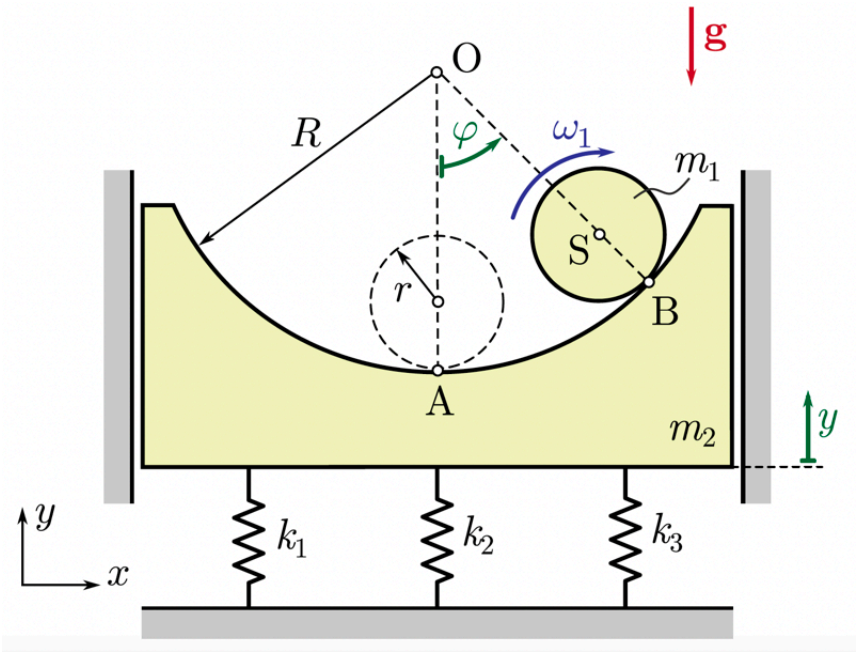
10. Gyakorlat - 2 DoF nemlineáris rendszer

2021.04.16.

Feladat:



A mellékelt ábrán egy két szabadságfokú mechanikai lengőrendszer látható, mely egy m_1 tömegű és r sugarú korongból áll, ami egy R belső sugarú mozgó felületen gördül. Ez az m_2 tömeg a k_1, k_2 és k_3 rugómerevségű rugók által van alátámasztva. Ennek vízszintes irányú elmozdulását az y általánosított koordináta írja le, valamint a gördülő korong pozíciójának leírására a φ általános koordináta használatos. A gravitációs mezőben elhelyezett rendszer egyensúlyi állapota az $y = 0$ és a $\varphi = 0$ állapothoz tartozik. Itt egyedül a rúgókat terheli statikus deformáció.

Adatok:

Az m_1 és m_2 tömegek, az r és R sugarak, valamint a k_1, k_2, k_3 rugómerevségek ismertnek tekinthetőek.

Részfeladatok:

1. Határozza meg a rendszer nemlineáris mozgásegyenleteit a másodfajú Lagrange-egyenlettel, továbbá linearizálja az $y = 0$ és $\varphi = 0$ egyensúlyi helyzet körüli kis kitérések mellett.
2. Határozza meg a linearizált mozgásegyenleteket mátrix együtthatós alakban is.

Megoldás:

1. Feladat

In [1]:
```
import sympy as sp
from IPython.display import Math
sp.init_printing()
# a két általános koordináta felvétele

t = sp.Symbol('t')
y = sp.Function('y')(t)
φ = sp.Function('φ')(t)

m1, m2, r, R, k1, k2, k3, g = sp.symbols('m_1, m_2, r, R, k_1, k_2, k_3, g'
# Tároljuk ezeket a `q` általánosított koord. vektorban
q = sp.Matrix([[y],[φ]]) # a második réteg szögletes zárójel nem kötelező
display(Math('\mathbf{{q}} = {}'.format(sp.latex(q))))

# Figyeljünk rá, hogy mostantól `y` `q[0]` lesz, `φ` pedig `q[1]`.
```

$$\mathbf{q} = \begin{bmatrix} y(t) \\ \varphi(t) \end{bmatrix}$$

A kinetikus energia meghatározása:

$$T = \frac{1}{2} m_2 \dot{y}^2 + \frac{1}{2} m_1 v_{\mathrm{S}}^2 + \frac{1}{2} \Theta_{1\mathrm{Sz}} \omega_{1\mathrm{z}}^2$$

In [2]:
```
""" A kinetikus energiát az általánosított koordinákkal kell felírni,
így `v_S` és `ω_1z`-t azok segítségével kell kifejezni.
"""

# egy kis LaTeX-es formázás: a vektort félkövérrel íratjuk ki (bold font)
# valamint az index álló betű legyen (roman).
v_O = sp.MatrixSymbol('\mathbf{v}_\mathrm{O}',3,1) # Hasonló, mint az `sp.S
                                                   # 3x1-es mátrixként keze
                                                   # szimbolikusan egy megi
v_S = sp.MatrixSymbol('\mathbf{v}_\mathrm{S}',3,1)
r_OSx,r_OSy = sp.symbols('\mathbf{r}_{\mathrm{OS}z},\mathbf{r}_{\mathrm{OS}
r_OS = sp.Matrix([[r_OSx],[r_OSy],[0]])
v_OS_red = v_O + sp.Matrix([[0],[0],[q[1].diff(t)]]).cross(r_OS)
eq_vos = sp.Eq(v_S,v_OS_red)
eq_vos
```

Out[2]:

$$\mathbf{v}_\mathrm{S} = \begin{bmatrix} -\mathbf{r}_{\mathrm{OS}y}\frac{d}{dt}\varphi(t) \\ \mathbf{r}_{\mathrm{OS}z}\frac{d}{dt}\varphi(t) \\ 0 \end{bmatrix} + \mathbf{v}_\mathrm{O}$$

