test_mechanics

November 19, 2022

0.1 test_mechanics.py

[1]:
```
"""
test_mechanics.py

References:
    Vladimir Pletser - Lagrangian and Hamiltonian Analytical Mechanics Forty␣
↪Exercises Resolved and Explained-Springer Singapore (2018).pdf
"""
import copy
import sys
import os
lstPaths = ["../src"]
for ipath in lstPaths:
    if ipath not in sys.path:
        sys.path.append(ipath)
from libsympy import *
from mechanics import *
from sympy.physics import mechanics
mechanics.mechanics_printing()
#print(sys.version)
#print(sys.path)
# Execute jupyter-notebook related commands.
exec(open('libnotebook.py').read())
```

```
libsympy is loaded.
libnotebook is loaded.
```

0.1.1 Settings

[3]:
```
### Settings
class sets:
    """
    Setttings class.

    Instead of settings class, settings nametuble might be used.
    Settings = namedtuple("Settings", "type dropinf delta")
    sets = Settings(type="symbolic", dropinf=True, delta=0.1)
```

```
    """
    global dictflow, test_all

    def __init__(self):
        pass

    input_dir  = "input/mechanics"
    output_dir = "output/mechanics"

    # Plotting settings
    plot_time_scale = {1:"xy", 2:"xz", 3:"yz"}[3]

    # Execution settings.
    test_all = {0:False, 1:True}[1]
    dictflow = {100:"get_formulary", 150:"get_subformulary",
                200:"simple_harmonic_oscillator_scalar", 201:
↪"simple_harmonic_oscillator_vectorial",
                2321:"coordinate_systems"}
    flow = [dictflow[i] for i in [2321]]
    if test_all: flow = [dictflow[i] for i in dictflow.keys()]
```

[3]:
```
print("Test of the {0}.".format(sets.flow))
```

```
Test of the ['get_formulary', 'get_subformulary',
'simple_harmonic_oscillator_scalar', 'simple_harmonic_oscillator_vectorial',
'coordinate_systems'].
```

0.1.2 get_formulary

[ ]:
```
### get_formulary
if "get_formulary" in sets.flow:
    omech.class_type = "scalar"
    omech.__init__()
    omech.output_style = "latex"
    omech.get_formulary()
    omech.get_formulary(style="eq")

    omech.class_type = "vectorial"
    omech.__init__()
    omech.get_formulary()

    omech.class_type = "EulerLagrange"
    omech.__init__()
    omech.get_formulary()
```

0.1.3 get_subformulary

[ ]:
```
if "get_subformulary" in sets.flow:
    omech.__init__()
    omech.get_subformulary()
```

0.1.4 simple_harmonic_oscillator_scalar

[4]:
```
if "simple_harmonic_oscillator_scalar" in sets.flow:
    """
    Example: Solve a from F = ma
    """
#    omech = mechanics() # DO NOT create any instance.
    omech.class_type = "scalar"
    omech.__init__()
    omech.solver.verbose = True
    commands = ["solve", "NewtonsLaw2", omech.a.rhs]
    omech.process(commands)


    """
    Example: Solve position of a spring mass system.
    F = ma, F = -kx
    -kx = ma
    -kx = m d^2 x/dt^2
    w = sqrt(k/m)
    x(t) = C1*sin(wt) + C2*sin(wt)
    """
    # Scalar Way.
    omech.class_type = "scalar"
    omech.__init__()
    omech.solver.verbose = True
    display("Newton's 2nd Law", omech.NewtonsLaw2,
            "Hooke's Law", omech.HookesLaw)
    commands = ["Eq", "NewtonsLaw2", "HookesLaw"]
#    commands = ["subs", "omech.result", [(a, diff(x, t, 2, evaluate=False))]]
    res = omech.process(commands)
    simp = simplify(res.lhs/m)
    omech.result = Eq(simp, 0)
    commands = ["subs", "omech.result", [(k/m, w**2)]]
    omech.process(commands)
    commands = ["dsolve", "omech.result", x]
    print("Codes:\n", *omech.solver.get_codes())

    omech.x = omech.process(commands).rhs
    v = omech.v.evalf(subs={x:omech.x}).doit()
    a = omech.a.evalf(subs={x:omech.x}).doit()
```

```
    display(omech.result,v,a)

    # Numerical calculations
    [C1,C2] = symbols('C1 C2')
    numvals = {C1:1, C2:1, w:2}
#    commands = ["xreplace", "omech.x", numvals]
#    omech.process(commands)
    x = omech.x.evalf(subs=numvals).doit()
    v = v.evalf(subs=numvals).rhs
    a = a.evalf(subs=numvals).rhs
    plot(x, (t,0,4*pi,200), xlabel="$t$", ylabel="$x(t)$")
    plot(v, (t,0,4*pi,200), xlabel="$t$", ylabel="$v(t)$")
    plot(a, (t,0,4*pi,200), xlabel="$t$", ylabel="$a(t)$")
    plot_sympfunc([x.subs({t:var('x')}),], (0, float(4*pi), 200),
                  xlabel="$t$", ylabel="$x(t)$")
```

```
'solve NewtonsLaw2 Derivative(x(t), (t, 2))'


solve(Eq(F, m*Derivative(x(t), (t, 2))), Derivative(x(t), (t, 2)))
```

