Projects Statistical Analysis of Networks and Systems (SANS-MIRI) Homework 2

By

Umberto Salviati

Universitat Politècnica de Catalunya

Barcelona, October 2023

Contents

List of Figures

Chapter 1

Exercise 1

Chapter 2

Exercise 2

2.1 Assignemet

Let us assume the following matrices: $A =$

$$\begin{bmatrix} 4 & 3 & 2 & 1 \\ 1 & 2 & 1 & 3 \\ 3 & 1 & 0 & 2 \\ 2 & 3 & 4 & 5 \end{bmatrix}$$

$B =$

$$\begin{bmatrix} 5 & 0 & 2 & -1 \\ 0 & 2 & 0 & -1 \\ 2 & 0 & 1 & 0 \\ -1 & -1 & 0 & 2 \end{bmatrix}$$

$C =$

$$\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

$D =$

$$\begin{bmatrix} 4 & 3 \\ 3 & 1 \end{bmatrix}$$

a) specify the positive definiteness of each matrix

b) investigate whether the matrices produced by the eigendecomposition are orthogonal matrices

c) specify the projection of vector $b^\top = (1, 2, 1, 5, 5)$ onto the subspace generated by matrix E and vector $b^\top = (1, 2, 1, 5)$ onto the subspace generated by F (use

these matrices only in this subsection), $E =$

$$\begin{bmatrix} 4 & 3 & 2 & 1 \\ 1 & 2 & 1 & 3 \\ 3 & 1 & 0 & 2 \\ 2 & 3 & 4 & 5 \\ 2 & 1 & 3 & 2 \end{bmatrix}$$

$F =$

$$\begin{bmatrix} 5 & 0 & 2 & -1 \\ 0 & 2 & 0 & -1 \\ 2 & 0 & 1 & 0 \\ -1 & -1 & 0 & 2 \end{bmatrix}$$

d) take vectors $b_1^\top = (1, 0)$, $b_2^\top = (0, 1)$, $b_3^\top = (-1, 0)$, and $b_4^\top = (0, -1)$ and plot how these vectors are transformed using matrix D, i.e., $d = D \cdot b_i$. Repeat the process showing step by step the transformation using an eigendecomposition of D, i.e., show how the vector b is transformed by the eigenvector and eigenvalues matrices in the unit circle.

2.2 Solution exercise 2

Given a symmetruc real matrix A if An $n \times n$ symmetric real matrix A is said to be positive definite if

$$x^\top Ax > 0$$

for all non-zero vectors $x \in \mathbb{R}^n$, and it is said to be negative definite if

$$x^\top Ax < 0$$

An $n \times n$ symmetric real matrix A is said to be positive semi-definite if

$$x^\top Ax \geq 0$$

for all non-zero vectors $x \in \mathbb{R}^n$, and it is said to be negative semi-definite if

$$x^\top Ax \leq 0$$

An $n \times n$ symmetric real matrix which is neither positive semi-definite nor negative semi-definite is called indefinite.

A matrix A is positive (negative) definite if and only if all of its eigenvalues are > 0 (< 0).

A matrix A is positive (negative) semi-definite if and only if all of its eigenvalues are ≥ 0 (≤ 0).

You can find several interesting properties such as: i) if A and B are positive definite matrices, then the sum $A + B$ is a positive definite matrix; or ii) if A is a positive definite matrix, then the inverse A^{-1} is a positive definite matrix.

Finally, an interesting result is the following: let A be an $m \times n$ matrix. The matrix $S = A^\top A$ is positive definite (and then symmetric), and therefore S has orthonormal eigenvectors and positive eigenvalues.

a) Given the theory, finding the solution using Python is straightforward. We can obtain the results simply by using the `is_positive_definite` function. The output of my code is:

   A is not positive definite.

   B is positive definite.

   C is positive definite.

