nota_lezione_2

January 7, 2021

1 2nd lecture on python and sympy

In this lecture we'll continue exploring the functionalities of python and sympy. Here's the list of the topics you'll find in this notebook:

- control flow in python
- function calling inside loops
- 2D and 3D plotting with sympy
  - plots of functions
  - parametric plots
  - implicit plots
  - the hyperboloid as a ruled surface

1.1 Control flow

One of the main advantages of programming is the ability to repeat some task automatically. This can save you a lot of time, and automate your processes avoiding manual, boring and error-prone repetitions.

There are 3 tools you need to learn:

- **if-else** statements
- **for** loops
- **while** loops

1.1.1 if-else

It may happen that you want some code to be executed only if some condition is satisfied, and some other code to be executed elsewise. This mechanism is inspired on how humans think. For instance, let's consider the study of the stationary points of a function

$$f : \mathbb{R} \to \mathbb{R}$$

The steps you learned are more or less the following: * Evaluate the 1st derivative $f'(x)$ * Find the values x_i for which it vanishes * Study the $f''(x)$ at those points * For each or those points check the following conditions: * **if** $f''(x_i) > 0$ it corresponds to a minimum * **else if** $f''(x_i) < 0$ it corresponds to a maximum * **else** $f''(x_i) = 0$, and you need to study $f^{(n)}(x_i) \quad n > 2$ until the latter doesn't vanish at that point * When you find n such that $f^{(n)}(x_i) \neq 0$ - **if** n is even, the point is a maximum or minimum - **else**, it is an inflection point

We've used if-else statements all over the previous list of instructions (and also some hidden for and while). Let's now see them in action with python.

[3]:
```
print("Boolean values can control the conditional execution of some piece of␣
 ↪code:")
if 3>0:
  print("3 is greater than 0")
else:
  print("3 is less than 0")
```

```
Boolean values can control the conditional execution of some piece of code:
3 is greater than 0
```

[4]:
```
a = 5
b = -6
if a>0 and b>0:
  print(a,"and", b, "are both positive")
elif a>0 and b<0:
  print(a, "is positive,", b, "is negative")
elif a<0 and b>0:
  print(b, "is positive,", a, "is negative")
elif a==0 or b==0:
  print(a, "or", b, "is 0")
else:
  print(a, "and", b, "are both negative")
```

```
5 is positive, -6 is negative
```

[5]:
```
print("Now let's try with something non-trivial")
from datetime import date # module for date and time
today = date.today()
print("Today's date:", today)
```

```
Now let's try with something non-trivial
Today's date: 2021-01-06
```

[6]:
```
print("ciao")
```

```
ciao
```

[7]:
```
print("Note:", type(today), "is not a string")
AMD = str(today).split("-") # converting to string and splitting
D =  int(AMD[2]) # converting to int
if D%2==0: # D is even
  print("Today is an even day")
else:
  print("Today is an odd day")
```

```
Note: <class 'datetime.date'> is not a string
Today is an even day
```

[8]:
```
print("Now let's check greek pi")
from numpy import pi as pi
if type(pi)==int:
  print(pi,"is an integer")
else:
  print(pi, "is not an integer")
```

```
Now let's check greek pi
3.141592653589793 is not an integer
```

1.1.2 for and while loops

For and while loops allow the repetitive execution of some piece of code:

- **for** : run the code for a predefined list of values (e.g. all the names in a list)
- **while** : run the code until a condition is not valid anymore

[9]:
```
print("'for' loop")
for f in ["Al","John","Jack"]:
    print("Good morning", f, ", how are you?")
```

```
'for' loop
Good morning Al , how are you?
Good morning John , how are you?
Good morning Jack , how are you?
```

[10]:
```
print("It's useful to call functions inside loops")
def introduce(name):
  print("Hi, my name is", name)

for f in ["Al","John","Jack"]:
    introduce(f)
```

```
It's useful to call functions inside loops
Hi, my name is Al
Hi, my name is John
Hi, my name is Jack
```

[11]:
```
print("'while' loop")
T=0
print("You can start your test")
while T<7:
  print(7-T, "minutes left")
  T = T + 1
# end of while
print("Time is up!")
```

```
'while' loop
You can start your test
7 minutes left
6 minutes left
5 minutes left
4 minutes left
3 minutes left
2 minutes left
1 minutes left
Time is up!
```

1.2 Plotting with sympy

In the previous lecture we started to see how sympy can generate plots. Here we'll continue, exploring it's functionalities with 2D and 3D plots.

