PROYECTO DE INNOVACIÓN DOCENTE
SINGALCON - Desarrollo de programas en SINGULAR para el aprendizaje del Álgebra Conmutativa

Memoria Final

Esteban Gómez González
Ana Cristina López Martín
Darío Sánchez Gómez
Tomás Carlos Tejero Prieto

30 de junio de 2014

Índice general

1. Elaboración de material docente y distribución **5**

1.1. Puntos fuertes 6

1.2. Puntos débiles 7

2. Seminarios formativos **8**

2.1. Puntos fuertes 8

2.2. Puntos débiles 9

3. Anexo: Materiales didácticos **10**

El proyecto SINGALCON - Desarrollo de programas en SINGULAR para el aprendizaje del Álgebra Conmutativa se ha desarrollado con normalidad en el ámbito de actuación de las asignaturas "Álgebra"del segundo curso del Grado en Matemáticas, "Álgebra Conmutativa y Computacional" y "Ampliación de Álgebra Conmutativa"del tercer curso del Grado en Matemáticas. Se espera que la experiencia recogida durante la ejecución de este proyecto ayude al equipo docente implicado en su ejecución a mejorar, tanto el desarrollo de las mismas como a proponer propuestas de cambio en su programación de manera que los contenidos, competencias y habilidades sean más realistas y se garantice su consecución.

El desarrollo y la ejecución del proyecto se ha articulado sobre dos ejes de actuación, siguiendo la naturaleza de los objetivos propuestos en la memoria de solicitud del mismo, que enunciaremos a continuación y que serán analizados a lo largo de este documento:

1. Material docente y distribución.
    - Desarrollo de una colección de programas en SINGULAR que permitan resolver los principales problemas que se plantean a los estudiantes en las clases prácticas de las asignaturas: "Álgebra", "Álgebra Conmutativa y Computacional" y "Ampliación de Álgebra Conmutativa" del Grado en Matemáticas.
    - Redacción de un manual con la documentación necesaria por una parte para la instalación y manejo de SAGE y por otra para el uso autónomo de los programas desarrollados en SINGULAR.
    - Distribución de los programas y la documentación a los estudiantes de las asignaturas arriba mencionadas a través de la plataforma STUDIUM.
2. Seminarios tutelados.
    - Realización de seminarios formativos dirigidos a los estudiantes en los que se les expliquen las principales funcionalidades de los programas.

Esto ha contribuido a la adquisición de las siguientes competencias asociadas a los módulos "Estructuras Algebraicas" y "Ampliación de Álgebra"del actual plan de estudio de Grado, dentro del cual se encuentran las asignaturas de actuación de este proyecto:

1. Aprender de manera autónoma nuevos conocimientos y técnicas. (Competencia CG-5, Memoria del Grado en Matemáticas de la USAL).

2. Utilizar aplicaciones informáticas de análisis estadístico, cálculo numérico y simbólico, visualización gráfica, optimización u otras para experimentar en Matemáticas y resolver problemas (Competencia CE-3 según la memoria de enseñanzas de Grado en Matemáticas de la USAL).

3. Desarrollar programas que resuelvan problemas metemáticos utilizando en cada caso un entorno computacional adecuado (Competencia CE-4 según la memoria de enseñanzas de Grado en Matemáticas de la USAL).

Capítulo 1

Elaboración de material docente y distribución

El primer elemento desarrollado en este proyecto de innovación docente, ha consistido en el diseño de una colección de algoritmos y programas desarrollados en SINGULAR mediante el software de álgebra computacional de distribución libre SAGE, que permiten resolver los principales problemas que se proponen a los estudiantes en las clases prácticas de las asignaturas Álgebra", "Álgebra Conmutativa y Computacional" y "Ampliación de Álgebra Conmutativa" del Grado en Matemáticas. El diseño de este material ha permitido realizar actividades formativas de carácter práctico que han contribuido a la adquisición de las competencias propias de las asignaturas incluidas en el ámbito de actuación del mismo. Los aspectos computacionales de las materias justifican la utilización de paquetes de cálculo específicos con software de álgebra computacional.

La actuación por parte de los docentes del equipo en este apartado ha consistido en la recopilación de las colecciones de problemas disponibles, identificación de aquellos susceptibles de ser programados en SINGULAR para las asignaturas antes mencionadas, así como diseñar nuevo material necesario para la realización de prácticas de ordenador con programas de álgebra computacional.

Conviene señalar que en principio el proyecto estaba concebido para ser desarrollado integramente mediante programas en SINGULAR. Ahora bien, teniendo en cuenta la integración de SINGULAR en SAGE, las capacidades

interactivas de los cuadernos de éste y su similitud en el manejo con los cuadernos de MATHEMATICA con el que los estudiantes están ampliamente familiarizados, con objeto de reducir la curva de aprendizaje y facilitar el acceso a la documentación de modo dinámico, se ha optado finalmente por implementar los algoritmos mediante cuadernos de SAGE. No obstante, una vez que ya está desarrollada toda la documentación para el manejo de SAGE y SINGULAR para que los estudiantes dispongan de ella desde el inicio del curso, se espera que esto facilite que, en proyectos de innovación docente que den continuación al que aquí nos ocupa, se proceda a migrar paulatinamente estos cuadernos de SAGE a programas en SINGULAR.

En segundo lugar se ha desarrollado un manual para la instalación y el manejo de SAGE, incluyendo la documentación para el uso de los algoritmos en los propios cuadernos desarrollados en SINGULAR mediante SAGE.

Los programas elaborados, las práctivas desarrolladas y la documentación se han puesto a disposición de los estudiantes, de las asignaturas arriba mencionadas, a través de la plataforma STUDIUM.

No obstante, la falta de tiempo del profesorado participante en este proyecto ha impedido el trasaldo de todo el material diseñado a la plataforma STUDIUM. Es por ello que algunos de los test planteados a los estudiantes a modo de autocontrol han tenido que ser realizados bajo el sistema tradicional, que aunque siendo muy válido, deja fuera ciertas posibilidades que ofrecen las nuevas tecnologías.

1.1. Puntos fuertes

- Los programas informáticos de álgebra computacional permiten elaborar una gran variedad de material docente.
- La posibilidad de disponer de un mecanismo de corrección de problemas permite al estudiante conocer su nivel de conocimiento y comprensión de la asignatura.
- La resolución de problemas de manera autónoma e independiente ayuda al estudiante a tener un ritmo de trabajo más constante.
- La implementación de algoritmos que resuelvan problemas supone un

beneficio también para el docente al servirle como método rápido y efectivo para elaborar nuevos problemas.

- Los algoritmos se han implementado en SINGULAR mediante cuadernos de SAGE en lugar de programas con objeto de reducir la curva de aprendizaje y facilitar el acceso a la documentación de modo dinámico.
- Uso de la plataforma STUDIUM como canal óptimo de comunicación con los estudiantes, tanto para facilitarles material docente como para resolución de dudas y consultas.

1.2. Puntos débiles

- La falta de recursos humanos ha imposibilitado implementar algoritmos para todos los principales tipos de problemas que deben saber resolver los estudiantes.

Capítulo 2

Seminarios formativos

El segundo eje de actuación ha sido el desarrollo práctico de las actividades mencionadas en el primer capítulo.

Dicha actividad se ha llevado a cabo a través de la puesta en marcha de seminarios formativos y prácticos tutelados en el aula de informática del Departamento de Matemáticas. En estos seminarios los estudiantes se dedicaron a conocer, bajo la supervisión de uno de los docentes del equipo, el funcionamiento de SAGE y el de los distintos cuadernos de SINGULAR implementados en SAGE, una vez que los correspondientes contenidos teóricos eran conocidos.

2.1. Puntos fuertes

- Los estudiantes mostraron un alto índice de participación en el desarrollo de los seminarios.
- Se complementan los ejercicios desarrollados en clase con los realizados en estas sesiones.
- La flexibilidad y rápidez de cálculo que supone el uso de programas informáticos permite a los estudiantes disponer de un mayor número de ejercicios resueltos.
- El uso de software permite visualizar ciertos conceptos geométricos de una forma limpia, cómoda e interactiva.

- Mejora del aprendizaje autónomo de los estudiantes al disponer de una herramienta que les permita comprobar dinámicamente las competencias y conocimientos adquiridos a lo largo del desarrollo de la asignaturas.
- La realización de prácticas de manera autónoma e independiente muestra al estudiante un dato real de su comprensión de la asignatura.
- Incremento del atractivo de las asignaturas arriba mencionadas, por una parte mediante un mejor conocimiento de los algoritmos utilizados en la resolución de problemas y por otra mediante el uso de herramientas informáticas en la resolución de los mismos.

2.2. Puntos débiles

- Poca disponibilidad de horas para desarrollar las clases prácticas.
- Dificultad a la hora de desarrollar las sesiones de manera dinámica e interactiva debido a las carencias de algunos estudiantes en el manejo de los programas informáticos.
- Reacciones adversas al hecho de tener que saber manejar un determinado programa de cálculo científico.
- Reacciones adversas por parte de algunos estudiantes al hecho de que la asistencia a las prácticas les supone una carga lectiva extra.
- La baja repercusión de estas actividades en la evaluación final hace que algunos estudiantes pierdan el interés.

Capítulo 3

Anexo: Materiales didácticos

En este anexo, se adjuntan los siguientes documentos:

1. Ejemplos del material docente elaborado con los contenidos necesarios para el seguimiento de los seminarios prácticos.
   - *a*) Grupos abelianos finitos
   - *b*) Ecuaciones diofánticas
2. Ejemplos de algoritmos de SINGULAR implementados en cuadernos de SAGE.
   - *a*) Aritmética en cuerpos y anillos de polinomios
   - *b*) Bases de Gröbner
3. Manual de instalación e introducción a SAGE.

1. Ejemplos del material docente elaborado con los contenidos necesarios para el seguimiento de los seminarios prácticos.

Práctica con SAGE

Grupos abelianos finitos

Un grupo G se dice que es cíclico si está generado por alguno de sus elementos. El teorema de clasificación de los grupos cíclicos afirma que, salvo isomorfismos, estos son $\mathbb{Z}$ (si tiene infinitos elementos) o $\mathbb{Z}/n$, siendo $n \in \mathbb{N}$ el orden de cualquiera de sus generadores. Todo grupo cíclico, así como el producto directo de grupos cíclicos, es abeliano. El Teorema de estructura de los grupos abelianos finitos afirma que todo grupo abeliano finito es isomorfo a un producto directo de grupos cíclicos.

El siguiente comando nos permite construir grupos cíclicos finitos

```
G.<a>=AbelianGroup([n])
```

El resultado es un grupo abeliano G isomorfo a $\mathbb{Z}/n$ y con generador a. Análogamente el comando

```
G.<a,b,c> = AbelianGroup([n,m,k])
```

crea un grupo abeliano, de nombre G, isomorfo a $\mathbb{Z}/n \times \mathbb{Z}/m \times \mathbb{Z}/k$[1], con generadores a, b, c, respectivamente. La misma construcción se puede realizar con cualquier número de grupos cíclicos. Todo elemento del grupo se puede escribir como producto de potencias de los generadores. La operación en el grupo se realiza con el asterisco.

Dado un grupo, el método `.list()` nos da una lista con todos los elementos del grupo.

Si queremos saber cuantos elementos tiene el grupo utilizamos `.order()`

Con `.random_element()` extraemos, de modo aleatorio, un elemento del grupo.

[1]Sage denota $\mathbb{Z}/n$ como Cn

SAGE

```
sage: G.<a,b> = AbelianGroup([2,3]); G
Multiplicative Abelian Group isomorphic to C2 x C3
sage: G.order()
6
sage: G.list()
[1, b, b^2, a, a*b, a*b^2]
sage: G.<a,b,c> = AbelianGroup([67,980,23]); G
Multiplicative Abelian Group isomorphic to C67 X C980 x C23
sage: h = G.random_element(); h a^10*b^449*c^22
sage: g = G.random_element(); g a^62*b^391*c^20
sage:h*g,g*h #Deben conmutar
(a^5*b^840*c^19, a^5*b^840*c^19)
sage: i = g^(-1); i
a^5*b^589*c^3
sage: i * g
1
```

Para crear un subgrupo H de un grupo G basta con dar una lista de generadores del subgrupo. Esto es, si damos una lista de elementos del grupo, el subgrupo generado es el mínimo subgrupo que contiene a todos los elementos. Para hacer esto en Sage utilizamos el método

```
.subgroup(lista)
```

donde `lista` (que debe ir entre corchetes aunque tenga un solo elemento) es una lista de generadores del subgrupo. Dado un elemento g del grupo G llamaremos grupo generado por g al subgrupo formado por sus potencias. Cuando G es finito, el subgrupo generado por g también lo es. Llamaremos orden de g al menor entero positivo n tal que $g^n = 1$. El número de elementos del subgrupo generado por g (es decir el orden del subgrupo) coincide con el orden del elemento g. Para saber el orden de un elemento usaremos

```
.order()
```

aplicándolo sobre un elemento del grupo.

SAGE

```
sage: G.<a,b,c,d> = AbelianGroup([45,78,23,678]); G
Multiplicative Abelian Group isomorphic
to C45 x C78 x C23 x C678
sage: g = G.random_element()
sage: h = G.random_element()
sage: g, h
(a^4*b^76*c^21*d^548, a^22*b^75*c^17*d^129)
sage: H = G.subgroup([a,b]); H
Multiplicative Abelian Group isomorphic
to C2 x C3 x C5 x C9 x C13, which is the subgroup of
Multiplicative Abelian Group isomorphic
to C45 x C78 x C23 x C678 generated by [a, b]
sage: Hg = G.subgroup([g])
sage: Hg.order(), g.order()
(1520415, 1520415)
```

Dado un número entero positivo n llamaremos *Indicador de Euler* de n y lo denotamos por $\phi(n)$ al número de números naturales menores que n y primos con n. Dicho valor coincide con el número de unidades (esto es de elementos con inverso) del anillo $\mathbb{Z}/n$. El Indicador de Euler es una función multiplicativa, es decir $\phi(nm) = \phi(n)\phi(m)$ si n y m son primos entre si. Por tanto, por el teorema fundamental de la aritmética, si se conoce $\phi(p^k)$ para cualquier primo p, podemos conocer $\phi(n)$ para cualquier entero positivo n.

En Sage el Indicador de Euler se calcula con

```
euler_phi()
```

SAGE

```
sage: euler_phi(7)  # Sobre los numeros primos es sencilla
6
sage: euler_phi(23)
22
sage: # Ahora sobre potencias de primos
sage: euler_phi(7^2)
42
sage: euler_phi(7^3)
294
sage: euler_phi(776556979878787888999999876)
384456325242775348203782 40
```

Práctica con SAGE

Ecuaciones diofánticas.

