```
[118]: from sympy import solve
       from sympy import Symbol
       from math import cos, sin, pi, sqrt, tan, fabs
       import numpy as np
       import matplotlib.pyplot as plt
       from scipy import optimize
       from instances.parser import *
       from E3PNT.ThreePntEllipse import *

       %matplotlib inline
```

```
[119]: def isInc(f, x):
           eps=1e-9
           return f(x+eps) > f(x)

       from scipy.optimize import minimize_scalar


       def get_center(theta, a, b, h, x, y):

           m = tan(theta)

           B = a**2*m**2 + b**2
           A = 4*a**2*b**2*(1+m**2)
           D = 4*h**2
           c = sqrt((B*A-D*B**2)/A)
           alx = (-a**2*m*c+a*b*sqrt(a**2*m**2+b**2-c**2))/(a**2*m**2+b**2)
           aly = alx * m + c
           px = h * cos(theta)
           py = h * sin(theta)
           bx = alx - px
           by = aly - py
           xc = bx * cos(theta) + by * sin(theta)
           yc = bx * sin(theta) - by * cos(theta)

           return xc, yc

       def fk(theta, a, b, h, x, y):

           #print("bla", theta, tu)
           xc, yc = get_center(theta, a, b, h, x, y)
           X = xc-x
           Y = yc-y
           return (X*cos(theta) + Y*sin(theta))**2/a**2 + (X*sin(theta) - Y*cos(theta))**2/b

       def g(a, b, h, x, y, tu):

           #res = minimize_scalar(fk, bounds=(0, tu-0.001), method='bounded', args=(a, b, h

           res = find_roots(fk, 0, tu-0.001, 0, (a, b, h, x, y))
           print(res)
```

```
    arr = np.linspace(0.0001, max(tu-0.00001, 0.0001), 2000)
    ds = [fk(t, a, b, h, x, y) for t in arr]
    #plt.axvline(x=tu*2/3)
    #plt.axvline(x=tu*1/3)
    plt.plot(arr, ds)
    nit = res[1]

    nsols = len(res[0])

    for s in res[0]:
        plt.scatter([s.root], [fk(s.root, a, b, h, x, y)])

    plt.show()
    return nit, nsols
```

```
[121]: I = parse_input('instances/CM3.txt')
       X=I.X
       Y=I.Y
       n = len(X)
       a = I.a[0]
       b = I.b[0]
       tot = 0
       nit = 0
       nsols = 0

       for i in range(n):
           for j in range(i+1, n):
               for k in range(j+1, n):
                   ctx = Context(a, b, X[i], Y[i], X[j], Y[j], X[k], Y[k])
                   ctx.x_mirror()
                   #ctx.y_mirror()
                   #ctx.x_mirror()
                   h = ctx.h
                   xa = ctx.p[2].x
                   ya = ctx.p[2].y

                   tu = get_upper_limit(ctx)

                   if tu < 1e-5:
                       continue



                   print(f"h: {h}, x: {xa}, y: {ya}")
                   ret = g(a, b, h, xa, ya, get_upper_limit(ctx))
                   nit += ret[0]
                   nsols += ret[1]
                   tot += 1
```