

---

# **Sage Tutorial**

***Publicación 7.1***

**The Sage Development Team**

20 de March de 2016



<b>1. Introducción</b>	<b>3</b>
1.1. Instalación . . . . .	4
1.2. Formas de usar Sage . . . . .	4
1.3. Metas a largo plazo de Sage . . . . .	4
<b>2. Un Tour Guiado</b>	<b>7</b>
2.1. Asignación, Igualdad y Aritmética . . . . .	7
2.2. Obteniendo Ayuda . . . . .	9
2.3. Funciones, Indentación Y Conteo . . . . .	10
2.4. Álgebra Y Cálculo Básicos . . . . .	14
<b>3. Indices and tables</b>	<b>21</b>
<b>Bibliografía</b>	<b>23</b>



Sage es un software matemático libre y de código abierto que apoya la investigación y la enseñanza en álgebra, geometría, teoría de números, criptografía, computación numérica y áreas relacionadas. Tanto el modelo de desarrollo como la tecnología en Sage se distinguen por un énfasis extremadamente fuerte en la apertura, comunidad, cooperación y colaboración: estamos construyendo el carro, no reinventando la rueda. La meta de Sage en su conjunto es la de crear una alternativa viable, libre y de código abierto a Maple, Mathematica, Magma y MATLAB.

Este tutorial es la mejor manera de familiarizarse con Sage en unas cuantas horas. Puedes leerlo en versión HTML o PDF, o desde el notebook de Sage (haz click en `Help`, luego haz click en `Tutorial` para trabajar interactivamente en el tutorial desde dentro de Sage).

Este trabajo está licenciado bajo una licencia [Creative Commons Attribution-Share Alike 3.0](#).



---

## Introducción

---

Completar este tutorial debería llevarte unas 3 o 4 horas. Puedes leerlo en versión HTML o PDF, o desde el notebook (interfaz interactiva vía web) de Sage (Haz click en `Help`, luego haz click en `Tutorial` para trabajar interactivamente en el tutorial desde dentro de Sage).

Aunque gran parte de Sage está implementado usando el lenguaje de programación Python, no es necesario ningún conocimiento previo de Python para poder leer este tutorial. En algún punto seguramente querrás aprender Python (¡un lenguaje muy divertido!), y hay muchos recursos gratuitos excelentes para hacerlo, incluyendo [\[PyT\]](#) y [\[Dive\]](#). Si tan solo quieres experimentar ligeramente con Sage, este tutorial es el lugar justo para empezar. Por ejemplo:

```
sage: 2 + 2
4
sage: factor(-2007)
-1 * 3^2 * 223

sage: A = matrix(4, 4, range(16)); A
[ 0  1  2  3]
[ 4  5  6  7]
[ 8  9 10 11]
[12 13 14 15]

sage: factor(A.charpoly())
x^2 * (x^2 - 30*x - 80)

sage: m = matrix(ZZ, 2, range(4))
sage: m[0,0] = m[0,0] - 3
sage: m
[-3  1]
[ 2  3]

sage: E = EllipticCurve([1, 2, 3, 4, 5]);
sage: E
Elliptic Curve defined by y^2 + x*y + 3*y = x^3 + 2*x^2 + 4*x + 5
over Rational Field
sage: E.anlist(10)
[0, 1, 1, 0, -1, -3, 0, -1, -3, -3]
sage: E.rank()
1

sage: k = 1/(sqrt(3)*I + 3/4 + sqrt(73)*5/9); k
36/(20*sqrt(73) + 36*I*sqrt(3) + 27)
sage: N(k)
0.165495678130644 - 0.0521492082074256*I
sage: N(k, 30)      # 30 "bits"
```

```
0.16549568 - 0.052149208*I
sage: latex(k)
\frac{36}{20} \sqrt{73} + 36 i \sqrt{3} + 27
```

## 1.1 Instalación

Si no tienes instalado Sage en tu computador y sólo quieres probar algunos comandos, usa la versión en línea en <http://www.sagenb.org>.

Mira la Guía De Instalación Para Sage en la sección de documentación de la página web principal de [Sage] para obtener instrucciones sobre cómo instalar Sage en tu computador. Aquí hacemos simplemente dos comentarios:

