

# week14-02-springs-bridges

April 14, 2025

## 1 George McNinch Math 87 - Spring 2025

## 2 Week 14

## 3 Spring networks

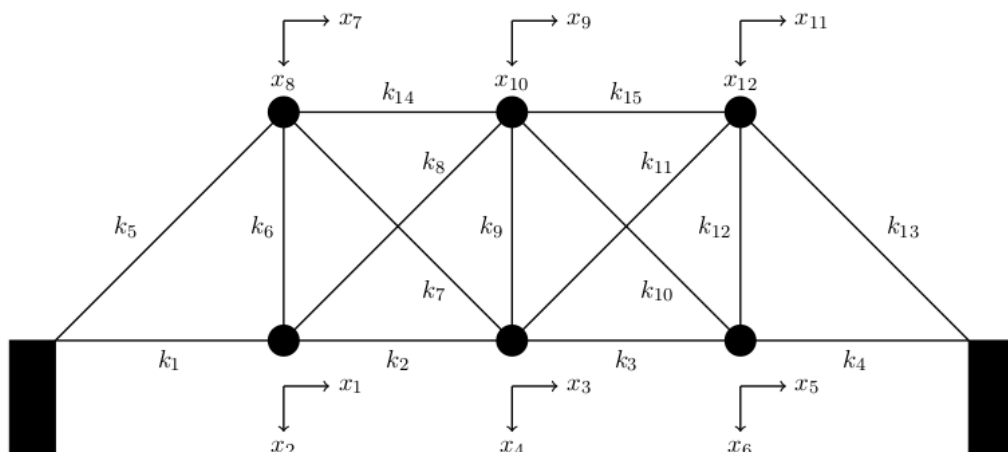
## 4 Least squares and bridges

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

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)])
```

Suppose we are building a bridge, which we'll view as a spring-system as before:



For simplicity we are going to assume that all angles in the diagram are  $\pi/2$  or  $\pi/4$ .

Using the same material for each “beam” of the bridge, the spring constants are inversely proportional to their length.

There are 15 springs/beams in total, and we take them as follows:

$$k_i = \begin{cases} 9.7500 \cdot 10^6 \text{ N/m} & \text{for } i = 1, 2, 3, 4, 6, 9, 12, 14, 15 \\ 6.8943 \cdot 10^6 \text{ N/m} & \text{for } i = 5, 7, 8, 10, 11, 13 \end{cases}$$

In order to test the bridge for structural integrity, we load the three lower masses (where traffic would be) individually with forces of  $2 \cdot 10^6 \text{ N}$ , and measure the resulting displacements.

**Remark** The average car in the US weighs about 4,000 lbs ( $\approx 1,814 \text{ kg}$ ), which corresponds to  $\approx 1.778 \cdot 10^4 \text{ N}$ . The average semi-truck weighs about 80,000 lbs ( $\approx 36,287 \text{ kg}$ ), which corresponds to  $\approx 3.556 \cdot 10^5 \text{ N}$ .

```
[7]: # The matrix B with Bx = e is given as follows:
def sbv(i,n):
    return np.array([1 if i == j else 0 for j in range(n)])

def sbvarray(x,l):
    return sbv(l.index(x),len(l))

a = list(range(1,13))

