DSC 210 Numerical Linear Algebra, Fall 2025

Homework problems for Topic 1: *Linear Algebra Basics*

Student Name (PID): James Doan (A15903661)

Write your solutions to the following problems by typing them in LaTeX. Unless otherwise noted by the problem's instructions, show your work and provide justification for your answer. Homework is due via Gradescope at **23rd October 2025, 11:59 PM**.
Late Policy: If you submit your homework after the deadline we will apply a late penalty of 10% per day.

Guidelines for Homework Related Questions:

(a) As a general rule, we can help you understand the homework problems and explain the material from the corresponding lectures, but we cannot give you the entire solution.

(b) Regarding debugging programming questions: We ask you to do some debugging on your own first, including printing out intermediate values in your algorithms, trying a simpler version of the problem, etc.

(c) We will not be pre-grading the homework, i.e. we won't confirm if the answer you have is correct.

AI Usage Policy:

(a) Code: You may use LLMs to debug your code; however, you may not use LLMs to generate your entire code, and code must be reviewed and tested.

(b) Writing: You may use LLMs to correct grammar, style and latex issues; however, you may not use LLMs to generate entire solutions, sentences or paragraphs. All writing must be in your own voice.

Academic Integrity Policy:

The UC San Diego Academic Integrity Policy (formerly the Policy on Integrity of Scholarship) is effective as of September 25, 2023 and applies to any cases originating on or after September 25, 2023. The university expects both faculty and students to honor the policy. For students, this means that all academic work will be done by the individual to whom it's assigned, without unauthorized aid of any kind. If violations of academic integrity occur, the same Sanctioning Guidelines apply regardless of which policy was effective for that case.

For more information on how the policy is implemented, refer to the most current procedures. Remember: When in doubt about what constitutes appropriate collaboration or resource use, please ask TAs before proceeding. It's always better to clarify expectations than to risk an academic integrity violation. Academic integrity violations can have serious consequences for your academic record, and you will get zero grades.

You can access the Homework Template using the following link: `https://www.overleaf.com/read/vfhcmsppvskp`

Question 1: Property of triangular matrices (20 points)

Given L_1 and L_2 are two lower triangular matrices of size $n \times n$, prove that $L_1 L_2$ is also a lower triangular matrix. Further, prove by induction that multiplication of $m\,(m > 2)$ lower triangular matrices $(L_1, L_2, ..., L_m)$ is also a lower triangular matrix.

Solution:

Base Case: Suppose that for two matricies A, B of size m = 2, they are defined as

$$\mathbf{A} = \begin{bmatrix} a_{11} & 0 \\ a_{21} & a_{22} \end{bmatrix}$$

,

$$\mathbf{B} = \begin{bmatrix} b_{11} & 0 \\ b_{21} & b_{22} \end{bmatrix}$$

, The matrix product C is calcuated as follows:

$$\begin{aligned} C &= AB \\ &= \begin{bmatrix} a_{11} & 0 \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & 0 \\ b_{21} & b_{22} \end{bmatrix} \\ &= \begin{bmatrix} a_{11}b_{11} & 0 \\ a_{21}b_{11} + a_{22}b_{21} & a_{22}b_{22} \end{bmatrix} \end{aligned}$$

which indicates that C is also a lower triangular matrix.

We can then generalize this to $L_1, L_2 \in \mathbb{R}^{n \times n}$ where the ij-th entry of L_1, L_2 is

$$(L_1, L_2)_{ij} = \sum_{k=1}^{n} (L_1)_{ik}(L_2))_{kj}$$

where $i \geq k \geq j$, $\therefore i \geq j$. Should there by any term where $i < j$, it implies that there is no k that satisfies the previous statement, so every term in the sum is 0. This is what defines a lower triangular matrix.

Induction Assumption: Assume that for some m = k, consider the product of a combination of lower triangular matricies $L_1, L_2, \ldots, Ł_k$ is also a lower triangular matrix, as proven in the base case. Let this matrix be denoted as P. Now consider the lwoer triangular matricies $L_1, L_2, \ldots, Ł_k, L_{k+1}$, again a lower triangular matrix. let this matrix be denoted as Q. The product of P and Q (two lower triangular matricies) will result in another lower triangular matrix, as proven in the base case. Therefore, by induction, for lower triangular matricies with $m \geq 2$ the products are also lower triangular.

