Capítulo 6

Apéndice

6.1 Sistemas de Álgebra Computacional

Los Sistemas de Computación Algebraica (CAS, por sus siglas en inglés) son herramientas poderosas que utilizan algoritmos y técnicas computacionales para realizar cálculos simbólicos en matemáticas y ciencias afines. Esto significa que el computador puede efectuar operaciones con ecuaciones y fórmulas simbólicamente, es decir, $a+b = c$ se interpreta como la suma de variables y no como la suma de números previamente asignados. Por lo tanto, estos sistemas son capaces de manipular expresiones algebraicas, resolver ecuaciones, derivar e integrar funciones, realizar operaciones matriciales, entre otras tareas, todo de manera simbólica. De hecho, la computación algebraica es parte de una disciplina mas amplia, la computación simbólica.

La principal diferencia entre los sistemas numéricos y los sistemas algebraicos (o simbólicos) radica en la memoria necesaria para almacenar una variable. En los algoritmos numéricos, es posible predecir la memoria requerida para su ejecución y así decidir si se necesita un supercomputador o si es suficiente con un ordenador personal. Por ejemplo, cuando se realiza una multiplicación numérica $a \times b = c$, las variables a , b y el resultado c son números que se almacenan en 64 bits[1].

En contraste, en la computación simbólica, la situación es diferente. Una variable a puede ser un polinomio de grado 10 y la variable b un polinomio de grado 6; por lo tanto, el resultado c será un polinomio de grado 16. El tamaño de la memoria necesaria para almacenar esta multiplicación de variables simbólicas no se puede predecir fácilmente, ya que depende de la estructura y complejidad de las expresiones involucradas.

Otra diferencia entre la manipulación numérica y la simbólica radica en los algoritmos de simplificación. En computación simbólica, cada vez que encontramos una expresión como $\cos^2 x + \sin^2 x$, debemos reconocerla y simplificarla a 1, utilizando identidades trigonométricas. En el caso del cómputo numérico, no existe esa dificultad, ya que la suma de dos números es simplemente otro número y no requiere simplificación adicional.

La historia de los sistemas de cómputo algebraico se remonta casi al origen mismo de los sistemas computacionales[2]. El desarrollo de un lenguaje diferente a FORTRAN, que permitiera el manejo de listas

[1]Este es el valor por omisión para el caso de un lenguaje como FORTRAN.

[2]El lector puede consultar parte de esa historia en MacCallum, M. A. (2018). Computer algebra in gravity research. Living Reviews in Relativity, 21(1), 6.

y no solo de números, marcó una diferencia significativa. LISP (acrónimo de List Processing[3]) es un lenguaje vinculado a los albores de la inteligencia artificial, nacido en 1960 y que perdura hasta nuestros días. LISP hizo posible el desarrollo de REDUCE y MAXIMA (originalmente MACSYMA) a mediados de los años 60. Actualmente, disponemos de estos dos sistemas de cómputo algebraico en versiones de software libre.

En el ámbito de la enseñanza, los CAS facilitan el aprendizaje y la comprensión de conceptos matemáticos y científicos. Al proporcionar una plataforma interactiva y dinámica, los estudiantes pueden explorar conceptos abstractos de manera visual y práctica, lo que les ayuda a internalizar y aplicar esos conceptos de manera más efectiva. Además, los CAS permiten a los estudiantes experimentar con diferentes escenarios y casos, lo que fomenta la resolución de problemas cada vez más complejos.

En el ámbito profesional, los CAS son herramientas indispensables para realizar cálculos complejos y llevar a cabo análisis simbólicos detallados. Los investigadores pueden utilizar estos sistemas para explorar nuevas teorías, validar resultados teóricos, resolver ecuaciones diferenciales y realizar simulaciones, entre otras aplicaciones. La capacidad de automatizar tareas repetitivas y manipular expresiones simbólicas de manera eficiente ahorra tiempo y esfuerzo, lo que permite a los investigadores concentrarse en aspectos más creativos y analíticos de su trabajo.

Estas herramientas computacionales permiten operar de manera exacta con símbolos que representan objetos matemáticos tales como: Números (Enteros, racionales, reales, complejos . . .), Polinomios, Funciones Racionales, Sistemas de Ecuaciones. Grupos, Anillos, Álgebras . . .

Otra característica principal radica en el hecho de que son interactivos (interpretados o ejecutados al momento de proveer una instrucción).

Existe, o han existido, una enorme cantidad de sistemas algebraicos[4] donde destacan: Maple, Mathematica, Matlab que son pagos y los que son software libre como REDUCE, MAXIMA y SymPy.

6.1.1 Python, SymPy y el cómputo científico

En casi cualquier área del cómputo científico confluyen tres elementos fundamentales: desarrollos algebraicos, cálculo numérico y visualización de resultados. Estas tres características, integradas en un ecosistema, son proporcionadas por el lenguaje Python en conjunto con los cuadernos Jupyter. Python es un lenguaje de programación orientado a objetos, de alto nivel, que cuenta con un amplio ecosistema de bibliotecas de terceros que simplifican tareas complejas y mejoran la productividad. Además, posee una amplia y activa comunidad de desarrolladores que contribuyen a su crecimiento y proporcionan soporte a través de foros, tutoriales y documentación. Este enfoque basado en la comunidad garantiza la mejora continua y el acceso a una gran cantidad de recursos.

SymPy es una biblioteca de computación simbólica en Python que ofrece una amplia gama de herramientas para realizar cálculos matemáticos y simbólicos. Fue desarrollada por Ondřej Čertík cuando era estudiante de física en la Universidad de Nevada, Reno, como una alternativa ligera a otros sistemas de álgebra computacional, como Mathematica y Maple. La idea era crear una biblioteca que estuviera completamente escrita en Python y que fuera fácil de usar. Desde entonces, SymPy ha crecido vertiginosamente, integrándose en otros proyectos y conformándose con otras herramientas en el ecosistema científico de Python, como SciPy, NumPy y Jupyter. La comunidad sigue activa, contribuyendo con nue-

[3] https://es.wikipedia.org/wiki/Lisp.

[4] https://en.wikipedia.org/wiki/List_of_computer_algebra_systems.

vas funcionalidades, mejorando la documentación y asegurando la compatibilidad con nuevas versiones de Python.

Esta biblioteca permite realizar una variedad de operaciones simbólicas, como manipulación de expresiones algebraicas, resolución de ecuaciones, cálculo diferencial e integral, álgebra lineal, optimización, entre otras. Además, SymPy está diseñada para ser fácilmente extensible, lo que permite a los usuarios crear y agregar sus propias funciones y módulos personalizados.

Como hemos dicho, una de las características más destacadas de SymPy es su integración con el ambiente de trabajo Jupyter-Python. Esto significa que los usuarios pueden combinar la potencia de SymPy con las capacidades de implementación de algoritmos en Python, lo que facilita la automatización de tareas, la creación de scripts personalizados y el desarrollo de aplicaciones más complejas.

SymPy es ampliamente utilizado en entornos académicos, de investigación y de desarrollo, tanto para el aprendizaje y la enseñanza de conceptos matemáticos como para la resolución de problemas del mundo real. Su naturaleza de código abierto fomenta la colaboración y el intercambio de ideas entre la comunidad de usuarios, lo que contribuye a su constante mejora y evolución.

