SageMath è un computer algebra system: effettua calcoli anche con espressioni simboliche, permette di disegnare grafici ed è dotato di un linguaggio di programmazione

Modalità di accesso:

Online: https://sagecell.sagemath.org/ o https://cocalc.com/

Offline: https://www.sagemath.org/download.html, da cui scaricare il software da installare e da cui si può anche accedere ai servizi online indicati sopra

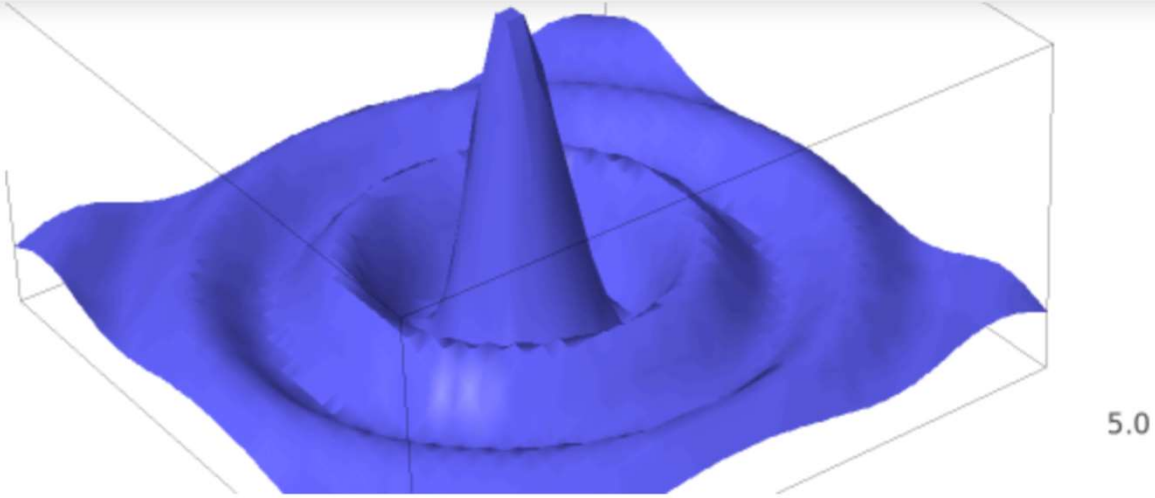
SageMath (in breve Sage) nasce da un progetto iniziato nel 2005 da William Stein con lo scopo di creare un software libero per i calcoli matematici

jupyter a notebook (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

SageMath 7.6



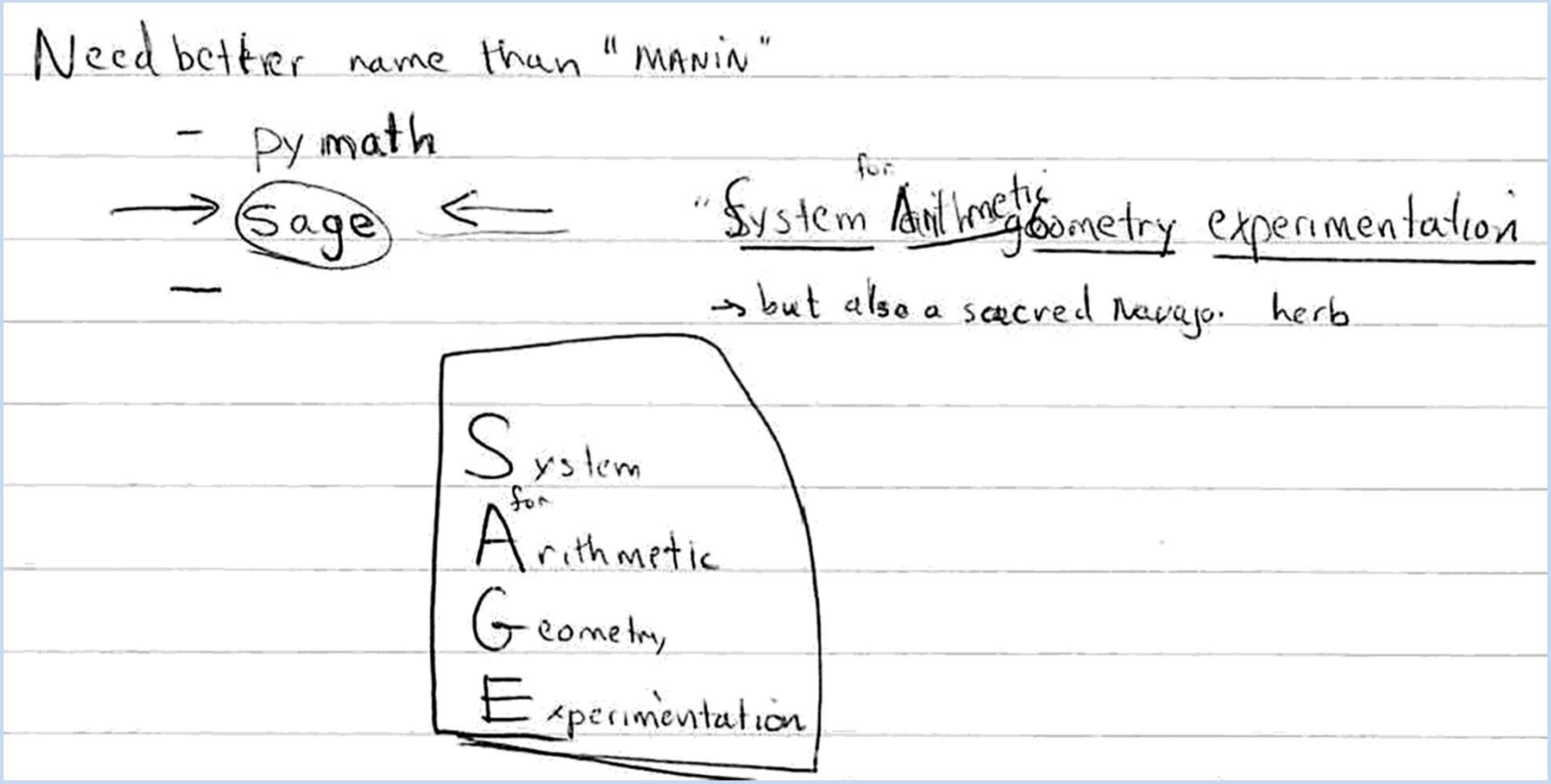
This graph of the function $\frac{sin(\pi * \sqrt{x^2+y^2})}{\sqrt{x^2+y^2}}$ was drawn with `plot3d`. See the documentation:

In [1]:
```
plot3d?
```

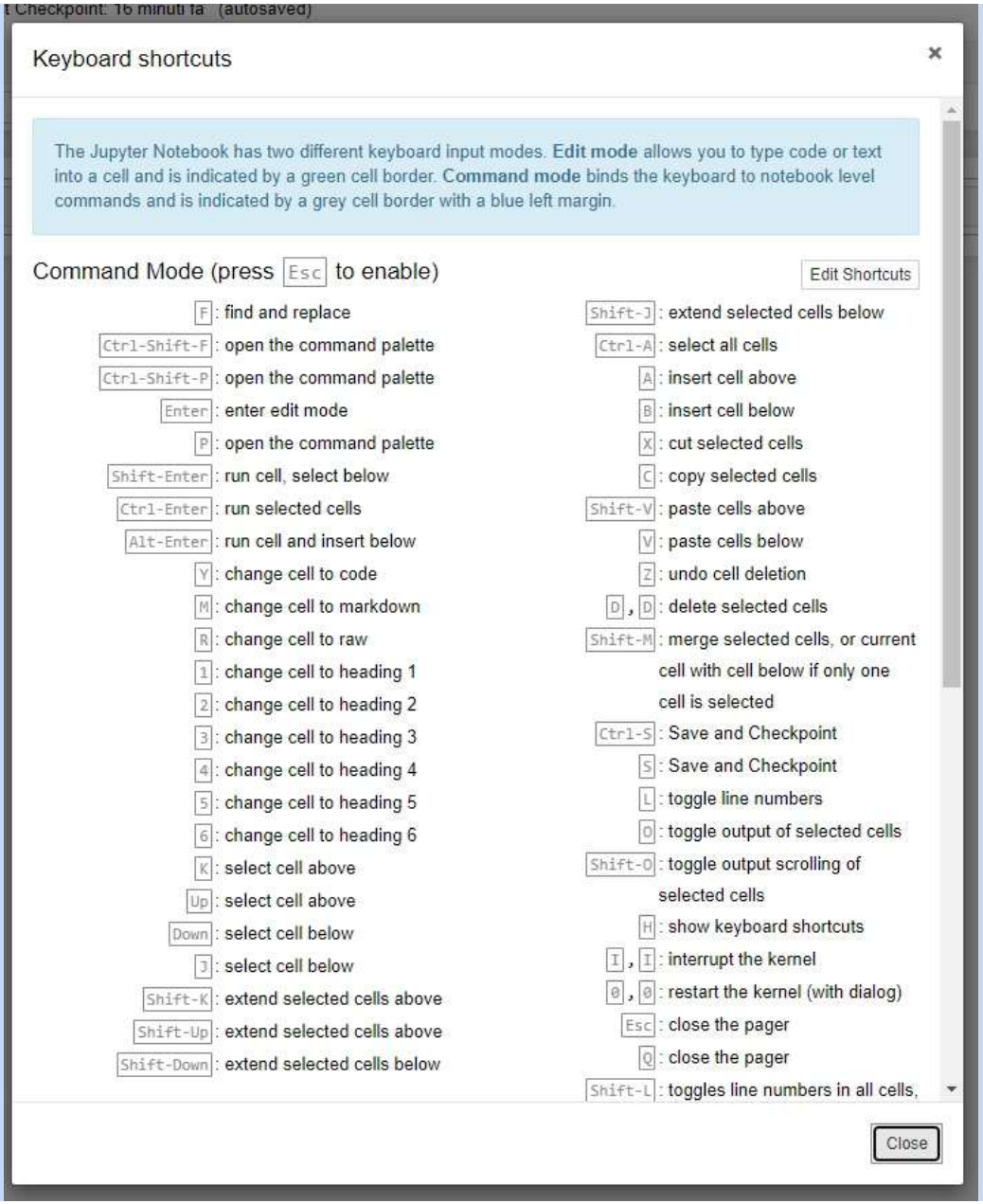
In [4]:
```
factor(x^128-1)
```

Out[4]: $(x^{64}+1)(x^{32}+1)(x^{16}+1)(x^8+1)(x^4+1)(x^2+1)(x+1)(x-1)$

```
Signature:      plot3d(f, urange, vrange, adaptive=False, transformation=None, **kwds)
```



Sage è sia un acronimo sia un riferimento alla pianta della salvia



Ctrl + Enter : esegue il comando nella cella (o nelle celle selezionate)

Alt + Enter: esegue il comando nella cella e inserisce una nuova cella

Shift + Enter: esegue il comando nella cella e seleziona la cella successiva

A: inserisce una cella sopra

B: inserisce una cella sotto

C e V: copia e incolla (sotto) le celle selezionate

X: taglia le celle selezionate

D 2 volte: elimina le celle selezionate

NB: se necessario, passare in modalità comando (margine sinistro della cella blu) premendo ESC

Operatori aritmetici

Operatori aritmetici di base: +, -, *, /, ^, - (unario)

per somma, sottrazione, moltiplicazione, divisione, elevamento a potenza, numeri negativi

```
sage:1+1
2
sage: 103-101
2
sage: 7*9
63
sage: 7337/11
667
sage: 11/4
11/4
sage: 2^5
32
```

NB: rende una frazione!

Sage rispetta l'ordine standard delle operazioni e l'uso delle parentesi

```
sage: -6
-6
sage: -11+9
-2
sage: 2*4^2+1
 33
sage: (2*4)^2+1
 65
sage: 2*4^(2+1)
128
sage: -3^2
 -9
sage: (-3)^2
 9
```

Vari modi di avere l'espressione decimale invece della frazione

```
sage: 11/4
11/4

sage: 11/4.0

2.75000000000000

sage: 11/4.

2.75000000000000

sage: 11.0/4

2.75000000000000

sage: 11/4*1.

2.75000000000000
```

```
sage: 14 // 4
```
Per avere il quoziente della divisione intera

```
3
```

```
sage:14 % 4
```
Per avere il resto della divisione intera

```
2
```

```
sage: divmod(14,4)
```
Per avere entrambi

```
(3, 2)
```

```
sage: 3.divides(15)
```
Per verificare se un numero divide un altro

```
True
```

```
sage: 5.divides(17)
```
NOTAZIONE PUNTATA

```
False
```

```
sage: 12.divisors()
[1,2, 3, 4, 6, 12]
sage: 101.divisors()
[1,101]
```

Per avere i divisori di un numero

```
sage: (2*19-1).is_prime()
True
sage: 153.is_prime()
False
```

Per verificare se un numero è primo

```
sage: 62.factor()
2 * 31
sage: 63.factor()
3*2 * 7
```

Per avere la fattorizzazione in numeri primi

```
sage: a=2; a
2
sage: 1 + 1
2
sage: _ + 1
3
sage: __
2
```

Si usa il punto e virgola per indicare più comandi sulla stessa linea

Il carattere di sottolineatura _ è una variabile predefinita che conserva l'ultimo risultato; se raddoppiato (__) conserva il penultimo risultato e se triplicato (___) il terzultimo.

Principali funzioni matematiche

```
sage: 24.prime_divisors()
[2, 3]
sage: 63.prime_factors()
[3, 7]
```

Per i divisori/fattori primi

```
sage: gcd(14,63)
7
```

Massimo comun dividsore

```
sage: gcd(15,19)
1
```

gcd = greatest common divisor

```
sage: lcm(4,5)
20
```

Minimo comune multiplo

```
sage: lcm(14,21)
42
```

lcm = least common multiple

```
sage: max(1,5,8)
8
sage: min(1/2,1/3)
 1/3
sage: abs(-10)
10
sage: abs(4)
4
```

Nelle funzioni max e min si possono inserire quanti argomenti si desidera

```
sage: floor(2.1)
2
sage: ceil(2.1)
3
sage: floor(1 /(2.1-2))
9
sage: 1/(2.1-2)
9.99999999999999
sage: 1/(21/10-2)
10
```

NB: 2.1-2=0.1 e 1/0.1=10, quindi floor(10)=10.

Cosa è successo?

La rappresentazione dei numeri in macchina è binaria!

Usando frazioni (2.1=21/10) il risultato è però corretto.

```
sage: 3^(1/2)
sqrt(3)
sage: (3.0)^(1/2)
1.73205080756888
sage: 8^(1/2)
2*sqrt(2)
sage: 8^(1/3)
2
```

```
sage: sin(1)
sin(1)
sage: sin(1.0)
0.841470984807897
sage: cos(3/2)
cos(3/2)
sage: cos(3/2.0)
0.0707372016677029
```

```
sage: pi
pi
sage: sin(pi/3)
1/2*sqrt(3)
```

Sage dà in genere una risposta esatta. Se l'operando è decimale il risultato è invece decimale (e generalmente approssimato).