■

Question 2: Matrix operations (20 points)

Let $\mathbf{B}$ be a 4×4 matrix to which we apply the following 7 operations sequentially and get a final matrix $\mathbf{D}$:

(i) double column 1,

(ii) halve row 3,

(iii) add row 1 to row 4,

(iv) interchange columns 2 and 3,

(v) subtract row 2 from each of the other rows,

(vi) replace column 4 by column 1,

(vii) delete column 2 (so that the column dimension is reduced by 1).

(a) Express each operation (i) to (vii) as a matrix and the final matrix $\mathbf{D}$ as a product of 8 matrices. (10 points)

(b) Write the final result again as a product of $\mathbf{ABC}$, i.e. write matrix $\mathbf{D} = \mathbf{ABC}$ and find $\mathbf{A}, \mathbf{C}$. (5 points)

(c) Write Python code to verify your answers in parts a, and b. Show the answers and code. (5 points) Let

$$\mathbf{B} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

Hint: You can use NumPy for matrix operations.

Solution:

(a) (i) double column 1

$$\mathbf{B_i} = \begin{bmatrix} 2 & 2 & 3 & 4 \\ 10 & 6 & 7 & 8 \\ 18 & 10 & 11 & 12 \\ 26 & 14 & 15 & 16 \end{bmatrix}$$

(ii) halve row 3

$$\mathbf{B_{ii}} = \begin{bmatrix} 2 & 2 & 3 & 4 \\ 10 & 6 & 7 & 8 \\ 9 & 5 & 5.5 & 6 \\ 26 & 14 & 15 & 16 \end{bmatrix}$$

(iii) add row 1 to row 4

$$\mathbf{B_{iii}} = \begin{bmatrix} 2 & 2 & 3 & 4 \\ 10 & 6 & 7 & 8 \\ 9 & 5 & 5.5 & 6 \\ 28 & 16 & 18 & 20 \end{bmatrix}$$

(iv) interchange columns 2 and 3

$$\mathbf{B_{iv}} = \begin{bmatrix} 2 & 3 & 2 & 4 \\ 10 & 7 & 6 & 8 \\ 9 & 5.5 & 5 & 6 \\ 28 & 18 & 16 & 20 \end{bmatrix}$$

(v) subtract row 2 from each of the other rows

$$\mathbf{B_v} = \begin{bmatrix} -8 & -4 & -4 & -4 \\ 10 & 7 & 6 & 8 \\ -1 & -1.5 & -1 & -2 \\ 18 & 11 & 10 & 12 \end{bmatrix}$$

(vi) replace column 4 by column 1

$$\mathbf{B_{vi}} = \begin{bmatrix} -8 & -4 & -4 & -8 \\ 10 & 7 & 6 & 10 \\ -1 & -1.5 & -1 & -1 \\ 18 & 11 & 10 & 18 \end{bmatrix}$$

(vii) delete column 2 (so that the column dimension is reduced by 1)

$$\mathbf{B_{vii}} = \begin{bmatrix} -8 & -4 & -8 \\ 10 & 6 & 10 \\ -1 & -1 & -1 \\ 18 & 10 & 18 \end{bmatrix}$$

(viii) matrix D, a result of multiplying all 8 previous matricies

$$\mathbf{D} = \begin{bmatrix} 16834799830 & 9851435670 & 16834799830 \\ 38819874470 & 22716723814 & 38819874470 \\ 60804949110 & 35582011958 & 60804949110 \\ 82790023750 & 48447300102 & 82790023750 \end{bmatrix}$$

(b) A is the product of all row elementary operations, which are given below.

i. halve row 3

$$\mathbf{E_1} = \mathbf{B_{ii}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ii. add row 1 to row 4

$$\mathbf{E_2} = \mathbf{B_{iii}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

iii. subtract row 2 from each of the other rows

$$\mathbf{E_3} = \mathbf{B_v} = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

C is the product of all column elementary operations, which are given below.

