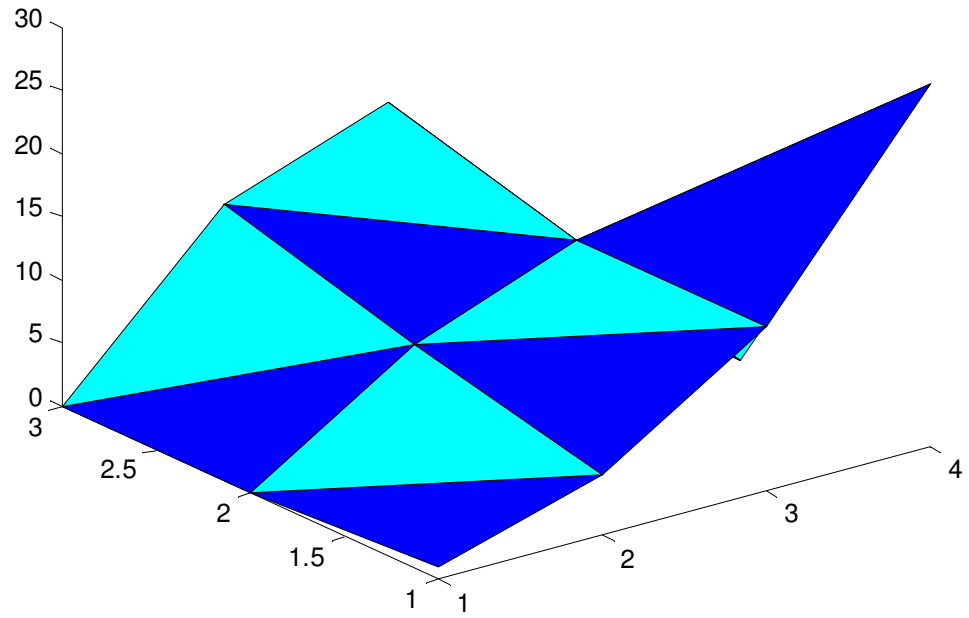


Figura 5.43

Lógicamente *MATLAB* tiene rutinas de minimización como *fmin*

5.19 INTEGRALES DE CAMPOS VECTORIALES SOBRE CURVAS Y SUPERFICIES

5.19.1 Campos vectoriales

Un campo vectorial bidimensional se escribe en coordenadas cartesianas con la funciones $u(x,y)$ y $v(x,y)$, en cada punto(x,y) en un conjunto G, se tiene
F=[*u(x,y)* , *v(x,y)***]**, se muestran con el comando quiver, con las funciones como *inline* o en *m.*, llamándolas con *quiver(X,Y,u(X,Y),v(X,Y)).*

Ejemplo. F=[1,x+y²] sobre R=[-2,3]x[-1,3]

```
>> u=inline('0*x+1','x','y');
>> v=inline('x+y.^2','x','y');
>> x=linspace(-2,3,11);
>> y=linspace(-1,2,11);
>> [X,Y]=meshgrid(x,y);
>> U=u(X,Y);V=v(X,Y);
>> quiver(X,Y,U,V)
>> axis image
```

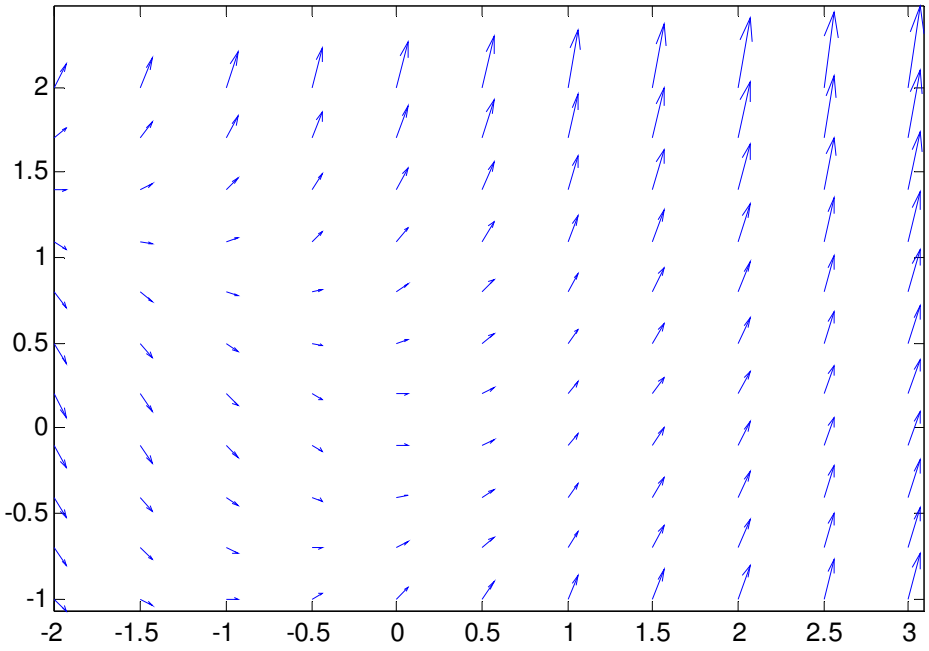


Figura 5.44

Axis image cumple la misma función que *axis equal*, pero no deja un espacio extra en los bordes de la parcela, donde las flechas pueden colgarse, la longitud de los vectores está escalada, para ver un campo vectorial unidad en la misma dirección de *f*, se procede:

```
>> U1=U./sqrt(U.^2+V.^2)
>> V1=V./sqrt(U.^2+V.^2);
>> quiver(X,Y,U1,V1);axis image
```

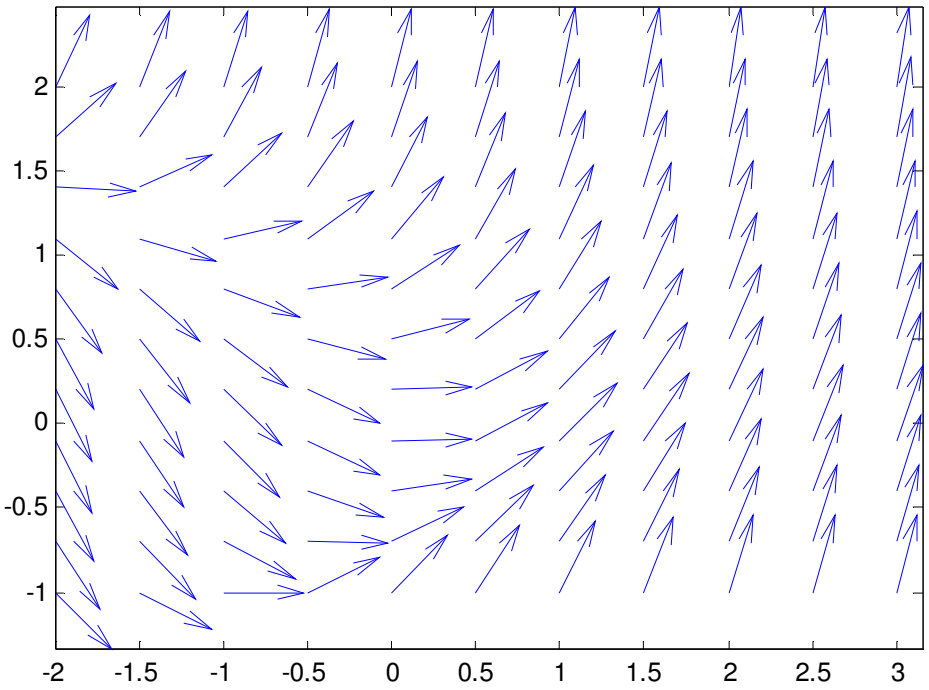


Figura 5.45

Ejemplo para *3D*: sea $F(x,y,z)=[1,x+y^2,z]$, n el rectángulo $R=[-2,-1,-1]x[3,2,1]$, el *script(d3)*

```
>> u=inline('1+0*x','x','y','z');
v=inline('x+y.^2','x','y','z');
w=inline('z','x','y','z');
x=linspace(-2,3,6);
y=linspace(-1,2,6);
[X,Y]=meshgrid(x,y);
for z=-1:.4:1
    Z=z+0*X;
    U=u(X,Y,Z);
    V=v(X,Y,Z);
    W=w(X,Y,Z);
    quiver3(X,Y,Z,U,V,W)
    hold on
end
```

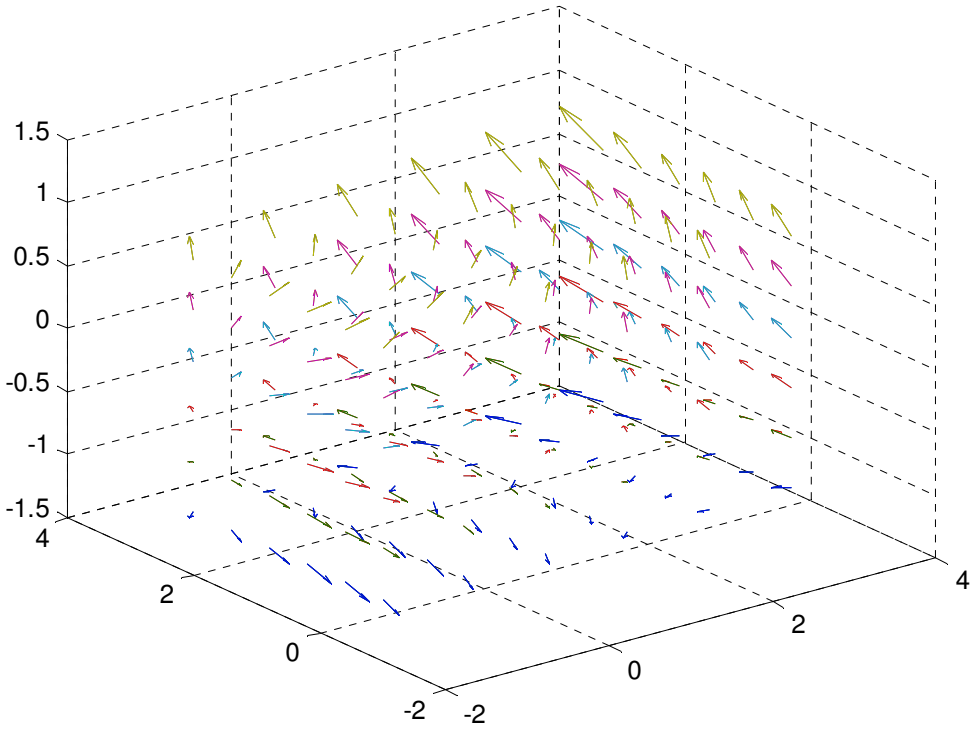


Figura 5.46

Llamamos desde elprompt *d3* y enter, para la figura 5.46.
Alternativamente, para hacerlo más dinámico, se maneja el script:

```
>> z=linspace(-1,1,6);
>> [X,Y,Z]=meshgrid(x,y,z);
>> U=u(X,Y,Z);
>> V=v(X,Y,Z);
>> W=w(X,Y,Z);
>> quiver3(X,Y,Z,U,V,W)
```

5.19.2 Integrales de línea

Para el campo *F=[u,v,w]* orientado en una curva C de *P* a *Q* se tiene

$$\int_C \vec{F}.d\vec{r} = \int_C udx + vdy + wdz \quad (11)$$

Como límite de sumas: $\int_C \vec{F}.d\vec{r} = \lim_{n\to\infty} \sum_{i=1}^{n} F(P_i).(P_{i+1} - P_i)$, P_i secuencia de puntos sobre C, entonces de la definición $\int_C \vec{F}.d\vec{r} = \int_C \vec{F}.\vec{T}ds$, con *T* vector unitario tangente, la integral de línea representa el trabajo realizado por $\boldsymbol{F}$ sobre un cuerpo siguiendo la línea C en una orientación dadda.

Si C es parametrizada por una función continuamente diferenciable $\boldsymbol{r}(t)$ en *[a,b]*, con $\boldsymbol{r}(a)=P$ y $\boldsymbol{r}(b)=Q$, se escribirá: $\int_C \vec{F}.d\vec{r} = \int_C \vec{F}(\vec{r}(t))\vec{r}(t)dt$ (12), que se resuelve simbólica o numéricamente

Ejemplo: sea el campo $\boldsymbol{F}(x,y,z)=[x,z,exp(x+y)]$, parametrizada por $\boldsymbol{r}(t)=[tcost,tsent,t^2]$ en $[0,2\pi]$;

```
>> %definir los componentes de f
>> u=inline('x','x','y','z');
>> v=inline('z','x','y','z');
>> w=inline('exp(x+y)','x','y','z');
>> %elegir los valores para regla simpson
>> n=200;
>> t=linspace(0,2*pi,n+1);dt=2*pi/n;
>> s=simpvec(n);
>> %calcular los valores x,y,z a lo largo de la curva
>> x=t.*cos(t);y=t.*sin(t);z=t.^2;
>> %calcular los componentes de rdot a lo largo de la curva
>> xdot=cos(t)-t.*sin(t);
>> ydot=sin(t)+t.*cos(t);
>> zdot=2*t;
>> %calcular los términos del integrando
>> I1=u(x,y,z).*xdot;
>> I2=v(x,y,z).*ydot;
>> I3=w(x,y,z).*zdot;
>> %calcular la integral con simpson
>> integral=dot(s,(I1+I2+I3))*dt/3;
>> integral=dot(s,(I1+I2+I3))*dt/3

integral =
  1.0505e+003
```

Si C no está parametrizada, se puede estimar del siguiente modo, aproximar la curva por un conjunto de segmentos, parametrizando la línea segmento L_j de P_j a

P_{j+1} por

$\mathbf{r}_j(t)=(1-t)P_j+tP_{j+1}\ \ 0\le t\le 1$
con $\mathbf{r}'_j(t)=P_{j+1}-P_j$

$$\int_C \vec{F}.d\vec{r} \approx \sum_{j=1}^{n}\int_{L_j}\vec{F}.d\vec{r}$$

$$= \sum_{j=1}^{n}\int_o^1 \vec{F}(\vec{r}_j(t)).(P_{j+1}-P_j)dt$$

$$= \sum_j (x_{j+1}-x_j)\int_0^1 u\vec{r}_j(t))dt + \sum_j (y_{j+1}-y_j)\int_0^1 v\vec{r}_j(t))dt + \sum_j (z_{j+1}-z_j)\int_0^1 w\vec{r}_j(t))dt$$

Estas integrales se resuelven simbólica o numéricamente
Ejemplo, una curva C define la trayectoria de una partícula, se dan las coordenadas de los puntos en la curva se dan en la tabla:

x	0	0.1	.25	.4	.54	.76	.82	.93	1
y	0	.005	.0312	.0800	.1458	.2888	.3362	.4352	.5

Para el campo de fuerzas ***F****(x,y)=[xcosy,x+y]*, se desea estimar el trabajo sobre la curva C de *P1*=(0,0) a *P9*=(1,.5), con el script:

```
>> x=[0 .1 .25 .4 .54 .76 .82 .93 1];
>> y=[0 0.005 .0312 .0800 .1458 .2888 .3362 .4352 .5];
>> % definir las componentes de F
>> u=inline('x.*cos(y)','x','y');
>> v=inline('x+y','x','y');
>> % formar el vector simpson
>> n=50;s=simpvec(n);dt=1/n;
>> %empezar Trabajo
>> trabajo=0;
>> for j=1:8
%calcular los valores de x e y sobre el segmento j
xx=linspace(x(j),x(j+1),n+1);
yy=linspace(y(j),y(j+1),n+1);
%calcular el integrando
I=(x(j+1)-x(j))*u(xx,yy)+(y(j+1)-y(j))*v(xx,yy)
%calcular la j sima integral
integral=dot(s,I)*dt/3;
trabajo=trabajo+integral
end
trabajo =
   0.9353
```

El archivo *lint* usa el procedimiento del ejemplo para la integral de línea de un campo en *2D*, ***F****=[u(x,y),v(x,u)]* a lo largo de un camino poligonal determinado por el usuario con el clickeado en la figura; se llama con *lint*(*u*,*v*,*esquinas*), en archivo m o inline para *u*,*v*, se debe entrar el número de segmentos *N*, el archivo usa quiver para

graficar el vector campo sobre *R*. También con el archivo se puede encontrar $\int_c xdy$, donde C es cerrado y forma un polígono orientado positivamente, siendo la integral de línea el área del polígono. Recordar que si el campo es conservativo *(u=f_x(x,y),v=f_y(x,y))*, si el campo es el gradiente de un potencial escalar, para cada recinto cerrado la integral será 0.

Teorema de *Green* y *rotor*

El rotor de un campo vectorial tridimensional ***F****(x,y,z)=[u(x,y,z),v(x,y,z),w(x,y,z)], u,v,w* continuamente diferenciables es: *rot(F)=[w_y-v_z,u_z-w_x,v_x-u_y]*, el teorema de *Green* nos da un significado geométrico y físico a esta cantidad.
Sea el caso de sola dependencia de *x,y*: ***F****=[u(x,y),v(x,y),0]*, será *rot(F)=[0,0,v_x-u_y]*, para esta situación sobre C con orientación positiva respecto al conjunto G del plano *xy* donde está definido el campo, el teorema de *Green* establece:

$$\int_C udx + vdy = \iint_G (v_x - u_y)dA(x,y) \quad (13), \text{ equivalentemente}$$

$$\int_C \vec{F}.\vec{T}ds = \int_C \vec{F}.dr = \iint_G rot(\vec{F}).\vec{k}dA(x,y) \qquad /14)$$

La integral de línea del lado izquierda de 13 es el trabajo hecho por *F* alrededor del camino cerrado C; en flujo de fluídos, *F=[u,v]* es el campo de velocidad del fluído, *u,v* componentes de la velocidad en *x,y*, entones la integral de tiempo es la circulación del fluído alrededor del camino cerrado C, ***k****=[0,0,1]*. Con el teorema de *Green*, se toma un pequeño cuadrado de lado h centrado en *(x_0,y_0)*, llamado R_h, dividiendo por h^2, ambos lados de (13), se tiene:

$$\frac{1}{h^2}\int_C udx + vdy = \frac{1}{h^2}\iint_G (v_x - u_y)dA(x,y) = \text{promedio sobre } R_h \text{ de } v_x\text{-}u_y$$

El lado derecho de esta ecuación converge a *v_x(x_0,y_0)-u_y(x_0,y_0)* cuando h tiende a cero, interpretándose el rot(*F*(x_0,y_0)) como la circulación por unidad de area en el punto(*x_0,y_0*)

Ejemplo: sea ***F****=[1-y^2,0]* en la región *-1≤y≤1*, el campo representa el flujo bidimensional de un fluido viscoso de izquierda a derecha en un ducto cuyas paredes son las líneas *y±1*
El fluido pegado a las paredes(sin CF separadas), el *rot(F)* vemos que da *2y****k.***
Usamos el *lint* para obtener la circulación para senderos cada vez menores centrados en (0,1/2), donde *rot(****F****).****k****=1*
El script también calcula el área de los polígonos encerrados.
Primero se toma *esquinas*=[-1 1 -1 1] y calcular la circulación alrededor de de un triángulo con vértices en *P1*=(-.5,0),*P2*=(.5,0),*P3*=(0,1), al dividir la circulación por el área da .6747(valor exacto es 2/3), si se repite en puntos
P1=(-.25,.3),*P2*=(.25,.3),*P3*=(0,.7),dará .8655(valor exacto .8666),

```
>> u=inline('1-y^.2','x','y');v=inline('0*x','x','y');
>> esquinas=[-1 1 -1 1];lint(u,v,esquinas)
enter the number of segments in your path 3
```

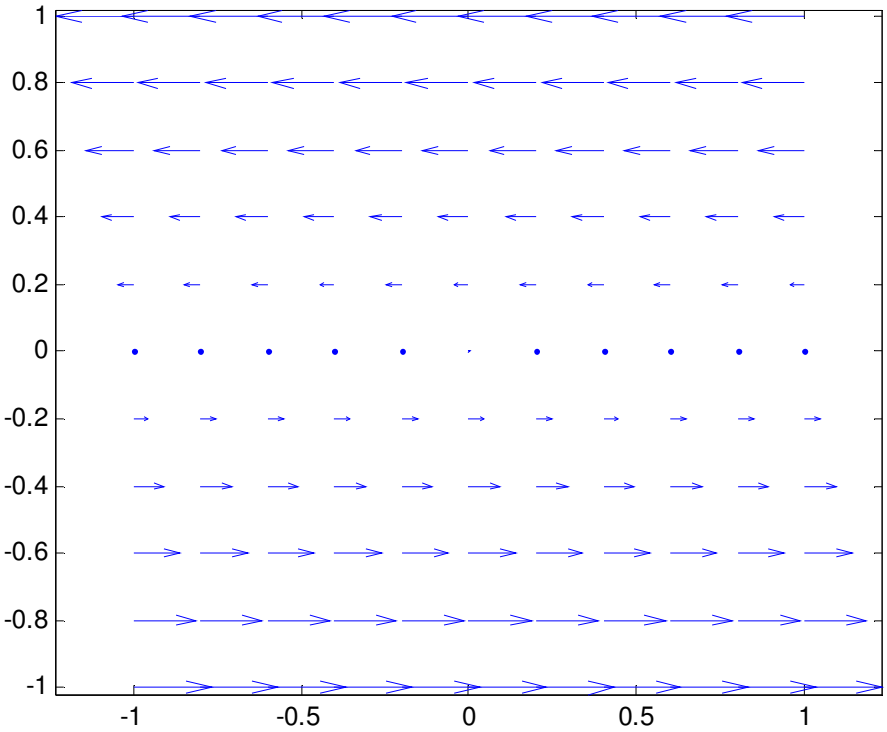


Figura 5.47

El archivo *curl* determina cuando un campo vectorial bidimensional tiene un rotor no cero e indica el signo y magnitud por color; se llama igual que *lint*.

