General constraints for code submissions Please adhere to these rules to make our and your life easier! We will deduct points if your solution does not fulfill the following:

- If not stated otherwise, we will use exclusively Python 3.6.
- If not stated otherwise, we expect a Python script, which we will invoke exactly as stated on the exercise sheet.
- Your solution exactly returns the required output (neither less nor more) – you can implement a `--verbose` option to increase the verbosity level for developing.
- Add comments and docstrings, so we can understand your solution.
- (If applicable) The `README` describes how to install requirements or provides addition information.
- (If applicable) Add required additional packages to `requirements.txt`. Explain in your `README` what this package does, why you use that package and provide a link to it's documentation or GitHub page.
- (If applicable) All prepared unittests have to pass.
- (If applicable) You can (and sometimes have to) reuse code from previous exercises.

Now that you have learned about the theory behind GPs, you will have to use that theory to implement GPs yourself.

1. **Gaussian Processes** [14 points]

   The exercise is mostly concerned with equations 2.11 and 2.12 in chapter 2 of "Gaussian Processes for Machine Learning[1]"

   (a) Your first task is to implement 2.11 to predict the mean and variance given some observations. [4pt.]

   $$f_\star|\boldsymbol{x}_\star, \boldsymbol{X}, \boldsymbol{y} \sim \mathcal{N}(\frac{1}{\sigma_n^2}\phi(\boldsymbol{x}_\star)^\intercal A^{-1}\Phi\boldsymbol{y}, \phi(\boldsymbol{x}_\star)^\intercal A^{-1}\phi(\boldsymbol{x}_\star))$$

   where $\Phi = \phi(\boldsymbol{X})$ and $A = \sigma_n^{-2}\Phi\Phi^\intercal + \Sigma_p^{-1}$.

   We provide a function prototype that takes the observations, a function ϕ, and an array of points, $[\mathbf{x}_1, \mathbf{x}_2, \dots]$, as arguments. Compute the mean μ and the variance σ^2 at all $\mathbf{x}$. For matrix inversion you can use `numpy.linalg.inv`. Use the data provided in your source folder to create a plot that shows the mean and the 2σ confidence interval around it for a feature space of size $N = 2$. Use $\sigma_n = 1$ and $\Sigma_p = I$ and

   $$\phi(x) = (1, x)^T$$

   which corresponds to Bayesian *linear* regression for one dimensional input.

   This exercise will not be automatically graded. To let us quickly grade your exercises please add your solutions to a PDF and upload that PDF to your repository. Further we expect all plots to have axes labels and a legend.

   (b) Implement a second function based on equation 2.12: [5pt.]

   $$f_\star|\boldsymbol{x}_\star, \boldsymbol{X}, \boldsymbol{y} \sim \mathcal{N}(\phi_\star^\intercal\Sigma_p\Phi(\Phi^\intercal\Sigma_p\Phi + \sigma_n^2 I)^{-1}\boldsymbol{y}, \phi_\star^\intercal\Sigma_p\phi_\star - \phi_\star^\intercal\Sigma_p\Phi(\Phi^\intercal\Sigma_p\Phi + \sigma_n^2 I)^{-1}\Phi^\intercal\Sigma_p\phi_\star)$$

   Convince yourself, that both yield the same values, by checking the output for the given input[2]. Compare the time it takes for both implementations to compute the output for the given data, and points $\mathbf{x}_i$, for a growing number of features. Again use $\sigma_n = 1$ and $\Sigma_p = I$ and

   $$\phi_n(x) = (1, x, x^2, \dots, x^{n-1})^T \tag{1}$$

   to plot the computing time for $n = 2, 4, 8, \dots, 2048, 4096$. Your implementation for both 2.11 and 2.12 should pass the tests in `test_runtime` and `test_prediction_equality`.

[1] `http://www.gaussianprocess.org/gpml/chapters/RW2.pdf`

[2] By the nature of numerical calculations, the results will not be identical, but the difference will be very small. Use `numpy.allclose` to test for equality.

(c) Apart from the reliable uncertainty estimates, a GP has the additional advantage to allow for easy optimization of its own hyperparameters by gradient based optimization of the log marginal likelihood (equation 5.8). [5pt.]

$$\log p(\mathbf{y}|X,\theta) = -\frac{1}{2}\mathbf{y}^T K_y^{-1}\mathbf{y} - \frac{1}{2}\log|K_y| - \frac{n}{2}\log 2\pi$$

where θ denotes the kernel hyperparameters (see eq. 5.1). In this exercise you will use equation 2.12 to optimize hyperparameters of the kernel. By replacing the inner products with an explicit kernel function equation $K_f = k(X,X) = \Phi^\intercal \Sigma_p \Phi$ where $K_y = K_f + \sigma_n^2 I$, (σ_n noise level of the data) equation 2.12 becomes:

$$f_\star|\boldsymbol{X}_\star, \boldsymbol{X}, \boldsymbol{y} \sim \mathcal{N}(K_*^T K_y^{-1}\mathbf{y}, K_{**} - K_*^T K_y^{-1} K_*)$$

where $K_* = k(\mathbf{X}, \mathbf{X}^\star) = \phi_\star^\intercal \Sigma_p \Phi$ and $K_{**} = k(\mathbf{X}^\star, \mathbf{X}^\star) = \phi_\star^\intercal \Sigma_p \phi_\star$. Here we will use the squared exponential kernel:

$$k(\mathbf{X_p}, \mathbf{X_q}) = \sigma_f^2 \exp\left(-\frac{1}{2}(\mathbf{X_p} - \mathbf{X_q})^T l^{-2} I (\mathbf{X_p} - \mathbf{X_q})\right)$$

For this exercise you are given the implementation of 2.12 in the kernel formulation and are asked to implement the squared exponential kernel and the log marginal likelihood (σ_f, L). The optimization is carried out via scipy optimize and already implemented for you. Try to play around with the random seed to see how different training data effects to the HPO of the GP.

Your implementation should satisfy the test in `test_gp_hpo.py`.

2. **Code Style** [1 point]

On every exercise sheet we will also make use of `pycodestyle`[3] to adhere to a common python standard. Your code will be automatically evaluated on every push and you will be informed if the test fails. To check it locally, first run `pip install pycodestyle` and then run `pycodestyle --max-line-length=120 src/` to check your source file folder. Alternatively run `make checkstyle`

[3] former pep8