Gaussian process regression

Exercise 03

Submitted by:
Hitik Panchal (50230156, s17hpanc)
Rouy Alfahel (3389227, s6roalfa)
Orhan Ugur Aydin (3209243, s6oraydi)
Sebastian Sauerbier (2973879, s6sesaue)
Anand Karna (50393435, s03akarn)
Khashayar Alavi (50071735, S76kalav)
Chenfang Wang, (50292921, s59cwang)
Feiyue Cheng (3305865, s6fechen)

Task 3.1 - Basic Functionalities

Goal: Implementations of diffMatrix and prodMatrix matrices;

$$[M]_{ij} = u_i - v_j \quad , \quad [M]_{ij} = u_i . v_j \quad , \quad M \in \mathbb{R}^{n_u \times n_v}$$

where,

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

! Without using loops !

Solution → Broadcasting

Implementation

```python
def diffMatrix(u,v):
    u = np.array(u)
    v = np.array(v)
    matrix = u[:, np.newaxis] - v[np.newaxis, :] # Broadcasting -> u[n_u,1] - v[1,n_v]
    return matrix

def prodMatrix(u,v):
    u = np.array(u)
    v = np.array(v)
    matrix = u[:, np.newaxis] * v[np.newaxis, :] # Broadcasting -> u[n_u,1] * v[1,n_v]
    return matrix
```

- All combinations of pairwise differences
- Outer Product $\equiv u.vT$

```
u = [1,2,3,4,5]
v = [5,10,15,20]
```

```
Difference Matrix:
[[ -4  -9 -14 -19]
 [ -3  -8 -13 -18]
 [ -2  -7 -12 -17]
 [ -1  -6 -11 -16]
 [  0  -5 -10 -15]]
```

```
Product Matrix:
[[  5  10  15  20]
 [ 10  20  30  40]
 [ 15  30  45  60]
 [ 20  40  60  80]
 [ 25  50  75 100]]
```

Task 3.2 - Kernel Matrices

Goal: Implementations of Linear Kernel matrix

$$K = K(u, v|\alpha) \in \mathbb{R}^{n_u \times n_v} \quad [K]_{ij} = \alpha . u_i . v_j$$

and Gauss Kernel matrix

$$K = K(u, v|\alpha, \sigma) \in \mathbb{R}^{n_u u \times n_v} \quad [K]_{ij} = \alpha . \exp(-\frac{(u_i - v_j)^2}{2\sigma^2})$$

! using diffMatrix and prodMatrix

Implementation

```python
def linearKernelMatrix(u,v,alpha):
    u, v = np.array(u), np.array(v)
    prod = prodMatrix(u,v)
    matrix = alpha * prod
    return matrix

def gausKernelMatrix(u,v,alpha,sigma):
    u, v = np.array(u), np.array(v)
    diff = diffMatrix(u,v)
    matrix = alpha * np.exp(- (diff ** 2) / (2 * sigma ** 2))
    return matrix
```

- Compute similarity between inputs via kernels
- Global vs local similarity

```
u = [1,2,3,4,5]
v = [5,10,15,20]
alpha = 2.0
sigma = 3.0
```

```
Linear Kernel Matrix:
[[ 10.  20.  30.  40.]
 [ 20.  40.  60.  80.]
 [ 30.  60.  90. 120.]
 [ 40.  80. 120. 160.]
 [ 50. 100. 150. 200.]]
```

```
Gaussian Kernel Matrix:
[[8.22224581e-01 2.22179931e-02 3.73289382e-05 3.89953557e-09]
 [1.21306132e+00 5.71310016e-02 1.67296694e-04 3.04599595e-08]
 [1.60147481e+00 1.31457057e-01 6.70925256e-04 2.12907428e-07]
 [1.89191894e+00 2.70670566e-01 2.40771999e-03 1.33167229e-06]
 [2.00000000e+00 4.98704418e-01 7.73184028e-03 7.45330634e-06]]
```