Integral de flujo: se determina por la expresión $\int_C \vec{F}.\vec{n}ds$, (15) con $\boldsymbol{n}$ vector *normal*(grad/su norma)

El archivo *flux2* calcula el flujo de un campo2D a través de una frontera compuesta de segmentos de línea, se opera igual que *lint*.

Ej. Sea el vector $F(x,y)=[u,v]=[x^2/2,y^2/2]$, buscaremos el flujo a través de las fronteras de dos triángulos contenidos en el cuadrado $-2\leq x,y\leq 2$.

```
>> u=inline('.5*x.^2','x','y');
>> v=inline('.5*y.^2','x','y');
>> esquinas=[-2 2 -2 2];
>> flux2(u,v,esquinas) % con enter aparece
enter the number of segments in the path  3   % por ejemplo 3
```

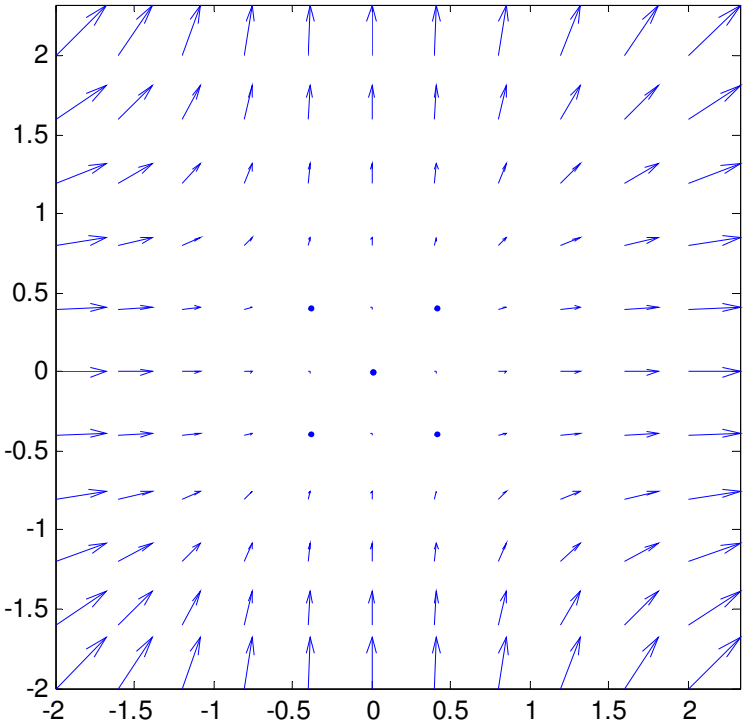


Figura 5.48

Sobre este gráfico, con el ratón generamos los segmentos(se usa el *ginput*), al completar nos dispará el valor de *flux*,
0.6418

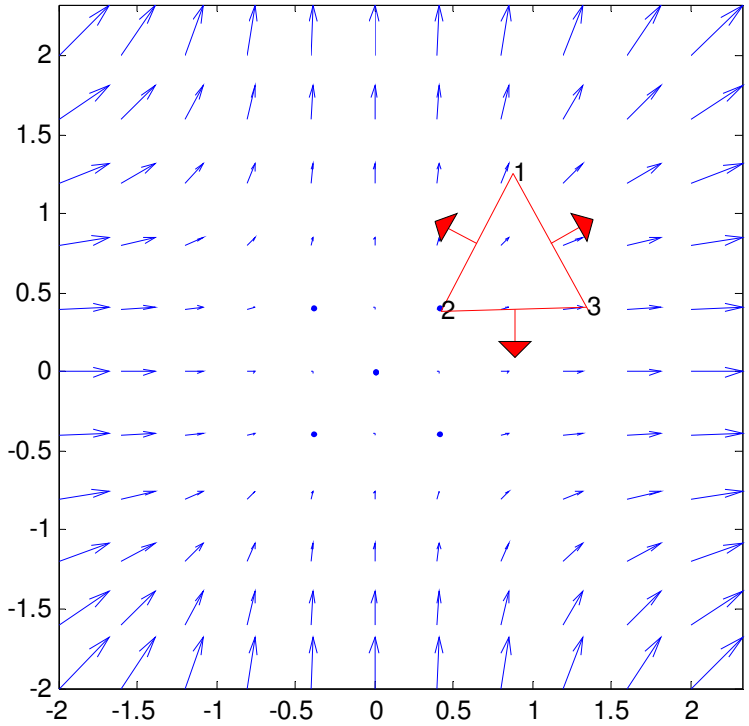


Figura 5.49

5.19.3 Teorema de la divergencia

Para un campo *F* en 3D, *div(F)=)=[u_x(x,y,z)+v_y(x,y,z)+w_z(x,y,z)], función escalar*
Para G un conjunto acotado en el espacio *xyz*, limitado por una superficie orientable S, ***n*** el normal exterior unitario a S, el teorema de la divergencia establece:

$$\iiint_G div(\vec{F})dV(x,y,z) = \iint_S \vec{F} \bullet \vec{n} dS$$

Sea R_h un cubo pequeño con lado h centrado en (x_0,y_0,z_0),al aplicar el teorema sobre R_h y dividiendo por el volumen h^3, se tiene:

$$\frac{1}{h^3}\iiint_{R_h} div(\vec{F})dV = \frac{1}{h^3}\iint_S \vec{F} \bullet \vec{n} dS \quad (17)$$

El lado derecho es el promedio de *div(**F**)* sobre el cubo, si las componentes de ***F*** son contiuamente diferenciables, este valor converge a *div(**F**)* (x_0,y_0,z_0) cuando $h \to 0$, es dcir que para h pequeños: *div(**F**)* $(x_0,y_0,z_0) \approx \frac{1}{h^3}\iint_S \vec{F}.\vec{n} dS$

Las unidades de *div(**F**)* flujo/unidad de volumen, o sea que ***F*** es el vector velocidad de fluido,*u, v* y *w* las componentes sobre *x,y,z*, representando el volumen de fluido a través de la frontera de s en unidad de tiempo.
El archivo *flux2* puede colaborar para el manejo de la divergencia en *2D*
Ejemplo: con *F(x,y)=[x^2/2,sen(πy/2)]*, se debe calcular el flujo y dividir por el área del polígono encerrado para tener una aproximación a *div(**F**)(x,y)=u_x(x,y)+v_y(x,y).*
Tomado los puntos, de acuerdo a la figura, se tiene un valor de flujo:

```
>>u=inline('.5*x.^2','x','y');
>> v=inline('sin(.5*pi*y)','x','y');
>> esquinas=[-1.5 -.5 -1.5 -.5];
>> flux2(u,v,esquinas)
enter the number of segments in the path  4
N = 4
```

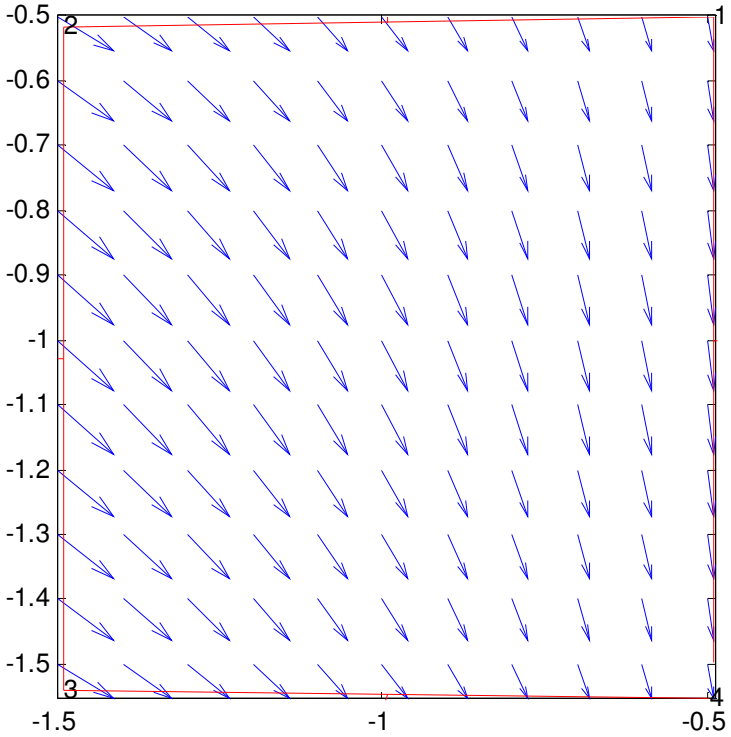


Figura 5.50

>>*area* =

1.0664

total_flux =

-1.1027

Haciendo el cociente *flujo/area*=-1.03, el valor exacto de la divergencia en el centro del cuadrado es -1

CAPITULO 6 - SERIES DE FOURIER

En 1807, Fourier, establece en los trabajos presentados en el Instituto de Francia que cualquier señal periódica puede ser representada por una serie de sumas trigonométricas en senos y cosenos relacionadas armónicamente.
Los argumentos establecidos por Fourier eran imprecisos y en 1829 Dirichlet proporcionó las condiciones precisas para que una señal periódica pueda ser representada por una serie de Fourier.

Fourier obtuvo además, una representación para señales no periódicas, no como suma de senoides relacionadas armónicamente, sino como integrales de senoides, las cuales no todas están relacionadas armónicamente. Al igual que las series de Fourier, la integral de Fourier, llamada *Transformada de Fourier*, es una de las herramientas más poderosas para el análisis de sistemas *LTI (Sistema Lineal Invariante en el Tiempo)*.

6.1 REPRESENTACIÓN DE UNA SEÑAL PERIÓDICA

Una señal es periódica si para algún valor positivo *T*, diferente de cero, se verifica que:

$x(t) = x(t + T)$ para toda *t*.

Para que una señal periódica pueda representarse por una serie de Fourier, debe respetar las condiciones de Dirichlet:

- Que tenga un número finito de discontinuidades en el periodo *T*, en caso de ser discontinua.
- El valor medio en el periodo *T*, sea finito.
- Que tenga un número finito de máximos positivos y negativos.

Si se satisfacen estas condiciones, existe la serie de Fourier y puede escribirse en la forma trigonométrica como:

$$x(t) = a_0 + 2(a_1 \cos wt + a_2 \cos 2wt + a_3 \cos 3wt + \ldots + b_1 senwt + b_2 sen2wt + b_3 sen3wt + \ldots$$

Es decir: $x(t) = a_0 + 2\sum_{k=1}^{\infty} a_k.\cos(kwt) + b_k sen(kwt)$

Los coeficientes a_k y b_k, se obtienen mediante el siguiente cálculo integral:

$$a_k = \frac{1}{T}\int x(t)\cos(kwt)dt \qquad b_k = \frac{1}{T}\int x(t)sen(kwt)dt$$

6.2 CONDICIONES DE DIRICHLET

Sea $f : R \rightarrow R$ una función periódica de período *T*. Se dice

que *f* satisface las condiciones de Dirichlet si en cada período la función *f : [0, T] → R* es continua salvo en un número finito de discontinuidades todas ellas de salto y sólo tiene una cantidad finita de máximos y mínimos locales estrictos. Puede probarse, en particular, que si una función periódica es tal que ella y su derivada están definidas y son continuas salvo un número finito de discontinuidades de salto, entonces dicha función verifica las condiciones de Dirichlet. Prácticamente todas las funciones –señales– de interés en las aplicaciones las verifican.

6.3 TEOREMA DE CONVERGENCIA DE DIRICHLET

Sea $f : R \to R$ una función periódica de período T que satisface las condiciones de Dirichlet y sea

$$f(t) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty}[a_n.\cos(nwt) + b_n sen(nwt)]$$ con $\omega = 2\pi/T$, su serie de Fourier(1)

Si *f* es continua en un punto *t*, entonces la serie de Fourier converge en ese punto a *f (t)*, o sea, $\frac{1}{2}a_0 + \sum_{n=1}^{\infty}[a_n.\cos(nwt) + b_n sen(nwt)] = f(t)$ (2)

Si *f* tiene una discontinuad de salto en un punto *t*, entonces la serie de Fourier converge en ese punto al punto medio del salto, o sea,

$$\frac{1}{2}a_0 + \sum_{n=1}^{\infty}[a_n.\cos(nwt) + b_n sen(nwt)] = \frac{f(t^+) + f(t^-)}{2}$$

donde, $f(t^-) = lim_{\tau \to 0, \tau > 0} f(t^{-\tau})$ indica el límite de *f* en t por la izquierda y $f(t+) = lim_{\tau \to 0, \tau > 0} f(t^{+\tau})$ indica el límite de *f* en *t* por la derecha.

El teorema nos dice, en particular, que si f satisface las condiciones de Dirichlet y redefinimos el valor de *f* en cada punto de discontinuidad como el punto medio del salto,o sea, poniendo $f(t) = f(t^-) + f(t^+)/2$,, entonces la suma de la serie de Fourier coincide con *f (t)* en cada $t \varepsilon R$. Por eso, en lo que sigue, y salvo que se diga lo contrario, supondremos que esto se cumple.

Desarrollos de medio intervalo. Sea f : [0, L] → R una función continua a trozos. Sea T = 2L, entonces la extensión *T -periódica y par de f* es la función definida en el intervalo $[-L, L] = [-T/2, T/2]$ por $fpar(t) = f(-t)$ si $-L \le t \le 0$, $= f(t)$ si $0 \le t \le L$, y extendida periódicamente a toda la recta real *R*. Obviamente, *fpar* es una función par,así que su desarrollo en serie de Fourier es una serie de cosenos que se conoce, en este contexto, como el desarrollo de *f en serie de cosenos* en medio intervalo [0, *L*] y viene dado por

$$f(t) \approx \frac{1}{2}a_0 + \sum_{n=1}^{\infty}a_n.\cos(nwt)] \text{ en } [0, L]$$

siendo $\omega = \pi/L$ y

$$a_n = \frac{2}{L}\int_0^L f(t)\cos(nwt)dt \quad n = 0.1,2,...$$

La extensión *T -periódica e impar de f* es la función definida en $[-L, L] = [-T/2, T/2]$

Por $fimpar(t) = -f(-t)$ si $-L \le t \le 0$,

$=f(t)$ *si* $0 \le t \le L$,

y extendida periódicamente a toda la recta real *R*. Obviamente, *fimpar* es una función impar, así que su desarrollo en serie de Fourier es una serie de senos que se conoce, en este contexto, como el desarrollo de *f* en *serie de senos* en medio intervalo *[0, L]* y viene dado por

$$f(t) \approx \frac{1}{2}b_0 + \sum_{n=1}^{\infty} b_n.sen(nwt)] \text{ en } [0, L]$$

siendo $\omega = \pi/L$ y

$$b_n = \frac{2}{L}\int_0^L f(t)sen(nwt)dt \quad n = 0,1,2,...$$

Ejercicios: Considera la función $f : [0, 2] \longrightarrow R$ dada por $f(x) = 1 - x/2$.
Un cálculo elemental muestra que los coeficientes del desarrollo en serie de Fourier de senos de *f* vienen dados por $b_n = 2/n\pi$
Diseñar una función de *MATLAB* que dibuje la suma parcial de los *N* primeros sumandos de la serie de Fourier de senos de la función y muestre simultáneamente la gráfica de *f*.
Hacemos uso del archivo m ***fousen***, en términos de *N*, para hallar explícitamente el valor de los coeficientes de Fourier de la función *f*. Usando *MATLAB* podemos aproximar dichos coeficientes mediante las diferentes funciones de integración numérica entre las *quad* y *quadl*
>>*fousen(20)*

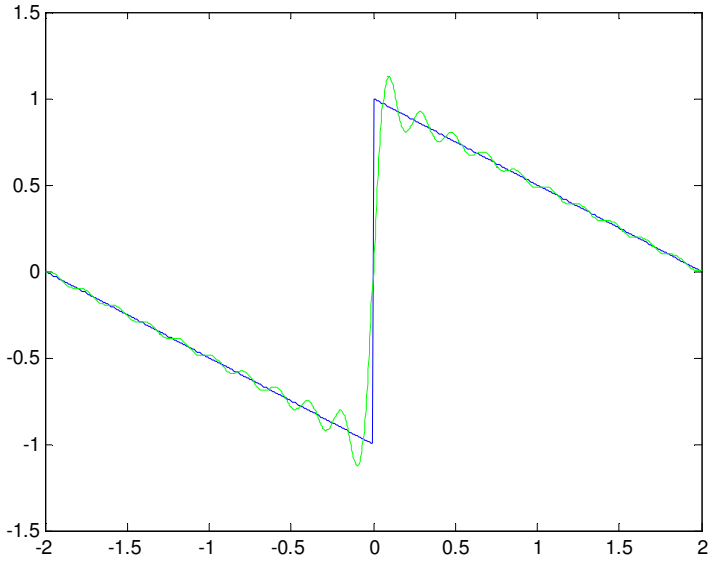


Figura 6.1

Considera la función 2-periódica $f : R \longrightarrow R$ tal que $f(x) = x^2$ si $-1 < x < 1$. Diseñar una función de *MATLAB* que aproxime numéricamente los *N* primeros

coeficientes de la serie de Fourier de la función y que dibuje la suma parcial frente a *f*.

La función es par

$$a_0 = \frac{2}{3}, a_n = \frac{2}{2}\int_{-1}^{1} f(t)\cos(n\pi t)dt = 2\int_{0}^{1} t^2\cos(n\pi t)dt = 1,2,\ldots \quad \text{y}$$

$$b_n = \frac{2}{2}\int_{-1}^{1} f(t)sen(nwt)dt = 0, \quad n = 1,2,\ldots$$

De esta forma, la solución del problema viene dada por la siguiente función en un archivo four.m.

>>*four*(20)

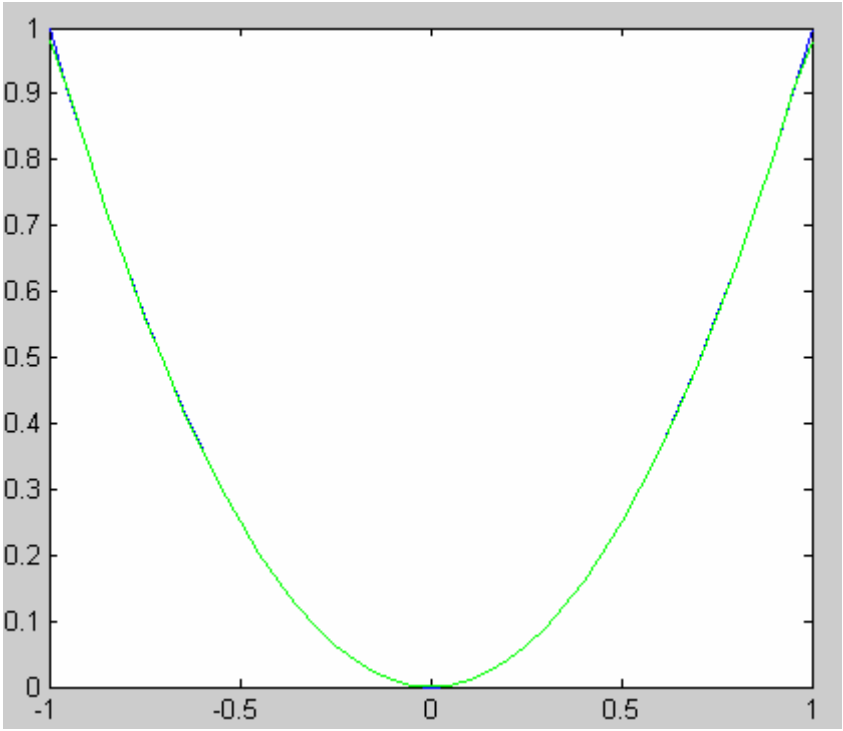


Figura 6.2

6.4 FENÓMENO DE GIBBS

El teorema de Dirichlet nos dice que, en los puntos de discontinuidad, la gráfica de la suma de la serie de Fourier pasa por el punto medio del salto. Si se dibujan las sumas parciales se ve que en las cercanías de los puntos de discontinuidad se reduce la velocidad de convergencia de la serie y que la gráfica de la suma parcial oscila alrededor de la gráfica de la función. Cuando se aumenta el número de términos,

las oscilaciones se condensan a ambos lados del punto pero su amplitud no parece decrecer. Esto se conoce como el fenómeno de Gibbs,en honor de *J.W. Gibbs (1839–1903)* que lo analizó en 1899 probando que la amplitud de la oscilación a cada lado de la gráfica de la función tiende a ser $\frac{1}{2\pi}\int_0^{\pi}\frac{sen(t)}{t}dt - 1 \approx 0.0895$ veces el tamaño del salto, o sea, aproximadamente el 9% del tamaño del salto.
Veremos sobre la función $f : [0, 2] \rightarrow R$ dada por $f(x) = 1 - x/$, *para* $N = 4, 8, 16, 32$ y verifique el fenómeno de *Gibbs* en $x = 0$. Estime gráficamente el valor del salto.

