Homework 2

September 22, 2021

Gabe Morris

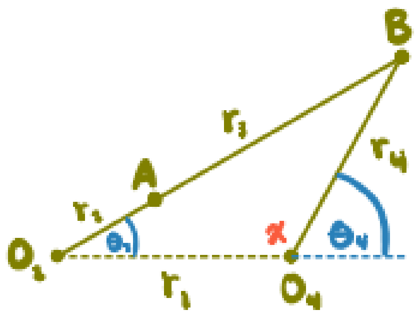
gnm54

[1]:
```
# Importing packages that will be used throughout the document.
import matplotlib.pyplot as plt
from scipy.optimize import fsolve
import sympy as sp
import numpy as np
from mech import *  # This package was created by me for this class.
plt.rcParams['figure.dpi'] = 300

import warnings
warnings.filterwarnings('ignore')
```

1 Problem 1

1.1 Part A

1.1.1 Limit Position 1



The value of x may be solved by the Law of Cosines. Then, the value of θ_4 may be solved by subtracting it from 180°. Finally, the value of θ_2 can be solved by the Law of Sines.

[2]:
```python
# Solving for the value of x
from sympy.solvers import solve
x = sp.Symbol('x')
solve(sp.Pow(180 + 520, 2) - sp.Pow(400, 2) - sp.Pow(400, 2) + 2*400*400*sp.
 ↪cos(x), x)[0]
```

[2]:

$$-\operatorname{acos}\left(-\frac{17}{32}\right) + 2\pi$$

[3]:
```python
# The value of x solved above is the conjugate. Here is the value of x:
x = sp.N(sp.acos(-17/32)*180/sp.pi)
x
```

[3]:

122.08995125628

[4]:
```python
# Calculating theta_4
th4_1 = 180 - x
th4_1
```

[4]:

57.9100487437197

[5]:
```python
# Calculating the value of theta_2
th2_1 = sp.N(sp.asin(sp.sin(x*sp.pi/180)/700*400)*180/sp.pi)
th2_1
```

[5]:

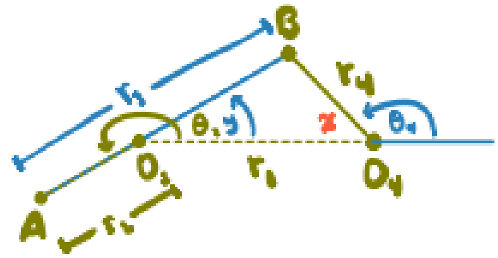
28.9550243718598

1.1.2 Answers:

$\theta_2 = 28.9550243718598°$

$\theta_4 = 57.9100487437197°$

1.1.3 Limit Position 2



This is the same process as above except this time, the the side opposite to θ_4 is $r^3 - r^2$.

[6]:
```
# Solving for the value of x
x2 = sp.N(sp.acos((340**2 - 400**2 - 400**2)/(-2*400*400))*180/sp.pi)
x2
```

[6]: 50.3013268250874

[7]:
```
# Solving for theta_4
th4_2 = 180 - x2
th4_2
```

[7]: 129.698673174913

[8]:
```
# Solving for theta_2
# You have to add 180 to get the value of the angle shown in the figure above.
th2_2 = sp.N(sp.asin(400*sp.sin(th4_2*sp.pi/180)/340)*180/sp.pi) + 180
th2_2
```

[8]: 244.849336587456

1.1.4 Answers:

$\theta_2 = 244.849336587456°$

$\theta_4 = 129.698673174913°$

1.2 Part B

The rocker angle is the maximum angle of θ_4 minus the minimum θ_4.

[9]:
```
th4_2 - th4_1
```

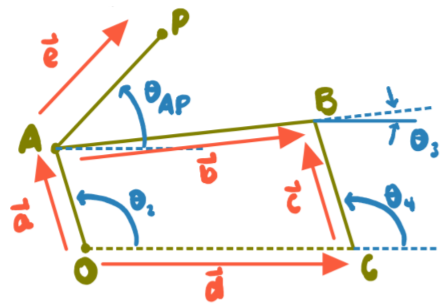
[9]: 71.7886244311929

1.2.1 Answer:

The rocker angle is 71.7886244311929°

2 Problem 2

Kinematic Diagram:



Defining Vectors and Joints for this mechanism:

[10]:
```
O, A, B, C, P = get_joints('OABCP')
a = Vector((O, A), length=2)
b = Vector((A, B), length=4.1)
c = Vector((C, B), length=3)
d = Vector((O, C), length=4, angle=0, ls='--', color='black')
e = Vector((A, P), length=2.5)
# Vector f is not shown in the figure but is the vector from O to P
mech = Mechanism(vectors=(a, b, c, d, e), input_vector=a)
```

Position Loop Equation:

$$\begin{cases} 2cos(\theta_2) + 4.1cos(\theta_3) - 3cos(\theta_4) - 4 = 0 \\ 2sin(\theta_2) + 4.1sin(\theta_3) - 3sin(\theta_4) = 0 \end{cases}$$

Knowns: a, b, c, d, e, and θ_1 which is 0

Unknowns: θ_2, θ_3, and θ_4, but θ_2 is the input angle

[11]:
```
position_loop = lambda x, i: np.array((a.get_x(i) + b.get_x(x[0]) - c.
 ↪get_x(x[1]) - d.length,
                                       a.get_y(i) + b.get_y(x[0]) - c.
 ↪get_y(x[1])))
```

2.1 $\theta_2 = 45°$

[12]:
```
# Solving for the values of theta_3 and theta_4 when theta_2 is 45 degrees
guess = np.deg2rad([20, 60])
solution = fsolve(position_loop, guess, args=(np.pi/4, ))
np.rad2deg(solution)
```

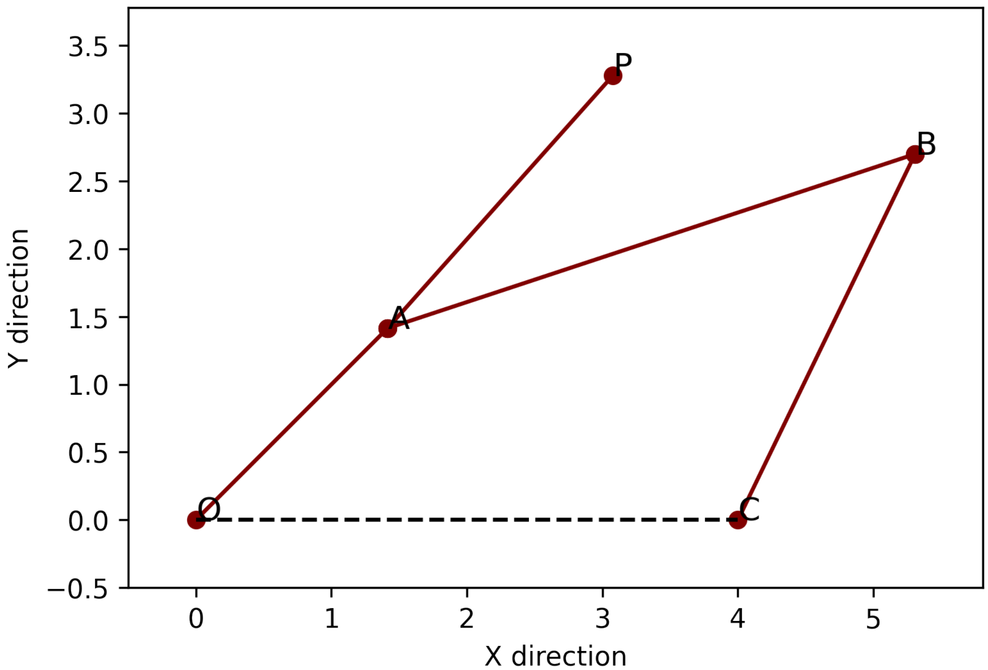
[12]:
```
array([18.27915529, 64.16535766])
```

The position of point P can be determined by summing vectors a and e. The angle for vector e is $\theta_3 + 30°$.

[13]:
```
e.get_x(np.deg2rad(30) + solution[0])
e.get_y(np.deg2rad(30) + solution[0])
```

[13]: 1.8659902906492987

[14]:
```
mech.fix_position()  # This fixes the positions of all the joints.
mech.plot(cushion=0.5)  # This produces a plot
mech.tables(position=True)  # Displays tables
```




```
POSITION
--------

Vector | Length | Angle             | x                  | y
-------+--------+-------------------+--------------------+-------------------
R_OA   | 2      | 45.0              | 1.4142135623730951 | 1.4142135623730951
R_AB   | 4.1    | 18.27915529442809 | 3.8931125568365097 | 1.2859528062109784
R_CB   | 3      | 64.16535766360016 | 1.307326119209602  | 2.7001663685840476
R_OC   | 4      | 0.0               | 4.0                | 0.0
R_AP   | 2.5    | 48.27915529442809 | 1.6637548603092185 | 1.8659902906492987

Joint | x                  | y
------+--------------------+-------------------
A     | 1.4142135623730951 | 1.4142135623730951
B     | 5.307326119209602  | 2.7001663685840476
C     | 4.0                | 0.0
```

```
O      | 0                  | 0
P      | 3.0779684226823134 | 3.280203853022394
```

To prove that this is the correct answer, this method will calculate the distances between all the points so that we can see that they match the lengths provided.