Dado un anillo euclídeo A y $a, b, c \in A$ se trata de calcular los $x, y \in A$ tales que $ax + by = c$. Se verifica que la ecuación $ax + by = c$ tiene solución si y sólo si c es un múltiplo del máximo común divisor de a y b.

Resolución de la ecuación: En el caso de que la ecuación $ax + by = c$ tenga solución, el algoritmo de Euclides nos permite obtener x'_0, y'_0 tales que $ax'_0 + by'_0 = d$, siendo d el el máximo común divisor de a y b. Por tanto $x_0 = \frac{c}{d}x'_0$, $y_0 = \frac{c}{d}y'_0$ es una solución particular de la ecuación. Otra solución cualquiera x, y verifica que $ax + by = c = ax_0 + by_0$, es decir, $a(x - x_0) = b(y_0 - y)$. Como $\frac{a}{d}$ y $\frac{b}{d}$ son primos entre sí se sigue que $\frac{a}{d}$ tiene que dividir a $y_0 - y$. Luego $\frac{a}{d}t = y_0 - y$ y $\frac{b}{d}t = x - x_0$. Así todas las soluciones de la ecuación $ax + by = c$ son

$$x = \frac{1}{d}(cx'_0 + bt), \quad y = \frac{1}{d}(cy'_0 - at) \quad \forall t \in A$$

Por tanto el algoritmo de Euclides nos permite por un lado comprobar si la ecuación tiene solución y por otro lado calcular una solución particular y por tanto todas.

Veamos cómo resolver ecuaciones diofánticas usando SAGE.

Si $a, b \in A$ tales que $a = bc + r$ entonces m.c.d(a,b) = m.c.d(b,r). Repitiendo recursivamente este argumento, para los valores de los divisores y restos, el Algoritmo de Euclides dice que el máximo común divisor de a y b será el último resto no nulo.

La función `a % b` devuelve el resto de la división euclídea de a y b

La función `gcd(a,b)` calcula el máximo común divisor de a y b mientras que la función `xgcd(a,b)` devuelve una lista de tres elementos, siendo el primero el máximo común divisor de a y b y los dos otros son los valores x e y, obtenidos por el algoritmo de Euclides, tales que $ax + by = m.c.d(a, b)$.

SAGE

```
var('t')
def diofantica(a,b,c):
    d=gcd(a,b)
    if c % d!=0:
        return "NO HAY SOLUCION"
    else:
x=(xgcd(a,b)[1]*c+b*t)/d
y=(xgcd(a,b)[2]*c-a*t)/d
print "x=", x, "," , "y=", y

diofantica(24,18,12)
x= 3*t + 2 , y= -4*t - 2
```

Si trabajamos con polinomios (en lugar de números enteros) debemos indicar el anillo en el que están definidos los coeficientes de los polinomios. Para hacer esto en SAGE debemos incluir al principio la instrucción

```
R.<x>=A[]
```

que define el anillo de polinomios `R` con coeficientes en el anillo `A` y en la variable `x`. Posibles anillos `A` definidos en SAGE son

- ZZ para $\mathbb{Z}$
- QQ para $\mathbb{Q}$
- Integers(n) para $\mathbb{Z}/n$
- RR para $\mathbb{R}$, representados por números reales de doble precisión.
- CC para $\mathbb{C}$

Aritmética en Cuerpos y Anillos de Polinomios

Aritmética en cuerpos

Para hacer aritmética en un cuerpo con Singular, siempre hay que definir un anillo de polinomios en al menos una variable con coeficientes en dicho cuerpo.

(1) Cuerpo de los números racionales

```
# Cuerpo de los números racionales dotado del orden lexicográfico
inverso según el grado

Q=singular.ring(0, 'x', 'dp');
```

```
Q;
```

```
//   characteristic : 0
//   number of vars : 1
//        block   1 : ordering dp
//                  : names    x
//        block   2 : ordering C
```

```
q=singular.number('12345/6789');
```

```
# Los factores comunes se simplifican automáticamente.
q;
```

```
4115/2263
```

```
q^5;
```

```
1179910858126071875/59350279669807543
```

(2) Cuerpos finitos

```
# Cuerpo finito Z/32003

A1=singular.ring(32003, 'x', 'dp');
```

```
p=singular.number(123456789);
```

```
p;
```

```
-10785
```

```
p^5;
```

```
8705
```

```
# Cuerpo finito de Galois GF(8), a es un generador del grupo
GF(8)-{0}

A2=singular.ring('(2^3,a)','x','dp');
```

```
A2;
```

```
//   # ground field : 8
//   primitive element : a
//   minpoly           : 1*a^3+1*a^1+1*a^0
//   number of vars : 1
//        block   1 : ordering dp
//                  : names    x
//        block   2 : ordering C
```

```
p=singular.number('a+a2');
```

```
p;
```

```
   a^4
```

```
p^5;
```

```
   a^6
```

```
#Polinomio mínimo de GF(8)

singular.eval('minpoly');
```

```
   '1*a^3+1*a^1+1*a^0'
```

```
# Cuerpo infinito Z/2(a) de característica 2

A3=singular.ring('(2,a)','x','dp');
```

```
#Definimos un polinomio mínimo

singular.eval('minpoly=a20+a3+1');
```

```
   'minpoly=a20+a3+1;'
```

```
n=singular.number('a+a2');
```

```
n^5;
```

```
   (a^10+a^9+a^6+a^5)
```

(2.1) Irreducibilidad

```
A4=singular.ring(2,'a','dp');
```

```
# Un polinomio reducible

singular.eval('factorize(a20+a2+1,1)');
```

```
   '_[1]=a^7+a^5+a^4+a^3+1\n_[2]=a^3+a+1'
```

```
# Podemos obtener la multiplicidad de las raices con:

singular.eval('factorize(a20+a2+1)');
```

```
   '[1]:\n   _[1]=1\n   _[2]=a^7+a^5+a^4+a^3+1\n   _[3]=a^3+a+1\n[2
```

```
1,2,2'
```

```
# Un polinomio irreducible

singular.eval('factorize(a20+a3+1,1)');
```

```
'_[1]=a^20+a^3+1'
```

(3) Cuerpo de los números reales

```
# Cuerpo de los números reales en precisión flotante de 30
dígitos de precisión

R=singular.ring('(real,30)','x','dp');
```

```
R;
```

```
//   characteristic : 0 (real:30 digits, additional 30 digits)
//   number of vars : 1
//        block   1 : ordering dp
//                  : names    x
//        block   2 : ordering C
```

```
r=singular.number('123456789.0');
```

```
r^5;
```

```
0.28679718602997181072337614380 9e+41
```

(4) Cuerpo de los números complejos

```
# Cuerpo de los números complejos en precisión flotante de 30
dígitos de precisión

C=singular.ring('(complex,30,I)','x','dp');
```

```
z=singular.number('123456789.0+0.0001*I');
```

```
z^5;
```

```
(0.28679718602997181072337426213 3e+41+I*11615286139912962207504
610)
```

(5) Cuerpos de fracciones

```
# Cuerpo Q(a,b,c) de fracciones de Q[a,b,c]

Q1=singular.ring('(0,a,b,c)','x','dp');
```

```
singular.eval('number q=12345a+12345/(78bc)');
```

```
'number q=12345a+12345/(78bc);'
```

```
q^2;
```

```
    Traceback (click to the left of this block for traceback)
    ...
       ? error occurred in or before STDIN line 74: `def sage41=sage
    sage40;`
```

```
singular.eval('q/9c');
```

```
    '(320970*a*b*c+4115)/(234*b*c^2)'
```

Aritmética en anillos de polinomios

Tal como hemos comentado antes, ya hemos definido anillos de polinomios en una variable con coeficientes en un cuerpo; pues para hacer aritmética en un cuerpo K, Singular entiende este como un subanillo del anillo K[x] de polinomios en una variable con coeficientes en dicho cuerpo; es decir, $K \subset K[x]$. Así pues, ahora simplemente generalizamos la construcción anterior a varias variables.

(1) Anillo de polinomios en tres variables con coeficientes en el cuerpo de los racionales

```
A=singular.ring(0,'(x,y,z)', 'dp');
```

```
A;
```

```
    //   characteristic : 0
    //   number of vars : 3
    //        block   1 : ordering dp
    //                  : names    x y z
    //        block   2 : ordering C
```

```
f=singular('x^3+y^2+z^2');
```

```
f*f-f;
```

```
    x^6+2*x^3*y^2+2*x^3*z^2+y^4+2*y^2*z^2+z^4-x^3-y^2-z^2
```

(2) Anillo de polinomios en tres variables con coeficientes en el cuerpo de fracciones del anillo de polinomios con coeficientes racionales en tres variables

```
B=singular.ring('(0,a,b,c)','(x,y,z)','dp');
```

```
B;
```

```
    //   characteristic : 0
    //   3 parameter    : a b c
    //   minpoly        : 0
    //   number of vars : 3
    //        block   1 : ordering dp
    //                  : names    x y z
    //        block   2 : ordering C
```

```
f=singular('x^3+y^2+z^2');
```

```
f^3-f^2;
```

```
x^9+3*x^6*y^2+3*x^6*z^2+3*x^3*y^4+6*x^3*y^2*z^2+3*x^3*z^4-x^6+y^
y^4*z^2+3*y^2*z^4+z^6-2*x^3*y^2-2*x^3*z^2-y^4-2*y^2*z^2-z^4
```

Bases de Gröbner

Bases de Gröbner

Sage llama internamente a Singular para calcular las bases de Gröbner, así que podemos utilizar directamente las instrucciones de Sage para estos cálculos.

Podemos utilizar los distintos tipos de ordenes en los anillos de polinomios: lexicográfico, lexicográfico negativo, lexicográfico según el grado, etc.

```
# Anillo de polinomios en cuatro variables con coeficientes
racionales y orden lexicográfico
A = PolynomialRing(QQ, 4, 'abcd', order='lp');
```

```
A;
```

```
Multivariate Polynomial Ring in a, b, c, d over Rational Field
```

```
# Generadores del anillo A

A.gens();
```

```
(a, b, c, d)
```

```
# Introduzcamos un ideal del anillo A
I = (a+b+c+d, a*b+a*d+b*c+c*d, a*b*c+a*b*d+a*c*d+b*c*d, a*b*c*d-
1)*A;
```

```
I;
```

```
Ideal (a + b + c + d, a*b + a*d + b*c + c*d, a*b*c + a*b*d + a*c
b*c*d, a*b*c*d - 1) of Multivariate Polynomial Ring in a, b, c,
over Rational Field
```

```
# Cálculo de la base de Gröbner del ideal I
B = I.groebner_basis();
```

```
B;
```

```
[a + b + c + d, b^2 + 2*b*d + d^2, b*c - b*d + c^2*d^4 + c*d -
2*d^2, b*d^4 - b + d^5 - d, c^3*d^2 + c^2*d^3 - c - d, c^2*d^6 -
c^2*d^2 - d^4 + 1]
```

```
# Anillo de polinomios en cuatro variables con coeficientes
racionales y orden lexicográfico inverso según el grado
A = PolynomialRing(QQ, 4, 'abcd', order='dp');
```

```
A;
```

```
Multivariate Polynomial Ring in a, b, c, d over Rational Field
```

```
# Generadores del anillo A
```

```
A.gens();
```

```
(a, b, c, d)
```

```
# Introduzcamos un ideal del anillo A
I = (a+b+c+d, a*b+a*d+b*c+c*d, a*b*c+a*b*d+a*c*d+b*c*d, a*b*c*d-
1)*A;
```

```
I;
```

```
Ideal (a + b + c + d, a*b + b*c + a*d + c*d, a*b*c + a*b*d + a*c
b*c*d, a*b*c*d - 1) of Multivariate Polynomial Ring in a, b, c,
over Rational Field
```

```
# Cálculo de la base de Gröbner del ideal I
B = I.groebner_basis();
```

```
B;
```

```
[c^2*d^4 + b*c - b*d + c*d - 2*d^2, c^3*d^2 + c^2*d^3 - c - d, b
+ d^5 - b - d, b*c*d^2 + c^2*d^2 - b*d^3 + c*d^3 - d^4 - 1, b*c^
c^2*d - b*d^2 - d^3, b^2 + 2*b*d + d^2, a + b + c + d]
```

```
# Anillo de polinomios en cuatro variables con coeficientes
racionales y orden lexicográfico según el grado
A = PolynomialRing(QQ, 4, 'abcd', order='Dp');
```

```
A;
```

```
Multivariate Polynomial Ring in a, b, c, d over Rational Field
```

```
# Generadores del anillo A

A.gens();
```

```
(a, b, c, d)
```

```
# Introduzcamos un ideal del anillo A
I = (a+b+c+d, a*b+a*d+b*c+c*d, a*b*c+a*b*d+a*c*d+b*c*d, a*b*c*d-
1)*A;
```

```
I;
```

```
Ideal (a + b + c + d, a*b + b*c + a*d + c*d, a*b*c + a*b*d + a*c
b*c*d, -1 + a*b*c*d) of Multivariate Polynomial Ring in a, b, c,
over Rational Field
```

```
# Cálculo de la base de Gröbner del ideal I
B = I.groebner_basis();
```

```
B;
```

```
[c^2*d^4 + b*c - b*d + c*d - 2*d^2, c^3*d^2 + c^2*d^3 - c - d, b
+ d^5 - b - d, b*c*d^2 + c^2*d^2 - b*d^3 + c*d^3 - d^4 - 1, b*c^
```

```
c^2*d - b*d^2 - d^3, b^2 + 2*b*d + d^2, a + b + c + d]
```

```
# Anillo de polinomios en cuatro variables con coeficientes
racionales y orden lexicográfico negativo. Este es un orden local
y por tanto produce bases estándar pero no bases de Gröbner.
A = PolynomialRing(QQ, 4, 'abcd', order='ls');
```

```
A;
```

```
Multivariate Polynomial Ring in a, b, c, d over Rational Field
```

```
# Generadores del anillo A

A.gens();
```

```
(a, b, c, d)
```

```
# Introduzcamos un ideal del anillo A
I = (a+b+c+d, a*b+a*d+b*c+c*d, a*b*c+a*b*d+a*c*d+b*c*d, a*b*c*d-
1)*A;
```

```
I;
```

```
Ideal (a + b + c + d, a*b + b*c + a*d + c*d, a*b*c + a*b*d + a*c
b*c*d, -1 + a*b*c*d) of Multivariate Polynomial Ring in a, b, c,
over Rational Field
```

```
# Cálculo de la base de Gröbner del ideal I
B = I.groebner_basis();
```

```
B;
```

```
[1]
```

```
# Anillo de polinomios en cuatro variables con coeficientes
racionales y orden lexicográfico negativo inverso según el grado.
Este es un orden local y por tanto produce bases estándar pero no
bases de Gröbner.
A = PolynomialRing(QQ, 4, 'abcd', order='ds');
```

```
A;
```

```
Multivariate Polynomial Ring in a, b, c, d over Rational Field
```

```
# Generadores del anillo A

A.gens();
```

```
(a, b, c, d)
```

```
# Introduzcamos un ideal del anillo A
I = (a+b+c+d, a*b+a*d+b*c+c*d, a*b*c+a*b*d+a*c*d+b*c*d, a*b*c*d-
1)*A;
```

```
I;
```

Ideal (a + b + c + d, a*b + b*c + a*d + c*d, a*b*c + a*b*d + a*c
b*c*d, -1 + a*b*c*d) of Multivariate Polynomial Ring in a, b, c,
over Rational Field

```
# Cálculo de la base de Gröbner del ideal I
B = I.groebner_basis();
```

```
B;
```

[1]

```
# Anillo de polinomios en cuatro variables con coeficientes
racionales y orden lexicográfico negativo según el grado. Este es
un orden local y por tanto produce bases estándar pero no bases
de Gröbner.
A = PolynomialRing(QQ, 4, 'abcd', order='Ds');
```

```
A;
```

Multivariate Polynomial Ring in a, b, c, d over Rational Field

```
# Generadores del anillo A

A.gens();
```

(a, b, c, d)

```
# Introduzcamos un ideal del anillo A
I = (a+b+c+d, a*b+a*d+b*c+c*d, a*b*c+a*b*d+a*c*d+b*c*d, a*b*c*d-
1)*A;
```

```
# Cálculo de la base de Gröbner del ideal I
B = I.groebner_basis();
```

```
B;
```

[1]

```

```

Introducción a Sage

Carlos Tejero Prieto
Departamento de Matemáticas
Universidad de Salamanca

¿Qué es Sage?