Utilizamos la función *ginput* de *MATLAB*

```
>> fousen(32),A=ginput(2)
salto=A(2,2)-A(1,2)

A =
   -0.0046   -0.9956
   -0.0138   -0.9956

salto =
    0
```

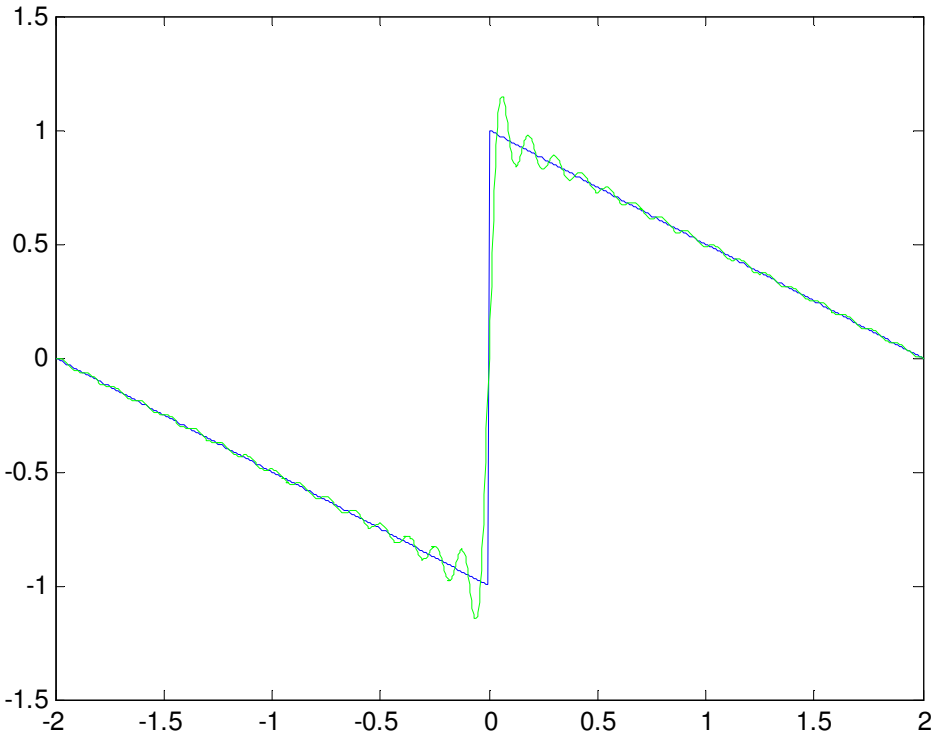


Figura 6.3

Nota: [X, Y] = *GINPUT* (N) *N* recibe puntos de los ejes y devuelve las coordenadas de longitud *N* de los vectores *X* e *Y*. El cursor de se puede colocar con el ratón. Los puntos de datos se especifican presionando un botón del ratón o cualquier tecla del teclado

Problema
Hallar la serie de Fourier para la señal representada en la figura:

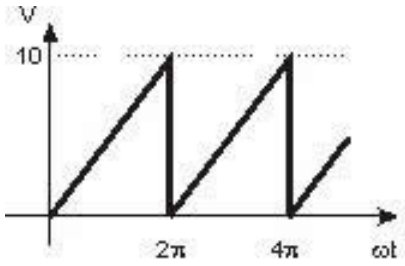


Figura 6.4

La señal de este ejemplo continua para $0 < wt < 2pi$, y su expresión es

$$x(t) = \frac{10}{2\pi} wt$$

Las condiciones de Dirichlet se satisfacen y la serie de Fourier está dada por:

$$x(t) - a_0 + 2\sum_{n=1}^{\infty} a_k.\cos(kwt) + b_k sen(kwt)]$$

$$a_0 = \frac{1}{T}\int_T x(t)dt$$

$$a_k = \frac{2}{T}\int_0^L x(t)\cos(kwt)dt \ , \ b_k = \frac{1}{T}\int_0^L x(t)sen(kwt)dt$$

Multiplicando y dividiendo por w, se puede tomar a wt como variable y el período corresponde a 2π radianes.

Teniendo en cuenta que $wT = 2pi$, se obtienen las siguientes fórmulas

$$a_0 = \frac{1}{2\pi}\int_0^{2\pi} x(t)d(wt)$$

$$a_k = \frac{1}{2\pi}\int_0^{2\pi} x(t)\cos(kwt)d(wt) \ , \ b_k = \frac{1}{2\pi}\int_0^{2\pi} x(t)sen(kwt)d(wt)$$

Entonces la solución será:

$$a_0 = \frac{1}{2\pi}\int_0^{2\pi} x(t)d(wt) = \frac{1}{2\pi}\int_0^{2\pi}\frac{10}{2\pi}(wt)d(wt) = \frac{10}{4\pi^2}\left[\frac{(wt)^2}{2}\right]_0^{2\pi} = 5$$

$$a_k = \frac{1}{2\pi}\int_0^{2\pi} x(t)\cos(kwt)d(wt) = \frac{10}{4\pi^2}\int_0^{2\pi}(wt)\cos(kwt)d(wt)$$

Se resuelve por partes, haciendo:

$u=wt$ con $du=d(wt)$

$dv=\cos(kwt)d(wt)$, de donde $v=sen(kwt)/k$

$$a_k = \frac{10}{4\pi^2}\left[wt.\frac{sen(kwt)}{k} - \int_0^{2\pi}\frac{sen(kwt)}{k}d(wt)\right]_0^{2\pi} = 0$$

Para la integral de b_k

$$b_k = \frac{1}{2\pi}\int_0^{2\pi} x(t)sen(kwt)d(wt)$$

$u=wt$, $du=d(wt)$

$dv=sen(kwt)d(wt)$, donde $v=-\cos(kwt)/k$

$$b_k = \frac{10}{4\pi^2}\left[-wt.\frac{\cos(kwt)}{k} + \int_0^{2\pi}\frac{\cos(kwt)}{k}d(wt)\right]_0^{2\pi} =$$

$$b_k - \frac{10}{4\pi^2}\left[wt.\frac{\cos(kwt)}{k} - \frac{sen(kwt)}{k^2}\right]_0^{2\pi} =$$

$$b_k = \frac{-5}{k\pi}\cos(k2\pi) = \begin{Bmatrix} b_1 = 5/\pi \\ b_2 = -5/2\pi \\ b_3 = -5/3\pi \end{Bmatrix}$$

La serie de Fourier será:

$$x(t) = 5 + 2\sum_{k=1}^{\infty}\frac{-5}{k\pi}sen(kwt) = 5 - \frac{10}{\pi}\sum_{k=1}^{\infty} sen(kwt)/k$$

$$x(t) = 5 - \frac{10}{\pi}sen(wt)\frac{10}{2\pi}sen(2wt)\frac{10}{3\pi}sen(3wt) - ...$$

$$x(t) = 5 - 3.18309sen(wt) - 1.59154sen(2wt) - 1.06103sen(3wt) - ...$$

6.5 GRAFICACIÓN SERIES CON MATLAB

Utilizando el análisis de Fourier podemos descomponer una señal en sus componentes armónicos, por ejemplo si tenemos una señal cuadrada la cual podemos escribir como

$$f(t) = \frac{4v}{\pi}\sum_{n=1}^{\infty}\frac{1}{n}sennw_0t = \frac{4v}{\pi}(sen\omega_0t + \frac{1}{3}sen3\omega_0t + \frac{1}{5}sen5\omega_0t + ...)$$

Vamos a graficar los armónicos de esta función generando uno por uno y posteriormente haremos un programa para obtenerlos

```
>>t = 0:.1:10;
><y = sin(t);
>>plot(t,y),
```

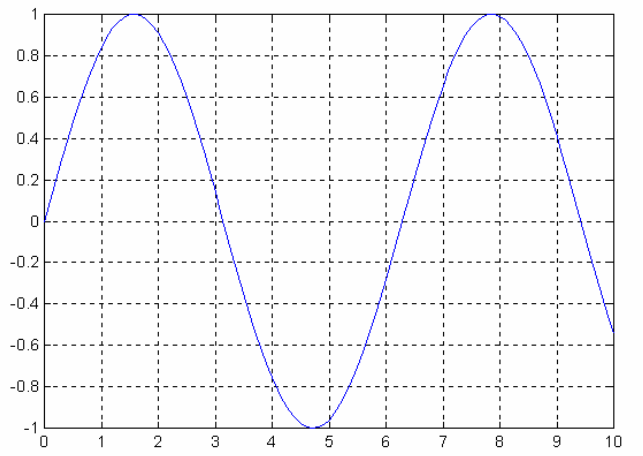


Figura 6.5

Primer armónico

```
>>y = sin(t) + sin(3*t)/3;
>>plot(t,y)
```

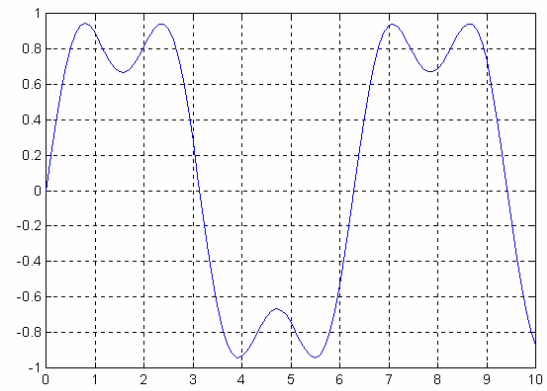


Figura 6.6

Adición de dos armónicos

```
>>y = sin(t) + sin(3*t)/3 + sin(5*t)/5 + sin(7*t)/7 + sin(9*t)/9;
>>plot(t,y)
```

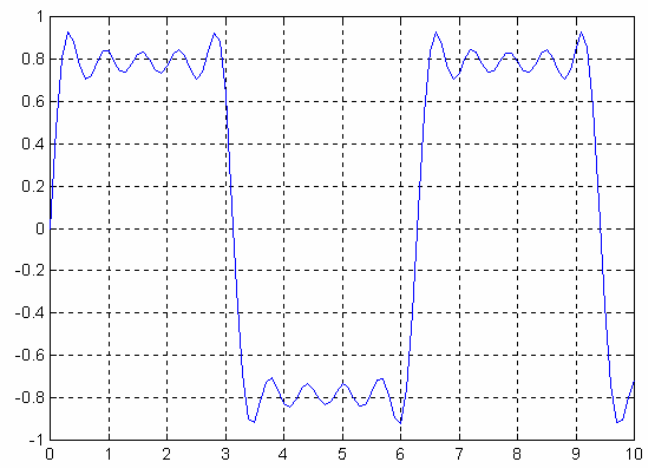


Figura 6.7

Adición de tres armónicos

```
t = 0:.02:3.14;
y = zeros(10,max(size(t)));          %abriendo espacio para almacenar los
                                     valores de los armónicos
x = zeros(size(t));

% programa para generar los armónicos de una señal cuadrada (armonicos.m)
%Programa para obtener los armónicos de una señal cuadrada
%con un periodo de T = 1 seg
Wo = 2*pi/T;
t = 0:0.001:pi;
y = square(2*pi*t/T);
plot(t,y,'r-')
grid
hold on
x = 0;
for k = 1:2:10
      x = x + (4*sin(k*Wo*t))/(pi*k);
   y((k+1)/2,:) = x;
   plot(t,x), pause
end
title('componentes armónicos')
xlabel('tiempo'), ylabel('amplitud')

>>T=1;

>>armonicos
```

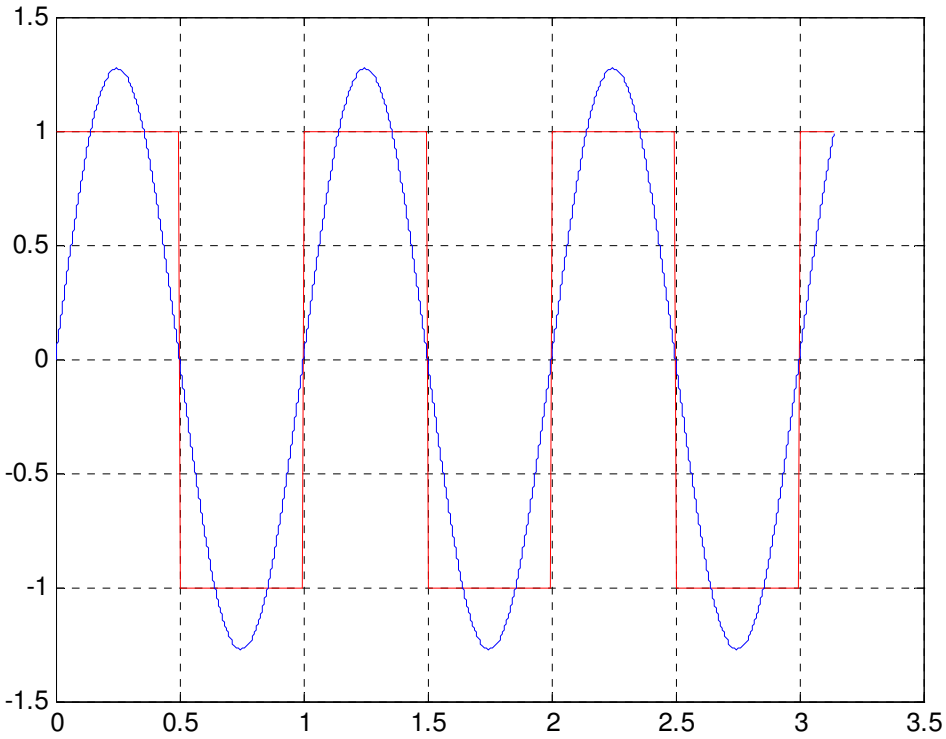


Figura 6.8

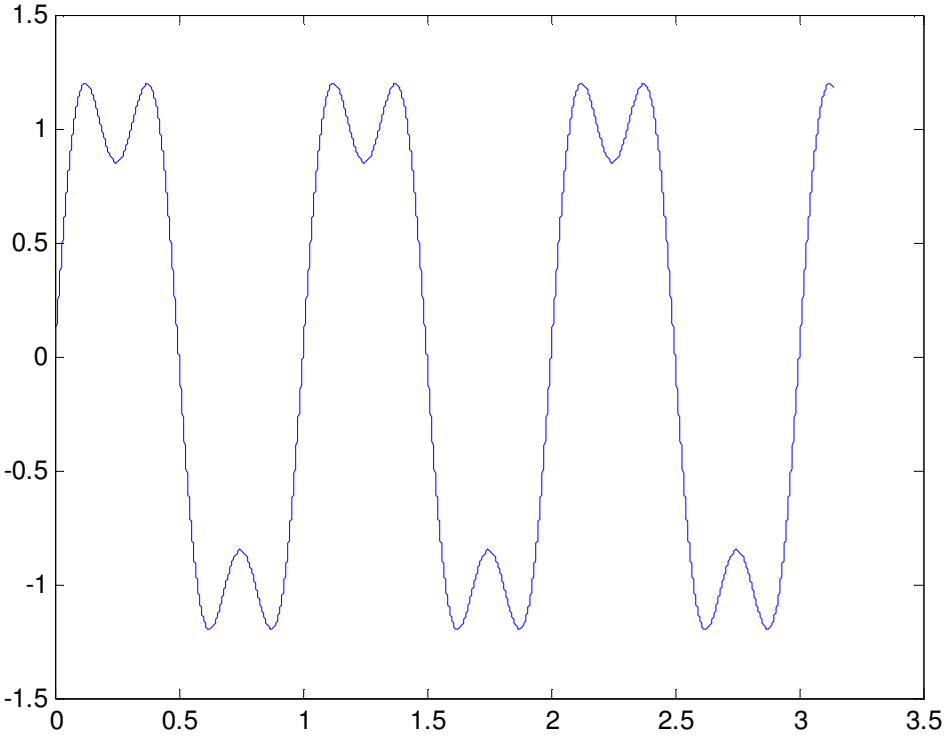


Figura 6.9

Ahora usaremos el archivo *fourier_comp* para obtener los coeficientes de la serie de Fourier., con forma general

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{inf} a_n * sin(n*pi*x/L) + b_n * consn*pi*x/L)$$

Se debe ingresar la función handle, ej $f(x)=x*\text{sen}(x)$, como primer argumento, el número de términos como segundo y el intervalo como tercero:

```
>> f = @(x)x.*sin(x);
>> coeff = fourier_comp(f,3,pi/2)
coeff =

    a0: 1.2732
    an: [0 7.0679e-017 0]
    bn: [-0.7074 0.0962 -0.0385]
```

Con los archivos fourier, graficador y movi (m) se pueden graficar también las series conociendo los coeficientes

CAPÍTULO 7 - LA TRANSFORMADA

7.1 INTRODUCCION

En esta sección veremos algunos formas de *MATLAB* para el manejo de transformadas y su aplicación, no nos excedemos en el desarrollo minucioso de conceptos y definiciones (del curso de análisis), lo que no impide una breve recordatorio, como así también nos reducimos principalmente a Laplace y Fourier.
Una **transformada integral** es cualquier transformada *T*, en definitiva un funcional *T(f)* .aplicada sobre la función *f(x)* de la manera

$$T(f(t)) = \int_{t_1}^{t_2} K(v,t) f(t) dt = F(v)$$

Entre las transformadas se mecionan a las de Fourier, discreta de Fourier, rápida de Fourier, de Hilbert, Laplace, Zeta, Wavelet, polinómica, etc.

Para la entrada de T se tiene una función f(t), y la salida otra función F(v), representando un particular operador matemático, el rango de variación de t está entre -∞ y +∞.

Cada transformada depende de la función K, llamada la función ***núcleo*** *de la transformación.*

Una propiedad inmediata de estas transformadas es que son lineales, es decir, cualesquiera sean los números *a, β* y y las funciones *f(x)* y *g(x)* de R [*a;b*],

$T(a f + \beta g) = a T(f) + \beta T(g)$

Algunos núcleos tienen una *K* inversa asociada, $K^{-1}(v,t)$, que (más o menos) da una transformada inversa:

$$f(t) = \int_{v_1}^{v_2} K^{-1}(v,t)((f(t))dv$$

Un núcleo ***simétrico*** es el que es inalterado cuando las dos variables son permutadas.

7.2 POR QUE LAS TRANSFORMADAS?

Hay muchas clases de los problemas que son difíciles o engorrosos para solucionar en sus presentaciones originales.

Una *transformada integral* "*mapea*" una ecuación de su dominio en otro dominio adecuado (por ejemplo, una función seno en “el dominio del tiempo" puede ser representada como un vector representando una onda "en el dominio de la frecuencia"). La manipulación y la solución de la ecuación en el dominio objetivo son, siempre y cuando la selección es adecuada, mucho más fáciles que la

manipulación y la solución en el dominio original. La solución entonces es *mapeada* al dominio original con la transformada inversa.

La *transformada integral* funciona porque están basadas sobre el concepto de la "*factorización espectral*" sobre bases ortonormales, permitiendo la representación de funciones complicadas como sumas de funciones sencillas. Recordemos que la ortogonalidad indica que el producto de dos funciones de la base distintas integrada sobre su dominio debe ser el cero y la transformada integral modifica la representación de una base ortogonal a otra, entonces, en el dominio objetivo, cada punto de la función transformada es el aporte de una función de base ortogonal dada a la *expansión* de la función de origen en la suma de funciones ortonormales.

Una aplicación de las transformadas pasa por llevar una EDO a una ecuación algebraica en general sencilla de resolver y luego invertir el proceso de forma que obtengamos la solución pretendida, hurgaremos el proceso con una particular transformada integral

7.3 TRANSFORMADA INTEGRAL DE LAPLACE

Sea f una función definida en [0;∞). La transformada de Laplace *L [f]* o

F(t) es la transformada integral:

$F(t) = L(f(t)) = \int_0^{\infty} e^{-tx} f(x)dx$, recordando que

$$\int_0^{\infty} e^{-tx} f(x)dx = \lim_{x\to\infty} \int_0^{x} e^{-tx} f(x)dx$$

Esta integral a distinción de la transformada anterior, puede no existir para ciertas funciones, debido a la no existencia del límite

Una pregunta natural es por tanto que clase de funciones tienen transformadas de Laplace, nos ocuparemos principalmente de las funciones de orden exponencial.
Lema: Si *f* es una función continua a trozos de orden exponencial entonces *f* tiene transformada de Laplace para todo *t* suficientemente grande.
La siguiente tabla de Transformadas de Laplace es de gran inter_es. En ella est_an las principales transformadas de Laplace que usaremos incluida la de la función escalón *χK(s)* definida como 1 si *x* ε [0; c] y 0 en el resto.
Haciendo F= $\mathcal{F}$ y *L*=$\mathcal{L}$

1. $\mathfrak{L}[f'](t) = t\mathfrak{L}[f](t) - f(0) = t\mathcal{F}(t) - f(0)$,

2. $\mathfrak{L}[f^{(n)}](t) = t^n\mathcal{F}(t) - \sum_{k=0}^{n-1} t^k f^{(n-k-1)}(0) = t^n\mathcal{F}(t) - t^{n-1}f(0) - t^{n-2}f'(0) - \cdots - f^{(n-1)}(0)$,

3. $\mathfrak{L}[e^{ax}f(x)](t) = \mathcal{F}(t-a)$,

4. $\mathfrak{L}[xf(x)](t) = -\mathcal{F}'(t)$,

5. $\mathfrak{L}[x^n f(x)](t) = (-1)^n\mathcal{F}^{(n)}(t)$,

6. $\mathfrak{L}[\chi_c(x)f(x-c)](t) = e^{-ct}\mathcal{F}(t)$,

7. $\mathfrak{L}[\chi_c(x)f(x)](t) = e^{-ct}\mathfrak{L}[f(x+c)](t)$.