[15]:
```
mech.calculate()
```

```
Distances:
- O to A: 2.0
- A to B: 4.099999999999989
- C to B: 3.0000000000000004
- O to C: 4.0
- A to P: 2.5
```

[16]:
```
# Getting vector A to P
f = a + e
f.get_length()
f
```

[16]:
```
Vector(joints=(Joint(name=O), Joint(name=P))), length=4.498180401941702,
angle=0.8171946078320964)
```

[17]:
```
# Vector f angle in degrees
np.rad2deg(f.angle)
```

[17]:
```
46.82180206962757
```

2.1.1 Answers:

$\theta_3 = 18.27915529442809°$

$\theta_4 = 64.16535766360016°$

The angle of vector $\overrightarrow{e}$ is $\theta_{AP} = 48.27915529442809°$

The polor coordinates of the position vector from O to P is (4.498180401941702, 46.82180206962757°)

2.2 $\theta_2 = 87°$

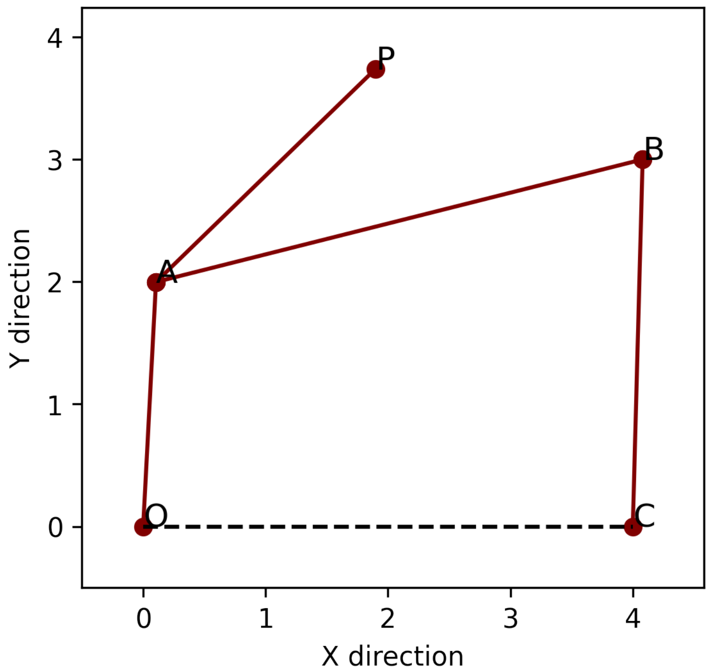
[18]:
```
# Solving for the values of theta_3 and theta_4 when theta_2 is 87 degrees
mech.clear_joints()
guess = np.deg2rad([20, 60])
solution = fsolve(position_loop, guess, args=(np.deg2rad(87), ))
np.rad2deg(solution)
```

[18]:
```
array([14.14093814, 88.46366837])
```

[19]:
```
e.get_x(np.deg2rad(30) + solution[0])
e.get_y(np.deg2rad(30) + solution[0])
```

[19]: 1.7410643097965348

[20]:
```
mech.fix_position()
mech.plot(cushion=0.5)
mech.tables(position=True)
```




```
POSITION
--------

Vector | Length | Angle              | x                   | y
-------+--------+--------------------+---------------------+-------------------
R_OA   | 2      | 87.0               | 0.10467191248588793 | 1.9972590695091477
R_AB   | 4.1    | 14.140938135101566 | 3.9757605844603874  | 1.0016625055632233
R_CB   | 3      | 88.46366836775566  | 0.08043249694627211 | 2.998921575072444
R_OC   | 4      | 0.0                | 4.0                 | 0.0
R_AP   | 2.5    | 44.140938135101564 | 1.7940722028816778  | 1.7410643097965348

Joint | x                   | y
------+---------------------+-------------------
A     | 0.10467191248588793 | 1.9972590695091477
B     | 4.080432496946272   | 2.998921575072444
C     | 4.0                 | 0.0
```

```
O      | 0                   | 0
P      | 1.8987441153675657  | 3.7383233793056823
```

[21]:
```
mech.calculate()
```

```
Distances:
- O to A: 1.9999999999999998
- A to B: 4.100000000000014
- C to B: 3.0
- O to C: 4.0
- A to P: 2.5
```

[22]:
```
f = a + e
f.get_length()
f
```

[22]:
```
Vector(joints=(Joint(name=O), Joint(name=P))), length=4.192885748968891,
angle=1.1008381471903976)
```

[23]:
```
np.rad2deg(f.angle)
```

[23]:
```
63.07337976101109
```

2.2.1 Answers:

$\theta_3 = 14.140938135101566°$

$\theta_4 = 88.46366836775566°$

The angle of vector $\vec{e}$ is $\theta_{AP} = 44.140938135101564°$

The polor coordinates of the position vector from O to P is (4.192885748968891, 63.07337976101109°)

2.3 $\theta_2 = 134°$

[24]:
```
# Solving for the values of theta_3 and theta_4 when theta_2 is 134 degrees
mech.clear_joints()
guess = np.deg2rad([20, 60])
solution = fsolve(position_loop, guess, args=(np.deg2rad(134), ))
np.rad2deg(solution)
```

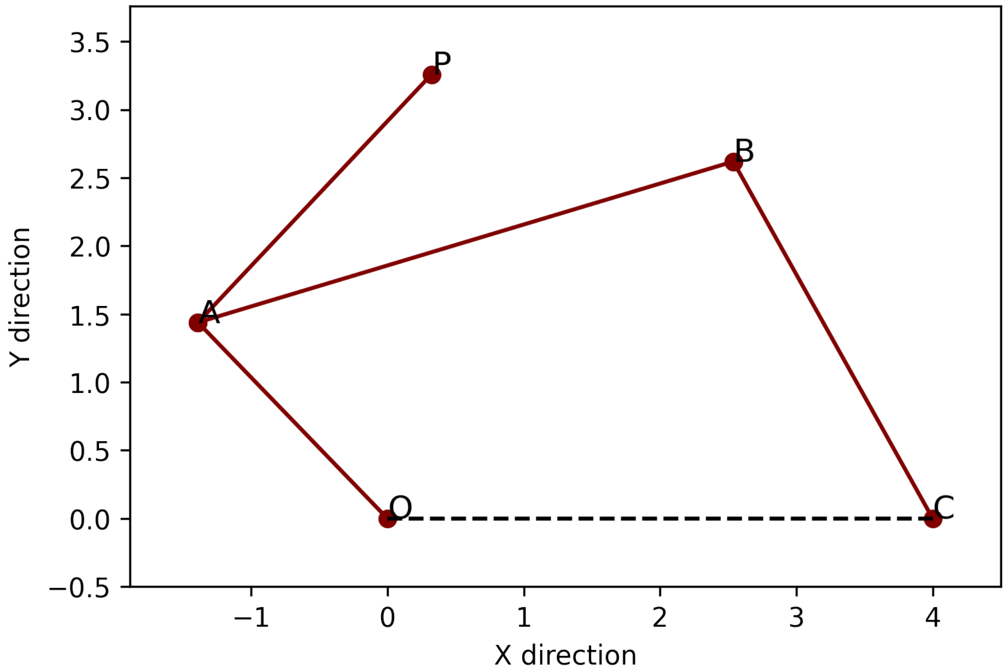
[24]:
```
array([ 16.73316184, 119.18573817])
```

[25]:
```
e.get_x(np.deg2rad(30) + solution[0])
e.get_y(np.deg2rad(30) + solution[0])
```

[25]:
```
1.8204239399750564
```

[26]:

```
mech.fix_position()
mech.plot(cushion=0.5)
mech.tables(position=True)
```




```
POSITION
--------

Vector | Length | Angle              | x                   | y
-------+--------+--------------------+---------------------+-------------------
R_OA   | 2      | 134.0              | -1.3893167409179947 | 1.4386796006773022
R_AB   | 4.1    | 16.733161844004133 | 3.9263896612363984  | 1.1804508579927901
R_CB   | 3      | 119.18573817282972 | -1.4629270796817053 | 2.619130458670197
R_OC   | 4      | 0.0                | 4.0                 | 0.0
R_AP   | 2.5    | 46.73316184400413  | 1.713492538287136   | 1.8204239399750564

Joint | x                   | y
------+---------------------+-------------------
A     | -1.3893167409179947 | 1.4386796006773022
B     | 2.5370729203182947  | 2.619130458670197
C     | 4.0                 | 0.0
O     | 0                   | 0
P     | 0.32417579736914126 | 3.2591035406523585
```

[27]:

```
mech.calculate()
```

```
Distances:
- O to A: 2.0
- A to B: 4.099999999999925
- C to B: 3.0
- O to C: 4.0
- A to P: 2.5
```

[28]:
```
f = a + e
f.get_length()
f
```

[28]:
```
Vector(joints=(Joint(name=O), Joint(name=P))), length=3.275186381916708,
angle=1.4716546514890805)
```

[29]:
```
np.rad2deg(f.angle)
```

[29]:
```
84.31960043112036
```

2.3.1 Answers:

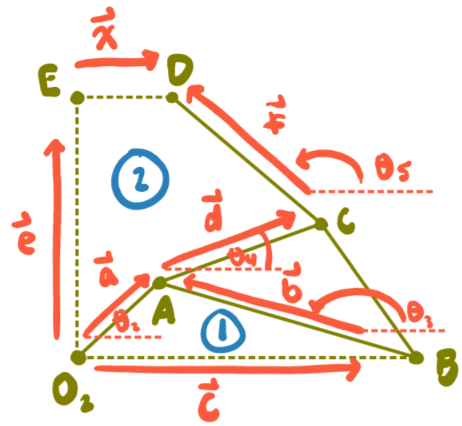
$\theta_3 = 16.733161844004133°$

$\theta_4 = 119.18573817282972°$

The angle of vector $\overrightarrow{e}$ is $\theta_{AP} = 46.73316184400413°$

The polor coordinates of the position vector from O to P is (3.275186381916708, 84.31960043112036°)

3 Problem 3



The numbers in blue are the loop reference.

3.1 Knowns and Unknowns:

3.1.1 Answer:

A letter without the vector notation denotes a length. If the cell contains a variable then it is unknown.

Vector	Length	Angle
$\overrightarrow{a}$	2	θ_2
$\overrightarrow{b}$	10	θ_3
$\overrightarrow{c}$	c	$0°$
$\overrightarrow{d}$	4	θ_4
$\overrightarrow{e}$	e	$90°$
$\overrightarrow{f}$	8	θ_5
$\overrightarrow{x}$	3	$0°$

3.2 Vector Loops

3.2.1 Loop 1

$\overrightarrow{a} - \overrightarrow{b} - \overrightarrow{c} = 0$

3.2.2 Answer:

In the complex vector form:

$ae^{i\theta_2} - be^{i\theta_3} - c = 0$

Applying Euler's Identity and Magnitudes:

$2cos(\theta_2) + 2isin(\theta_2) - 10cos(\theta_3) - 10isin(\theta_3) - c = 0$

Collecting the real and imaginary terms, a system of equations may be defined:

$$\begin{cases} 2cos(\theta_2) - 10cos(\theta_3) - c = 0 \\ 2sin(\theta_2) - 10sin(\theta_3) = 0 \end{cases}$$

3.2.3 Loop 2

$\overrightarrow{e} + \overrightarrow{x} - \overrightarrow{f} - \overrightarrow{d} - \overrightarrow{a} = 0$

3.2.4 Answer:

In the complex vector form:

$ee^{90i} + x - fe^{i\theta_5} - de^{i\theta_4} - ae^{i\theta_2} = 0$

Applying Euler's Identity and Magnitudes:

$ie + 3 - 8cos(\theta_5) - 8isin(\theta_5) - 4cos(\theta_4) - 4isin(\theta_4) - 2cos(\theta_2) - 2isin(\theta_2) = 0$

Collecting the real and imaginary terms, a system of equations may be defined:

$$\begin{cases} 3 - 8cos(\theta_5) - 4cos(\theta_4) - 2cos(\theta_2) = 0 \\ e - 8sin(\theta_5) - 4sin(\theta_4) - 2sin(\theta_2) = 0 \end{cases}$$

3.3 Finding the Stroke of Point B

The stroke of point B is calculated by finding the maximum length of $\overrightarrow{c}$ and subtracting the its minimum length. Because points O_2 and B are in-line, the maximum value occurs when θ_2 is 0° and the minmum occurs when θ_2 is 180°. This means that the maximum value is $a + b = 2 + 10 = 12in$ and the minimum value is $b - a = 10 - 2 = 8in$.

