Exercises consist of both pen&paper and computer assignments. Pen&paper ques-
tions are solved at home before exercises, while computer assignments are solved
during exercise hours. The computer assignments are marked by text [python] and
Pen&paper questions by text [pen&paper]

1. [pen&paper] Consider the model
\[ x[n] = As[n] + w[n], \quad n = 0, 1, \ldots, N - 1, \]
where \( w[n] \sim N(0, \sigma^2) \), \( s[n] \) is a known signal, and \( A \) is the parameter to be
estimated. Derive the maximum likelihood estimator of \( A \).

*Hint:* The same procedure as on the Monday 15.1. lecture applies: Just write the
probability of observing \( x = (x[0], \ldots, x[N-1]) \), and maximize with respect to
\( A \). The only difference to lecture case is that you have the known signal \( s[n] \) as
an additional variable. However, it is known, so just treat it as at as a constant (it
will be part of the end result.

The lecture slides describe an optimal detector for a known waveform \( s[n] \). Ap-
ply it to design the optimal detector for a step edge:
\[ s[n] = \begin{cases} -1, & \text{for } 0 \leq n < 10 \\ 1, & \text{for } 10 \leq n < 20 \end{cases} \]
Simplify the expression as far as you can.

3. [python] Estimate sinusoidal parameters.
   a) Generate a 100-sample long synthetic test signal from the model:
\[ x[n] = \sin (2\pi f_0 n) + w[n], \quad n = 0, 1, \ldots, 99 \]
with \( f_0 = 0.017 \) and \( w[n] \sim N(0, 0.25) \). Note that \( w[n] \) is generated by
\( w = \text{numpy.sqrt}(0.25) \times \text{numpy.random.randn}(100) \). Plot the
result.

   b) Implement code from estimating the frequency of \( x \) using the maximum
likelihood estimator:
\[ \hat{f}_0 = \text{value of } f \text{ that maximizes } \left| \sum_{n=0}^{N-1} x(n) e^{-2\pi i f n} \right|. \]
Implementation is straightforward by noting that the sum expression is in fact a dot product:

$$\hat{f}_0 = \text{value of } f \text{ that maximizes } |x \cdot e|,$$

with $x = (x_0, x_1, \ldots, x_{N-1})$ and $e = (e^{-2\pi i f_0}, e^{-2\pi i f_1}, \ldots, e^{-2\pi i f (N-1)})$.

Use the following template and fill in the blanks.

```python
scores = []
frequencies = []

for f in numpy.linspace(0, 0.5, 1000):
    # Create vector e. Assume data is in x.
    n = numpy.arange(100)
    z = # <compute -2*pi*i*f*n. Imaginary unit is 1j>
    e = numpy.exp(z)
    score = # <compute abs of dot product of x and e>
    scores.append(score)
    frequencies.append(f)

fHat = frequencies[np.argmax(scores)]
```

c) Run parts (a) and (b) a few times. Are the results close to true $f_0 = 0.017$?

4. **python** Load a dataset of images split to training and testing.

We will train a classifier to classify hand written digits. Scikit-learn provides a number of sample datasets. Load the `digits`-dataset as follows.

```python
from sklearn.datasets import load_digits
digits = load_digits()
```

The result is a `dict` structure that can be accessed using `keys`. Find all keywords of the dict with `print(digits.keys())`. The interesting ones for us are: `’images’, ‘data’ and ‘target’`.

Plot the first image of the 1797 numbers like this.

```python
import matplotlib.pyplot as plt
plt.gray()
plt.imshow(digits.images[0])
plt.show()
```

Check that this corresponds to the label `digits.target[0]`. 

The images are vectorized as rows in the matrix digits.data, whose size is $1797 \times 64$ (1797 images of size $8 \times 8$).

Split the data to training and testing sets, such that the training set consists of 80% and test set 20% of the data. Use sklearn.cross_validation.train_test_split to do this and create variables $\text{x\_train, y\_train, x\_test, y\_test}$.

5. **python**  
   *Train a classifier using the image data.*
   
   In this exercise we will train a nearest neighbor classifier with the data arrays of exercise 4.

   a) Initiate a KNN classifier with
   ```python
   from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier()
   ```

   b) Train the classifier using the training data.

   c) Predict the labels for the test data.

   d) Compute the accuracy using sklearn.metrics.accuracy_score.