Import relevant packages here.

In [1]:
```python
import matplotlib.pyplot as plt
import random
import pandas as pd
import numpy as np
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 [2]:
```python
data = pd.read_csv('cf_data.csv')
print(data.head())
print(data.tail())
```

```
         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
             dv        s         a
73903  5.19874  116.139 -0.795081
73904  5.10428  115.627 -0.314263
73905  5.13764  115.118  0.232283
73906  5.15348  114.599  0.262078
73907  5.25868  113.112 -0.612440
```

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 [3]:
```python
dv = np.linspace(-10, 10, 41)
s = np.linspace(0, 200, 21)
a = np.zeros((41, 21))

print("dv:", dv)
print("s:", s)
print("Grid a (first 5 rows):\n", a[:5, :])
```

```
dv: [-10.   -9.5  -9.   -8.5  -8.   -7.5  -7.   -6.5  -6.   -5.5  -5.   -4.5
  -4.   -3.5  -3.   -2.5  -2.   -1.5  -1.   -0.5   0.    0.5   1.    1.5
   2.    2.5   3.    3.5   4.    4.5   5.    5.5   6.    6.5   7.    7.5
   8.    8.5   9.    9.5  10. ]
s: [  0.  10.  20.  30.  40.  50.  60.  70.  80.  90. 100. 110. 120. 130.
 140. 150. 160. 170. 180. 190. 200.]
Grid a (first 5 rows):
 [[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

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 [11]:
```python
dv_data = data.dv.to_numpy()
s_data = data.s.to_numpy()
a_data = data.a.to_numpy()

print("dv:", dv_data)
print("s:", s_data)
print("a (shape):", a_data.shape)
```

```
dv: [-0.74324  -0.55723  -0.454769 ...  5.13764   5.15348   5.25868 ]
s: [ 53.5427  53.612   53.6541 ... 115.118  114.599  113.112 ]
a (shape): (73908,)
```

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.

