



UNIVERSITÀ
DI TRENTO

Dipartimento di
Matematica

Bachelor's Degree in Mathematics

GAUSSIAN PROCESSES AND SUPERVISED
LEARNING: APPLICATION TO THE
WINDKESSEL MODEL

Candidate:
Stefano Costa

Supervisor:
Prof. Lucas Omar Müller

Academic Year 2021-22

Acknowledgements

I thank my thesis advisor, Professor Lucas Omar Müller, for his patience, guidance and support. I have benefited greatly from his knowledge and advice, without which this thesis would not have been possible.

Due thanks also go to Dr. Christian Contarino, who was not only crucial in choosing and developing the topic, but also gave me constant moral support.

Contents

	Pag.
List of codes	viii
List of figures	xiv
List of tables	xv
I Introduction	1
I.1 Topics covered and motivation	2
I.2 Organization of the work	5
II Gaussian distribution	7
II.1 Univariate Gaussian distribution	8
II.2 Multivariate Gaussian distribution	9
II.3 Correlation for multivariate vectors	11
III Processi gaussiani	17
III.1 Definizione e motivazione	18
III.2 Introduzione ai processi gaussiani e alla notazione	20
III.3 Sulle covariance function	24
III.4 Predizioni con osservazioni senza rumore	44
III.5 Predizioni con osservazioni rumorose	47
IV Machine learning	51
IV.1 Introduzione al machine learning	52
IV.2 Dataset	52
IV.3 Modelli parametrici e non parametrici	55
IV.4 Ottimizzazione degli iperparametri	57
IV.5 Metodo di ottimizzazione	61
V Modello Windkessel	69
V.1 Introduzione	70
V.2 Concetti di fisiologia	72
V.3 Modello Windkessel a due elementi	79
V.4 Forzante periodica	83
V.5 Analisi di sensitività locale	86

VI Metodologia e risultati training	91
VI.1 Problema di regressione	92
VI.2 Dettagli del training	92
VI.3 Risultati del training	95
VI.4 Tempo di esecuzione: approssimazione di MAP, DBP, SBP, PP	124
VII Conclusions and future directions	125
VII.1 Conclusions	126
VII.2 Future directions	127
Bibliography	131
Appendice	135

List of codes

VII.1	Import necessario per la generazione di immagini dei kernel introdotti nel capitolo III	136
VII.2	Import necessario per la generazione di immagini sulla predi- zione del capitolo III	136
VII.3	Definizione della media cubica	136
VII.4	Codice per generare la figura III.3	137
VII.5	Codice per generare la figura III.4	138
VII.6	Codice per generare la figura III.5	138
VII.7	Codice per generare la figura III.6	139
VII.8	Codice per generare la figura III.7	139
VII.9	Codice per generare la figura III.8	140
VII.10	Codice per generare la figura III.9	140
VII.11	Codice per generare la figura III.10	141
VII.12	Codice per generare la figura III.11	141
VII.13	Codice per generare la figura III.12	142
VII.14	Codice per generare la figura III.13	142
VII.15	Codice per generare la figura III.14	143
VII.16	Codice per generare la figura III.15	144
VII.17	Codice per generare la figura III.16	144
VII.18	Codice per generare la figura III.17	145
VII.19	Codice per generare la figura III.18	145
VII.20	Codice per generare la figura III.19	146
VII.21	Codice per generare la figura III.20	146
VII.22	Codice per generare la figura III.22	147
VII.23	Codice per generare la figura III.23	147
VII.24	Codice per generare la figura III.24	148
VII.25	Codice per generare la figura III.25	148
VII.26	Codice per generare la figura III.30	149
VII.27	Codice per generare la figura III.31	150
VII.28	Codice per generare la figura III.32	151
VII.29	Codice per generare la figura III.33	152
VII.30	Codice per importare le librerie necessarie	153
VII.31	Codice per il plot dei dati reali	153
VII.32	Codice per il calcolo della resistenza periferica totale	153
VII.33	Codice per la definizione della ODE	153
VII.34	Codice per definire la funzione di flusso	154

VII.35	Codice per la definizione della funzione f_C	154
VII.36	Plot soluzione del modello "semplice"	155
VII.37	Codice per generare la figura V.9.	155
VII.38	Codice per la stima di C che minimizza f_C	155
VII.39	Codice per generare la figura V.10	156
VII.40	Codice per la definizione della funzione $f_{C,\alpha}$	156
VII.41	Codice per la stima di C e α	157
VII.42	Codice per generare la figura V.11.	157
VII.43	Codice per il metodo delle differenze finite centrate riadattato al calcolo della sensitività locale.	158
VII.44	Codice per il calcolo della sensitività di C	158
VII.45	Codice per il calcolo della sensitività di R_1	159
VII.46	Codice per il calcolo della sensitività di R_2	159
VII.47	Codice per il calcolo della sensitività di P_d	160
VII.48	Codice per definire la funzione di flusso per il modello Wind- kessel ciclico	160

List of Figures

II.1	Distribution function and probability density of a standard normal distribution [Wik22b].	8
II.2	Point cloud generated by a bivariate Gaussian distribution with uncorrelated components [Wil20].	11
II.3	Point cloud generated by a bivariate Gaussian distribution with correlated components [Wil20].	12
II.4	Point cloud generated by a bivariate Gaussian distribution with strongly correlated components [Wil20].	13
II.5	Two different visual approaches to correlation of components of multivariate Gaussian vectors: $n = 2$ [Wil20].	14
II.6	Visualization using segments of the correlation of components of multivariate Gaussian vectors: $n = 5$ [Wil20].	15
II.7	Weak and strong correlation of components of multivariate Gaussian vectors visualized by segments [Dam16].	16
II.8	Visualization by segments of the correlation of components of multivariate Gaussian vectors: $n = 50$ [Wil20].	16
III.1	Regressione nonlineare [Tur16].	18
III.2	Regressione nonlineare con processi gaussiani [Tur16].	19
III.3	Quattro vettori generati da un processo gaussiano definito come nell'esempio III.2.11. Codice VII.4.	23
III.4	Grafico di $k(x, x')$ linear kernel, $\sigma_b^2 = 1$, $\sigma_v^2 = 1$, $c = -1$ e $x' = 1$. Codice VII.5	25
III.5	Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è il linear kernel e $\sigma_b^2 = 0$, $\sigma_v^2 = 1$, $c = 0$. Codice VII.6.	26
III.6	Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è il linear kernel e $\sigma_b^2 = 0$, $\sigma_v^2 = 1$, il parametro c viene variato. Codice VII.7.	27
III.7	Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è il linear kernel e $\sigma_v^2 = 1$, $c = 0$, il parametro σ_b^2 viene variato. Codice VII.8.	27
III.8	Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è il linear kernel e $\sigma_b^2 = 0$, $c = 0$, il parametro σ_v^2 viene variato. Codice VII.9.	28

III.9	Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(m, k)$ dove $m(x) = x^3$ e $k(x, x')$ è il linear kernel, $\sigma_b^2 = 1$, $\sigma_v^2 = 10$, $c = 0$. Codice VII.10.	28
III.10	Grafico di $k(x, x')$ squared-exponential kernel, $\sigma^2 = 1$ e $l^2 = 1$. Codice VII.11.	29
III.11	Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è lo squared-exponential kernel e $l^2 = 1$, $\sigma^2 = 1$. Codice VII.12.	30
III.12	Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è lo squared-exponential kernel e $l^2 = 1$, il parametro σ^2 viene variato. Codice VII.13.	31
III.13	Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è lo squared-exponential kernel e $\sigma^2 = 1$, il parametro l^2 viene variato. Codice VII.14.	31
III.14	Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(m, k)$ dove $m(x) = x^3$ e $k(x, x')$ lo squared-exponential kernel, $\sigma^2 = 7$ e $l = 0.3$. Codice VII.15.	32
III.15	Grafico di $k(x, x')$ periodic kernel, $\sigma^2 = 1$, $l^2 = 1$, $p = 2$. Codice VII.16.	33
III.16	Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è il periodic kernel e $\sigma^2 = 1$, $l^2 = 2$, $p = 1$. Codice VII.17.	34
III.17	Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è il periodic kernel e $l^2 = 1$, $p = 1$, il parametro σ^2 viene variato. Codice VII.18.	34
III.18	Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è il periodic kernel e $\sigma^2 = 1$, $l^2 = 1$ e il parametro p viene variato. Codice VII.19.	35
III.19	Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è il periodic kernel e $\sigma^2 = 1$, $p = 1$ e il parametro l^2 viene variato. Codice VII.20.	35
III.20	Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(m, k)$ dove $m(x) = x^3$ e $k(x, x')$ il periodic kernel, $\sigma^2 = 8$, $l^2 = 1$ e $p = 0.5$. Codice VII.21.	36
III.21	Grafico di funzione con distribuzione $f \sim \mathcal{GP}(0, k)$ con $k(x, x')$ lo squared-exponential kernel in due dimensioni in cui $M = \text{diag}(1, 3)^{-2}$. La funzione tende a cambiare più velocemente lungo la direzione x_1 che lungo la direzione x_2 . [Mur12]	37
III.22	Grafico di $k(x, x')$ squared-exponential kernel sommato a periodic kernel. $\sigma^2 = 1$, $l = 2$, $p = 1$. Codice VII.22.	39
III.23	Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è lo squared-exponential sommato al periodic kernel e $l^2 = 1.1$, $\sigma^2 = 1$, $p = 1.1$. Codice VII.23.	39
III.24	Grafico di $k(x, x')$ linear kernel moltiplicato a linear kernel. $\sigma_b^2 = 0$, $\sigma_v^2 = 1$, $c = 0$, $x' = 1$. Codice VII.24.	40
III.25	Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è il linear kernel moltiplicato al linear kernel e $\sigma_b^2 = 0$, $\sigma_v^2 = 1$, $c = 0$. Codice VII.25.	40

III.26	545 osservazioni delle medie mensili della concentrazione atmosferica di CO_2 tra il 1958 e il 2003, inoltre viene mostrata la regione di confidenza del 95% per un modello di regressione di processo gaussiano a 20 anni nel futuro. [RW06]	41
III.27	Comparazione della predizione della concentrazione di CO_2 con i dati reali fino a maggio 2022.	42
III.28	Comparazione della predizione della concentrazione di CO_2 con i dati reali fino dal 1995 a maggio 2022.	43
III.29	Spiegazione grafica di come viene incorporata la conoscenza a priori [GKD19].	44
III.30	Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(\cdot, \cdot)$ è il squared-exponential kernel; il processo gaussiano è stato condizionato per interpolare sei punti. Viene mostrata in rosso la funzione da cui sono stati scelti i punti da interpolare, in blu la media del processo gaussiano condizionato, come linee tratteggiate alcuni sample del processo gaussiano. Codice VII.26.	46
III.31	Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(\cdot, \cdot)$ è il squared-exponential kernel; il processo gaussiano è stato condizionato per interpolare sei punti. Viene mostrata in blu la regione di confidenza al 95%, in blu la media del processo gaussiano condizionato e come linee tratteggiate alcuni sample. Codice VII.27.	47
III.32	Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(\cdot, \cdot)$ è il squared-exponential kernel; il processo gaussiano è stato condizionato per predire una funzione a partire dalle sue osservazioni rumorose. Viene mostrata in rosso la funzione da predire, in rosso i punti osservati della funzione con le barre rappresentanti il rumore, in blu la media del processo gaussiano condizionato, come linee tratteggiate alcuni sample del processo gaussiano. Codice VII.28.	48
III.33	Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(\cdot, \cdot)$ è il squared-exponential kernel; il processo gaussiano è stato condizionato per predire una funzione a partire dalle sue osservazioni rumorose. Viene mostrata in blu la regione di confidenza al 95%, in rosso con le barre di errore le osservazioni, in blu la media del processo gaussiano condizionato e come linee tratteggiate alcuni sample. Codice VII.29.	49
IV.1	Esempio di overfitting. I dati (approssimativamente lineari) sono approssimati da una funzione lineare e da una polinomiale. Anche se la funzione polinomiale fornisce un adattamento quasi perfetto, ci si può aspettare che la funzione lineare generalizzi meglio i dati. [Wik22d]	53

IV.2	Overfitting nell'apprendimento supervisionato. Il training error (errore sul training set) è mostrato in blu, il validation error (errore su validation set) in rosso, entrambi in funzione del numero di cicli di training. [Wik22d]	54
IV.3	Illustrazione del metodo del gradiente su degli insiemi di livello, ad ogni iterazione viene aggiornato il learning rate.[Wik22c]	63
IV.4	Metodo dei momenti applicato al metodo stocastico del gradiente. [Rud16]	65
IV.5	Comparazione di metodi di minimizzazione di una cost function in una rete neurale. [KB17]	68
V.1	Pressione sistemica e flusso aortico misurati in un paziente. Codice VII.31.	70
V.2	Grafico della pressione reale e output del modello semplice. Codice VII.36.	71
V.3	Anatomia del cuore umano [Wik22a].	72
V.4	Diagrammi che riassumono la sistole e la diastole di un cuore umano [Wik22a].	73
V.5	Esempio di diagramma di Wiggers [Wik21a].	75
V.6	Illustrazione dell'analogia sull'effetto windkessel [Wik21b]. . . .	77
V.7	Illustrazione dell'effetto windkessel [AW20].	78
V.8	Forma circuitale del modello Windkessel a due elementi. . . .	81
V.9	Grafico di f_C . Codice VII.37.	82
V.10	Grafico della soluzione approssimata dell'equazione (3) con C stimata. Codice VII.39.	82
V.11	Grafico della soluzione approssimata dell'equazione (3) con $C = 2,11579mL/mmHg$ e $\alpha = 0,97134$. Codice VII.42. . . .	83
V.12	Grafico della soluzione dell'equazione (3) approssimata dopo venti cicli cardiaci con $C = 2,03424mL/mmHg$ e $\alpha = 0,97354$. .	85
VI.1	Distribuzione dei dati nel database.	94
VI.2	MAP: andamento del training e validation loss, early stopper, R2Score e MSE.	95
VI.3	MAP: predizioni sui dati di input.	96
VI.4	Dipendenza di MAP da C sull'intervallo di training e due intervalli attigui.	97
VI.5	Dipendenza di MAP da C sull'intervallo di training.	97
VI.6	Dipendenza di MAP da C sull'intervallo attiguo a sinistra dell'intervallo di training.	98
VI.7	Dipendenza di MAP da C sull'intervallo attiguo a destra dell'intervallo di training.	98
VI.8	Dipendenza di MAP da $R1$ sull'intervallo di training e due intervalli attigui.	99
VI.9	Dipendenza di MAP da $R1$ sull'intervallo di training.	99
VI.10	Dipendenza di MAP da $R1$ sull'intervallo attiguo a sinistra dell'intervallo di training.	100

VI.11	Dipendenza di MAP da R_1 sull'intervallo attiguo a destra dell'intervallo di training.	100
VI.12	Dipendenza di MAP da R_2 sull'intervallo di training e due intervalli attigui.	101
VI.13	Dipendenza di MAP da R_2 sull'intervallo di training.	101
VI.14	Dipendenza di MAP da R_2 sull'intervallo attiguo a sinistra dell'intervallo di training.	102
VI.15	Dipendenza di MAP da R_2 sull'intervallo attiguo a destra dell'intervallo di training.	102
VI.16	DBP: andamento del training e validation loss, early stopper, R2Score e MSE.	103
VI.17	DBP: predizioni sui dati di input.	103
VI.18	Dipendenza di DBP da C sull'intervallo di training e due intervalli attigui.	104
VI.19	Dipendenza di DBP da C sull'intervallo di training.	104
VI.20	Dipendenza di DBP da C sull'intervallo attiguo a sinistra dell'intervallo di training.	105
VI.21	Dipendenza di DBP da C sull'intervallo attiguo a destra dell'intervallo di training.	105
VI.22	Dipendenza di DBP da R_1 sull'intervallo di training e due intervalli attigui.	106
VI.23	Dipendenza di DBP da R_1 sull'intervallo di training.	106
VI.24	Dipendenza di DBP da R_1 sull'intervallo attiguo a sinistra dell'intervallo di training.	107
VI.25	Dipendenza di DBP da R_1 sull'intervallo attiguo a destra dell'intervallo di training.	107
VI.26	Dipendenza di DBP da R_2 sull'intervallo di training e due intervalli attigui.	108
VI.27	Dipendenza di DBP da R_2 sull'intervallo di training.	108
VI.28	Dipendenza di DBP da R_2 sull'intervallo attiguo a sinistra dell'intervallo di training.	109
VI.29	Dipendenza di DBP da R_2 sull'intervallo attiguo a destra dell'intervallo di training.	109
VI.30	PP: andamento del training e validation loss, early stopper, R2Score e MSE.	110
VI.31	PP: predizioni sui dati di input.	110
VI.32	Dipendenza di PP da C sull'intervallo di training e due intervalli attigui.	111
VI.33	Dipendenza di PP da C sull'intervallo di training.	111
VI.34	Dipendenza di PP da C sull'intervallo attiguo a sinistra dell'intervallo di training.	112
VI.35	Dipendenza di PP da C sull'intervallo attiguo a destra dell'intervallo di training.	112
VI.36	Dipendenza di PP da R_1 sull'intervallo di training e due intervalli attigui.	113
VI.37	Dipendenza di PP da R_1 sull'intervallo di training.	113

VI.38	Dipendenza di PP da R_1 sull'intervallo attiguo a sinistra dell'intervallo di training.	114
VI.39	Dipendenza di PP da R_1 sull'intervallo attiguo a destra dell'intervallo di training.	114
VI.40	Dipendenza di PP da R_2 sull'intervallo di training e due intervalli attigui.	115
VI.41	Dipendenza di PP da R_2 sull'intervallo di training.	115
VI.42	Dipendenza di PP da R_2 sull'intervallo attiguo a sinistra dell'intervallo di training.	116
VI.43	Dipendenza di PP da R_2 sull'intervallo attiguo a destra dell'intervallo di training.	116
VI.44	SBP: andamento del training e validation loss, early stopper, R2Score e MSE.	117
VI.45	SBP: predizioni sui dati di input.	117
VI.46	Dipendenza di SBP da C sull'intervallo di training e due intervalli attigui.	118
VI.47	Dipendenza di SBP da C sull'intervallo di training.	118
VI.48	Dipendenza di SBP da C sull'intervallo attiguo a sinistra dell'intervallo di training.	119
VI.49	Dipendenza di SBP da C sull'intervallo attiguo a destra dell'intervallo di training.	119
VI.50	Dipendenza di SBP da R_1 sull'intervallo di training e due intervalli attigui.	120
VI.51	Dipendenza di SBP da R_1 sull'intervallo di training.	120
VI.52	Dipendenza di SBP da R_1 sull'intervallo attiguo a sinistra dell'intervallo di training.	121
VI.53	Dipendenza di SBP da R_1 sull'intervallo attiguo a destra dell'intervallo di training.	121
VI.54	Dipendenza di SBP da R_2 sull'intervallo di training e due intervalli attigui.	122
VI.55	Dipendenza di SBP da R_2 sull'intervallo di training.	122
VI.56	Dipendenza di SBP da R_2 sull'intervallo attiguo a sinistra dell'intervallo di training.	123
VI.57	Dipendenza di SBP da R_2 sull'intervallo attiguo a destra dell'intervallo di training.	123

List of Tables

V.1	Tabella riassuntiva ciclo cardiaco [Wik22a].	74
V.2	Tempo di esecuzione per trovare l'approssimazione della soluzione del modello Windkessel su diversi cicli cardiaci.	86
V.3	Valori variabili calcolati con il modello Windkessel e parametri di input (C, α) stimati. I valori reali sono ripresi da [AW20]. . .	87
V.4	Sensitività locale delle variabili $\mathcal{M} = \{MAP, DBP, SBP, PP\}$ al variare del valore dei parametri $\mathcal{P} = \{C, R_1, R_2, P_d\}$. I parametri sono inseriti in ordine decrescente (in modulo) di sensitività. . .	87
V.5	Variazione delle variabili all'aumento del dieci percento dei parametri. $\mathcal{M}_S = \{MAP_S, DBP_S, SBP_S, PP_S\}$ sono le variabili ottenute con parametri standard (stimati precedentemente), $\mathcal{M}_{+10\%} = \{MAP_{+10\%}, DBP_{+10\%}, SBP_{+10\%}, PP_{+10\%}\}$ sono le variabili ottenute con singoli parametri aumentati del dieci percento, $\Delta = \{\Delta_{MAP}, \Delta_{DBP}, \Delta_{SBP}, \Delta_{PP}\}$ sono le variazioni percentuali della variabile a pedice.	88
VI.1	Tempo di esecuzione per trovare l'approssimazione dei parametri usando i processi gaussiani addestrati.	124

I

Introduction

In the introductory chapter, the topics covered in the paper are motivated and introduced, and the main objective is explained: the application to the Windkessel model of supervised learning with Gaussian processes. The organization of the paper in terms of chapters, sources, images and code is then shown.

Breve dialogo con punto di circonferenza

- Per il Centro, mi scusi, qual è la via?
- Oh non si affanni, qui è tutto periferia.

Marco Furgeri

I.1 Topics covered and motivation

I.1.1 Gaussian processes

One of the main topics covered in the paper is supervised learning, that is, the problem of learning relationships between inputs and outputs from an example dataset and then making predictions about new inputs that the machine has never seen. It is therefore clear that the problem at hand is inductive: one must define (finite) training data D and a function f that makes predictions for all possible input values.

One approach to the problem is to define a class of functions from which to draw (e.g., linear functions), but this presents a problem: the choice of class. Indeed, this choice is very delicate since it can lead, for example, to a model based on functions that fail to accurately model the target function; in that case, predictions will be inaccurate. Furthermore, increasing the size of the class of functions (e.g., by increasing their parameters in a parametric regression context) does not necessarily improve the predictions, since there is a risk of *overfitting*, in which a good fit to the training data is obtained but a poor result in predictions on new data.

A second approach is to assign a probability to each possible function, where higher probabilities are assigned to functions that are considered more likely, for example, because they are smoother (in terms of continuity). This approach is not without problems: there are an infinite number of possible functions and one is interested in evaluating this set in finite time. Gaussian processes elegantly solve this problem by generalizing the Gaussian probability distribution. While a probability distribution describes random variables that are scalars or vectors, this type of stochastic process governs the properties of functions. Intuitively, one can think of a function as a very long (infinite) vector in which each component is the value of the function $f(x)$ for some x . Throughout this work, it is explained how, although a simple idea, this solves the above problem. In fact: inference in Gaussian processes is able to draw conclusions from the properties of the function on a finite number of points, thus ignoring infinitely many. To do so, the function in question is considered as a vector with $f(x_i)$ components for $i = 1, \dots, N$ which, because of the properties of Gaussian processes, is manipulated using multivariate Gaussian random variable theory, i.e., a relatively simple theory. This is extremely powerful because it is perfectly adaptable computationally. While not a well-known approach, in reality many models commonly used in machine learning and statistics are actually special cases or limited types of Gaussian processes.

I.1.2 Windkessel model

Haemodynamic models based on simplified representations of the components of the cardiovascular system can contribute strongly to the study and understanding of circulatory physiology and pathology. These models can be derived from the Navier-Stokes equations by exploiting specific features of blood flow, such as the cylindrical morphology of vessels, and can offer a great level of detail and a potentially accurate description of relevant quantities, but their numerical discretization is very complex and requires high computational resources.

The representation and analysis of the cardiovascular system (within what are known as zero-dimensional models) began with the modeling of arterial flow using the Windkessel model. In particular, it is the two-element Windkessel model, first proposed by Stephen Hales in 1733 and later mathematically formulated by Otto Frank in 1899, that is among the simplest and best known (zerodimensional) models. It consists of a capacitor C , which describes the elasticity properties of large arteries, and a resistor R (divided into two resistors R_1 and R_2), which describes the dissipative nature of small peripheral vessels, including arterioles and capillaries. The modeling was then expanded to cover the modeling of other cardiovascular components, such as the heart, heart valves, and veins to simulate the overall hemodynamics of the entire circulatory system.

However, modeling blood flow in highly complex networks can involve computationally expensive simulations. The situation worsens when, for example, one wants to integrate different mechanisms in the cerebral microcirculation, such as cerebral perfusion or solute exchange between the blood and various tissue beds.

Several papers can be found in the literature concerning models for the simulation of arterial blood flow that address the issues of runtime and topological complexity optimization. Of interest in this work, however, is to exploit the capabilities of the Windkessel model (with easy generalization to other hemodynamic models) without having to solve the differential equation that describes it. In the Windkessel model, there is only one differential equation, so the computational cost and time required to get an approximation of the solution are very low. However, exploiting supervised learning with Gaussian processes yields the same results as in the Windkessel model (within a certain region of uncertainty) without solving the differential equation, thus decreasing the running time. In this simple case, there is no noticeable improvement, but thinking about very complex models with many differential equations, which generally require long run times and even hosting on supercomputers, acceptable run times in clinical settings can be achieved with this approach.

I.1.3 Justification of the topic

Computational Life is a company founded in 2018 by Dr. Christian Contarino (Ph.D. at the University of Trento in 2018) focusing on biomedical applications of mathematics. Most recently, he is working on AltegosTM, a patient-specific predictive decision support software that leverages the predictive technology of supervised learning in Gaussian processes.

As anticipated, complex hemodynamic models (e.g., comprehensive models that model all components of the circulation) take a long time to run, which can be up to several hours. In a research setting this is not problematic, but in a clinical setting where one interfaces with patients who need urgent care this wait is incompatible. As anticipated then, Gaussian processes solve the run-time problem, allowing for support of a hemodynamic model that would potentially take hours to run.

In addition, Gaussian processes (and specifically the library used in the paper under VI) allow for the study of the *global sensitivity analysis* of parameters, making it possible to understand which among them actually have influence in a given output and which can be discarded from the study. This makes it possible to lighten machine learning by providing the statistical model with fewer parameters, sometimes many fewer, speeding up the process and improving accuracy (similar to what was concluded about P_d in V.5).

These features were implemented in the research enterprise context by the research team coordinated by Dr. Contarino to create the AltegosTM product. In addition, the choice of Gaussian processes is justified by the fact that they have already shown excellent results in domains similar to the one studied in the paper (e.g., in [Lon+20] and in [Yuh+22]) and allow for an indication of prediction accuracy in the form of mean and standard deviation. This makes them a preferable choice to other machine learning approaches.

It is therefore evident that this technology and its application are novelties in the world of research and that its study constitutes an important addition to the academic background of a bachelor's degree student. This work, therefore, aims to study this technology applied in a simplified context, namely that of the Windkessel model, so that it can be easily generalized to more real-world and complex contexts such as those addressed in the master's degree.

I.2 Organization of the work

I.2.1 Sources

Each chapter is given an introductory page in which the content is anticipated and the sources used in writing it are cited. The main source used for the part of Gaussian processes is [RW06]; for the part related to hemodynamics and the Windkessel model important source of information was Professor Lucas Omar Müller author of [GTM22] and of the jupyter notebook that provided practical results on the Windkessel model (later extensively modified); the python library "GPERks," which can be found on GitHub, was used in the chapter "Methodology and Training Results".

I.2.2 Images

Most of the images were generated by the author of the work; of these the python code is often given in the appendix¹. Of all images not generated by the author, the source can be found in the caption.

I.2.3 Code

The code written for generating the images and the results obtained is written in python. This choice was made because python allows many options in graph creation and because the GPERks library is written in python.

¹Not of all the images are the codes for generating them given because in some cases it was too massive to include in the paper and sometimes not very useful: the codes for the GPERks library, for example, can be found on the GitHub page and there is no need to include them in the paper

I.2.4 Structure of the work

The first two chapters are intended to introduce the broad topic of Gaussian processes. The discussion will not devote itself to framing Gaussian processes in the broad context of stochastic processes but will limit itself to a focused study of its peculiarities useful for the purposes of supervised learning.

The chapter "Machine learning" introduces the concepts of Bayesian statistics underlying supervised learning focusing on the case of Gaussian processes. Some optimization methods including the one used to obtain the results shown in the last chapter are also introduced for informative purposes.

Next, the Windkessel model is introduced, showing only the differential equation (thus without explaining how to deduce it from the Navier-Stokes equation). Practical results on its use in predicting a patient's blood pressure from his flow are then illustrated. The chapter concludes with a study of the local sensitivity of MAP, DBP, SBP, and PP with respect to C (compliance), R_1 (proximal resistance), R_2 (peripheral resistance), and P_d (distal pressure), concluding that distal pressure has little influence on the variables, which is why it was excluded from the input parameters in supervised learning.

The last chapter shows the results obtained from training Gaussian processes following the GPERks library approach to study the C , R_1 and R_2 dependence of blood pressure.

The appendix contains most of the codes used for generating the images and results used in the paper.

II

Gaussian distribution

In this chapter, the basic properties of Gaussian distributions are given, with special attention in regard to the **multivariate Gaussian distribution**. As the name suggests, **Gaussian processes** are in fact based on these multivariate distributions, exploiting properties that will be shown throughout the chapter.

The sources used in writing the chapter are [Gut09], [Wil20], [Mur22].

At a purely formal level, one could call probability theory the study of measure spaces with total measure one, but that would be like calling number theory the study of strings of digits which terminate.

Terence Tao

II.1 Univariate Gaussian distribution

Definition II.1.1 (Univariate Gaussian distribution). The **univariate Gaussian distribution** is a continuous probability distribution.

Given $Y \sim \mathcal{N}(\mu, \sigma^2)$, its probability density function is expressed as:

$$f_Y(y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2}\frac{(y-\mu)^2}{\sigma^2}\right\}.$$

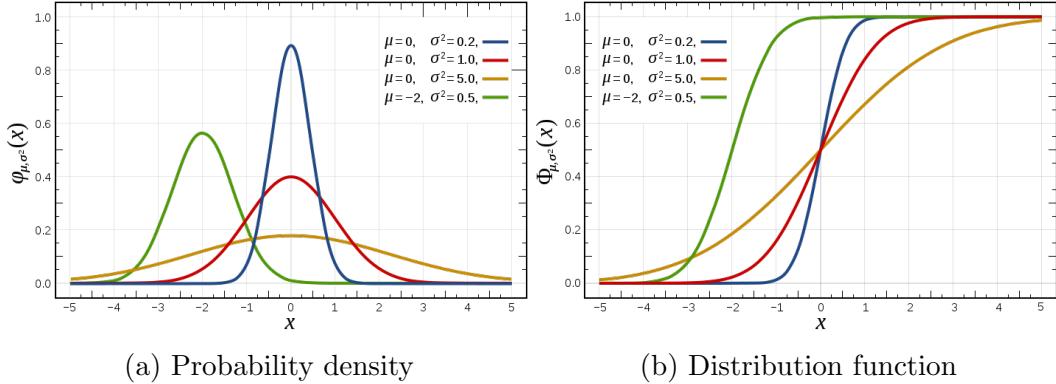


Figure II.1: Distribution function and probability density of a standard normal distribution [Wik22b].

Remark II.1.2. Given $Z \sim \mathcal{N}(0, 1)$ it follows: $Y = \mu + \sigma Z \sim \mathcal{N}(\mu, \sigma^2)$.

Remark II.1.3. Informally speaking, the Gaussian distribution is "convenient" mathematically because of its properties:

- the normal distribution has two parameters that are easy to interpret: the mean and the variance;
- the normal distribution is closed under linear operations;
- the normal distribution is closed by marginalization and conditioning (see the proposition II.2.5);
- at equal mean and variance, the normal distribution has maximum entropy;
- from **central limit theorem**, the normal distribution is the limit of a sum of random variables;
- the normal distribution has a simple mathematical form that facilitates its implementation.

II.2 Multivariate Gaussian distribution

The multivariate Gaussian distribution is a generalization of the normal (univariate) distribution.

Definition II.2.1 (Multivariate Gaussian distribution). An n -dimensional \mathbf{X} -vector of random variables is said to be normal (**multivariate normal**) if and only if $\forall \mathbf{a} \in \mathbb{R}^n$ the random variable $\mathbf{a}^T \mathbf{X}$ is a normal distribution. The density¹ of a **multivariate Gaussian distribution** is expressed as

$$f_{\mathbf{X}}(\mathbf{x}) = \frac{1}{(2\pi)^{n/2} \det(\Sigma)^{1/2}} \exp \left[-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right]$$

where $\boldsymbol{\mu} = \mathbb{E}[\mathbf{X}] \in \mathbb{R}^n$ is the *mean vector*, $\Sigma = \text{Cov}[\mathbf{X}]$ is a $n \times n$ matrix called *covariance matrix*, defined as:

$$\begin{aligned} \text{Cov}[\mathbf{X}] &= \mathbb{E} [(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{X} - \mathbb{E}[\mathbf{X}])^T] \\ &= \begin{pmatrix} \mathbb{V}[X_1] & \text{Cov}[X_1, X_2] & \dots & \text{Cov}[X_1, X_n] \\ \text{Cov}[X_2, X_1] & \mathbb{V}[X_2] & \dots & \text{Cov}[X_2, X_n] \\ \vdots & \vdots & \ddots & \vdots \\ \text{Cov}[X_n, X_1] & \text{Cov}[X_n, X_2] & \dots & \mathbb{V}[X_n] \end{pmatrix} \end{aligned}$$

where:

$$\begin{aligned} \text{Cov}[X_i, X_j] &= \mathbb{E}[(X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}[X_j])] = \mathbb{E}[X_i X_j] - \mathbb{E}[X_i] \mathbb{E}[X_j] \\ \mathbb{V}[X_i] &= \text{Cov}[X_i, X_i]. \end{aligned}$$

Remark II.2.2. The covariance matrix is symmetrical and semidefinite positive, i.e., $\forall \mathbf{a} \in \mathbb{R}^n$ one has $\mathbf{a}^T \mathbf{a} \geq 0$.

Corollary II.2.3. Given \mathbf{X} a multivariate normal vector, from the definition of multivariate Gaussian distribution follow immediately:

1. each component of \mathbf{X} is a Gaussian random variable;
2. $\sum_{i=1}^n a_i X_i$ is a Gaussian random variable $\forall a_i \in \mathbb{R}$;
3. If the components of \mathbf{X} are independent Gaussian random variables, then \mathbf{X} is a multivariate normal vector.

Remark II.2.4. The third point of the previous corollary does not hold if the components are not independent. For example: $X \sim \mathcal{N}(0, 1)$, $Z \perp\!\!\!\perp X$, $\mathbb{P}(Z = 1) = \mathbb{P}(Z = -1) = 1/2$. It is easily seen that $Y = ZX$ is not independent of X and $\begin{pmatrix} X \\ Y \end{pmatrix}$ is not multivariate normal.

¹In the generalized case it is not possible to define the density of the multivariate Gaussian distribution. As will be shown later, in Gaussian processes the density is not as important as the covariance matrix and the mean vector.

Of critical importance is the next proposition. The proof is omitted because it consists of lengthy calculations that are outside the scope of the paper. For the proof, please refer to [Mur22].