1. El archivo de descarga de Sage viene con “baterías incluidas”. En otras palabras, aunque Sage utiliza Python, IPython, PARI, GAP, Singular, Maxima, NTL, GMP, etc., no necesitas instalarlos por separado pues ya están incluidos con la distribución de Sage. Sin embargo, para utilizar ciertas características de Sage, por ejemplo, Macaulay o KASH, debes instalar el paquete opcional relevante o al menos tener los programas pertinentes ya instalados en tu computador. Macaulay y KASH son paquetes opcionales de Sage (para una lista de los paquetes opcionales disponibles, teclea `sage -optional`, o navega por la página de descarga “Download” en el sitio web de Sage).
2. La versión binaria precompilada de Sage (que se encuentra en el sitio web de Sage) puede ser más rápida y fácil de instalar que la versión en código fuente. Sólo desempaqueta el archivo y ejecuta `sage`.
3. Si quieres utilizar el paquete SageTeX (el cual te permite insertar los resultados de tus cálculos con Sage en un archivo LaTeX), necesitarás hacerle conocer SageTeX a tu distribución de TeX. Para hacer esto, consulta la sección “Haciendo que TeX conozca a SageTeX” en la guía de instalación de Sage [Sage installation guide](#) (Este enlace debería llevarte a tu copia local de la guía de instalación). Es bastante sencillo: sólo necesitas establecer una variable de entorno o copiar un solo archivo en un directorio en el que TeX va a buscar.

La documentación para usar SageTeX se encuentra en `$SAGE_ROOT/local/share/texmf/tex/generic/sagetex/`, donde “`$SAGE_ROOT`” se refiere al directorio donde Sage está instalado – por ejemplo, `/opt/sage-4.2.1`.

## 1.2 Formas de usar Sage

Puedes usar Sage de varias maneras.

- **Interfaz gráfico del Notebook:** Permite usar Sage en forma interactiva desde el navegador web. Véase la sección que trata sobre el Notebook en el manual de referencia,
- **Línea de comandos interactiva;**
- **Programas:** Escribiendo programas compilados e interpretados en Sage y
- **Scripts:** Escribiendo scripts (archivos de órdenes) independientes en Python que utilizan la biblioteca Sage.

## 1.3 Metas a largo plazo de Sage

- **Útil:** La audiencia a la que está destinado Sage son los estudiantes de matemáticas (desde la secundaria hasta la universidad), profesores y matemáticos (para la investigación). El objetivo es proveer un software que pueda usarse para explorar y experimentar con construcciones matemáticas en álgebra, geometría, teoría de números, cálculo, computación numérica, etc. Sage facilita la experimentación interactiva con objetos matemáticos.



- **Eficiente:** Queremos que sea rápido. Sage utiliza software maduro y altamente optimizado: GMP, PARI, GAP y NTL, por lo que es muy rápido en ciertas operaciones.
- **Libre y de código abierto:** El código fuente debe ser legible y libremente disponible, de modo que los usuarios puedan entender qué está haciendo realmente el sistema y así poder extenderlo fácilmente. Tal como los matemáticos logran un entendimiento más profundo de un teorema al leerlo cuidadosamente o, por lo menos, al echarle una ojeada a la prueba, la gente que efectúa cálculos debe ser capaz de comprender cómo funcionan los cálculos leyendo el código fuente documentado. Si utilizas Sage para hacer cálculos en un artículo que vas a publicar, puedes estar seguro que tus lectores siempre tendrán libre acceso a Sage y a todo su código fuente, y hasta se te permite archivar y re-distribuir la versión de Sage que usaste.
- **Fácil de compilar:** Sage tiene que ser fácil de compilar desde el código fuente para los usuarios de Linux, OS X y Windows. Esto provee a los usuarios de una mayor flexibilidad para que modifiquen el sistema.
- **Cooperación con otros programas:** Sage debe proveer interfaces robustos a la mayoría de sistemas algebraicos de cómputo, incluyendo PARI, GAP, Singular, Maxima, KASH, Magma, Maple y Mathematica. Sage pretende unificar y extender el software matemático existente.
- **Bien documentado:** Debemos proveer un tutorial, una guía de programación, un manual de referencia y documentos sobre cómo hacer cosas específicas, con numerosos ejemplos y discusiones de las bases matemáticas.
- **Extensible:** Debe ser posible definir nuevos tipos de datos o derivar de tipos incorporados y utilizar código escrito en una amplia gama de lenguajes.
- **Fácil de usar:** Debe de ser fácil comprender qué funcionalidad se ha provisto para un objeto dado y examinar la documentación y el código fuente, así como alcanzar un alto nivel de soporte al usuario.



---

## Un Tour Guiado

---

Esta sección es un tour guiado de las opciones disponibles en Sage. Para muchos más ejemplos, véase “Construcciones En Sage”, con la cual se intenta responder a la pregunta general “¿Cómo hago para construir ...?”. Véase también el “Manual De Referencia De Sage”, el cual tiene miles de ejemplos. Observa también que puedes trabajar interactivamente con este tour en el notebook de Sage haciendo click en el enlace [Help](#).

(Si estás viendo el tutorial en el notebook de Sage, presiona `shift-enter` para evaluar cualquier celda de entrada de datos. Incluso puedes editar la entrada de datos antes de presionar `shift-enter`. En algunas Macs tendrías que presionar `shift-return` en lugar de `shift-enter`.)

