Prácticas con Python

Jose Salvador Cánovas Peña.
Departamento de Matemática Aplicada y Estadística.

Índice general

Capítulo 1

Prácticas con Python

1.1. Operaciones básicas

Python permite realizar operaciones de cálculo básicas con números reales. Dado dos números reales a y b, la siguiente tabla muestra como hacerlo

a+b	suma
a-b	resta
a*b	producto
a/b	división
a//b	proporciona la parte entera del cociente
a%b	proporciona el resto de la división entera
a**b	a^b

El orden a la hora de hacer las operaciones es la usual. Por ejemplo, 2*2+3 realizará primero la multiplicación y luego la suma. Los paréntesis también se usan de la forma usual, con lo que $2 * (2 + 3)$ realizará primero la suma y luego el producto. La potencia tiene prevalencia frente a la división, por ejemplo 2**3//2 dará 4 mientras que 2**(3//2) proporcionará el resultado 2.

Para obtener la aproximación de un número real a con n cifras tenemos la expresión round(a, n). Por ejemplo, round(16,555, 2) = 16,55 y round(16,556, 2) = 16,56.

Actividad 1 *Realizar las siguientes operaciones con Python:*

(a) $(2^4 + 3)^2$ *(b)* $\frac{2+4^4}{1+\frac{2}{4\cdot 3^3}}$ *(c)* $\left(\frac{4+4^3}{2} + 5^2\right)^6$

(d) $1+2\frac{7}{2^4+5}$ *(e)* $(2 + 3^2 + 5^3)^{1/3}$ *(f)* $\left(1 + 2^3\frac{5}{2^4+1}\right)^{1/2}$

Actividad 2 *Obtener el resto y el cociente de las siguientes divisiones enteras:*

(a) 45 entre 3 *(b) 111 entre 67*
(c) 99 entre 54 *(d) 103964 entre 78*

1.1.1. Definición de constantes

A continuación, una reglas básicas para la definición de constantes y variables en Python.

Para asignar un valor a una variable tenemos el =. Por ejemplo $a = 10$ asignará a a el valor 10. Así, tecleando a=11, b=2 y luego a//b proporciona la salida 5.

Por defecto, Python asigna al símbolo "_" el valor de la última operación realizada. Si después de hacer la operación a=11, b=2, a//b tecleamos _ obtendremos el valor 5.

Si se utiliza una variable no definida Python enviará un mensaje de error.

Se pueden asignar varias constantes a la vez, por ejemplo a,b,c=1,3,4 asignará a a el valor 1, a b el 3 y a c el 4.

Para quitar el valor a una variable o constante a teclearemos del(a).

1.1.2. Funciones y constantes predefinidas en Python

Python tiene predefinidas una serie de funciones que pueden usarse sin tener que importarlas desde ningún módulo o paquete. Enumeramos las más interesantes para nosotros.

- abs(x). Proporciona el valor absoluto de x.
- divmod(a,b). Para dos números enteros a y b retorna el cociente y el resto de la división entera de a entre b.
- float(x). Proporciona la expresión decimal de x.
- len(s). Retorna el tamaño (el número de elementos) de una lista u objeto s.
- min(a,b,...) y max(a,b,...). Retornan el máximo y el mínimo de una lista a,b,....
- pow(x,y). Se trata de x^y. Equivalente a x**y.
- round(a,n). Proporciona a con n cifras decimales.
- sum(lista). Suma los elementos de un lista.

Respecto a las constantes, tenemos las siguientes:

- True y False. Son constantes de verdadero y falso.
- None. Se usa para indicar que ausencia de valor.

1.1.3. Operaciones lógicas y comparaciones

Las operaciones lógicas son las siguientes

x or y	x | y	$x \vee y$
x and y	x & y	$x \wedge y$
not x	$\sim x$	$\neg x$

En cuanto a las comparaciones tenemos las siguientes

x<y	x menor que y
x>y	x mayor que y
x<=y	x menor o igual que y
x>=y	x mayor o igual que y
x==y	x igual a y
x!=y	x distinto de y

Estas operaciones lógicas y comparaciones se pueden utilizar a la hora de realizar programas en Python, como en los ejemplos anteriores.

1.1.4. Listas o conjuntos

Las listas o conjuntos en Python se construyen utilizando corchetes y separando los elementos de la lista con comas. En la lista definida cada elemento tiene un lugar asignado según el orden en que esté escrito. Dicho orden empieza por 0. Por ejemplo, definimos la lista

$$\text{primos} = [2, 3, 5, 7, 11]$$

con los cinco primeros números primos. Si tecleamos primos[0] obtendremos el valor 2, y el último 11 será primos[4]. Además, los elementos también se pueden llamar con índices negativos. Así, primos[-1]=11 y primos[-5]=2. Los índices fuera del rango de valores que va desde -5 a 4 proporcionará error. Nótese que el rango de índices depende del número de elementos de la lista.

A continuación vamos a dar una serie de sentencias o instrucciones para manipular listas.

- Para extraer un subconjunto tenemos ":". Dada la lista de nombre lista y $n > m > 0$, la sentencia lista[m:n] será una nueva lista que contendrá todos los elementos que van de lista[n] a lista[m-1]. Así, primos[2:3] dará la salida [5] y primos[1:4] proporciona la salida [3,5,7]. Si tecleamos lista[m:n] dará la lista vacía [].

- Si se teclea lista[n:] tendremos todos los elementos de la lista a partir de lista[n]. Tecleando lista[:m] dará todos los elementos de la lista desde el primero a lista[m-1]. En el ejemplo, primos[2:] dará la salida [5,7,11] y primos[:2] es [2,3].

- Las listas se pueden manipular tanto con índice negativos como con positivos. En el ejemplo, primos[-3:-1] da la misma salida que primos[3:5]. Además primos[-3:5] da la misma salida que primos[2:5] ya que primos[-3]=primos[2]. No obstante, es recomendable no mezclar índices positivos con negativos.

- Si tenemos dos listas de nombres lista1 y lista2, se pueden concatenar usando el símbolo "+". Si definimos la lista

  $$\text{cuad} = [1, 4, 9, 16]$$

  con los cuatro primeros cuadradados, podemos definir

  $$\text{union} = \text{primos} + \text{cuad}$$

como la lista

$$[2, 3, 5, 7, 11, 1, 4, 9, 16].$$

- Si queremos reemplazar un elemento de la lista en la posición n, basta escribir lista[n] con el valor deseado. Por ejemplo, primos[1]=4 modificaría la lista de primos que sería ahora [2,4,5,7,11]. Podemos reemplazar varios elementos consecutivos de la lista utilizando :. Si tecleamos primos[1:4]=[3,7,5] el nuevo conjunto de primos será [2,3,7,5,11]. Aquí podríamos poner más elementos de los que contiene la lista inicial. Por ejemplo, en la lista original de primos [2,3,5,7,11], si tecleamos primos[1:3]=[1,4,8] obtenemos la nueva lista [2,1,4,8,7,11], es decir, con un elemento más. Esta operación también se puede utilizar para eliminar elementos de la lista tecleando despúes del igual el conjunto vacío []. Por ejemplo, en la lista original de primos [2,3,5,7,11], tecleando primos[1:3]=[] eliminamos 3 y 5 y tendríamos la nueva lista [2,7,11]. Si hubiésemos tecleado primos[1:]=[] los eliminamos todos desde el 3, siendo la nueva lista [2].

- Para añadir elementos a una lista tenemos la sentencia append. Tecleando lista.append(a) añadimos a al último lugar de la lista. Por ejemplo, tecleando cuad.append(25) añadimos 25 a la lista cuad, que ahora será [1,4,9,16,25].

- Para obtener el número de elementos de una lista tenemos la sentencia len(lista). Por ejemplo, tecleando len(cuad) obtenemos el resultado 5.

- Es posible construir listas cuyos elementos son listas. Por ejemplo, definimos las listas primos=[2,3,5,7,11] y cuad=[1,4,9,16] y la lista conj=[primos,union]. Se tiene que conj será

  $$[[2, 3, 5, 7, 11], [1, 4, 9, 16]]$$

  y su tamaño será de 2 elementos. conj[0] es [2,3,5,7,11] y conj[1] es [1,4,9,16]. Si queremos obtener el elemento 1, debemos teclear conj[1][0]. La manipulación con este tipo de lista se hace como siguendo los casos anteriores. Por ejemplo, conj[0][2:4] es la lista [5,7].

Actividad 3 *Dadas las listas A de los 10 primeros números naturales pares y B de los 5 primeros múltiplos de 3, hacer las siguientes operaciones:*

1. *Hacer la unión de A y B. Llamar C a esta nueva lista.*

2. *Eliminar los elementos repetidos de C eliminado el elemento repetido que aparece en primer lugar.*

3. *Añadir a la lista resultante los números 5 y 7 al final de la lista.*

4. *Añadir a la lista resultante los números 3,4 y 5 al primcipio de la lista.*

5. *Eliminar los elementos repetidos de C eliminado el elemento repetido que aparece en último lugar.*

6. *Crear una nueva lista D con los elementos pares de C, sin escribir el número en cuestión, sino seleccionándolo de la lista C.*

1.1.5. Elementos básicos de programación

Describimos aquí elementos básicos de la programación en Python que son análagos a otros lenguajes de programación.

- print. La senetencia print(argumento) escribirá en pantalla el argumento que le pongamos. Por ejemplo, print(7) escribirá el número 7. Para escribir un texto lo tenemos que escribir entre el símbolo '. Por ejemplo, print('Python es rollero') escribirá Python es rollero en pantalla. Para escribir varios argumentos a la vez hemos de separarlos por comas. Por ejemplo, print('Python es rollero',7).

- while. La sentencia while ejecutará una expresión mientras se den las condiciones lógicas para ello. Vamos a verlo con un ejemplo obteniendo los elementos de la sucesión de Fibonacci menores de 100. La sucesión de Fibonacci x_n se construye con los valores iniciales $x_0 = x_1 = 1$ y la recurrencia $x_{n+2} = x_n + x_{n+1}$. Esto es $x_n = (1, 1, 2, 3, 5, 8, ...)$. Con el siguiente programa obtenemos los elementos de la sucesión menores que 20. Primero introducimos a,b=1,1 y posteriormente tecleamos

```
while a<20:
    print(a)
    a,b=b,a+b
```

  que escribirá en columna los números. Aquí los : se ponen al final de cada sentencia y se ejecutará todo lo que haya a continuación que esté con al menos un espacio a la derecha. Es importante dejar este espacio porque si no tendremos un error, como por ejemplo si lo que escribimos es

```
while a<20:
print(a)
a,b=b,a+b
```

- if. La sentencia if es un condicional que puede ir acompañado por else y elif. Veamos con un ejemplo. En primer lugar introducimos $x = 40$. A continuación escribimos el programa

```
if x<10:
    print(1)
```

  el programa no devolverá salida porque x es menor que 10. El programa

```
if x<10:
    print(1)
else:
    print(2)
```

  devolverá ahora 2, escribiéndolo en pantalla. Ahora el programa

```
if x<10:
    print(1)
elif x<20:
    print(2)
else:
    print(3)
```

devolverá la salida 3 al ser x menor que 10 y 20. La sentencia elif es abreviatura de else if.

- for. Esta sentencia es un poco diferente a la que podemos encontrar en otros lenguajes de programación ya que va asociada a un conjunto o lista. Veamos cómo se emplea. Creamos la lista primos=[2,3,5,7]. Escribimos el programa

```
for i in primos:
    print(i)
```

  que devolverá como salida los números primos 2, 3, 5 y 7 en forma de columna. Nótese que for siempre va a ir asociada con la sentencia in.

- range. La sentencia range sirve para construir listas de números naturales para ser utilizadas por for. Así, range(10) creará un conjunto con los números del 0 al 9. Si queremos empezar por un valor distinto de 0, por ejemplo 5, tecleamos range(5,9) que creará un conjunto del 5 al 9. Si deseamos crear un conjunto que empiece por 5, con cuatro elementos y vaya añadiendo naturales de dos en dos, tecleamos range(5,9,2) que dará la lista con los números 5, 7 y 9. Combinado con for nos permite escribir el siguiente programa

```
for i in range(3):
    print(i)
```

  escribirá los números 0, 1 y 2. El programa

```
for i in range(3,6):
    print(i)
```

  escribirá los números 3, 4 y 5. El programa

```
for i in range(3,10,3):
    print(i)
```

  escrbirá los números 3, 6 y 9.

  Un observación sobre esta sentencia es que no es una lista, pero podemos usarla para crear éstas con la sentencia list. Por ejemplo, pares=list(range(2,10,2)) introducirá la variable pares como la lista [2,4,6,8]. Si simplemente escribimos pares=range(2,10,2) y tecleamos pares, nos devolverá range(2,10,2).

- break, else y continue en bucles. La sentencia break se usa en un bucle para terminar de ejecutar el mismo antes de que se llegue al final. Por ejemplo en el programa

```
for i in range(1,10):
    if i>5:
        break
    else:
        print(i)
```

escribirá los números de 1 al 5 ya que termina el bucle si i es mayor que 5. Nótese que el else está asociado al if. La sentencia else se puede asociar al for como en el programa siguiente:

```
for i in range(1,10):
    a=i+1:
else:
    print(a)
```

que devolverá en pantalla 10, esl resultado de la última suma i+1. La sentencia continue produce la continuación del bucle después de comprobar una condición y proporcionar una salida. Por ejemplo el programa

```
for i in range(2,4):
  if i % 2==0:
        print(i,' es par')
        continue
   print(i,' es impar')
```

da la salida 2 es par, 3 es impar. Nótese que el programa

```
for i in range(2,4):
   if i % 2==0:
        print(i,' es par')
   else:
        print(i,' es impar')
```

produce el mismo resultado.

Actividad 4 *Dada la función* $f(x) = 3{,}95x(1-x)$ *y* $x_0 = 0{,}5$*, obtener los 100 primeros elementos de la recursión*

$$x_{n+1} = f(x_n).$$

Actividad 5 *Dada la recusión de la actividad 4, obtener los 100 primeros elementos pares, es decir, los de la sucesión* $x_0, x_2, x_4, x_6, ...$

Actividad 6 *Dada la recusión de la actividad 4, obtener los 100 primeros elementos multiplos de 4, es decir, los de la sucesión* $x_0, x_4, x_8, x_{12}, ...$

Actividad 7 *Dada la función* $f(x) = 3{,}95x(1-x)$ *y* $x_0 = 0{,}5$*,* $x_1 = 0{,}25$*, obtener los 100 primeros elementos de la recursión*

$$x_{n+1} = 025 \cdot x_{n-1} + 0{,}75 \cdot f(x_n).$$

Actividad 8 *Dados los elementos obtenidos en las activiadades 4 y 7, obtener una lista que resulte de multiplicar los elementos de las dos lista dos a dos.*

1.1.6. Definición de funciones

Para definir funciones en Python tenemos la sentencia def, junto con return. Si queremos definir por ejemplo la función $f(x) = x^2$ tecleamos

```
def f(x):
    return x**2
```

Si tenemos más variables, por ejemplo $f(x, y) = x \cdot y$, tecleamos

```
def f(x,y):
    return x*y
```

Las funciones pueden tener definiciones más sofisticadas. Por ejemplo, vamos a definir una función que nos de los n primeros elementos de la sucesión de Fibonacci. Para ello escribimos

```
def fib(n):
    seq=[]
    a,b=1,1
    for i in range(n):
        seq.append(a)
        a,b=b,a+b
    return seq
```

Actividad 9 *Definir las funciones siguientes*

$$f_1(x) = 3x^2 + x - 1 \qquad f_2(x) = \tfrac{2x+1}{x^2+1}$$

$$f_3(x) = \begin{cases} 2x & si \quad x \leq 0, \\ x^2 & si \quad x > 0. \end{cases} \qquad f_4(x) = \begin{cases} \frac{2x}{x+1} & si \quad 0 < x \leq -2, \\ x^2 + 3 & si \quad x > -2. \end{cases}$$

$$f_5(x) = \begin{cases} 2x & si \quad x \leq 0, \\ x^2 & si \quad 0 < x < 2, \\ x^3 + 1 & si \quad x \geq 2. \end{cases} \qquad f_5(x) = \begin{cases} \frac{2x+1}{x^2} & si \quad x \leq -1, \\ x^2 & si \quad 0 < x < 2, \\ 0 & si \quad x \geq 3. \end{cases}$$

Actividad 10 *Definir las funciones siguientes*

$$f_1(x, y) = xy^2 \qquad f_2(x, y) = \tfrac{x+y^2}{x-y}$$

$$f_3(x, y, z) = xy^2 + zy^3 \qquad f_4(x, y, z, t) = x^2 + y^2 - z^{t-x}$$

$$f_5(x, y) = \begin{cases} 2xy & si \quad xy \leq 0, \\ xy^2 & si \quad xy > 0. \end{cases} \qquad f_6(x, y) = \begin{cases} 2x^y & si \quad x + y^2 \leq 1, \\ x^{y^2} & si \quad x + y^2 > 1. \end{cases}$$

1.2. Más sobre listas y conjuntos

1.2.1. Listas

Describimos en este apartado diversas sentecias para manipular listas. Supongamos que tenemos una lista llamada lista y enumeramos las diferentes sentencias que se le pueden aplicar tal y como se escriben en Python.

- lista.append(x). Añade a lista el elemento x en la última posición.
- lista.insert(i, x). Añade a lista el elemento x en la posición i.
- lista.remove(x). Elimina el primer elemento de la lista cuyo valor sea x.
- lista.pop(i). Elimina el elemento de la lista de la posición i. Si no se indica posición, es decir, lista.pop( ) elimina el último elemento de la lista.
- lista.clear(). Elimina todos los elementos de la lista.
- lista.index(x). Retorna el índice del primer elemento cuyo valor sea igual a x.
- lista.count(x). Retorna el número de veces que x aparece en la lista.
- lista.reverse(). Invierte los elementos de la lista.
- list.sort(). Ordena los elementos de la lista de menor a mayor.
- Podemos utilizar del para borrar los elementos de una lista. Escribiendo del lista[i] eliminamos el elemento i, del lista[i:j] elimina los elementos del i al j-1. Así con todas las posibles formas de trabajar con los elementos de las listas.

Las listas largas pueden generarse fácilmente con bucles en su interior. Veamos un ejemplo. Supongamos que queremos crear una lista con lo 10 primeros cuadrados 1,4,9,...,81. El siguiente programa permite hacerlo

```
cuad=[]
for i in range(10):
    squares.append(i**2)
```

Es posible hacer esto más cómodamente de la siguiente manera:

```
cuad=[i**2 for i in range(10)]
```

Es posible incorporar condiciones como por ejemplo

```
[(x,y) for x in [1,2,3] for y in [3,1,4] if x!=y]
```

que crea la lista de los pares (x,y) tales que x es distinto de y.

Actividad 11 *Dadas las listas A de los 10 primeros números naturales impares y B de los 5 primeros múltiplos de 4, hacer las siguientes operaciones:*

1. *Insertar en A el número 10 en la posición 2 y llamar A a la lista resultante.*
2. *Eliminar de B el primer y último elemento y llamar B a la lista resultante.*
3. *Añadir a A los dos primeros elementos de B y llamar A a la lista resultante.*
4. *Definir C como la unión de B y A, por este orden.*
5. *Añadir a la lista resultante los números 3,4 y 5 al final de la lista.*

1.2.2. Conjuntos

Python también incluye un tipo de dato para conjuntos. Un conjunto es una colección no ordenada y sin elementos repetidos. Las llaves o la función set() pueden usarse para crear conjuntos. Para crear un conjunto vacío se usa set(). Por ejemplo, un conjunto

A={'a','b','c','d','e','e','b'}

será {'a','b','c','d','e'}, nótese que se eliminan elementos repetidos y que los elementos van entre '. Otro

B=set('caravana')

que será {'a','c','n','r','v'}.

Para comprobar que un elemento está en A preguntamos con in, y se devolverá el valor True si está y False si no lo esta. Por ejemplo 'a' in A retorna True y 'v' in A retorna False. Las operaciones son las siguientes:

A|B	$A \cup B$
A&B	$A \cap B$
A-B	$A \setminus B$
A^B	$A \triangle B$

En el ejemplo, A|B retorna {'a','b','c','d','e','n','r','v'}, A&B da {'a','c'}, A-B da por solución {'b','d','e'} y A^B proporciona el valor {'b','d','e','n','r','v'}.

Actividad 12 *Dados los conjuntos*

$$A = \{1, 2, 3, 4, 5\},$$

$$B = \{2, 4, 6, 8, 10, 12\}$$

y

$$C = \{1, 9, 4, 3, 2, 5, 11\},$$

obtener:

(a) $A \cap B \cup C$ *(b)* $B \setminus C \cup A$
(c) $(B \setminus C) \cup A$ *(d)* $(A \cup C) \triangle B$
(e) $A \cap (C \triangle B)$ *(f)* $(A \triangle B) \cup (B \setminus C)$

1.3. Creación de módulos

Es posible guardar nuestro trabajo en Python en un archivo con la extensión .py. En él podemos guardar nuestros programas y funciones definidas y llamarlas desde Python cada vez que las necesitemos. Por ejemplo, vamos a definir 3 funciones en Python

$$f_1(x) = x^2,$$

$$f_2(x) = 2x$$

y

$$f_3(x) = x^2 + 2x + 1.$$

Como sabemos, estás funciones se definen en Python como

```
def f1(x):
    return x**2
```

```
def f2(x):
    return 2*x
```

y

```
def f3(x):
    return x**2+2*x+1
```

Vamos a crear el fichero mifun.py con un editor de textos plano incluyendo las tres funciones. Ahora podemos importar el fichero desde la carpeta donde lo hemos guardado tecleando import mifun. Una vez cargado, podemos usar cualquiera de las tres funciones del siguiente modo. Por ejemplo, para calcular $f_3(10)$ tecleamos

```
mifun.f3(10)
```

y obtendremos el valor 100.

No es esta la única manera de importar las funciones que hemos predefinido. Pordemos usar la expresión

```
from mifun import f3
```

Ahora, para obtener el valor $f_3(10)$ basta teclear f3(10) para obtener el 100. Podemos importar todas las funciones escribiendo

```
from mifun import *
```

Python tiene predefinidos módulos con funciones y programas que podemos utilizar. El funcionamiento es análogo aunque no es necesario buscar la ruta donde el fichero está guardado. Una vez que se ha importado un módulo, podemos ver qué funciones tiene definidas tecleando dir(modulo).

Todos los módulos se pueden cargar y usar con un nombre diferente usando la sentencia as. Por ejemplo, podemos cargar el módulo creado tecleando

```
import mifun as func
```

y ahora sólo hemos de teclear

```
func.f3(10)
```

para obtener su valor 100. De igual forma podemos importar como

```
from mifun import f3 as f
```

y ahora basta escribir f(10) para obtener su valor 100.

Actividad 13 *Crear un módulo llamado fun1var.py con las funciones definidas en la actividad 9.*

Actividad 14 *Crear un módulo llamado fun2var.py con las funciones definidas en la actividad 10.*

1.4. Módulo math

En este módulo están definidas la mayoría de las funciones matemáticas usuales. A continuación describimos algunas de las funciones que más se van a utilizar.

1.4.1. Teoría de números

Las funciones para la teoría de números son las siguientes

ceil(x)	menor entero mayor o igual que x
floor(x)	mayor entero menor o igual que x
gcd(n,m,...)	máximo común divisor de un lista de enteros n,m,...
lcm(n,m,...)	mínimo común múltiplo de un lista de enteros n,m,...
prod(lista)	multiplica los elementos de una lista
fsum(lista)	suma los elementos de una lista
remainder(x, y)	resto de la división entera de x entre y

Para combinatoria tenemos

comb(n,k)	combinaciones de n elementos de k en k
factorial(n)	factorial de n
perm(n, k)	variaciones de n elementos de k en k
perm(n)	permutaciones de n elementos.

1.4.2. Funciones logarítmicas y exponenciales

Las funciones logarítmicas y exponenciales son las siguientes

exp(x)	función exponencial e^x
log(x)	logaritmo neperiano de x
log(x,b)	logaritmo en base b de x
sqrt(x)	$\sqrt{x}$

1.4.3. Funciones trigonométricas

Las funciones trigonométricas son las siguientes

sin(x)	seno de x en radianes
cos(x)	coseno de x en radianes
tan(x)	tangente de x en radianes
asin(x)	arcoseno de x
acos(x)	arcocoseno de x
atan(x)	arcotangente de x
degrees(x)	convierte x de radianes a grados
radians(x)	convierte x de grados a radianes

1.4.4. Constantes

Las constantes predefinidas en Python son las siguientes:

pi	π
e	número e$\sim$2.718281828459045
inf	infinito
nan	No un número.