   D is not positive definite.

b) By extracting the eigenvalues and eigenvectors for our symmetric matrices we can express (S) as: $(S = Q\Lambda Q^\top)$, where Q is an $n \times n$ matrix with columns being the eigenvectors of S, $Q = [q_1, q_2, ..., q_n]$, and Λ is a diagonal matrix with diagonal elements $\Lambda_{i,i} = \lambda_i$.

   Using Python, particularly with the NumPy library, we can perform eigen decomposition. Thanks to this, I obtained the following output:

   Van is not orthogonal

   Lan is not orthogonal

   Vbn is orthogonal

   Lbn is not orthogonal

   Vcn is orthogonal

   Lcn is not orthogonal

   Vdn is orthogonal

   Ldn is not orthogonal

   Where V is the matrix composed of the eigenvectors, and L is the lambda matrix. It is easy to notice that the matrix containing the eigenvalues is correctly not orthogonal.

c) For the general case of a non-unitary vector v, the projection matrix is defined as:

$$P = \frac{vv^\top}{\|v\|_2^2}$$

So the projection of a vector onto the vector v would be given by: $\hat{t} = Pt = \frac{vv^\top t}{\|v\|_2^2}$

In the case that we want to project a vector onto a subspace generated by the matrix A, the projection matrix will be given by:

$$P = A(A^\top A)^{-1}A^\top$$

And the projection of the vector t onto the subspace generated by the matrix A will be given by:

$\hat{t} = A(A^\top A)^{-1}A^\top t$

so in our case we are in the second case so by computing P as in the theory we get: The projection of b on E is:

$$\begin{bmatrix} \frac{667}{649} \\ \frac{866}{649} \\ \frac{757}{649} \\ \frac{3587}{649} \\ \frac{2921}{649} \end{bmatrix}$$

The projection of b on F is:

$$\begin{bmatrix} 1 \\ 2 \\ 1 \\ 5 \end{bmatrix}$$

d) Now I take vectors $b_1^\top = (1, 0)$, $b_2^\top = (0, 1)$, $b_3^\top = (-1, 0)$, and $b_4^\top = (0, -1)$ and plot how these vectors are transformed using matrix D.

As we can see from the image 2.2, simply applying the matrix V results in a rotation of the vector.

on the other hand we can observe from the image 2.3, simply applying the matrix Λ implies a scaling operation along the dimensions of the eigenvalues of the vector.

Here we can observe the full rotation that is applied using V, Λ, and $V^\top$, and that is actually the same result as applying just the diagonal matrix Λ

