Technical University of Denmark

Course name	Introduction to programming and data processing
Course number	02631, 02632, 02633, 02634, 02692
Aids allowed	All Aids
Exam duration	2 hours
Weighting	All exercises have equal weight

Contents

Submission details

You must hand in your solution electronically:

1. You can test your solutions individually on CodeJudge

   https://dtu.codejudge.net/

   under *Assignments* and *Exam*. When you hand in a solution on CodeJudge, the test example given in the assignment description will be run on your solution. If your solution passes this single test, it will appear as *Submitted*. This means that your solution passes on this single test example. You can upload to CodeJudge as many times as you like during the exam.

2. You must upload all your solutions at Eksamen site. Each assignment must be uploaded as one separate .py file, given the same name as the function in the assignment:

   (a) `fit_windows.py`
   (b) `count_novel.py`
   (c) `count_peaks.py`
   (d) `classify_flower.py`
   (e) `qualify.py`

   The files must be handed in separately (*not* as a zip-file) and must have these exact filenames.

After the exam, your solutions submitted to DTU Inside will be automatically evaluated on CodeJudge on a range of different tests, to check that they work correctly in general. The assessment of your solution is based only on how many of the automated tests it passes.

- Make sure that your code follows the specifications exactly.
- Each solution shall not contain any additional code beyond the specified function, though `import` statements can be included.
- Remember, you can check if your solutions follow the specifications by uploading them to CodeJudge.
- Note that all vectors and matrices used as input or output must be numpy arrays.

Assignment A Fit Windows

We are given a building with rectangular sides that should be fitted with as many rectangular windows as possible on either all the four sides or two of the sides (not the top and the bottom of the building). The building is specified by length, width and height and the windows are specified by width and height. The windows cannot be rotated. They can be fitted as close as possible. The number of windows must be an integer number, so the total sum of the window dimensions is less than or equal to the building dimensions.

An additional parameter determines whether the windows should be fitted at the two "length" sides, the two "width" sides, or with the value "all" on all four sides.

The total number of windows, c, for all the four sides is (when the "side" parameter is set to "all"):

$$c = 2 \times c_{length} \times c_{height} + 2 \times c_{width} \times c_{height}.$$

Problem definition

Create a function called `fit_windows` which as input takes a 3-element vector **b** with length, width and height for the building and a 2-element vector **w** with width and height of the windows. A "side" parameter with the value of either "length", "width" or "all" should determine which sides of the building the windows should be fitted to. The function should return the number of windows that can be fitted to the building.

Solution template

```
def fit_windows(b, w, side):
  #insert your code
  return c
```

Input	
`b`	3-element vector with decimal numbers specifying length, width and height.
`w`	2-element vector with decimal numbers specifying width and height of the windows.
`side`	String, that is either "length" (indicating windows on the two "length" sides), "width" (indicating windows on the two "width" sides) or "all" (indicating windows on all four sides).

Output	
`c`	Integer with the number of windows that can be fitted.

Example

Consider the building vector $\mathbf{b} = [37, 20, 10]$, the window vector $\mathbf{w} = [1.1, 1.75]$ and the "side" parameter to have the value "length". The number of windows that can be fitted in the height is

$$c_{height} = 5 \qquad \text{as } 5 \times 1.75 = 8.75 \leq 10.$$

Windows fitted on the "length" side:

$$c_{length} = 33 \qquad \text{as } 33 \times 1.1 = 36.3 \leq 37$$

Windows that could be fitted on the "width" side (but we ignore this as "side" is set to "length"):

$$c_{width} = 18 \qquad \text{as } 18 \times 1.1 = 19.8 \leq 20$$

The total number windows are:

$$c = 2 \times c_{length} \times c_{height} = 2 \times 33 \times 5 = 330 \tag{1}$$

Thus the value 330 is returned.

A

Assignment B Count novel words

Given a text represented in a string with words separated with spaces, we want to return a vector with the count of novel words seen so far in the text for each word.