1.5. Módulo sympy

Este módulo permite trabajar con Pyhton en modo simbólico, de forma que se puede trabajar de igual modo que con programas de cálculo simbólico como maxima o mathematica. Contiene todas las funciones definidas en math, aunque en algunos casos funciones con el mismo nombre dan lugar a resultados diferentes, es dcir, no es en realidad la misma función. El ejemplo paradigmatico es sqrt. Si tecleamos

```
from math import sqrt
sqrt(2)
```

da la salida 1.4142135623730951 que es una aproximación del valor $\sqrt{2}$. Si tecleamos ahora

```
from sympy import sqrt
sqrt(2)
```

da la salida $\sqrt{2}$, que es el valor real sin aproximaciones. Así, el módulo sympy permite trabajar en modo simbólico o precisión infinita, y permite hacer gran número de cálculos complejos como derivadas, integrales, resolver ecuaciones, dibujar funciones, etcétera. Vamos a ver en estas notas cómo hacer estas operaciones.

De hecho, este módulo permite trabajar con variables que no están asignadas a números con la sentencia symbols. Por ejemplo, tecleamos

```
from sympy import symbols
x, y= symbols('x y')
expr=x+2*y
expr-1+x
```

da la salida final $2x + 2y - 1$. Es decir, se puede trabajar en forma simbólica y hacer diferentes operaciones. Veremos que es posible hacer mucho más.

1.5.1. Lógica

Como sabemos, True es la expresión para verdadero y False los es para falso. Son las proposiciones verdadera y falsa. Recordemos las sentencias lógicas de Python

x or y	x | y	$x \vee y$
x and y	x & y	$x \wedge y$
not x	$\sim x$	$\neg x$

Sin embargo, or, and y not debemos usarlos al programar, mientras que para ejercicios de lógica usaremos |, & y ~.

Veamos que otras sentencias lógicas podemos encontrar en este módulo. Dadas las proposiciones p y q, la implicación p implica q se escribe p>>q o q<<p. Con el siguiente texto se puede construir la tabla de verdad de $p \to q$.

```
from sympy import symbols
p,q=symbols('p q')
(p>>q).subs({p:True, q:True })
True
(p>>q).subs({p:True, q:False })
False
(p>>q).subs({p:False, q:True })
True
(p>>q).subs({p:False, q:False })
True
```

Aquí, hemos escrito en negrita las salidas que proporciona Python. La sentencia de Python subs() sirve para asignar valores a variables que a continuación son utilizados para verificar la operación a la izquierda.

Actividad 15 *Construir con Python las tablas de verdad de* $p \vee q$, $p \wedge q$, *y* $p \leftrightarrow q$.

Actividad 16 *Comprobar que* $p \vee \neg(p \wedge q)$ *es una tautología.*

Para ver si dos proposiciones son lógicamente equivalentes podemos usar la sentencia de Python equals() que permite comparar dos expresiones *que deben tener las mismas variables.* Por ejemplo,

```
from sympy import symbols
p,q=symbols('p q')
(p >> q).equals(~q >> ~p)
True
(p|False).equals(p)
True
(p&False).equals(p)
False
```

Debemos decir aquí que s tecleamos (p).equals(p&False) Python dará error. Las proposiciones neutras True y False deben usarse en la primera expresión, y no en la segunda.

Para trabajar con implicaciones lógicas más complicadas tenemos a nuestra disposición en sympy las sentencias And(), Not(), Or(). Por ejemplo

```
from sympy import symbols
p,q,r=symbols('p q r')
from sympy import And, Or, Not
Not(And(p, q, r)).equals(And(Not(p), Not(q), Not(r)))
False
Not(And(p, Not(p))).equals(Or(q, Not(q)))
False
```

Así, en el primer ejemplo comparamos la tabla de verdad de $p \to q$ con la de $(\neg q) \to (\neg p)$, concluimos que son iguales por lo que son equivalentes. Del mismo modo se concluye que no son equivalentes las proposiciones $\neg(p \wedge q \wedge r)$ y $(\neg p) \wedge (\neg q) \wedge (\neg r)$, y las proposiciones $\neg(p \wedge (\neg p))$ y $q \vee (\neg q)$.

Es importante darse cuenta de que al usar equals se tienen que tener las mismas variables en las dos fórmulas a comparar. Si no es así, el programa puede darnos un resultado erróneo. Por ejemplo, la proposición $(\neg p \wedge \neg q \wedge r) \vee (\neg p \wedge \neg q \wedge \neg r)$ es quivalente a $\neg p \wedge \neg q$. Sin embargo, si tecleamos

```
from sympy import symbols
p,q,r=symbols('p q r')
from sympy import And, Or, Not
Or(And(Not(p),Not(q),r),And(Not(p),Not(q),Not(r))).equals(And(Not(p),Not(q)))
False
```

encontramos que el programa nos dice erróneamente que no son equivalentes.

Actividad 17 *Demostrar que las proposiciones $\neg(p \wedge q)$ y $\neg p \vee \neg q$ son lógicamente equivalentes.*

Con And(), Not() y Or() se pueden construir las tablas de verdad de proposiciones. Por ejemplo

```
from sympy import And, Or, Not
And(True,True)
True
And(True,False)
False
And(False,True)
False
And(False,False)
False
```

construye la tabla de verdad de $p \wedge q$. Como permiten incluir más argumentos, es posible trabajar con proposiciones más complicadas. Por ejemplo, vamos a ver que $((p \to q) \wedge (q \to r))$ implica $(p \to r)$. Para ello tecleamos

```
from sympy import symbols
p,q,r=symbols('p q r')
from sympy import And, Not, Or
(And((p>>q),(q>>r)))>>(p>>r).subs({p:True, q:True, r:True})
True
```

y cambiando los valores de {p:True, q:True, r:True} incluyendo todas las combinaciones con True y False vemos que siempre se obtiene True de salida, por lo que se tratará de una tautología y la implicación es verdadera.

Al igual que los otros operadores lógicos, la implicación $\rightarrow$ también está definida en sympy. Se trata de la funcion Implies() y es equivalente a usar >>. Por ejemplo

```
from sympy import Implies, And, symbols
Implies(True,True)
True
p,q,r=symbols('p q r')
Implies(And(Implies(p,q),Implies(q,r)),Implies(p,r)).subs({p:False, q:False, r:False})
True
```

Similarmente, la doble implicación $\leftrightarrow$ se define como la función Equivalent(). Por ejemplo

```
from sympy import Equivalent
Equivalent(True,True)
True
```

Actividad 18 *Demostrar que el argumento* $\{p \rightarrow q, \neg p\}$ *implica* $\neg q$ *es una falacia.*

Actividad 19 *Determinar la validez del argumento* $\{p \rightarrow q, \neg p\}$ *implica* $\neg p$.

Actividad 20 *Demostrar que el argumento* $\{p \rightarrow \neg q, rq, r\}$ *implica* $\neg p$.

Finalmente, mostramos la sentencia simplify_logic(), que permite simplificar expresiones lógicas, pero que solo opera con los símbolos |, & y $\sim$. Veamos un ejemplo

```
from sympy import symbols, simplify_logic
p,q,r=symbols('p q r')
simplify_logic((~p & ~q & ~r) | ( ~p & ~q & r))
~p & ~q
```

Esto es $(\neg p \wedge \neg q \wedge \neg r) \vee (\neg p \wedge \neg q \wedge r)$ es equivalente a $\neg p \wedge \neg q$.

Actividad 21 *Demostrar que* $(p \vee q \wedge \neg r) \wedge (p \wedge q)$ *es equivalente a* $p \wedge q$.

Actividad 22 *Demostrar que* $(p \vee q \wedge \neg r) \wedge (p \vee q)$ *es equivalente a* $p \vee (q \wedge \neg r)$.

Actividad 23 *Demostrar que* $(p \vee q \wedge \neg r) \vee (p \vee q)$ *es equivalente a* $p \vee q$.

1.5.2. Conjuntos

Los conjuntos se pueden estudiar en Python como vimos anteriormente. No obstante, puede usarse la sentencia simplify_logic para estudiarlos en Python. Esto se debe a que el álgebra de los operadores lógicos $\vee$, $\wedge$ y $\neg$ es análoga al álgebra de conjuntos con $\cup$, $\cap$ y complementario. De ejemplo anterior

$$(\tilde{}p\&\tilde{}q\&\tilde{}r)|(\tilde{}p\&\tilde{}q\&r) \equiv \tilde{}p\&\tilde{}q$$

tiene su análogo en teoría de conjuntos

$$(A^c \cap B^c \cap C^c) \cup (A^c \cap B^c \cap C) = A^c \cap B^c,$$

y se puede obtener esta simplificación a partir de la obtenida usando las sentencias de lógica.

Actividad 24 *Simplificar los siguientes conjuntos*

(a) $(A \cap B \cup C^c) \cap (C \cup A)$	*(b)* $(A^c \cap B^c \cup C) \triangle (C \cup A)$
(c) $(A \cap B \cup C^c) \cap ((C \cup A) \setminus B)$	*(d)* $(A \cap B \cup C^c) \cap (C \cup A^c)$
(e) $(A \cap B \cup C^c) \cup (C \cup A)^c$	*(f)* $(A \cap B)^c \cap (C \cup A)^c \setminus (C \cup B)$

Nota: *Se debe usar que* $A \setminus B = A \cap B^c$ *y* $A \triangle B = (A \setminus B) \cup (B \setminus A)$.

1.5.3. Teoría de números

Uno de los temas centrales en la teoría de números tiene que ver con los números primos. Veamos como trata el módulo sympy este tema. En primer lugar tenemos la sentencia prime(n) que te da el n-ésimo primo en la lista de números primos. A modo de ejemplo, si tecleamos

```
from sympy import prime
prime(10)
29
prime(100000)
1299709
prime(1)
2
```

donde como siempre, en negrita se escriben las salidas proporcionadas por Python. Para comprobar si un número natural n es primo tenemos la sentencia isprime(n). Por ejemplo

```
from sympy import isprime
isprime(5)
True
isprime(25)
False
```

La sentencia primepi(n) nos permite saber cuántos primos hay menores o iguales que n. Por ejemplo

```
from sympy import primepi
primepi(25)
9
```

La sentencias prevprime(n) y nextprime(n) nos dan, respectivamente, el mayor número primo menor que n y el menor primo mayor que n. Por ejemplo

```
from sympy import prevprime, nextprime
prevprime(25)
23
nextprime(25)
29
```

La sentencia primerange(n) da todos los primos menores que n, mientras que primerange(n,m) da todos los primos mayores o iguales que n y menores que m, con n<m. La salida es de tipo range, por lo que para visualizar la solución hay que usar la sentencia list(). Por ejemplo

```
from sympy import primerange
list(primerange(25))
[2, 3, 5, 7, 11, 13, 17, 19, 23]
list(primerange(5,25))
[5, 7, 11, 13, 17, 19, 23]
```

Finalmente, la sentencia randprime(n,m) da un primo aleatorio entre n y m. A modo de ejemplo

```
from sympy import randprime
randprime(6, 25)
13
```

Los números naturales que no son primos se llaman compuestos. Al igual que los primos, buen número de ellos estan listados en el módulo y pueden ser llamados con la sentencia composite(n), como muestra el siguiente ejemplo

```
from sympy import composite
composite(36)
52
composite(1)
4
composite(17737)
20000
```

De forma similar al caso de los primos, tenemos la sentencia compositepi(n) que da el número de números compuestos menores o iguales que n. Por ejemplo

```
from sympy import compositepi
compositepi(24)
14
compositepi(23)
13
```

Para encontrar los divisores de un número natural n tenemos la sentencia divisors(n), y los divisores distintos de n se calculan con proper_divisors(n), como muestra el siguiente ejemplo

```
from sympy import divisors, proper_divisors
divisors(24)
[1, 2, 3, 4, 6, 8, 12, 24]
proper_divisors(24)
[1, 2, 3, 4, 6, 8, 12]
```

y para calcular en número de divisores de n tenemos divisor_count(n), y proper_divisor_count(n) para el número de divisores de n distintos de n. Como ejemplo

```
from sympy import divisor_count, proper_divisor_count
divisor_count(24)
8
proper_divisor_count(24)
7
```

Para factorizar un número natural n tenemos factorint(n). Por ejemplo

```
from sympy import factorint
factorint(153)
{3: 2, 17: 1}
```

donde la salida se lee que hay dos divisores, 3 con potencia 2 y 17 con potencia 1, esto es, $153 = 3^2 \cdot 17$. Recordemos que en el módulo math tenemos definido el máximo común divisor y mínimo común múltiplo, pero tambibén podemos importarlos desde el modulo sympy. Es decir, se puede hacer el ejemplo

```
from sympy import gcd, lcm
gcd(24,64)
8
lcm(24,64)
192
```

Actividad 25 *Calcular* $\gcd(215, 36)$ *y* $\gcd(334, 562)$. *Encontrar el mínimo común múltiplo de ambos pares de números y los* x_0 *e* y_0 *del Teorema de Bezout.*

Actividad 26 *Calcular* $\gcd(18, 256)$ *y* $\gcd(8316, 10920)$. *Encontrar el mínimo común múltiplo de ambos pares de números y los* x_0 *e* y_0 *del Teorema de Bezout.*

Actividad 27 *Enumerar todos los primos menores de 100.*

Actividad 28 *Establecer cuáles de las siguientes congruencias son verdaderas.*

1. $446 \equiv 278 (\mathrm{mod}\, 7)$.
2. $269 \equiv 413 (\mathrm{mod}\, 12)$.
3. $445 \equiv 536 (\mathrm{mod}\, 18)$.
4. $793 \equiv 682 (\mathrm{mod}\, 9)$.
5. $473 \equiv 369 (\mathrm{mod}\, 26)$.
6. $383 \equiv 126 (\mathrm{mod}\, 15)$.

La función de Euler es totient(n) para n natural. Así, tecleando

```
from sympy import totient
totient(1)
1
totient(5)
4
totient(26)
12
```

Actividad 29 *Encontrar* $\varphi(37)$, $\varphi(137)$, $\varphi(275)$, $\varphi(700)$ *y* $\varphi(201)$.

Actividad 30 *Encontrar* a^{-1} *en* $\mathbb{Z}_m$ *donde: a)* $a = 37$ *y* $m = 249$*; b)* $a = 15$ *y* $m = 234$.

Actividad 31 *Encontrar los inversos siguientes a partir de la función* φ *de Euler:*

1. 20^{-1} *en* $\mathbb{Z}_9$.
2. 7^{-1} *en* $\mathbb{Z}_{12}$.

Actividad 32 *Resolver las siguientes ecuaciones:*

1. $3x+1 \equiv 6 (\mathrm{mod}\, 7)$.
2. $2x+1 \equiv 3 (\mathrm{mod}\, 6)$.
3. $3x+2 \equiv 6 (\mathrm{mod}\, 8)$.
4. $23x+1 \equiv 6 (\mathrm{mod}\, 67)$.
5. $3x+18 \equiv 6 (\mathrm{mod}\, 1645)$.

También tenemos implementado el Teorema chino de los restos, con la sentencia crt(lista1,lista2), donde lista1 contiene los módulos y la lista2 los restos. Por ejemplo, si queremos encontrar x tal que

$$\begin{cases} x \equiv 49 (\mathrm{mod}\, 99), \\ x \equiv 76 (\mathrm{mod}\, 97), \\ x \equiv 65 (\mathrm{mod}\, 95), \end{cases}$$

tecleamos

```
from sympy.ntheory.modular import crt
crt([99, 97, 95], [49, 76, 65])
(639985, 912285)
```

cuyo resultado es $x = 639985$. Nótese que el módulo desde el que se importa es sympy.ntheory.modular, que es un paquete de sympy. Para comprobar que 639985 es la solución podemos testear

```
[639985 % m for m in [99, 97, 95]]
[49, 76, 65]
```

por lo que efectivamente, se trata de la solución. Nótese que $912285 = 99 \cdot 97 \cdot 95$.

Alternativamente, tenemos la sentencia solve_congruence(par1,par2,...) donde introducimos pares con el resto y el módulo correspondiente. Por ejemplo, para resolver

$$\begin{cases} x \equiv 2 (\mathrm{mod}\, 3), \\ x \equiv 3 (\mathrm{mod}\, 5), \\ x \equiv 2 (\mathrm{mod}\, 7), \end{cases}$$

tecleamos

```
from sympy.ntheory.modular import solve_congruence
solve_congruence((2, 3), (3, 5), (2, 7))
(23, 105)
```

cuyo resultado es $x = 23$. Para comprobar que se trata de la solución podemos teclear

```
[23 % m for m in [3, 5, 7]]
[2, 3, 2]
```

Nótese que $105 = 3 \cdot 5 \cdot 7$.

Actividad 33 *Encontrar soluciones enteras de las siguientes ecuaciones, cuando sea posible.*

1. $5x + 7y = 4$.
2. $6x + 24y = 21$.
3. $14x + 21y = 49$.

Actividad 34 *Resolver el siguiente sistema:*

$$\begin{cases} 2x \equiv 3 (\mathrm{mod}\, 7), \\ x \equiv 1 (\mathrm{mod}\, 9), \\ x \equiv 3 (\mathrm{mod}\, 8), \\ x \equiv 0 (\mathrm{mod}\, 11). \end{cases}$$

Podemos resolver congruencias del tipo

$$x^n \equiv a (\mathrm{mod}\, m)$$

donde $a, n, m \in \mathbb{N}$. Para ello, tenemos las sentencias sqrt_mod(par), donde par=(a,m), y nthroot_mod(triple), donde triple=(a,n,m). En ambos casos, se proporciona una solución en caso de existir. Para encontrarlas todas debemos añadir True después de introducir los datos. A modo de ejemplo, para resolver la ecuación

$$x^2 \equiv 11 (\mathrm{mod}\, 43)$$

tecleamos

```
from sympy.ntheory import sqrt_mod
sqrt_mod(11, 43)
21
sqrt_mod(11, 43, True)
[21,22]
```

Para resolver

$$x^n \equiv a(\mathrm{mod}\, m)$$

tecleamos

```
from sympy import nthroot_mod
nthroot_mod(11, 4, 19)
8
nthroot_mod(11, 4, 19, True)
[8,11]
```

Actividad 35 *Encontrar soluciones enteras de las siguientes ecuaciones, cuando sea posible.*

1. $x^2 \equiv 3(\mathrm{mod}\, 29)$.
2. $x^3 \equiv 3(\mathrm{mod}\, 21)$.
3. $x^5 \equiv 3(\mathrm{mod}\, 19)$.

Terminamos esta sección con una sentencia para estudiar los elementos del grupo $\mathbb{Z}_m$. Dado $n \in \mathbb{Z}_m$ se llama orden de n al menor número natural k tal que $n^k \equiv 1(\mathrm{mod}\, m)$. Para calcular el orden tenemos la sentencia n_order(par), donde par=(n,m). A modo de ejemplo

```
from sympy import n_order
n_order(3, 7)
6
n_order(4, 7)
3
```

por lo que en $\mathbb{Z}_7$ el orden de 3 es 6 y el de 4 es 3.

Actividad 36 *Encontrar el orden de todos los elementos de* $\mathbb{Z}_{15}$.

1.5.4. Criterios de divisibilidad

Uno de los temas más importantes de la aritmética es encontrar condiciones para que un número natural k divida a otro número n. Son los llamados criterios de divisibilidad. En general si expresamos el número n como

$$n = a_i \cdot 10^i + a_{i-1} \cdot 10^{i-1} + ... + a_1 \cdot 10 + a_0$$

se verifica que $k|n$ si la división de n por k es exacta, es decir, si

$$\begin{aligned} \frac{n}{k} &= \frac{a_i \cdot 10^i + a_{i-1} \cdot 10^{i-1} + ... + a_1 \cdot 10 + a_0}{k} \\ &= a_i \cdot \frac{10^i}{k} + a_{i-1} \cdot \frac{10^{i-1}}{k} + ... + a_1 \cdot \frac{10}{k} + a_0 \cdot \frac{1}{k} \end{aligned}$$

es un número entero. Calculamos los restos

$$\begin{cases} 10^i \equiv r_i (\mathrm{mod}\, k), \\ 10^{i-1} \equiv r_{i-1} (\mathrm{mod}\, k), \\ ..... \\ 10 \equiv r_1 (\mathrm{mod}\, k), \\ 1 \equiv r_0 (\mathrm{mod}\, k). \end{cases}$$

Entonces $k|n$ si y sólo si k divide a $a_i \cdot r_i + a_{i-1} \cdot r_{i-1} + ... + a_1 \cdot r_1 + a_0$. Se construyen así las reglas de divisibilidad que conocemos:

- $k = 2$. Tenemos que $10^i \equiv 0 (\mathrm{mod}\, 2)$ para todo $i \geq 1$. Por lo tanto $2|n$ si y sólo si $2|a_0$, es decir, si a_0 es par.
- $k = 3$. Tenemos que $10^i \equiv 1 (\mathrm{mod}\, 3)$ para todo $i \geq 1$.Por lo tanto $3|n$ si y sólo si $3|a_i + a_{i-1} + ... + a_1 + a_0$. Por ejemplo, $3|111$ mientras que no divide a 112.
- $k = 4$. Tenemos que $10 \equiv 2 (\mathrm{mod}\, 4)$ y $10^i \equiv 0 (\mathrm{mod}\, 4)$ para todo $i \geq 2$. Así, $4|n$ si y sólo si $4|2a_1 + a_0$. Por ejemplo, $4|236$ ya que $2a_1 + a_0 = 12$, que es divisible por 4.
- $k = 5$. Tenemos que $10^i \equiv 0 (\mathrm{mod}\, 5)$ para todo $i \geq 1$. Por lo tanto $5|n$ si y sólo si $5|a_0$, es decir, si a_0 es 0 o 5.

Actividad 37 *Probar que* $10^i \equiv 1 (\mathrm{mod}\, 9)$ *si* $i \geq 1$. *Deducir que* $9|n$ *si y sólo si* $9|a_i + a_{i-1} + ... + a_1 + a_0$. *Determinar si los siguientes números son divisibles por 9: 2312342123, 923649264, 45702357 y 982675625662956.*

Actividad 38 *Probar que* $10^i \equiv (-1)^i (\mathrm{mod}\, 11)$ *si* $i \geq 1$. *Deducir que* $11|n$ *si y sólo si* $11|(-1)^i \cdot a_i + (-1)^{i-1} \cdot a_{i-1} + ... + a_2 - a_1 + a_0$. *Determinar si los siguientes números son divisibles por 11: 2312342123, 923649264, 45702357 y 982675625662956.*

Actividad 39 *Probar que* $10 \equiv 3 (\mathrm{mod}\, 7)$, $100 \equiv 2 (\mathrm{mod}\, 7)$, $1000 \equiv -1 (\mathrm{mod}\, 7)$, $10000 \equiv 1 (\mathrm{mod}\, 7)$, *y* $10^i \equiv (-1)^{i-2} (\mathrm{mod}\, 7)$ *si* $i \geq 3$. *Deducir que* $7|n$ *si y sólo si* $7|a_i \cdot (-1)^{i-2} + a_{i-1} \cdot (-1)^{i-3} + ... - a_3 + 2 \cdot a_2 + 3 \cdot a_1 + a_0$. *Determinar si los siguientes números son divisibles por 7: 2312342123, 923649264, 45702357 y 982675625662956.*

Actividad 40 *Deducir las reglas de divisibilidad de 13 y 17. Dar un número de 25 cifras que sea múltiplo de 13 y otro que sea múltiplo de 17.*

1.5.5. Algoritmo RSA

El algoritmo RSA fue introducido en 1977 y se utiliza para codificar mensajes usando números primos. Supongamos que queremos enviar una palabra o mensaje, por ejemplo "leon" de forma codificada. En primer lugar, asignamos a cada letra un número de dos cifras, por ejemplo asignamos a la letra "a" la cifra 11, a "b" la cifra 12, y así consecutivamente. La palabra leon se convierte en el número $x = 22152624$. Este número es el que vamos a codificar con el algoritmo RSA.