i. double column 1

$$\mathbf{F_1} = \mathbf{B_i} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ii. interchange columns 2 and 3

$$\mathbf{F_2} = \mathbf{B_{iv}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

iii. replace column 4 with column 1

$$\mathbf{F_3} = \mathbf{B_{vi}} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Deleting a column cannot be represented by multiplying with a square matrix

i. delete Column 2

$$\mathbf{B_{vii}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Therefore,

$$\begin{aligned} D &= ABC \\ &= E_3E_2E_1BF_1F_2F_3 \\ &= \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 16834799830 & 9851435670 & 16834799830 \\ 38819874470 & 22716723814 & 38819874470 \\ 60804949110 & 35582011958 & 60804949110 \\ 82790023750 & 48447300102 & 82790023750 \end{bmatrix} \text{after column 2 is removed from the final matrix} \end{aligned}$$

(c) For code you may use:

```
# imports
import numpy as np

# variable initalization
b = np.array([[1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12],
              [13, 14, 15, 16]], dtype=float)

# verify part a
## double column 1
b_i = b.copy()
b_i[:, 0] *= 2

## halve row 3
b_ii = b_i.copy()
b_ii[2] /= 2

## add row 1 to row 4
b_iii = b_ii.copy()
```

```python
    b_iii[3] += b_iii[0]

    ## interchange columns 2 and 3
    b_iv = b_iii.copy()
    b_iv[:, [1, 2]] = b_iv[:, [2, 1]]

    ## subtract row 2 from each of the other rows
    b_v = b_iv.copy()
    b_v[[0, 2, 3]] -= b_v[1]

    ## replace column 4 by column 1
    b_vi = b_v.copy()
    b_vi[:, 3] = b_vi[:, 0]

    ## delete column 2 (so that the column dimension is reduced by 1)
    b_vii = np.delete(b_vi, 1, axis=1)

    ## compute matrix D, product of 8 matricies b to b_vii
    d = b @ b_i @ b_ii @ b_iii @ b_iv @ b_v @ b_vi @ b_vii

    q2_dict = {
        "b": b,
        "b_i": b_i,
        "b_ii": b_ii,
        "b_iii": b_iii,
        "b_iv": b_iv,
        "b_v": b_v,
        "b_vi": b_vi,
        "b_vii": b_vii,
        "d": d
    }
    for k, v in q2_dict.items():
        print(f"{k}:\n{v}\n")

    # verify part (b)
    ## elementary row operations
    A = np.eye(4)
    A[2,2] = 0.5 # halve row 3
    A[3,0] = 1 # add row 1 to row 4
    A[[0,2,3],1] = -1 # subtract row 2 from others

    ## elementary column operations
    C = np.eye(4)
    C[0,0] = 2 # double column 1
    C[:, [1,2]] = C[:, [2,1]] # swap columns 2 and 3
    C[:,3] = C[:,0] # replace column 4 by column 1

    ## deleting column 2 to compute final matrix D
    D_full = A @ b @ C # because it was lowercase earlier
    D = np.delete(D_full, 1, axis=1)
    print("Final matrix D:\n", D)
```

Question 3: Matrix properties (20 points)

Prove that if a matrix $\mathbf{A}$ is triangular (upper or lower) then $\mathbf{A}^{-1}$ is also triangular. Further, use the result to show that if $\mathbf{A}$ is both triangular and orthogonal, then it is diagonal.

Solution:

Theorem. The product of two more more lower triangular matricies also is a lower triangular matrix. (Proved in Question 1)

Proof. Given that a matrix $\mathbf{A}$ is a lower triangular matrix, there exists a possible sequence of elementary row operations that transforms A into an identity matrix I_n, given that $A \in \mathbb{R}^{n\times n}$. This sequence of elementary row operations can be defined as:

$$E_p E_{p-1} \dots E_2 E_1 A = I_n$$

This is known as forward substitution. To get the inverse, that is to say get $\mathbf{A}^{-1}$, divide the identity matrix I_n by $\mathbf{A}$. When looking at the above equation, we see that this operation is done on the left hand side; the left hand side must also be divided by A.

$$\therefore \mathbf{A}^{-1} = E_p E_{p-1} \dots E_2 E_1$$

■

*Corollary.*This proof was done for a lower triangular matrix, however due to the properties of triangular matricies having 0 values on either side of the diagonal, the reasoning is valid for upper triangular matricies as well when back substitution is used.

Proof. If a matrix $\mathbf{A}$ is lower triangular, then matrix $\mathbf{A}^\top$ is upper triangular (and vice versa). Additionally, if a matrix $\mathbf{A}$ is orthogonal, then $\mathbf{A}^{-1} = \mathbf{A}^\top$.

