2 DOF System, Alternative Procedure

G. Boffi

1 Alternative Solution to 2DOF Problem

If you get lost in the solution proposed for the first exercise, here it is an alternative solution based on a direct kinematic analysis and a somewhat labourious summing up of all the inertial contributions to the equation of equilibrium written using the PVD.

We don't want to use symbolic programming so we are going to attribute a unit value to the physical constants, the displacements and the accelerations so that we can use their names in the various expressions below w/o modifying the expressions' values.

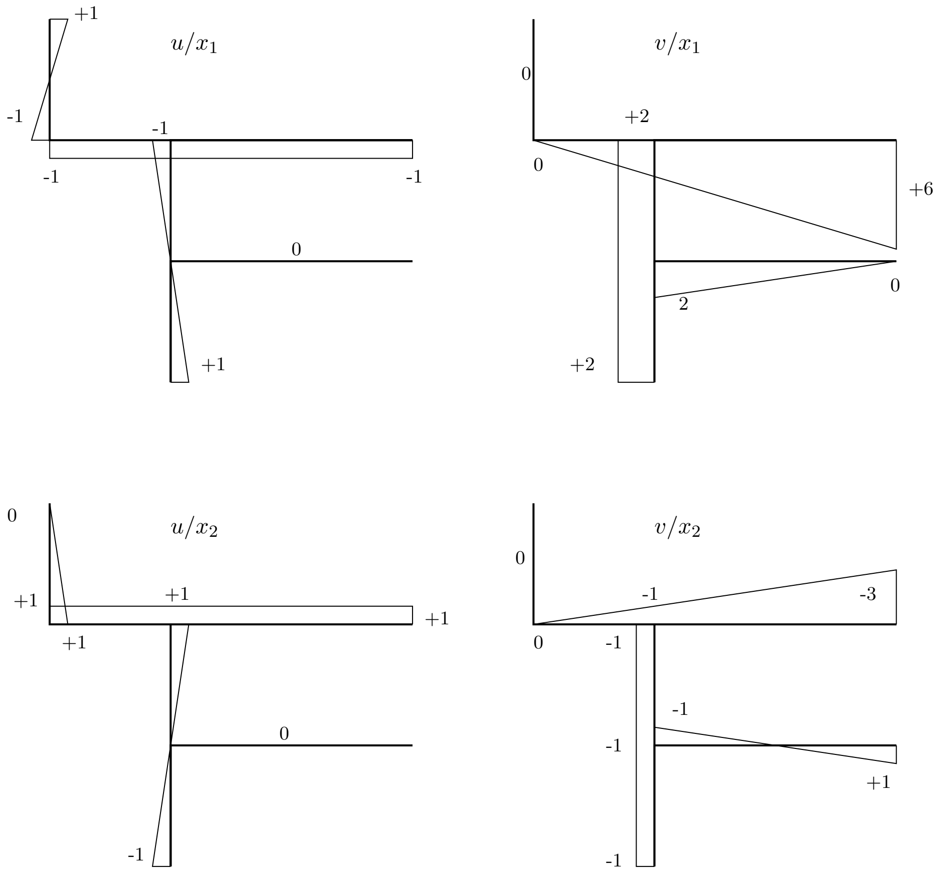
```
m, k = 1.0, 1.0
x1, x2 = 1, 1
a1, a2 = 1, 1
```

To represent a virtual displacement, we'll use a (dummy) function to compute the increment of a quantity, that it's here defined as follows:

```
def delta(x): return x
```

1.1 Kinematic Analysis Results

First row, the (small) displacements when $x_2 = 0$, normalized w/r to x_1; second row, ditto for $x_1 = 0$, normalized w/r to x_2.



1.2 Elastic Forces

The displacements in $\mathscr{A}$ and $\mathscr{B}$ due to x_1 and x_2 and the resulting spring forces on the structure

```
uA1, uA2 = 1*x1, 0*x2
fxA1, fxA2 = -k*uA1, -k*uA2
```

```
vB1, vB2 = 0*x2, 1*x2
fyB1, fyB2 = -k*vB1, -k*vB2
```

1.3 Inertial Forces

The displacements (we'll need them for the PVD), the accelerations and the inertial forces in $\mathscr{D}$ due to $\ddot{x}_1$ and $\ddot{x}_2$

```
uD1, uD2 = -1*x1, +1*x2
auD1, auD2 = -1*a1, +1*a2
fxD1, fxD2 = -m*auD1, -m*auD2

vD1, vD2 =   +6*x1, -3*x2
avD1, avD2 = +6*a1, -3*a2
fyD1, fyD2 = -m*avD1, -m*avD2
```

The displacements, the accelerations and the inertial forces in $\mathscr{E}$ due to $\ddot{x}_1$ and $\ddot{x}_2$

```
uE1, uE2 = 1*x1, -1*x2
auE1, auE2 = 1*a1, -1*a2
fxE1, fxE2 = -m*auE1, -m*auE2

vE1, vE2 = 2*x1, -1*x2
avE1, avE2 = 2*a1, -1*a2
fyE1, fyE2 = -m*avE1, -m*avE2
```

1.4 Equation of motion and Structural Matrices

The structural matrices can be computed , coefficient by coefficient, by the application of the PVD

```
k11 = - fxA1*delta(uA1)
k12 = - fxA2*delta(uA1)
k21 = - fyB1*delta(vB1)
k22 = - fyB2*delta(vB2)

m11 = - fxD1*delta(uD1) - fyD1*delta(vD1) - fxE1*delta(uE1) - fyE1*delta(vE1)
m12 = - fxD2*delta(uD1) - fyD2*delta(vD1) - fxE2*delta(uE1) - fyE2*delta(vE1)
m21 = - fxD1*delta(uD2) - fyD1*delta(vD2) - fxE1*delta(uE2) - fyE1*delta(vE2)
m22 = - fxD2*delta(uD2) - fyD2*delta(vD2) - fxE2*delta(uE2) - fyE2*delta(vE2)

K = array(((k11, k12), (k21, k22)))
M = array(((m11, m12), (m21, m22)))

print(K)
print(M)
```

```
[[ 1.  0.]
 [ 0.  1.]]
[[ 42. -22.]
 [-22.  12.]]
```

As you can see, the structural matrices are equal to the ones computed in another fashion. The remaining part of the solution is the same.