### 2.1 Asignación, Igualdad y Aritmética

Con algunas excepciones menores, Sage utiliza el lenguaje de programación Python, de modo que la mayoría de los libros introductorios sobre Python te ayudarán a aprender Sage.

Sage utiliza `=` para la asignación. Utiliza `==`, `<=`, `>=`, `<` y `>` para la comparación:

```
sage: a = 5
sage: a
5
sage: 2 == 2
True
sage: 2 == 3
False
sage: 2 < 3
True
sage: a == 5
True
```

Sage provee todo lo relacionado con las operaciones matemáticas básicas:

```
sage: 2**3      # ** significa exponente
8
sage: 2^3       # ^ es un sinónimo de ** (diferente de Python)
8
sage: 10 % 3    # para argumentos enteros, % significa mod, es decir, residuo
1
sage: 10/4
5/2
sage: 10//4     # para argumentos enteros, // devuelve el cociente de enteros
2
sage: 4 * (10 // 4) + 10 % 4 == 10
```

```
True
sage: 3^2*4 + 2%5
38
```

El cálculo de una expresión tal como  $3^2 \cdot 4 + 2 \% 5$  depende de el orden en que las operaciones son aplicadas.

Sage también provee muchas funciones matemáticas conocidas; he aquí solo unos cuantos ejemplos:

```
sage: sqrt(3.4)
1.84390889145858
sage: sin(5.135)
-0.912021158525540
sage: sin(pi/3)
1/2*sqrt(3)
```

Como demuestra el último ejemplo, algunas expresiones matemáticas devuelven valores ‘exactos’, en lugar de aproximaciones numéricas. Para obtener una aproximación numérica, utilice la función `n` o el método `n` (ámbas tienen un nombre más largo, `numerical_approx`, y la función `N` es la misma que `n`). Éstas toman argumentos opcionales `prec`, que es el número requerido de bits de precisión, y `digits`, que es el número requerido de dígitos decimales de precisión; el número predeterminado es de 53 bits de precisión.

```
sage: exp(2)
e^2
sage: n(exp(2))
7.38905609893065
sage: sqrt(pi).numerical_approx()
1.77245385090552
sage: sin(10).n(digits=5)
-0.54402
sage: N(sin(10), digits=10)
-0.5440211109
sage: numerical_approx(pi, prec=200)
3.1415926535897932384626433832795028841971693993751058209749
```

Python es un lenguaje de tipado dinámico, de modo que el valor referido por cada variable tiene un tipo asociado. Pero una variable dada puede contener valores de cualquier tipo Python dentro de un ámbito dado:

```
sage: a = 5    # a es un entero
sage: type(a)
<type 'sage.rings.integer.Integer'>
sage: a = 5/3  # ahora es un número racional
sage: type(a)
<type 'sage.rings.rational.Rational'>
sage: a = 'hello' # ahora es una cadena
sage: type(a)
<type 'str'>
```

El lenguaje de programación C, que es un lenguaje de tipado estático, es muy diferente; una variable declarada como `int` solo puede contener un `int` en su ámbito.

Una fuente posible de confusión en Python es el que un entero literal que comienza con un cero es tratado como un número octal, es decir, un número en base 8.

```
sage: 011
9
sage: 8 + 1
9
sage: n = 011
```

```
sage: n.str(8)    # representación en cadena de n en base 8
'11'
```

Esto es consistente con el lenguaje de programación C.

## 2.2 Obteniendo Ayuda

Sage posee una extensa documentación incorporada, accesible con solo teclear el nombre de una función o una constante (por ejemplo), seguido de un signo de interrogación:

```
sage: tan?
Type:      <class 'sage.calculus.calculus.Function_tan'>
Definition: tan( [noargspec] )
Docstring:
```

The tangent function

EXAMPLES:

```
sage: tan(pi)
0
sage: tan(3.1415)
-0.0000926535900581913
sage: tan(3.1415/4)
0.999953674278156
sage: tan(pi/4)
1
sage: tan(1/2)
tan(1/2)
sage: RR(tan(1/2))
0.546302489843790
```

```
sage: log2?
Type:      <class 'sage.functions.constants.Log2'>
Definition: log2( [noargspec] )
Docstring:
```

The natural logarithm of the real number 2.

EXAMPLES:

```
sage: log2
log2
sage: float(log2)
0.69314718055994529
sage: RR(log2)
0.693147180559945
sage: R = RealField(200); R
Real Field with 200 bits of precision
sage: R(log2)
0.69314718055994530941723212145817656807550013436025525412068
sage: l = (1-log2)/(1+log2); l
(1 - log(2))/(log(2) + 1)
sage: R(l)
0.18123221829928249948761381864650311423330609774776013488056
sage: maxima(log2)
log(2)
sage: maxima(log2).float()
.6931471805599453
```

```
sage: gp(log2)
      0.6931471805599453094172321215          # 32-bit
      0.69314718055994530941723212145817656807 # 64-bit
sage: sudoku?
File:      sage/local/lib/python2.5/site-packages/sage/games/sudoku.py
Type:      <type 'function'>
Definition: sudoku(A)
Docstring:
```

Solve the 9x9 Sudoku puzzle defined by the matrix A.