Dicho algoritmo se basa en la elección de dos números primos distintos p y q y tomamos $n = p \cdot q$. A continuación elegimos d tal que es coprimo con $(p-1)(q-1)$ y su inverso $d^{-1} = e$ en $\mathbb{Z}_{(p-1)(q-1)}$. Se hacen públicos los valores de n y e, y sólo el receptor del mensaje conoce el valor de d. Para cifrar el mensaje hacemos la cuenta

$$f_C(x) = x^e (\mathrm{mod}\, n),$$

y para descifrarlo

$$f_D(x) = x^d (\mathrm{mod}\, n).$$

El código es muy díficil de descodificar si no se tiene el valor de d cuando los primos p y q son suficientemente grandes. Nótese que se conoce el valor de n, pero no de p y q, que son necesarios para obtener el valor de d. Veamos que la descodificación puede hacerse, es decir que

$$x = f_D(f_C(x)).$$

Para ello, calculamos

$$f_D(f_C(x)) = (f_C(x))^d = (x^e)^d = x^{d \cdot e}.$$

Ahora bien

$$d \cdot e \equiv 1 (\mathrm{mod} (q-1)(p-1)),$$

de donde existirá un entero r tal que

$$d \cdot e = 1 + r \cdot (p-1) \cdot (q-1) = 1 + r \cdot \varphi(p) \cdot \varphi(q) = 1 + r \cdot \varphi(n).$$

Así,

$$x^{d \cdot e} = x^{1 + r \cdot \varphi(n)}.$$

Así, hemos de probar que

$$x^{1 + r \cdot \varphi(n)} \equiv x (\mathrm{mod}\, n).$$

Distinguimos los siguientes casos:

1. Si $\gcd(x, n) = 1$, entonces por el Teorema de Euler se verifica que $x^{\varphi(n)} \equiv 1 (\mathrm{mod}\, n)$, por lo que
$$x^{1 + r \cdot \varphi(n)} = x \cdot (x^{\varphi(n)})^r \equiv x \cdot 1 (\mathrm{mod}\, n) \equiv x (\mathrm{mod}\, n).$$

2. Si $\gcd(x, n) = p \cdot q$, entonces $x \equiv 0 (\mathrm{mod}\, n)$, y por tanto $x^{1 + r \cdot \varphi(n)} = 0^{1 + r \cdot \varphi(n)} \equiv 0 (\mathrm{mod}\, n)$.

3. Si $\gcd(x, n) = p$, entonces $x = p \cdot x'$ para algún entero x' de forma que $\gcd(x', n) = 1$. Entonces
$$x^{1 + r \cdot \varphi(n)} = (p \cdot x')^{1 + r \cdot \varphi(n)} = p^{1 + r \cdot \varphi(n)} \cdot (x')^{1 + r \cdot \varphi(n)}$$
y por el Teorema de Euler
$$(x')^{1 + r \cdot \varphi(n)} \equiv x' (\mathrm{mod}\, n).$$
Utilizando el Teorema de Euler para p en $\mathbb{Z}_q$ tenemos que
$$p^{\varphi(q)} \equiv 1 (\mathrm{mod}\, q),$$

de donde

$$p^{q-1} = 1 + s \cdot q$$

para algún entero s. Entonces

$$p^{1+r(p-1)(q-1)} - p = p\left(p^{r(p-1)(q-1)} - p\right) = p\left((1+sq)^{r(p-1)} - 1\right).$$

Tenemos que $(1+sq)^{r(p-1)} - 1$ es multiplo de q ya que

$$\begin{aligned}(1+sq)^{r(p-1)} - 1 &= 1 + \sum_{k=1}^{r(p-1)} \left(\begin{array}{c} r(p-1) \\ k \end{array} (sq)^k \right) - 1 \\ &= \sum_{k=1}^{r(p-1)} \left(\begin{array}{c} r(p-1) \\ k \end{array} (sq)^k \right).\end{aligned}$$

Por lo tanto $p\left((1+sq)^{r(p-1)} - 1\right)$ es múltiplo de $pq = n$. Así,

$$p^{1+r\cdot\varphi(n)} - p \equiv 0 (\mathrm{mod}\, n),$$

o equivalentemente

$$p^{1+r\cdot\varphi(n)} \equiv p (\mathrm{mod}\, n).$$

Así

$$x^{1+r\cdot\varphi(n)} = p^{1+r\cdot\varphi(n)} \cdot (x')^{1+r\cdot\varphi(n)} \equiv p \cdot x' (\mathrm{mod}\, n) \equiv x (\mathrm{mod}\, n).$$

4. Si $\gcd(x, n) = q$, se procede como en el caso anterior cambiando los papeles de p y q.

Vamos a codificar nuestra palabra. Tomamos primos grandes tecleando

```
from sympy import prime, gcd
p=prime(10000)
q=prime(1000)
n=p*q
m=(p-1)*(q-1)
d,e=9999,61867215
x,y=22152624,1
for i in range(e):
    y=y*x %n
y
```
84131476

Así, el mensaje codificado es 84131476. Para descodificarlo basta teclear

```
84131476**d %n
```
22152624

Para obtener el valor de e hemos tenido que utilizar el algoritmo de Euclides extendido y obtener este mediante el Teorema de Bezout. Para codificar el mensaje hay que hacer la operación

$$22152624^{61867215}$$

y tardaría mucho tiempo de computación hacerlo de forma directa. Por este motivo se ha calculado con el for.

Actividad 41 *Utlizando el algoritmo anterior, se ha recibido el siguiente mensaje codificado que se manda silaba a silaba*

$$459841177 - 191069077 - 815795976$$

$$305847131 - 783303881 - 720588579$$

$$340174716 - 380043468$$

$$4058580 - 368119658$$

Actividad 42 *Construir un sistema de codificación con los primos número 2000 y 5000. Asignando a la letra "a" el valor 11, a "b" el 12 y así consecutivamente, enviar el mensaje "me gusta el helado de chocolate" sílaba a sílaba. Proporcionar la clave para que el receptor pueda descodificarlo.*

1.5.6. Teoría de grafos

Aquí, vamos a construir la matriz de adyacencia de un grafo. Para ello recordemos que para construir un matriz en Python debemos cargar de sympy la sentencia Matrix. Así, para introducir la matrix

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ -3 & -2 & -1 \end{pmatrix}$$

en Python tecleamos

```
from sympy import Matrix
A=Matrix([[1,2,3],[-3,-2,-1]])
```

Tecleando ahora

$$\text{A}$$
$$\begin{pmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} \\ \mathbf{-3} & \mathbf{-2} & \mathbf{-1} \end{pmatrix}$$

vemos que la matriz está introducida en el programa.

Para eliminar filas o columnas de la matriz tenemos las sentencias col_del(n) y row_del(n), donde n indica el número de fila o columna que queremos eliminar. Recordar aquí que las listas en Python empiezan por 0, por lo que para eliminar la primera columna tecleamos

$$\text{A.col\_del(0)}$$
$$\begin{pmatrix} \mathbf{2} & \mathbf{3} \\ \mathbf{-2} & \mathbf{-1} \end{pmatrix}$$

Para añadir filas o columnas tenemos las sentencias row_insert(n,b) y col_insert(n,A), donde n es el número de fila o columna y b es la fila o columna de deseamos insertar. Así, para añadir una tercera fila con ceros tecleamos

```
A.row_insert(2,Matrix([[0, 0]]))
```

$$\begin{pmatrix} \mathbf{2} & \mathbf{3} \\ \mathbf{-2} & \mathbf{-1} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}$$

Aquí hemos de hacer la siguiente aclaración. Al eliminar filas o columnas, Python asigna a la variable A la nueva matriz, pero al insertarlas no. Así, si tecleamos

```
A
```

$$\begin{pmatrix} \mathbf{2} & \mathbf{3} \\ \mathbf{-2} & \mathbf{-1} \end{pmatrix}$$

obtenemos el valor anterior de A.

Las matrices podemos introducirlas también indicando su tamaño y una lista con sus elementos. Python distribuirá los elementos de la lista dividiendo éstos por filar. Por ejemplo, tecleando

```
from sympy import Matrix
C=Matrix(2,3,[1,2,3,4,5,6])
```

introducimos la matriz

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}.$$

Los elementos de la matriz los podemos obtener tecleando C[i,j], donde i indicará el número de fila y j el de columna. Ambos, como todas las listas en Python, empiezan por 0, por lo que C[0,0] es el 1 mientras que C[1,2] es el 6. Para reemplazar un elemento de la matriz, por ejemplo en la matriz C cambiamos el 3 por un 0, bastará con teclear

```
C[0,2]=0
```

y ahora la matriz C será

$$\begin{pmatrix} 1 & 2 & 0 \\ 4 & 5 & 6 \end{pmatrix}.$$

Operaciones con matrices

Para sumar o restar matrices tenemos los símbolos usuales + y -. Tanto para multiplicar matrices por escalares como por matrices tenemos el símbolo del producto *. Es preciso recordar que las operaciones suma y producto de matrices sólo pueden realizarse si las dimensiones de las matrices implicadas lo permiten.

Actividad 43 *Dados las matrices* $A = \begin{pmatrix} 1 & 2 & 4 \\ 2 & 3 & -1 \end{pmatrix}$, $B = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$, $C = \begin{pmatrix} -1 & 1 \\ 2 & -1 \end{pmatrix}$ *y* $D = \begin{pmatrix} 0 & -1 & 0 \\ 2 & 0 & 5 \\ 2 & 1 & 0 \end{pmatrix}$ *calcular:*

$$(a)\ CAB \quad (b)\ A(B+C) \quad (c)\ C^2 + 2C$$

$$(d)\ 2B - 4D^4 \quad (e)\ ADB \quad (f)\ A(B+D)$$

Matrices especiales

En nuestros problemas vamos a tener que construir matrices, por lo que es útil saber cómo se construyen determinadas matrices especiales. Así, tenemos que

- eye(n) es la matriz identidad de dimensión n.
- zeros(n,m) es una matrix de tamaño $n \times m$ con ceros en todas las posiciones.
- ones(n,m) es una matrix de tamaño $n \times m$ con unos en todas las posiciones.
- diag$(a_1, a_2, ..., a_n)$ es una matriz diagonal donde los elementos de su diagonal principal son $a_1, a_2, ..., a_n$. Los elementos de la diagonal principal pueden ser matrices, por lo que la diagonal principal puede entenderse como cajas de matrices.

Por ejemplo, tecleando

diag(1,2,3)

$$\begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{2} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{3} \end{pmatrix}$$

y tecleando

diag(-1,ones(2,2),Matrix([5,7,5]))

$$\begin{pmatrix} \mathbf{-1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{5} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{7} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{5} \end{pmatrix}$$

Determinantes y valores y vectores propios

Para calcular el determinante de una matriz cuadrada tenemos la sentencia det(). Así, para calcular el determinante de la matriz

$$A = \begin{pmatrix} 2 & 3 \\ -2 & -1 \end{pmatrix}$$

tecleamos

```
from sympy import Matrix
A=Matrix([[2,3],[-2,-1]])
A.det()
```
4

Para calcular sus valores propios tenemos la sentencia eigenvals. Dada la matriz anterior, sus valores propios son

```
A.eigenvals()
```
$$\{1/2 - \sqrt{15}i/2 \ : \ 1, 1/2 + \sqrt{15}i/2 : 1\}$$

donde la salida muestra los valores propios y su multiplicidad. Para obtener los vectores propios tenemos la sentencia eigenvects. Por ejemplo, si tecleamos

```
Matrix([[1,0],[1,1]]).eigenvects()
```
$$\left[\left(1, 2, \left[\begin{bmatrix} 0 \\ 1 \end{bmatrix}\right]\right)\right]$$

Python nos devuelve los valores propios, su multiplicidad y una base de vectores propios de la matriz

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

En este caso tenemos un único valor propio 1 de multiplicidad 2 cuya base de vectores propios está dada por el vector $(0, 1)$.

Para obtener la forma diagonal de una matriz cuadrada tenemos la sentencia diagonalize. Así, tecleando

```
Matrix([[0,1],[1,0]]).diagonalize()
```
$$\left(\begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}\right)$$

obtenemos la matriz de cambio de base

$$\begin{pmatrix} -1 & 1 \\ 1 & 1 \end{pmatrix}$$

y la forma diagonal

$$\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$$

de la matriz

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

Actividad 44 *Calcular el determinante de las siguientes matrices cuadradas*

$$(a)\begin{vmatrix} 3 & 5 & 7 & 2 \\ 2 & 4 & 1 & 1 \\ -2 & 0 & 0 & 0 \\ 1 & 1 & 3 & 4 \end{vmatrix} \quad (b)\begin{vmatrix} 1 & \cos x & \cos 2x \\ \cos x & \cos 2x & \cos 3x \\ \cos 2x & \cos 3x & \cos 4x \end{vmatrix} \quad (c)\begin{vmatrix} x & a & b & c \\ a & x & 0 & 0 \\ b & 0 & x & 0 \\ c & 0 & 0 & x \end{vmatrix}$$

Actividad 45 *Calcular los vectores y valores propios, y en caso de existir, la forma diagonal de las matrices B y D del ejercicio 43.*

Aplicación a la teoría de grafos

La matriz de adyacencia de un grafo se usa para trabajar con éste desde un punto de vista computacional. Recordemos que si un grafo simple tiene n vértices y su matriz de adyacencia es A, entonces este grafo es conexo si y sólo si $A + A^2 + ... + A^n$ tiene todas sus entradas no nulas.

Actividad 46 *Representar gráficamente el grafo determinado por las siguientes matrices y determinar si son o no conexos:*

$$(a)\begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \quad (b)\begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$(c)\begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} \quad (d)\begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Actividad 47 *El radio espectral de una matriz es el máximo de los módulos de sus valores propios. Determinar el radio espectral de las matrices de la actividad 46.*

Actividad 48 *Dados los grafos de la figura 1.1, determinar su matriz de adyacencia, si son o no conexos a partir de ésta y calcular su radio espectral.*

Actividad 49 *La gráfica de la figura 49 se corresponde con el patrón de plegado de una teselación del plano. Cada vez que dos aristas se cortan tenemos un vértice, por lo que el grafo es plano. Determinar cuántos vértices de grados 2, 3, 4, 5 y 6 existen. Etiquetar los vértices de la figura empezando la esquina superior izquierda y acabando por la inferior derecha siguiendo el orden habitual de lectura (de izquierda a derecha). Construir su matriz de adyacencia y probar*

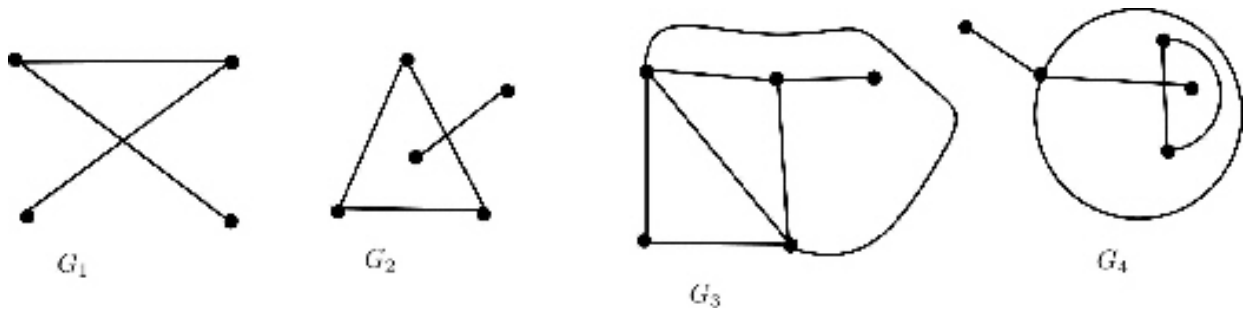
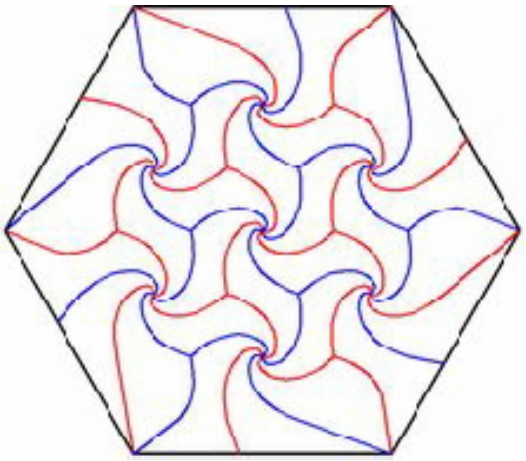


Figura 1.1: Grafos diversos.

que el grafo es conexo y calcular su radio espectral.



Grafo simple

1.5.7. Manipulaciones algebraicas

Antes de proceder a explicar el cálculo de una y varias funciones con el módulo sympy de Python vamos a dar agunas sentencias o instrucciones que permiten manipular y simplificar expresiones y fórmulas. La lista no es exhaustiva, por lo que se refiere a [1] en caso de que se necesite alguna sentencia no contenida en la siguiente tabla.

simplify(expr)	Simplifica la expresión expr
expand(expr)	Desarrolla la expresión expr
factor(expr)	Saca factores comunes en la expresión expr
cancel(expr)	Simplifica factores en la fracción expr
apart(expr)	Obtiene fracciones simples de la fracción expr
trigsimp(expr)	Simplifica la expresión trigonométrica expr
expand_trig(expr)	Desarrolla la expresión trigonométrica expr

Veamos algunos ejemplos

```
from sympy import symbols, simplify, expand, factor
x=symbols('x')
simplify((x + 1)**2-x**2 + 2*x)
4*x + 1
expand((x + 1)**2-2*x**2 + 2*x)
-x**2 + 4*x + 1
factor(x**2-2*x +1)
(x - 1)**2
```

Para fracciones

```
from sympy import symbols, cancel, apart
x=symbols('x')
cancel((x**2+2*x+1)/((x+1)*(x-2)))
(x + 1)/(x - 2)
apart((x**2+2*x+1)/((x+1)*(x-2)))
1 + 3/(x - 2)
```

Para expresiones trigonométricas

```
from sympy import symbols, trigsimp, expand_trig, sin, cos, tan
x=symbols('x')
trigsimp(sin(x)**4 - 2*cos(x)**2*sin(x)**2 + cos(x)**4)
cos(4*x)/2 + 1/2
expand_trig(tan(2*x))
2*tan(x)/(1 - tan(x)**2)
```

Finalmente, si tenemos una expresión algebraica expr y queremos sustituir alguna de las variables que la componen por un número u otra variable, tenemos la sentencia subs(). Los siguientes ejemplos muestran cómo funciona esta sentencia.

```
from sympy import symbols
x,y,z=symbols('x y z')
(x**2+y**2).subs(x,1)
y**2+1
(x**2+y**2).subs(x,z)
y**2+z**2
```

donde tenemos que indicar la variable que queremos sustituir y por qué valor.

Actividad 50 *Simplificar las siguientes expresiones utilizando las sentencias apropiadas.*

1. $(x+1)^3 - x^3 - 3x$.
2. $\sin(2x)^2 - 2\cos(x)^2\sin(x)^2 + \sin(x)^2$.
3. $(x+2)^4 - (x-2)^4 - 1$.

4. $\frac{2x^3-6x^2+2x-6}{2x^3-4x^2+2x-4}$.

5. $\frac{\cos(2x)-3\cos^2 x+1}{1-\sin x}$.

6. $(\sin x + 1)^3 - \sin^3 x - 3\cos x$.

7. $\frac{(x-y)^2+(x+y)^2}{x^2+y^2}$.

8. $\sin(x + y) - \sin(x - y) - 2\cos(2x)$.

Actividad 51 *Desarrollar las siguientes expresiones utilizando la sentencia apropiada.*

1. $(x + 4)^4 + 3(x - 3)^3 + 2(x + 2)^2 + x - 1$.

2. $\sin(x + y + z)$.

3. $(x + 2)^3 + (x + 2)^2 + (x + 2) + 2$.

4. $\frac{\tan(-y)}{\tan(x+y)}$.

5. $\frac{(x+2)^3+(x+2)^2+(x+2)}{x+2}$.

6. $\frac{\tan\left(\frac{\pi}{3}-x\right)}{\sin\left(\frac{\pi}{4}+x\right)}$.

7. $(x + y)^4 + 3(x - y)^3 + 2(x + y)^2 + x - 1$.

Actividad 52 *Descomponer en fracciones simples las siguientes fracciones.*

1. $\frac{x^2+2x+1}{x^4+8x^3+23x^2+28x+12}$.

2. $\frac{x^2-2x+1}{x^4+8x^3+23x^2+28x+12}$.

3. $\frac{x^2-2x+1}{x^4+4x^3+5x^2+4x+4}$.

4. $\frac{x^2-2x-1}{x^6+4x^5+6x^4+8x^3+9x^2+4x+4}$

1.5.8. Definición de funciones en sympy

Además de la definición de funciones en Python, sympy dispone de una sentencia para indicar que una variable es en realidad una función. Por ejemplo, si tecleamos

$$\mathrm{f} = \mathrm{Function}('\mathrm{f}')$$

definimos una función en la que no decimos qué variables la definen. A partir de este momento, podemos asignarle valores, como por ejemplo

```
from sympy import symbols
x,y=symbols('x y')
f(1)
f(1)
f(x,y)
f(x,y)
f(1,y)
f(1,y)
```

La función queda así parcialmente evaluada.

Si queremos que la función se evalue completamente, tenemos la sentencia def explicada anteriormente. Aquí puede ser útil la sentencia lambdify, que combierte la función en otra que puede ser evaluada por el paquete numpy. Por ejemplo, tecleamos

```
from sympy import symbols, lambdify
x,y=symbols('x y')
f=lambdify([x,y],x+y)
f(1,1)
2
```

Aquí hemos tomado la función $x + y$ y la hemos preparado para que se pueda emplear con el paquete numpy, que como veremos es útil para hacer representaciones gráficas de funciones. Nótese que en la sentencia lamdify hemos de decir qué variables son las que definen la función en una lista, en este caso, [x,y].

1.5.9. Límites

Para el cálculo de límites tenemos la sentencia limit(f(x), x, x0), donde f(x) es la función, x la variable y x0 el punto donde queremos calcular el límite. Supongamos que queremos calcular

$$\lim_{x\to 0} \frac{\sin x}{x}.$$

Veamos cómo hacer este límite usando Python. En primer lugar, cargamos

```
from sympy import symbols, limit
x=symbols('x ')
limit(sin(x)/x,x,0)
1
```

donde **1** es el resultado. En la sentencia limit hay que explicitar la función, la variable y el punto donde queremos calcular el límite, separados por comas. Es posible hacer límites en ∞ como muestra el siguiente ejemplo

```
from sympy import symbols, limit, oo
x=symbols('x ')
limit(x**2+1,x,oo)
oo
```

donde, como vemos oo es ∞. También podemos hacer el límite importando ind del módulo math. Para hacer límites por la derecha e izquierda tenemos la sentencia limit(f(x), x, x0, '±'). Por ejemplo,

$$\lim_{x\to 0^-} \frac{1}{x}$$

se calculará como

```
from sympy import symbols, limit
x=symbols('x ')
limit(1/x,x,0,'-')
-oo
```

Para calcular límites en varias variables tenemos que usar límites iterados. Por ejemplo, del límite

$$\lim_{(x,y)\to(0,0)} \frac{x^3+y^3}{(x^2+y^2)}$$

podemos calcular sus límites iterados

$$\lim_{x\to 0}\lim_{y\to 0} \frac{x^3+y^3}{(x^2+y^2)} \quad y \quad \lim_{y\to 0}\lim_{x\to 0} \frac{x^3+y^3}{(x^2+y^2)}$$

y el límite al pasar a coordenadas polares

$$\lim_{r\to 0} r(\cos^3\theta+\sin^3\theta),$$

aunque el alumno deberá saber si estos límites son suficientes para determinar el valor del límite doble. En Python

```
from sympy import symbols, limit, simplify
x,y,r,theta=symbols('x y r theta')
f=(x**3+y**3)/(x**2+y**2)
limit(limit(f,x,0),y,0)
0
limit(limit(f,y,0),x,0)
0
F=simplify(f.subs({x:r*cos(theta), y:r*sin(theta)}))
limit(F,r,0)
0
```

Ahora bien, debemos acotar

$$|r(\cos^3\theta+\sin^3\theta)-0| \leq 2r,$$

y como $\lim_{r\to 0} 2r = 0$, se tiene que

$$\lim_{(x,y)\to(0,0)} \frac{x^3+y^3}{(x^2+y^2)} = 0.$$

Actividad 53 *Calcular los siguientes límites:*

1. $\lim_{x\to 0} \frac{1-\cos 2x}{\sin x^2} =$
2. $\lim_{x\to \infty} \frac{\log x}{x} =$
3. $\lim_{x\to 1} \frac{1-x^2}{1-x} =$
4. $\lim_{x\to 0} \frac{1}{x} =$

Actividad 54 *Calcular los siguientes límites de dos variables:*

1. $\lim_{(x,y)\to(0,0)} \frac{x^3+y^3}{x^2+y^2+5x^4y^4}$.
2. $\lim_{(x,y)\to(0,0)} (x^2+y^2)\sin \frac{1}{\sqrt{x^2+xy+y^2}}$.
3. $\lim_{(x,y)\to(0,0)} \frac{x^2+y^2+x^4}{x^2+y^2-y^3}$.
4. $\lim_{(x,y)\to(1,-2)} \frac{x^2y+2x^2-2xy-4x+y+2}{x^2-2x+y^4+8y^3+24y^2+32y+17}$.

