week14–springs

April 22, 2024

1 George McNinch Math 87 - Spring 2024

2 Week 13

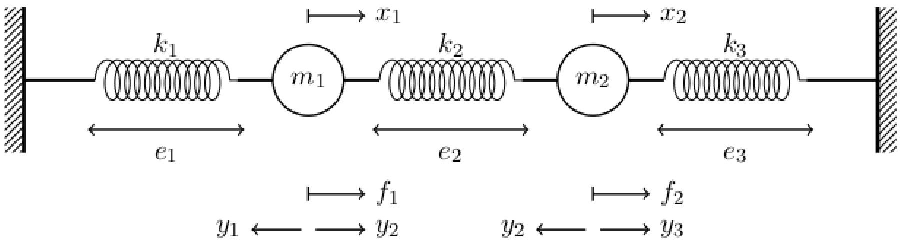
3 Spring networks

4 Modeling spring networks

Examples of "Structural modeling" that will ultimately involve *least squares* approximation.

4.1 One-dimensional models

Let's consider a linear network of 3 springs and 2 masses:



Here are the variables:

- f_j = applied load or force to mass (in `N` = `Newtons`), for $j = 1, 2$
- k_i = spring constant (in `N/m` = `Newtons per meter`), for $i = 1, 2, 3$
- e_i = elongation of spring i from equilibrium (in `m` = `meters`)
- x_j = displacement of mass j from equilibrium (in `m` = `meters`)
- y_i = restoring force on spring i (in `N` = `Newtons`)

The *"inputs"* are the applied forces f_j which cause the masses to move, resulting in elongation of springs.

We'll take "movement to the right" to be *positive*, and a stretch as *positive* elongation.

Thus we have the equations:

$$e_1 = x_1, \quad e_2 = x_2 - x_1, \quad e_3 = -x_2.$$

(This third equation reflects the fact that spring 3 compresses when m_2 moves to the right.)

Let's put this in matrix form:

$$\mathbf{e} = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -1 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = B\mathbf{x}.$$

Now, let's recall that according to Hooke's Law, the elongation of the spring causes a restoring force on the mass, determined by the *spring constant* $k_i > 0$. Thus we get equations

$$y_j = k_j e_j \quad \text{for } j = 1, 2, 3.$$

In matrix form, these equations read:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} k_1 & 0 & 0 \\ 0 & k_2 & 0 \\ 0 & 0 & k_3 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = K\mathbf{e}.$$

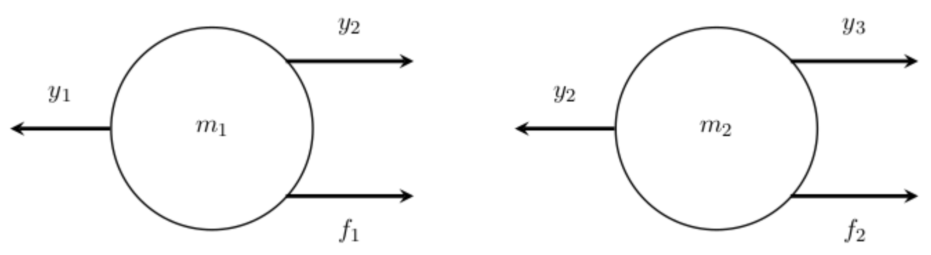
Combining these euqations gives

$$\mathbf{y} = K\mathbf{e} = KB\mathbf{x}.$$

[15]:
```
import numpy as np
import numpy.linalg as la

import sympy as sp

def sbv(i,n):
    return np.array([1 if j == i else 0 for j in range(n)])

def makeK(kar):
    n = len(kar)
    return np.array([kar[i]*sbv(i,n) for i in range(n)])

def makeB(m,n):
    return np.array([(-1)*sbv(i-1,m) + sbv(i,m) for i in range(n)])

k1, k2, k3 = sp.symbols('k1 k2 k3')

K = makeK([k1,k2,k3])
B = makeB(2,3)
(K,B)
```

[15]:
```
(array([[k1, 0, 0],
        [0, k2, 0],
        [0, 0, k3]], dtype=object),
 array([[ 1,  0],
        [-1,  1],
        [ 0, -1]]))
```

Next, we assume that the system is at rest after the loads are applied (i.e. the forces f_i).