Task 3.3 - Sampling Simple Gaussian Processes

Goal: Sample vectors (functions) from a multivariate Gaussian distribution

$$N(0, K)$$

using different kernel matrices to visualize the behavior imposed by each kernel

$$K_L = K(x, x|\alpha_L, \sigma_L)\ ,\ K_G = K(x, x|\alpha_G, \sigma_G)\ ,\ K_L + K_G$$

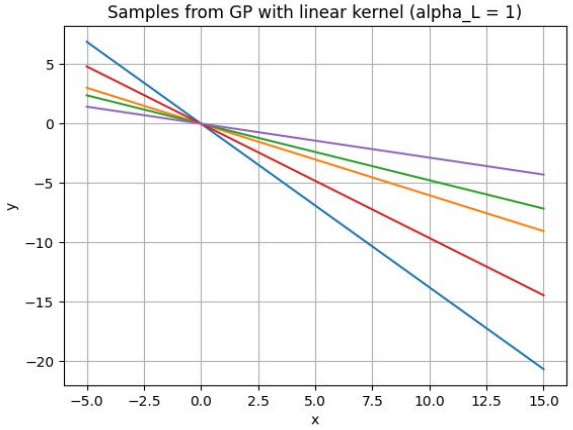
! All kernel matrices are square

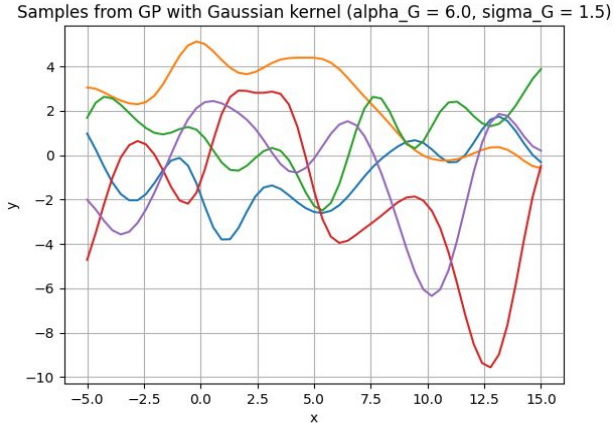
! Implementation achieved by
multivariate_normal function from numpy.random

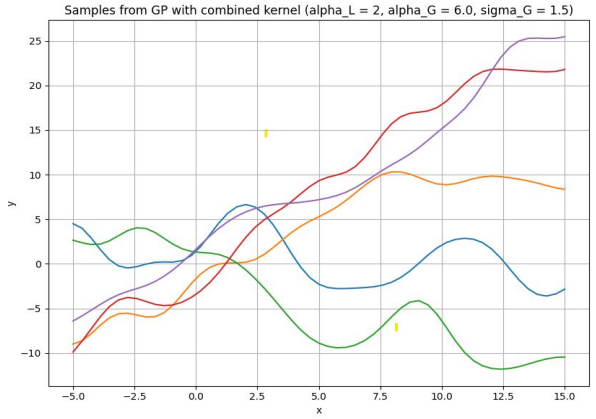
```
x = np.linspace(-5.0, 15.0, 55)
n = len(x)
mean = np.zeros(n)
```

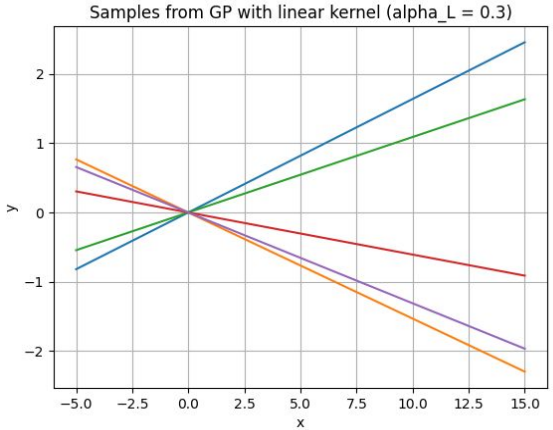
```
num_samples = 5
Y = np.random.multivariate_normal(mean=mean, cov=K_l, size=num_samples)
```