Actividad 55 *Probar que los siguientes límites de dos variables no existen:*

1. $\lim_{(x,y)\to(0,0)} \frac{x^2-xy}{x^2+3y}$.
2. $\lim_{(x,y)\to(0,0)} \frac{\log(1+x^2y^2)}{x^4+y^4+2x^2y^2}$.
3. $\lim_{(x,y)\to(0,0)} \frac{x^4-y^4}{x^2+y^3}$.
4. $\lim_{(x,y)\to(0,0)} \frac{xy^2}{x^2+y^4}$.

Actividad 56 *Calcular, en caso de que existan, los siguientes límites:*

1. $\lim_{(x,y)\to(0,0)} \frac{7x^3-6y^2}{4x^3+y^2}$.
2. $\lim_{(x,y)\to(0,0)} \left(\frac{x^2-y^2}{x^2+y^2}\right)^2$.
3. $\lim_{(x,y)\to(0,0)} \frac{2x^3-x^2+y}{x^3-x^2+y}$.
4. $\lim_{(x,y)\to(1,1)} \frac{xy-y-2x+2}{x-1}$.

Actividad 57 *Estudiar la continuidad de la función*

$$f(x,y) = \begin{cases} \frac{\sin(xy)}{x} & si \quad x \neq 0, \\ y & si \quad x = 0. \end{cases}$$

1.5.10. Derivadas

Para el cálculo de derivadas tenemos la sentencia diff(f(x),x), donde f(x) es la función a derivar y x es la variable de derivación. Por ejemplo, para calcular la derivada de $e^x \sin x$ debemos teclear

```
from sympy import symbols, diff, sin, exp
x=symbols('x')
diff(sin(x)*exp(x), x)
exp(x)*sin(x) + exp(x)*cos(x)
```

Para calcular la derivada n-ésima, tenemos la sentencia diff(f(x),x,n), donde n es el orden de la derivada que queremos calcular. Tambié puede calcularse añadiendo la variable que queremos derivar tantas veces como derivadas queremos hacer. Por ejemplo, a derivada segunda de $e^x \sin x$ se calcula tecleando

```
from sympy import symbols, diff, sin, exp
x=symbols('x')
diff(sin(x)*exp(x), x,2)
2*exp(x)*cos(x)
diff(sin(x)*exp(x), x,x)
2*exp(x)*cos(x)
```

Si tenemos funciones de varias variables, las derivadas las hacemos indicando las variables que queremos derivar en el orden en que queremos hacer las derivadas separadas por comas. Por ejemplo

$$\frac{\partial}{\partial x \partial y} \sin(xy)$$

se calcula como

```
from sympy import symbols, diff, sin
x,y=symbols('x y')
diff(sin(x*y), x,y)
-x*y*sin(x*y) + cos(x*y)
```

Actividad 58 *Calcula las derivadas de las siguientes funciones:*

1. $f(x) = \log(\sin x)$.
2. $f(x) = \frac{\log(\cos x) \arcsin x}{e^x \log_4(x^2+10)}$.
3. $f(x) = 1 + \left(\frac{3x+e^x}{x^2+\tan\sqrt{x}}\right)$.

Actividad 59 *Calcula la derivada cuarta y sexta de cada una de las funciones del ejercicio 58.*

Actividad 60 *Calcula* $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial y}$, $\frac{\partial^2 f}{\partial x \partial y}$, $\frac{\partial^2 f}{\partial x^2}$, $\frac{\partial^3 f}{\partial x \partial y \partial x}$ *y* $\frac{\partial^4 f}{\partial x^4}$ *de cada una de las siguientes funciones:*

1. $f(x,y) = \cos(\sin x - y)$.

2. $f(x,y) = \arcsin\left(\frac{1}{x^2+y^2}\right)$.

3. $f(x,y) = \frac{1+\log(x+e^y)}{\sqrt{x^2-\tan xy}}$.

Actividad 61 *Estudiar la existencia y continuidad de las derivadas parciales de la función*

$$f(x,y) = \begin{cases} \frac{xy^2}{x^2+y^4} & si \quad (x,y) \neq (0,0), \\ 0 & si \quad (x,y) = (0,0). \end{cases}$$

Actividad 62 *Dada la función*

$$f(x,y) = x^2 \tan\left(\frac{x^2}{x^2+y^2}\right),$$

comprobar que se cumple para todo punto de su dominio la relación

$$x\frac{\partial f}{\partial x}(x,y) + y\frac{\partial f}{\partial y}(x,y) = 2f(x,y).$$

Actividad 63 *Dada la función*

$$f(x,y) = \frac{\sqrt{x}+\sqrt{y}}{x+y},$$

comprobar que se cumple para todo punto de su dominio la relación

$$x\frac{\partial g}{\partial x}(x,y) + y\frac{\partial g}{\partial y}(x,y) = -\frac{g(x,y)}{2}.$$

Actividad 64 *Sea $f : \mathbb{R}^3 \to \mathbb{R}$ una función dada por $f(x,y,z) = \log(\varphi(x+yz))$, donde $\varphi : \mathbb{R} \to (0,+\infty)$, es una función positiva de clase C^2. Probar que*

$$z^2\frac{\partial^2 f}{\partial x^2}(x,y,z) - \frac{\partial^2 f}{\partial y^2}(x,y,z) + \frac{1}{y}\frac{\partial f}{\partial z}(x,y,z) = \frac{1}{\varphi(x+yz)}\frac{d\varphi}{dt}(x+yz).$$

Nota: *Aquí, teniendo en cuenta que $\varphi(t)$ es una función real de una variable, hay que hacer el cambio de variable $t = x+yz$ para construir la función f.*

1.5.11. Diferencial

Dada una función diferenciable $\mathbf{f} : A \subseteq \mathbb{R}^n \to \mathbb{R}^m$ podemos calcular la diferencial en un punto $\mathbf{x}_0 \in A$ obteniendo la matriz Jacobiana de $\mathbf{f}$ en dicho punto. Para ello hay que introducir la función en forma de matriz y usar la sentencia jacobian.

Veamos por ejemplo como obtener la diferencial de la función $\mathbf{f} : \mathbb{R} \to \mathbb{R}^3$ dada por

$$\mathbf{f}(t) = (\cos t, e^t, t)$$

en el punto $t = 1$. Para ello tecleamos

```
from sympy import symbols, cos, exp, Matrix
t=symbols('t')
f=Matrix([[cos(t),exp(t),t]])
f.jacobian([t])
```

$$\begin{bmatrix} -\sin(t) \\ e^t \\ 1 \end{bmatrix}$$

y obtenemos la matriz Jacobiana de la función en un punto arbitrario t. Nótese que hay que indicar en la sentencia jacobian las variables de las que depende la función. Si tecleamos

```
f.jacobian([t]).subs({t:1})
```

$$\begin{bmatrix} -\sin(1) \\ e \\ 1 \end{bmatrix}$$

obtendremos la matriz Jacobiana en el punto indicado. La diferencial será la aplicación lineal dada por

$$df(1)(t) = (-t\sin 1, te, t).$$

Si ahora queremos calcular la diferencial de la función $\mathbf{f}(x, y) = (x^y, xy)$ en un punto arbitrario, tecleamos

```
from sympy import symbols, Matrix
x,y=symbols('x y')
f=Matrix([[x**y,x*y]])
f.jacobian([x,y])
```

$$\begin{bmatrix} \frac{x^y y}{x} & x^y \log(x) \\ y & x \end{bmatrix}$$

Actividad 65 *Obtener la diferencial de las siguientes funciones:*

1. $\mathbf{f}(x, y, z) = (zx, xy)$.
2. $\mathbf{f}(x, y, z) = (\cos(zx), \sin(xy), z\sin(xy))$.
3. $\mathbf{f}(x, y) = (\frac{x}{e^y}, x + y^2 \log x)$.
4. $\mathbf{f}(x, y, z, t) = (\frac{t}{x^2+y^2}, xy^z, t + z\tan(xyzt))$.

Actividad 66 *Utilizar la regla de la cadena para obtener la diferencial de la función* $\mathbf{f} \circ \mathbf{g}$ *en los siguientes casos:*

1. $\mathbf{f}(x, y, z) = (zx, xy)$ *y* $\mathbf{g}(x, y) = (x^2 + y, x + y, x - y^2)$.
2. $\mathbf{f}(x, y) = (\frac{x}{e^y}, x + y^2 \log x)$ *y* $\mathbf{g}(x, y, z) = (zx^2 + y, x + yz)$.
3. $\mathbf{f}(x, y, z) = (x + y^2z, zx, y^2z \log x)$ *y* $\mathbf{g}(t) = (t^2 + 1, t - t^3, e^{t-1})$.

1.5.12. Integrales

La sentencia para calcular integrales es integrate, que tiene diferentes sintaxis según se trate de una integral definida o indefinida. Para el cálculo de primitivas, por ejemplo

$$\int e^x x dx$$

tenemos la sentencia integrate(f(x),x), donde debemos indicar la función f(x) y la variable de integración. Por ejemplo, la integral anterior se calcula tecleando

```
from sympy import symbols, integrate, exp
x=symbols('x')
integrate(x*exp(x),x)
(x - 1)*exp(x)
```

Si lo que tenemos es una integral definida, por ejemplo

$$\int_0^1 e^x x dx$$

debemos añadir los límites de integración separados por comas según la sintaxis integrate(f(x),(x,x0,x1)), donde xo es el límite inferior y x1 el superior. Debemos teclear

```
from sympy import symbols, integrate, exp
x=symbols('x')
integrate(x*exp(x),(x,0,1))
1
```

Es posible calcular integrales impropias como

$$\int_{-\infty}^0 x e^x dx$$

tecleando

```
from sympy import symbols, integrate, exp,oo
x=symbols('x')
integrate(x*exp(x),(x,-oo,0))
-1
```

La integrales dobles y triples se calculan mediante el uso del teorema de Fubbini. Tenemos dos alternativas a la hora de hacer las integrales, bien usando dos veces la integral simple, bien explicitando el recinto de integración como se muestran en los siguientes ejemplos, donde se hace el cálculo por ambos medios. Por ejemplo, para calcular

$$\int\int_{[-1,1]\times[0,1]} xy dx dy$$

tecleamos

```
from sympy import symbols, integrate
x,y=symbols('x y')
integrate(integrate(x*y,(x,-1,1)),(y,0,1))
0
integrate(x*y,(x,-1,1),(y,0,1))
0
```

Para calular

$$\int\int_{\Omega} xydxdy$$

donde

$$\Omega = \{(x,y) : y < x,\ 0 \leq x \leq 1\}$$

tecleamos

```
from sympy import symbols, integrate
x,y=symbols('x y')
integrate(integrate(x*y,(y,0,x)),(x,0,1))
1/8
integrate(x*y,(y,0,x),(x,0,1))
1/8
```

Actividad 67 *Calcular las primitivas de las siguientes funciones:*

1. $f(x) = \cos x \sin(2x) \cos 3x \tan 4x$
2. $f(x) = \frac{x+x^2}{1+x^{10}}$
3. $f(x) = \cos x e^{10x}$
4. $f(x) = \sin x \cos^2(2x) \sin^3(3x)$.

Actividad 68 *Sea D la región plana acotada por el eje Y y la parábola $x = -4y^2 + 3$. Calcular el área de dicha región y la integral*

$$\int\int_{D} x^3 y dxdy.$$

Actividad 69 *Calcular*

$$\int\int_{D} (\sin x + y + 3)dxdy$$

donde $D = \{(x,y) \in \mathbb{R}^2 : x^2 + y^2 \leq 4\}$.

Actividad 70 *Calcular*

$$\int\int_{D} x^2 dxdy$$

donde D es el recinto limitado por $y = x$, $y = -x$ y $0 \leq x \leq 1$.

Actividad 71 *Calcular el área comprendida entre las circunferencias* $x^2+y^2 = 2x$, $x^2+y^2 = 4x$ *y las rectas* $y = x$ *e* $y = 0$.

Actividad 72 *Hallar el volumen de la región dentro de la superficie* $z = x^2 + y^2$ *con* z *entre 0 y 10.*

Actividad 73 *Calcular*

$$\int\int\int_D xdxdydz$$

donde D *es la región acotada por los planos* $x = 0$, $y = 0$ *y* $z = 2$ *y por la superficie* $z = x^2+y^2$ *definida en el primer cuadrante* $x, y \geq 0$.

Actividad 74 *Evaluar*

$$\int\int\int_D (1 - z^2)dxdydz$$

donde D *es la pirámide con vértice superior* $(0, 0, 1)$ *y vértices de la base* $(1, 1, 0)$, $(1, -1, 0)$, $(-1, 1, 0)$ *y* $(-1, -1, 0)$.

Actividad 75 *Calcular*

$$\int\int\int_D xdxdydz$$

donde

$$D = \{(x, y, z) \in \mathbb{R}^3 : x^2 + z^2 \leq z^2,\ 0 \leq z \leq 2\}.$$

Actividad 76 *Calcular*

$$\int\int\int_D xdxdydz$$

donde

$$D = \{(x, y, z) \in \mathbb{R}^3 : x^2 + z^2 \leq z^2,\ x^2 + y^2 + z^2 \leq 1,\ 0 \leq z\}.$$

Actividad 77 *Calcular el volumen de la región*

$$D = \{(x, y, z) \in \mathbb{R}^3 : x^2 + z^2 \leq z,\ 2 - x^2 - y^2 \geq z\}.$$

Actividad 78 *Calcular*

$$\int\int\int_D zdxdydz$$

donde

$$D = \{(x, y, z) \in \mathbb{R}^3 : x^2 + z^2 + z^2 \leq 9,\ x \geq 0,\ y \geq 0,\ z \geq 0\}.$$

1.5.13. Resolución de ecuaciones

Para escribir una ecuación en Python tenemos que hacerlo con la sentencia Eq que debemos importar desde sympy. Si tenemos la ecuación

$$eq1 = eq2,$$

ésta se escribe en Python Eq(eq1,eq2) y el programa iguala ambas expresiones. Para resolver la ecuación tenemos la sentencia solveset que debemos importar desde sympy. Por ejemplo, para resolver la ecuación $x^3 + x = 0$ tecleamos

```
from sympy import symbols, Eq, solveset
x=symbols('x')
solveset(Eq(x**3+x,0),x)
{0, -I, I}
```

Notar que debemos indicar la variable que deseamos calcular, ya que es posible reolver la ecuación

$$x^2 = ax + b$$

tecleando

```
from sympy import symbols, Eq, solveset
x,a,b=symbols('x a b')
solveset(Eq(x**2,a*x+b),x)
{a/2 - sqrt(a**2 + 4*b)/2, a/2 + sqrt(a**2 + 4*b)/2}
```

Si la ecuación tiene una de los lados de la igualdad igual a 0, como en el ejemplo $x^3 + x = 0$, solveset puede usarse de forma más simple

```
from sympy import symbols, solveset
x=symbols('x')
solveset(x**3+x,x)
{0, -I, I}
```

Tambien podemos usar la sentencia solve, de una sintaxis similar a solveset, que permite encontrar soluciones de ecuaciones de la forma

$$ecuacion = 0$$

y sistemas de ecuaciones definidos de la forma de la ecuación anterior. Por ejemplo, para resolver la ecuación $x^3 + x = 0$ podemos teclear

```
from sympy import symbols, solve
x=symbols('x')
solve(x**3+x,x)
[0, -I, I]
```

y para resolver el sistema

$$\begin{cases} x + y^2 - 1 = 0, \\ x + y - 1 = 0 \end{cases}$$

tecleamos

```
from sympy import symbols, solve
x,y=symbols('x y')
solve([x+y**2-1,x+y-1],x,y)
[(0, 1), (1, 0)]
```

La sentencias solve y solveset tienen limitaciones pues permiten obtener soluciones de algunos tipos de ecuaciones polinómicas. Por ejemplo, la sentencia

```
solveset(cos(x)-x,x)
```

no proporciona ninguna solución. Estas ecuaciones, y bastante de las polinómicas de grado mayor que 4 se deben resolver numéricamente.

1.5.14. Desarrollo en serie de potencias

Dada una función f(x) podemos encontrar su desarrollo en serie de potencias con la sentencia series, que tiene la sintaxis f(x).series(x,x0,n), donde x es la variable de la función f(x), x0 es el centro de la serie y n es el número de términos de la serie. Por ejemplo, para obtener la serie de potencias de grado 5 centrada en 0 de la función $e^x \sin x$ tecleamos

```
from sympy import symbols, exp, sin, series
x=symbols('x')
(exp(x)*sin(x)).series(x,0,6)
x + x**2 + x**3/3 - x**5/30 + O(x**6)
```

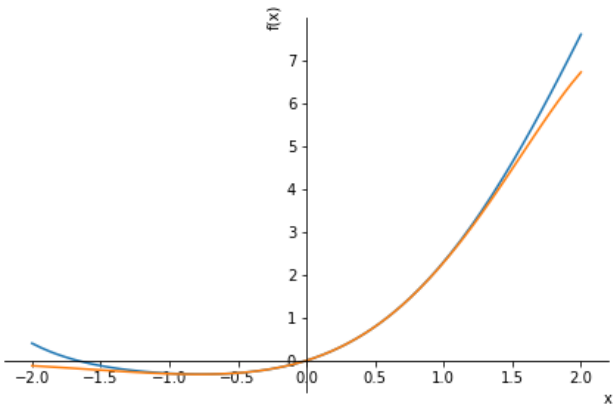
que nos da la serie de potencias

$$p(x) = x + x^2 + \frac{1}{3}x^3 - \frac{1}{30}x^5$$

con el resto $O(x^6)$. Podemos representar la función y la serie de potencias teclealndo

```
from sympy import symbols, exp, sin, series, plot
x,t=symbols('x t')
(exp(x)*sin(x)).series(x,0,6)
x + x**2 + x**3/3 - x**5/30 + O(x**6)
t=x + x**2 + x**3/3 - x**5/30
plot(t,exp(x)*sin(x),(x,-2,2))
```

que nos da la gráfica en el dominio (-2,2) siguiente



que muestra como la aproximación es buena cerca de 0, y varía en los extremos del intervalo.

Si no estamos interesados en el resto, podemos utilizar la sentencia remove para quitar la O(). Así al teclear

```
from sympy import symbols, exp, sin, series, plot
x,t=symbols('x t')
t=(exp(x)*sin(x)).series(x,0,6).removeO()
x + x**2 + x**3/3 - x**5/30
plot(t,exp(x)*sin(x),(x,-2,2))
```

obtenemos la misma gráfica.

Actividad 79 *Calcular los polinomios de Taylor de grado 2, 5 y 8 en los punto 0 y 1 de las siguientes funciones*

$$(a)\ f(x) = \log(1+x) \quad (b)\ f(x) = e^{x^2+1} \quad (c)\ f(x) = \sin(\cos x)$$

$$(d)\ f(x) = e^x \log(1+x^2) \quad (e)\ f(x) = \tfrac{2x}{1+x^2} \quad (f)\ f(x) = \sin(e^x)$$

Hacer una representación conjunta de las 4 funciones en los intervalos $(-1,1)$, $(-2,2)$ *y* $(-5,5)$.

Capítulo 2

Gráficas con Python

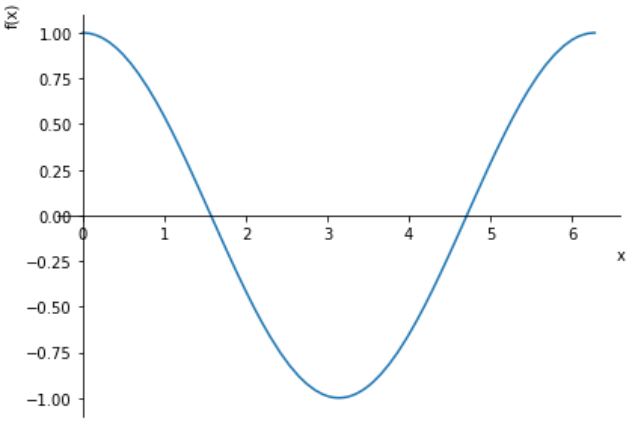
2.1. Graficas con el módulo sympy

2.1.1. Representación de funciones reales

Para representar funciones de una variable tenemos incorporado en sympy la sentencia plot. Todas las figuras representadas pueden guardarse en formato png. Por ejemplo, para representar $\cos x$ en el intervalo $[0, 2\pi]$ tecleamos

```
from sympy import symbols, plot, cos
x=symbols('x')
plot(cos(x),(x,0,2*pi))
```

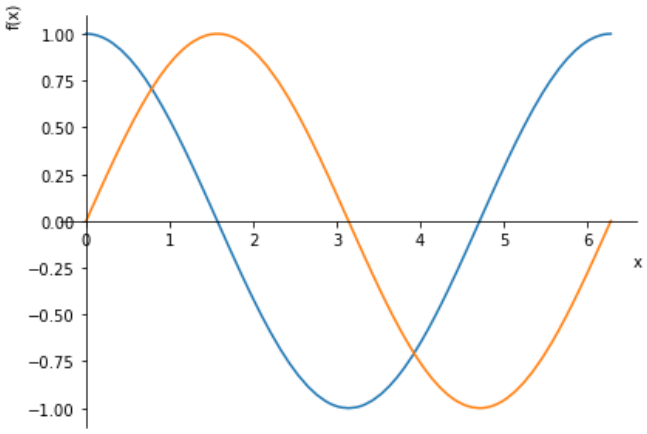
y obtenemos la representación



Como vemos, tenemos que indicar el intervalo donde queremos representar la función entre paréntesis. Podemos representar varias funciones a la vez, por ejemplo

```
from sympy import symbols, plot, cos, sin
x=symbols('x')
plot(cos(x),sin(x),(x,0,2*pi))
```

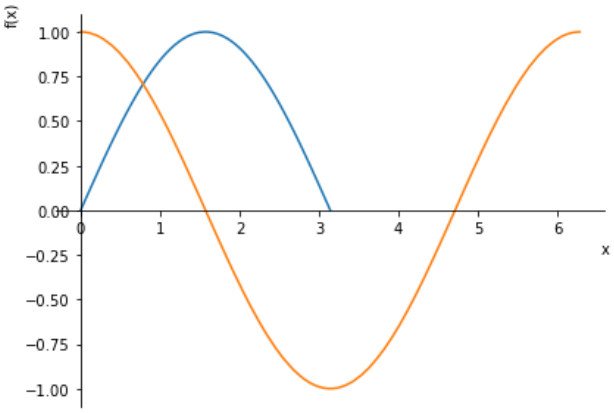
representa a la vez el seno y el coseno como muestra la siguiente figura



Si lo deseamos, podemos representar funciones en diferentes dominios. Por ejemplo, tecleando

```
from sympy import symbols, plot, cos, sin
x=symbols('x')
plot((sin(x),(x,0,pi)),(cos(x),(x,0,2*pi)))
```

obtenemos la representación del seno entre 0 y π y la del coseno entre 0 y 2π, como muestra la siguiente figura



Actividad 80 *Representar gráficamente las siguientes funciones de una variable:*

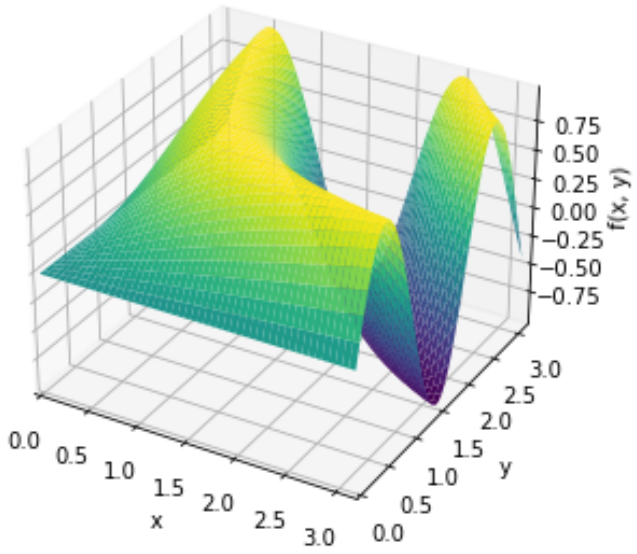
1. $f(x) = \frac{1+x}{1-x^2}$ *en el dominio* $[-2, 2]$.
2. $f(x) = e^{x^2}\frac{1+x}{1-x^2}$ *en el dominio* $[-2, 2]$.
3. $f(x) = \sin\left(\frac{1+x}{1-x^2}\right)$ *en el dominio* $[-2, 2]$.
4. $f(x) = e^{x\cos x}$ *en el dominio* $[-5, 5]$.
5. $f(x) = \frac{e^x}{\cos x}$ *en el dominio* $[-\pi, \pi]$.

2.1.2. Representación de funciones reales de varias variables

Para representar funciones de dos variables tenemos la sentencia plot3d, que tenemos que importar del módulo sympy.plotting. Habrá que indicar tanto la función como los intervalos de ambas variables donde queremos hacer la representación. Por ejemplo, para representar la función $\sin(xy)$ en $[0,\pi]^2$ tecleamos

```
from sympy import symbols, cos, sin
from sympy.plotting import plot3d
x,y=symbols('x y')
plot3d(sin(x*y),(x,0,pi),(y,0,pi))
```

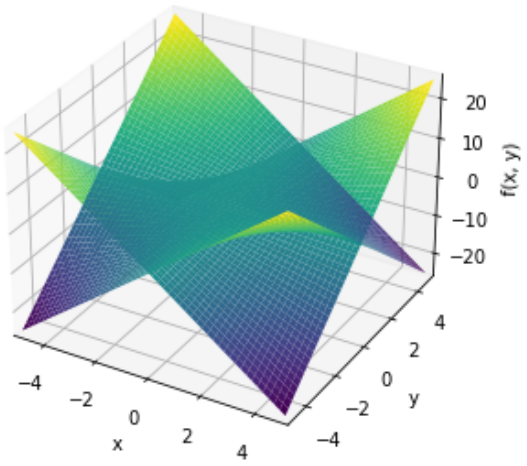
obteniendo la figura siguiente



De nuevo, es posible representar varias funciones separándolas por comas. Por ejemplo, tecleando

```
from sympy import symbols
from sympy.plotting import plot3d
x,y=symbols('x y')
plot3d(x*y, -x*y, (x, -5, 5), (y, -5, 5))
```

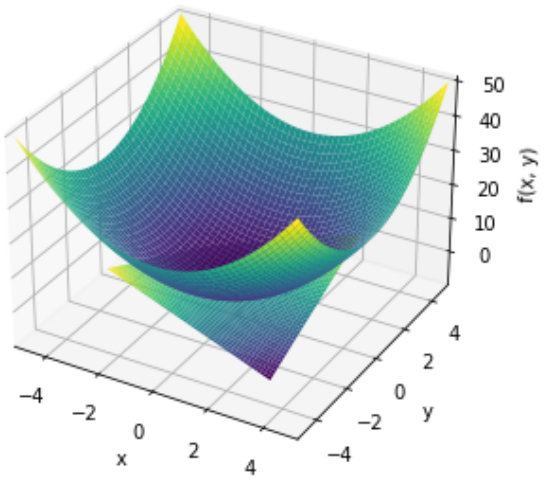
obtenemos la gráfica



que se corresponde a las gráficas de las funciones xy y $-xy$ en el dominio $[-5,5]^2$. Es posible representar varias funciones en diferentes dominios, como por ejemplo

```
from sympy import symbols
from sympy.plotting import plot3d
x,y=symbols('x y')
plot3d((x**2 + y**2, (x, -5, 5), (y, -5, 5)),(x*y, (x, -3, 3), (y, -3, 3)))
```

que da la gráfica



de la función x^2+y^2 en $[-5,5]^2$ y xy en el dominio $[-3,3]^2$.

Actividad 81 *Representar gráficamente las siguientes funciones de varias variables:*

1. $f(x,y) = \log(x^2+y^2)$ *en el dominio* $[-2,2]\times[-2,2]$.
2. $f(x,y) = x^2+y^2$ *en el dominio* $[-2,2]\times[-2,2]$.

3. $f(x,y) = e^{\frac{1}{x^2+y^2}}$ *en el dominio* $[-2,2] \times [-2,2]$.

4. $f(x,y) = (x^2+y^2)\sin\frac{1}{x^2+y^2}$ *en el dominio* $[-1,1] \times [-1,1]$.

5. *Representar conjuntamente las gráficas 2 y 3.*

6. *Representar conjuntamente las gráficas 2 y 4.*

2.1.3. Visualización de recintos en $\mathbb{R}^3$

Para visualizar conjuntos en $\mathbb{R}^3$ usamos las gráficas disponibles en el módulo sympy. Veamos algunos ejemplos. Dado el conjunto

$$A = \{(x,y,z) \in \mathbb{R}^3 : 0 < x+y+z < 10,\ 0 < x < 1,\ -1 < y < 2\}$$

procedemos de la siguiente manera. Consideramos las expresiones $x+y+z = 0$ y $x+y+z = 10$ y definimos las funciones

$$z = -x - y$$

y

$$z = 10 - x - y.$$

El conjunto estará contenido entre las gráficas de ambas funciones en el rectángulo $[0,1]\times[-1,2]$. Para representarlo con Python tecleamos

```
from sympy import symbols
x,y=symbols('x y')
def f(x,y):
    return -x-y
def g(x,y):
    return 10-x-y
from sympy.plotting import plot3d
plot3d(f(x,y),g(x,y), (x, 0, 1), (y, -1, 2))
```

obteniendose la gráfica de la figura 2.1.

En general, así sólo podremos visualizar conjuntos cuando se puedan expresar como gráficas de funciones reales del plano. Veremos más adelante cómo visualizar estos conjuntos usando un concepto más general de superficie parametrizada. No obstante, es útil a la hora de representar el siguiente conjunto

$$B = \{(x,y,z) \in \mathbb{R}^3 : x^2+y^2 < z < x+y+10\}.$$

Aquí definimos las funciones

$$z = x^2 + y^2$$

y

$$z = 10 + x + y.$$

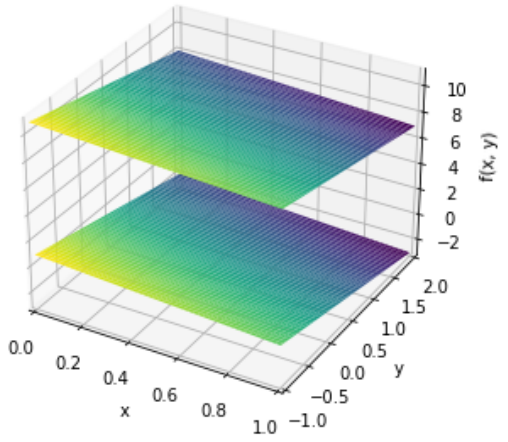


Figura 2.1: Visualización del conjunto A.

En principio no sabemos donde representarlas. Un comienzo es encontrar la intersección de ambas superficies, es decir, resolver el sistema

$$\begin{cases} z = x^2 + y^2, \\ z = 10 + x + y, \end{cases}$$

que nos da la curva

$$x^2 + y^2 - x - y = 10.$$

Procedemos a completar cuadrados para ver qué curva es. Tomamos

$$\begin{aligned} x^2 + y^2 - x - y &= x^2 + y^2 - 2x\frac{1}{2} - 2y\frac{1}{2} \\ &= x^2 - 2x\frac{1}{2} + \frac{1}{4} - \frac{1}{4} + y^2 - 2y\frac{1}{2} + \frac{1}{4} - \frac{1}{4} \\ &= \left(x - \frac{1}{2}\right)^2 + \left(y - \frac{1}{2}\right)^2 - \frac{1}{2}, \end{aligned}$$

por lo que la curva es

$$\left(x - \frac{1}{2}\right)^2 + \left(y - \frac{1}{2}\right)^2 = 10 + \frac{1}{2} = \frac{21}{2},$$

es decir una circunferencia de centro $(1/2, 1/2)$ y radio $\sqrt{21/2}$. Los valores máximos y mínimos de x e y son

$$\frac{1}{2} + \sqrt{\frac{21}{2}} \approx 3{,}74037034920393$$

y

$$\frac{1}{2} + \sqrt{\frac{21}{2}} \approx -2{,}74037034920393,$$

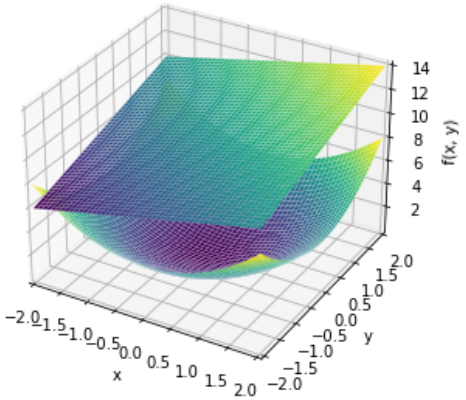


Figura 2.2: Representación del plano $z = 10 + x + y$ y $z = x^2 + y^2$.

por lo que un dominio apropiado para representar las funciones es $[-3, 4] \times [-3, 4]$. Para representar tecleamos

```
from sympy import symbols
x,y=symbols('x y')
def f(x,y):
    return x**2+y**2
def g(x,y):
    return 10+x+y
from sympy.plotting import plot3d
plot3d(f(x,y),g(x,y), (x, -3, 4), (y, -3, 4))
```

pero vemos que la imagen no es nítida. Disminuyendo el dominio donde se hace la representación tecleando

```
plot3d(f(x,y),g(x,y),(x,-2,2),(y,-2,2))
```

obtenemos la figura 2.2, que nos da una idea de qué conjunto se trata.

2.1.4. Plano tangente a la gráfica de una función $z = f(x, y)$

Dada una función $f : A \subseteq \mathbb{R}^2 \to \mathbb{R}$, hemos visto como podemos visualizar con Python la gráfica del conjunto

$$graf(f) = \{(x, y, z) \in \mathbb{R}^3 : z = f(x, y) \ \wedge \ (x, y) \in A\},$$

es decir la gráfica de una función real de dos variables. También cómo representar sus curvas de nivel dadas por la expresión

$$f(x, y) = c,$$

donde c es un número real. A continuación, vamos a ver cómo calcular el plano tangente a la superficie en un punto (x_0, y_0, z_0) de la misma.

Una primera advertencia es que no todo punto $(x_0, y_0, z_0) \in \mathbb{R}^3$ está en la superficie. Debe verificarse que $z_0 = f(x_0, y_0)$. Por ejemplo, dada la función $f(x, y) = \sin(xy)$ definida en todo el plano real, el punto $(0, 1, 0)$ está en la superficie ya que $f(0, 1) = 0$, mientras que el punto $(0, 1, 2)$ no lo está. Debemos siempre verificar que el punto esté contenido en la superficie. Veamos qué es el plano tangente.

Dados (x_0, y_0, z_0) en la superficie determinada por la gráfica de la función f de clase C^1, de la noción de diferencial tenemos que, dado $\mathbf{h} = (h_1, h_2) \in \mathbb{R}^2$, ésta es

$$\begin{aligned} Df(x_0, y_0)(h_1, h_2) &= Jf(x_0, y_0) \cdot \begin{pmatrix} h_1 \\ h_2 \end{pmatrix} \\ &= \left(\frac{\partial f}{\partial x}(x_0, y_0), \frac{\partial f}{\partial y}(x_0, y_0) \right) \cdot \begin{pmatrix} h_1 \\ h_2 \end{pmatrix} \\ &= \frac{\partial f}{\partial x}(x_0, y_0)h_1 + \frac{\partial f}{\partial y}(x_0, y_0)h_2. \end{aligned}$$

De la definición de diferencial tenemos que

$$\lim_{\mathbf{h} \to (0,0)} \frac{\left| f(x_0 + h_1, y_0 + h_2) - f(x_0, y_0) - \left(\frac{\partial f}{\partial x}(x_0, y_0)h_1 + \frac{\partial f}{\partial y}(x_0, y_0)h_2 \right) \right|}{||\mathbf{h}||} = 0,$$

o equivalentemente

$$\lim_{(x,y) \to (x_0,y_0)} \frac{\left| f(x, y) - f(x_0, y_0) - \left(\frac{\partial f}{\partial x}(x_0, y_0)(x - x_0) + \frac{\partial f}{\partial y}(x_0, y_0)(y - y_0) \right) \right|}{||(x, y)||} = 0.$$

Entonces

$$\lim_{(x,y) \to (x_0,y_0)} f(x, y) - f(x_0, y_0) - \left(\frac{\partial f}{\partial x}(x_0, y_0)(x - x_0) + \frac{\partial f}{\partial y}(x_0, y_0)(y - y_0) \right) = 0,$$

lo que significa que la función

$$z = f(x_0, y_0) + \frac{\partial f}{\partial x}(x_0, y_0)(x - x_0) + \frac{\partial f}{\partial y}(x_0, y_0)(y - y_0)$$

es una aproximación de $f(x, y)$. La gráfica de dicha aproximación es un plano que se llama plano tangente de f en $(x_0, y_0, f(x_0, y_0))$.

A continuación vamos a ver cómo calcular con Python el plano tangente de la función $f(x, y) = x^2 + y^3 + 3xy$ el el punto $(1, 1, 5)$ y vamos a hacer una representación conjunta de ambas funciones. Para ello debemos importar del módulo sympy las sentencias para derivar y

representar funciones gráficamente. Tecleamos

```
from sympy import symbols, diff
x,y=symbols('x y')
def f(x,y):
    return x**2+y**3+3*x*y
diff(f(x,y), x)
2*x+3*y
diff(f(x,y), y)
3*x+3*y**2
a=(2*x + 3*y).subs({x:1,y:1})
b=(3*x + 3*y**2).subs({x:1,y:1})
def p(x,y):
    return a*(x-1)+b*(y-1)+f(1,1)
p(x,y)
5*x + 6*y - 6
```

por lo que la ecuación del plano tangente es

$$z = 5x + 6y - 6.$$

Vamos a continuación a representar gráficamente ambas funciones en un entorno del punto $(1, 1)$, por ejemplo el cuadrado $[-5, 5] \times [-5, 5]$. Para ello tecleamos

```
from sympy.plotting import plot3d
plot3d(f(x,y),p(x,y), (x, -5, 5), (y, -5, 5))
```

obteniendo la gráfica de la figura 2.3.

Una representación gráfica en un conjunto más pequeño conteniendo al $(1, 1)$, por ejemplo el cuadrado $(0{,}5, 1{,}5) \times (0{,}5, 1{,}5)$ nos permite darnos cuenta de que efectivamente, el plano tangente aproxima a la función. Para ello tecleamos

```
plot3d(f(x,y),p(x,y), (x,0.5,1.5),(y,0.5,1.5))
```

y obtenemos la gráfica de la figura 2.4.

Actividad 82 *Determinar si los siguientes puntos pertencen a la gráfica de la función que se indica:*

1. *El punto* $(1, 1, 2)$ *a la gráfica* $z = x^2 + y^4 + 2xy$.
2. *El punto* $(1, 1, 0)$ *a la gráfica* $z = x^2 + y^3 - 2xy$.
3. *El punto* $(0, 1, 1)$ *a la gráfica* $z = e^x + yx$.

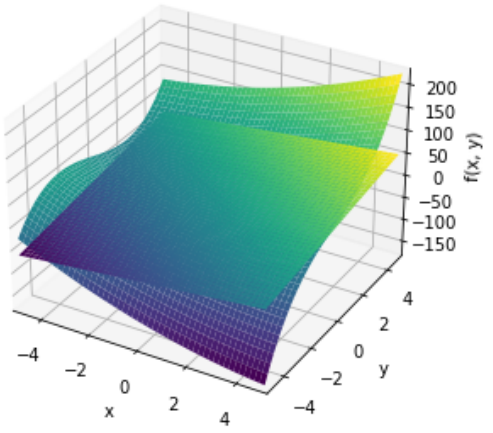


Figura 2.3: Representación conjunta de la función y el plano tangente a la gráfica de la función en el punto $(1, 1, 5)$.

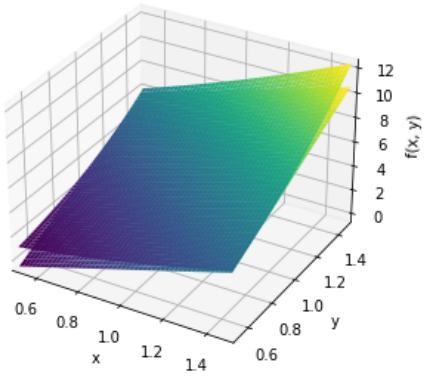


Figura 2.4: Representación conjunta del función y plano tangente en un conjunto más pequeño.

4. *El punto* $(1,1,2)$ *a la gráfica* $z = x^2 - y^2 - 5x + 2y - xy$.

Actividad 83 *Obtener el plano tangente a la gráfica de las siguientes funciones en los puntos que se indican:*

1. *La función* $z = x^2 + y^4 + 2xy$ *y el punto* $(0,1,1)$.
2. *La función* $z = x^2 + y^3 - 2xy$ *y el punto* $(6,-1,47)$.
3. *La función* $z = e^x + yx$ *y el punto* $(0,-1,-1)$.
4. *La función* $z = x^2 - y^2 - 5x$ *y el punto* $(-1,1,5)$.

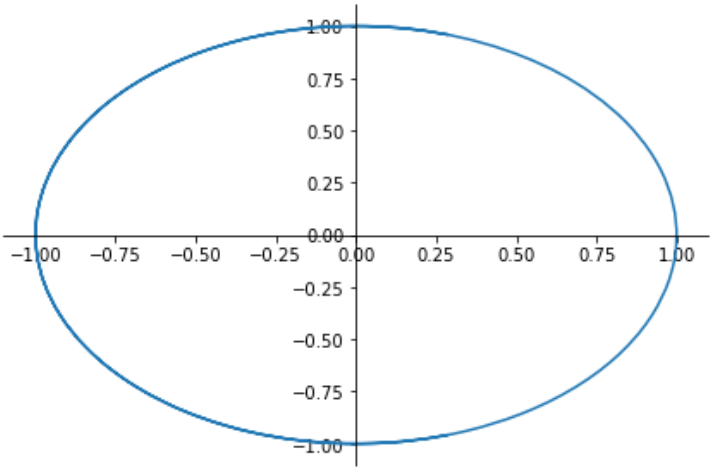
Actividad 84 *Dibujar conjuntamente las gráficas de las funciones y planos tangentes del ejercicio anterior en un dominio apropiado. Realizar un zoom de la representación gráfica anterior que muestre que el plano tangente es una aproximación de la función.*

2.1.5. Representación de curvas

Para representar curvas en el plano tenemos la sentencia plot_parametric. Lo podemos importar de sympy y debemos de indicar las coordenadas de la curva y el dominio donde deseamos representarla. Por ejemplo, para representar la curva $(x(t), y(t)) = (\cos t, \sin t)$ con $t \in (0, 2\pi)$ tecleamos

```
from sympy import symbols, plot_parametric, sin, cos
t=symbols('t')
plot_parametric((cos(t), sin(t)), (u, 0, 2*pi))
```

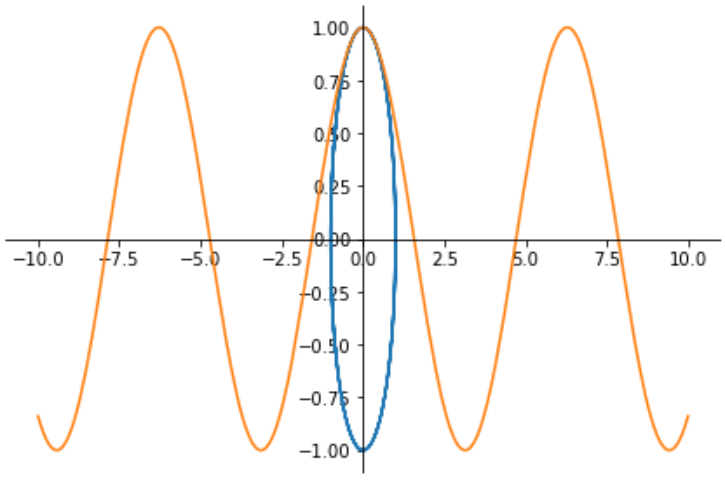
y obtenemos la gráfica



Como en los casos anteriores, podemos representar de forma análoga varias curvas a la vez, en el mismo dominio, o en dominios diferentes. Por ejemplo

```
from sympy import symbols, plot_parametric, sin, cos
t=symbols('t')
plot_parametric((cos(t), sin(t)), (t, cos(t)), (t, -10, 10))
```

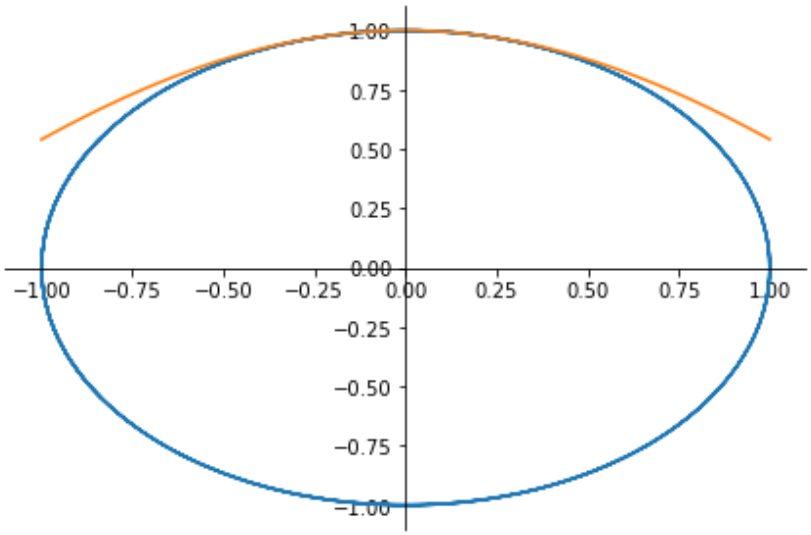
y obtener la gráfica siguiente, con dos curvas representadas en el mismo dominio.



