Laboratory 1: Introduction to SAGE

Computation in SAGE

Basic arithmetic operations

```
“four operations”     a+b, a-b, a*b, a/b

power                 a^b or a**b

square root           sqrt(a)

n-th root             a^(1/n)
```

In the notebook, you can enter the commands in a computation cell, and the result is obtained by clicking on Run button or using the Shift+Enter key combination. The combination Alt+Enter not only executes the command of the current cell, but also creates a new cell just below.

In [1]:
```
1+2-3
```
Out[1]: 0

In [2]:
```
2*3/7+3^2
```
Out[2]: 69/7

In [3]:
```
(1 + 2*(3 + 5^2))*sqrt(9)
```
Out[3]: 171

To obtain a numerical approximation, one simply writes one of the numbers with a decimal point

In [4]:
```
15/6
```
Out[4]: 5/2

In [5]:
```
15./6
```
Out[5]: 2.50000000000000

Also, the numerical_approx function gives a numerical approximation of an expression

In [6]:
```
numerical_approx(15/6)
```
Out[6]: 2.50000000000000

In [7]:
```
numerical_approx(44/13, digits=60)
```
Out[7]: 3.38461538461538461538461538461538461538461538461538461538462

Integer operations

integer division	a // b
remainder	a % b
quotient and remainder	divmod(a,b)
factorial n!	factorial(n)
binomial coefficient	binomial(n,k)

Usual mathematical functions

integer part	floor(a)
absolute value, modulus	abs(a)
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

Symbolic Variables

The symbolic variables should be explicitly declared before being used (SR abbreviates Symbolic Ring):

In [8]:
```
x = SR.var('x')
p1=(x+1)^2
p1
```
Out[8]: (x + 1)^2

If we need to expand the expression we use the command variable.expand()

In [9]:
```
p1.expand()
```
Out[9]:
```
x^2 + 2*x + 1
```

In [10]:
```
p2=x^2 + 2*x + 1
p2
```
Out[10]:
```
x^2 + 2*x + 1
```

The factor decomposition of a given expression is obtained by variable.factor()

In [11]:
```
p2.factor()
```
Out[11]:
```
(x + 1)^2
```

In [12]:
```
p3=x^2-5*x + 6
p3.factor()
```
Out[12]:
```
(x - 2)*(x - 3)
```

The subs method is used in the case when we need to evaluate an expression by giving a value to some of its parameters.

In [13]:
```
p1.subs(x=-1)
```
Out[13]:
```
0
```

In [14]:
```
p3.subs(x=2)
```
Out[14]:
```
0
```

In [15]:
```
p3.subs(x=-3)
```
Out[15]:
```
30
```

The command var('x') is a convenient alternative for x = SR.var('x')

In [16]:
```
x=var('x')
p=(2*x-1)^3
p.expand()
```
Out[16]:
```
8*x^3 - 12*x^2 + 6*x - 1
```

In [17]:
```
x,y=var('x,y')
p=(2*x+y)^2
p.expand()
```
Out[17]:
```
4*x^2 + 4*x*y + y^2
```

In [18]:
```
p.subs(x=1,y=1)
```
Out[18]:
```
9
```

Equations

An equation is defined using double equality sign, f(x)==g(x).

The most used commands to solve an equation are:

```
Symbolic solution             solve

Roots (with multiplicities)   roots

Numerical solving             find_root
```

In [19]:
```
x=var('x')
eq1=x^2+x+1==0
solve(eq1,x)
```

Out[19]: `[x == -1/2*I*sqrt(3) - 1/2, x == 1/2*I*sqrt(3) - 1/2]`

Not all equations can be solved by Sage, in the case of the next example Sage does not return any solution

In [20]:
```
eq2=exp(-x)==x
solve(eq2,x)
```

Out[20]: `[x == e^(-x)]`

If we want to find a numerical solution we can use find_root(equation,a,b), which determine a solution in the interval [a,b]

In [21]:
```
find_root(eq2,0,2)
```

Out[21]: `0.5671432904098384`

The solve command can also solve systems of equations, these can be defined in Sage using [ ]

[eq1,eq2,...,eqn]

In [22]:
```
x,y=var('x,y')
syst=[x+2*y==1,x-y==3]
solve(syst,x,y)
```

Out[22]: `[[x == (7/3), y == (-2/3)]]`

Limits

To determine a limit, we use the limit command

In [23]:
```
n,x=var('n,x')
```

In [24]:
```
limit(1/n,n=infinity)
```
Out[24]: 0

In [25]:
```
limit(sin(x)/x,x=0)
```
Out[25]: 1

In [26]:
```
limit(1/x, x=0)
```
Out[26]: Infinity

The last output says that one of the limits to the left or to the right is infinite. To know more about this, we study the limits to the left (minus) and to the right (plus), with the dir option

In [27]:
```
limit(1/x, x=0,dir='minus')
```
Out[27]: -Infinity

In [28]:
```
limit(1/x, x=0,dir='plus')
```
Out[28]: +Infinity

Useful functions in analysis:

Derivative	diff(f(x), x)
n-th derivative	diff(f(x), x, n)
Antiderivative	integrate(f(x), x)
Numerical integration	integral_numerical(f(x), a, b)
Symbolic summation	sum(f(i), i, imin, imax)
Limit	limit(f(x), x=a)
Taylor expansion	taylor(f(x), x, a, n)
Power series expansion	f.series(x==a, n)

Derivatives

In [29]:
```
f(x)=exp(x^2)+3
```

In [30]:
```
diff(f(x),x,2)
```
Out[30]: 4*x^2*e^(x^2) + 2*e^(x^2)

In [31]:
```
diff(f(x),x,3)
```
Out[31]:
```
8*x^3*e^(x^2) + 12*x*e^(x^2)
```

Integrals

In [32]:
```
x=var('x')
```
In [33]:
```
integrate(cos(x),x)
```
Out[33]:
```
sin(x)
```
In [34]:
```
f(x)=exp(x)*cos(x)
integrate(f(x),x)
```
Out[34]:
```
1/2*(cos(x) + sin(x))*e^x
```

For definite integral we use integrate(f(x),x,a,b)

In [35]:
```
integrate(cos(x),x,0,pi/2)
```
Out[35]:
```
1
```

Not alwaysSage can compute the definite integral value

In [36]:
```
integrate(sin(sqrt(1 - x^3)), x, 0,1)
```
Out[36]:
```
integrate(sin(sqrt(-x^3 + 1)), x, 0, 1)
```

To compute numerically an integral on an interval, we use the integral_numerical function, which returns a pair, the first value is the approximation of the integral, the second value is an estimate of the corresponding error:

In [37]:
```
integral_numerical(sin(sqrt(1 - x^3)), 0, 1)
```
Out[37]:
```
(0.7315380084233594, 3.953379981670976e-07)
```

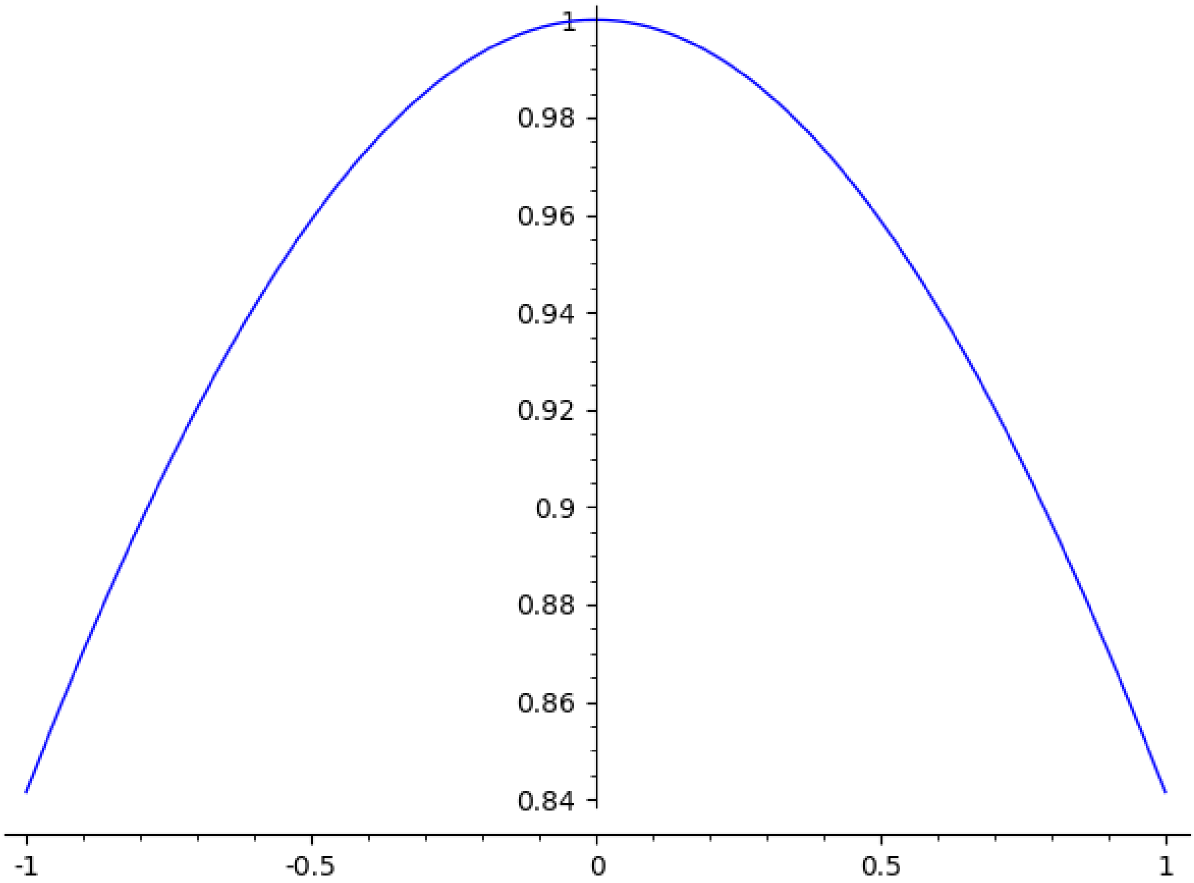
2D Graphics

Graphical Representation of a Function

For a graphical representation of a function f(x) on an interval [a, b], we use plot(f(x), a, b) or the alternative syntax plot(f(x), x, a, b).

In [38]:
```
f(x)=sin(x)/x
plot(f(x),-1,1)
```

Out[38]:



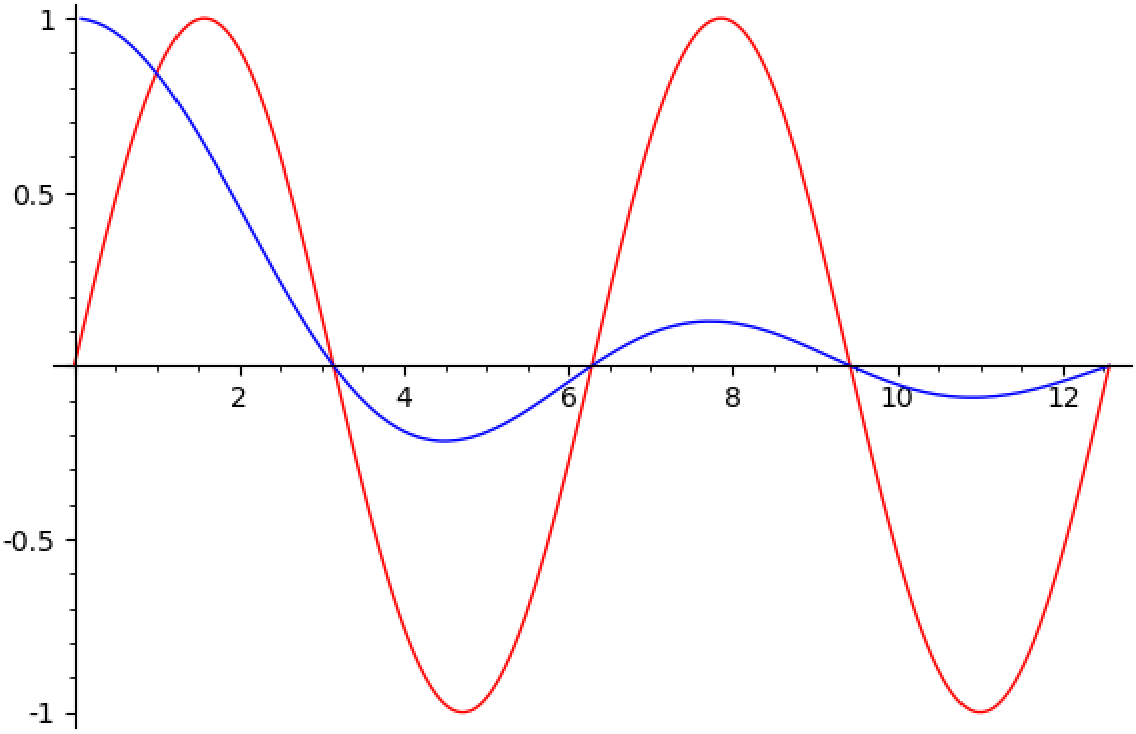
The plot command has many options, most important are:

```
plot_points (default value 200): minimal number of computed points;

xmin and xmax: interval bounds over which the function is displayed;

color: colour of the graph, either a RGB triple, a character string such as
'blue', or an HTML colour like '#aaff0b';

detect_poles (default value False): enables to draw a vertical asymptote at
poles of the function;

alpha: line transparency;

thickness: line thickness;

linestyle: style of the line, either dotted with ':', dash-dotted with '-.', or
solid with the default value '-'.
```

We can plot more than one function in the same window specifying the functions list [f1(x),f2(x),...,fn(x)] and the corresponding color list ['color_1','color_2','color_n']:

In [39]:
```
plot([sin(x),f(x)],0,4*pi,color=['red','blue'])
```

Out[39]:



If the functions list is larger, we index the list and we use for command to generate it. For example, for the function f_n(x)=x/(1+x^2)^n if we want to generate the graphical representations of the functions f_1(x),...,f_10(x) first we construct the list and then the graphs using plot command:

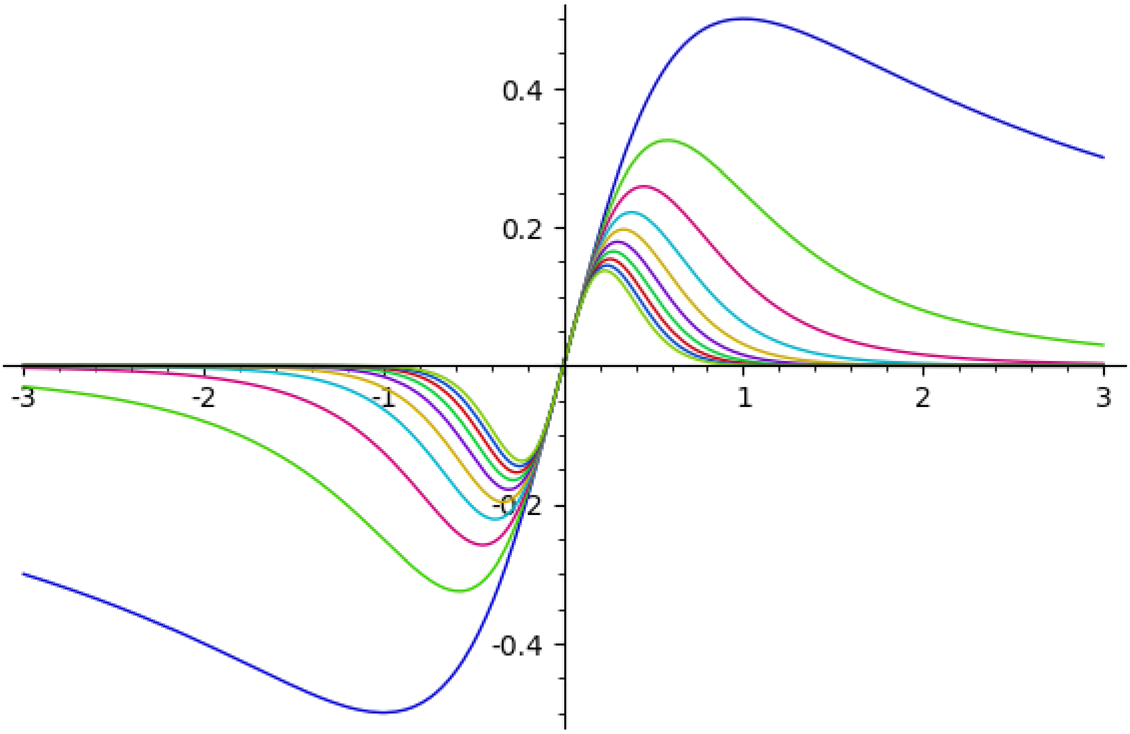
In [40]:
```
x,n=var('x,n')
f(x,n)=x/(1+x^2)^n
```

In [41]:
```
f_list=[f(x,n) for n in [1..10]]
f_list
```

Out[41]:
```
[x/(x^2 + 1),
 x/(x^2 + 1)^2,
 x/(x^2 + 1)^3,
 x/(x^2 + 1)^4,
 x/(x^2 + 1)^5,
 x/(x^2 + 1)^6,
 x/(x^2 + 1)^7,
 x/(x^2 + 1)^8,
 x/(x^2 + 1)^9,
 x/(x^2 + 1)^10]
```

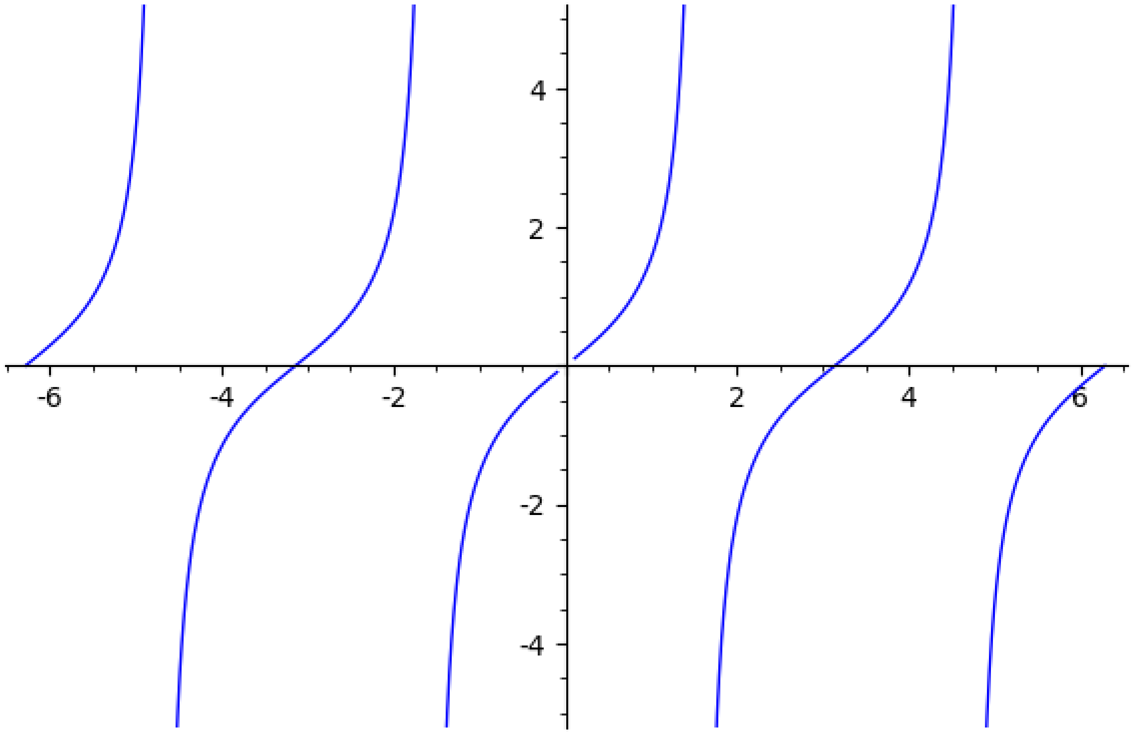
In [42]:
```
plot(f_list,-3,3)
```

Out[42]:



In the case of discontinuous points we need to use the option detect_poles=True and we can give bounds for the y-axis specifying the values for ymin and ymax

In [43]:
```