Looking at the diagram, we see that the following equations must hold:

(The first diagram gives:)

$$y_1 = y_2 + f_1 \implies$$
$$y_1 - y_2 = f_1$$

(The second diagram gives:)

$$y_2 = y_3 + f_2 \implies$$
$$y_2 - y_3 = f_2$$

In matrix form this reads

$$\begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix},$$

i.e.

$$B^T \mathbf{y} = \mathbf{f}$$

Combined with our earlier equation

$$\mathbf{y} = K\mathbf{e} = KB\mathbf{x}$$

we now see

$$B^T K B \mathbf{x} = \mathbf{f}.$$

```
[2]: A=B.transpose() @ K @ B
     A
```

```
[2]: array([[k1 + k2, -k2],
            [-k2, k2 + k3]], dtype=object)
```

Thus we have

$$A = B^T K B = \begin{bmatrix} k_1 + k_2 & -k_2 \\ -k_2 & k_2 + k_3 \end{bmatrix}.$$

```
[16]: def findDisplacements(kar,far):
          # kar = array of spring constants
          # far = array of initial forces.
          m = len(far)
          n = len(kar)
          B = makeB(m,n)
          K = makeK(kar)
          A = B.transpose() @ K @ B
          f = np.array(far)
          return la.solve(A,f)

      # Let's find the displacements for spring constants `k = [1,1,1]`
      # and forces `f = [3,-3]`

      findDisplacements([1,1,1],[3,-3])
```

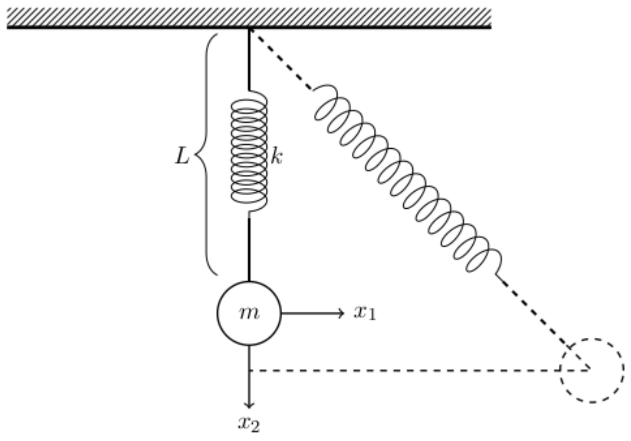
```
[16]: array([ 1., -1.])
```

```
[4]: findDisplacements([1,1,1],[3,-2])
```

```
[4]: array([ 1.33333333, -0.33333333])
```

4.2 Two dimensional models

Now let's allow the mass to move in two dimensions:



L - resting length of spring
k - spring constant
x_1 - displacement to the right
x_2 - displacement downward
e - elongation.

We see in this case that the elongation e satisfies

$$e = \sqrt{x_1^2 + (L + x_2)^2} - L$$

Since this does not express a linear relationship between e and the displacements x_1, x_2, we can't express this relationship using a matrix.

But we can *linearize*. Recall that the linearization (first-order Taylor polynomial) about $t = 0$ of the function $y = \sqrt{1+t}$ is given by

$$(\clubsuit) \quad \sqrt{1+t} \approx 1 + \frac{t}{2} + O(t^2).$$

Let's use this linearization to rewrite the expression for e given above.

We first rewrite

$$\begin{aligned} x_1^2 + (L+x_2)^2 &= x_1^2 + L^2 + 2Lx_2 + x_2^2 \\ &= L^2\left(\frac{x_1^2}{L^2} + 1 + \frac{2x_2}{L} + \frac{x_2^2}{L^2}\right) \\ &= L^2\left(1 + \frac{2x_2}{L} + \frac{x_1^2}{L^2} + \frac{x_2^2}{L^2}\right) \end{aligned}$$

so that

$$\begin{aligned} e &= \sqrt{x_1^2 + (L+x_2)^2} - L \\ &= \sqrt{L^2\left(1 + \frac{2x_2}{L} + \frac{x_1^2}{L^2} + \frac{x_2^2}{L^2}\right)} - L \\ &= L\sqrt{1 + \frac{2x_2}{L} + \frac{x_1^2}{L^2} + \frac{x_2^2}{L^2}} - L \end{aligned}$$

Now taking $t = \dfrac{2x_2}{L} + \dfrac{x_1^2}{L^2} + \dfrac{x_2^2}{L^2}$ the approximation $(\clubsuit)$ gives

$$\begin{aligned} e &\approx L\left(1 + \frac{1}{2}t\right) - L \\ &= L\left(1 + \frac{1}{2}\frac{2x_2}{L} + \frac{x_1^2}{L^2} + \frac{x_2^2}{L^2}\right) - L \\ &= x_2 + \frac{x_1^2 + x_2^2}{2L} \end{aligned}$$

Now, this is of course still not a linear relationship between e and x_1, x_2. Note that the approximation $(\clubsuit)$ depends on the assumption that $t = \dfrac{2x_2}{L} + \dfrac{x_1^2}{L^2} + \dfrac{x_2^2}{L^2} \approx 0$.

If we suppose that the displacements x_1, x_2 are small compared to the resting length L of the spring, then $x_1^2 + x_2^2$ is even smaller compared to L, so making one more approximation, we eliminate the quadratic term and so we get

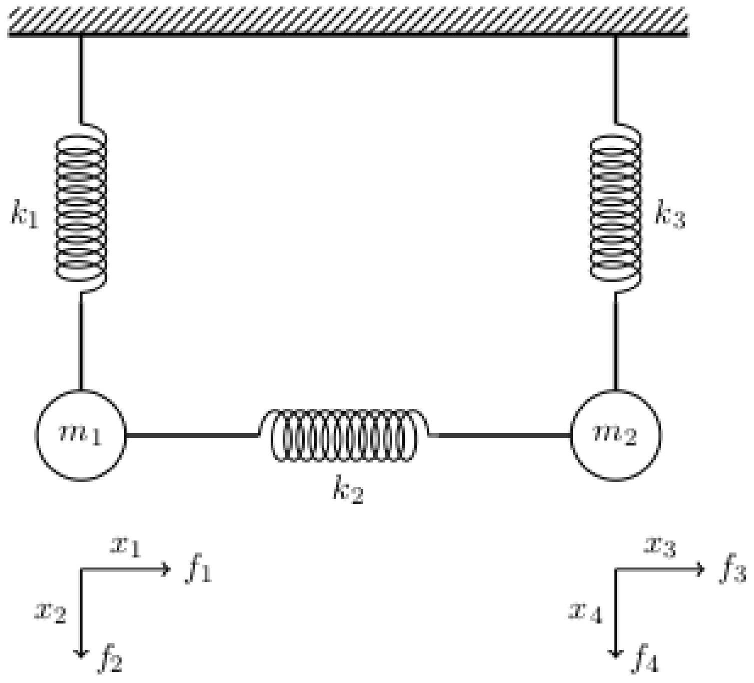
$$e \approx x_2.$$

Remark The approximation $e \approx x_2$ is equivalent to a small-angle approximation. It essentially says that the horizontal displacements are negligible in the elongation, but vertical displacements are important.

Remark This assumption is of course **"wrong"**, but can still be useful. Especially, it is now *linear*

4.3 Spring networks

Let's consider a network of springs in the two-dimensional setting.



$$
\begin{aligned}
e_1 &= x_2 \\
e_2 &= x_3 - x_1 \\
e_3 &= x_4
\end{aligned}
$$

The *linearization* discussed in the previous section says that the elongation is determined by the displacement in the "1 dimensional direction of the spring".

We get the matrix equation

$$
\mathbf{e} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = B\mathbf{x}.
$$

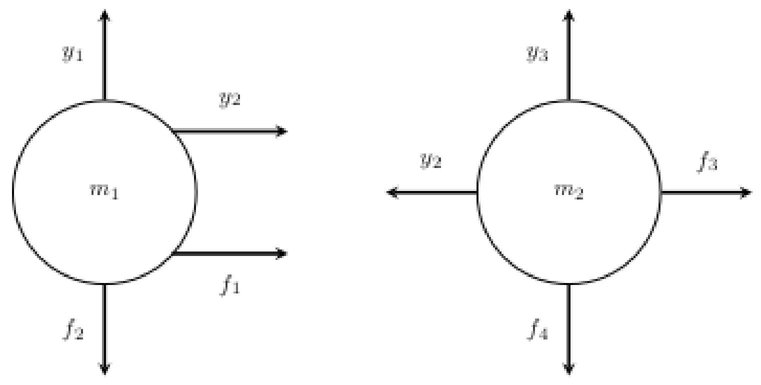
i.e.

$$
B = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

As before, Hooke's Law gives

$$
\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} k_1 & 0 & 0 \\ 0 & k_2 & 0 \\ 0 & 0 & k_3 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = K\mathbf{e}.
$$

We see again that the equations to balance the forces come from the transposed matrix B^T:



$$\begin{aligned} y_1 &= f_2 \\ -y_2 &= f_1 \\ y_2 &= f_3 \\ y_3 &= f_4 \end{aligned}$$

This gives

$$\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{y} = B^T \mathbf{y} = \mathbf{f}.$$

