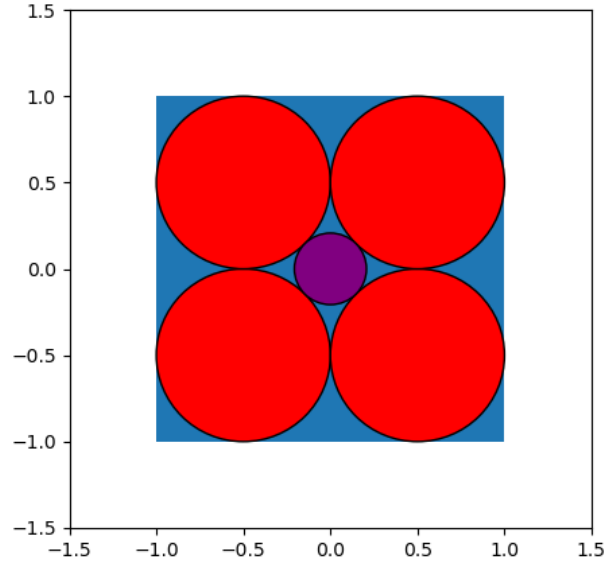


## Exercise 1

(a)

Situation for  $l=2$  and  $s=2$ :

$$Q_{2,2} = \{[x_1 x_2]^T \in \mathbb{R}^2 \mid |x_i| \leq 1 \text{ for all } i = 1, 2\} = [-1, 1]^2$$



(b)

First, let's calculate the radius of the inner hyperball for any  $l \in \mathbb{N}$ ,  $s \in \mathbb{R}_{>0}$ :

The distance from the center of the inner hyperball (equal to the center of the hypercube) to the center of one of the  $2^l$  outer hyperballs (doesn't matter which one) can be calculated the following:

$$d := \sqrt{l \cdot \left(\frac{s}{4}\right)^2}$$

Thus, the radius of the inner hyperball is equal to:

$$r := d - \frac{s}{4} = \sqrt{l \cdot \left(\frac{s}{4}\right)^2} - \frac{s}{4}$$

Now, we simply must solve the following inequality to find an  $l \in \mathbb{N}$  for an arbitrary but fixed  $s \in \mathbb{R}_{>0}$  such that  $B(Q_{l,s}) \not\subseteq Q_{l,s}$ :

$$\begin{aligned}
 \frac{s}{2} &< r \\
 \frac{s}{2} &< d - \frac{s}{4} \\
 \frac{s}{2} &< \sqrt{l \cdot \left(\frac{s}{4}\right)^2} - \frac{s}{4} \\
 \frac{3 \cdot s}{4} &< \sqrt{l} \cdot \frac{s}{4} \\
 3 &< \sqrt{l} \\
 l &> 9
 \end{aligned}$$

## Exercise 2

(a)

Let's first find all eigenvalues of  $A_c$ :

$$\begin{aligned}
& \{\lambda \in \mathbb{R} \mid \det(A_c - \lambda \cdot I_3) = 0\} \\
& \{\lambda \in \mathbb{R} \mid \det \begin{bmatrix} 2 - \lambda & 0 & c \\ 0 & 1 - \lambda & 0 \\ c & 0 & 1 - \lambda \end{bmatrix} = 0\} \\
& \{\lambda \in \mathbb{R} \mid -\lambda^3 + 4 \cdot \lambda^2 + c^2 \cdot \lambda - 5 \cdot \lambda + 2 - c^2\} \\
& \{\lambda \in \mathbb{R} \mid (\lambda - 1) \cdot (-\lambda^2 + 3\lambda + c^2 - 2)\} \\
& \left\{1, \frac{3 - \sqrt{4 \cdot c^2 + 1}}{2}, \frac{3 + \sqrt{4 \cdot c^2 + 1}}{2}\right\} =: \Lambda_{A_c}
\end{aligned}$$

So, if for all  $\lambda \in \Lambda_{A_c}$  it must hold that  $\lambda \geq 0$ , we must set  $c$  such that the following inequality holds:

$$\begin{aligned}
\frac{3 - \sqrt{4 \cdot c^2 + 1}}{2} &\geq 0 \\
3 - \sqrt{4 \cdot c^2 + 1} &\geq 0 \\
3 &\geq \sqrt{4 \cdot c^2 + 1} \\
9 &\geq 4 \cdot c^2 + 1 \\
8 &\geq 4 \cdot c^2 \\
2 &\geq c^2 \\
c &\in (-\sqrt{2}, \sqrt{2}) \subseteq \mathbb{R}
\end{aligned}$$

(b)

Using Theorem 5.19, we create an orthogonal matrix  $U \in \mathbb{R}^{n \times n}$  with the columns being the eigenvectors of  $A$ . We can now use Theorem 5.20 to create the diagonal matrix  $\Lambda \in \mathbb{R}^{n \times n}$  where the eigenvalue-entry  $\lambda_{i,i}$  corresponds to eigenvector in the column  $i$  of  $U$ .