By the first proof in this section, $\mathbf{A}^{-1}$ is also lower triangular. Therefore, $\mathbf{A}^{-1} = \mathbf{A}^\top$ is both upper and lower triangular. By definition of diagonal matricies, only lower and upper triangular matricies are diagonal matricies. By implication, if a matrix is both diagonal and triangular, it would logically conclude that the matrix is also orthogonal.

■

Question 4: p-norm inequalities (20 points)

Let $\mathbf{x}$ be a real m-vector, the vector p-norms $\|\mathbf{x}\|_p$ are related by various inequalities, often involving the dimension of the vector, i.e. m. For each of the following, prove the inequality and give an example of a nonzero vector $\mathbf{x}$ for which *equality* is satisfied.

(a) $\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_2$. (7 points)

(b) $\|\mathbf{x}\|_2 \leq \sqrt{m} \cdot \|\mathbf{x}\|_\infty$. (7 points)

(c) Plot a 2D contour of $\|\mathbf{x}\|_\infty = 1$, on the same chart also highlight regions where $\|\mathbf{x}\|_2 < 1$, $\|\mathbf{x}\|_2 = 1$ and $\|\mathbf{x}\|_2 > 1$. (6 points)

Solution:

Proof 1. Let x be the m-vector norm $x = (x_1, x_2, \ldots, x_m)^\top \in \mathbb{R}^m$. By definition, a norm is a function $\|x\| \cdot \mathbb{R}^m \to \mathbb{R}$ that assigns a real-valued length to each vector, measuring the value of x. Also by definition, let

$$\|\mathbf{x}\|_\infty = \max_{1 \leq i \leq m} |x_i|, \quad \text{and} \quad \|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{m} x_i^2}.$$

Now, let p represent the norm where $|x_p| = \|\mathbf{x}\|_\infty$. Thus by comparing the two terms we see that

$$\|\mathbf{x}\|_2^2 = \sum_{i=1}^{m} x_i^2 > x_p^2 = \|\mathbf{x}\|_\infty^2$$

When p = 2 (the Euclidean norm),

$$\|\mathbf{x}\|_2 > \|\mathbf{x}\|_\infty$$

$$\|\mathbf{x}\|_\infty = \max_i |x_i|^p$$

■

Proof 2. Building off of proof 1, let $\|\mathbf{x}\|_\infty = \max_i |x_i|_p$. Assume then that $|x_i| \leq \|\mathbf{x}\|_\infty$.

$$\Sigma_{i=1}^m x_i \leq m\|\mathbf{x}\|_\infty = \Sigma_{i=1}^m \|\mathbf{x}\|_\infty$$

$$\therefore \Sigma_{i=1}^m x_i^2 \leq m\|\mathbf{x}\|_\infty^2 = \Sigma_{i=1}^m \|\mathbf{x}\|_\infty^2$$

$$\therefore \|\mathbf{x}\|_2 \leq \sqrt{m}\|\mathbf{x}\|_\infty$$

■

Code

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

x = np.linspace(-1.5, 1.5, 400)
X, Y = np.meshgrid(x, x)
norm_inf = np.maximum(np.abs(X), np.abs(Y))
norm2 = np.sqrt(X**2 + Y**2)

plt.figure(figsize=(6,6))
plt.contour(X, Y, norm_inf, levels=[1], colors='black', linewidths=2, label='||x||_inf=1')
plt.contour(X, Y, norm2, levels=[1], colors='blue', linewidths=2, label='||x||_2=1')
plt.contourf(X, Y, norm2, levels=[0,1], colors=['#a8dadc'], alpha=0.5)  # ||x||_2 < 1 region
plt.contourf(X, Y, norm2, levels=[1,2], colors=['#e63946'], alpha=0.3)  # ||x||_2 > 1 region

