In [4]:
```
from sympy import *
from sympy.plotting import (plot, plot_parametric)
```

OPTIMIZATION AND ANTIDERIVATIVES IN PYTHON

Optimization is the perfect blend of brain and technology. Oddly enough, if you type a word problem into Python and press "Enter", it doesn't solve it! Your job is to determine the mathematical setup. Python's job is to do the grunge work (the algebraic and calculus manipulations).

To help you set up these types of problems, the following general strategies may be useful:

1) Read the entire problem and make sure you understand the story

2) If necessary, draw a diagram and label all constants (lengths/angles that stay the same) and all variables (lenghts/angles you can change).

3) Write an equation relating your variables (in these examples, it is based on the "goal": to maximize or minimize some quantity)

4) If there is more than one variable, write equations for any restrictions given in the problem. These may be explicit ("The box has to have a volume of...") or implicit based on the geometry of the diagram.

5) Solve the restrictions (**solve**) for one variable and substitute (**subs**) into the goal until you have one variable left.

6) Optimize as explained in the Lab 7 Overview. Recall the process:

a) Differentiate the function (**diff**)

b) Solve the derivative = 0 (**solve**)

c) Show the critical value is the max/min you want:

i) If closed, bounded interval, substitute CV and endpoints in the original function (**subs** and **for**)

ii) If not, substitute CV into the second derivative (**diff** and **subs**)

NOTE: The examples here are NOT copy/paste to solve the problems in lab. However, they will USE many of the features you will use to solve your problems.

EXAMPLE 1:

77 in the textbook, available at https://calclab.math.tamu.edu/Python/maxminExample.png (https://calclab.math.tamu.edu/Python/maxminExample.png)

Assuming you've read and understand the problem, you've done step 1. For step 2, the constants are already labeled, so label AP = x and BP = y. For step 3, the key to coming up with an equation is that theta and the angles on either side of it form a straight line (pi radians). So theta = pi - arctan(2/y) - arctan(5/x). This has 2 variables in it, so step 4 needs a relationship between y and x-clearly, x + y = 3 (NOTE: at the beginning, you could have said AP = x and BP = 3-x if you wanted to). Step 5 can easily be done by hand, but if you needed to, we show how to apply it in Python.

In [6]:
```
x,y=symbols('x y')
# Define the goal (theta) and restriction
theta=pi-atan(2/y)-atan(5/x)
restriction=x+y-3
# Step 5a: solve restriction for one variable
y_ofx=solve(restriction,y)
print(y_ofx)  # Good idea to see what results you get
# Step 5b: substitute into goal
theta_ofx=theta.subs(y,y_ofx[0])
#REMEMBER: "solve" returns a LIST of solutions, even if there is just one!
#ALSO REMEMBER: the first element in a list is element 0!
print('The function theta(x) is',theta_ofx)
```

```
[-x + 3]
The function theta(x) is -atan(5/x) - atan(2/(-x + 3)) + pi
```

In [7]:
```
# Now ready to find the critical values.  NOTICE also that x has a practical domain of [0,3]: and
 BOTH endpoints are valid!
# Step 6a
dtheta=diff(theta_ofx,x)
# Step 6b
cvals=solve(dtheta,x)
print(cvals) # Again, good to see what we are working with before proceeding.  We see that we need
to convert to float
cvals_float=[i.evalf() for i in cvals]  # REMEMBER: We cannot evalf a list.  Use list comprehensio
n.
print(cvals_float) # Only the first critical is in the domain.
```

```
[-2*sqrt(5) + 5, 2*sqrt(5) + 5]
[0.527864045000421, 9.47213595499958]
```

So the only critical value in the domain is -2+sqrt(5). Since we have a closed, bounded interval, we use Step 6(c)(i) to verify our answer is a maximum.

In [8]:
```
testpoints=[0,-2+sqrt(5),3]
yvals=[theta_ofx.subs({x:i}).evalf() for i in testpoints] #Notice we are converting to float right
away here
print('The y-values are',yvals)
```

```
The y-values are [-atan(zoo) + 2.55359005004223, 0.991580500951695, -atan(zoo) + 2.11121582706548]
```

What is atan(zoo)? If you evaluated by hand, you would get 0 in the denominator of your arctangent expression, which could be +oo OR -oo...and Python doesn't know which one you want! But you know we are approaching 0 from the right and approaching 3 from the left, so let's just approach them numerically.

