trigonometric_method

April 26, 2023

1 Trigonometrical method

Tasks: 1. numerical analysis of mechanism 2. optimization of mechanism 3. control of mechanism ## Python Intro ## Why Python -> it is free! - libraries - variables, numbers, lists … - flow control - function definition - plotting

1.1 Libraries

- use the available useful functions
- the functions are gather in the libraries
- the libraries are usually open-source projects
- you can download them directly (from Git etc.) or using pip (https://pypi.org/project/pip/)

[ ]:
```
# The Python Standard Libraries                    https://docs.python.org/3/
 ↪library/index.html
# - standard built-in, without installation
import math

# Basic library for numerical calculation          https://numpy.org/doc/stable/
 ↪user/quickstart.html
# - Numpy vs. Matlab                               https://numpy.org/doc/stable/
 ↪user/numpy-for-matlab-users.html
import numpy

# Basic library for plotting                       https://matplotlib.org/
from matplotlib import pyplot as plt
```

1.2 Variables, Numbers, Lists …

- you can easily define necessary variables
- the variables are usually defined without data type definition (boolean, int, string …) - all types are objects

[ ]:
```
a = 0.1 # [m]
b = 0.5 # [m]
c = 0.3 # [m]

angle_fi_0_s = 0 # [rad]
```

```
angle_fi_0_e = 2*math.pi # [rad], pi is defined in the library math as constant
angle_fi_0_num_division = 100 # number of sections between the angle_fi_0 and␣
 ↪the angle_fi_1

# using function from the imported library numpy
# each function has specified parameters
angles_fi_0 = numpy.linspace(angle_fi_0_s, angle_fi_0_e,␣
 ↪angle_fi_0_num_division) # get list of angles between the angle_fi_0 and the␣
 ↪angle_fi_1

# basic printing function - print()
# parameter can be a functional string - f''
print(f'List of angles: {angles_fi_0} rad')
```

```
List of angles: [0.         0.06346652 0.12693304 0.19039955 0.25386607
0.31733259
 0.38079911 0.44426563 0.50773215 0.57119866 0.63466518 0.6981317
 0.76159822 0.82506474 0.88853126 0.95199777 1.01546429 1.07893081
 1.14239733 1.20586385 1.26933037 1.33279688 1.3962634  1.45972992
 1.52319644 1.58666296 1.65012947 1.71359599 1.77706251 1.84052903
 1.90399555 1.96746207 2.03092858 2.0943951  2.15786162 2.22132814
 2.28479466 2.34826118 2.41172769 2.47519421 2.53866073 2.60212725
 2.66559377 2.72906028 2.7925268  2.85599332 2.91945984 2.98292636
 3.04639288 3.10985939 3.17332591 3.23679243 3.30025895 3.36372547
 3.42719199 3.4906585  3.55412502 3.61759154 3.68105806 3.74452458
 3.8079911  3.87145761 3.93492413 3.99839065 4.06185717 4.12532369
 4.1887902  4.25225672 4.31572324 4.37918976 4.44265628 4.5061228
 4.56958931 4.63305583 4.69652235 4.75998887 4.82345539 4.88692191
 4.95038842 5.01385494 5.07732146 5.14078798 5.2042545  5.26772102
 5.33118753 5.39465405 5.45812057 5.52158709 5.58505361 5.64852012
 5.71198664 5.77545316 5.83891968 5.9023862  5.96585272 6.02931923
 6.09278575 6.15625227 6.21971879 6.28318531] rad
```

1.3 Programme flow control

- the standard flow control tools are defined in Python https://docs.python.org/3/tutorial/controlflow.html
- programme blocks are defined by indentation

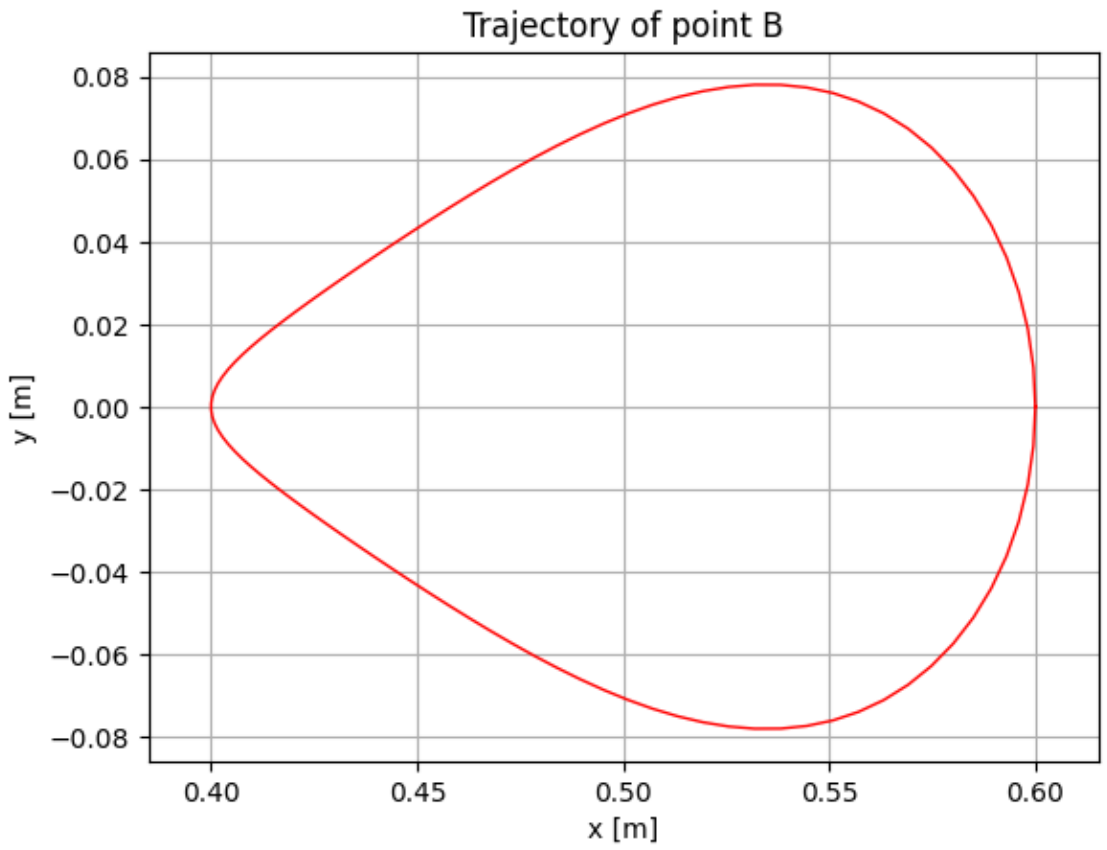
[ ]:
```
# empty list of coordinates defition
x_B = []
y_B = []

for phi_1 in angles_fi_0:
    # calculate defined values
    dis_AS2 = math.sqrt((a*math.cos(phi_1)-c)**2 + (a * math.sin(phi_1))**2)
    dis_BS2 = b - dis_AS2
    phi_2 = math.atan(a * math.sin(phi_1) / (c - a * math.cos(phi_1)))
```

```
    # append new values to existing lists
    x_B.append(dis_BS2 * math.cos(-phi_2) + c)
    y_B.append(dis_BS2 * math.sin(-phi_2))
```

1.4 Function definition and Plotting

- you can easily define own functions -> do it!
- decompose your code into function blocks
- add description of your function into the code
- describe inputs and outputs of the function

[ ]:
```
def make_2D_graph(x, y, x_name, y_name, title):
    """
    Funtion plots the data [x, y] into 2D graph.

    param x: x data
    type  x: int
    param y: y data
    type  y: int
    param x_name: name of the x axis
    param x_name: str
    param y_name: name of the y axis
    param y_name: str
    param title: title of the graph
    param title: str
    """
    plt.figure(1, dpi=100)
    ax = plt.axes()
    ax.set_xlabel(x_name)
    ax.set_ylabel(y_name)
    ax.set_title(title)
    # plot line - parameters https://matplotlib.org/stable/api/_as_gen/
 ↪matplotlib.pyplot.plot.html
    ax.plot(x, y, 'r', linewidth=1)
    ax.grid(True)
    ax.axis('equal')

# example of usage the defined function
# plot the trajectory of point B[x_B, y_B]
make_2D_graph(x_B, y_B, 'x [m]', 'y [m]', 'Trajectory of point B')
```

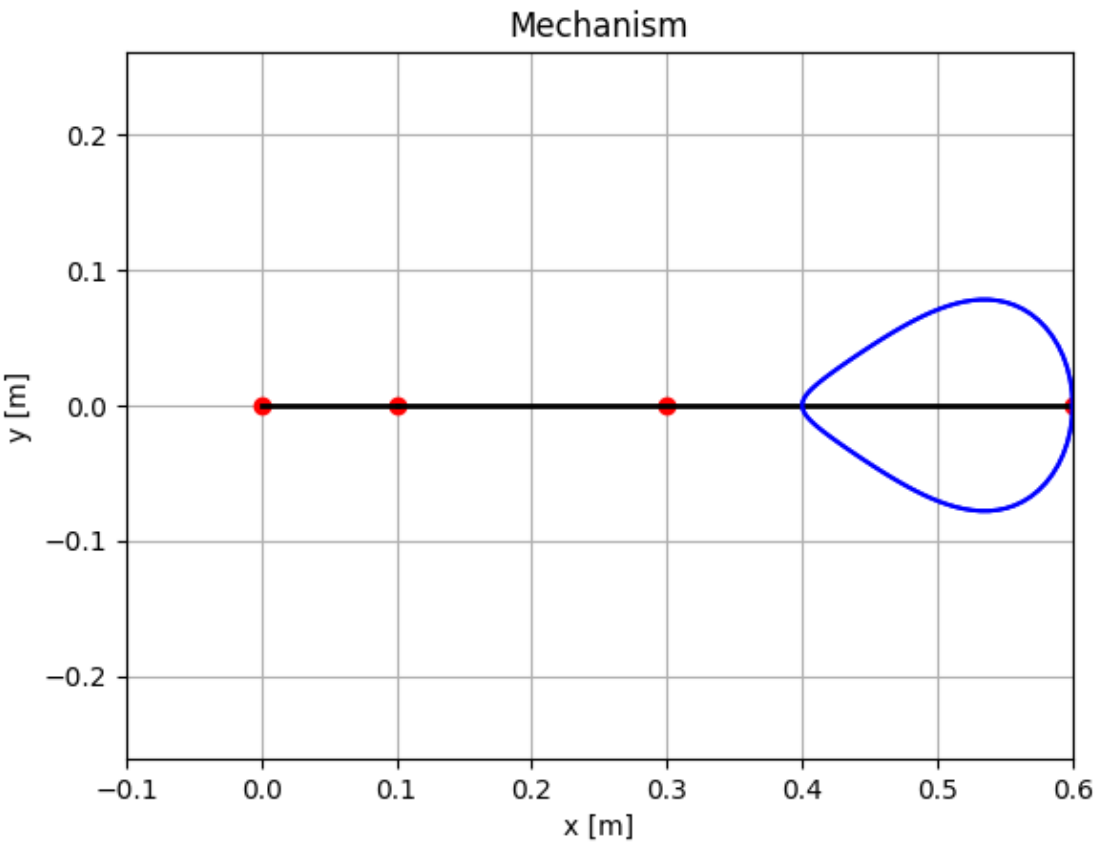


[ ]:
```python
# Animation of the mechanism
from matplotlib.animation import FuncAnimation
# init figure
fig, ax = plt.subplots()

# create dict of lines and points
objects = {
    'point_A' : {
        'label': 'A',
        'style': 'ro',
        'x': a,
        'y': 0,
        'point': True,
        'obj': None,
    },
    'point_B' : {
        'label': 'A',
        'style': 'ro',
        'x': a + b,
        'y': 0,
```

```
        'point': True,
        'obj': None,
    },
    'point_S1' : {
        'label': '$S_1$',
        'style': 'ro',
        'x': 0,
        'y': 0,
        'point': True,
        'obj': None,
    },
    'point_S2' : {
        'label': '$S_2$',
        'style': 'ro',
        'x': c,
        'y': 0,
        'point': True,
        'obj': None,
    },
    'line_S1A' : {
        'style': 'k',
        'x': [0, a],
        'y': [0, 0],
        'point': False,
        'obj': None,
    },
    'line_AB' : {
        'style': 'k',
        'x': [a, a + b],
        'y': [0, 0],
        'point': False,
        'obj': None,
    },
}

# create graphic objects
for object in objects:
    objects[object]['obj'] = ax.plot(objects[object]['x'],␣
 ↪objects[object]['y'], objects[object]['style'], linewidth=2)

def init():
    """
    Function is called before update().
    """
    ax.set_xlabel('x [m]')
    ax.set_ylabel('y [m]')
    ax.set_title('Mechanism')
```

```
    ax.plot(x_B, y_B, 'b')
    ax.grid(True)
    ax.axis('equal')
    ax.set_xlim(-a, a+b)
    ax.set_ylim(-b, b)

    return [objects[object]['obj'][0] for object in objects]

def update(frame):
    """
    Function generates each frame in the animation.
    """
    objects['point_A']['x'] = a * math.cos(angles_fi_0[frame])
    objects['point_A']['y'] = a * math.sin(angles_fi_0[frame])

    objects['point_B']['x'] = x_B[frame]
    objects['point_B']['y'] = y_B[frame]

    objects['line_S1A']['x'] = [0, objects['point_A']['x']]
    objects['line_S1A']['y'] = [0, objects['point_A']['y']]

    objects['line_AB']['x'] = [objects['point_A']['x'], objects['point_B']['x']]
    objects['line_AB']['y'] = [objects['point_A']['y'], objects['point_B']['y']]

    for object in objects:
        objects[object]['obj'][0].set_data(objects[object]['x'],␣
 ↪objects[object]['y'])

    return [objects[object]['obj'][0] for object in objects]

# object generates animation
# def of number of frames and FPS
animation = FuncAnimation(fig, update, frames=range(len(angles_fi_0)),␣
 ↪init_func=init, blit=True)

# save the animation into .gif
animation.save('motion.gif', fps=15)
plt.show()
```

```
MovieWriter ffmpeg unavailable; using Pillow instead.
```


Mechanism
0.2
0.1
0.0
−0.1
−0.2
y [m]
−0.1
0.0
0.1
0.2
0.3
0.4
0.5
0.6
x [m]