In [3]:
```
""" Sajnos a mátrix szimbólumos út nem járható
a jelenlegi `sympy` verzió mellett (1.7.1).
Remélhetőleg a jövőben jobban ki lesz dolgozva
ez a megközelítés is. Az egyik legfőbb hiányosságot
az alábbi kódsor szemlélteti:
"""
sp.solve(eq_vos,v_O) # Azaz egyszerűen fejezzük ki a `v_O` vektort.

""" Látható, hogy hibát kapunk. A `sympy solve`
metódusa nincs felkészítve a mátrix szimbólumokkal
való számolásra. Ez is jól mutatja, hogy a `sympy`
közel sem tökéletes modul, bár folyamatosan fejlesztik.
Remélhetőleg egy jövőbeli verzióba belekerül ez a funkció is.
"""
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-3-4fcdda7f1235> in <module>
      5 az alábbi kódsor szemlélteti:
      6 """
----> 7 sp.solve(eq_vos,v_O) # Azaz egyszerűen fejezzük ki a `v_O` vektort.
      8
      9 """ Látható, hogy hibát kapunk. A `sympy solve`

~/anaconda3/lib/python3.8/site-packages/sympy/solvers/solvers.py in solve(f
, *symbols, **flags)
    992
    993     # we can solve for non-symbol entities by replacing them with D
ummy symbols
--> 994     f, symbols, swap_sym = recast_to_symbols(f, symbols)
    995
    996     # this is needed in the next two events

~/anaconda3/lib/python3.8/site-packages/sympy/solvers/solvers.py in recast_
to_symbols(eqs, symbols)
    103             isubs = getattr(i, 'subs', None)
    104             if isubs is not None:
--> 105                 new_f.append(isubs(swap_sym))
    106             else:
```

```
    107             new_f.append(i)

~/anaconda3/lib/python3.8/site-packages/sympy/core/basic.py in subs(self, *
args, **kwargs)
    946             rv = self
    947             for old, new in sequence:
--> 948                 rv = rv._subs(old, new, **kwargs)
    949                 if not isinstance(rv, Basic):
    950                     break

~/anaconda3/lib/python3.8/site-packages/sympy/core/cache.py in wrapper(*arg
s, **kwargs)
     70         def wrapper(*args, **kwargs):
     71             try:
---> 72                 retval = cfunc(*args, **kwargs)
     73             except TypeError:
     74                 retval = func(*args, **kwargs)

~/anaconda3/lib/python3.8/site-packages/sympy/core/basic.py in _subs(self,
old, new, **hints)
   1060         rv = self._eval_subs(old, new)
   1061         if rv is None:
-> 1062             rv = fallback(self, old, new)
   1063         return rv
   1064

~/anaconda3/lib/python3.8/site-packages/sympy/core/basic.py in fallback(sel
f, old, new)
   1037                     args[i] = arg
   1038             if hit:
-> 1039                 rv = self.func(*args)
   1040                 hack2 = hints.get('hack2', False)
   1041                 if hack2 and self.is_Mul and not rv.is_Mul:  # 2-ar
g hack

~/anaconda3/lib/python3.8/site-packages/sympy/core/cache.py in wrapper(*arg
s, **kwargs)
     70         def wrapper(*args, **kwargs):
     71             try:
---> 72                 retval = cfunc(*args, **kwargs)
     73             except TypeError:
     74                 retval = func(*args, **kwargs)

~/anaconda3/lib/python3.8/site-packages/sympy/core/operations.py in __new__
(cls, evaluate, _sympify, *args)
     86         is_commutative = not nc_part
     87         obj = cls._from_args(c_part + nc_part, is_commutative)
---> 88         obj = cls._exec_constructor_postprocessors(obj)
     89
     90         if order_symbols is not None:

~/anaconda3/lib/python3.8/site-packages/sympy/core/basic.py in _exec_constr
uctor_postprocessors(cls, obj)
   1806
   1807         for f in postprocessors.get(clsname, []):
-> 1808             obj = f(obj)
   1809
   1810         return obj

~/anaconda3/lib/python3.8/site-packages/sympy/matrices/expressions/matexpr.
py in _postprocessor(expr)
    632
    633         if mat_class == MatAdd:
--> 634             return mat_class(*matrices).doit(deep=False)
    635         return mat_class(cls._from_args(nonmatrices), *matrices).do
```

```
it(deep=False)
    636     return _postprocessor

~/anaconda3/lib/python3.8/site-packages/sympy/matrices/expressions/matadd.p
y in doit(self, **kwargs)
     81         else:
     82             args = self.args
---> 83         return canonicalize(MatAdd(*args))
     84
     85     def _eval_derivative_matrix_lines(self, x):

~/anaconda3/lib/python3.8/site-packages/sympy/strategies/core.py in exhaust
ive_rl(expr)
      9     """ Apply a rule repeatedly until it has no effect """
     10     def exhaustive_rl(expr):
---> 11         new, old = rule(expr), expr
     12         while new != old:
     13             new, old = rule(new), new

~/anaconda3/lib/python3.8/site-packages/sympy/strategies/core.py in conditi
oned_rl(expr)
     31     def conditioned_rl(expr):
     32         if cond(expr):
---> 33             return rule(expr)
     34         else:
     35             return expr

~/anaconda3/lib/python3.8/site-packages/sympy/strategies/core.py in do_one_
rl(expr)
     83     def do_one_rl(expr):
     84         for rl in rules:
---> 85             result = rl(expr)
     86             if result != expr:
     87                 return result

~/anaconda3/lib/python3.8/site-packages/sympy/strategies/rl.py in conglomer
ate(expr)
     67         groups = sift(expr.args, key)
     68         counts = dict((k, sum(map(count, args))) for k, args in
groups.items())
---> 69         newargs = [combine(cnt, mat) for mat, cnt in counts.items()
]
     70         if set(newargs) != set(expr.args):
     71             return new(type(expr), *newargs)

~/anaconda3/lib/python3.8/site-packages/sympy/strategies/rl.py in <listcomp
>(.0)
     67         groups = sift(expr.args, key)
     68         counts = dict((k, sum(map(count, args))) for k, args in
groups.items())
---> 69         newargs = [combine(cnt, mat) for mat, cnt in counts.items()
]
     70         if set(newargs) != set(expr.args):
     71             return new(type(expr), *newargs)

~/anaconda3/lib/python3.8/site-packages/sympy/matrices/expressions/matadd.p
y in combine(cnt, mat)