k = np.sqrt(2)/2

B_bridge = np.array([ sbvarray(1,a),                                # e1 = x1
    ↪x1
                        sbvarray(3,a) - sbvarray(1,a),             # e2 = x3 - x1
                        sbvarray(5,a) - sbvarray(3,a),             # e3 = x5 - x3
                        sbvarray(5,a),                             # e4 = x5
                        k*(sbvarray(7,a) - sbvarray(8,a)),         # e5 = k*x7 - x8
    ↪k*x8
                        sbvarray(2,a) - sbvarray(8,a),             # e6 = x2 - x8
                        k*(sbvarray(3,a) + sbvarray(4,a)
                          - sbvarray(7,a) - sbvarray(8,a)),       # e7 = k*x3 + x4 - k*x7 - k*x8
    ↪k*x4 - k*x7 - k*x8
                        k*(-sbvarray(1,a) + sbvarray(2,a)
                          + sbvarray(9,a) - sbvarray(10,a)),      # e8 = -k*x1 + x2 + k*x9 - k*x10
    ↪k*x2 + k*x9 - k*x10
                        sbvarray(4,a) - sbvarray(10,a),            # e9 = x4 - x10
    ↪x10
                        k*(sbvarray(5,a) + sbvarray(6,a)
                          - sbvarray(9,a) - sbvarray(10,a)),      # e10 = k*x5 + x6 - k*x6 - k*x9 - k*x10
    ↪k*x6 - k*x9 - k*x10
                        k*(-sbvarray(3,a) + sbvarray(4,a)
                          + sbvarray(11,a) - sbvarray(12,a)),     # e11 = -k*e3 + k*e4 + k*e11 - k*e12
    ↪+ k*e4 + k*e11 - k*e12
                        sbvarray(6,a) - sbvarray(12,a),            # e12 = x6 - x12
    ↪x12
```

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

3

```
[8]: ff = 10**6*np.array([0, 2, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0])

K2 = makeK([ 9.7* 10**6 if i in [1,2,3,4,6,9,12,14,15] else 6.8943 * 10**6 for
↪ i in range (1,16)])

A2 = B_bridge.transpose() @ K2 @ B_bridge
la.solve(A2,ff)
```

```
[8]: array([-0.0245008 ,  1.48884085, -0.          ,  1.74923139,  0.0245008 ,
 1.48884085,  0.36336953,  1.23365368,  0.          ,  1.65122818, -0.36336953,
 1.23365368])
```

## 4.1 inverse problem

The above shows that, given knowledge of the “spring constants”  $k_i$ , and the applied forces  $f_i$ , we can estimate the displacements  $x_i$ . This is the “*forward problem*”.

The inverse problem is this: given measurements of the displacements  $x_i$ , find the spring constants  $k_i$ .

There are many applications of such *inverse problems*.

For our bridge problem, note that consider the system

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

We consider a general case where there are  $m$  springs and  $n/2$  masses, so that there are  $n$  “displacement components” (in 2 dimensions).

Notice that

$$\begin{aligned} K B \mathbf{x} &= \begin{bmatrix} k_1 & 0 & \cdots & 0 \\ 0 & k_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & k_m \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \\ &= \text{diag}(B \mathbf{x}) \begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_m \end{bmatrix} \end{aligned}$$

where for a vector  $\mathbf{w} \in \mathbb{R}^m$ , we obtain an  $m \times m$  matrix

$$\text{diag}(\mathbf{w}) = \begin{bmatrix} w_1 & 0 & \cdots & 0 \\ 0 & w_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & w_m \end{bmatrix}$$

Thus we have

$$B^T K B \mathbf{x} = B^T \text{diag}(B \mathbf{x}) \mathbf{k} = \mathbf{f}$$

which gives some hope for finding the vector  $\mathbf{k}$  given knowledge of  $\mathbf{x}$  and  $\mathbf{f}$ .

## 4.2 the difficulty

Note that in the case of our bridge example, the number of “displacement components” we are tracking is  $n = 12$ , while the number of springs is  $m = 15$ .

Note that  $B^T$  is an  $n \times m$  matrix, that  $B \mathbf{x}$  is in  $\mathbb{R}^m$ , so that  $B^T \text{diag}(B \mathbf{x})$  is an  $n \times m$  matrix.

In our example, the linear equation  $B^T \text{diag}(B \mathbf{x}) \mathbf{k} = \mathbf{f}$  amounts to a system of 12 linear equations in 15 unknowns.

In general, if  $n < m$ , the system is *underdetermined*. As a consequence, the linear equation does not uniquely determine the spring constants  $k_i$  – i.e. the entries in the vector  $\mathbf{k}$ .

One way to fix this:

take measurements  $\mathbf{x}$  for *various different force loads*  $\mathbf{f}$ .

More precisely, consider different force loads  $\mathbf{f}_1, \dots, \mathbf{f}_p$ . For each of these force loads, determine the displacement vectors  $\mathbf{x}_1, \dots, \mathbf{x}_p$ .

We now obtain a  $pn \times m$  matrix system

$$\begin{bmatrix} B^T \text{diag}(B \mathbf{x}_1) \\ B^T \text{diag}(B \mathbf{x}_2) \\ \vdots \\ B^T \text{diag}(B \mathbf{x}_p) \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_m \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_p \end{bmatrix}.$$

If we choose  $p$  sufficiently large and if we make sure that the  $np \times m$  coefficient matrix has rank at least  $m$ , we expect to find a solution.