Proposition II.2.5 (Marginal and conditional distribution). *Let $\mathbf{Y} = \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix}$ Gaussian multivariate vector where:*

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix} \quad \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix} \quad \boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1} = \begin{pmatrix} \boldsymbol{\Lambda}_{11} & \boldsymbol{\Lambda}_{12} \\ \boldsymbol{\Lambda}_{21} & \boldsymbol{\Lambda}_{22} \end{pmatrix}$$

Then the marginal distributions are:

$$\mathbf{y}_1 \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11}) \quad \mathbf{y}_2 \sim \mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{22})$$

While conditional distributions are:

$$\mathbf{y}_1|\mathbf{y}_2 \sim \mathcal{N}(\boldsymbol{\mu}_{1|2}, \boldsymbol{\Sigma}_{1|2}) \quad \begin{aligned} \boldsymbol{\mu}_{1|2} &= \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\mathbf{y}_2 - \boldsymbol{\mu}_2) \\ \boldsymbol{\Sigma}_{1|2} &= \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21} = \boldsymbol{\Lambda}_{11}^{-1} \end{aligned}$$

$$\mathbf{y}_2|\mathbf{y}_1 \sim \mathcal{N}(\boldsymbol{\mu}_{2|1}, \boldsymbol{\Sigma}_{2|1}) \quad \begin{aligned} \boldsymbol{\mu}_{2|1} &= \boldsymbol{\mu}_2 + \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}(\mathbf{y}_1 - \boldsymbol{\mu}_1) \\ \boldsymbol{\Sigma}_{2|1} &= \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12} = \boldsymbol{\Lambda}_{22}^{-1} \end{aligned}$$

II.3 Correlation for multivariate vectors

Definition II.3.1 (Pearson's correlation index). The **Pearson's correlation index** between two random variables is an index expressing a linearity relationship between them. It is expressed as:

$$\rho_{XY} = \frac{\text{Cov}[X, Y]}{\sqrt{\text{V}[X]\text{V}[Y]}}.$$

Remark II.3.2 (Meaning of ρ_{XY}). Because of the Cauchy-Schwarz inequality it holds: $-1 \leq \rho_{XY} \leq 1$. There are three main cases: $\rho_{XY} = 1$ indicates a perfect positive linear relationship; $\rho_{XY} = -1$ indicates a perfect negative linear relationship; $\rho_{XY} = 0$ indicates no linear correlation.

The remainder of the chapter will focus on the analysis of bivariate Gaussian vectors and the correlation of their components and then generalize the visual approach to multiple dimensions.

Consider a two-dimensional Gaussian vector:

$$\mathbf{Y} = \begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix} \quad \text{con} \quad \boldsymbol{\mu} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \boldsymbol{\Sigma} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Generating points from the distribution of \mathbf{Y} (so each point consists of a two-dimensional vector) yields what is shown in figure II.2.

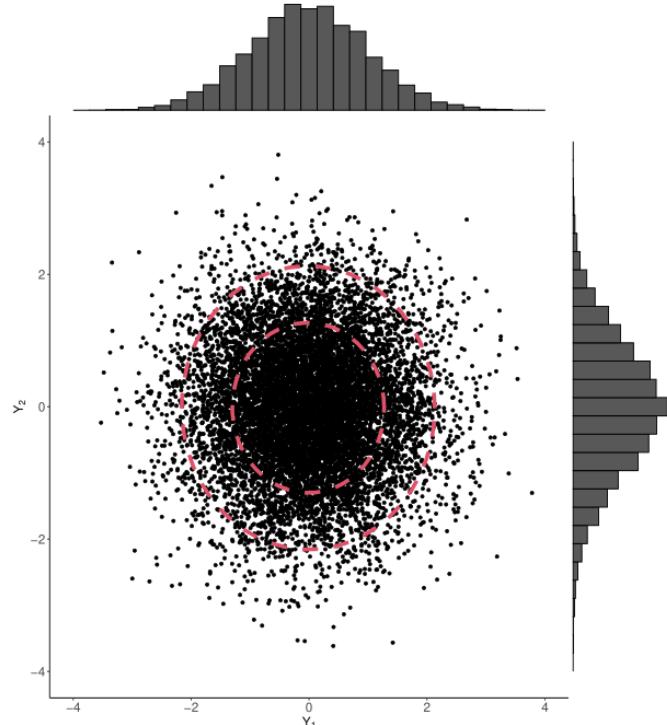


Figure II.2: Point cloud generated by a bivariate Gaussian distribution with uncorrelated components [Wil20].

Note that the point cloud is centered in zero as a consequence of the choice of μ . It is easy to compute $\rho_{Y_1, Y_2} = 0$ which guarantees the incorrelation of the components of the Gaussian vector. This result can be guessed from the shape of the cloud: given a point in the cloud, knowledge of its Y_1 coordinate gives no information about its Y_2 coordinate.

Now consider $\mu = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ e $\Sigma = \begin{pmatrix} 1 & 0.9 \\ 0.9 & 1 \end{pmatrix}$. Generating a cloud of points again yields what is shown in figure II.3.

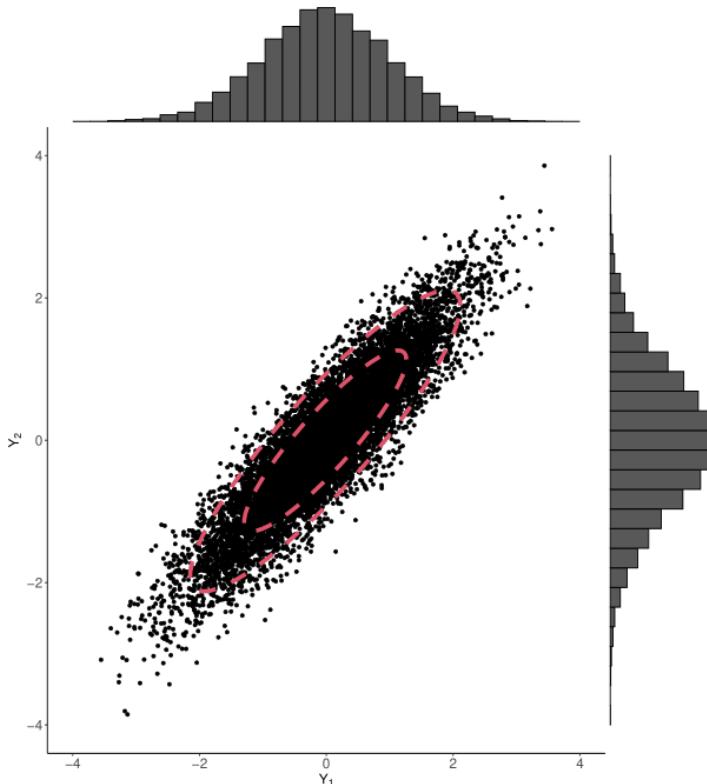


Figure II.3: Point cloud generated by a bivariate Gaussian distribution with correlated components [Wil20].

From the shape of the cloud it is now possible to see the correlation between the components of the Gaussian vector. Given a point on the cloud, in fact, the first of the two components gives an approximate idea of the value of the second component (and vice versa): the cloud, to a certain approximation, thickens around a line passing through the origin.

The elliptical shape of the cloud is a consequence of the correlation index value: $\rho_{Y_1, Y_2} = 0.9$, that is, there is a large linear correlation between the two components.

Evidently, the elliptical shape becomes more pronounced as the value of the correlation index increases, as can be seen from the figure II.4, in which $\Sigma = \begin{pmatrix} 1 & 0.99 \\ 0.99 & 1 \end{pmatrix}$ and thus $\rho_{Y_1, Y_2} = 0.99$.

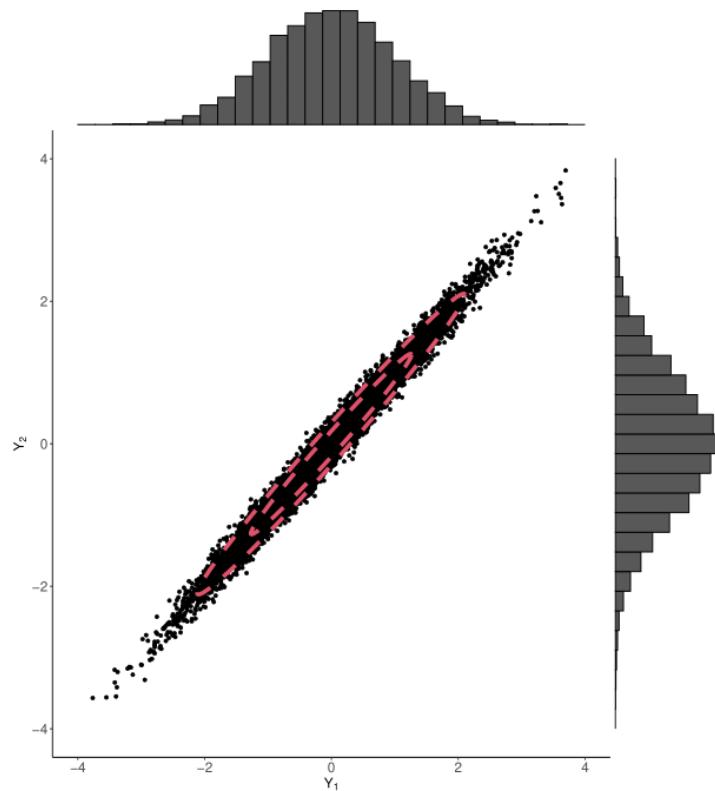


Figure II.4: Point cloud generated by a bivariate Gaussian distribution with strongly correlated components [Wil20].

In order to generalize to more than two dimensions the visual process of interpreting the correlation between the components of a multivariate Gaussian vector, it is necessary to adopt a different strategy: for each sample (i.e., for each vector) generated by the multivariate Gaussian distribution, the value of the components is plotted in a graph by connecting the respective values by a segment.

In figure II.5 the example in two dimensions with $\mu = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ and $\Sigma = \begin{pmatrix} 1 & 0.54 \\ 0.54 & 0.3 \end{pmatrix}$.

The graph on the left shows the approach used so far; the graph on the right shows for each point on the left graph the value of the first component at index 1 on the abscissae and the value of the second component at index 2 on the abscissae.

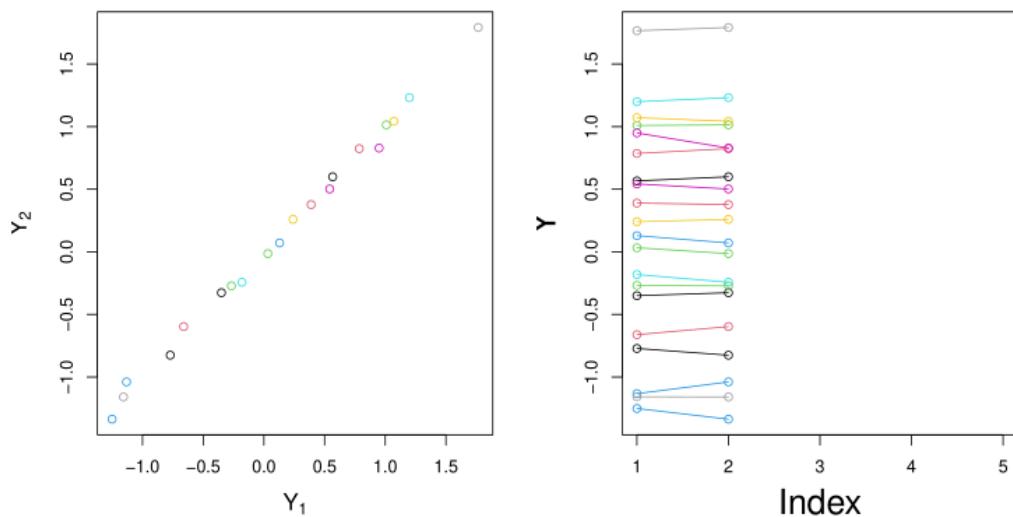


Figure II.5: Two different visual approaches to correlation of components of multivariate Gaussian vectors: $n = 2$ [Wil20].

This approach allows generalization to dimensions $n > 2$. Let it be now:

$$\boldsymbol{\mu} = \mathbf{0} \quad \boldsymbol{\Sigma} = \begin{pmatrix} 1 & 0.99 & 0.98 & 0.97 & 0.96 \\ 0.99 & 1 & 0.99 & 0.98 & 0.97 \\ 0.98 & 0.99 & 1 & 0.99 & 0.98 \\ 0.97 & 0.98 & 0.99 & 1 & 0.99 \\ 0.96 & 0.97 & 0.98 & 0.99 & 1 \end{pmatrix}$$

Where the components are strongly correlated with each other. We obtain what is shown in the figure II.6.

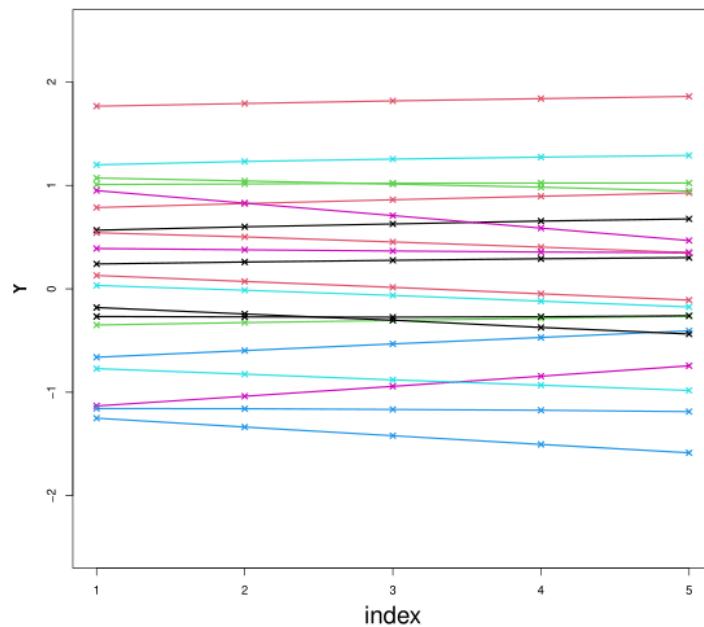


Figure II.6: Visualization using segments of the correlation of components of multivariate Gaussian vectors: $n = 5$ [Wil20].

Being in dimension $n = 5$, each sample generated by the Gaussian vector has five components, so when a sample is plotted in the graph it has five points at indices 1, 2, 3, 4, 5.

In this case, the strong correlation between the components of the Gaussian vector is reflected in the segment joining the individual points of each sample, as shown in Figure II.7.

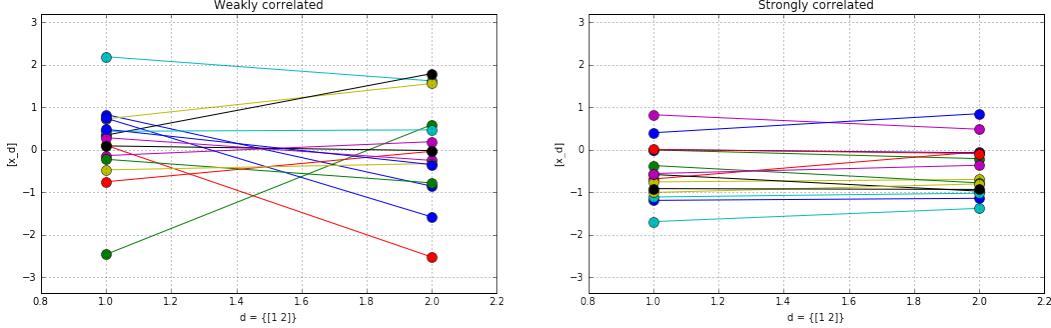


Figure II.7: Weak and strong correlation of components of multivariate Gaussian vectors visualized by segments [Dam16].

In the case $n = 50$ (with null norm vector but omitting the covariance matrix, which continues to have one on the diagonal and values close to one in the other entries), we obtain what is in the figure II.8.

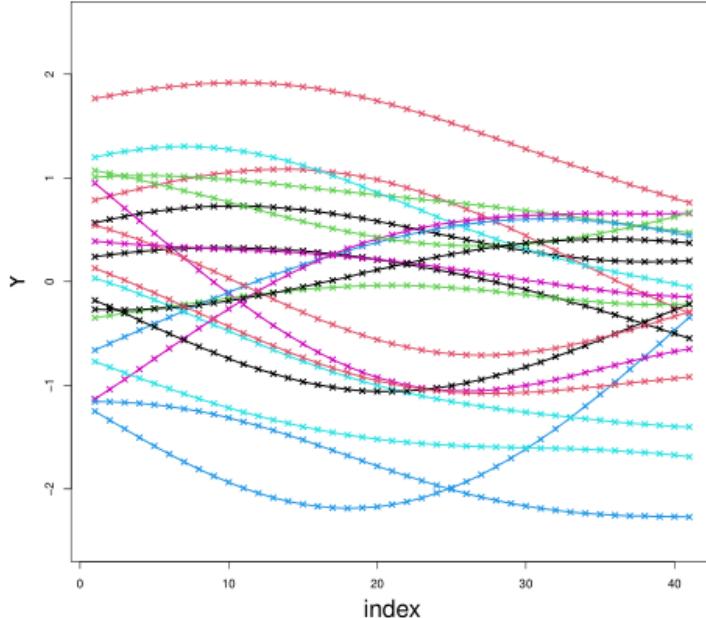


Figure II.8: Visualization by segments of the correlation of components of multivariate Gaussian vectors: $n = 50$ [Wil20].

Notice then that as n increases what obtained begins to resemble a function for each sample generated.

From this approach it is possible to interpret Gaussian processes either as functions or as infinite-dimensional multivariate Gaussian distributions ($n = \infty$) with a continuous index (introducing a *mean function* and a *covariance function*). This interpretation will be clarified in the next chapter.

III

Processi gaussiani

In questo capitolo vengono introdotti i **processi gaussiani**. Inevitabilmente il capitolo non sarà esaustivo dell'argomento ma saranno trattate le sue caratteristiche principali e interessanti ai fini dell'elaborato. Per questo motivo la trattazione non si dedicherà ad inquadrare i processi gaussiani nel vasto contesto dei processi stocastici e si limiterà ad uno studio mirato delle sue peculiarità.

I codici python usati nel capitolo richiedono di eseguire i codici VII.1 e VII.2 di import delle librerie. Inoltre la media cubica viene per tutti i kernel definita come nel codice VII.3. Il codice per le immagini dei kernel si ispira a quello scritto da Peter Roelants nel suo blog alla pagina "Gaussian processes - From scratch". Sono state fatte importanti modifiche al codice, in particolare vengono usati i kernel implementati dalla libreria scikit-learn. Il codice per la predizione di dati prende invece spunto dal sito W3cubDocs alla pagina "Gaussian Processes regression: basic introductory example". Anche in questo caso sono state fatte importanti modifiche nel codice. Il codice sulla concentrazione di CO_2 è una modifica del codice "Gaussian process regression (GPR) on Mauna Loa CO2 data" presente nella documentazione di scikit-learn.

Le fonti usate per la stesura del capitolo sono: [RW06], [Mur22], [Ras04], [Duv14], [GKD19], [Mur12].

In applying mathematics to subjects such as physics or statistics we make tentative assumptions about the real world which we know are false but which we believe may be useful nonetheless. The physicist knows that particles have mass and yet certain results, approximating what really happens, may be derived from the assumption that they do not. Equally, the statistician knows, for example, that in nature there never was a normal distribution, there never was a straight line, yet with normal and linear assumptions, known to be false, he can often derive results which match, to a useful approximation, those found in the real world.

George E. P. Box

III.1 Definizione e motivazione

Definition III.1.1 (Processo gaussiano). Un **processo gaussiano** è un insieme di variabili casuali tali che ogni suo sottoinsieme finito abbia distribuzione gaussiana multivariata.

Dalla definizione risulta evidente come la teoria delle distribuzioni gaussiane multivariate abbia notevole importanza nello studio dei processi gaussiani.

Similmente alla distribuzione gaussiana, completamente determinata dal suo vettore media e dalla sua matrice di covarianza, i processi gaussiani sono completamente determinati da una *mean function* (che ne determina la media), denotata $m(x)$, e da una *covariance function* (che ne determina la covarianza), denotata $k(x, x')$. Verrà approfondito il ruolo delle due funzioni nella prossima sezione.

Nonostante questa similarità però, le distribuzioni gaussiane e i processi gaussiani differiscono per una importante caratteristica: le prime lavorano con vettori, i secondi con funzioni.

Remark III.1.2 (Motivazione: regressione nonlineare). La principale applicazione dei processi gaussiani, nonché il tema principale di questo capitolo, è quello della **regressione nonlineare**.

Mentre il modello di regressione lineare cerca, a partire da dati osservati, una relazione lineare tra una variabile dipendente Y e una variabile indipendente x , tenendo conto di un errore statistico ϵ , in forma: $Y_i = \beta_0 + \beta_1 x_i + \epsilon_i$, dove le incognite sono β_0 e β_1 ; la regressione nonlineare è una forma di regressione in cui la funzione che esprime la relazione tra variabili indipendenti e dipendenti è una combinazione non lineare dei parametri del modello e dipende da una o più variabili indipendenti, dunque in forma: $Y = f(X, \theta) + \epsilon$, dove le incognite sono θ e la funzione f .

Una importante differenza tra i due tipi di regressione è che per quella nonlineare, a differenza di quella lineare, non esiste un metodo generale per la determinazione dei valori dei parametri.

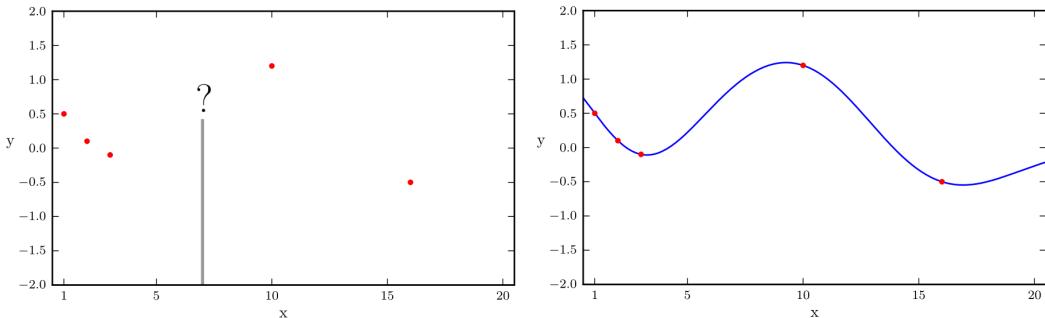


Figure III.1: Regressione nonlineare [Tur16].

I **processi gaussiani** forniscono un buon metodo per la determinazione dei parametri nella regressione nonlineare non solo perché sfruttano la teoria della distribuzione gaussiana multivariata, complessivamente semplice, ma perché oltre ad una funzione che approssima i dati in modo nonlineare con massima probabilità (concetto chiarito successivamente) forniscono un'indicazione della confidenza della regressione in forma di un'area all'interno della quale funzioni interpolanti meno probabili (ma comunque possibili) risiedono, come illustrato in figura III.2.

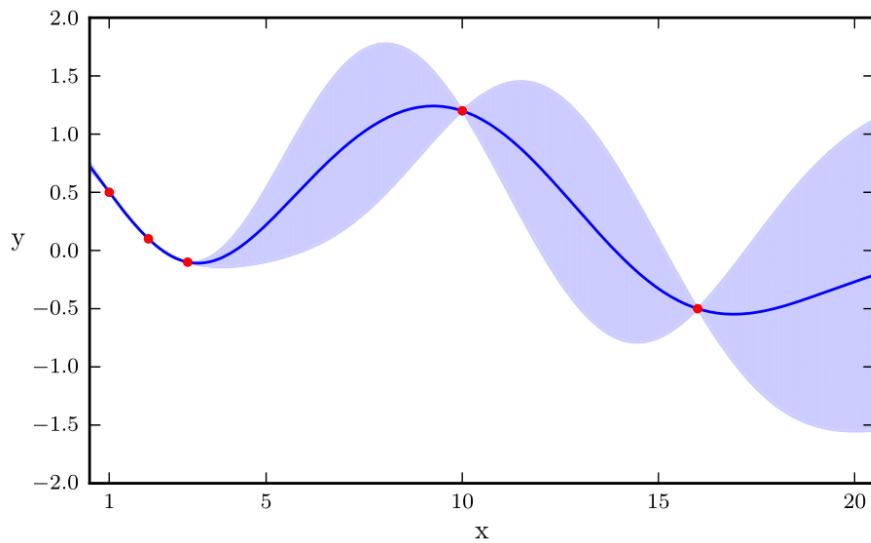


Figure III.2: Regressione nonlineare con processi gaussiani [Tur16].

III.2 Introduzione ai processi gaussiani e alla notazione

Seppur non rientri negli scopi dell'elaborato approfondire questo aspetto, viene riportata la definizione di processo stocastico per facilitare l'introduzione della notazione usata.

Definition III.2.1 (Processo stocastico). Un **processo stocastico** su uno spazio di probabilità $(\Omega, \mathcal{F}, \mathbb{P})$ è una famiglia $\{X_t\}_{t \in T \subset \mathbb{R}}$ di variabili casuali indicizzata da un parametro t .

È comune chiamare t l'indice per sottolineare il ruolo del tempo nei processi stocastici: un processo stocastico, generalmente, descrive matematicamente l'evoluzione temporale di un sistema caratterizzato dall'essere soggetto al caso, un sistema, cioè, del quale lo stato al tempo t non può essere determinato con certezza ma solo da una variabile casuale.

In un sistema stocastico è dunque possibile solo calcolare la *probabilità* che il sistema si trovi in uno degli stati possibili ad un tempo $t > t_0$ se è noto lo stato al tempo t_0 ; in un sistema deterministico invece l'evoluzione viene descritta da regole (tipicamente in forma di equazione differenziale) che consentono di determinare con precisione lo stato del sistema ad ogni tempo $t > t_0$ noto lo stato al tempo t_0 .

Notation III.2.2 (Funzioni e indicizzazione nei processi gaussiani). Scrivendo $f \sim \mathcal{GP}(m, k)$ si intende che "*la funzione f è distribuita come un processo gaussiano con mean function $m(\cdot)$ e covariance function $k(\cdot, \cdot)$* ".

La funzione così ottenuta si può descrivere come:

$$f : \chi \rightarrow \mathbb{R},$$

dove χ è un qualsiasi dominio. Per ogni elemento x nel dominio χ esiste una variabile casuale $f(x)$ a cui è associato.

Remark III.2.3 (Versatilità dei processi gaussiani e generalizzazione dei vettori gaussiani). Si noti che il dominio χ non ha restrizioni, caratteristica che rende i processi gaussiani molto versatili. Si noti inoltre che con un dominio finito un processo gaussiano è un vettore gaussiano multivariato. In questo senso, come detto in precedenza, i processi gaussiani sono una generalizzazione dei vettori gaussiani multivariati.

Definition III.2.4 (Mean function). Dato un processo gaussiano $f \sim \mathcal{GP}(m, k)$, la **mean function** è una funzione

$$m : \chi \rightarrow \mathbb{R}$$

dove $m(x) = \mathbb{E}[f(x)]$.

Remark III.2.5. Per la *mean function* non vi è alcuna richiesta in termini di proprietà della funzione. Tipiche scelte sono $m(x) = 0$ oppure $m(x) = c$ con c costante.

Diversamente dalla *mean function*, la scelta della *covariance function* è ristretta a una determinata classe di funzioni, perciò prima della definizione vengono introdotti alcuni concetti preliminari.

Definition III.2.6 (Mercer kernel/positive definite kernel). Si definisce **Mercer kernel** (o **positive definite kernel**) qualunque funzione simmetrica

$$\mathcal{K} : \chi \times \chi \rightarrow \mathbb{R}^+$$

tale che: $\sum_{i=1}^N \sum_{j=1}^N \mathcal{K}(x_i, x_j) c_i c_j \geq 0$ per qualunque insieme di elementi distinti $\{x_i\}_{i=1}^N \subset \chi$, $\{c_i\}_{i=1}^N \subset \mathbb{R}$.

Una definizione alternativa si basa sul concetto di *matrice di Gram*:

Definition III.2.7 (Matrice di Gram). Data una funzione $\mathcal{K} : \chi \times \chi \rightarrow \mathbb{R}^+$, sia $\{x_i\}_{i=1}^N \subset \chi$ un insieme qualsiasi di elementi distinti, si definisce la **matrice di Gram** di \mathcal{K} la seguente:

$$\mathbf{K} = \begin{pmatrix} \mathcal{K}(x_1, x_1) & \mathcal{K}(x_1, x_2) & \dots & \mathcal{K}(x_1, x_N) \\ \mathcal{K}(x_2, x_1) & \mathcal{K}(x_2, x_2) & \dots & \mathcal{K}(x_2, x_N) \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{K}(x_N, x_1) & \mathcal{K}(x_N, x_2) & \dots & \mathcal{K}(x_N, x_N) \end{pmatrix}$$

Definition III.2.8 (Mercer kernel/positive definite kernel). Data una funzione $\mathcal{K} : \chi \times \chi \rightarrow \mathbb{R}^+$, \mathcal{K} si dice **Mercer kernel** se e solo se la sua matrice di Gram è semidefinita positiva.

È ora possibile definire la covariance function.

Definition III.2.9 (Covariance/kernel function). Dato un processo gaussiano $f \sim \mathcal{GP}(m, k)$, la **covariance function** è una funzione

$$k : \chi \times \chi \rightarrow \mathbb{R}$$

tale che $k(\cdot, \cdot)$ è un *Mercer kernel*. Si ha

$$k(x, x') = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))] = \text{Cov}[f(x), f(x')].$$

Remark III.2.10 (Similarità tra processi gaussiani e distribuzione gaussiana multivariata). Si è definita la *mean function* senza fare restrizioni sulle proprietà della funzione mentre per la *covariance function* è stato richiesto che la sua matrice di Gram sia semidefinita positiva.

Questo non deve sorprendere: come precedentemente detto, i processi gaussiani generalizzano la distribuzione gaussiana multivariata e come spiegato nel precedente capitolo (e enfatizzato nell'osservazione II.2.2) quest'ultima è definita da due parametri: un vettore media, senza alcuna restrizione, e una matrice di covarianza, che deve essere simmetrica e semidefinita positiva. Vi è dunque, anche in questo senso, similarità tra processi gaussiani e distribuzioni gaussiane multivariate.

Example III.2.11 (Esempio di processo gaussiano). Si consideri $f \sim \mathcal{GP}(m, k)$, dove:

$$m(x) = \frac{x^2}{4} \quad k(x, x') = \exp\left(-\frac{1}{2}(x - x')^2\right).$$

Per comprendere questo esempio di processo gaussiano si consideri il grafico di alcuni campioni della funzione f . Per non lavorare nel caso infinito, si consideri un dominio finito¹: $\chi = \{x_i\}_{i=1}^n$. Essendo χ finito, quello che si ottiene valutando $m(\cdot)$ e $k(\cdot, \cdot)$ sul dominio sono un vettore μ e una matrice Σ dove:

$$\begin{aligned} \mu_i &= m(x_i) = \frac{x_i^2}{4} & i = 1, \dots, n \\ \Sigma_{ij} &= k(x_i, x_j) = \exp\left(-\frac{1}{2}(x_i - x_j)^2\right) & i, j = 1, \dots, n \end{aligned}$$

Si ottiene quindi, come anticipato nell'osservazione III.2.3, che per ogni x_i nel dominio χ la variabile casuale $f(x_i)$ è un variabile casuale gaussiana di media μ_i e varianza Σ_{ii} ; chiamando $\mathbf{f} = (f(x_1), \dots, f(x_n))^T$ si ha cioè:

$$\mathbf{f} \sim \mathcal{N}(\mu, \Sigma).$$

¹Si tenga conto che dal punto di vista teorico i processi gaussiani che vengono considerati in questo elaborato si costruiscono su un dominio infinito di numeri reali; tuttavia quando si generano i grafici si è costretti a considerare un insieme finito di punti che poi vengono uniti per generare la curva. Con questa (obbligata) filosofia vengono costruiti i grafici della prossima sezione.

Avendo ottenuto un vettore gaussiano multivariato n -dimensionale risulta naturale utilizzare lo stesso metodo introdotto nella sezione II.3 per ricavare il grafico della "funzione" (è in realtà un vettore) \mathbf{f} . Nella figura III.3 vengono riportati i "grafici" di quattro campioni diversi di vettore gaussiano di dimensione $n = 60$. Si noti che la forma dei quattro grafici è ben diversa da quella della figura II.8 come conseguenza del fatto che le distribuzioni hanno matrice di covarianza diversa. Questo sottolinea quanto sia importante la covariance function nei processi gaussiani, essendo responsabile della matrice di covarianza. Questo argomento verrà analizzato in dettaglio successivamente. Con questo esempio si è chiarito cosa si intende quando si è detto che i processi gaussiani generalizzano la distribuzione gaussiana multivariata.

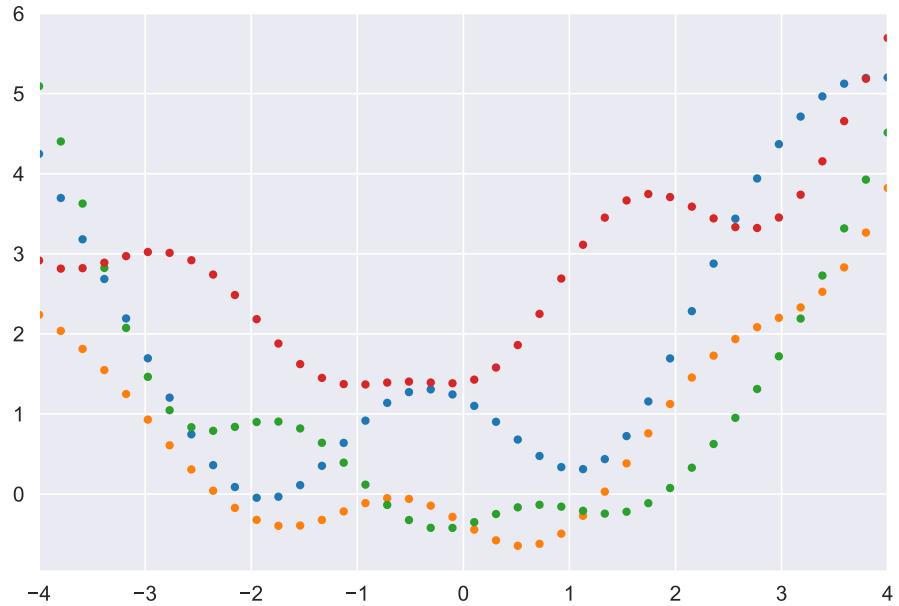


Figure III.3: Quattro vettori generati da un processo gaussiano definito come nell'esempio III.2.11. Codice VII.4.