A simple verification:

$$\begin{aligned}
A &= U \Lambda U^\top \\
A &= U \Lambda U^{-1} \\
AU &= U \Lambda \\
Av_i &= v_i \lambda_{i,i} \quad , \forall i \in [n], v_i := \text{col}_i(U) \\
Av_i &= \lambda_{i,i} v_i \quad , \forall i \in [n], v_i := \text{col}_i(U)
\end{aligned}$$

Since  $A$  is positive semi-definite, all entries  $\lambda_{i,i} \geq 0$ . Thus, we can create a diagonal matrix  $\Lambda'$  consisting of the entries  $\lambda'_{i,i} := \sqrt{\lambda_{i,i}}$ . Thus,  $\Lambda = \Lambda' \Lambda'^\top$ . Let  $B := U \Lambda'$ . Now:

$$\begin{aligned} A &= U \Lambda U^\top \\ A &= U (\Lambda' \Lambda'^\top) U^\top \\ A &= U (\Lambda' \Lambda'^\top) U^\top \\ A &= (U \Lambda') (\Lambda'^\top U^\top) \\ A &= (U \Lambda') (U \Lambda')^\top \\ A &= B B^\top \end{aligned}$$

(c)

Since  $A$  is symmetric, the following holds  $\forall x, y \in \mathbb{R}^n$ :

$$\langle Ax, y \rangle = \langle x, Ay \rangle$$

Let  $v_1 \in E_1, v_2 \in E_2$ . Then  $Av_1 = \lambda_1 v_1$  and  $Av_2 = \lambda_2 v_2$ .

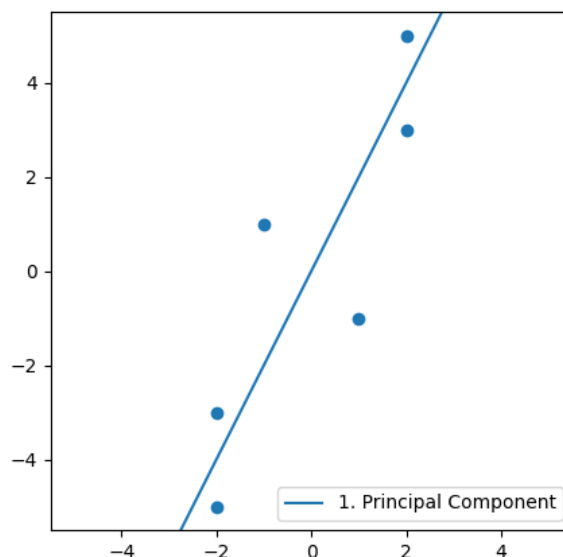
$$\begin{aligned} 0 &= \langle Av_1, v_2 \rangle - \langle v_1, Av_2 \rangle \\ &= \langle \lambda_1 v_1, v_2 \rangle - \langle v_1, \lambda_2 v_2 \rangle \\ &= (\lambda_1 - \lambda_2) \langle v_1, v_2 \rangle \end{aligned}$$

Since  $\lambda_1 \neq \lambda_2$  it follows, that  $\langle v_1, v_2 \rangle = 0$ .

## Exercise 3

(a)

Plot and estimate first principal component  $(\frac{1}{\sqrt{5}}, \frac{2}{\sqrt{5}})$



(b)

$$\begin{aligned}
 C &:= A^T A \\
 &= \begin{bmatrix} -2 & -2 & -1 & 1 & 2 & 2 \\ -5 & -3 & 1 & 1 & 3 & 5 \end{bmatrix} \begin{bmatrix} -2 & -5 \\ -2 & 3 \\ -1 & 1 \\ 1 & -1 \\ 2 & 3 \\ 2 & 5 \end{bmatrix} \\
 &= \begin{bmatrix} 18 & 18 \\ 18 & 70 \end{bmatrix}
 \end{aligned}$$

(c)

View code in the appendix for the calculations.

The eigenvector we got is:  $\begin{pmatrix} 0.298 \\ 0.954 \end{pmatrix}$

(d)

Well, they both point in the same general direction with a slope greater than 1. Our slope came out to be at 2, since eye-balling stuff tends to round to integer values. The more optimal slope however seems to be even greater than  $3 < \frac{0.954}{0.298}$ .

## Exercise 4

(a)

(b)

(c)

(d)

(e)

## Exercise 5

## Appendix

### Code for Exercise 3

```

1 import functools
2 import math
3
4
5 def length(v):
6     return math.sqrt(functools.reduce(lambda a, b: a + b, map(lambda v_i:
7         v_i ** 2, v)))
8

```

```
9 def truncate(number, decimals=0):
10     """
11     Returns a value truncated to a specific number of decimal places.
12     """
13     if not isinstance(decimals, int):
14         raise TypeError("decimal places must be an integer.")
15     elif decimals < 0:
16         raise ValueError("decimal places has to be 0 or more.")
17     elif decimals == 0:
18         return math.trunc(number)
19
20     factor = 10.0 ** decimals
21     return math.trunc(number * factor) / factor
22
23
24 def matrix_times_vector(A, v):
25     Av = list()
26     for i in range(len(A)):
27         assert(len(A[i]) == len(v))
28         Av.append(functools.reduce(
29             lambda a, b: a + b, map(lambda item: A[i][item[0]] * item[1],
30                                     enumerate(v))))
31     return Av
32
33 def power_iteration(A, x):
34     x_length = length(x)
35     v = tuple(map(lambda x_i: x_i / x_length, x))
36     old_trunc_v = tuple(map(lambda v_i: truncate(v_i, 3), v))
37     while True:
38
39         Av = matrix_times_vector(A, v)
40         Av_length = length(Av)
41         v = tuple(map(lambda v_i: v_i / Av_length, Av))
42
43         # check if does not converge no more
44         new_trunc_v = tuple(map(lambda v_i: truncate(v_i, 3), v))
45         if(old_trunc_v == new_trunc_v):
46             break
47         old_trunc_v = new_trunc_v
48     return v
49
50
51 if __name__ == '__main__':
52     C = [
53         [18, 18],
54         [18, 70]
55     ]
56     v = (1,1)
57     v_power = power_iteration(C,v)
58     print(tuple(map(lambda v_i: round(v_i, 3), v_power)))
```