exercise 3

Gaussian process regression

solutions due

until **November 25, 2025** at **23:59** via **ecampus**

general remarks

These exercises are about implementing Gaussian process models from scratch. In fact, you will learn how to reproduce all the relevant figures we saw in lecture 07.

Note: Please do *not* use `sklearn` or other fancy GP libraries which you may find on the Web. Rather, implement all your solutions using only plain vanilla `numpy`, `scipy`, `matplotlib`, ... functionalities.

One of the goals of all our exercises in this course is for you to become able to (vibe-)code fancy libraries yourself (if need be). Indeed, while there exist machine learning libraries which are developed by professionals (say at Google, Meta, ...), many other machine learning libraries on the Web are created by people who do not always seem to know what they are doing and we do not want you to become like these people.

If you like to read up on further details behind Gaussian processes, then this standard textbook

> C.E. Rasmussen & K.I. Williams, "Gaussian Processes for Machine Learning", MIT Press, 2006

is recommended. It is also freely available on the Web (just search for it).

task 3.1 [5 points]

basic functionalities

Note: Do not use `for` loops in your solutions to this task but implement properly vectorized `numpy` or `scipy` code. (This can be done! Consult the Web or an AI f you need hints or inspiration).

In this and in the following task, we let

$$\boldsymbol{u} = \big[u_1, u_2, \ldots, u_{n_u}\big]^\intercal \in \mathbb{R}^{n_u}$$
$$\boldsymbol{v} = \big[v_1, v_2, \ldots, v_{n_v}\big]^\intercal \in \mathbb{R}^{n_v}$$

be two arbitrary, real valued vectors of dimensions n_u and n_v, respectively.

task 3.1.1 [2.5 points]

Implement a function `diffMatrix` which takes inputs $\boldsymbol{u}$ and $\boldsymbol{v}$ and returns

$$\boldsymbol{M} \in \mathbb{R}^{n_u \times n_v}$$

with entries

$$\big[\boldsymbol{M}\big]_{ij} = u_i - v_j$$

task 3.1.2 [2.5 points]

Implement a function `prodMatrix` which takes inputs $\boldsymbol{u}$ and $\boldsymbol{v}$ and returns

$$\boldsymbol{M} \in \mathbb{R}^{n_u \times n_v}$$

with entries

$$\big[\boldsymbol{M}\big]_{ij} = u_i \cdot v_j$$

task 3.2 [5 points]

kernel matrices

Note: Do not use `for` loops in your solutions to this task but implement properly vectorized `numpy` or `scipy` code. (Again, this is doable!)

Use your solutions from task 3.1, to accomplish the following . . .

task 3.2.1 [2.5 points]

Implement a function `linearKernelMatrix` which takes two inputs $\boldsymbol{u}$ and $\boldsymbol{v}$ and one parameter α and returns a matrix

$$\boldsymbol{K} = \boldsymbol{K}\big(\boldsymbol{u}, \boldsymbol{v} \;\big|\; \alpha\big) \in \mathbb{R}^{n_u \times n_v}$$

with entries

$$\big[\boldsymbol{K}\big]_{ij} = \alpha\, u_i\, v_j$$

task 3.2.2 [2.5 points]

Implement a function `gaussKernelMatrix` which takes two inputs $\boldsymbol{u}$ and $\boldsymbol{v}$ and two parameters α and σ returns a matrix

$$\boldsymbol{K} = \boldsymbol{K}\big(\boldsymbol{u}, \boldsymbol{v} \;\big|\; \alpha, \sigma\big) \in \mathbb{R}^{n_u \times n_v}$$

with entries

$$\big[\boldsymbol{K}\big]_{ij} = \alpha\, \exp\left(-\frac{(u_i - v_j)^2}{2\,\sigma^2}\right)$$

task 3.3 [10 points]

sampling simple Gaussian processes

Let

$$\boldsymbol{x} = \big[x_1, x_2, \ldots, x_n\big]^\intercal \in \mathbb{R}^n$$

be a vector of n input values and

$$\boldsymbol{0} = \big[0, 0, \ldots, 0\big]^\intercal \in \mathbb{R}^n$$

the n-dimensional vector of all 0s.

For instance, here is a simple *numpy* implementation of vectors like these

```
vecX = np.linspace(-5.0, 15.0, 55)
vec0 = np.zeros_like(vecX)
```

Now, implement and run *numpy* / *scipy* code for the following subtasks.

task 3.3.1 [3 points]

Consider a kernel matrix

$$\boldsymbol{K}_L = \boldsymbol{K}\big(\boldsymbol{x}, \boldsymbol{x} \;\big|\; \alpha_L\big) \in \mathbb{R}^{n \times n}$$

with entries

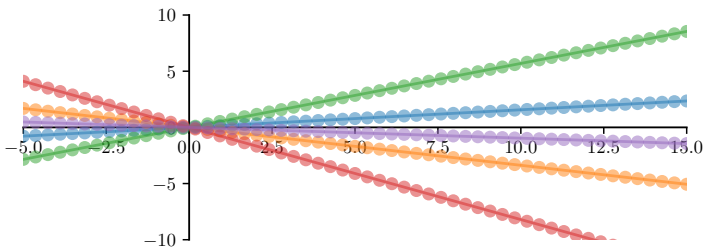
$$\big[\boldsymbol{K}_L\big]_{ij} = \alpha_L \, x_i \, x_j$$

and sample 5 vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_5$ from the Gaussian $\mathcal{N}\big(\boldsymbol{0}, \boldsymbol{K}_L\big)$.

Hint: Just work with the *numpy.random* function *multivariate_normal*.

For each resulting vector $\boldsymbol{y}$, plot the pairs (x_j, y_j). Experiment with different choices of the kernel parameter α_L.

For instance, for $\alpha_L = 1$, your plots could / should look something like this:



Repeat your experiments several times. Are your results always the same?

task 3.3.2 [3 points]

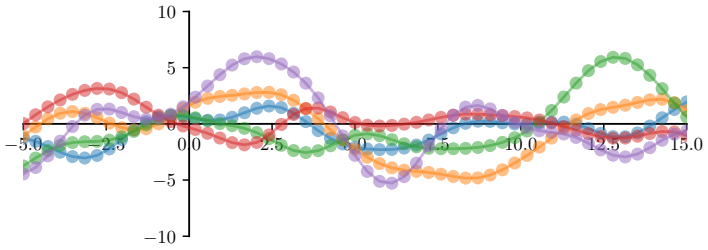
Consider a kernel matrix

$$\boldsymbol{K}_G = \boldsymbol{K}\big(\boldsymbol{x}, \boldsymbol{x} \mid \alpha_G, \sigma_G\big) \in \mathbb{R}^{n \times n}$$

with entries

$$\big[\boldsymbol{K}_G\big]_{ij} = \alpha_G \, \exp\left(-\frac{(x_i - x_j)^2}{2\,\sigma_G^2}\right)$$

and sample 5 vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_5$ from the Gaussian $\mathcal{N}\big(\boldsymbol{0}, \boldsymbol{K}_G\big)$.
For each resulting vector $\boldsymbol{y}$, plot the pairs (x_j, y_j). Experiment with different choices of the kernel parameters α_G and σ_G.
For instance, for $\alpha_G = 6, \sigma_G = 1.5$, your plots could / should look like this:

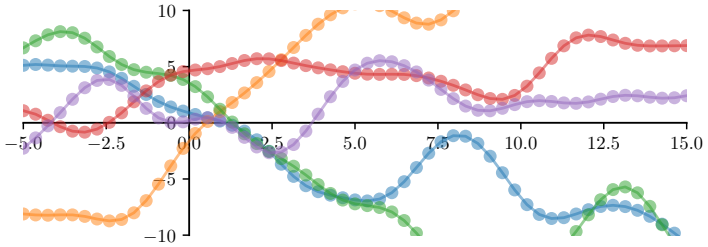


task 3.3.3 [4 points]

Consider a kernel matrix

$$\boldsymbol{K}_{LG} = \boldsymbol{K}_L + \boldsymbol{K}_G$$

and sample 5 vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_5$ from the Gaussian $\mathcal{N}\big(\boldsymbol{0}, \boldsymbol{K}_{LG}\big)$.
For each resulting vector $\boldsymbol{y}$, plot the pairs (x_j, y_j). Experiment with different choices of the three kernel parameters.
For instance, for $\alpha_L = 2, \alpha_G = 6, \sigma_G = 1.5$, your plots could look like this:



task 3.4 [10 points]

fitting a Gaussian processes model to data

In this task, you are supposed to fit a Gaussian process model to the body height and -weight data known from the lectures.

To begin with, read the content of the file

`whData.dat`

into memory. Remove the two outliers and collect the remaining height- and weight values into two vectors

$$\boldsymbol{x} \in \mathbb{R}^n$$
$$\boldsymbol{y} \in \mathbb{R}^n$$

Important trick of the trade: Normalize the entries of $\boldsymbol{y}$ to zero mean. That is, compute

$$\bar{\boldsymbol{y}} = \boldsymbol{y} - \tfrac{1}{n}\,\mathbf{1}\mathbf{1}^\intercal\boldsymbol{y}$$

In your code / implementation, you would of course want to do this using the *numpy* function *mean*.

Now, implement code that uses the data in $\boldsymbol{x}$ to compute a kernel matrix $\boldsymbol{K} = \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{x} \mid \theta_1, \theta_2, \theta_3)$ where

$$\big[\boldsymbol{K}\big]_{ij} = \theta_1\,\exp\left(-\frac{(x_i - x_j)^2}{2\,\theta_2^2}\right) + \theta_3\,x_i\,x_j$$

Extend you code such that it can compute

$$\boldsymbol{C} = \boldsymbol{K} + \theta_4\,\boldsymbol{I}$$

Next, implement a function *negLikelihood* that takes $\boldsymbol{\theta} = [\theta_1, \theta_2, \theta_3, \theta_4]^\intercal$, $\boldsymbol{x}$, and $\bar{\boldsymbol{y}}$ as inputs (in this order) and computes and returns

$$-\mathcal{L}\big(\boldsymbol{\theta} \;\big|\; \boldsymbol{x}, \bar{\boldsymbol{y}}\big) = \tfrac{1}{2}\,\log\big(\det\big[\boldsymbol{C}\big]\big) + \tfrac{1}{2}\,\bar{\boldsymbol{y}}^\intercal\boldsymbol{C}^{-1}\bar{\boldsymbol{y}}$$

Next, realize code that uses the `scipy.optimize` function `minimize` and your function `negLikelihood` to determine optimal parameters $\hat{\boldsymbol{\theta}}$. Consider the following initial guess

$$\boldsymbol{\theta} = \big[1.0,\ 20.0,\ 0.5,\ 1.0\big]^\mathsf{T}$$

and **make sure that the entries of the resulting $\hat{\theta}$ will be non-negative**. To this end, set appropriate bounds when calling `minimize`.

Finally, run all your code and print the resulting vector $\hat{\boldsymbol{\theta}}$.

Note: If your code is bug free and you use `minimize` appropriately, then the result for this task should be deterministic. That is, you should get exactly the same result as your instructors.

Be aware of the following: The negative log-likelihood $-\mathcal{L}\big(\boldsymbol{\theta} \;\big|\; \boldsymbol{x}, \bar{\boldsymbol{y}}\big)$ has numerous local minima. That is, if you you consider different initial guesses than the one suggested above, you will get different optimization results. Try it out and ponder what this would mean if you were to release your code on the Web.

task 3.5 [5 points]

sampling a fitted Gaussian processes model

In this task, you are supposed to sample from the Gaussian process you fitted in the previous task.

Begin by reading the outlier-free height- and weight values into two vectors

$$\boldsymbol{x} \in \mathbb{R}^n$$
$$\boldsymbol{y} \in \mathbb{R}^n$$

Letting $\hat{\boldsymbol{\theta}} = [\hat{\theta}_1, \hat{\theta}_2, \hat{\theta}_3, \hat{\theta}_4]^\intercal$ be parameters you learned / estimated in the previous task, compute the matrix

$$\boldsymbol{C} = \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{x} \mid \hat{\theta}_1, \hat{\theta}_2, \hat{\theta}_3) + \hat{\theta}_4 \, \boldsymbol{I}$$

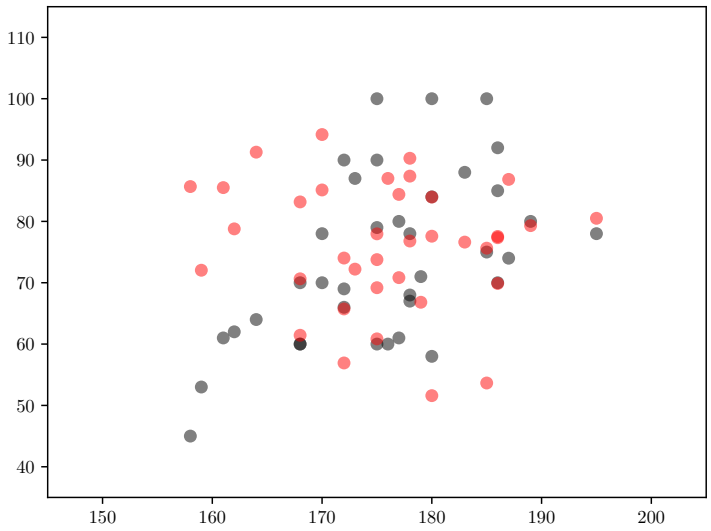
Now, work with the `numpy.random` function `multivariate_normal` to sample a vector

$$\bar{\boldsymbol{y}}' \sim \mathcal{N}\big(\boldsymbol{0}, \boldsymbol{C}\big)$$

and de-normalize it

$$\boldsymbol{y}' = \bar{\boldsymbol{y}}' + \tfrac{1}{n} \, \boldsymbol{1}\boldsymbol{1}^\intercal \boldsymbol{y}$$

Create a plot that shows all pairs (x_j, y_j) in black and all pairs (x_j, y'_j) in red. If all goes well, your plot should look something like this:



Another trick of the trade: Instead of sampling $\bar{\boldsymbol{y}}' \sim \mathcal{N}(\mathbf{0}, \boldsymbol{C})$ you can also proceed like this: Use the `numpy.linalg` function `cholesky` to compute the Cholesky factorization

$$\boldsymbol{C} = \boldsymbol{L}\boldsymbol{L}^\intercal$$

Given $\boldsymbol{L}$, compute

$$\bar{\boldsymbol{y}}' = \boldsymbol{L}\,\boldsymbol{w}$$

where $\boldsymbol{w} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.

This should be faster than sampling from $\sim \mathcal{N}(\mathbf{0}, \boldsymbol{C})$ because it avoids the need for inverting matrix $\boldsymbol{C}$ (which `multivariate_normal` tacitly does "under the hood" and can be expensive for large problem sizes n).

Given $\bar{\boldsymbol{y}}'$, de-normalize it

$$\boldsymbol{y}' = \bar{\boldsymbol{y}}' + \tfrac{1}{n}\,\mathbf{1}\mathbf{1}^\intercal\boldsymbol{y}$$

and create a plot that shows all pairs (x_j, y_j) in black and all pairs (x_j, y'_j) in red. If all goes well, your result should look similar to the above example.

task 3.6 [10 points]

predicting with a fitted Gaussian processes model

In this task, you are supposed to use the fitted Gaussian process model from task 3.4 to predict the most likely output y for any input x.

Begin by reading the outlier-free height- and weight values into two vectors

$$\boldsymbol{x} \in \mathbb{R}^n$$
$$\boldsymbol{y} \in \mathbb{R}^n$$

and normalize $\boldsymbol{y}$ to zero mean

$$\bar{\boldsymbol{y}} = \boldsymbol{y} - \tfrac{1}{n}\,\mathbf{1}\mathbf{1}^\intercal \boldsymbol{y}$$

Let $\boldsymbol{x}^* \in \mathbb{R}^N$ be a vector of new input values. Here is an exemplary *numpy* implementation of such a vector with $N = 200$ entries

```
vecXs = np.linspace(140, 210, 200)
```

Letting $\hat{\boldsymbol{\theta}} = [\hat{\theta}_1, \hat{\theta}_2, \hat{\theta}_3, \hat{\theta}_4]^\intercal$ be parameters you learned in task 3.4, compute

$$\boldsymbol{K}_{\boldsymbol{xx}} = \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{x} \mid \hat{\theta}_1, \hat{\theta}_2, \hat{\theta}_3)$$
$$\boldsymbol{K}_{\boldsymbol{x*}} = \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{x}^* \mid \hat{\theta}_1, \hat{\theta}_2, \hat{\theta}_3)$$
$$\boldsymbol{K}_{\boldsymbol{*x}} = \boldsymbol{K}(\boldsymbol{x}^*, \boldsymbol{x} \mid \hat{\theta}_1, \hat{\theta}_2, \hat{\theta}_3)$$
$$\boldsymbol{K}_{\boldsymbol{**}} = \boldsymbol{K}(\boldsymbol{x}^*, \boldsymbol{x}^* \mid \hat{\theta}_1, \hat{\theta}_2, \hat{\theta}_3)$$
$$\boldsymbol{C} = \boldsymbol{K}_{\boldsymbol{xx}} + \hat{\theta}_4\,\boldsymbol{I}$$

Next, compute the vector and the matrix

$$\bar{\boldsymbol{\mu}}^* = \boldsymbol{K}_{\boldsymbol{*x}}\,\boldsymbol{C}^{-1}\,\bar{\boldsymbol{y}}$$
$$\boldsymbol{\Sigma}^* = \boldsymbol{K}_{\boldsymbol{**}} - \boldsymbol{K}_{\boldsymbol{*x}}\,\boldsymbol{C}^{-1}\boldsymbol{K}_{\boldsymbol{x*}}$$

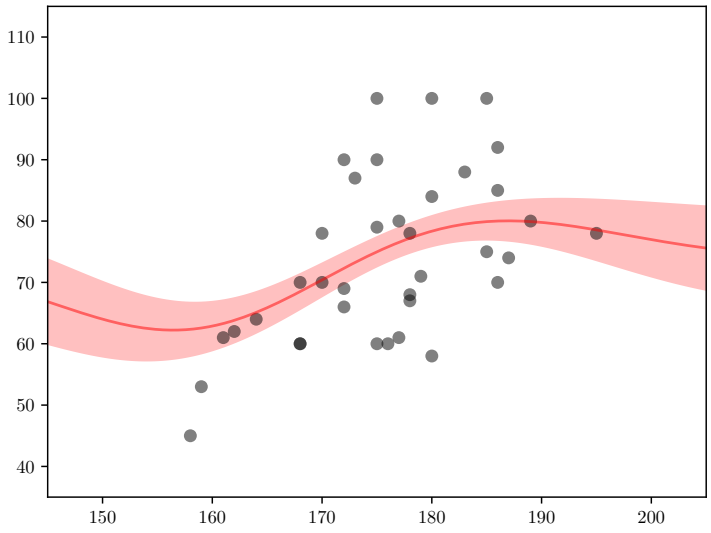
Furthermore, compute the vector

$$\boldsymbol{\sigma}^* = \sqrt{\mathrm{diag}\big[\boldsymbol{\Sigma}^*\big]}$$

Given $\bar{\boldsymbol{\mu}}^*$, de-normalize it

$$\boldsymbol{\mu}^* = \bar{\boldsymbol{\mu}} * + \tfrac{1}{n}\,\mathbf{1}\mathbf{1}^\intercal \boldsymbol{y}$$

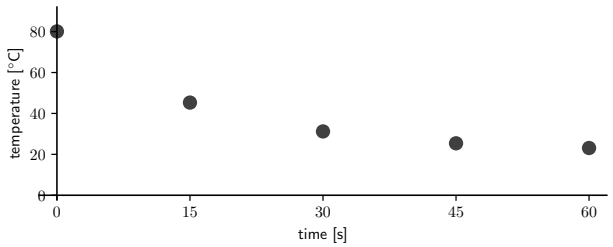
Finally, create a plot that shows all pairs (x_j, y_j) as black dots and a "line" through all pairs (x_j^*, μ_j^*) , say, in red. Also plot the one-sigma confidence interval $(x_j^*, \mu_j^* \pm \sigma_j^*)$. For the latter, you may want to use the `matplotlib` function `fill_between`. If all goes well, you plot should look something like this



task 3.7 [5 points]

yet another problem setting

The following data shows how the liquid in a (poorly isolated) cup of coffee is cooling over time.



time t [s]	temperature T [°C]
0	80.1
15	45.3
30	31.2
45	25.4
60	23.1

Use your code from the previous tasks to fit a Gaussian process regression model to this data.

For parameter optimization with `scipy.optimize.minimize`, consider the following initial guess

$$\boldsymbol{\theta} = \big[10.0,\ 4.0,\ 1.0,\ 1.0\big]^\mathsf{T}$$

Once you have fitted your model, plot the given data together with your model predictions over the time interval $0 \leq t \leq 120$. Please also plot the one-sigma confidence interval.

If you used the above initial guess, your plot should show a “strange” result.

But why would we call this result “strange”? Discuss this in your own words. Also, test if you get more reasonable results for other choices of the initial parameter guess.

task 3.8 [no points, not mandatory]

what about scikit.learn ?

This task is for those who want to know why we discouraged the naïve use of *sklearn.gaussian_process*. To make a long story short: on the Web (and in your instructors' heads), there is quite some confusion about how the *sklearn* developers chose to name and set the different kinds of parameters required for Gaussian process regression; the *sklearn* API apparently does not exactly follow the conventions in the standard textbook by Rasmusen and Williams or similar resources.

For our body height/weight setting, this does not really cause a lot of problems. Here is the best we came up with . . .

```
# minimal required imports
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF

# height / weight training data
vecX = ...
vecY = ...

# setting up matrix C
matC = 1 * RBF(length_scale=20.0, length_scale_bounds=(1e-7, 1e2))

# instantiate a Gaussian process regressor object
gp = GaussianProcessRegressor(kernel=matC,
                              alpha=1.0,
                              n_restarts_optimizer=9,
                              normalize_y=True)

# fit its parameters to the training data at hand
gp.fit(vecX.reshape(-1, 1), vecY)


# test inputs (heights)
vecXtest = np.linspace(140, 210, 200)

# expected test outputs (weights) and standard deviations
vecYmean, vecYsigm = gp.predict(vecXtest.reshape(-1,1), return_std=True)
```

If you want, experiment with this code. Especially see what happens if you choose different values for the parameters `alpha` and `normalize_y`.

task 3.9

submission of presentation and of code

As always, prepare a presentation / set of slides on your solutions for all the mandatory tasks and submit them to eCampus. Also as always, put your code into a ZIP file and submit it to eCampus.