The deadline for this exercise sheet is **Monday, 18.06.2018, 10:00.**

Introductory Words

Remember that you need proper documentation to pass the homework. The documentation doesn't need to be *perfect*, but everything that needs a docstring, should have a docstring.

Please make sure that your libraries are up to date. For numpy your version must be ≥ 1.13 and for matplotlib ≥ 2.2. You can check the installed version either using pip: `pip show numpy matplotlib` or directly in Python:

```
1 import numpy as np
2 import matplotlib as mpl
3
4 print("Numpy:", np.__version__)
5 print("Matplotlib:", mpl.__version__)
```

If you need to upgrade either, use conda or pip: `conda install numpy matplotlib` or `pip install numpy matplotlib`. This should install the latest version.

1 Warm-Up

1.1 Number Stuff with NumPy

Given the following ndarrays:

```
1 import numpy as np
2
3
4 a = np.array([1, 2, 3])
5 b = np.array([-1, 2, -3])
6 c = np.arange(1, 10).reshape(3,3)
7 d = np.array([[1,2],[-2,-1],[5,5]])
```

calculate the following operations. Order matters.

- The dot product of a and b **Solution:** np.dot(a, b)
- The dot product of c and a **Solution:** np.dot(c, a)
- The matrix multiplicative of c and d **Solution:** np.matmul(c,d)
- The matrix multiplicative of the transpose of d and c **Solution:** np.matmul (d.T, c.T)
- the overall mean of a and b together **Solution:** np.mean([a,b])

Solution:

```python
"""A simple script to test some numpy functionality"""
import numpy as np


# the arrays we are going to work with
a = np.array([1, 2, 3])
b = np.array([-1, 2, -3])
c = np.arange(1, 10).reshape(3, 3)
d = np.array([[1, 2], [-2, -1], [5, 5]])

# sine none of the results are further needed we do the calculation directy
# in the output function
# Note that f-strings evaluate the statements in curly braces as python
# statements before converting them to string. Therefore, we can do all the
# numpy calculations inside the f-strings
print(
    f"a: {a}", f"b: {b}", f"c:\n{c}", f"d:\n{d}",
    f"dot(a, b): {np.dot(a, b)}",
    f"dot(c, a): {np.dot(c, a)}",
    f"c * d:\n{np.matmul(c, d)}",
    f"d^T * c^T:\n{np.matmul(d.T, c.T)}",
    f"mean(a, b): {np.mean([a,b]): .3f}",
    sep="\n"
)
```

1.2 Plot, plot away

1.2.1 Crossed out

Plot the following functions from -4 to 8 into one figure. Use at least 100 points.
Note: You only need to create *one* ndarray for the x-coordinates.

- $y_1 = 2x - 2$
- $y_2 = -2x + 6$
- $y_3 = 0.25(x - 2)^2 + 2$
- $y_4 = -0.25(x - 2)^2 + 2$

1.2.2 Markers all around

From NumPy's random module use the normal function to create 1000 randomly drawn numbers. Draw from a normal distribution with center (loc) 0 and a standard deviation (scale) of 5. Then plot those numbers into a histogram, with 25 bins, and a bar width of 0.9.
Have a look at the numpy documentation for random.normal.

1.2.3 Who is that plot?!

In the accompanying .zip archive you can find the `plot_me` module. Import the `get_x_y` function from it. The function will return two numpy ndarrays in the form `x, y`, which are x and y coordinate pairs. Plot those pairs using a scatter plot, with a scale (`s`) of 0.5.
Note: The function can take a few seconds to compute.