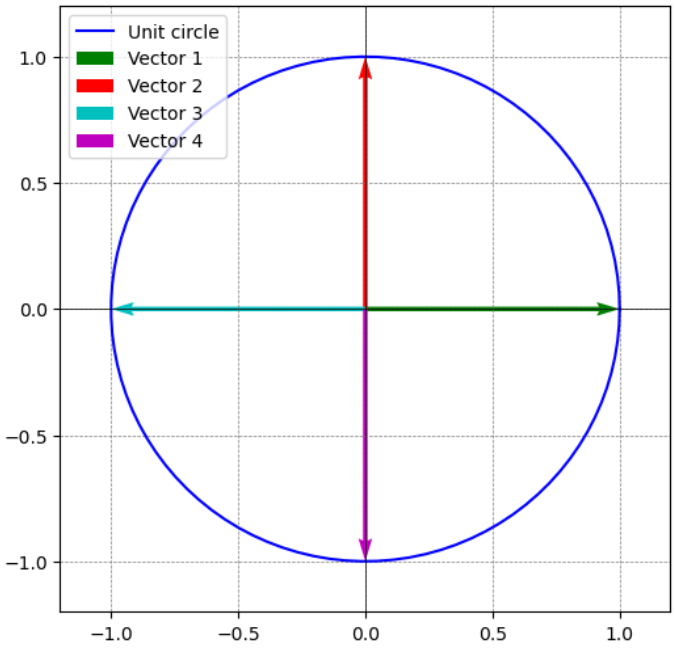

Figure 2.1: Vectors b_i plotted within the unit circle

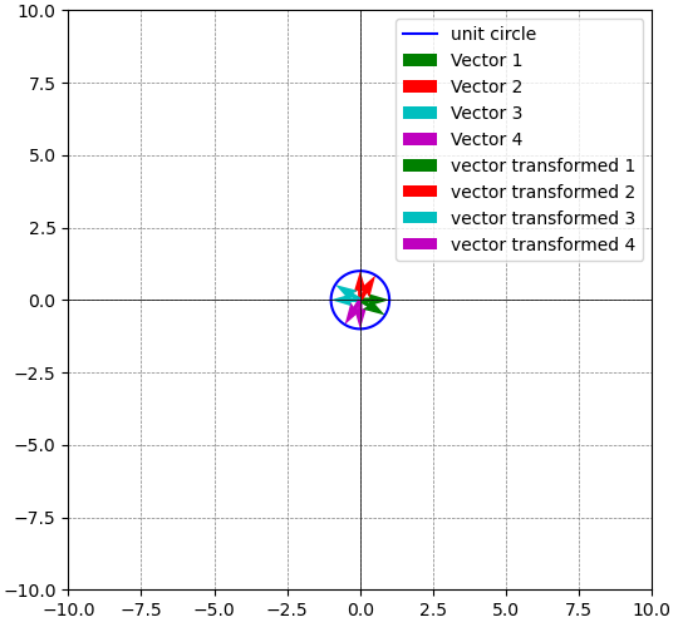

Figure 2.2: Vectors b_i transformed by the matrix V of the eigen decomposition plotted within the unit circle

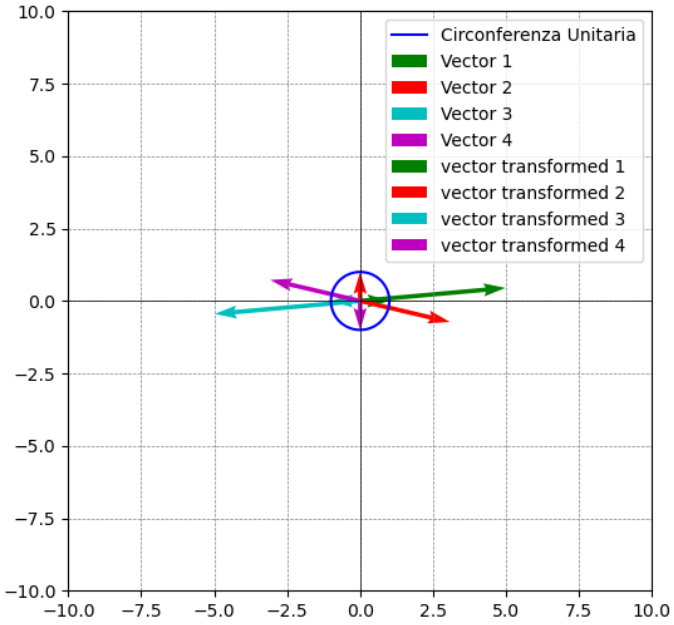

Figure 2.3: Vectors b_i transformed by the matrix V and matrix Λ of the eigen decomposition plotted with the unit circle

