362982 (Pedro Carvalho) **CK0146: Reality Check**

Exercise R.1. Let Ω be a finite set of all possible outcomes and $P: \Omega \to \mathbb{R}$ a probability measure that (by definition) satisfies $P(\omega) \geq 0$ and $\sum_{\omega \in \Omega} P(\omega) = 1$. Let $f: \Omega \to \mathbb{R}$ be an arbitrary function o Ω. Define the *expectation* of function f by $E[f] = \sum_{\omega \in \Omega} P(\omega) f(\omega)$. The *variance* of function f is defined by $Var[f] = E[(f - E[f])^2]$.

Use such definitions to show that:

- $E[\cdot]$ is a linear operator.
- $Var[f(\omega)] = E[f(\omega)^2] - E[f(\omega)]^2$

Solution:

- For the first item, we must first define what is a linear operator. We can say that $\overrightarrow{L}$ is a linear operand if for every scalar t and pair of functions f and g, the following statements hold:
  - $\overrightarrow{L}(f+g) = \overrightarrow{L}(f) + \overrightarrow{L}(g)$
  - $\overrightarrow{L}(tf) = t\overrightarrow{L}(f)$.

  Let's show that the first condition hold for $E[f]$. Let f and g be two arbitrary functions with domain on Ω. From the definition of expectation of a function, we have

$$E[f+g] = \sum_{\omega \in \Omega} P(\omega)[(f+g)(\omega)]$$
$$= \sum_{\omega \in \Omega} P(\omega)[f(\omega) + g(\omega)]$$
$$= \sum_{\omega \in \Omega} [P(\omega)f(\omega) + P(\omega)g(\omega)]$$
$$= \sum_{\omega \in \Omega} P(\omega)f(\omega) + \sum_{\omega \in \Omega} P(\omega)g(\omega)$$
$$= E[f] + E[g]$$

  Let's now show that the second condition hold for $E[f]$. Let $t \in \mathbb{R}$ be a scalar and f, a function with domain on Ω. From the definition of $E[f]$, we have

$$E[tf] = \sum_{\omega \in \Omega} P(\omega)[tf(\omega)]$$
$$= \sum_{\omega \in \Omega} tP(\omega)f(\omega)$$
$$= t\sum_{\omega \in \Omega} P(\omega)f(\omega)$$
$$= tE[f]$$

  And with that, we have proved that $E[\cdot]$ is a linear operator. ■

- For the second item, we will use the fact that $E[\cdot]$ is a linear operator in our favour. We know that $Var[f(\omega)] = E[(f(\omega) - E[f(\omega)])^2]$. Expanding the square power, we have

$$Var[f(\omega)] = E[f(\omega)^2 + E[f(\omega)]^2 - 2f(\omega)E[f(\omega)]]$$

  Using the linearity of the $E[\cdot]$ operand, we can separate the expectation operand in three other operands. Note that $E[E[f(x)]] = E[f(x)]$ since the expectation of a real value is the value itself. With that in mind, we have

$$Var[f(\omega)] = E[f(\omega)^2] + E[f(\omega)^2] - 2 * E[f(\omega)]E[f(\omega)]$$

$$= E[f(\omega)^2] + E[f(\omega)]^2 - 2 * E[f(\omega)]^2$$

$$= E[f(\omega)^2] - E[f(\omega)^2]$$

■

Exercise R.2. Let $\mathbf{A} \in \mathbb{R}^{N\times N}$ be a symmetric matrix with distinct eigenvalues $\lambda_i \in \mathbb{R}$ ($i = 1, 2, \ldots, N$), defined by $\mathbf{A}\mathbf{v}_i = \lambda_i \mathbf{v}_i$ and eigenvectors that satisfy $\mathbf{v}_i^T \mathbf{v}_j = \delta_{ij}$ where the Kronecker delta is $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ otherwise. Let the spectral decomposition

$$\mathbf{B} = \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \mathbf{v}_i^T \mathbf{v}_j$$

Show that $\mathbf{B}$ has the same eigenvectors and eigenvalues as $\mathbf{A}$ (Showing that $\mathbf{A} = \mathbf{B}$ is not needed).