No nos extenderemos en la demostración de estos resultados
Transformadas de Laplace de ciertas funciones

$f(x)$	$\mathcal{F}(t) = \mathfrak{L}[f](t)$
c	$\frac{c}{t}$
x^n	$\frac{n!}{t^{n+1}}$
e^{ax}	$\frac{1}{t-a}, \quad t > a$
$\operatorname{sen}(ax)$	$\frac{a}{t^2+a^2}$
$\cos(ax)$	$\frac{t}{t^2+a^2}$
$\operatorname{senh}(ax)$	$\frac{a}{t^2-a^2}, \quad t > a$
$\cosh(ax)$	$\frac{t}{t^2-a^2}, \quad t > a$
$x^{-1/2}$	$\sqrt{\frac{\pi}{t}}, \quad t > 0$
$\chi_c(x)$	$\frac{e^{-ct}}{t}, \quad t > 0$

Habíamos mencionado la transformada inversa: dada la función $\mathcal{F}(t)$, $f(x)$ es la antitransformada de $\mathcal{F}(t)$ si $\mathcal{F}(t) = \mathcal{L}(f(t))$, escribiendo $f(x) = \mathcal{L}^{-1}(\mathcal{F}(t))$

Concurrimos ahora a aplicar a lo mencionado, como la resolución de un *PVI*

$y'' + ay' + by = f(x)$; $y(0) = y_0$; $y'(0) = y_{00}$

Usando que $\mathcal{L}(y') = t\mathcal{L}(y) - y(0)$, que $\mathcal{L}(y'') = t^2\mathcal{L}(y) - ty(0) - y'(0)$, como $\mathcal{F} = \mathcal{L}(f)$

la *EDO* se transforma en la siguiente ecuación algebraica

$t^2\mathcal{L}(y) - ty(0) - y'(0) + a\,t\mathcal{L}(y) - ay(0) + b\mathcal{L}(y) = \mathcal{F}(t)$

la transformada de la solución será del tipo

$\mathcal{L}(y)=(\mathcal{F}(t)+(t+a)y(0)+y'(0))/(t^2+at+b)$ (1)

Yendo a un caso específico de PVI

$y'' + 3y' + 2y = 0$ con $y(0) = y_0$ e $y'(0) = y'_0$

De (1)

$\mathcal{L}(y)=(t-3)y(0)+y'(0))/(t^2+3t+2)$

Llevando a fracciones simples

$\mathcal{L}(y)=(A/t+1)+(B/t+2)$
Pero $\mathcal{L}(e^x)=1/(t+1)$ y $\mathcal{L}(e^{2x})=1/(t+2)$

Tendremos

$\mathcal{L}(y)= \mathcal{L}(A\ e^x+B\ e^{2x})\rightarrow y(x)=Ae^x+B\ e^{2x}$
Vale citar que la transformada de Laplace no sólo está bien definida para muchas funciones continuas, sino que también lo está para funciones continuas a trozos lo cual nos permite resolver directamente ciertos *PVI* cuando la función *f* sea continua a trozos.
Definición: Dadas dos funciones integrables *f* y g, definimos la convolución (*f**g) de ambas a la función

$$(f * g)(x) = \int_0^x f(x-z)g(z)dz$$

Si además, *f* y g son de orden exponencial siendo $\mathcal{F}$ (t) y G (*t*) las transformadas de Laplace de *f* y g, respectivamente, entonces existe la transformada de Laplace de (*f* * *g*)(*x*) :

$\mathcal{L}(f * g)= \mathcal{F}(t)\ G(t)$
Calcular $\mathcal{L}^{-1}[1/(1+t^2 2)^2]$
Como $\mathcal{L}(senx)= \mathcal{F}(t)=1/(1+t^2)$
$1/(1+t^2 2)^2=1/(1+t^2).\ 1/(1+t^2)= \mathcal{F}(t).\ \mathcal{F}(t)$

$\mathcal{L}^{-1}[1/(1+t^2 2)^2]=sen(x).sen(x)= \int_0^x sen(x-z)g(z)dz =(-1/2)x\cos x +(1/2)sen\ x$

7.4 USO DE MATLAB

De acuerdo a lo visto hasta ahora.la transformada de Laplace de una función *F(t)* definida en [0;+∞) es una nueva función que denotaremos por *f(s)* o *L(F(t))(s)*, definida por la integral

$$f(s) = \int_0^\infty e^{-st}F(t)dt.$$

El dominio de *f(s) = L(F)(s)* está formado por los valores de *s* para los cuales la integral existe.

La integral viene dada en términos de una integral impropia de primera especie (intervalo no acotado).
Recordar que si G es integrable en cada *[0; b],* entonces

$$\int_0^{\infty} G(t)dt = \lim_{b\to\infty} \int_0^{b} G(t)dt$$

Siempre que exista el límite

Para el cálculo *MATLAB* ofrece el comando *laplace* de tipo simbólico, cuya sintaxis es
>>*f=laplace(F)*
donde *F* es una función escalar de la variable simbólica *t*, previamente declarada, y *f* es una función cuya variable es por defecto *s*.
El siguiente ejemplo muestra que se pueden elegir las variables libremente.
>> *syms u v*
>>*f=laplace(u^2,v)*
f = 2/v^3
-Calcular la transformada de Laplace de la función *F(t) = 1*.
>>*syms t s*
>>*laplace(1,t,s)*
ans =1/s

Calcular la transformada de Laplace de la función $F(t) = e^{-at}$.

>>*syms a;*
>>*laplace(exp(-a*t),t,s)*
ans =1/(s+a)

La función objeto *F(t)* se puede crear como una cadena de caracteres (string)

>>*syms* t s
>>*F='exp(-a*t)'*
>>*laplace(F,t,s)*
ans =1/(s+a)
Con lo cual no tenemos que declarar previamente como simbólico a ningún parámetro que aparezca en la expresión que define a la función, como ocurre en este caso con *a*.
Invocando al núcleo Maple podemos trabajar exclusivamente con strings.
>>*maple('f:=t->exp(-a*t)*sin(b*t)')*
ans =
*f := t -> exp(-a*t)*sin(b*t)*

>>*f=maple('laplace(f(x),x,s)')*
f =
b/((s+a)^2+b^2)
El comando *ilaplace* ha sido diseñado para calcular exactamente, cuando ello sea posible, la transformada inversa de Laplace. La sintaxis es la siguiente
>>*ilaplace(expresion_simbolica,s,t) o*

>>*ilaplace(expresion_simbolica)*
Observar que el comando *ilaplace* calcula la transformada inversa de Laplace de una expresión *f(s)*, produciendo otra expresión *F(t)*.
La versión Maple es como sigue
>>*F=maple('invlaplace(f(s),s,t)')*

Calcular la transformada inversa de $f(s) = s + 3/s^2$

```
>> syms s;
>> f=(s+3)/s^2;
>> F=ilaplace(f)

F =
  1+3*t
```

Con maple

```
>> maple('invlaplace((s+3)/s^2,s,t)')
ans =
  1+3*t
```

7.4.1Laplace y los sistemas de control

En años recientes, los sistemas de control han asumido un papel cada vez más importante en el desarrollo y avance de la civilización moderna y la tecnología.
Los sistemas de control se encuentran en gran cantidad en todos los sectores de la industria: tales como control de calidad de los productos manufacturados, líneas de ensamble automático, control de máquinas-herramienta, tecnología espacial y sistemas de armas, control por computadora, sistemas de transporte, sistemas de potencia, robótica y muchos otros.
por qué es necesario controlar un proceso :incremento de la productividad, alto costo de mano de obra, seguridad, alto costo de materiales, mejorar la calidad, reducción de tiempo de manufactura, reducción de inventario en proceso, certificación (mercados internacionales), protección del medio ambiente (desarrollo sustentable)
En el estudio de los procesos es necesario considerar **modelos dinámicos**, es decir, modelos de comportamiento variable respecto al tiempo. Esto trae como consecuencia el uso de **ecuaciones diferenciales respecto al tiempo** para representar matemáticamente el comportamiento de un proceso.
El comportamiento dinámico de los procesos en la naturaleza puede representarse de manera aproximada por el siguiente **modelo general de comportamiento dinámico lineal**:

$$a_n \frac{d^n y(t)}{dt^n} + a_n - 1\frac{d^{n-1} y(t)}{dt^{n-1}} + a_{n-2}\frac{d^{n-2} y(t)}{dt^{n-2}} + \ldots\ldots + a_0 y(t) = x(t)$$

Cómo mencionamos al hablar de transformadas integrales, la transformada de Laplace permite resolver ecuaciones diferenciales lineales mediante la transformación en ecuaciones algebraicas con lo cual se facilita su estudio.

Una vez que se ha estudiado el comportamiento de los sistemas dinámicos, se puede proceder a diseñar y analizar los sistemas de control de manera simple.
Para poder diseñar un sistema de control automático, se requiere

- Conocer el proceso que se desea controlar, es decir, conocer la ecuación diferencial que describe su comportamiento, utilizando las leyes físicas, químicas y/o eléctricas.
- A esta ecuación diferencial se le llama modelo del proceso.
- Una vez que se tiene el modelo, se puede diseñar el controlador.

7.4.2 Propiedades y teoremas de la transformada de Laplace más utilizados en al ámbito de control

- TEOREMA DE TRASLACIÓN DE UNA FUNCIÓN

  (Nos indica cuando el proceso tiene un retraso en el tiempo)

$$L\{f(t-\alpha)1(t-\alpha)\} = e^{-\alpha s F(s)} \quad \text{para } \alpha \geq 0$$

- TEOREMA DE DIFERENCIACIÓN REAL

  (Es uno de los más utilizados para transformar las ecuaciones diferenciales)

$$L\left\{\frac{d^n}{dt^n} f(t)\right\} = s^n F(s) - s^{n-1} f(0) - s^{n-2} f'(0) - \ldots - s f^{n-2}(0) - f^{n-1}(0)$$

- TEOREMA DE VALOR FINAL

  (Nos indica el valor en el cual se estabilizará la respuesta)

$$\lim_{t\to\infty} f(t) = \lim_{s\to 0} sF(s)$$

- TEOREMA DE VALOR INICIAL

  (Nos indica las condiciones iniciales)

$$\lim_{t\to 0} f(t) = \lim_{s\to\infty} sF(s)$$

Tomaremos un modelo sencillo: **control de nivel de un tanque**

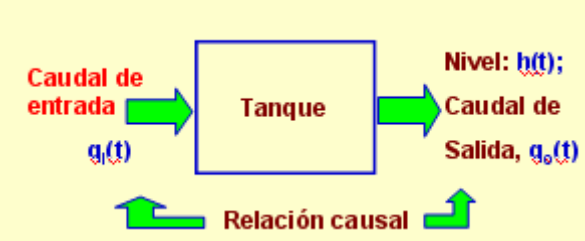


Figura 7.1

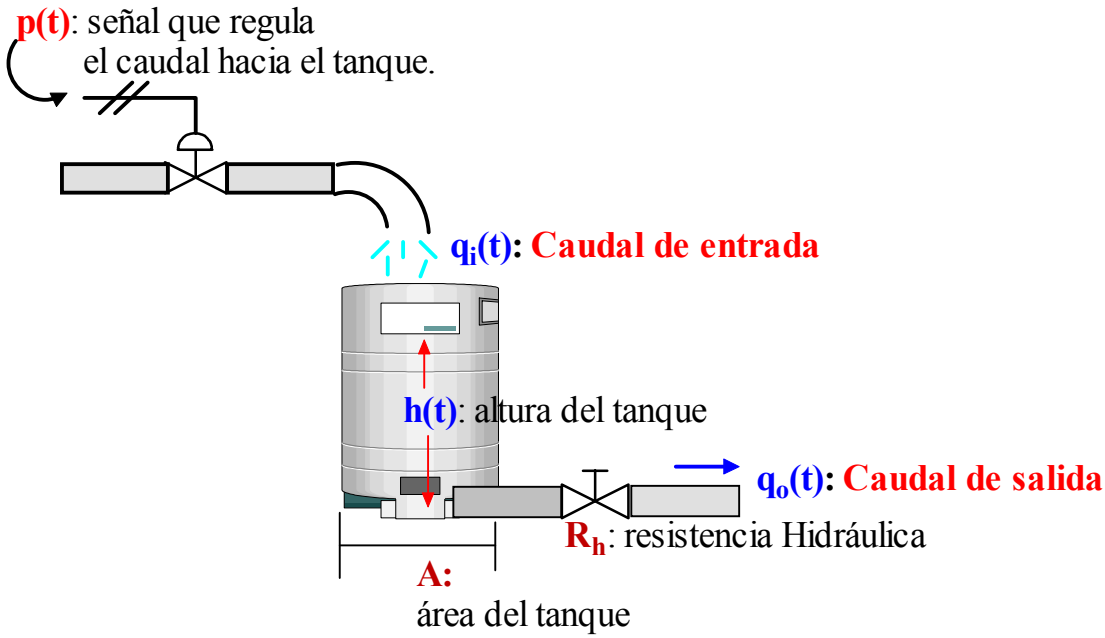


Figura 7.2

Haciendo un balance.
Caudal de entrada – caudal de salida=acumulación

$$q_i(t) - q_0(t) = q_{acum}(t) = Av(t) = A\frac{dh(t)}{dt} \quad \ldots\ldots (1) \qquad q_0(t) = \frac{h(t)}{Rh} \quad \ldots\ldots (2)$$

$$Q_i(s) - Q_o(s) = \boldsymbol{A}\ s\ H(s),\ (\boldsymbol{c.\ i.} = 0); \qquad Q_0(s) = \frac{H(s)}{\boldsymbol{Rh}}$$

Función de transferencia

- Representa el comportamiento dinámico del proceso
- Nos indica como cambia la salida de un proceso ante un cambio en la entrada

Diagrama de bloques

Usando bloques

$$\frac{Y(s)}{X(s)} = \frac{\text{Cambio en la salida del proceso}}{\text{Cambio en la entrada del proceso}}$$

$$\frac{Y(s)}{X(s)} = \frac{\text{Respuesta del proceso}}{\text{Función forzante}}$$

Figura 7.3

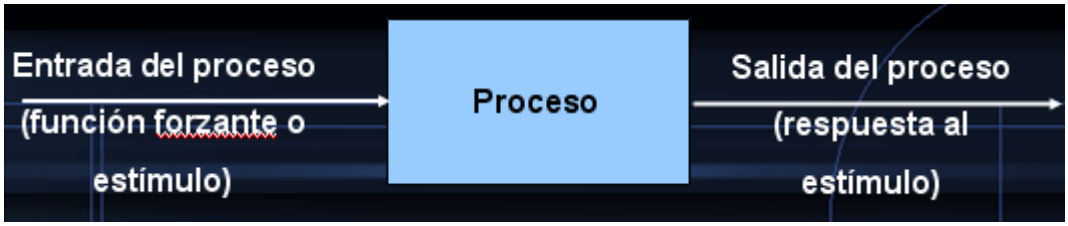


Figura 7.4

Esta función forzante puede ser una función escalón(valor cte), una senoidal o una función producto exponencial por senoidal, por ejemplo
Entonces en general para un modelo de primer orden:

$\tau \frac{dc}{dt} + c(t) = K.u(t)$ donde τ =cte. de tiempo y K=ganancia en estado estable

En el caso del tanque

$$\tau \frac{dq_0(t)}{dt} + q_0(t) = q_i(t)$$

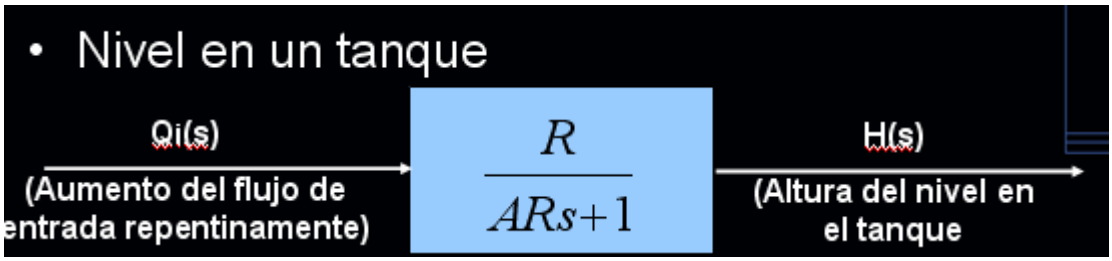


Figura 7.5

Llevando a nuestro ejemplo, el diagrama sería

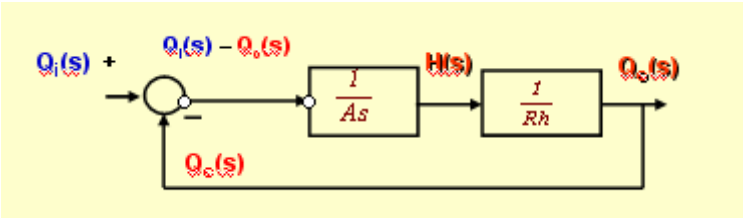


Figura 7.6

Haciendo su correlato con la transformada
$\mathcal{L}(f(t)) \rightarrow \mathcal{F}(s)$, a través de la integral
Recurrimos a Simulink para su resolución, tomando como datos área del tanque
A=2 y H/Q_0=Rh=3

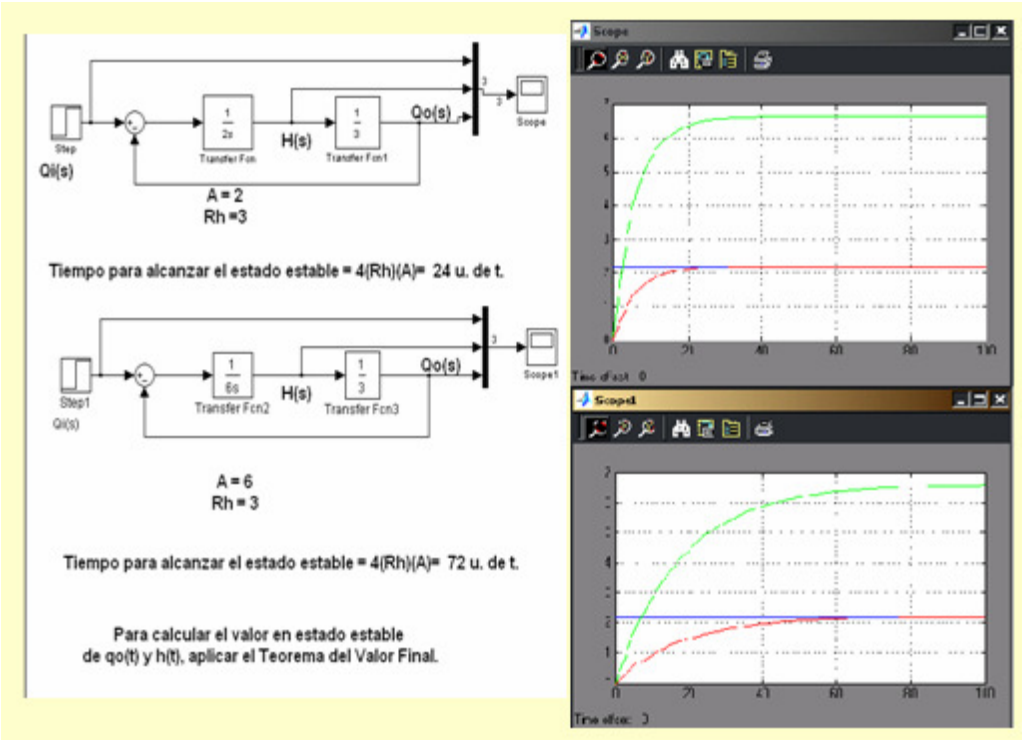


Figura 7.7

7.5 TRANSFORMADA DE FOURIER

Sea x(t) una función del espacio $L^1(-\infty,+\infty)$, función *R* o C sobre la recta real que satisface $\int_{-\infty}^{+\infty}|x(t)|dt<\infty$ (i)

Restringiéndola tal que en $(-T/2,T/2)$, $x(t)=x(t)$ para todo *t* del intervalo y $x(t)=0$ en cualquier otra situación ; al desarrollar $x_T(t)$ en términos de las exponenciales complejas

$$\{\phi_k(t)\}_{k\varepsilon Z}=\left\{e^{j2\pi\frac{k}{T}t}\right\}_{k\varepsilon Z}$$

Que forman una base del espacio hilbertiano $L^2(-T/2,T/2)$, se tiene

$$x_T(t)=\left\{\sum_{k=-\infty}^{+\infty}\gamma[k]e^{j2\pi\frac{k}{T}t}\right\} \quad \text{(ii)}$$

Con $\gamma[k]$ es el correspondiente coeficiente de Fourier

$$\gamma[k]=\frac{\left\langle x(t),e^{j2\pi\frac{k}{T}t}\right\rangle}{(e^{j2\pi\frac{k}{T}t})}=\frac{1}{T}\int_{-T/2}^{T/2}x(t)e^{j2\pi\frac{k}{T}t} \quad \text{(iii)}$$

Para cada *k* de Z, el pulso ω_k de la exponencial compleja ϕ_k es $\omega_k=2\pi f_k=2\pi k/T$

Entonces la frecuencia asociada $f_k=k/T=k/f_1$
Donde f_1 (inverso del periodo fundamental T) se conoce como frecuencia fundamental o del primer armónico,f_k es la frecuencia del k-simo armónico

La diferencia entre dos armónicos consecutivos $\Delta f = f_{k+1} - f_k = \frac{k+1}{T} - \frac{k}{T} = \frac{1}{T}$

Si $T\to\infty, \Delta f \to 0$, por lo tanto para no trabajar con f_k nulas,se supone que $k\to\infty$ a la vez, así: $k\Delta f = \frac{k}{T} \to f$ con f una cantidad finita que representa una frecuencia genérica(f_k sólo toma múltiplos de Δf ,f puede tomar cualquier valor)
También el módulo de los coeficientes de (iii) tiende a cero pues:

$\gamma[k] \le \frac{1}{T}\int_{-T/2}^{T/2} \|x(t)\| \left| e^{j2\pi\frac{k}{T}t} \right| dt = \frac{1}{T}\int_{-\infty}^{\infty} |x(t)| dt$ convergiendo la integral a un valor finito

Por todo ello se define coeficiente por unidad de frecuencia(función discreta)

$$Xf[k] = \gamma[k]T = \frac{\gamma[k]1}{\Delta f} = \int_{-T/2}^{T/2} x(t)e^{j2\pi f_k t}dt \quad \text{(iv)}$$

que para $T,k\to\infty$, en el límite

$$Xf = \lim_{T,k\to\infty} X[f_k] = \int_{-\infty}^{\infty} x(t)e^{j2\pi f t}dt \quad \text{(v)}$$

Es la llamada transformada de Fourier de *x(t)*, X=*Fr(x), representando X(f)* una función de *densidad de coeficiente de Fourier*
De la serie a la transformada de Fourier
De (iv), la serie de Fourier (ii) de $x_T(t)$ se puede escribir como :

$$x_T(t) = \sum_{k=-\infty}^{\infty} \Delta f X[f_k] e^{j2\pi f_k t}dt \qquad \text{(vi)}$$

para $T,k\to\infty$,resulta que $X[f_k]\to X(f_k)$ y la serie anterior puede escribirse como

$$\sum_{k=-\infty}^{\infty} \Delta f X(f_k)e^{j2\pi f_k t}dt = \sum_{k=-\infty}^{\infty} \Delta f \psi(k\Delta f)$$

Donde $\psi(f) = X(f)e^{j2\pi fkt}$
En el límite cunado $\Delta f \to 0$, esta serie representa la integral de Riemann de la función $\psi(f)$ y se obtiene la fórmula que nos permite recuperar *x(t)* a partir de *X(f)*:

$$x(t) = \int_{-\infty}^{\infty} X(f)e^{j2\pi f t}df \quad \text{(vii)}$$

que representa la *transformada inversa* de Fourier,$x==Fr^{-1}X$

Tabla de Transformadas de Fourier

FUNCIÓN	TRANSFORMADA	PARÁMETROS
$f(x)$	$\int_{-\infty}^{\infty} f(x)e^{-itx}\,dx$	
$e^{-\lvert x\rvert}$	$\frac{2}{1+t^2}$	
$\frac{1}{1+x^2}$	$\pi e^{-\lvert t\rvert}$	
e^{-ax^2}	$\sqrt{\frac{\pi}{a}}\, e^{-t^2/4a}$	$a>0$
$e^{iax}f(x)$	$\hat{f}(t-a)$	$a\in\mathbb{R}$
$f(x+a)$	$e^{iat}\hat{f}(t)$	$a\in\mathbb{R}$
$f(ax+b)$	$\frac{1}{\lvert a\rvert}\, e^{\frac{ibt}{a}}\hat{f}(\frac{t}{a})$	$a\neq 0\ ,\ b\in\mathbb{R}$
$x^n f(x)$	$i^n\hat{f}^{(n)}(t)$	$n\in\mathbb{N}$
$f^{(n)}$	$(it)^n\hat{f}(t)$	$n\in\mathbb{N}$
$f(x)\operatorname{sen}(ax)$	$\frac{\hat{f}(t-a)-\hat{f}(t+a)}{2i}$	$a\in\mathbb{R}$
$f(x)\cos(ax)$	$\frac{\hat{f}(t-a)+\hat{f}(t+a)}{2}$	$a\in\mathbb{R}$
$\begin{cases}1 & ,\ \lvert x\rvert\leqslant 1\\ 0 & ,\ \lvert x\rvert>1\end{cases}$	$2\frac{\operatorname{sen} t}{t}$	

Transformada	**Símbolo**	K	t_1	t_2	K^{-1}	u_1	u_2
Transformada de Fourier	$\mathcal{F}$	$\frac{e^{+iut}}{\sqrt{2\pi}}$	$-\infty$	∞	$\frac{e^{-iut}}{\sqrt{2\pi}}$	$-\infty$	∞
Transformada de Laplace	$\mathcal{L}$	e^{-ut}	0	∞	$\frac{e^{+ut}}{2\pi i}$	$c-i\infty$	$c+i\infty$

7.5.1 Aplicaciones

En la teoría de sistemas lineales es fundamental la representación de una señal en términos de sinusoides o exponenciales complejas. Ello es debido a que una exponencial compleja es una autofunción de cualquier sistema lineal e invariante con el tiempo, mientras que la respuesta a una sinusoide es otra sinusoide de la misma frecuencia, con fase y amplitud determinadas por el sistema. De este modo, la representación en frecuencia de la señales, a través de la Transformada de Fourier, resulta imprescindible para analizar las señales y los sistemas.

La transformada de Fourier de una señal discreta (DTFT) es una señal periódica de período 2π. Así, la ecuación de síntesis de $x[n]$ a partir de su transformada se puede ver como el cálculo de los coeficientes de la serie de Fourier de la señal periódica $X(e^{jw})$, mientras que la ecuación de análisis refleja el desarrollo en serie de la transformada en función de los coeficientes $x[n]$.

A la hora de plantear la DTFT computacionalmente cabe hablar de dos problemas: la transformada de señales infinitas, y el hecho de que la transformada es continua, cuando solo podemos trabajar de forma discreta. Ante el primer problema sólo cabe decir que se podrá evaluar la transformada de señales infinitas cuando esta se pueda representar analticamente. En cuanto a la naturaleza discreta de los cálculos, aunque la transformada es continua sólo podremos obtener muestras de la misma, que pueden constituir una buena aproximación si se toman suficientes.

La función *fft* calcula la transformada de Fourier de una señal finita en el número de puntos equiespaciados especificado en la llamada a la función.

Ejemplo: Veamos como se puede visualizar la transformada de Fourier de una señal discreta, que necesariamente debe ser calculada en un conjunto finito de frecuencias. Así, sea la señal

$h[n] = \delta[n] + 0.5\delta[n-1] + 0.2\delta[n-2]$

La siguiente sentencia nos permite calcular 114 valores de su transformada de Fourier:

```
>> h = [1 0.56 0.2];
>> H=fft(h,114);
```

El vector H recoge los valores de la función $H(e^{jw})$ en las siguientes frecuencias:

$w_k = 2\pi k/114$, $k = 0, \cdots, 113$

Para ver la transformada el vector H contiene valores complejos, por lo que tendremos que representar por separado su magnitud y su fase:

```
>> plot(2*pi*(0:113)/114,abs(H));
```

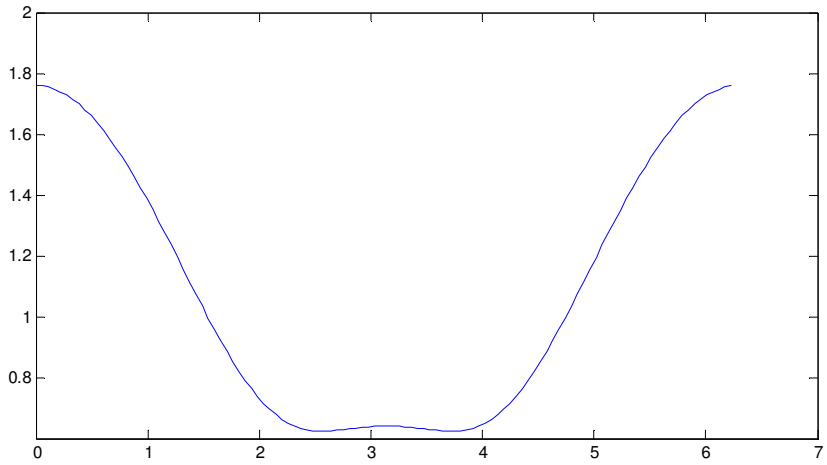


Figura 7.8

>> *plot*(2**pi**(0:113)/114,*angle*(*H*))

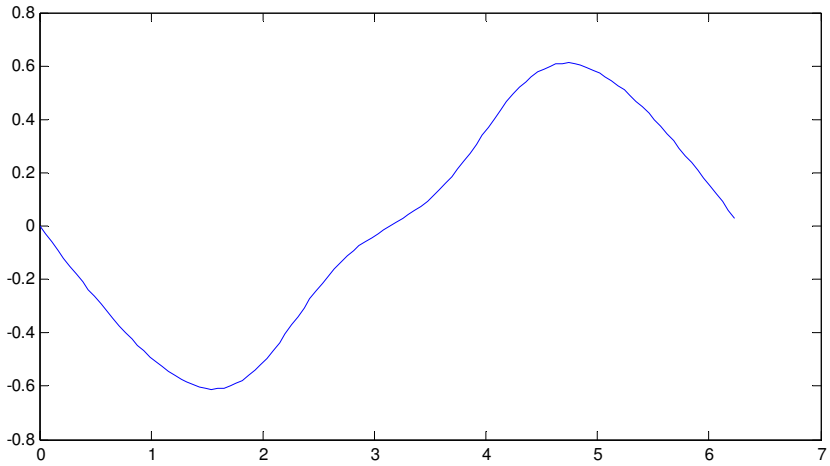


Figura 7.9

En las abcisas se incluyen las frecuencias a las que se evalúa la transformada, En las ordenadas se coloca o bien la magnitud o bien la fase. El comando *plot*() crea una curva continua, la interpolación entre los valores discretos (114) de la transformada que han sido calculados. De esta manera se obtiene una representación de la transformada entre 0 y 2π.

Aplicación a filtros: Los filtros digitales son sistemas *LIT* (sistemas lineales invariantes con el tiempo) descritos por una ecuación en diferencias lineal con coeficientes constantes:

$$\sum_{k=0}^{N} a_k y[n-k] = \sum_{k=0}^{M} b_k x[n-k] \quad \text{(vii)}$$

Utilizando las propiedades de desplazamiento y linealidad de la transformada de Fourier, la respuesta en frecuencia de un sistema *LIT* descrito por una ecuación en diferencias con coeficientes constantes puede expresarse como:

$$H(e^{jw}) = \frac{\sum_{k=0}^{M} b_k e^{-jw}}{\sum_{k=0}^{N} a_k e^{-jw}}$$

Si necesitamos representar una respuesta en frecuencia de un filtro expresado de esta forma, se debe hacer:

>> *H = freqz(b,a,114,"whole"); % a,b* son los coeficientes de la ecuación en diferencias

>> *plot(2*pi*(0:113)/114,abs(H));*

>> *plot(2*pi*(0:113)/114,angle(H));*

El concepto de autofunción de un sistema LIT es la base para comprender su respuesta en frecuencia. Así, para un sistema con respuesta impulsional *h[n]*, la salida ante una exponencial compleja de la forma $e^{jw_0 n}$ será

$$y[n] = \sum_{k=0}^{\infty} h[k] e^{jw_0(n-k)} = H(e^{jw_0}) e^{jw_0 n} \quad \text{(viii)}$$

Un sistema *LIT* tiene como autofunciones el conjunto de exponenciales complejas de la forma $z_0{}^n$, que sólo se ven modificadas a su paso por el sistema por una constante compleja. En el caso de $z_0 = e^{jw_0}$, esa constante es el valor de la respuesta en frecuencia (transformada de Fourier de la respuesta impulsional) a la frecuencia w_0.

Por tanto, la Transformada de Fourier de una señal nos informa de como responde el sistema descrito por esa señal para cada frecuencia de entrada, o lo que es lo mismo, nos aporta el contenido en frecuencia de la señal, ya que con la ecuación de síntesis se puede reconstruir la señal con exponenciales complejas, de la manera siguiente:

$$X(e^{jw}) = \sum_{n=-\infty}^{\infty} x[n] e^{-jwn} \quad \text{(ix)}$$

$$x[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(e^{jw}) e^{jw} dw \quad \text{(x)}$$

Ejemplo: En este ejercicio se va a identificar la respuesta en frecuencia de un sistema (la Transformada de Fourier de su respuesta al impulso) a determinadas frecuencias. Para ello, sólo podemos introducir señales a su entrada y observar lo que ocurre a su salida.

Utilizaremos exponenciales complejas como entradas, dado su carácter de autofunciones. Observando la salida, podremos obtener información sobre como se comporta el sistema para cada frecuencia de interés. Así, considerar el sistema con respuesta impulsional

h[n] = 0.03δ[n] + 0.4 δ [n − 1] +0.54 δ [n − 2] + 0.2 δ [n − 3] −0.2 δ [n − 4]
+0.1 δ [n − 5] + 0.2 δ [n − 6]

Para definir la respuesta impulsional, basta con efectuar en *MATLAB*

>> *h=[0.03 0.4 0.54 0.2 -0.2 0.1 0.2]*

Generar un conjunto de 10 exponenciales complejas con 48 puntos de longitud de la forma $e^{jw_k n}$, para las siguientes frecuencias discretas: $w_k = 2\pi k/10$, k = 0,...,9:

```
>> omegas =(2*pi*(0:9))/10;
>> e1 = exp(j*omegas(1)*(0:47));
>> e2 = exp(j*omegas(2)*(0:47));
>> ...
```

Calcular la salida del filtro para cada una de las exponenciales, convolucionando cada señal de entrada con el filtro definido al comienzo:

```
>> y1 = conv(e1,h);
>> y2 = conv(e2,h);
>>...
```

Superponer en la misma gráfica la entrada e1 y la salida y1:

```
>> plot(e1);
>> hold;
>> plot(y1);
>> hold;
```

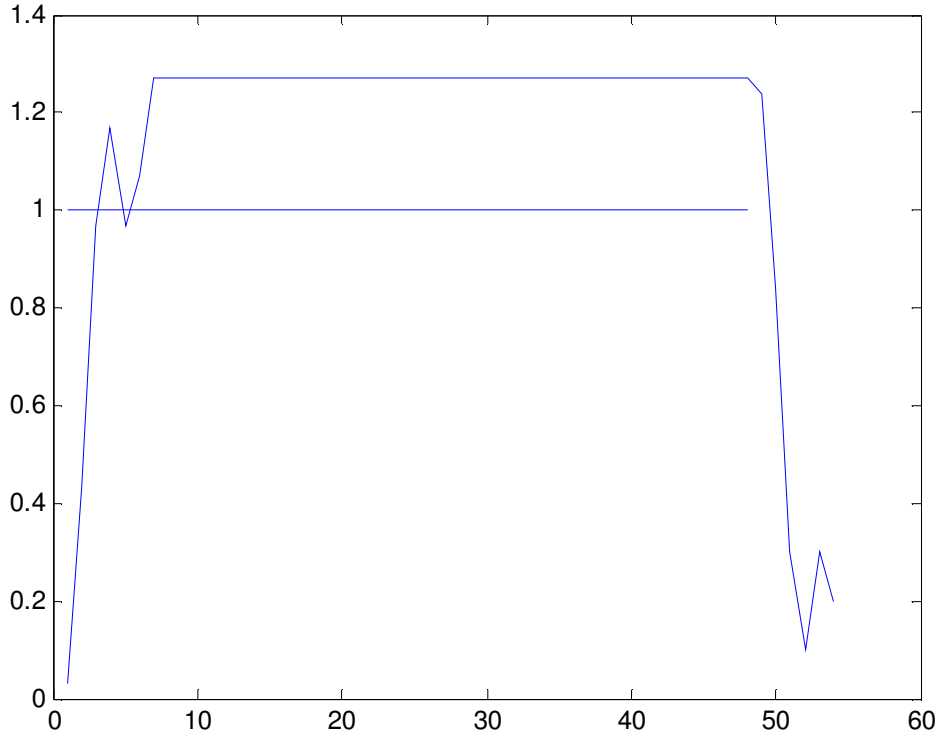


Figura 7.10

En el ejercicio anterior la fase y amplitud de la señal de salida son diferentes a las de la señal de entrada. Esa diferencia viene determinada por la respuesta en frecuencia para *w = omegas(2).*

Con *MATLAB*, se trabaja con la función FFT Transformada discreta de Fourier discreta.

FFT (X) es la transformada discreta de Fourier (DFT) del vector X. Para matrices, la operación de FFT se aplica a cada columna. Para N-D matrices, la operación de FFT opera en el primer no-singleton dimensión FFT (X, N) es la N-FFT, rellena con ceros si X tiene menos de N puntos y truncada si tiene más.

FFT (X, [], DIM) o FFT (X, N, DIM), se aplica la operación a través de la FFT dimensión DIM.

Para un vector de entrada x de longitud, la DFT es un vector de longitud N X, con los elementos

$$X(k) = \sum_{n=1}^{N} x(n) * \exp(-j * 2 * \pi * (k-1) * (n-1) / N),\ 1 <= k <= N.$$

El DFT inversa (calculadas por la IFFT) viene dada por la

$$x(n) = (1 / N) \sum_{k=1}^{N} X(k) * \exp(j * 2 * \pi * (k-1) * (n-1) / N),\ 1 <= n <= N.$$

Un uso común de las transformadas de Fourier es encontrar los componentes de frecuencia de una señal encerrado en una señal de dominio temporal de ruido. Considerar que los datos de una muestra a 1000 Hz. Formar una señal que contiene una sinusoide de 50 Hz de amplitud 0,7 y 120 Hz sinusoidal de amplitud 1 y corruptos con un ruido aleatorio medio no nulo

```
>>Fs = 1000;                    % frecuencia de muestra
>>T = 1/Fs;                     % tiempo de muestra
>>L = 1000;                     % Longitud señal
>>t = (0:L-1)*T;                % vector tiempo
% Suma e una sinusoide de 50 Hz y una sinusoide de 120 Hz
>>x = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t);
>>y = x + 2*randn(size(t));     % ruido sinusoide
>>plot(Fs*t(1:50),y(1:50))
title('Señal corrupta con ruido medio aleatorio no cero')
xlabel('tiempo (milisegundos)')
```

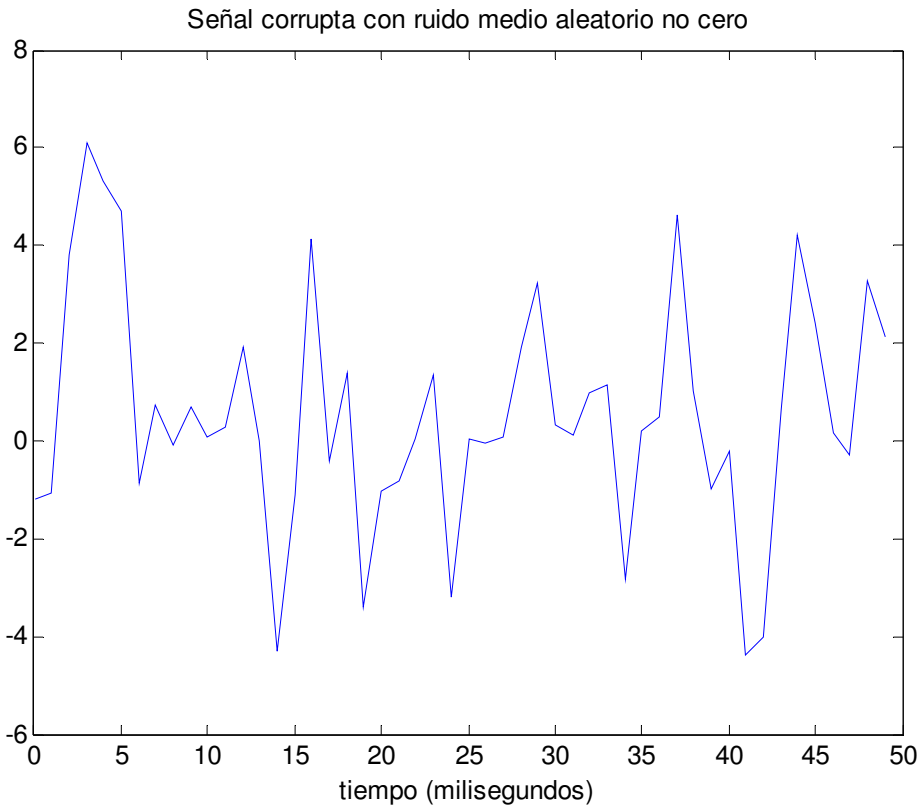


Figura 7.11

El algoritmo *FFT* es una manera eficiente de calcular la *DFT*. En *MATLAB* la función es

X=fft(x,N)

- Calcula la *FFT* de *N* puntos del vector x.
- El resultado *X* es un vector de números complejos ordenados con índice *k*=0,1, ...*N*-1.
- Si no se da el segundo parámetro se considera como N la longitud del vector. Para que el algoritmo sea eficiente *N* debe ser potencia de 2.
- Si la longitud de x es menor que *N*, el vector se rellena con ceros. Si es mayor el vector es truncado.

x = ifft(X)

Calcula la transformada de Fourier inversa del vector X. También se puede especificar el número de puntos N con *ifft(X,N)*

X=fftshift(x)

Reordena el vector X en orden creciente de frecuencias de tal manera que la componente continua queda centrada.

