Import relevant packages here.

In [161…
```
import matplotlib.pyplot as plt
import numpy
import pandas as pd
import math
```

Load the data and verify it is loaded correctly.

- Print it (head, tail, or specific rows, choose a sensible number of rows).
- Compare it to the source file.

In [215…
```
file_path = 'cf_data.csv'
data = pd.read_csv(file_path)
data.head()  # print the first 5 lines of the data
```

Out[215]:

	dv	s	a
0	-0.743240	53.5427	1.242570
1	-0.557230	53.6120	1.777920
2	-0.454769	53.6541	0.544107
3	-0.525396	53.7030	-0.294755
4	-0.601285	53.7592	-0.290961

In the ensuing, you will use `numpy`.

Let's create a grid for the values to plot. But first create **two arrays named `dv` and `s`** using `numpy.linspace` that hold the grid values at the relevant indices in their respective dimension of the grid.

Create a **grid named `a`** with zeros using `numpy.zeros` in to which calculated acceleration values can be stored.

Let the grid span:

- Speed difference `dv` [m/s]
  - From -10 till 10
  - With 41 evenly spaced values
- Headway `s` [m]
  - From 0 till 200
  - With 21 evenly spaced values

In [210…
```
dv = numpy.linspace(-10, 10, 41)
s = numpy.linspace(0, 200, 21)
a = numpy.zeros((21, 41))  # Makes a 21 by 41 array and intializes all elements of this array to zero.
```

Create from the imported data 3 separate `numpy` arrays for each column `dv`, `s` and `a`. (We do this for speed reasons later.)

- Make sure to name them differently from the arrays that belong to the grid as above.
- You can access the data of each column in a `DataFrame` using `data.xxx` where `xxx` is the column name (not as a string).
- Use the method `to_numpy()` to convert a column to a `numpy` array.

In [216…
```
DV = data.dv.to_numpy()
S = data.s.to_numpy()
A = data.a.to_numpy()
```

Create an algorithm that calculates all the acceleration values and stores them in the grid. The algorithm is described visually in the last part of the lecture. At each grid point, it calculates a weighted mean of all measurements. The weights are given by an exponential function, based on the 'distance' between the grid point, and the measurement values of `dv` and `s`. To get you started, how many `for`-loops do you need?

For this you will need `math`.
Use an *upsilon* of 1.5m/s and a *sigma* of 30m.

Warning: This calculation may take some time. So:

- Print a line for each iteration of the outer-most `for`-loop that shows you the progress.
- Test you code by running it only on the first 50 measurements of the data.

In [220…
```
UPSILON = 1.5  #m/s
SIGMA = 30  # m


def smoothing_filter():
    #This nested for loop loops through all places in the s-dv grid.
    for s_grid in s:
        for dv_grid in dv:
            omega_sum = 0
            omega_A_multiplication_sum = 0
            for dv_value, s_value, a_value in zip(DV, S, A):
            # In this for loop, the value of omega is calculated at a specific point in the grid,
            # with all different values in DV, S and A.

                omega = math.exp(
                    -(abs(dv_value-dv_grid))/UPSILON
                    -(abs(s_value-s_grid))/SIGMA
                )

                omega_A_multiplication_sum += (omega * a_value)  # omega*a_value is added to omega_A_multiplication_sum
                omega_sum += omega  #omega is added to omega_sum

            weighted_sum = omega_A_multiplication_sum / omega_sum  #calculates the weighted sum at a specific point on the grid

            a[int(s_grid/10), int((dv_grid)*2 + 20)] = weighted_sum
            # The weighted mean is calculated and placed added to a.
            #
            # int(s_grid/10) and int((dv_grid)*2 + 20) are used
            # to calculate the correct index for a,
            # as the index of an array starts at 0 and goes with steps of 1.

        print(s_grid)  # This line prints the values of s_grid
        # When this prints 200, the script has finished running
    return

smoothing_filter()
```

```
0.0
10.0
20.0
30.0
40.0
50.0
60.0
70.0
80.0
90.0
100.0
110.0
120.0
130.0
140.0
150.0
160.0
170.0
180.0
190.0
200.0
```