$$\left[\frac{F}{m}\right]$$

```
"Newton's 2nd Law"
```

$$F = m\frac{d^2}{dt^2}x(t)$$

```
"Hooke's Law"
```

$$F = -kx(t)$$

```
'Eq NewtonsLaw2 HookesLaw'


Equality(k*x(t) + m*Derivative(x(t), (t, 2)), 0)
```

$$kx(t) + m\frac{d^2}{dt^2}x(t) = 0$$

```
'subs omech.result [(k/m, w**2)]'


Eq(k*x(t)/m + Derivative(x(t), (t, 2)), 0)(subs, [(k/m, w**2)])
```

$$w^2x(t) + \frac{d^2}{dt^2}x(t) = 0$$

```
Codes:
 Equality(k*x(t) + m*Derivative(x(t), (t, 2)), 0)
 Eq(k*x(t)/m + Derivative(x(t), (t, 2)), 0)(subs, [(k/m, w**2)])
```

```
'dsolve omech.result x(t)'
```

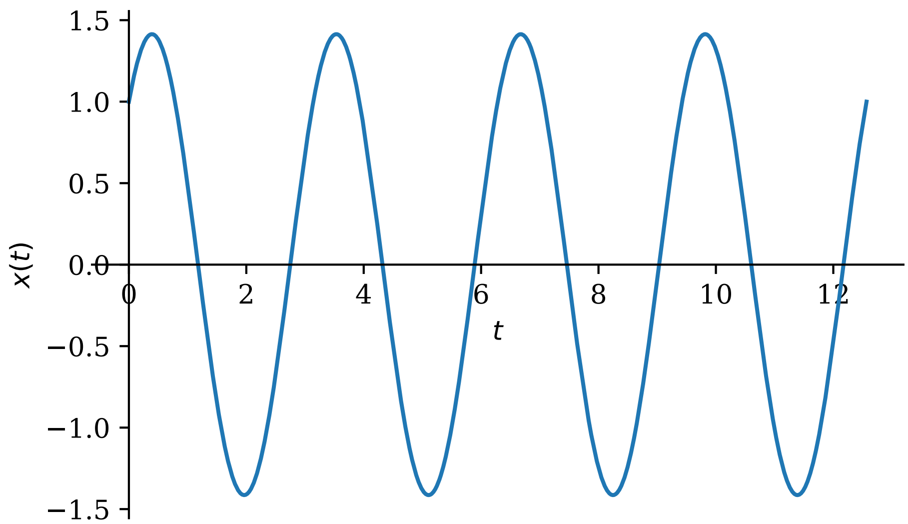
```
dsolve(Eq(w**2*x(t) + Derivative(x(t), (t, 2)), 0), x(t))
```