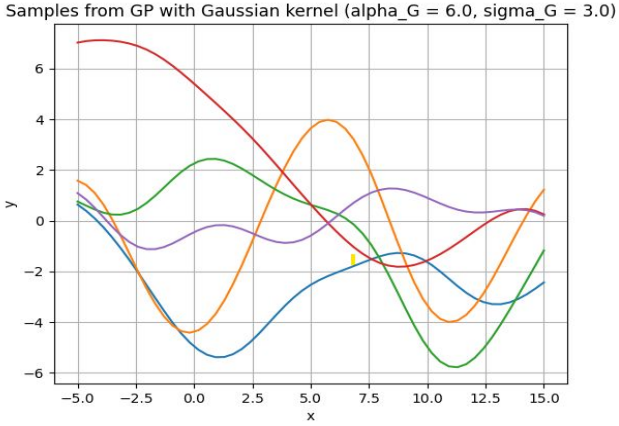
Task 3.3 - Sampling Simple Gaussian Processes (Implementation)

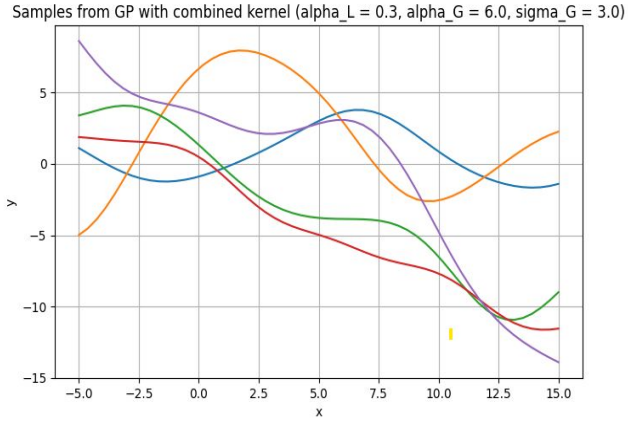












Task 3.4 Fitting a Gaussian Processes Model to Data

Problem Setup

We are given a dataset:

- Inputs: height (in cm) $\mathbf{x} = (x_i),\ i = 1, \ldots, n$
- Targets: weight (in kg) $\mathbf{y} = (y_i),\ i = 1, \ldots, n$

centralize y and get

$$\bar{\mathbf{y}} = \mathbf{y} - \frac{1}{n}\mathbf{1}\mathbf{1}^\top \mathbf{y}$$

Gaussian Process model

$$\mathbf{y} \sim \mathcal{N}(0, \boldsymbol{C}(\boldsymbol{\theta})), \qquad \boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3, \theta_4)$$

$$\boldsymbol{C}(\boldsymbol{\theta}) = \boldsymbol{K_{xx}} + \theta_4 \boldsymbol{I_n}, \qquad [\boldsymbol{K_{xx}}]_{ij} = \theta_1 \exp\left(-\frac{(x_i - x_j)^2}{2\theta_2^2}\right) + \theta_3 x_i x_j$$

The kernel $\boldsymbol{K_{xx}}$ used here is a linear combination of gaussian kernel and linear kernel

Predictive Distribution

Let $\boldsymbol{x}^*$ denote the test inputs, and $\boldsymbol{K_{xx^*}}$, $\boldsymbol{K_{x^*x}}$, $\boldsymbol{K_{x^*x^*}}$ defined analogously to $\boldsymbol{K_{xx}}$.
Predictive distribution of $\bar{\boldsymbol{y}}^* \mid \boldsymbol{x}, \boldsymbol{x}^*, \bar{\boldsymbol{y}} \sim \mathcal{N}(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*)$.