Problem definition

Create a function called `count_novel` which takes a string as input and returns a vector with the count of novel words seen so far in the text for each word.

Solution template

```
def count_novel(text):
  # insert your code
  return c
```

Input

`text`	String with a text with one or more words. Words consist of lowercase letters from a to z and they are separated with one space character.

Output

`c`	Vector with the same length as the number of words in the string and with an integer number with the count of novel words seen so far in each element.

Example

Regard the text

"the man and another man walked down the street"

The novel words in order are:

the, man, and, another, (already seen), walked, down, (already seen), street

The first word "the" is a novel word and counts as one. The second word is "man" and also novel so there are two novel words so far. When we come to the fifth word, which is "man", this word is not novel and thus for the text until the fifth word there are 4 novel words. The full result is:

$$c = [1, 2, 3, 4, 4, 5, 6, 6, 7] \tag{2}$$

B

Assignment C Count peaks

Given a matrix, we want to count the number of peaks, defined as elements where the difference between the element itself and at least one of its neighbors is larger than or equal to 2. The neighborhood for a matrix element is defined as the four matrix elements immediately above, below, to the right and to the left (For border elements, there is only 2 or 3 neighboring elements).

Problem definition

Create a function called `count_peaks` that takes matrix as input and returns the number of peaks where the difference between the element and one or more of its neighbors is larger than or equal to 2.

Solution template

```
def count_peaks(A):
  # insert your code
  return c
```

Input	
`A`	Matrix with one or more columns and one or more rows an decimal elements.

Output	
`c`	Non-negative integer with count.

Example

Consider the matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 & 5 & 3 & 7 & 7.1 \\ 3 & 3 & 3 & 4 & 4 & 7 & 7.3 \end{bmatrix}. \tag{3}$$

Here the element $a_{2,1} = 3$ has a neighbor, $a_{1,1}$, where the difference is ≥ 2: $a_{2,1} - a_{1,1} = 3 - 1 = 2$. The element $a_{1,4} = 5$ has two neighboring elements where the difference is larger than or equal to two, e.g., $a_{1,4} - a_{1,3} = 3 - 1 = 2$. Furthermore, the element $a_{1,6}$ has a neighbor sufficiently smaller: $a_{1,6} - a_{1,5} = 7 - 3 = 4$. Lastly, the element $a_{2,6}$ also has a small neighbor: $a_{2,6} - a_{2,5} = 7 - 4 = 3$.

In total, the following elements are peaks:

$$\text{peaks} = \{a_{2,1}, a_{1,4}, a_{1,6}, a_{2,6}\} \tag{4}$$

The total number of elements in this set is $\underline{4}$ which is the value returned.

C

Assignment D Classify flower

We want to categorize a flower with respect to a database represented in a matrix $\mathbf{D}$ of size $(N \times F)$, where N is the number of different flower species and F is the number of features. Except for the last column in the matrix, the features are encoded as non-negative integers. The last column of the matrix is special and encodes the typical height of the flower as a decimal number.

The query flower is represented with a vector of length F, where the first $F-1$ elements are either non-negative integers or have a value zero which indicates undetermined elements (features). The last element contains the height as a decimal number.

We want to know which rows in the database match the query vector, ignoring the elements (features) in the query vector that is undetermined and indicated with zero. Furthermore, we want to sort the matching rows (flowers) in the database according to height, so row indexes for the species closest in height are returned as the first element in the vector.

Problem definition

Create a function called `classify_flower` that takes as input a matrix $\mathbf{D}$, a query vector $\mathbf{q}$ and returns a vector, $\mathbf{s}$, with row indexes (indexed from one) for the species represented in rows in $\mathbf{D}$ that matches $\mathbf{q}$, ignoring elements in the query flower that is zero. The row indices in $\mathbf{s}$ should be sorted in ascending order based on height distance between the query and the database flowers.