    104         return mat
    105     else:
--> 106         return cnt * mat
    107
    108

~/anaconda3/lib/python3.8/site-packages/sympy/core/numbers.py in __mul__(se
lf, other)
```

```
   2215             elif isinstance(other, Rational):
   2216                 return Rational(self.p*other.p, other.q, igcd(self.
p, other.q))
-> 2217             return Rational.__mul__(self, other)
   2218         return Rational.__mul__(self, other)
   2219

~/anaconda3/lib/python3.8/site-packages/sympy/core/decorators.py in __sympi
fyit_wrapper(a, b)
     94                 if not hasattr(b, '_op_priority'):
     95                     b = sympify(b, strict=True)
---> 96                 return func(a, b)
     97             except SympifyError:
     98                 return retval

~/anaconda3/lib/python3.8/site-packages/sympy/core/numbers.py in __mul__(se
lf, other)
   1755                 return other*self
   1756             else:
-> 1757                 return Number.__mul__(self, other)
   1758         return Number.__mul__(self, other)
   1759     __rmul__ = __mul__

~/anaconda3/lib/python3.8/site-packages/sympy/core/decorators.py in __sympi
fyit_wrapper(a, b)
     94                 if not hasattr(b, '_op_priority'):
     95                     b = sympify(b, strict=True)
---> 96                 return func(a, b)
     97             except SympifyError:
     98                 return retval

~/anaconda3/lib/python3.8/site-packages/sympy/core/numbers.py in __mul__(se
lf, other)
    767         elif isinstance(other, Tuple):
    768             return NotImplemented
--> 769         return AtomicExpr.__mul__(self, other)
    770
    771     @_sympifyit('other', NotImplemented)

~/anaconda3/lib/python3.8/site-packages/sympy/core/decorators.py in _func(s
elf, other)
    265             if not isinstance(other, expectedcls):
    266                 return retval
--> 267             return func(self, other)
    268
    269         return _func

~/anaconda3/lib/python3.8/site-packages/sympy/core/decorators.py in binary_
op_wrapper(self, other)
    134                     f = getattr(other, method_name, None)
    135                     if f is not None:
--> 136                         return f(self)
    137             return func(self, other)
    138         return binary_op_wrapper

~/anaconda3/lib/python3.8/site-packages/sympy/matrices/expressions/matexpr.
py in __sympifyit_wrapper(a, b)
     26             try:
     27                 b = _sympify(b)
---> 28                 return func(a, b)
     29             except SympifyError:
     30                 return retval

~/anaconda3/lib/python3.8/site-packages/sympy/core/decorators.py in binary_
op_wrapper(self, other)
```

```
    135                     if f is not None:
    136                         return f(self)
--> 137             return func(self, other)
    138         return binary_op_wrapper
    139     return priority_decorator

~/anaconda3/lib/python3.8/site-packages/sympy/matrices/expressions/matexpr.
py in __rmul__(self, other)
    133     @call_highest_priority('__mul__')
    134     def __rmul__(self, other):
--> 135         return MatMul(other, self).doit()
    136
    137     @_sympifyit('other', NotImplemented)

~/anaconda3/lib/python3.8/site-packages/sympy/matrices/expressions/matmul.p
y in doit(self, **kwargs)
    170         deep = kwargs.get('deep', True)
    171         if deep:
--> 172             args = [arg.doit(**kwargs) for arg in self.args]
    173         else:
    174             args = self.args

~/anaconda3/lib/python3.8/site-packages/sympy/matrices/expressions/matmul.p
y in <listcomp>(.0)
    170         deep = kwargs.get('deep', True)
    171         if deep:
--> 172             args = [arg.doit(**kwargs) for arg in self.args]
    173         else:
    174             args = self.args

~/anaconda3/lib/python3.8/site-packages/sympy/matrices/expressions/matadd.p
y in doit(self, **kwargs)
     81         else:
     82             args = self.args
---> 83         return canonicalize(MatAdd(*args))
     84
     85     def _eval_derivative_matrix_lines(self, x):

~/anaconda3/lib/python3.8/site-packages/sympy/strategies/core.py in exhaust
ive_rl(expr)
      9     """ Apply a rule repeatedly until it has no effect """
     10     def exhaustive_rl(expr):
---> 11         new, old = rule(expr), expr
     12         while new != old:
     13             new, old = rule(new), new

~/anaconda3/lib/python3.8/site-packages/sympy/strategies/core.py in conditi
oned_rl(expr)
     31     def conditioned_rl(expr):
     32         if cond(expr):
---> 33             return rule(expr)
     34         else:
     35             return expr

~/anaconda3/lib/python3.8/site-packages/sympy/strategies/core.py in do_one_
rl(expr)
     83     def do_one_rl(expr):
     84         for rl in rules:
---> 85             result = rl(expr)
     86             if result != expr:
     87                 return result

~/anaconda3/lib/python3.8/site-packages/sympy/strategies/rl.py in conglomer
ate(expr)
     65     def conglomerate(expr):
```

```
     66         """ Conglomerate together identical args x + x -> 2x """
---> 67         groups = sift(expr.args, key)
     68         counts = dict((k, sum(map(count, args))) for k, args in
groups.items())
     69         newargs = [combine(cnt, mat) for mat, cnt in counts.items()
]

~/anaconda3/lib/python3.8/site-packages/sympy/utilities/iterables.py in sif
t(seq, keyfunc, binary)
    809         m = defaultdict(list)
    810         for i in seq:
--> 811             m[keyfunc(i)].append(i)
    812         return m
    813     sift = F, T = [], []

~/anaconda3/lib/python3.8/site-packages/sympy/matrices/expressions/matadd.p
y in <lambda>(arg)
     99
    100 factor_of = lambda arg: arg.as_coeff_mmul()[0]
--> 101 matrix_of = lambda arg: unpack(arg.as_coeff_mmul()[1])
    102 def combine(cnt, mat):
    103     if cnt == 1:

AttributeError: 'Dummy' object has no attribute 'as_coeff_mmul'
```

In [4]:

```
# Most tehát minden szimbolikus vektort/mátrixot a benne lévő elemek
# szimbolikussá tételével fogunk kezelni

v_Sx, v_Sy = sp.symbols('v_{\mathrm{S}x}, v_{\mathrm{S}y}')
v_Ox, v_Oy = sp.symbols('v_{\mathrm{O}x}, v_{\mathrm{O}y}')
v_Bx, v_By = sp.symbols('v_{\mathrm{B}x}, v_{\mathrm{B}y}')
r_OSx, r_OSy = sp.symbols('r_{\mathrm{OS}x}, r_{\mathrm{OS}y}')
r_BSx, r_BSy = sp.symbols('r_{\mathrm{BS}x}, r_{\mathrm{BS}y}')
ω_1z = sp.Symbol('ω_1z')

v_S = sp.Matrix([[v_Sx],[v_Sy],[0]])
v_O = sp.Matrix([[v_Ox],[v_Oy],[0]])
v_B = sp.Matrix([[v_Bx],[v_By],[0]])
r_OS = sp.Matrix([[r_OSx],[r_OSy],[0]])
r_BS = sp.Matrix([[r_BSx],[r_BSy],[0]])
```

2 sebességredukciós képlet segítségével számíthatjuk ki a megfelelő sebességeket a kinetikus energia kifejezésében, valamint ezek felhasználásával határozhatjuk meg a kapcsolatot az általános koordináták és ezen sebességek között. Az ábra alapján:

$$\mathbf{v}_\mathrm{S} = \mathbf{v}_\mathrm{O} + \begin{bmatrix} 0 \\ 0 \\ \dot{\varphi} \end{bmatrix} \times \mathbf{r}_\mathrm{OS},$$

valamint

$$\mathbf{v}_\mathrm{S} = \mathbf{v}_\mathrm{B} + \begin{bmatrix} 0 \\ 0 \\ \omega_{1z} \end{bmatrix} \times \mathbf{r}_\mathrm{BS}.$$

Megállapítható továbbá, hogy

$$\mathbf{v}_\mathrm{O} = \mathbf{v}_\mathrm{B} = \begin{bmatrix} 0 \\ \dot{y} \\ 0 \end{bmatrix}.$$

In [5]:
```
v_OS_red = sp.Matrix([[0],[q[0].diff(t)],[0]]) + sp.Matrix([[0],[0],[q[1].d

v_BS_red = sp.Matrix([[0],[q[0].diff(t)],[0]]) + sp.Matrix([[0],[0],[ω_1z]]

# A helyvektorok koordinátái az ábráról leolvashatóak:
vekt_koord = [(r_OSx, (R-r) * sp.sin(q[1])), (r_OSy,  -(R-r) * sp.cos(q[1])
              (r_BSx, -r*sp.sin(q[1])), (r_BSy, r*sp.cos(q[1]))]

# A fentiek alapján az alábbi egyenlőség áll fenn:
sp.Eq(v_OS_red.subs(vekt_koord),v_BS_red.subs(vekt_koord))
```

Out[5]:
$$\begin{bmatrix} -\left(-R+r\right)\cos\left(\varphi(t)\right)\frac{d}{dt}\varphi(t) \\ \left(R-r\right)\sin\left(\varphi(t)\right)\frac{d}{dt}\varphi(t) + \frac{d}{dt}y(t) \\ 0 \end{bmatrix} = \begin{bmatrix} -r\omega_{1z}\cos\left(\varphi(t)\right) \\ -r\omega_{1z}\sin\left(\varphi(t)\right) + \frac{d}{dt}y(t) \\ 0 \end{bmatrix}$$

In [6]:
```
# Fejezzük ki `ω_1z` és `φ.diff(t)` közötti kapcsolatot, felhasználva
# (pl.) az x komponensek egyenlőségét:

tmp = sp.Eq(v_OS_red.subs(vekt_koord)[0],v_BS_red.subs(vekt_koord)[0])
display(tmp)

ω_1z_expr = sp.solve(tmp, ω_1z)[0]

display(Math('ω_{{1z}} = {}'.format(sp.latex(ω_1z_expr))))
```

$$-\left(-R+r\right)\cos\left(\varphi(t)\right)\frac{d}{dt}\varphi(t) = -r\omega_{1z}\cos\left(\varphi(t)\right)$$

$$\omega_{1z} = \frac{\left(-R+r\right)\frac{d}{dt}\varphi(t)}{r}$$

In [7]:
```
# Ezt felhasználva a B és S pontok közötti sebességred. képletből:
v_S_expr = v_BS_red.subs(vekt_koord).subs(ω_1z,ω_1z_expr)
display(Math('\mathbf{{v}}_\mathrm{{S}} = {}'.format(sp.latex(v_S_expr))))

# Mostmár felírható a kinetikus energia az általánosított
# koordináták segítségével

θ_1Sz = sp.Rational(1,2)*m1*r**2
T = (sp.Rational(1,2)*m2 * q[0].diff(t)**2
     + sp.Rational(1,2)*m1 * v_S_expr.dot(v_S_expr)
     + sp.Rational(1,2)*θ_1Sz * ω_1z_expr**2)
```

$$\mathbf{v}_S = \begin{bmatrix} -\left(-R+r\right)\cos\left(\varphi(t)\right)\frac{d}{dt}\varphi(t) \\ -\left(-R+r\right)\sin\left(\varphi(t)\right)\frac{d}{dt}\varphi(t) + \frac{d}{dt}y(t) \\ 0 \end{bmatrix}$$

In [8]:
```
## A potenciális energia

"""A potenciális energia a rugókban felhalmozódó potenciális energia
és a gravitációs erő potenciális energiájából tevődik össze.
Mivel a rugó előterhelt állapotban van az egyensúlyi pozícióban, ezért
célszerű a potenciális energiának bevezetni egy új koordináta rendszert,
aminek a függőleges nullpontja ott van, ahol a rugó hossza megegyegyzik
a terheletlen hosszával. Ezzel a transzformációval
az új koordináta-rendszerben a nullszinttől való eltérést a z koordináta mé

z_st = sp.symbols("z_st")
z = q[0] - z_st

ke = k1 + k2 + k3
U = sp.Rational(1,2)*ke*z**2 + m1*g*(z+r_OSy) + m2*g*z

# Fejezzük ki a statikus deformációt az egyensúlyi egyenletből:

e_egy = sp.Eq((m1 + m2)*g, ke*z_st) # azaz egyensúlyban `(m1 + m2)*g = ke*z
z_st_expr = sp.solve(e_egy,z_st)[0]

# Írjuk vissza U-ba:
U = U.subs(z_st,z_st_expr).subs(vekt_koord)
```

In [9]:
```
# Minden adott a Lagrange-egyenletben; írjuk fel a mozgásegyenleteket

eom_y = T.diff(q[0].diff(t)).diff(t) - T.diff(q[0]) + U.diff(q[0]) # `y`-ho
eom_φ = T.diff(q[1].diff(t)).diff(t) - T.diff(q[1]) + U.diff(q[1]) # `φ`-he
eom_φ = eom_φ.simplify()
eom_y = eom_y.simplify()
```