$$\begin{aligned}\boldsymbol{\mu}^* &= \mathbb{E}[\bar{\boldsymbol{y}}^* \mid \boldsymbol{x}, \boldsymbol{x}^*, \bar{\boldsymbol{y}}] \\ &= \boldsymbol{K}_{x^*x}\, \boldsymbol{C}(\boldsymbol{\theta})^{-1}\, \bar{\boldsymbol{y}} \\ \boldsymbol{\Sigma}^* &= \boldsymbol{K}_{x^*x^*} - \boldsymbol{K}_{x^*x}\, \boldsymbol{C}(\boldsymbol{\theta})^{-1}\, \boldsymbol{K}_{xx^*}\end{aligned}$$

$$\mathbb{E}[\boldsymbol{y}^* \mid \boldsymbol{x}, \boldsymbol{x}^*, \bar{\boldsymbol{y}}] = \boldsymbol{\mu}^* + \frac{1}{n}\mathbf{1}\mathbf{1}^\top \mathbf{y}$$

$$\mathrm{Var}(y_i^*) = [\boldsymbol{\Sigma}^*]_{ii}$$

Hyperparameter Optimization:

$$-\mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{x}, \bar{\mathbf{y}}) = \frac{1}{2}\bar{\mathbf{y}}^\top \boldsymbol{C}(\boldsymbol{\theta})^{-1}\bar{\mathbf{y}} + \frac{1}{2}\log\det \boldsymbol{C}(\boldsymbol{\theta}).$$

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \ \big(-\mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{x}, \bar{\mathbf{y}})\big),$$

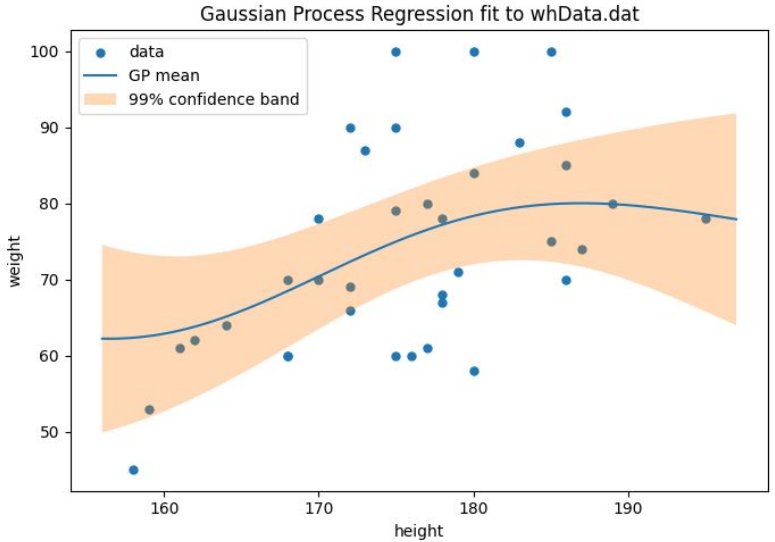
In the implementation, we:

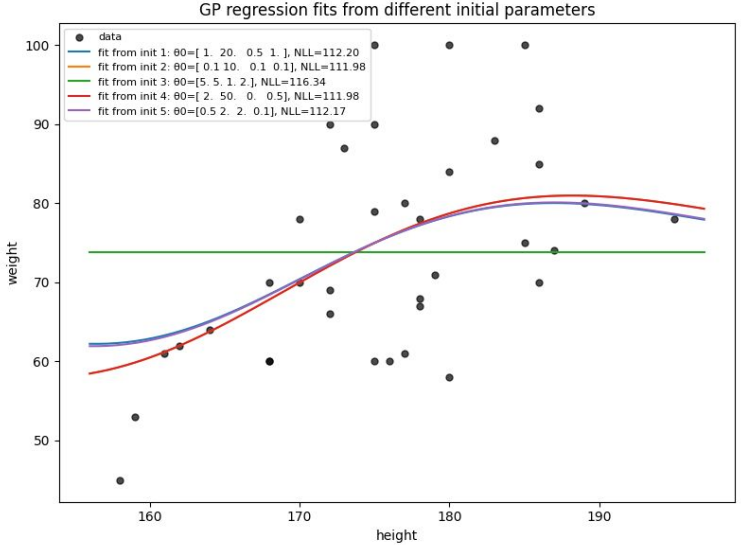
- build $C(\theta)$ using the kernel,
- use `np.linalg.slogdet` for $\log\det C$,
- use `np.linalg.solve` instead of explicitly computing C^{-1},
- call `scipy.optimize.minimize` with bounds to ensure $\theta_i \geq 0, i = 1, 3, 4$ and $\theta_2 > 0$.

