prácticaJP1

Práctica 1. Introducción a Sage.

1.1. Primeras operaciones.

Empecemos con algunas operaciones numéricas. Se emplea (/) para la división, (*) para el producto y (^) o (**) para la exponenciación.

```
2+3
```
```
5
```
```
4*6
```
```
24
```
```
12/6
```
```
2
```
```
3^2^5
```
```
1853020188851841
```

La principal diferencia de Sage con una calculadora estriba en su capacidad de realizar cálculos exactos o con precisión arbitraria. Así, si operamos con números racionales el resultado obtenido es un número racional y no su representación o aproximación decimal, como proporciona una calculadora. Con las sencillas operaciones siguientes se ilustra el hecho de que Sage trabaja con cantidades exactas (para escribir varias instrucciones en una misma celda de entrada podemos escribir cada instrucción en una línea diferente o separarlas con ';'):

```
12/8; 3/4+1/3; 2*sqrt(8); (12/8)*(2/3)
```
```
3/2
13/12
4*sqrt(2)
1
```

Con algunas calculadoras, es muy posible que esta última operación se hubiera realizado en la siguiente forma aproximada: 12/8=1.5, 2/3=0.6666667, y por tanto el producto es 1.0000005. Por supuesto que Sage permite utilizar aproximaciones decimales de un número. Esto es:

```
n(11/43)    # las órdenes n, N, numerical_approx son sinónimas
```
```
0.255813953488372
```

Las órdenes de Sage siempre encierran sus argumentos entre paréntesis. En una celda de entrada, Sage no ejecuta el texto que se inicia con #. Cuando en una operación aparece algún número real (de punto flotante) el resultado es siempre real. El resultado aparece por defecto con unas 15 cifras significativas, pero podemos conseguir mayor precisión especificando el número de dígitos como argumento opcional.

```
11./43
n(11/43,digits=50)
```
```
0.25581395348837209302325581395348837209302325581395
```

Sage sólo muestra la salida de la última instrucción de la celda, si queremos que muestre la salida de alguna operación que no sea la última debemos emplear la orden **`print`**

```
print 11./43
n(11/43,digits=50)
```
```
0.255813953488372
0.25581395348837209302325581395348837209302325581395
```

Parece más interesante que Sage trabaje con números exactos. De hecho, opera y simplifica (el símbolo _ hace referencia al resultado de la última celda ejecutada):

```
3/7*5/9; n(_)
```
```
5/21
0.238095238095238
```

Como es regla general en matemáticas, Sage realiza primero las operaciones entre paréntesis, después las potencias de derecha a izquierda, continúa con las multiplicaciones y divisiones de izquierda a derecha y, por último las sumas y restas de izquierda a derecha. Para agrupar operaciones en Sage se usan exclusivamente paréntesis, nada de corchetes y llaves.

```
((3+5)*4)^2
```
```
1024
```

Sage también calcula el valor absoluto, la parte entera y el signo de un número dado:

```
abs(3); abs(-2/3); floor(2/3); floor(-4/3); sgn(4); sgn(-56)
```
```
3
2/3
0
-2
1
-1
```

Podemos trabajar con números complejos:

```
z1=2+3*I; z2=4+5*I; z1+z2; z1*z2; z1/z2; conjugate(z1)
```
```
8*I + 6
22*I - 7
2/41*I + 23/41
-3*I + 2
```

Funciones elementales

Hasta ahora hemos utilizado Sage como una calculadora. También podemos usarlo como una calculadora científica exacta, ya que tiene predefinidas las funciones elementales y algunas constantes; para usarlas sólo hay que conocer la sintaxis apropiada.