[12]:
```
import sympy
from sympy import * # importing the whole sympy
from sympy.plotting import * # importing the plotting utilities
x, y, z = symbols("x, y, z") # symbols for the coordinates
t, s = symbols("t, s") # symbols for the parameters (parametric curves and␣
 ↪surfaces)
```

1.2.1 Plot of explicit functions

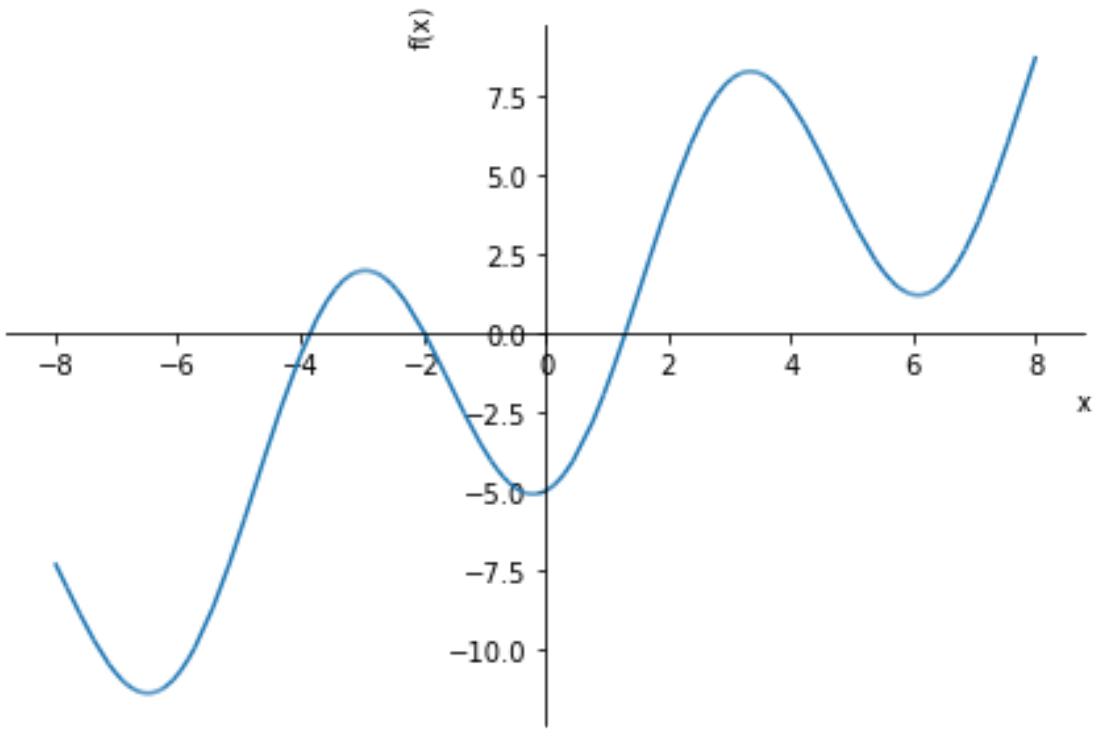
sympy.plotting.plot and sympy.plotting.plot3d accept explicit expressions and a range of values for the independent variable(s)

2D plots We now plot the function:

$$f(x) = x + 5\sin\left(x - \frac{\pi}{2}\right)$$

[13]:
```
def fun(x):
    return x + 5*sin(x - pi/2)
```

[14]:
```
# %matplotlib notebook
plot(fun(x), (x,-8,8))
```

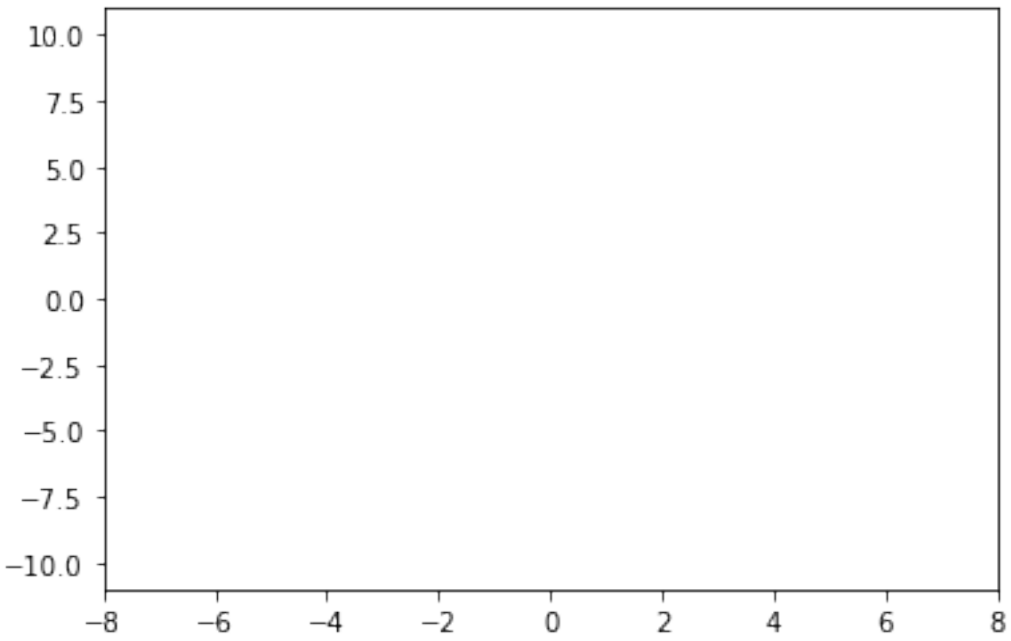


[14]:
```
<sympy.plotting.plot.Plot at 0x7f0013e68400>
```

Animations We can build animations for an improved visualization. Let's now consider the above function. we can animate it such that it seems to be drawn by hand:

[15]:
```
%matplotlib inline
import numpy as np # numpy (arrays of numbers)
import matplotlib.pyplot as plt # plotting with matplotlib

from matplotlib import animation, rc  # animations with matplotlib
from IPython.display import HTML # HTML and javascript support
```

[16]:
```
fig, ax = plt.subplots() # assigninf the figure and axes to 2 variables

ax.set_xlim((-8, 8)) # limits on the x-axis
ax.set_ylim((-11, 11)) # limits on the y-axis

# initializing the variable line with an empty plot:
# it will be updated in the following lines of code
line, = ax.plot([], [], lw=2)
```



[17]:
```
print("Here we define the initialization function:")
def init():
    line.set_data([], [])
    return (line,)
```

Here we define the initialization function:

[18]:
```
print("Here we define the animation function, which depend on the evolution␣
 ↪parameter 'i'")
def animate(i):
    x0 = -8
    m = i/100
    x = np.linspace(x0, x0 + i/10, 100)
    y = x + 5*np.sin(x - np.pi/2)
    line.set_data(x, y)
    return (line,)
```

Here we define the animation function, which depend on the evolution parameter 'i'

[19]:
```
frn = 180 # i=0,...,179
anim = animation.FuncAnimation(fig, animate, init_func=init,
                               frames=frn, interval=20,
                               blit=True)
```

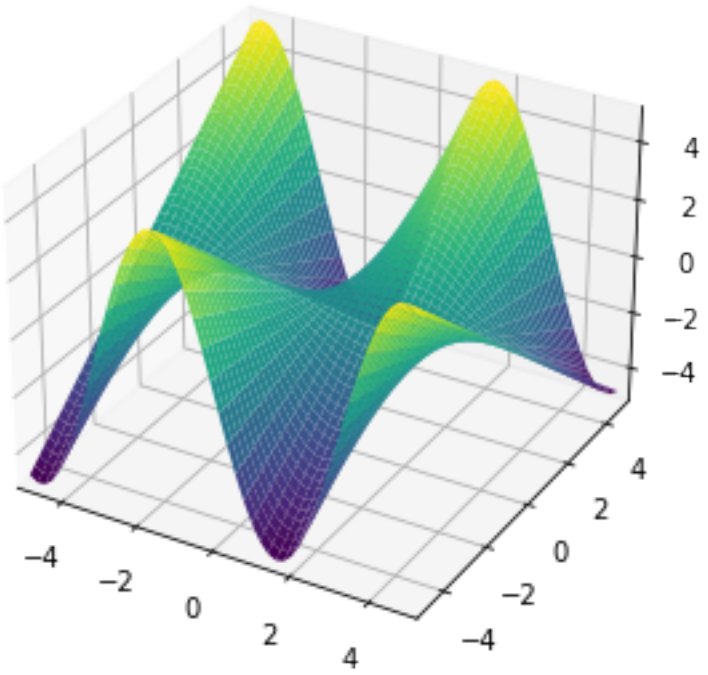
[20]:
```
jsanim = HTML(anim.to_jshtml())
display(jsanim)
```

```
<IPython.core.display.HTML object>
```