Initial Parameters: $\boldsymbol{\theta} = (1.0, 20.0, 0.5, 1.0)$

Estimated Parameters: $\hat{\boldsymbol{\theta}} = (57.0622, 12.3984, 0.0000, 139.5032)$

Task 3.4 Fitting a Gaussian Processes Model to Data





The neg-likelihood funktion not globally convex. Different initial hyperparameters may lead the optimizer to different local minima, which correspond to qualitatively different explanations of the data. Solutions with lower NLL values tend to be more explanatory. When releasing codes on Web, reasonable initial guess of the parameters coming from the knowledge of the background problem should be suggested.

Task 3.5: Sampling from a Fitted Gaussian Process Model

Goal

Generate synthetic weight samples from the fitted Gaussian Process (GP) model and compare them with the original height–weight data to check whether the GP captures the underlying data structure.

Approach

- Loaded height-weight data and removed outliers: $x \in \mathbb{R}^n$, $y \in \mathbb{R}^n$.
- Built covariance matrix $C = K(x, x|\hat{\theta}_1, \hat{\theta}_2, \hat{\theta}_3) + \hat{\theta}_4 I$ using the $\hat{\theta}$ from Task 3.4
- Generated GP samples using two methods `np.random.multivariate_normal` and Cholesky: $C = LL^\top$, $\tilde{y} = Lw$, $w \sim \mathcal{N}(0, I)$
- De-normalized via $y' = \tilde{y} + \frac{1}{n}\mathbf{1}\mathbf{1}^\top y$
- Plotted original data in black and GP samples in red.

Code snippet:

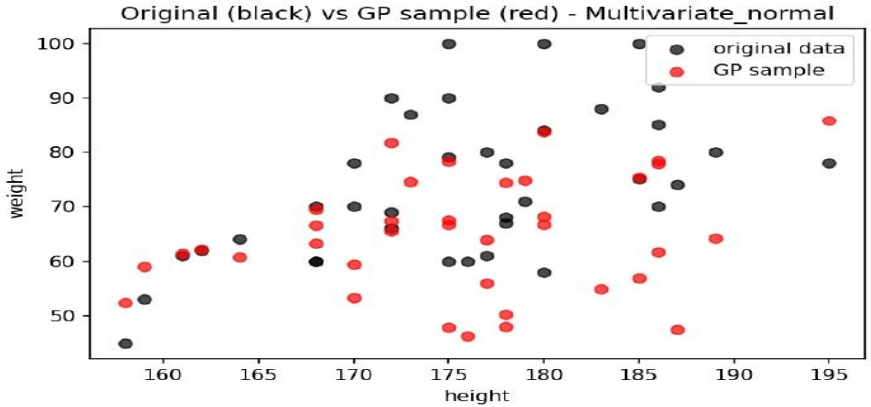
```python
# Kernel and covariance matrix

def kernel_matrix(x: np.ndarray, theta1: float, theta2: float, theta3: float) -> np.ndarray:
    """
        [K]_ij = theta1 * exp(-(xi - xj)^2 / (2 theta2^2)) + theta3 * xi * xj
    """
    x = x.reshape(-1, 1)
    dists2 = (x - x.T) ** 2
    K = theta1 * np.exp(-dists2 / (2.0 * theta2 ** 2))
    K += theta3 * (x @ x.T)
    return K


def covariance_matrix(x: np.ndarray, theta: np.ndarray) -> np.ndarray:
    """
    C = K + theta4 * I
    """
    theta1, theta2, theta3, theta4 = theta
    K = kernel_matrix(x, theta1, theta2, theta3)
    n = len(x)
    C = K + theta4 * np.eye(n)
    return C
```