Thus

$$B^T K B \mathbf{x} = \mathbf{f}.$$

[73]:
```
B = np.array([[0,1,0,0],[-1,0,1,0],[0,0,0,1]])

K=makeK([1,1,1])            ## springs of equal strength
f = np.array([0,1,0,1])     ## downward forces only

A = B.transpose() @ K @ B

try:
   la.solve(A , f)
except:
   print(f"the matrix B^T K B = \n\n{A}\n\n is singular")
```

```
the matrix B^T K B =

[[ 1  0 -1  0]
 [ 0  1  0  0]
 [-1  0  1  0]
 [ 0  0  0  1]]

 is singular
```

[69]:
```
# numpy confirms that the null space has dimension 1

la.matrix_rank(A)
```

```
[69]: 3
```

```
[74]: # so we just need to find a non-zero vector in the null space

      # now, the Null space of A can be obtained from the *singular value␣
       ↪decomposition* of A.

      # la.svd(A) returns a tuple ( U , S, Vh )
      # where e.g. U is a unitary matrix

      U,S,Vh = la.svd(A)

      # notice that the last column of U is in the null space:

      A@U
```

```
[74]: array([[-1.41421356,  0.        ,  0.        ,  0.        ],
             [ 0.        ,  1.        ,  0.        ,  0.        ],
             [ 1.41421356,  0.        ,  0.        ,  0.        ],
             [ 0.        ,  0.        ,  1.        ,  0.        ]])
```

```
[75]: nullA = U[:,3]
      A @ nullA
```

```
[75]: array([0., 0., 0., 0.])
```

```
[76]: # now we just need a solution to A @ x = f
      # let's use `lstsq` to get one

      x,_,_,_ = la.lstsq(A,f,rcond=None)
      (x,nullA)
```

```
[76]: (array([0., 1., 0., 1.]),
       array([0.70710678, 0.        , 0.70710678, 0.        ]))
```

Thus we see that the general solution to

$$A\mathbf{x} = \mathbf{f}$$

is given by

$$x = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} + t \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} t \\ 1 \\ t \\ 1 \end{bmatrix}.$$

What is the physical meaning of the solutions given by the null space of A? i.e. the solutions

$$t\mathbf{x}_n = t \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

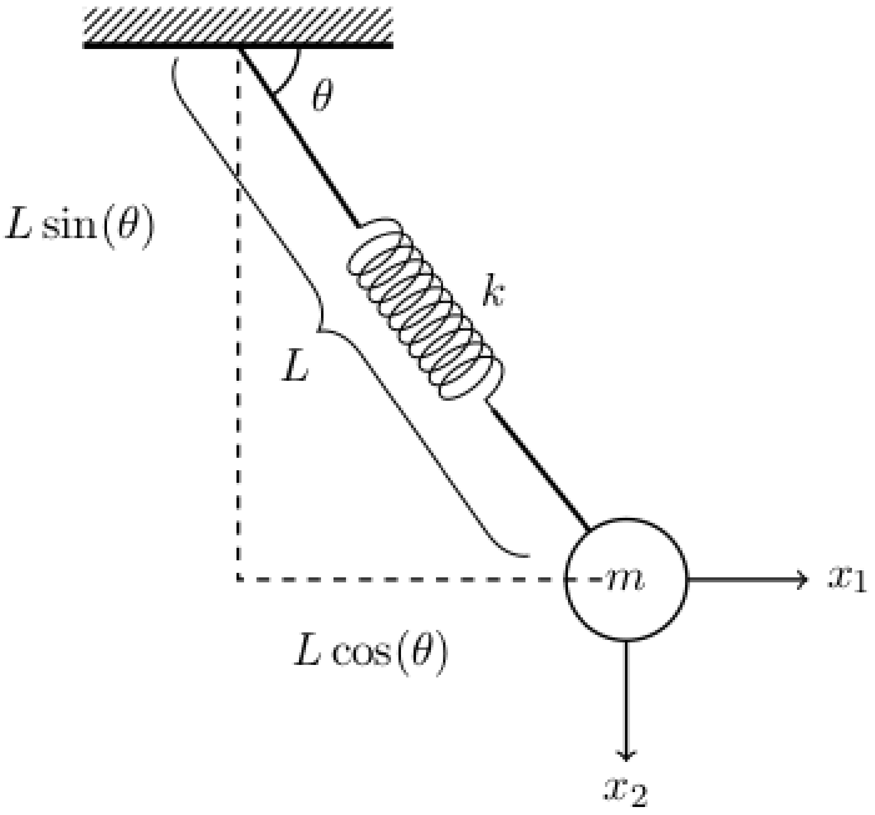
Well, since $A\mathbf{x}_n = \mathbf{0}$, we see that a non-zero displacement can result from application of *zero force* (!).