plt.gca().set_aspect('equal', adjustable='box')
plt.title(r"Contours of $\|x\|_\infty=1$ and $\|x\|_2$")
plt.xlabel("$x_1$")
plt.ylabel("$x_2$")
plt.grid(True)
plt.show()
```

Question 5: Basic vector operations (20 points)

Given two 3-dimensional vectors $\mathbf{a}, \mathbf{b}$, and three matrices $A \in \mathbb{R}^{2\times3}, B \in \mathbb{R}^{3\times2}, C \in \mathbb{R}^{2\times3}$, scalars β_1, β_2 with the values below:

$$\mathbf{a} = \begin{bmatrix}1\\3\\5\end{bmatrix} \mathbf{b} = \begin{bmatrix}2\\4\\6\end{bmatrix} \mathbf{A} = \begin{bmatrix}1 & 2 & 3\\2 & 4 & 6\end{bmatrix} \mathbf{B} = \begin{bmatrix}7 & 8\\9 & 10\\11 & 12\end{bmatrix} \mathbf{C} = \begin{bmatrix}1 & 0 & 0\\0 & 0 & 1\end{bmatrix}$$
$$\beta_{\mathbf{1}} = 4, \beta_{\mathbf{2}} = 5$$

(a) Compute the following operations by hand and show your work: (15 points)

(i) Vector operations: $\mathbf{a} + \mathbf{b}, \quad \beta_1\mathbf{a}, \quad \mathbf{a} \circ \mathbf{b}, \quad \beta_1\mathbf{a} + \beta_2\mathbf{b}$ where $\circ$ denotes component-wise multiplication. (3 points)

(ii) Matrix operations: $\beta_1\mathbf{A}, \quad \mathbf{A} + \mathbf{B}, \quad \mathbf{A} + \mathbf{C}$. (3 points)

(iii) Transpose operations: $(\mathbf{AB})^\top, \quad \mathbf{B}^\top\mathbf{A}^\top, \quad (\mathbf{A}^\top)^\top, \quad (\mathbf{A} + \mathbf{C})^\top$. (3 points)

(iv) Inner products and outer product: $\langle\mathbf{a}, \mathbf{b}\rangle, \quad \langle\mathbf{b}, \mathbf{a}\rangle, \quad \langle\mathbf{a}, \mathbf{a}\rangle, \quad \langle\mathbf{b}, \mathbf{b}\rangle, \quad \beta_1\langle\mathbf{a}, \mathbf{b}\rangle, \quad \langle\beta_1\mathbf{a}, \mathbf{b}\rangle, \quad \mathbf{b}\mathbf{a}^\top$. (3 points)

(v) Determinants: $\det(\mathbf{AB}), \quad \det(\mathbf{BC})$. (3 points)

(b) Implement all the parts above using python (any programming language of your choice) and show the answers and code. (5 points)

Solution:

1. (a) **I: Vector Operations**

i.

$$\mathbf{a} + \mathbf{b} = \begin{bmatrix}1\\3\\5\end{bmatrix} + \begin{bmatrix}2\\4\\6\end{bmatrix}$$
$$= \boxed{\begin{bmatrix}3\\7\\11\end{bmatrix}}$$

ii.

$$\beta_1\mathbf{a} = 4 * \begin{bmatrix}1\\3\\5\end{bmatrix}$$
$$= \boxed{\begin{bmatrix}4\\12\\20\end{bmatrix}}$$

iii.

$$\mathbf{a} \circ \mathbf{b} = \begin{bmatrix} 1 \cdot 2 \\ 3 \cdot 4 \\ 5 \cdot 6 \end{bmatrix}$$
$$= \boxed{\begin{bmatrix} 2 \\ 12 \\ 30 \end{bmatrix}}$$

iv.

$$\beta_1 \mathbf{a} + \beta_2 \mathbf{b} = 4 * \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} + 5 \cdot \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}$$
$$= \begin{bmatrix} 4 \\ 12 \\ 20 \end{bmatrix} + \begin{bmatrix} 10 \\ 20 \\ 30 \end{bmatrix}$$
$$= \boxed{\begin{bmatrix} 14 \\ 32 \\ 50 \end{bmatrix}}$$

(b) **II: Matrix Operations**

i.

$$\beta_1 \mathbf{A} = 4 * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$
$$= \boxed{\begin{bmatrix} 4 & 8 & 12 \\ 16 & 20 & 30 \end{bmatrix}}$$

ii.

$$\mathbf{A} + \mathbf{B} = \boxed{NotPossible, sizeA \neq sizeB}$$

iii.

$$\mathbf{A} + \mathbf{C} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
$$= \boxed{\begin{bmatrix} 2 & 2 & 3 \\ 4 & 5 & 7 \end{bmatrix}}$$

(c) **III: Transpose Operations**

i.

$$(\mathbf{AB})^\top = \begin{bmatrix} 1 \cdot 7 + 2 \cdot 9 + 3 \cdot 11 & 1 \cdot 8 + 2 \cdot 10 + 3 \cdot 12 \\ 4 \cdot 7 + 5 \cdot 9 + 6 \cdot 11 & 4 \cdot 8 + 5 \cdot 10 + 6 \cdot 12 \end{bmatrix}^\top$$
$$= \begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix}^\top$$
$$= \boxed{\begin{bmatrix} 58 & 139 \\ 64 & 154 \end{bmatrix}}$$

ii.

$$
\begin{aligned}
\mathbf{B}^\top \mathbf{A}^\top &= \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix}^\top \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}^\top \\
&= \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \\
&= \begin{bmatrix} 7 \cdot 1 + 8 \cdot 3 + 9 \cdot 5 & 7 \cdot 2 + 8 \cdot 4 + 9 \cdot 6 \\ 4 \cdot 1 + 11 \cdot 3 + 12 \cdot 5 & 10 \cdot 2 + 11 \cdot 4 + 12 \cdot 6 \end{bmatrix} \\
&= \boxed{\begin{bmatrix} 58 & 139 \\ 64 & 154 \end{bmatrix}}
\end{aligned}
$$

iii.

$$
\begin{aligned}
(\mathbf{A}^\top)^\top &= (\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}^\top)^\top \\
&= \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^\top \\
&= \boxed{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}}
\end{aligned}
$$

iv.

$$
\begin{aligned}
(\mathbf{A} + \mathbf{C})^\top &= (\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix})^\top \\
&= (\begin{bmatrix} 2 & 2 & 3 \\ 4 & 5 & 7 \end{bmatrix})^\top \\
&= \boxed{\begin{bmatrix} 2 & 2 \\ 3 & 4 \\ 5 & 7 \end{bmatrix}}
\end{aligned}
$$

(d) **IV: Inner Products and Outer Product**

i.

$$
\begin{aligned}
\langle \mathbf{a}, \mathbf{b} \rangle &= 1 \cdot 2 + 3 \cdot 4 + 5 \cdot 6 \\
&= \boxed{44}
\end{aligned}
$$

ii.

$$
\begin{aligned}
\langle \mathbf{b}, \mathbf{a} \rangle &= 2 \cdot 1 + 4 \cdot 3 + 6 \cdot 5 \\
&= \boxed{44}
\end{aligned}
$$

iii.

$$
\begin{aligned}
\langle \mathbf{a}, \mathbf{a} \rangle &= 1^2 + 3^2 + 5^2 \\
&= \boxed{35}
\end{aligned}
$$

iv.

$$\begin{aligned}\langle \mathbf{b}, \mathbf{b} \rangle &= 2^2 + 4^2 + 6^2 \\ &= \boxed{56}\end{aligned}$$

v.

$$\begin{aligned}\beta_1 \langle \mathbf{a}, \mathbf{b} \rangle &= 4 \cdot (1 \cdot 2 + 3 \cdot 4 + 5 \cdot 6) \\ &= \boxed{176}\end{aligned}$$

vi.

$$\begin{aligned}\langle \beta_1 \mathbf{a}, \mathbf{b} \rangle & 4 \cdot 1 \cdot 2 + 4 \cdot 3 \cdot 4 + 4 \cdot 5 \cdot 6 \\ &= \boxed{176}\end{aligned}$$

vii.

$$\begin{aligned}\mathbf{b}\mathbf{a}^\top &= \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix} \begin{bmatrix} 1 & 3 & 5 \end{bmatrix} \\ &= \boxed{\begin{bmatrix} 2 & 6 & 10 \\ 4 & 12 & 20 \\ 6 & 18 & 30 \end{bmatrix}}\end{aligned}$$

(e) **V: Determinants**

i.

$$\begin{aligned}\det(\mathbf{AB}) &= \det(\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix}) \\ &= \det(\begin{bmatrix} 1 \cdot 7 + 2 \cdot 9 + 3 \cdot 11 & 1 \cdot 8 + 2 \cdot 10 + 3 \cdot 12 \\ 4 \cdot 7 + 5 \cdot 9 + 6 \cdot 11 & 4 \cdot 8 + 5 \cdot 10 + 6 \cdot 12 \end{bmatrix}) \\ &= \det(\begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix}) \\ &= 58 \cdot 154 - 64 \cdot 139 \\ &= \boxed{36}\end{aligned}$$

ii.

$$\begin{aligned}\det(\mathbf{BC}) &= \det(\begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}) \\ &= \det(\begin{bmatrix} 7 \cdot 1 + 9 \cdot 0 + 11 \cdot 0 & 8 \cdot 1 + 10 \cdot 0 + 12 \cdot 0 \\ 7 \cdot 0 + 9 \cdot 0 + 11 \cdot 1 & 8 \cdot 0 + 10 \cdot 0 + 12 \cdot 1 \end{bmatrix}) \\ &= \det(\begin{bmatrix} 7 & 0 & 8 \\ 9 & 0 & 10 \\ 11 & 0 & 12 \end{bmatrix}) \\ &= \boxed{0}\end{aligned}$$

2.

