Solving Differential Equations with Sage

Define and solve a first order differential equation. The command desolve

The command desolve is used to find the general solution of a scalar differential equation. Also, it can be used to find the solution of an IVP or of a problem formed by a differential equation and any other type of conditions (as we can see in the following paragraph).

First we show how to introduce a differential equation in SAGE. Note that, first we have to clarify the notations for the unknowns and, respectively, for the independent variable. An important thing to remember is that we have to write each time the independent variable.

The first order differential equation x'=x , where the unknown is the function x of independent variable t, can be introduced in Sage as follows:

x'(t)= x(t)

In [8]:
```
t=var('t')
x=function('x')(t)
eqd=diff(x,t)==x
desolve(eqd,x)
```

Out[8]:
```
_C*e^t
```

The solving command in Sage is desolve(equation, variable, ics = ..., ivar = ..., show_method = ..., contrib_ode = ...) where: equation is the differential equation. Equality is designated by ==; variable is the dependent variable, i.e., x as x(t); ics is optional and stands for initial conditions. For a first-order equation, write [t0,x0] and for a second-order equation write [t0,x0,t1,x1] or [t0,x0,x0']; ivar is optional and stands for the independent variable, i.e., x in x(t). It must be specified if there is more than one independent variable or parameters as in x; show_method is an optional boolean set to false. If true, then Sage returns a pair "[solution, method]", where method is the string describing the method which has been used to get a solution. The method can be one of the following: linear, separable, exact, homogeneous, bernoulli, generalized homogeneous; contrib_ode is an optional boolean set to false. If true, desolve allows to solve Clairaut, Lagrange, Riccati and some other equations. This may take a long time and is thus turned off by default.

If we want to see the used method for finding solution we can add the option show_method = True. The answer is given as a list, first component is the solution expression, the second component is the used method to find it.

In [9]:
```
desolve(eqd,x,show_method=True)
```

Out[9]:
```
[_C*e^t, 'linear']
```

Define and solve a second order differential equation.

A second order differential equation is defined in same way as we did in the case of a first order differential equation, where the unknown is the function x of independent variable t, in addition the term of x'' will appear in this case.

Let us consider the second order differential equation: x''+5x'-6x=0.

In order to find the general solution of of a second order differential equations we use desolve. Notice that the general solution evidentiate as many arbitrary real constants, as many as the order of the differential equation.

In [10]:
```
t=var('t')
x=function('x')(t)
eqd=diff(x,t,2)+5*diff(x,t)-6*x==0
desolve(eqd,x)
```

Out[10]:
```
_K2*e^(-6*t) + _K1*e^t
```

Solving Initial Value Problems. Ploting solution of Initial Value Problem

Initial Value Problems for a first order differential equation

In order to obtain the solution of a problem consisting of a differential equations and one or more conditions (like, for example, an IVP) we use again the command desolve. The solving command to find the solution for a given IVP for a first order differential equation is: desolve(equation, variable, ics = ...) where: ics stands for initial condition. For a first-order equation, write ics=[t0,x0] for a given initial condition x(t0)=x0

For example, if we want to obtain the solution of the following IVP x'=x x(0)=-3 we have:

In [20]:
```
t=var('t')
x=function('x')(t)
eqd=diff(x,t)==x
desolve(eqd,x,ics=[0,-3])
```

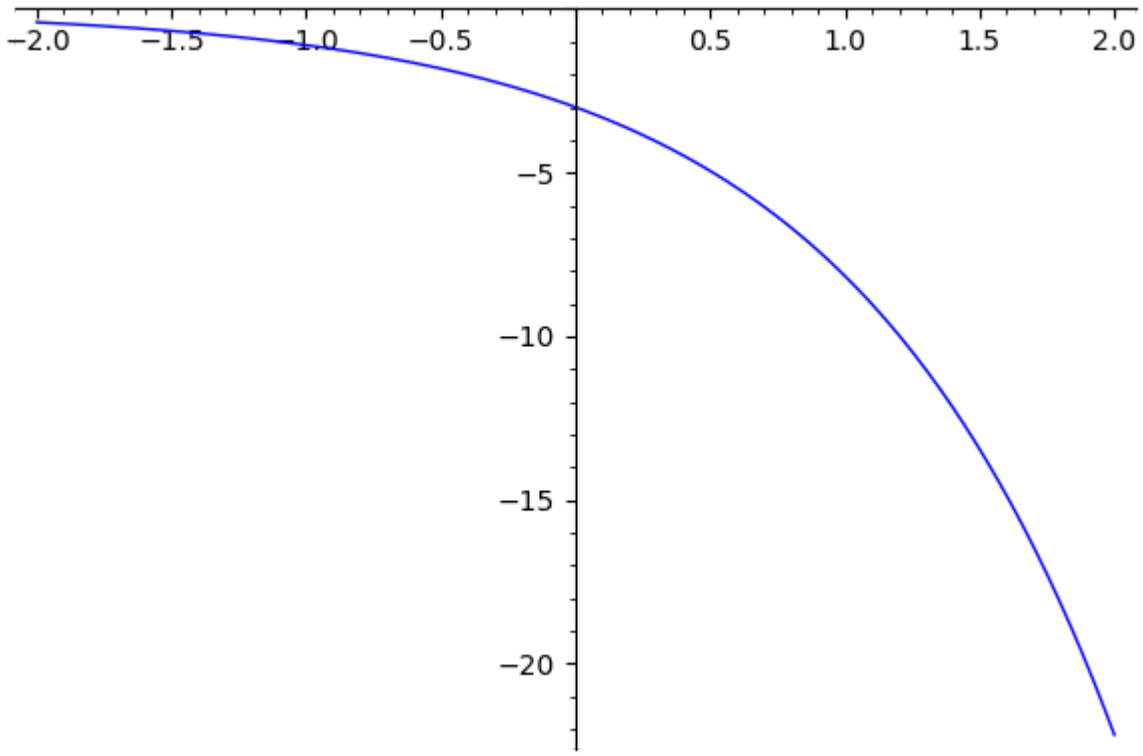
Out[20]:
```
-3*e^t
```

In [12]:
```
sol=desolve(eqd,x,ics=[0,-3])
sol
```

Out[12]:
```
-3*e^t
```

In [13]:
```
plot(sol,t,-2,2)
```

Out[13]:



In the case when the differential equation depends on some parameter it is interesting to study the solution dependence with respect to that parameter. For example, let's study the dependence of the IVP solution

x'=k*x

x(0)=1

with respect to the parameter k.

Remark: When the differential equation depends on some parameter in desolve command we have to specify the list of variable, i.e, if we want to find the solution x=x(t) we use the desolve command as follows desolve(eqd, [x,t])

In [21]:
```
t,k=var('t,k')
x=function('x')(t)
eqd=diff(x,t)==k*x
desolve(eqd,[x,t],ics=[0,1])
```

Out[21]:
```
e^(k*t)
```

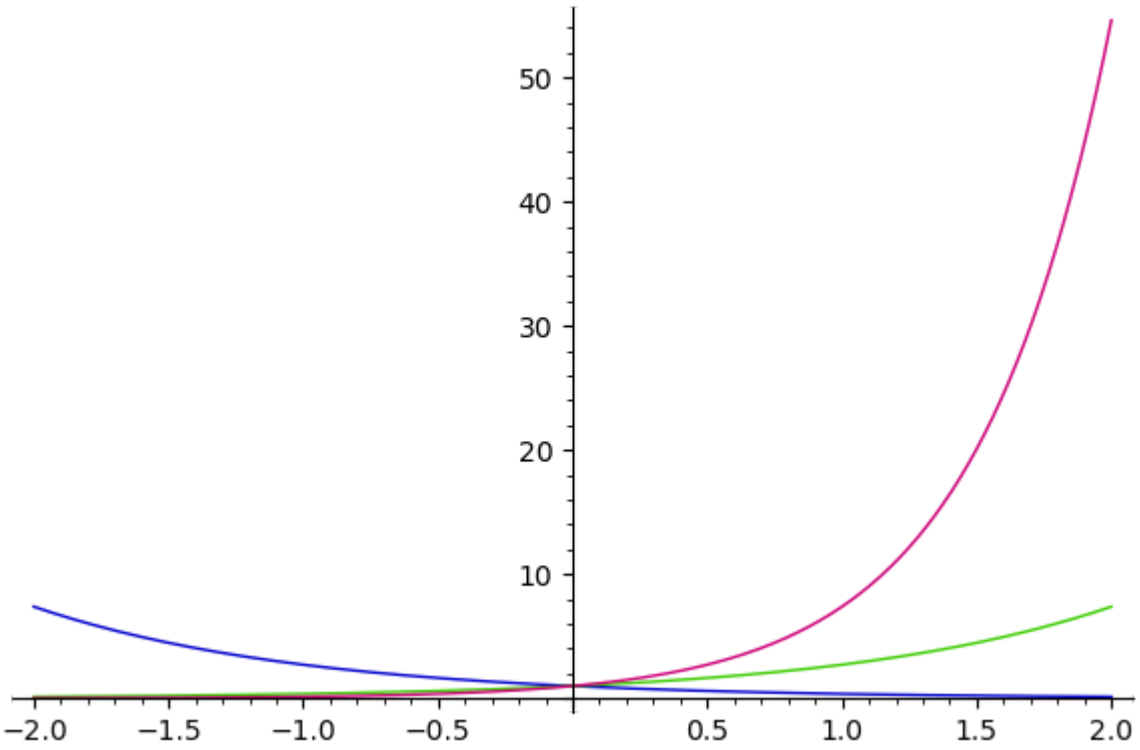
In [22]:
```
sol(t,k)=desolve(eqd,[x,t],ics=[0,1])
sol(t,k)
```

Out[22]:
```
e^(k*t)
```

To study the IVP solution dependence with respect to the parameter k means to plot solutions for defferent values of k

In [23]:
```
plot([sol(t,-1),sol(t,1),sol(t,2)],t,-2,2)
```

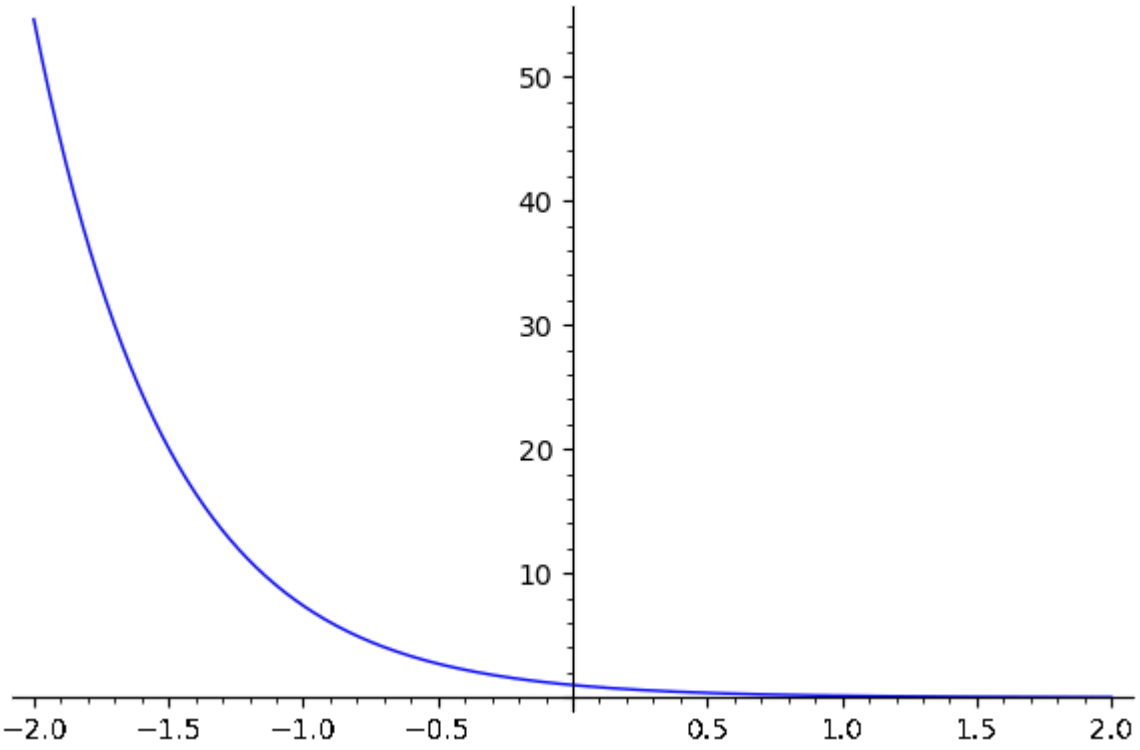
Out[23]:



Using animate command we can see how solution is changing when the value of the parameter k is changing

In [25]:
```
sols=[plot(sol(t,k/20),t,-2,2,xmin=0,xmax=50) for k in [-40..40]]
animate(sols)
```

Out[25]:



Initial Value Problems for a second order differential equation

The solving command to find the solution for a given IVP for a second order differential equation is: desolve(equation, variable, ics = ...) where: ics=[t0,x0,x1] for a given initial condition x(t0)=x0 x'(t0)=x1

Let's find the solution of the IVP:

x''+5x'-6x=0

x(0)=0

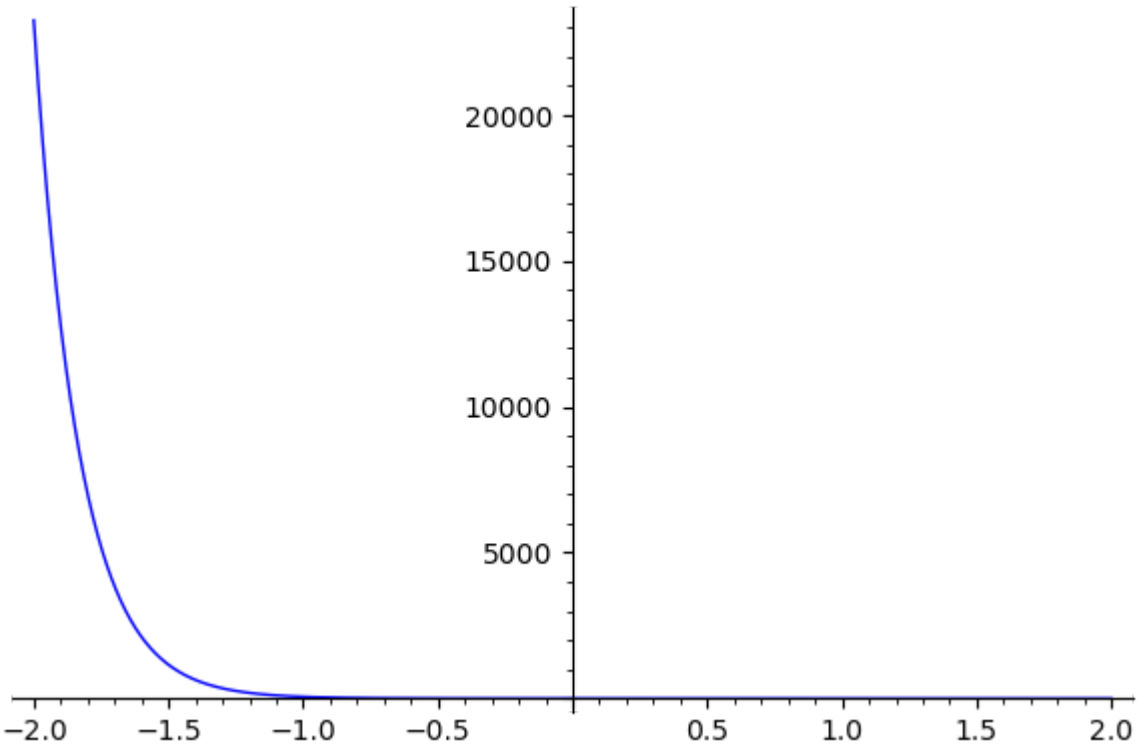
x'(0)=-1

In [19]:
```
t=var('t')
x=function('x')(t)
eqd=diff(x,t,2)+5*diff(x,t)-6*x==0
desolve(eqd,x,ics=[0,0,-1])
```

