Université de Genève

Analyse et Traitement de l'Information
14X026

TP1: Linear Algebra

Antoine Sutter
antoine.sutter@etu.unige.ch

October 2024

UNIVERSITÉ
DE GENÈVE
FACULTÉ DES SCIENCES
Département d'informatique

Contents

1. Matrix

1.1. Quadratic polynomial

Find a quadratic polynomial, say $f(x) = ax^2 + bx + c$, such that

$$f(1) = -5, f(2) = 1, f(3) = 11$$

First lets fill the values:

1. $f_1 = f(1) = a + b + c = -5$
2. $f_2 = f(2) = 4a + 2b + c = 1$
3. $f_3 = f(3) = 9a + 3b + c = 11$

First subscract f_1 from f_2 to get rid of c:

- $4a + 2b + c = 1$
- $a + b + c = -5$

That gives us equation f_4:

- $3a + b = 6$

Then subscract f_2 from f_3 to again get rid of c:

- $9a + 3b + c = 11$
- $4a + 2b + c = 1$

That gives us equation f_5:

- $5a + b = 10$

Now let's subscract f_4 from f_5 to solve a:

- $5a + b = 10$
- $3a + b = 6$

Which gives:

- $2a = 4$
- $a = 2$

Now that we know that $a = 2$ we can solve b by injecting a into f_4:

- $3a + b = 6$
- $6 + b = 6$
- $b = 0$

Finally we can inject a and b into f_1 to solve c:

- $a + b + c = -5$
- $2 + c = -5$
- $c = -7$

Therefore the quatdratic polynomial is the following:

- $f(x) = 2x^2 - 7$

1.2. Linear equation system

Let a, b be some fixed parameters. Solve the system of linear equations

$$\begin{cases} x + ay = 2 \\ bx + 2y = 3 \end{cases}$$

First I solve for x:

$$\begin{aligned} x + ay &= 2 \\ x &= 2 - ay \end{aligned}$$

Then substitude x into the 2nd equation

$$\begin{aligned} bx + 2y &= 3 \\ b(2 - ay) + 2y &= 3 \\ 2b - aby + 2y &= 3 \\ 2b + 2y - aby &= 3 \end{aligned}$$

Now move all the y on one side

$$2y - aby = 3 - 2b$$

Then factor y out and isolate y

$$\begin{aligned} y(2 - ab) &= 3 - 2b \\ y &= \frac{3 - 2b}{2 - ab} \end{aligned}$$

We can now take back the equation were we isolated x and inject y into it

$$\begin{aligned} x &= 2 - ay \\ x &= 2 - a\frac{3 - 2b}{2 - ab} \\ x &= 2 - \frac{a(3 - 2b)}{2 - ab} \\ x &= \frac{2(2 - ab)}{2 - ab} - \frac{a(3 - 2b)}{2 - ab} \\ x &= \frac{2(2 - ab) - a(3 - 2b)}{2 - ab} \\ x &= \frac{4 - 2ab - 3a + 2ab}{2 - ab} \\ x &= \frac{4 - 3a}{2 - ab} \end{aligned}$$

We now have a solution for both x and y with the condition that $2 - ab \neq 0$

$$\begin{aligned} x &= \frac{4 - 3a}{2 - ab} \\ y &= \frac{3 - 2b}{2 - ab} \end{aligned}$$

1.3. Matrix inverse

Find the inverse of the following matrix, if it exists

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 5 & 6 & 0 \end{pmatrix}$$

First let's assign a letter to each value in A

$$a = 1, b = 2, c = 3$$
$$d = 0, e = 1, f = 4$$
$$g = 5, h = 6, i = 0$$

From there we can calulate the deteminant and check if it's not 0. Otherwise it has no inverse solution.

$$\det(A) = a(ei - fh) - b(di - fg) + c(dh - eg)$$
$$\det(A) = 1(1 \cdot 0 - 4 \cdot 6) - 2(0 \cdot 0 - 4 \cdot 5) + 3(0 \cdot 6 - 1 \cdot 5)$$
$$\det(A) = (1 \cdot -24) - (2 \cdot -20) + (3 \cdot -5)$$
$$\det(A) = (-24) - (-40) + (-15)$$
$$\det(A) = 1$$

Because the deteminant isn't 0 we know there is a solution. We can now calculate the cofactor matrix of A. The cofactor Cij is calculated by eliminating the i-th row and j-th column, then calculating the determinant of the resulting 2x2 matrix, applying a sign change based on position.

$$C11 : \det(1 \cdot 0 - 4 \cdot 6) = -24$$
$$C12 : -\det(0 \cdot 0 - 4 \cdot 5) = 20$$
$$C13 : \det(0 \cdot 6 - 1 \cdot 5) = -5$$
$$C21 : -\det(2 \cdot 0 - 3 \cdot 6) = 18$$
$$C22 : \det(1 \cdot 0 - 3 \cdot 5) = -15$$
$$C23 : -\det(1 \cdot 4 - 2 \cdot 5) = 4$$
$$C31 : \det(2 \cdot 4 - 3 \cdot 1) = 5$$
$$C32 : -\det(1 \cdot 4 - 3 \cdot 0) = -4$$
$$C33 : \det(1 \cdot 1 - 2 \cdot 0) = 1$$

This gives us the $\text{Cof}(A)$ matrix:

$$\text{Cof}(A) = \begin{pmatrix} -24 & 20 & -5 \\ 18 & -15 & 4 \\ 5 & -4 & 1 \end{pmatrix}$$

We can then transpose the $\text{Cof}(A)$:

$$\text{Adj}(A) = \begin{pmatrix} -24 & 18 & 5 \\ 20 & -15 & -4 \\ -5 & 4 & 1 \end{pmatrix}$$

And that gives us thus the inverse of A.

1.4. Matrix equation for X

Solve the following matrix equation for X:

$$AX = B, \text{where } A = \begin{pmatrix} 1 & 3 \\ 2 & 5 \end{pmatrix}, B = \begin{pmatrix} 7 & 8 \\ 9 & 10 \end{pmatrix}$$

Let's start with the determinant of A:

$$\det(A) = (1 \cdot 5) - (3 \cdot 2) = 5 - 6 = -1$$

From there we can get A^{-1}

$$A^{-1} = \frac{1}{\det(A)} \cdot \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

With values filled we get the following

$$A^{-1} = \frac{1}{-1} \cdot \begin{pmatrix} 5 & -3 \\ -2 & 1 \end{pmatrix}$$

$$A^{-1} = \begin{pmatrix} -5 & 3 \\ 2 & -1 \end{pmatrix}$$

Now we need to multiply A^{-1} by B to find X

$$X = \begin{pmatrix} -5 & 3 \\ 2 & -1 \end{pmatrix} \cdot \begin{pmatrix} 7 & 8 \\ 9 & 10 \end{pmatrix}$$

$$X[0][0] = (-5 \cdot 7) + (3 \cdot 9) = -35 + 27 = -8$$
$$X[0][1] = (-5 \cdot 8) + (3 \cdot 10) = -40 + 30 = -10$$
$$X[1][0] = (2 \cdot 7) + (-1 \cdot 9) = 14 - 9 = 5$$
$$X[1][1] = (2 \cdot 8) + (-1 \cdot 10) = 16 - 10 = 6$$

Which means we have our solution:

$$X = \begin{pmatrix} -8 & -10 \\ 5 & 6 \end{pmatrix}$$

Because we can verify that $AX = B$

$$\begin{pmatrix} 1 & 3 \\ 2 & 5 \end{pmatrix} \begin{pmatrix} -8 & -10 \\ 5 & 6 \end{pmatrix} = \begin{pmatrix} 7 & 8 \\ 9 & 10 \end{pmatrix}$$

2. Math concepts behind code

Firsly before running the `some_script.py` file, I added the following code to supress the deprecation warnings as the output is really noisy otherwise. This really shouldn't be a thing but the point of this exercice is to explain the code, not fix it.

```
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

Example output with the warnings:

```
some_script.py:10: DeprecationWarning: datetime.datetime.utcnow() is deprecated
and scheduled for removal in a future version. Use timezone-aware objects to
represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
  print('START - %s'%datetime.utcnow().strftime('%Y-%m-%d %H:%M:%S.%f')[:-3]);
START - 2024-10-01 16:11:56.612
Result = 0.0
some_script.py:53: DeprecationWarning: datetime.datetime.utcnow() is deprecated
and scheduled for removal in a future version. Use timezone-aware objects to
represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
  print('END - %s\n'%datetime.utcnow().strftime('%Y-%m-%d %H:%M:%S.%f')[:-3]);
END - 2024-10-01 16:11:56.756
```

And without the warnings:

```
START - 2024-10-01 16:12:33.537
Result = 0.0
END - 2024-10-01 16:12:33.584
```

2.1. `def project_on_first(u, v)`

Like the same suggests, this fucking allows you to project a verctor into another vector, in this case project v onto u. This returns a new vector w which can be thought of as the shadow of u.

This consists of a few steps

1. dot product between u and v
2. dot product between u and u
3. Scalar product using the two dot products from above
4. Projecting the vector

A dot product between two matrices consist in multiplying each cell with the corresponding cell of the other vectoc and summing all the results. Consider u and v:

$$u = (3 \;\; 4), v = (1 \;\; 2)$$

Then a dot product of $u \cdot v$ is done as follow:

$$u \cdot v = (3 \;\; 4) \cdot (1 \;\; 2) = 3 \cdot 1 + 4 \cdot 2 = 11$$

Then doing the dot project $u \cdot u$ is done as follow:

$$u \cdot v = (3 \;\; 4) \cdot (3 \;\; 4) = 3 \cdot 3 + 4 \cdot 4 = 25$$

After that, we can do the scalar product, which consist of the following:

$$\text{scalar} = \frac{u \cdot v}{u \cdot u} \rightarrow \frac{11}{25}$$

The projection is then a multiplication of the result above with the u vector.

$$w = \frac{11}{25} \cdot (3 \;\; 4) = \left(\frac{33}{25} \;\; \frac{44}{25}\right) \approx (1.32 \;\; 1.76)$$

In summary, the function projects vector v onto vector u, resulting in a new vector that represents the part of v that is aligned with u.

2.2. Python function into math

```
# import numpy library as np in order to work on tensors
import numpy as np
# create 2 random vectors of length 7
u = np.random.randn(7)
v = np.random.randn(7)
# perform some computations
r=0
for ui, vi in zip(u, v):
  r += ui * vi
```

The code above is an implementation of a dot product between two matrices. Mathematically this can be expressed as such:

$$u \cdot v$$

If we want to be pedentic, we could rewite the last tree lines into one as such:

```
r = sum([ui * vi for ui, vi in zip(u, v)])
```

But of course would should use `numpy` for this instead:

```
r = np.dot(u, v)
```

2.3. Orthogonal vector

Complete the code to construct a vector v orthogonal to the vector u and of the same norm. Comment each line of your code.

```
# create 2 random vectors of length 7
u = np.random.randn(7)
v = np.random.randn(7)

# Perform some computations
r = np.dot(u, v)

# Creating a vector orthogonal to u and of the same norm
# Step 1: Generate a random vector
w = np.random.randn(7)

# Step 2: Make w orthogonal to u by subtracting its projection on u
proj_u_w = (np.dot(w, u) / np.dot(u, u)) * u
v_orthogonal = w - proj_u_w

# Step 3: Normalize v_orthogonal to make it of the same norm as the original v
v_orthogonal = v_orthogonal / np.linalg.norm(v_orthogonal) * np.linalg.norm(v)

# Check orthogonality and norm
dot_product = np.dot(u, v_orthogonal)
norm_v_orthogonal = np.linalg.norm(v_orthogonal)

print("Dot product (should be close to 0):", dot_product)
print("Norm of v (original):", np.linalg.norm(v))
print("Norm of v (orthogonal):", norm_v_orthogonal)
```

Example output from a few runs I've done:

```
Dot product (should be close to 0): 0.0
Norm of v (original): 3.692870668658719
Norm of v (orthogonal): 3.6928706686587187
```

2.4. Cosine of vectors

The cosine between two vectors u and v is defined as such:

$$\cos \alpha = \frac{u \cdot v}{\|u\| \cdot \|v\|}$$

In order to do that in python, we can use the following code. It's important to check that neither the norm of u or v is zero as the computation would be impossible otherwise.

```python
def cosine_similarity(u: np.ndarray, v: np.ndarray) -> np.float64:
    norm_u = np.linalg.norm(u)
    norm_v = np.linalg.norm(v)

    if norm_u == 0 or norm_v == 0:
        raise ValueError("cannot use vectors with norm 0")

    return np.dot(u, v) / (norm_u * norm_v)

u = np.array([1, 2, 3])
v = np.array([4, 5, 6])
cos_theta = cosine_similarity(u, v)
print(f'Cosine of the angle between u and v: {cos_theta}')
```

In this case, the angle is `0.9746318461970762`.

3. Eigenvalues, Eigenvectors, and Determinants

3.1. Find the determinant, eigenvectors, and eigenvalues of the matrix

$$A = \begin{pmatrix} 5 & 6 & 3 \\ -1 & 0 & 1 \\ 1 & 2 & -1 \end{pmatrix}$$

3.1.1. Determinant

The deteminant of a 3x3 matrix is defined as such:

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = aei + bfg + cdh - ceg - bdi - afh$$

Which gives us the following result

$$0 + 6 + -6 - 0 - 6 - 10$$

$$0 + 6 - 6 - 0 - 6 - 10 = -16$$

We can verify using the following python code:

```
import numpy as np

A = np.array([
    [5, 6, 3],
    [-1, 0, 1],
    [1, 2, -1],
])

print(np.linalg.det(A))
```

And gives us a similar result:

```
-15.999999999999998
```

3.1.2. Eigen values

First we need to find the eigen values of A. To do so, we need to solve the deteminant for $A - \lambda I$ where λ is the unknown eigen value and I is the indentity matrix.

$$A - \lambda I = \begin{pmatrix} 5 & 6 & 3 \\ -1 & 0 & 1 \\ 1 & 2 & -1 \end{pmatrix} - \lambda \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = 0$$

$$\det \begin{pmatrix} 5-\lambda & 6 & 3 \\ -1 & -\lambda & 1 \\ 1 & 2 & -1-\lambda \end{pmatrix} = 0$$

We can then compute the deteminant and solve for λ

$$(5-\lambda)(-\lambda)(-1-\lambda) + 6 - 6 + 3\lambda + 6(-1-\lambda) - 2(5-\lambda)$$

$$(5-\lambda)(-\lambda)(-1-\lambda) + 3\lambda - 6 - 6\lambda - 10 + 2\lambda$$

$$(5-\lambda)(-\lambda)(-1-\lambda) - \lambda - 16$$

$$(5-\lambda)(\lambda + \lambda^2) - \lambda - 16$$

$$5\lambda + 5\lambda^2 - \lambda^2 - \lambda^3 - \lambda - 16$$

$$-\lambda^3 + 4\lambda^2 + 4\lambda - 16$$

Therefore, the polynomial function of A is:

$$p(\lambda) = -\lambda^3 + 4\lambda^2 + 4\lambda - 16 = 0$$

Because this example is being solved by hand, we can safely assume that the roots we're looking for are integers. We also know that the roots have to be factors of the 16 in this example. We can therefore just try out all the factors of 16 and find out if any of them work.

- $p(1) = -9$
- $p(-1) = -15$
- $p(2) = 0$
- $p(-2) = 0$
- $p(4) = 0$
- $p(-4) = 96$
- $p(8) = -240$
- $p(-8) = 720$
- $p(16) = -3024$
- $p(-16) = 5040$

Therefore the eigen values of A are 2, –2 and 4.

We can verify that we got it correctly with the following python code

```
import numpy as np

A = np.array([
    [5, 6, 3],
    [-1, 0, 1],
    [1, 2, -1],
])

print(np.linalg.eigvals(A))
```

```
[-2.  4.  2.]
```

3.1.3. Eigen vectors

Now that we have the eigen values of A, we need to solve the following:

$$\begin{pmatrix} 5-\lambda & 6 & 3 \\ -1 & -\lambda & 1 \\ 1 & 2 & -1-\lambda \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

In order to find a solution for $\lambda = 2$, let's lock $X = 1$.

$$\begin{pmatrix} 3 & 6 & 3 \\ -1 & -2 & 1 \\ 1 & 2 & -3 \end{pmatrix} \begin{pmatrix} 1 \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

We can get our first 2 equations like so:

$$f_1 = 3 + 6Y + 3Z = 0$$
$$f_2 = -1 - 2Y + Z = 0$$

Then $f_1 - 3f_2$ to extract Y

$$f_1 - 3f_2 = (3 + 6Y + 3Z) - (-3 - 6Y + 3Z) = 6 + 12Y = 0$$
$$12Y = -6$$
$$Y = -\frac{6}{12} = -\frac{1}{2}$$

Now let's insert Y into f_2:

$$f_2 = -1 - 2\left(-\frac{1}{2}\right) + Z = 0$$
$$f_2 = -1 + 1 + Z = 0$$
$$Z = 0$$

We can now verify with f_3:

$$1 + 2\left(-\frac{1}{2}\right) = 0$$
$$1 - 1 = 0$$

Therefore, our eigen vector for the eigen value $\lambda = 2$ is:

$$\begin{pmatrix} 1 \\ -\frac{1}{2} \\ 0 \end{pmatrix}$$

We can scale it up to get rid of the fraction

$$\begin{pmatrix} 2 \\ -1 \\ 0 \end{pmatrix}$$

We can now verify that our original equation is true:

$$\begin{pmatrix} 3 & 6 & 3 \\ -1 & -2 & 1 \\ 1 & 2 & -3 \end{pmatrix} \begin{pmatrix} 2 \\ -1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

We can then verify we got it correctly using the following python code:

```python
import numpy as np

# Original vector
A = np.array([
    [5, 6, 3],
    [-1, 0, 1],
    [1, 2, -1],
])

# Add the eigen value
B = A - np.identity(3) * 2
print(B)

# Our result we're testing
eig_vec_1 = np.array([2, -1, 0])
print(eig_vec_1)
print(np.dot(B, eig_vec_1))
```

```
[[ 3.  6.  3.]
 [-1. -2.  1.]
 [ 1.  2. -3.]]
[ 1.  -0.5  0. ]
[0. 0. 0.]
```

Now we need to do the same for $\lambda = -2$. This time we lock $Y = 1$ because locking $X = 1$ would lead too all variables disapearing when doing subscractions, meaning X is 0 which we will confirm is true later. I will not comment every steps from now one since it's similar to the previous example.

$$\lambda = -2 \rightarrow \begin{pmatrix} 7 & 6 & 3 \\ -1 & 2 & 1 \\ 1 & 2 & 1 \end{pmatrix} \begin{pmatrix} X \\ 1 \\ Z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$f_1 = 7X + 6 + 3Z = 0$$

$$f_2 = -1X + 2 + Z = 0$$

$$f_1 - 3f_2 = (7X + 6 + 3Z) - (-3X + 6 + 3Z) = 10X = 0$$

$$X = 0$$

$$f_2 = 2 + Z = 0$$

$$Z = -2$$

$$f_3 = 2 - 2 = 0$$

Therefore the eigen vector for $\lambda = -2$ is:

$$\begin{pmatrix} 0 \\ 1 \\ -2 \end{pmatrix}$$

And we can again confirm via python that the result is correct.

```python
import numpy as np

# Original vector
A = np.array([
    [5, 6, 3],
    [-1, 0, 1],
    [1, 2, -1],
])

# Add the eigen value
B = A - np.identity(3) * -2
print(B)

# Our result we're testing
eig_vec_1 = np.array([0, 1, -2])
print(eig_vec_1)
print(np.dot(B, eig_vec_1))
```

```
[[ 7.  6.  3.]
 [-1.  2.  1.]
 [ 1.  2.  1.]]
[ 0  1 -2]
[0. 0. 0.]
```

Now we need to do the same for $\lambda = 4$

$$\lambda = 4 \rightarrow \begin{pmatrix} 1 & 6 & 3 \\ -1 & -4 & 1 \\ 1 & 2 & -5 \end{pmatrix} \begin{pmatrix} 1 \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$f_1 = 1 + 6Y + 3Z = 0$$

$$f_2 = -1 - 4Y + Z = 0$$

$$f_1 - 3f_2 = (1 + 6Y + 3Z) - (-3 - 12Y + 3Z) = 18Y + 4 = 0$$

$$Y = -\frac{4}{18} = -\frac{2}{9}$$

$$f_2 = -1 - 4\left(-\frac{2}{9}\right) + Z$$

$$f_2 = -1 - 4\left(-\frac{2}{9}\right) + Z$$

$$f_2 = -1 + \frac{8}{9} + Z = 0$$

$$Z = 1 - \frac{8}{9}$$

$$f_3 = 1 + 2\left(-\frac{2}{9}\right) - 5\left(1 - \frac{8}{9}\right) = 0$$

Therefore the eigen vector for $\lambda = 4$ is:

$$\begin{pmatrix} 1 \\ -\frac{2}{9} \\ 1 - \frac{8}{9} \end{pmatrix}$$

And we can again confirm via python that the result is correct. This time however we do not get exactly the vector `[0. 0. 0]` but this is because of the limitation of floating point numbers. We are close enough.

```
import numpy as np

# Original vector
A = np.array([
    [5, 6, 3],
    [-1, 0, 1],
    [1, 2, -1],
])

# Add the eigen value
B = A - np.identity(3) * 4
print(B)

# Our result we're testing
eig_vec_1 = np.array([1, -(2/9), 1-(8/9)])
print(eig_vec_1)
print(np.dot(B, eig_vec_1))
```

```
[[ 1.  6.  3.]
 [-1. -4.  1.]
 [ 1.  2. -5.]]
[ 1.         -0.22222222  0.11111111]
[ 2.22044605e-16  0.00000000e+00 -2.22044605e-16]
```

3.2. Positive semidefinite proof

The covariance matrix for the n samples $x_1, ..., x_n$, each represented by a $d \times 1$ column vector, is given by

$$C = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \mu)(x_i - \mu)^T$$

where C is a $d \times d$ matrix and $\mu = \sum_{i=1}^{n} \frac{x_i}{n}$ is the sample mean. Prove that C is always positive semidefinite. (Note: A symmetric matrix C of size $d \times d$ is *positive semidefinite* if $v^T C v \geq 0$ for every $d \times 1$ vector v.)

3.2.1. C is symmetric

We can see that C is symmetric because $(x_i - \mu)(x_i - \mu)^T$ is a $d \times d$ matrix and symmetric itself because transposing it gives back the same result.

$$\left((x_i - \mu)(x_i - \mu)^T\right)^T = (x_i - \mu)(x_i - \mu)^T$$

Therefore, C being a sum of symmetric matrices, C is symmetric.

3.2.2. Positive semidefinite

Proving that C is semidefinite means that $v^T C v \geq 0$ where v is a $d \times 1$ vector. We can then do the following:

$$v^T C v = v^T \left(\tfrac{1}{n-1} \textstyle\sum_{i=1}^{n} (x_i - \mu)(x_i - \mu)^T \right) v$$

We can move the multiplication and sum outside:

$$v^T C v = \frac{1}{n-1} v^T \left(\textstyle\sum_{i=1}^{n} (x_i - \mu)(x_i - \mu)^T \right) v$$

From there, move v^T inside the sum:

$$v^T C v = \frac{1}{n-1} \sum_{i=1}^{n} v^T (x_i - \mu)(x_i - \mu)^T v$$

We can then factor v and simplify like this:

$$v^T C v = \frac{1}{n-1} \sum_{i=1}^{n} \left(v(x_i - \mu)^T \right)^2$$

Because squaring can only produce positive values, we know that C is semipositive.

$$v^T C v = \frac{1}{n-1} \sum_{i=1}^{n} \left(v(x_i - \mu)^T \right)^2 \geq 0$$

3.3. Eigenvalues of covariance matrices

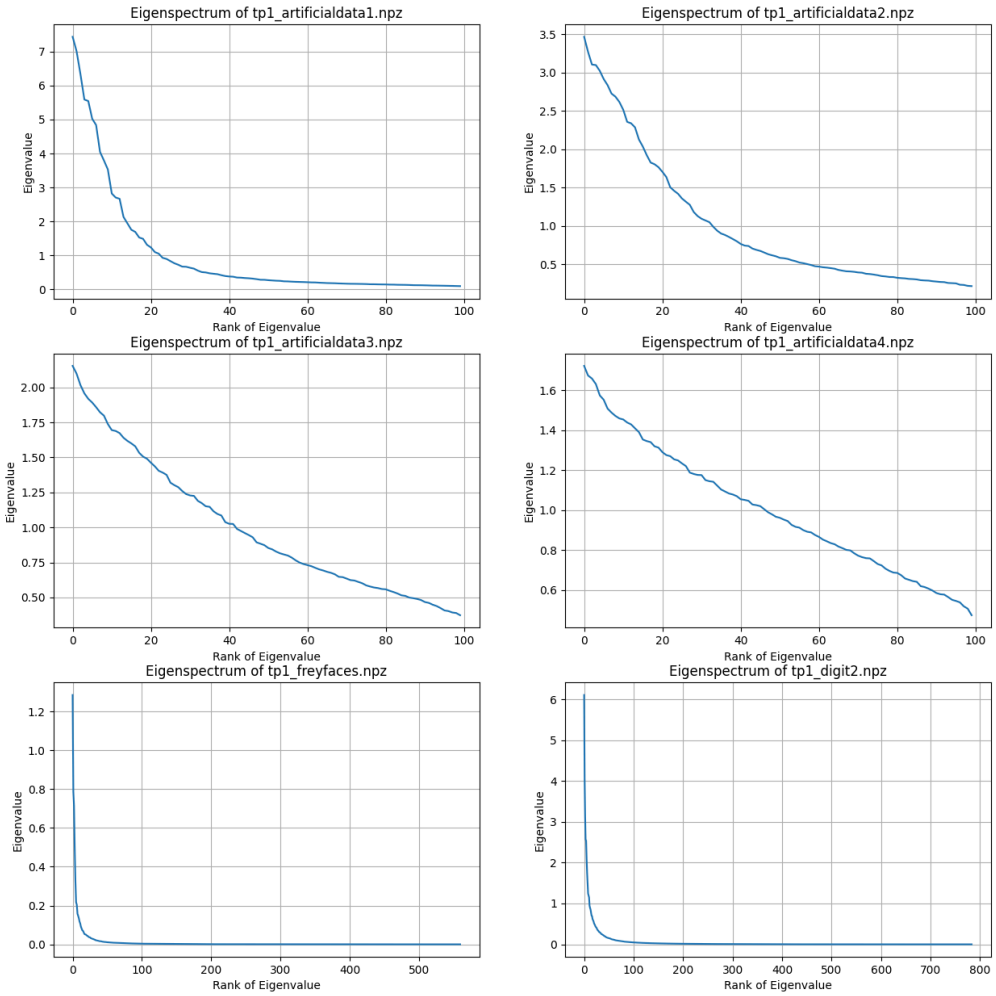

Figure 2: Eigenspecta of the data sets