ANEXO 1

qsurf

```
%  Function mfile       qsurf (quick surf)
%
%     Aligerador para gráfico de funciones de dos variables
%    se llama qsurf(f,esquinas, n)  f es una inline función,
%     qsurf('f', esquinas, n) con f en archivo m.
%     esquinas es el vector [a b c d], suponiendo a < b y c < d. f se grafica en el rectángulo
%
%                (a,d)  ~~~~~~~~ (b,d)
%                  |                |
%                  |                |
%                  |                |
%                (a,c)  ~~~~~~~~ (b,c)
%
%    la malla es n veces n, por default n es 50

 function out = qsurf(f,esquinas,n)
       if nargin < 3
          n = 50;
       end
       a = esquinas(1); b = esquinas(2); c = esquinas(3); d = esquinas(4);
       x = linspace(a,b,n+1); y = linspace(c,d,n+1);
       [X,Y] = meshgrid(x,y);
       Z = feval(f,X,Y);
       surf(X,Y,Z)
```

Armónicos

```
% Programa para obtener los armónicos de una señal cuadrada
% con un periodo de T = 1 seg
Wo = 2*pi/T;
t = 0:0.001:pi;
y = square(2*pi*t/T);
plot(t,y,'r-')
grid
hold on
x = 0;
for k = 1:2:10
    x = x + (4*sin(k*Wo*t))/(pi*k);
   y((k+1)/2,:) = x;
   plot(t,x), pause
end
title('componentes armónicos')
xlabel('tiempo'), ylabel('amplitud')
```

four

```
function four(N)
x=-1:0.05:1;
a=zeros(1,N);
sumparcial=1/3;
```

```
for k=1:N
a(k)=quadl(@fun,-1,1,1e-9,[],k);
sumparcial=sumparcial+a(k)*cos(k*x*pi);
end
f=x.^2;
plot(x,f,'b',x,sumparcial,'g'),shg
function y=fun(t,n)
y=(t.^2).*cos(n*pi*t);
```

fourier_comp

```
% function coeff = fourier_comp(func,m,L)
% fourier_comp evalúa numéricamente los coeficientes de la serie de Fourier.
% coeff = fourier_comp(func,m,L) busca aproximar la función func
% de -L a L con m terminus de la serie de Fourier usando quad (MATLAB functions).
% func es una function handle, debe acepatr un argumento vector x y devuelve un vector resultado y
% Esta funcion devuelve una estructura con estos campos: % coeff.a0
% coeff.an
% coeff.bn
% Esta es la serie de Fourier.
%                    inf
%                   ~~~
%                   \
%       f(x) = a0/2 +  \  an * sin(n*pi*x/L) + bn * con(n*pi*x/L)
%                   /
%                   /
%                   ~~~
%                   n = 1
%
%   Ejemplo:
%
%   f = @(x)x.*cos(x);
%   coeff = fourier_comp(f,3,pi)
%     coeff =
%     a0: 0
%     an: [-0.5000 1.3333 -0.7500]
%     bn: [1.4136e-016 0 7.0679e-017]
%
%   ver además
%   fourier_gui
%

% func debe ser una función handle

for n = 1:m
   an(n) = quad(@fa,-L,L)/L;
   bn(n) = quad(@fb,-L,L)/L;
end
a0 = quad(func,-L,L)/L;
coeff.a0 = a0;
coeff.an = an;
coeff.bn = bn;
   function y = fa(t)
      y = func(t).*sin(n*pi*t/L);
   end
```

```
    function y = fb(t)
        y = func(t).*cos(n*pi*t/L);
    end
end
```

fourier_gui

```
% function fourier_gui
% calculadora serie de Fourier
% FOURIER_GUI inicia un  GUI que grafica una function contra la n-th suma parcial de su serie
% de fourier._gui usó fourier_comp para los coefcientes de la serie de Fourier de –L a L.
% f(x): Funciónon handle o nombre de function M, debe aceptar un argumento vector x y devuelve
% un vector resultado y
% n: Número de terminos de la serie de fourier
% 2L: período de la función
% fourier_gui agrega menus de la serie de Fourier sobre la ventana de figures, consisten en:
% Export: exportar coeficiente serie de Fourier
% Help: muestra ayuda de fourier_gui
% About: displays copyright and contact information.
%
%   Ejemplo
%   hallar la serie de Fourier de la función [1]
%          |  0   -pi < x <= 0
%   f(x) = |
%          |  x   0 <= x < pi
%
% function y = fx(x)
% i = find((x > -pi) & (x <= 0));
% y = 2*x;
% y(i) = 0;
% y(1) = y(end);
%
% Ver también
%
%   fourier_comp
%
if findall(0,'tag','fourier')
    figure(findall(0,'tag','fourier'));
else
    figure('units','pixel','position',[100 80 600 600],'tag','fourier');
end
informpanelH = uipanel('units','pixel','position',[20 450 560 140],...
    'BackgroundColor','white');
uicontrol('unit','pixel','position',[75 107 30 16],...
    'parent',informpanelH,'style','text','string','f(x)')
fH = uicontrol('unit','pixel','position',[110 100 200 30],...
    'parent',informpanelH,'style','edit','string','@(x)x.*cos(x)');
uicontrol('unit','pixel','position',[315 107 20 16],...
    'parent',informpanelH,'style','text','string','n')
nH = uicontrol('unit','pixel','position',[340 100 30 30],...
    'parent',informpanelH,'style','edit','string','3');
uicontrol('unit','pixel','position',[375 107 20 16],...
    'parent',informpanelH,'style','text','string','L')
LH = uicontrol('unit','pixel','position',[400 100 30 30],...
    'parent',informpanelH,'style','edit','string','pi');
```

```
fmenu = uimenu('Label','Fourier series');
uimenu(fmenu,'Label','Export','Callback',@export);
uimenu(fmenu,'Label','Help','Callback',@help);
uimenu(fmenu,'Label','About','Callback',@about);
uicontrol('unit','pixel','position',[435 100 50 30],'parent',...
   informpanelH,'style','pushbutton','string','plot','Callback',@fplot)
plotpanelH = uipanel('units','pixel','position',[20 20 560 420],...
   'BackgroundColor','white');
axes('parent',plotpanelH);
   function fplot(h,e)
      try
         func =  evalin('base',get(fH,'string'));
      catch
         func = str2func(get(fH,'string'));
      end
      m = str2num(get(nH,'string'));
      L = str2num(get(LH,'string'));
      y = fourier_comp(func,m,L);
      set(gcf,'UserData',y)
      a0 = y.a0;
      an = y.an;
      bn = y.bn;
      dx = 2*L/50;
      xspan = -L:dx:L;
      fx = [];
      for x = xspan
         fx = [fx a0/2+sum(an(1:m).*sin(pi*(1:m)*x/L)...
            +bn(1:m).*cos(pi*(1:m)*x/L))];
      end
      plot(xspan,fx,'b',xspan,func(xspan),'r','LineWidth',2)
      grid on
      legend('Fourier series','User function',0)
   end
   function export(h,e)
      y = get(gcf,'UserData');
      assignin('base', 'an', y.an)
      assignin('base', 'bn', y.bn)
      assignin('base', 'a0', y.a0)
   end
   function help(h,e)
      doc fourier_gui
   end
   function about(h,e)
      msgbox({'   Fourier series calculator Version 1.0 '})
   end
end
```

fousen

```
function fousen(N)
x=-2:0.005:2;
sumparcial=0;
b=zeros(1,N);
for k=1:N
b(k)=2/(k*pi);
```

```
sumparcial=sumparcial+b(k)*sin(k*x*pi/2);
end
f=(x<0).*(-1-x/2)+(x>=0).*(1-x/2);
plot(x,f,'b',x,sumparcial,'g'),shg
```

fourier

```
function fourier(a0,an,bn,w,t,titulo)
sum=0;
an=inline(an);
bn=inline(bn);
n=400;
for k=1:n
   sum=sum+a0+an(k)*cos(w*k*t)+bn(k)*sin(w*k*t);
end
plot(t,sum),xlabel('t'),ylabel('f(t)'),grid on,title(titulo)
```

graficador

```
%  Programa que realiza la petición de una función
%  para graficarla y darle movimiento con ayuda de la función movi

clear;
clc;
opcion='s';
while opcion~=('n')
fprintf('\n\n Programa para graficar funciones periodicas y darles movimiento');
fprintf('\n\n\n');

%  Da a escoger que tipo de funciones se quiere graficar

fprintf('\n a.- Funciones trigonometricas, polinomiales o exponenciales');
fprintf('\n b.- Series de Fourier de forma trigonometrica');
fprintf('\n c.- Salir');
fprintf('\n');
opcion=input(' Elige una opcion ','s');

%  Realizando acciones de la opción a

if opcion==('a')
   clc;close all; clear;
   fprintf('RECUERDA QUE SI QUIERES ELEVAR A UNA POTENCIA LA VARIABLE "X"
DENTRO DE LA FUNCION');
   fprintf('\nDEBERAS PONER UN PUNTO ENTRE "X" Y EL SIMBOLO "^" ');
   disp(' ');

   %  Pidiendo los datos a utilizar

   fun=input('Introduce tu funcion correctamente: ','s');
   x=input('Introduce el rango de valores de x de la siguiente manera: "valor inicial:incremento:valor
final" ');
   titulo=input('¿Que titulo quieres que lleve la grafica? ','s');
   equis=input('¿Cual sera el nombre de las abcisas? ','s');
   ye=input('¿Cual sera el nombre de las ordenadas? ','s');
```

```
    %  Llamando a la funcion movi

    movi(x,fun,equis,ye,titulo)
    fprintf('\n\n Ahi tienes tu grafica espero te agrade');

%  Realizando las acciones de la acción b

elseif opcion==('b')
    clc;close all;clear;
    fprintf('RECUERDA QUE SI QUIERES ELEVAR A UNA POTENCIA LA VARIABLE "n"
DENTRO DE LA FUNCION');
    fprintf('\nDEBERAS PONER UN PUNTO ENTRE "n" Y EL SIMBOLO "^" ');
    disp(' ');

    %  Pidiendo los datos a utilizar

    a0=input('Dame el valor de a0 ');
    an=input('Cual es la funcion an ','s');
    bn=input('Cual es la funcion bn ','s');
    w=input('dame el valor para w ');
    t=input('Introduce el rango de valores de t de la siguiente manera: "valor inicial:incremento:valor
final" ');
    titulo=input('¿Que titulo quieres que lleve la grafica? ','s');

    %  Llamando a la función fourier

    fourier(a0,an,bn,w,t,titulo)
    fprintf('\n\n Ahi tienes tu grafica espero te agrade');
end

opcion=input('¿deseas hacer otra grafica? (s/n) ','s');
end
```

movi

```
%  Función que aparenta darle movimiento a una función periódica o a una oscilación
%
function movi(x,fun,equis,ye,titulo)
for n=linspace(0,2*pi,100)
    y=inline(fun);
    y1=y(x+n);
    subplot(2,1,1),plot(x,y1),xlabel(equis),ylabel(ye),title(titulo),grid
    subplot(2,1,2),plot(x,y(x)),xlabel(equis),ylabel(ye),title(titulo),grid,legend(fun)
    pause(0.01)
end
figure(2)
plot(x,y(x)),xlabel(equis),ylabel(ye),title(titulo),grid,legend(fun)
return
```

frenet

```
% esta function toma como entrada la función en tres coordenadas
% de una curva parametrizada en 3D. Llama  frenet(x,y,z)
% si x,y,z se dan como funciones inline,   frenet('x','y', z'),
% si x,y,z están en archivos m.  El usuario es interrogado
% para entrar un valor de  t.  T,N,B  se calculan en este valor
```

```
% usando cocientes en diferencias. Los vectores se adjuntan al gráfico de la curva
% en el punto  x(t), y(t), z(t).  Esto puede efectuarse  4 veces
% las components de  T,N,B se imprimen como un vector columna
% como la curvatura kappa, y las components tangencial y normal de la aceleración

function w = frenet(x,y,z)
 hold on
 for n = 1:4
     t = input('enter a value of  t   ')
     h = 1e-6;
     p0 = [feval(x, t), feval(y,t), feval(z,t)];
     p1 = [feval(x, t+h), feval(y,t+h), feval(z,t+h)];
     p2 = [feval(x, t-h), feval(y,t-h), feval(z,t-h)];

     v = (p1-p2)/(2*h);
     a = (p1 - 2*p0 + p2)/h^2;

     T = v/norm(v);
     projection = a - dot(a,T)*T;

     N = projection/norm(projection);

     B = cross(T,N);

     kappa = norm(cross(v,a))/norm(v)^3;

     arrow3(p0, T, 'r')
     arrow3(p0, N, 'k')
     arrow3(p0, B, 'g')

     text(p0(1) + T(1), p0(2) + T(2), p0(3) + T(3), ' T ')
     text(p0(1) + N(1), p0(2) + N(2), p0(3) + N(3), ' N ')
     text(p0(1) + B(1), p0(2) + B(2), p0(3) + B(3), ' B ')
     disp('      T            N            B')
     frame = [T', N', B']
     disp('  ')
     disp(' kappa       a_T           a_N  ')
     [kappa, dot(a,T),   dot(a,N)]

end
hold off

 function out = arrow3(P,V,color)

if nargin < 3
   color = 'b';
end

x0 =P(1); y0 = P(2); z0 = P(3);
a = V(1); b = V(2); c = V(3);
l = max(norm(V), eps);

x = [x0 x0+a]; y = [y0 y0+b]; z = [z0 z0+c];
hchek = ishold;
plot3(x,y,z,color)
```

```
hold on

h = l - min(.2*l, .2) ;
v = min(.2*l/sqrt(3), .2/sqrt(3) );

upper = [h, v*tan(pi/6), 0]';
lower = [h, -v*tan(pi/6), 0]';

r = sqrt(a^2 +b^2);
if r > 0

   col1 = [a b c]/l;
   col2 = [-b/r, a/r, 0];
   col3 = [-a*c/(l*r), -b*c/(l*r), r/l];

   Q = [col1; col2; col3]' ;

else
   Q = [0 0 -1; 0 1 0; 1 0 0];
end

p = Q*upper; q = Q*lower;

plot3([x0+p(1), x0+a], [y0+p(2), y0+b], [z0+p(3), z0+c], color)
plot3([x0+q(1), x0+a], [y0+q(2), y0+b], [z0+q(3), z0+c], color)

if hchek == 0
   hold off
end
```

arrow

```
%  dibuja una flecha de desde(x0, y0) a (x0+a, y0+b).  Escribiendo P = [x0, y0] y
%  V = [a,b], el llamado complete es(P,V, color).  Color es opcional
%  si se llama arrow(P,V), el color es azul.

function y = arrow(P,V,color)

if nargin < 3
   color = 'b';
end
x0 = P(1); y0 = P(2);
a = V(1); b = V(2);

l = max(norm(V), eps);
u = [x0 x0+a]; v = [y0 y0+b];
hchek = ishold;

plot(u,v,color)
hold on
h = l - min(.2*l, .2) ; v = min(.2*l/sqrt(3), .2/sqrt(3) );

a1 = (a*h -b*v)/l;
b1 = (b*h +a*v)/l;
```

```
plot([x0+a1, x0+a], [y0+b1, y0+b], color)

a2 = (a*h +b*v)/l;
b2 = (b*h -a*v)/l;

plot([x0+a2, x0+a], [y0+b2, y0+b], color)
if hchek == 0
hold off
end
```

arrow3

```
%   dibuja una flecha de desde(x0, y0,z0) a (x0+a, y0+b,z0+c).  Escribiendo P = [x0, y0,z0] y
%  V = [a,b,c], el llamado complete es(P,V, color).  Color es opcional
%  si se llama arrow(P,V), el color es azul

 function out = arrow3(P,V,color)

if nargin < 3
    color = 'b';
end

x0 =P(1); y0 = P(2); z0 = P(3);
a = V(1); b = V(2); c = V(3);
l = max(norm(V), eps);

x = [x0 x0+a];
y = [y0 y0+b];
z = [z0 z0+c];
hchek = ishold;
plot3(x,y,z,color)
hold on
h = l - min(.2*l, .2) ;
v = min(.2*l/sqrt(3), .2/sqrt(3) );

upper = [h, v*tan(pi/6), 0]';
lower = [h, -v*tan(pi/6), 0]';

r = sqrt(a^2 +b^2);

if r > 0
   col1 = [a b c]/l;
   col2 = [-b/r, a/r, 0];
   col3 = [-a*c/(l*r), -b*c/(l*r), r/l];
   Q = [col1; col2; col3]' ;

else
   if c > 0
      Q = [0 0 -1; 0 1 0; 1 0 0];
   else
      Q = [0 0 1; 0 1 0; -1 0 0];
   end

end
```

```
p = Q*upper; q = Q*lower;

plot3([x0+p(1), x0+a], [y0+p(2), y0+b], [z0+p(3), z0+c], color)
plot3([x0+q(1), x0+a], [y0+q(2), y0+b], [z0+q(3), z0+c], color)

if hchek == 0
  hold off
end
```

curl

```
%   se llama   curl(u,v,esquinas)   cone u = u(x,y) y v = v(x,y)
%   so en campo vectorial F = [u,v].  El archivo calcula el curl
%   con diferencias centradas. El campo vectorial e muestra usando
%   quiver , un mapa pcolor se hace del curl, mostrando según color  los valores del curl.

function out = curl(u,v, corners)
a = corners(1); b = corners(2); c = corners(3); d = corners(4);

x = linspace(a,b,51); y = linspace(c,d,51);

[X,Y] = meshgrid(x,y);
h = 10^(-6);

C1 = (feval(v,X+h, Y) - feval(v, X-h,Y))/(2*h);
C2 = (feval(u,X, Y+h) - feval(u,X, Y-h))/(2*h);
C = C1 - C2;
close
pcolor(X,Y,C); map = jet;
myjet = map(33:64,:);
colormap(myjet);
shading interp
colorbar
hold on

xx = linspace(a,b, 11); yy = linspace(c,d,11);

[XX,YY] = meshgrid(xx,yy);

U = feval(u,XX,YY); V = feval(v, XX,YY);
quiver(XX,YY,U,V, 'k');
axis image
hold off
```

d3

```
u=inline('1+0*x','x','y','z');
v=inline('x+y.^2','x','y','z');
w=inline('z','x','y','z');
x=linspace(-2,3,6);
y=linspace(-1,2,6);
[X,Y]=meshgrid(x,y);
for z=-1:.4:1
    Z=z+0*X;
    U=u(X,Y,Z);
```

```
    V=v(X,Y,Z);
    W=w(X,Y,Z);
    quiver3(X,Y,Z,U,V,W)
    hold on
end
```

da

```
function out=da(u,v)
n1=3*cos(u).*cos(v).^2;
n2=3*sin(u).*cos(v).^2;
n3=cos(v).*sin(v)-4*cos(u).*sin(v).*cos(v).^2;
out=sqrt(n1.^2+n2.^2+n3.^2);
```

Descent

```
%    Implementa el método de descenso más rápido 2D.
%    Se debe ingresar una function inline  f(x,y), sólo.
%    y el vector esquinas = [a b c d] del rectángulo donde
%    se cree cae el mínimo. N es el número de iteraciones.
%    se llama descent(f,corners, N)  function out = descent(f,corners,N)
%    hacer expression simbólica para f(x,y) para calcular derivadas
 syms x y;
 ff = sym(f(x,y));
 ffx = diff(ff,x);
 ffy = diff(ff,y);
% hacer funciones inline numérica para fx y fy
 fx = inline(vectorize(ffx),'x', 'y')
 fy = inline(vectorize(ffy),'x', 'y')

 % plot contours de f(x,y)
 a = corners(1); b = corners(2); c = corners(3); d = corners(4);

 xmesh = linspace(a,b,101);
 ymesh = linspace(c,d,101);

 [X,Y] = meshgrid (xmesh, ymesh);
 Z = feval(f,X,Y);
 f1 = min(min(Z)); f2 = max(max(Z));
 levels = linspace(log(f1+.01),log(f2), 21);
 levels = exp(levels);
    contour(X,Y,Z,levels)
    hold on
    axis equal
    hold on
    % iniciar el punto de arranque clickeando el plot contour    p = ginput(1);
    plot(p(1), p(2), '*r')
 x = zeros(N,1);
 y = x;

 x(1) = p(1); y(1) = p(2);

for n = 2:N
   % v es la dirección del descenso más rápido
   v = -[feval(fx, x(n-1),y(n-1)), feval(fy,x(n-1), y(n-1))];
```

```
  l = norm(v);
  if l < eps
    sprintf(' grad f = 0, n = %2d   x = %2.5f  y = %2.5f', n-1, x(n-1), y(n-1))
    break
  end

  v = v/l;
  tmax = .1*max(b-a, d-c);
  % construye la línea de valores  x y sobre la cual se investiga el mínimo de f
  M = 10001;
  t = linspace(0, tmax, M);
  xvals = t*v(1); yvals = t*v(2);
  xx = x(n-1) + xvals;
  yy = y(n-1) + yvals;

  fvals = feval(f,xx,yy);
  % plot(xx,fvals)
  [m,j]= min(fvals);

  for k = 1:10
    % si el mínimo ocurre al punto final, extendemos la línea de búsqueda y
    % miramos  después Se puede hacer al menos 10 veces
    if j == M
      xx = xx(M) + xvals;
      yy = yy(M) + yvals;
      fvals = feval(f,xx,yy);
     % plot(xx,fvals)
      [m,j] = min(fvals);
    else
      break
    end
  end

  x(n) = xx(j);
  y(n) = yy(j);

  disp('hit return to continue ')
  pause

  plot(x(n), y(n), '*r')
end

plot(x,y, 'r')

disp( '    x      y        f(x,y)   ')

[x,y,feval(f,x,y)]
hold off
```

ejemplo fft

```
%Ejemplo, fft
N=128;
t=linspace(0,3,N);
f=2*exp(-3*t);
```

```
Ts = t(2) - t(1);
Ws = 2*pi/Ts;
F=fft(f);
Fp=F(1:N/2+1)*Ts;
W=Ws*(0:N/2)/N;
Fa=2./(3+j*W);
plot(W,abs(Fa),W,abs(Fp),'+')
xlabel('Frequencia,Rad/s')
ylabel('|F(\omega)|')
title('Figura 1.1; Aproximación transformada Fourier')
```

findcrit

```
%  findcrit
%  la función f(x,y) en archive m o inline.
%  Usuario debe elegir las ''esquinas = [a b c d]"
%  que son las del reactángulo
%
%               (a,d)~~~~~~~~~(b,d)
%                 |               |
%                 |               |
%                 |               |
%               (a,c)~~~~~~~~~(b,c)
%
%  Se llama findcrit(f,esquinas) donde f es function inline
%  y findcrit('f',esquinas)  cuando se da f en archive m.
%  El archive halla el maximo y el mínimo en una malla de 50 x 50
%  sobre este rectángulo y muestra las curvas de nivel de ese rectángulo.
%  el usuario luego decide sobre el menor y cliquea sobre las esquinas inferiores
%  izquierda y derecha. Un tercer click en cualquier lugar. El rectángulo menor
%  se levanta, las curvas de nivel de fse muestran este menor rectángulo, hallándose
%  el máx y mín de f sobre la malla 50x50. esto puede hacerse hasta 4 veces.

function  out = findcrit(f, esquinas)

a = esquinas (1); b = esquinas (2); c = esquinas (3); d = esquinas (4);
m = .5*(c+d);
c = m -.4*(b-a);
d = m+ .4*(b-a);

x = linspace(a,b,50);
y = linspace(c,d,50);
[X,Y] = meshgrid(x,y);
Z = feval(f, X,Y);
contour(X,Y,Z);
disp(' ')
     disp('  x          y          fmax       x          y          fmin')
disp('  ')
for n = 1:4
   [x esquinas, y esquinas]  = ginput(2);
   a = x esquinas (1);
   b = x esquinas (2);
   c = y esquinas (1);
   d = y esquinas (2);
   m = .5*(c+d);
```

```
  c = m -.4*(b-a);
  d = m +.4*(b-a);
  hold on
  plot([a b b a a], [c c d d c])
  hold off
  ginput(1);
  x = linspace(a,b,50);
  y = linspace(c,d,50);
  [X,Y] = meshgrid(x,y);
  Z = feval(f,X,Y);
  contour(X,Y,Z);
  hold on
        [row,I] = max(Z);
        [fmax, j] = max(row);
        plot([x(j)], [y(I(j))], '*')
        text(x(j), y(I(j)), '\fontsize{14pt}max')

        [row,J] = min(Z);
        [fmin,k] = min(row);
        plot([x(k)], [y(J(k))], '*')
        text(x(k), y(J(k)), '\fontsize{14pt}min')
        [x(j), y(I(j)), fmax, x(k), y(J(k)), fmin]

end

hold off
```