In [10]:
```
display(sp.Eq(eom_y,0),sp.Eq(eom_φ,0))
```

$$gm_1 + gm_2 - g\left(m_1 + m_2\right) + m_1\left(\left(R-r\right)\sin\left(\varphi(t)\right)\frac{d^2}{dt^2}\varphi(t) + \left(R-r\right)\cos\left(\varphi(t)\right)\left(\right.\right.$$

$$\frac{m_1\left(R-r\right)\left(3R\frac{d^2}{dt^2}\varphi(t) + 2g\sin\left(\varphi(t)\right) - 3r\frac{d^2}{dt^2}\varphi(t) + 2\sin\left(\varphi(t)\right)\frac{d^2}{dt^2}y(t)\right)}{2} = 0$$

In [11]:
```
"""A linearizáláshoz írjuk fel az egyensúlyi helyzet körüli Taylor sorfejté
tartózó Taylor polinomot az elsőfokú tagig. Ehhez a `sympy` `sp.series` met
használható. Demonstráció: linearizáljuk a `sin(a(t))` függvényt `a(t) = 0`

a = sp.Function('a')(t)
a_sym = sp.Symbol('a_sym')
expr = sp.sin(a)

""" Jelen helyzetben `a` `t`-nek a függvénye, így a `sympy` nem szimbólumké
Létre kell hozni egy átmeneti szimbólumot, amire kicseréljük a függvényünke
formaiság. A `sp.series` az argumentumába először egy függvényt vár (itt tö
majd várja a függvény argumentumát. Ezt követően meg kell adni, hogy mely p
sorba (itt a_sym = 0), valamint hogy hanyadrendig szeretnénk megtenni a sor

a_lin_sym = sp.series(expr.subs(a,a_sym),a_sym,0,2)

# Alternatív függvény hívás:
expr.subs(a,a_sym).series(a_sym,0,2) # azaz közvetlenül a `sympy` objektum
```

Out[11]:
$$a_{sym} + O\left(a_{sym}^2\right)$$

In [12]:
```
"""A magasabb rendű tagokat a `.removeO()` metódussal hagyhatjuk el (figyel
majd cseréljük vissza a szimbólumot a függvényünkre:"""

a_lin = a_lin_sym.removeO().subs(a_sym,a)
a_lin
```

Out[12]:
$$a(t)$$

In [13]:
```
"""Linearizáljuk a mozgásegyenletetek bal oldalát (mint függvényeket), a
q(t) = 0, és q.diff(t) = 0 egyensúlyi helyzet körül. Arra kell itt figyelni
deriváltjait külön változóként kell kezelni."""

# Cseréljük ki a deriváltakat szimbólumokra. Fontos, hogy a legmagasabb re
# kezdjünk, mert ha pl. az első deriválttal kezdünk, a második deriváltat
φ, dφ, ddφ = sp.symbols('φ, dφ, ddφ')
φ_csere = [(q[1].diff(t,2),ddφ),(q[1].diff(t),dφ),(q[1],φ)]
φ_csere_vissza = [(elem[1], elem[0]) for elem in φ_csere] # hasznos lesz, l

eom1_csere = eom_y.subs(φ_csere)

eom1_taylor = eom1_csere.series(dφ,0,2).removeO().series(φ,0,2).removeO() #
eom1_taylor = eom1_taylor.subs(φ_csere_vissza).simplify() # visszacsere

display(eom1_taylor)
# Van még egy másodrendben kicsi tagunk, amit kézzel kell nullázni. Ejtsük
# pl. φ helyére 0-t helyettesítünk. (Ha lenne máshol is φ ez nem lenne jó

eom1_lin = sp.Eq(eom1_taylor.subs(q[1],0).simplify().collect(q[0]),0)
eom1_lin # megküzdöttünk vele (:
```

$$gm_1 + gm_2 - g\left(m_1 + m_2\right) + m_1\left(R - r\right)\varphi(t)\frac{d^2}{dt^2}\varphi(t) + m_1\frac{d^2}{dt^2}y(t) + m_2\frac{d^2}{dt^2}y(t) -$$

Out[13]:
$$\left(m_1 + m_2\right)\frac{d^2}{dt^2}y(t) + \left(k_1 + k_2 + k_3\right)y(t) = 0$$

In [14]:
```
# Végezzünk hasonlót a másik mozgásegyenletünk esetében:

eom2_taylor = eom_φ.subs(φ_csere).series(φ,0,2).removeO().subs(φ_csere_viss
display(eom2_taylor)

# Másodrendben kicsi tagunk itt akkor lesz, ha `φ`-t szorozzunk y''-tal.
# Kézzel ezeket a tagokat jelen esetben pl. úgy tudjuk eltüntetni, hogy
# y'' helyére 0-t helyettesítünk:
eom2_lin = sp.Eq(eom2_taylor.subs(q[0].diff(t,2),0).collect(q[1]),0)
eom2_lin
```

$$\frac{3R^2m_1\frac{d^2}{dt^2}\varphi(t)}{2} - 3Rm_1r\frac{d^2}{dt^2}\varphi(t) + \frac{3m_1r^2\frac{d^2}{dt^2}\varphi(t)}{2} + \left(Rgm_1 + Rm_1\frac{d^2}{dt^2}y(t) - gm$$

Out[14]:

$$\left(Rgm_1 - gm_1r\right)\varphi(t) + \left(\frac{3R^2m_1}{2} - 3Rm_1r + \frac{3m_1r^2}{2}\right)\frac{d^2}{dt^2}\varphi(t) = 0$$

2. Feladat

Adjuk meg a mozgásegyenleteket mátrix együtthatós alakban, melynek alakja jelen esetben (disszipatív potenciál, és nem potenciálos aktív erők nélkül):

$$\mathbf{M\ddot{q}} + \mathbf{Kq} = \mathbf{0}.$$

In [15]:
```
# Tegyük meg ezt kétféleképpen:

# Elsőnek parciálisan deriváljuk a kinetikus energiát a 2 ált. sebesség sze
# majd helyettesítsünk 0-t a helyükre.
# Szintax: egymásba ágyazott 2 db `list comprehension`. Olyan mint 2 egymás
# ágyazott `for` ciklus.
M_array = [[T.expand().diff(q1.diff()).diff(q2.diff()).simplify().subs([(q1

K_array = [[U.expand().diff(q1).diff(q2).simplify().subs([(q1,0),(q2,0)]) f

M = sp.Matrix(M_array)
K = sp.Matrix(K_array)

display(Math('\mathbf{{M}} = {}'.format(sp.latex(M))))
display(Math('\mathbf{{K}} = {}'.format(sp.latex(K))))

# Ránézve erre a cellára, gondoljuk meg, mennyire egyszerű felírni a mátri
# mozgásegyenletet (2 érdemi sor), ha ismert a kinetikus és a potenciális e
# (+ a disszipatív potenciál, ha releváns), valamint ha nincs nem potenciá
```

$$\mathbf{M} = \begin{bmatrix} m_1 + m_2 & 0 \\ 0 & \frac{3m_1(-R+r)^2}{2} \end{bmatrix}$$

$$\mathbf{K} = \begin{bmatrix} k_1 + k_2 + k_3 & 0 \\ 0 & gm_1\left(R - r\right) \end{bmatrix}$$

In [16]:
```
# Másik módszer, a már linearizált mozgásegyenletekből (amit a Lagrange-egy

# fűzzük őket össze egy vektorba:
eom_12 = sp.Matrix([[eom1_lin.lhs],[eom2_lin.lhs]]) # kell a `left hand sid
M2 = eom_12.jacobian(q.diff(t,2)) # Jacobi-mátrix az általános gyorsulások
K2 = eom_12.jacobian(q)

display(Math('\mathbf{{M}} = {}'.format(sp.latex(M2))))
display(Math('\mathbf{{K}} = {}'.format(sp.latex(K2))))
```

$$\mathbf{M} = \begin{bmatrix} m_1 + m_2 & 0 \\ 0 & \frac{3R^2 m_1}{2} - 3Rm_1 r + \frac{3m_1 r^2}{2} \end{bmatrix}$$

$$\mathbf{K} = \begin{bmatrix} k_1 + k_2 + k_3 & 0 \\ 0 & Rgm_1 - gm_1 r \end{bmatrix}$$

In [17]:
```
display(sp.simplify(M-M2))
display(sp.simplify(K-K2))  # Ha nem bíznánk magunkban (:
```

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Készítette:

Csuzdi Domonkos (Alkalmazott Mechanika Szakosztály)
Balogh Tamás (BME MM) kidolgozása alapján.

Hibák, javaslatok:
amsz.bme@gmail.com
csuzdi02@gmail.com

2021.04.04