La capacità di trattare risultati esatti consente a Sage di elaborare e semplificare espressioni

```
sage: pi.n()
3.14159265358979
sage: sin(pi)
0
sage: sin(pi.n())
1.22464679914735e-16
sage: sin(pi/4)
1/2*sqrt(2)
sage: numerical_approx(sin(pi/4), digits = 5)
0.70711
```

.n() consente di ottenere valori approssimati
Si può usare anche nella forma n(pi).
In alternativa si può usare numerical_approx

Attenzione però alle approssimazioni successive.

In Sage sono disponibili tutte le più comuni funzioni come:

sin, cos, tan, arcsin, …, log (o ln),

log e ln rendono sempre il logaritmo in base e. Per cambiare base, si inserisca un secondo parametro.

```
sage: log(10.0)
2.30258509299405
sage: log(100,10)
2
```

e e pi sono due costanti predefinite per il numero di Nepero e Pi Greco.

Operatori di confronto

```
sage: 9 == 9
True
sage: 9 <= 10
True
sage: 3*x - 10 == 5
3*x - 10 == 5
```

Operatori di confronto: ==, <=, >=, >, <, !=, <>

```
sage: solve(3*x - 2== 5,x)
[x == (7/3)]
sage: solve(2*x - 5>= 17,x)
[[x >= 11]]
```

Si usa solve per risolvere equazioni e disequazioni, indicando anche la variabile rispetto la quale risolvere

Risolvere equazioni e disequazioni

```
sage: solve( x^2 + x == 6, x)
[x == -3, x == 2]
sage: solve(2*x^2 - x + 1 == 0, x)
[x == -1/4*I*sqrt(7) + 1/4, x == 1/4*I*sqrt(7) + 1/4]
sage: solve(exp(x) == -1, x)
[x == I*pi]
```

Se si hanno più soluzioni, queste vengono restituite in forma di lista

```
sage: solve( x^2 - 6 >= 3, x )
[[x <= -3], [x >= 3]]
```

Soluzione che è unione di intervalli

```
sage: solve( x^2 - 6 <= 3, x )
[[x >= -3, x <= 3]]
```

Soluzione che è intersezione di intervalli

```
sage: solve(sin(x) == x, x)
[x == sin(x)]
sage: solve(cos(x) - exp(x) == 0, x)
[cos(x) == e^x]
```

Sage tenta sempre di trovare una soluzione non approssimata. Se non gli è possibile, rende una soluzione simbolica.

```
sage: find_root(sin(x) == x, -pi/2 , pi/2)
0.0
sage: find_root(sin(x) == cos(x), pi, 3*pi/2)
3.9269908169872414
```

Per trovare una soluzione numerica (approssimata), si usa find_root, indicando l'intervallo nel quale va cercata la soluzione.

```
sage: arccos(sin(pi/3))
arccos(1/2*sqrt(3))
sage: simplify(arccos(sin(pi/3)))
1/6*pi
sage: numerical_approx(arccos(sin(pi/3)),
digits=10)
0.5235987756
```

Non sempre vengono eseguite tutte le semplificazioni. In tal caso è possibile chiederle esplicitamente

Resta possibile anche trovare valori approssimati

Variabili simboliche

```
sage: y,z,t = var("y z t")
sage: phi, theta, rho = var("phi theta rho")
sage: x1, x2 = var("x1 x2")
sage: u
NameError: name 'u' is not defined
sage: solve (u^2-1,u)
NameError    Traceback (most recent call last)
NameError: name 'u' is not defined
sage: restore('phi')
sage: phi
NameError: name 'phi' is not defined
```

Le variabili simboliche vanno dichiarate, con l'eccezione di x che è automaticamente dichiarata (si può anche scrivere soltanto var("y z t") e si può anche usare ' (apice semplice) invece di " (doppio apice)

Una variabile non dichiarata produce una segnalazione di errore

Si può annullare la dichiarazione di una variabile con restore

Variabili simboliche e variabili Python

```
sage: b = var("a"); b
a
sage: b+b
2*a
sage: var("c")
c
sage: c
c
sage: c=2; c
2
```

```
sage: f(c)=c^2+c-1
x |--> c^2 + c - 1
sage: f(1)
1
```

Le variabili simboliche sono distinte dalle variabili usuali (dette in Sage variabili Python), che non è necessario dichiarare.

Ne primo esempio a fianco viene creata una variabile simbolica a «puntata» da una variabile Phyton b.

Nel secondo esempio viene creata automaticamente una variabile Python c che «punta» a una variabile simbolica c.
La variabile simbolica viene poi usata per creare una funzione, che viene successivamente valutata.

```
sage: solve( [3*x - y == 2, -2*x -y == 1 ], x,y)
[[x == (1/5), y == (-7/5)]]
```

Con solve si possono risolvere sistemi di equazioni, usando come parametro una lista contenente tali equazioni

```
sage: solve( [ 2*x + y == -1 , -4*x - 2*y == 2],x,y)
[[x == -1/2*r1 - 1/2, y == r1]]
```

<- Soluzione parametrica (1 parametro)

```
sage: solve([ 2*x + 3*y + 5*z == 1, 4*x + 6*y +
10*z == 2, 6*x + 9*y + 15*z == 3], x,y,z)
[[x == -5/2*r1 - 3/2*r2 + 1/2, y == r2, z == r1]]
```

<- Soluzione parametrica (2 parametri)

```
sage: solve( [ 2*x - y == -1 , 2*x - y == 2],x,y)
[]
```

<- Nessuna soluzione

solve risolve anche sistemi di disequazioni, ma le soluzioni possono assumere forme alquanto complesse

Definire funzioni matematiche

Definizione di funzioni

```
sage: f(x) = x*exp(x)
sage:f
x |--> x*e^x
sage: g(x) = (x^2)*cos(2*x)
sage:g
x |--> x^2*cos(2*x)
sage: h(x) = (x^2 + x - 2)/(x-4)
sage:h
x |--> (x^2 + x - 2)/(x-4)
```

Valutazione di funzioni

```
sage: h(-1)
2/5
sage: k = h+2
sage: k(1)
2
sage: k(x=1)
2
sage: k.subs(x=y)
x |--> (y^2 + y - 2)/(y - 4) + 2
```

Funzioni max e min che operano con simboli

Nel caso di espressioni simboliche le funzioni max e min non funzionano.
In questo caso si usano max_symbolic e min_symbolic

```
sage: f(x) = max(x,x^2); f
x |--> x
sage: max_symbolic(x,x^2)
x |--> max(x, x^2)
sage: f(x)=max_symbolic(x,x^2); f(1/2)
1/2
```

La risposta ottenuta con max è errata, perché la risposta corretta dipende dal valore di x

max_symbolic attende invece di conoscere il valore di x per dare la risposta corretta