- Es una distribución de software libre para Matemáticas que consta de casi 100 paquetes de software bajo una interfaz común para todos ellos desarollada en el lenguaje de programación orientada a objetos Python.
- Gran cantidad de áreas de las Matemáticas: Álgebra básica, Análisis, Teoría de Números, Criptografía, Cálculo Numérico, Álgebra Conmutativa, Teoría de Grupos, Combinatoria, Teoría de Grafos, Álgebra Lineal, etc.

Instalación

Sage está disponible en forma de archivos binarios pre-compilados para los principales sistemas operativos: Windows, Linux, Mac OS X, etc.

Para instalar Sage descargamos los archivos binarios para nuestro sistema operativo desde el repositorio español de Sage en el servidor de REDIRIS:
http://sunsite.rediris.es/mirror/sagemath/index.html

Microsoft Windows: Para poder usar Sage en Windows es necesario instalar previamente VirtualBox para Windows
http://www.virtualbox.org/wiki/

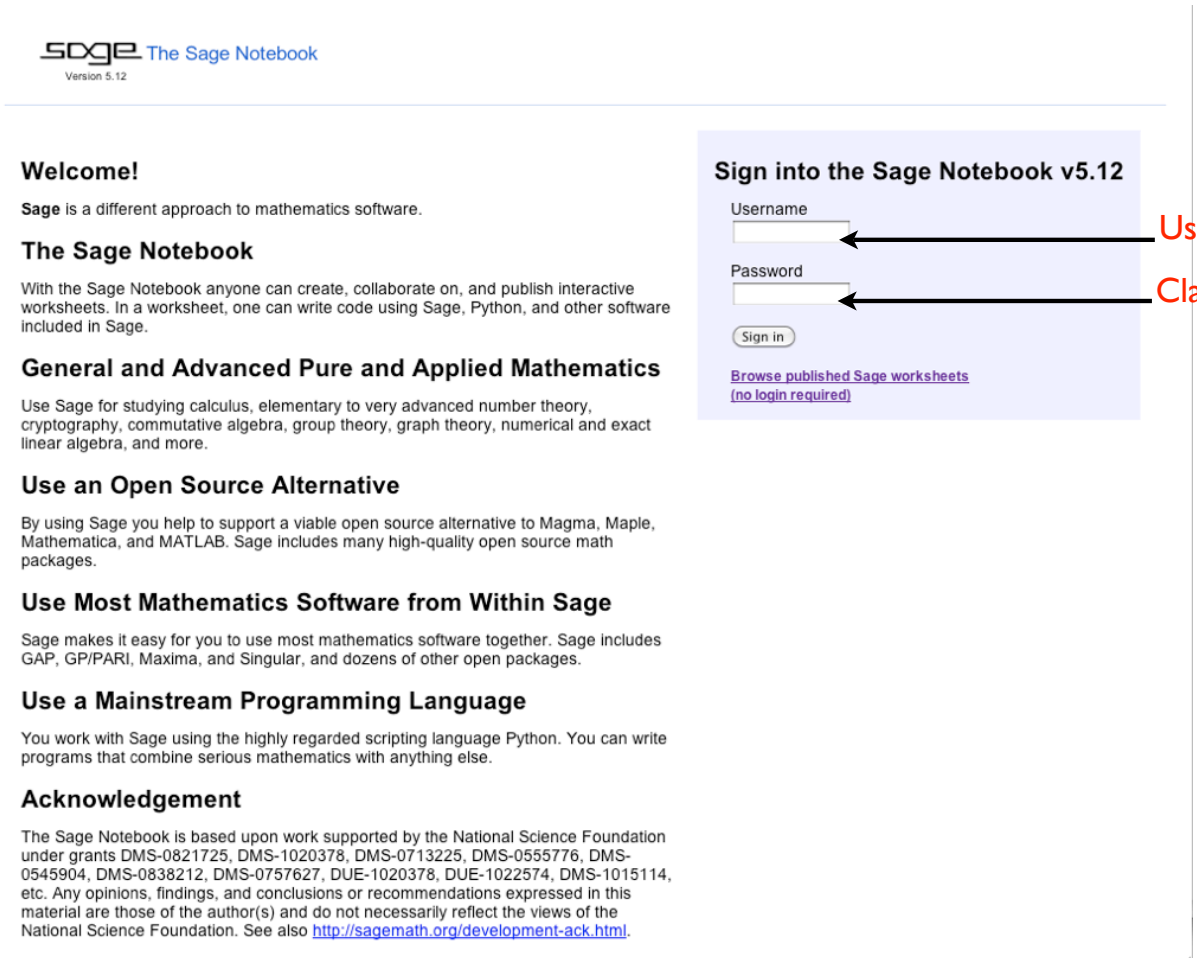
y después descargarse e instalar el binario de la distribución VirtualBox de Sage.

Linux & Mac OS X: Basta descargarse e instalar los archivos binarios.

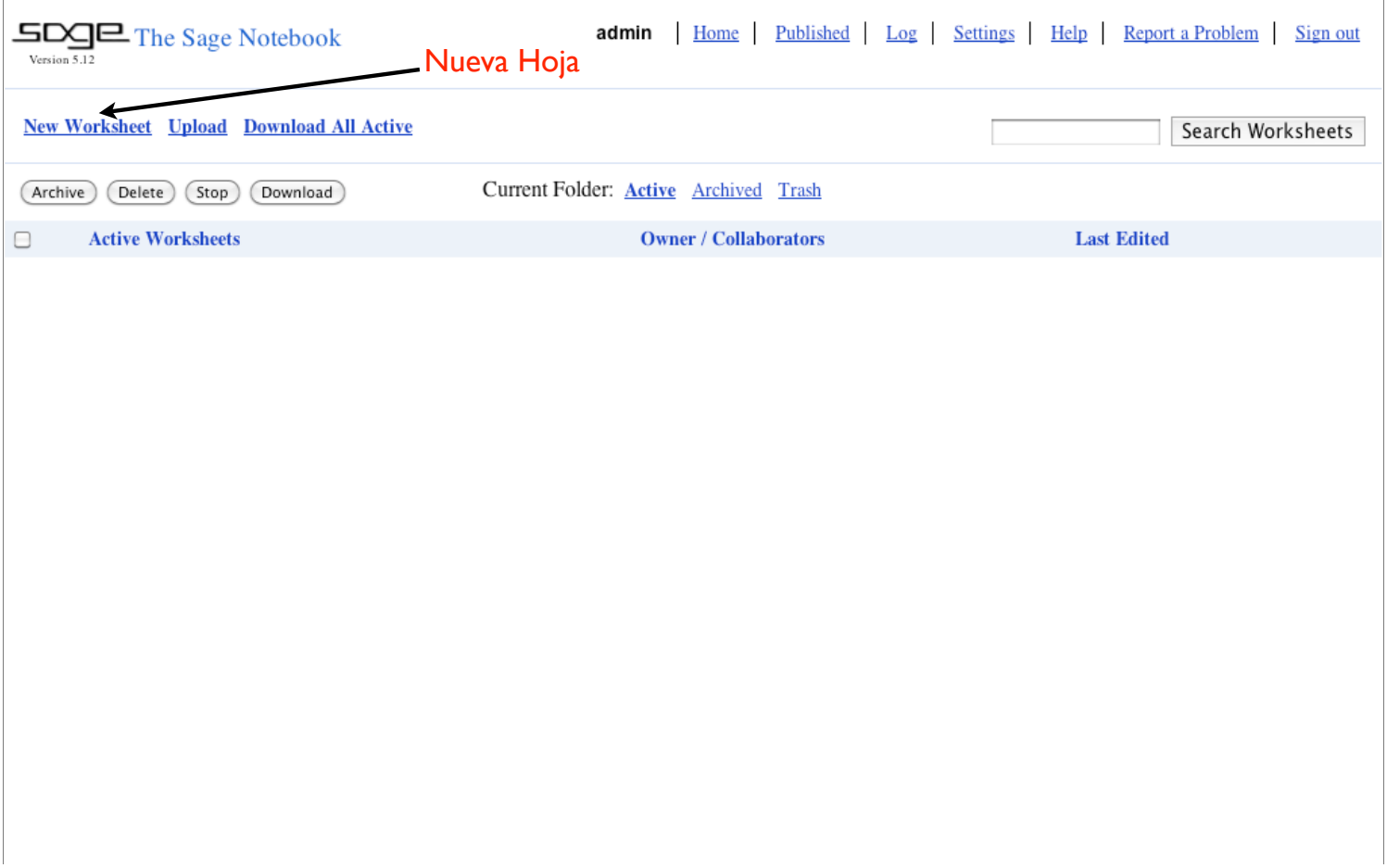
La interfaz del usuario

- Se puede acceder a Sage en línea de comandos si sólo necesitamos una presentación básica.
- Para presentaciones más avanzadas Sage usa un cuaderno como interfaz de usuario al cual se accede desde un navegador.
- Dentro del cuaderno de Sage se pueden incluir gráficos, expresiones matemáticas de alta calidad tipográfica y podemos compartir nuestro trabajo en la red.

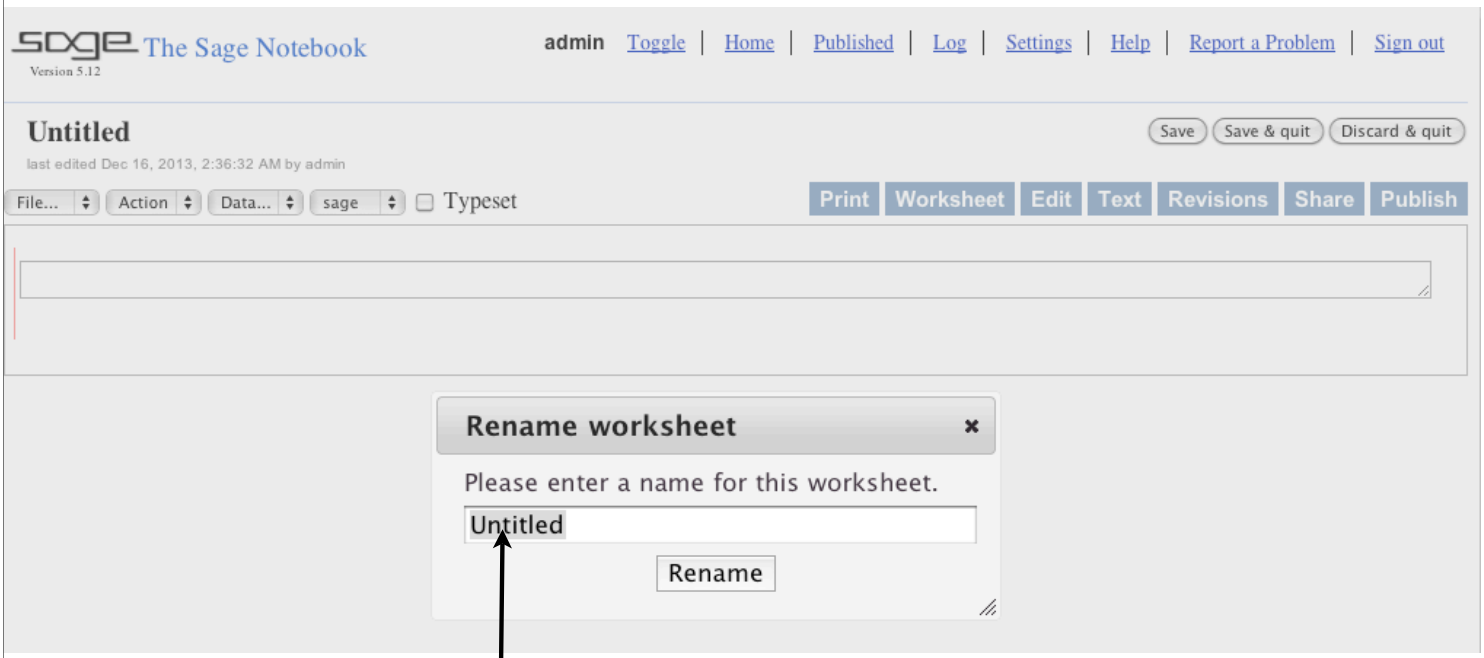
Página de acceso al cuaderno de SAGE



El gestor de hojas de trabajo



Crear una hoja de trabajo nueva en Sage



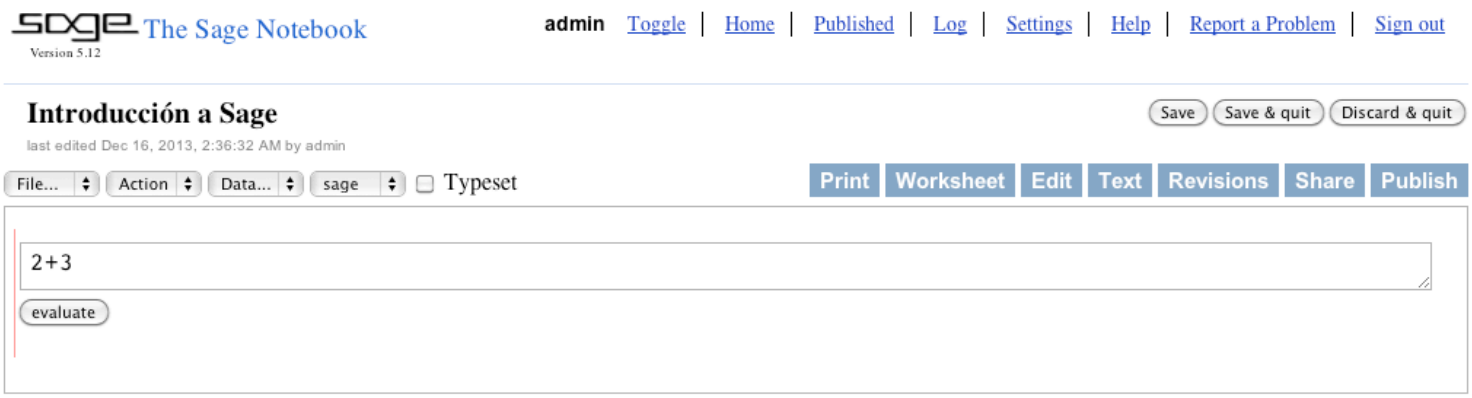
Sustituir por el nombre que queremos para la nueva hoja de trabajo