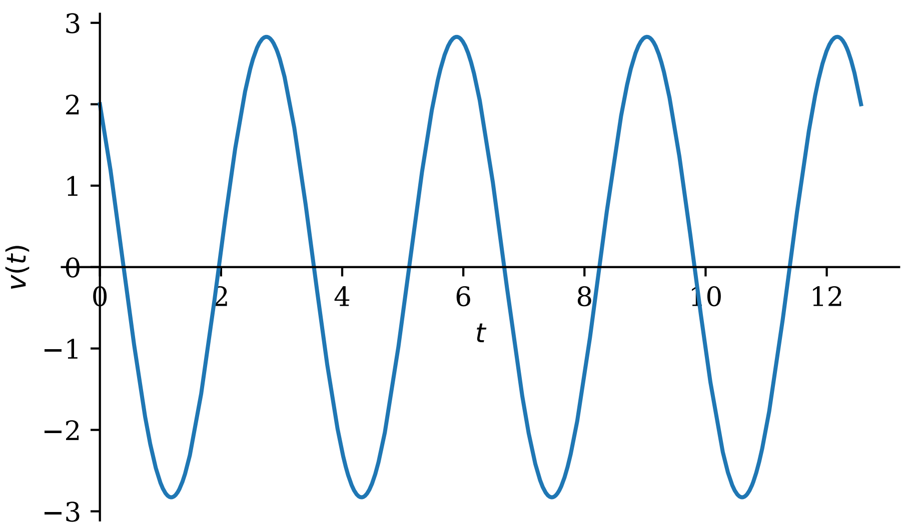
$x(t) = C_1 \sin(tw) + C_2 \cos(tw)$

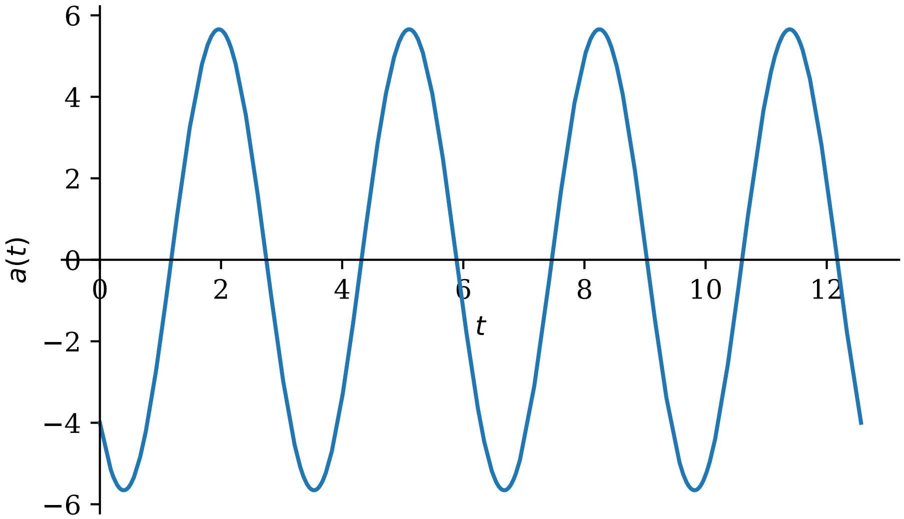
$x(t) = C_1 \sin(tw) + C_2 \cos(tw)$

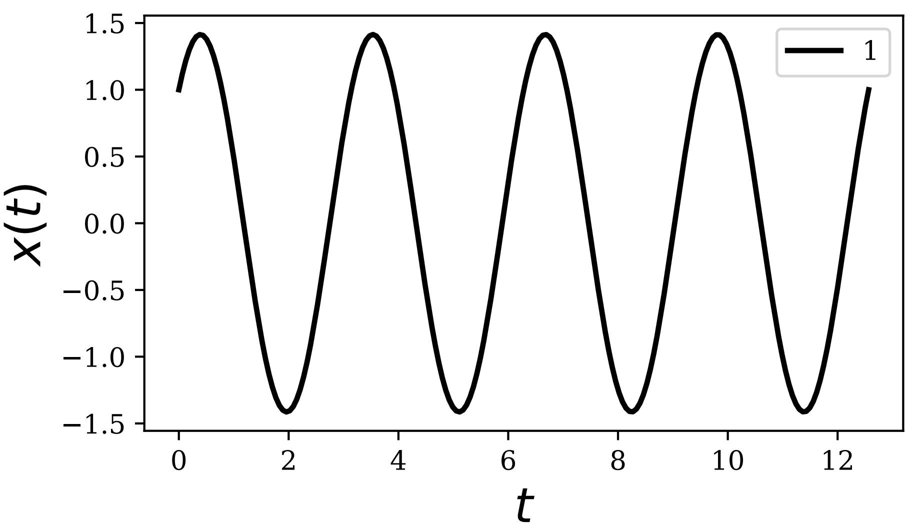
$v(t) = C_1 w \cos(tw) - C_2 w \sin(tw)$