Solution: We begin this proof by analysing the law of formation of $\mathbf{B}$. It is multiplying an eigenvector v_i by its correspondent eigenvalue λ_i and another eigenvector v_j. With a little of imagination, we can be rewrite $\mathbf{B}$ in matrix form as

$$\mathbf{B} = \mathbf{Q}\Lambda\mathbf{Q}^T,$$

where $\mathbf{Q}$ is a $N \times N$ matrix such that the i_{th} column is v_i and Λ is a diagonal $N \times N$ matrix where where the element in position ii is equal to λ_i.

Now let's take a closer look at $\mathbf{Q}$. From the information given, the product between two distinct eigenvectors of $\mathbf{A}$ is equal to 0, and the product between two equal eigenvectors is 1. That is enough to show that $\mathbf{Q}$ is an orthogonal matrix. Then, from the properties of orthogonal matrices, we also know that $Q^T = Q^{-1}$.

Let v_i be an eigenvector of $\mathbf{A}$. We will show that it is also an eigenvector of $\mathbf{B}$ and that its correspondent eigenvalue will be λ_i. Let's then multiply $\mathbf{B}$ by v_i:

$$\mathbf{B}v_i = \mathbf{Q}\Lambda\mathbf{Q}^T v_i = \mathbf{Q}\Lambda(\mathbf{Q}^T v_i) = \mathbf{Q}\Lambda v_{prod}$$

where v_{prod} is a n-dimensional vector with 1 in the i_{th} position and 0 elsewhere. Multiplying Λ by v_{prod}, we will get a vector v_{λ_i}, with only λ_i in the i_{th} spot and 0 elsewhere. Then, we stay with

$$\mathbf{B}v_i = \mathbf{Q}v_{\lambda_i} = v_i\lambda_i.$$

That concludes our proof, since we know that every eigenvector of $\mathbf{A}$ is also an eigenvector of $\mathbf{B}$ and the correspondent eigenvalues are the same. ■

Exercise R.3. Fibonacci numbers $F(i)$ are defined for $i \in \mathbb{N}$ recursively as $F(i+2) = F(i+1) + F(i)$, with $F(1) = F(2) = 1$. Using pseudo-code, write down an algorithm that takes $n \in \mathbb{N}$ as input and outputs the Fibonacci numbers from $F(1)$ to $F(n)$. Analyse the time complexity of your algorithm using $\mathbb{O}$ notation. Briefly discuss the efficiency of your algorithm.

Solution: Let's begin by presenting an algorithm for the Fibonacci numbers problem.

```
f := list(n);
f[1] := 1;
if n := 1 then $return$ 1 end;
f[2] := 1;
if n := 2 then $return\ f$; end;
for i := 3 to n do
    $f[i] := f[i-1] + f[i]$;
end;
$return\ f$
```

This algorithm works by creating a list that will store the Fibonacci numbers. It first checks to see if n is equal to 1 or 2, which are the base cases and do not require any calculation. For the other numbers (from 3 to n), we run through the positions of the list and calculate the values using the calculations from past iterations. At the end of the process, our list is populated with the Fibonacci numbers from 1 to n.

Let's have a look at the time complexity of our algorithm. The first five instructions are all $\mathbb{O}(1)$, because they are only stand-alone attributions and comparisons. The for loop runs for $n-2$ steps (from 3 to n), which is $\mathbb{O}(n)$. Then, combining it all, the algorithm is $\mathbb{O}(n)$. That is actually a pretty good approach, given that there are recursive algorithms that are exponential or factorial. Implementing it in python and checking the execution time for $n = 1$ and $n = 10000$