Las celdas de una hoja de trabajo

Introducir código en una celda

Introduce el siguiente texto, pero no pulses la tecla intro:
2+3

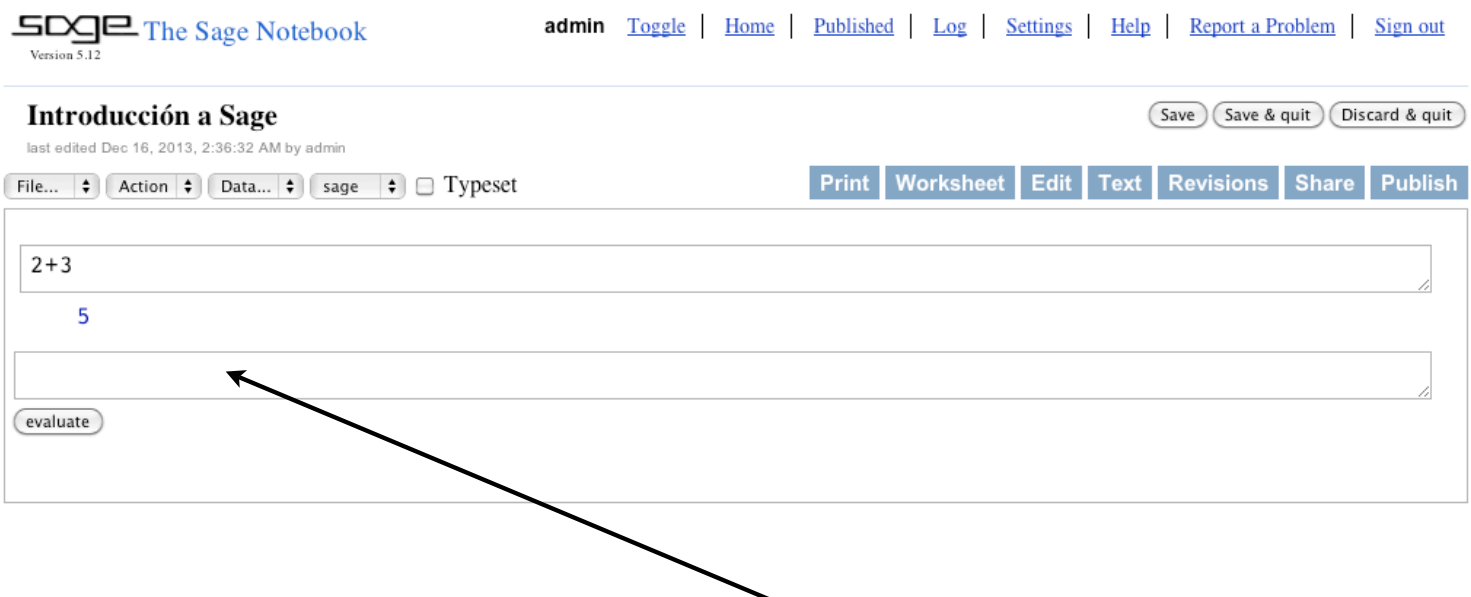
Ahora la hoja de trabajo tiene el siguiente aspecto:



Para evaluar el resultado tenemos dos posibilidades. Podemos presionar Mayúsculas+Intro, o bien podemos hacer uso del botón evaluate que aparece debajo de la celda. Sin embargo, si sólo pulsamos la tecla Intro, entonces simplemente se genera una nueva línea dentro de la celda y el resultado no se evalúa.

La hoja después de evaluar una celda

Justo después de proceder a evaluar la celda, veremos que aparece debajo de ella en el lado izquierdo una barra vertical de color verde | que desaparece una vez que el ordenador termina de hacer los cálculos y nos devuelve el resultado. El aspecto final de la hoja de trabajo es el siguiente:



Fijémonos que debajo del resultado ha aparecido una nueva celda en la que podemos seguir introduciendo código.

Operadores

Tecleemos en la nueva celda el siguiente texto: `5 + 6*21/18 - 2^3` y evaluemos la celda.

En las expresiones anteriores, 2, 3, 5, 6, 21 y 18 son objetos de tipo **entero**, los caracteres +, –, *, /, ^ se denominan operadores y su propósito es decirle a Sage qué operaciones tiene que ejecutar sobre los objetos de una expresión. El tipo de un objeto se obtiene con la función **type( )**

Operadores: Suma +, resta –, multiplicación *, división /, exponenciación ^, resto de una división %

Orden de precedencia de operadores

Los operadores anteriores se comportan en Sage del modo habitual. La siguiente tabla muestra dichos operadores en orden decreciente de precedencia y sus reglas.

Operadores	Reglas
^	Se evalúan de derecha a izquierda
*, /, %	Se evalúan de izquierda a derecha
+, -	Se evalúan de izquierda a derecha

Introducción a Sage

```
2+3
```

```
5
```

```
5 + 6*21/18 - 2^3
```

```
4
```

```

```

Orden de precedencia de operadores en orden decreciente:

( ) Los paréntesis se evalúan desde dentro hacia afuera

^ Se evalúan de derecha a izquierda.

*,%,/ Se evalúan de izquierda a derecha

+, – Se evalúan de izquierda a derecha.

```
((2+4)*3*5)
```

```
90
```

Variables

Una variable es un nombre que puede asociarse con una dirección de memoria en la que almacenar información.

En Sage la forma de crear variables es mediante la asignación de contenido.

Ejemplo: creemos una variable *b* con valor *7*

```
b=7
```

Nótese que al evaluar la celda no hemos obtenido nada en pantalla.

Si queremos ver el contenido de la variable *b* hay que escribir su

nombre en una nueva celda y evaluarla.

```
b
```

```
7
```

Podemos tener más de una variable.

```

```

```
a = 2
b = 3
```

Podemos operar simbólicamente con las variables.

```
a+b
```
```
5
```

El tipo de una variable es el del objeto al cual está asignada:

```
type(a)
```
```
<type 'sage.rings.integer.Integer'>
```

Operaciones con enteros

```
125/7
```
```
125/7
```

Si la división no es exacta tenemos un número racional.

```
type(125/7)
```
```
<type 'sage.rings.rational.Rational'>
```

Si la fracción es reducible, Sage la reduce.

```
190/24
```
```
95/12
```

Podemos escribir dos o más operaciones separadas por punto y coma.

```
2^34; 2**34
```
```
17179869184
17179869184
```

Tambien se pueden separar por comas, y el resultado aparece en forma de n-upla, esto es, entre paréntesis.

```
2^34, 2**34
```
```
(17179869184, 17179869184)
```

Las potencias de exponente negativo producen fracciones.

```
2^(-4)
```
```
1/16
```

Los parentesis y la jerarquia de operaciones funcionan como se espera.

```
(2 + 6)^2 - 30*2 + 9
```
```
13
```

Operaciones con números racionales y reales

```
```

```
4/7 + 8/3
```
```
68/21
```
```
3/5 - 6/11
```
```
3/55
```
```
(3/4 - 4/5)^3 + (2/3) / (34/89)
```
```
711949/408000
```

Un número real se representa como un decimal con precisión fijada.

```
type(1.5)
```
```
<type 'sage.rings.real_mpfr.RealLiteral'>
```
```
1.563 + 3.89
```
```
5.45300000000000
```

2*10^5 se escribe como número real en notación científica 2e5

```
1.456e25 / 2.456e-12
```
```
5.92833876221498e36
```
```
(1.123782)^100
```
```
117005.737177117
```

Si aparecen numeros racionales y decimales en una operación, Sage da la solucion en numeros decimales, i.e reales

```
4/5 + 0.87
```
```
1.67000000000000
```

La raíz cuadrada se obtiene con la función **sqrt( )**

```
sqrt(4), sqrt(8), sqrt(32)
```
```
(2, 2*sqrt(2), 4*sqrt(2))
```

Para raíces de orden superior se utiliza la notación de potencias fraccionarias:

```
#La raíz cuarta
1024^(1/4)
```
```
4*4^(1/4)
```
```
#La raíz septima
1024^(1/7)
```
```
2*8^(1/7)
```

En Matemáticas existen distintos conjuntos de numéros: enteros $\mathbb{Z}$, racionales $\mathbb{Q}$, reales $\mathbb{R}$, complejos $\mathbb{C}$, etc. En Sage hay algo similar, aunque no totalmente igual: en $\mathbb{Z}$ y en $\mathbb{Q}$ el programa utiliza aritmética exacta, sin embargo en $\mathbb{R}$ y en $\mathbb{C}$ utiliza aritmética aproximada, por lo que pueden producirse errores de redondeo. El conjunto de los enteros se denota en el programa por ZZ. Los racionales, los reales y los complejos se denotan QQ, RR y CC. Para saber en que conjunto considera Sage que esá un número utilizamos el método

.base_ring()

aunque también se puede utilizar la función

type()

La diferencia entre étodo y función es su forma de utilización. Para usar una función escribimos el nombre de la función y entre paréntesis el objeto al que se le aplica la función. Sin embargo para utilizar un método, primero escribimos el objeto sobre el que va a actuar, después escribimos un punto y finalmente el método.

Para obtener aproximaciones decimales utilizamos la función (y a la vez método)

n()

Esta función admite como opción **digits** para obtener las aproximaciones con distinto número de cifras.

```
a=2; type(a); a.base_ring()
```

```
    <type 'sage.rings.integer.Integer'>
    Integer Ring
```

```
b=2/3; type(b); b.base_ring()
```

```
    <type 'sage.rings.rational.Rational'>
    Rational Field
```

```
c=1.5; type(c); c.base_ring()
```

```
    <type 'sage.rings.real_mpfr.RealLiteral'>
    Real Field with 53 bits of precision
```

El comando **in** se utiliza como el símbolo pertenece ∈.

```
a in ZZ, a in QQ, a in RR, a in CC; b in ZZ, b in QQ, b in RR, b
in CC; c in ZZ, c in QQ, c in RR, c in CC
```

```
    (True, True, True, True)
    (False, True, True, True)
    (False, True, True, True)
```

Precisión por defecto

```
n(sqrt(2))
```

```
    1.41421356237310
```

Elegimos los dígitos de precisión

```
n(sqrt(2),digits=50)
```

```
    1.4142135623730950488016887242096980785696718753769
```

Actuando **n()** como método

```
sqrt(2).n(); sqrt(2).n(digits=50)
```

```
    1.41421356237310
    1.4142135623730950488016887242096980785696718753769
```

Para extraer el numerador y el denominador de un número racional usamos las funciones, que también son métodos,

numerator(), denominator()

respectivamente.

```
numerator(3/4); denominator(3/4)
```

```
3
4
```

Redondeos a enteros

Por defecto **floor()**

Por exceso **ceil()**

Redondeo **round()** podemos especificar el número de cifras decimales que usamos**.** Mediante números negativos podemos especificar el redondeo a la decena, centena, etc. mediante -1, -2, etc.

Son tanto funciones como métodos, aunque el método **.round()** no admite la opción de especificar el número de cifras decimales.

```
floor(3.4), round(3.4), ceil(3.4); floor(3.5), round(3.5),
ceil(3.5); floor(3.6), round(3.6), ceil(3.6);
```

```
(3, 3, 4)
(3, 4, 4)
(3, 4, 4)
```

```
round(sqrt(2)), round(sqrt(2),1), round(sqrt(2),2)
```

```
(1, 1.4, 1.41)
```

```
a=sqrt(2)
```

```
d = 3456.876543
round(d), round(d,2), round(d,5); round(d,-1), round(d,-2),
round(d,-4)
```

```
(3457, 3456.88, 3456.87654)
(3460.0, 3500.0, 0.0)
```

Edición de la hoja de trabajo

Para borrar una celda primero borramos todo su contenido y después usamos la tecla de borrado ←

Para añadir una nueva celda entre dos existentes, colocamos el cursor entre las dos hasta que vemos una línea de color azul, hacemos click sobre ella con el ratón y aparecerá una nueva celda.

Para insertar texto entre dos celdas hacemos como en el caso anterior pero ahora hacemos click sobre la línea azul a la vez que presionamos Maúsculas. Esto mostrará una celda de escritura con un editor de texto básico.

Para volver a editar una celda de texto basta hacer doble click sobre cualquier parte del texto.

La ayuda interactiva en Sage

Sage dispone de ayuda interactiva a la que se accede escribiendo el nombre de una función, sin los paréntesis, seguida de una interrogación.