En el siguiente ejemplo

```
from sympy import symbols, plot_parametric, sin, cos
t=symbols('t')
plot_parametric((cos(t), sin(t), (t, -10, 10)), (t, cos(t), (t, -1, 1)))
```

para obtener la gráfica de curvas en diferentes dominios



Como sabemos, es posible que una curva venga expresada en forma implícita. Por ejemplo, una circunferencia de centro (0,0) y radio 4 viene dada por la ecuación

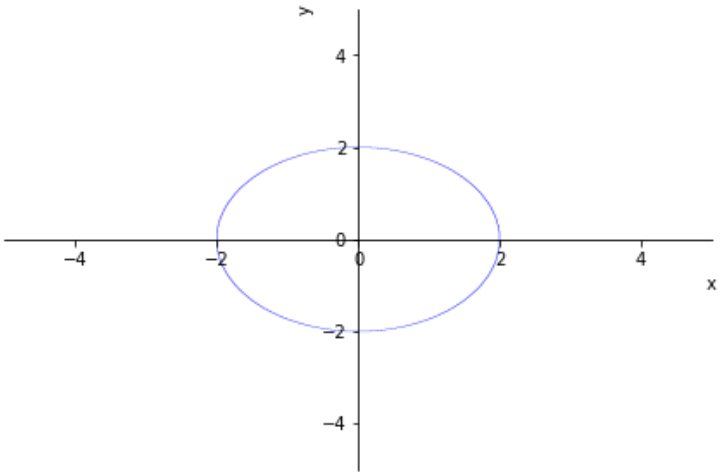
$$x^2 + y^2 = 4.$$

Para estos casos tenemos la sentencia plot_implicit, que debemos importar del módulo sympy.plotting. Para utlizar esta sentencia tenemos que importar de sympy la sentencia Eq(eq1,eq2) que permite introducir la ecuación eq1=eq2. No hace falta introducir entonces dominio alguno. Veámos

un ejemplo de como representar la curva anterior

```
from sympy import symbols
from sympy.plotting import plot_implicit
x,y=symbols('x y')
plot_implicit(Eq(x**2 + y**2, 4))
```

que da la curva



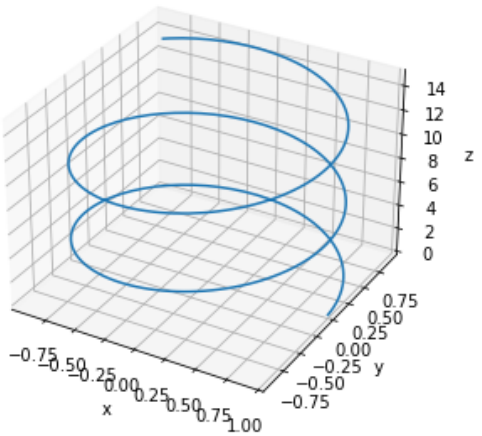
Para curvas en el espacio tridimensional tenemos la sentencia plot3d_parametric_line. Como en el caso de plot3d es necesario importarlo desde el módulo sympy.plotting. La sintaxis es similar a la de los casos anteriores pudiendo representarse una o varias curvas sobre el mismo o diferente dominio. Veamos un par de ejemplos. Para representar la curva

$$(x(t), y(t), z(t)) = (\cos t, \sin t, t), \quad t \in [0, 15]$$

tecleamos

```
from sympy import symbols, sin, cos
from sympy.plotting import plot3d_parametric_line
t=symbols('t')
plot3d_parametric_line(cos(t), sin(t), t, (t, 0, 15))
```

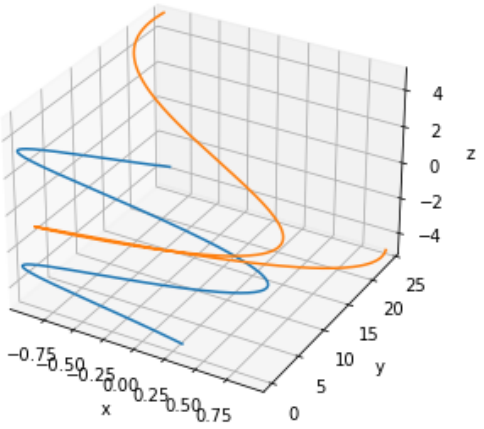
obteniéndose la gráfica



Otro ejemplo

```
from sympy import symbols, sin, cos
from sympy.plotting import plot3d_parametric_line
t=symbols('t')
plot3d_parametric_line((cos(t), sin(t), t, (t, -5, 5)),(sin(t), t**2, t, (t, -5, 5)))
```

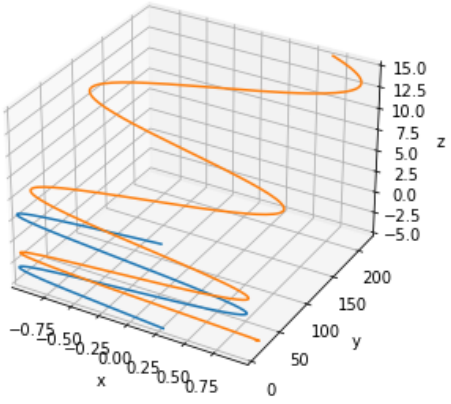
que representa dos curvas sobre el mismo dominio como se muestra en la en la siguiente gráfica



Un ejemplo de representación sobre dominios diferentes es

```
from sympy import symbols, sin, cos
from sympy.plotting import plot3d_parametric_line
t=symbols('t')
plot3d_parametric_line((cos(t), sin(t), t, (t, -5, 5)),(sin(t), t**2, t, (t, -5, 15)))
```

que da la gráfica



Actividad 85 *Representar gráficamente las siguientes curvas en el plano y el espacio:*

1. $\begin{cases} x(t) = \cos 2t \\ y(t) = \sin t \end{cases}$ *en el dominio* $[0, 2\pi]$.

2. $\begin{cases} x(t) = \frac{1}{t} \\ y(t) = t^2 \end{cases}$ *en el dominio* $[-1, 1]$.

3. $\begin{cases} x(t) = t^3 \\ y(t) = \sqrt{t} \end{cases}$ *en el dominio* $[0, 2]$.

4. $\begin{cases} x(t) = \cos t^2 \\ y(t) = \sin t^2 \\ z(t) = \sqrt{t} \end{cases}$ *en el dominio* $[0, 2\pi]$.

5. $\begin{cases} x(t) = t^2 \\ y(t) = t \end{cases}$ *en el dominio* $[0, 2]$.

2.1.6. Representación de superficies parametrizadas

Para representar superficies en forma paramétrica de la forma

$$(x(u, v), y(u, v), z(u, v)), \quad (u, v) \in (a, b) \times (c, d)$$

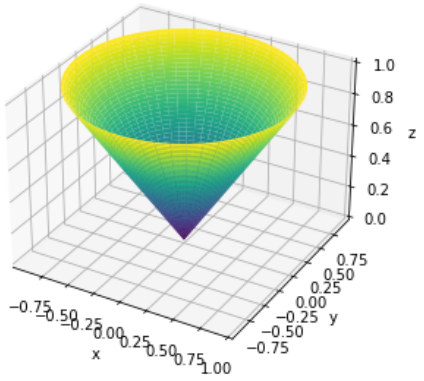
tenemos la sentencia plot3d_parametric_surface, que debemos importar del módulo sympy.plotting. Por ejemplo, si queremos representar la superficie

$$(x(u, v), y(u, v), z(u, v)) = (u \cos v, u \sin v, u), \quad v \in [0, 2\pi], \quad u \in [0, 1]$$

debemos teclear

```
from sympy import symbols, sin, cos
from sympy.plotting import plot3d_parametric_surface
u,v=symbols('u v')
plot3d_parametric_surface(u*cos(v), u*sin(v),u, (u, 0,1), (v,0,2*pi))
```

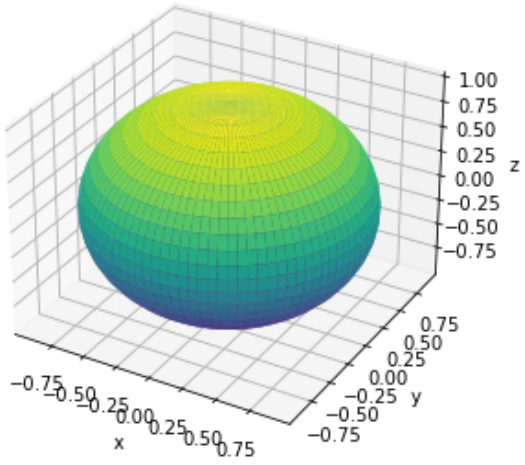
que nos da el cono



Para representar una esfera de radio uno y centro el origen de coordenadas escribimos

```
from sympy import symbols, sin, cos
from sympy.plotting import plot3d_parametric_surface
u,v=symbols('u v')
plot3d_parametric_surface(sin(u)*sin(v), cos(u)*sin(v),cos(v), (u, 0,pi), (v,0,2*pi))
```

que da la gráfica



Actividad 86 *Representar gráficamente las siguientes superficies parametrizadas:*

1. $\begin{cases} x(u,v) = u \\ y(u,v) = v \\ z(u,v) = u^2 + v^2 \\ (u,v) \in [-2,2] \times [-2,2]. \end{cases}$

2. $\begin{cases} x(u,v) = u \\ y(u,v) = v \\ z(u,v) = u^2 - v^2 \\ (u,v) \in [-4,4] \times [-4,4]. \end{cases}$

3. $\begin{cases} x(u,v) = u \\ y(u,v) = v \\ z(u,v) = e^{-(u^2+v^2)} \\ (u,v) \in [-2,2] \times [-2,2]. \end{cases}$

4. $\begin{cases} x(u,v) = u \\ y(u,v) = v \\ z(u,v) = \sin u^2 - \sin v^2 \\ (u,v) \in [-2,2] \times [-2,2]. \end{cases}$

5. $\begin{cases} x(u,v) = u \\ y(u,v) = v \\ z(u,v) = \sin(uv) \\ (u,v) \in [-2,2] \times [-2,2]. \end{cases}$

6. $\begin{cases} x(u,v) = 16 \sin u \cos v \\ y(u,v) = 10 \sin u \sin v \\ z(u,v) = 10 \cos u \\ (u,v) \in [0,\pi] \times [0,2\pi]. \end{cases}$

7. $\begin{cases} x(u,v) = 3\cos v \\ y(u,v) = 3\sin v \\ z(u,v) = u \\ (u,v) \in [0,10] \times [0,2\pi]. \end{cases}$

8. $\begin{cases} x(u,v) = 3\cos v \\ y(u,v) = 8\sin v \\ z(u,v) = u \\ (u,v) \in [0,10] \times [0,2\pi]. \end{cases}$

9. $\begin{cases} x(u,v) = v\cos u \\ y(u,v) = v\sin u \\ z(u,v) = v^2 \\ (u,v) \in [0,2\pi] \times [0,2]. \end{cases}$

10. $\begin{cases} x(u,v) = \cos u \cosh v \\ y(u,v) = \sin u \cosh v \\ z(u,v) = \sinh v \\ (u,v) \in [0,2\pi] \times [0,2]. \end{cases}$

11. $\begin{cases} x(u,v) = u\cos v \\ y(u,v) = u\sin v \\ z(u,v) = \log(\cos v + u) \\ (u,v) \in [0,3] \times [-\pi/2, \pi/2]. \end{cases}$

12. *El toro es una superficie producida al girar una circunferencia de radio r alrededor de un eje situado a una distancia a de la misma. Puede parametrizarse por*

$$\begin{cases} x(u,v) = (r\cos u + a)\cos v \\ y(u,v) = (r\cos u + a)\sin v \\ z(u,v) = r\sin u \\ (u,v) \in [0,2\pi] \times [0,2\pi]. \end{cases}$$

Obtener la gráfica del toro con $a = 3$ y $r = 1$.

2.2. Gráficas con el módulo matplotlib

2.2.1. Representación de funciones reales con el paquete matplotlib

Con el paquete de Python matplotlib podemos representar gráficas de la forma $y = f(x)$. Para ello debemos de hacer un mallado del intervalo en el que queramos hacer la representación, para lo cual usaremos el modulo **numpy**. Posteriomente utilizaremos el paquete **matplotlib**, en particular la sentecia subplots.

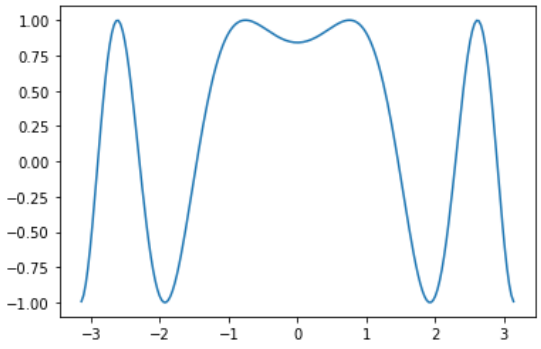


Figura 2.5: Representación gráfica de la función $\sin(x^2+1)$ en el intervalo $[-\pi, \pi]$.

Por ejemplo, vamos a representar la función $f(x) = \sin(x^2+1)$ en el intervalo $[-\pi, \pi]$. En primer lugar hacemos el mallado del intervalo con la sentencia linspace tecleando

```
from numpy import linspace,sin,pi
x = linspace(-pi, pi, 200)
y=sin(x**2+1)
```

Así, dividimos el intervalo en subintervalos, seleccionando 200 puntos equidistribuidos mediante la sentencia linspace. A continuación, calculamos los valores de la función en dichos puntos.

Una vez definidos los puntos en los que vamos a dibujar y el valor de la función en dichos puntos, procedemos a su representación con la sentencia subplots. Ésta tiene que ser importada de matplotlib.pyplot escribiendo

```
from matplotlib.pyplot import subplots
```

Finalmente, ejecutando conjuntamente

```
fig, ax = subplots()
ax.plot(x, y)
```

obtenemos la representación gráfica que se muestra en la figura 2.5.

Alternativamente, podemos definir la función mediante una función lambda tecleando

```
f=lambda x:sin(x**2+1)
```

y entonces la representación gráfica se obtiene tecleando

```
fig, ax = subplots()
ax.plot(x, f(x))
```

Aquí, la idea que subyace es la de crear un mallado en el conjunto donde se va a representar la función. A continuación se calcula el valor de la función en todos los puntos del mallado y se construyen los puntos en el plano de coordenadas (x,f(x)) donde x es un punto del mallado. Se

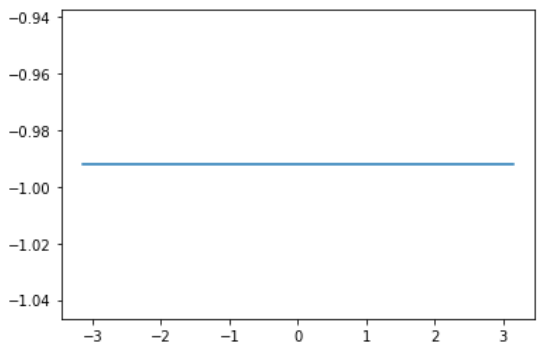


Figura 2.6: Gráfica de la función $\sin(1 + x^2)$ con un mallado insuficiente.

representan esos puntos y se unen mostrando la gráfica final. Obviamente, la gráfica obtenida depende de los puntos que se toman en el mallado ya que si no son suficiente no va a dar una buena representación gráfica. Por ejemplo, haciendo

```
from numpy import linspace,sin,pi
x = linspace(-pi, pi, 2)
y=sin(x**2+1)
```

es dcir, tomando un mallado con dos puntos y haciendo la representación gráfica tecleando

```
fig, ax = subplots()
ax.plot(x,y)
```

obtenemos la gráfica de la figura 2.6, que como vemos no es para nada la función que queremos representar. Al final, siempre es un problema de equilibrio entre el número de puntos del mallado y el tiempo de computación necesario para producir el resultado.

Si no se explicitan argumentos en la sentencia subplots, ésta define una figura y unos ejes. Esta sentencia admite otros argumentos para hacer gráficas variadas. Tecleando subplots(n,m) indicamos que vamos a representar conjuntamente $n \cdot m$ gráficas en n filas y m columnas. Por ejemplo, si ejecutamos

```
from numpy import linspace
x = linspace(-1, 1, 200)
```

y posteriormente tecleamos

```
from matplotlib.pyplot import subplots
fig, ((ax1, ax2, ax3), (ax4, ax5, ax6)) = subplots(2, 3)
ax1.plot(x, x)
ax2.plot(x, x**2)
ax3.plot(x, -x)
ax4.plot(x, -x**2)
ax5.plot(x, x**3)
ax6.plot(x, -x**4)
```

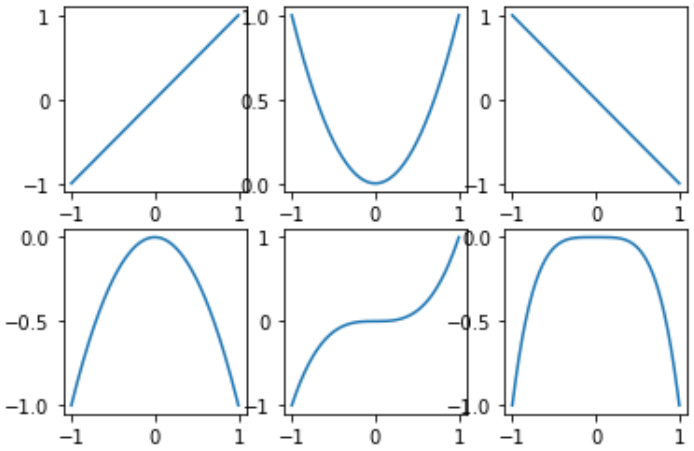


Figura 2.7: Varias representaciones gráficas.

obtenemos las representaciones gráficas de las funciones $y = x$, $y = x^2$, $y = -x$, $y = -x^2$, $y = x^3$ e $y = -x^4$ en el intervalo $[-1, 1]$ según muestra la figura 2.7.

Tecleando

```
from matplotlib.pyplot import subplots
fig, (ax1, ax2) = subplots(2, 1)
ax1.plot(x, x)
ax2.plot(x, x**2)
```

Se obtienen las figuras representadas una debajo de la otra, mientras que

```
from matplotlib.pyplot import subplots
fig, (ax1, ax2) = subplots(1, 2)
ax1.plot(x, x)
ax2.plot(x, x**2)
```

dará la representación de las funciones en una fila y una al lado de la otra.

Actividad 87 *Representar la siguientes funciones:*

1. $f(x) = \sin(x^2)$ *y* $g(x) = \sin^2 x$ *en el intervalo* $[-2\pi, 2\pi]$, *una gráfica encima de la otra.*
2. $f(x) = \sin(x^2)$ *y* $g(x) = \sin^2 x$ *en el intervalo* $[-2\pi, 2\pi]$, *una gráfica al lado de la otra.*
3. $f(x) = \sin(x^2)$ *y* $g(x) = \sin^2(x^2)$ *en el intervalo* $[-2\pi, 2\pi]$, *una gráfica encima de la otra.*
4. $f_1(x) = x^2 + 1$, $f_2(x) = x^3 + 1$, $f_3(x) = x^2 - 1$, *y* $f_4(x) = \sin^2(x^2)$ *en el intervalo* $[-2, 2]$, *en un matriz* 2×2.

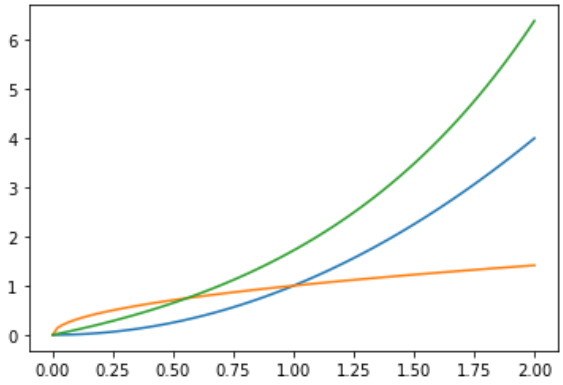


Figura 2.8: Representación conjunta de tres funciones.

También es posible representar varias gráficas en el mismo dibujo. Por ejemplo, supongamos que queremos representar las funciones $y = x^2$, $y = \sqrt{x}$ y $y = e^x - 1$ a la vez en el intervalo $[0, 1]$. Para ello tecleamos

```
from numpy import linspace, sqrt, exp
x = linspace(0, 1, 100)
y1=x**2
y2=sqrt(x)
y3=exp(x)-1
```

A continuación, hacemos la representación tecleando

```
from matplotlib.pyplot import subplots
fig, ax = subplots()
ax.plot(x,y1)
ax.plot(x,y2)
ax.plot(x,y3)
```

obteniéndose la gráfica de la figura 2.8.

Actividad 88 *Representar conjuntamente, es decir, en el mismo eje de coordenadas, la siguientes funciones:*

1. $f(x) = \sin(x^2)$ *y* $g(x) = \sin^2 x$ *en el intervalo* $[-2\pi, 2\pi]$.
2. $f(x) = x^2$ *y* $g(x) = \sin^2 x$ *en el intervalo* $[-2, 2]$.
3. $f(x) = \sin(x^2)$ *y* $g(x) = \sin^2(x^2)$ *en el intervalo* $[-5\pi, 2\pi]$.

2.2.2. Visualización de conjuntos planos

Python puede ayudarnos a visualizar recintos planos que nos puedan ser útilies a la hora de, por ejemplo, resolver integrales dobles. Para ello debemos de hacer un mallado de una región del plano, para lo cual usaremos el modulo **numpy**. Definiremos las funciones mediante la sentencia de Python **lambda**, y posteriomente utilizaremos el paquete **matplotlib**, en particular las sentecias contour y contourf.

Veamos con un ejemplo práctico cómo hacer estas representaciones gráficas. Por ejemplo, vamos a representar circunferencias concéntricas centradas en el origen de coordenadas y de diferentes radios. Como sabemos, vendrán dadas por la ecuación

$$x^2 + y^2 = R^2,$$

donde R es el radio de la circunferencia.

En primer lugar seleccionamos un rectángulo del plano donde vamos a representar las figuras y haceos un mallado del mismo. Para ello recurrimos al módulo numpy, tecleando

```
from numpy import linspace, meshgrid
x = linspace(-1.5, 1.5, 200)
y = linspace(-1.5, 1.5, 200)
X, Y = meshgrid(x, y)
```

dividimos el intervalo $[-1{,}5, 1{,}5]$ en 200 subintervalos mediante la sentencia linspace, y con la sentencia meshgrid hacemos un mallado de 200 por 200 puntos del cuadrado $[-1{,}5, 1{,}5] \times [-1{,}5, 1{,}5]$, que es la región donde vamos a representar las circunferencias.

Una vez definida la región, definimos la función $f(x, y) = x^2 + y^2$. De esta función vamos a representar sus curvas de nivel de la forma $f(x, y) = c$ para distintos valores de c. Evidentemente, $c = R^2$, por lo que será el cuadrado del radio de cada circunferencia. Definimos la función usando la sentencia lambda tecleando

```
f = lambda x, y: x**2 + y**2
```

Finalmente, importamos la sentencia contour del paquete matplotlib.pyplot. Esta sentencia tiene las entradas

```
contour(X, Y, f(X, Y), lista, colors='xxxx')
```

donde las tres primeras son las variables de la malla y la función, lista es una lista de los distintos valores que toma c, y colors se usa para indicar el color a utilizar en inglés, predefinidos en el paquete. Así, tecleando

```
from matplotlib.pyplot import contour
contour(X, Y, f(X, Y), [1], colors='blue')
```

representamos la circunferencia $x^2 + y^2 = 1$ en la figura 2.9

Si queremos representar circunferencias de distintos radios, tecleamos

```
contour(X,Y,f(X,Y),[0.1,0.3,0.5,0.7,1],colors='blue')
```

obteniendo el resultado de la figura 2.10.

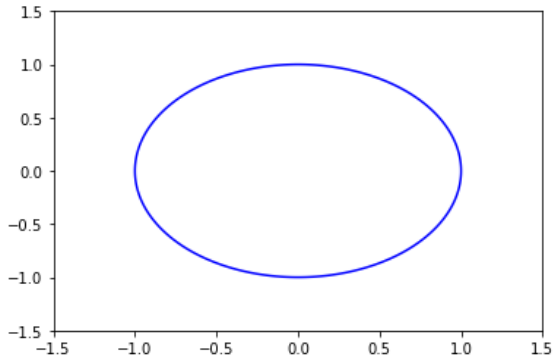


Figura 2.9: Circunferencia de radio 1.

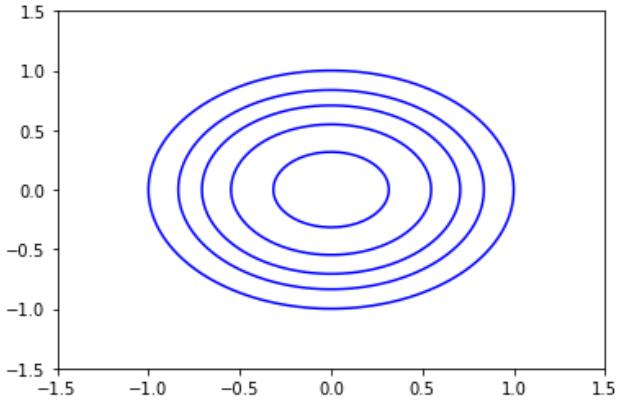


Figura 2.10: Circunferencias de distintos radios.