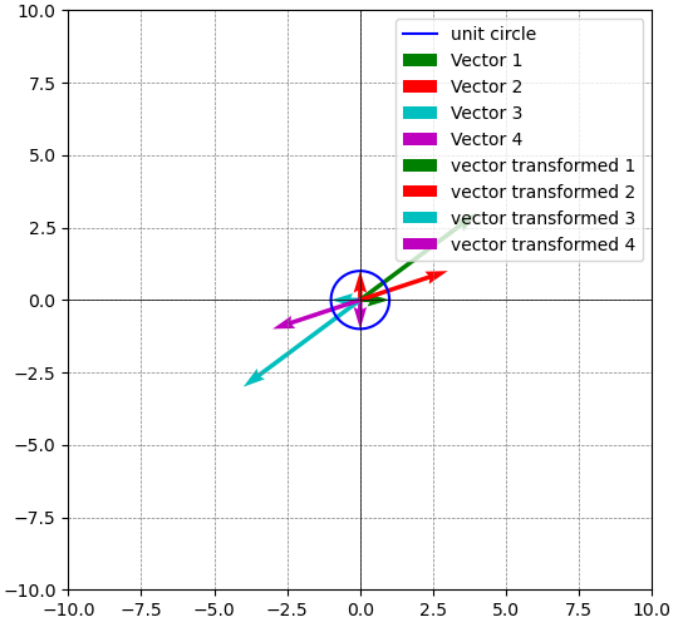

Figure 2.4: Vectors b_i transformed by the D with the unit circle

to the vector. As is possible to see between 2.3 and 2.4 the matrix $V^\top$ is just contributing rotating the vectros

Chapter 3

Exercise 3

3.1 Assignemet

Solve the following short questions:

a) Take matrices A and B from Exercise 2 and obtain:

$$\text{trace}(A), \text{trace}(B), \text{trace}(A+B), \text{trace}(AB), \text{trace}(BA)$$

b) Obtain the Hadamard product of matrices A and B, i.e., $A \odot B$.

c) Write in matricial form the following quadratic function:

$$f(x) = x_1^2 + 2x_2^2 + 4x_3^2 + 2x_1x_2 - 2x_1x_3 + 2x_1 + 2x_2 - x_3 + 3.$$

Check whether the matrix P of the quadratic form, i.e., $x^\top Px$, is positive definite, and thus, this quadratic form is a convex function.

3.2 Solution exercise 3

For a square matrix $n \times n$ A, we define the trace as the sum of the elements of its diagonal:

$$\text{tr}(A) = \sum_{k=1}^{n} a_{kk} = a_{11} + a_{22} + \ldots + a_{nn}$$

a) So, in our case, using SymPy, we can compute the trace. We will observe that some properties of the trace are verified by the numeric Python output, such as the fact that the trace of the sum is the sum of the traces, etc. The output for out specific exercise is:

trace of A is 11

trace of B is 10

trace of A+B is 21

trace of AB is 35

trace of BA is 35

- The Hadamard product (also known as the element-wise product, entry-wise product, or Schur product) returns a matrix of the multiplied corresponding elements. It is defined with the symbol $\odot$ (sometimes also with symbol $\circ$): $(A \odot B)_{ij} = (A \circ B)_{ij} = (A)_{ij}(B)_{ij}$. In our case, once again utilizing the `hadamard_product` function, we obtain our solution:

$$\begin{bmatrix} 20 & 0 & 4 & -1 \\ 0 & 4 & 0 & -3 \\ 6 & 0 & 0 & 0 \\ -2 & -3 & 0 & 10 \end{bmatrix}$$

- A quadratic function has the form of

$$f(x) = \frac{1}{2} x^\top P x + b^\top x + c$$

In this case, to obtain the matrix P, I had to create it manually since there is no automatic way to represent it. The matrix P was that reprenset the funtion

$$f(x) = x_1^2 + 2x_2^2 - 4x_3^2 + 2x_1x_2 - 2x_1x_3 + 2x_1 + 2x_2 - x_3 + 3$$

is:

$$\begin{bmatrix} 2 & 2 & -2 \\ 2 & 4 & 0 \\ -2 & 0 & -8 \end{bmatrix}$$

and using the python code i was able to determine the this matrix in not positive definite. This matrix is actually semi-postive so the function is actually convex.

Chapter 4

Exercise 4

4.1 Assignemet

Let us assume matrices A and B:

$$A = \begin{bmatrix} 4 & 3 & 2 & 1 \\ 3 & 1 & 4 & 2 \end{bmatrix}$$

$$B = \begin{bmatrix} 4 & 3 & 2 \\ 3 & 1 & 5 \\ 1 & 4 & 7 \\ 5 & 2 & 6 \\ 3 & 4 & 2 \end{bmatrix}$$

a) Obtain the Singular Value Decomposition (SVD) of matrices A and B, and check orthogonality of eigenvectors.

b) Write economy, compact, and truncated SVD representations.

c) Obtain the rank-1 and rank-2 best matrix approximations.

d) Write the ℓ_1, ℓ_2, ℓ_∞ norms, and the spectral, Frobenius, and nuclear norms.

e) Obtain the condition number of the matrices.

