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/prog-f20/assignments

   under *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 DTU's Online exam site. Each assignment must be uploaded as one separate .py file, given the same name as the function in the assignment:

   (a) `sequence_cut.py`
   (b) `geometry.py`
   (c) `count_words.py`
   (d) `maximum_pair.py`
   (e) `peak.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 Sequence cut

We want to cut a sequence of numbers in two parts. The sequence should be cut at the point where the sum of the first part of the sequence is closest to half of the sum of the full sequence. Closeness is measured as the smallest absolut value.

Problem definition

Create a function called `sequence_cut` which takes as input: $\mathbf{x}$ which is a vector representing the sequence of numbers. The function should return a new sequence, $\mathbf{y}$, that starts with the first element of $\mathbf{x}$ and takes elements from $\mathbf{x}$ in order so the sum of $\mathbf{y}$ is closest to the half of the sum of $\mathbf{x}$. If two cuts are equally close the longest $\mathbf{y}$ should be chosen.

Solution template

```
def sequence_cut(x):
  #insert your code
  return y
```

Input	
`x`	Vector with one of more numbers.

Output	
`y`	Vector with one of more numbers.

Example

Consider the sequence

$$\mathbf{x} = [2, 3, 1, 1, 4, 5, 23, 6, 3, 40]$$

The sum is 88 and half of that is 44. We consider the accumulated sum of the sequence

$$\begin{aligned}\tilde{\mathbf{x}} &= [x_1,\ x_1 + x_2,\ x_1 + x_2 + x_3,\ \ldots] \\ &= [2, 5, 6, 7, 11, 16, 39, 45, 48, 88]\end{aligned}$$

Here is 45 the value closest to half the sum (44). The sum at 45 corresponds to a cut between the 8th and the 9th element in the sequence $\mathbf{x}$. Thus $\mathbf{y}$ is returned as:

$$\mathbf{y} = \underline{[2, 3, 1, 1, 4, 5, 23, 6]}$$

A

Assignment B Geometry selection

We want to choose the largest two-dimensional geometric form in terms of area among four given forms. The areas are computed as:

$$
\begin{aligned}
A_1 &= ab && \text{Rectangle} \\
A_2 &= \pi r^2 && \text{Circle} \\
A_3 &= \frac{1}{2} n r^2 \sin\frac{2\pi}{n} && \text{Inscribed (regular polygon)} \\
A_4 &= n r^2 \tan\frac{\pi}{n} && \text{Circumscribed (regular polygon)}
\end{aligned}
$$

Problem definition

Create a function called `geometry` which takes four inputs: parameters of a rectangle, circle, polygon inscribed in a circle and a polygon circumscribed around a circle. The function should return the area of the largest geometric form.

Solution template

```
def geometry(rectangle, circle, inscribed, circumscribed):
  # insert your code
  return area
```

Input	
`rectangle`	2-element vector with $[a, b]$ for rectangle.
`circle`	Scalar with radius r for circle.
`inscribed`	2-element vector with $[n, r]$ for n sides and radius r for the inscribed polygon.
`circumscribed`	2-element vector with $[n, r]$ for n sides and radius r for the circumscribed polygon

Output	
`area`	Area of the geometric form with the largest area.

Example

Regard a rectangle $a = 3$ and $b = 5$, a circle $r = 2$, an inscribed polygon $n = 4$ and $r = 2.5$, and an circumscribed polygon $n = 6$ and $r = 2.2$

$$
\begin{aligned}
A_1 &= ab = 3 \cdot 5 = 15 && \text{Rectangle} \\
A_2 &= \pi r^2 = \pi 2^2 = 4\pi = 12.57 && \text{Circle} \\
A_3 &= \frac{1}{2} n r^2 \sin\frac{2\pi}{n} = \frac{1}{2} 4 \cdot 2.5^2 \sin\frac{2\pi}{4} = 12.5 \cdot 1 = 12.5 && \text{Inscribed} \\
A_4 &= n r^2 \tan\frac{\pi}{n} = 6 \cdot 2.2^2 \tan\frac{\pi}{6} = 29.04 \cdot 0.577 = 16.77 && \text{Circumscribed}
\end{aligned}
$$

Here the polygon circumscribed in a circle has the largest area and 16.77, or more precisely 16.7662518, and this is returned as the area.

B

Assignment C Count frequent word

Given a text with words separated by a space, we want to count the number of times the most frequent word occurs. Words starting with `no` must be ignored.

Problem definition

Create a function called `count_words` that takes a text as a string input. The function should return the count of the most frequent word. Words starting with `no` should be ignored. String with zero words, i.e., empty strings, should return the number zero. It can be assumed that the text consists of lowercase letters from `a` to `z` and the only other character is the space character.