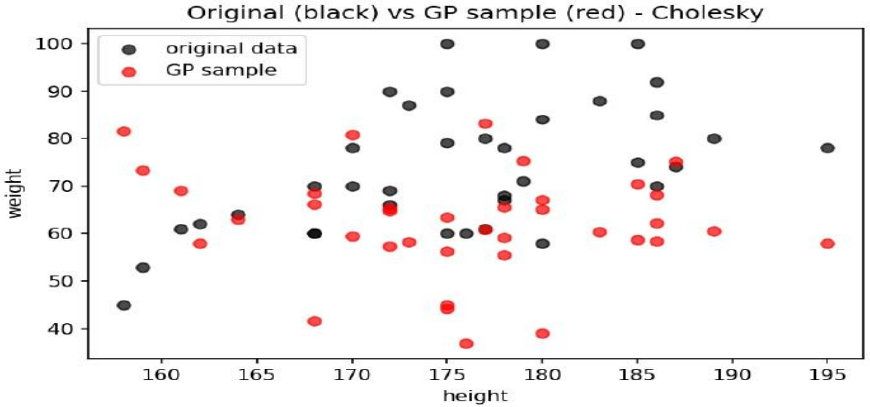
```python
# Generate the covariance matrix with the new theta_hat

C = covariance_matrix(x, theta_hat)
```

```python
# Sample using multivariate_normal

np.random.seed(42)

n = len(x)                          # number of data points
mean = np.zeros(n)                  # 0 vector in R^n

y_tilde_sample = np.random.multivariate_normal(mean=mean, cov=C)

# de-normalize: y' = ỹ' + 1/n * 11^T y
y_sample = y_tilde_sample + y_mean
```

```python
# Sample using Cholesky

L = np.linalg.cholesky(C)    # C = L L^T
w = np.random.randn(n)       # w ~ N(0, I)
y_tilde_sample = L @ w       # ỹ' ~ N(0, C)

# de-normalize: y' = ỹ' + 1/n * 11^T y
y_sample = y_tilde_sample + y_mean
```

Results and Observations





Observations and Learnings:

- The samples preserve the positive relationship between height and weight, showing that the learned covariance structure is applied correctly.
- Variance and spread of the red points match the real data, especially in the central height range where most observations lie.
- Differences between the two plots are due to independent random draws; both methods sample from the same $\mathcal{N}(0, C)$ distribution.
- The Cholesky-based sample looks slightly more structured, while the multivariate sample shows marginally higher local variability.
- Overall, the GP generates realistic, statistically coherent synthetic data, confirming the correctness of the fitted model and sampling implementation.

Task 3.6: predicting with a fitted Gaussian processes model

➔ Goal: Predict weight from height and show the uncertainty of the prediction using a Gaussian Process (GP)

```python
# --- 3) Kernel: RBF(theta1, theta2) + Linear(theta3) ---
def K_mix(a, b, theta1, theta2, theta3):
    A = a.reshape(-1, 1)
    B = b.reshape(1, -1)
    sq = (A - B) ** 2
    rbf = theta1 * np.exp(-sq / (2.0 * theta2 ** 2))
    linear = theta3 * (A * B)  # elementwise x_i * x_j
    return rbf + linear
```

Data: File whData.dat

- ➔ Inputs: height [cm] → x
- ➔ Targets: weight [kg] → y
  - Preprocessing: Subtract the mean of y (center) so the model doesn't need an extra bias term
- ➔ This function tells the GP how similar two heights x_i, x_j are

a.The **kernel** in `K_mix(...)` is the sum of an RBF part and a linear part, so the GP can model a smooth shape and an overall trend at the same time
b.The **RBF** term `theta1 * exp(-(A-B)**2 / (2*theta2**2))` makes nearby heights behave similarly, where **theta1** sets how large the variations can be and **theta2** controls how smooth the curve is
c.The **linear** term `theta3 * (A * B)` adds a simple global height→weight trend
d.We also add `+ theta4 * I` later to **model measurement noise** and natural variability in the data