Si queremos conocer el código que implementa dicha función entonces tecleamos la función seguida de dos interrogaciones.

```
n?
```

File: /sagenb/sage_install/sage-5.11-boxen-x86_64-Linux/local/lib/python2.7/site-packages/sage/misc/func

Type: <type 'function'>

Definition: n(x, prec=None, digits=None)

Docstring:

Returns a numerical approximation of an object `x` with at least `prec` bits (or decimal `digits`) of pr

Note

Both upper case `N` and lower case `n` are aliases for `numerical_approx()`, and all three may be

INPUT:

- `x` - an object that has a numerical_approx method, or can be coerced into a real or complex
- `prec (optional)` - an integer (bits of precision)
- `digits (optional)` - an integer (digits of precision)

If neither the `prec` or `digits` are specified, the default is 53 bits of precision. If both are specified

EXAMPLES:

```
sage: numerical_approx(pi, 10)
3.1
sage: numerical_approx(pi, digits=10)
3.141592654
sage: numerical_approx(pi^2 + e, digits=20)
12.587886229548403854
sage: n(pi^2 + e)
12.5878862295484
sage: N(pi^2 + e)
12.5878862295484
sage: n(pi^2 + e, digits=50)
12.587886229548403854194778471228813633070946500941
sage: a = CC(-5).n(prec=100)
sage: b = ComplexField(100)(-5)
sage: a == b
True
sage: type(a) == type(b)
True
sage: numerical_approx(9)
9.00000000000000
```

You can also usually use method notation.

```
sage: (pi^2 + e).n()
12.5878862295484
sage: (pi^2 + e).N()
12.5878862295484
sage: (pi^2 + e).numerical_approx()
12.5878862295484
```

Vectors and matrices may also have their entries approximated.

```
sage: v = vector(RDF, [1,2,3])
sage: v.n()
(1.00000000000000, 2.00000000000000, 3.00000000000000)

sage: v = vector(CDF, [1,2,3])
sage: v.n()
(1.00000000000000, 2.00000000000000, 3.00000000000000)
sage: _.parent()
Vector space of dimension 3 over Complex Field with 53 bits of precision
sage: v.n(prec=75)
(1.0000000000000000000, 2.0000000000000000000, 3.0000000000000000000
```

```
sage: u = vector(QQ, [1/2, 1/3, 1/4])
sage: n(u, prec=15)
(0.5000, 0.3333, 0.2500)
sage: n(u, digits=5)
(0.50000, 0.33333, 0.25000)

sage: v = vector(QQ, [1/2, 0, 0, 1/3, 0, 0, 0, 1/4], sparse=True)
sage: u = v.numerical_approx(digits=4)
sage: u.is_sparse()
True
sage: u
(0.5000, 0.0000, 0.0000, 0.3333, 0.0000, 0.0000, 0.0000, 0.2500)

sage: A = matrix(QQ, 2, 3, range(6))
sage: A.n()
[0.000000000000000  1.00000000000000  2.00000000000000]
[ 3.00000000000000  4.00000000000000  5.00000000000000]

sage: B = matrix(Integers(12), 3, 8, srange(24))
sage: N(B, digits=2)
[0.00  1.0  2.0  3.0  4.0  5.0  6.0  7.0]
[ 8.0  9.0  10.  11. 0.00  1.0  2.0  3.0]
[ 4.0  5.0  6.0  7.0  8.0  9.0  10.  11.]
```

Internally, numerical approximations of real numbers are stored in base-2. Therefore, numbers whic
different:

```
sage: x=N(pi, digits=3); x
3.14
sage: y=N(3.14, digits=3); y
3.14
sage: x==y
False
sage: x.str(base=2)
'11.001001000100'
sage: y.str(base=2)
'11.001000111101'
```

As an exceptional case, `digits=1` usually leads to 2 digits (one significant) in the decimal output

```
sage: N(pi, digits=1)
3.2
sage: N(pi, digits=2)
3.1
sage: N(100*pi, digits=1)
320.
sage: N(100*pi, digits=2)
310.
```

In the following example, `pi` and `3` are both approximated to two bits of precision and then subtrac

```
sage: N(pi, prec=2)
3.0
sage: N(3, prec=2)
3.0
sage: N(pi - 3, prec=2)
0.00
```

TESTS:

```
sage: numerical_approx(I)
1.00000000000000*I
sage: x = QQ['x'].gen()
sage: F.<k> = NumberField(x^2+2, embedding=sqrt(CC(2))*CC.0)
sage: numerical_approx(k)
1.414213562373095*I

sage: type(numerical_approx(CC(1/2)))
<type 'sage.rings.complex_number.ComplexNumber'>
```

The following tests trac ticket #10761, in which `n()` would break when called on complex-valued a

```
sage: E = matrix(3, [3,1,6,5,2,9,7,3,13]).eigenvalues(); E
[18.168153650888222?, -0.084076825444106502? - 0.21902614848029062?*I, -0.0
sage: E[1].parent()
Algebraic Field
sage: [a.n() for a in E]
[18.168153650888222, -0.084076825444106502 - 0.21902614848029062*I, -0.0840768
```

Make sure we've rounded up log(10,2) enough to guarantee sufficient precision (trac #10164):

```
sage: ks = 4*10**5, 10**6
sage: check_str_length = lambda k: len(str(numerical_approx(1+10**-k,dig
sage: check_precision = lambda k: numerical_approx(1+10**-k,digits=k+1)-
sage: all(check_str_length(k) and check_precision(k) for k in ks)
True
```

Testing we have sufficient precision for the golden ratio (trac #12163), note that the decimal point a

sage: len(str(n(golden_ratio, digits=5000))) 5001 sage: len(str(n(golden_ratio, digits=5000000

```
n??
```

File: /Applications/Sage-5.0-OSX-32bit-10.5.app/Contents/Resources/sage/local/lib/python2.7/site-package

Source Code (starting at line 1193):

```
def numerical_approx(x, prec=None, digits=None):
    """
    Returns a numerical approximation of x with at least prec bits of
    precision.

    .. note::

       Both upper case N and lower case n are aliases for
       :func:`numerical_approx`.

    INPUT:


    -  ``x`` - an object that has a numerical_approx
       method, or can be coerced into a real or complex field

    -  ``prec (optional)`` - an integer (bits of
       precision)

    -  ``digits (optional)`` - an integer (digits of
       precision)


    If neither the prec or digits are specified, the default is 53 bits
    of precision.

    EXAMPLES::

        sage: numerical_approx(pi, 10)
        3.1
        sage: numerical_approx(pi, digits=10)
        3.141592654
        sage: numerical_approx(pi^2 + e, digits=20)
        12.587886229548403854
        sage: n(pi^2 + e)
        12.5878862295484
        sage: N(pi^2 + e)
        12.5878862295484
        sage: n(pi^2 + e, digits=50)
        12.587886229548403854194778471228813633070946500941
        sage: a = CC(-5).n(prec=100)
```

```
        sage: b = ComplexField(100)(-5)
        sage: a == b
        True
        sage: type(a) == type(b)
        True
        sage: numerical_approx(9)
        9.00000000000000


    You can also usually use method notation::

        sage: (pi^2 + e).n()
        12.5878862295484

    TESTS::

        sage: numerical_approx(I)
        1.00000000000000*I
        sage: x = QQ['x'].gen()
        sage: F.<k> = NumberField(x^2+2, embedding=sqrt(CC(2))*CC.0)
        sage: numerical_approx(k)
        1.41421356237309*I

        sage: type(numerical_approx(CC(1/2)))
        <type 'sage.rings.complex_number.ComplexNumber'>

    The following tests Trac 10761, in which n() would break when
    called on complex-valued AlgebraicNumbers::

        sage: E = matrix(3, [3,1,6,5,2,9,7,3,13]).eigenvalues(); E
        [18.16815365088822?, -0.08407682544410650? - 0.2190261484802906?*I, -
        sage: E[1].parent()
        Algebraic Field
        sage: [a.n() for a in E]
        [18.1681536508882, -0.0840768254441065 - 0.219026148480291*I, -0.084(
    """
    if prec is None:
        if digits is None:
            prec = 53
        else:
            prec = int((digits+1) * 3.32192) + 1
    try:
        return x._numerical_approx(prec)
    except AttributeError:
        from sage.rings.complex_double import is_ComplexDoubleElement
        from sage.rings.complex_number import is_ComplexNumber
        if not (is_ComplexNumber(x) or is_ComplexDoubleElement(x)):
            try:
                return sage.rings.real_mpfr.RealField(prec)(x)
            # Trac 10761: now catches ValueErrors as well as TypeErrors
            except (TypeError, ValueError):
                pass
        return sage.rings.complex_field.ComplexField(prec)(x)
```

También podemos conocer los métodos que le son aplicables a un determinado **objeto**. Para ello escribimos **objeto.**+Tabulador

Por ejemplo, si **a** es un número racional obtendremos

a.

a.abs a.inverse_mod a.period

a.additive_order a.is_integral a.py

a.base_extend a.is_nilpotent a.quo_rem

a.base_ring a.is_one a.rename

a.category a.is_square a.reset_name

a.ceil a.is_unit a.round

a.charpoly a.is_zero a.save

a.conjugate a.lcm a.sobj

a.copy a.list a.sqrt

a.db a.minpoly a.sqrt_approx

a.denom a.mod a.squarefree_part

a.denominator a.mod_ui a.str

a.divides a.multiplicative_order a.subs

a.dump a.n a.substitute

a.dumps a.norm a.support

a.factor a.nth_root a.trace

a.floor a.numer a.val_unit

a.gamma a.numerator a.valuation

a.gcd a.order a.version

a.height a.parent

Ahora los métodos que empiezan por d

a.d

a.db a.denominator a.dump

a.denom a.divides a.dumps

Pedimos ayuda sobre el metodo denominator

```
a.denominator?
```

File: /Applications/Sage-5.0-OSX-32bit-10.5.app/Contents/Resources/sage/devel/sage/sage/rings/rational.pyx

Type: <type 'builtin_function_or_method'>

Definition: a.denominator()

Docstring:

Returns the denominator of this rational number.

EXAMPLES:

```
sage: x = -5/11
sage: x.denominator()
11
sage: x = 9/3
sage: x.denominator()
1
```

Comparación de números

Los operadores para comparar números son

Menor <

Mayor >

Menor o igual <=

Mayor o igual >=

Igual ==

Distinto != (también vale <>)

```
5 < 6
```

```
True
```

```
-5 > 6
```

```
False
```

```
78 < 78, 78 <= 78
```

```
(False, True)
```

```
45 == 9 * 5
```

```
True
```

```
78 != 78
```

```
False
```

```
78<>78, 78 <> 123
```

```
(False, True)
```

Primos y factorización

Para saber si un número es primo debemos emplear el método

.is_prime()

Sage también puede decidir si un número es una potencia de un número primo

.is_prime_power()

Dado un número entero **n**, el primer número mayor que **n** y primo se obtiene con

.next_prime()

Análogamente el primo inmediatamente anterior a n se calcula con la función (no es un método)

previous_prime()

Las siguientes funciones, que como la anterior no son métodos, nos dan el primer número que es mayor o menor que **n,** respectivamente, y es una potencia de un primo

next_prime_power(), previous_prime_power()

```
7.is_prime()
```
```
True
```
```
49.is_prime()
```
```
False
```
```
49.is_prime_power()
```
```
True
```
```
49.next_prime()
```
```
53
```
```
previous_prime(49)
```
```
47
```
```
previous_prime_power(49)
```
```
47
```
```
next_prime_power(49)
```
```
53
```

Para obtener la factorización de un número como producto de potencias de primos de acuerdo con el Teorema Fundamental de la Aritmética, disponemos del método, que también es función

.factor()

```
factor(factorial(6))
```
```
2^4 * 3^2 * 5
```
```
factorial(100).factor()
```
```
2^97 * 3^48 * 5^24 * 7^16 * 11^9 * 13^7 * 17^5 * 19^5 * 23^4 * 2
* 31^3 * 37^2 * 41^2 * 43^2 * 47^2 * 53 * 59 * 61 * 67 * 71 * 73
79 * 83 * 89 * 97
```
```
factorial(100)
```
```
933262154439441526816992388562667004907159682643816214685929638
599993229915608941463976156518286253697920827223758251185210916
8000000000000000000000
```

Para obtener todos los divisores de un número tenemos el método, que también es función,

.divisors()

```
a=factorial(10)
```
```
a.divisors()
```
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 24,
27, 28, 30, 32, 35, 36, 40, 42, 45, 48, 50, 54, 56, 60, 63, 64,
72, 75, 80, 81, 84, 90, 96, 100, 105, 108, 112, 120, 126, 128, 1
```

```
140, 144, 150, 160, 162, 168, 175, 180, 189, 192, 200, 210, 216,
224, 225, 240, 252, 256, 270, 280, 288, 300, 315, 320, 324, 336,
350, 360, 378, 384, 400, 405, 420, 432, 448, 450, 480, 504, 525,
540, 560, 567, 576, 600, 630, 640, 648, 672, 675, 700, 720, 756,
768, 800, 810, 840, 864, 896, 900, 945, 960, 1008, 1050, 1080, 1
1134, 1152, 1200, 1260, 1280, 1296, 1344, 1350, 1400, 1440, 1512
1575, 1600, 1620, 1680, 1728, 1792, 1800, 1890, 1920, 2016, 2025
2100, 2160, 2240, 2268, 2304, 2400, 2520, 2592, 2688, 2700, 2800
2835, 2880, 3024, 3150, 3200, 3240, 3360, 3456, 3600, 3780, 3840
4032, 4050, 4200, 4320, 4480, 4536, 4725, 4800, 5040, 5184, 5376
5400, 5600, 5670, 5760, 6048, 6300, 6400, 6480, 6720, 6912, 7200
7560, 8064, 8100, 8400, 8640, 8960, 9072, 9450, 9600, 10080, 103
10800, 11200, 11340, 11520, 12096, 12600, 12960, 13440, 14175,
14400, 15120, 16128, 16200, 16800, 17280, 18144, 18900, 19200,
20160, 20736, 21600, 22400, 22680, 24192, 25200, 25920, 26880,
28350, 28800, 30240, 32400, 33600, 34560, 36288, 37800, 40320,
43200, 44800, 45360, 48384, 50400, 51840, 56700, 57600, 60480,
64800, 67200, 72576, 75600, 80640, 86400, 90720, 100800, 103680,
113400, 120960, 129600, 134400, 145152, 151200, 172800, 181440,
201600, 226800, 241920, 259200, 302400, 362880, 403200, 453600,
518400, 604800, 725760, 907200, 1209600, 1814400, 3628800]
```

```
180.divisors(); divisors(180)
```

```
[1, 2, 3, 4, 5, 6, 9, 10, 12, 15, 18, 20, 30, 36, 45, 60, 90, 18
[1, 2, 3, 4, 5, 6, 9, 10, 12, 15, 18, 20, 30, 36, 45, 60, 90, 18
```

División euclídea

Dados dos números

enteros arbitrarios **a, b** podemos encontrar dos números, únicos, **c** y **r** que verifican:

a = c b + r.

c se denomina cociente de la división de **a** entre **b** y **r** es un entero positivo o nulo ymenor que **|b|**, denominado resto de la división.

Para obtener el cociente de la división de **a** entre **b** se emplea el comando

//

Para obtener el resto se pueden emplear dos formas alternativas

.mod(), %

```
23 // 5, 23 % 5
```

```
(4, 3)
```

Comprobemos

```
4 * 5 + 3
```

```
23
```

También funciona con números negativos

```
-26 // 5, -26 % 5
```
```
(-6, 4)
```

Comprobamos

```
-6 * 5 + 4
```
```
-26
```

Lo mismo lo podemos realizar con el metodo **.mod()**

```
(-26).mod(5)
```
```
4
```

En n-ésimo número de Fermat es F_n=2^(2^n)+1. Fermat pensaba que todos los F_n eran primos. Sin embargo Euler demostró que F_5 no es primo.

```
(2^(2^1)+1).factor(),(2^(2^2)+1).factor(), (2^(2^3)+1).factor(),
(2^(2^4)+1).factor(), (2^(2^5)+1).factor()
```
```
(5, 17, 257, 65537, 641 * 6700417)
```

Máximo común divisor y Mínimo común multiplo

El máximo común divisor de dos números se calcula con el método, y también función,

.gcd()

El mínimo común múltiplo de dos números se calcula con el método, y también función,

.lcm()

El producto del mínimo común múltiplo por el máximo común múltiplo da el producto de los numeros

Dados dos enteros **a, b**, cuyo máximo común divisor es **d,** el Lema de Bezout nos dice que existen enteros **m. n**, no únicos, tales que

d= m a+ n b

Además, cualquier otra pareja de enteros que satisfaga la igualdad **d=m' a+ n' b=d** es de la forma

m' = m + t (b/d),

n' = n - t (a/d).

El método, y función, **.xgcd()** nos da una tres-upla **a.xgcd(b)=(d,m,n)** tal que **d= m a+ n b.**

```
(78).gcd(24), (78).lcm(24); 78*24, (78).gcd(24)*(78).lcm(24)
```
```
(6, 312)
(1872, 1872)
```
```
xgcd(78,24)
```
```
(6, 1, -3)
```

Comprobemos

```
1*78-3*24
```

```
6
```

La sucesión de Fibonacci está definida de modos que sus dos primeros terminos son 1 y los siguientes se obtienen sumando los dos anteriores. Es decir, es la sucesión (1,1,2,3, 5, 8, ...)

En Sage la sucesión de Fibonacci se obtiene con el método, y función,

.fibonacci()

```
fibonacci(10), fibonacci(20), fibonacci(30)
```

```
(55, 6765, 832040)
```

Números complejos

En Sage la unidad imaginaria se escribe tanto como **i**, como **I** (este es el formato que utiliza para presentarnos los resultados).

Dado el número complejo **z = a + bi** su parte real es a y su parte imaginaria es b. Para obtenerlas

utilizamos los métodos

.real(), .imag()

El módulo o valor absoluto es **(a^2 + b^2)^(1/2)** y se obtiene con

.abs()

El conjugado es **a – bi**

.conjugate()

La norma de un número complejo se obtiene multiplicando dicho número por su complejo

conjugado. Es siempre un número real no negativo, que coincide con **a^2 + b^2**

.norm()

Si imaginamos el número complejo **z = a + bi** como el vector del plano **(a, b),** dicho vector forma con

la parte positiva del eje de las **X** un cierto ángulo. Es el denominado argumento del número

complejo (Sage lo mide en radianes)

.arg(), argument()

```
(2+3*i)*(4+5*i)
```

```
22*I - 7
```

```
(2+3*i).real(), (2+3*i).imag(), abs(2+3*i), conjugate(2+3*i),
norm(2+3*i), arg(2+3*i)
```

```
(2, 3, sqrt(13), -3*I + 2, 13, arctan(3/2))
```

Operaciones con números complejos de precisión finita

Hasta ahora todas las operaciones que hemos hecho con números complejos han sido simbólicas.

Sin embargo hay ocasiones en las que tenemos que operar con números complejos de precisión finita; es decir, sus partes reales e imaginarias son números reales de precisión finita. Para indicarle a Sage que

trate a **i** como un número complejo de precisión finita tendremos que ejecutar la siguiente orden

i=CC(i)

```
i^2, type(2+3*i); i=CC(i); i^2, type(2+3*i)
```

```
(-1, <type 'sage.symbolic.expression.Expression'>)
(-1.00000000000000, <type
'sage.rings.complex_number.ComplexNumber'>)
```

Raíces m-ésimas de un número complejo

Una raíz m-éssima de un número complejo **z de** precisión finita se obtienen con

z.nth_root(m)

Si queremos todas las raíces m-ésimas entonces

z.nth_root(m,all=True)

```
a=4+9*i; a.nth_root(5); a.nth_root(5,all=True)
```

```
1.53827855658067 + 0.361012484681860*I
[1.53827855658067 + 0.361012484681860*I, 0.132010940045404 +
1.57454883806137*I, -1.45669130874579 + 0.612112214186722*I,
-1.03229567996687 - 1.19624268476502*I, 0.818697492086580 -
1.35143085216493*I]
```

```
a=(sqrt(2)-1)/(sqrt(2)+1)
```

Racionalización de expresiones

```
maxima_calculus('algebraic: true;') #Cargamos un módulo de Maxima
para poder simplificar
a.simplify_rational()
```

```
-2*sqrt(2) + 3
```

Resolución de ecuaciones elementales de modo exacto

En sage las dos partes de una ecuación están separadas por ==

El comando para resolver ecuaciones es

solve(ecuacion, x)

Este comando no tiene en cuenta la multiplicidad de las soluciones.

```
solve(x^2 - 5*x +6 == 0, x)
```

```
[x == 3, x == 2]
```

```
solve(x^2 - 5*x + 6, x)
```

```
[x == 3, x == 2]
```

```
solve(x^2 - 4*x + 4 == 0, x) # Ecuación con solución doble
```

```
[x == 2]
```

```
solve(x^3 - 3*x + 2 == 0, x) # Ecuación de tercer grado con
solución doble
```

```
[x == -2, x == 1]
```

```
solve(x^8 - 10*x^6 + 37*x^4 - 60*x^2 + 36, x) # Ecuación reducible a una de cuarto grado
```

```
    [x == -sqrt(2), x == sqrt(2), x == -sqrt(3), x == sqrt(3)]
```

```
solve(x^2 +4,x)  # Soluciones complejas
```

```
    [x == (-2*I), x == (2*I)]
```

```
solve(3*x^3 + 5*x^2 - 4*x +5,x) # Cúbica con soluciones complejas
```

```
    [x == -1/2*(1/54*sqrt(1423)*sqrt(3) - 2005/1458)^(1/3)*(I*sqrt(3
    1) - 1/162*(-61*I*sqrt(3) + 61)/(1/54*sqrt(1423)*sqrt(3) -
    2005/1458)^(1/3) - 5/9, x == -1/2*(1/54*sqrt(1423)*sqrt(3) -
    2005/1458)^(1/3)*(-I*sqrt(3) + 1) - 1/162*(61*I*sqrt(3) +
    61)/(1/54*sqrt(1423)*sqrt(3) - 2005/1458)^(1/3) - 5/9, x ==
    (1/54*sqrt(1423)*sqrt(3) - 2005/1458)^(1/3) +
    61/81/(1/54*sqrt(1423)*sqrt(3) - 2005/1458)^(1/3) - 5/9]
```

Declaración de incógnitas

Hasta ahora en todas las ecuaciones que hemos resuelto la incógnita o variable ha sido **x**. Si intentamos repetir esto con otra letra, el programa dará un error. Se pueden resolver ecuaciones con otro nombre de variable, pero para ello, antes de intentar resolverla debemos declararla mediante la instrucción

var('letra')

La letra debe ir necesariamente entre comillas simples. Se pueden declarar varias incognitas a la vez escribiendolas separadas por comas o por espacios.

```
var('a')
```

```
    a
```

```
solve(a^2 - 5*a + 6, a)
```

```
    [a == 3, a == 2]
```

```
var('b,c,d,e') # Ya podemos utilizar todas estas letras como incógnitas
```

```
    (b, c, d, e)
```

```
solve(d^2 - 5*d + 6, d)
```

```
    [d == 3, d == 2]
```

```

```

```
    [x == (-2*I), x == (2*I)]
```

Sistemas de ecuaciones

Un sistema de ecuaciones es simplemente una colección finita de ecuaciones. En Sage las distintas ecuaciones de un sistema se escriben como los elementos de una lista. Debemos escribir las ecuaciones entre corchetes separándolas por comas. Para resolver un sistema utilizamos

solve([ecuación1, ecuación2, ...], incognita1, incognita2, ...)

```
var('x,y')
```

```
    (x, y)
```

```
solve([x+y==2,x-y==0],x,y)
```

```
    [[x == 1, y == 1]]
```

```
solve(x^2 + y == 2, y) # Despejamos una incógnita
```

```
    [y == -x^2 + 2]
```

```
solve([x + y == 2, 2*x + 2*y == 4], x, y) # Un sistema
indeterminado, los parámetros son r1,...
```

```
    [[x == -r1 + 2, y == r1]]
```

```
solve([x^2 + y^2 == 2, 2*x + 2*y == 3], x, y) # Un sistema no
lineal
```

```
    [[x == -1/4*sqrt(7) + 3/4, y == 1/4*sqrt(7) + 3/4], [x ==
    1/4*sqrt(7) + 3/4, y == -1/4*sqrt(7) + 3/4]]
```

Para desarrollar (expandir) ambos miembros de una ecuación se emplea el método

.expand()

Si queremos quedarnos con el miembro izquierdo de una ecuación

.left()

y para obtener el derecho

.right()

Para sustituir la variable por un valor particular

.subs(x = valor)

```
f = (x-3)^2 + (3*x-5)/89 == (2*x - 5)*(x-2)
```

```
f
```

```
    (x - 3)^2 + 3/89*x - 5/89 == (2*x - 5)*(x - 2)
```

```
f.expand()
```

```
    x^2 - 531/89*x + 796/89 == 2*x^2 - 9*x + 10
```

```
f.left()
```

```
    (x - 3)^2 + 3/89*x - 5/89
```

```
f.right()
```

```
    (2*x - 5)*(x - 2)
```

```
f.subs(x=7) # Sustituimos x por el valor 7
```

```
    (1440/89) == 45
```

Resolución numérica de ecuaciones

El método para encontrar la solución numérica de una ecuación es

.find_root(intervalo)

```
f = (x^2 -5*x + 6 == 0)
```

```
f.find_root(-20,5) # Encontrar soluciones en el intervalo (-
20,5)
```

```
    2.999999999999999
```

```
f.find_root(0,2.5) # Encontrar soluciones en el intervalo
```

```
(0,2.5)
```
```
    2.0
```
```
g = (sin(x^2)+3 == exp(x)) # Una ecuación más complicada
```

```
g.find_root(-10,10)
```
```
    1.3737974476331927
```
```
g.subs(x=_) # La variable _ guarda el valor del último cálculo en Sage
```
```
    3.95032339468241 == 3.9503233946824143
```
```
h = x^6 - 5*x^4 - 3*x^2 + 5*x - 12 # Una ecuación de sexto grado
```

```
h.find_root(-10,20) # El problema de este método es fijar un intervalo adecuado
```
```
    2.3554150099757294
```

Para hacernos una idea de cómo elegir un intervalo adecuado podemos utilizar las capacidades gráficas de Sage.

Funciones

Para definir una función **f** de la variable **x** escribimos

f(x)= expresión algebraica de la función

Podemos definir una función a trozos mediante la instrucción **Piecewise** que agrupa en una lista las funciones componentes juntos con sus intervalos de definición

f(x)=Piecewise([[(x1,x2),f1], [(x2,x3),f2], ...)

```
f(x)=x^2-2*x+3
```

```
f(2) # Cáculo del valor de la función en x=2
```
```
    3
```
```
f = Piecewise([[(-5,-1),sin(x)],[(-1,2),x^2],[(2,4),x]]); f
```
```
    Piecewise defined function with 3 parts, [[(-5, -1), sin(x)], [(
    2), x^2], [(2, 4), x]]
```
```
f(-2), f(2/3), f(9/8)
```
```
    (-sin(2), 4/9, 81/64)
```

Gráficos bidimensionales

Para representar la gráfica de una función **f** de la variable **x** escribimos

plot(f, xmin,xmax, opciones)

Sólo es obligatorio el primer argumento. Si sólo escribimos la función entre paréntesis, el programa dibujará la gráfica en el intervalo [−1,1]. El eje **y** lo adaptará a la función. Normalmente la escala no será la misma en el eje **x** y en el eje **y**.

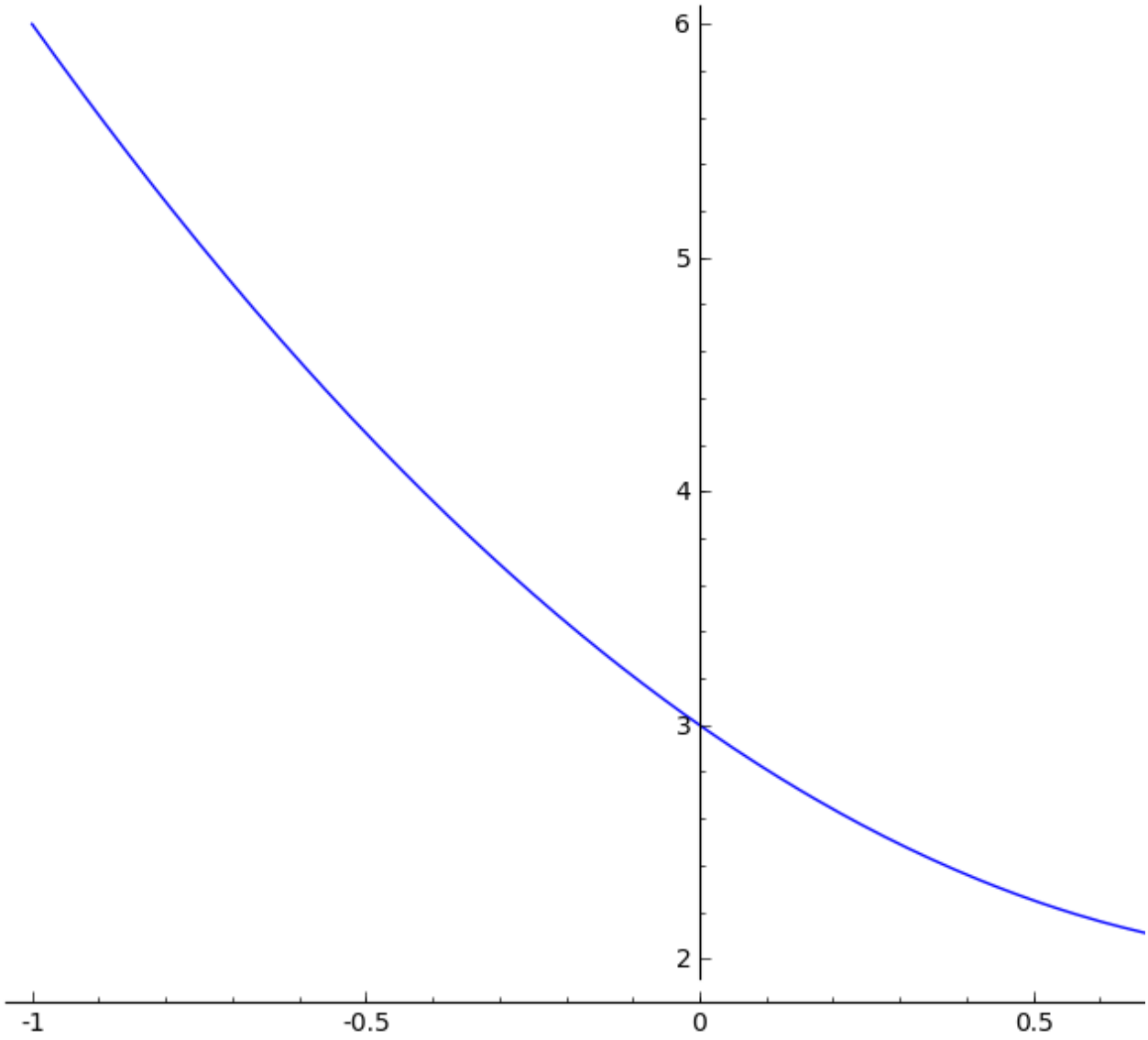
Para conocer las opciones de un **comando** en Sage escribimos **comando.options**