Solution template

```
def count_words(text):
  # insert your code
  return count
```

Input	
`text`	String with words separated by one space.

Output	
`count`	Scalar with the number of times of the most frequent word, ignoring words starting with `no`

Example

Consider the text

`none none is here but none is coming here again`

Here `none` occurs 3 times but since it is starting with `no` it is ignored. The second most frequent words are `here` and `is` which both occur 2 times. So the number 2 is returned.

C

Assignment D Maximum pair

Given a matrix we want to find the two neighboring numbers that has the heighest sum. Neighboring numbers are defined as those elements that are immediate neighbors in vertical or horizontal directions. The set of neighbors of the matrix element, x_{ij} is

$$\mathcal{N}(x_{i,j}) = \{x_{i-1,j}, x_{i+1,j}, x_{i,j-1}, x_{i,j+1}\}$$

Problem definition

Create a function `maximum_pair` which takes a matrix as input and returns as output the pair of neighbors which sum is the maximum. The pair must be returned in order so the element with the lowest index is first. You may assume that the pair of neighbors with the highest sum is unique.

Solution template

```
def maximum_pair(x):
    #insert your code
    return pair
```

Input	
`x`	2D array representing a matrix containing at least two elements.

Output	
`pair`	1D array with two elements representing the pair.

Example

Consider the matrix

$$\mathbf{X} = \begin{bmatrix} 2 & 2 & 29 & -4 \\ 1 & -6 & 7 & 20 \\ 9 & -37 & 8 & 15 \end{bmatrix}$$

Of some of the possible pairs are $2+29 = 31$ from the first row, $7+20 = 27$ from the second row, $29+7 = 36$ from the third column and $20+15 = 35$ from the fourth column. Considering all possible pairs, 36 is the maximum value so $\underline{[29, 7]}$ is returned.

D

Assignment E Convolution

Given a signal represented by the vector $\mathbf{x}$ and a filter represented by the vector $\mathbf{h}$ with M elements, we want to find the peak amplitude of the *convolution*, where the peak amplitude is the maximum of the absolute value of the convolution. The convolution is given as:

$$y[n] = \sum_{m=0}^{M-1} h[m]\, x[n-m]$$

where the indices m and n start at zero, and where

$$x[n] = 0 \qquad \text{if } n < 0$$

and where $\mathbf{y}$ has the same length as $\mathbf{x}$.

Problem definition

Create a function called `peak` which takes a signal vector $\mathbf{x}$ and a filter vector $\mathbf{h}$, computes the convolution $\mathbf{y}$ with the same length as $\mathbf{x}$ and returns the maximum of the absolute values of the convolution.

Solution template

```
def peak(x, h):
  # insert your code
  return p
```

Input	
`x`	Vector with the signal $\mathbf{x}$.
`h`	Vector with the filter $\mathbf{h}$.

Output	
`p`	Peak amplitude as the maximum of the absolute values of the convolution.

Example

Consider the signal and the filter given by

$$\mathbf{x} = [1, 2, -1, -3, -4] \qquad \mathbf{h} = [0, 1, 3]$$

The convolution, $\mathbf{y}$, (when indexed from zero) is

$$y[0] = \sum_{m=0}^{M-1} h[m]\, x[0-m] = h_0x_0 + h_1x_{(-1)} + h_2x_{(-2)} = 0\cdot 1 + 1\cdot 0 + 3\cdot 0 = 0$$
$$y[1] = \sum_{m=0}^{M-1} h[m]\, x[1-m] = h_0x_1 + h_1x_0 + h_2x_{(-1)} = 0\cdot 2 + 1\cdot 1 + 3\cdot 0 = 1$$
$$y[2] = \sum_{m=0}^{M-1} h[m]\, x[2-m] = h_0x_2 + h_1x_1 + h_2x_0 = 0\cdot (-1) + 1\cdot 2 + 3\cdot 1 = 5$$
$$y[3] = \sum_{m=0}^{M-1} h[m]\, x[3-m] = h_0x_3 + h_1x_2 + h_2x_1 = 0\cdot (-3) + 1\cdot (-1) + 3\cdot 2 = 5$$
$$y[4] = \sum_{m=0}^{M-1} h[m]\, x[4-m] = h_0x_4 + h_1x_3 + h_2x_2 = 0\cdot (-4) + 1\cdot (-3) + 3\cdot (-1) = -6$$

Which gives the full convolution

$$\mathbf{y} = [0, 1, 5, 5, -6]$$

Here the peak amplitude $|-6| = 6$, so $p = \underline{6}$ is returned.

E