Limiti – derivate - integrali

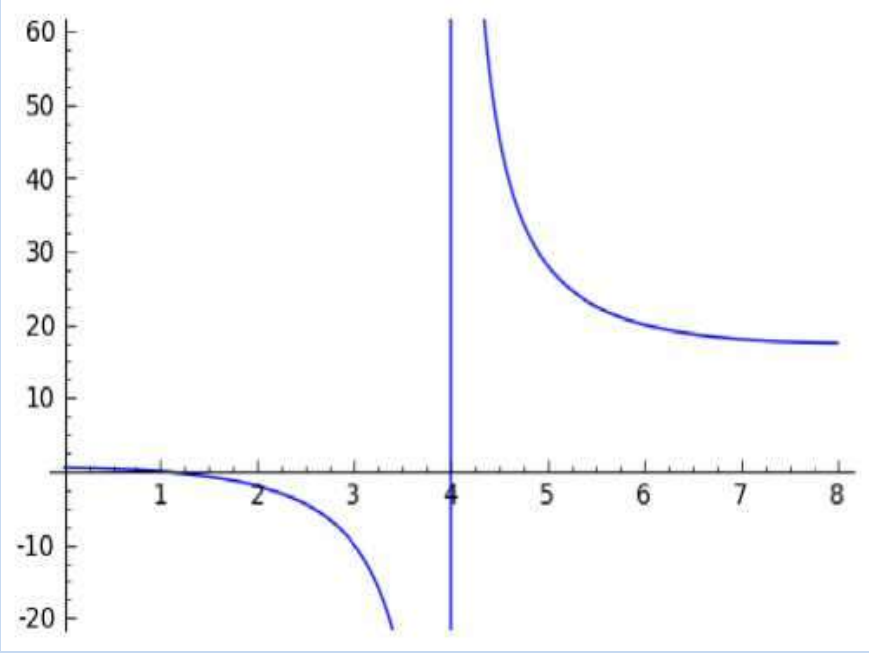
```
sage: limit(f(x), x=1)
e
sage: limit(g(x), x=2)
4*cos(4)

sage: limit(g, x=2)
x |--> 4*cos(4)

sage: limit(h(x), x = 4)
Infinity

sage: limit(h(x), x = Infinity)
+Infinity
```

```
sage: limit(h, x=4, dir="right")
x |--> +Infinity
sage: limit(h, x=4, dir="left")
x |--> -Infinity
```



Differenza tra espressioni e funzioni simboliche

```
sage: y, u = var('y, u')

sage: u = sin(x) + x*cos(y)

sage: u
x*cos(y) + sin(x)

sage: v = u.function(x, y); v
(x, y) |--> x*cos(y) + sin(x)

sage: w(x, y) = u; w
(x, y) |--> x*cos(y) + sin(x)
```

u è un'espressione simbolica

Per trasformare un'espressione simbolica in una funzione simbolica si può usare function o semplicemente assegnare l'espressione ad una funzione

```
sage: f(x) = x*exp(x); f
x |--> x*e^x
sage: fp= derivative(f,x); fp
x |--> x*e^x + e^x
sage: fp(10)
11*e^10
sage: derivative(f, x, 2)
x |--> x*e^x + 2*e^x
sage: y = var('y')
sage: derivative (f,y)
x |--> 0
```

Per derivare si usa la funzione derivative (o diff), che rende una funzione, che poi si può valutare

Nell'esempio viene calcolata la derivata seconda

Derivando rispetto ad una variabile che non compare nell'espressione della funzione si sta derivando una costante

```
sage: f(x,y)=ln(x)+3*y
sage: derivative(f,x)
(x, y) |--> 1/x
sage: derivative(f)
(x, y) |--> (1/x, 3)
sage: derivative(f)[0]
(x, y) |--> 1/x
sage: f(x,y)=(ln(x)+3*y, sin(x)+cos(y))
sage: derivative(f)
[ (x, y) |--> 1/x       (x, y) |--> 3]
[ (x, y) |--> cos(x) (x, y) |--> -sin(y)]
```

Calcolo di una derivata parziale

Calcolo del gradiente

Calcolo di una derivata parziale come componente del gradiente

Calcolo della matrice jacobiana

Assumption

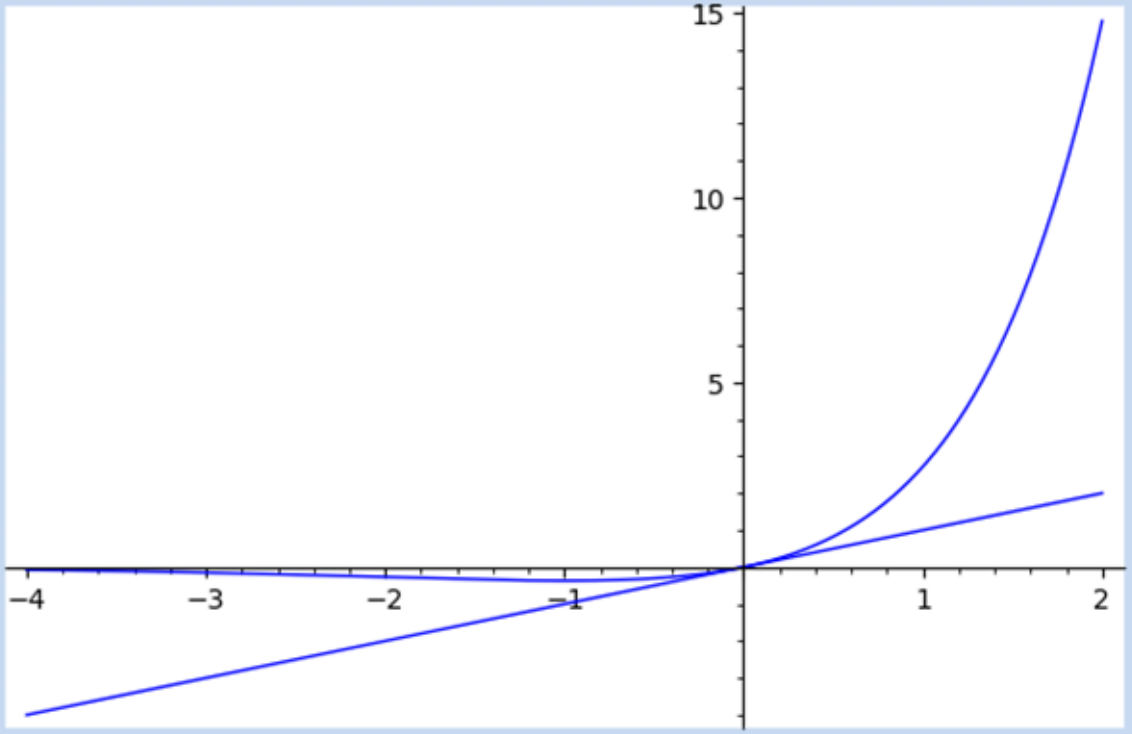
```
sage: bool(sqrt(x^2) == x)
False
sage: assume(x>0); bool(sqrt(x^2) == x)
True
sage: forget(x > 0); bool(sqrt(x^2) == x)
False
sage: n = var('n'); assume(n, 'integer');
sin(n*pi)
0
```

Le «assumption» consentono di restringere il dominio di appartenenza di una variabile

Si usa assume per introdurre una assumption, forget per eliminarla

Esempio: trovo i punti critici di f e l'equazione della tangente in (0,0)

```
sage: f(x) = x*exp(x); f
x |--> x*e^x
sage: fp= derivative(f,x); fp
x |--> x*e^x + e^x
sage: solve( fp(x) == 0, x)
[x == -1]
sage: T_f = fp(0)*( x - 0 ) + f(0); T_f
x
sage: plot(f,x,-4,2)+plot(T_f,x,-4,2)
```



```
sage: f
x |--> x*e^x
sage: integral(f,x)
 x |--> (x - 1)*e^x
sage: derivative(integral(f,x), x )
 x |--> (x - 1)*e^x + e^x
sage: integral(f, x,0,2)
e^2+1
sage: integral(f, x,0,2).n()
8.38905609893065
```

L'integrale indefinito (una primitiva) si ottiene con integral o anche con integrate

Per l'integrale definito si indicano gli estremi di integrazione come parametri di integral o di integrate