4.2 Solution exercise 4

a) As we've seen, when S is a symmetric $n \times n$ matrix, we can find a set of orthonormal vectors u_k that serve as both left and right eigenvectors of S, associated with the real eigenvalues λ_k. This means they satisfy the equations:

$$Su_i = \lambda_i u_i, \quad i \in \{1, \ldots, n\}$$

The Singular Value Decomposition (SVD) generalizes these equalities for a general $m \times n$ (i.e., m rows and n columns) matrix $A \in \mathbb{R}^{m \times n}$ of rank $r \leq \min(m, n)$. The concept is to find orthonormal matrices $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$, and a diagonal matrix $\Sigma \in \mathbb{R}^{m \times n}$, such that:

$$A = U \Sigma V^{\top}$$

Now using python I can calulate this matricies and check the orthogonality of the eigenvectors. My output is: U of A

$$\begin{bmatrix} 0.70710678 & -0.70710678 \\ 0.70710678 & 0.70710678 \end{bmatrix}$$

S of A

$$\begin{bmatrix} 7.41619849 & 0. \\ 0. & 2.23606798 \end{bmatrix}$$

VT of A

$$\begin{bmatrix} 0.66742381 & 0.38138504 & 0.57207755 & 0.28603878 \\ -0.31622777 & -0.63245553 & 0.63245553 & 0.31622777 \\ -0.60302269 & 0.60302269 & 0.44120908 & -0.27939546 \\ -0.30151134 & 0.30151134 & -0.27939546 & 0.86030227 \end{bmatrix}$$

U of A is orthogonal VT of A is orthogonal

U of B

$$\begin{bmatrix} -0.34257033 & 0.58307731 & 0.09625668 & -0.38316715 & -0.62175881 \\ -0.4064537 & -0.17291495 & -0.44522011 & -0.66625951 & 0.40345153 \\ -0.53578801 & -0.63122898 & 0.51385337 & 0.01149083 & -0.22428522 \\ -0.56226218 & 0.0866505 & -0.52728265 & 0.62645945 & -0.07664528 \\ -0.33798434 & 0.47345849 & 0.50044123 & 0.12921969 & 0.62806408 \end{bmatrix}$$

S of B

$$\begin{bmatrix} 13.98490244 & 0. & 0. \\ 0. & 4.18231663 & 0. \\ 0. & 0. & 3.30616567 \end{bmatrix}$$

VT of B

$$\begin{bmatrix} -0.49701415 & -0.43287956 & -0.75205866 \\ 0.72590472 & 0.26744412 & -0.63366867 \\ -0.47543588 & 0.86086522 & -0.1813058 \end{bmatrix}$$

U of B is orthogonal VT of B is orthogonal

b) Defining $\Sigma_j = \text{diag}(\sigma_1, \ldots, \sigma_k)$, $U_k = [u_1, \ldots, u_k]$, and $V_k = [v_1, \ldots, v_k]$, we obtain:

$$A = U_k \Sigma_k V_k^\top$$

which is known as the economy SVD. On the other hand, if the rank r of A is $r < \min(m, n)$, then there are only r singular values different from zero, and then:

$$A = U_r \Sigma_r V_r^\top$$

which is known as the compact SVD. This fact is useful when obtaining matrices U and V since only u_r and v_r vectors have to be calculated.

Finally, in low-rank approximation, only the t highest singular values are considered, and:

$$A = U_t \Sigma_t V_t^\top$$

which is known as the truncated SVD.