Flow1

```
%   Se llama flow1(esquinas, T) donde esquinas = [a,b,c,d] es el
%   vector de coordenadas esquinas del  rectangulo R donde
%   se muestra el campo vectorial.  T intervalo de tpo. en el que se
%   se sigue el flujo.
%   El flujo se genera por un campo vector [u,v].  u(x,y)
%   y v(x,y) son los provistos en archivos m,u.m,v.m.
%   También se necesita wdot.m
%   Luego de llamar, el campo vectorial se plotea, y el programa pide
%   el número de puntos de arranque de  líneas de corriente. Luego que se
%   ingresa el número M, hit return. El programa entonces espera que el usuario
%   cliquee en la figura en los puntos deseados de arranque.
%   Las líneas arrancan en este punto calculando con ode45 y dibujando, hasta M veces.

 function out = flow1(esquinas,T)
 a = esquinas (1); b = esquinas (2); c = esquinas (3); d = esquinas (4);
 x = linspace(a,b,16); y = linspace(c,d,16);
 [X,Y] = meshgrid(x,y);

 U = u(X,Y); V = v(X,Y);

quiver(X,Y,U,V)
axis([a b c d])
hold on

M = input('entre el número de puntos de arranque    ')
```

```
m = 1;
while m < M+1
  [aa bb] = ginput(1);
  x0 = aa(1); y0 = bb(1);
  w0 = [x0, y0]';

  [t,w] = ode45('wdot', [0, T], w0);
  plot( w(:,1), w(:,2), 'r')
  nsteps = size(w,1)
  m = m+1;
end

hold off
```

flow2

```
%   archivo de función para calcular la deformación de un disco siguiendo
%   el flujo de un campo vectorial.  Se llama  flow2(esquinas, times) donde
%   esquinas = [a,b,c,d],es el vector de coordenadas de las esquinas del rectángulo
%   donde se muestra el campo vectorial, times = [t1 t2 t3 t4] es un vector de 4 tpos
%   El disco deformado se muestra a estos tpos.   t1, t2, t3,t4.   El campo debe darse
%   en archive m u.m y v.m.,también usa wdot.m . t1,t2,t3,t4 son los tpos en los cuales
%   se observa el disco deformado. Luego de llamarlo, el programa pide un lugar de
%   colocación del disco. La imagen del disco en el flujo se calcula y dibuja usando odee45
%   sobre los puntos frontera del disco, el área de cada imagen se calcula aproximadamente

function out = flow2(esquinas, times)
 xmin = esquinas (1); xmax = esquinas (2);
 width = xmax - xmin;
 ymin = esquinas (3); ymax = ymin+.8*width;

 x = linspace(xmin,xmax,16); y = linspace(ymin,ymax ,16);
 [X,Y] = meshgrid(x,y);

 U = u(X,Y); V = v(X,Y);

quiver(X,Y,U,V)
axis([xmin,xmax, ymin, ymax])
hold on

%center = input('entrar coordenadas del centro de círculo [a,b]  ')
%a = centro(1); b = centro(2);

disp(' clicquee donde desee inicie el flujo  ')
disp('  ')
[a,b] = ginput(1)
r = width/20
n = 20; theta = 0:2*pi/n: 2*pi;

x0 = a + r*cos(theta); y0 = b + r*sin(theta) ;
fill(x0, y0, 'r')

x1 = zeros(1,n); y1 = x1;
x2 = x1; y2 = y1;
x3 = x2; y3 = y2;
```

```
x4 = x3; y4 = y3;
tspan = [0 times];

for j = 1:n+1

   w0 = [x0(j), y0(j)];
   [t,w] = ode45('wdot', tspan, w0);
   x1(j) = w(2,1); y1(j) = w(2,2);
   x2(j) = w(3,1); y2(j) = w(3,2);
   x3(j) = w(4,1); y3(j) = w(4,2);
   x4(j) = w(5,1); y4(j) = w(5,2);
end

area0 = 0;
for j = 1:n
    area0 = area0 +.5*(x0(j+1) +x0(j))*(y0(j+1) - y0(j));
end
area0

fill(x1,y1, 'r')
area1 = 0;
for j = 1:n
    area1 = area1 + .5*(x1(j+1) + x1(j))*(y1(j+1) - y1(j));
end
area1

fill(x2,y2, 'r')
area2 =0;
for j = 1:n
    area2 = area2 + .5*(x2(j+1) +x2(j))*(y2(j+1) - y2(j));
end
area2

fill(x3,y3, 'r')
area3 = 0;
for j = 1:n
    area3 = area3 + .5*(x3(j+1) + x3(j))*(y3(j+1) - y3(j));
end
area3

fill(x4,y4, 'r')
area4 = 0;
for j = 1:n
  area4 = area4 + .5*(x4(j+1) + x4(j))*(y4(j+1) - y4(j));
end
area4

hold off
```

flux2

```
%    Se llama flux2(u,v,esquinas).  u = u(x,y) y v = v(x,y) son
%    las componentes de un campo vectorial F = [u,v]. u y v se escriben
%    como archivos m o funciones inline. esquinas [a,b,c,d]
%    es el vector de coordenadas de las esquinas del rectángulo R.
```

```
%    Luego del llamado, se entra el número de segmentos de línea que forman
%    el camino, usualmente sera la frontera de un polígono. Si se va en sentido
%    antihorario, los puntos normales n salen.
%    El usuario usa click izquierdo sobre la figura, hasta completer el número de
%    segmentos , el archive estima la integral de flujo F en el camino C y el area del
%    polígono encerrado

function out = flux2(u,v, esquinas)

  a = esquinas (1); b = esquinas (2); c = esquinas (3); d = esquinas (4);

  N = input('entre el número de segmentos del camino o trayectoria  ')

  x = linspace(a,b,11);
  y = linspace(c,d,11);

  [X,Y] = meshgrid(x,y);
  U = feval(u,X,Y);
  V = feval(v,X,Y);
  quiver(X,Y,U,V)
  axis image
  hold on

  sumflux = 0;
  A = 0;

  [x0,y0] = ginput(1);

  n = 50;
  simpvec = 2*ones(1,n+1);
  simpvec(2:2:n) = 4*ones(1,n/2);
  simpvec(1) = 1; simpvec(n+1) = 1;

  for j = 1:N
    [x1, y1] = ginput(1);
    plot([x0,x1], [y0,y1], 'r')

    xm = .5*x0 + .5*x1; ym = .5*y0 + .5*y1;
    denom = norm([x1-x0, y1-y0]);
    normal = [y1-y0, x0-x1]/denom;
    l = .025*max(b-a,d-c);
    xtip = xm +3*l*normal(1); ytip = ym + 3*l*normal(2);
    plot([xm, xtip], [ym, ytip], 'r')

    nx = l*normal(1); ny = l*normal(2);
    xedge = [xtip, xtip-nx+ny, xtip-nx-ny];
    yedge = [ytip, ytip-ny-nx, ytip-ny+nx];
    fill(xedge, yedge, 'r')
    segment_number = num2str(j);
    text(x1,y1, segment_number)


    dx = (x1-x0)/n; dy = (y1-y0)/n;
    xx = linspace(x0, x1,n+1);
    yy = linspace(y0 ,y1,n+1);
```

```
  f1 = simpvec*feval(u,xx,yy)'*dy/3;
  f2 = simpvec*feval(v,xx,yy)'*dx/3;
  flux = f1 - f2
  sumflux = sumflux + flux;
  A = A + .5*(x1 + x0)*(y1 - y0);
  x0 = x1; y0 = y1;
end
area = A
total_flux = sumflux
hold off
```

frenet

```
% La entrada son las funciones coordenadas de una curva parametrizada 3D
% Se llama  frenet(x,y,z) para  x,y,z  inline , y  frenet('x','y', z'),si x,y,z son m.
% Se pide luego un valor de t, los vectores T,N,B del frenet frame se calculan
% a este valor de t por cocientes en diferencias. Los vectores se adjuntan al
% gráfico de la curva en el punto x(t), y(t), z(t)., hasta 4 veces.
% Las componentes de T,N,B se imprimen como vectores columnas como
% la curvatura kappa, las componentes normal y tangencial de la aceleración.

function w = frenet(x,y,z)
hold on
for n = 1:4
   t = input('entre un valor de  t   ')
   h = 1e-6;
   p0 = [feval(x, t), feval(y,t), feval(z,t)];
   p1 = [feval(x, t+h), feval(y,t+h), feval(z,t+h)];
   p2 = [feval(x, t-h), feval(y,t-h), feval(z,t-h)];

   v = (p1-p2)/(2*h);
   a = (p1 - 2*p0 + p2)/h^2;

   T = v/norm(v);
   projection = a - dot(a,T)*T;

   N = projection/norm(projection);

   B = cross(T,N);

   kappa = norm(cross(v,a))/norm(v)^3;

   arrow3(p0, T, 'r')
   arrow3(p0, N, 'k')
   arrow3(p0, B, 'g')

   text(p0(1) + T(1), p0(2) + T(2), p0(3) + T(3), ' T ')
   text(p0(1) + N(1), p0(2) + N(2), p0(3) + N(3), ' N ')
   text(p0(1) + B(1), p0(2) + B(2), p0(3) + B(3), ' B ')
   disp('      T           N           B')
   frame = [T', N', B']
   disp('  ')
   disp(' kappa       a_T          a_N  ')
```

```
    [kappa, dot(a,T),   dot(a,N)]

end
hold off

 function out = arrow3(P,V,color)

if nargin < 3
   color = 'b';
end

x0 =P(1); y0 = P(2); z0 = P(3);
a = V(1); b = V(2); c = V(3);
l = max(norm(V), eps);

x = [x0 x0+a]; y = [y0 y0+b]; z = [z0 z0+c];
hchek = ishold;
plot3(x,y,z,color)
hold on

h = l - min(.2*l, .2) ;
v = min(.2*l/sqrt(3), .2/sqrt(3) );

upper = [h, v*tan(pi/6), 0]';
lower = [h, -v*tan(pi/6), 0]';

r = sqrt(a^2 +b^2);
if r > 0

   col1 = [a b c]/l;
   col2 = [-b/r, a/r, 0];
   col3 = [-a*c/(l*r), -b*c/(l*r), r/l];

   Q = [col1; col2; col3]' ;

else
   Q = [0 0 -1; 0 1 0; 1 0 0];
end

p = Q*upper; q = Q*lower;

plot3([x0+p(1), x0+a], [y0+p(2), y0+b], [z0+p(3), z0+c], color)
plot3([x0+q(1), x0+a], [y0+q(2), y0+b], [z0+q(3), z0+c], color)

if hchek == 0
   hold off
end
```

gdome

```
%  El programa usa la función de Matlab bucky  que da las coordenadas de
%  sobre la esfera unidad de un poliedro conformado por 12  pentágonos
%  y 20 hexágonos. Este poliedro se muestra.
%  Se hace hit return, y cada uno de los hexágonos se divide en 6 isosceles
%  Nuevamente hit return, los pentágomos se divden en triángulos, dando una
```

```
%  superficie de 180 triángulos de dos tipos.

   [B,V] = bucky;
   P = V';

   for i = 1:12
      x = P(1,(i-1)*5+1:i*5);
      y = P(2,(i-1)*5+1:i*5);
      z = P(3,(i-1)*5+1:i*5);
      fill3(x,y,z,'b')
      hold on
   end
  % for i = 1:60
  %   vertexnumber = num2str(i);
  %   text(V(i,1), V(i,2), V(i,3),vertexnumber)
  % end

   H1 = zeros(3,6);
   H1(:,1) = P(:,1);
   H1(:,2) = P(:,6);
   H1(:,3) = P(:,10);
   H1(:,4) = P(:,12);
   H1(:,5) = P(:,11);
   H1(:,6) = P(:,2);


  R = [ cos(.4*pi), -sin(.4*pi), 0; sin(.4*pi), cos(.4*pi),0; 0 0 1];
  H = zeros(3,120);
  for j =1:5
     H(:,(j-1)*6+1: j*6) = R^(j-1)*H1;
     xedge = H(1, (j-1)*6 +1: j*6);
     yedge = H(2, (j-1)*6 +1: j*6);
     zedge = H(3, (j-1)*6+ 1: j*6);
     fill3(xedge, yedge, zedge, 'w')
   end

   H6 = zeros(3,6);
   H6(:,1) = P(:,10);
   H6(:,2) = P(:,9);
   H6(:,3) = P(:,38);
   H6(:,4) = P(:,37);
   H6(:,5) = P(:,13);
   H6(:,6) = P(:,12);
   c1 = [.9 .9 .9];
   for j = 6:10
      H(:, (j-1)*6+1:6*j) = R^(j-6)*H6;
      xedge = H(1, (j-1)*6+1: j*6);
      yedge = H(2, (j-1)*6+1: j*6);
      zedge = H(3, (j-1)*6+1: j*6);
      fill3(xedge, yedge, zedge, c1)
   end

   H11 = zeros(3,6);
   H11(:,1) = P(:,13);
   H11(:,2) = P(:,37);
```

```
H11(:,3) = P(:,36);
H11(:,4) = P(:,34);
H11(:,5) = P(:,33);
H11(:,6) = P(:,14);
c2 = [.8 .8 .8];
for j = 11:15
   H(:, (j-1)*6 +1: j*6) = R^(j-11)*H11;
   xedge = H(1, (j-1)*6+1:j*6);
   yedge = H(2, (j-1)*6+1:j*6);
   zedge = H(3, (j-1)*6+1:j*6);
   fill3(xedge, yedge, zedge, c2)
end
 Q = [cos(.2*pi) -sin(.2*pi) 0; sin(.2*pi) cos(.2*pi) 0; 0 0 -1];

H16 = Q*H1;
c3 =  [.7 .7 .7];
for j = 16:20
   H(:,(j-1)*6+1: j*6) = R^(j-16)*H16;
   xedge = H(1, (j-1)*6+1:j*6);
   yedge = H(2, (j-1)*6+1:j*6);
   zedge = H(3, (j-1)*6+1:j*6);
   fill3(xedge, yedge, zedge, c3)
end

for i = 1:30
    for j = 1:i
        if B(i,j) > 0
            L = [V(i,:); V(j,:)];
            x = L(:,1);
            y = L(:,2);
            z = L(:,3);
            plot3(x,y,z)
        end
     end
end
view (128,15)
axis equal
disp('hit return para seguir ')
pause

for j = 1:20

    if j <= 5
       color = [1 1 1];
    elseif 6 <= j  & j <= 10
       color = c1;
    elseif 11 <= j  & j <= 15
       color = c2;
    else
         color = c3;
    end

p1 = H(:, (j-1)*6+1);
p2 = H(:, (j-1)*6+2);
p3 = H(:, (j-1)*6+3);
```

```
p4 = H(:, (j-1)*6+4);
p5 = H(:, (j-1)*6+5);
p6 = H(:, (j-1)*6+6);
A = [p1'; p2'; p3'];
d = .5*norm(p1+p4);

p = d*(A\[1 1 1]');

U = [p1,p,p2,p,p3,p,p4, p,p5, p,p6,p,p1];


x1 = U(1,1:3);
y1 = U(2,1:3);
z1 = U(3,1:3);
fill3(x1,y1,z1,color)

x2 = U(1,3:5);
y2 = U(2,3:5);
z2 = U(3,3:5);
fill3(x2,y2,z2,color)

x3 = U(1,5:7);
y3 = U(2,5:7);
z3 = U(3,5:7);
fill3(x3,y3,z3, color)

x4 = U(1,7:9);
y4 = U(2,7:9);
z4 = U(3,7:9);
fill3(x4,y4,z4,color)

x5 = U(1, 9:11);
y5 = U(2, 9:11);
z5 = U(3, 9:11);
fill3(x5,y5,z5, color)

x6 = U(1,11:13);
y6 = U(2,11:13);
z6 = U(3,11:13);
fill3(x6, y6, z6, color)

end
disp('hit return para seguir ')
pause

c4 = [0 .8  1];

for j = 1:12

   p1 = P(:,(j-1)*5+1);
   p2 = P(:,(j-1)*5+2);
   p3 = P(:,(j-1)*5+3);
   p4 = P(:,(j-1)*5+4);
   p5 = P(:,(j-1)*5+5);
```

```
    d = .2*norm(p1+p2+p3+p4+p5);

    A = [p1'; p2';p3'];
    p = d*(A\[1 1 1]');

    UU = [p1,p,p2,p,p3,p,p4,p,p5,p,p1];
    x1 = UU(1,1:3);
    y1 = UU(2,1:3);
    z1 = UU(3,1:3);
    fill3(x1,y1,z1,c4)

    x2 = UU(1,3:5);
    y2 = UU(2,3:5);
    z2 = UU(3,3:5);
    fill3(x2,y2,z2,c4)

    x3 = UU(1,5:7);
    y3 = UU(2,5:7);
    z3 = UU(3,5:7);
    fill3(x3,y3,z3,c4);

    x4 = UU(1,7:9);
    y4 = UU(2,7:9);
    z4 = UU(3,7:9);
    fill3(x4,y4,z4, c4);

    x5 = UU(1,9:11);
    y5 = UU(2,9:11);
    z5 = UU(3,9:11);
    fill3(x5,y5,z5,c4)

  end


  hold off
```

lagrange

```
%     f(x,y) como m o  inline.
%    dada una función restricta g(x,y) de la misma forma.
%    se provee un vector de esquinas = [a b c d] ,las esquinas de un
%     rectángulo
%
%                 (a,d)~~~~~~~~~(b,d)
%                   |               |
%                   |               |
%                   |               |
%                 (a,c)~~~~~~~~~(b,c)
%
%   el rectángulo se elige tal que incluya la curva de nivel
%   g(x,y) = 0.
%   Se llama lagrange(f,g,esquinas) , f y g dadas inline
%   lagrange('f', 'g', esquinas) si f  y g como m.
%         Cliquear sobre los puntos de la curva de nivel cero de g
%   la curva de nivel de f en el punto se muestra con el valor
```

```
%   de f en esta curva, se puede hacer hasta 5 veces.

function out = lagrange(f,g, esquinas)

a = esquinas (1);
b = esquinas (2);
c = esquinas (3);
d = esquinas (4);

x = linspace(a,b,50);
y = linspace(c,d,50);

[X,Y] = meshgrid(x,y);

Z = feval(f,X,Y);
W = feval(g,X,Y);
disp(' ')
disp('  x          y         f(x,y)  ')
disp('  ')

contour(X,Y,W, [0 0], 'k')
axis equal
hold on

for n = 1:5
   [p,q] = ginput(1);
   z0 = feval(f, p,q);
   contour(X,Y,Z, [z0 z0], 'r')
   z1 = num2str(z0);
   text(p,q, z1)
   [p,q,z0]
end
 xlabel('x eje ')
 ylabel('y eje ')

hold off
```