Out[19]:
```
1/7*e^(-6*t) - 1/7*e^t
```

In [73]:
```
sol=desolve(eqd,x,ics=[0,0,-1])
plot(sol,t,-2,2)
```

Out[73]:



The command piecewise

Here we show how to define in Sage a piecewise-valued function. The command is f=piecewise(...) See the below example.

In [82]:
```
f = piecewise([((0,1), x^3), ([-1,0], -x^2)]);
f
```

Out[82]:
```
piecewise(x|-->x^3 on (0, 1), x|-->-x^2 on [-1, 0]; x)
```

In [83]:
```
2*f
```

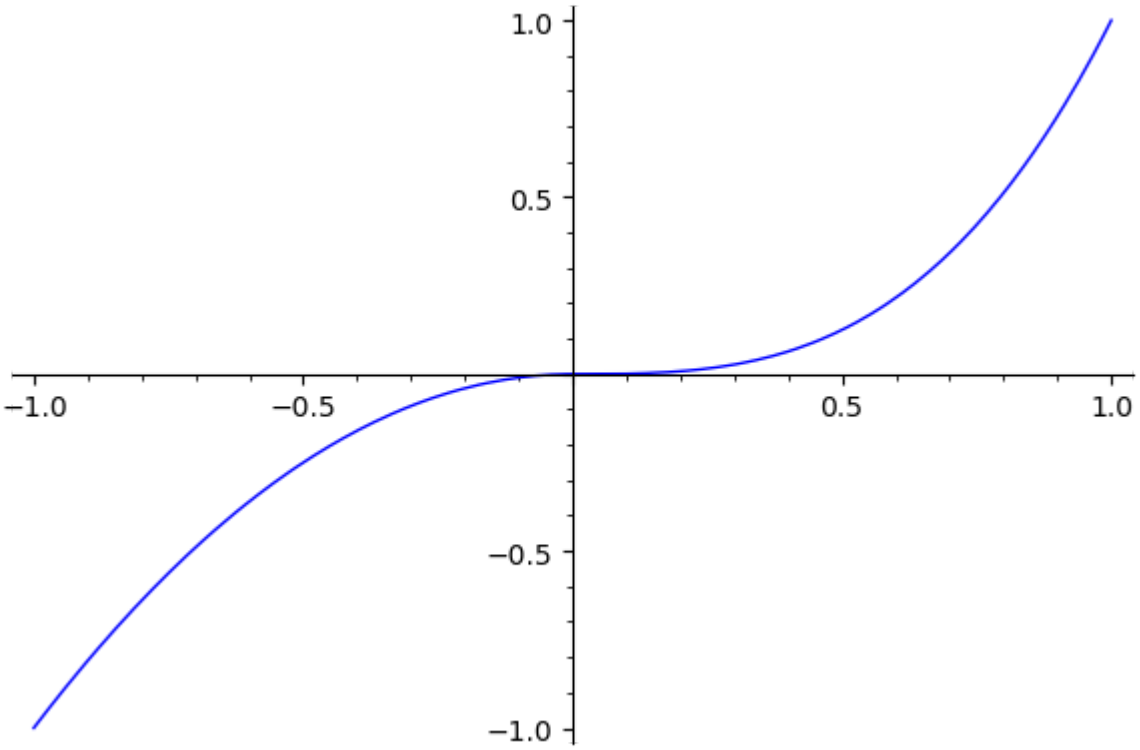
Out[83]:
```
2*piecewise(x|-->x^3 on (0, 1), x|-->-x^2 on [-1, 0]; x)
```

In [84]:
```
f(x=1/2)
```

Out[84]:
```
1/8
```

In [85]:
```
plot(f)
```

Out[85]:



In [ ]:
```

```