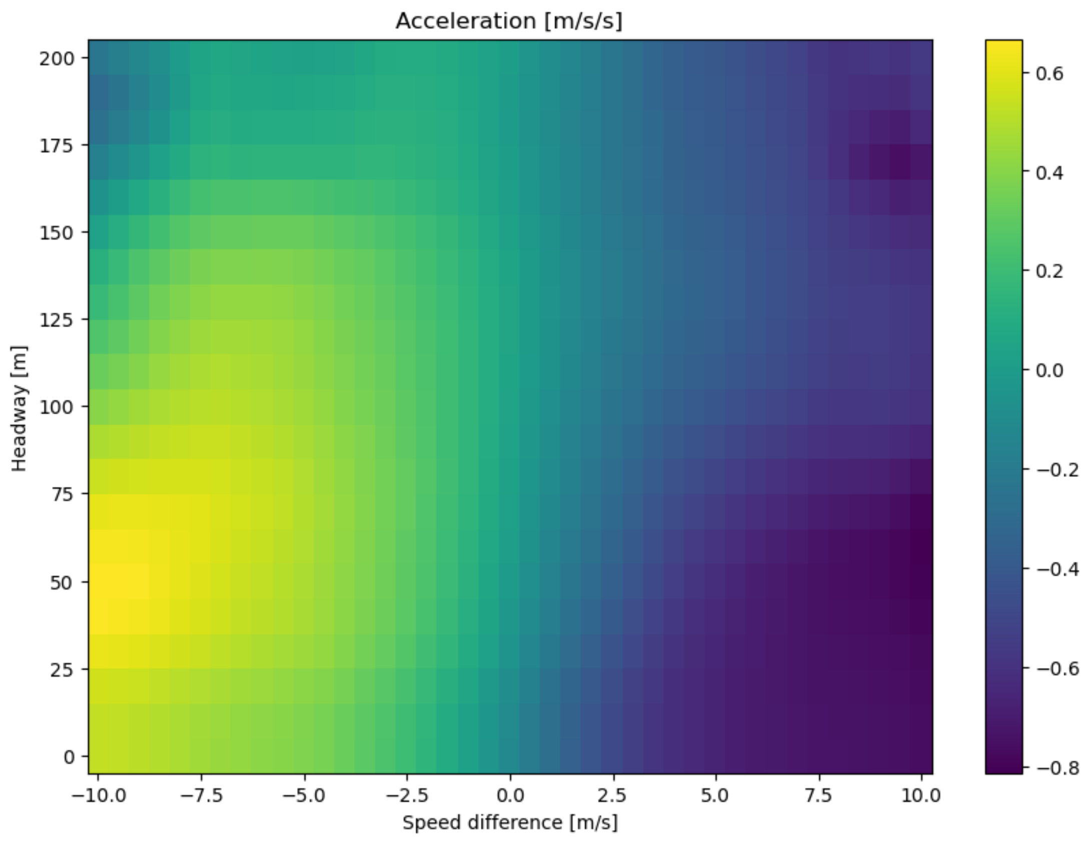
The following code will plot the data for you. Does it make sense when considering:

- Negative (slower than leader) and positive (faster than leader) speed differences?
- Small and large headways?

In [221…

```
X, Y = numpy.meshgrid(dv, s)
axs = plt.axes()
p = axs.pcolor(X, Y, a, shading='nearest')
axs.set_title('Acceleration [m/s/s]')
axs.set_xlabel('Speed difference [m/s]')
axs.set_ylabel('Headway [m]')
axs.figure.colorbar(p);
axs.figure.set_size_inches(10, 7)

# Negative speed differences give a positive value for acceleration
# positive speed differences give a negative value for acceleration
#Small headways gives a high value for acceleration (positive or negative)
#large headways gives a smaller value for acceleration
```


Acceleration [m/s/s]
Headway [m]
Speed difference [m/s]
200
175
150
125
100
75
50
25
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8

```
In [ ]:
```

```
In [ ]:
```