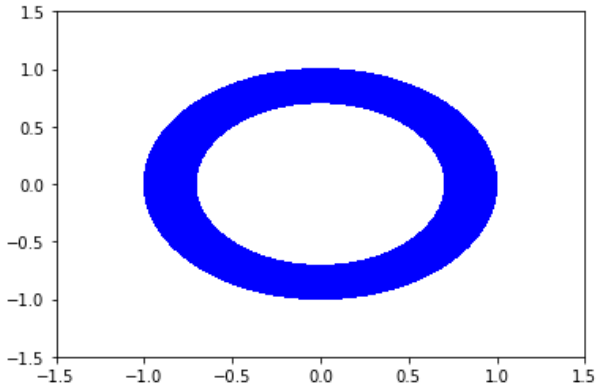


Figura 2.11: Anillo comprendido entre dos circunferencias.

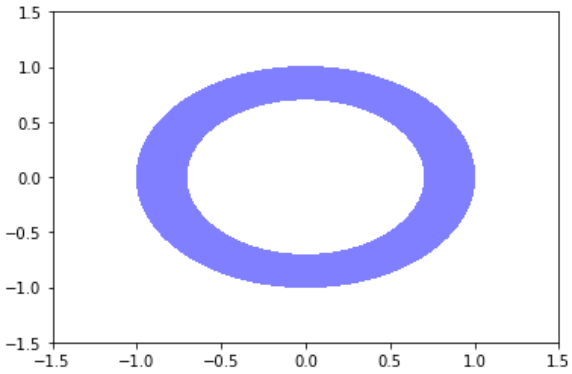


Figura 2.12: Anillo con una menor opacidad.

Si lo que deseamos es dibujar las regiones entre dos valores, es decir una anillo, tenemos la sentencia contourf, de sintaxis parecida. Por ejemplo, tecleando

```
from matplotlib.pyplot import contour
contourf(X, Y, f(X, Y), [0.5,1], colors='blue')
```

obtenemos en la figura 2.11 el anillo comprendido entre las circunferencias de radio $\sqrt{0{,}5}$ y 1.

Ambas sentencias pueden llevar además una componente adicional, llamada **alpha**, para determinar la opacidad, y que viene dada por un número comprendido entre 0 y 1, de manera que 0 es nada opaco y 1 lo es mucho. Por ejemplo,

```
contourf(X,Y,f(X,Y),[0.5,1],colors='blue', alpha=0.5)
```

nos da el mismo anillo de la figura 2.11 con una menor opacidad, como puede verse en la figura 2.12.

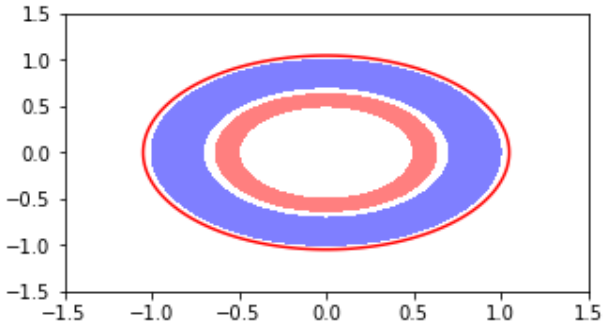


Figura 2.13: Tres gráficas combinadas.

Los límites contenidos en la tabla de contourf pueden ser $-\infty$ (escrito '-inf') y $+\infty$ (introducido como 'inf'). Por ejemplo, tecleando

contourf(X,Y,f(X,Y),[0.5,1],colors='blue')

obtendremos el disco exterior a la circunferencia $x^2 + y^2 = 1$.

Finalmente, pueden combinarse varias gráficas a la vez usando la sentencia **figure**. Para ello, hemos de escribir las gráficas que queremos y ejecutarlas todas a la vez después de la sentencia figure, que sigue la estructura figure(figsize=(a,b)), donde a y b son las longitudes del rectángulo en el que se van a representar las figuras. Por ejemplo, tecleando

```
from matplotlib.pyplot import contour, contourf, figure
figure(figsize=(5, 2.7))
contourf(X, Y, f(X, Y), [0.5,1], colors='blue',alpha=0.5)
contourf(X, Y, f(X, Y), [0.25,0.4], colors='red',alpha=0.5)
contour(X,Y,f(X,Y),[1.1],colors='red')
```

obtenemos la gráfica conjunta de la figura 2.13.

Es necesario aclarar que precisamos importar las funciones del modulo numpy, y no del sympy. Por ejemplo, si queremos usar la exponencial, ésta debe ser importada de numpy, en otro caso dará error.

Otra cuestión a tener en cuenta es que el dominio en el que dibujar las curvas normalmente se determina por tanteo. Es decir, en los ejemplos anteriores sabemos que las circunferencias que hemos utilizado están contenidas en el cuadrado $[-2{,}5, 2{,}5]$. Para otras curvas es posible que este conjunto dominio no esté claro, y haya que ir variando éste hasta obtener una buena representación gráfica.

Veamos cómo representar el conjunto

$$A = \{(x, y) \in \mathbb{R}^2 : x^2 + y^2 < 1 \text{ y } x > 0\}.$$

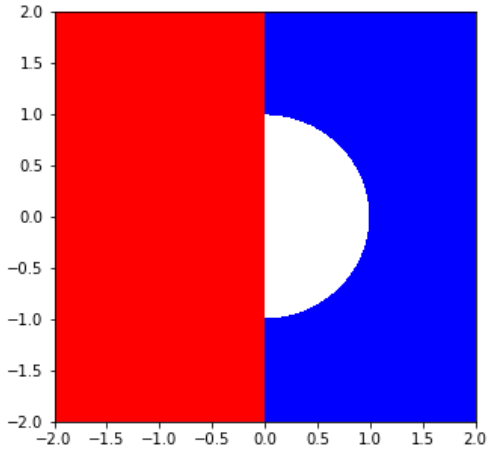


Figura 2.14: El conjunto A, que es la parte del dibujo que queda sin colorear.

Para ello tecleamos

```
f1 = lambda x, y: x**2 + y**2
f2 = lambda x, y: x
from numpy import linspace, meshgrid
y = linspace(-2, 2, 200)
x = linspace(-2, 2, 200)
X, Y = meshgrid(x, y)
from matplotlib.pyplot import contourf, figure
figure(figsize=(5, 5))
contourf(X, Y, f1(X, Y), [1,'inf'], colors='blue')
contourf(X, Y, f2(X, Y), ['-inf',0], colors='red')
```

El conjunto A es la región que queda sin colorear en la figura 2.14.

Si el conjunto fuera de la forma

$$A = \{(x, y) \in \mathbb{R}^2 : x^2 + y^2 < 1 \text{ y } x \geq 0\},$$

es decir, las desigualdades no fueran estrictas, podemo incluir la frontera correspondiente tecleando

```
f1 = lambda x, y: x**2 + y**2
f2 = lambda x, y: x
from numpy import linspace, meshgrid
y = linspace(-2, 2, 200)
x = linspace(-2, 2, 200)
X, Y = meshgrid(x, y)
from matplotlib.pyplot import contour, contourf, figure
figure(figsize=(5, 5))
contourf(X, Y, f1(X, Y), [1,'inf'], colors='blue',alpha=0.5)
contourf(X, Y, f2(X, Y), ['-inf',0], colors='red',alpha=0.5)
contour(X,Y,f2(X,Y),[0])
```

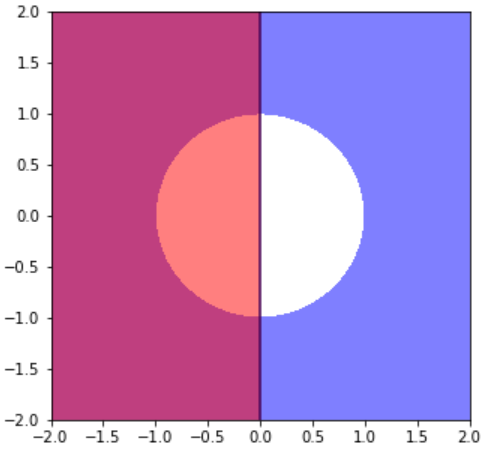


Figura 2.15: Recinto con una frontera incorporada.

obteniéndose la figura 2.15 donde la frontera $x = 0$ aparece representada, a la vez que se difuminan los colores de las otras dos regiones.

Actividad 89 *Dibujar las siguientes curvas de nivel:*

1. $x^2 + y^4 + 2xy = c$, *con* $c \in \{0{,}1, 0{,}5, 1\}$.
2. $x^2 + y^3 - 2xy = c$, *con* $c \in \{1, 2, 3\}$.
3. $e^x + yx = c$, *con* $c \in \{1, 2, 5\}$.
4. $x^2 - y^2 - 5x + 2y - xy = c$, *con* $c \in \{0{,}1, 0{,}4, 1{,}1\}$.

Actividad 90 *Dibujar los recintos (sólo cuando sea posible y tomando las constantes* c_1 *y* c_2 *dos a dos):*

1. $c_1 \leq x^2 + y^4 + 2xy \leq c_2$, *con* $c_1 \in \{0{,}1, 0{,}5, 1\}$ *y* $c_2 = \{2, 3\}$.
2. $x^2 + y^3 - 2xy \leq c$, *con* $c \in \{1, 2, 3\}$.
3. $c_1 \leq e^x + yx \leq c_2$, *con* $c_1 \in \{0{,}5, 2, 4\}$ *y* $c_2 \in \{1, 2, 5\}$.
4. $x^2 - y^2 - 5x \geq c$, *con* $c \in \{0{,}1, 0{,}4, 1{,}1\}$.

Actividad 91 *Dibujar los conjuntos siguientes:*

1. $A = \{(x, y) \in \mathbb{R}^2 : x + y \leq 1,\ x \geq 0,\ y \geq 0\}$.
2. *El conjunto* $A = \{(x, y) \in \mathbb{R}^2 : x^2 \leq y \leq x\}$.
3. $A = \{(x, y) \in \mathbb{R}^2 : x^2 + y^2 \leq 1,\ x \geq 0,\ y \leq 0\}$.
4. *El conjunto* A *comprendido entre la función* $y = x^3$ *y la recta* $y = x$.
5. *La parte del triángulo de vértices* $(0, 0)$, $(2, 0)$ *y* $(1, 1)$ *que está por debajo de la curva* $y = x^4$.

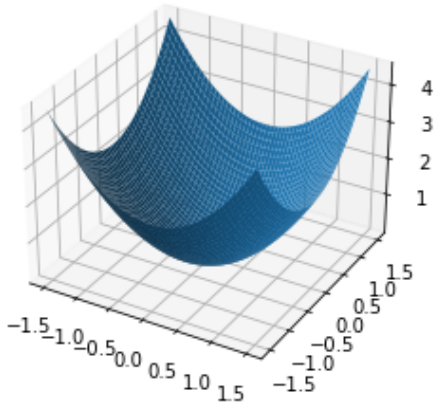


Figura 2.16: Gráfica de la función $f(x, y) = x^2 + y^2$ en $[-1{,}5, 1{,}5] \times [-1{,}5, 1{,}5]$.

2.2.3. Representación de funciones en 3D

Veamos como representar ahora funciones en varias variables. Por ejemplo $z = x^2 + y^2$ en el recinto $[-1{,}5, 1{,}5] \times [-1{,}5, 1{,}5]$. En primer lugar, hacemos un mallado del recinto y calculamos el valor de la función en los puntos del mallado tecleando

```
from numpy import linspace, meshgrid
x = linspace(-1.5, 1.5, 200)
y = linspace(-1.5, 1.5, 200)
X, Y = meshgrid(x, y)
Z=X**2+Y**2
```

Posteriormente, utilizamos la sentencia subplots, indicando en el argumento que vamos a hacer una representación en tres dimensiones, tal y como se indica en el siguiente ejemplo. Vemos además que utilizamos plot_surface ya que queremos representar una superficie. Tecleando

```
from matplotlib.pyplot import subplots
fig, ax = subplots(subplot_kw={"projection": "3d"})
ax.plot_surface(X, Y, Z)
```

obtenemos la gráfica de la figura 2.16.

A veces es útil representar la figura en un dominio circular. Para ello necesitamos coordenadas polares. Veámos como proceder en ese caso con un ejemplo. Tomamos la función $f(x, y) = \sin(-xy)$ y vamos a representarla en el círculo de centro el origen de coordenadas y radio 1. En primer lugar tecleamos

```
from numpy import linspace, newaxis, pi
r = linspace(0.125, 1.0, 8)
ang = linspace(0, 2*pi, 36, endpoint=False)[..., newaxis]
```

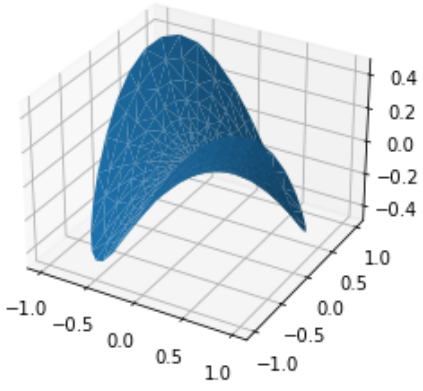


Figura 2.17: Gráfica en un recinto circular.

Con esto definimos las coordenadas polares. El radio será r y el ángulo lo hemos denotado por ang. Hemos toamdo 8 valores para el radio y 36 para el ángulo y hacemos un mallado en las coordenadas polares (r,ang). La sentencia endpoint=False es para no incluir el valor de 2π, porque puede dar poblemas por el teorema de la función inversa.

A continuación, pasamos a coordenadas cartesianas tecleando

```
from numpy import sin, cos, append
x = append(0, (r*cos(ang)).flatten())
y = append(0, (r*sin(ang)).flatten())
z = sin(-x*y)
```

Aquí, flatten se usa para colapsar el array en una dimensión. Notar que también hemos definido los puntos (x,y,z) donde vamos a dibujar la función. Para dibujarla escribimos

```
from matplotlib.pyplot import subplots
fig, ax = subplots(subplot_kw={'projection': '3d'})
ax.plot_trisurf(x, y, z)
```

donde ahora usamos la sentencia plot_trisurf en vex de plot_surface. Aquí la diferencia está en cómo es el mallado en cada caso, en uno triangular y en el otro rectangular. Obtenemos la gráfica de la figura 2.17.

Es interesante ver cómo quedaría la anterior gráfica si hubiéramos usado plot_surface. Para ello tecleamos

```
from numpy import meshgrid
X, Y = meshgrid(x, y)
Z=sin(-X*Y)
```

para el mallado, y

```
from matplotlib.pyplot import subplots
fig, ax = subplots(subplot_kw={'projection': '3d'})
ax.plot_surface(X, Y, Z)
```

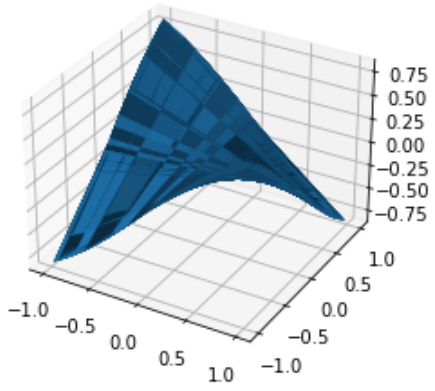


Figura 2.18: Otra representación de la función $\sin(-xy)$ usando plot_surface.

y obtenemos la gráfica de la figura 2.18, que como vemos es bastante peor. En cualquier caso, tenemos dos posibilidades para representar funciones de dos variables en el espacio y siempre podemos usar la que mejor resultado proporcione.

Actividad 92 *Representar la siguientes funciones en los recintos que se indican:*

1. $f(x,y) = \sin(yx^2)$ *en* $[0,1] \times [-1,1]$.
2. $f(x,y) = y + x^2 + xy)$ *en* $[-1,1] \times [-1,1]$.
3. $f(x,y) = xe^{x^2+y^2}$ *en* $[0,2] \times [-1,1]$.
4. $f(x,y) = \log(yx^2)$ *en* $[-1,1] \times [1,2]$.

Actividad 93 *Representar la siguientes funciones en los recintos que se indican:*

1. $f(x,y) = \sin(y + x^2)$ *en el círculo de radio 2 y centro* $(0,0)$.
2. $f(x,y) = y + x^2$ *en el círculo de radio 1 y centro* $(1,0)$.
3. $f(x,y) = \cos(yx)$ *en el círculo de radio 2 y centro* $(0,1)$.
4. $f(x,y) = x^2 + y^3$ *en el círculo de radio 3 y centro* $(-1,1)$.

Es posible visualizar varias gráfica a la vez. Por ejemplo, dada la función $f(x,y) = x^2 + y^3 + 3xy$ veremos que el punto $(1,1,5)$ está en la superficie definida por $z = f(x,y)$ y el plano tangente en dicho punto viene dada por la ecuación $z = 5x + 6y - 6$ (ver la sección 2.1.4). Vamos

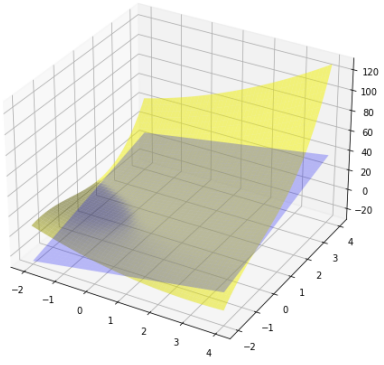


Figura 2.19: Representación conjunta de la función y el plano tangente.

a hacer una representación conjunta de ambas superficies utilizando la sentencia figure. Para ello vamos a selecionar un dominio centrado en $(1, 1)$, por ejemplo $[-2, 4] \times [-2, 4]$. Ejecutamos

```
from numpy import linspace, meshgrid
f = lambda x, y: x**2 + y**3 + 3*x*y
p = lambda x, y: 5*x + 6*y-6
x = linspace(-2, 4, 50)
y = linspace(-2, 4, 50)
X, Y = meshgrid(x, y)
```

y posteriormente tecleando

```
from matplotlib.pyplot import figure, show
fig = figure(figsize=(7, 6))
ax = fig.add_axes([0, 0, 1, 1], projection='3d')
ax.plot_surface(X, Y, f(X, Y), color='yellow', alpha=0.5)
ax.plot_surface(X, Y, p(X, Y), color='blue', alpha=0.25)
show()
```

obtenemos la gráfica de la figura 2.19.

Haciendo el conjunto donde representamos las funciones más pequeño, por ejemplo $[0, 2] \times [0, 2]$ podemos ver como las gráficas de las funciones se parecen mucho ya que el plano tangente es una aproximación local lineal de la función. Simplemente tecleamos

```
from numpy import linspace, meshgrid
f = lambda x, y: x**2 + y**3 + 3*x*y
p = lambda x, y: 5*x + 6*y-6
x = linspace(0, 2, 50)
y = linspace(0, 2, 50)
X, Y = meshgrid(x, y)
```

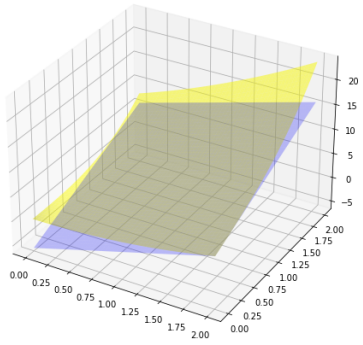


Figura 2.20: Representación gráfica de plano tangente y función en un dominio más reducido.

y

```
from matplotlib.pyplot import figure, show
fig = figure(figsize=(7, 6))
ax = fig.add_axes([0, 0, 1, 1], projection='3d')
ax.plot_surface(X, Y, f(X, Y), color='yellow', alpha=0.5)
ax.plot_surface(X, Y, p(X, Y), color='blue', alpha=0.25)
show()
```

para obtener la representación de la gráfica 2.20.

Actividad 94 *Obtener el plano tangente a la gráfica de las siguientes funciones en los puntos que se indican:*

1. *La función* $z = x + y^4 + 2xy$ *y el punto* $(0, 1, 1)$.
2. *La función* $z = x^2 + y^3$ *y el punto* $(1, -1, 2)$.
3. *La función* $z = e^x + \sin(yx)$ *y el punto* $(0, -1, 1)$.
4. *La función* $z = x^2 - y^2 + 5x$ *y el punto* $(-1, 1, 5)$.

2.2.4. Visualización de conjuntos en el espacio

Veamos cómo usar las sentencias anteriores para visualizar conjuntos en $\mathbb{R}^3$. Por ejemplo, tomamos el conjunto

$$A = \{(x, y, z) \in \mathbb{R}^3 : x + y \leq z \leq 4 - x^2 - xy,\ x, y \in [0, 1]\}$$

y vamos a visualizarlo. Para ello definimos las funciones siguientes, junto con un mallado tecleando

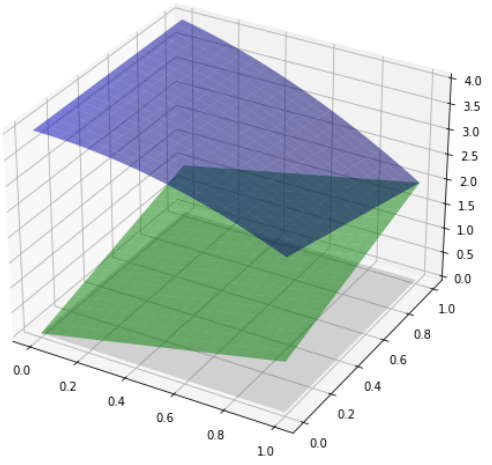


Figura 2.21: Gráfica del recinto A.

```
from numpy import linspace, meshgrid
f = lambda x, y: 4 - x**2 - x*y
g = lambda x, y: x + y
h = lambda x,y: 0.000001*x
x = linspace(0, 1, 20)
y = linspace(0, 1, 20)
X, Y = meshgrid(x, y)
```

Aquí la función h la definimos para mostrar el dominio de definición. Tomamos una función como vemos con coordenada z prácticamente nula. A continuación, tecleamos

```
from matplotlib.pyplot import figure, show
fig = figure(figsize=(7, 6))
ax = fig.add_axes([0, 0, 1, 1], projection='3d')
ax.plot_surface(X, Y, h(X, Y), color='gray', alpha=0.25)
ax.plot_surface(X, Y, f(X, Y), color='blue', alpha=0.5)
ax.plot_surface(X, Y, g(X, Y), color='green', alpha=0.5)
show()
```

y obtenemos el recinto que se muestra en la figura 2.21. En la representación aparecen todos los elementos definidos entre la sentencias figure y show. La primera sentencia fig.add_axes([0, 0, 1, 1], projection='3d') define unos ejes en tres dimensiones donde la tupla [0,0,1,1] representan la dimensiones izquierda, suelo, anchura y altura de los mismos. Una vez definidos los ejes ax se les va añadiendo superficies con la sentencia plot_surface, que tiene la posibilidad de elegir color y opacidad.

Podemos embellecer el recinto si además tecleamos los siguiente

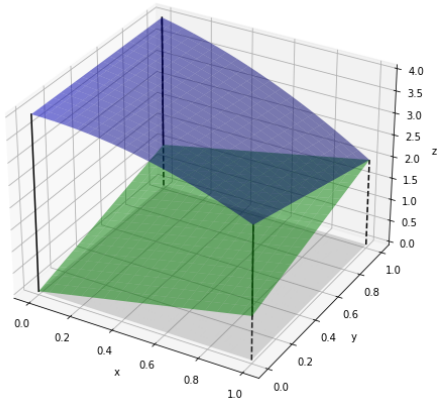


Figura 2.22: Recinto en el espacio tridimensional.

```
from matplotlib.pyplot import figure, show
fig = figure(figsize=(7, 6))
ax = fig.add_axes([0, 0, 1, 1], projection='3d')
ax.plot_surface(X, Y, h(X, Y), color='gray', alpha=0.25)
ax.plot_surface(X, Y, f(X, Y), color='blue', alpha=0.5)
ax.plot_surface(X, Y, g(X, Y), color='green', alpha=0.5)
ax.plot3D([0,0], [0,0], [f(0,0),g(0,0)], color='black')
ax.plot3D([0,0], [1,1], [0,f(0,1)], color='black', linestyle='dashed')
ax.plot3D([0,0], [1,1], [f(0,1),g(0,1)], color='black')
ax.plot3D([1,1], [0,0], [0,f(1,0)], color='black', linestyle='dashed')
ax.plot3D([1,1], [0,0], [f(1,0),g(1,0)], color='black')
ax.plot3D([1,1], [1,1], [0,f(1,1)], color='black', linestyle='dashed')
ax.plot3D([1,1], [1,1], [f(1,1),g(1,1)], color='black')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
show()
```

y obtenemos la gráfica de la figura 2.22. Aquí hemos añadido etiquetas en los ejes con las sentencias set_xlabel, set_ylabel y set_zlabel. La sentencia plot3($[x_1,x_2]$,$[y_1,y_2]$,$[z_1,z_2]$) dibuja un segmento uniendo los puntos (x_1, y_1, z_1) y (x_2, y_2, z_2). Entre diversas opciones, se puede elegir color (por ejemplo color='black') y tipo de línea (por ejemplo linestyle='dashed'), aunque si no se pone nada en la línea se dibuja continua, y la elección dashed implica que ésta se representará con puntos.