`log(x), sin(x), cos(x), tan(x), asin(x), acos(x), atan(x), exp(x), e , pi ...`

Sage distingue entre mayúsculas y minúsculas. Los argumentos de las funciones van entre paréntesis. Huelga decir que si no escribimos bien la expresión, los resultados obtenidos no serán correctos. Podemos trabajar de modo exacto o aproximado:

```
sin(pi/3); exp(3); log(exp(3)); N(exp(3))
```
```
1/2*sqrt(3)
e^3
3
20.0855369231877
```

1.2. Variables. Concatenación de instrucciones.

Asignación de variables.

```
a=5 # crea la variable 'a' de tipo entero que tiene el valor 5
```

Al hacer una asignación de variable Sage no muestra una salida.

```
a; a+27
```

```
5
32
```

```
b=7; c=3; a*b*c
```

```
105
```

Sage incorpora características de lenguaje de programación (basado en Python). Sage puede trabajar en simbólico, pero es un lenguaje de programación tipado. Es decir, cualquier símbolo (excepto 'x', que por defecto se considera una variable) que usemos en Sage ha de ser definido previamente como miembro de un tipo de datos. De momento veremos dos formas de hacerlo:

1. Asignando un valor a la variable. P. ej. escribiremos `a=5`. o también `poli=x^2+3*x`.

2. Si queremos que una variable sea genérica escribiremos p. ej. `y=var('y')`

```
factor(x^3-1)  #la orden factor factoriza un polinomio.
```

```
(x - 1)*(x^2 + x + 1)
```

```
factor(y^3-1)
```

```
Traceback (click to the left of this block for traceback)
...
NameError: name 'y' is not defined
```

Da error, ya que no está definido 'y'. Ahora lo definimos primero.

```
y=var('y'); factor(y^3-1)
```

```
(y - 1)*(y^2 + y + 1)
```

El comando **`reset()`** borra todas las variables definidas. Si se especifica la variable solo se borra el valor de dicha variable y las demás quedan intactas.

```
reset('y'); factor(y^3-1)
```

```
Traceback (click to the left of this block for traceback)
...
NameError: name 'y' is not defined
```

1.3. Cálculo simbólico

Aquí empezamos a atisbar la importancia del uso de Sage. Esta clase de programas pueden operar con símbolos y hacer simplificaciones y manipulaciones como las harías en un papel. Las órdenes que aprenderemos en este apartado son:

- `expr.`**`full_simplify()`** aplica varios métodos de simplificación sobre "expr": simplify_factorial, simplify_trig, simplify_rational y simplify_radical.
- `expr.`**`expand()`** ejecuta las multiplicaciones y potencias de "expr" para transformarlo en sumas.
- `poli.`**`factor()`** ó **`factor(`**`poli)` factoriza el polinomio "poli".
- `expr.`**`partial_fraction()`** descompone en fracciones simples "expr".

En general Sage es muy cuidadoso y no simplificará una fracción racional a menos que le indiquemos que es una fracción racional o que le pidamos explícitamente que simplifique

```
(x^2-1)/(x-1); _.full_simplify()
```

```
(x^2 - 1)/(x - 1)
x + 1
```

También puede simplificar expresiones no polinómicas

```
ex=(1+(tan(x))^2)*cos(x)*e^(x^2-x)/(e^x)
ex.simplify_full()  # simplify_full es sinónimo de full_simplify
```

```
e^(x^2 - 2*x)/cos(x)
```

Si lo que queremos es desarrollar un producto utilizamos la orden `expand`

```
poli=((x+1)*(x+2)-(x+2)^2)^3
expand(poli)
```

```
-x^3 - 6*x^2 - 12*x - 8
```

También podemos factorizar polinomios (si sus raíces son enteras, si no, no funciona)

```
factor(x^4-1)
```

```
(x - 1)*(x + 1)*(x^2 + 1)
```

O descomponer en fracciones simples (método estándar que se emplea, por ejemplo, para calcular primitivas de funciones racionales o sumar series).

```
frac=1/((3+x)*(1+x)^3)
frac.partial_fraction()
```

```
1/8/(x + 1) - 1/4/(x + 1)^2 + 1/2/(x + 1)^3 - 1/8/(x + 3)
```

1.4 Resolución de ecuaciones e inecuaciones.

1.4.1. Resolución de ecuaciones y sistemas.

Sage permite resolver ecuaciones y sistemas (sobre los números complejos).