Semplificazioni

Polynomial $p = zx^2 + x^2 - (x^2 + y^2)(ax - 2by) + zy^2 + y^2$	
`p.expand()`	$-ax^3 + 2\,bx^2y - axy^2 + 2\,by^3 + x^2z + y^2z + x^2 + y^2$
`p.expand().collect(x)`	$-ax^3 - axy^2 + 2\,by^3 + (2\,by + z + 1)x^2 + y^2z + y^2$
`p.collect(x).collect(y)`	$2\,bx^2y + 2\,by^3 - (ax - z - 1)x^2 - (ax - z - 1)y^2$
`p.factor()`	$-(ax - 2\,by - z - 1)\left(x^2 + y^2\right)$
`p.factor_list()`	$\left[(ax - 2\,by - z - 1, 1), \left(x^2 + y^2, 1\right), (-1, 1)\right]$

Fraction $r = \frac{x^3+x^2y+3\,x^2+3\,xy+2\,x+2\,y}{x^3+2\,x^2+xy+2\,y}$	
`r.simplify_rational()`	$\frac{x^2+(x+1)y+x}{x^2+y}$
`r.factor()`	$\frac{(x+y)(x+1)}{x^2+y}$
`r.factor().expand()`	$\frac{x^2}{x^2+y} + \frac{xy}{x^2+y} + \frac{x}{x^2+y} + \frac{y}{x^2+y}$

Fraction $r = \frac{(x-1)x}{x^2-7} + \frac{y^2}{x^2-7} + \frac{b}{a} + \frac{c}{a} + \frac{1}{x+1}$	
`r.combine()`	$\frac{(x-1)x+y^2}{x^2-7} + \frac{b+c}{a} + \frac{1}{x+1}$

Fraction $r = \frac{1}{(x^3+1)y^2}$	
`r.partial_fraction(x)`	$\frac{-(x-2)}{3\left(x^2-x+1\right)y^2} + \frac{1}{3\,(x+1)y^2}$

Alcune delle funzioni disponibili in SageMath per semplificare espressioni

Usual mathematical functions	
Exponential and logarithm	`exp, log`
Logarithm in base a	`log(x, a)`
Trigonometric functions	`sin, cos, tan`
Inverse trigonometric functions	`arcsin, arccos, arctan`
Hyperbolic functions	`sinh, cosh, tanh`
Inverse hyperbolic functions	`arcsinh, arccosh, arctanh`
Integer part, etc.	`floor, ceil, trunc, round`
Square and n-th root	`sqrt, nth_root`

Rewriting trigonometric expressions	
Simplification	`simplify_trig`
Linearisation	`reduce_trig`
Anti-linearisation	`expand_trig`

Le principali funzioni matematiche disponibili in SageMath e le funzioni per semplificare espressioni trigonometriche

Tra le altre funzioni per semplificare espressioni, simplify() è di uso generale:

```
sage: (x^x/x).simplify()
 x^(x - 1)
```

In presenza di radici, esponenziali, logaritmi, si usa piuttosto canonicalize_radical():

```
sage: f = (e^x-1) / (1+e^(x/2)); f.canonicalize_radical()
e^(1/2*x) – 1
```

Esiste anche il commando simplify_full, che applica in sequenza 5 comandi di semplificazione, tra cui simplify_trig e simplify_rational

Alcune funzioni statistiche

```
sage: data = [1, -1, -7, 0, -4, -1, -2, 1, 3, 5, -1, 25,-5, 1,2, 0, 1, -1,-1, -1]
sage: mean(data)
```

Media

```
3/4
sage: median(data)
```

Mediana

```
-1/2
sage: mode(data)
```

Moda

```
[-1]
sage: variance(data)
```

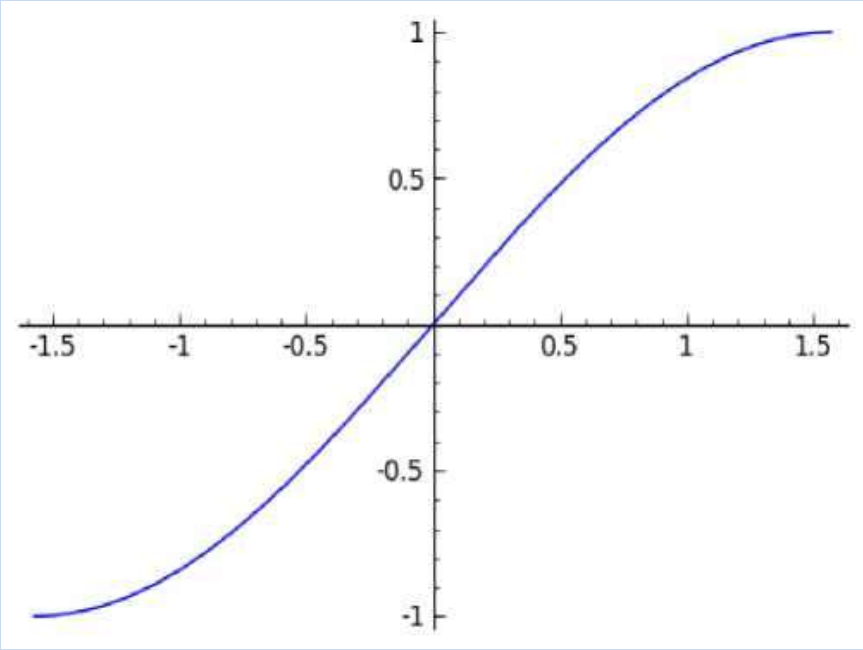
Varianza

```
 3023/76
sage: std(data) 1/2*sqrt(3023/19)
```