Solution:

```
"""
This script implements the functionality of the whole task 1.2

Each subtask is plotted in its own figure so everytime one is closed the
next one opens to show the result.

Only works with the module plot_me present.
"""
import numpy as np
import matplotlib.pyplot as plt

from plot_me import get_x_y # convenience

#------------------------------------------------------------------------

# Task 1.2.1: Crossed Out
#------------------------------------------------------------------------

# since the functions are rather simple we can implement them as lambdas
y1 = lambda x: 2*x - 2
y2 = lambda x: -2*x + 6
y3 = lambda x: -0.25*(x - 2)**2 + 2
y4 = lambda x: 0.25*(x - 2)**2 + 2

# 100 should be enough for a rather smooth curve
x = np.linspace(-4, 8, 100)

# and plot them all. We could have done that in one plot function, but I prefer
# the longer version.
fig = plt.figure()
plt.title("Crossed Out") # they all meet in one point :)
plt.plot(x, y1(x))
plt.plot(x, y2(x))
plt.plot(x, y3(x))
plt.plot(x, y4(x))
plt.show()

#------------------------------------------------------------------------

# Task 1.2.2: Markers All Around
#
```

```
39 fig = plt.figure()
40 # draw the sample from a normal distribution, with center 0 and std 5
41 # we want a 1d array with 10000 entries
42 y = np.random.normal(0, 5, (10000))
43 plt.title("Markers All Around")
44 # and plot the thing. bins as an integer automatically divides the
       whole range
45 # into evenely spaced bins. rwidth as .9 so they are cleared
       distinguishable
46 plt.hist(y, bins=25, rwidth=.9)
47 plt.show()
48
49 #
   ------------------------------------------------------------------------

50 # Task 1.2.3: Who Is That Plot
51 #
   ------------------------------------------------------------------------

52 fig = plt.figure()
53 x, y = get_x_y() # get the data from the supplied function
54 plt.title("Who Is That Plot?!") # it's a pikachu!
55 # setting the scale makes it more visible as the dots are very close
56 plt.scatter(x, y, s=0.5)
57 plt.show()
```

2 The great Plotting

2.1 Wavy Waves

Using your new gained knowledge, recreate the figure in the image below 1. The graphs are plotted from -2π to 2π. You do not have to modify the ticks on the x-axis. If you want to recreate those as well, look into the rc doc, which allows you to set all kind of formatting. Especially interesting is the `usetex` boolean property of `text`. Also see this explanation.

2.2 Bonus: All alive.

Let's bring this plot to life! Animate the sine waves in such a way, that the top left sine wave moves to the left and the top right sine wave moves to the right. Wave-y!

2.3 Bonus-Bonus: Wave the wave

Now that we got that animation up and going, let's animate the bottom sine wave as well. This time not just left and right, but up and down as well. Move the whole sine wave on a sine wave. So that the whole curve shifts to the left

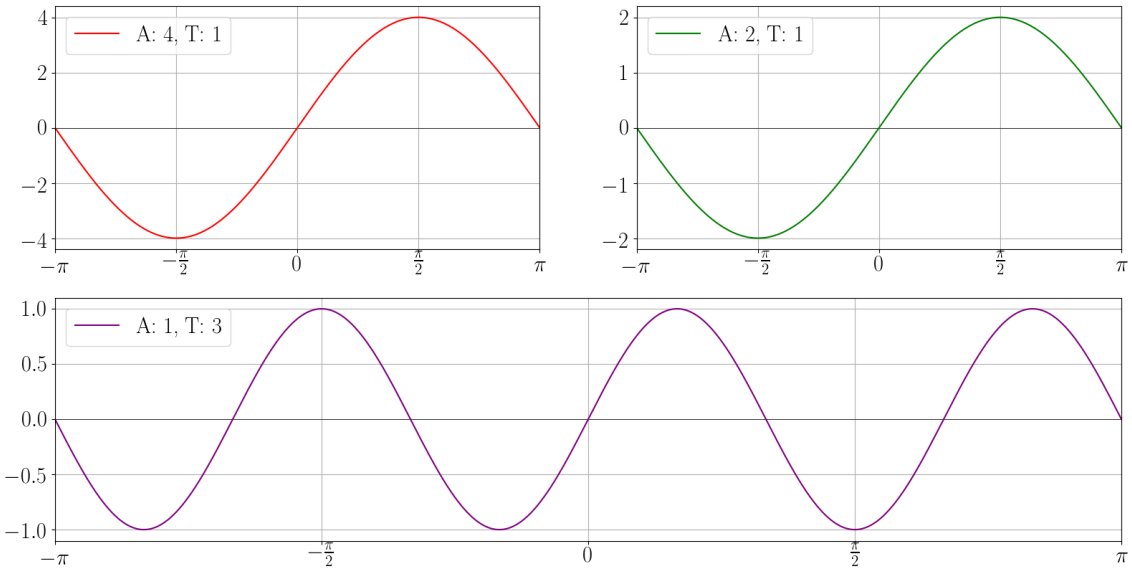

Figure 1: Three sine waves with different properties

as well as up and down as if it was moving along another sine wave.
This is a tricky one.

Solution:

```
"""
This script plots three animated sine curves.

The sine curves have different amplitudes and periods, but are animated
using the same amount of time to complete one period. Thus one sine wave moves
"faster".

There are constants at the top of the file which change the output. Since the
animation of three waves with a somewhat complex function (sine is not that
easy to calculate) is computation heavy, the animation might be lagging.

Bundled with this file comes the "animation.mp4" file, which contains the full
animation, without lag. If ffmpeg is present it can be recreated by uncommenting
the second last line of the file.
"""
```

```
16 import numpy as np
17 import matplotlib.pyplot as plt
18 import matplotlib.animation as animation
19
20
21 #
    ------------------------------------------------------------------------------

22 # constants
23 #
    ------------------------------------------------------------------------------

24 # how many samples to draw and thus how many frames there will be
25 # higher numbers take more computation time, but is a more accurate
       curve
26 # Defaulted to 500.
27 SAMPLES = 500
28 # How long one circle of animation should play in ms. Defaulted to 4
       seconds.
29 ANIM_LENGTH = 4000
30 # How big the ouput figure should be in inches. Defaults to a 12 by 8
       canvas
31 FIG_SIZE = (12, 8)
32 # The font size of the labels
33 FONT_SIZE = 14
34
35 #
    ------------------------------------------------------------------------------

36 # config of matplotlib parameters
37 #
    ------------------------------------------------------------------------------

38 # we will mark the x-axis every half-pi steps
39 xtick = np.arange(-np.pi, np.pi+np.pi/2, np.pi/2)
40 # we will use mathtext to display the pi symbol and fractions
41 # one label for each tick in xtick. The syntax is similar to latex
       maths.
42 xlabel = [r'$-\pi$', r'$-\frac{\pi}{2}$',
43           r'$0$', r'$\frac{\pi}{2}$', r'$\pi$']
44 # config paramters for matplotlib to change the font
45 font = {'family': 'sans-serif',
46         'sans-serif': ['Calibri', 'Ubuntu'],
47         'size': FONT_SIZE}
48
49 plt.rc('font', **font)
50 #
    ------------------------------------------------------------------------------

51 # functions
52 #
    ------------------------------------------------------------------------------

53 def create_sine_func(x, amplitudes, periods):
54     """
55     Creates a function to create sine waves with the given parameters
           and
```

```
56      changeable offsets.
57
58      Creates a function that creates a sine wave for each entry in
            amplitudes and
59      periods. The amplitudes and periods, as well as the x-values that
            the sine
60      waves are calculated off, are unchangeable. But the sine waves can
            still be
61      shifted along the x- and y-axis. This allows for animation of those
             waves.
62
63      Args:
64          x: 1d array-like
65              The list of x values to use to calculate the sine waves
66          amplitudes: 1d array-like
67              The list of amplitudes to use for the different sine waves
68              Must be the same size as amplitudes.
69          periods:
70              The list of periods to use for the different sine waves.
71              Must be the same size as amplitudes.
72
73      Returns:
74          sine_waves: function
75              A function which takes two lists the same size as
                    amplitudes
76              and then returns a list of sine curves' y-values.
77      """
78      def sine_waves(x_offs, y_offs):
79          """
80          Creates a list of sine-wave arrays offset by the values in
                x_offs and
81          y_offs.
82
83          For each entry in amplitudes there must be a value in x_offs
                and y_offs.
84          The function than calculates that many sine waves with the
                amplitude and
85          period from the outer function's lists. Each sine wave can be
                offset on
86          the x- and the y-axis, which the values from x_offs and y_offs
                are used
87          for.
88
89          Args:
90              x_offs: 1d array-like
91                  The x-offsets for the sine waves. Must be the same
                        length as
92                  amplitudes.
93              y_offs: 1d array-like
94                  The y-offsets for the sine waves. Must be the same
                        length as
95                  amplitudes.
96
97          Returns:
98              sines: 1d list
99                  The list containing the sine arrays. Each entry is the
                        array of
```

```
100                 the y-values of a sine wave with the supplied
                        parameters.
101         """
102         sines = []
103         # a set of amplitude, period, x offset and y offset defines one
                sine curve
104         # so for all sets, create one sine wave, based on the supplied
                x values
105         for a, T, x_off, y_off in zip(amplitudes, periods, x_offs,
                y_offs):
106             # amplitude changes the "height" of the curve into positive
                    and
107             # negative direction. period changes how long the curve
                    needs
108             # for one repetition. x-offset moves the curve along the x-
                    axis,
109             # and y-offset along the y-axis respectively.
110             sine = a * np.sin(T * (x + x_off)) + y_off
111             sines.append(sine)
112         return sines
113
114     return sine_waves
115
116
117 def animate(idx, *fargs):
118     """
119     Animate the sine waves by shifting them along the axes.
120
121     Calculates the sine waves anew shifted a tiny bit more.
122     fargs needs to supply the tuple: (sine_waves, lines, x_offsets,
            y_offsets)
123     The order is important. sine_calc is the output of create_sine_func
            , lines
124     is the list of the plots in the figure, and x- and y-offsets are
            the list
125     of offsets to be applied to the sine waves.
126
127     Since the top-left and bottom curve move to the left, the x-offset
            is
128     supplied as is. For the top-right curve it is negated to move the
            curve to
129     the right. Only the bottom curve is shifted on the y-axis, hence
            the
130     y-offsets for the top curves are 0.
131
132     The index of the animation is important to access how much to shift
             the curves.
133
134     Args:
135         idx: int
136             The index of the animation playing. Used to access offsets.
137         fargs: tuple
138             A tuple containing the sine_calc function, a list of 2D-
                    lines to
139             replace the y-data on, the x-offsets and y-offsets to index
                     with idx
140
```

```
141     Returns:
142         lines: list
143             The list of altered 2D-Line objects to be plotted in the
                    next frame
144     """
145     sine_waves = fargs[0] # extract function object
146     lines = fargs[1] # extract list of artists
147     x_off = fargs[2][idx] # extract offset for this animation frame
148     y_off = fargs[3][idx]
149 
150     # create the lists supplied to the sine_waves function
151     x_offs = [x_off, -x_off, x_off]
152     y_offs = [0, 0, y_off]
153 
154     # calculate the new sine waves
155     sines = sine_waves(x_offs, y_offs)
156 
157     # and replace the data
158     for i in range(len(lines)):
159         lines[i].set_ydata(sines[i])
160 
161     return lines
162 
163 #
        ------------------------------------------------------------------------

164 # set-up of figure and axes objects
165 #
        ------------------------------------------------------------------------

166 # create a new figure object
167 fig = plt.figure("Inspecting Sines", figsize=FIG_SIZE)
168 # ... and give it a title
169 fig.suptitle("The Waving Sines")
170 # create the subplots for the three waves
171 # tl: top-left
172 ax_tl = plt.subplot(221)
173 # tr: top_right
174 ax_tr = plt.subplot(222)
175 # b: bottom
176 ax_b = plt.subplot(212)
177 
178 # And iterable for the loops
179 axes = [ax_tl, ax_tr, ax_b]
180 
181 #
        ------------------------------------------------------------------------

182 # Setting up and initialising plots
183 # WIHTOUT THE BONUS TASK THIS IS ALL THAT WAS NEEDED
184 #
        ------------------------------------------------------------------------

185 # create the SAMPLES x-values
186 x = np.linspace(-np.pi, np.pi, SAMPLES)
187 # the amplitudes our sine waves will have in order tl -> tr -> b
188 amps = [4, 2, 1]
```

```
189 # the periods of the sine waves
190 periods = [1, 1, 3]
191 # the colour of the graphs
192 colours = ['red', 'green', 'purple']
193
194 # create the sine_waves functions with the x-values and the amplitudes
        and
195 # periods we want
196 sine_waves = create_sine_func(x, amps, periods)
197 # initialise the unshifted sine waves
198 sines = sine_waves([0, 0, 0], [0, 0, 0])
199
200 # create a list of all the plots
201 lines = []
202 for i in range(len(amps)):
203     # the label to describe the sine wave
204     label = f"A: {amps[i]}, T: {periods[i]}"
205     # and plot the plot
206     line = axes[i].plot(x, sines[i], color=colours[i], label=label)
207     lines += line
208
209 # all subplots (axes objects) share some layout options
210 # so we can define them in a loop. No need to write the same code
        thrice.
211 for ax in axes:
212     # set the show-limits from -pi to pi
213     ax.set_xlim((-np.pi, np.pi))
214     # show a grid for better readability
215     ax.grid(True)
216     # usually matplotlib leaves a bit of space on the side of the
            graphs
217     # this would look silly with animations though. Therefore we get
            rid of them
218     ax.margins(xmargin=0)
219     # up above we defined the xticks and their labels, here we apply
            them
220     ax.set_xticks(xtick)
221     ax.set_xticklabels(xlabel)
222     # pack the legend into the top left corner of each graph
223     ax.legend(loc=2)
224     # just so that there is no confusion...
225     ax.set_xlabel("x")
226     ax.set_ylabel("y")
227
228 ###### MAIN TASK IS DONE HERE
229
230 # a small addon for the bottom graph. Since we defined the layout now
        with the
231 # unshifted curve, moving it along the y-axis will put it out of bounds
         and clip
232 # the curve. Since we move the bottom curve on a sine wave, the maximum
         offset in
233 # y is 1. So we take the amplitude (the maximum height of the unshifted
         curve)
234 # and add 1.1 and make those the limit, and having some margin.
235 b_yrange = amps[2] + 1.1
236 ax_b.set_ylim((-b_yrange, b_yrange))
```

```
237
238 #
    ------------------------------------------------------------------------------

239 # initialise figures
240 #
    ------------------------------------------------------------------------------

241 # we need as many x_offsets as we will have frames.
242 # we have the same amount of frames as datapoints. This is so that
        there is no
243 # weird behaviour when a curve is shifted inbetween two data points (
        sometimes
244 # that leads to a small "bump" in the animation)
245 x_offs = np.linspace(0, 2*np.pi, SAMPLES)
246 # the bottom sine wave is supposed to move on a sine wave. So the y-
        offsets
247 # must be sine values
248 y_offs = np.sin(x_offs)
249
250 # build the tuple to be supplied to our animation function
251 fargs = (sine_waves, lines, x_offs, y_offs)
252
253 # Phew. Done. And finally: Here. We. Go.
254 anim = animation.FuncAnimation(
255     fig,
256     animate,
257     frames=SAMPLES,
258     interval=(ANIM_LENGTH / SAMPLES),
259     fargs=fargs,
260     save_count=25
261 )
262
263 # if you have ffmpeg in this folder or your PATH you can save the
        animation into
264 # a video file.
265 # anim.save('animation.mp4')
266
267 # actually we only start going here. but it was more dramatic up there.
268 plt.show()
```