3D plots We can do 3D plots as well. Below we plot

$$f(x,y) = y\sin(x)$$

[21]:
```
# %matplotlib notebook
x,y = symbols("x, y")
plot3d(y*sin(x), (x,-5,5),(y,-5,5))
```



[21]:
```
<sympy.plotting.plot.Plot at 0x7f0011d92910>
```

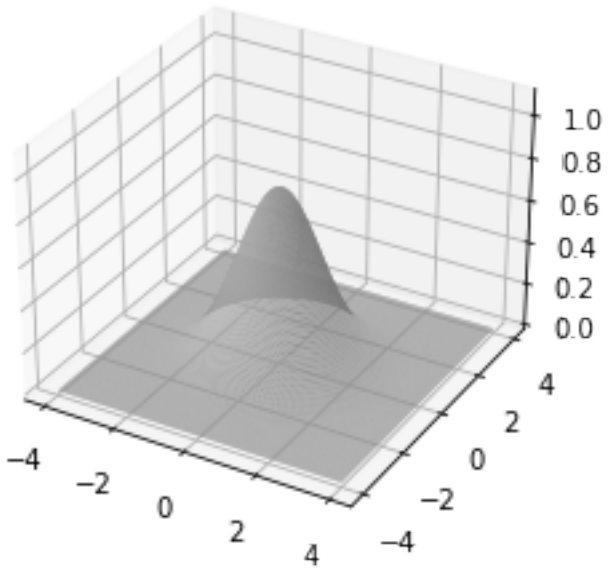
Let's see an animation for 3d plots:

[22]:
```
print("Again, we can animate our plot:")
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.animation as animation
import matplotlib
```

Again, we can animate our plot:

[23]:
```
N = 150 # size of our lattice of points in the xy-plane
fps = 10 # frames per second
frn = 50 # number of frames from the animation

x = np.linspace(-4,4,N) # N points between -4 and 4
x, y = np.meshgrid(x, x) # lattice of points in [-4,4]x[-4,4]
zarray = np.zeros((N, N, frn)) # NxNxfrn numbers

f = lambda x,y,sig : 1/np.sqrt(sig)*np.exp(-(x**2+y**2)/sig**2) # our function

for i in range(frn):
    zarray[:,:,i] = f(x,y,1.5+np.sin(i*2*np.pi/frn)) # sigma oscillating in time␣
 ↪(with 'i')
```

[24]:
```
def update_plot(frame_number, zarray, plot):
    plot[0].remove() # removing the old plot
    plot[0] = ax.plot_surface(x, y, zarray[:,:,frame_number], cmap="magma")

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

plot = [ax.plot_surface(x, y, zarray[:,:,0], color='0.75', rstride=1, cstride=1)]
ax.set_zlim(0,1.1)
anim_3D = animation.FuncAnimation(fig, update_plot, frn, fargs=(zarray, plot),␣
 ↪interval=1000/fps)
```



[25]:
```
jsanim = HTML(anim_3D.to_jshtml())
display(jsanim)
```

```
<IPython.core.display.HTML object>
```

Here we plot:

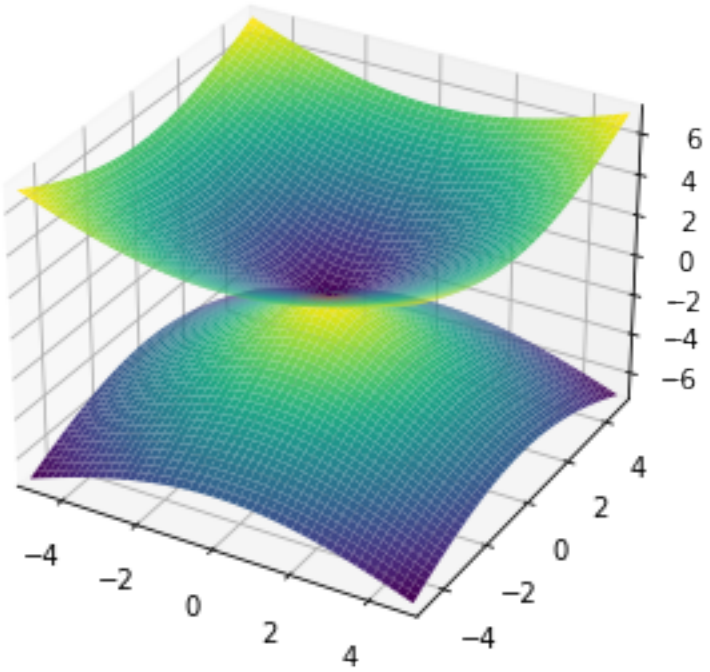
$$f_1(x,y) = +\sqrt{x^2+y^2}$$

$$f_2(x,y) = -\sqrt{x^2+y^2}$$

which are the explicit solutions of:

$$x^2 + y^2 - z^2 = 0$$

[26]:
```
# %matplotlib notebook
x,y = symbols("x, y")
plot3d(sqrt(x**2+y**2),-sqrt(x**2+y**2), (x,-5,5),(y,-5,5))
```



[26]: <sympy.plotting.plot.Plot at 0x7f0011de5940>

1.2.2 Plot of implicit expressions

What if our curve or surface is given with an implicit formula? Sympy can handle it.

2D plots Now we draw an circumference:

$$x^2 + y^2 = 16$$

[ ]:
```
plot_implicit(Eq(x**2+y**2,16),(x,-5,5),(y,-5,5), aspect_ratio=(1,1))
```

Here an hyperbola:

$$x^2 - y^2 = 1$$

with asymptotes:

$$y = \pm x$$

[ ]:
```
hyp = (x**2-y**2-1)
as1 = (y-x)
as2 = (y+x)

plot_implicit(Eq(hyp*as1*as2,0), (x,-5,5),(y,-5,5), aspect_ratio=(1,1))
```

1.2.3 3D plots

The implementation of implicit generic 3D plots is not always straightforward: it's not always possible to find an analytic expression for the folds $z(x,y)$. For instance, as we saw before:

$$x^2 + y^2 - z^2 = 0$$

can be plotted doing the usual plot for the 2 functions:

$$z(x,y) = +\sqrt{x^2+y^2}$$
$$z(x,y) = -\sqrt{x^2+y^2}$$

But how should we plot the following curve?

$$x^2\, tan(yz) + log(x+z)\,\sqrt{y} + sin(z) = \frac{x}{z}$$

The trick here is to define function which can plot a class of 3D curves, such as the quadrics.

[ ]:
```
print("Here we define a quadric plotter.")
def plot3d_quadric(eq,x,a,b,y,c,d,z):
    sol = solve(eq,z) # solving for 'z'
    print("Plotting the following curves:")
    for s in sol:
      display(Eq(z,s)) # displaying the solution
```

```
    plot3d(sol[0],sol[1],(x,a,b),(y,c,d)) # plotting the solution
```

[ ]:
```
plot3d_quadric(x**2+y**2+z**2-16,x=x,a=-5,b=5,y=y,c=-5,d=5,z=z)
```

[ ]:
```
Q1 = z**2-x**2-y**2
display(Eq(Q1, 0))
plot3d_quadric(z**2-x**2-y**2,x,-5,5,y,-5,5,z)
```

[ ]:
```
Q2 = x**2+y**2-z**2-1
display(Eq(Q2, 0))
print("Hyperboloid from the 2 z(x,y) curves:")
plot3d_quadric(Q2,x,-5,5,y,-5,5,z)
```

1.2.4 Parametric plots

We can draw curves using the parametric representation:

$$\vec{r}(t) = \vec{f}(t)$$

Consider the simple case of a circumference of radius $R = 1$. The parametrization is:

$$x(t) = \cos(t)$$
$$y(t) = \sin(t)$$

for $t \in [0, 2\pi)$

[ ]:
```
plot_parametric(cos(t),sin(t),(t,0,2*pi),aspect_ratio=(1,1))
```

Now let's dive in to some more complicated example: a cardioid with radius $a = 1$. This is the movement of a point of a circumference rounding up to another.

$$x(t) = (1 - \cos(t))\cos(t)$$
$$y(t) = (1 - \cos(t))\sin(t)$$

[ ]:
```
x_card = (1 - cos(t))*cos(t)
y_card = (1 - cos(t))*sin(t)
plot_parametric(x_card, y_card, (t,0,10), aspect_ratio=(1,1))
```