For Matrix A: Economy-sized U of A:

$$\begin{bmatrix} 0.70710678 & -0.70710678 \\ 0.70710678 & 0.70710678 \end{bmatrix}$$

Economy-sized VT of A:

$$\begin{bmatrix} 0.66742381 & 0.38138504 & 0.57207755 & 0.28603878 \\ -0.31622777 & -0.63245553 & 0.63245553 & 0.31622777 \end{bmatrix}$$

For Matrix B: Economy-sized U of B:

$$\begin{bmatrix} -0.34257033 & 0.58307731 & 0.09625668 \\ -0.4064537 & -0.17291495 & -0.44522011 \\ -0.53578801 & -0.63122898 & 0.51385337 \\ -0.56226218 & 0.0866505 & -0.52728265 \\ -0.33798434 & 0.47345849 & 0.50044123 \end{bmatrix}$$

Economy-sized VT of B:

$$\begin{bmatrix} -0.49701415 & -0.43287956 & -0.75205866 \\ 0.72590472 & 0.26744412 & -0.63366867 \\ -0.47543588 & 0.86086522 & -0.1813058 \end{bmatrix}$$

c) To obtaing the rank-1 and rank-2 approximation of my matricies i gust use the TurncatedSVD in the library Numpy and the result is: Lower approximation of A, rank 1:

$$\begin{bmatrix} 5.24404424 \\ 5.24404424 \end{bmatrix}$$

Lower approximation of A, rank 2:

$$\begin{bmatrix} 5.24404424 & -1.58113883 \\ 5.24404424 & 1.58113883 \end{bmatrix}$$

Lower approximation of B, rank 1:

$$\begin{bmatrix} 4.79081258 \\ 5.68421529 \\ 7.49294299 \\ 7.8631818 \\ 4.72667799 \end{bmatrix}$$

Lower approximation of B, rank 2:

$$\begin{bmatrix} 4.79081258 & -2.43861391 \\ 5.68421529 & 0.72318506 \\ 7.49294299 & 2.63999946 \\ 7.8631818 & -0.36239983 \\ 4.72667799 & -1.98015332 \end{bmatrix}$$

d) Write the ℓ_1, ℓ_2, ℓ_∞ norms, and the spectral, Frobenius, and nuclear norms is very simle using Numpy since i just have to write few lines of code:

```
\begin{lstlisting}[language=Python]
# Calculate the l1, l2, l_inf, Frobenius, and nuclear norm of A
def norms(matrix, name):
    l1 = np.linalg.norm(matrix, ord=1)
    l2 = np.linalg.norm(matrix, ord=2)
    l_inf = np.linalg.norm(matrix, ord=np.inf)
    frobenius = np.linalg.norm(matrix, ord='fro')
    nuclear = np.linalg.norm(matrix, ord='nuc')
    print("l1 norm of Matrix " + name)
    print(l1)
    print("l2 norm of Matrix " + name)
    print(l2)
```

```
    print("l_inf norm of Matrix " + name)
    print(l_inf)
    print("Frobenius norm of Matrix " + name)
    print(frobenius)
    print("Nuclear norm of Matrix " + name)
    print(nuclear)

norms(A, "A")
norms(B, "B")
\end{lstlisting}
```

and the output is: l1 norm of Matrix A 7.0

l2 norm of Matrix A 7.416198487095664

linf norm of Matrix A 10.0

frobenius norm of Matrix A 7.745966692414834

nuclear norm of Matrix A 9.652266464595453

l1 norm of Matrix B 22.0

l2 norm of Matrix B 13.984902437006962

linf norm of Matrix B 13.0

frobenius norm of Matrix B 14.966629547095765

nuclear norm of Matrix B 21.47338473715689

e) We define the condition number of a matrix A as:

$$\kappa(A) = \frac{\|A\|_2}{\|A^{-1}\|_2} = \frac{\sigma_{\max}}{\sigma_{\min}}$$

Where $\sigma_{\max}$ and $\sigma_{\min}$ are the maximum and minimum singular values of matrix A. A large condition number, $\kappa(A)$, results in a highly sensitive system, meaning that small changes in A or b may result in very large changes in the solution x. On the other hand, a large condition number, $\kappa(A)$, implies that $\sigma_{\max} \gg \sigma_{\min}$, and then the matrix A is almost singular, meaning it is not invertible.