plot(tan(x), -2*pi,2*pi,detect_poles=True,ymin=-5,ymax=5)
```
Out[43]:



Parametric curve

If a curve is given in a parametric form

x(t)=f(t), y(t)=g(t), t in [a,b]

we use the command parametric_plot((f(t), g(t)), (t, a, b)).

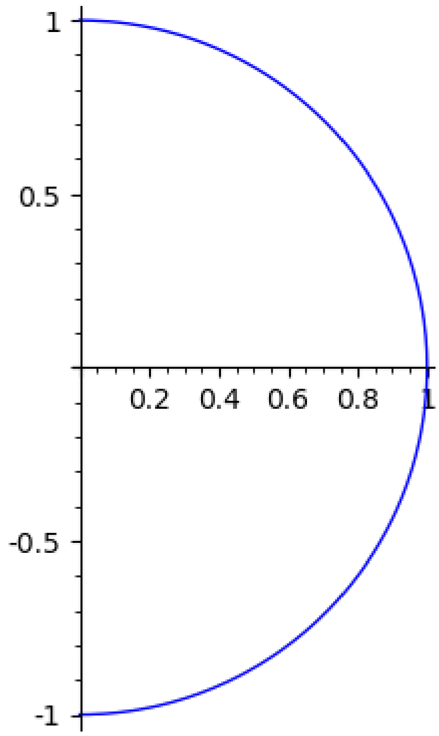
For example in the case of semicircle defined by

x(t)=cos(t), y(t)=sin(t), t in [-pi/2,b]

we have:

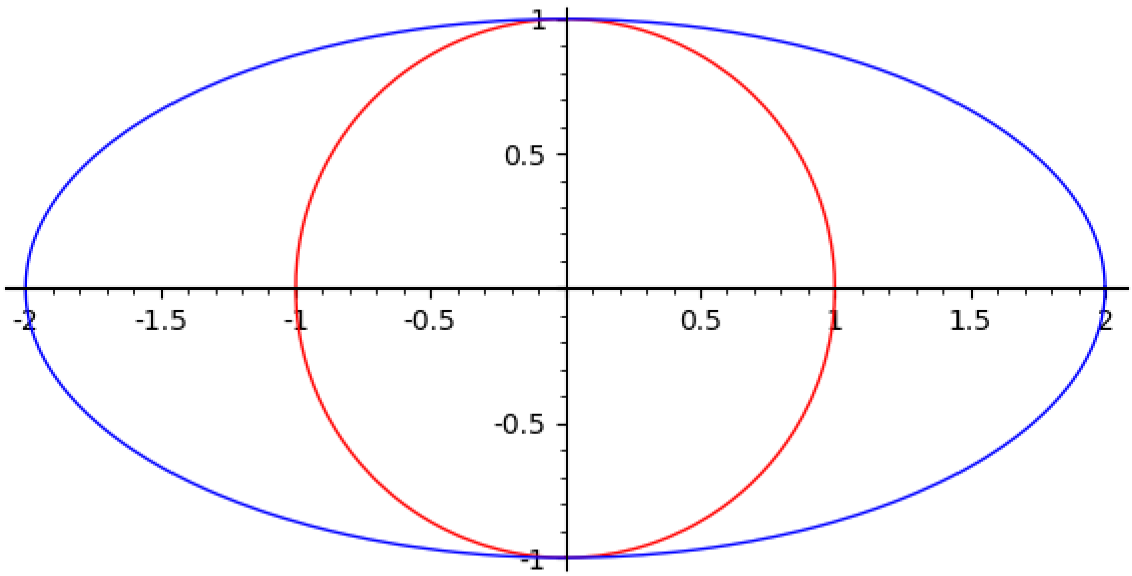
In [44]:
```
t=var('t')
x(t)=cos(t)
y(t)=sin(t)
parametric_plot((x(t), y(t)), (t, -pi/2, pi/2))
```

Out[44]:



In order to represent in the same graph several curves given in parametric we assing each graph to a variable, we combine them by adding and we use show command:

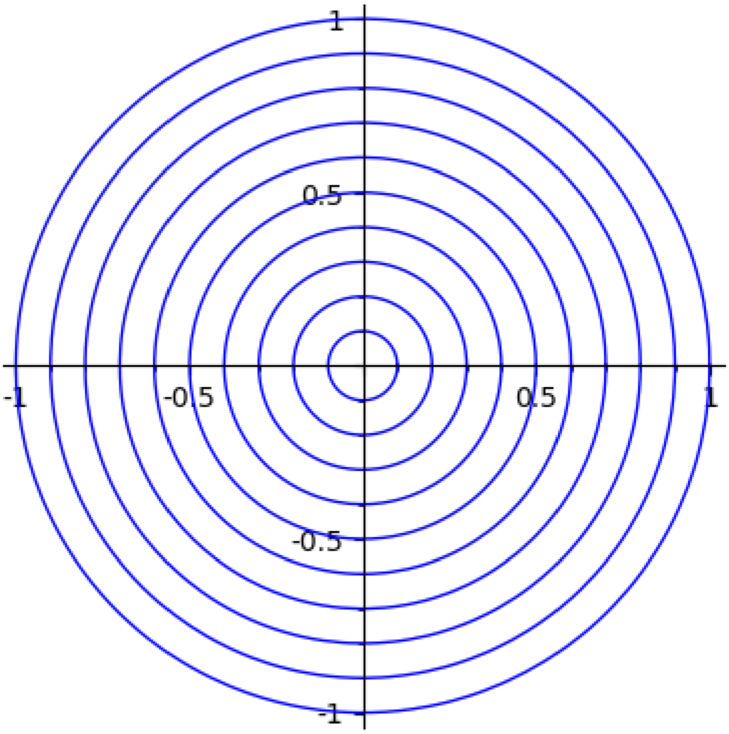
In [45]:
```
g1=parametric_plot((x(t), y(t)), (t, -pi/2, 3*pi/2),color='red')
g2=parametric_plot((2*x(t), y(t)), (t, 0, 2*pi),color='blue')
show(g1+g2)
```



When the list of parametric curves is large we can use "for" command to generate the curves list and we combine them by adding, for example let's plot the family of circles x(t)=r * cos(t) y(t)=r * sin(t), t in [0 , 2 * pi]

for r=0.1, 0.2, ..., 1

In [46]:
```
g=parametric_plot((1/10*cos(t),1/10*sin(t)),(t,0,2*pi))
for k in [2..10]:
    g1=parametric_plot((k/10*cos(t),k/10*sin(t)),(t,0,2*pi))
    g=g+g1
show(g)
```



Curves in implicit form

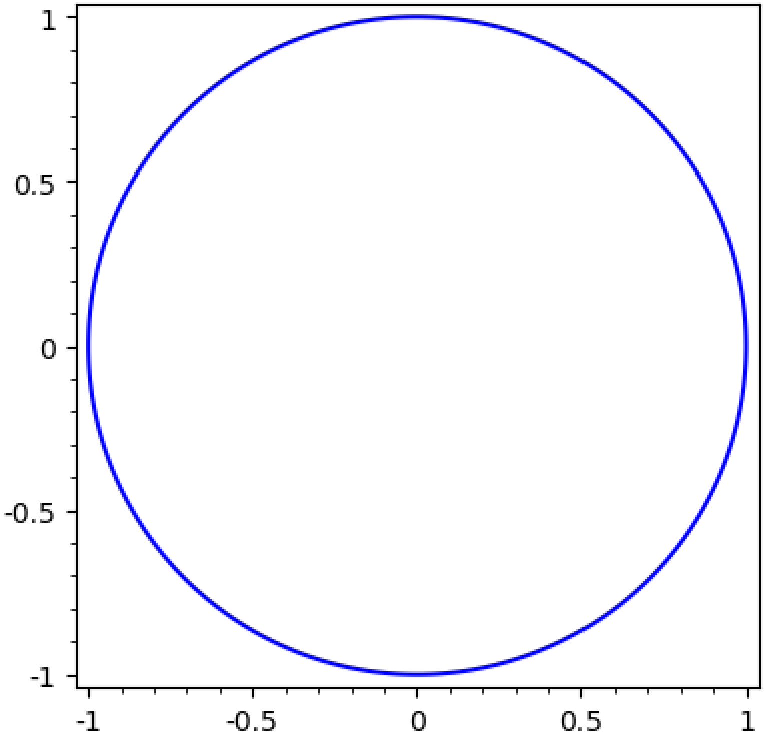
To draw a curve given by an implicit equation, you need to call the function

implicit_plot(f(x, y), (x, a, b), (y, c, d));

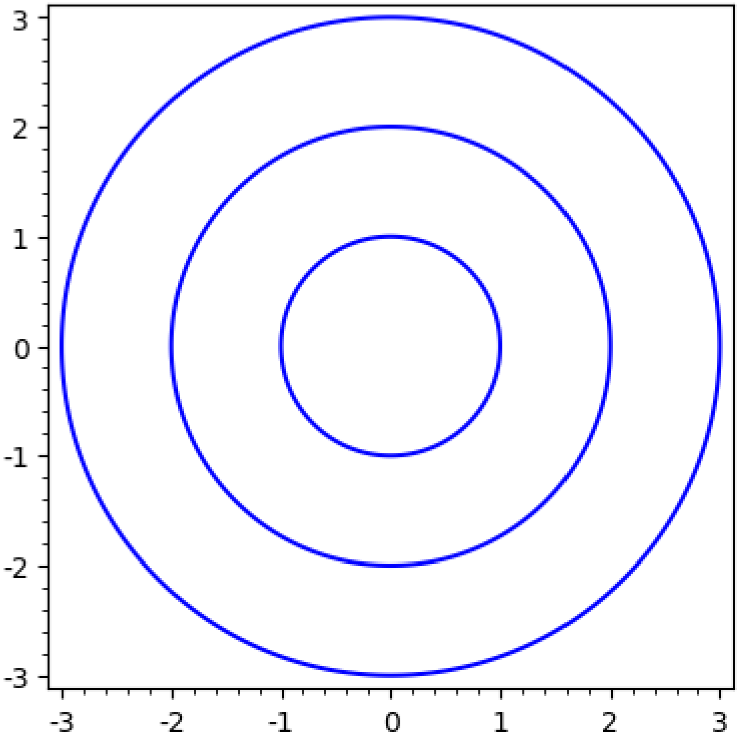
In [47]:

```
x,y=var('x,y')
f(x,y)=x^2+y^2
implicit_plot(f(x,y)==1,(x,-1,1),(y,-1,1))
```

Out[47]:



In [48]:
```
g1=implicit_plot(f(x,y)==1,(x,-1,1),(y,-1,1))
g2=implicit_plot(f(x,y)==4,(x,-2,2),(y,-2,2))
g3=implicit_plot(f(x,y)==9,(x,-3,3),(y,-3,3))
show(g1+g2+g3)
```



In [ ]:
```
```