Now a cycloid:

$$x(t) = r(t - \sin(t))$$
$$y(t) = r(1 - \cos(t))$$

we choose $r = 1$

[ ]:
```
x_cycl = (t - sin(t))
y_cycl = (1 - cos(t))
plot_parametric(x_cycl, y_cycl, (t,0,10), aspect_ratio=(1,1))
```

Now we draw a spiral:

$$r(\theta) = A\theta$$

(we'll choose $A = 1$)

How do we parametrize in terms of x, y ?

$$x(t) = r(\theta)\cos(\theta) = A\theta\cos(\theta)$$
$$y(t) = r(\theta)\sin(\theta) = A\theta\sin(\theta)$$

[ ]:
```
theta = symbols("theta")
x_spir = theta*cos(theta)
y_spir = theta*sin(theta)
plot_parametric(x_spir, y_spir, (theta,0, 10), aspect_ratio=(1,1))
```

1.3 3D plotting

We can plot in 3D. Let's start with a line:

$$x(t) = -2t$$
$$y(t) = 3 + t$$
$$z(t) = 1 + t$$

[ ]:
```
plot3d_parametric_line(-2*t, 3+t, 1+t, (t, -5, 5))
```

And last, but not least, an **helix**. This is the motion of an electric charge with a magnetic field along the z axis:

$$x(t) = \cos(t)$$
$$y(t) = \sin(t)$$
$$z(t) = t$$

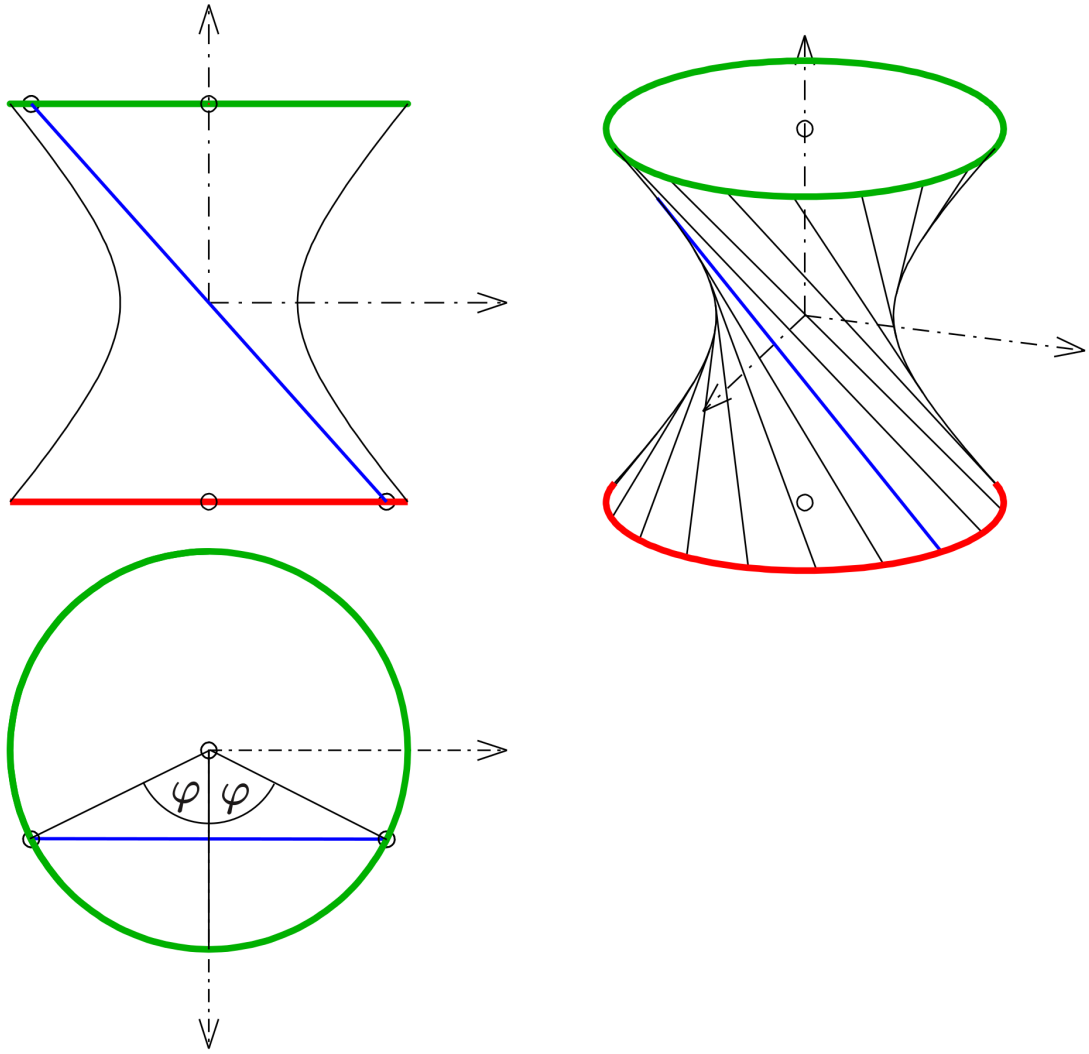
[ ]:
```
x_hel = cos(t)
y_hel = sin(t)
z_hel = t

plot3d_parametric_line(x_hel, y_hel, z_hel, (t, 0, 20))
```