Por ser una biblioteca de Python, puede ser utilizada en una variedad de entornos de desarrollo y plataformas que admitan Python. Algunas de las opciones comunes para usar SymPy incluyen:

1. Entornos de desarrollo integrado (IDE):
    - *Visual Studio Code*: *Visual Studio Code* es un popular IDE de código abierto que admite Python y proporciona funcionalidades avanzadas para el desarrollo de software, incluida la integración con SymPy.
    - *PyCharm*: PyCharm es otro IDE ampliamente utilizado para el desarrollo de Python que ofrece características avanzadas como depuración, análisis estático y soporte para SymPy.
2. Cuadernos Jupyter
    - Cuadernos Jupyter (*Jupyter Notebooks*): Los cuadernos Jupyter son una herramienta poderosa para la computación interactiva que permite crear y compartir documentos que contienen código ejecutable, visualizaciones y texto explicativo. SymPy puede ser utilizado en cuadernos Jupyter para realizar cálculos simbólicos interactivos.
    - *Google Colab*: *Google Colab* es una plataforma de Google que permite ejecutar cuadernos Jupyter en la nube de forma gratuita. Proporciona acceso gratuito a recursos de cómputo, lo que lo convierte en una opción popular para trabajar con SymPy y otros paquetes de Python.
3. Entornos de desarrollo Python:
    - Python en línea de comandos: SymPy se puede utilizar directamente en el intérprete de Python en la línea de comandos para realizar cálculos rápidos y experimentar con la biblioteca.
    - *Scripts* de Python: SymPy también se puede utilizar en *scripts* de Python para automatizar tareas y realizar cálculos en lotes. Plataformas en la nube y servicios de alojamiento:
    - Plataformas en la nube como AWS, Azure y otros servicios de alojamiento de Python ofrecen la posibilidad de ejecutar código Python, incluido SymPy, en entornos en la nube escalables y seguros.

6.2 Guía rápida sobre SymPy

Este apéndice tiene como objetivo brindar una introducción a SymPy para usuarios no especializados en el uso de herramientas computacionales.

La documentación oficial se encuentra en: https://docs.sympy.org/latest/index.html. Adicionalmente, todos los códigos los pueden descargar del siguiente enlace https://github.com/nunezluis/CodigosLibroMatematicas/tree/main/Capitulo06.

Primero que todo debemos incorporar, de la enorme cantidad de bibliotecas que existen para Python, la librería ***sympy***

[1]:
```
# 6/4/2024 <= Esta linea es un comentario gracias al #
import sympy
from sympy import *
```

[2]:
```
__version__  # Esto es opcional y permite ver la versión de Sympy que estamos␣
↪usando
```

[2]: '1.12'

6.2.1 Sintáxis básica

Si queremos calcular: $3! + 2^3 - 1$ debemos escribir:

[3]:
```
factorial(3) + 2**3 - 1
```

[3]: 13

El valor de: $\sqrt{8}$

[4]:
```
sqrt(8)
```

[4]: $2\sqrt{2}$

Si queremos el valor numérico podemos usar **float**

[5]:
```
float(sqrt(8))
```

[5]: 2.8284271247461903

Otras variantes

[6]:
```
N(sqrt(8),10)
```

[6]: 2.828427125

[7]:
```
sqrt(8).evalf(10)
```

[7]: 2.828427125

[8]:
```
N(sqrt(8),3)
```

[8]: 2.83

[9]:
```
pi # La constante σ?
```

[9]: π

[10]:
```
pi.evalf(50)
```

[10]: $3.1415926535897932384626433832795028841971693993751$

[11]:
```
ln(E)
```

[11]: 1

[12]:
```
log(E)
```

[12]: 1

[13]:
```
ln(E).evalf(2)
```

[13]: 1.0

[14]:
```
log(10)
```

[14]: $\log(10)$

[15]:
```
log(10).evalf(5)
```

[15]: 2.3026

[16]:
```
log(10,10)
```

[16]: 1

Consideremos ahora una combinación de operaciones matemáticas

$$\frac{\sqrt{8}}{3} + \ln(e+1) + \log_{10}(3) + e^{\pi^2} + (e^{\pi})^2 - 6!\sin\left(\frac{\pi}{3}\right) + \sqrt{-1}$$

[17]:
```
sqrt(8)/3+ln(E+1)+log(3,10)+E**(pi**2)+exp(pi)**2-factorial(6)*sin(pi/
↪3)+sqrt(-1)
```

[17]: $-360\sqrt{3} + \frac{\log(3)}{\log(10)} + \frac{2\sqrt{2}}{3} + \log(1+e) + e^{2\pi} + e^{\pi^2} + i$

Aquí aprenderemos un atajo. Queremos reutilizar la última salida en el siguiente comando.

Para hacer esto se utiliza _ como el argumento del comando. En este caso el comando que utilizaremos es **round**, que nos dará el valor numérico de la expresión anterior con los decimales que especifiquemos.

round() devuelve un número en coma flotante pero redondeado y con el número de decimales especificado.

[18]:
```
round(_,8)
```

[18]: $19248.37563115 + i$

Escribamos nuevamente la expresión, pero sin la parte imaginaria, y la asignaremos a la variable x

[19]:
```
x=sqrt(8)/3+ln(E+1)+log(3,10)+E**(pi**2)+exp(pi)**2
```

[20]:
```
x
```

[20]: $\frac{\log(3)}{\log(10)} + \frac{2\sqrt{2}}{3} + \log(1+e) + e^{2\pi} + e^{\pi^2}$

[21]:
```
float(_)
```

[21]: 19871.91392187373

[22]:
```
round(x, 2)
```

[22]: 19871.91

[23]:
```
N(_,2)
```

[23]: $2.0 \cdot 10^4$

Como pudimos ver, el programa se entiende perfectamente con los números imaginarios, por ejemplo: $\sqrt{-1} + 2i$

[24]:
```
sqrt(-1)+2*I
```

[24]: $3i$

Una de las ecuaciones más bonitas de las matemáticas:

$e^{2\pi i} = 1$

[25]:
```
exp(2*pi*I)-1
```

[25]: 0

A diferencia de muchos sistemas de manipulación simbólica, en SymPy las variables deben definirse antes de usarse.

[26]:
```
t, x, y, z, n, a, b, c = symbols('t x y z n a b c')
```

[27]:
```
r = x + 2*y
```

[28]:
```
r
```

[28]: $x + 2y$

[29]:
```
p=x**2*r**3
p
```

[29]: $x^2 (x + 2y)^3$

Algunas funciones de manipulación simbolica

[30]:
```
expand(p)
```

[30]: $x^5 + 6x^4y + 12x^3y^2 + 8x^2y^3$

Lo inverso de expandir es factorizar

[31]:
```
factor(_)
```

[31]: $x^2 (x + 2y)^3$

[32]:
```
expand(_)
```

[32]: $x^5 + 6x^4y + 12x^3y^2 + 8x^2y^3$

[33]:
```
collect(_,x*y)
```

[33]: $x^5 + 6x^4y + x^2y^2 \cdot (12x + 8y)$

[34]:
```
simplify(_)
```

[34]: $x^2 \left(x^3 + 6x^2y + y^2 \cdot (12x + 8y)\right)$

Escribamos el siguiente polinomio

[35]:
```
P = (x-1)*(x-2)*(x-3)
P
```

[35]: $(x-3)(x-2)(x-1)$

[36]:
```
P.expand()
```

[36]: $x^3 - 6x^2 + 11x - 6$

[37]:
```
raices = solve(P, x)
raices
```

[37]: `[1, 2, 3]`

La solución es escrita en el formato de una lista: $[a, b, c, d]$, donde el primer elemento está etiquetado con cero, el segundo con uno, el tercero con dos. . .

Por lo tanto, para extraer los elementos de una lista podemos hacer lo siguiente:

[38]:
```
raices[0]
```

[38]: 1

[39]:
```
raices[0]+raices[1]+raices[2]
```

[39]: 6

Por lo tanto: $P - (x - x_0)(x - x_1)(x - x_2) = 0$

[40]:
```
simplify(P - (x-raices[0])*(x-raices[1])*(x-raices[2]) )
```

[40]: 0

Podemos simplificar expresiones trigonométricas

[41]:
```
simplify( sin(x)*cos(y)+cos(x)*sin(y) )
```

[41]: $\sin(x+y)$

[42]:
```
expr = 2*sin(x)**2+2*cos(x)**2
expr
```

[42]: $2\sin^2(x) + 2\cos^2(x)$

[43]:
```
trigsimp(expr)
```

[43]: 2

[44]:
```
(sin(x)**4-2*cos(x)**2*sin(x)**2+cos(x)**4).simplify()
```

[44]: $\frac{\cos(4x)}{2} + \frac{1}{2}$

[45]:
```
expand_trig(sin(2*x))
```

[45]: $2\sin(x)\cos(x)$

6.2.2 Cálculos elementales

Vamos a ver una forma de trabajar con funciones como una asignación a una variable. Luego mostraremos otras posibilidades

[46]:
```
f=sin(x)/exp(x)
```

[47]:
```
f
```

[47]: $e^{-x}\sin\left(x\right)$

La primera derivada

[48]:
```
df=diff(f,x)
```

[49]:
```
df
```

[49]: $-e^{-x}\sin\left(x\right)+e^{-x}\cos\left(x\right)$

Otra manera de hacer lo mismo es

[50]:
```
df=f.diff(x)
df
```

[50]: $-e^{-x}\sin\left(x\right)+e^{-x}\cos\left(x\right)$

Simplificamos

[51]:
```
df.simplify()
```

[51]: $\sqrt{2}e^{-x}\cos\left(x+\frac{\pi}{4}\right)$

La integral

[52]:
```
integrate(df,x)
```

[52]: $e^{-x}\sin\left(x\right)$

Derivadas de orden superior

[53]:
```
diff(f,x,5)
```

[53]: $4\left(\sin\left(x\right)-\cos\left(x\right)\right)e^{-x}$

Para evaluar la funcíon en un punto

[54]:
```
f.subs(x, pi/2)
```

[54]: $e^{-\frac{\pi}{2}}$

Para funciones de varias variables

[55]:
```
f=exp(x**2+y**2)/(x-y)
f
```

[55]: $\frac{e^{x^2+y^2}}{x-y}$

La derivadas cruzadas:

[56]:
```
diff(diff(f,y),x).factor()
```

[56]: $\frac{2\cdot\left(2x^3y-4x^2y^2+x^2+2xy^3-2xy+y^2-1\right)e^{x^2}e^{y^2}}{\left(x-y\right)^3}$

[57]:
```
(diff(f,x) + diff(f,y)).factor()
```

[57]:
$$\frac{2\left(x+y\right)e^{x^2}e^{y^2}}{x-y}$$

Las derivadas también pueden ejecutarse de la siguiente manera

[58]:
```
f.diff(x) + f.diff(y)
```

[58]:
$$\frac{2xe^{x^2+y^2}}{x-y}+\frac{2ye^{x^2+y^2}}{x-y}$$

[59]:
```
factor(_)
```

[59]:
$$\frac{2\left(x+y\right)e^{x^2}e^{y^2}}{x-y}$$

Si queremos dejar indicadas las operaciones matemáticas:

[60]:
```
Derivative(f,x)+Derivative(f,y)
```

[60]:
$$\frac{\partial}{\partial x}\frac{e^{x^2+y^2}}{x-y}+\frac{\partial}{\partial y}\frac{e^{x^2+y^2}}{x-y}$$

Y luego le pedimos al programa que ejecute la operación indicada con anterioridad

[61]:
```
(_).doit()
```

[61]:
$$\frac{2xe^{x^2+y^2}}{x-y}+\frac{2ye^{x^2+y^2}}{x-y}$$

[62]:
```
Integral(f,x)
```

[62]:
$$\int\frac{e^{x^2+y^2}}{x-y}\,dx$$

Para asignarle valores a las variables

[63]:
```
f.subs([(x, 2), (y, 1)])
```

[63]:
$$e^5$$

[64]:
```
f.evalf(subs={x:2,y:1})
```

[64]:
148.413159102577

Consideremos otra función y varios cálculos con ella.

[65]:
```
σ=2*x/sqrt(x**2+1)
σ
```

[65]:
$$\frac{2x}{\sqrt{x^2+1}}$$

Calculamos la primera derivada y la asignaremos a la variable $d\sigma$

[66]:
```
dσ= σ.diff(x).factor()
dσ
```

[66]:
$$\frac{2}{\left(x^2+1\right)^{\frac{3}{2}}}$$

La derivada cuarta y al mismo tiempo que se haga la respectiva factorización

[67]:
```
σ.diff(x,4).factor()
```

[67]: $$-\frac{30x\left(4x^2-3\right)}{\left(x^2+1\right)^{\frac{9}{2}}}$$

[68]:
```
integrate(σ ,x)
```

[68]: $$2\sqrt{x^2+1}$$

La integral definida

$\int_a^b \sigma dx$

[69]:
```
integrate(σ ,[x,a,b]).factor()
```

[69]: $$-2\left(\sqrt{a^2+1}-\sqrt{b^2+1}\right)$$

lím_{$x \to 1/2$}σ

[70]:
```
limit(σ ,x, 1/2 )
```

[70]: 0.894427190999916

SymPy no nos devolvió un resultado simbólico sino un número punto flotante. Esto es porque SymPy define tres tipos de números: Real, Racional y Entero. Para SymPy el $\frac{1}{2}$ significa $0,5$ y para evitar que haga esta interpretación podemos escribir

[71]:
```
Rational(1,2)
```

[71]: $$\frac{1}{2}$$

De manera equivalente, también se puede escribir:

[72]:
```
S(1)/2
```

[72]: $$\frac{1}{2}$$

Por lo tanto:

[73]:
```
limit(σ ,x, S(1)/2 )
```

[73]: $$\frac{2\sqrt{5}}{5}$$

Los límites por la izquierda y por la derecha

[74]:
```
limit(σ ,x, 1 , dir='+')
```

[74]: $$\sqrt{2}$$

[75]:
```
limit(σ ,x, 1 , dir='-')
```

[75]: $$\sqrt{2}$$

[76]:
```
Limit(σ ,x, oo)
```

[76]: $$\lim_{x\to\infty}\left(\frac{2x}{\sqrt{x^2+1}}\right)$$

[77]:
```
(_).doit()
```

[77]: 2

Para las series de Taylor al rededor de $x=0$

[78]:
```
series(σ ,x,0,8)
```

[78]:
$$2x - x^3 + \frac{3x^5}{4} - \frac{5x^7}{8} + O\left(x^8\right)$$

Al rededor de $x = 4$

[79]:
```
series(σ ,x,4,3)
```

[79]:
$$\frac{8\sqrt{17}}{17} + \frac{2\sqrt{17}\,(x-4)}{289} - \frac{12\sqrt{17}\,(x-4)^2}{4913} + O\left((x-4)^3 ; x \to 4\right)$$

Series de Taylor de funciones elementales

[80]:
```
series(sin(x), x, 0, 8)
```

[80]:
$$x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} + O\left(x^8\right)$$

[81]:
```
series(cos(x), x, 0, 8)
```

[81]:
$$1 - \frac{x^2}{2} + \frac{x^4}{24} - \frac{x^6}{720} + O\left(x^8\right)$$

[82]:
```
series(E**(x), x, 0, 8)
```

[82]:
$$1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + \frac{x^6}{720} + \frac{x^7}{5040} + O\left(x^8\right)$$

[83]:
```
series(ln(x+1), x, 0, 6)
```

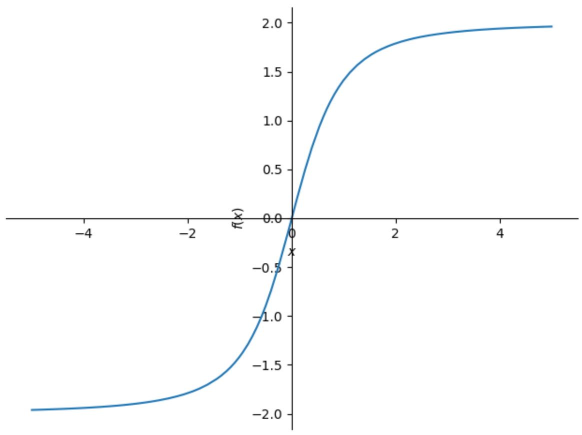
[83]:
$$x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} + O\left(x^6\right)$$