In [9]:
```
# Take two...use 0.00001 for 0 and 2.9999 for 3
testpoints=[0.00001,-2+sqrt(5),2.9999]
yvals=[theta_ofx.subs({x:i}).evalf() for i in testpoints]
print('The y-values are',yvals)  #NOTE: I ran the code here to see which answer was largest!
print('So the maximum theta is',yvals[1],'occuring when x=',testpoints[1])
```

```
The y-values are [0.982794184782240, 0.991580500951695, 0.540454794258432]
So the maximum theta is 0.991580500951695 occuring when x= -2 + sqrt(5)
```

Example 2 (incorporating antiderivatives as well, which are done using the Python command **integrate**) A projectile is launched with initial velocity v0 at an angle theta. Assuming gravity is the only force acting on the projectile, at what angle should it be launched to maximize the horizontal distance traveled.

We've read and understand the story (step 1). No picture needed for step 2. To get our equations, we need to know that when the projectile is launched, the acceleration is [0, -g], the intial velocity is [v0 *cos(theta), v0* sin(theta)], and we'll call the initial position [0, 0] (the origin). Antidifferentiate (also called integrate-hence the command name) each component of the acceleration twice WITH RESPECT TO TIME. We will also demonstrate for vx how to find the arbitrary constant even though in this case (NOT ALWAYS IN GENERAL!!!!) the arbitrary constants are the initial conditions.

In [10]:
```
v0,theta,g,t,C=symbols('v0 theta g t C')
ax=0
ay=-g
vx=integrate(ax,t)+C #NOTE: Python does NOT put the arbitrary constant at the end
# Use the initial velocity to solve for C
Ceqn=vx.subs(t,0)-v0*cos(theta)
Csoln=solve(Ceqn,C) # print here to confirm C value
print(Csoln)  # only one solution: the 0th one
vx=vx.subs(C,Csoln[0])
print('The x-component of v(t) is',vx)
# For the rest, we will use the fact that C is the initial condition-REMEMBER this is NOT always t
rue!
vy=integrate(ay,t)+v0*sin(theta)
rx=integrate(vx,t)+0   #No, you don't have to put the +0. I'm relating it to the initial position
 for clarity.
ry=integrate(vy,t)+0   #See comment above.
print('The acceleration is',[ax,ay])
print('The velocity is',[vx,vy])
print('The position is',[rx,ry])
```

```
[v0*cos(theta)]
The x-component of v(t) is v0*cos(theta)
The acceleration is [0, -g]
The velocity is [v0*cos(theta), -g*t + v0*sin(theta)]
The position is [t*v0*cos(theta), -g*t**2/2 + t*v0*sin(theta)]
```

So according to the problem, our goal (step 3) is to maximize rx, but there are two variables (t and theta-remember v0 is a fixed value). For step 4, we have the restriction that the projectile hits the ground, i.e., ry = 0. So we solve this equation for t and substitute into our goal.

In [7]:
```
# Back to our max/min problem: Step 5a
t_oftheta=solve(ry,t)
print(t_oftheta) # We will get 2 solutions, so we run it now to see which is the useful one...the
 second one.
# Step 5b
rx_oftheta=rx.subs(t,t_oftheta[1])  #Again, count the numbers in the list starting with 0
print('rx(theta) =',rx_oftheta)
```

```
[0, 2*v0*sin(theta)/g]
rx(theta) = 2*v0**2*sin(theta)*cos(theta)/g
```

v0 and g are constants, so theta is our only variable. Use the same procedure as before...find the critical values and evaluate rx(theta) at the critical value(s) AND ENDPOINTS (in this case, the practical interval for theta is [0, pi/2])

In [8]:
```
# Step 6a
drx=diff(rx_oftheta,theta)
# Step 6b
cvals=solve(drx,theta)
print(cvals) # Ran it here.  The only critical value in the domain is pi/4
# Step 6c(i)
testpoints=[0,pi/4,pi/2]
xvals=[rx_oftheta.subs({theta:i}) for i in testpoints] #No evalf here because there are symbolic c
onstants!
print('The x-values are',xvals)
print('So the maximum distance is',xvals[1],'when theta=',testpoints[1])
```

```
[-3*pi/4, -pi/4, pi/4, 3*pi/4]
The x-values are [0, v0**2/g, 0]
So the maximum distance is v0**2/g when theta= pi/4
```

In [ ]:
```
```