```python
# imports
import numpy as np

# variable initialization
vector_a = np.array([[1],
                     [3],
                     [5]])

vector_b = np.array([[2],
                     [4],
                     [6]])

matrix_a = np.array([[1, 2, 3],
                     [4, 5, 6]])

matrix_b = np.array([[7, 8],
                     [9, 10],
                     [11, 12]])

matrix_c = np.array([[1, 0 , 0],
                     [0, 0, 1]])

beta_a, beta_b = 4, 5

## verify part (i), vector operations
vector_add = vector_a + vector_b
vector_scalar = beta_a * vector_a
dot_prod = vector_a * vector_b
linear_combo = (vector_scalar) + (beta_b * vector_b)
vector_operations_dict = {
    "vector_add": vector_add,
    "vector_scalar": vector_scalar,
    "dot_prod": dot_prod,
    "linear_combo": linear_combo
}

for k, v in vector_operations_dict.items():
    print(f"{k}:\n{v}\n")

## verify part (ii), matrix operations
matrix_scalar = beta_a * matrix_a
# matrix_add_b = matrix_a + matrix_b # incompatible matrix
matrix_add_ac = matrix_a + matrix_c
matrix_operations_dict = {
    "matrix_scalar": matrix_scalar,
    "matrix_add_ac": matrix_add_ac
}

for k, v in matrix_operations_dict.items():
    print(f"{k}:\n{v}\n")

## verify part (iii), transpose operations
transpose_prod = np.transpose(matrix_a @ matrix_b)
transpose_prod_new = transpose_prod
transpose_transpose = np.transpose(np.transpose(matrix_a))
transpose_sum = np.transpose(matrix_a + matrix_c)
transpose_dict = {
    "transpose_prod": transpose_prod,
    "transpose_prod_new": transpose_prod_new,
    "transpose_transpose": transpose_transpose,
    "transpose_sum": transpose_sum
```

```python
    }
    for k, v in transpose_dict.items():
        print(f"{k}:\n{v}\n")

    ## verify part (iv), inner and outer products
    inner_ab = np.inner(vector_a, vector_b)
    inner_ba = np.inner(vector_b, vector_a)
    inner_aa = np.inner(vector_a, vector_a)
    inner_bb = np.inner(vector_b, vector_b)
    scalar_out = beta_a * inner_ab
    scalar_in = np.inner((beta_a * vector_a), vector_b)
    outer_prod = vector_b @ np.transpose(vector_a)
    inner_prod_dict = {
        "inner_ab": inner_ab,
        "inner_ba": inner_ba,
        "inner_aa": inner_aa,
        "inner_bb": inner_bb,
        "scalar_out": scalar_out,
        "scalar_in": scalar_in,
        "outer_prod": outer_prod
    }

    for k, v in inner_prod_dict.items():
        print(f"{k}:\n{v}\n")

    # verify part (v), determinants
    det_ab = np.linalg.det(matrix_a @ matrix_b)
    det_bc = np.linalg.det(matrix_b @ matrix_c)
    det_dict = {
        "det_ab": det_ab,
        "det_bc": det_bc
    }

    for k, v in det_dict.items():
        print(f"{k}:\n{v}\n")

```