[84]:
```
summation(σ?,(x,0,6))
```

[84]:
$$\sqrt{2} + \frac{4\sqrt{5}}{5} + \frac{3\sqrt{10}}{5} + \frac{8\sqrt{17}}{17} + \frac{5\sqrt{26}}{13} + \frac{12\sqrt{37}}{37}$$

La gráfica más sencilla que podemos hacer es de la manera siguiente

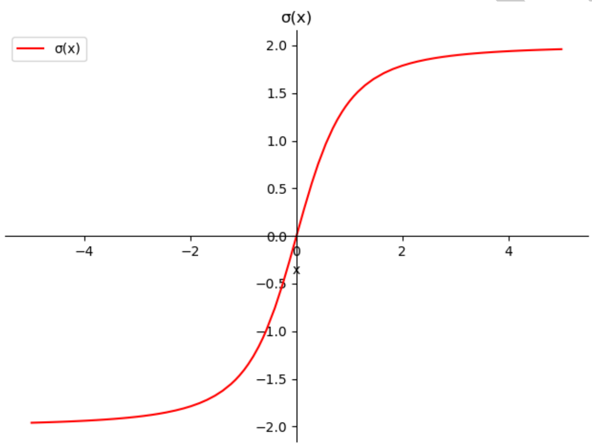
[85]:
```
plot(σ,(x,-5,5))
```



[85]: <sympy.plotting.plot.Plot at 0x118aad110>

O si lo preferimos

[86]:
```
# Creamos la gráfica
p = plot(σ,(x,-5,5), show=False)
# Cambiamos los colores y etiquetas de cada función
p[0].line_color = 'red'
p[0].label = 'σ(x)'
# Agregamos un título a la gráfica y a los ejes
p.title = 'σ(x)'
p.xlabel = 'x'
p.ylabel = False
# Agregamos una leyenda
p.legend = True
p.legend_loc = 'upper left'
# Mostramos la gráfica
p.show()
```



También podemos definir funciones de manera abstracta para usarlas como objetos matemáticos

[87]:
```
f = Function('f')
g = Function('g')(x)
```

[88]:
```
f
```

[88]: f

[89]:
```
g
```

[89]: $g(x)$

[90]:
```
(f(x)+g).diff()
```

[90]: $\frac{d}{dx}f(x) + \frac{d}{dx}g(x)$

[91]:
```
f = Function('f')(x)
f
```

[91]: $f(x)$

[92]:
```
integrate(g,x)
```

[92]: $\int g(x)\,dx$

La regla de la cadena

[93]:
```
diff(f*g,x)
```

[93]: $f(x)\frac{d}{dx}g(x) + g(x)\frac{d}{dx}f(x)$

[94]:
```
diff(g/f,x).factor()
```

[94]: $-\frac{-f(x)\frac{d}{dx}g(x) + g(x)\frac{d}{dx}f(x)}{f^2(x)}$

6.2.3 Ecuaciones algebraicas

Para escribir ecuaciones usamos la siguiente sintaxis

[95]:
```
Eq(Function("f")(x),Sum(x,(x,1,10)))
```

[95]: $f(x) = \sum_{x=1}^{10} x$

[96]:
```
Eq(x+5, 3)
```

[96]: $x + 5 = 3$

[97]:
```
solveset(Eq(x+5, 3), x)
```

[97]: $\{-2\}$

Otro ejemplo la ecuación: $cos(x) = 1$, la resolveremos para x

[98]:
```
solveset(Eq(cos(x),1 ), x, domain=S.Reals)
```

[98]: $\{2n\pi \mid n \in \mathbb{Z}\}$

Aquí “solveset” devuelve un objeto “set”. Para los casos en los que no “conoce” todas las soluciones devuelve un “ConditionSet” con una solución parcial. Para la entrada sólo toma la ecuación, las variables a resolver y el argumento opcional “dominio” sobre el que se va a resolver la ecuación.

Consideremos el siguiente ejemplo

[99]:
```
Ec1=Eq(x**2+2*x, 1)
Ec1
```

[99]: $x^2 + 2x = 1$

La función “solve” se utiliza principalmente para resolver ecuaciones algebraicas y sistemas de ecuaciones algebraicas. Puede manejar una amplia gama de ecuaciones algebraicas y puede devolver soluciones en forma de símbolos, números reales o complejos. La salida es una lista.

[100]:
```
solve(Ec1,x)
```

[100]: [-1 + sqrt(2), -sqrt(2) - 1]

[101]:
```
Ec=Eq(a*x**2+b*x+c,0)
Ec
```

[101]: $ax^2 + bx + c = 0$

[102]:
```
solveset(Ec, x)
```

[102]: $$\left\{-\frac{b}{2a} - \frac{\sqrt{-4ac + b^2}}{2a}, -\frac{b}{2a} + \frac{\sqrt{-4ac + b^2}}{2a}\right\}$$

[103]:
```
solveset(Ec1, x)
```

[103]: $\left\{-1 + \sqrt{2}, -\sqrt{2} - 1\right\}$

[104]:
```
Ec2=Eq(2*x**2+2*x, -1)
Ec2
```

[104]: $2x^2 + 2x = -1$

[105]:
```
solve(Ec2, x)
```

[105]: [-1/2 - I/2, -1/2 + I/2]

[106]:
```
Ec3= Eq(x**6+x**4-x**3+x-2,0)
Ec3
```

[106]: $x^6 + x^4 - x^3 + x - 2 = 0$

[107]:
```
factor(Ec3)
```

[107]: $(x-1)(x+1)\left(x^2 - x + 1\right)\left(x^2 + x + 2\right) = 0$

[108]:
```
solveset(Ec3, x)
```

[108]: $$\left\{-1, 1, -\frac{1}{2} - \frac{\sqrt{7}i}{2}, -\frac{1}{2} + \frac{\sqrt{7}i}{2}, \frac{1}{2} - \frac{\sqrt{3}i}{2}, \frac{1}{2} + \frac{\sqrt{3}i}{2}\right\}$$

[109]:
```
Ec4= Eq(x**7+x**4-x**3+x,2)
Ec4
```

[109]: $x^7 + x^4 - x^3 + x = 2$

[110]:
```
solve(Ec4, x)
```

[110]:
```
[1,
 CRootOf(x**6 + x**5 + x**4 + 2*x**3 + x**2 + x + 2, 0),
 CRootOf(x**6 + x**5 + x**4 + 2*x**3 + x**2 + x + 2, 1),
 CRootOf(x**6 + x**5 + x**4 + 2*x**3 + x**2 + x + 2, 2),
 CRootOf(x**6 + x**5 + x**4 + 2*x**3 + x**2 + x + 2, 3),
 CRootOf(x**6 + x**5 + x**4 + 2*x**3 + x**2 + x + 2, 4),
```

```
 CRootOf(x**6 + x**5 + x**4 + 2*x**3 + x**2 + x + 2, 5)]
```

[111]:
```
sols = solveset(Ec4, x)
```

[112]:
```
sols.evalf(3)
```

[112]: $\{1.0, -1.12-0.339i, -1.12+0.339i, -0.0158-1.16i, -0.0158+1.16i, 0.633-0.83i, 0.633+0.83i\}$

Para resolver sistemas de ecuaciones, como por ejemplo:

$$x+y+z-1=0\,,\; x+y+2z-3=0\,,\; x-y+z-1=0\,.$$

[113]:
```
linsolve([x + y + z - 1, x + y + 2*z - 3, x-y+z-1 ], (x, y, z))
```

[113]: $\{(-1,\ 0,\ 2)\}$

[114]:
```
Ec1=2*x-2*y+z+3
Ec2=x+3*y-2*z-1
Ec3=3*x-y-z-2
linsolve([Ec1, Ec2, Ec3 ], (x, y, z))
```