Task 3.6: predicting with a fitted Gaussian processes model

•**what I did:**

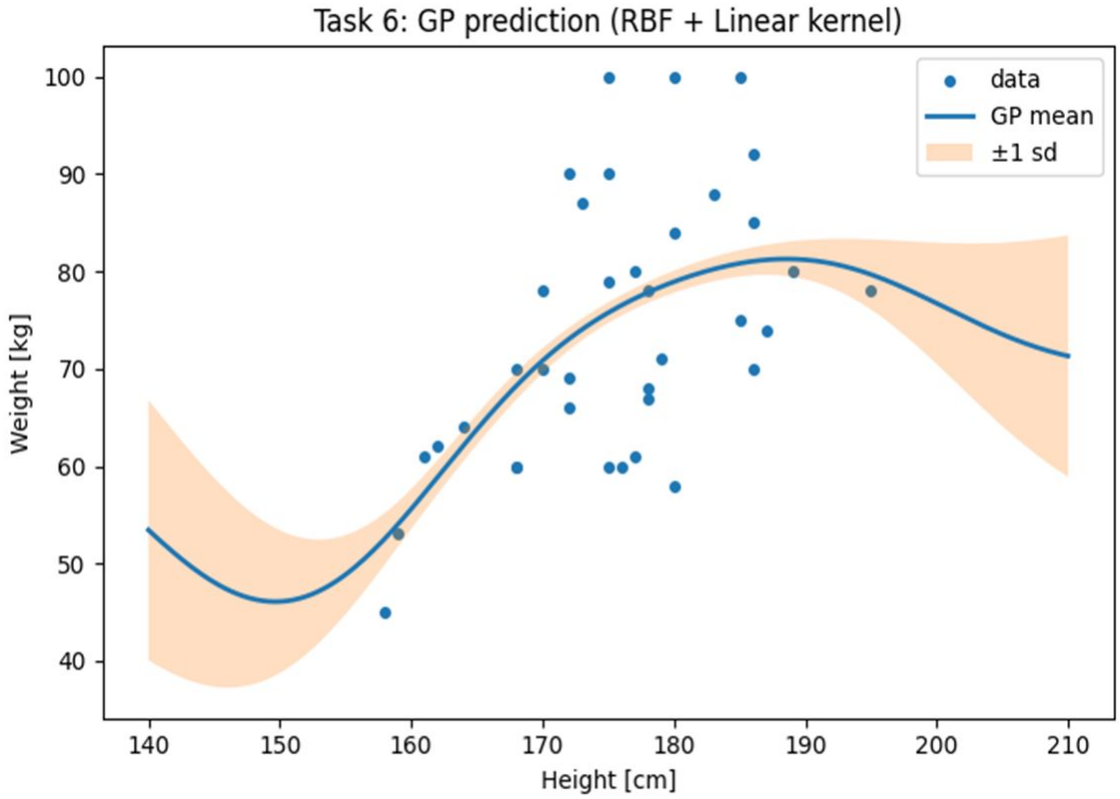
i.Load and clean the data, then **center y** (remove its mean)

ii.Use a **GP with RBF + Linear** kernel (K_mix in the code)

iii.Pick hyperparameters **θ = (θ1, θ2, θ3, θ4)** by **random search** that minimizes the negative log marginal likelihood

iv.Compute the **posterior** on heights 140–210 cm: mean **μ(x)** and uncertainty **σ(x)** using **Cholesky** solves.

v.Plot data points, the **GP mean** curve, and the **±1 sd** band




```
Hyperparameters (theta1, theta2, theta3, theta4):
248.88091949767613 14.211464028160336 0.0007364939914934126 20.578300634680105
```

Task 3.7: Fitting a Gaussian process regression model to the data

Goal: To predict the temperature trend beyond the measured time (extrapolation up to t=120s).