EXAMPLE:

```
sage: A = matrix(ZZ,9,[5,0,0, 0,8,0, 0,4,9, 0,0,0, 5,0,0,
0,3,0, 0,6,7, 3,0,0, 0,0,1, 1,5,0, 0,0,0, 0,0,0, 0,0,0, 2,0,8, 0,0,0,
0,0,0, 0,0,0, 0,1,8, 7,0,0, 0,0,4, 1,5,0, 0,3,0, 0,0,2,
0,0,0, 4,9,0, 0,5,0, 0,0,3])
sage: A
[5 0 0 0 8 0 0 4 9]
[0 0 0 5 0 0 0 3 0]
[0 6 7 3 0 0 0 0 1]
[1 5 0 0 0 0 0 0 0]
[0 0 0 2 0 8 0 0 0]
[0 0 0 0 0 0 0 1 8]
[7 0 0 0 0 4 1 5 0]
[0 3 0 0 0 2 0 0 0]
[4 9 0 0 5 0 0 0 3]
sage: sudoku(A)
[5 1 3 6 8 7 2 4 9]
[8 4 9 5 2 1 6 3 7]
[2 6 7 3 4 9 5 8 1]
[1 5 8 4 6 3 9 7 2]
[9 7 4 2 1 8 3 6 5]
[3 2 6 7 9 5 4 1 8]
[7 8 2 9 3 4 1 5 6]
[6 3 5 1 7 2 8 9 4]
[4 9 1 8 5 6 7 2 3]
```

Sage también provee ‘Autocompletado con el tabulador’: teclea las primeras letras de una función y luego oprime la tecla del tabulador. Por ejemplo, si tecleas `ta` seguido por TAB, Sage imprimirá `tachyon`, `tan`, `tanh`, `taylor`. Esto proporciona una buena manera de encontrar los nombres de funciones y otras estructuras en Sage.

## 2.3 Funciones, Indentación Y Conteo

Para definir una nueva función en Sage, utilice el comando `def` y el signo de dos puntos después de la lista de nombres de variable. Por ejemplo:

```
sage: def is_even(n):
....:     return n % 2 == 0
....:
sage: is_even(2)
True
sage: is_even(3)
False
```

Nota: Dependiendo de la versión del tutorial que estás leyendo, puede que veas puntos `....:` en la segunda línea de este ejemplo. No los incluyas; son solo para enfatizar que el código está indentado. Siempre que este sea el caso,

presiona [Return/Enter] una vez al final del bloque para insertar una línea en blanco y concluir la definición de la función.

No tienes que especificar los tipos de ninguno de los argumentos de entrada. Puedes especificar múltiples entradas, cada una de las cuales puede tener un valor predeterminado opcional. Por ejemplo, la función de abajo tiene un valor predeterminado `divisor=2` si no se especifica el valor de `divisor`.

```
sage: def is_divisible_by(number, divisor=2):
.....:     return number % divisor == 0
sage: is_divisible_by(6,2)
True
sage: is_divisible_by(6)
True
sage: is_divisible_by(6, 5)
False
```

También puedes especificar explícitamente una o ambas de las entradas cuando llames a la función; si especificas las entradas explícitamente, puedes darlas en cualquier orden:

```
sage: is_divisible_by(6, divisor=5)
False
sage: is_divisible_by(divisor=2, number=6)
True
```

En Python, los bloques de código no se encierran entre llaves o bloques `begin...end` como en muchos otros lenguajes. En vez de ello, los bloques de código se indican por medio de la indentación, la cual se debe agrupar con exactitud. Por ejemplo, el siguiente es un error de sintaxis porque la declaración `return` no está indentada al mismo nivel que las otras líneas por encima de ella.

```
sage: def even(n):
.....:     v = []
.....:     for i in range(3,n):
.....:         if i % 2 == 0:
.....:             v.append(i)
.....:     return v
Syntax Error:
return v
```

Si arreglas la indentación, la función se ejecutará:

```
sage: def even(n):
.....:     v = []
.....:     for i in range(3,n):
.....:         if i % 2 == 0:
.....:             v.append(i)
.....:     return v
sage: even(10)
[4, 6, 8]
```

El punto y coma no es necesario al final de las líneas. Una línea termina, en muchos casos, por un carácter de nueva línea. Sin embargo, puedes poner múltiples declaraciones en una línea, separadas por punto y coma:

```
sage: a = 5; b = a + 3; c = b^2; c
64
```

Si quisieras que una simple línea de código abarque múltiples líneas, utiliza una barra invertida como terminación:

```
sage: 2 + \
....: 3
5
```

En Sage, se cuenta iterando sobre un rango de enteros. Por ejemplo, la primer línea de abajo es exactamente igual a `for(i=0; i<3; i++)` en C++ o Java:

```
sage: for i in range(3):
....:     print i
0
1
2
```

La primer línea de abajo es igual a `for(i=2; i<5; i++)`.

```
sage: for i in range(2,5):
....:     print i
2
3
4
```

El tercer argumento controla el incremento, de modo que lo siguiente es igual a `for(i=1; i<6; i+=2)`.

```
sage: for i in range(1,6,2):
....:     print i
1
3
5
```

A menudo, querrás crear una tabla para presentar números que has calculado utilizando Sage. Una manera sencilla de hacer esto es usando el formateado de cadenas. Abajo, creamos tres columnas, cada una con un ancho exácto de 6 caracteres y hacemos una tabla de cuadrados y cubos.

```
sage: for i in range(5):
....:     print '%6s %6s %6s'%(i, i^2, i^3)
0      0      0
1      1      1
2      4      8
3      9     27
4     16     64
```

La estructura de datos más básica en Sage es la lista, la cual es – como sugiere su nombre – solo una lista de objetos arbitrarios. Por ejemplo, el comando `range` que hemos usado crea una lista:

```
sage: range(2,10)
[2, 3, 4, 5, 6, 7, 8, 9]
```

He aquí una lista más complicada:

```
sage: v = [1, "hello", 2/3, sin(x^3)]
sage: v
[1, 'hello', 2/3, sin(x^3)]
```

El indexado de una lista comienza en el cero, como en muchos lenguajes de programación.

```
sage: v[0]
1
```



```
sage: v[3]
sin(x^3)
```

La función `len(v)` devuelve la longitud de `v`. Utiliza `v.append(obj)` para añadir un nuevo objeto al final de `v`, y utiliza `del v[i]` para borrar el  $i$ -ésimo elemento de `v`:

```
sage: len(v)
4
sage: v.append(1.5)
sage: v
[1, 'hello', 2/3, sin(x^3), 1.500000000000000]
sage: del v[1]
sage: v
[1, 2/3, sin(x^3), 1.500000000000000]
```

Otra estructura de datos importante es el diccionario (o array asociativo). Funciona como una lista, excepto que puede ser indexado con casi cualquier objeto (los índices deben ser inmutables):

```
sage: d = {'hi':-2, 3/8:pi, e:pi}
sage: d['hi']
-2
sage: d[e]
pi
```

También puedes definir nuevos tipos de datos usando clases. El encapsulado de objetos matemáticos con clases es una técnica potente que puede ayudar a simplificar y organizar tus programas en Sage. Abajo, definimos una clase que representa la lista de enteros positivos pares hasta  $n$ ; se deriva de el tipo básico `list`.

```
sage: class Evens(list):
....:     def __init__(self, n):
....:         self.n = n
....:         list.__init__(self, range(2, n+1, 2))
....:     def __repr__(self):
....:         return "Even positive numbers up to n."
```

El método `__init__` se llama para inicializar al objeto cuando es creado; el método `__repr__` imprime el objeto. Llamamos al método constructor de listas en la segunda línea del método `__init__`. A continuación, creamos un objeto de clase `Evens`:

```
sage: e = Evens(10)
sage: e
Even positive numbers up to n.
```

Observe que `e` se imprime usando el método `__repr__` que hemos definido. Para ver la lista subyacente de números, utilice la función `list`:

```
sage: list(e)
[2, 4, 6, 8, 10]
```

También podemos acceder al atributo `n` o tratar a `e` como una lista.

```
sage: e.n
10
sage: e[2]
6
```

## 2.4 Álgebra Y Cálculo Básicos

Sage puede efectuar cálculos relacionados al álgebra y cálculo básicos: por ejemplo, encontrar soluciones de ecuaciones, diferenciación, integración y transformadas de Laplace. Véa la documentación “Construcciones En Sage” para más ejemplos.

### 2.4.1 Resolviendo Ecuaciones

#### Resolviendo Ecuaciones De Manera Exacta

La función `solve` resuelve ecuaciones. Para usarla, primero no olvides especificar algunas variables. Los argumentos de `solve` son una ecuación (o un sistema de ecuaciones), junto con las variables a resolver:

```
sage: x = var('x')
sage: solve(x^2 + 3*x + 2, x)
[x == -2, x == -1]
```

Puedes resolver ecuaciones en una variable respecto de las demás:

```
sage: x, b, c = var('x b c')
sage: solve([x^2 + b*x + c == 0], x)
[x == -1/2*b - 1/2*sqrt(b^2 - 4*c), x == -1/2*b + 1/2*sqrt(b^2 - 4*c)]
```

Puedes también resolver ecuaciones en varias variables:

```
sage: x, y = var('x, y')
sage: solve([x+y==6, x-y==4], x, y)
[[x == 5, y == 1]]
```

El siguiente ejemplo del uso de Sage para resolver un sistema de ecuaciones no-lineales fue proporcionado por Jason Grout: primero, resolvemos el sistema simbólicamente:

```
sage: var('x y p q')
(x, y, p, q)
sage: eq1 = p+q==9
sage: eq2 = q*y+p*x==6
sage: eq3 = q*y^2+p*x^2==24
sage: solve([eq1,eq2,eq3,p==1], p,q,x,y)
[[p == 1, q == 8, x == -4/3*sqrt(10) - 2/3, y == 1/6*sqrt(5)*sqrt(2) - 2/3], [p == 1, q == 8, x == 4/3*sqrt(10) - 2/3, y == 1/6*sqrt(5)*sqrt(2) + 2/3]]
```

Si queremos aproximaciones numéricas de las soluciones, podemos usar lo siguiente:

```
sage: solns = solve([eq1,eq2,eq3,p==1], p,q,x,y, solution_dict=True)
sage: [[s[p].n(30), s[q].n(30), s[x].n(30), s[y].n(30)] for s in solns]
[[1.0000000, 8.0000000, -4.8830369, -0.13962039],
 [1.0000000, 8.0000000, 3.5497035, -1.1937129]]
```

(La función `n` imprime una aproximación numérica, y el argumento es el número de bits de precisión.)

#### Resolviendo Ecuaciones Numéricamente

A menudo, `solve` no podrá encontrar una solución exacta para la ecuación o ecuaciones especificadas. Cuando falla, puedes usar `find_root` para encontrar una solución numérica. Por ejemplo, `solve` no devuelve nada interesante para la siguiente ecuación:

```
sage: theta = var('theta')
sage: solve(cos(theta)==sin(theta), theta)
[sin(theta) == cos(theta)]
```

Por otro lado, podemos usar `find_root` para encontrar una solución a la ecuación de arriba en el rango  $0 < \theta < \pi/2$ :

```
sage: phi = var('phi')
sage: find_root(cos(phi)==sin(phi), 0, pi/2)
0.785398163397448...
```

## 2.4.2 Diferenciación, Integración, etc.

Sage sabe cómo diferenciar e integrar muchas funciones. Por ejemplo, para diferenciar  $\sin(u)$  con respecto a  $u$ , haz lo siguiente:

```
sage: u = var('u')
sage: diff(sin(u), u)
cos(u)
```

Para calcular la cuarta derivada de  $\sin(x^2)$ :

```
sage: diff(sin(x^2), x, 4)
16*x^4*sin(x^2) - 48*x^2*cos(x^2) - 12*sin(x^2)
```

Para calcular las derivadas parciales de  $x^2 + 17y^2$  con respecto a  $x$  e  $y$ , respectivamente:

```
sage: x, y = var('x, y')
sage: f = x^2 + 17*y^2
sage: f.diff(x)
2*x
sage: f.diff(y)
34*y
```

También podemos calcular integrales, tanto indefinidas como definidas. Para calcular  $\int x \sin(x^2) dx$  y  $\int_0^1 \frac{x}{x^2+1} dx$

```
sage: integral(x*sin(x^2), x)
-1/2*cos(x^2)
sage: integral(x/(x^2+1), x, 0, 1)
1/2*log(2)
```

Para calcular la descomposición en fracciones simples de  $\frac{1}{x^2-1}$ :

```
sage: f = 1/((1+x)*(x-1))
sage: f.partial_fraction(x)
-1/2/(x + 1) + 1/2/(x - 1)
```

## 2.4.3 Resolviendo Ecuaciones Diferenciales

Puedes usar a Sage para investigar ecuaciones diferenciales ordinarias. Para resolver la ecuación  $x' + x - 1 = 0$ :

```
sage: t = var('t')      # defina una variable t
sage: x = function('x')(t)  # defina x como una función de esa variable
sage: DE = diff(x, t) + x - 1
sage: desolve(DE, [x, t])
(_C + e^t)*e^(-t)
```

Esto utiliza el interfaz a Maxima de Sage [Max], por lo que el resultado puede diferir de otros resultados de Sage. En este caso, la salida nos dice que la solución general a la ecuación diferencial es  $x(t) = e^{-t}(e^t + c)$ .

También puedes calcular transformadas de Laplace; la transformada de Laplace de  $t^2 e^t - \sin(t)$  se calcula como sigue:

```
sage: s = var("s")
sage: t = var("t")
sage: f = t^2*exp(t) - sin(t)
sage: f.laplace(t,s)
-1/(s^2 + 1) + 2/(s - 1)^3
```

Veamos un ejemplo más complicado. El desplazamiento desde el punto de equilibrio de dos resortes acoplados, sujetos a una pared a la izquierda

```
|-----\\/\ /\ /\ /\----|masa1|----\\/\ /\ /\ /\----|masa2|
          resorte1                resorte2
```

está modelado por el sistema de ecuaciones diferenciales de segundo orden

$$\begin{aligned} m_1 x_1'' + (k_1 + k_2)x_1 - k_2 x_2 &= 0 \\ m_2 x_2'' + k_2(x_2 - x_1) &= 0, \end{aligned}$$

donde  $m_i$  es la masa del objeto  $i$ ,  $x_i$  es el desplazamiento desde el equilibrio de la masa  $i$ , y  $k_i$  es la constante de elasticidad del resorte  $i$ .