[114]: $\left\{\left(-\frac{7}{5},\ -2,\ -\frac{21}{5}\right)\right\}$

6.2.4 Ecuaciones diferenciales

Definamos la función $f=f(x)$

[115]:
```
f = Function('f')(x)
```

Resolvamos la ecuación diferencial

$$f(x)-\frac{df(x)}{dx}=0$$

[116]:
```
dsolve( f - diff(f, x), f )
```

[116]: $f(x)=C_1e^x$

También podemos escribir primero la ecuación diferencial

[117]:
```
ed = Eq(f - diff(f, x),0)
ed
```

[117]: $f(x)-\frac{d}{dx}f(x)=0$

[118]:
```
dsolve(ed, f)
```

[118]: $f(x)=C_1e^x$

Veamos la ecuación diferencial para el oscilador armónico simple

$$\frac{d^2y(t)}{dt^2}+\omega_0^2y(t)=0$$

[119]:
```
ω0 = symbols('ω0')
y = Function('y')(t)
ed1 = Eq(y.diff(t,2)  + ω0**2*y,0)
ed1
```

[119]:
$$\omega_0^2 y(t) + \frac{d^2}{dt^2} y(t) = 0$$

[120]:
```
dsolve(ed1, y)
```

[120]:
$$y(t) = C_1 e^{-it\omega_0} + C_2 e^{it\omega_0}$$

Una de las ventajas de los sistemas de manipulación simbólica es que son de gran utilidad cuando necesitamos hacer cálculos largos y tediosos.

Por ejemplo, queremos demostrar que la función

$$F(x,y,z) = \frac{\sin\left(\frac{nz\sqrt{z^2+y^2+x^2}}{\sqrt{z^2+y^2}}\right)}{\sqrt{z^2+y^2+x^2}}$$

satisface la ecuación diferencial:

$$\frac{\partial^4 F}{\partial x^4} + \frac{\partial^4 F}{\partial y^2 x^2} + \frac{\partial^4 F}{\partial z^2 x^2} + n^2\left[\frac{\partial^2 F}{\partial x^2} + \frac{\partial^2 F}{\partial y^2}\right] = 0$$

Primero debemos escribir la función F

[121]:
```
F=sin(n*z*sqrt(x**2+y**2+z**2)/sqrt(y**2+z**2))/sqrt(x**2+y**2+z**2)
F
```

[121]:
$$\frac{\sin\left(\frac{nz\sqrt{x^2+z^2+y^2(t)}}{\sqrt{z^2+y^2(t)}}\right)}{\sqrt{x^2+z^2+y^2(t)}}$$

Luego podríamos usar **Derivative** para escribir la ecuación y verificar que la escribimos bien

[122]:
```
Derivative(F,x,4)+Derivative(Derivative(F,x,2),y,2)+ \
Derivative(Derivative(F,x,2),z,2) + \
n**2*(Derivative(F,x,2)+Derivative(F,y,2))
```

[122]:
$$n^2\left(\frac{\partial^2}{\partial x^2}\frac{\sin\left(\frac{nz\sqrt{x^2+z^2+y^2(t)}}{\sqrt{z^2+y^2(t)}}\right)}{\sqrt{x^2+z^2+y^2(t)}} + \frac{\partial^2}{\partial y(t)^2}\frac{\sin\left(\frac{nz\sqrt{x^2+z^2+y^2(t)}}{\sqrt{z^2+y^2(t)}}\right)}{\sqrt{x^2+z^2+y^2(t)}}\right) + \frac{\partial^4}{\partial x^4}\frac{\sin\left(\frac{nz\sqrt{x^2+z^2+y^2(t)}}{\sqrt{z^2+y^2(t)}}\right)}{\sqrt{x^2+z^2+y^2(t)}} + \frac{\partial^4}{\partial z^2\partial x^2}\frac{\sin\left(\frac{nz\sqrt{x^2+z^2+y^2(t)}}{\sqrt{z^2+y^2(t)}}\right)}{\sqrt{x^2+z^2+y^2(t)}} + \frac{\partial^4}{\partial y(t)^2\partial x^2}\frac{\sin\left(\frac{nz\sqrt{x^2+z^2+y^2(t)}}{\sqrt{z^2+y^2(t)}}\right)}{\sqrt{x^2+z^2+y^2(t)}}$$

Ahora le pedimos al programa que realice todas las derivadas indicadas y que luego aplique la factorización respectiva.

[123]:
```
factor((_).doit())
```

[123]:
$$0$$

En este caso el programa demora bastante tiempo en realizar los cálculos porque ha representado todas las derivadas y luego hace la factorización.

También se puede hacer la demostración de manera más directa y el tiempo de ejecución será mucho menor

[124]:
```
(diff(F,x,4)+diff(diff(F,x,2),y,2)+diff(diff(F,x,2),z,2)+
 n**2*(diff(F,x,2)+diff(F,y,2))).factor()
```

[124]: 0

6.2.5 Álgebra vectorial y matricial

A los vectores podemos tratarlos como objetos matriciales

[125]:
```
A = Matrix([[4, 5, 6]]) # un vector fila 1x3
A
```

[125]: $\begin{bmatrix}4 & 5 & 6\end{bmatrix}$

[126]:
```
B = Matrix ([[7] ,[8] , [9]]) # un vector columna 3x1
B
```

[126]: $\begin{bmatrix}7\\8\\9\end{bmatrix}$

[127]:
```
B.T # vector traspuesta de B
```

[127]: $\begin{bmatrix}7 & 8 & 9\end{bmatrix}$

[128]:
```
A[0] # Primera componente del vector A (índice 0)
```

[128]: 4

[129]:
```
A.norm() # norma del vector A
```

[129]: $\sqrt{77}$

[130]:
```
Ahat = A/A.norm() # vector unitario asociada a A
Ahat
```

[130]: $\begin{bmatrix}\frac{4\sqrt{77}}{77} & \frac{5\sqrt{77}}{77} & \frac{6\sqrt{77}}{77}\end{bmatrix}$

[131]:
```
Ahat.norm()
```

[131]: 1

Definamos los siguientes vectores

$$\mathbf{a} = 2\mathbf{i} + 4\mathbf{j} + 6\mathbf{k}, \quad \mathbf{b} = 5\mathbf{i} + 7\mathbf{j} + 9\mathbf{k} \;\; \text{y} \;\; \mathbf{c} = \mathbf{i} + 3\mathbf{j}$$

[132]:
```
a = Matrix([2,4,6])
b = Matrix([5,7,9])
c = Matrix([1,3,0])
```

[133]:
```
a+b+c
```

[133]:
$$\begin{bmatrix}8\\14\\15\end{bmatrix}$$

[134]:
```
3*a+5*b-c
```

[134]:
$$\begin{bmatrix}30\\44\\63\end{bmatrix}$$

Recordemos que el primer elemento será la primer componente del vector
El producto escalar:

$$\mathbf{a}\cdot\mathbf{b} \equiv a_x b_x + a_y b_y + a_z b_z \equiv \|\mathbf{a}\|\|\mathbf{b}\|\cos(\varphi) \in \mathbb{R}$$

[135]:
```
a.dot(b)
```

[135]: 92

[136]:
```
b.dot(a)
```

[136]: 92

El ángulo entre los vectores

$$\varphi = \arccos\left[\frac{\mathbf{a}\cdot\mathbf{b}}{\|\mathbf{a}\|\|\mathbf{b}\|}\right]$$

[137]:
```
acos(a.dot(b)/(a.norm()*b.norm())).round(3)  # En radianes
```

[137]: 0.158

En grados sería:

[138]:
```
((_)*180/pi).evalf(4)  # grados
```

[138]: 9.052

El producto vectorial de dos vectores en 3 dimensiones es:

$$\mathbf{a}\times\mathbf{b} = a_y b_z - a_z b_y,\ a_z b_x - a_x b_z,\ a_x b_y - a_y b_x$$

[139]:
```
a.cross(b)
```

[139]:
$$\begin{bmatrix}-6\\12\\-6\end{bmatrix}$$

[140]:
```
b.cross(a)
```

[140]:
$$\begin{bmatrix}6\\-12\\6\end{bmatrix}$$

El producto triple:

$$\mathbf{c}\cdot(\mathbf{a}\times\mathbf{b})$$

[141]:
```
(a.cross(b)).dot(c)
```

[141]: 30

Tenemos otra opción para el cálculo vectorial, esta vez utilizando una librería llamada **sympy.vector**, como se muestra a continuación

[142]:
```
from sympy.vector import *
R = CoordSys3D('R')
```

[143]:
```
A = 2*R.i + 4*R.j - 6*R.k
B = R.i - 3*R.j + 5*R.k
C= R.i + R.j + R.k
```

R.i, R.j y R.k representan los vectores unitarios i, j, k

[144]:
```
# El vector A
A
```

[144]: $(2)\,\hat{\mathbf{i}}_{\mathbf{R}} + (4)\,\hat{\mathbf{j}}_{\mathbf{R}} + (-6)\,\hat{\mathbf{k}}_{\mathbf{R}}$

Operaciones básicas con vectores

[145]:
```
A + 2*B - C
```

[145]: $(3)\,\hat{\mathbf{i}}_{\mathbf{R}} + (-3)\,\hat{\mathbf{j}}_{\mathbf{R}} + (3)\,\hat{\mathbf{k}}_{\mathbf{R}}$

El módulo del vector $|\vec{A}| = \sqrt{\vec{A} \cdot \vec{A}}$

[146]:
```
sqrt(A.dot(A))
```

[146]: $2\sqrt{14}$

Aunque tenemos funciones en Sympy para la magnitud

[147]:
```
A.magnitude()
```

[147]: $2\sqrt{14}$

El vector unitario asociado a **A**

[148]:
```
A.normalize()
```

[148]: $$\left(\frac{\sqrt{14}}{14}\right)\hat{\mathbf{i}}_{\mathbf{R}} + \left(\frac{\sqrt{14}}{7}\right)\hat{\mathbf{j}}_{\mathbf{R}} + \left(-\frac{3\sqrt{14}}{14}\right)\hat{\mathbf{k}}_{\mathbf{R}}$$

[149]:
```
(_).magnitude()
```

[149]: 1

El producto escalar $A.B$

[150]:
```
A.dot(B)
```

[150]: -40

El producto vectorial $A \times B$

[151]:
```
A.cross(B)
```

[151]: $(2)\,\hat{\mathbf{i}}_{\mathbf{R}} + (-16)\,\hat{\mathbf{j}}_{\mathbf{R}} + (-10)\,\hat{\mathbf{k}}_{\mathbf{R}}$

[152]:
```
B.cross(A)
```

[152]: $(-2)\,\hat{\mathbf{i}}_{\mathbf{R}} + (16)\,\hat{\mathbf{j}}_{\mathbf{R}} + (10)\,\hat{\mathbf{k}}_{\mathbf{R}}$

El producto triple $(A \times B) \cdot C$

[153]:
```
A.cross(B).dot(C)
```

[153]: -24

Veamos ahora algunos operaciones con matrices. Definamos la matriz A

[154]:
```
init_printing(use_unicode=True)
A = Matrix([ [ 2,-3,-8, 7], [-2,-2, 2,-7],[ 1, 0,-5, 6] ])
A
```

[154]:
$$\begin{bmatrix} 2 & -3 & -8 & 7 \\ -2 & -2 & 2 & -7 \\ 1 & 0 & -5 & 6 \end{bmatrix}$$

He utilizado la función “init_printing(use_unicode=True)” que configura la salida de las expresiones simbólicas para que se muestren de manera más legible cuando se imprimen en la consola o en entornos interactivos como cuadernos Jupyter. Cuando se agrega “use_unicode=True”, SymPy utiliza caracteres Unicode para representar símbolos matemáticos como letras griegas, operadores y símbolos especiales. Esto mejora significativamente la legibilidad de las expresiones matemáticas impresas.

[155]:
```
A[0,1] # fila 0, columna 1 de A
```

[155]: -3

[156]:
```
A[0:2, 0:3] # submatriz 2x3 de A
```

[156]:
$$\begin{bmatrix} 2 & -3 & -8 \\ -2 & -2 & 2 \end{bmatrix}$$

La matiz identidad

[157]:
```
eye(4) # 4x4
```

[157]:
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[158]:
```
zeros(2, 3) # 2x3
```

[158]:
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

[159]:
```
A.transpose()
```

[159]:

$$\begin{bmatrix} 2 & -2 & 1 \\ -3 & -2 & 0 \\ -8 & 2 & -5 \\ 7 & -7 & 6 \end{bmatrix}$$

[160]:
```
B = Matrix([ [-24, 18, 5],[20,-15, -4], [-5, 4, 1] ])
B
```

[160]:
$$\begin{bmatrix} -24 & 18 & 5 \\ 20 & -15 & -4 \\ -5 & 4 & 1 \end{bmatrix}$$

[161]:
```
B.det()
```

[161]: 1

[162]:
```
B.inv()
```

[162]:
$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 5 & 6 & 0 \end{bmatrix}$$

La función "rref()" calcula la forma escalonada reducida por filas de una matriz. La forma escalonada reducida por filas (en inglés, Reduced Row Echelon Form o RREF) es una forma canónica de representar una matriz en álgebra lineal, que simplifica su análisis y facilita la resolución de sistemas de ecuaciones lineales y otras operaciones. La forma RREF de una matriz tiene las siguientes propiedades:

- Cada fila no nula comienza con un "1" (llamado pivote), y los "1" de cada fila están a la derecha de los "1" de las filas superiores.
- En cada columna que contiene un "1" (pivote), todas las otras entradas son "0".
- Las filas con todos ceros están al final, si existen.

[163]:
```
B.rref()
```

[163]:
$$\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, (0, 1, 2)\right)$$

La primera primera salida es la forma escalonada reducida de la matriz y la segunda salida es una tupla de índices de las columnas pivotables. Esto significa que las columnas pivotables están en las columnas con índices 0, 1 y 2 respectivamente. Las columnas que no están en pivots son columnas libres (que no contienen pivotes).

Veamos otros ejemplos:

[164]:
```
M = Matrix([[1, 2, 3], [-2, 3, 1], [-5, 4, 1]])
N = Matrix([[4, 5,6], [0, 7,1], [-5, 4, 1]])
```

[165]:
```
α,β = symbols('α β')
α*M + β*N
```

[165]:

$$\begin{bmatrix} \alpha+4\beta & 2\alpha+5\beta & 3\alpha+6\beta \\ -2\alpha & 3\alpha+7\beta & \alpha+\beta \\ -5\alpha-5\beta & 4\alpha+4\beta & \alpha+\beta \end{bmatrix}$$

[166]:
```
M*N
```

[166]:

$$\begin{bmatrix} -11 & 31 & 11 \\ -13 & 15 & -8 \\ -25 & 7 & -25 \end{bmatrix}$$

[167]:
```
M**2
```

[167]:

$$\begin{bmatrix} -18 & 20 & 8 \\ -13 & 9 & -2 \\ -18 & 6 & -10 \end{bmatrix}$$

[168]:
```
M**(-1)
```

[168]:

$$\begin{bmatrix} -\frac{1}{14} & \frac{5}{7} & -\frac{1}{2} \\ -\frac{3}{14} & \frac{8}{7} & -\frac{1}{2} \\ \frac{1}{2} & -1 & \frac{1}{2} \end{bmatrix}$$

[169]:
```
M*N**(-1)
```

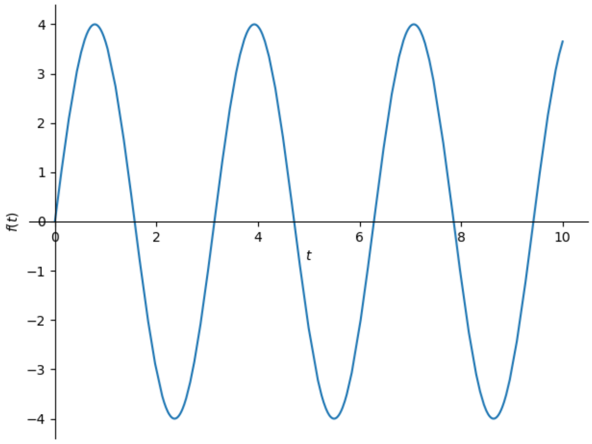
[169]:

$$\begin{bmatrix} \frac{98}{197} & -\frac{36}{197} & \frac{39}{197} \\ \frac{14}{197} & \frac{23}{197} & \frac{90}{197} \\ 0 & 0 & 1 \end{bmatrix}$$

6.2.6 Gráficos

El hecho de que SymPy sea una biblioteca de Python hace que se abran infinitas posibilidades para hacer gráficas con diferentes complejidades. Lo más sencillo es con el uso del comando **Plot**, pero mostraremos algunos ejemplos más elaborados.

[170]:
```
f=4*sin(2*t)
plot(f, (t, 0, 10))
```

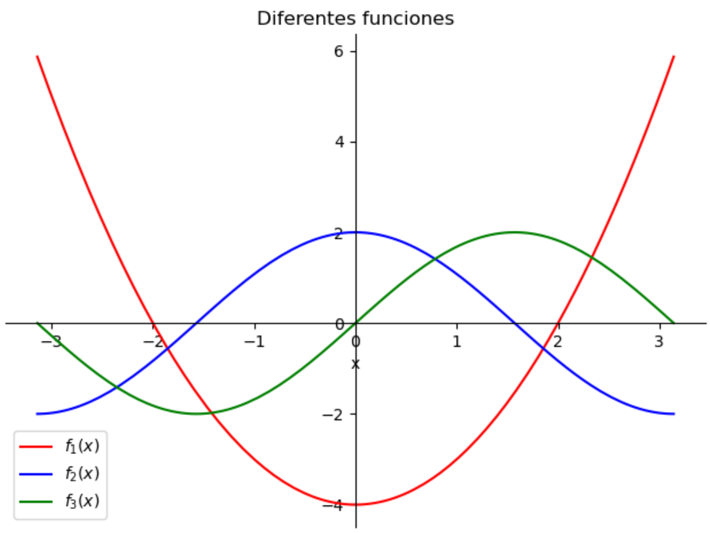


[170]: <sympy.plotting.plot.Plot at 0x11fbecd90>

La siguiente figura es con más elaboración, en este caso queremos graficar varias funciones

[171]:
```
f1=x**2-4
f2=2*cos(x)
f3=2*sin(x)
```

[172]:
```
# Tres funciones en una misma gráfica
p = plot(f1, f2, f3, (x, -pi, pi),show=False)
# Ponemos colores y etiquetamos cada función
p[0].line_color = 'red'
p[0].label = '$f_1(x)$'
p[1].line_color = 'blue'
p[1].label = '$f_2(x)$'
p[2].line_color = 'green'
p[2].label = '$f_3(x)$'
# El título de la gráfica y de los ejes
p.title = 'Diferentes funciones'
p.xlabel = 'x'
p.ylabel = False
# Agregamos una leyenda
p.legend = True
p.legend_loc = 'upper left'
# Mostramos la gráfica
p.show()
```



Existe una librería muy potente para graficar que se llama **matplotlib**. Más información en: https://matplotlib.org/

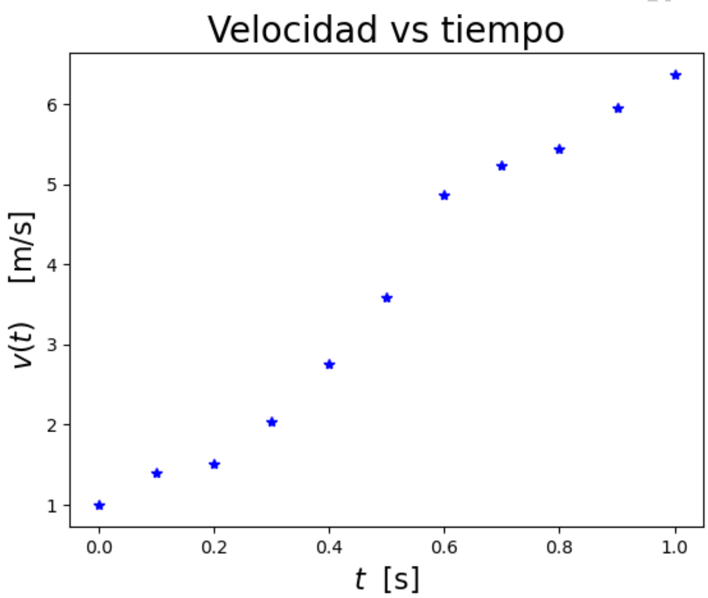
Primero que todo, se debe cargar la librería:

[173]:
```
import matplotlib.pyplot as plt
```

Si queremos graficar un conjunto de datos podemos hacer lo siguiente:

[174]:
```
# Los datos se escriben como una lista [dato1,dato2,...]
yn =[1.00, 1.40, 1.51, 2.03, 2.75, 3.59, 4.87, 5.23, 5.44, 5.95, 6.37]
xn =[0.00, 0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 1.00]
```

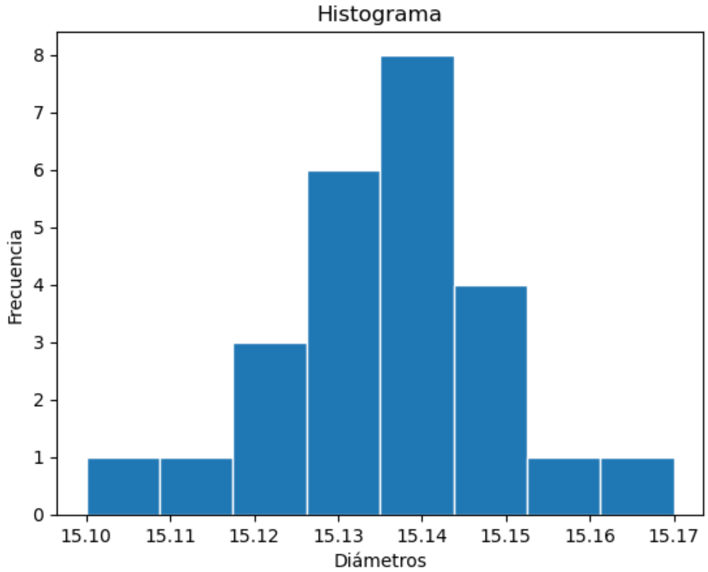
[175]:
```
plt.plot(xn, yn, '*',color="blue")
plt.title('Velocidad vs tiempo', fontsize=20)
plt.xlabel('$t$  [s]', fontsize=16)
plt.ylabel('$v(t)$    [m/s]', fontsize=16)
plt.show()
```



A continuación veamos un ejemplo para hacer un histograma.