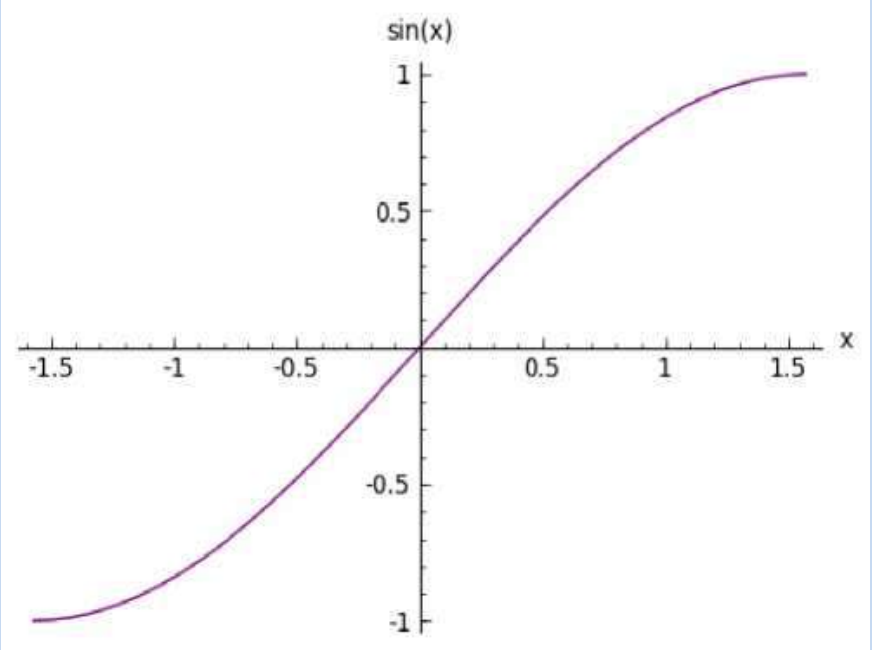
Deviazione standard

Disegnare grafici di funzioni

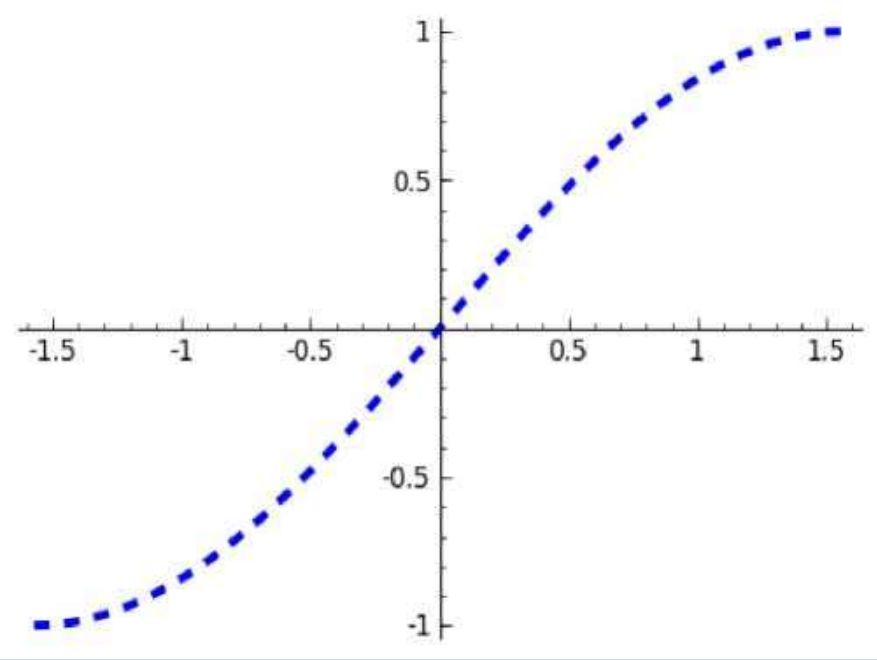
```
sage: f(x) = sin(x)
sage: p = plot(f(x), (x, -pi/2, pi/2))
sage: p.show()
```



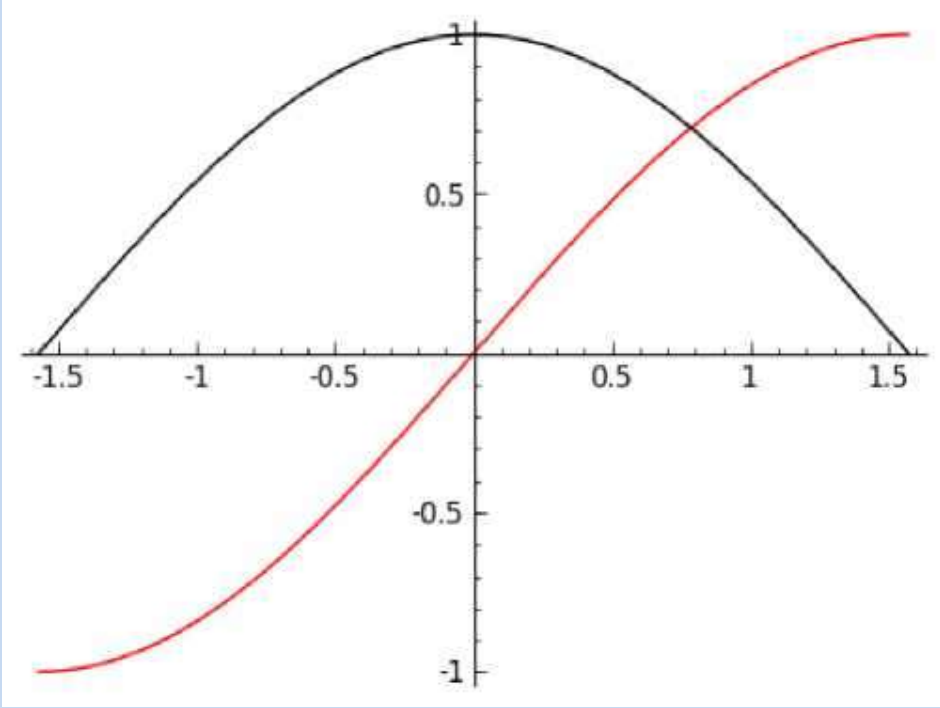

```
sage:p = plot(f(x), (x,-pi/2, pi/2), axes_labels
=['x','sin(x)'], color='purple')
sage: p.show()
```



```
sage: p = plot(f(x), (x,-pi/2, pi/2), linestyle='--', thickness=3)
sage: p.show()
```



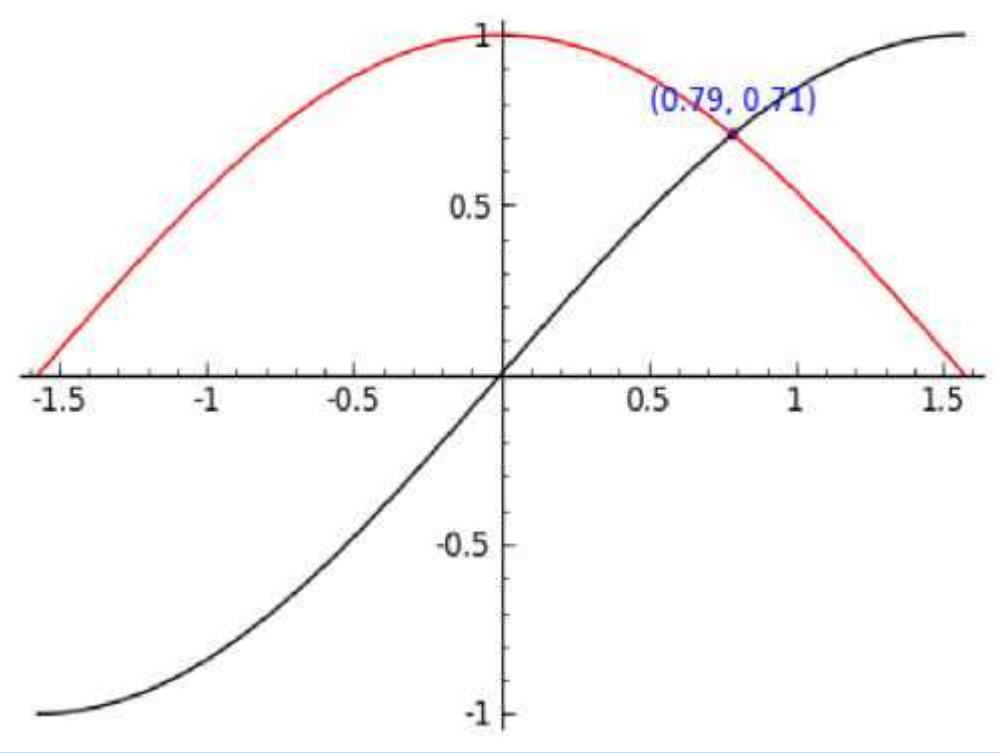
```
sage: f(x) = sin(x)
sage: g(x) = cos(x)
sage: p = plot(f(x),(x,-pi/2,pi/2), color='black')
sage: q = plot(g(x), (x,-pi/2, pi/2), color='red')
sage: r = p + q
sage: r.show()
```