$a(t) = -w^2 \left(C_1 \sin(tw) + C_2 \cos(tw)\right)$









0.1.5 simple_harmonic_oscillator_vectorial

[5]:
```
if "simple_harmonic_oscillator_vectorial" in sets.flow:
#    todo error occurs at "Eq" due to undefined 0 vector. ???
    # Vectorial Way.
    omech.class_type = "vectorial"
    omech.__init__()
    omech.solver.verbose = True
    commands = ["Eq", "NewtonsLaw2", "HookesLaw"]
    omech.process(commands)
```

```python
#    commands = ["subs", "omech.result", [(a, diff(x, t, 2, evaluate=False))]]
    res = omech.process(commands)
    simp = simplify(res.lhs/m)
    omech.result = Eq(simp, 0)
    commands = ["subs", "omech.result", [(k/m, w**2)]]
    omech.process(commands)
    commands = ["dsolve", "omech.result", omech.x]
    print("Codes:\n", *omech.solver.get_codes())

    omech.x = omech.process(commands).rhs
    v = omech.v.evalf(subs={x:omech.x}).doit()
    a = omech.a.evalf(subs={x:omech.x}).doit()
    display(omech.result,v,a)

    # Numerical calculations
    [C1,C2] = symbols('C1 C2')
    numvals = {C1:1, C2:1, w:2}
#    commands = ["xreplace", "omech.x", numvals]
#    omech.process(commands)
    x = omech.x.evalf(subs=numvals).doit()
    v = v.evalf(subs=numvals).rhs.components[C.i]
    a = a.evalf(subs=numvals).rhs.components[C.i]
    plot(x, (t,0,4*pi,200), xlabel="$t$", ylabel="$x(t)$")
    plot(v, (t,0,4*pi,200), xlabel="$t$", ylabel="$v(t)$")
    plot(a, (t,0,4*pi,200), xlabel="$t$", ylabel="$a(t)$")
    plot_sympfunc([x.subs({t:var('x')}),], (0, float(4*pi), 200),
                  xlabel="$t$", ylabel="$x(t)$")
```

```
'Eq NewtonsLaw2 HookesLaw'


Equality(k*x(t) + m*Derivative(x(t), (t, 2)), 0)
```

$$kx(t) + m\frac{d^2}{dt^2}x(t) = 0$$

```
'Eq NewtonsLaw2 HookesLaw'


Equality(k*x(t) + m*Derivative(x(t), (t, 2)), 0)
```

$$kx(t) + m\frac{d^2}{dt^2}x(t) = 0$$

```
'subs omech.result [(k/m, w**2)]'


Eq(k*x(t)/m + Derivative(x(t), (t, 2)), 0)(subs, [(k/m, w**2)])
```

$$w^2x(t) + \frac{d^2}{dt^2}x(t) = 0$$

```
Codes:
 Equality(k*x(t) + m*Derivative(x(t), (t, 2)), 0)
 Equality(k*x(t) + m*Derivative(x(t), (t, 2)), 0)
 Eq(k*x(t)/m + Derivative(x(t), (t, 2)), 0)(subs, [(k/m, w**2)])
```

```
'dsolve omech.result x(t)'
```

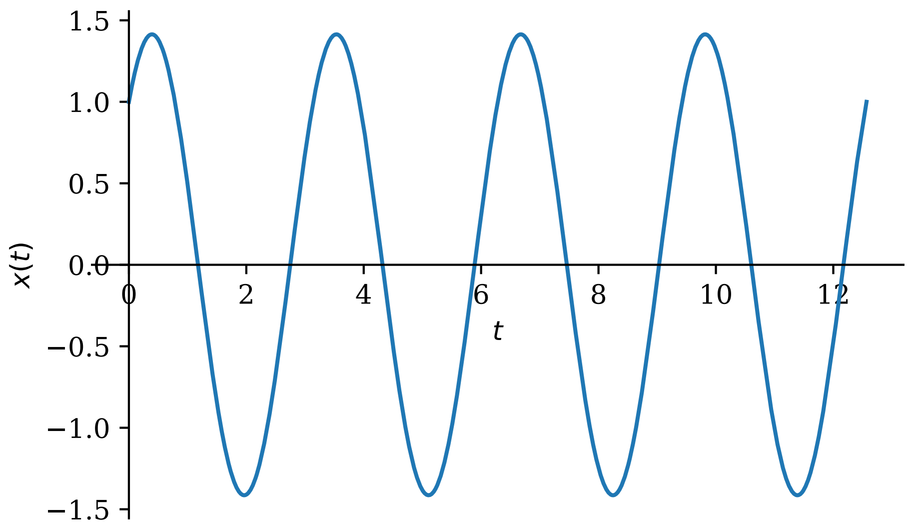
```
dsolve(Eq(w**2*x(t) + Derivative(x(t), (t, 2)), 0), x(t))
```

$x(t) = C_1 \sin(tw) + C_2 \cos(tw)$

$x(t) = C_1 \sin(tw) + C_2 \cos(tw)$

$(v_x)\,\widehat{\mathbf{i}}_{\mathbf{C}} + (v_y)\,\widehat{\mathbf{j}}_{\mathbf{C}} + (v_z)\,\widehat{\mathbf{k}}_{\mathbf{C}} = \left(\frac{d}{dt}x(t)\right)\widehat{\mathbf{i}}_{\mathbf{C}} + \left(\frac{d}{dt}y(t)\right)\widehat{\mathbf{j}}_{\mathbf{C}} + \left(\frac{d}{dt}z(t)\right)\widehat{\mathbf{k}}_{\mathbf{C}}$

$(a_x)\,\widehat{\mathbf{i}}_{\mathbf{C}} + (a_y)\,\widehat{\mathbf{j}}_{\mathbf{C}} + (a_z)\,\widehat{\mathbf{k}}_{\mathbf{C}} = \left(\frac{d^2}{dt^2}x(t)\right)\widehat{\mathbf{i}}_{\mathbf{C}} + \left(\frac{d^2}{dt^2}y(t)\right)\widehat{\mathbf{j}}_{\mathbf{C}} + \left(\frac{d^2}{dt^2}z(t)\right)\widehat{\mathbf{k}}_{\mathbf{C}}$