Se realizaron varias medidas del diámetro de un vaso, los datos y el histograma se muestran a continuación:

[176]:
```
d = [15.12, 15.10, 15.15, 15.17, 15.14, 15.16, 15.14, 15.12,
     15.12, 15.14, 15.15, 15.14, 15.13, 15.14, 15.13, 15.13,
     15.14, 15.14, 15.14, 15.13, 15.15, 15.13, 15.11, 15.13, 15.15]
plt.hist(d,edgecolor = "white", bins=8)
plt.xlabel('Diámetros')
plt.ylabel('Frecuencia')
plt.title("Histograma")
plt.show()
```

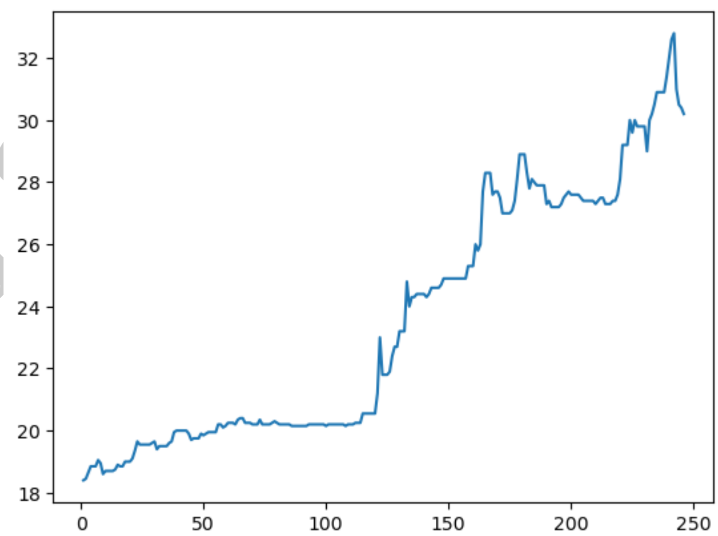


El siguiente ejemplo contiene un mayor número de datos, en este caso, temperaturas de un lugar. Obviamente también existe la posibilidad de cargar los datos desde un archivo de datos.

[177]:
```
# temp1 son los datos ha representar en el eje de las ordenadas (eje y)
temp1 =[18.40, 18.45, 18.65, 18.85, 18.85, 18.85, 19.05, 18.95, 18.60, 18.70,␣
 ↪18.70, 18.70,
        18.70, 18.75, 18.90, 18.85, 18.85, 19.00, 19.00, 19.00, 19.10, 19.35,␣
 ↪19.65, 19.55,
        19.55, 19.55, 19.55, 19.55, 19.60, 19.65, 19.40, 19.50, 19.50, 19.50,␣
 ↪19.50, 19.60,
        19.65, 19.95, 20.00, 20.00, 20.00, 20.00, 20.00, 19.90, 19.70, 19.75,␣
 ↪19.75, 19.75,
        19.90, 19.85, 19.90, 19.95, 19.95, 19.95, 19.95, 20.20, 20.20, 20.10,␣
 ↪20.15, 20.25,
        20.25, 20.25, 20.20, 20.35, 20.40, 20.40, 20.25, 20.25, 20.25, 20.20,␣
 ↪20.20, 20.20,
        20.35, 20.20, 20.20, 20.20, 20.20, 20.25, 20.30, 20.25, 20.20, 20.20,␣
 ↪20.20, 20.20,
        20.20, 20.15, 20.15, 20.15, 20.15, 20.15, 20.15, 20.15, 20.20, 20.20,␣
 ↪20.20, 20.20,
        20.20, 20.20, 20.20, 20.15, 20.20, 20.20, 20.20, 20.20, 20.20, 20.20,␣
 ↪20.20, 20.15,
        20.20, 20.20, 20.20, 20.25, 20.25, 20.25, 20.55, 20.55, 20.55, 20.55,␣
 ↪20.55, 20.55,
        21.20, 23.00, 21.80, 21.80, 21.80, 21.90, 22.40, 22.70, 22.70, 23.20,␣
 ↪23.20, 23.20,
```

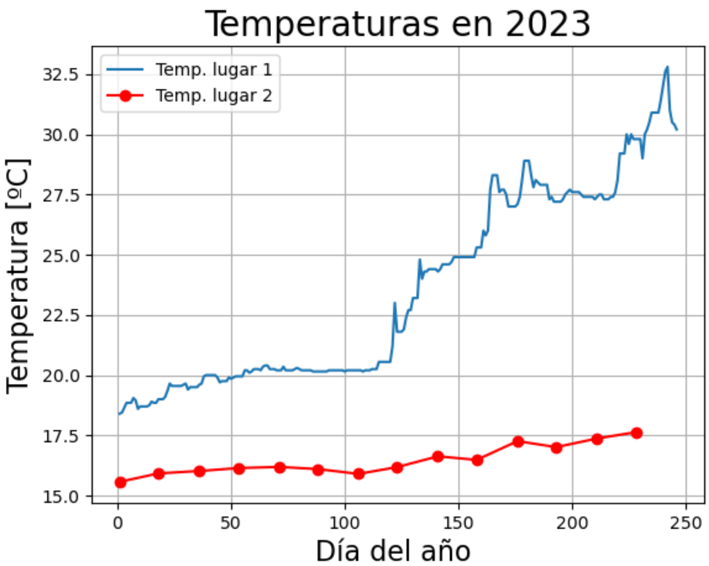
```
        24.80, 24.00, 24.30, 24.30, 24.40, 24.40, 24.40, 24.40, 24.30, 24.40,␣
↪24.60, 24.60,
        24.60, 24.60, 24.70, 24.90, 24.90, 24.90, 24.90, 24.90, 24.90, 24.90,␣
↪24.90, 24.90,
        24.90, 25.30, 25.30, 25.30, 26.00, 25.80, 26.00, 27.70, 28.30, 28.30,␣
↪28.30, 27.60,
        27.70, 27.70, 27.50, 27.00, 27.00, 27.00, 27.00, 27.10, 27.40, 28.10,␣
↪28.90, 28.90,
        28.90, 28.30, 27.80, 28.10, 28.00, 27.90, 27.90, 27.90, 27.90, 27.30,␣
↪27.40, 27.20,
        27.20, 27.20, 27.20, 27.30, 27.50, 27.60, 27.70, 27.60, 27.60, 27.60,␣
↪27.60, 27.50,
        27.40, 27.40, 27.40, 27.40, 27.40, 27.30, 27.40, 27.50, 27.50, 27.30,␣
↪27.30, 27.30,
        27.40, 27.40, 27.60, 28.10, 29.20, 29.20, 29.20, 30.00, 29.60, 30.00,␣
↪29.80, 29.80,
        29.80, 29.80, 29.00, 30.00, 30.20, 30.50, 30.90, 30.90, 30.90, 30.90,␣
↪31.40, 32.00,
        32.60, 32.80, 31.00, 30.50, 30.40, 30.20]
# La variable 'dia' lo representarmos en el eje de las abscisas (eje x):
dia = range(1,247)  # días del 1 al 247
plt.plot(dia, temp1)
```

[177]: [<matplotlib.lines.Line2D at 0x14e81b890>]



Tenemos otro conjunto de datos con temperaturas de otro lugar, y queremos comparar el perfil de temperaturas de ambos lugares en el mismo gráfico.

[178]:
```
temp2 = [15.57, 15.92, 16.02, 16.14, 16.19, 16.10, 15.90, 16.18, 16.63,
         16.48, 17.26, 17.01, 17.37, 17.63]
# ... y un registro del tiempo diferente
dia_2 = [1, 18, 36, 53, 71, 88, 106, 123, 141, 158, 176, 193, 211, 228]
plt.plot(dia  , temp1 , '-'  , label='Temp. lugar 1')
plt.plot(dia_2, temp2 , 'o-' , color="r", label='Temp. lugar 2')
plt.xlabel('Día del año ', fontsize=16)
plt.ylabel('Temperatura [ºC]', fontsize=16)
plt.title('Temperaturas en 2023', fontsize=20)
plt.legend()
plt.grid(True)
```



Índice alfabético