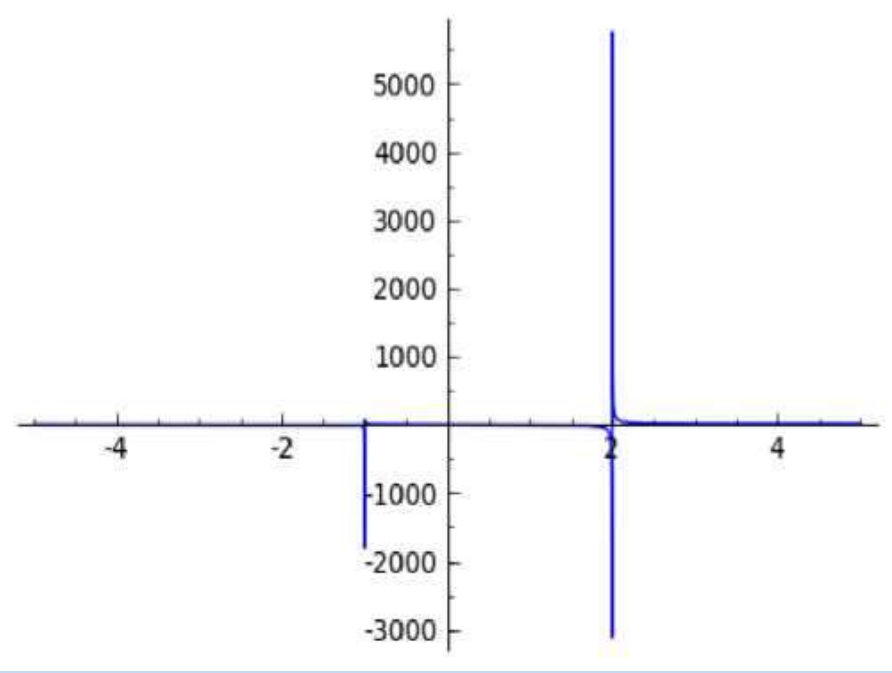
```
sage: find_root( sin(x) == cos(x),-pi/2, pi/2 )
0.78539816339744839
sage: P = point( [(0.78539816339744839, sin(0.78539816339744839))] )
sage: T = text("(0.79,0.71)", (0.78539816339744839, sin(0.78539816339744839) + .10))
sage: s = P + r + T
sage: s.show()
```

Per definire un punto solo non occorre la struttura di lista

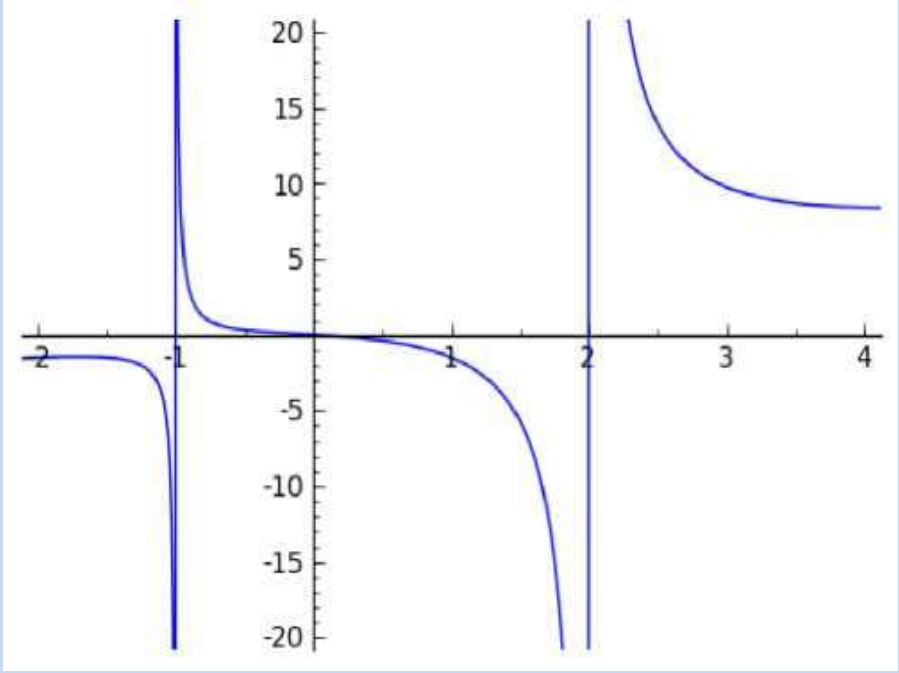
Posizione del testo



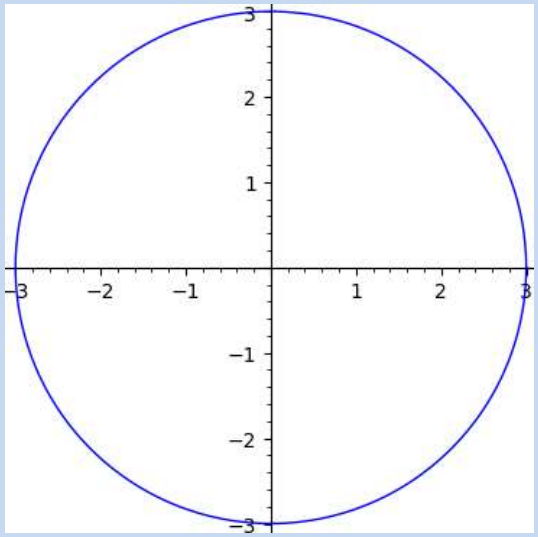
```
sage: f(x) = (x^3 + x^2 + x)/(x^2 - x -2 )
sage: p = plot(f(x), (x, -5,5))
sage: p.show()
```




```
sage: p.show(xmin=-2, xmax=4,
 ymin=-20, ymax=20)
```



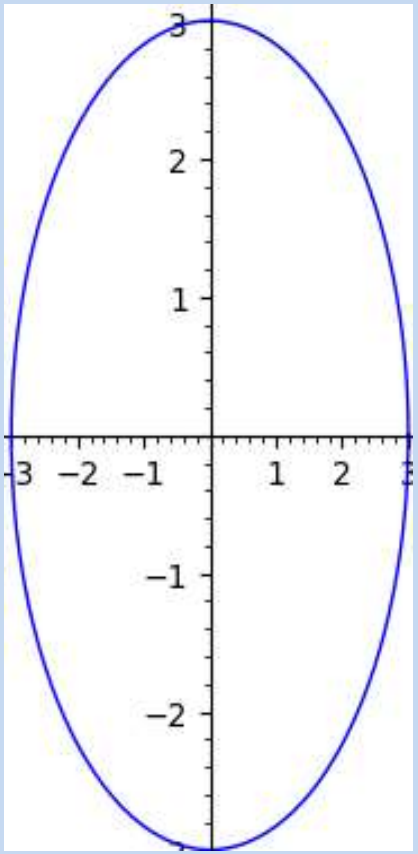
```
sage: t = var('t')
sage:p=parametric_plot( [3*cos(t), 3*sin(t)], (t, 0, 2*pi) )
sage: p.show()
```



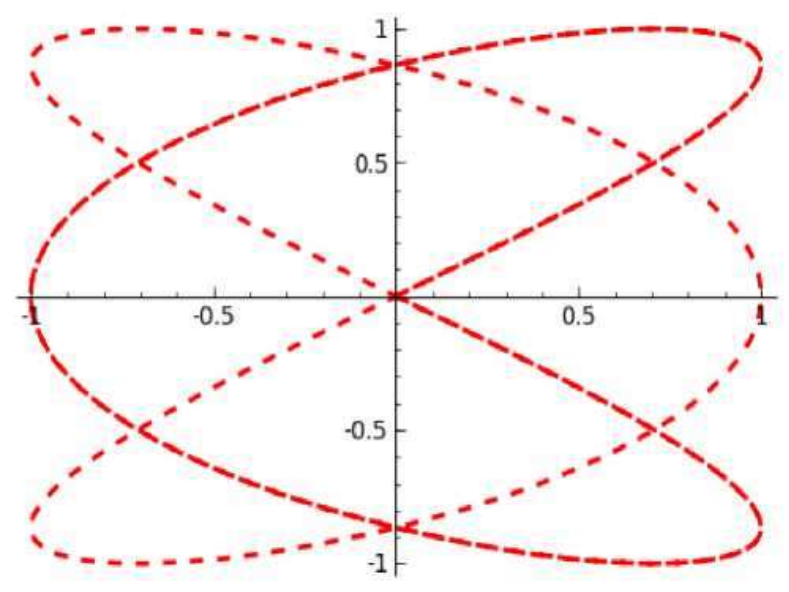
NB: Opzioni utili di show sono:

- aspect_ratio (1 per avere la stessa scala sui due assi)
- figsize (per default = 4) che modifica il rapporto altezza/larghezza della figura
- xmin, xmax, ymin, ymax per i valori minimi/massimi sugli assi

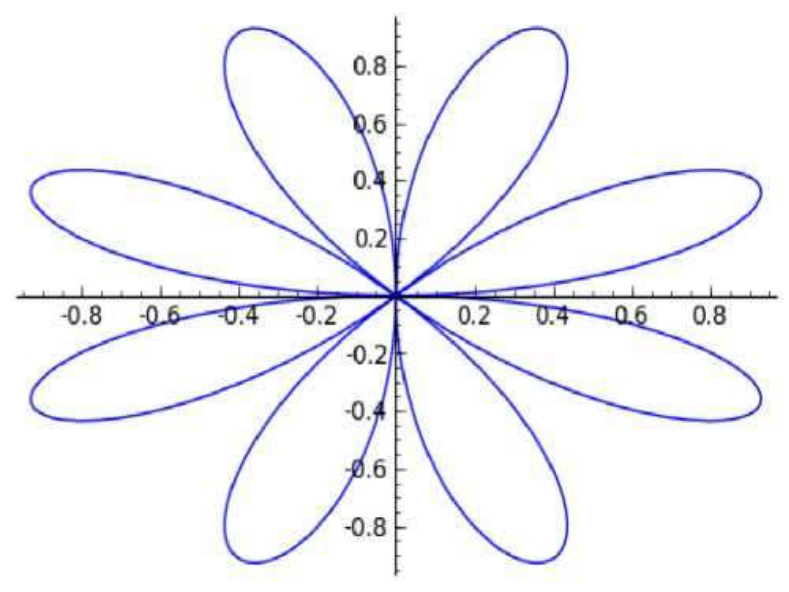
```
sage: p.show(aspect_ratio=2)
```



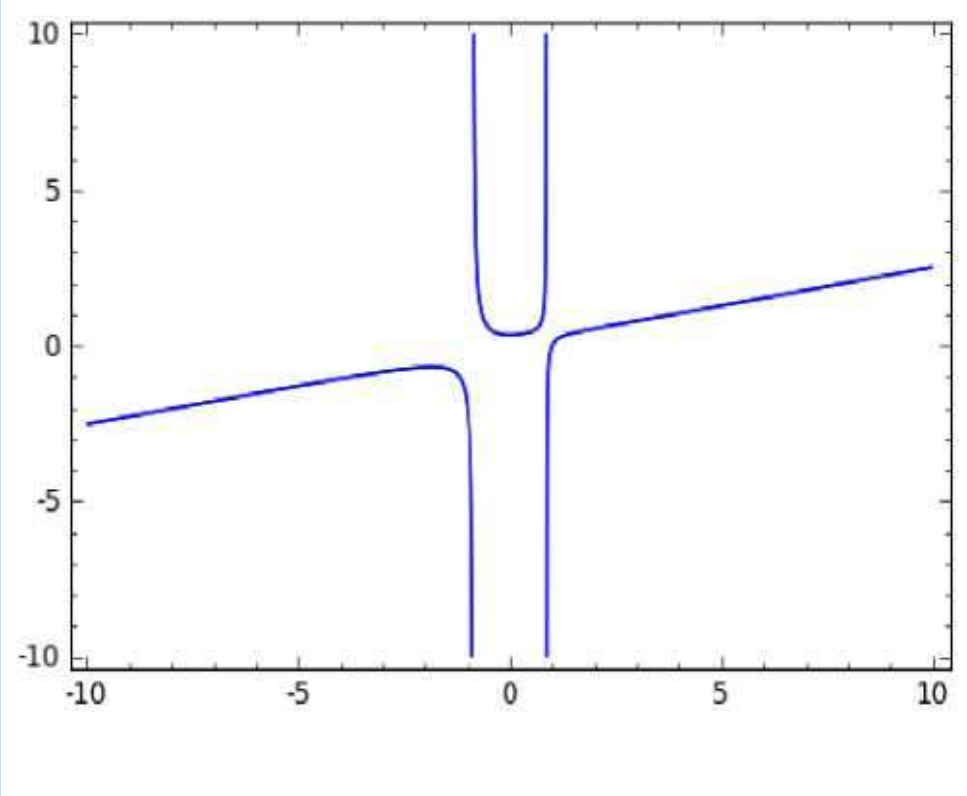
```
sage: p = parametric_plot( [sin(3*t), sin(2*t)],
(t, 0, 3*pi), thickness=2, color='red',
linestyle="--")
sage: p.show()
```



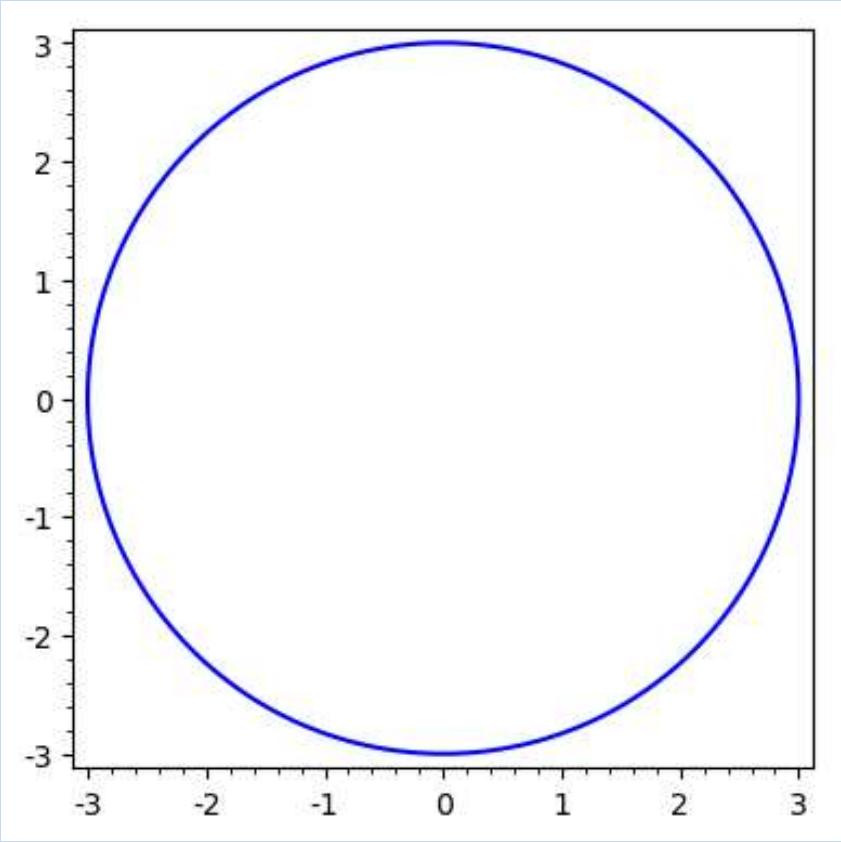

```
sage: theta = var("theta")
sage: r(theta) = sin(4*theta)
sage: p = polar_plot((r(theta)),
(theta, 0, 2*pi) )
sage: p.show()
```



```
sage: implicit_plot(4*x^2*y - 3*y == x^3 - 1, (x,-10,10),(y,-10,10))
```



```
sage: implicit_plot(x^2 + y^2 == 9, (x,-3,3),(y,-3,3))
```



```
sage: x,y = var("x y")
sage: f(x,y) = x^2 – y^2
sage: p = plot3d(f(x,y), (x,-5,5), (y,-5,5))
sage: p.show(frame = false)
```