So in our case is really simple to compute using python:

```
#calculate the condition number of A
def cond(matrix):
    return  np.linalg.cond(matrix)

print("condition number of A")
print(cond(A))
print("condition number of B")
print(cond(B))
```

and the ouput is :

condition number of A 3.3166247903554003

condition number of B 4.229946052622944

Chapter 5

Exercise 5

5.1 Assignement

a)

$$\binom{n-1}{k-1} + \binom{n-1}{k} = \binom{n}{k}$$

b)

$$\sum_{i=r}^{n} \binom{i}{r} = \binom{n+1}{r+1}, \quad \text{where } n > r.$$

c)

$$\sum_{k=0}^{m} \binom{m}{k}\binom{n}{r-k} = \binom{m+n}{r}$$ (Hint: use $(1+x)^m(1+x)^n = (1+x)^{m+n}$ and equate coefficients)

5.2 Solution

a) To prove the identity

$$\binom{n-1}{k-1} + \binom{n-1}{k} = \binom{n}{k},$$

we can use the definition of binomial coefficients and some algebraic manipulation. Here's a step-by-step proof:

Start with the definition of binomial coefficients:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Now, we'll prove the identity by manipulating the left-hand side of the identity and showing that it equals the right-hand side:

Begin with the right-hand side of the identity:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

While the left-hand side is

$$\begin{aligned}
\binom{n-1}{k-1} + \binom{n-1}{k} &= \frac{(n-1)!}{(k-1)!(n-k)!} + \frac{(n-1)!}{k!(n-k-1)!} \\
&= \frac{(n-1)!}{(k-1)!(n-k)!} + \frac{(n-1)!}{k(n-k)!} \\
&= \frac{k(n-1)!}{k(n-k)!} + \frac{(n-1)!}{k(n-k)!} \\
&= \frac{k(n-1)! + (n-1)!}{k(n-k)!} \\
&= \frac{(n-1)!(k+1)}{k(n-k)!} \\
&= \frac{(n-1)!(n-k)}{k(n-k)!} \\
&= \frac{n!}{k!(n-k)!} = \binom{n}{k}.
\end{aligned}$$

So, by manipulating the factorials and using the definition of binomial coefficients, we have shown that $\binom{n-1}{k-1} + \binom{n-1}{k} = \binom{n}{k}$ which is the first combinatorial identity.

b) **Proof by Induction:**

Base Case: $n = r$

$$\begin{aligned}
\sum_{i=r}^{n} \binom{i}{r} = \sum_{i=r}^{r} \binom{i}{r} = \binom{r}{r} = 1 \\
\binom{n+1}{r+1} = \binom{r+1}{r+1} = 1
\end{aligned}$$

So, this proves that the base case is true.

Induction Hypothesis: Now, we will assume that

$$\sum_{i=r}^{n} \binom{i}{r} = \binom{r+1}{n+1},$$

where $n > r$. is true

Now, we need to prove:

$$\binom{n+1}{r} + \sum_{i=r}^{n} \binom{i}{r} = \binom{n+2}{r+1}$$

By our induction hypothesis, we have:

$$\binom{n+1}{r} + \binom{r+1}{n+1} = \binom{n+2}{r+1}$$

And this is true since we proved it in the point A

c) We can prove this using the bionomial theorem so:

$$(1+x)^m(1+x)^n$$

$$\sum_{k=0}^{\infty} \binom{m}{k} x^k \sum_{j=0}^{\infty} \binom{n}{j} x^j =$$

$$\sum_{k=0}^{\infty} \sum_{j=0}^{\infty} \binom{m}{k} \binom{n}{j} x^{k+j} =$$

$$\sum_{k=0}^{\infty} \sum_{r=k}^{\infty} \binom{m}{k} \binom{n}{r-k} x^r =$$

$$\sum_{r=0}^{\infty} \sum_{k=0}^{r} \binom{m}{k} \binom{n}{r-k} x^r$$

$$(1+x)^{m+n} = \sum_{r=0}^{\infty} (m+n)^r x^r$$

.