```
plot.options
```

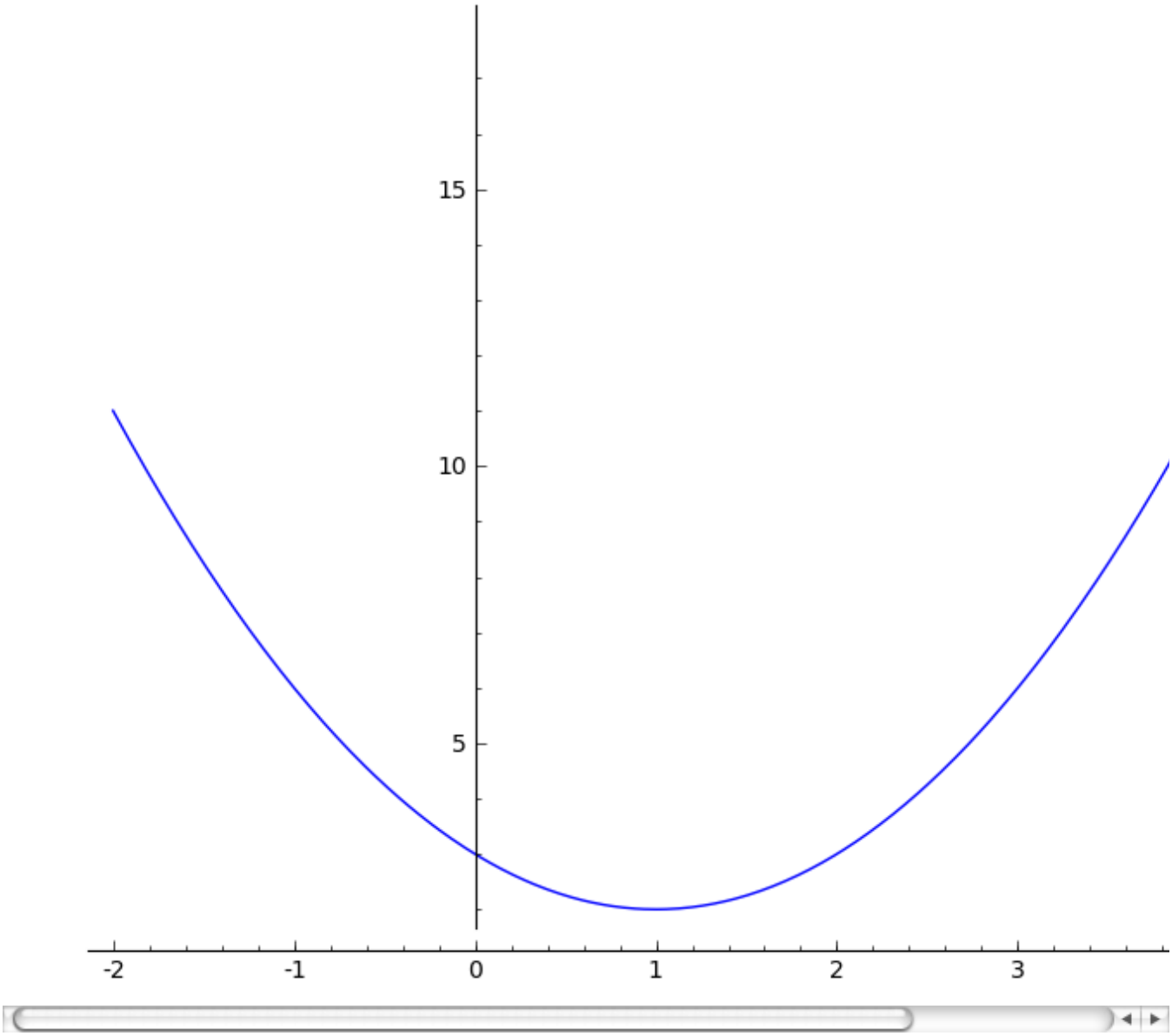
```
{'fillalpha': 0.5, 'detect_poles': False, 'plot_points': 200,
'thickness': 1, 'alpha': 1, 'adaptive_tolerance': 0.01, 'fillcol
'automatic', 'adaptive_recursion': 5, 'aspect_ratio': 'automatic
'exclude': None, 'legend_label': None, 'rgbcolor': (0, 0, 1),
'fill': False}
```

```
f(x) = x^2 -2*x +3
```

```
plot(f)
```

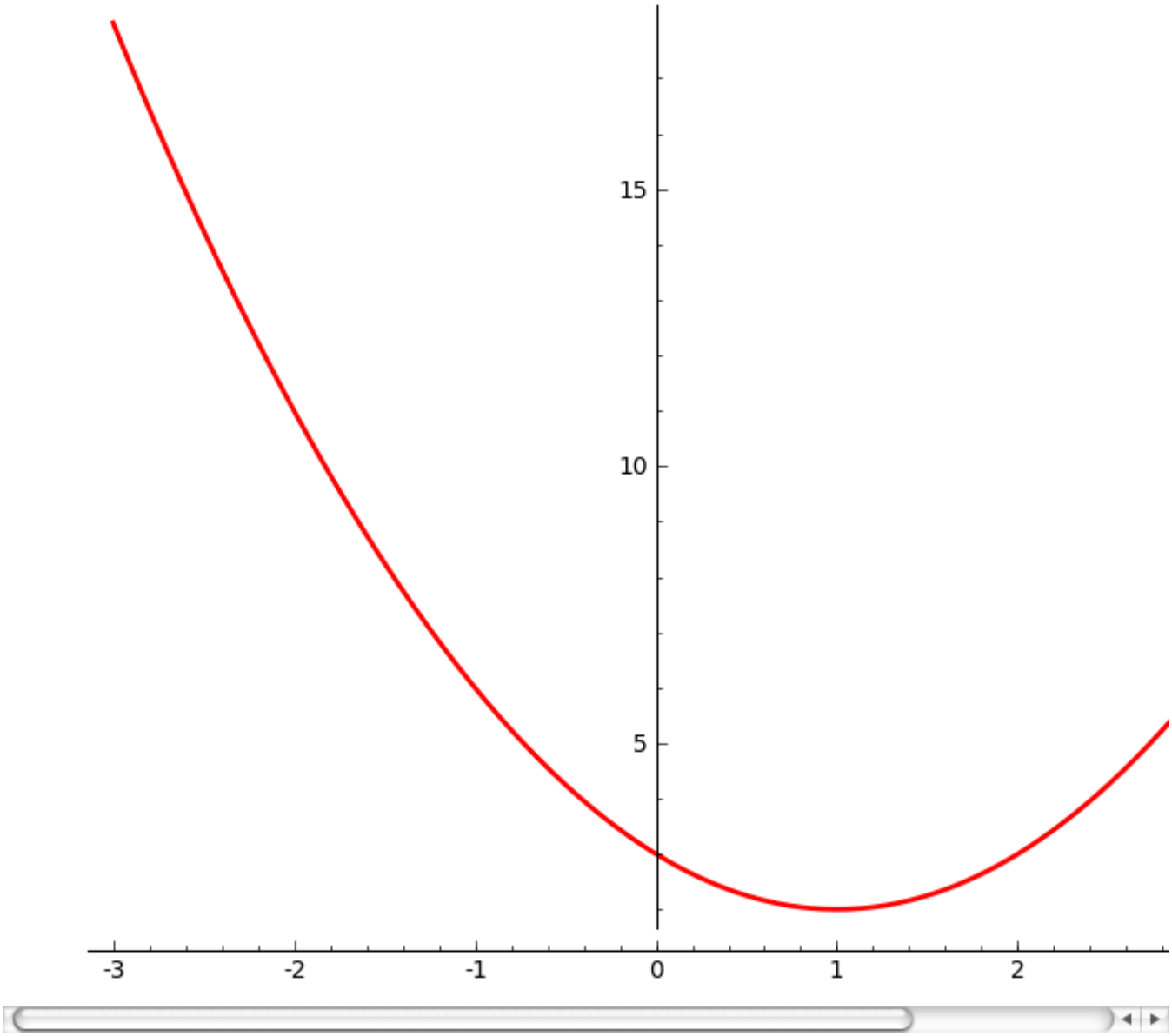



```
f.plot(-2,5) # Dibujo en el intervalo [-2,5]
```



Podemos cambiar el color de gráfica añadiendo la opción color. El nombre del color debe figurar entre comillas simples. Existen otras opciones como thickness que afecta al grosor de la gráfica, etc.

```
plot(f, -3, 4, color = 'red', thickness = 2)
```



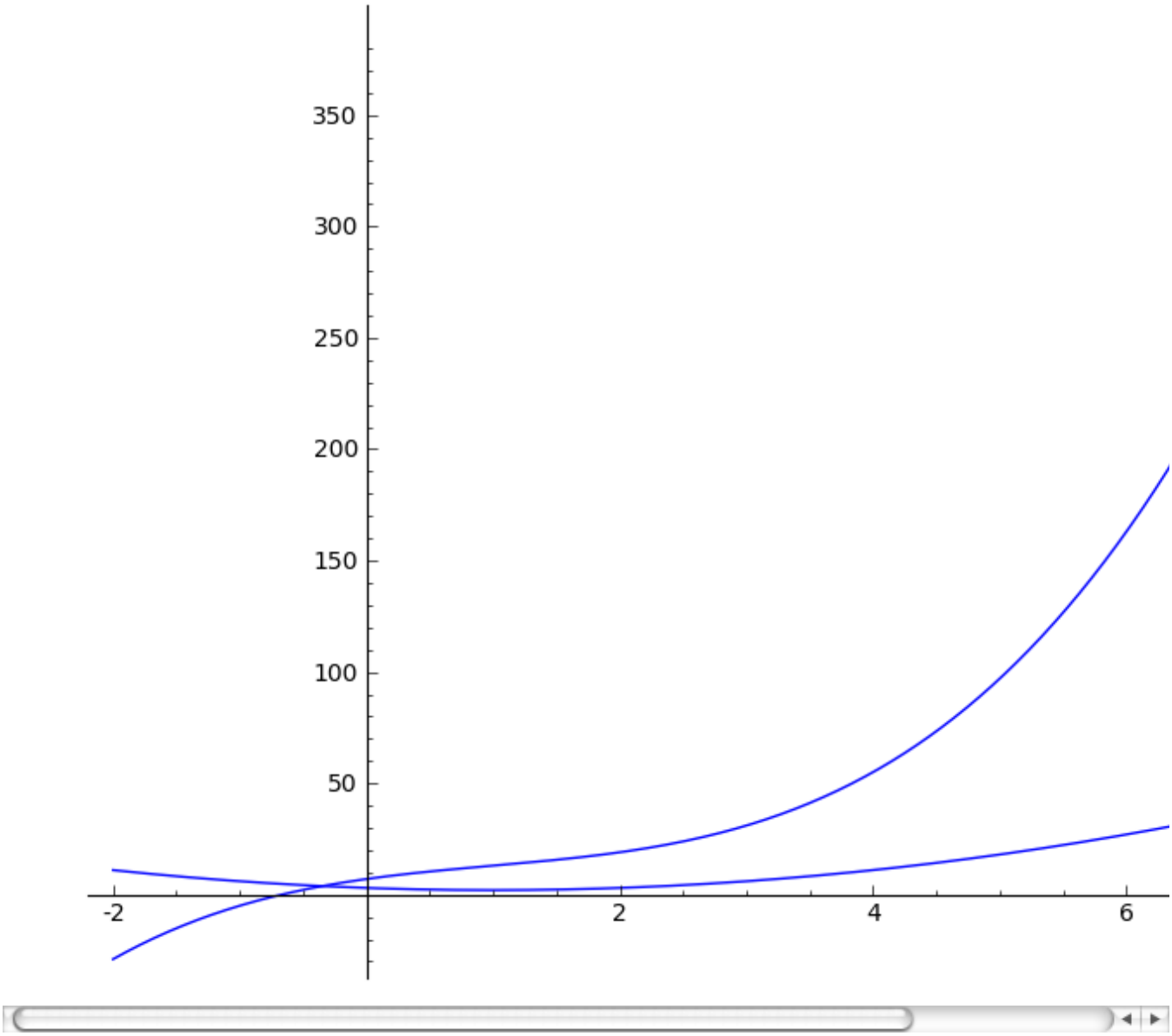
Gráficas simultáneas de varias funciones

Para representar la gráfica de las funciones **f1, f2, ...** de la variable **x** escribimos

plot([f1,f2,...], xmin,xmax, opciones)