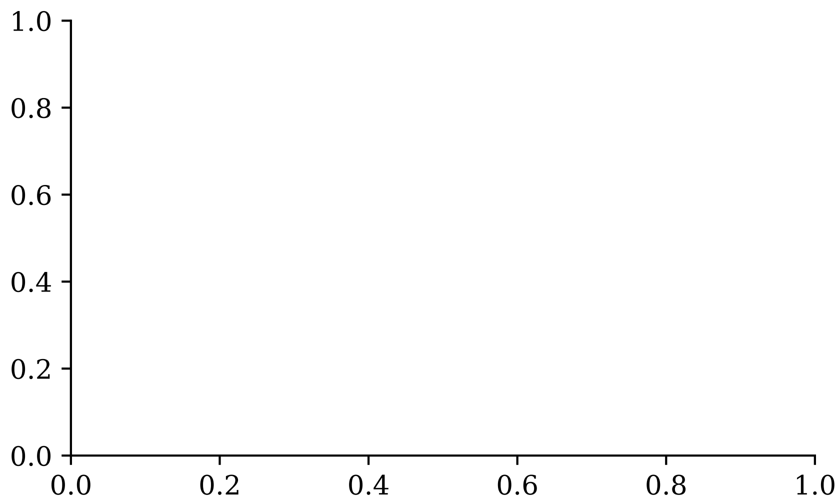



```
␣
↪---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call␣
↪last)

        Input In [5], in <cell line: 1>()
         30 a = a.evalf(subs=numvals).rhs.components[C.i]
         31 plot(x, (t,0,4*pi,200), xlabel="$t$", ylabel="$x(t)$")
    ---> 32 plot(v, (t,0,4*pi,200), xlabel="$t$", ylabel="$v(t)$")
```

```
     33 plot(a, (t,0,4*pi,200), xlabel="$t$", ylabel="$a(t)$")
     34 plot_sympfunc([x.subs({t:var('x')}),], (0, float(4*pi), 200),
     35                xlabel="$t$", ylabel="$x(t)$")

File /usr/local/lib/python3.8/dist-packages/sympy/plotting/plot.py:1873, in plot(show, *args, **kwargs)
   1871 plots = Plot(*series, **kwargs)
   1872 if show:
-> 1873     plots.show()
   1874 return plots

File /usr/local/lib/python3.8/dist-packages/sympy/plotting/plot.py:251, in Plot.show(self)
    249     self._backend.close()
    250 self._backend = self.backend(self)
--> 251 self._backend.show()

File /usr/local/lib/python3.8/dist-packages/sympy/plotting/plot.py:1549, in MatplotlibBackend.show(self)
   1548 def show(self):
-> 1549     self.process_series()
   1550     #TODO after fixing https://github.com/ipython/ipython/issues/1255
   1551     # you can uncomment the next line and remove the pyplot.show() call
   1552     #self.fig.show()
   1553     if _show:

File /usr/local/lib/python3.8/dist-packages/sympy/plotting/plot.py:1546, in MatplotlibBackend.process_series(self)
   1544 if isinstance(self.parent, PlotGrid):
   1545     parent = self.parent.args[i]
-> 1546 self._process_series(series, ax, parent)

File /usr/local/lib/python3.8/dist-packages/sympy/plotting/plot.py:1367, in MatplotlibBackend._process_series(self, series, ax, parent)
   1364 for s in series:
   1365     # Create the collections
   1366     if s.is_2Dline:
-> 1367         x, y = s.get_data()
   1368         if (isinstance(s.line_color, (int, float)) or
   1369                 callable(s.line_color)):
   1370             segments = self.get_segments(x, y)
```

```
    File /usr/local/lib/python3.8/dist-packages/sympy/plotting/plot.py:605,␣
↪in Line2DBaseSeries.get_data(self)
    591 """ Return lists of coordinates for plotting the line.
    592
    593 Returns
   (…)
    602         List of z-coordinates in case of Parametric3DLineSeries
    603 """
    604 np = import_module('numpy')
--> 605 points = self.get_points()
    606 if self.steps is True:
    607     if len(points) == 2:


    File /usr/local/lib/python3.8/dist-packages/sympy/plotting/plot.py:779,␣
↪in LineOver1DRangeSeries.get_points(self)
    776         x_coords.append(q[0])
    777         y_coords.append(q[1])
--> 779 f_start = f(self.start)
    780 f_end = f(self.end)
    781 x_coords.append(self.start)


    File /usr/local/lib/python3.8/dist-packages/sympy/plotting/
↪experimental_lambdify.py:176, in lambdify.__call__(self, args)
    173 def __call__(self, args):
    174     try:
    175         #The result can be sympy.Float. Hence wrap it with complex␣
↪type.
--> 176         result = complex(self.lambda_func(args))
    177         if abs(result.imag) > 1e-7 * abs(result):
    178             return None


    File /usr/local/lib/python3.8/dist-packages/sympy/plotting/
↪experimental_lambdify.py:272, in Lambdifier.__call__(self, *args, **kwargs)
    271 def __call__(self, *args, **kwargs):
--> 272     return self.lambda_func(*args, **kwargs)


    File <string>:1, in <lambda>(x0)


    File /usr/local/lib/python3.8/dist-packages/sympy/core/function.py:1339,␣
↪in Derivative.__new__(cls, expr, *variables, **kwargs)
```

```
   1337     if not v._diff_wrt:
   1338         __ = ''  # filler to make error message neater
-> 1339         raise ValueError(filldedent('''
   1340             Can't calculate derivative wrt %s.%s''' % (v,
   1341             __)))
   1343 # We make a special case for 0th derivative, because there is no
   1344 # good way to unambiguously print this.
   1345 if len(variable_count) == 0:


    ValueError:
Can't calculate derivative wrt 0.0.
```