- **`solve(`**`eqns, v1, v2,...)` resuelve las ecuaciones 'eqns' en las variables v1, v2, ...

Algunas observaciones

1. Las ecuaciones se escriben en la forma: parte izda. **==** parte dcha.
2. ecuaciones simultáneas se escriben entre corchetes y separadas por comas, e.d. `[eqn1,eqn2,eqn3]`

En general, la orden `solve` está poco desarrollada y sólo resuelve ecuaciones y sistemas polinómicos y algún otro caso sencillo. Veamos algunos ejemplos.

```
solve(x^3-2*x+1==0,x)
```

```
[x == -1/2*sqrt(5) - 1/2, x == 1/2*sqrt(5) - 1/2, x == 1]
```

Por defecto no muestra multiplicidades: x=1 es una raíz doble del polinomio x^2-2x+1, sin embargo, si queremos que nos muestre que es una solución doble, tenemos que indicárselo.

```
solve(x^2-2*x+1==0,x); solve(x^2-2*x+1==0,x,multiplicities=True)
```

```
[x == 1]
([x == 1], [2])
```

```
solve(x^8-256==0,x)  #posee 8 raíces complejas, 2 de ellas reales.
```

```
[x == (I + 1)*sqrt(2), x == (2*I), x == (I - 1)*sqrt(2), x == -2,
== -(I + 1)*sqrt(2), x == (-2*I), x == -(I - 1)*sqrt(2), x == 2]
```

En cuanto la ecuación es un poco complicada, para resolverla tenemos que activar el procedimiento "`to_poly_solve`" de Maxima. Este procedimiento es incompatible con que muestre multiplicidades.

```
solve(sqrt(x+1)-2*sqrt(x-1)==1,x,to_poly_solve=True)
```

```
[x == -4/9*sqrt(7) + 20/9]
```

Al resolver $\sqrt{1/4-x}=\sqrt{x-3/4}$ Sage nos da la solución x=1/2. Notar que aunque la solución sea un número real, la igualdad es de números complejos $\sqrt{-1/4}=\sqrt{-1/4}$

```
solve(sqrt(1/4-x)==sqrt(x-3/4),x,to_poly_solve=True)
```

```
[x == (1/2)]
```

Más ejemplos:

```
solve(log(2*x)+log(4+x)==log(x),x,to_poly_solve=True);
```

```
[x == (-7/2)]
```

```
solve(abs(1-abs(1-x))==10,x,to_poly_solve=True)
```

```
[x == -10, x == 12]
```

Veamos cómo se desenvuelve con un sistema de ecuaciones dependiente de un parámetro

```
x,y,a=var('x,y,a'); solve([2*a*x-y==1, 3*a*y+x==1],x,y)
```

```
[[x == (3*a + 1)/(6*a^2 + 1), y == (2*a - 1)/(6*a^2 + 1)]]
```

Es posible que no encuentre soluciones. Porque no las haya o porque no sea capaz de obtenerlas:

```
solve([2*x+y==1,4*x+2*y==3],x)
```

```
[]
```

```
solve(2*x==sin(x),x)
```

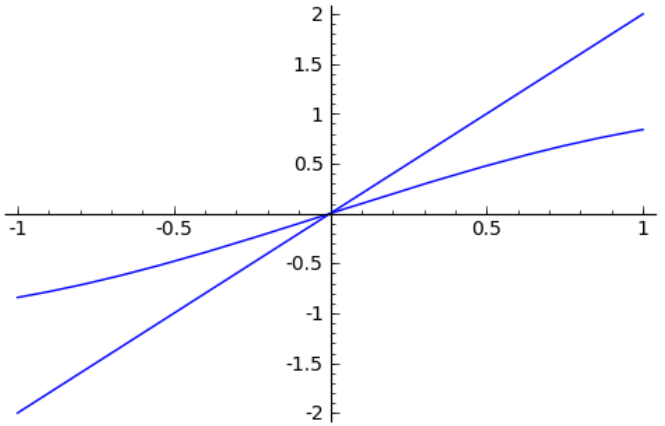
```
[x == 1/2*sin(x)]
```

Cuando no somos capaces de resolver de modo exacto usamos métodos numéricos.