mslice

```
%  la entrada es una función f(x,y) en un archive m o como inline
%  y un punto P = (x0, y0).  Se llama mslice(f,P) para f como inline
%   mslice('f', P) para f en archivo m .
%    Al llamar a  mslice ,la gráfica de f cerca de P se muestra en la figura
%   superior, dando la chance de modificar el tamaño de la ventana de figura.
%    Entonces, se debe hit return.
%   la figura inferior aparece con varias curves de nivel de f y el pto. P.
%  Cliqueando con botón derecho sobre la figura inferior en distintas
%  direcciones alrededor de  P. El corte correspondiente aparece en la
%  figura superior,la dirección u y la derivada direccional Duf
%  se muestran en pantalla, puede hacerse hasta 5 veces.

function out = mslice(f,P)

global  z0 z1 z2
```

```
x0 = P(1); y0 = P(2);

x = x0-1:.05:x0+1;
y = y0-1:.05:y0+1;
[X,Y] = meshgrid(x,y);
Z = feval(f,X,Y);

z0 = feval(f,x0,y0);
z1 = max(max(Z));
z1 = max([z1,z0+1,0]);
z2 = min(min(Z));
z2 = min([z2,z0-1,0]);


subplot(2,1,1)
surf(X,Y,Z); shading interp; colormap(gray)
xlabel(' x eje ')
ylabel(' y eje ')
zlabel(' z eje ')
disp('puede agrandar la ventana y rotar la figura ')
disp('al terminar, de return ')
rotate3d on
pause
rotate3d off
disp('  ')
disp('    u1        u2        Duf    ')
disp('  ')
format compact

ylower = y0-.5:.02:y0+.5;
[XX,YY] = meshgrid(x,ylower);
ZZ = feval(f,XX,YY);

subplot(2,1,2)
  contour(XX,YY,ZZ, [.5*z0, z0, 1.5*z0], 'k')
  hold on
  plot([x0], [y0], '*')
  axis equal
  xlabel(' x eje ')
  ylabel(' y eje ')
  text(x0+.1, y0+.1, 'P')


for j = 1:5
  subplot(2,1,2)
    [q1 q2] = ginput(1);
    Q = [q1 q2];
    u = (Q-P)/norm(Q-P);
    arrow(P,Q-P)
  subplot(2,1,1)
    surf(X,Y,Z); shading flat; colormap(gray)
    hold on
    myslice(f, P, u)
    xlabel(' x eje ')
    ylabel(' y eje ')
```

```
      zlabel(' z eje ')
end
subplot(2,1,1)
hold off
subplot(2,1,2)
hold off
format loose

function out = myslice(f, P, u)
global a b c d z0 z1 z2

ztip = (z1 +z2)/2;

u = u/norm(u);
u1 = u(1); u2 = u(2);

x0 = P(1); y0 = P(2);

x1 = x0 -u1;
x2 = x0 + u1;
y1 = y0-u2;
y2 = y0+u2;

xedge = [x1 x0+.6*u1 x0+.9*u1 x0+.6*u1 x1];
yedge = [y1 y0+.6*u2 y0+.9*u2 y0+.6*u2 y1];
zedge = [z1 z1  ztip z2 z2];
fill3(xedge, yedge, zedge, 'c')
hold on
plot3([x0,x0], [y0,y0], [z1,z2],'r')
xx = [x0+.6*u1, x0+.7*u1, x2, x0+.7*u1, x0+.6*u1, x0+.9*u1];
yy = [y0+.6*u2, y0+.7*u2, y2, y0+.7*u2, y0+.6*u2, y0+.9*u2];
zz = [z1,z1,ztip,z2,z2, ztip];
fill3(xx,yy,zz, 'r')

hold off
h = 1e-6;
delz = feval(f, P(1) +h*u(1), P(2)+h*u(2)) -z0;
Duf = delz/h;
[u, Duf]

function out = arrow(P,V,color)
if nargin < 3
   color = 'b';
end

x0 = P(1); y0 = P(2);
a = V(1); b = V(2);

l = max(norm(V), eps);

u = [x0 x0+a]; v = [y0 y0+b];
hchek = ishold;
plot(u,v,color)
hold on
```

```
h = l - min(.2*l, .2) ; v = min(.2*l/sqrt(3), .2/sqrt(3) );

a1 = (a*h -b*v)/l;
b1 = (b*h +a*v)/l;

plot([x0+a1, x0+a], [y0+b1, y0+b], color)

a2 = (a*h +b*v)/l;
b2 = (b*h -a*v)/l;

plot([x0+a2, x0+a], [y0+b2, y0+b], color)

if hchek == 0
  hold off
end
```

riemann

```
%   Ccalcula la suma de Riemann y grafica la aproximación de a trozos la función.
%  Se llama  riemann(f,esquinas,graph) para f inline, y riemann('f', esquinas, graph)
%   para f como archivo m.
%    esquinas = [a b c d] es un vector de las coordenadas de las esquinas de un
%   rectángulo con esquinas (a,c), (b,c), (b,d), (a,d). Se supone a < b y c < d.
%    El tercer   argumento es opcional. Si se entra algún número para  graph,
%    se muestra el gráfico de la aproximación de a trozos.Si no se da, no se muestra
%    el gráfico.
%    Luego del llamado, se pide el número de subdivisiones n en la dirección x
%    y el número m de subdivisiones en la dirección y.
%    Las sumas de Riemann se calculan como la suma en los puntos medios de cada subrectángulo

function out = riemann(f, esquinas,graph)

 a = esquinas(1); b = esquinas (2); c = esquinas (3); d = esquinas (4);

 subdiv = input('entre el número de subdivisions en x e y como [n m] ')
 n = subdiv(1); m = subdiv(2);

  delx = (b-a)/n;  dely = (d-c)/m;

  x = a:delx:b;
  y = c:dely:d;

  p = a +.5*delx: delx : b-.5*delx;
  q = c +.5*dely: dely : d-.5*dely;

  [P,Q] = meshgrid(p,q);
  Z = feval(f,P,Q);

  disp(' valor aproximado de la integral ')
  out = sum(sum(Z))*delx*dely;

  if nargin < 3
    return
  end
```

```
 xplot = zeros(1,2*n);
 for j = 1:2:2*n-1
   xplot(j) = x((j+1)/2);
 end
 for j = 2:2:2*n
   xplot(j) = a + (j-2)*delx/2 + .99*delx;
 end

 yplot = zeros(1,2*m);
 for i = 1:2:2*m-1
    yplot(i) = y((i+1)/2);
 end
 for i = 2:2:2*m
    yplot(i) = c + (i-2)*dely/2 + .99*dely;
 end

[Xplot,Yplot] = meshgrid(xplot, yplot);

 Zplot = zeros(size(Xplot));

 j = 1:2:2*n-1;
 k = 2:2:2*n;

 for i = 1:m
   Zplot(2*i-1,j) = Z(i,:);
   Zplot(2*i-1,k) = Z(i,:);
   Zplot(2*i, :) = Zplot(2*i-1,:);
 end

  surf(Xplot, Yplot, Zplot); colormap(cool)
```

rrho

```
function out=rrho(r,theta,phi)
x=r.*cos(theta).*sin(phi);
out=(1+cos(x)).*r.^2.*sin(phi);
```

Simp2

```
%  la regla bidimensional de Simpson sobre un rectángulo.
%  Se llama simp2(f, esquinas) para f inline y simp2('f', esquinas).
%   para f en archive m, esquinas = [a b c d] es el vector de coord de
%   as esquinas del rectángulo con esquinas( a,c), (b,c), (b,d),(a,d).
%   se supone a< , c < d.
%   Luego nos pedirá el número de subdivisions, n y m, en x e y(impares)

function out = simp2(f, esquinas)

 a = esquinas (1); b = esquinas(2); c = esquinas (3); d = esquinas (4);
 disp('  ')
 disp('el número de subdivisiones n y m en cada dirección debe ser impar ')
 subdiv = input('entre n y m como [n m] ')
 n = subdiv(1); m = subdiv(2);
```

```
x = linspace(a,b, n+1); y = linspace(c,d,m+1);
[X,Y] = meshgrid(x,y);

svecx = 2*ones(size(x));
svecx(2:2:n) = 4*ones(1,n/2);
svecx(1) = 1; svecx(n+1) = 1;

svecy = 2*ones(size(y));
svecy(2:2:m) = 4*ones(1,m/2);
svecy(1) = 1; svecy(m+1) = 1;

S = svecy'*svecx;
T = S.*feval(f,X,Y);
disp('valor aproximado de la integral usando regal de Simpsons  ')

out = sum(sum(T))*(b-a)*(d-c)/(9*m*n);
```

simp 3

```
%    Regla de Simpson sobre un rectángulo sólido. Se llama simp3(f,esquinas)
%    para f inline, y simp3('f', esquinas) para  f como archivo m .
%    esquinas = [xmin, xmax, ymin, ymax, zmin, zman] define el rango de
%    integración.
%     Nos pide el número n de subdivisiones en la dirección x , m en y , p, en z
%     m,n, p impares.

function out = simp3(f, esquinas)

xmin =  esquinas (1); xmax = esquinas (2); ymin = esquinas (3);  ymax  = esquinas(4);
zmin  = esquinas (5); zmax = esquinas (6);
disp('  ')
disp('el nro. De subdivisiones m,n, p en cada dirección debe ser impar')
subdiv = input('entre el número de subdivisions como [n m p]  ')

n = subdiv(1); m = subdiv(2); p = subdiv(3);

x = linspace(xmin, xmax, n+1);
y = linspace(ymin, ymax, m+1);
z = linspace(zmin, zmax, p+1);

[X,Y,Z] = meshgrid(x,y,z);

svecx = 2*ones(1,n+1);
svecx(2:2:n) = 4*ones(1,n/2);
svecx(1) = 1; svecx(n+1) = 1;

svecy = 2*ones(1,m+1);
svecy(2:2:m) = 4*ones(1,m/2);
svecy(1) = 1; svecy(m+1) = 1;

svecz = 2*ones(1,p+1);
svecz(2:2:p) = 4*ones(1,p/2);
svecz(1) = 1; svecz(p+1) = 1;

S2 = svecy'*svecx;
```

```
S3 = zeros(m+1,n+1,p+1);

for k = 1:p+1
  S3(:,:, k) = svecz(k)*S2;
end

T = S3.*feval(f,X,Y,Z);
V = (xmax - xmin)*(ymax-ymin)*(zmax - zmin);
out = sum(sum(sum(T)))*V/(27*m*n*p);
```

simpvec

```
 function s=simpvec(n);
s=2*ones(1,n+1);
s(2:2:n)=4*ones(1,n/2);
s(1)=1;svec(n+1)=1;
```

trf

```
%   este archive dibuja un rectángulo con subrectángulos en la ventana izquierda
%   y su imagen curvilínea image con imagen de los subrectángulos en la derecha
%   la transformación es (u,v) - > (x,y) = (f(u,v), g(u,v)).
%    se llama is trf(f,g,esquinas) para f y g como inline, y trf('f', 'g',esquinas) para f
%    g como archivos m.
%    esquinas = [a b c d] vector de coordenadas del rectángulo a < x < b,  c < y <d.
%    Nos pedirá n,m, nro. Subdivisions en dirección x e y,respectivamente
%   Los gráficos se dibujan, con pausa en el programa.
%   Para seguir hits return. Entonces las imagines afinadas de los subrectángulos se
%   dibujan en la figura derecha y se calcula la suma de sus áreas
%    Esto da una aproximación de la región curvilínea en la figura derecha

function out = trf(f,g, esquinas)

subdiv = input(' entre el número de subdivisiones [n m]  ')

a =  esquinas (1); b = esquinas (2); c = esquinas(3); d = esquinas (4);
n = subdiv(1); m = subdiv(2);

t = linspace(a,b,101); s = linspace(c,d,101);
u = linspace(a,b, n+1); v = linspace(c,d,m+1);

subplot(1,2,1)
%  fill([a a+.01*(b-a), a+.01*(b-a), a], [c,c,d,d],'r')
  plot([a a], [c d], 'r')
  hold on
  for j = 2:n+1
     plot(u(j) +0*s, s,'k')
     hold on
  end
 % fill([a b b a], [c c c+.015*(d-c), c+.015*(d-c)], 'b')
  plot([a b], [c c], 'b')
  for i = 2:m+1
     plot(t, v(i)+0*t, 'k')
  end
  hold off
```

```
  xlabel( 'u' )
  ylabel( 'v' )
  axis equal

subplot(1,2,2)

  plot(feval(f, u(1),s), feval(g,u(1),s), 'r')
  hold on

  for j = 2:n+1
    plot(feval(f,u(j), s), feval(g,u(j), s), 'k')
    hold on
  end
  plot(feval(f, t,v(1)), feval(g,t,v(1)), 'b')
  for i = 2:m+1
    plot(feval(f, t,v(i)), feval(g,t,v(i)), 'k')
    hold on
  end
  axis equal

  disp('Hit return para seguir ')
  pause
  A = 0;
  du = (b-a)/n; dv = (d-c)/m; h = 10^(-6);
  umid = linspace(a +.5*du, b - .5*du, n);
  vmid = linspace(c +.5*dv, d - .5*dv, m);
  hold on
  for i = 1:m
     for j = 1:n
        uu = umid(j); vv = vmid(i);
        fu = .5*(feval(f,uu+h, vv) - feval(f, uu-h, vv))/h;
        fv = .5*(feval(f,uu,vv+h) - feval(f, uu, vv-h))/h;
        gu = .5*(feval(g,uu+h,vv)  - feval(g, uu-h,vv))/h;
        gv = .5*(feval(g,uu, vv+h) - feval(g, uu, vv-h))/h;
        J = [fu, fv; gu, gv];
        p = J*[du/2,0]';  q = J*[0 dv/2]';

        xplot = [p(1)+q(1), -p(1)+q(1), -p(1)-q(1), p(1)-q(1)] +feval(f,uu,vv);
        yplot = [p(2)+q(2), -p(2)+q(2), -p(2)-q(2), p(2)-q(2)] +feval(g,uu,vv);
        fill(xplot, yplot, 'g')

        A = A + abs(det(J))*du*dv;
     end
  end
  xlabel('x')
  ylabel('y')
  axis equal
  hold off
  A
```

tsurf

```
%  [X,Y] es una malla de puntos en el rectángulo R. Las alturas de
%  la superficie en los puntos de malla se dan en la matriz Z.
%  Frecuentemente Z se calculará calculada por la función Z = f(X,Y).
```

```
%  El llamado tsurf(X,Y,Z) calcula el área de la superficie consistiendo
%  de triangulos que cubren los puntos (xj,yi,f(xj,yi)).
%  la superficie consistente en triángulos también se grafica

function  out = tsurf(X,Y,Z)

    n = size(X,2)-1;  m = size(Y,1)-1;

    a = X(1,1); b = X(1,n+1); c = Y(1,1); d = Y(m+1,1);

    dx = (b-a)/n; dy = (d-c)/m;
    A = 0;

    for i = 1:m
       for j = 1:n
          x1 = X(i,j); x2 = x1+dx;
          y1 = Y(i,j); y2 = y1+dy;
          z1 = Z(i,j); z2 = Z(i,j+1);
          z3 = Z(i+1, j+1); z4 = Z(i+1,j);
          xedge = [x1 x2 x1];
          yedge = [y1 y1 y2];
          zedge = [z1 z2 z4];
          fill3(xedge,yedge, zedge, 'b')
          hold on
          xedge = [x2 x2 x1];
          yedge = [y1 y2 y2];
          zedge = [z2 z3 z4];
          fill3(xedge, yedge, zedge, 'c')
          dAlower = .5*sqrt(dx^2*dy^2 + (z2-z1)^2*dy^2 + (z4-z1)^2*dx^2);
          dAupper = .5*sqrt(dx^2*dy^2 + (z4-z3)^2*dy^2 + (z2-z3)^2*dx^2);
       A = A + dAlower + dAupper;
       end
     end
     out= A;

     hold off
```

wdot

```
% requiere  u.m y v.m para las componentes de la velocidad.

function z = wdot(t,w)

   z = [u(w(1), w(2)); v(w(1), w(2)) ] ;
```

xslice

```
% corta el gráfico de una función dada con un plano paralelo
%  al plano yz . se supone la función graficada sobre un rectángulo
%  [a,b]  [c,d].
% se llama is xslice('f',x,y0) para f en archivo m, y xslice(f, x,y0) para
% f inline, x es un vector de coordenadas x de a –b;y0 es el valor fijado
% de y.

function out = xslice(f, x, y0)
```

```
    z = feval(f,x,y0);
    high = max([z+1,0]);
    low = min([z-1,0]);
    a = min(x);
    b = max(x);

    xedge = [a b b a a];
    yedge = [y0 y0 y0 y0 y0];
    zedge = [low low high high low];

    fill3(xedge, yedge, zedge,'r')
```

yslice

```
% Corta el gráfico de f con un plano paralelo al plano xz
%  Se supone que la function se graficó sobre un rectángulo
% [a,b],[c,d].  Se llama is yslice(f,x0,y) para f inline, y yslice
%  ('f',x0,y) para f como archive m, x0 es el valor cte. de x,  y
%  es el vector de coordenadas y que va de c a d.

function out = yslice(f, x0, y)

  z = feval(f, x0, y);
  high = max([z+1 0]);
  low =  min([z-1,0]);

  c = min(y);  d = max(y);

  xedge = [x0 x0 x0 x0 x0];
  yedge = [c d d c c];
  zedge = [low low high high low];

  fill3(xedge, yedge, zedge, 'b')
```

lint

```
%    estima la integral de línea de un campo vectorial bidimensional %  F = [u(x,y), v(x,y)] a lo
%    largo trayectoria C. Se llama
%    lint(u,v,esquinas) con u y v funciones de  x,y  en archivos m o
%    inline.  esquinas = [a b c d], vector coordenadas del rectángulo R
%     Luego se pide el número de segmentos en la trayectoria
%    Se cliquea con botón izquierdo sobre la figura
%    hasta completar el número de segmentos. el valor de la integral de
%    línea sobre cada segmento se muestra en pantalla
%    El programa también estima la integral de xdy en C, cuando C es un
%    polígono cerrado, la integral de línea es el área de este polígono

 function out = lint(u,v,esquinas)

   a = esquinas(1); b = esquinas (2); c = esquinas (3); d = esquinas (4);
   x = linspace(a,b,11);
   y = linspace(c,d,11);

   [X,Y] = meshgrid(x,y);
```

```
U = feval(u,X,Y);
V = feval(v,X,Y);
quiver(X,Y,U,V)
axis image
hold on

N = input('entre el número de segmentos de la trayectoria  ')
linesum = 0;
A = 0;

[x0,y0] = ginput(1);

n = 50;
simpvec = 2*ones(1,n+1);
simpvec(2:2:n) = 4*ones(1,n/2);
simpvec(1) = 1; simpvec(n+1) = 1;
for j = 1:N
  [x1, y1] = ginput(1);
  plot([x0,x1], [y0,y1], 'r')

  xm = .4*x0 + .6*x1; ym = .4*y0 + .6*y1;
  d = norm([x1-x0, y1-y0]);
  l = .025*max([b-a, d-c]);
  delx = (l/d)*(x1-x0); dely = (l/d)*(y1-y0);
  xedge = [xm, xm-delx-dely, xm-delx+dely];
  yedge = [ym, ym-dely+delx, ym-dely-delx];
  fill(xedge, yedge, 'r')
  segment_number = num2str(j);
  text(x1,y1, segment_number)

  dx = (x1-x0)/n; dy = (y1-y0)/n;
  xx = linspace(x0, x1,n+1);
  yy = linspace(y0 ,y1,n+1);
  Fdx = simpvec*feval(u,xx,yy)'*dx/3;
  Fdy = simpvec*feval(v,xx,yy)'*dy/3;
  line_integral = Fdx + Fdy
  linesum = linesum + line_integral;
  dA = .5*(x1+x0)*(y1-y0);
  A = A +dA;
  x0 = x1; y0 = y1;
end

area = A
total_line_integral = linesum
hold off
```

Impl

```
function out = impl(f, esquinas,c)

% Esta función de gráfico muestra una superficie de nivel f(x,y,z) = c .

% Se llama impl(f,esquinas, c) si f está como inline,e
%impl('f',esquinas,c) si f está como archivo m. 'corners'
% es un vector de 6 componentes [xmin, xmax, ymin, ymax, zmin, zmax]
```

```
% %que define el domino en el que se muestra la superficie de nivel.
% 'c' es su valor.
%  Los max y min de la función sobre este dominio son estimados, si el
%valor c no cae en este rango, el programa se

xmin = esquinas (1); xmax = esquinas (2);  ymin = esquinas (3); ymax = esquinas (4);zmin = esquinas
(5); zmax = esquinas (6);

x = linspace(xmin, xmax ,21); y = linspace(ymin,ymax,21);
z = linspace(zmin,zmax,21);
[XX,YY,ZZ] = meshgrid(x,y,z);
W = feval(f, XX,YY,ZZ);
m = min(min(min(W)));
M = max(max(max(W)));
sprintf('El max sobre este dominio es %5.5f%', M)
sprintf('El mínimo sobre este dominio es %5.5f%', m)

if c < m | c > M
sprintf('En este dominio, la function no toma el valor%5.5f%', c)
else

[X,Y] = meshgrid(x,y);
dz = (zmax-zmin)/40;
  for zz = zmin:dz:zmax
    Z = feval(f,X,Y,zz);
    con = contours(X,Y,Z, [c,c]);
    nn = size(con,2);
    if nn > 0
      j = 1;
      while j < nn
        npairs = con(2,j);
        xdata = con(1, j+1: j+npairs);
        ydata = con(2, j+1: j+npairs);
        plot3(xdata, ydata, zz+ 0*xdata)
        j = j+npairs +1;
        hold on
      end
    end
  end
end
axis(esquinas)
xlabel('x')
ylabel('y')
zlabel('z')
hold off
```