```
def intersectLineCylinder(line=None, cylinder=None):
    import numpy as np
    import math as m
    import linePosition3d


    #INTERSECTLINECYLINDER Compute intersection points between a line and a
cylinder
    #
    #   POINTS = intersectLineCylinder(LINE, CYLINDER)
    #   Returns intersection points between a line and a cylinder.
    #
    #   Input parameters:
    #   LINE     = [x0 y0 z0  dx dy dz]
    #   CYLINDER = [x1 y1 z1 x2 y2 z2 R]
    #
    #   Output:
    #   POINTS   = [x1 y1 z1 ; x2 y2 z2]
    #
    #   POINTS = intersectLineCylinder(LINE, CYLINDER, 'checkBounds', B)
    #   Where B is a boolean (TRUE by default), check if the points are within
    #   the bounds defined by the two extreme points. If B is false, the
    #   cylinder is considered as infinite.
    #
    #   Example
    #     % Compute intersection between simple cylinder and line
    #     line = [60 60 60 1 2 3];
    #     cylinder = [20 50 50 80 50 50 30];
    #     points = intersectLineCylinder(line, cylinder);
    #     % Display the different shapes
    #     figure;
    #     drawCylinder(cylinder);
    #     hold on; light;
    #     axis([0 100 0 100 0 100]);
    #     drawLine3d(line);
    #     drawPoint3d(points, 'ko');
    #
    #
    #     % Compute intersections when one of the points is outside the
    #     % cylinder
    #     line = [80 60 60 1 2 3];
    #     cylinder = [20 50 50 80 50 50 30];
    #     intersectLineCylinder(line, cylinder)
    #     ans =
    #           67.8690   35.7380   23.6069
    #
    #
    #   See also
    #   lines3d, intersectLinePlane
    #
    #   References
    #   See the link:
    #   http://www.gamedev.net/community/forums/topic.asp?topic_id=467789
    #
    # ---
    # Author: David Legland, from a file written by Daniel Trauth (RWTH)
    # e-mail: david.legland@grignon.inra.fr
    # Created: 2007-01-27

    # HISTORY
    # 2010-10-21 change cylinder argument convention, add bounds check and doc
    # 2010-10-21 add check for points on cylinders, update doc
    # 2018-07-06 translated to python by Valerie Martin
```

```
#% Parse input arguments

# default arguments
checkBounds = 1

#% Parse cylinder parameters

# Starting point of the line
l0 = np.asarray(line[0:3])

# Direction vector of the line
dl = np.asarray(line[3:6])

# Starting position of the cylinder
c0 = np.asarray(cylinder[0:3])

# Direction vector of the cylinder
dc = np.subtract(cylinder[3:6], c0)

# Radius of the cylinder
r = cylinder[6]


#% Resolution of a quadratic equation to find the increment

# Substitution of parameters
e = dl - (np.dot(dl, dc) / np.dot(dc, dc)) * dc
f = (l0 - c0) - (np.dot(l0 - c0, dc) / np.dot(dc, dc)) * dc

# Coefficients of 2-nd order equation
A = np.dot(e, e)
B = 2 * np.dot(e, f)
C = np.dot(f, f) - r ** 2

# compute discriminant
delta = B ** 2 - 4 * A * C

# check existence of solution(s)
if delta < 0:
    points = np.zeros((0, 3))


# extract roots
x1 = (-B + m.sqrt(delta)) / (2 * A)
x2 = (-B - m.sqrt(delta)) / (2 * A)
x = np.asarray([x1, x2])


#% Estimation of points position

# process the smallest position
x1 = x.min()

# Point on the line: l0 + x*dl = p
point1 = l0 + x1 * dl

# process the greatest position
x2 = x.max()

# Point on the line: l0 + x*dl = p
point2 = l0 + x2 * dl

# Format result
```

```
    points = np.asarray([point1, point2])


    #% Check if points are located between bounds

    if checkBounds:
        # cylinder axis
        axis = np.concatenate((c0, dc), axis=0)

        # compute position on axis
        ts = np.asarray([linePosition3d.linePosition3d(point1, axis),
linePosition3d.linePosition3d(point2, axis)])


        # check bounds
        ind = np.logical_and(ts>=0, ts<= 1)
        points = points[ind, :]

    return points
```