1.3.1 Hyperboloid as a ruled surface

Here we show how an hyperboloid can be drawn. We can use parametric plot utilities from sympy.plotting, or draw it from the straight lines that generate it.

[ ]:
```
t,s = symbols('t, s')
a=b=c=1
from numpy import pi as pi
n=20
print("Hyperboloid drawn as a parametric curve:")
x_hyp = a*sqrt(1+t**2)*cos(s)
y_hyp = b*sqrt(1+t**2)*sin(s)
z_hyp = c*t
plot3d_parametric_surface(x_hyp,y_hyp, z_hyp, (t, -5, 5), (s,0,2*pi))
```

In order to draw the hyperboloid as a ruled surface, keep in mind the following image:



Our steps are then:

- generate n points along 2 circumferences, with some phase difference between the corresponding points

- draw a straight line connecting the couples of points of the circumference

[ ]:
```
print("We define here a function which draws a circumference on the xy plane at␣
 ↪height z")
import numpy as np
def circ_points(n, phi, h):
  points = [] # list of points
  for i in range(0,n):
    theta = phi + i*(2*np.pi/n)
    x = cos(theta)
    y = sin(theta)
    z = h
    points.append([x,y,z])
  return points

print("Check of the function:")
Pi = circ_points(10, 0.0, 0.0)
print(Pi)
```

[ ]:
```
print("Here we define a function that plot a line passing for A and B")
import numpy as np
def get_plot_line(A0,B0, I=[-5,5]):
  A = np.array(A0)
  B = np.array(B0)
  v = (A - B)
  t = symbols("t")
  rt = A + v*t
  return plot3d_parametric_line(rt[0], rt[1], rt[2], (t, I[0], I[1]), show=False)


print("testing")
p1 = get_plot_line([0,0,0],[1,2,3])
# p1.show()
```

[ ]:
```
N = 40
phi = 0.5
h = 1
c1 = circ_points(N,0.0,0.0)
c2 = circ_points(N,phi,h)

P_hyp = sympy.plotting.plot(show=False)
for i in range(0,N):
    P_hyp.extend(get_plot_line(c1[i], c2[i]))
P_hyp.show()
```

Now we can think to automate the process:

[1]:
```
print("Defining a function for generic hyperboloid as ruled surface")
def hyp_ruled(N, phi, h):
    c1 = circ_points(N,0.0,0.0)
    c2 = circ_points(N,phi,h)
    P_hyp = sympy.plotting.plot(show=False)
    for i in range(0,N):
        P_hyp.extend(get_plot_line(c1[i], c2[i]))
    P_hyp.show()
```

```
Defining a function for generic hyperboloid as ruled surface
```

[2]:
```
print("Testing:")

hyp_ruled(40, 1.0, 25)
```

```
Testing:

---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-2-25b3985cfade> in <module>
  1 print("Testing:")
  2
----> 3 hyp_ruled(40, 1.0, 25)


<ipython-input-1-17a9137e00f8> in hyp_ruled(N, phi, h)
  1 print("Defining a function for generic hyperboloid as ruled surface")
  2 def hyp_ruled(N, phi, h):
----> 3     c1 = circ_points(N,0.0,0.0)
  4     c2 = circ_points(N,phi,h)
  5     P_hyp = sympy.plotting.plot(show=False)


NameError: name 'circ_points' is not defined
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: