Import relevant packages here.

In [3]:
```
import matplotlib.pyplot as plt
import numpy as np
```

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 [4]:
```
data = np.loadtxt('cf_data.csv', dtype=np.float64, delimiter=',', skiprows=1)
print(data)
```

```
[[ -0.74324   53.5427     1.24257 ]
 [ -0.55723   53.612      1.77792 ]
 [ -0.454769  53.6541     0.544107]
 ...
 [  5.13764  115.118      0.232283]
 [  5.15348  114.599      0.262078]
 [  5.25868  113.112     -0.61244 ]]
```

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 [5]:
```
dv = np.linspace(-10, 10, 41)
s = np.linspace(0, 200, 21)
a = np.zeros([dv.size, s.size])
```

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 [6]:
```
DV = data[:, 0]
S = data[:, 1]
A = data[:, 2]
```

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 [45]:
```
import time
upsilon = 1.5
sigma = 30

def filter_a(dv, s):


    dv_grid, s_grid = np.meshgrid(dv, s)

    # timer, progress
    start = time.time()
    progress, total = 0, np.size(dv_grid)

    def one_step_filter(dv_item, s_item):
        nonlocal progress

        # main body
        omega = lambda DV_item, S_item: np.exp(- np.abs(dv_item - DV_item)/upsilon - np.abs(s_item - S_item)/sigma)
        smooth_A_item = np.sum(omega(DV, S) * A) / np.sum(omega(DV, S))

        # print progress

        progress += 1

        if progress % 1e2 == 0:
            print("progress now: %.2f %%" % (progress / total * 1e2), ", time elapsed %.2f s" % (time.time() - start))

        return smooth_A_item

    one_step_filter = np.vectorize(one_step_filter)

    return one_step_filter(dv_grid, s_grid)


a = filter_a(dv, s)

print("complete!")
print("a shape:", a.shape)
```

```
progress now: 11.61 % , time elapsed 0.23 s
progress now: 23.23 % , time elapsed 0.41 s
progress now: 34.84 % , time elapsed 0.60 s
progress now: 46.46 % , time elapsed 0.77 s
progress now: 58.07 % , time elapsed 0.94 s
progress now: 69.69 % , time elapsed 1.13 s
progress now: 81.30 % , time elapsed 1.30 s
progress now: 92.92 % , time elapsed 1.47 s
complete!
a shape: (21, 41)
```

In [41]:
```
# test case
a_test = filter_a(dv[:25], s[:2])
print(a_test)
```

```
[[ 0.52879029  0.52197296  0.51196083  0.49443067  0.47489727  0.4564993
   0.43853545  0.42333708  0.41087822  0.39771161  0.3804174   0.35928689
   0.32570546  0.28687769  0.24632719  0.20100141  0.14857471  0.09012394
   0.02700055 -0.04124758 -0.11562593 -0.19879562 -0.28130451 -0.35755138
  -0.42593754]
 [ 0.53152837  0.52472414  0.51473605  0.49722638  0.47772383  0.4593541
   0.44141404  0.42624089  0.41381434  0.40068399  0.38342171  0.3623419
   0.32881282  0.29005594  0.24939012  0.2039263   0.15133874  0.09271452
   0.0295022  -0.03888711 -0.11350169 -0.19686542 -0.27981293 -0.3564541
  -0.42518631]]
```

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 [42]:
```
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)
```