- **`find_root`**`(eqn,a,b)` encuentra una solución aproximada de 'eqn' en [a,b].

Es importante elegir bien el intervalo en el que buscar la solución. Además hay que tener en cuenta que sólo nos devuelve una solución, aunque hubiera varias. Por todo esto, es muy conveniente hacer primero una representación gráfica.

```
plot([2*x,sin(x)],-1,1)
```



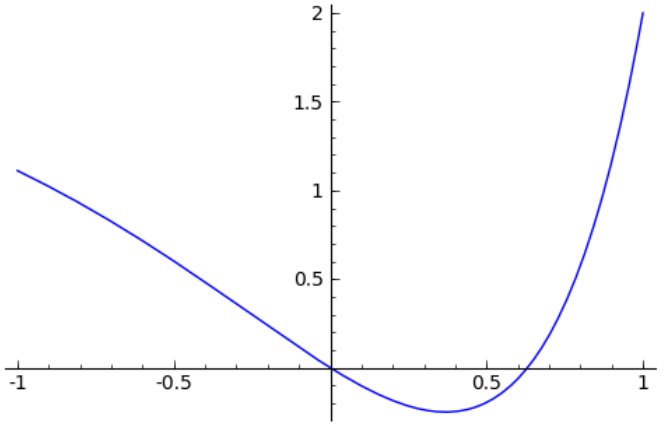

```
find_root(2*x==sin(x),-1/2,1/2)
```

```
0.0
```

```
solve(9^x-3^(x+1)+2==0,x,to_poly_solve=True)
```

```
Traceback (click to the left of this block for traceback)
...
TypeError: 'sage.symbolic.expression.Expression' object does not
support indexing
```

```
plot(9^x-3^(x+1)+2,-1,1)
```




```
find_root(9^x-3^(x+1)+2==0,-1,1)
```

```
0.63092975357145686
```

```
find_root(9^x-3^(x+1)+2==0,-1,1/2)
```

```
3.7478822620183693e-16
```

En realidad la solución exacta es cero, pero hay un error del orden 10^(-16).

1.4.2. Resolución de inecuaciones.

- la instruccion **`solve`** explicada en el apartado anterior resuelve inecuaciones sencillas
- si solve falla, también podemos emplear **`solve_ineq`**

```
solve(x*(x^2-4)*(x^2-3)>0,x)
```

```
[[x > -2, x < -sqrt(3)], [x > 0, x < sqrt(3)], [x >
2]]
```

```
solve_ineq(x*(x^2-4)*(x^2-3)>0,x)
```

```
[[x > -2, x < -sqrt(3)], [x > 0, x < sqrt(3)], [x >
2]]
```

En general, las desigualdades deberán ser sencillas para que Sage sea capaz de resolverlas. Muchas veces es mejor hacerlas a mano. En el siguiente ejemplo avanza un paso, pero deja la solución a medias.

```
solve([x/abs(x-1)>=0,1/x<x+1],x)
```

```
[[0 < x, x < 1, x^2 + x - 1 > 0], [1 < x, x^2 + x - 1
> 0]]
```

Así que la completamos nosotros

```
solve([x^2 + x - 1 > 0],x)
```

```
[[x < -1/2*sqrt(5) - 1/2], [x > 1/2*sqrt(5) - 1/2]]
```

Por tanto la solución del sistema $\frac{x}{\mathrm{abs}(x-1)} \geq 0$, $\frac{1}{x} < x+1$ es la unión $(\frac{\sqrt{5}-1}{2}, 1) \cup (1, \infty)$.

```

```