This is clearly "wrong", and is a consequence of our linearization, **but** these displacement are referred to as *unstable modes*, and they do correspond to some real physical phenomena –

see e.g. Tacoma bridge collapse (1940)

5 Stabilizing the unstable modes

One way to try to stabilize the unstable modes is to add a diagonal spring:



For the indicate angle θ, the elongation is given by

$$e = \sqrt{(L\cos(\theta) + x_1)^2 + (L\sin(\theta) + x_2)^2}$$

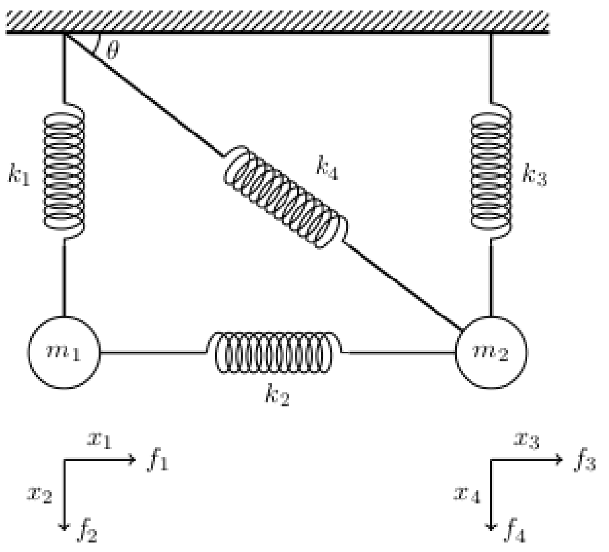
Now our linearization assumption gives

$$e \approx x_1 \cos(\theta) + x_2 \sin(\theta).$$

Observe that under the linearization, the elongation is again in the "spring direction".

Note e.g. if $\theta = 0$ then $e = x_1$ and if $\theta = \dfrac{\pi}{2}$ then $e = x_2$ (as before).

Now return to our unstable network, and add a diagonal spring:



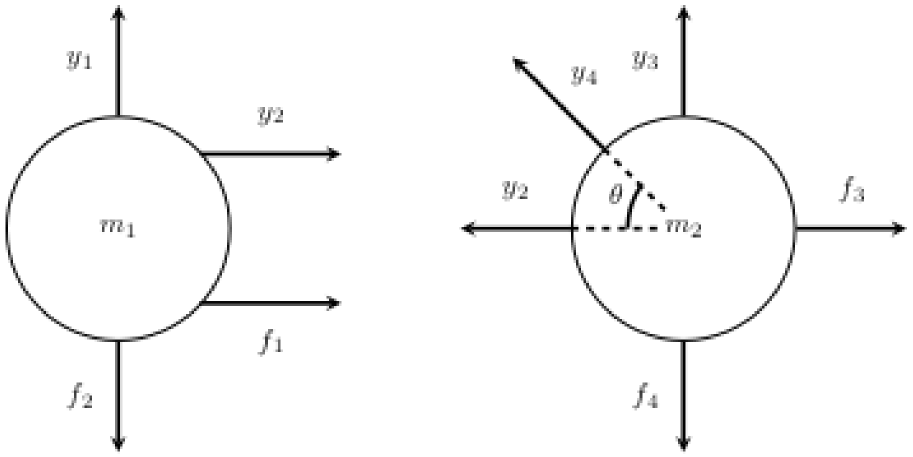
$$
\begin{aligned}
e_1 &= x_2 \\
e_2 &= x_3 - x_1 \\
e_3 &= x_4 \\
e_4 &= x_3 \cos(\theta) + x_4 \cos(\theta)
\end{aligned}
$$

The new parameter here is the elongation e_4.

Now we see that $\mathbf{e} = B\mathbf{x}$ where

$$
B = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \cos(\theta) & \sin(\theta) \end{bmatrix}.
$$

Let's inspect the force balancing conditions:



As before, we see that

$$B^T\mathbf{y} = \mathbf{f}$$

so that

$$B^T K B\mathbf{x} = \mathbf{f}.$$

[82]:
```
# let's suppose theta = pi/3
theta = np.pi/3
BB = np.array([[0,1,0,0],[-1,0,1,0],[0,0,0,1],[0,0,np.cos(theta),np.
 ↪sin(theta)]])

K=makeK([1,1,1,1])           ## springs of equal strength
f = np.array([0,1,0,1])    ## downward forces only

AA = BB.transpose() @ K @ BB

## AA is non-singular!
la.matrix_rank(AA)
```

[82]:
```
4
```

So there are no *unstable modes* for this system. Non-zero displacements **x** are only determined by non-zero force vectors **f**.

[ ]: