Solutions

Luxformel

Vectors and Matrices in Python Solutions

Exercise 1 – Vectors in $\mathbb{R}^7$

Consider the following vectors:

$$u = (0.5, 0.4, 0.4, 0.5, 0.1, 0.4, 0.1), \quad v = (-1, -2, 1, -2, 3, 1, -5)$$

Using **Python and NumPy**:

1. Check whether u and v are unit vectors.

2. Compute the dot product of u and v.

3. Determine if u and v are orthogonal.

```python
import numpy as np

u = np.array([0.5, 0.4, 0.4, 0.5, 0.1, 0.4, 0.1])
v = np.array([-1, -2, 1, -2, 3, 1, -5])

# 1. Check if u and v are unit vectors
norm_u = np.linalg.norm(u)
norm_v = np.linalg.norm(v)
print(norm_u, norm_v)

# 2. Dot product
dot_uv = np.dot(u, v)
print(dot_uv)

# 3. Orthogonality
print(np.isclose(dot_uv, 0))
```

Exercise 2 – Norms and Orthogonality

Consider the following vectors in $\mathbb{R}^9$:

$$u = (1, 2, 5, 2, -3, 1, 2, 6, 2), \quad v = (-4, 3, -2, 2, 1, -3, 4, 1, -2), \quad w = (3, 3, -3, -1, 6, -1, 2, -5, -7)$$

Using Python:

1. Check which pairs of these vectors are orthogonal.

2. Calculate the **Euclidean norm** of u.

3. Calculate the **infinity norm** of w.

u = np.array([1, 2, 5, 2, -3, 1, 2, 6, 2]) v = np.array([-4, 3, -2, 2, 1, -3, 4, 1, -2]) w = np.array([3, 3, -3, -1, 6, -1, 2, -5, -7])

```
# 1. Orthogonality
pairs = {'u·v': np.dot(u, v), 'u·w': np.dot(u, w), 'v·w': np.dot(v, w)}
print(pairs)

# 2. Euclidean norm of u
print(np.linalg.norm(u))

# 3. Infinity norm of w
print(np.linalg.norm(w, np.inf))
```

Exercise 3 – Matrix Operations

Consider the matrices:

$$A = \begin{pmatrix} 2 & -2 \\ 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 1 \\ 6 & 2 \end{pmatrix}, \quad C = \begin{pmatrix} 4 & 1 & -1 \\ 2 & 5 & -2 \\ 1 & 1 & 2 \end{pmatrix}, \quad D = \begin{pmatrix} -3 & 1 & -1 \\ -7 & 5 & -1 \\ -6 & 6 & -2 \end{pmatrix}$$

$$E = \begin{pmatrix} 2 \\ -1 \\ 3 \end{pmatrix}, \quad F = \begin{pmatrix} -2 & 1 & 0 \end{pmatrix}, \quad G = \begin{pmatrix} 1 & -1 & 0 & 0 \\ 1 & 4 & 0 & 0 \end{pmatrix}$$

Using Python:

1. Compute (if possible):

- $A+B$, $B-A$, $B+C$, AB, BA, BG, CE, EF, FE

2. Compute the transposes of A and B and then their product.
   Observe and explain any property you find.

```
A = np.array([[2, -2], [0, 1]])
B = np.array([[3, 1], [6, 2]])
C = np.array([[4, 1, -1], [2, 5, -2], [1, 1, 2]])
D = np.array([[-3, 1, -1], [-7, 5, -1], [-6, 6, -2]])
E = np.array([[2], [-1], [3]])
F = np.array([[-2, 1, 0]])
G = np.array([[1, -1, 0, 0], [1, 4, 0, 0]])

# 1. Basic operations
print(A + B)
print(B - A)
print(np.dot(A, B))
print(np.dot(B, A))

# Some operations will raise shape errors (e.g. B + C), check compatibility before computing:
def try_op(func, *args):
    try:
        print(func(*args))
    except ValueError as e:
        print(f"Operation not possible: {e}")

try_op(np.add, B, C)
try_op(np.dot, B, G)
try_op(np.dot, C, E)
try_op(np.dot, E, F)
try_op(np.dot, F, E)

# 2. Transpose and product
AT, BT = A.T, B.T
print(np.dot(AT, BT))
```

Exercise 4 – Matrix Rank and Norms

Consider the following matrices:

$$A = \begin{pmatrix} 2 & -2 \\ -3 & 1 \\ 5 & -3 \end{pmatrix}, \quad B = \begin{pmatrix} 4 & 4 & 4 \\ -2 & 3 & -7 \\ 2 & 5 & -7 \end{pmatrix}, \quad C = \begin{pmatrix} 4 & -1 & 2 \\ -8 & 2 & -4 \\ 2 & 1 & -4 \end{pmatrix}$$

Using Python:

1. Compute $A^T B$ and $C + B$.

2. Determine which of A, B, and C are full rank.

3. Compute the **Frobenius norm** of C and the **spectral norm** of A.

4. Attempt to compute the inverse of B.

```
A = np.array([[2, -2], [-3, 1], [5, -3]])
B = np.array([[4, 4, 4], [-2, 3, -7], [2, 5, -7]])
C = np.array([[4, -1, 2], [-8, 2, -4], [2, 1, -4]])

# 1. A^T B and C + B
print(A.T @ B)
print(C + B)

# 2. Rank
for name, M in {'A': A, 'B': B, 'C': C}.items():
    print(f"rank({name}) =", np.linalg.matrix_rank(M))

# 3. Norms
frobenius_C = np.linalg.norm(C, 'fro')
spectral_A = np.linalg.norm(A, 2)
print(frobenius_C, spectral_A)

# 4. Inverse of B (if invertible)
try:
    B_inv = np.linalg.inv(B)
    print(B_inv)
except np.linalg.LinAlgError:
    print("B is not invertible.")
```

Exercise 5 – Determinants

Using the matrices from **Exercise 3**, and Python:

1. Compute $\det(A)$, $\det(B)$, and $\det(AB)$.

2. Compute $\det(C)$ and $\det(D)$.

```
A = np.array([[2, -2], [0, 1]])
B = np.array([[3, 1], [6, 2]])
C = np.array([[4, 1, -1], [2, 5, -2], [1, 1, 2]])
D = np.array([[-3, 1, -1], [-7, 5, -1], [-6, 6, -2]])

print(np.linalg.det(A))
print(np.linalg.det(B))
print(np.linalg.det(A @ B))
print(np.linalg.det(C))
print(np.linalg.det(D))
```

Exercise 6 – Inverses

Consider the matrices:

$$A = \begin{pmatrix} 2 & -1 \\ 4 & 3 \end{pmatrix}, \quad B = \begin{pmatrix} 2 & 0 \\ 4 & 5 \end{pmatrix}, \quad C = \begin{pmatrix} 6 & -9 \\ -4 & 6 \end{pmatrix}, \quad D = \begin{pmatrix} -1 & 6 & 2 \\ 0 & 1 & 0 \\ 3 & 0 & -5 \end{pmatrix}$$

Using Python, calculate (if possible) the inverses of A, B, C, and D.

```
A = np.array([[2, -1], [4, 3]])
B = np.array([[2, 0], [4, 5]])
C = np.array([[6, -9], [-4, 6]])
D = np.array([[-1, 6, 2], [0, 1, 0], [3, 0, -5]])

for name, M in {'A': A, 'B': B, 'C': C, 'D': D}.items():
    try:
        print(f"{name} inverse:\n", np.linalg.inv(M))
    except np.linalg.LinAlgError:
        print(f"{name} is not invertible.")
```

Exercise 7 – Invertibility

Consider the matrix:

$$A = \begin{pmatrix} 2 & 2 & 3 \\ -2 & 7 & 4 \\ -3 & -3 & -4 \\ -8 & 2 & 3 \end{pmatrix}$$

Using Python:

1. Add a column to A to make it invertible.

2. Remove a row from A to make it invertible.

3. Compute AA^T and check if it is invertible.

4. Compute A^TA and check if it is invertible.

```
A = np.array([[2, 2, 3],
              [-2, 7, 4],
              [-3, -3, -4],
              [-8, 2, 3]])

# 1. Add a column (example: zeros)
A_aug = np.hstack([A, np.ones((4, 1))])
print("New A with extra column:\n", A_aug)

# 2. Remove a row (example: first row)
A_red = A[1:, :]
print("Reduced A:\n", A_red)

# 3. AA^T and 4. A^T A
AAT = A @ A.T
ATA = A.T @ A
print("AA^T invertible?", np.linalg.matrix_rank(AAT) == AAT.shape[0])
print("A^T A invertible?", np.linalg.matrix_rank(ATA) == ATA.shape[0])
```

Exercise 8 – Matrix Inversion and Systems of Equations

Using Python:

1. Compute the inverse of

$$M = \begin{pmatrix} 3 & 2 & -1 \\ 1 & -1 & 1 \\ 2 & -4 & 5 \end{pmatrix}.$$

2. Use this inverse to solve the linear system:

$$\begin{cases} 3x + 2y - z = 5 \\ x - y + z = 1 \\ 2x - 4y + 5z = -3 \end{cases}$$

```
M = np.array([[3, 2, -1],
              [1, -1, 1],
              [2, -4, 5]])

b = np.array([5, 1, -3])

M_inv = np.linalg.inv(M)
print("M inverse:\n", M_inv)

# Solve using inverse
x = M_inv @ b
print("Solution:", x)

# Verify using numpy solver
print("Check:", np.allclose(M @ x, b))
```

Exercise 9 – Solving Linear Systems

Use Python to solve the following systems:

1.

$$\begin{cases} 2x + 3y + 5z = 2 \\ 7x + z = -1 \\ -2y + 2z = 3 \end{cases}$$

2.

$$\begin{cases} x + 2y - z = 2 \\ 2x + 5y + 4z = 3 \\ 3x + 7y + 4z = 1 \end{cases}$$

```
# 1st system
```

```
A1 = np.array([[2, 3, 5],
               [7, 0, 1],
               [0, -2, 2]])
b1 = np.array([2, -1, 3])

x1 = np.linalg.solve(A1, b1)
print("System 1 solution:", x1)

# 2nd system
A2 = np.array([[1, 2, -1],
               [2, 5, 4],
               [3, 7, 4]])
b2 = np.array([2, 3, 1])

x2 = np.linalg.solve(A2, b2)
print("System 2 solution:", x2)
```