**Ejemplo:** Utiliza Sage para resolver el problema de arriba con  $m_1 = 2$ ,  $m_2 = 1$ ,  $k_1 = 4$ ,  $k_2 = 2$ ,  $x_1(0) = 3$ ,  $x_1'(0) = 0$ ,  $x_2(0) = 3$ ,  $x_2'(0) = 0$ .

Solución: Toma la transformada de Laplace de la primera ecuación (con la notación  $x = x_1$ ,  $y = x_2$ ):

```
sage: de1 = maxima("2*diff(x(t),t, 2) + 6*x(t) - 2*y(t)")
sage: lde1 = de1.laplace("t","s"); lde1
2*(-%at('diff(x(t),t,1),t=0)+s^2*'laplace(x(t),t,s)-x(0)*s)-2*'laplace(y(t),t,s)+6*'laplace(x(t),t,s)
```

El resultado puede ser difícil de leer, pero significa que

$$-2x'(0) + 2s^2 * X(s) - 2sx(0) - 2Y(s) + 6X(s) = 0$$

(donde la transformada de Laplace de una función en letra minúscula como  $x(t)$  es la función en letra mayúscula  $X(s)$ ). Toma la transformada de Laplace de la segunda ecuación:

```
sage: de2 = maxima("diff(y(t),t, 2) + 2*y(t) - 2*x(t)")
sage: lde2 = de2.laplace("t","s"); lde2
-%at('diff(y(t),t,1),t=0)+s^2*'laplace(y(t),t,s)+2*'laplace(y(t),t,s)-2*'laplace(x(t),t,s)-y(0)*s
```

Esto dice

$$-Y'(0) + s^2 Y(s) + 2Y(s) - 2X(s) - sy(0) = 0.$$

Introduce las condiciones iniciales para  $x(0)$ ,  $x'(0)$ ,  $y(0)$  y  $y'(0)$  y resuelve las dos ecuaciones resultantes:

```
sage: var('s X Y')
(s, X, Y)
sage: eqns = [(2*s^2+6)*X-2*Y == 6*s, -2*X+(s^2+2)*Y == 3*s]
sage: solve(eqns, X,Y)
[[X == 3*(s^3 + 3*s)/(s^4 + 5*s^2 + 4),
  Y == 3*(s^3 + 5*s)/(s^4 + 5*s^2 + 4)]]
```

Ahora toma la transformada inversa de Laplace para obtener la respuesta:

```
sage: var('s t')
(s, t)
sage: inverse_laplace((3*s^3 + 9*s)/(s^4 + 5*s^2 + 4), s, t)
cos(2*t) + 2*cos(t)
sage: inverse_laplace((3*s^3 + 15*s)/(s^4 + 5*s^2 + 4), s, t)
-cos(2*t) + 4*cos(t)
```

Por tanto, la solución es

$$x_1(t) = \cos(2t) + 2\cos(t), \quad x_2(t) = 4\cos(t) - \cos(2t).$$

La solución puede dibujarse paramétricamente usando

```
sage: t = var('t')
sage: P = parametric_plot((cos(2*t) + 2*cos(t), 4*cos(t) - cos(2*t)), \
....: (0, 2*pi), rgbcolor=hue(0.9))
sage: show(P)
```

Los componentes individuales pueden dibujarse usando

```
sage: t = var('t')
sage: p1 = plot(cos(2*t) + 2*cos(t), 0, 2*pi, rgbcolor=hue(0.3))
sage: p2 = plot(4*cos(t) - cos(2*t), 0, 2*pi, rgbcolor=hue(0.6))
sage: show(p1 + p2)
```

REFERENCIAS: Nagle, Saff, Snider, Fundamentos De Ecuaciones Diferenciales, 6a ed, Addison-Wesley, 2004. (véase § 5.5).

## 2.4.4 Método De Euler Para Sistemas De Ecuaciones Diferenciales

En el siguiente ejemplo, ilustraremos el método de Euler para EDOs de primer y segundo orden. Primero, recordemos la idea básica para ecuaciones de primer orden. Dado un problema con valor inicial de la forma

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

queremos encontrar el valor aproximado de la solución en  $x = b$  con  $b > a$ .