3.3.1 Answer:

- $B_{max} = 12in$
- $B_{min} = 8in$
- $stroke = B_{max} - B_{min} = 12 - 8 = 4in$

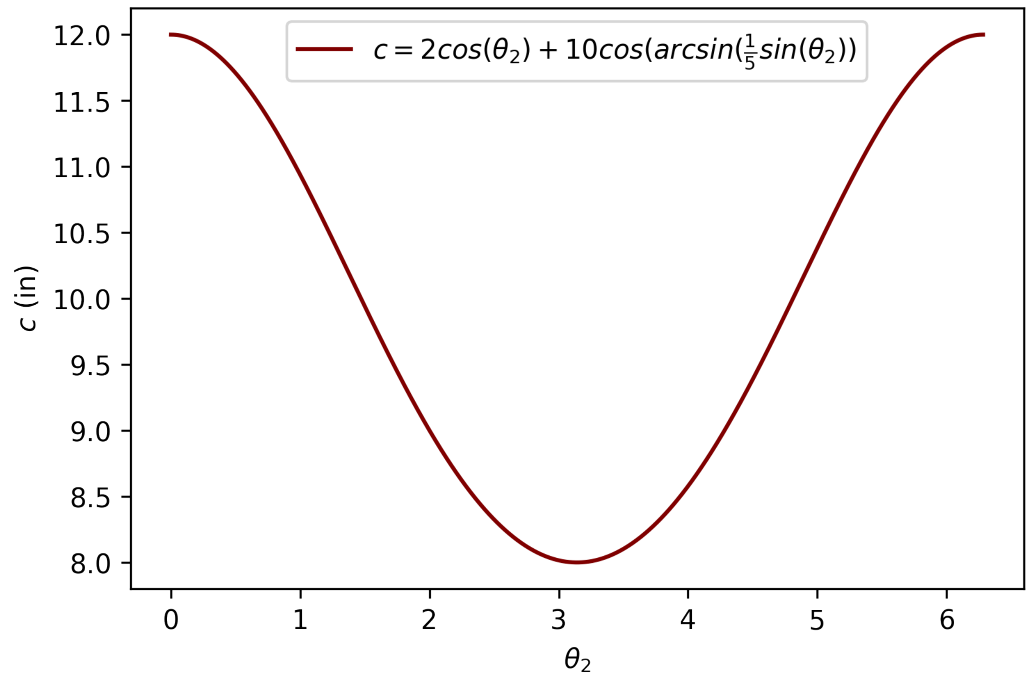
3.4 Finding B_x as a Function of θ_2

In order to obtain the length equation of $\overrightarrow{c}$ in the x direction ($c_x = c = f(\theta_2)$), the position loop equation (loop 1) must be solved analytically. Here is the analytical solution: $c = 2cos(\theta_2) - 10cos(arcsin(\frac{1}{5}sin(\theta_2)))$.

3.4.1 Answer (Plotting the Function):

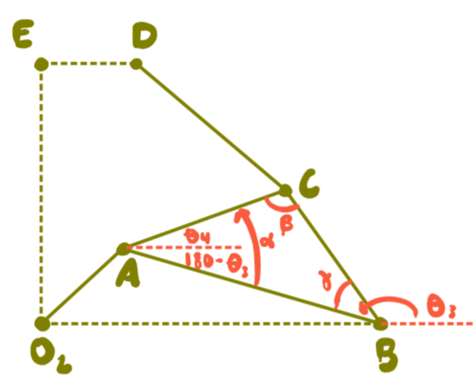
[30]:
```
# Plotting on the interval from 0 to 2pi
f = lambda x: 2*np.cos(x)+10*np.cos(np.arcsin(0.2*np.sin(x)))
d = np.linspace(0, 2*np.pi, 1000)
plt.plot(d, f(d), color='maroon',␣
 ↪label=r'$c=2cos(\theta_2)+10cos(arcsin(\frac{1}{5}sin(\theta_2))$')
plt.legend()
plt.xlabel(r'$\theta_2$')
plt.ylabel(r'$c$ (in)')
```

```
plt.show()
# Note: I changed it to + 10cos(arcsin...) so that we wouldn't see the crossed
↪solution from page 186.
```



3.5 Finding D_y as a Function of θ_2 BONUS

In order to find $e_y(\theta_2) = e(\theta_2)$, the relationship between θ_4 and θ_3 must be established from the Law of Cosines in the figure below. Also, notice that there is also this relationship between θ_3 and θ_2 from the first postition loop equation: $\theta_3 = arcsin(\frac{1}{5}sin(\theta_2))$



From the above figure: $\alpha = \theta_4 + 180° - \theta_3$.

[31]:
```
# Solving for alpha using the law of cosines.
alpha = np.arccos((7**2-10**2-4**2)/(-2*4*10))
```

```
alpha  # This is in radians
```

[31]: 0.5781043645663436

Now we can begin by using the loop 2 equations in this form:

$$\begin{cases} 8cos(\theta_5) = 3 - 4cos(\theta_4) - 2cos(\theta_2) \\ 8sin(\theta_5) = e - 4sin(\theta_4) - 2sin(\theta_2) \end{cases}$$

Now if you square both sides of the equation then add, the θ_5 terms go away via the pythagorean identity.

$$64 = (3 - 4cos(\theta_4) - 2cos(\theta_2))^2 + (e - 4sin(\theta_4) - 2sin(\theta_2))^2$$

$$e = \sqrt{64 - (3 - 4cos(\theta_4) - 2cos(\theta_2))^2} + 4sin(\theta_4) + 2sin(\theta_2)$$

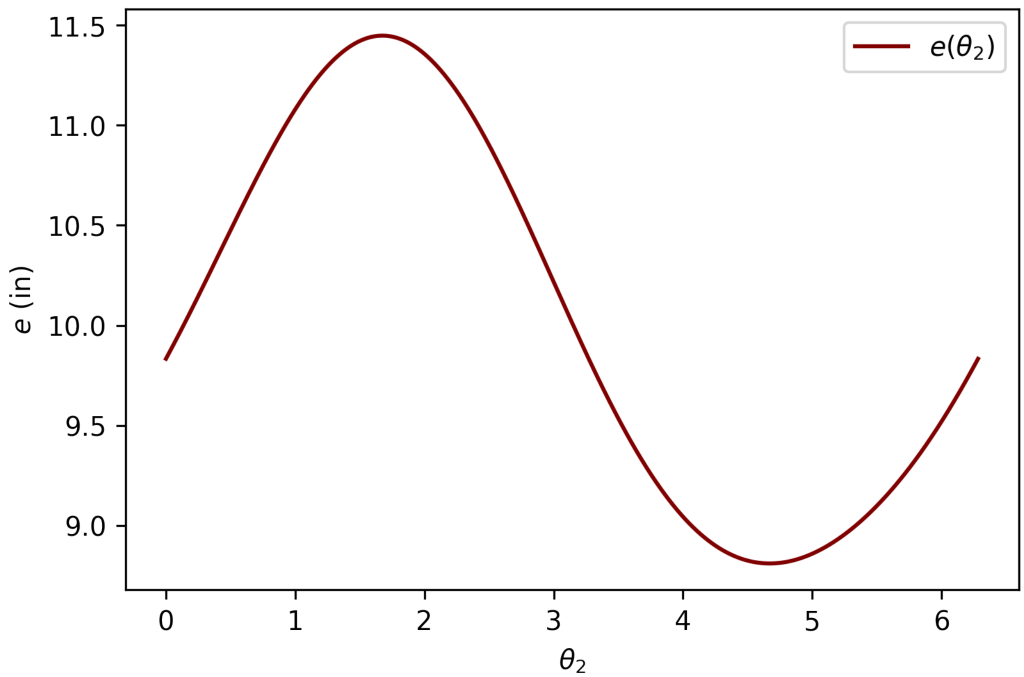
With the above relationships between θ_2, θ_3, and θ_4, e can now be determined as a function of θ_2 alone by substituting $\theta_4 = \alpha - \pi + arcsin(\frac{1}{5}sin(\theta_2))$.

3.6 Finding the Stroke of Point D BONUS

The stroke can be found by determining the maximum and minimum value of $e(\theta_2)$ and taking the difference between the two. Note: The length of e is the y value of point F.

3.6.1 Plotting $e(\theta_2)$ BONUS

[32]:
```
def e(t2):
    # Because theta_3 is in the 2nd and 3rd quadrant, you have to do some hand␣
↪waving to get the real value of
    # theta_3 which affects the real value of theta_4
    t4 = alpha - np.arcsin(0.2*np.sin(t2))

    return np.sqrt(64 - (3 - 4*np.cos(t4) - 2*np.cos(t2))**2) + 4*np.sin(t4) +␣
↪2*np.sin(t2)

d = np.linspace(0, 2*np.pi, 10_000)
e_values = e(d)
plt.plot(d, e_values, label=r'$e(\theta_2)$', color='maroon')
plt.legend()
plt.xlabel(r'$\theta_2$')
plt.ylabel(r'$e$ (in)')
plt.show()
```



[33]:
```
# Getting the maximum and minimum values
e_max = np.max(e_values)
e_max
```

[33]: 11.448596166062995

[34]:
```
e_min = np.min(e_values)
e_min
```

[34]: 8.809289404889084

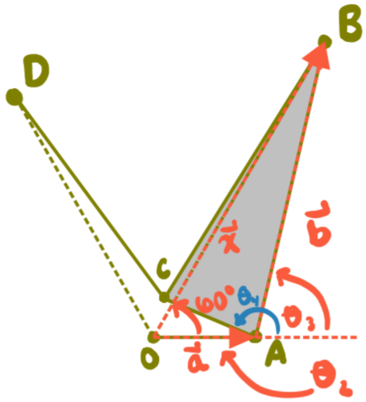
[35]:
```
stroke = e_max - e_min
stroke
```

[35]: 2.639306761173911

3.6.2 Answers BONUS:

- $D_{max} = 11.448596166062995 in$
- $D_{min} = 8.809289404889084 in$
- $stroke = 2.639306761173911 in$