The determinant of a covariance matrix is equal to the product of its eigenvalues. Large eigenvalues indicate directions with high variance, while small or zero eigenvalues suggest low or no variance, indicating potential linear dependence. The spectrum of eigenvalues helps understand the dimensionality and variance structure: a steep drop-off implies effective lower-dimensional representation, while a gradual decline indicates spread across dimensions. Differences in eigenvalues between datasets arise from intrinsic dimensionality, data variance, feature correlations, and noise levels, providing insights into each dataset's properties.

3.4. Signular matrix for $\lambda = 0$

An eigenvalue λ of a matrix A is defined as a scalar such that there exists a non-zero vector x for which the following equation is true:

$$Ax = \lambda x$$

Imagine the case $\lambda = 0$:

$$Ax = 0x$$
$$Ax = 0$$

This implies that the matrix A has a non-zero vector x in its null space. In other words, there is some non-zero vector x such that $Ax = 0$. By definition, a matrix A is singular if it does not have full rank, or equivalently, if its null space contains a non-zero vector. Since we have found a non-zero vector x such that $Ax = 0$, this implies that A is singular.

3.5. Proof that the determinant of A must be zero if any eigenvalue is zero

The determinant of a matrix A, denoted as $\det(A)$, can be expressed in terms of its eigenvalues. For an $n \times n$ matrix, if $\lambda_1, \lambda_2, ..., \lambda_n$ are the eigenvalues of A, then the determinant of A is given by the product of its eigenvalues:

$$\det(A) = \lambda_1 \lambda_2 ... \lambda_n$$

Therefore, if any of the eigen value is 0, the product will be 0 and the deteminant will also be 0.

4. Computing Projection Onto a Line

There are given a line $\alpha : 3x + 4y = -6$ and a point A with the coordinates $(-1, 3)$.

4.1. Distance from point A

Let's first rewrite α as such:

$\alpha : 3x + 4y + 6 = 0$

Then we can use the Point-Line Distance Formula, defined as such:

$$d = \frac{|ax_1 + by_1 + c|}{\sqrt{a^2 + b^2}}$$

Given the following variables:

$$a = 3, b = 4, c = 6$$
$$x_1 = -1, y_1 = 3$$

We get the following equation that we can then simpliy to get our result.

$$d = \frac{3 \cdot -1 + 4 \cdot 3 + 6}{\sqrt{3^2 + 4^2}}$$
$$d = 3(-1) + 4(3) + 6$$
$$d = -3 + 12 + 6$$
$$d = 15$$

Therefore the distance from A to α is 3.

4.2. Python visualized

We can visualize the projection of A using the following python code:

```python
import numpy as np
import matplotlib.pyplot as plt

if __name__ == "__main__":
    # Define the line parameters and point A
    a, b, c = 3, 4, 6
    point = np.array([-1., 3.], dtype=float)

    def line(x: float) -> float:
        return (-a * x - c) / b

    # Normal vector to the line
    normal_vector = np.array([a, b])

    # Compute the numerator as dot product plus the constant term
    num = np.dot(normal_vector, point) + c

    # Compute the denominator as the norm of the normal vector
    den = np.linalg.norm(normal_vector)

    # Distance from point to line
    distance = abs(num) / den

    print(f"dot product of normal vector and point: {num - c}")
    print(f"constant term: {c}")
    print(f"nominator (dot product + constant): {num}")
    print(f"denominator (norm of normal vector): {den}")
    print(f"distance: {distance}")

    # Calculate the perpendicular point projection onto the line
    x_perp = (b * (b * point[0] - a * point[1]) - a * c) / (a ** 2 + b ** 2)
    y_perp = (a * (a * point[1] - b * point[0]) - b * c) / (a ** 2 + b ** 2)

    x_vals = np.linspace(-10, 10, 400)
    y_vals = line(x_vals)

    plt.plot(x_vals, y_vals, label='3x + 4y + 6 = 0')

    # Plot point A
    plt.scatter(point[0], point[1], color='red', zorder=5)
    plt.text(point[0], point[1], '(-1, 3)', fontsize=12,
verticalalignment='bottom')

    # Plot the perpendicular line projection
    plt.scatter(x_perp, y_perp, color='blue', zorder=5)
    plt.plot([point[0], x_perp], [point[1], y_perp], linestyle='--', color='gray',
             label=f'Distance = {round(distance, 2)}')

    # Annotation for the projected point
    plt.text(x_perp, y_perp, 'P(Projection)', fontsize=12,
verticalalignment='bottom')
```

```
    plt.xlabel('x')
    plt.ylabel('y')
    plt.axhline(0, color='black', linewidth=0.5)
    plt.axvline(0, color='black', linewidth=0.5)
    plt.grid(color='gray', linestyle='--', linewidth=0.5)
    plt.legend()
    plt.title('Projection of Point A onto Line')
    plt.show()
```

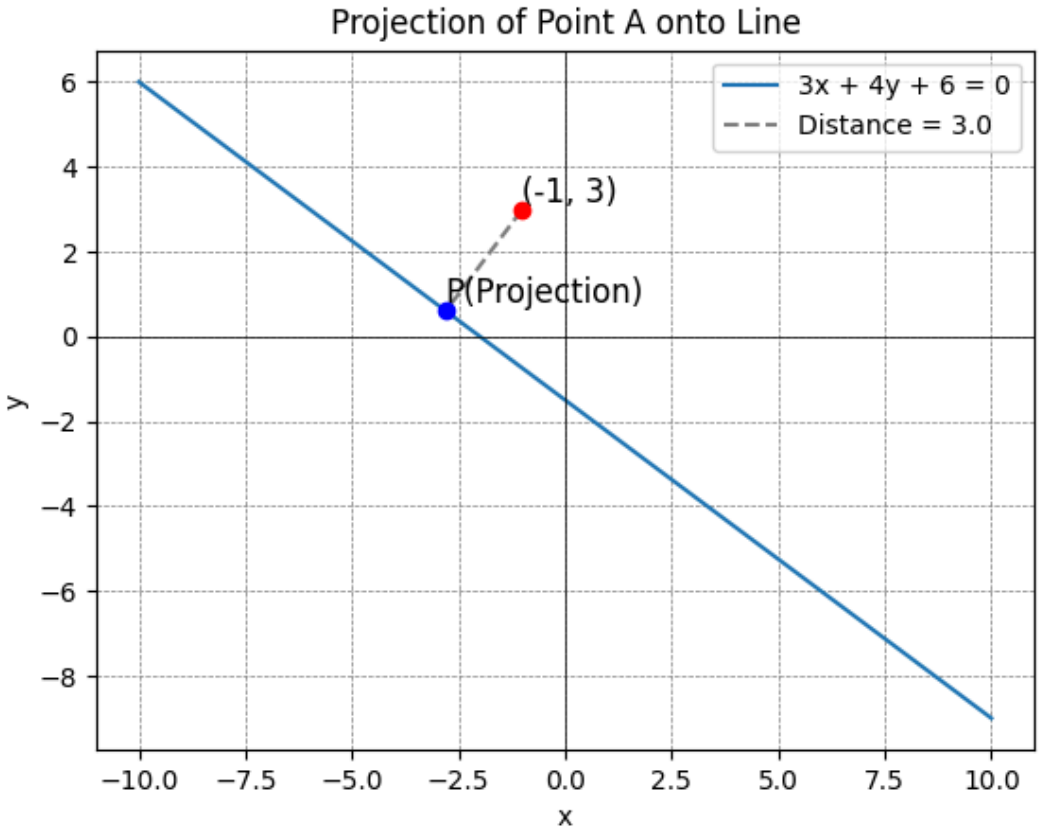

Figure 3: Projection of point A