0.1.6 coordinate_systems

[4]:
```python
if "coordinate_systems" in sets.flow:
    print("Coordinate Systems")

    print("Polar Coordinates")
    omech.class_type = "vectorial"
    omech.__init__()
    omech.solver.verbose = False

    xreplaces = {x:r*cos(theta)*C.i,
                 y:r*sin(theta)*C.j,
                 z:0}
    xreplaces = {x:omech.subformulary.pol_to_cart_x,
                y:omech.subformulary.pol_to_cart_y,
```

```
                z:0} # C.k
    display(omech.r, omech.v, omech.a)
    display(xreplaces)

    commands = ["xreplace", "omech.r", xreplaces]
    r = omech.process(commands).doit()
    commands = ["xreplace", "omech.v", xreplaces]
    v = omech.process(commands).doit()
    commands = ["xreplace", "omech.a", xreplaces]
    a = omech.process(commands).doit()
    display(x,y,z,r,v,a)

    print("Components of r")
    [display(r.rhs.args[i]) for i in range(2)]
    print("Components of v")
    [display(v.rhs.args[i]) for i in range(2)]
    print("Components of a")
    [display(a.rhs.args[i]) for i in range(2)]
```

```
Coordinate Systems
Polar Coordinates
```

$$(r_x)\hat{\mathbf{i}}_{\mathbf{C}} + (r_y)\hat{\mathbf{j}}_{\mathbf{C}} + (r_z)\hat{\mathbf{k}}_{\mathbf{C}} = (x(t))\hat{\mathbf{i}}_{\mathbf{C}} + (y(t))\hat{\mathbf{j}}_{\mathbf{C}} + (z(t))\hat{\mathbf{k}}_{\mathbf{C}}$$

$$(v_x)\hat{\mathbf{i}}_{\mathbf{C}} + (v_y)\hat{\mathbf{j}}_{\mathbf{C}} + (v_z)\hat{\mathbf{k}}_{\mathbf{C}} = \left(\frac{d}{dt}x(t)\right)\hat{\mathbf{i}}_{\mathbf{C}} + \left(\frac{d}{dt}y(t)\right)\hat{\mathbf{j}}_{\mathbf{C}} + \left(\frac{d}{dt}z(t)\right)\hat{\mathbf{k}}_{\mathbf{C}}$$

$$(a_x)\hat{\mathbf{i}}_{\mathbf{C}} + (a_y)\hat{\mathbf{j}}_{\mathbf{C}} + (a_z)\hat{\mathbf{k}}_{\mathbf{C}} = \left(\frac{d^2}{dt^2}x(t)\right)\hat{\mathbf{i}}_{\mathbf{C}} + \left(\frac{d^2}{dt^2}y(t)\right)\hat{\mathbf{j}}_{\mathbf{C}} + \left(\frac{d^2}{dt^2}z(t)\right)\hat{\mathbf{k}}_{\mathbf{C}}$$

$$\{x(t) : r(t)\cos(\theta(t)),\ y(t) : r(t)\sin(\theta(t)),\ z(t) : 0\}$$

$$(r_x)\hat{\mathbf{i}}_{\mathbf{C}} + (r_y)\hat{\mathbf{j}}_{\mathbf{C}} + (r_z)\hat{\mathbf{k}}_{\mathbf{C}} = (r(t)\cos(\theta(t)))\hat{\mathbf{i}}_{\mathbf{C}} + (r(t)\sin(\theta(t)))\hat{\mathbf{j}}_{\mathbf{C}}$$

$$(v_x)\hat{\mathbf{i}}_{\mathbf{C}} + (v_y)\hat{\mathbf{j}}_{\mathbf{C}} + (v_z)\hat{\mathbf{k}}_{\mathbf{C}} = \left(\frac{d}{dt}r(t)\cos(\theta(t))\right)\hat{\mathbf{i}}_{\mathbf{C}} + \left(\frac{d}{dt}r(t)\sin(\theta(t))\right)\hat{\mathbf{j}}_{\mathbf{C}} + \left(\frac{d}{dt}0\right)\hat{\mathbf{k}}_{\mathbf{C}}$$

$$(a_x)\hat{\mathbf{i}}_{\mathbf{C}} + (a_y)\hat{\mathbf{j}}_{\mathbf{C}} + (a_z)\hat{\mathbf{k}}_{\mathbf{C}} = \left(\frac{d^2}{dt^2}r(t)\cos(\theta(t))\right)\hat{\mathbf{i}}_{\mathbf{C}} + \left(\frac{d^2}{dt^2}r(t)\sin(\theta(t))\right)\hat{\mathbf{j}}_{\mathbf{C}} + \left(\frac{d^2}{dt^2}0\right)\hat{\mathbf{k}}_{\mathbf{C}}$$