4 Problem 4



4.1 Loop 1 Known and Unknowns

Vector	Length	Angle
$\vec{a}$	2	-50πt
$\vec{b}$	6	$\theta_3(t)$
$\vec{x}$	$x(t)$	60°

The $\theta_2(t)$ comes from integrating $\omega_2(t) = -1500rmp = -50\pi\frac{rad}{s}$

[36]:
```
t = sp.Symbol('t')
t2 = sp.Symbol('theta_2')
t2_t = -50*sp.pi*t
x = sp.Symbol('x')
t3 = sp.Symbol('theta_3')

a, b = 2, 6
x_angle = sp.pi/3
```

4.2 Position Loop 1

$\vec{a} + \vec{b} - \vec{x} = 0$

$$\begin{cases} 2cos(\theta_2) + 6cos(\theta_3) - xcos(60) = 0 \\ 2sin(\theta_2) + 6sin(\theta_3) - xsin(60) = 0 \end{cases}$$

[37]:
```
# Solving for x and theta_3 as functions of theta_2
p1 = a*sp.cos(t2) + b*sp.cos(t3) - x*sp.cos(x_angle)
p2 = a*sp.sin(t2) + b*sp.sin(t3) - x*sp.sin(x_angle)

p_loop1 = solve([p1, p2], (x, t3), dict=True)
p_loop1
```

[37]:
```
[{theta_3: -acos(-sqrt(34 - 2*cos(2*theta_2 + pi/3))/12 - sqrt(3)*cos(theta_2 +
pi/6)/6) + 2*pi,
  x: -sqrt(34 - 2*cos(2*theta_2 + pi/3)) + 2*sin(theta_2 + pi/6)},
 {theta_3: -acos(sqrt(34 - 2*cos(2*theta_2 + pi/3))/12 - sqrt(3)*cos(theta_2 +
pi/6)/6) + 2*pi,
  x: sqrt(34 - 2*cos(2*theta_2 + pi/3)) + 2*sin(theta_2 + pi/6)},
 {theta_3: acos(-sqrt(34 - 2*cos(2*theta_2 + pi/3))/12 - sqrt(3)*cos(theta_2 +
pi/6)/6),
  x: -sqrt(34 - 2*cos(2*theta_2 + pi/3)) + 2*sin(theta_2 + pi/6)},
 {theta_3: acos(sqrt(34 - 2*cos(2*theta_2 + pi/3))/12 - sqrt(3)*cos(theta_2 +
pi/6)/6),
  x: sqrt(34 - 2*cos(2*theta_2 + pi/3)) + 2*sin(theta_2 + pi/6)}]
```

I will chose the last solution because x and θ_3 are positive for all values of θ_2.

[38]:
```python
x_t2 = p_loop1[-1][x]   # x(theta_2)
t3_t2 = p_loop1[-1][t3]   # t3(theta_2)

# Substituting -50pi*t in for theta_2
x_t = x_t2.subs(t2, t2_t)
t3_t = t3_t2.subs(t2, t2_t)
```

[39]:
```python
# x as a function of time
x_t
```

[39]:

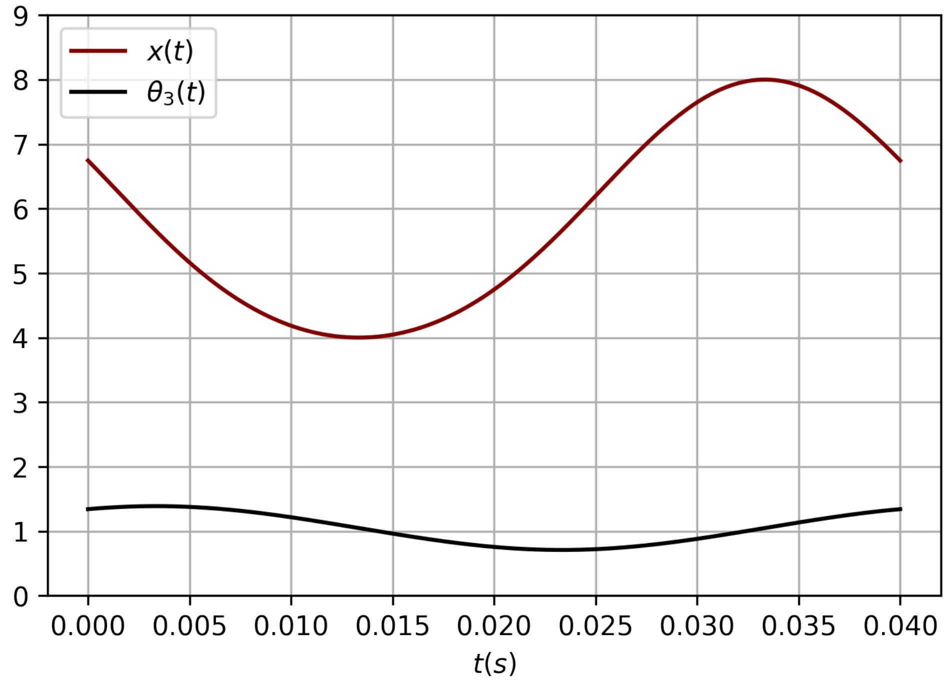
$$\sqrt{34 - 2\sin\left(100\pi t + \frac{\pi}{6}\right)} + 2\cos\left(50\pi t + \frac{\pi}{3}\right)$$

[40]:
```python
# Theta_3 as a function of time
t3_t
```

[40]:

$$\operatorname{acos}\left(\frac{\sqrt{34 - 2\sin\left(100\pi t + \frac{\pi}{6}\right)}}{12} - \frac{\sqrt{3}\sin\left(50\pi t + \frac{\pi}{3}\right)}{6}\right)$$

[41]:
```python
# Plotting x(t) and t3(t)
x_t_lamb = sp.lambdify(t, x_t, modules=['numpy'])
t3_t_lamb = sp.lambdify(t, t3_t, modules=['numpy'])

d = np.linspace(0, 0.04, 1000)
plt.plot(d, x_t_lamb(d), color='maroon', label=r'$x(t)$')
plt.plot(d, t3_t_lamb(d), color='black', label=r'$\theta_3(t)$')
plt.legend()
plt.xlabel(r'$t(s)$')
plt.ylim(0, 9)
plt.grid()
plt.show()
```



4.2.1 Acceleration at Point B Answer:

Just from this information alone, the acceleration of point B can now be determined.

$\overrightarrow{a_B} = \overrightarrow{a_{BO}} + \overrightarrow{a_O}$

The acceleration of point O is zero.

$\overrightarrow{a_B} = \overrightarrow{a_{BO}}$

Since the acceleration from O to B is purely translational, $\overrightarrow{a_{BO}} = \ddot{x}$. Therefore, $\overrightarrow{a_B} = \ddot{x}$.

[42]:
```
# Getting the 2nd order derivative of x with respect to time
x_t__ = x_t.diff(t, 2)  # Two underscores at the end of the variable␣
↪declaration denotes a second order derivative
x_t__
```

[42]:

$$2500\pi^2 \left(-2\cos\left(\pi\left(50t + \frac{1}{3}\right)\right) + \frac{2\sqrt{2}\sin\left(\pi\left(100t + \frac{1}{6}\right)\right)}{\sqrt{17 - \sin\left(\pi\left(100t + \frac{1}{6}\right)\right)}} - \frac{\sqrt{2}\cos^2\left(\pi\left(100t + \frac{1}{6}\right)\right)}{\left(17 - \sin\left(\pi\left(100t + \frac{1}{6}\right)\right)\right)^{\frac{3}{2}}} \right)$$

4.2.2 Another Way to Get the Answer

The acceleration of B can also be found by using this path:

$\overrightarrow{a_B} = \overrightarrow{a_{AB}} + \overrightarrow{a_A}$

Now we must find $\overrightarrow{a_A}$ which is equal to $\overrightarrow{a_{OA}}$. Because $\alpha_2 = 0$, the acceleration will only consist of the radial component.

$\overrightarrow{a_A} = -2\omega_2^2 e^{i\theta_2}$

[43]:
```
# Getting acceleration of point A in the cartesian form.
e1 = lambda x: sp.Matrix([sp.cos(x), sp.sin(x)])  # Returns <sin(theta),␣
↪cos(theta)>
e2 = lambda x: sp.Matrix([-sp.sin(x), sp.cos(x)])  # Returns <-sin(theta),␣
↪cos(theta)>
mag = lambda m: sp.sqrt(m[0]**2 + m[1]**2)  # Returns the magnitude of a vector

A_vector = -a*t2_t.diff(t)**2*e1(t2_t)
A_vector
```

[43]: $\begin{bmatrix} -5000\pi^2 \cos(50\pi t) \\ 5000\pi^2 \sin(50\pi t) \end{bmatrix}$

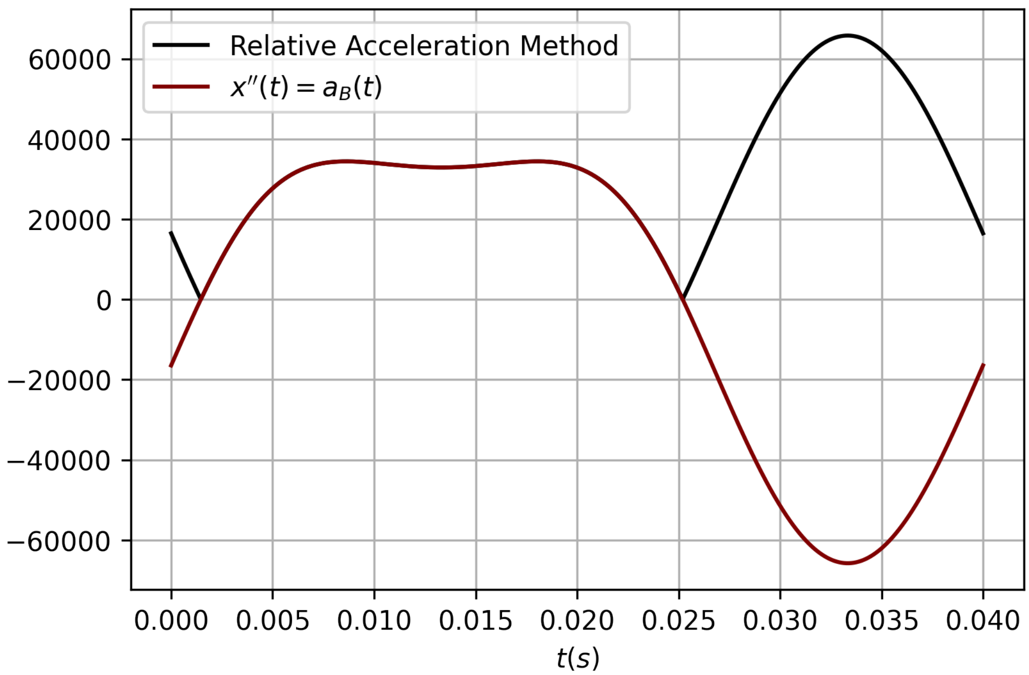
[44]:
```
mag(A_vector).simplify()  # Just making sure that this is working
```