Recuerda de la definición de derivada que

$$y'(x) \approx \frac{y(x+h) - y(x)}{h},$$

donde  $h > 0$  está dado y es pequeño. Esto, junto con la ED, dan  $f(x, y(x)) \approx \frac{y(x+h)-y(x)}{h}$ . Ahora resuelve para  $y(x+h)$ :

$$y(x+h) \approx y(x) + h * f(x, y(x)).$$

Si llamamos a  $hf(x, y(x))$  el “término de corrección” (a falta de algo mejor), llamamos a  $y(x)$  “el valor viejo de  $y$ ”, y llamamos a  $y(x+h)$  el “nuevo valor de  $y$ ”, entonces, esta aproximación puede re-expresarse como

$$y_{nuevo} \approx y_{viejo} + h * f(x, y_{viejo}).$$

Si descomponemos el intervalo desde  $a$  a  $b$  en  $n$  pasos, de modo que  $h = \frac{b-a}{n}$ , podemos guardar la información dada por este método en una tabla.

$x$	$y$	$hf(x, y)$
$a$	$c$	$hf(a, c)$
$a + h$	$c + hf(a, c)$	...
$a + 2h$	...	
...		
$b = a + nh$	???	...

La meta es llenar todos los espacios de la tabla, una fila cada la vez, hasta que lleguemos a la casilla ???, que será la aproximación del método de Euler para  $y(b)$ .

La idea para los sistemas de EDOs es similar.

**Ejemplo:** Aproxima numéricamente  $z(t)$  en  $t = 1$  usando 4 pasos del método de Euler, donde  $z'' + tz' + z = 0$ ,  $z(0) = 1$ ,  $z'(0) = 0$ .

Debemos reducir la EDO de segundo orden a un sistema de dos EDs de primer orden (usando  $x = z$ ,  $y = z'$ ) y aplicar el método de Euler:

```
sage: t, x, y = PolynomialRing(RealField(10), 3, "txy").gens()
sage: f = y; g = -x - y * t
sage: eulers_method_2x2(f, g, 0, 1, 0, 1/4, 1)
      t          x          h*f(t, x, y)          y          h*g(t, x, y)
      0          1          0.00          0          -0.25
      1/4        1.0        -0.062        -0.25        -0.23
      1/2        0.94        -0.12        -0.48        -0.17
      3/4        0.82        -0.16        -0.66        -0.081
      1          0.65        -0.18        -0.74        0.022
```

Por tanto,  $z(1) \approx 0.75$ .

También podemos dibujar los puntos  $(x, y)$  para obtener una representación aproximada de la curva. La función que hace esto es `eulers_method_2x2_plot`. Para poder usarla, necesitamos definir las funciones  $f$  y  $g$  que toman un argumento con tres coordenadas:  $(t, x, *y*)$ .

```
sage: f = lambda z: z[2]          # f(t, x, y) = y
sage: g = lambda z: -sin(z[1])    # g(t, x, y) = -sin(x)
sage: P = eulers_method_2x2_plot(f, g, 0.0, 0.75, 0.0, 0.1, 1.0)
```

A estas alturas,  $P$  está guardando dos gráficas:  $P[0]$ , el gráfico de  $x$  vs.  $t$ , y  $P[1]$ , el gráfico de  $y$  vs.  $t$ . Podemos mostrar ámbas como sigue:

```
sage: show(P[0] + P[1])
```

## 2.4.5 Funciones Especiales

Se han implementado varios polinomios ortogonales y funciones especiales, utilizando tanto PARI [GAP] como Maxima [Max]. Estas funciones están documentadas en las secciones apropiadas (“Polinomios Ortogonales” y “Funciones Especiales”, respectivamente) del manual de referencia de Sage.

```
sage: x = polygen(QQ, 'x')
sage: chebyshev_U(2, x)
4*x^2 - 1
sage: bessell_I(1, 1).n(250)
0.56515910399248502720769602760986330732889962162109200948029448947925564096
sage: bessell_I(1, 1).n()
0.565159103992485
sage: bessell_I(2, 1.1).n() # los últimos dígitos son al azar
0.16708949925104...
```

Hasta este punto, Sage únicamente ha encapsulado estas funciones para uso numérico. Para uso simbólico, por favor utiliza directamente la interfaz a Maxima, como en el siguiente ejemplo:

```
sage: maxima.eval("f:bessel_y(v, w)")
'bessel_y(v,w)'
sage: maxima.eval("diff(f,w)")
'(bessel_y(v-1,w)-bessel_y(v+1,w))/2'
```





---

## Indices and tables

---

- *genindex*
- *modindex*
- *search*



[Dive] Sumérgete en Python, libremente disponible online en <http://diveintopython.net>

[PyT] El Tutorial De Python, <http://www.python.org/>

[Sage] Sage, <http://www.sagemath.org>

[GAP] El Grupo GAP, GAP – Grupos, Algoritmos y Programación, <http://www.gap-system.org>

[Max] Maxima, <http://maxima.sf.net/>