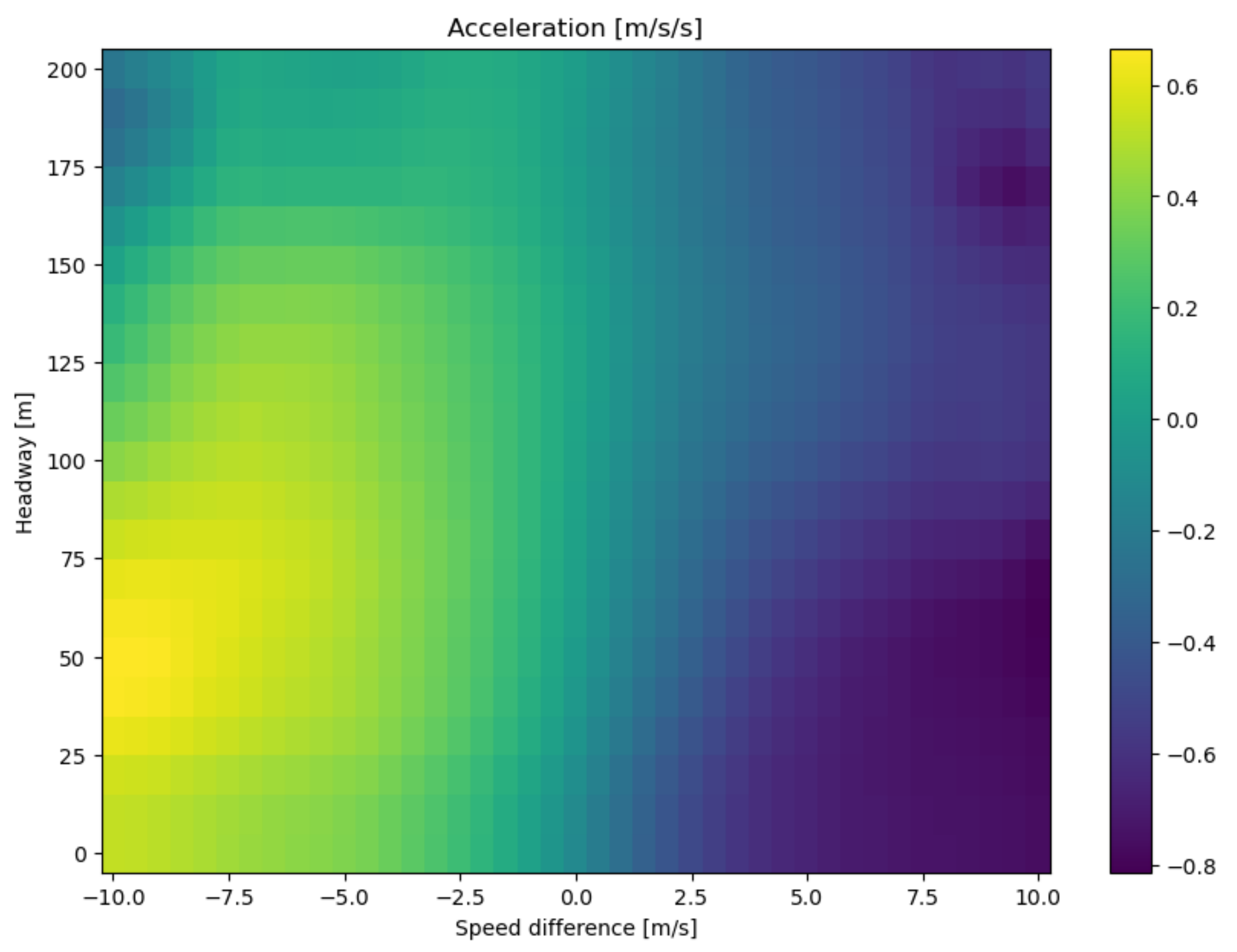
In [17]:
```python
upsilon = 1.5  # m/s
sigma = 30  # m

a = np.zeros((len(s), len(dv)))

for i in range(len(dv)):
    print(f"Iteration {i+1} van {len(dv)}")
    for j in range(len(s)):
        dv_grid = dv[i]
        s_grid = s[j]

        # Calculate weights based on the difference between dv, s and the data
        weights = np.exp(-np.abs((dv_grid - dv_data) / upsilon) - np.abs((s_grid - s_data) / sigma))

        weighted_sum = np.sum(weights * a_data)
        weight_total = np.sum(weights)

        if weight_total > 0:
            a[j, i] = weighted_sum / weight_total
        else:
            a[j, i] = 0
```

```
Iteration 1 van 41
Iteration 2 van 41
Iteration 3 van 41
Iteration 4 van 41
Iteration 5 van 41
Iteration 6 van 41
Iteration 7 van 41
Iteration 8 van 41
Iteration 9 van 41
Iteration 10 van 41
Iteration 11 van 41
Iteration 12 van 41
Iteration 13 van 41
Iteration 14 van 41
Iteration 15 van 41
Iteration 16 van 41
Iteration 17 van 41
Iteration 18 van 41
Iteration 19 van 41
Iteration 20 van 41
Iteration 21 van 41
Iteration 22 van 41
Iteration 23 van 41
Iteration 24 van 41
Iteration 25 van 41
Iteration 26 van 41
Iteration 27 van 41
Iteration 28 van 41
Iteration 29 van 41
Iteration 30 van 41
Iteration 31 van 41
Iteration 32 van 41
Iteration 33 van 41
Iteration 34 van 41
Iteration 35 van 41
Iteration 36 van 41
Iteration 37 van 41
Iteration 38 van 41
Iteration 39 van 41
Iteration 40 van 41
Iteration 41 van 41
```

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 [18]:
```python
X, Y = np.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)
```



Considering Negative (slower than leader) and positive (faster than leader) speed differences: np.abs() ensures that positive and negative deviations are accounted for symmetrically

Considering small and large headways: sigma (headway sensitivity) determines how strongly headway differences affect the weights