[44]: $5000\pi^2$

Now the value of $\overrightarrow{a_{AB}}$ can be considered. Since the vector from A to B is pure rotation, the following is true:

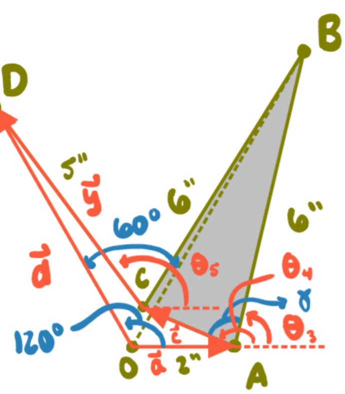
$\overrightarrow{a_{AB}} = 6\alpha_3 i e^{i\theta_3} - 6\omega_3^2 e^{i\theta_3}$

[45]:
```
# Getting the acceleration vector from A to B
AB_vector = b*t3_t.diff(t, 2)*e2(t3_t) - b*t3_t.diff(t)**2*e1(t3_t)

# Summing the two vectors
B_vector = AB_vector + A_vector
B_other = mag(B_vector)

# Plotting the two functions to compare
B_other_lamb = sp.lambdify(t, B_other, modules=['numpy'])
x_t__lamb = sp.lambdify(t, x_t__, modules=['numpy'])

plt.plot(d, B_other_lamb(d), color='black', label='Relative Acceleration␣
↪Method')
plt.plot(d, x_t__lamb(d), color='maroon', label=r"$x''(t)=a_{B}(t)$")
plt.legend()
plt.xlabel(r'$t(s)$')
plt.grid()
plt.show()
```



As seen from above, the two methods agree with each other. The relative acceleration is a magnitude which is why all the values are above 0.

4.3 Loop 2 Known and Unkowns



Vector	Length	Angle
$\vec{a}$	2	-$50\pi t$
$\vec{c}$	2	$\theta_4(t)$
$\vec{y}$	5	$\theta_5(t)$
$\vec{d}$	$d(t)$	120°

```
[46]: # Solving for gamma via law of cosines
gamma = sp.acos((6**2 - 6**2 - 2**2)/(-2*6*2))
gamma
```

```
[46]: 1.40334824757521
```

```
[47]: t4_t = t3_t + gamma
t4, t5, d = sp.symbols('theta_4 theta_5 d')

d_angle = 2*sp.pi/3
c, y = 2, 5
```

4.4 Position Loop 2

$\vec{a} + \vec{c} + \vec{y} - \vec{d} = 0$

$$\begin{cases} 2cos(\theta_2) + 2cos(\theta_4) + 5cos(\theta_5) - dcos(120) = 0 \\ 2sin(\theta_2) + 2sin(\theta_4) + 5sin(\theta_5) - dsin(120) = 0 \end{cases}$$

```
[48]: # Solving the system for d and theta_5
p3 = a*sp.cos(t2_t) + c*sp.cos(t4_t) + y*sp.cos(t5) - d*sp.cos(d_angle)
p4 = a*sp.sin(t2_t) + c*sp.sin(t4_t) + y*sp.sin(t5) - d*sp.sin(d_angle)

# p_loop2 = solve([p3, p4], (d, t5), dict=True, quick=True)
# p_loop2
```

4.4.1 Acceleration of Point D Answer

I tried running the above cell, but sympy cannot handle solving this symbolically. Nevertheless, the solution of that system would get $d(t)$ then differentiating with respect to time twice would get the acceleration of point D because $\overrightarrow{a_D} = \overrightarrow{a_{OD}} + \overrightarrow{a_O}$. With $\overrightarrow{a_O}$ being equal to zero, that would mean that $\overrightarrow{a_{OD}} = \overrightarrow{a_D}$ and $\overrightarrow{a_{OD}}$ is purely translational. This is similar to the way the acceleration of point B was solved.

The above system can be solved for $d(t)$ by hand. First, place the system in this form:

$$\begin{cases} 2cos(\theta_2) + 2cos(\theta_4) - dcos(120) = -5cos(\theta_5) \\ 2sin(\theta_2) + 2sin(\theta_4) - dsin(120) = -5sin(\theta_5) \end{cases}$$

Now if you square both sides then add the two equations, the θ_5 can be eliminated via Pythagorean Identity. This results in an equation with d as the only unknown.

$(2cos(\theta_2) + 2cos(\theta_4) - dcos(120))^2 + (2sin(\theta_2) + 2sin(\theta_4) - dsin(120))^2 = 25$

Sympy will solve for d in the cell below. This will result in a function of time.

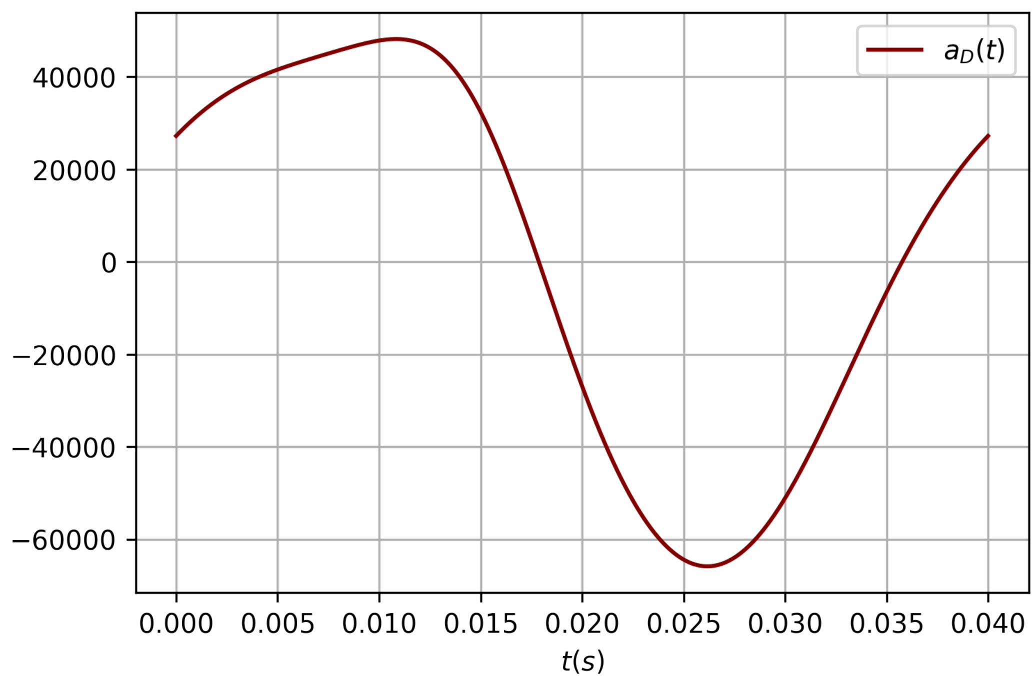
```
[49]: # Solving for d as a function of time
d_solutions = solve((2*sp.cos(t2_t) + 2*sp.cos(t4_t) - d*sp.cos(d_angle))**2 +␣
↪(2*sp.sin(t2_t) + 2*sp.sin(t4_t) - d*sp.sin(d_angle))**2 - 25, d, dict=True)
# d_solutions  Suppressing this output because it's too large
```

[50]:
```python
d_t = d_solutions[1][d]
# d_t  Suppressing this output because it's too large
```

The acceleration of d is $\ddot{d}$.

[51]:
```python
# getting the acceleration of point d as a function of time
d_t__ = d_t.diff(t, 2)
# d_t__  I am suppressing this output because it's too large
```

[52]:
```python
# Plotting the acceleration of point D
d_t__lamb = sp.lambdify(t, d_t__, modules=['numpy'])
period = np.linspace(0, 0.04, 1000)

plt.plot(period, d_t__lamb(period), color='maroon', label=r'$a_{D}(t)$')
plt.grid()
plt.legend()
plt.xlabel(r'$t(s)$')
plt.show()
```



4.4.2 Acceleration of Point C Answer

The acceleration of point C, however, can be solved by using relative acceleration with the already solved information. Consider the following relative acceleration equation:

$\overrightarrow{a_C} = \overrightarrow{a_{AC}} + \overrightarrow{a_A}$

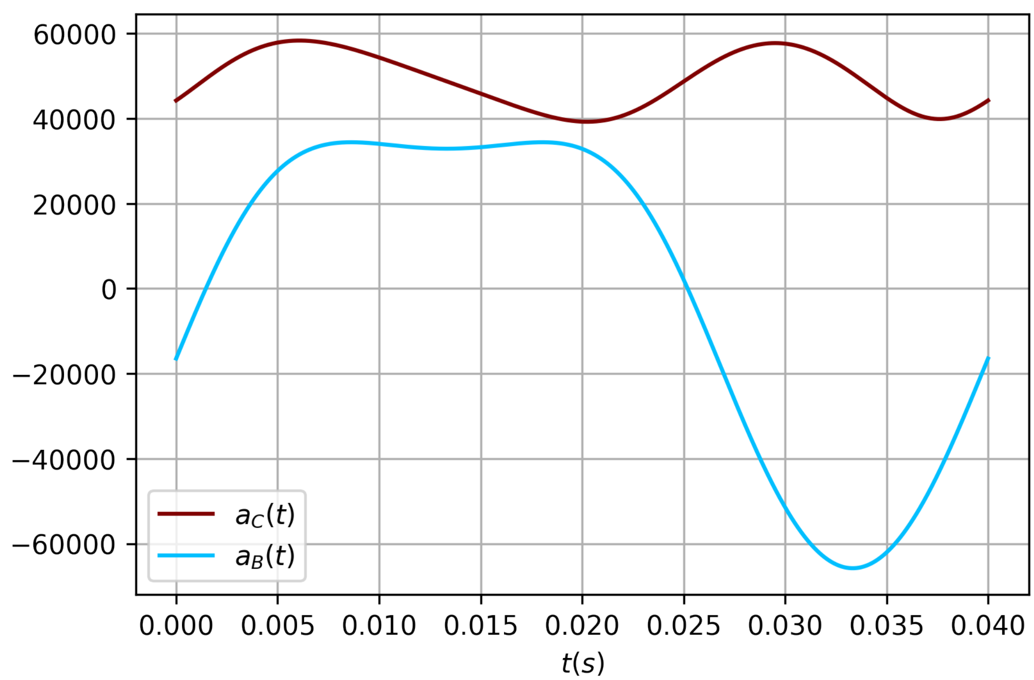
$\overrightarrow{a_A}$ was solved earlier, but since $\overrightarrow{a_{AC}}$ is purely rotational, the following is true:

$\overrightarrow{a_{AC}} = 2\alpha_3 i e^{i(\theta_3+\gamma)} - 2\omega_3^2 e^{i(\theta_3+gamma)}$

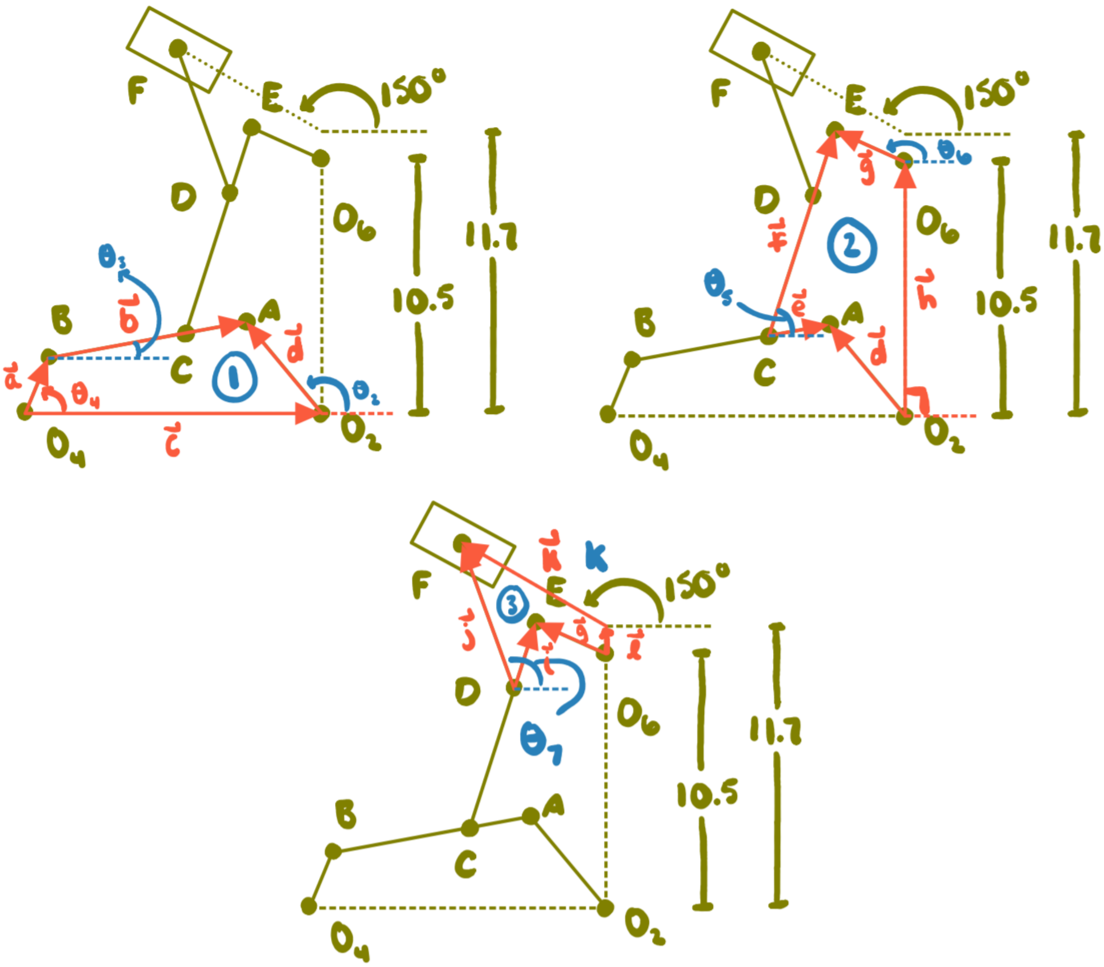
[53]:
```
# Calculating the acceleration of point C
C_vector = c*t4_t.diff(t, 2)*e2(t4_t) - c*t4_t.diff(t)**2*e1(t4_t) + A_vector
# mag(C_vector)  Suppressing this output because it's too large
```

This solution is ugly, but graphing it will show that it is correct.

[54]:
```
# Plotting acceleration of point C
C_vector_lamb = sp.lambdify(t, mag(C_vector), modules=['numpy'])
time = np.linspace(0, 0.04, 1000)  # This is the period
plt.plot(time, C_vector_lamb(time), label=r"$a_C(t)$", color='maroon')
plt.plot(time, x_t__lamb(time), label=r"$a_B(t)$", color='deepskyblue')
plt.legend()
plt.grid()
plt.xlabel(r'$t(s)$')
plt.show()
```



5 Problem 5



5.1 Known and Unkown Quantities

Vector	Length	Angle
$\overrightarrow{a}$	2.5	θ_4
$\overrightarrow{b}$	8.4	θ_3
$\overrightarrow{c}$	12.5	$0°$
$\overrightarrow{d}$	5	θ_2
$\overrightarrow{e}$	2.4	θ_3
$\overrightarrow{f}$	8.9	θ_5
$\overrightarrow{g}$	3.2	θ_6
$\overrightarrow{h}$	10.5	$90°$
$\overrightarrow{i}$	3	θ_5
$\overrightarrow{j}$	6.4	θ_7
$\overrightarrow{k}$	k	$150°$
$\overrightarrow{l}$	1.2	$0\circ$

There are 6 unknowns (excluding θ_4 because it's postion is the input) and each loop provides 2 equations which means that there are 6 equations, making this a consistent system.

5.2 Finding the Limit Position of θ_4

The linkage system for loop 1 is not a Grashof linkage.

[55]:
```
2.5 + 12.5 < 8.4 + 5
```

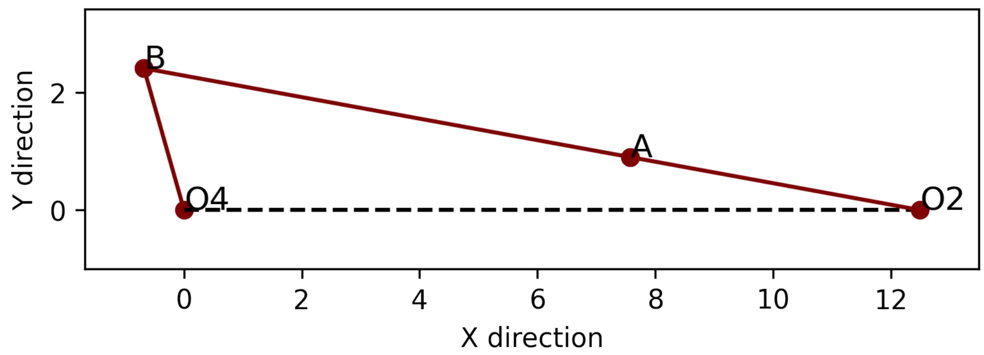
[55]:
```
False
```

The maximum value of θ_4 occurs when $\vec{d}$ and $\vec{b}$ are co-linear. The Law of Cosines could be used to determine this value, but a positon loop could also solve it.

[56]:
```
# Creating the limit position
A, B = get_joints('AB')
O2, O4 = Joint('O2'), Joint('O4')
a_limit = Vector((O4, B), length=2.5)
b_limit = Vector((A, B), length=8.4)
c_limit = Vector((O4, O2), length=12.5, angle=0, ls='--', color='black')
d_limit = Vector((O2, A), length=5)
limit = Mechanism(vectors=(a_limit, b_limit, c_limit, d_limit),␣
 ↪input_vector=a_limit)

# Implementing the position loop equation
limit_position_loop = lambda x: np.array([a_limit.get_x(x[0]) - b_limit.
 ↪get_x(x[1]) - d_limit.get_x(x[1]) - c_limit.length,
                                          a_limit.get_y(x[0]) - b_limit.
 ↪get_y(x[1]) - d_limit.get_y(x[1])])

solution = fsolve(limit_position_loop, np.deg2rad([120, 160]))
np.rad2deg(solution)
```

[56]:
```
array([105.8404803 , 169.66039052])
```

[57]:
```
limit.fix_position()
limit.plot()
limit.tables(position=True)
```




```
POSITION
--------

Vector | Length | Angle              | x                   | y
-------+--------+--------------------+---------------------+--------------------
R_O4B  | 2.5    | 105.84048029945892 | -0.6823999999999741 | 2.4050634586222532
R_AB   | 8.4    | 169.66039051940945 | -8.26359402985071   | 1.5076517203305604
R_O4O2 | 12.5   | 0.0                | 12.5                | 0.0
R_O2A  | 5      | 169.66039051940945 | -4.918805970149232  | 0.8974117382920002

Joint | x                   | y
------+---------------------+-------------------
A     | 7.581194029850735   | 0.8974117382916929
B     | -0.6823999999999741 | 2.4050634586222532
O2    | 12.5                | 0.0
O4    | 0                   | 0
```

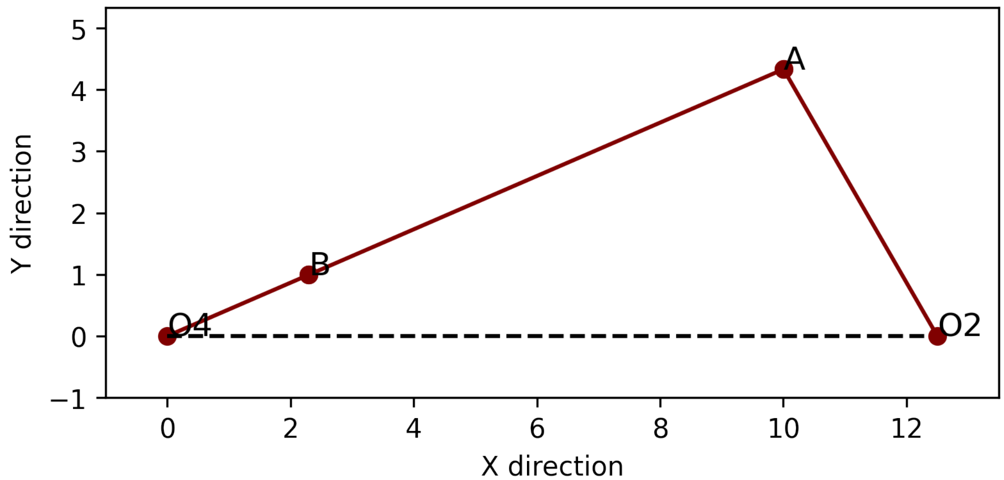
From the above data, $\theta_4 = 105.84048029945892°$. Lets calculate the other limit position to check if the solution half way between $0°$ and θ_4 max exists.