The Approach:

- Data Processing
  - Input vector **x**: Time in seconds (0 to 60).
  - Target vector **y**: Temperature in °C.
- **The Model**
  - Used a composite Kernel function:
  - **Components:** Radial Basis Function (RBF) for local smoothness + Linear Kernel for global trends + White

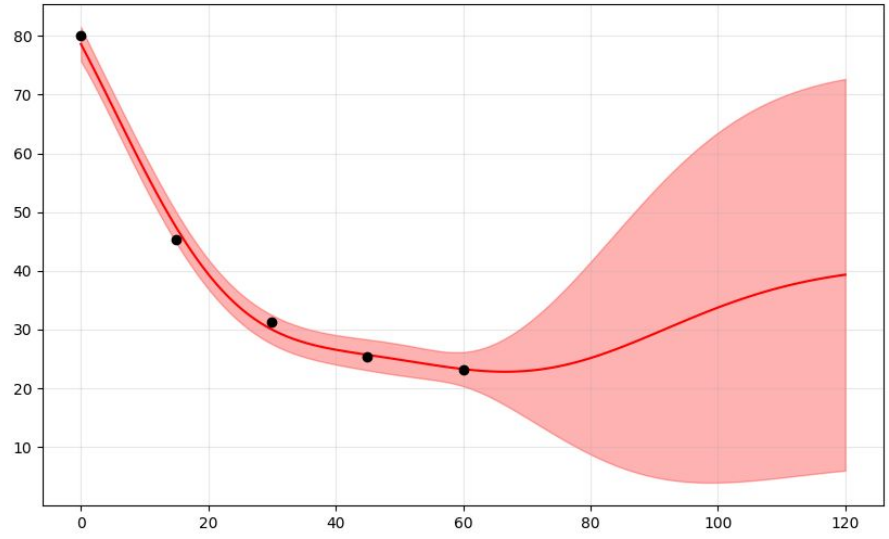
$$K(x, x') = \theta_1 \exp\left(-\frac{(x - x')^2}{2\theta_2^2}\right) + \theta_3(x \cdot x') + \theta_4 I$$

- **Optimization**: Maximized the Log-Likelihood using scipy.optimize.minimize (L-BFGS-B) to find optimal hyperparameters.

```
def compute_kernel_matrix(u, v, theta):
    # Extract parameters: Amplitude, Length-scale, Linear-scale
    theta1, theta2, theta3 = theta[0], theta[1], theta[2]

    # Vectorized RBF (Gaussian) Part
    diff = u[:, np.newaxis] - v[np.newaxis, :]
    K_rbf = theta1 * np.exp(-(diff**2) / (2 * theta2**2))

    # Vectorized Linear Part (The cause of the "strange" result)
    K_lin = theta3 * (u[:, np.newaxis] * v[np.newaxis, :])

    return K_rbf + K_lin
```

Task 3.7: Fitting a Gaussian process regression model to the data

Results



Conclusion: For physical processes like cooling, "standard" zero-mean normalization can be misleading. We should ideally normalize around the **asymptote** (room temperature,≈ 20℃) rather than the data mean.

Optimized Parameters: . $\hat{\theta} \approx [1129.7, 25.9, \mathbf{0.0}, 8.9]$
The Linear Term: The optimizer set $\theta_3 \approx 10^{-5}$, effectively disabling the linear kernel to avoid predicting a crash to negative temperatures.

Observations:

- **The Fit (0 < t < 60):** The model captures the exponential decay perfectly within the training data.
- **The Prediction (t > 60):** The prediction curve (red line) dips to≈ 22℃ but then strangely **rises back up** to≈ 40℃ as time progresses. The confidence interval (pink) explodes.

Key Learning:

- **Reversion to the Mean:** The RBF kernel is a "local" kernel. As we move far away from the training data (measurements), the correlation drops to zero.
- **The Normalization Trap:** When the kernel correlation fades, the GP predicts the **prior mean**.
    - We normalized the data based on the *average of the observed samples* ($\text{mean}([80, 45, 31, 25, 23]) \approx 41$℃).
    - Therefore, in the absence of data, the model assumes the coffee will warm back up to the "average temperature" of 41℃.