Solution template

```
def classify_flower(D, q):
    #insert your code
    return s
```

Input	
`D`	Matrix with size $(N \times F)$ where $N \geq 1$ and $F \geq 2$
`q`	F-sized vector representing the query flower where zero indicates unknown features.

Output	
`s`	Vector with row indices for the species that matches the query flower and sorted according to how close they are in height to the query vector. Indexed from one.

Example

As an example, consider a database representing 4 flowers and their 5 features:

$$\mathbf{D} = \begin{bmatrix} 4 & 1 & 2 & 3 & 10.1 \\ 4 & 1 & 2 & 2 & 20.5 \\ 4 & 2 & 3 & 3 & 25.7 \\ 5 & 2 & 3 & 1 & 19.4 \end{bmatrix} \tag{5}$$

For an example query vector $\mathbf{q}$ features #2 and #3 are unknown (and indicated with a zero) and the height of the flower is 21.1:

$$\mathbf{q} = [4, 0, 0, 3, 21.1] \tag{6}$$

Here the first feature ($q_1 = 4$) matches row (species) 1, 2 and 3, while the fourth feature ($q_4 = 3$) matches row 1 and 3, so only row 1 and 3 matches all known features. When we compare the heights, the differences are:

$$\Delta_1 = |d_{1,5} - q_5| = |10.1 - 21.1| = 10, \qquad \text{and} \qquad \Delta_3 = |d_{3,5} - q_5| = |25.7 - 21.1| = 4.6. \tag{7}$$

Here the species in the third row is closest, so we sort the results as $\mathbf{s} = [3, 1]$, i.e., we return $\underline{[3, 1]}$.

D

Assignment E Football qualification

Given two matrices, $\mathbf{A}$ and $\mathbf{R}$, encoding team allocation and number of goals in a group tournament, we want to determine which two teams qualify for advancement in the tournament. As an example, consider UEFA Euro 1992 group 1 with Sweden, France, Denmark and England, that is encoding as 1, 2, 3 and 4, respectively. In the first game Sweden played against France with the result 1 – 1. The next game was Denmark against England with the result 0 – 0. The fourth game was Sweden against Denmark with the result 1 – 0. With the full set of results, the $\mathbf{A}$ and $\mathbf{R}$ matrices are:

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 2 & 4 \\ 1 & 3 \\ 1 & 4 \\ 2 & 3 \end{bmatrix} \qquad \mathbf{R} = \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 2 & 1 \\ 1 & 2 \end{bmatrix} \tag{8}$$

For each game, the winning team receives two points and the loosing team zero points. A tie will result in one point to each of the teams.

In the above example, Sweden (1) won two games and had one tie, resulting in 5 points. France (2) had two ties and one loss resulting in 2 points. Denmark (3) had one tie, one loss and one winning resulting in 3 points, while England had two ties and one loss resulting in 2 point. Sweden and Denmark is the two top teams and qualify with Sweden first.

Problem definition

Create a function called `qualify` which takes a matrix, $\mathbf{A}$ of size $(G \times 2)$, with team allocation for G games and a matrix, $\mathbf{R}(G \times 2)$, indicating the number of goals. The function should return the team identifiers of the two teams that qualify, and sorted so the team with the most points is first. It can be assumed that winning teams are always separated by points (so it is not necessary to look at the differences in goals as in real life tournaments). The teams are encoded from 1 and in steps of one up to the number of teams. Not all teams combinations might be played.

Solution template

```
def qualify(A, R):
  # insert your code
  return t
```

Input

`A`	$(G \times 2)$-matrix where each row contains identifiers for teams playing a specific game, where $G \geq 1$.
`R`	$(G \times 2)$-matrix with goals scored in each game, where $G \geq 1$.

Output

`t`	2-element vector with identifiers for teams that qualifies to advance in the tournament.

Example

In the above UEFA Euro 1992 group 1 example, the identifiers for Sweden and Denmark should be returned with Sweden first in the vector, $t = [1, 3]$.

E