Consideremos ahora el siguiente conjunto

$$B = \{(x, y, z) \in \mathbb{R}^3 : x^2 + y^2 < z < 2 - x^2 - y^2\}$$

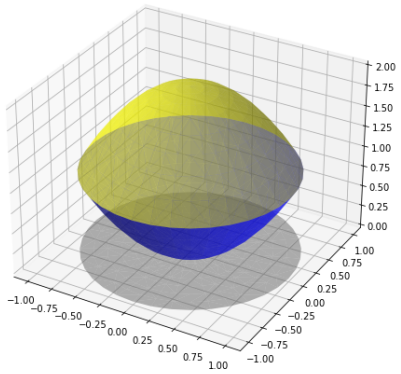


Figura 2.23: Gráfica del conjunto B.

tal que B es acotado. La intersección de las superficies $z = x^2 + y^2$ y $z = 2 - x^2 - y^2$ es la circunferencia $x^2 + y^2 = 1$, por lo que un recinto plano para representarlo es el círculo de radio 1 y centro $(0, 0)$. Por tanto utilizamos coordenadas polares y representación triangular para una mejor representación gráfica. Tecleamos

```
from numpy import linspace, newaxis, pi
r = linspace(0.125, 1.0, 8)
ang = linspace(0, 2*pi, 36, endpoint=False)[..., newaxis]
```

para definir las coordenadas polares. A continuación

```
from numpy import sin, cos, append
x = append(0, (r*cos(ang)).flatten())
y = append(0, (r*sin(ang)).flatten())
z1 = x**2+y**2
z2 = 2-x**2-y**2
z3=0.000000001*x
```

para pasar a coordenadas cartesianas y obtener los puntos en el espacio para hacer las representaciones gráficas. Finalmente

```
from matplotlib.pyplot import figure, show
fig = figure(figsize=(7, 6))
ax = fig.add_axes([0, 0, 1, 1], projection='3d')
ax.plot_trisurf(x, y, z1, color='blue', alpha=0.5)
ax.plot_trisurf(x, y, z2, color='yellow', alpha=0.5)
ax.plot_trisurf(x, y, z3, color='gray', alpha=0.5)
show()
```

para obtener la gráfica de la figura 2.23.

Actividad 95 *Visualizar los siguientes conjuntos en el espacio tridimensional.*

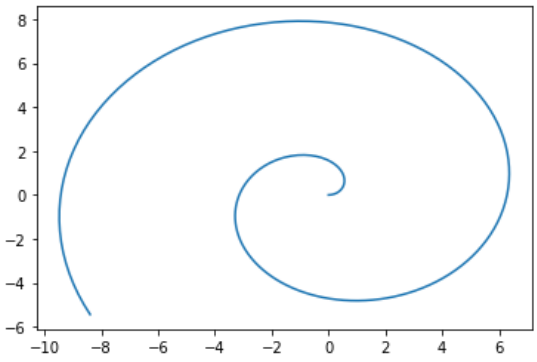


Figura 2.24: Curva parametrizada.

1. $A = \{(x, y, z) \in \mathbb{R}^3 : \sqrt{x^2 + y^2} < z < x + y,\ x, y \in [0, 1]\}$.
2. $A = \{(x, y, z) \in \mathbb{R}^3 : 0 < z < \sqrt{x^2 + 1},\ x \in [-1, 1], y \in [0, 1]\}$.
3. $A = \{(x, y, z) \in \mathbb{R}^3 : xy < z < 1,\ x \in [1, 2], y \in [-1, 0]\}$.
4. $A = \{(x, y, z) \in \mathbb{R}^3 : -\sqrt{x^2 + y^2} < z < \sqrt{x^2 + y^2},\ x^2 + y^2 \leq 1\}$.
5. $A = \{(x, y, z) \in \mathbb{R}^3 : -\sqrt{x^2 + y^2} < z < x^2 + y^2,\ x^2 + y^2 \leq 4\}$.

2.2.5. Representación de curvas planas

Para representar una curva plana seguiremos los mismos pasos que para representar funciones reales de una variable real. A modo de ejemplo, vamos a representar la curva dada por la parametrización

$$\begin{cases} x(t) = t\cos t \\ y(t) = t\sin t \end{cases}$$

donde $t \in [0, 10]$. Para ello tecleamos

```
from numpy import linspace,sin,cos
t = linspace(0, 10, 200)
x=cos(t)*t
y=sin(t)*t
```

para crear un mallado del intervalo $[0, 10]$ y calcular en los puntos del mismo los valores de la curva que vamos a representar. A continuación hacemos

```
from matplotlib.pyplot import subplots
fig, ax = subplots()
ax.plot(x, y)
```

de donde obtenemos la gráfica de la figura 2.24.

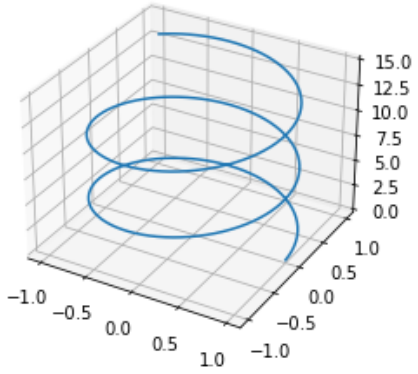


Figura 2.25: Curva en el espacio tridimensional.

Dado que las sentencias son análogas a las usadas para representar funciones reales, pueden realizarse las mismas operaciones que se podían realizar en ese caso. Por ejemplo, representar varias curvas a la vez en el mismo sistema de ejes coordenados, o agrupar las figuras matricialmente.

2.2.6. Representación de curvas en el espacio

Supongamos a continuación que tenemos una curva en el espacio, por ejemplo, dada por la parametrización

$$\begin{cases} x\left(t\right) = \cos t \\ y\left(t\right) = \sin t \\ z(t) = t \end{cases}$$

con $t \in [0, 15]$. Vamos a ver cómo representarla. En primer lugar, tecleamos

```
from numpy import linspace,sin,cos
t = linspace(0, 25, 200)
x=cos(t)
y=sin(t)
z=t
```

para hacer un mallado y calcular las coordenadas de la curva. Posteriormente, escribimos

```
from matplotlib.pyplot import figure, show
ax = figure().add_subplot(projection='3d')
ax.plot(x, y, z)
show()
```

obteniendo la gráfica de la figura 2.25.

Actividad 96 *Representar gráficamente las siguientes curvas en el plano y el espacio:*

1. $\begin{cases} x(t) = \cos 2t \\ y(t) = \sin t \end{cases}$ *en el dominio* $[0, 2\pi]$.

2. $\begin{cases} x(t) = \frac{1}{t} \\ y(t) = t^2 \end{cases}$ *en el dominio* $[-1, 1]$.

3. $\begin{cases} x(t) = t^3 \\ y(t) = \sqrt{t} \end{cases}$ *en el dominio* $[0, 2]$.

4. $\begin{cases} x(t) = \cos t^2 \\ y(t) = \sin t^2 \\ z(t) = \sqrt{t} \end{cases}$ *en el dominio* $[0, 2\pi]$.

5. $\begin{cases} x(t) = t^2 \\ y(t) = t \\ z(t) = 1 + t + t^2 \end{cases}$ *en el dominio* $[0, 2]$.

2.2.7. Vector y recta tangentes a una curva

A partir de ahora consideraremos curvas en el espacio, de forma que si es plana basta con quitar la variable z de lo que se describa a continuación. Dada una curva parametrizada $\gamma : [a, b] \subseteq \mathbb{R} \to \mathbb{R}^3$ suficientemente derivable, se define el vector tangente a la curva $\gamma(t) = (x(t), y(t), z(t))$ como su derivada $\gamma'(t) = (x'(t), y'(t), z'(t))$. Dado $t_0 \in [a, b]$ se define la recta tangente al punto $\gamma(t_0) = (x_0, y_0, z_0)$ como la curva

$$\begin{cases} x(t) = x_0 + x'(t_0)t \\ y(t) = y_0 + y'(t_0)t \\ z(t) = z_0 + z'(t_0)t \end{cases}$$

con $t \in \mathbb{R}$. Por ejemplo, vamos a considerar la curva anterior dada por la parametrización

$$\begin{cases} x(t) = \cos t \\ y(t) = \sin t \\ z(t) = t \end{cases}$$

con $t \in [0, 15]$ y el punto de la curva obtenido cuando $t = 2\pi$, esto es, $(1, 0, 2\pi)$. Su vector tangente en dicho punto es $\gamma'(t_0) = (0, 1, 1)$, por lo que su recta tangente vendrá dada por la parametrización

$$\begin{cases} x(t) = 1 \\ y(t) = t \\ z(t) = 2\pi + t \end{cases}$$

con $t \in \mathbb{R}$. Para representarlas conjuntamente tecleamos

```
from numpy import linspace,sin,cos
t = linspace(0, 15, 200)
x=cos(t)
y=sin(t)
z=t
```

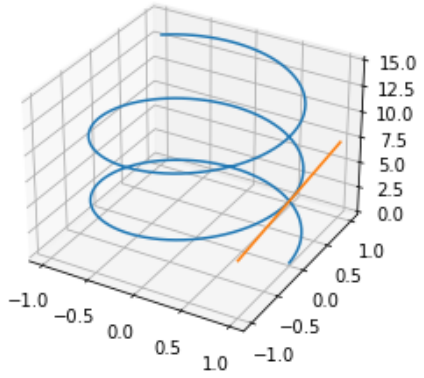


Figura 2.26: Recta tangente y curva.

para el mallado y obtención de los puntos de la curva,

```
from numpy import linspace,pi
t1 = linspace(-1, 1, 100)
x1=1+0*t1
y1=t1
z1=2*pi+t1
```

para el mallado y cómputo de los puntos de la recta tangente. Para representarlos conjuntamente tecleamos

```
from matplotlib.pyplot import figure, show
ax = figure().add_subplot(projection='3d')
ax.plot(x, y, z)
ax.plot(x1, y1, z1)
show()
```

obteniendo la representación conjunta de la figura 2.26.

2.2.8. Representación de superficies

Para representar una superficie parametrizada, por ejemplo

$$\begin{cases} x\,(u,v) = 10v\cos u \\ y\,(u,v) = 10v\sin u \\ z(u,v) = v \end{cases}$$

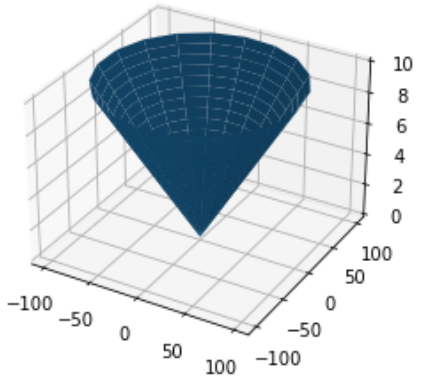


Figura 2.27: Representación gráfica de un cono.

donde $u \in [0, 2\pi]$ y $v \in [0, 10]$ se procede de un modo análogo al que se seguía para representar funciones. Empezamos definiendo el mallado y las funciones a representar tecleando

```
from numpy import linspace, meshgrid, sin, cos, pi
u = linspace(0, 2*pi, 20)
v = linspace(0, 10, 20)
U, V = meshgrid(u, v)
X=10*V*cos(U)
Y=10*V*sin(U)
Z=V
```

Posteriormente tecleamos

```
from matplotlib.pyplot import subplots
fig, ax = subplots(subplot_kw={"projection": "3d"})
ax.plot_surface(X, Y, Z)
```

obteniendose el cono de la figura 2.27.

De nuevo aquí pueden considerarse todos los casos que había cuando vimos la representación de funciones de la forma $z = f(x, y)$. Podemos representar varias superficies a la vez, y utilizar la sentencia para hacer mallados triangulares. Por ejemplo, vamos a obtener la superficie generada por las superficies

$$\begin{cases} x\,(u, v) = v \cos u \\ y\,(u, v) = v \sin u \\ z(u, v) = v \end{cases}$$

donde $u \in [0, 2\pi]$ y $v \in [0, 1]$ y

$$\begin{cases} x\,(u, v) = \cos u \sin v \\ y\,(u, v) = \sin u \sin v \\ z(u, v) = 1 + \cos v \end{cases}$$

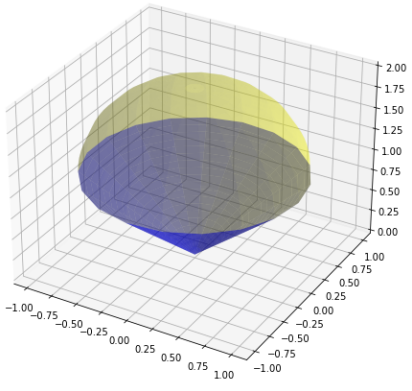


Figura 2.28: Superficie unión de una semiesfera y un cono.

con $u \in [0, 2\pi]$ y $v \in [0, \pi/2]$. Tecleamos

```
from numpy import linspace, meshgrid, sin, cos, pi
u1= linspace(0, 2*pi, 20)
v1 = linspace(0, 1, 20)
U1, V1 = meshgrid(u1, v1)
X1=V1*cos(U1)
Y1=V1*sin(U1)
Z1=V1
```

para el mallado de la primera superficie y

```
u2= linspace(0, 2*pi, 20)
v2 = linspace(0, pi/2, 20)
U2, V2 = meshgrid(u2, v2)
X2=sin(V2)*cos(U2)
Y2=sin(V2)*sin(U2)
Z2=1+cos(V2)
```

para el de la segunda. A continuación, tecleamos

```
from matplotlib.pyplot import figure, show
fig = figure(figsize=(7, 6))
ax = fig.add_axes([0, 0, 1, 1], projection='3d')
ax.plot_surface(X1, Y1, Z1, color='blue', alpha=0.5)
ax.plot_surface(X2, Y2, Z2, color='yellow', alpha=0.25)
show()
```

obteniendo el cucurucho de la figura 2.28.

Actividad 97 *Representar gráficamente las siguientes superficies parametrizadas:*

1. $\begin{cases} x(u,v) = u \\ y(u,v) = v \\ z(u,v) = u^2 + v^2 \\ (u,v) \in [-2,2] \times [-2,2]. \end{cases}$

2. $\begin{cases} x(u,v) = u \\ y(u,v) = v \\ z(u,v) = u^2 - v^2 \\ (u,v) \in [-4,4] \times [-4,4]. \end{cases}$

3. $\begin{cases} x(u,v) = u \\ y(u,v) = v \\ z(u,v) = e^{-(u^2+v^2)} \\ (u,v) \in [-2,2] \times [-2,2]. \end{cases}$

4. $\begin{cases} x(u,v) = u \\ y(u,v) = v \\ z(u,v) = \sin u^2 - \sin v^2 \\ (u,v) \in [-2,2] \times [-2,2]. \end{cases}$

5. $\begin{cases} x(u,v) = u \\ y(u,v) = v \\ z(u,v) = \sin(uv) \\ (u,v) \in [-2,2] \times [-2,2]. \end{cases}$

6. $\begin{cases} x(u,v) = 16 \sin u \cos v \\ y(u,v) = 10 \sin u \sin v \\ z(u,v) = 10 \cos u \\ (u,v) \in [0,\pi] \times [0,2\pi]. \end{cases}$

7. $\begin{cases} x(u,v) = 3 \cos v \\ y(u,v) = 3 \sin v \\ z(u,v) = u \\ (u,v) \in [0,10] \times [0,2\pi]. \end{cases}$

8. $\begin{cases} x(u,v) = 3 \cos v \\ y(u,v) = 8 \sin v \\ z(u,v) = u \\ (u,v) \in [0,10] \times [0,2\pi]. \end{cases}$

9. $\begin{cases} x(u,v) = v \cos u \\ y(u,v) = v \sin u \\ z(u,v) = v^2 \\ (u,v) \in [0,2\pi] \times [0,2]. \end{cases}$

10. $\begin{cases} x(u,v) = \cos u \cosh v \\ y(u,v) = \sin u \cosh v \\ z(u,v) = \sinh v \\ (u,v) \in [0, 2\pi] \times [0, 2]. \end{cases}$

11. $\begin{cases} x(u,v) = u \cos v \\ y(u,v) = u \sin v \\ z(u,v) = \log(\cos v + u) \\ (u,v) \in [0, 3] \times [-\pi/2, \pi/2]. \end{cases}$

12. El toro es una superficie producida al girar una circunferencia de radio r alrededor de un eje situado a una distancia a de la misma. Puede parametrizarse por

$$\begin{cases} x(u,v) = (r \cos u + a) \cos v \\ y(u,v) = (r \cos u + a) \sin v \\ z(u,v) = r \sin u \\ (u,v) \in [0, 2\pi] \times [0, 2\pi]. \end{cases}$$

Obtener la gráfica del toro con $a = 3$ y $r = 1$.

2.2.9. Vectores tangente y normal a una superficie parametrizada. Plano tangente

Dada una superficie dada por una parametrización plana de la forma

$$\Phi(u,v) = (x(u,v), y(u,v), z(u,v)),$$

$(u,v) \in \Omega \subseteq \mathbb{R}^2$, donde Ω es un conjunto abierto. Los vectores tangentes a la superfice vienen dados por

$$\mathbf{T}_u(u,v) = \left(\frac{\partial x(u,v)}{\partial u}, \frac{\partial y(u,v)}{\partial u}, \frac{\partial z(u,v)}{\partial u} \right)$$

y

$$\mathbf{T}_v(u,v) = \left(\frac{\partial x(u,v)}{\partial v}, \frac{\partial y(u,v)}{\partial v}, \frac{\partial z(u,v)}{\partial v} \right),$$

siendo el vector normal a la superficie

$$\mathbf{n}(u,v) = \mathbf{T}_u(u,v) \times \mathbf{T}_v(u,v).$$

Dado $(u_0, v_0) \in \Omega$, la superficie se dice regular en $(x_0, y_0, z_0) = (x(u_0, v_0), y(u_0, v_0), z(u_0, v_0))$ si

$$\mathbf{n}(u_0, v_0) = (n_1, n_2, n_3) = \mathbf{T}_u(u_0, v_0) \times \mathbf{T}_v(u_0, v_0) \neq \mathbf{0}.$$

En los puntos donde la superficie es regular se define el plano tangente como

$$n_1(x - x_0) + n_2(y - y_0) + n_3(z - z_0) = 0.$$

Veamos con un ejemplo como representar conjuntamente la superficie y el plano tangente a un punto regular de la misma. Por ejemplo, tomamos la superficie

$$\begin{cases} x(u,v) = 3\cos v \\ y(u,v) = 3\sin v \\ z(u,v) = u \\ (u,v) \in [0,10] \times [0,2\pi]. \end{cases}$$

Tomamos $u_0 = 5$ y $v_0 = 5\pi/4$, por lo que el punto $(-3\sqrt{2}/2, -3\sqrt{2}/2, 5)$ pertenece a la superficie. Sabemos que

$$\mathbf{T}_u(u,v) = (0,0,1)$$

y

$$\mathbf{T}_v(u,v) = (-3\sin v, 3\cos v, 0)$$

por lo que

$$\begin{aligned} \mathbf{n}(5,5\pi/4) &= (0,0,1) \times (3\sqrt{2}/2, -3\sqrt{2}/2, 0) \\ &= \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ 0 & 0 & 1 \\ 3\sqrt{2}/2 & -3\sqrt{2}/2 & 0 \end{vmatrix} = (3\sqrt{2}/2, 3\sqrt{2}/2, 0). \end{aligned}$$

El plano tangente es por tanto

$$3\sqrt{2}/2(x + 3\sqrt{2}/2) + 3\sqrt{2}/2(y + 3\sqrt{2}/2) = 0,$$

o equivalentemente

$$x + y + 3\sqrt{2} = 0,$$

que en ecuaciones paramétricas es

$$\begin{cases} x(u,v) = u \\ y(u,v) = -3\sqrt{2} - u \\ z(u,v) = v \\ (u,v) \in \mathbb{R}^2. \end{cases}$$

Vamos a representar conjuntamente ambas superficies parametrizadas. Para ello tecleamos

```
from numpy import linspace, meshgrid, sin, cos, pi
u1= linspace(0, 2*pi, 20)
v1 = linspace(0, 10, 20)
U1, V1 = meshgrid(u1, v1)
X1=3*cos(U1)
Y1=3*sin(U1)
Z1=V1
```

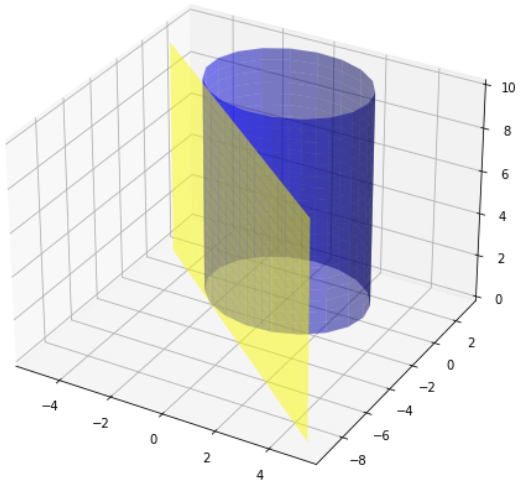


Figura 2.29: Plano tangente del cilindro.

para el mallado y determinación de los puntos de la superfice y

```
from numpy import linspace, meshgrid, sqrt
u2= linspace(-5, 5, 20)
v2 = linspace(-5, 5, 20)
U2, V2 = meshgrid(u2, v2)
X2=U2
Y2=-3*sqrt(2)-U2
Z2=V2
```

para el mallado y cálculo de los puntos de la superficie de una porción del plano tangente. Para representarlas conjuntamente tecleamos

```
from matplotlib.pyplot import figure, show
fig = figure(figsize=(7, 6))
ax = fig.add_axes([0, 0, 1, 1], projection='3d')
ax.plot_surface(X1, Y1, Z1, color='blue', alpha=0.5)
ax.plot_surface(X2, Y2, Z2, color='yellow', alpha=0.5)
show()
```

y obtenemos la representación gráfica de la figura 2.29.

Actividad 98 *Calcular el plano tangente de las siguiente superficies en los puntos que se indican y hacer una representación gráfica conjunta de la superficie y su plano tangente.*

1. $\begin{cases} x(u,v) = u \\ y(u,v) = v \\ z(u,v) = u^2 + v^2 \\ (u,v) \in [-2,2] \times [-2,2]. \end{cases}$ *en el punto* $(1,1,2)$.

2. $\begin{cases} x(u,v) = u \\ y(u,v) = v \\ z(u,v) = u^2 - v^2 \\ (u,v) \in [-4,4] \times [-4,4]. \end{cases}$ *en el punto* $(1,1,0)$.

3. $\begin{cases} x(u,v) = u \\ y(u,v) = v \\ z(u,v) = \sin u^2 - \sin v^2 \\ (u,v) \in [-2,2] \times [-2,2]. \end{cases}$ *en el punto* $(0,0,0)$.

4. $\begin{cases} x(u,v) = 16 \sin u \cos v \\ y(u,v) = 10 \sin u \sin v \\ z(u,v) = 10 \cos u \\ (u,v) \in [0,\pi] \times [0,2\pi]. \end{cases}$ *en el punto* $(0,0,10)$.

5. $\begin{cases} x(u,v) = 3 \cos v \\ y(u,v) = 8 \sin v \\ z(u,v) = u \\ (u,v) \in [0,10] \times [0,2\pi]. \end{cases}$ *en el punto* $(3,0,1)$.

6. $\begin{cases} x(u,v) = v \cos u \\ y(u,v) = v \sin u \\ z(u,v) = v^2 \\ (u,v) \in [0,2\pi] \times [0,2]. \end{cases}$ *en el punto* $(0,1,1)$.

Bibliografía

[1] SymPy Documentation, Release 1.11.1, SymPy Development Team.

[2] The Python Library Reference, Versión 3.10.5, Guido van Rossum and the Python development team.

[3] Python Tutorial, Versión 3.10.5, Guido van Rossum and the Python development team.

[4] https://matplotlib.org/