[58]:
```
A, B = get_joints('AB')
O2, O4 = Joint('O2'), Joint('O4')
a_limit = Vector((O4, B), length=2.5)
b_limit = Vector((B, A), length=8.4)
c_limit = Vector((O4, O2), length=12.5, angle=0, ls='--', color='black')
d_limit = Vector((O2, A), length=5)
limit2 = Mechanism(vectors=(a_limit, b_limit, c_limit, d_limit),␣
↪input_vector=a_limit)

limit_2 = lambda x: np.array([a_limit.get_x(x[0]) + b_limit.get_x(x[0]) -␣
↪d_limit.get_x(x[1]) - c_limit.length,
                              a_limit.get_y(x[0]) + b_limit.get_y(x[0]) -␣
↪d_limit.get_y(x[1])])
```

```python
solution = fsolve(limit_2, np.deg2rad([50, 40]))
np.rad2deg(solution)
```

[58]: array([23.41489115, 119.96824853])

[59]:
```python
# The solution is possible because theta_4 is 23.41 degrees which is less than half of its max
limit2.fix_position()
limit2.plot()
```



[60]:
```python
# Getting the half way point between 0 and theta_4's limit position
t4 = 105.84048029945892/2
t4
```

[60]: 52.92024014972946

5.3 Position Analysis

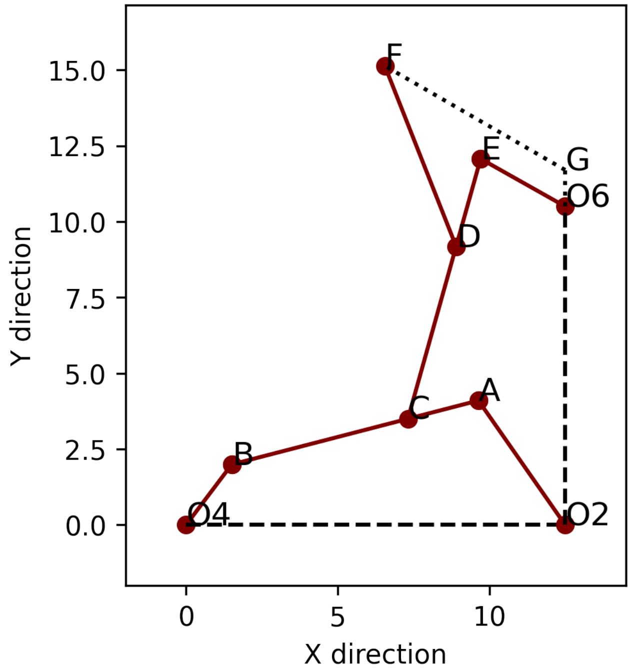
Position Loop Equations:

$$\begin{cases} 2cos(52.92°) + 8.4cos(\theta_3) - 5cos(\theta_2) - 12.5 = 0 \\ 2sin(52.92°) + 8.4sin(\theta_3) - 5sin(\theta_2) = 0 \\ 8.9cos(\theta_5) - 3.2cos(\theta_6) + 5cos(\theta_2) - 2.4cos(\theta_3) = 0 \\ 8.9sin(\theta_5) - 3.2sin(\theta_6) - 10.5 + 5sin(\theta_2) - 2.4sin(\theta_3) = 0 \\ 6.4cos(\theta_7) - kcos(150°) + 3.2cos(\theta_6) - 3cos(\theta_5) = 0 \\ 6.4sin(\theta_7) - ksin(150°) - 1.2 + 3.2sin(\theta_6) - 3sin(\theta_5) = 0 \end{cases}$$

[61]:
```python
# Defining the mechanism
A, B, C, D, E, F, G = get_joints('ABCDEFG')
O2, O4, O6 = Joint('O2'), Joint('O4'), Joint('O6')
a = Vector((O4, B), length=2.5)
b = Vector((B, A), length=8.4)
c = Vector((O4, O2), length=12.5, angle=0, ls='--', color='black')
d = Vector((O2, A), length=5)
e = Vector((C, A), length=2.4, show=False)
f = Vector((C, E), length=8.9)
g = Vector((O6, E), length=3.2)
h = Vector((O2, O6), length=10.5, angle=np.pi/2, ls='--', color='black')
i = Vector((D, E), length=3, show=False)
j = Vector((D, F), length=6.4)
k = Vector((G, F), angle=np.deg2rad(150), ls=':', color='black')
l = Vector((O6, G), length=1.2, angle=np.pi/2, ls=':', color='black')
mechanism = Mechanism(vectors=(a, b, c, d, e, f, g, h, i, j, k, l),␣
↪input_vector=a, omega=-30, alpha=0)

order = ('Theta 2', 'Theta 3', 'Theta 5', 'Theta 6', 'Theta 7', 'k')

position_loops = lambda t: np.array([
    a.get_x(np.deg2rad(t4)) + b.get_x(t[1]) - d.get_x(t[0]) - c.length,
    a.get_y(np.deg2rad(t4)) + b.get_y(t[1]) - d.get_y(t[0]),
    f.get_x(t[2]) - g.get_x(t[3]) + d.get_x(t[0]) - e.get_x(t[1]),
    f.get_y(t[2]) - g.get_y(t[3]) - h.length + d.get_y(t[0]) - e.get_y(t[1]),
    j.get_x(t[4]) - k.get_unknown_length(t[5])*np.cos(k.angle) + g.get_x(t[3])␣
↪- i.get_x(t[2]),
    j.get_y(t[4]) - k.get_unknown_length(t[5])*np.sin(k.angle) - l.length + g.
↪get_y(t[3]) - i.get_y(t[2]),
])

guess = np.concatenate((np.deg2rad([120, 20, 70, 170, 120]), np.array([7])))
p_loop = fsolve(position_loops, guess)
p_loop
```

[61]:
```
array([2.17998043, 0.25342828, 1.29821101, 2.62870235, 1.94696281,
       6.86701739])
```

[62]:
```python
mechanism.fix_position()
mechanism.plot(cushion=2)
mechanism.tables(position=True, to_five=True)
```




```
POSITION
--------

Vector | Length   | Angle     | x         | y
-------+----------+-----------+-----------+---------
R_O4B  | 2.50000  | 52.92024  | 1.50732   | 1.99449
R_BA   | 8.40000  | 14.52037  | 8.13169   | 2.10608
R_O4O2 | 12.50000 | 0.00000   | 12.50000  | 0.00000
R_O2A  | 5.00000  | 124.90368 | -2.86099  | 4.10058
R_CA   | 2.40000  | 14.52037  | 2.32334   | 0.60174
R_CE   | 8.90000  | 74.38201  | 2.39608   | 8.57140
R_O6E  | 3.20000  | 150.61355 | -2.78826  | 1.57023
R_O2O6 | 10.50000 | 90.00000  | 0.00000   | 10.50000
R_DE   | 3.00000  | 74.38201  | 0.80767   | 2.88923
R_DF   | 6.40000  | 111.55275 | -2.35109  | 5.95251
R_GF   | 6.86702  | 150.00000 | -5.94701  | 3.43351
R_O6G  | 1.20000  | 90.00000  | 0.00000   | 1.20000

Joint | x                  | y
------+--------------------+-------------------
A     | 9.63900738965331   | 4.100575725863581
B     | 1.5073154945133524 | 1.9944924166313514
C     | 7.315666849322855  | 3.4988376360150624
D     | 8.904077792318697  | 9.180998376100744
E     | 9.711744373503024  | 12.070232650720582
```

```
F      | 6.552988491303877  | 15.133508695172324
G      | 12.499999999972339 | 11.699999999768787
O2     | 12.5               | 0.0
O4     | 0                  | 0
O6     | 12.5               | 10.5
```

[63]:
```
# Checking the distances
mechanism.calculate()
```

```
Distances:
- O4 to B: 2.5
- B to A: 8.399999998982823
- O4 to O2: 12.5
- O2 to A: 5.0
- C to A: 2.399999998982824
- C to E: 8.9
- O6 to E: 3.2000000025126614
- O2 to O6: 10.5
- D to E: 3.0
- D to F: 6.3999999999999995
- G to F: 6.867017390807077
- O6 to G: 1.199999999768787
```

5.4 Velocity Analysis

The velocity analysis will consist of the same loops defined earlier. Every unkown angle defined will now have an unkown angular velocity corresponding to the same number. $\overrightarrow{k}$ will only have an uknown slip velocity.

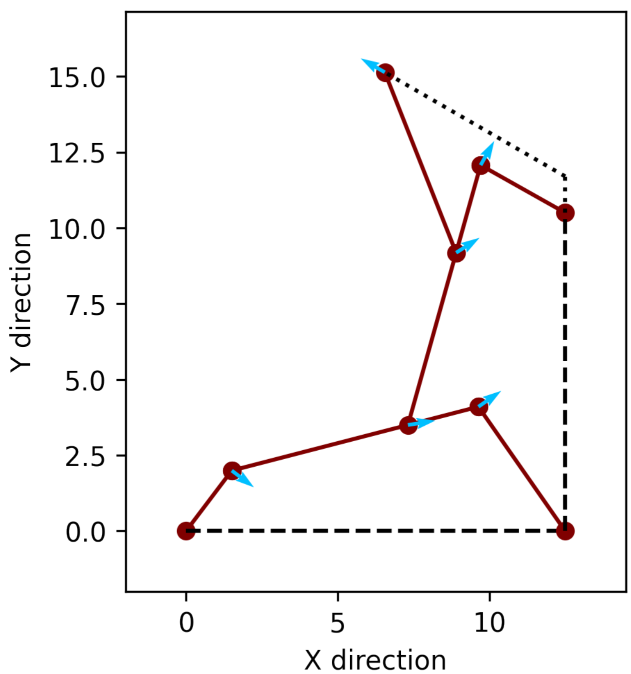
From the position analysis, all the values of θ_n are known.

$\omega_4 = -30\frac{rad}{s}$

$$\begin{cases} 2.5(-30)(-sin(52.92°)) + 8.4\omega_3(-sin(\theta_3)) - 5\omega_2(-sin(\theta_3)) = 0 \\ 2.5(-30)(cos(52.92°)) + 8.4\omega_3(cos(\theta_3)) - 5\omega_2(cos(\theta_3)) = 0 \\ 8.9\omega_5(-sin(\theta_5)) - 3.2\omega_6(-sin(\theta_6)) + 5\omega_2(-sin(\theta_2)) - 2.4\omega_3(-sin(\theta_3)) = 0 \\ 8.9\omega_5(cos(\theta_5)) - 3.2\omega_6(cos(\theta_6)) + 5\omega_2(cos(\theta_2)) - 2.4\omega_3(cos(\theta_3)) = 0 \\ 6.4\omega_7(-sin(\theta_7)) - \dot{k}cos(150°) + 3.2\omega_6(-sin(\theta_6)) - 3\omega_5(-sin(\theta_5)) = 0 \\ 6.4\omega_7(cos(\theta_7)) - \dot{k}sin(150°) + 3.2\omega_6(cos(\theta_6)) - 3\omega_5(cos(\theta_5)) = 0 \end{cases}$$

[64]:
```
a_v, b_v, c_v, d_v, e_v, f_v, g_v, h_v, i_v, j_v, k_v, l_v = mechanism.
↪get_velocities()

order2 = ('Omega 2', 'Omega 3', 'Omega 5', 'Omega 6', 'Omega 7', 'k_dot')

velocity_loops = lambda x: np.array([
    a_v.tan_x(a_v.omega) + b_v.tan_x(x[1]) - d_v.tan_x(x[0]),
```

```
    a_v.tan_y(a_v.omega) + b_v.tan_y(x[1]) - d_v.tan_y(x[0]),
    f_v.tan_x(x[2]) - g_v.tan_x(x[3]) + d_v.tan_x(x[0]) - e_v.tan_x(x[1]),
    f_v.tan_y(x[2]) - g_v.tan_y(x[3]) + d_v.tan_y(x[0]) - e_v.tan_y(x[1]),
    j_v.tan_x(x[4]) - k_v.slip_x(x[5]) + g_v.tan_x(x[3]) - i_v.tan_x(x[2]),
    j_v.tan_y(x[4]) - k_v.slip_y(x[5]) + g_v.tan_y(x[3]) - i_v.tan_y(x[2]),
])

guess = np.array([15, 15, 30, 12, 30, 3])
v_loop = fsolve(velocity_loops, guess)
mechanism.fix_velocity()
mechanism.plot(velocity=True, cushion=2)
mechanism.tables(velocity=True, to_five=True)
```




```
VELOCITY
--------

Vector | Mag      | Angle     | x         | y
-------+----------+-----------+-----------+----------
V_04B  | 75.00000 | 322.92024 | 59.83477  | -45.21946
V_BA   | 76.08683 | 104.52037 | -19.07681 | 73.65650
V_0402 | 0.00000  | 90.00000  | 0.00000   | 0.00000
V_02A  | 49.69785 | 34.90368  | 40.75796  | 28.43704
V_CA   | 21.73909 | 104.52037 | -5.45052  | 21.04472
V_CE   | 37.71928 | 164.38201 | -36.32661 | 10.15487
V_06E  | 20.13840 | 60.61355  | 9.88187   | 17.54719
```

```
V_O2O6 | 0.00000  | 90.00000  | 0.00000   | 0.00000
V_DE   | 12.71436 | 164.38201 | -12.24493 | 3.42299
V_DF   | 29.74448 | 201.55275 | -27.66474 | -10.92687
V_GF   | 6.39467  | 150.00000 | -5.53795  | 3.19734
V_O6G  | 0.00000  | 90.00000  | 0.00000   | 0.00000

Vector | Omega     | Slip Vel
-------+-----------+---------
V_O4B  | -30.00000 | 0.00000
V_BA   | 9.05796   | 0.00000
V_O4O2 | 0.00000   | 0.00000
V_O2A  | -9.93957  | 0.00000
V_CA   | 9.05796   | 0.00000
V_CE   | 4.23812   | 0.00000
V_O6E  | -6.29325  | 0.00000
V_O2O6 | 0.00000   | 0.00000
V_DE   | 4.23812   | 0.00000
V_DF   | 4.64758   | 0.00000
V_GF   | 0.00000   | 6.39467
V_O6G  | 0.00000   | 0.00000

Joint | Mag      | Angle     | x        | y
------+----------+-----------+----------+----------
A     | 49.69785 | 34.90368  | 40.75796 | 28.43704
B     | 75.00000 | 322.92024 | 59.83477 | -45.21946
C     | 46.79605 | 9.08903   | 46.20848 | 7.39232
D     | 26.25049 | 32.55132  | 22.12679 | 14.12420
E     | 20.13840 | 60.61355  | 9.88187  | 17.54719
F     | 6.39467  | 150.00000 | -5.53795 | 3.19734
G     | 0.00000  | 90.00000  | 0.00000  | 0.00000
O2    | 0.00000  | 90.00000  | 0.00000  | 0.00000
O4    | 0.00000  | 90.00000  | 0.00000  | 0.00000
O6    | 0.00000  | 90.00000  | 0.00000  | 0.00000
```

5.4.1 Velocity of Link 8 Answer

The velocity of link 8 is $6.39467\frac{units}{s}$

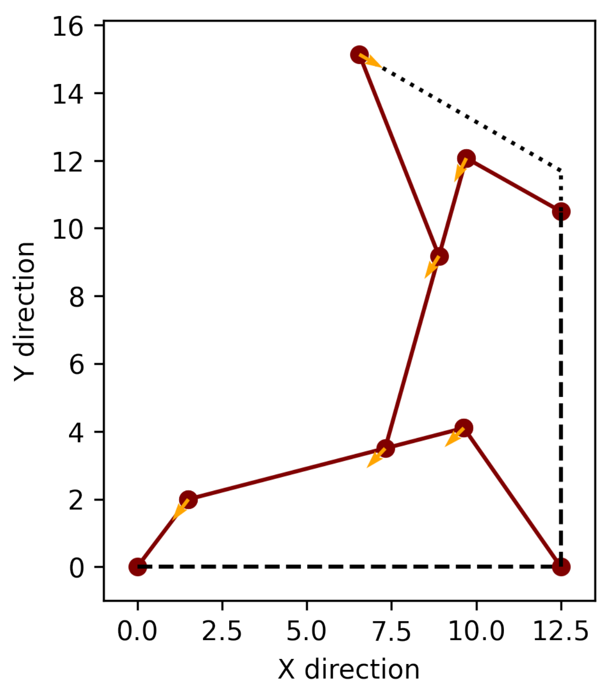
5.5 Acceleration Analysis

Similarly to the velocity analysis, only the angular accelerations for all the vectors are unkown except for $\vec{k}$ which has an unkown slip acceleration $\ddot{k}$. The value of $\ddot{k}$ is the acceleration of link 8.

$\alpha_4 = 0\frac{rad}{s^2}$

Look at the set up for the acceleration loop equation in the code below. All vectors will only have a normal and tangental component of acceleration with the exception of $\vec{k}$.

[65]:
```
a_a, b_a, c_a, d_a, e_a, f_a, g_a, h_a, i_a, j_a, k_a, l_a = mechanism.
↪get_accelerations()

order3 = ('Alpha 2', 'Alpha 3', 'Alpha 5', 'Alpha 6', 'Alpha 7', 'k_ddot')

acceleration_loops = lambda x: np.array([
    a_a.normal_tan_x(a_a.alpha) + b_a.normal_tan_x(x[1]) - d_a.
↪normal_tan_x(x[0]),
    a_a.normal_tan_y(a_a.alpha) + b_a.normal_tan_y(x[1]) - d_a.
↪normal_tan_y(x[0]),
    f_a.normal_tan_x(x[2]) - g_a.normal_tan_x(x[3]) + d_a.normal_tan_x(x[0]) -␣
↪e_a.normal_tan_x(x[1]),
    f_a.normal_tan_y(x[2]) - g_a.normal_tan_y(x[3]) + d_a.normal_tan_y(x[0]) -␣
↪e_a.normal_tan_y(x[1]),
    j_a.normal_tan_x(x[4]) - k_a.slip_x(x[5]) + g_a.normal_tan_x(x[3]) - i_a.
↪normal_tan_x(x[2]),
    j_a.normal_tan_y(x[4]) - k_a.slip_y(x[5]) + g_a.normal_tan_y(x[3]) - i_a.
↪normal_tan_y(x[2])
])

guess = np.array([10, 10, 30, -30, 20, 10])
a_loop = fsolve(acceleration_loops, guess)
mechanism.fix_acceleration()
mechanism.plot(acceleration=True)
mechanism.tables(acceleration=True, to_five=True)
```



ACCELERATION

Vector	Mag	Angle	x	y
A_O4B	2250.00000	232.92024	-1356.58395	-1795.04317
A_BA	690.38940	197.89664	-656.98313	-212.15724
A_O4O2	0.00000	90.00000	0.00000	0.00000
A_O2A	2843.11552	224.90928	-2013.56707	-2007.20042
A_CA	197.25412	197.89664	-187.70947	-60.61635
A_CE	762.80051	332.28495	675.28555	-354.75913
A_O6E	2572.93540	243.43693	-1150.57205	-2301.34320
A_O2O6	0.00000	90.00000	0.00000	0.00000
A_DE	257.12377	332.28495	227.62434	-119.58173
A_DF	3404.97848	19.22594	3215.07384	1121.23978
A_GF	2121.04337	330.00000	1836.87744	-1060.52169
A_O6G	0.00000	90.00000	0.00000	0.00000

Vector	Alpha	Slip Acc
A_O4B	0.00000	0.00000
A_BA	-4.84036	0.00000
A_O4O2	0.00000	0.00000
A_O2A	559.97479	0.00000
A_CA	-4.84036	0.00000
A_CE	-83.80469	0.00000
A_O6E	803.06630	0.00000
A_O2O6	0.00000	0.00000
A_DE	-83.80469	0.00000
A_DF	-531.58924	0.00000
A_GF	0.00000	-2121.04337
A_O6G	0.00000	0.00000

Joint	Mag	Angle	x	y
A	2843.11552	224.90928	-2013.56707	-2007.20042
B	2250.00000	232.92024	-1356.58395	-1795.04317
C	2668.88470	226.83297	-1825.85761	-1946.58406
D	2580.60233	237.71982	-1378.19640	-2181.76146
E	2572.93540	243.43693	-1150.57205	-2301.34320
F	2121.04337	330.00000	1836.87744	-1060.52169
G	0.00000	90.00000	0.00000	0.00000
O2	0.00000	90.00000	0.00000	0.00000
O4	0.00000	90.00000	0.00000	0.00000
O6	0.00000	90.00000	0.00000	0.00000

5.5.1 Acceleration of Link 8 Answer

The magnitude of the acceleration of link 8 is $2121.04337 \frac{units}{s^2}$