$$x(t)$$

$$y(t)$$

$$z(t)$$

$$(r_x)\hat{\mathbf{i}}_{\mathbf{C}} + (r_y)\hat{\mathbf{j}}_{\mathbf{C}} + (r_z)\hat{\mathbf{k}}_{\mathbf{C}} = (r(t)\cos(\theta(t)))\hat{\mathbf{i}}_{\mathbf{C}} + (r(t)\sin(\theta(t)))\hat{\mathbf{j}}_{\mathbf{C}}$$

$$(v_x)\hat{\mathbf{i}}_{\mathbf{C}} + (v_y)\hat{\mathbf{j}}_{\mathbf{C}} + (v_z)\hat{\mathbf{k}}_{\mathbf{C}} = \left(-r(t)\sin(\theta(t))\frac{d}{dt}\theta(t) + \cos(\theta(t))\frac{d}{dt}r(t)\right)\hat{\mathbf{i}}_{\mathbf{C}} + \left(r(t)\cos(\theta(t))\frac{d}{dt}\theta(t) + \sin(\theta(t))\frac{d}{dt}r(t)\right)\hat{\mathbf{j}}_{\mathbf{C}}$$

$$(a_x)\,\widehat{\mathbf{i}}_{\mathbf{C}}+(a_y)\,\widehat{\mathbf{j}}_{\mathbf{C}}+(a_z)\,\widehat{\mathbf{k}}_{\mathbf{C}} = \left(-\left(\sin\left(\theta(t)\right)\frac{d^2}{dt^2}\theta(t)+\cos\left(\theta(t)\right)\left(\frac{d}{dt}\theta(t)\right)^2\right)r(t)-2\sin\left(\theta(t)\right)\frac{d}{dt}r(t)\frac{d}{dt}\theta(t)+\cos\left(\theta(t)\right.$$

$$\left(-\left(\sin\left(\theta(t)\right)\left(\frac{d}{dt}\theta(t)\right)^2-\cos\left(\theta(t)\right)\frac{d^2}{dt^2}\theta(t)\right)r(t)+\sin\left(\theta(t)\right)\frac{d^2}{dt^2}r(t)+2\cos\left(\theta(t)\right)\frac{d}{dt}r(t)\frac{d}{dt}\theta(t)\right)\widehat{\mathbf{j}}_{\mathbf{C}}$$

`Components of r`

$$(r(t)\cos\left(\theta(t)\right))\,\widehat{\mathbf{i}}_{\mathbf{C}}$$

$$(r(t)\sin\left(\theta(t)\right))\,\widehat{\mathbf{j}}_{\mathbf{C}}$$

`Components of v`

$$\left(-r(t)\sin\left(\theta(t)\right)\frac{d}{dt}\theta(t)+\cos\left(\theta(t)\right)\frac{d}{dt}r(t)\right)\widehat{\mathbf{i}}_{\mathbf{C}}$$

$$\left(r(t)\cos\left(\theta(t)\right)\frac{d}{dt}\theta(t)+\sin\left(\theta(t)\right)\frac{d}{dt}r(t)\right)\widehat{\mathbf{j}}_{\mathbf{C}}$$

`Components of a`

$$\left(-\left(\sin\left(\theta(t)\right)\frac{d^2}{dt^2}\theta(t)+\cos\left(\theta(t)\right)\left(\frac{d}{dt}\theta(t)\right)^2\right)r(t)-2\sin\left(\theta(t)\right)\frac{d}{dt}r(t)\frac{d}{dt}\theta(t)+\cos\left(\theta(t)\right)\frac{d^2}{dt^2}r(t)\right)\widehat{\mathbf{i}}_{\mathbf{C}}$$

$$\left(-\left(\sin\left(\theta(t)\right)\left(\frac{d}{dt}\theta(t)\right)^2-\cos\left(\theta(t)\right)\frac{d^2}{dt^2}\theta(t)\right)r(t)+\sin\left(\theta(t)\right)\frac{d^2}{dt^2}r(t)+2\cos\left(\theta(t)\right)\frac{d}{dt}r(t)\frac{d}{dt}\theta(t)\right)\widehat{\mathbf{j}}_{\mathbf{C}}$$

[ ]:

```

```