```
[4]: def diag(w):
      n = len(w)
      return np.array([ w[i]*sbv(i,n) for i in range(n) ])

diag([1,2,3,4])
```

```
[4]: array([[1, 0, 0, 0],
           [0, 2, 0, 0],
           [0, 0, 3, 0],
           [0, 0, 0, 4]])
```

Let’s consider some data. We have collected displacement measurements for three different force loads. The displacement measurements are outside of engineering tolerances!

We want to know which spring (=bridge component) is defective.

```
[11]: # measurements

      # for these forces
```

```

f1 = 10**6*np.array([0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
f2 = 10**6*np.array([0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0])
f3 = 10**6*np.array([0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0])

# measured the following displacements
x1 = np.array([ 0.04465819,  0.69735727,  0.08117014,  0.49972439,  0.0584346 ,
                0.29520329,  0.09315958,  0.52830165, -0.04706315,  0.50115528,
                -0.11445688,  0.25950424])

x2 = np.array([-0.03453953,  0.49972439,  0.01867222,  0.76169019,  0.03453953,
                0.49283618,  0.16411865,  0.45421336,  0.0034441 ,  0.66577233,
                -0.15233463,  0.44242935])

x3 = np.array([-0.04465819,  0.29520329, -0.08117014,  0.49283618, -0.0584346 ,
                0.69735727,  0.11302599,  0.25807335,  0.04706315,  0.49140529,
                -0.09172868,  0.52687076])

(f1,x1,f2,x2,f3,x3)

```

```

[11]: (array([      0, 2000000,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0]),
       array([ 0.04465819,  0.69735727,  0.08117014,  0.49972439,  0.0584346 ,
              0.29520329,  0.09315958,  0.52830165, -0.04706315,  0.50115528, -0.11445688,
              0.25950424]),
       array([      0,      0,      0, 2000000,      0,      0,      0,      0,
              0,      0,      0,      0]),
       array([-0.03453953,  0.49972439,  0.01867222,  0.76169019,  0.03453953,
              0.49283618,  0.16411865,  0.45421336,  0.0034441 ,  0.66577233, -0.15233463,
              0.44242935]),
       array([      0,      0,      0,      0,      0,      0, 2000000,      0,      0,
              0,      0,      0]),
       array([-0.04465819,  0.29520329, -0.08117014,  0.49283618, -0.0584346 ,
              0.69735727,  0.11302599,  0.25807335,  0.04706315,  0.49140529, -0.09172868,
              0.52687076]))

```

```

[6]: coeffMatrix = np.concatenate([ B_bridge.transpose() @ diag( B_bridge @ x ) for
    ↪x in [x1,x2,x3] ] )

kvalues,_,_,_ = la.lstsq(coeffMatrix , np.concatenate([f1,f2,f3]),rcond=None)

[ f"k{i+1} = {kvalues[i]}" for i in range(len(kvalues))]

```

```

[6]: ['k1 = 9699998.607722549',
      'k2 = 2000000.4090927793',
      'k3 = 9700000.122052612',
      'k4 = 9699998.86756195',

```

```
'k5 = 6894300.281687661',  
'k6 = 9700000.126240244',  
'k7 = 6894300.249842933',  
'k8 = 6894299.831309714',  
'k9 = 9699998.96686751',  
'k10 = 6894300.258596171',  
'k11 = 6894299.903274672',  
'k12 = 9700000.298712648',  
'k13 = 6894300.008658496',  
'k14 = 9700000.174220743',  
'k15 = 9700000.624519458']
```

We see that **k2** is different than expected and needs to be replaced!