```
g(x)=x^3-3*x^2+8*x+7
```

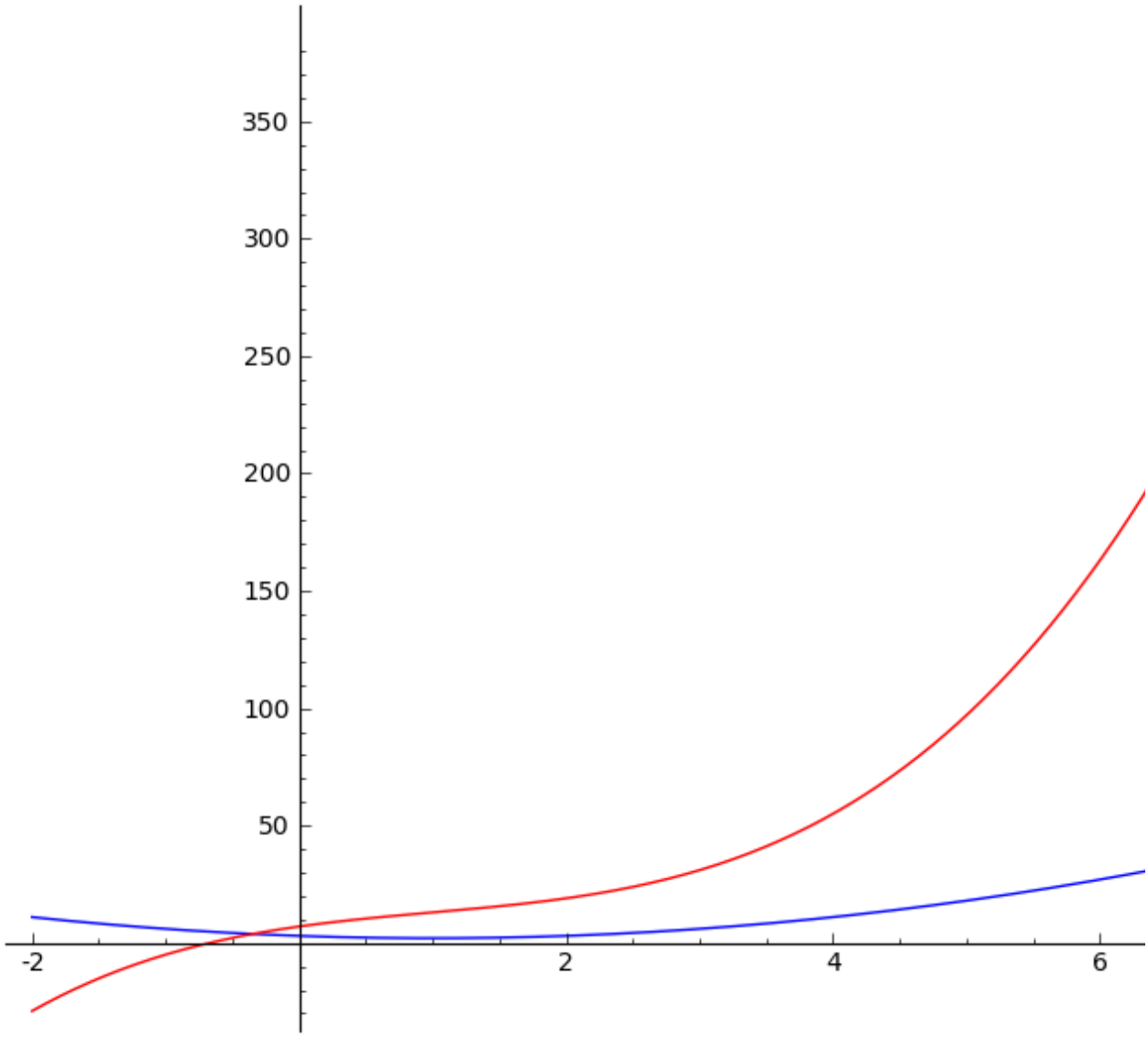
```
plot([f,g],-2,8)
```



Pero también podemos hacer cada gráfica de varias funciones por separado y después unirlas

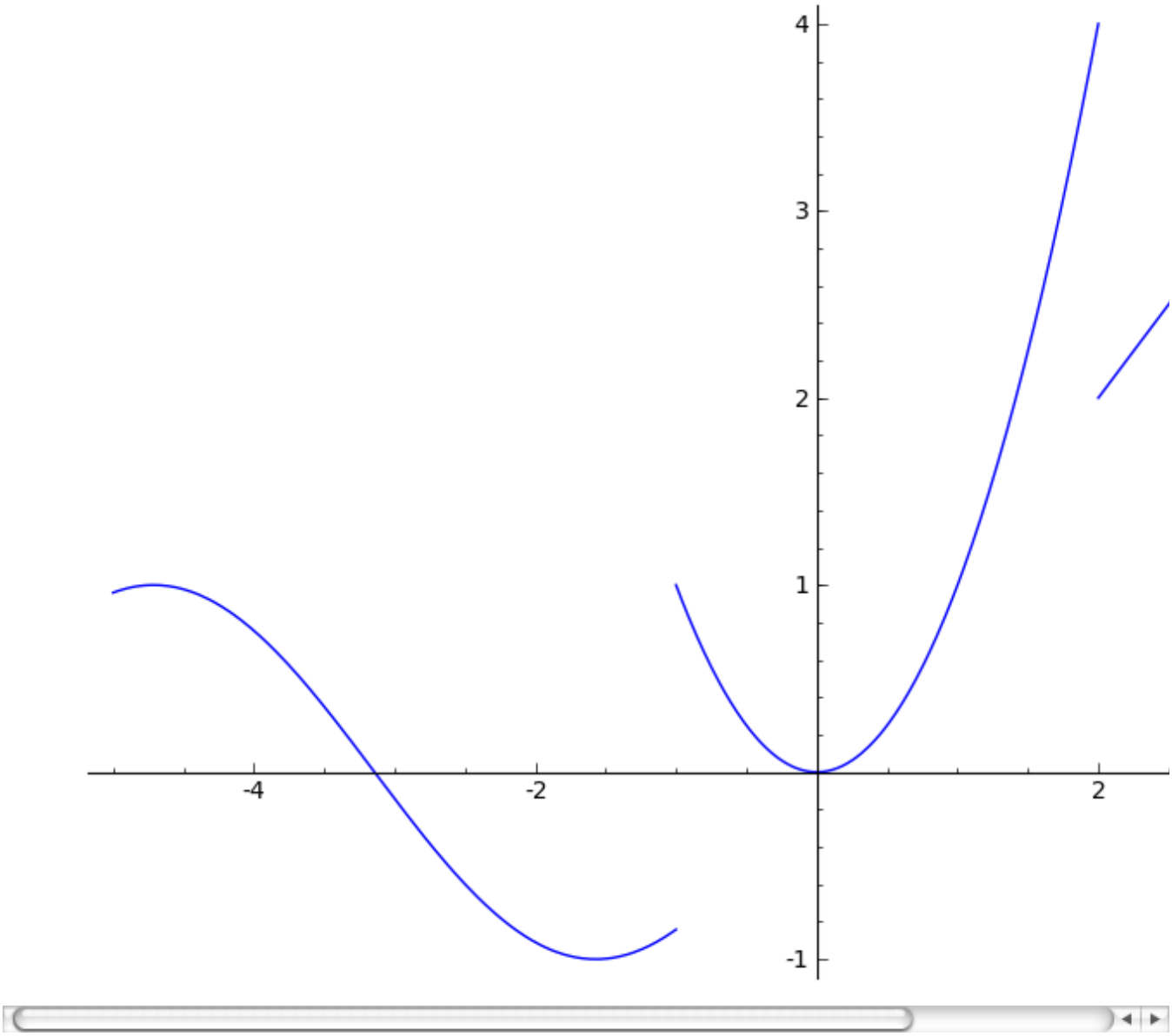
plot(f1,xmin1,xmax1, opciones1)+plot(f2,xmin2,xmax2, opciones2)+....+plot(fk,xmink,xmaxk, opcionesk)

```
plot(f,-2,8)+plot(g,-2,8, color = 'red')
```




```
f = Piecewise([[(-5,-1),sin(x)],[(-1,2),x^2],[(2,4),x]]) #
Función a trozos
```

```
plot(f) # Nótese que no hace falta especificar el valor mínimo y
máximo de la variable
```



Objetos gráficos

Los objetos gráficos pueden dibujarse directamente con los comandos que vamos a ver a continuación. Sin embargo, los ejes se adaptan al dibujo, lo que conlleva que la apariencia de los dibujos sea muy distinta a la real (una circunferencia parece una elipse, la pendiente de una recta no es la que corresponde,...). Por ello lo mejor es guardar cada uno de los objetos gráficos en una variable y después utilizar el comando **show()** con la opción aspect_ratio = 1 que hace que los ejes tengan la misma escala.

Para dibujar un punto p de coordenadas **p=(x1,y1)**

point2d((x1,y1))

Se le pueden añadir varias opciones que podemos conocerlas todas (y sus valores por defecto) con **point2d.options**

Para dibujar un segmento de una recta

line2d([p,q])

donde **p** es un extremo del segmento de la recta y **q** el otro. Ambos puntos deben ir colocados dentro de corchetes. Para ver las opciones se hace lo mismo que en el caso anterior. Para dibujar una circunferencia

circle(centro, radio)

donde **centro** es un punto que corresponde al centro y **radio** un número positivo que define el radio.

```
point2d.options
```

```
    {'legend_color': None, 'aspect_ratio': 'automatic', 'alpha': 1,
    'legend_label': None, 'faceted': False, 'rgbcolor': (0, 0, 1),
    'size': 10}
```

```
line2d.options
```

```
    {'legend_color': None, 'aspect_ratio': 'automatic', 'alpha': 1,
    'legend_label': None, 'rgbcolor': (0, 0, 1), 'thickness': 1}
```

```
circle.options
```

```
    {'edgecolor': 'blue', 'legend_color': None, 'facecolor': 'blue',
    'clip': True, 'legend_label': None, 'thickness': 1, 'zorder': 5,
    'aspect_ratio': 1.0, 'alpha': 1, 'linestyle': 'solid', 'fill':
    False}
```

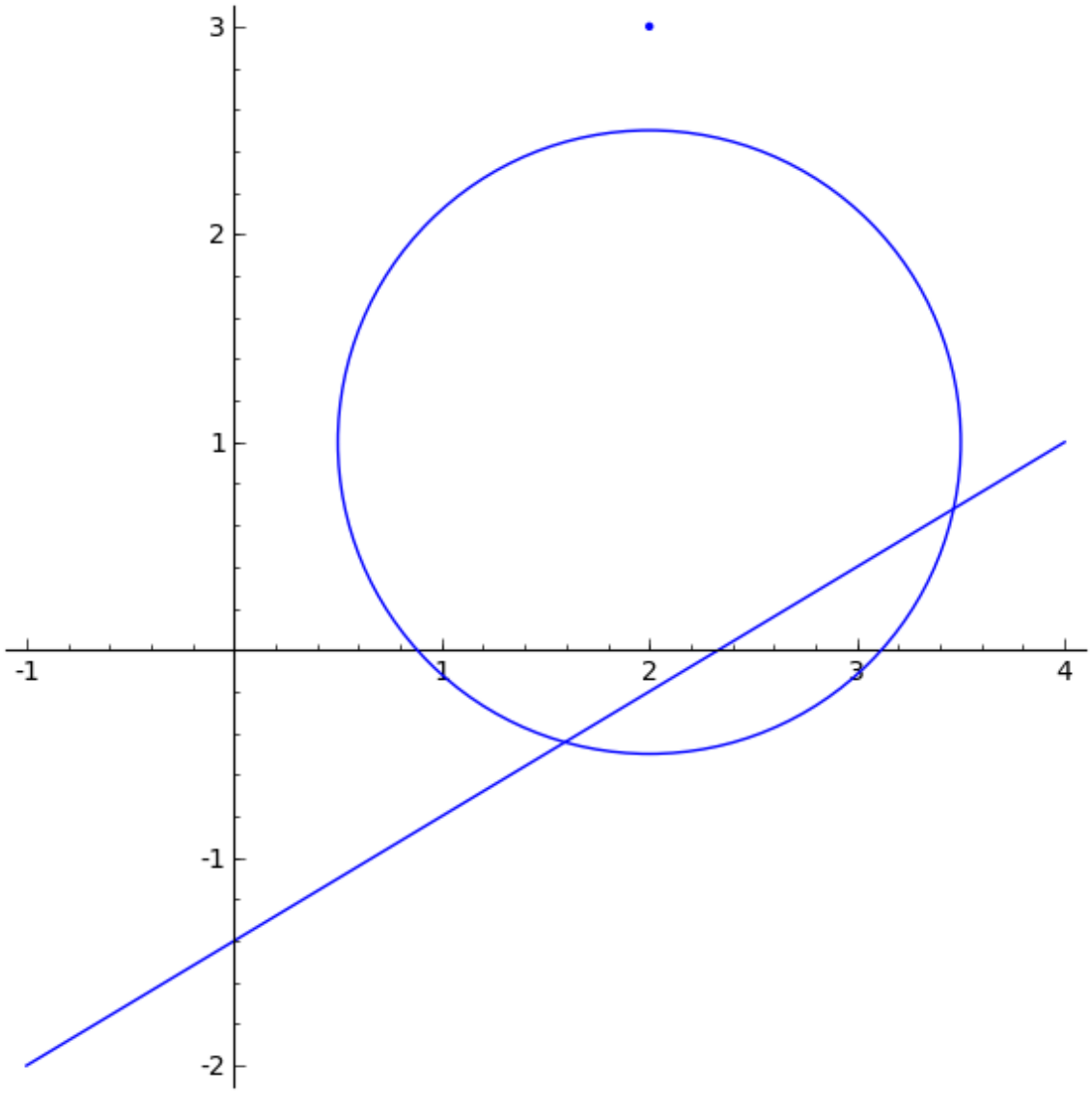
```
a = point2d((2,3));
b = line2d([(-1,-2),(4,1)]); # Un segmento
c = circle((2,1), 1.5);
```

```
type(a); type(b); type(c)
```

```
    <class 'sage.plot.graphics.Graphics'>
    <class 'sage.plot.graphics.Graphics'>
    <class 'sage.plot.graphics.Graphics'>
```

El comando **show** muestra los objetos graficos. La suma de objetos gráficos significa que los muestra todos a la vez.

```
show(a + b + c, aspect_ratio = 1) # Ejes con la misma escala
```




```
var('x')
x0  = 0
f   = sin(x)*e^(-x)
p   = plot(f,-1,5, thickness=2)
dot = point((x0,f(x=x0)),pointsize=80,rgbcolor=(1,0,0))
@interact
def _(order=(1..12)):
  ft = f.taylor(x,x0,order)
  pt = plot(ft,-1, 5, color='green', thickness=2)
  html('$f(x)\;=\;%s$'%latex(f))

html('$\hat{f}(x;%s)\;=\;%s+\mathcal{O}(x^{%s})$'%(x0,latex(ft),order+1))

  show(dot + p + pt, ymin = -.5, ymax = 1)
```