Si noti che il codice VII.4 non genererà lo stesso grafico della figura III.3 in quanto c'è una componente randomica! Lo stesso vale per gli altri codici citati in questo capitolo.

III.3 Sulle covariance function

Per un processo gaussiano è di fondamentale importanza la covariance function: è infatti la funzione $k(\cdot, \cdot)$ che determina come il processo gaussiano interpreta i dati: a partire dalla definizione della covariance function è possibile ricavare diversi modelli, ad esempio la regressione lineare² o le splines³.

Remark III.3.1 (Importanza della covarianza). Nel precedente capitolo, alla sezione II.3, è stata mostrata un'interpretazione visiva dell'indice di correlazione di Pearson ed è stato chiarito come la covarianza influisca sulla correlazione di due variabili casuali.

Risulta quindi evidente l'importanza della covariance function nella correlazione tra le variabili casuali $f(x)$ e $f(x')$ per ogni $x, x' \in \chi$. Per questo motivo è significativo, nella scelta della covariance function, tenere conto di come questa dipenda dalla coppia (x, x') .

Seguono i tre principali tipi di covariance function in base alla loro dipendenza da (x, x') dove $x, x' \in \mathbb{R}^D$.

Definition III.3.2 (Covariance function stazionaria). Una covariance function **stazionaria** è una covariance function che dipende da $x - x'$.

Una covariance function di questo tipo è invariante per traslazioni.

Definition III.3.3 (Covariance function isotropica). Una covariance function **isotropica** è una covariance function che dipende da $\|x - x'\|$.

Una covariance function di questo tipo è invariante per movimenti rigidi.

Definition III.3.4 (Dot product covariance function). Una **dot product** covariance function è una covariance function che dipende da x e x' solo tramite $x \cdot x'$.

Una covariance function di questo tipo è invariante per rotazioni centrate nell'origine ma non per traslazioni.

Di seguito vengono mostrate le principali covariance function. Come già detto in precedenza, questa non può che essere un'introduzione alle possibili scelte: importanti aspetti delle covariance function riguardano la loro generalizzazione data dall'uso della distanza di Mahalanobis, una corretta scelta del kernel per l'ottimizzazione numerica, relazione tra scelta del kernel e deep learning per i processi gaussiani⁴, adattamento dei kernel a modelli a più dimensioni⁵...

In merito alle covariance function, è negli interessi di questo elaborato comprendere come queste influiscano nel processo gaussiano. Insieme ai prossimi esempi verranno quindi riportati dei grafici a supporto di questo interesse.

²Per approfondire si veda [RW06] e [Wil98]

³Per approfondire si veda [KW70]

⁴Per approfondire si veda [Mur22]

⁵Per approfondire si veda [Duv14]

III.3.1 Linear kernel

Definition III.3.5 (Linear kernel). Il **linear kernel** ha forma

$$k(x, x') = \sigma_b^2 + \sigma_v^2(x - c)(x' - c).$$

Viene riportato il grafico della funzione $k(x, x')$. Si ottiene una retta con gli usuali parametri.

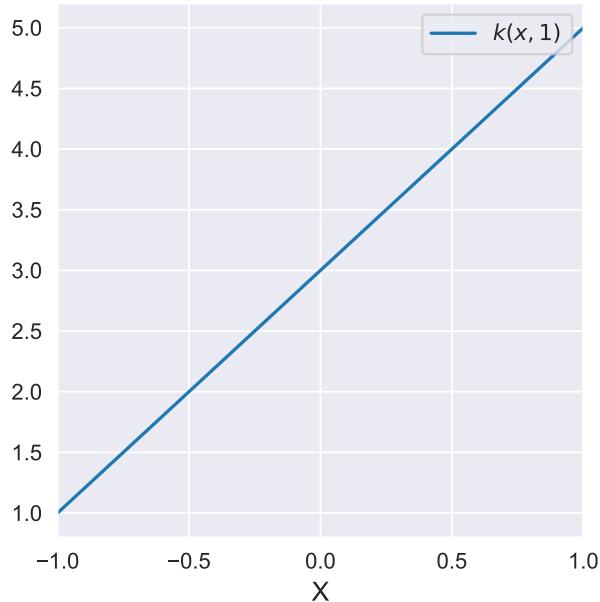


Figure III.4: Grafico di $k(x, x')$ linear kernel, $\sigma_b^2 = 1$, $\sigma_v^2 = 1$, $c = -1$ e $x' = 1$.
Codice VII.5

Nella figura III.5 vengono mostrati grafici di funzioni con distribuzione $f \sim \mathcal{GP}(m, k)$ dove $m(x) = 0$ e $k(x, x')$ è il linear kernel, $\sigma_b^2 = 0$, $\sigma_v^2 = 1$, $c = 0$.

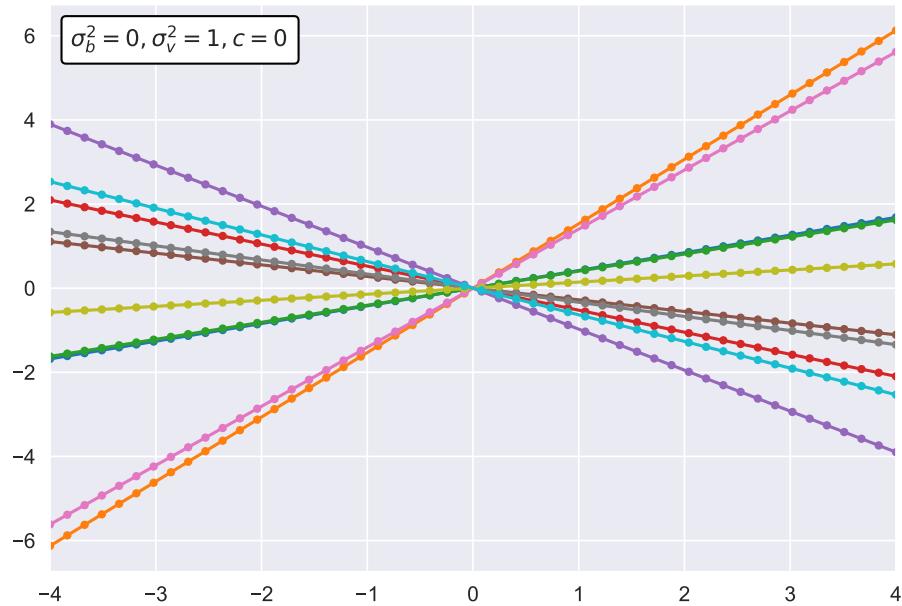


Figure III.5: Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è il linear kernel e $\sigma_b^2 = 0$, $\sigma_v^2 = 1$, $c = 0$. Codice VII.6.

Si noti che il linear kernel genera delle rette, da cui il nome.

Imporre la mean function uguale a zero aumenta la tendenza delle linee a passare per l'origine. Imponendo $m(x) = \alpha \in \mathbb{R}$ si aumenta la tendenza delle rette a passare per il punto $(0, \alpha)$.

Per comprendere l'influenza del parametro c , vengono mostrati grafici di funzioni con distribuzione $f \sim \mathcal{GP}(m, k)$ dove $m(x) = 0$ e $k(x, x')$ è il linear kernel e viene variato il valore di c .

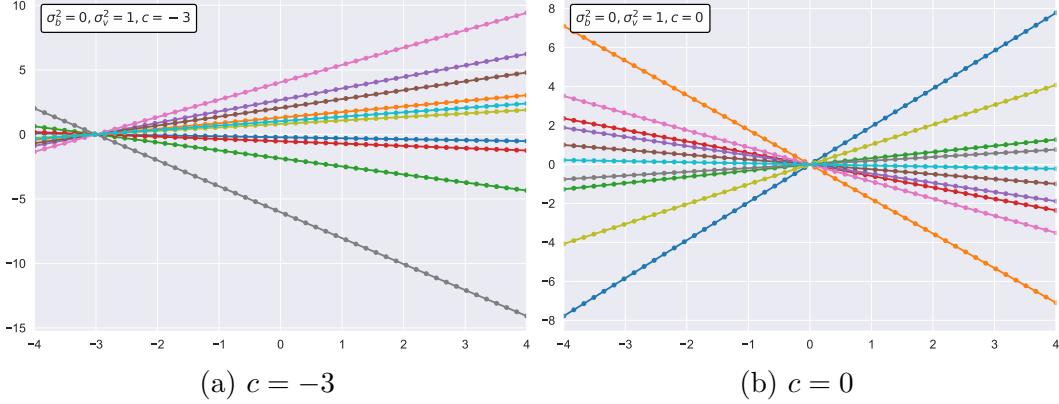


Figure III.6: Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è il linear kernel e $\sigma_b^2 = 0$, $\sigma_v^2 = 1$, il parametro c viene variato. Codice VII.7.

Il parametro c dunque impone un punto di passaggio per tutte le rette. Dunque c svolge lo stesso ruolo della mean function $m(\cdot)$.

Per comprendere l'influenza del parametro σ_b^2 , vengono mostrati grafici di funzioni con distribuzione $f \sim \mathcal{GP}(m, k)$ dove $m(x) = 0$ e $k(x, x')$ è il linear kernel e viene variato il valore di σ_b^2 .

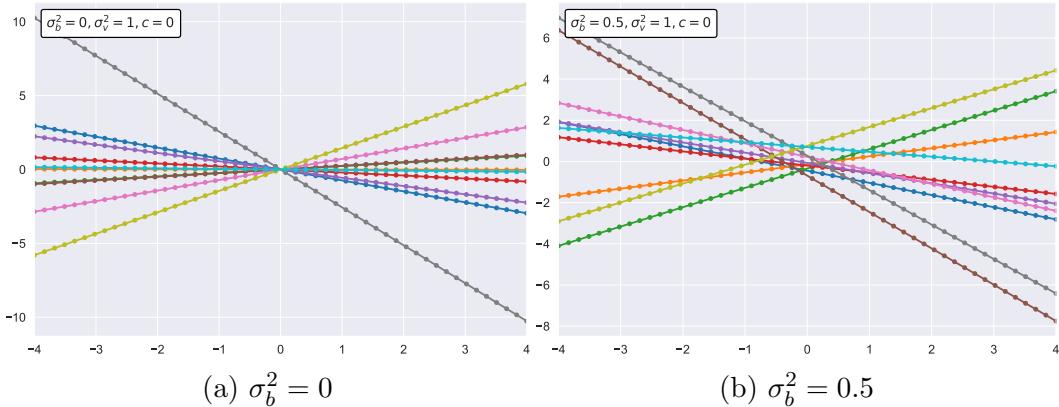


Figure III.7: Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è il linear kernel e $\sigma_v^2 = 1$, $c = 0$, il parametro σ_b^2 viene variato. Codice VII.8.

Dunque il parametro σ_b influenza la precisione con cui le funzioni tendono a passare per il punto $(0, c)$.

Per comprendere l'influenza del parametro σ_v^2 , vengono mostrati grafici di funzioni con distribuzione $f \sim \mathcal{GP}(m, k)$ dove $m(x) = 0$ e $k(x, x')$ è il linear kernel e viene variato il valore di σ_v^2 .

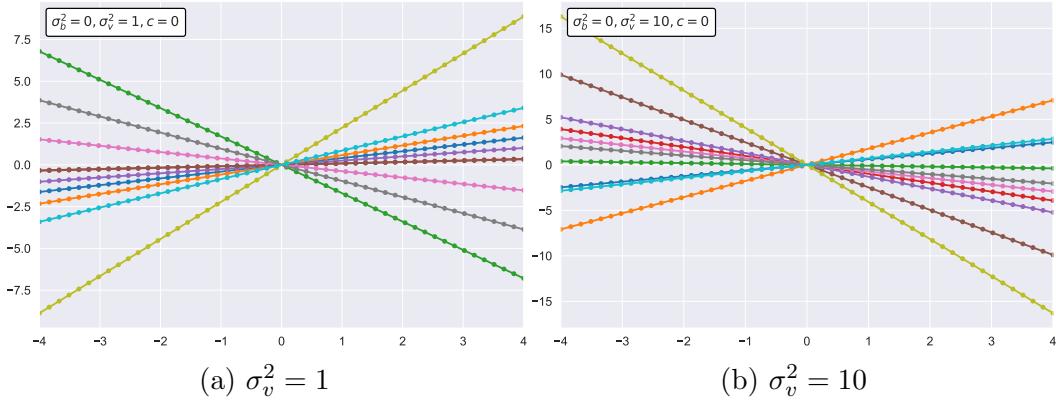


Figure III.8: Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è il linear kernel e $\sigma_b^2 = 0$, $c = 0$, il parametro σ_v^2 viene variato. Codice VII.9.

Dunque il parametro σ_v^2 influenza la pendenza delle rette, che è proporzionale al suo valore.

Per comprendere l'influenza della mean function, vengono di seguito mostrati grafici di funzioni con distribuzione $f \sim \mathcal{GP}(m, k)$ dove $m(x) = x^3$ e $k(\cdot, \cdot)$ è il linear kernel.

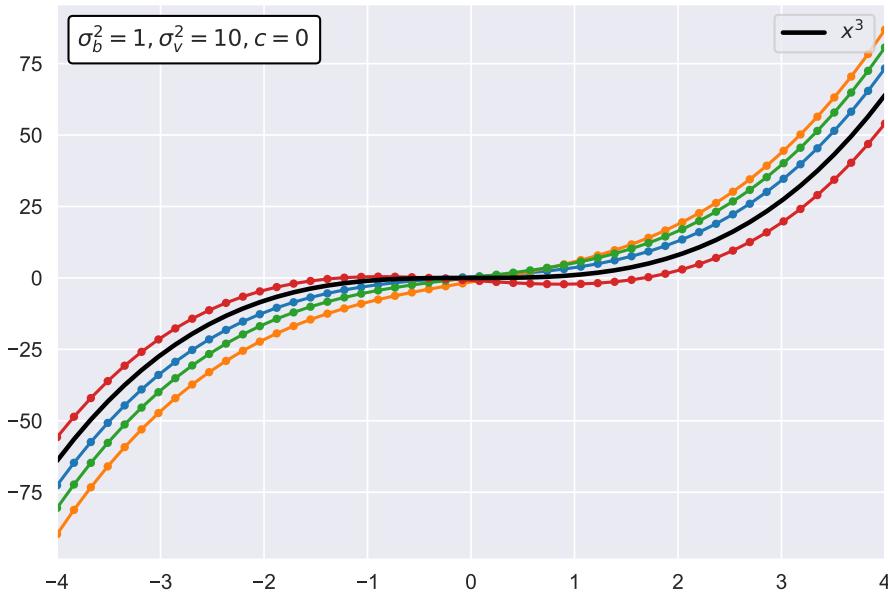


Figure III.9: Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(m, k)$ dove $m(x) = x^3$ e $k(x, x')$ è il linear kernel, $\sigma_b^2 = 1$, $\sigma_v^2 = 10$, $c = 0$. Codice VII.10.

Dal grafico è evidente che i grafici assomigliano alla funzione x^3 . Sarà più chiaro nei prossimi esempi di kernel, ma dalla definizione di processo gaussiano (pensando alla distribuzione gaussiana multivariata, che generalizza) ogni punto è interpretabile come campione di una distribuzione gaussiana. Ricordando l'osservazione II.1.2, sappiamo che ogni distribuzione normale (univariata) è scomponibile in $Y = \mu + \sigma Z$ dove $Z \sim \mathcal{N}(0, 1)$; dunque ogni punto x_i del grafico di una funzione con distribuzione il processo gaussiano come in figura III.9 ha scomposizione $x_i^3 + k(x_i, x_i)Z$. È dunque chiaro dalla scomposizione di ogni punto che i grafici aggiungeranno alla funzione x^3 un addendo dovuto alla covariance function.

III.3.2 Squared-exponential kernel

Definition III.3.6 (Squared-exponential kernel). Il **squared-exponential kernel** ha forma:

$$k(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2l^2}\right).$$

È dunque una covariance function isotropica.

Viene riportato il grafico della funzione $k(x, x')$. Si noti che il parametro σ^2 influisce sul picco della funzione, mentre il parametro l influisce indirettamente modificandone la velocità con cui si annulla.

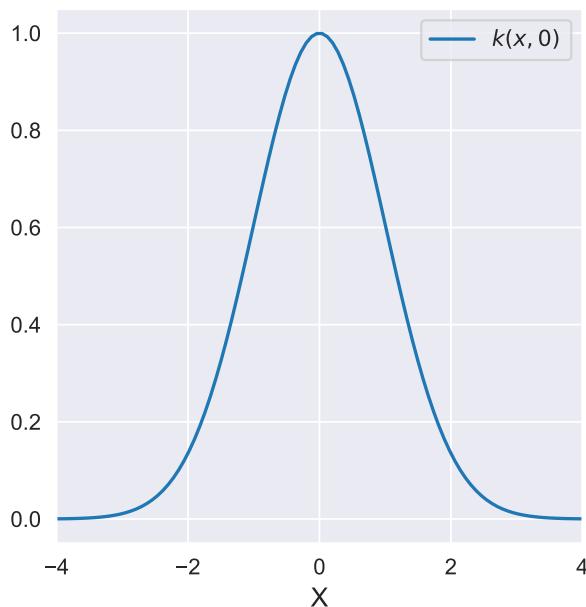


Figure III.10: Grafico di $k(x, x')$ squared-exponential kernel, $\sigma^2 = 1$ e $l^2 = 1$. Codice VII.11.

Una funzione distribuita come il processo gaussiano con questo tipo di kernel è C^∞ . Esistono diverse variazioni di questo kernel codificanti ipotesi leggermente diverse sulla continuità (anche locale) della funzione ma non sono negli interessi dell'elaborato.⁶

Vengono di seguito mostrati grafici di funzioni con distribuzione $f \sim \mathcal{GP}(m, k)$ dove $m(x) = 0$ e $k(x, x')$ è il squared-exponential kernel.

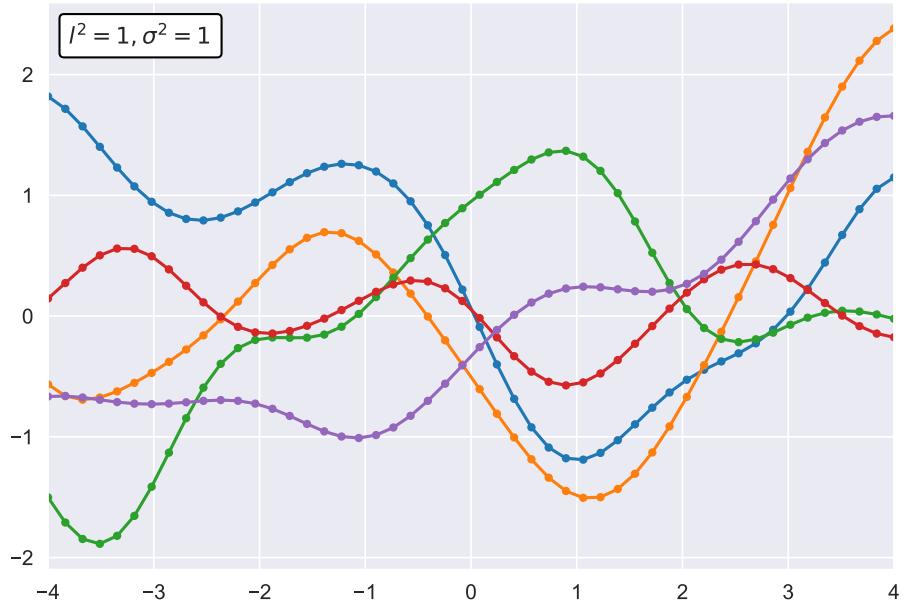


Figure III.11: Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è lo squared-exponential kernel e $l^2 = 1, \sigma^2 = 1$. Codice VII.12.

Dalla figura III.11 è possibile notare la differenza che il kernel ha causato nella forma del grafico delle funzioni: paragonandolo alla figura III.5 risulta evidente l'importanza della scelta del kernel in funzione del contesto del suo utilizzo.

Come nel caso del linear kernel, imporre la mean function ad un'altra costante comporterà la traslazione delle funzioni sull'asse delle y .

⁶Per approfondire si veda [Duv14]

Per comprendere l'influenza del parametro σ^2 vengono mostrati grafici di funzioni con distribuzione $f \sim \mathcal{GP}(m, k)$ dove $m(x) = 0$ e $k(x, x')$ è il squared-exponential kernel, con $l^2 = 1$ e due valori diversi di σ^2 .

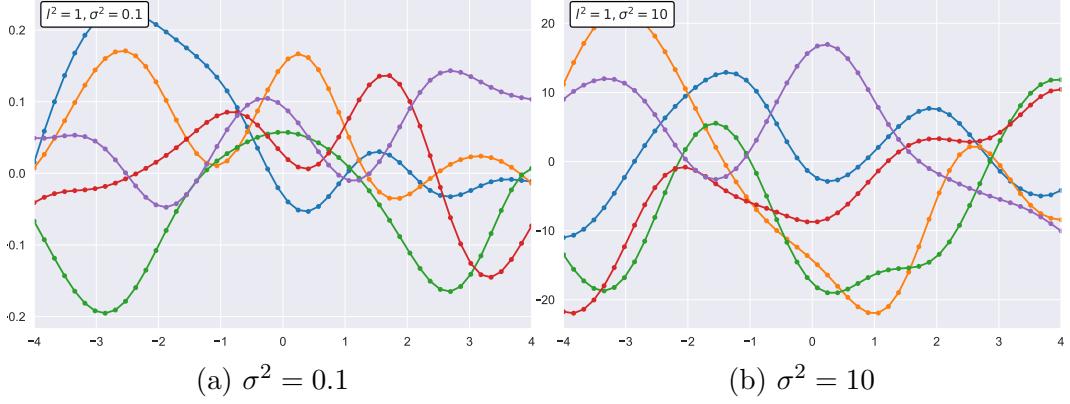


Figure III.12: Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è lo squared-exponential kernel e $l^2 = 1$, il parametro σ^2 viene variato. Codice VII.13.

Nei due casi dunque cambia quanto le funzioni si distanziano dalla retta $x = 0$, cioè proporzionalmente al valore di σ^2 . In realtà σ^2 influisce sulla tendenza delle funzioni a distanziarsi dalla media $m(x)$.

Per comprendere l'influenza del parametro l^2 , vengono di seguito mostrati grafici di funzioni con distribuzione $f \sim \mathcal{GP}(m, k)$ dove $m(x) = 0$ e $k(x, x')$ è il squared-exponential kernel e il parametro l^2 viene variato.

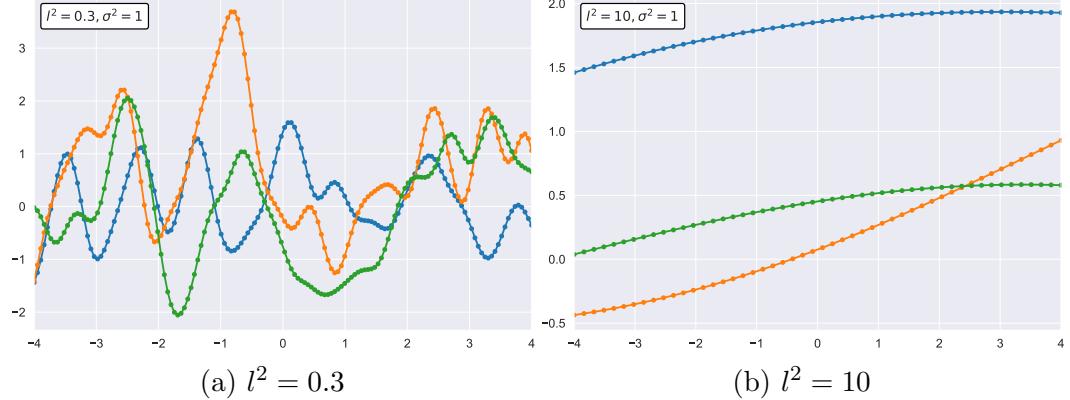


Figure III.13: Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è lo squared-exponential kernel e $\sigma^2 = 1$, il parametro l^2 viene variato. Codice VII.14.

Il parametro l^2 dunque modifica la frequenza di oscillazione delle funzioni.

Per comprendere l'influenza della mean function, vengono di seguito mostrati grafici di funzioni con distribuzione $f \sim \mathcal{GP}(m, k)$ dove $m(x) = x^3$ e $k(x, x')$ lo squared-exponential kernel.

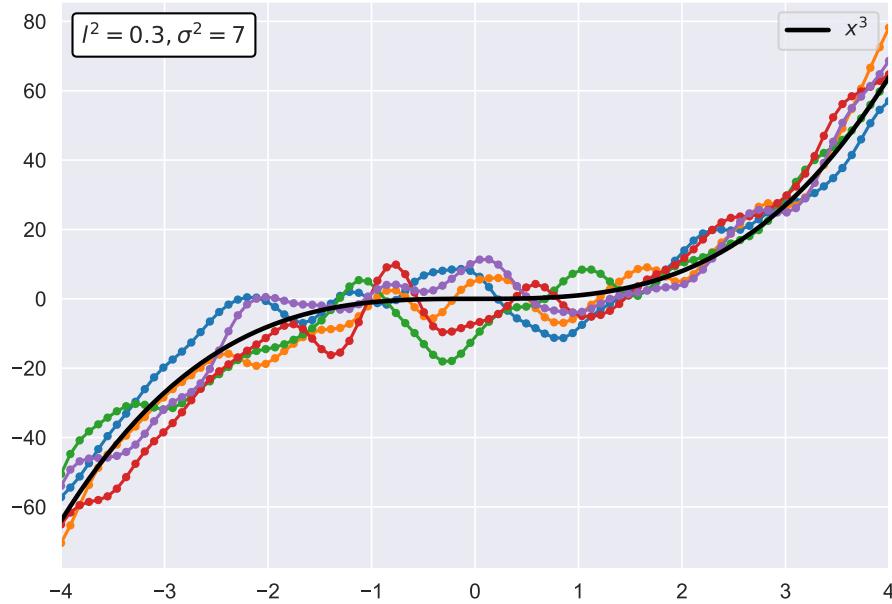


Figure III.14: Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(m, k)$ dove $m(x) = x^3$ e $k(x, x')$ lo squared-exponential kernel, $\sigma^2 = 7$ e $l = 0.3$. Codice VII.15.

Sono stati scelti dei parametri in modo da risaltare i grafici delle funzioni. Con $\sigma^2 = 7$ (quindi un grande distanziamento dalla mean function) e $l^2 = 0.3$ (quindi una grande frequenza di oscillazione) le funzioni tendono a emulare la media $m(x) = x^3$ mantenendo le proprietà indotte dal kernel.

III.3.3 Periodic kernel

Definition III.3.7 (Periodic kernel). Il **periodic kernel** ha forma:

$$k(x, x') = \sigma^2 \exp\left(-\frac{2}{l^2} \sin^2\left(\pi \frac{|x - x'|}{p}\right)\right).$$

Anche questo kernel è dunque isotropico.

Viene riportato il grafico della funzione $k(x, x')$. Si noti che il parametro σ^2 influisce sul picco della funzione come nel *squared-exponential kernel*, similmente il parametro l^2 influenza sulla funzione come nel precedente kernel, il parametro p influenza la periodicità del kernel.

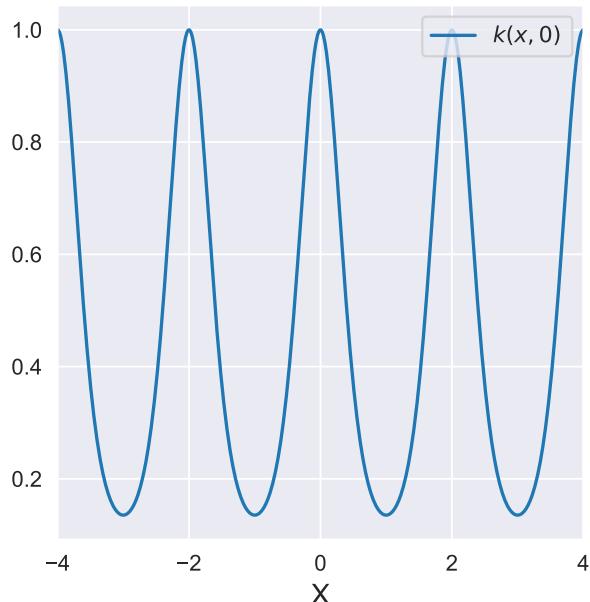


Figure III.15: Grafico di $k(x, x')$ periodic kernel, $\sigma^2 = 1$, $l^2 = 1$, $p = 2$. Codice VII.16.

Vengono di seguito mostrati grafici di funzioni con distribuzione $f \sim \mathcal{GP}(m, k)$ dove $m(x) = 0$ e $k(x, x')$ è il periodic kernel.

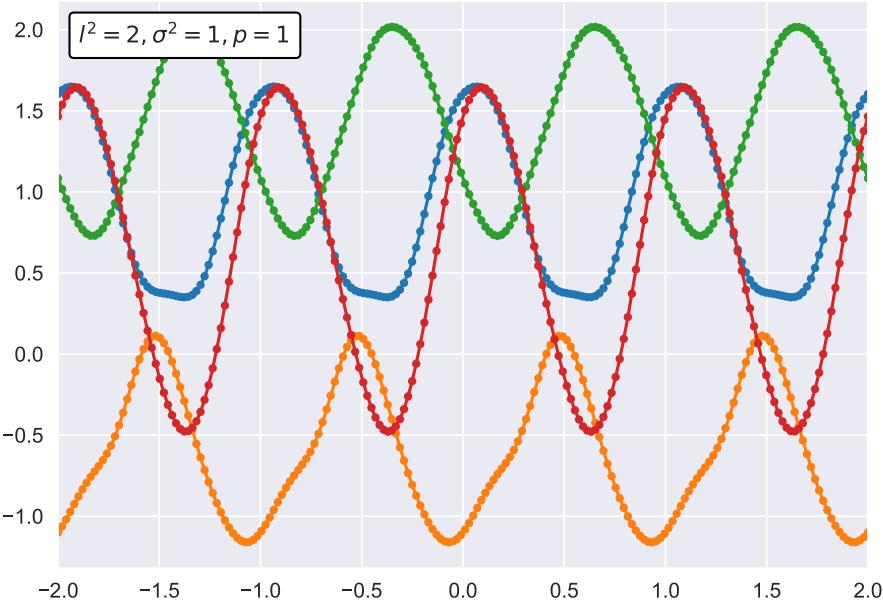


Figure III.16: Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è il periodic kernel e $\sigma^2 = 1$, $l^2 = 2$, $p = 1$. Codice VII.17.

Come suggerisce il nome del kernel, le funzioni hanno andamento periodico. Per comprendere l'influenza del parametro σ^2 , vengono di seguito mostrati grafici di funzioni con distribuzione $f \sim \mathcal{GP}(m, k)$ dove $m(x) = 0$ e $k(x, x')$ è il periodic kernel e il parametro σ^2 viene variato.

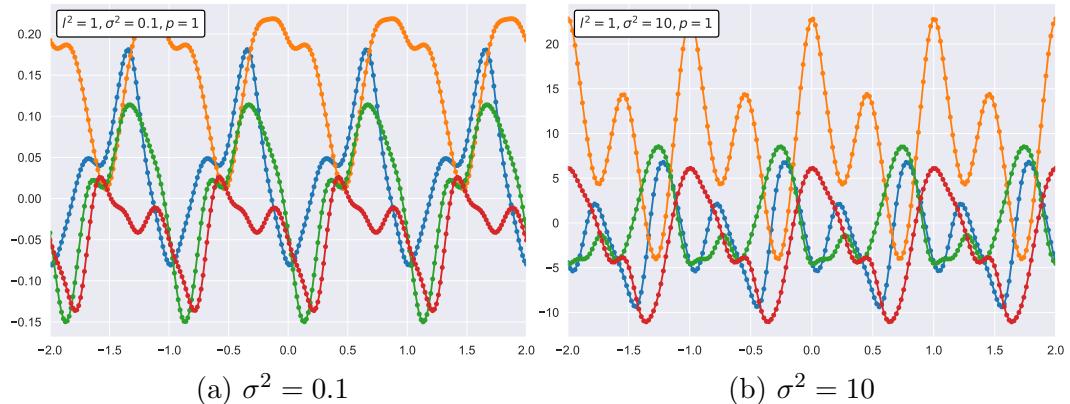


Figure III.17: Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è il periodic kernel e $l^2 = 1, p = 1$, il parametro σ^2 viene variato. Codice VII.18.

Il parametro σ^2 è dunque responsabile dell'allontamento delle funzioni dalla media, esattamente come per lo squared-exponential kernel (la figura III.12 riporta infatti risultati simili).

Per comprendere l'influenza del parametro p , vengono di seguito mostrati grafici di funzioni con distribuzione $f \sim \mathcal{GP}(m, k)$ dove $m(x) = 0$ e $k(x, x')$ è il periodic kernel e il parametro p viene variato.

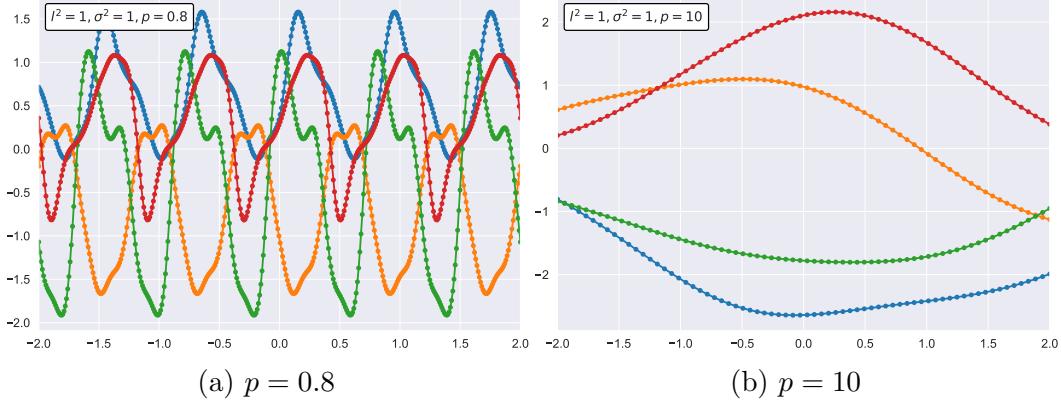


Figure III.18: Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è il periodic kernel e $\sigma^2 = 1$, $l^2 = 1$ e il parametro p viene variato. Codice VII.19.

Il parametro p influenza dunque il periodo delle funzioni.

Per comprendere l'influenza del parametro l^2 , vengono di seguito mostrati grafici di funzioni con distribuzione $f \sim \mathcal{GP}(m, k)$ dove $m(x) = 0$ e $k(x, x')$ è il periodic kernel e il parametro l^2 viene variato.

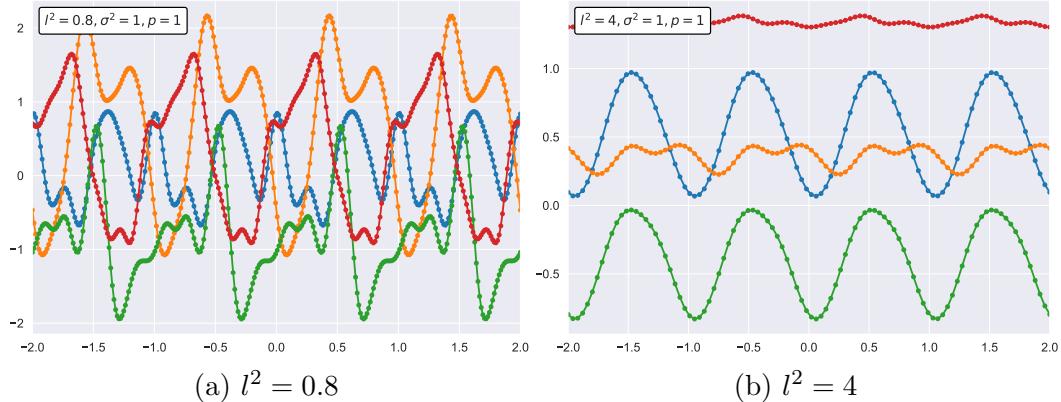


Figure III.19: Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è il periodic kernel e $\sigma^2 = 1$, $p = 1$ e il parametro l^2 viene variato. Codice VII.20.

Il parametro l^2 influenza dunque la "morbidezza" della frequenza delle funzioni.

Per comprendere l'influenza della mean function, vengono di seguito mostrati grafici di funzioni con distribuzione $f \sim \mathcal{GP}(m, k)$ dove $m(x) = x^3$ e $k(x, x')$ il periodic kernel.

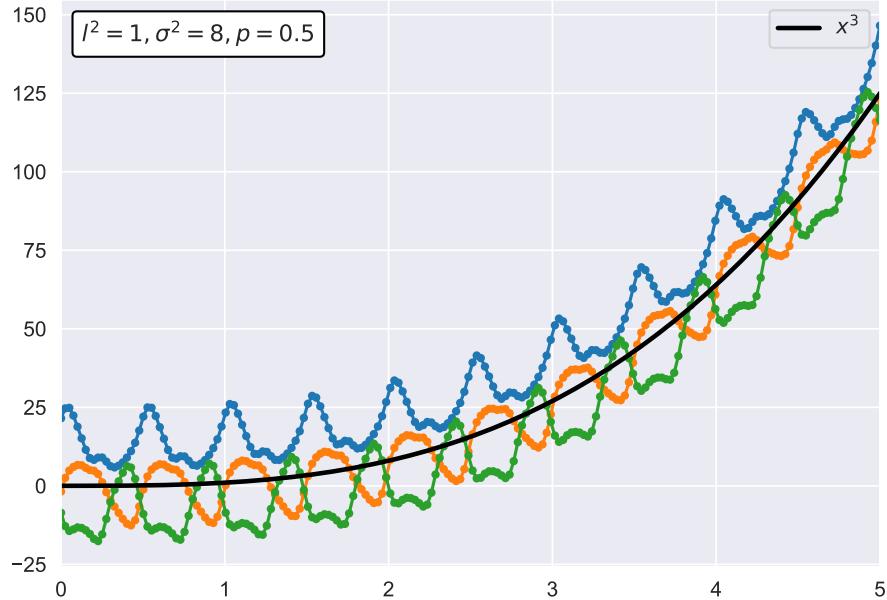


Figure III.20: Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(m, k)$ dove $m(x) = x^3$ e $k(x, x')$ il periodic kernel, $\sigma^2 = 8$, $l^2 = 1$ e $p = 0.5$. Codice VII.21.

Sono stati scelti dei parametri in modo da risaltare i grafici delle funzioni. Con $\sigma^2 = 8$ (quindi un grande distanziamento dalla mean function) e $p = 0.5$ (quindi una grande frequenza di oscillazione) e $l^2 = 1$ (quindi molto "spigolose") le funzioni tendono a emulare la media $m(x) = x^3$ mantenendo le proprietà indotte dal kernel.

III.3.4 Covariance function in più dimensioni

Si possono estendere le covariance function in più dimensioni in diversi modi. Poiché l'estensione a più dimensioni è pressocchè identica per ogni covariance function, viene mostrato il caso del squared-exponential kernel, in quanto verrà utilizzato nella parte di elaborato dedicata al training.

Per estendere il squared-exponential kernel è necessario estendere in più dimensioni la sottrazione $x - x'$. Il modo più semplice è considerare la norma $\|x - x'\|$, un modo più flessibile è considerare $(x - x')^T M (x - x')$ dove M è una matrice. In questo modo è possibile generalizzare il caso:

$$k(x, x') = \sigma^2 \exp\left(-\frac{\|x - x'\|^2}{2l^2}\right),$$

che si può ottenere come:

$$k(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^T M (x - x')}{2}\right)$$

imponendo $M = l^{-2}I$.

La struttura del kernel con matrice permette di dotare ogni dimensione di una differente scala di lunghezza l_i , imponendo $M = \text{diag}(\mathbf{l})^{-2}$, dove $\mathbf{l} = (l_1, \dots, l_n)^T$. Questa caratteristica risulta fondamentale nel training (trattato successivamente nell'elaborato) perché se uno di questi l_i diventa grande, la dimensione corrispondente (che si vedrà corrispondere ad un parametro da ottimizzare) "perde di rilevanza". L'immagine III.21 fornisce un esempio di un comportamento simile.

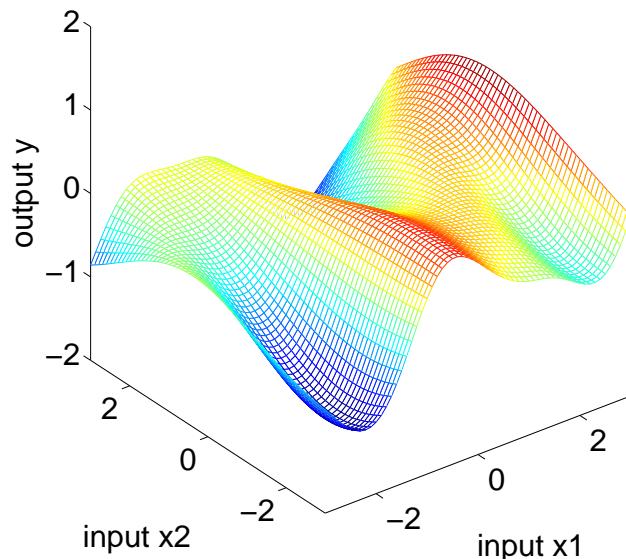


Figure III.21: Grafico di funzione con distribuzione $f \sim \mathcal{GP}(0, k)$ con $k(x, x')$ lo squared-exponential kernel in due dimensioni in cui $M = \text{diag}(1, 3)^{-2}$. La funzione tende a cambiare più velocemente lungo la direzione x_1 che lungo la direzione x_2 . [Mur12]

III.3.5 Combinare le covariance function

Nelle precedenti sezioni è stata spiegata l'influenza della covariance function nella forma del grafico delle funzioni. Tuttavia non è raro dover usare funzioni con forme diverse da quelle imposte dalle covariance function precedentemente introdotte. In tal caso è possibile costruire un nuovo kernel secondo la proposizione III.3.8.

Proposition III.3.8. *Dati due kernel $K_1(x, x')$ e $K_2(x, x')$, per le proprietà di un kernel i seguenti sono ancora kernel:*

$$\begin{aligned} K(x, x') &= c \cdot K_1(x, x') \quad \forall c > 0 \text{ costante} \\ K(x, x') &= f(x)K_1(x, x')f(x') \quad \forall f \text{ funzione} \\ K(x, x') &= q(K_1(x, x')) \quad \forall q \text{ funz. polin. a coeff. non negativi} \\ K(x, x') &= \exp(K_1(x, x')) \\ K(x, x') &= K_1(x, x') + K_2(x, x') \\ K(x, x') &= K_1(x, x') \times K_2(x, x') \end{aligned}$$

Non è negli interessi dell'elaborato studiare a fondo le combinazioni possibili dei kernel introdotti; vengono però portati due semplici casi di combinazioni di kernel a titolo puramente illustrativo.

Squared exponential + periodic kernel

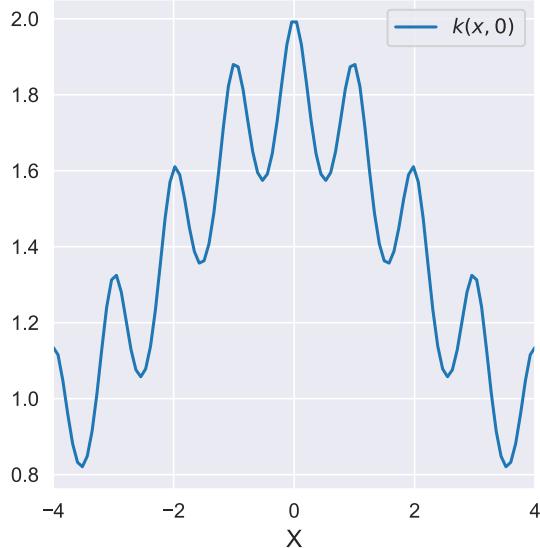


Figure III.22: Grafico di $k(x, x')$ squared-exponential kernel sommato a periodico kernel. $\sigma^2 = 1$, $l = 2$, $p = 1$. Codice VII.22.

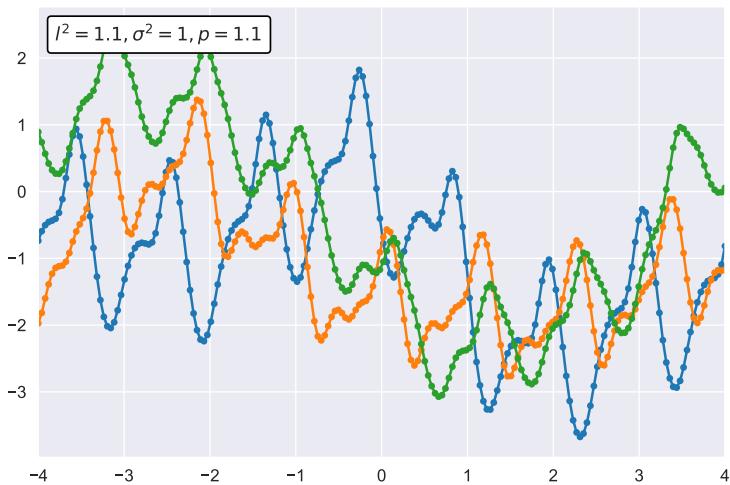


Figure III.23: Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è lo squared-exponential sommato al periodic kernel e $l^2 = 1.1$, $\sigma^2 = 1$, $p = 1.1$. Codice VII.23.

Combinare i due kernel genera dunque funzioni periodiche con delle perturbazioni sulla coppia degli assi.

Linear \times linear kernel

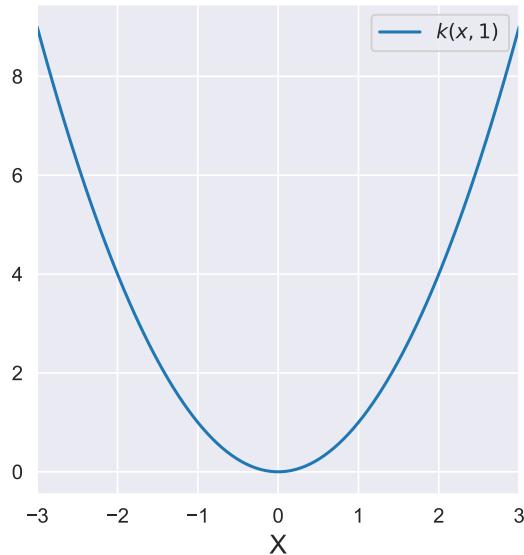


Figure III.24: Grafico di $k(x, x')$ linear kernel moltiplicato a linear kernel.
 $\sigma_b^2 = 0$, $\sigma_v^2 = 1$, $c = 0$, $x' = 1$. Codice VII.24.

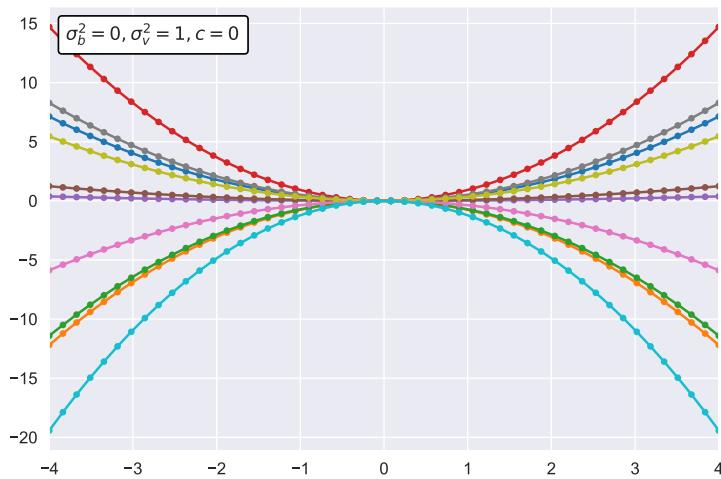


Figure III.25: Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(x, x')$ è il linear kernel moltiplicato al linear kernel e $\sigma_b^2 = 0$, $\sigma_v^2 = 1$, $c = 0$. Codice VII.25.

Combinare i due kernel genera dunque funzioni con comportamento parabolico. In un certo senso era possibile prevedere questo risultato ricordando che il linear kernel genera delle rette.

Esempio reale di covariance function composta

Viene ripreso da [RW06] un esempio di regressione in cui è necessario comporre più kernel function. I dati consistono in concentrazioni medie mensili di CO_2 atmosferica (in parti per milione di volume: *ppmv*) derivate da campioni d'aria raccolti presso l'osservatorio di Mauna Loa, nelle Hawaii, tra il 1958 e il 2003 (con alcuni valori mancanti). L'obiettivo è modellare la concentrazione di CO_2 in funzione del tempo x , cioè predire quella che è conosciuta come *curva di Keeling*. Dalla figura III.26 sono evidenti alcune caratteristiche: una tendenza all'aumento a lungo termine, una pronunciata variazione stagionale e alcune irregolarità minori.

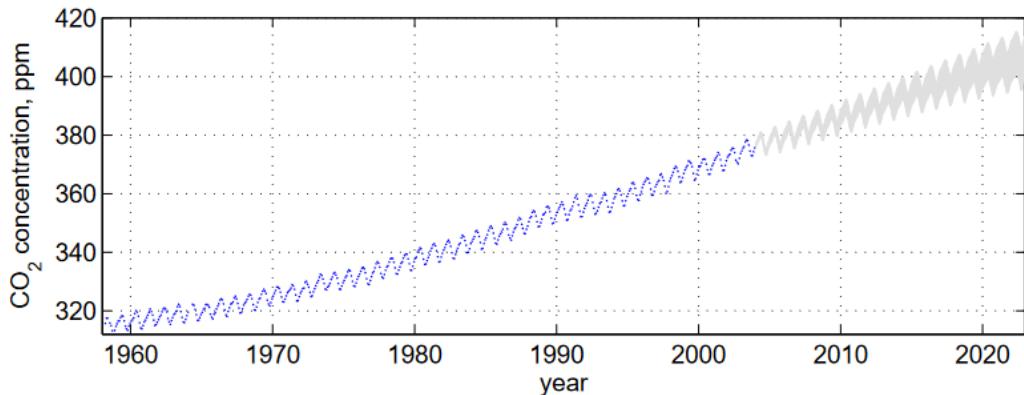


Figure III.26: 545 osservazioni delle medie mensili della concentrazione atmosferica di CO_2 tra il 1958 e il 2003, inoltre viene mostrata la regione di confidenza del 95% per un modello di regressione di processo gaussiano a 20 anni nel futuro. [RW06]

Per modellare l'andamento regolare e crescente a lungo termine viene utilizzato uno squared-exponential kernel:

$$k_1(x, x') = \theta_1^2 \exp\left(-\frac{(x - x')^2}{2\theta_2^2}\right).$$

Viene usato il periodic kernel con un periodo di un anno per modellare la variazione stagionale. Poiché l'andamento stagionale non è esattamente periodico, viene considerato il prodotto con un squared-exponential kernel per consentire un decadimento non esattamente periodico:

$$k_2(x, x') = \theta_3^2 \exp\left(-\frac{(x - x')^2}{2\theta_4^2} - \frac{2\sin^2(\pi(x - x'))}{\theta_5^2}\right).$$

Per modellare le (piccole) irregolarità a medio termine viene utilizzato un *rational quadratic* kernel (non è stato introdotto nell'elaborato):

$$k_3(x, x') = \theta_6^2 \left(1 + \frac{(x - x')^2}{2\theta_8\theta_7^2}\right)^{-\theta_8}.$$

Infine, si modella il rumore come somma di un contributo di uno squared-exponential e di una componente indipendente:

$$k_4(x_p, x_q) = \theta_9^2 \exp\left(\frac{-(x_p - x_q)^2}{2\theta_{10}^2}\right) + \theta_{11}^2 \delta_{pq}.$$

Il kernel complessivo risulta:

$$k(x, x') = k_1(x, x') + k_2(x, x') + k_3(x, x') + k_4(x, x'),$$

dove si hanno $\boldsymbol{\theta} = (\theta_1, \dots, \theta_{11})$ iperparametri⁷. Dopo una fase di training del modello vengono dati dei valori ai θ_i ⁸. La figura III.26 mostra come il modello predice l'andamento della concentrazione atmosferica di CO_2 nei venti anni successivi all'ultima misurazione, mostrando anche la regione di confidenza del novantacinque per cento. Si nota che più si va avanti nel tempo e più la regione di confidenza si allarga.

Intuitivamente il modello sembra seguire bene l'andamento del grafico, seppur più avanti negli anni la regione di incertezza diventa più ampia, informando poco sull'effettiva concentrazione di CO_2 . Nella figura III.27 viene comparata la predizione del modello con i dati reali presi dal sito del Global Monitoring Laboratory.

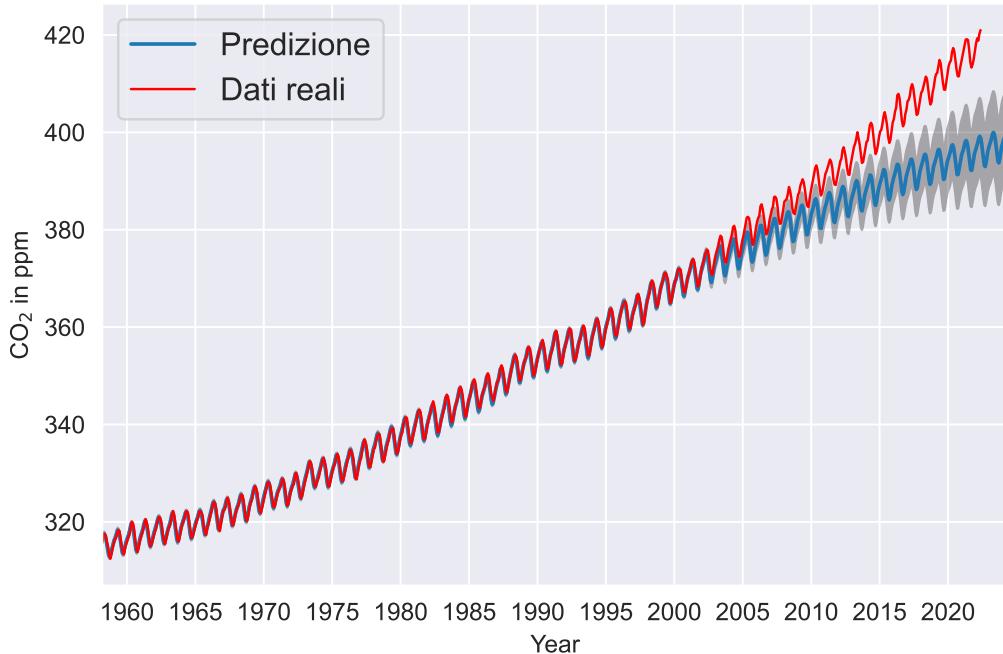


Figure III.27: Comparazione della predizione della concentrazione di CO_2 con i dati reali fino a maggio 2022.

⁷Si veda IV.3.3.

⁸Viene spiegato come questo venga fatto nel capitolo IV.

Si nota dunque che la previsione è stata conservativa nel lungo termine: il grande aumento nello concentrazione di CO_2 è dovuto, secondo alcune fonti, a come la flora ha risposto ai cambiamenti climatici (sicchezza e precipitazioni principalmente), ma anche alle grandi emissioni dovute all'uso di carburanti fossili.

In un certo senso, dunque, è ragionevole che la predizione non sia precisa negli ultimi anni poiché avrebbe dovuto prevedere eventi (dai cambiamenti climatici ai diversi tassi di consumo di carburante) che non erano presenti negli anni in cui il processo gaussiano "ha imparato".

In figura III.28 viene ingrandita l'immagine precedente dal 1995 al 2022, dunque enfatizzando il periodo di tempo che il processo gaussiano ha predetto.

Si nota che dopo il 2003 la predizione è piuttosto imprecisa, non riuscendo a stare dietro ai veloci cambiamenti umani (in termini di inquinamento).

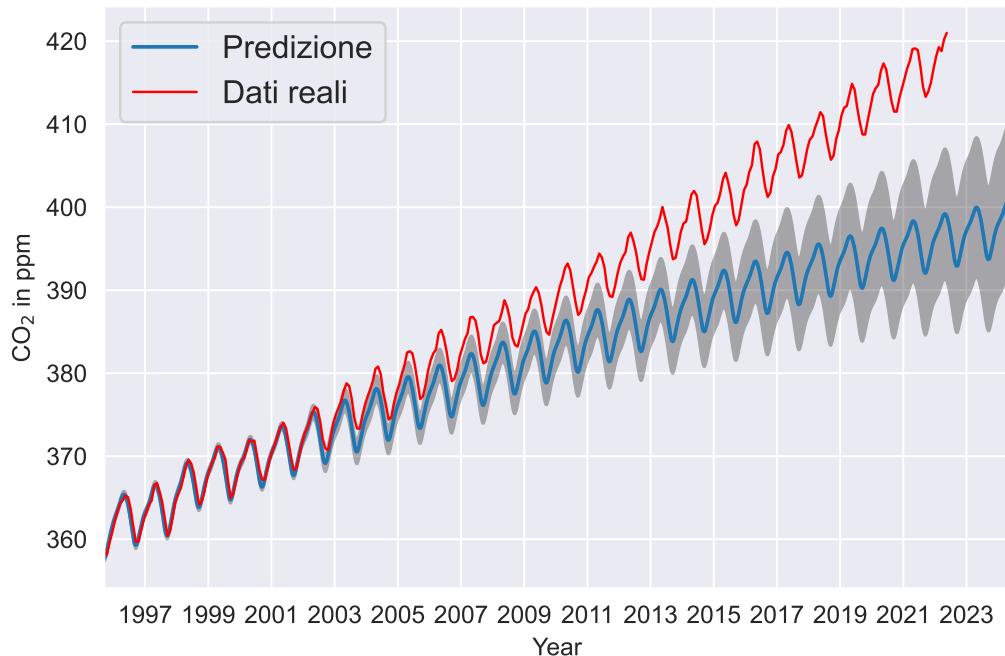


Figure III.28: Comparazione della previsione della concentrazione di CO_2 con i dati reali fino dal 1995 a maggio 2022.

III.4 Predizioni con osservazioni senza rumore

In questa sezione viene spiegato come sfruttare le informazioni fornite dai *training data* per generare una funzione che incorpori la conoscenza a priori nel caso di osservazioni senza rumore.

Definition III.4.1 (Noise-free training set). Il **noise-free training set** è l'insieme di osservazioni *noise-free* definito come: $\mathcal{D} = \{(x_n, y_n) : n = 1, \dots, N\}$ dove $y_n = f(x_n)$ è il valore osservato della funzione f nel punto x_n .

Lo scopo del (noise-free) training set è quello di definire un insieme di punti $\mathcal{P} = \{x_n : n = 1, \dots, N\}$ dei quali si conosce il valore della funzione per richiedere al processo gaussiano di generare funzioni che interpolino questo insieme di punti (dunque che abbiano valore y_n su ogni $x_n \in \mathcal{P}$ senza incertezza).

Definition III.4.2 (Test set). Il **test set** è l'insieme di valori $X_* = \{x_n : n = 1, \dots, N^*\}$ dei quali si vuole avere la predizione, cioè l'output della funzione.

Pragmaticamente, quello che si sta facendo definendo il *training set* \mathcal{D} è aggiungere i punti di \mathcal{P} all'insieme di punti su cui valutare la covariance function per costruire la matrice di covarianza⁹. Tramite il processo di condizionamento si ottiene la distribuzione *a posteriori*, la distribuzione cioè delle funzioni che interpolano i punti del training set, come illustrato nella figura III.29.

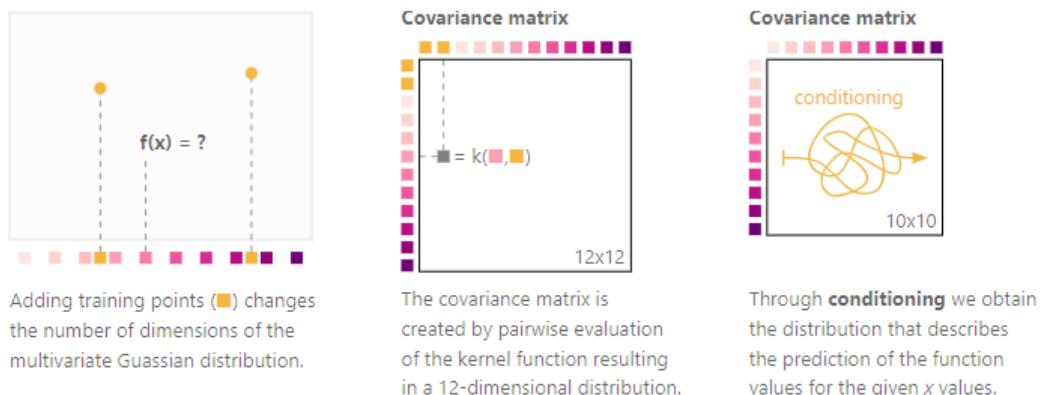


Figure III.29: Spiegazione grafica di come viene incorporata la conoscenza a priori [GKD19].

⁹Come accennato in 1, per ottenere un grafico è necessario partire da un dominio finito. Per questo motivo si parla di *matrice di covarianza*.

Si consideri ora una funzione distribuita come $f \sim \mathcal{GP}(m, k)$. Si consideri ora $X^* = \{x_i^* : i = 1, \dots, N^*\}$ un *test set* di cui si vuole predire gli output $\mathbf{f}^* = [f(x_1^*), \dots, f(x_{N^*}^*)]$; siano inoltre $\mathbf{f} = [f(x_1), \dots, f(x_N)]$ gli output del training set dove $x_i \in \mathcal{P}$. Ricordando che gli output della funzione f seguono una distribuzione gaussiana, è possibile ricavare la distribuzione congiunta:

$$\begin{pmatrix} \mathbf{f} \\ \mathbf{f}^* \end{pmatrix} = \mathcal{N} \left(\begin{pmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}_* \end{pmatrix}, \begin{pmatrix} \mathbf{K}_{X,X} & \mathbf{K}_{X,X^*} \\ \mathbf{K}_{X^*,X} & \mathbf{K}_{X^*,X^*} \end{pmatrix} \right)$$

dove $\boldsymbol{\mu} = \begin{pmatrix} m(x_1) \\ \vdots \\ m(x_N) \end{pmatrix}$, $\boldsymbol{\mu}_* = \begin{pmatrix} m(x_1^*) \\ \vdots \\ m(x_{N^*}^*) \end{pmatrix}$, $\mathbf{K}_{X,X} = k(X, X)$ è una matrice

$N \times N$, $\mathbf{K}_{X,X^*} = k(X, X^*)$ è una matrice $N \times N^*$, $\mathbf{K}_{X^*,X} = k(X^*, X)$ è una matrice $N^* \times N$, $\mathbf{K}_{X^*,X^*} = k(X^*, X^*)$ è una matrice $N^* \times N^*$.

Dalla proposizione II.2.5 ricaviamo la distribuzione condizionata di $\mathbf{f}^*|X^*, \mathcal{D}$:

$$\mathbf{f}^*|X^*, \mathcal{D} \sim \mathcal{N}(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*)$$

dove:

$$\begin{aligned} \boldsymbol{\mu}^* &= m(X^*) + \mathbf{K}_{X,X^*}^T \mathbf{K}_{X,X}^{-1} (\mathbf{f} - m(X)) \\ \boldsymbol{\Sigma}^* &= \mathbf{K}_{X^*,X^*} - \mathbf{K}_{X,X^*}^T \mathbf{K}_{X,X}^{-1} \mathbf{K}_{X,X^*} \end{aligned}$$

In figura III.30 viene mostrato un esempio grafico di interpolazione di sei punti.

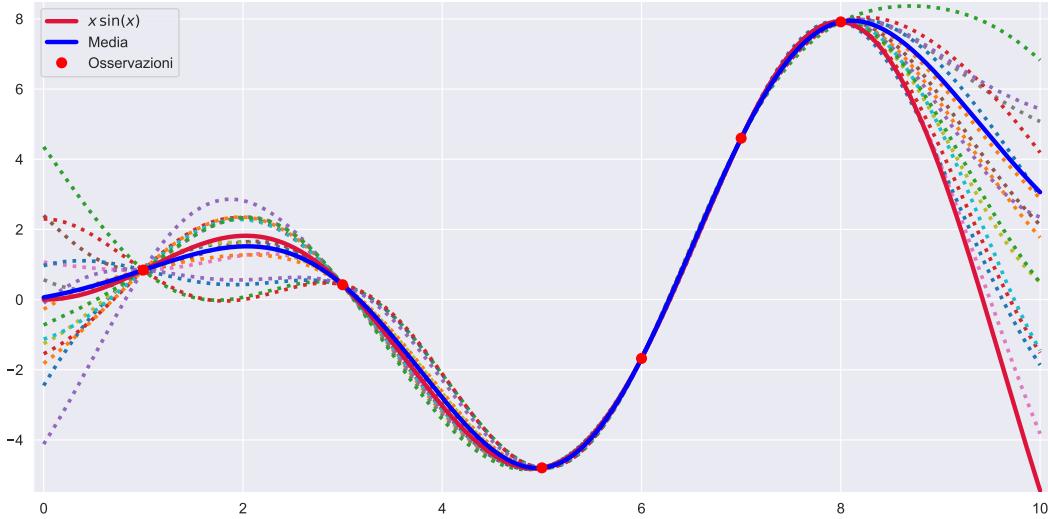


Figure III.30: Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(\cdot, \cdot)$ è il squared-exponential kernel; il processo gaussiano è stato condizionato per interpolare sei punti. Viene mostrata in rosso la funzione da cui sono stati scelti i punti da interpolare, in blu la media del processo gaussiano condizionato, come linee tratteggiate alcuni sample del processo gaussiano. Codice VII.26.

La media del processo gaussiano condizionato per interpolare i sei punti è la più affidabile nel predire la funzione $x \cdot \sin(x)$ da cui sono stati generati i punti da interpolare. Le funzioni con distribuzione il processo gaussiano condizionato hanno tutte la proprietà di interpolazione dei suddetti punti, tuttavia nel resto del piano si distanziano più facilmente dalla media, anche se tendono a restare nei suoi dintorni. Nella figura III.31 vengono sfruttate le informazioni della matrice di covarianza a posteriori per disegnare la regione di confidenza del 95%, dentro la quale la maggior parte dei sample risiede.

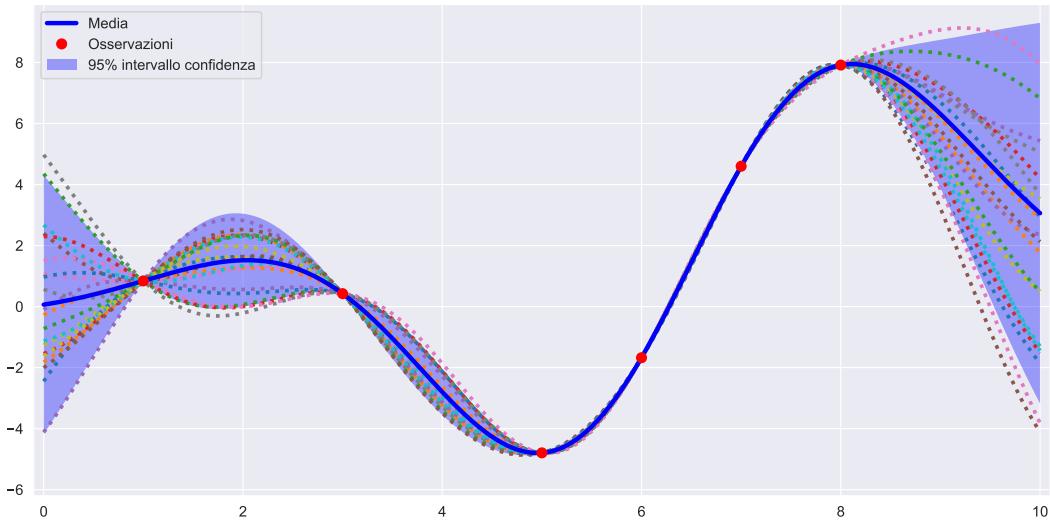


Figure III.31: Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(\cdot, \cdot)$ è il squared-exponential kernel; il processo gaussiano è stato condizionato per interpolare sei punti. Viene mostrata in blu la regione di confidenza al 95%, in blu la media del processo gaussiano condizionato e come linee tratteggiate alcuni sample. Codice VII.27.

Come anticipato, all'interno dell'area data da $\mu^*(x_i) \pm 1.96\Sigma^*(x_i, x_i)$ (regione di confidenza al 95%) risiedono la maggior parte dei sample della distribuzione a posteriori. Più ci si distanzia dai punti interpolati e più facilmente le funzioni si discostano dalla media, tendendo ad uscire dall'area di incertezza, come si osserva ad esempio nella zona nei pressi di $x = 10$ della figura III.31.

III.5 Predizioni con osservazioni rumorose

In questa sezione viene spiegato come sfruttare le informazioni fornite dai *training data* per generare una funzione che incorpori la conoscenza a priori nel caso di osservazioni con rumore.

Definition III.5.1 (Noisy training set). Il **noisy training set** è l'insieme di osservazioni *noisy* (cioè con una componente di errore) definito come: $\mathcal{D} = \{(x_n, y_n) : n = 1, \dots, N\}$ dove $y_n = f(x_n) + \epsilon$ è il valore osservato della funzione f nel punto x_n con una componente di rumore $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$ indipendente e identicamente distribuito.

In questo caso si ha:

$$\text{Cov}(y_p, y_q) = k(x_p, x_q) + \sigma_n^2 \delta_{pq},$$

o equivalentemente:

$$\text{Cov}(\mathbf{y}) = k(X, X) + \sigma_n^2 I.$$

Viene usata una notazione analoga alla sezione III.4: $X^* = \{x_i^* : i = 1, \dots, N^*\}$ è il *test set* di cui si vogliono predire gli output $\mathbf{f}^* = [f(x_1^*), \dots, f(x_{N^*}^*)]$; $\mathbf{y} = [f(x_1) + \epsilon, \dots, f(x_N) + \epsilon]$ gli output del training set.

Si procede come nel caso della predizione con osservazioni senza rumore:

$$\begin{pmatrix} \mathbf{y} \\ \mathbf{f}^* \end{pmatrix} = \mathcal{N} \left(\begin{pmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}_* \end{pmatrix}, \begin{pmatrix} \mathbf{K}_{X,X} + \sigma_n^2 I & \mathbf{K}_{X,X^*} \\ \mathbf{K}_{X^*,X} & \mathbf{K}_{X^*,X^*} \end{pmatrix} \right)$$

Si ricava:

$$\mathbf{f}^* | X^*, \mathcal{D} \sim \mathcal{N}(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*)$$

dove:

$$\begin{aligned} \boldsymbol{\mu}^* &= m(X^*) + \mathbf{K}_{X,X^*}^T (\mathbf{K}_{X,X} + \sigma_n^2 I)^{-1} (\mathbf{y} - m(X)) \\ \boldsymbol{\Sigma}^* &= \mathbf{K}_{X^*,X^*} - \mathbf{K}_{X,X^*}^T (\mathbf{K}_{X,X} + \sigma_n^2 I)^{-1} \mathbf{K}_{X,X^*} \end{aligned}$$

In figura III.32 viene mostrato un esempio grafico di come un processo gaussiano interpreti le informazioni date da osservazioni con rumore per predire una funzione. Come ci si può aspettare, essendo le osservazioni in questo caso con rumore, il processo gaussiano ha bisogno di più osservazioni per poter predire l'andamento della funzione. Non verificandosi interpolazione, il processo gaussiano predice la funzione con minore precisione.

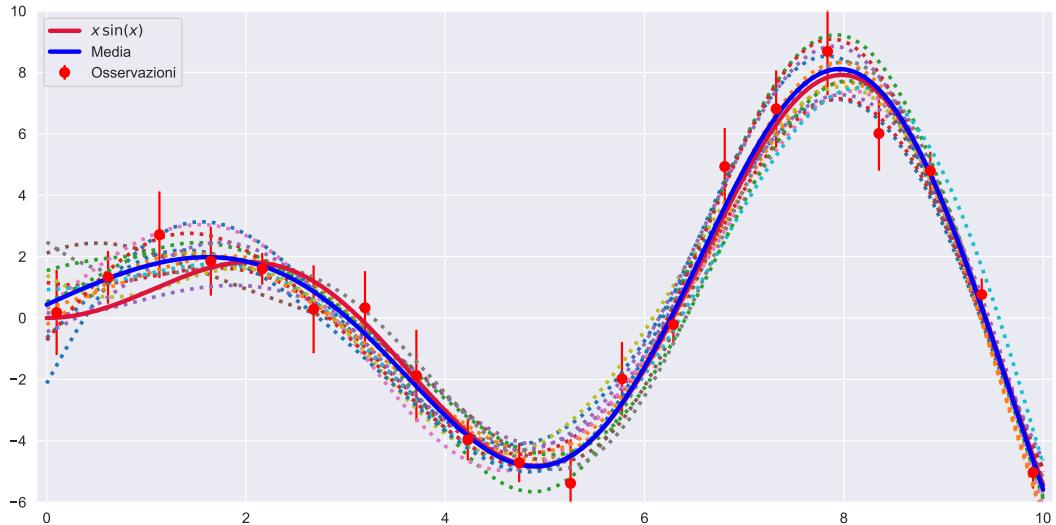


Figure III.32: Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(\cdot, \cdot)$ è il squared-exponential kernel; il processo gaussiano è stato condizionato per predire una funzione a partire dalle sue osservazioni rumorose. Viene mostrata in rosso la funzione da predire, in rosso i punti osservati della funzione con le barre rappresentanti il rumore, in blu la media del processo gaussiano condizionato, come linee tratteggiate alcuni sample del processo gaussiano. Codice VII.28.

Anche in questo caso la media del processo gaussiano condizionato rappresenta la funzione che meglio predice $x \cdot \sin(x)$. Poiché le osservazioni presentano un errore, non si ha interpolazione dei punti seppur, tendenzialmente, la media e i sample del processo gaussiano tocchino almeno le barre di errore di ogni punto. Sono stati presi in considerazione più punti del caso noise-free poiché con meno punti non si sarebbe ottenuto un risultato così "preciso" (comunque con un comportamento diverso da quello con osservazioni senza rumore).

Se nel caso senza rumore i sample tendevano a rimanere nei pressi della media nelle vicinanze di ogni punto interpolato, qui la vicinanza alla media dipende sia dai punti che dal loro errore: nei pressi di punti con minore rumore (cioè con barre più corte) le funzioni tendono ad avvicinarsi alla media. Viceversa, nei pressi di funzioni con un grande errore (barre più lunghe) le funzioni tendono ad allontanarsi con più facilità dalla media.

Anche in questo caso è possibile disegnare la regione di confidenza del 95%. La regione di confidenza ha ampiezza proporzionale alla lunghezza delle barre delle osservazioni; di conseguenza la regione di confidenza ha comportamento decisamente diverso dal caso noise-free.

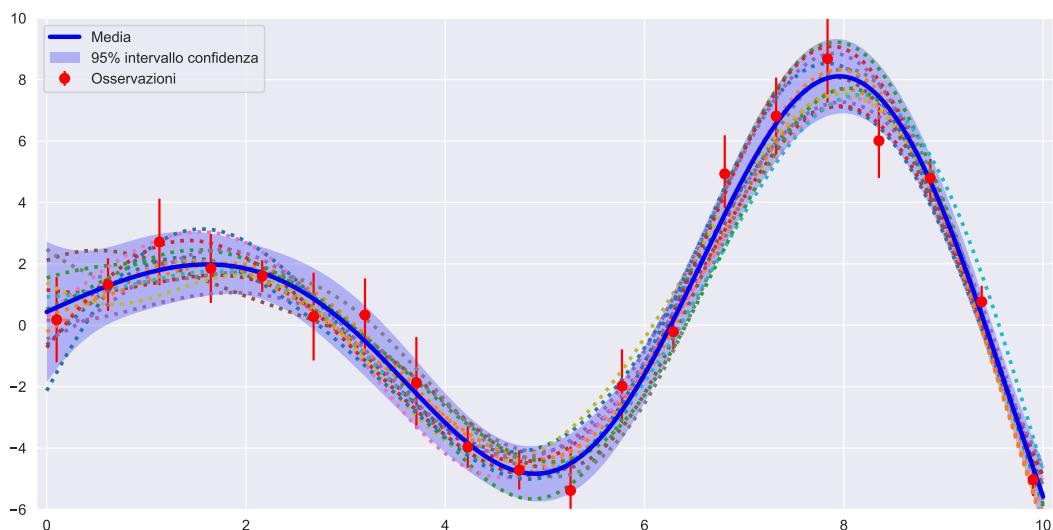


Figure III.33: Grafico di funzioni con distribuzione $f \sim \mathcal{GP}(\mathbf{0}, k)$ dove $k(\cdot, \cdot)$ è il squared-exponential kernel; il processo gaussiano è stato condizionato per predire una funzione a partire dalle sue osservazioni rumorose. Viene mostrata in blu la regione di confidenza al 95%, in rosso con le barre di errore le osservazioni, in blu la media del processo gaussiano condizionato e come linee tratteggiate alcuni sample. Codice VII.29.

IV

Machine learning

Il capitolo spiega in breve la teoria dell'**apprendimento supervisionato** focalizzandosi nel contesto di regressione con processi gaussiani. Viene introdotto teoricamente il modello statistico che verrà usato, nel capitolo VI, per produrre risultati legati al modello Windkessel visto nel capitolo V.

Le fonti usate per la stesura del capitolo sono: [Mur22], [Wik22e], [Wik22d], [Pyt22], [Wan22], [SSK18], [Mur12], [Gel+95], [Wik22c], [Rud16], [KB17], [DHS11], [Bot12].

Il sole è sorto e tramontato per miliardi di anni. Il sole è tramontato anche stanotte. Con un'elevata probabilità, il sole domani sorgerà. Ma questo numero è molto più grande per colui che, vedendo nella totalità dei fenomeni il principio che regola i giorni e le stagioni, si rende conto che nulla al momento attuale può arrestarne il corso.

Pierre Simon Laplace

IV.1 Introduzione al machine learning

Il **machine learning** (in italiano *apprendimento automatico*) è una branca dell'**intelligenza artificiale**, la quale è una disciplina che studia i fondamenti teorici e le tecniche che consentono la progettazione di sistemi capaci di fornire all'elaboratore prestazioni che sembrerebbero essere di pertinenza esclusiva dell'intelligenza umana.

Di questa vasta branca, l'elaborato si interessa ai soli problemi di regressione, contrapposti ai problemi di classificazione.

In termini più pratici, il principale scopo del machine learning è lo studio e la costruzione di algoritmi che possano imparare a fare previsioni su dei dati. Tali algoritmi funzionano basandosi su decisioni guidate dal dataset attraverso la costruzione di un modello matematico.

Nell'elaborato viene usato uno specifico tipo di apprendimento che è quello *supervisionato*.

Definition IV.1.1 (Apprendimento supervisionato). L'**apprendimento supervisionato** è una tecnica di apprendimento automatico che mira a istruire un sistema informatico in modo da consentirgli di elaborare previsioni sulla base di una serie di esempi costituiti da coppie di input e output.

IV.2 Dataset

Per ottimizzare la costruzione di algoritmi predittivi, i dati di input vengono divisi in più set di dati con ruoli diversi. Tipicamente viene usata una specifica partizione dei dati in tre dataset: **training**, **validation** e **test set**.

Definition IV.2.1 (Training set). Il **training set** è un insieme di esempi utilizzati durante il processo di apprendimento per determinare (o imparare) le combinazioni ottimali di parametri.

In pratica, è sui dati del training set che viene eseguito il metodo di ottimizzazione scelto, aggiornando quindi i valori dei parametri o degli iperparametri.

Definition IV.2.2 (Validation set). Il **validation set** è un set indipendente di dati usato per valutare il modello addestrato sul training set.

La valutazione (o *validazione*) del modello sul validation set porta a decidere quali siano i migliori valori per i parametri (o iperparametri) in base alla performance sul validation set, appunto. Nel caso dell'elaborato viene usato un early-stopper, che sceglie come miglior modello quello con minore errore subito prima che si verifichi overfitting.

Definition IV.2.3 (Test set). Un **test set** è un set indipendente di dati dal training set usato solo per valutare le prestazioni del modello.

Cioè il test set viene usato per valutare su un terzo set di dati indipendente il modello scelto in base alla performance sul validation set. Si ottengono così caratteristiche di prestazione come la precisione, la sensibilità, la specificità... Questo dataset è importante perché permette di valutare la performance del modello su un terzo insieme di dati indipendente dai precedenti evitando il rischio di overfitting: se un modello addestrato sul training test si adatta anche al test set, si è verificato un overfitting minimo.

È stato più volte citato il problema dell'**overfitting**, viene quindi riportata la definizione.

Definition IV.2.4 (Overfitting). L'**overfitting** è la generazione di un modello (o un'analisi) che corrisponde troppo strettamente a un particolare insieme di dati e può quindi non riuscire a prevedere osservazioni future in modo affidabile.

L'overfitting può verificarsi per esempio includendo troppi parametri regolabili o usando un approccio troppo complicato, come mostrato in figura IV.1. Chiaramente, quando si confrontano diversi tipi di modelli la complessità deve tenere conto dell'influenza di ogni parametro sull'output¹.

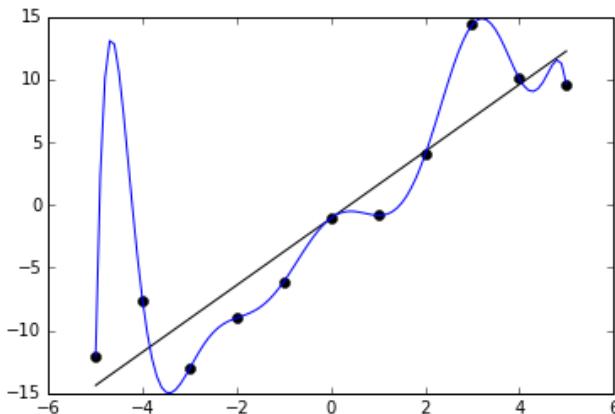


Figure IV.1: Esempio di overfitting. I dati (approssimativamente lineari) sono approssimati da una funzione lineare e da una polinomiale. Anche se la funzione polinomiale fornisce un adattamento quasi perfetto, ci si può aspettare che la funzione lineare generalizzi meglio i dati. [Wik22d]

¹Si noti che si può incorrere nel problema opposto usando un approccio troppo semplice: l'underfitting. Ad esempio cercando di approssimare con una regressione lineare un campione con un andamento parabolico.

L’overfitting è particolarmente probabile nei casi in cui l’apprendimento è stato eseguito troppo a lungo o in cui ci sono pochi dati per l’apprendimento, facendo sì che il modello si adatti a caratteristiche casuali molto specifiche dei dati di formazione che non hanno alcuna relazione causale con l’output. In questo caso di overfitting, la performance sul training set continua ad aumentare mentre la performance sul validation set peggiora, come mostrato in figura IV.2.

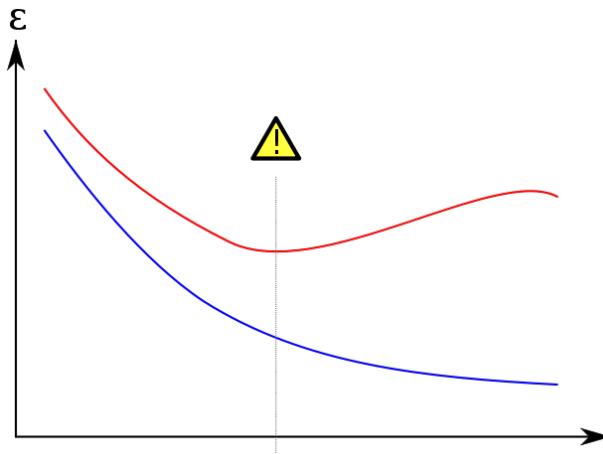


Figure IV.2: Overfitting nell’apprendimento supervisionato. Il training error (errore sul training set) è mostrato in blu, il validation error (errore su validation set) in rosso, entrambi in funzione del numero di cicli di training. [Wik22d]

IV.2.1 Loss function

Finora è stato delineato il problema che interessa l’elaborato: cercare una funzione f per predire l’output Y a partire dai valori dell’input X . Per farlo è necessaria una funzione di perdita, in inglese *loss function*, $L(Y, f(X))$ per penalizzare gli errori di previsione.

Definition IV.2.5 (Loss function). Una **loss function** è una funzione in forma $\mathcal{L}(y_{\text{true}}, y_{\text{guess}})$ che definisce la perdita subita (o l’errore commesso) prediendo il valore y_{guess} quando il valore vero è y_{true} , fornendo quindi una valutazione delle capacità predittive del modello.

Sarà questa la funzione da minimizzare con un algoritmo di ottimizzazione per l’adattamento degli iperparametri del processo gaussiano. Evidentemente la scelta della loss function dipende dal contesto.

IV.3 Modelli parametrici e non parametrici

Vi è una importante distinzione da fare sui modelli statistici: modelli parametrici e modelli non parametrici. Il tipo di modello, infatti, modifica notevolmente l'approccio teorico da usare e quindi i risultati applicativi.

IV.3.1 Modelli parametrici

I **modelli parametrici** presuppongono che la distribuzione dei dati possa essere modellata in termini di un insieme finito di parametri.

Un classico esempio è quello della regressione in cui, date delle osservazioni, si cerca di stimare un'eventuale relazione funzionale esistente tra la variabile dipendente e le variabili indipendenti. Se si assume un modello di regressione lineare, in cui la relazione è espressa dalla funzione $f(x) = \theta_1 + \theta_2 x$ dove x sono i dati di input, è necessario trovare i valori dei parametri θ_1 e θ_2 per definire f . In molti casi, l'assunzione del modello lineare non è sufficiente, ed è necessario un modello polinomiale (con più parametri): $f(x) = \theta_1 + \theta_2 x + \theta_3 x^2$.

Quindi dato un dataset D con n punti osservati, dopo il processo di *training* si assume che tutte le informazioni sulla relazione funzionale siano state catturate dai parametri θ_i . Quando si effettuano regressioni utilizzando modelli parametrici, la complessità e la flessibilità dei modelli è limitata dal numero di parametri.

IV.3.2 Modelli non parametrici

Contrariamente a quanto si possa pensare, un **modello non parametrico** non è un modello privo di parametri.

Intuitivamente, se il numero di parametri di un modello cresce con la dimensione del dataset di dati osservati, si tratta di un modello non parametrico. In via teorica, questo permette al modello di avere infiniti parametri e quindi di non dipendere da una struttura della funzione f rigida, aumentandone la flessibilità.

Di interesse per l'elaborato è la regressione tramite processi gaussiani, che segue un approccio bayesiano non parametrico (visto nella sezione III.4). È in grado di *apprendere* un'ampia varietà di relazioni tra input e output utilizzando un numero teoricamente infinito di parametri e lasciando che i dati determinino il livello di complessità attraverso i mezzi di inferenza bayesiana.

Si veda IV.3.3 per un chiarimento sulla differenza tra parametri e iperparametri nella regressione tramite processi gaussiani.

IV.3.3 Struttura gerarchica

È comune usare una struttura gerarchica dei parametri e degli iperparametri nei modelli di regressione.

I parametri risiedono al livello più basso. Per esempio, nel caso della regressione lineare, i parametri sono i θ_i , o nel caso di un modello di rete neurale i pesi associati ai neuroni.

Al secondo livello ci sono gli *iperparametri* che controllano la distribuzione dei parametri al livello inferiore e dunque il cui valore è usato per controllare il processo di apprendimento.

Nel caso dei processi gaussiani, poiché sono un modello non parametrico, non è ovvio quali siano i parametri del modello. Si possono considerare come parametri i valori della funzione (priva di rumore) valutata sui dati osservati. Quindi sia $X = \{(x_i, y_i) : i \in I\}$ l'insieme dei dati osservati, gli elementi y_i possono essere considerati i parametri del modello. Evidentemente, più è grande X , più parametri ci sono. In termini pratici, si è visto in III.4 come i dati osservati corrispondano alle informazioni sulla forma della funzione, quindi come la precisione dipenda dal numero di dati osservati.

È anche possibile dare una diversa lettura dei parametri del modello usando un'interpretazione teorica diversa (*weight-space view*), che non è stata introdotta nell'elaborato.²

Gli iperparametri, invece, sono i parametri della mean function e della kernel function. Pragmaticamente, per fare regressione è necessario lavorare sulla stima degli iperparametri.

²Per approfondire: [RW06]

IV.4 Ottimizzazione degli iperparametri

IV.4.1 Inferenza bayesiana

L'**inferenza bayesiana** è un metodo di inferenza statistica in cui il teorema di Bayes viene utilizzato per aggiornare la probabilità di un'ipotesi man mano che si rendono disponibili ulteriori informazioni.

Il processo bayesiano di analisi dei dati può essere idealizzato suddividendolo nelle seguenti tre fasi:

1. Creazione di un modello di probabilità completo

Il modello consiste in una distribuzione di probabilità congiunta per tutte le quantità relative al problema e deve essere coerente con le conoscenze del problema scientifico di base.

2. Condizionamento dei dati osservati

Viene calcolata e interpretata l'appropriata distribuzione a posteriori, ovvero la distribuzione di probabilità delle quantità non osservate condizionata rispetto i dati osservati.

3. Valutazione dell'adattamento del modello

Viene valutato quanto il modello si adatta ai dati e se le conclusioni sostanziali sono ragionevoli. In questa fase è possibile modificare o espandere il modello e ripetere le tre fasi.

L'approccio bayesiano può essere interpretato come una formalizzazione del metodo scientifico, in cui si cerca di aggiornare la conoscenza scientifica attraverso esperimenti e osservazioni.

Inoltre, questo approccio statistico gode di una flessibilità tale da essere utilizzabile in problemi complessi, anche con molti parametri.

Nell'ambito dei processi gaussiani viene studiato l'uso dell'inferenza bayesiana nell'ottimizzazione degli iperparametri.

IV.4.2 Teorema di Bayes

Theorem IV.4.1 (Teorema di Bayes). *Siano A, B eventi, sia $\mathbb{P}(B) \neq 0$, allora:*

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(B|A)\mathbb{P}(A)}{\mathbb{P}(B)}.$$

Proof. Dalla definizione di probabilità condizionata si ha: $\mathbb{P}(A \cap B) = \mathbb{P}(A|B)\mathbb{P}(B)$, da cui segue $\mathbb{P}(A|B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}$ se $\mathbb{P}(B) \neq 0$; analogamente: $\mathbb{P}(B|A) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(A)}$ se $\mathbb{P}(A) \neq 0$. Sostituendo $\mathbb{P}(A \cap B)$ nell'espressione di $\mathbb{P}(A|B)$ si conclude. \square

Nel contesto bayesiano lo scopo dell'analisi di un modello è quello di trarre conclusioni statistiche su un parametro θ (o analogamente su un vettore di parametri³) o su dati non osservati; le suddette conclusioni vengono formulate in termini di probabilità condizionate al valore osservato di y . Per poter fare affermazioni di probabilità su θ osservati i dati y è necessario iniziare con un modello che fornisca una distribuzione di probabilità congiunta per θ e y , fattorizzabile in due componenti: $\mathbb{P}(\theta)$, cioè la distribuzione a priori che descrive la conoscenza sulla distribuzione di θ prima di osservare i dati, e $\mathbb{P}(y|\theta)$, cioè la distribuzione dei dati (o campionaria):

$$\mathbb{P}(\theta, y) = \mathbb{P}(\theta)\mathbb{P}(y|\theta).$$

Applicando il teorema di Bayes si ottiene la *distribuzione a posteriori*:

$$\mathbb{P}(\theta|y) = \frac{\mathbb{P}(\theta)\mathbb{P}(y|\theta)}{\mathbb{P}(y)},$$

nella quale il denominatore può essere riscritto, per il teorema della probabilità assoluta (in inglese *law of total probability*), come:

$$\mathbb{P}(y) = \begin{cases} \sum_{\theta} \mathbb{P}(\theta)\mathbb{P}(y|\theta) & \text{in base a } \theta. \\ \int \mathbb{P}(\theta)\mathbb{P}(y|\theta)d\theta & \end{cases}$$

Poiché $\mathbb{P}(y)$ non dipende da θ e fissando y è possibile riscrivere la distribuzione a posteriori come:

$$\mathbb{P}(\theta|y) \propto \mathbb{P}(\theta)\mathbb{P}(y|\theta),$$

dove, avendo fissato y , $\mathbb{P}(y|\theta)$ è funzione di θ .

³Per il momento viene preso in considerazione il caso dei parametri, ma quanto detto si estende immediatamente al caso degli iperparametri, come visto in IV.4.3

IV.4.3 Evidenza bayesiana

Nel contesto dei modelli statistici, la reinterpretazione del teorema di Bayes (e di quanto detto nella sezione precedente) è:

$$\mathbb{P}(\theta|X, \alpha) = \frac{\mathbb{P}(X|\theta, \alpha)\mathbb{P}(\theta|\alpha)}{\mathbb{P}(X|\alpha)} \propto \mathbb{P}(\theta|\alpha)\mathbb{P}(X|\theta, \alpha),$$

dove: θ è il vettore di parametri, α il vettore degli iperparametri, X il campione di dati osservati. In questo caso la distribuzione a priori è composta da $\mathbb{P}(\theta|\alpha)$ ⁴, la distribuzione dei parametri prima di osservare i dati X ; $\mathbb{P}(X|\theta)$, la distribuzione campionaria che è la distribuzione dei dati osservati condizionata al valore dei parametri; e $\mathbb{P}(X|\alpha)$, l'**evidenza** (anche chiamata *verosimiglianza marginale*, in inglese *marginal likelihood*) che è la distribuzione dei dati condizionata rispetto gli iperparametri. Quest'ultima è possibile riscriverla come

$$\mathbb{P}(X|\alpha) = \int \mathbb{P}(X|\theta)\mathbb{P}(\theta|\alpha)d\theta,$$

cioè come la distribuzione dei dati marginalizzata sui parametri.

Oltre ad essere il termine di normalizzazione nel teorema di Bayes, un possibile altro modo di interpretare la marginal likelihood è quello di considerarla la probabilità di aver generato il dataset (osservato) X dati gli iperparametri α .

Utilizzando ancora il teorema di Bayes si ottiene:

$$\mathbb{P}(\alpha|X) = \frac{\mathbb{P}(X|\alpha)\mathbb{P}(\alpha)}{\mathbb{P}(X)} \propto \mathbb{P}(X|\alpha)\mathbb{P}(\alpha),$$

cioè la distribuzione a posteriori (che rappresenta la distribuzione degli iperparametri osservati i dati X) è proporzionale alla marginal likelihood. Poiché nel caso preso in considerazione dall'elaborato, cioè i processi gaussiani, uno degli obiettivi è trovare la distribuzione degli iperparametri (cioè la forma della covariance function, in altri termini $\mathbb{P}(\alpha|X)$), l'approccio adottato è quello di massimizzare la marginal likelihood, sfruttando la dipendenza evidenziata nella formula precedente. Questo approccio è chiamato **evidence approach**.

⁴Si noti che è condizionata rispetto al valore degli iperparametri. Questo è dovuto alla definizione degli iperparametri, che controllano la distribuzione dei parametri, come detto nella sezione IV.3.3.

IV.4.4 Alternative all'evidenza bayesiana

La scelta dell'approccio per stimare il valore di parametri o iperparametri dipende molto dalla situazione. Ad esempio:

- Massimizzazione della *likelihood*: si massimizza $L(\alpha, \theta|X)$ (la funzione di verosimiglianza), dunque si trovano valori di α e θ che la massimizzano;
- Massimizzazione parziale della *likelihood*: dato il caso in cui $L(\alpha, \theta|X) = L_1(\alpha|X)L_2(\theta|X)$, si massimizza solo il fattore di interesse, nel caso in esame si massimizza $L_1(\alpha|X)$;
- Massimizzazione della *marginal likelihood*: si marginalizza su θ , quindi si massimizza $\mathbb{P}(X|\alpha)$ che dipende solo da α .

L'elaborato applica l'ultimo caso poiché i calcoli necessari sono tutti analiticamente risolubili. In altre situazioni questo approccio richiede un'approssimazione numerica che può portare a problemi di stabilità, motivo per cui non è sempre un approccio usato.

IV.4.5 Nei processi gaussiani

L'approccio che i processi gaussiani seguono è quello di definire una *distribuzione sulle funzioni* (in inglese *distribution over functions*), aggiornarla a partire dai dati osservati e usarla per fare previsioni su nuovi input.

Si consideri quanto fatto nella sezione III.5: si è definita una distribuzione a priori sulle funzioni che poi è stata convertita in una distribuzione a posteriori aggiornando le informazioni grazie ai dati osservati. A differenza del caso *noise-free*, però, non si ottiene una funzione che interpoli perfettamente i dati. In questo caso è dunque necessario scegliere opportuni iperparametri, in modo da ottimizzare la forma della funzione: questa stima viene fatta tramite evidence approach sfruttando il fatto che le espressioni degli integrali sono tutte analiticamente risolubili.

In questo caso la marginal likelihood è:

$$\mathbb{P}(\mathbf{y}|\alpha) = \int \mathbb{P}(\mathbf{y}|\mathbf{f}, \alpha)\mathbb{P}(\mathbf{f}|\alpha)d\mathbf{f},$$

dove α sono gli iperparametri, \mathbf{f} sono dati osservati senza errore (sono i parametri), \mathbf{y} sono i dati osservati.

Si noti che $\mathbb{P}(\mathbf{f}|\alpha)$ è la distribuzione a priori della funzione priva di errori ed è una distribuzione gaussiana multivariata in quanto $\mathbf{f}|\alpha \sim \mathcal{N}(m, K)$ con K la matrice di Gram della covariance function e m la valutazione della mean function sul vettore di input. Inoltre, dalla definizione di \mathbf{y} si ha $\mathbf{y}|\mathbf{f} \sim \mathcal{N}(\mathbf{f}, \sigma_n^2 I)$, dove $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$.

Con i calcoli riportati su [RW06] si conclude:

$$L = \log(\mathbb{P}(\mathbf{y}|\alpha)) = -\frac{1}{2}\log(|K + \sigma_n^2 I|) - \frac{1}{2}(\mathbf{y} - \mathbf{m})^T(K + \sigma_n^2 I)^{-1}(\mathbf{y} - \mathbf{m}) - \frac{n}{2}\log(2\pi)$$

Lo stesso risultato si può ottenere evitando i conti analitici notando: $\mathbf{y} \sim \mathcal{N}(\mathbf{m}, K + \sigma_n^2 I)$. A questo punto è semplice calcolare le derivate parziali rispetto agli iperparametri della covariance function e della mean function. Questo è di fondamentale importanza per i metodi numerici di ottimizzazione.

Si noti che la massimizzazione della marginal likelihood è preferibile rispetto alla massimizzazione della likelihood (o altri metodi di likelihood) poiché con altre forme di likelihood è possibile ottenere un overfitting (si veda IV.2 per la definizione di overfitting). Al contrario, la marginal likelihood non esegue un *fitting* dei valori delle funzioni, ma li integra (li marginalizza), cioè tecnicamente non può fare overfitting perché non avviene alcun *fit*.

Per ulteriori considerazioni, dettagli sui conti e approfondimenti su come ottimizzare i calcoli numerici si rimanda a [RW06].

IV.5 Metodo di ottimizzazione

L'ottimizzazione è la branca della matematica che studia teoria e metodi per la ricerca dei punti di massimo e minimo di una funzione. Questa sezione si ripropone di ripercorrere i principali passi teorici che hanno portato alla costruzione del metodo di ottimizzazione usato successivamente nella massimizzazione della marginal likelihood: Adam.

IV.5.1 Forma della funzione da ottimizzare

Per comprendere la differenza tra i metodi di ottimizzazione che vengono riportati è importante comprendere la forma della funzione da ottimizzare. Nell'ambito dell'apprendimento supervisionato si ha un dataset di osservazioni da cui si vuole imparare. Per farlo, in breve, si definisce una funzione che rappresenta l'errore che il nostro modello compie sul dataset di valori osservati⁵: l'obiettivo è minimizzare questa funzione e per farlo è necessario, usando un opportuno algoritmo, cambiare i valori dei parametri (nel caso in esame, un modello non parametrico, vengono modificati gli iperparametri).

⁵Si veda la sezione IV.2, in cui viene spiegato come dividere un dataset in sottoinsiemi con diversi ruoli.

Siccome il dataset di dati osservati conterrà potenzialmente molti dati, è necessario tenere conto dell'errore commesso valutandolo in ogni dato osservato. Generalmente, infatti, la funzione di errore ha forma:

$$Q(w) = \sum_{i=1}^n Q_i(w),$$

dove tendenzialmente ogni addendo dipende da un singolo dato del dataset di input. Un esempio è:

$$f(m, b) = \sum_{i=1}^N (y_i - (mx_i + b))^2,$$

cioè il metodo dei minimi quadrati nella regressione lineare.

Si noti che nel caso in esame la funzione da ottimizzare, la marginal likelihood, è una somma in cui l'addendo i -esimo non dipende solo dall' i -esimo dato osservato.

IV.5.2 Learning rate

I metodi iterativi di ottimizzazione hanno una forma simile tra loro: dato θ_0 punto iniziale, ad ogni iterazione si aggiorna il punto come:

$$\theta_{t+1} = \theta_t + \eta_t d_t,$$

dove d_t rappresenta una direzione ed è definita in base al metodo, η_t è la lunghezza del passo (in inglese *step size*) o *learning rate* e indica di quanto ci si sposta nella direzione d_t . Il learning rate può essere costante oppure può essere aggiornato ogni iterazione.

IV.5.3 Metodo del gradiente [Rud16]

Il **metodo del gradiente** sfrutta il fatto che il gradiente è la direzione di massima crescita della funzione. Ha forma:

Dato un valore iniziale ω_0, η_0

Da ripetere fino a che si trova un'approssimazione del minimo:

$$\omega_{k+1} = \omega_k - \eta_k \cdot \nabla Q(\omega_k).$$

Con "Da ripetere fino a che si trova un'approssimazione del minimo" si intende che si ripete il metodo fino a che non viene soddisfatta una condizione di uscita che incorpora la precisione che si chiede all'approssimazione.

Poiché, come visto in IV.5.1, è necessario valutare il gradiente su ogni elemento del dataset, può essere un metodo oneroso soprattutto in termini di memoria. Il learning rate η_k può essere costante per ogni k oppure può essere modificato ad ogni iterazione. In entrambi i casi svolge un ruolo importante poiché può portare, se troppo piccolo, ad una convergenza troppo lenta oppure, se troppo grande, ad una divergenza. Nel caso di un learning rate aggiornato ad ogni iterazione, generalmente lo si diminuisce mano a mano che ci si avvicina al minimo; ad esempio si veda la figura IV.3.

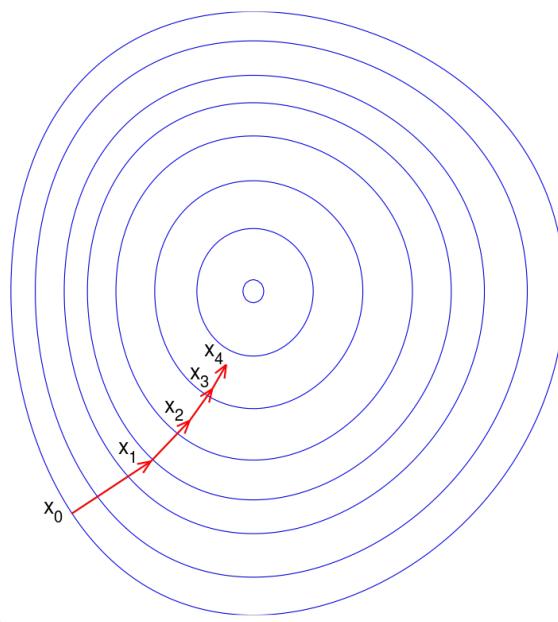


Figure IV.3: Illustrazione del metodo del gradiente su degli insiemi di livello, ad ogni iterazione viene aggiornato il learning rate.[Wik22c]

IV.5.4 Discesa stocastica del gradiente [Bot12]

Il metodo di **discesa stocastica del gradiente**, diversamente dal metodo del gradiente, esegue un aggiornamento per ogni Q_i ⁶. Ha forma:

Dato un valore iniziale ω_0, η_0

Da ripetere fino a che si trova un'approssimazione del minimo:

Mescola il dataset

Per $i = 1, \dots, n$:

$$\omega_{k+1} = \omega_k - \eta_k \cdot \nabla Q_i(\omega_k)$$

Dovendo calcolare il gradiente per un solo addendo di Q alla volta, è un metodo molto più veloce, seppur questi frequenti aggiornamenti causino un'alta fluttuazione della funzione obiettivo, come si può notare dalla figura IV.4.

Anche se la fluttuazione può sembrare uno svantaggio del metodo, è proprio questa che gli permette di evitare minimi locali e tendere a migliori approssimazioni. Il metodo del gradiente visto precedentemente invece converge al minimo del bacino in cui è stato scelto il dato iniziale.

D'altra parte, la fluttuazione complica la convergenza verso il minimo esatto a causa dei frequenti cambiamenti che la funzione subisce. Tuttavia è stato dimostrato che quando si diminuisce lentamente il learning rate il metodo mostra lo stesso comportamento di convergenza del metodo del gradiente.

⁶Nel caso in cui la Q si scriva come somma di termini che dipendono ognuno solo da un elemento del dataset, comporta che il metodo esegue un aggiornamento per ogni elemento del dataset.

IV.5.5 Metodo dei momenti [Rud16]

Il metodo stocastico del gradiente ha difficoltà a percorrere i *burroni*, cioè le aree in cui la superficie curva più ripidamente in una dimensione rispetto a un'altra. In questi scenari, il metodo oscilla lungo le pendici del burrone, mentre progredisce con esitazione lungo il fondo verso il minimo, come mostrato nella figura IV.4.

Il **metodo dei momenti** è un metodo che aiuta ad accelerare il metodo stocastico del gradiente nella direzione desiderata e a smorzare le oscillazioni, come si vede nella figura IV.4. Ha forma:

Dato un valore iniziale $\omega_0, \eta_0, v_0 = 0$

Da ripetere fino a che si trova un'approssimazione del minimo:

Mescola il dataset

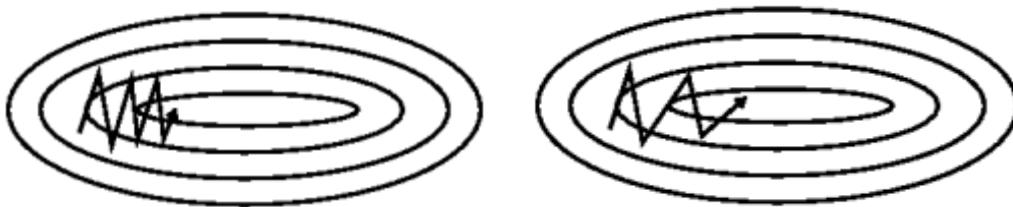
Per $i = 1, \dots, n$:

$$v_t = \gamma v_{t-1} + \eta_k \cdot \nabla Q_i(\omega_k)$$

$$\omega_{k+1} = \omega_k - v_t$$

Il termine di γ è chiamato *momento* (o *quantità di moto*) ed è solitamente impostato a 0.9.

In sostanza quello che avviene è analogo a quando si spinge una palla giù per una collina: la palla accumula quantità di moto mentre rotola in discesa diventando sempre più veloce (finché non raggiunge la sua velocità terminale, se c'è la resistenza dell'aria, cioè $\gamma < 1$). La stessa cosa accade agli aggiornamenti dei parametri: nel vettore il termine di "quantità di moto" favorisce le dimensioni in cui il gradiente punta nella stessa direzione e sfavorisce le altre. Di conseguenza, si ottiene una convergenza più rapida e una riduzione dell'oscillazione, riducendo i problemi del metodo stocastico del gradiente, come si mostra in figura IV.4.



(a) Senza momento.

(b) Con momento.

Figure IV.4: Metodo dei momenti applicato al metodo stocastico del gradiente. [Rud16]

IV.5.6 AdaGrad [DHS11]

Il metodo **AdaGrad** (abbreviazione di *Adaptive Gradient algorithm*) è un algoritmo di ottimizzazione basato sul metodo stocastico del gradiente che usa un learning rate indipendente per ogni ω_i . Per i termini ω_i che hanno gradiente più elevato o aggiornamenti frequenti, il metodo impone un learning rate più basso, in modo da non superare il minimo. Viceversa quelli con gradiente basso o aggiornamenti poco frequenti avranno un learning rate più alto, in modo da essere addestrati rapidamente.

Adagrad migliora la robustezza del metodo stocastico del gradiente e converge più rapidamente soprattutto quando la distribuzione dei dati è sparsa; ha dimostrato ottimi risultati nell’addestramento di reti neurali su larga scala (ad esempio in Google).

Dato un valore iniziale ω_0, η

Da ripetere fino a che si trova un’approssimazione del minimo:

Mescola il dataset

Per $i = 1, \dots, n$:

Per ogni componente j di ω :

$$g_{k,j} = (\nabla Q_i(\omega_k))_j$$

$$\omega_{k+1,j} = \omega_{k,j} - \frac{\eta}{\sqrt{G_{k,j}} + \epsilon} g_{k,j}$$

Dove $G_{k,j} = \sum_{t=1}^k g_{t,j}^2$ e ϵ un termine piccolo per evitare divisioni per zero (generalmente 10^{-8}).

Poiché adatta il learning rate, uno dei principali vantaggi del metodo è che elimina la necessità di regolare manualmente η_k : generalmente si imposta $\eta = 0,01$. Il punto debole di Adagrad è l’accumulo dei gradienti al quadrato nel denominatore: poiché ogni termine aggiunto è positivo, la somma accumulata continua a crescere durante l’addestramento. Questo fa sì che il learning rate si riduca e tenda a diventare infinitesimalmente piccolo, a quel punto l’algoritmo non è più in grado di acquisire ulteriori conoscenze.

IV.5.7 Adam [KB17]

Il metodo **Adam** (abbreviazione di *Adaptive Moment Estimation*) è, come il precedente, un metodo a learning rate adattivi per ogni parametro. Deriva da altri metodi (AdaDelta, RMSprop) che cercano di ridurre il tasso di apprendimento monotonicamente decrescente del metodo AdaGrad. Infatti: invece di accumulare tutti i gradienti passati al quadrato, la somma dei gradienti è ricorsivamente definita come una media decrescente di tutti i gradienti quadratici passati, che dipende quindi dalla media precedente e dal gradiente corrente, ed è chiamata v_k . Inoltre, il metodo di Adam prevede la stessa media decrescente dei gradienti passati, chiamata m_k .

Dato un valore iniziale $\omega_0, \eta, m_0 = 0, v_0 = 0$

Da ripetere fino a che si trova un'approssimazione del minimo:

Mescola randomicamente il dataset

Per $i = 1, \dots, n$:

Per ogni componente j di ω :

$$\begin{aligned} g_{k,j} &= (\nabla Q_i(\omega_k))_j \\ m_k &= \beta_1 m_{k-1} + (1 - \beta_1) g_{k,j} \\ v_k &= \beta_2 v_{k-1} + (1 - \beta_2) g_{k,j}^2 \\ \hat{m}_k &= \frac{m_k}{1 - \beta_1^k} \\ \hat{v}_k &= \frac{v_k}{1 - \beta_2^k} \\ \omega_{k+1,j} &= \omega_{k,j} - \frac{\eta}{\sqrt{\hat{v}_k} + \epsilon} \hat{m}_k \end{aligned}$$

I termini m_k e v_k , le medie decrescenti dei gradienti e dei gradienti quadrati, sono stime del primo momento e del secondo momento dei gradienti, come definito nel metodo dei momenti. Poiché m_k e v_k sono distorti verso valori prossimi allo zero, si usano estimatori corretti \hat{m}_k e \hat{v}_k . Generalmente si impongono valori $\beta_1 = 0.9$ e $\beta_2 = 0.999$.

In figura IV.5 vengono comparati alcuni metodi nella minimizzazione della cost function in una neural network. Si nota che il metodo di Adam è molto più efficiente degli altri metodi introdotti nell'elaborato.

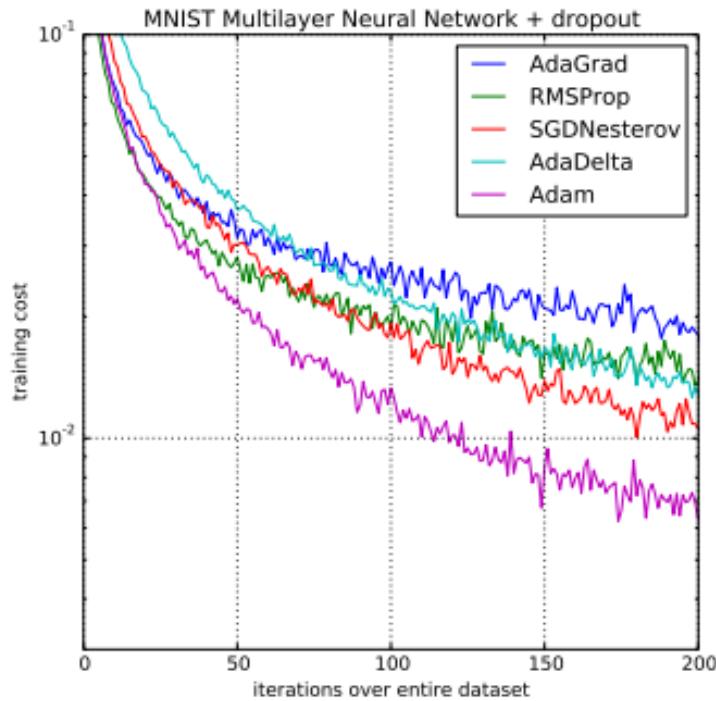


Figure IV.5: Comparazione di metodi di minimizzazione di una cost function in una rete neurale. [KB17]

V

Modello Windkessel

In questo capitolo vengono introdotti alcuni concetti di fisiologia necessari alla comprensione del **modello Windkessel**, successivamente introdotto. Il capitolo non vuole essere completo della teoria alla base del modello, ma vuole introdurne l'idea su cui si fonda: l'**effetto Windkessel**.

Nel capitolo viene fatto uso di una jupyter notebook usata nel corso *Introduzione a Python per il calcolo scientifico* tenuto dal professore Lucas Omar Müller nell'anno accademico 2021/2022.

Le fonti usate per la stesura del capitolo sono: [AW20], [Wikb], [Wika], [Wik21b], [Wik22a], [Wik21a], [WLW08], [GTM22].

Medicine is a science of uncertainty and an art of probability.

William Osler

V.1 Introduzione

Da [WLW08] si prendono in considerazione dati di un paziente nel quale è stato misurato il flusso aortico Q_{in} e la pressione sistematica P .

Visualizzando il grafico dei due valori si ottiene quanto in figura V.1.

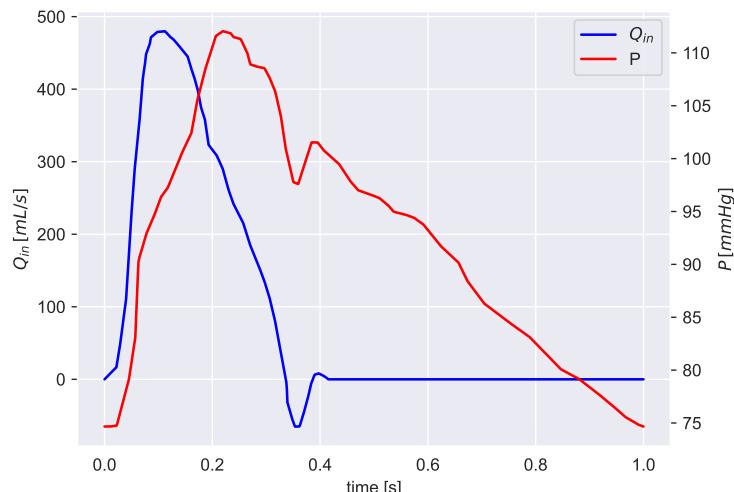


Figure V.1: Pressione sistematica e flusso aortico misurati in un paziente. Codice VII.31.

Viene considerato ora un semplice modello del sistema cardiovascolare:

$$P = Q_{\text{in}} R,$$

dove R è la resistenza periferica totale. Questo modello permette di calcolare la pressione arteriosa a partire dal flusso di sangue in entrata nel sistema e dalla resistenza periferica totale.

A partire dai dati reali del paziente, è possibile ricavare il valore della resistenza periferica totale R , o resistenza del sistema cardiovascolare. Data T la durata del ciclo cardiaco, si ha:

$$R = \frac{\frac{1}{T} \int_T P(t) dt}{\frac{1}{T} \int_T Q_{\text{in}}(t) dt}.$$

Ricavata $R = 1.037 \text{ mmHg/mL} \cdot \text{s}$ (come da codice VII.32) conoscendo Q_{in} è ora possibile usare il modello per ricavare l'andamento di P , come mostrato in figura V.2.

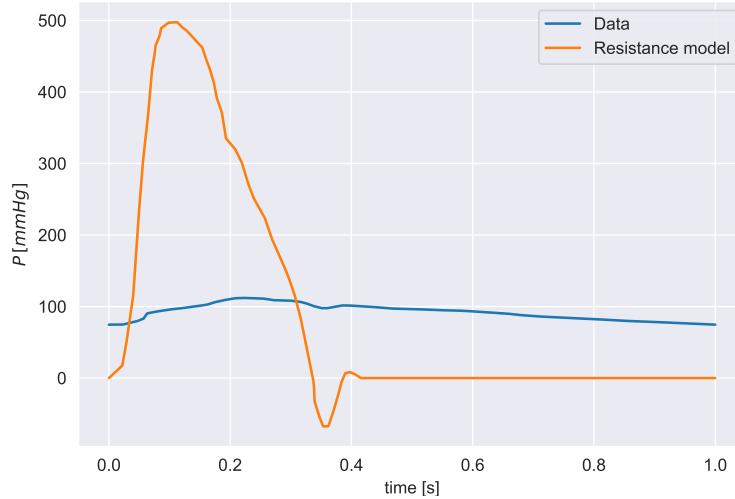


Figure V.2: Grafico della pressione reale e output del modello semplice. Codice VII.36.

Come è chiaro dalla figura, nella fase di sistole il modello prevede un picco molto alto di pressione (sull'ordine di cinque volte più alto del dato reale). Come verrà mostrato nel resto del capitolo, questo dipende dal fatto che il modello trascura un fondamentale aspetto della circolazione arteriosa: l'effetto Windkessel, cioè l'immagazzinamento di energia nelle arterie in fase di sistole. A fine capitolo verrà proposto un modello che tiene conto di questo effetto e fornisce quindi un'approssimazione molto più precisa.

V.2 Concetti di fisiologia

V.2.1 Ciclo cardiaco

Non verrà analizzato in dettaglio il ciclo cardiaco, del quale viene riportato un riassunto in tabella V.1, ma vengono brevemente descritti il periodo di sistole e diastole, utili alla comprensione dei prossimi argomenti. Potrà tornare utile l'immagine V.3, in modo da avere presente l'anatomia del cuore umano necessaria alla comprensione dei successivi concetti.

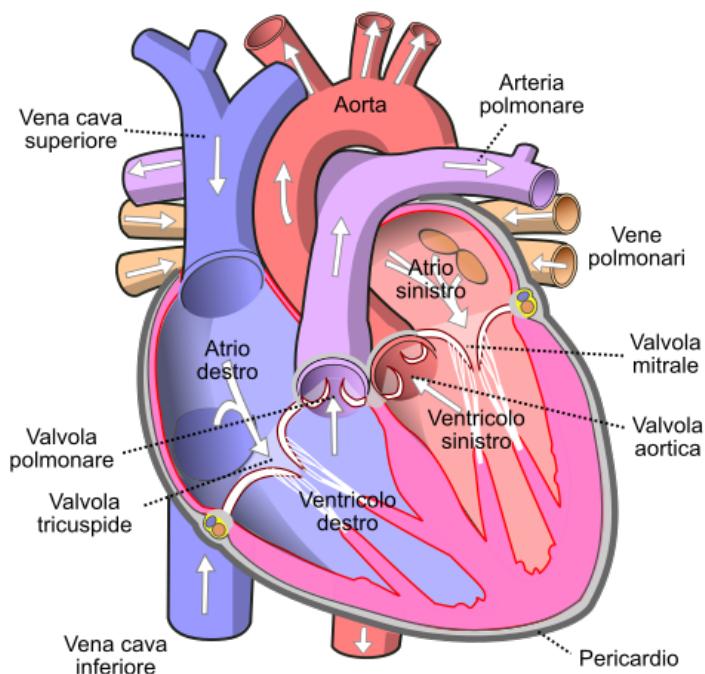


Figure V.3: Anatomia del cuore umano [Wik22a].

La diastole cardiaca è il periodo del ciclo cardiaco in cui, dopo la contrazione, il cuore si rilassa e si espande mentre si riempie di sangue di ritorno dal sistema circolatorio. Entrambe le valvole atrioventricolari (tricuspidale e mitrale) si aprono per facilitare il flusso "non pressurizzato" del sangue direttamente attraverso gli atrii in entrambi i ventricoli, dove viene raccolto per la successiva contrazione.

La sistole atriale è la contrazione delle cellule muscolari cardiache di entrambi gli atrii in seguito alla stimolazione elettrica e alla conduzione di correnti elettriche attraverso le camere atriali.

Definibile come parte della sequenza di contrazione ed espulsione, la sistole atriale in realtà svolge il ruolo di completare la diastole, finalizza cioè il riempimento di entrambi i ventricoli con il sangue mentre sono rilassati ed espansi per tale scopo.

La sistole atriale si sovrappone alla fine della diastole e applica una pressione di contrazione per spostare i volumi di sangue a entrambi i ventricoli; questo "calcio" (in inglese *kick*) atriale conclude la diastole immediatamente prima che il cuore ricominci a contrarsi e ad espellere il sangue dai ventricoli (sistole ventricolare) verso l'aorta e le arterie.

La sistole ventricolare è la contrazione, in seguito a stimolazioni elettriche, del sincizio ventricolare di cellule muscolari cardiache nei ventricoli destro e sinistro.

Le contrazioni del ventricolo destro garantiscono la circolazione polmonare facendo pulsare il sangue impoverito di ossigeno attraverso la valvola polmonare e poi attraverso le arterie polmonari fino ai polmoni. Contemporaneamente, le contrazioni della sistole ventricolare sinistra forniscono la circolazione sistemica di sangue ossigenato a tutti i sistemi del corpo pompando il sangue attraverso la valvola aortica, l'aorta e tutte le arterie.

In figura V.4 vengono riportati due semplici diagrammi che rappresentano lo spostamento del sangue ossigenato (rosso) e non ossigenato (blu) nelle fasi di sistole e diastole.

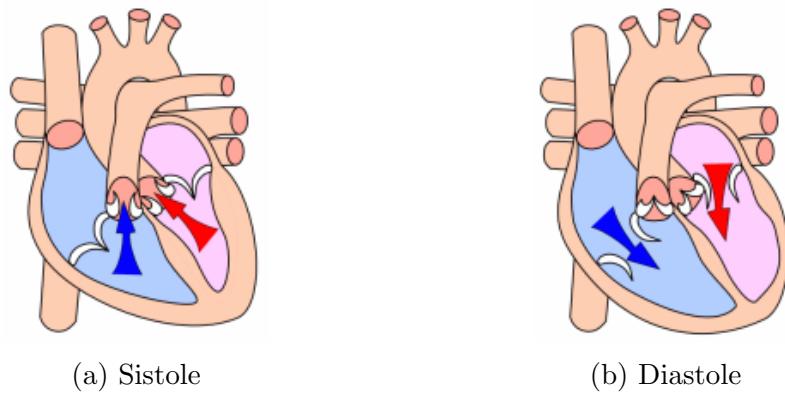


Figure V.4: Diagrammi che riassumono la sistole e la diastole di un cuore umano [Wik22a].

Fase	Valvole atrioventricolari (tricuspide e mitrale)	Valvole semilunari (polmonare e aortica)	Stato dei ventricoli e degli atri; flusso sanguigno
1 - Rilassamento isovolumetrico	chiuse	chiuse	Le valvole semilunari si chiudono alla fine della fase di eiezione; il flusso di sangue si ferma.
2a - Afflusso (riempimento ventricolare)	aperte	chiuse	I ventricoli e gli atri si rilassano e si espandono insieme; il sangue scorre nel cuore durante la diastole ventricolare e atriale.
2b - Infusso (Riempimento ventricolare con sistole atriale)	aperte	chiuse	Ventricoli rilassati ed espansi; la contrazione atriale (sistole) spinge il sangue sotto pressione nei ventricoli durante la diastole ventricolare.
3 - Contrazione isovolumetrica	chiuse	chiuse	Le valvole AV si chiudono alla fine della diastole ventricolare; il flusso di sangue si ferma; i ventricoli cominciano a contrarsi.
4 - Espulsione ventricolare	chiuse	aperte	I ventricoli si contraggono (sistole ventricolare); il sangue scorre dal cuore ai polmoni e al resto del corpo durante l'espulsione ventricolare.

Table V.1: Tabella riassuntiva ciclo cardiaco [Wik22a].

Utile alla comprensione del ciclo cardiaco è il diagramma di Wiggers: è un diagramma standard che viene utilizzato nell'insegnamento della fisiologia cardiaca nel quale l'asse orizzontale viene utilizzato per tracciare il tempo mentre l'asse verticale contiene tutti i seguenti elementi su una singola griglia: pressione sanguigna, pressione aortica, pressione ventricolare, pressione atriale, volume ventricolare, elettrocardiogramma, flusso arterioso.

Il diagramma di Wiggers illustra chiaramente la variazione coordinata di questi valori mentre il cuore batte, aiutando a comprendere l'intero ciclo cardiaco. Aiuta anche a familiarizzare con la forma delle curve di parametri che verranno utilizzati nell'elaborato. Un esempio di diagramma viene mostrato in figura V.5.

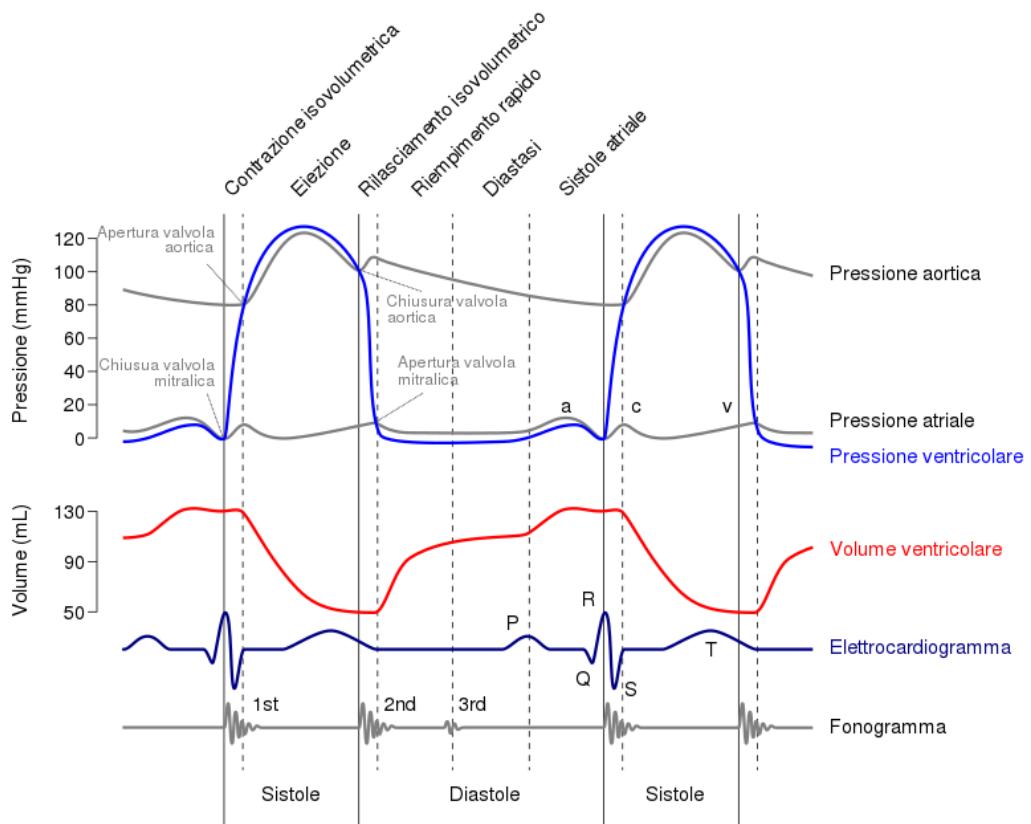


Figure V.5: Esempio di diagramma di Wiggers [Wik21a].

V.2.2 Terminologia utile

Volume sistolico

Il **volumen sistolico** (in inglese *stroke volume*, o SV) è la quantità di sangue pompato da un ventricolo ad ogni sistole ventricolare. Il volume sistolico è solitamente uguale nei due ventricoli, circa 70ml in un uomo sano di 70kg.

Arteria elastica

Un'**arteria elastica** è un'arteria formata da molti filamenti di collagene ed elastina, che le conferiscono la capacità di allungarsi in risposta ad ogni pulsazione.

È proprio in virtù di questa elasticità che si ha l'effetto Windkessel, il quale aiuta a mantenere una pressione costante nelle arterie nonostante la natura pulsante del flusso sanguigno.

Le arterie elastiche comprendono le arterie più grandi del corpo, quelle più vicine al cuore.

Le arterie polmonari, l'aorta e i suoi rami costituiscono insieme il sistema di arterie elastiche del corpo. Esempi sono: aorta, brachiocefalica, carotide comune, succlavia, iliaca comune.

Resistenza vascolare

La **resistenza vascolare** è la resistenza che deve essere superata per spingere il sangue attraverso il sistema circolatorio e creare il flusso.

La resistenza offerta dalla circolazione sistemica è nota come **resistenza vascolare sistemica** (in inglese *systemic vascular resistance* o SVR) o **resistenza periferica totale** (in inglese *total peripheral resistance* o TPR)¹.

La vasocostrizione (cioè la diminuzione del diametro dei vasi sanguigni) aumenta la SVR, mentre la vasodilatazione (aumento del diametro) diminuisce la SVR.

Vale la formula: $R = \frac{\Delta P}{Q}$, dove R è la resistenza, ΔP la variazione di pressione durante il loop circolatorio², Q è il flusso.

Si noti che questa è la versione idraulica della legge di Ohm, $V=IR$, in cui il differenziale di pressione è analogo alla caduta di tensione elettrica, il flusso è analogo alla corrente elettrica e la resistenza vascolare è analoga alla resistenza elettrica.

¹Vi è la resistenza offerta dalla circolazione polmonare nota come **resistenza vascolare polmonare**, in inglese *pulmonary vascular resistance* o PVR. Non sarà però oggetto di studio in questo elaborato.

²Cioè da subito dopo l'uscita dal ventricolo sinistro / ventricolo destro all'entrata nell'atrio destro / atrio sinistro

Capacitanza o compliance

La **capacitanza** (o compliance, in inglese) è la capacità di un organo cavo di distendersi e aumentare di volume con l'aumento della pressione. Essa è l'equivalente fluidodinamico della capacità elettrica.

Nel caso dei vasi sanguigni, questo significa fisicamente che i vasi con una maggiore compliance si deformano più facilmente dei vasi con minore compliance nelle stesse condizioni di pressione e volume.

La compliance venosa è circa 30 volte maggiore di quella arteriosa, in gran parte a causa delle loro pareti più sottili.

La compliance C di un vaso sanguigno è direttamente proporzionale all'elasticità delle sue pareti e costituisce una misura dei rapporti tra le variazioni di pressione e le variazioni di volume. Viene definita come: $C = \frac{\Delta V}{\Delta P}$, dove ΔV è la variazione di volume, ΔP è la variazione di pressione, cioè la differenza tra pressione intravasale e esterna.

Una caratteristica che rende la sua stima oggetto di studio è la difficoltà di misurazione: la compliance arteriosa può essere misurata con diverse tecniche ma la maggior parte di esse sono invasive e non sono clinicamente appropriate.

V.2.3 Effetto Windkessel

L'**effetto Windkessel** è un termine usato in medicina per spiegare la forma d'onda della pressione arteriosa in termini di interazione tra il volume sistolico e la compliance dell'aorta e delle grandi arterie elastiche e la resistenza delle arterie e arteriole più piccole.

Windkessel, dal tedesco, significa *camera d'aria* e veniva usato nel diciottesimo secolo dai pompieri per garantire una continua erogazione di acqua nel combattere gli incendi; nel caso cardiovascolare si tratta di un serbatoio elastico, ma il funzionamento è lo stesso. In figura V.6 viene illustrata questa analogia.

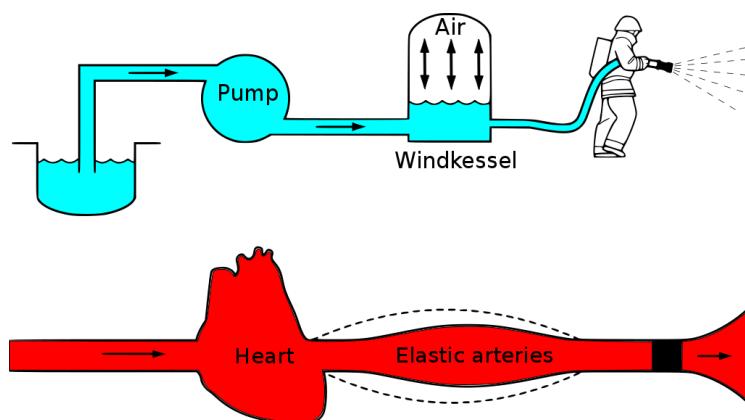


Figure V.6: Illustrazione dell'analogia sull'effetto windkessel [Wik21b].

Come introdotto precedentemente, le arterie elastiche si distendono quando la pressione sanguigna sale durante la sistole e si ritraggono quando la pressione sanguigna scende durante la diastole, come mostrato in figura V.7.

Poiché il tasso di sangue che entra in questo tipo di arterie supera il tasso di sangue che esce, c'è un deposito netto di sangue durante la sistole, che si scarica durante la diastole. La compliance dell'aorta e delle grandi arterie elastiche è quindi analoga a quella di un condensatore; in altre parole, queste arterie agiscono collettivamente come un accumulatore idraulico.

L'effetto Windkessel aiuta a smorzare la fluttuazione della pressione sanguigna durante il ciclo cardiaco e aiuta a mantenere la perfusione degli organi durante la diastole, quando l'espulsione cardiaca cessa.

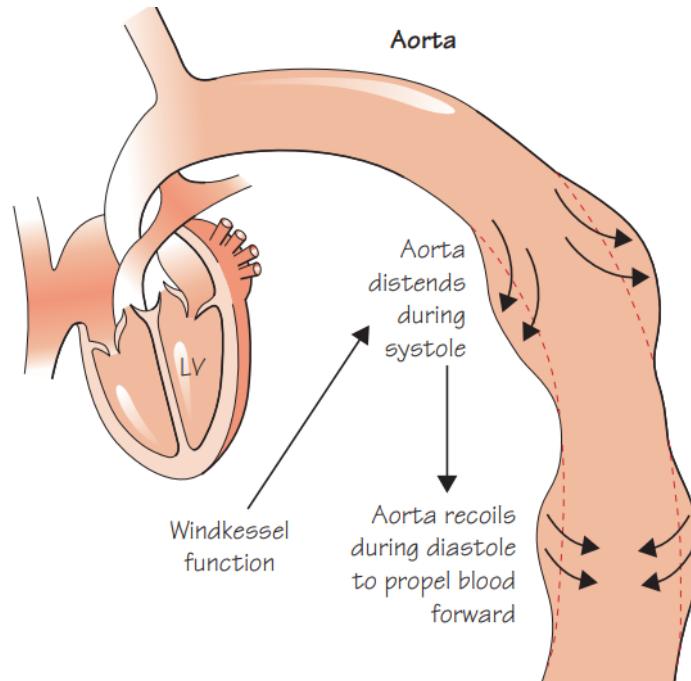


Figure V.7: Illustrazione dell'effetto windkessel [AW20].

V.3 Modello Windkessel a due elementi

Nel corso dei secoli il sistema arterioso è stato modellato in molti modi e con diverse strategie. Un esempio è stato mostrato nell'introduzione a questo capitolo. Stephen Hales³ fu il primo a misurare la pressione sanguigna e notò che la pressione nel sistema arterioso non è costante, ma varia nel corso del battito cardiaco, suggerendo che le variazioni di pressione potessero essere legate all'elasticità delle grandi arterie.

Fu Otto Frank⁴ che formulò quantitativamente e rese popolare il cosiddetto **modello Windkessel a due elementi** composto da un elemento di resistenza e uno di compliance.

La legge di Poiseuille afferma che la resistenza è inversamente proporzionale al raggio del vaso sanguigno alla quarta potenza. La resistenza al flusso nel sistema arterioso è quindi principalmente nei vasi di resistenza: le arterie più piccole e le arteriole. Quando tutte le resistenze individuali nella microcircolazione vengono sommate correttamente, si ottiene la resistenza dell'intero letto vascolare sistemico chiamata resistenza periferica (totale). La resistenza periferica, R , può essere calcolata come:

$$R = \frac{P_{\text{ao; mean}} - P_{\text{ven; mean}}}{CO} \approx \frac{P_{\text{ao; mean}}}{CO}, \quad (1)$$

dove $P_{\text{ao; mean}}$ è la pressione aortica media, mentre $P_{\text{ven; mean}}$ è la pressione venosa media, CO è l'output cardiaco.

La componente di compliance è principalmente determinata dall'elasticità delle grandi arterie. Può essere ottenuta sommando le compliance di tutti i vasi e viene quindi chiamata compliance arteriosa totale.

Per il calcolo di C (come mostrato in V.2.2) è molto difficile eseguire un esperimento in cui un volume viene iniettato nel sistema arterioso senza alcuna perdita nella periferia. Per questo motivo sono stati sviluppati diversi metodi per ricavare la compliance arteriosa totale senza ricorrere a esperimenti.

Il Windkessel a due elementi prevede che in diastole, quando la valvola aortica è chiusa, la pressione decada esponenzialmente con un tempo di decadimento caratteristico, con il quale è possibile calcolare la resistenza periferica con la pressione aortica in diastole e una stima della compliance arteriosa totale.

Il flusso medio, cioè l'output cardiaco CO , si ricava quindi da (1).

Il Windkessel è un cosiddetto *lumped* model. In altre parole, questo modello descrive l'intero sistema arterioso in termini di una relazione pressione-flusso al suo ingresso, sfruttando due parametri che hanno un significato fisiologico. È interessante notare che in passato la ricerca si è concentrata principalmente sulla resistenza periferica, mentre il contributo della compliance arteriosa totale è stato spesso trascurato.

³Stephen Hales (1677-1761) è stato un ecclesiastico inglese che ha dato importanti contributi ad una serie di campi scientifici tra cui la botanica, la chimica pneumatica e la fisiologia.

⁴Otto Frank (1865 - 1944) è stato un medico e fisiologo tedesco che ha contribuito alla fisiologia cardiaca e alla cardiologia.

Il modello Windkessel a due elementi mostra come il carico sul cuore consiste di resistenza periferica e totale arteriosa e che entrambi svolgono un ruolo importante.

Con lo sviluppo del flussometro elettromagnetico e quindi della misurazione del flusso aortico, è diventato chiaro che in sistole la relazione tra pressione e flusso era mal predetta dal modello Windkessel a due elementi. Le misurazioni del flusso aortico e gli sviluppi delle possibilità tecnologiche hanno portato a migliorie del modello a due elementi: il modello a tre elementi tiene conto anche della resistenza della valvola aortica, mentre il modello a quattro elementi tiene conto anche dell'inerzia del flusso sanguigno.

Per mostrare la precisione che può garantire il modello Windkessel a due elementi, viene mostrata a seguire la procedura in Python per ottenere la stima della pressione confrontandola con gli stessi dati reali usati nella sezione introduttiva.

Per il codice Python mostrato è necessario importare le giuste librerie mostrate nel codice VII.30; è necessario anche il codice VII.34 che definisce la funzione di flusso⁵.

⁵Il codice usa i dati reali ripresi da [WLW08], pertanto è necessario il codice VII.31.

V.3.1 Definizione del modello

Come chiarito nelle definizioni in V.2.2, il modello Windkessel può essere espresso in forma di circuito come in figura V.8.

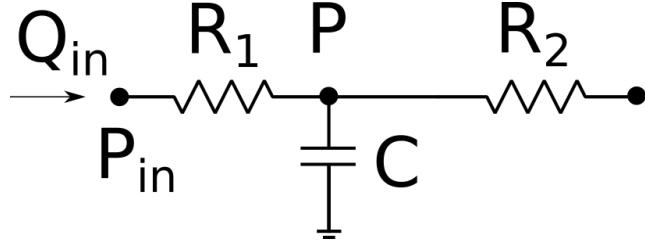


Figure V.8: Forma circuitale del modello Windkessel a due elementi.

Il modello Windkessel a due elementi si caratterizza dall'equazione:

$$\frac{dP}{dt} = \frac{1}{C}(Q_{in} - Q_{out}),$$

dove C è la compliance sistematica, R_1 e R_2 sono la resistenza prossimale e periferica (di tutte le arterie e arteriole), Q_{out} il flusso in uscita dal capacitatore. Consideriamo anche $Q_{out} = \frac{P - P_d}{R_2}$, dove P_d è la pressione distale (successiva cioè alla resistenza periferica, nel corso dell'elaborato, quando non specificato altrimenti, viene imposta a 5mmHg), e

$$P_{in} = P + R_1 Q_{in}, \quad (2)$$

da cui possiamo quindi scrivere:

$$\frac{dP}{dt} = \frac{1}{C} \left(Q_{in} - \frac{P - P_d}{R_2} \right). \quad (3)$$

V.3.2 Stima della compliance

Per lavorare con l'equazione del modello è necessario conoscere la compliance. Per farlo, si minimizza la funzione

$$f_C(C) = \sum_{n=1}^M \left(P_{in}(t_n) - \hat{P}_{in}(t_n) \right)^2,$$

dove \hat{P}_{in} è il valore misurato di pressione in M punti t_n e P_{in} è definita come in (2) e richiede di risolvere (3) con condizione iniziale $P(0) = \hat{P}_{in}(t_1)$.

Per questo primo approccio viene assunto $R_1 = 0$ e $R_2 = R$ dove R è la resistenza periferica totale. Da (2) si ricava che $P_{in} = P$.

Poiché è necessario risolvere l'equazione (3), è necessario definirla in Python come funzione, come nel codice VII.33. Si definisce poi la funzione f_C nel codice VII.35. Il grafico della funzione f_C è mostrato in figura V.9.

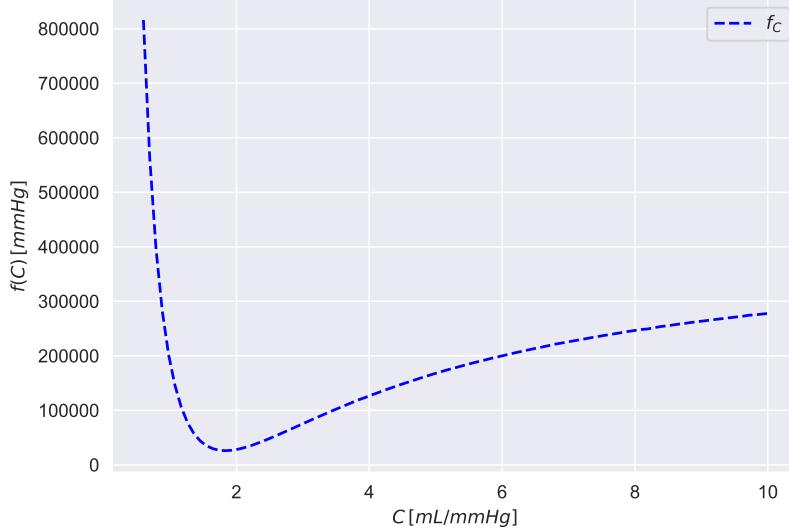


Figure V.9: Grafico di f_C . Codice VII.37.

Dunque è ora possibile stimare C che minimizza f_C appoggiandosi alla libreria di Python `scipy.optimize`, come fatto nel codice VII.38, in questo modo si trova $C = 1,83288mL/mmHg$.

Risolvendo ora l'equazione (3) con la C stimata si ottiene quanto in figura V.10.

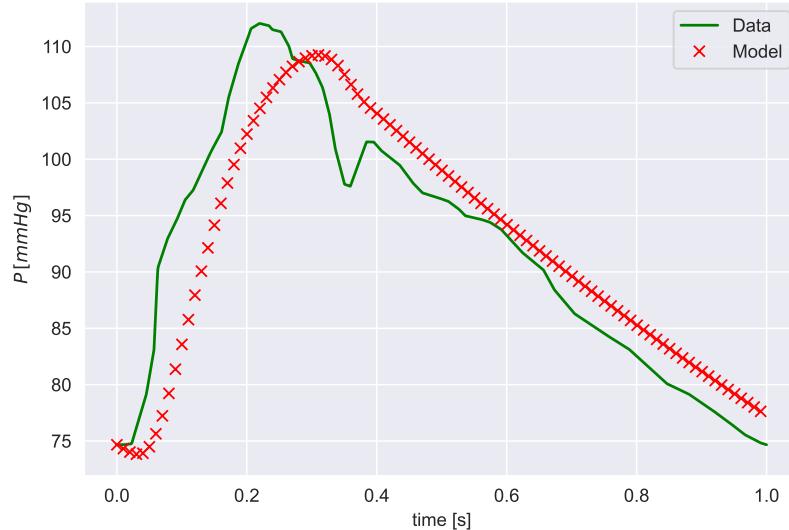


Figure V.10: Grafico della soluzione approssimata dell'equazione (3) con C stimata. Codice VII.39.

V.3.3 Stima della compliance e delle resistenze

Cambiando la definizione delle resistenze è possibile migliorare l'approssimazione. Si definiscono ora: $R_1 = (1 - \alpha)R$ e $R_2 = \alpha R$ dove $\alpha \in [0, 1]$. Dunque ora la funzione f_C dipende anche da α , ed è dunque necessario aggiornarla come mostrato nel codice VII.40. Nel codice VII.41 si stimano i parametri C e α . Si ottiene: $C = 2,11579mL/mmHg$ e $\alpha = 0,97134$. Risolvendo ora l'equazione (3) con i parametri stimati di C e α si ottiene un grafico che approssima con una notevole precisione i dati reali, come mostrato in figura V.11.

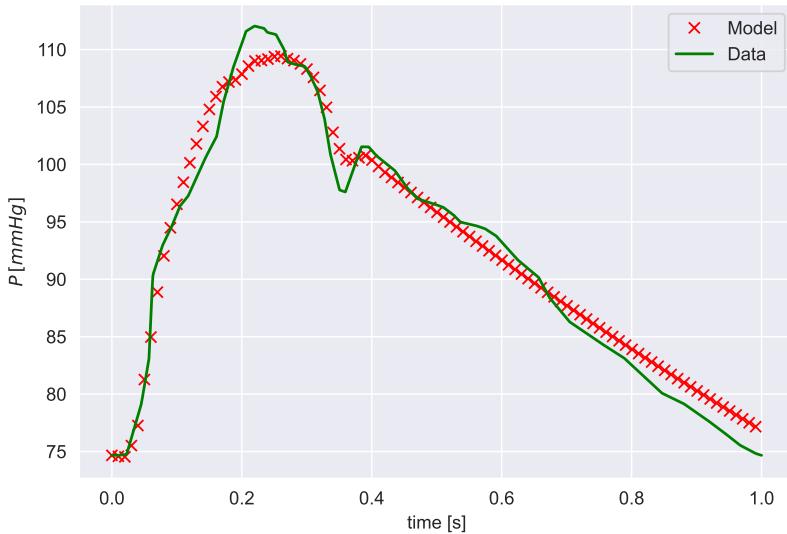


Figure V.11: Grafico della soluzione approssimata dell'equazione (3) con $C = 2,11579mL/mmHg$ e $\alpha = 0,97134$. Codice VII.42.

V.4 Forzante periodica

È un risultato noto che, dato un sistema lineare omogeneo a coefficienti costanti, cioè

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t), \quad (4)$$

la stabilità della soluzione esatta di tale sistema è determinata dalla parte reale degli autovalori della matrice di coefficienti A . In particolare, una condizione necessaria e sufficiente affinché il sistema sia asintoticamente stabile è che tutti gli autovalori di A abbiano parte reale strettamente negativa. In tal caso esistono costanti positive α, β tali che

$$\|e^{At}\| \leq \beta e^{-\alpha t} \quad t \geq 0. \quad (5)$$

Si dimostra ora che, quando una funzione forzante periodica $\mathbf{b}(t)$ viene aggiunta al sistema (4) per ottenere il sistema disomogeneo

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + \mathbf{b}(t), \quad (6)$$

se la parte omogenea è asintoticamente stabile, allora qualsiasi soluzione del sistema non omogeneo (6) converge a una soluzione periodica all'aumentare di t . La soluzione esatta del sistema disomogeneo, ottenuta con il metodo della variazione delle costanti (o metodo di Lagrange), è:

$$\mathbf{x}(t) = e^{At} \mathbf{x}_0 + e^{At} \int_0^t e^{A(t-s)} \mathbf{b}(s) ds$$

dove $\mathbf{x}(0) = \mathbf{x}_0$. Data $\mathbf{b}(t)$ funzione forzante periodica, cioè $\mathbf{b}(T_0) = \mathbf{b}(0)$ per un certo $T_0 > 0$, è possibile ottenere una soluzione periodica del sistema disomogeneo (6), dunque che soddisfa $\mathbf{x}(T_0) = \mathbf{x}(0)$, scegliendo opportunamente la condizione iniziale \mathbf{x}_0 . Con dei semplici conti si ottiene che la suddetta condizione iniziale \mathbf{x}_0^P per ottenere una soluzione periodica è data da

$$\mathbf{x}_0^P = (I - e^{AT_0})^{-1} e^{AT_0} \int_0^{T_0} e^{-As} \mathbf{b}(s) ds.$$

Si mostra ora che, a partire da una condizione iniziale diversa da \mathbf{x}_0^P , i valori della soluzione convergono ai valori della soluzione periodica all'aumentare di t . Sia \mathbf{x}^P la soluzione periodica corrispondente alla condizione iniziale \mathbf{x}_0^P , sia \mathbf{x}^{NP} qualsiasi soluzione del sistema disomogeneo associata a una qualche condizione iniziale \mathbf{x}_0^{NP} . Allora, calcolando la norma della differenza tra le due soluzioni e applicando la (5), si ottiene

$$\|\mathbf{x}^P(t) - \mathbf{x}^{NP}(t)\| = \|e^{At}(\mathbf{x}_0^P - \mathbf{x}_0^{NP})\| \leq \beta e^{-\alpha t} \|\mathbf{x}_0^P - \mathbf{x}_0^{NP}\|,$$

nella quale $\beta e^{-\alpha t} \rightarrow 0$ per $t \rightarrow +\infty$, cioè quanto si voleva dimostrare.

Quindi se la funzione forzante $\mathbf{b}(t)$ è periodica e se la parte omogenea del sistema è asintoticamente stabile, allora la soluzione esatta del problema disomogeneo convergerà alla soluzione periodica del sistema disomogeneo stesso al crescere di t per qualsiasi scelta ammissibile della condizione iniziale.

V.4.1 Applicazione al modello Windkessel

Nel modello Windkessel la forzante periodica è il flusso cardiaco, come si osserva nell'equazione (3). Dunque, a seguito di quanto detto, invece di trovare un'approssimazione della soluzione usando il flusso di un singolo ciclo cardiaco, si trova un'approssimazione su più cicli fino a che la soluzione è, a meno di una tolleranza, uguale a quella precedente.

Per farlo è sufficiente sostituire il codice della funzione flusso VII.34 con VII.48 e quando si risolve l'equazione differenziale sostituire `t_span` e di `t_eval` con `t_span = [time[0], numCicli+time[-1]]` e `t_eval = numCicli+time` dove `numCicli` è il numero di cicli cardiaci rispetto ai quali risolvere l'equazione differenziale.

Con queste modifiche al codice si ripete quanto visto precedentemente per l'approssimazione di α e C e si trova: $C = 2.03424$ e $\alpha = 0.97354$. L'approssimazione della soluzione dopo venti cicli cardiaci con questi parametri viene mostrata in figura V.12.

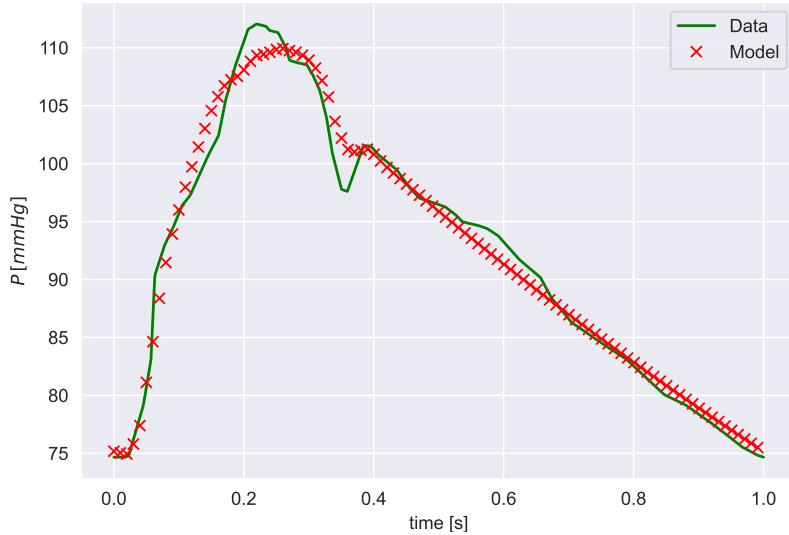


Figure V.12: Grafico della soluzione dell'equazione (3) approssimata dopo venti cicli cardiaci con $C = 2,03424mL/mmHg$ e $\alpha = 0,97354$.

Si noti che nella figura V.11 l'errore della soluzione approssimata dalla pressione reale è in norma infinito 5,005246 e in norma due 55,584739, mentre nella figura V.12 in norma infinito è 4,654959 e in norma due è 46,429220. Dunque il secondo approccio, quello che tiene conto della convergenza di ogni soluzione alla soluzione periodica, genera risultati più precisi a discapito di più cicli cardiaci da considerare.

V.4.2 Tempo di esecuzione

Con il comando `python %%timeit -r 20` (definito un *built-in magic command* delle jupyter notebook) è possibile calcolare il tempo medio e la deviazione standard di 20 esecuzioni del codice python contenuto in una cella di una jupyter notebook. Quando si cerca l'approssimazione della soluzione a convergenza si verifica che sia abbastanza simile a quella precedente, a meno di 5×10^{-3} . Come spiegato in VI.2.4, il dataset di input è costituito da 3375 combinazioni di parametri C, R_1, R_2 , per ognuna delle quali si cerca la soluzione fino a convergenza. Su tutte le combinazioni si ottengono diversi numeri di cicli necessari per arrivare a convergenza, tutti strettamente minori di cento.

Per dare un'idea di quanto tempo ci impieghi questo approccio, vengono mostrate in tabella V.2 la quantità di tempo necessaria a trovare la soluzione dopo diversi cicli cardiaci.

Cicli cardiaci	Tempo esecuzione (media + dev. std.)
1	17.1 ms + 1.23 ms
10	90.4 ms ± 3.75 ms
20	189 ms ± 10.3 ms
30	270 ms ± 12.9 ms
40	362 ms ± 13.3 ms
50	458 ms ± 15.1 ms
60	526 ms ± 21.7 ms
70	602 ms ± 15.9 ms
80	711 ms ± 17.3 ms
90	802 ms ± 32.3 ms
100	883 ms ± 17.6 ms

Table V.2: Tempo di esecuzione per trovare l'approssimazione della soluzione del modello Windkessel su diversi cicli cardiaci.

V.5 Analisi di sensitività locale

In questa sezione viene indagata la sensitività dell'output del modello ai cambiamenti dei parametri eseguendo un'*analisi di sensitività locale*.

L'output del modello (la pressione P) viene utilizzato per il calcolo di quattro variabili:

- MAP: Mean Arterial Pressure, ovvero pressione arteriosa media, si calcola come $\text{mean}(P)$;
- DBP: Diastolic Blood Pressure, ovvero la pressione sanguigna in diastole, si calcola come $\min(P)$;
- SBP: Systolic Blood Pressure, ovvero la pressione sanguigna in sistole, si calcola come $\max(P)$;
- PP: Pulse Pressure, ovvero la pressione di pulsazione, si calcola come $\max(P) - \min(P)$.

A partire dall'output ottenuto con i valori di C e α stimati nel codice VII.41, vengono riportati i valori delle variabili confrontandoli con i valori reali (di individui sani) nella tabella V.3.

Variabili	Unità misura	Modello	Reale
MAP	mmHg	92.91	65 - 110
DBP	mmHg	74.86	<80
SBP	mmHg	109.78	<120
PP	mmHg	34.91	~ 40

Table V.3: Valori variabili calcolati con il modello Windkessel e parametri di input (C, α) stimati. I valori reali sono ripresi da [AW20].

Il concetto di sensitività locale è formalizzato come

$$S_{\mathcal{M}}^{\mathcal{P}} = \frac{\hat{\mathcal{P}}}{\hat{\mathcal{M}}} \frac{\partial \mathcal{M}(\mathcal{P})}{\partial \mathcal{P}},$$

in cui $S_{\mathcal{M}}^{\mathcal{P}}$ è la sensitività della variabile \mathcal{M} al variare del parametro \mathcal{P} . I valori $\hat{\mathcal{M}}$ è il valore calcolato a partire dall'output del modello mentre $\hat{\mathcal{P}}$ è il valore impostato del parametro.

Per calcolare la derivata parziale si è fatto uso del *metodo delle differenze finite centrate* riadattato al calcolo della sensitività locale, come mostrato nel codice VII.43. Come variazione per ogni parametro si è usato il dieci per cento del suo valore.

Per ogni parametro $\mathcal{P} = \{C, R_1, R_2, P_d\}$ si è quindi risolta il problema ai valori iniziali (3) due volte: una con il parametro aumentato del dieci per cento, una con il parametro diminuito del dieci per cento; gli altri parametri vengono lasciati invariati. I due output di P ottenuti (insieme alla variazione del parametro) vengono poi usati nel metodo delle differenze finite centrate per l'approssimazione della derivata parziale. Infine si calcola la sensitività. L'esempio nel caso $\mathcal{P} = C$ è mostrato nel codice VII.44. La tabella V.4 riasume i valori di sensitività ottenuti in ordine decrescente in modulo.

Rank	MAP	DBP	SBP	PP
1	$R_2 (-0.1907)$	$R_2 (-0.0870)$	$C (0.2464)$	$C (0.7928)$
2	$C (0.1271)$	$C (-0.0085)$	$R_2 (-0.0871)$	$R_1 (-0.1695)$
3	$R_1 (-0.0285)$	$R_1 (-0.0066)$	$R_1 (-0.0584)$	$R_2 (-0.0871)$
4	$P_d (-0.0099)$	$P_d (-0.0006)$	$P_d (-0.0049)$	$P_d (-0.0141)$

Table V.4: Sensitività locale delle variabili $\mathcal{M} = \{MAP, DBP, SBP, PP\}$ al variare del valore dei parametri $\mathcal{P} = \{C, R_1, R_2, P_d\}$. I parametri sono inseriti in ordine decrescente (in modulo) di sensitività.

Nella tabella V.5 viene illustrata la variazione delle variabili in funzione della variazione dei parametri.

Variabili	MAP_S	$\text{MAP}_{+10\%}$	Δ_{MAP}	DBP_S	$\text{DBP}_{+10\%}$	Δ_{DBP}	SBP_S	$\text{SBP}_{+10\%}$	Δ_{SBP}	PP_S	$\text{PP}_{+10\%}$	Δ_{PP}
C	91.81	-1.18%		74.92	+0.08%		107.32	-2.24%		32.40	-7.19%	
R_1	93.18	+0.29%		74.91	+0.07%		110.43	+0.59%		35.52	+1.75%	
R_2	94.54	+1.75%		74.93	+0.09%		110.66	+0.80%		35.73	+2.35%	
P_d	93.01	+0.10%		74.86	+0.00%		109.84	+0.05%		34.98	+0.20%	

Table V.5: Variazione delle variabili all'aumento del dieci per cento dei parametri. $\mathcal{M}_S = \{\text{MAP}_S, \text{DBP}_S, \text{SBP}_S, \text{PP}_S\}$ sono le variabili ottenute con parametri standard (stimati precedentemente), $\mathcal{M}_{+10\%} = \{\text{MAP}_{+10\%}, \text{DBP}_{+10\%}, \text{SBP}_{+10\%}, \text{PP}_{+10\%}\}$ sono le variabili ottenute con singoli parametri aumentati del dieci per cento, $\Delta = \{\Delta_{MAP}, \Delta_{DBP}, \Delta_{SBP}, \Delta_{PP}\}$ sono le variazioni percentuali della variabile a pedice.

Dalla tabella V.5 si comprende che P_d ha bassa influenza in tutte le variabili.

La sensitività locale si comporta quindi come ci si aspettava: più questa è alta (in modulo) e maggiore è la variazione percentuale (in modulo) della variabile modificando il parametro.

Ad esempio nel caso $\mathcal{M} = \text{MAP}$, la variazione in percentuale è più alta modificando R_2 , seguita dalla variazione ottenuta modificando C , poi da quella modificando R_1 e P_d . Segue quindi lo stesso ordine della sensitività di MAP (in modulo).

Si nota che DBP dipende scarsamente da tutti i parametri.

Inoltre C ha forte influenza nella variazione di SBP e PP (quest'ultima influenzata non trascurabilmente anche da R_1 e R_2).

Complessivamente R_2 ha maggiore influenza di R_1 , seppur l'influenza di quest'ultima non sia trascurabile.

Si noti che per ogni parametro la variazione percentuale di ogni variabile ha segno inverso della sensibilità. Questo deriva dal metodo delle differenze finite centrate: con una modifica negativa si otterrebbe una variazione percentuale concorde con il segno della sensibilità.

Metodologia e risultati training

In questo capitolo viene spiegato il problema di regressione di interesse dell'elaborato e il processo di training del modello. Viene introdotta la libreria di python usata per la parte di training del modello riportando i dettagli di programmazione; vengono infine riportati i risultati ottenuti.

A baby learns to crawl, walk and then run. We are in the crawling stage when it comes to applying machine learning.

Dave Waters

VI.1 Problema di regressione

Nel capitolo V è stato mostrato il funzionamento del modello Windkessel: a partire da un flusso cardiaco, resistenza prossimale, resistenza periferica e compliance il modello è in grado di dare un'approssimazione dell'andamento della pressione durante il ciclo cardiaco.

Il problema che si pone l'elaborato è di ottenere MAP, DBP, SBP, PP senza dover risolvere l'equazione differenziale del modello. Per farlo viene usato l'approccio dell'apprendimento supervisionato: si genera un corposo database di combinazioni dei parametri di input associati all'output trovato risolvendo l'equazione del modello windkessel (come visto nel capitolo V), si divide il dataset in training, validation e testing set (come visto in IV.2), si esegue il training con processi gaussiani come spiegato in IV.4.5.

VI.2 Dettagli del training

VI.2.1 GPErks

Per il training è stata usata la libreria di python GPErks, progetto sviluppato dal dottor Stefano Longobardi e che ha usato in applicazioni mediche, ad esempio in [Lon+20].

Per la creazione della libreria è stata usato principalmente PyTorch, libreria di python molto comune nel machine learning.

Poiché l'elaborato avrebbe sfruttato la libreria GPErks, la base teorica sulla quale poggia il suo funzionamento è stata spiegata nei capitoli precedenti, in particolare nel capitolo IV.

In particolare GPErks divide il database di input (formato da coppie input - output di esempi) in training set, validation set e testing set. Definisce il processo gaussiano con una mean function lineare e uno squared exponential kernel (non sono scelte obbligate, ma sono le più comuni). Definisce la marginal likelihood e utilizza un metodo di ottimizzazione per la sua massimizzazione (è possibile scegliere il metodo di ottimizzazione, nel caso dell'elaborato è stato scelto il metodo di Adam). La libreria ha implementato tre diversi earlystopper utili a terminare il training prima di incorrere in overfitting.

GPErks mostra inoltre i risultati del training in forma di grafici della loss function e riporta anche i valori di *metriche* (come vengono chiamate nella libreria) come R2Score e il Mean Squared Error (le due scelte per i fini dell'elaborato). Genera infine risultati di tipo inferenziali sul training, mostrando come i risultati dell'apprendimento riescano a prevedere gli output a partire dai dati di input.

VI.2.2 Processo gaussiano

Viene usato un processo gaussiano con mean function $m(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$ e covariance function lo squared exponential kernel con un parametro di lengthscale per ogni dimensione (come descritto in III.3.4).

Nello specifico, nell'elaborato si segue la struttura del codice dell'esempio quattro presente nella pagina github della libreria.

VI.2.3 Ottimizzazione degli iperparametri

Viene massimizzata la marginal likelihood, seguendo quanto detto in IV.4.5. Per farlo viene utilizzato il metodo di Adams (spiegato in IV.5.7).

VI.2.4 Dataset di input

Il dataset di input è costituito da un file con i parametri di input: C , R_1 (resistenza prossimale), R_2 (resistenza distale). Non viene aggiunto P_d ai dati di input per quanto visto in V.5: la sensitività locale di P_d (cioè l'influenza sui parametri di output) è molto bassa (< 0.02 in modulo), che significa che non c'è una relazione causale tra la pressione distale e i parametri di output. Si noti che lasciare P_d nella lista dei parametri di input può peggiorare i risultati del training in quanto il modello cercherebbe di imparare una relazione causale tra P_d e gli output mentre vi è solo una relazione casuale.

Per generarlo si creano tre liste, una per ogni parametro di input; in particolare:

$$C \in [1.4, 2.6] \quad R_1 \in [0.01, 0.1] \quad R_2 \in [0.6, 1.3]$$

Per ogni parametro la lista consiste di quindici elementi. Vengono poi generate tutte le combinazioni di parametri, mescolate¹ e inserite in un file.

Per ogni combinazione di parametri di input viene poi risolto il modello Windkessel, come fatto nel capitolo V. Con il risultato vengono trovati MAP, DBP, SBP, PP e vengono salvati in un file per ogni output. In questo modo si ottengono quattro file con un input per ogni combinazione di parametri contenute nel primo file generato.

Il dataset di input viene poi diviso (da GPErks) in training set, validation set e testing set. La proporzione è 60% training set, 20% validation set, 20% testing set.

¹Mischiare le combinazioni prima di dividere il dataset in training, validation e testing set evita problemi di overfitting.

In figura VI.1 viene mostrata la distribuzione del database. Questa dipende dal fatto che vengono costruite tutte le combinazioni possibili di dati di input.

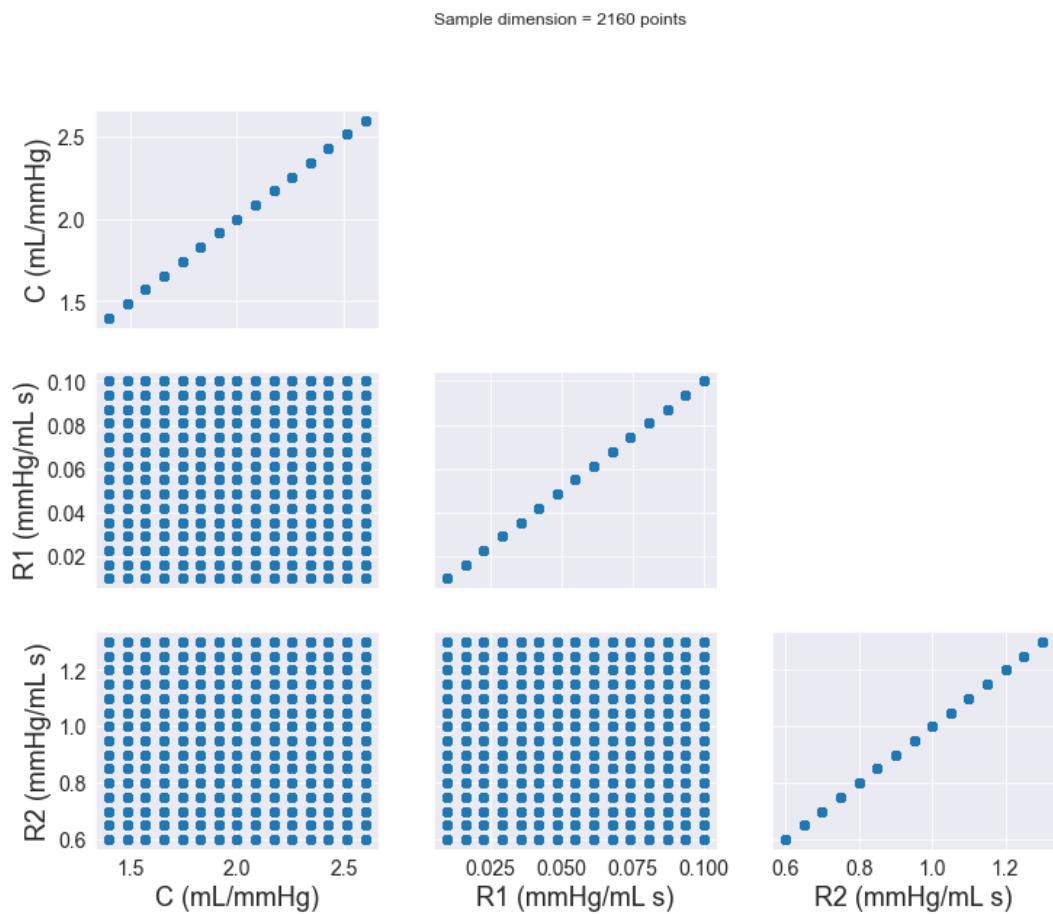


Figure VI.1: Distribuzione dei dati nel database.

VI.3 Risultati del training

VI.3.1 Mean arterial pressure (MAP)

Viene impostato lr = 0.1 e viene usato l'early stopper *GLEarlyStoppingCriterion* con parametri: $\alpha = 5$, patience = 2.

Training e validation loss

Il training ha necessitato cinquantadue EPOCHS (ogni EPOCH consiste in valutare il gradiente su ogni elemento del dataset), ha concluso con R2Score = 0.9999, MeanSquaredError = 0.0001. In figura VI.2 viene mostrato l'andamento del training e validation loss con MSE e R2Score; in verde l'andamento dell'early stopper.

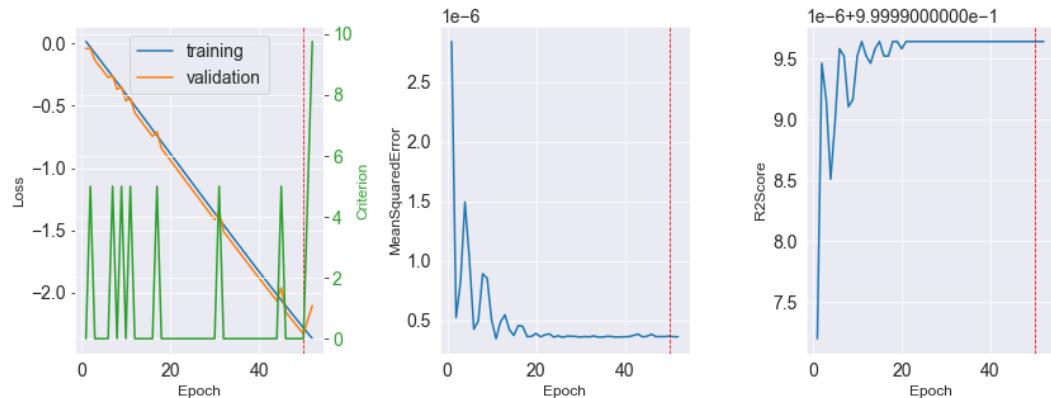


Figure VI.2: MAP: andamento del training e validation loss, early stopper, R2Score e MSE.

Approssimazione dei dati di input

In figura VI.3 viene mostrato come le predizioni approssimano i dati di input. La lunghezza delle barre di errore è 0.0015, quindi sono molto corte indicando un'alta precisione.

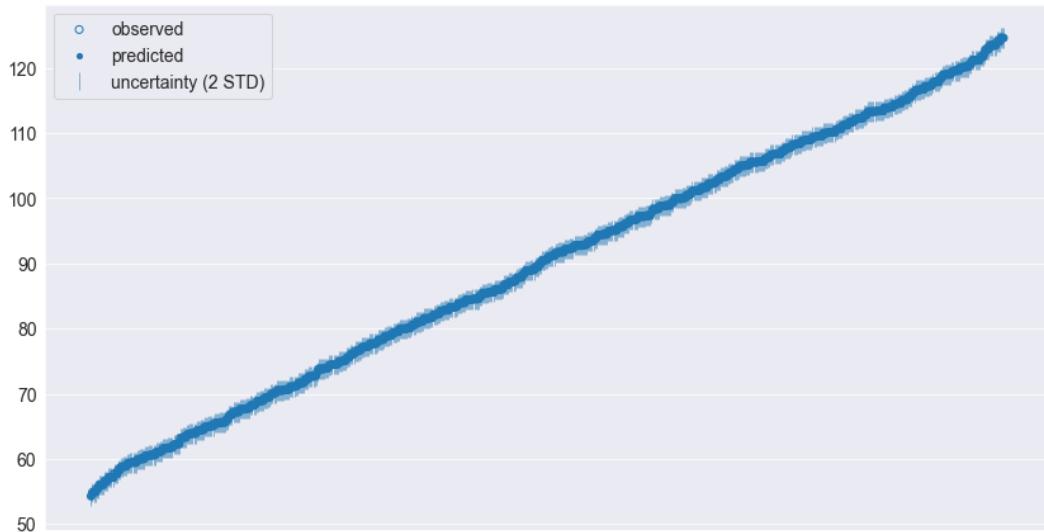


Figure VI.3: MAP: predizioni sui dati di input.

Dipendenza da C

Per studiare la dipendenza di MAP da C , vengono presi novanta valori di C equidistanziati nello stesso intervallo usato per la creazione del database di input e, fissati i valori di R_1 e R_2 a quelli trovati nella loro approssimazione in V.3.3, viene generato un file di combinazioni C , R_1 e R_2 . Per ogni combinazione viene trovata l'approssimazione della pressione con il modello Windkessel e calcolata la MAP; con la combinazione di compliance e resistenze viene poi stimata la MAP usando il modello già addestrato. Viene poi fatto lo stesso su due intervalli attigui a quello di training e ampi il 10% dell'intervallo di training. Il risultato complessivo è mostrato in figura VI.4, il risultato nel solo intervallo di training in VI.5, il risultato nei singoli intervalli attigui in VI.6 e VI.7.

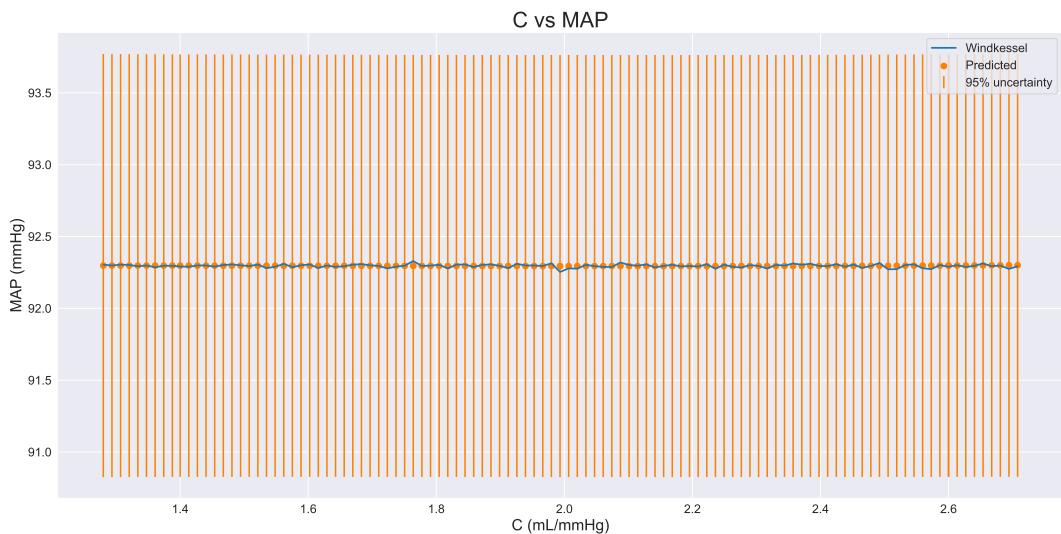


Figure VI.4: Dipendenza di MAP da C sull'intervallo di training e due intervalli attigui.

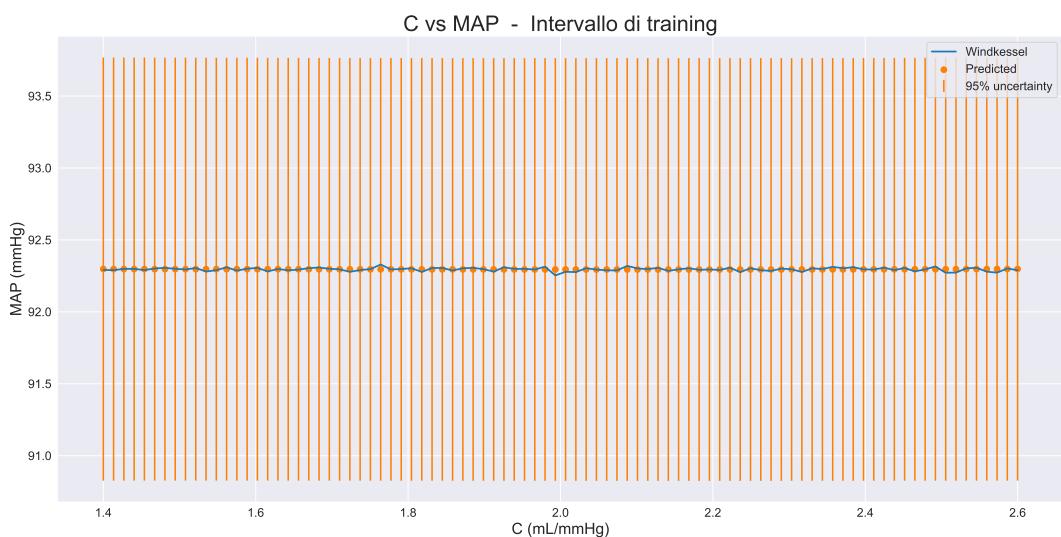


Figure VI.5: Dipendenza di MAP da C sull'intervallo di training.

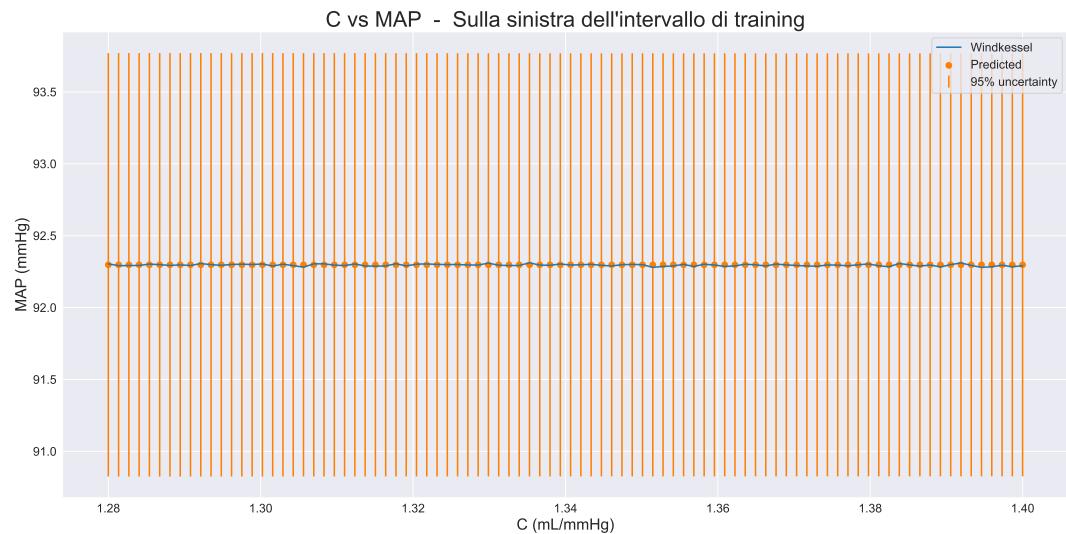


Figure VI.6: Dipendenza di MAP da C sull’intervallo attiguo a sinistra dell’intervallo di training.



Figure VI.7: Dipendenza di MAP da C sull’intervallo attiguo a destra dell’intervallo di training.

Dipendenza da R_1

Per studiare la dipendenza da R_1 viene usato lo stesso approccio. Il risultato complessivo è mostrato in figura VI.8, il risultato nel solo intervallo di training in VI.9, il risultato nei singoli intervalli attigui in VI.10 e VI.11.

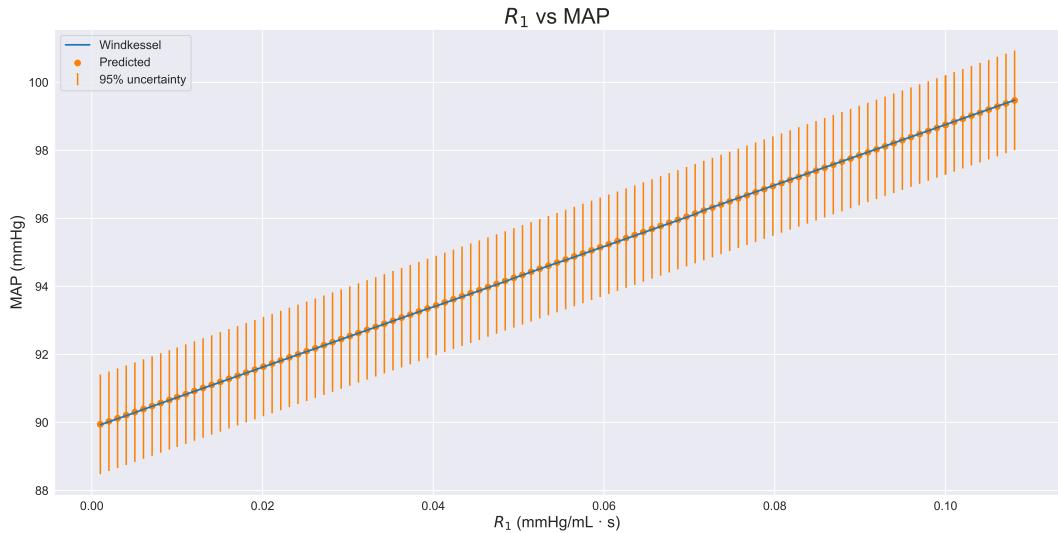


Figure VI.8: Dipendenza di MAP da R_1 sull’intervallo di training e due intervalli attigui.

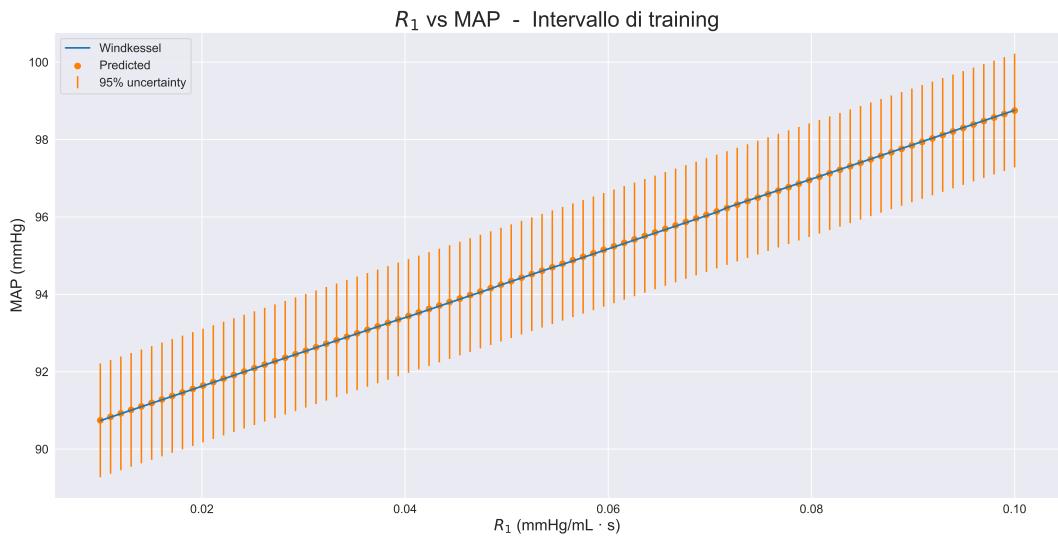


Figure VI.9: Dipendenza di MAP da R_1 sull’intervallo di training.

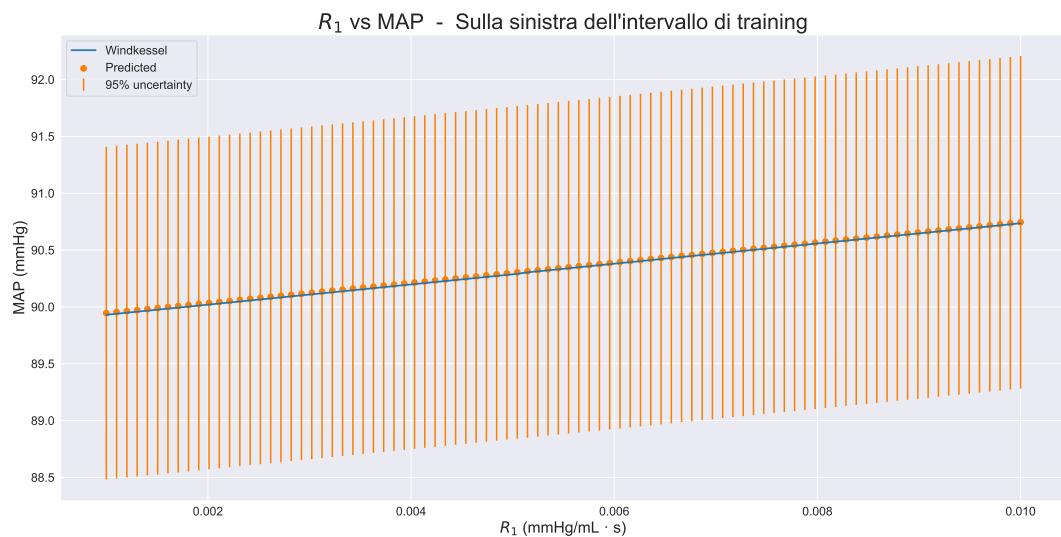


Figure VI.10: Dipendenza di MAP da R_1 sull’intervallo attiguo a sinistra dell’intervallo di training.

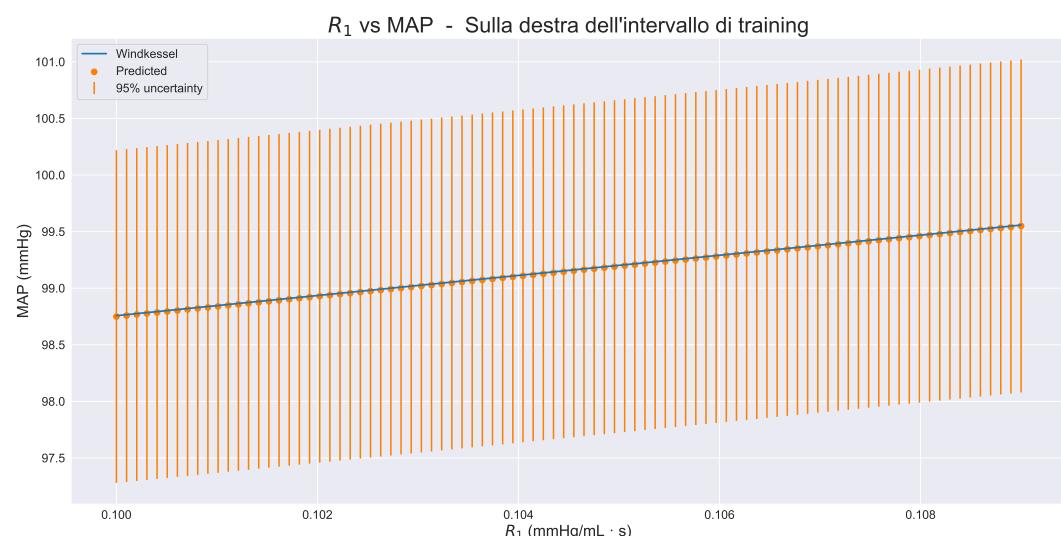


Figure VI.11: Dipendenza di MAP da R_1 sull’intervallo attiguo a destra dell’intervallo di training.

Dipendenza da R_2

Per studiare la dipendenza da R_2 viene usato lo stesso approccio. Il risultato complessivo è mostrato in figura VI.12, il risultato nel solo intervallo di training in VI.13, il risultato nei singoli intervalli attigui in VI.14 e VI.15.

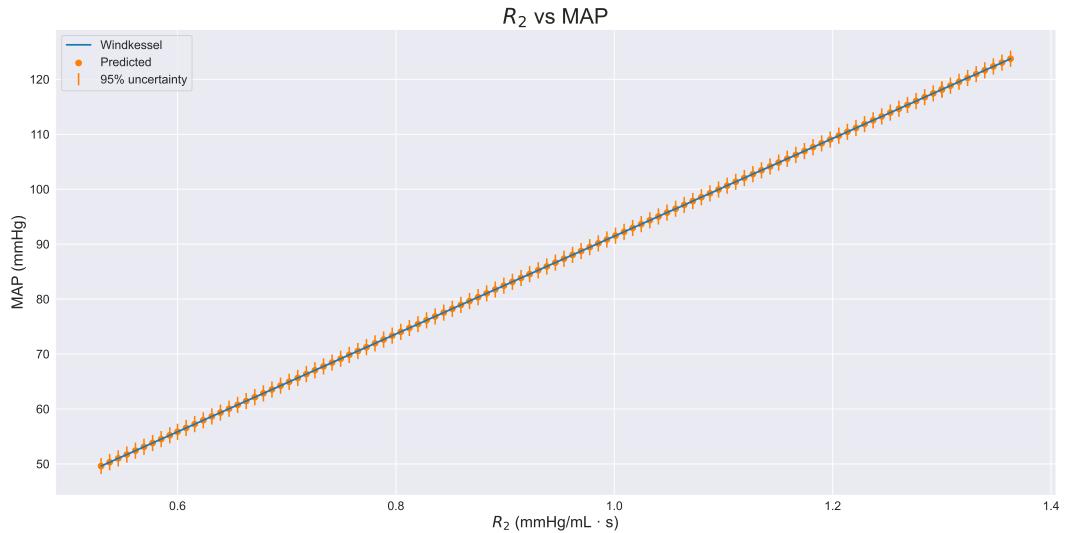


Figure VI.12: Dipendenza di MAP da R_2 sull’intervallo di training e due intervalli attigui.

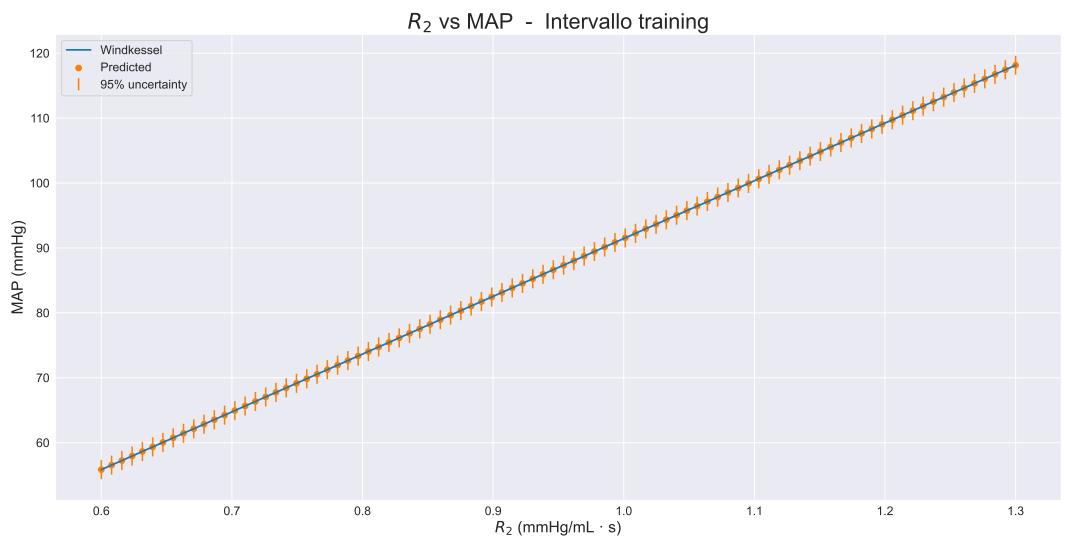


Figure VI.13: Dipendenza di MAP da R_2 sull’intervallo di training.

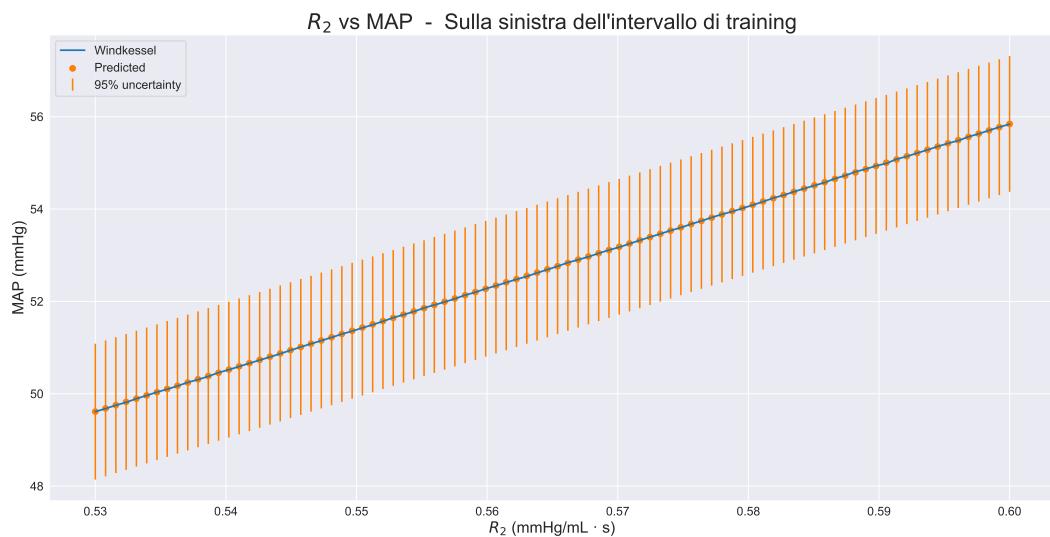


Figure VI.14: Dipendenza di MAP da R_2 sull'intervallo attiguo a sinistra dell'intervallo di training.

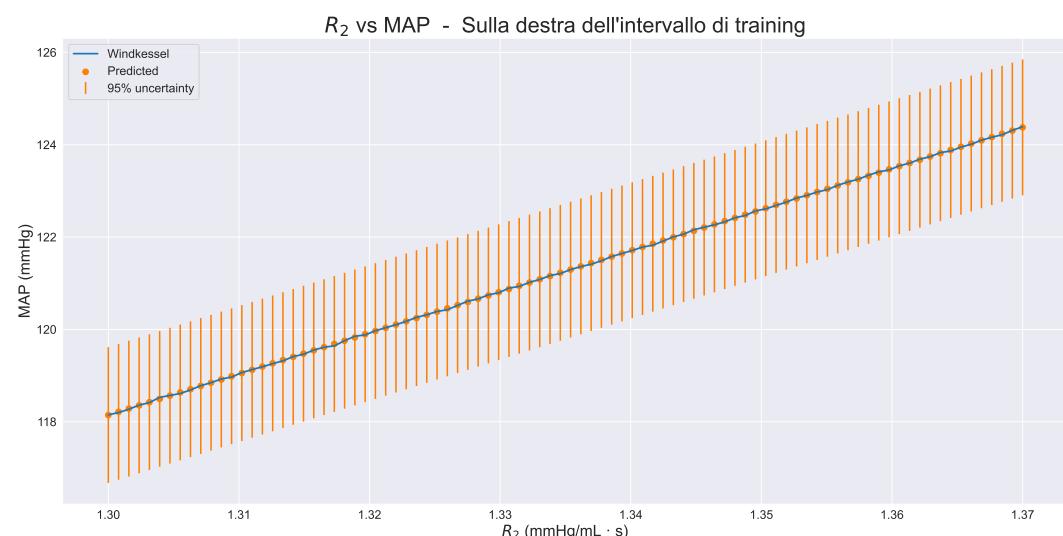


Figure VI.15: Dipendenza di MAP da R_2 sull'intervallo attiguo a destra dell'intervallo di training.

VI.3.2 Diastolic blood pressure (DBP)

Viene imposto lr = 0.07 e viene usato l'early stopper *GLEarlyStoppingCriterion* con parametri: $\alpha = 2$, patience = 2.

Training e validation loss

Il training ha necessitato centotrentuno EPOCHS, ha concluso con R2Score = 0.9999, MeanSquaredError = 0.0001. In figura VI.16 viene mostrato l'andamento del training e validation loss con MSE e R2Score; in verde l'andamento dell'early stopper.

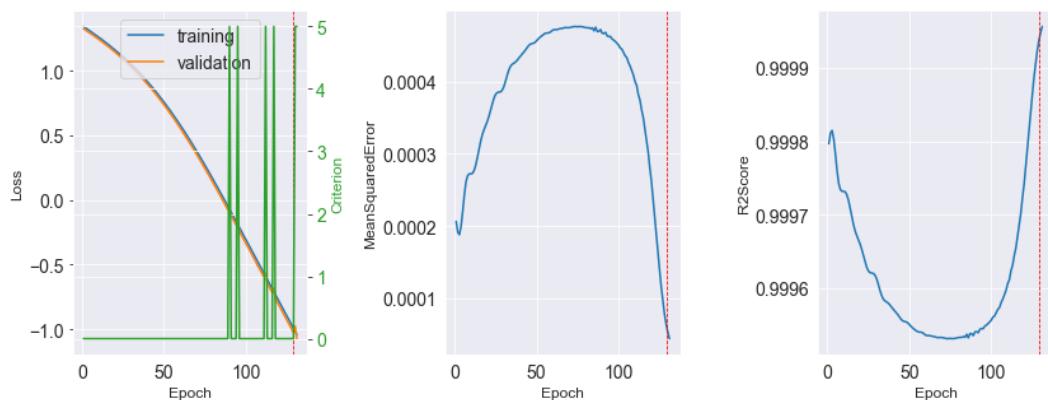


Figure VI.16: DBP: andamento del training e validation loss, early stopper, R2Score e MSE.

Approssimazione dei dati di input

In figura VI.17 viene mostrato come le predizioni approssimano i dati di input. La lunghezza delle barre di errore è 0.0028.



Figure VI.17: DBP: predizioni sui dati di input.

Dipendenza da C

Il risultato complessivo è mostrato in figura VI.18, il risultato nel solo intervallo di training in VI.19, il risultato nei singoli intervalli attigui in VI.20 e VI.21. In tutti gli intervalli il modello riesce a generare ottime predizioni con una bassa incertezza (rappresentata dalle barre di errore).

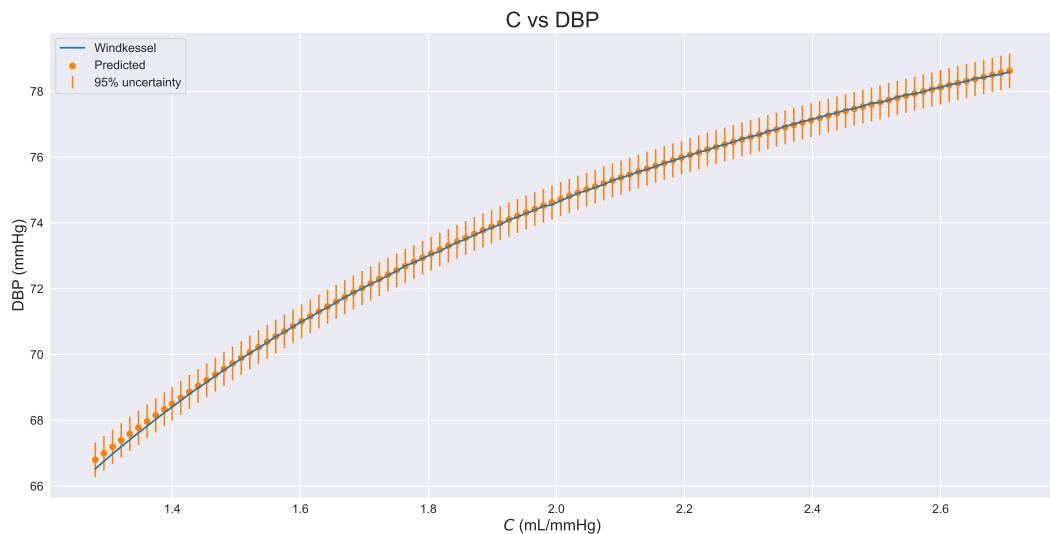


Figure VI.18: Dipendenza di DBP da C sull’intervallo di training e due intervalli attigui.



Figure VI.19: Dipendenza di DBP da C sull’intervallo di training.

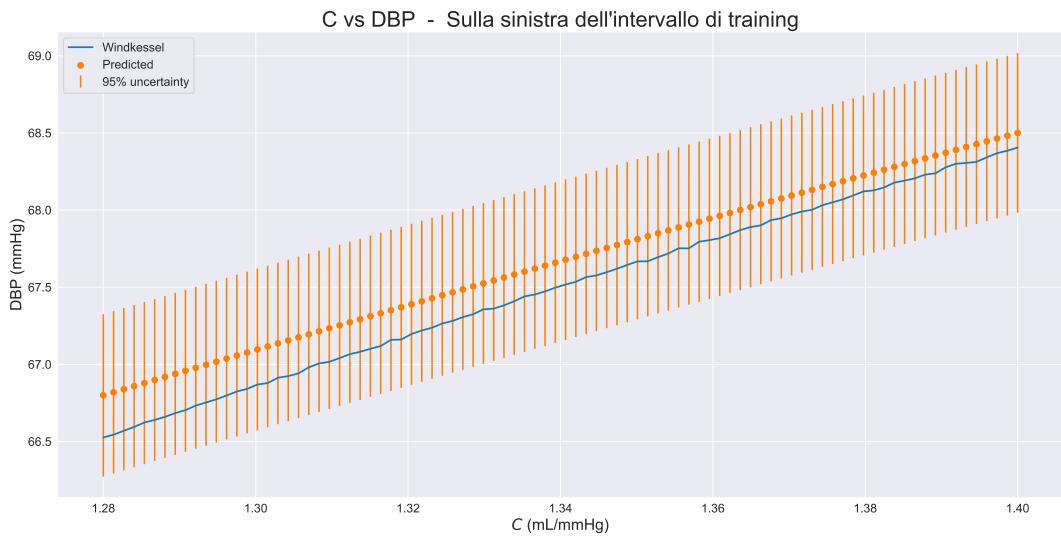


Figure VI.20: Dipendenza di DBP da C sull'intervallo attiguo a sinistra dell'intervallo di training.

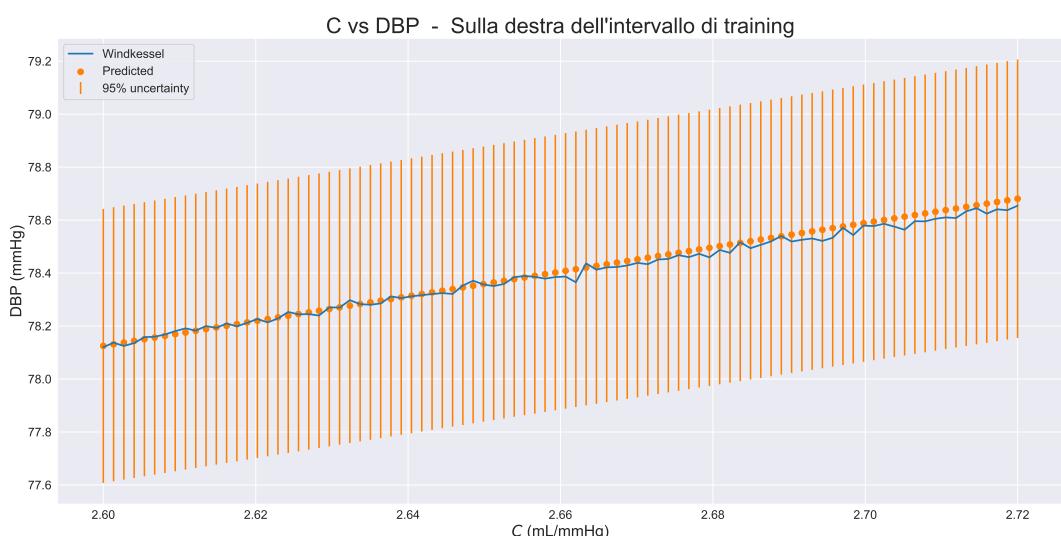


Figure VI.21: Dipendenza di DBP da C sull'intervallo attiguo a destra dell'intervallo di training.

Dipendenza da R_1

Il risultato complessivo è mostrato in figura VI.22, il risultato nel solo intervallo di training in VI.23, il risultato nei singoli intervalli attigui in VI.24 e VI.25. Anche in questo caso il modello è in grado di fare predizioni molto precise.

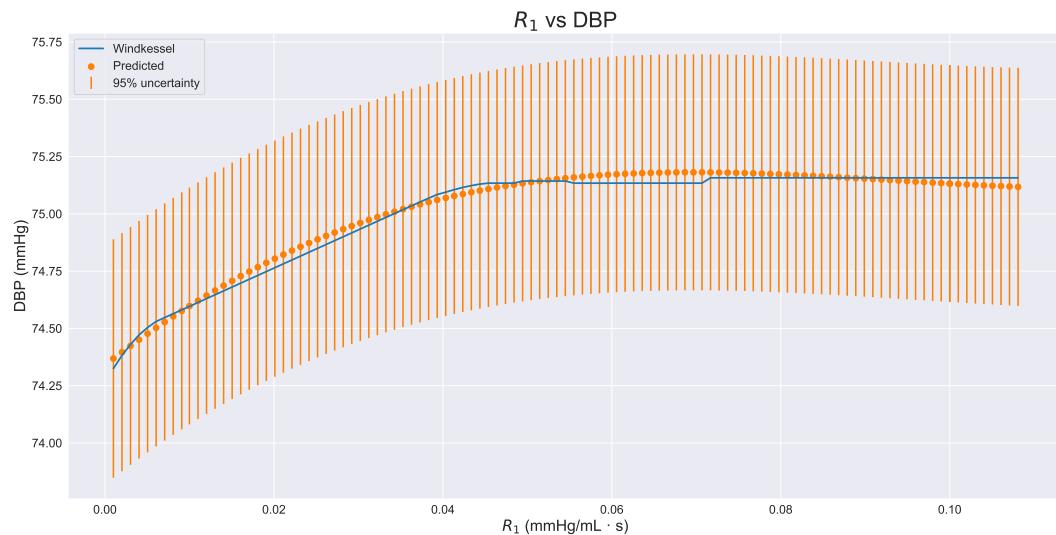


Figure VI.22: Dipendenza di DBP da R_1 sull’intervallo di training e due intervalli attigui.

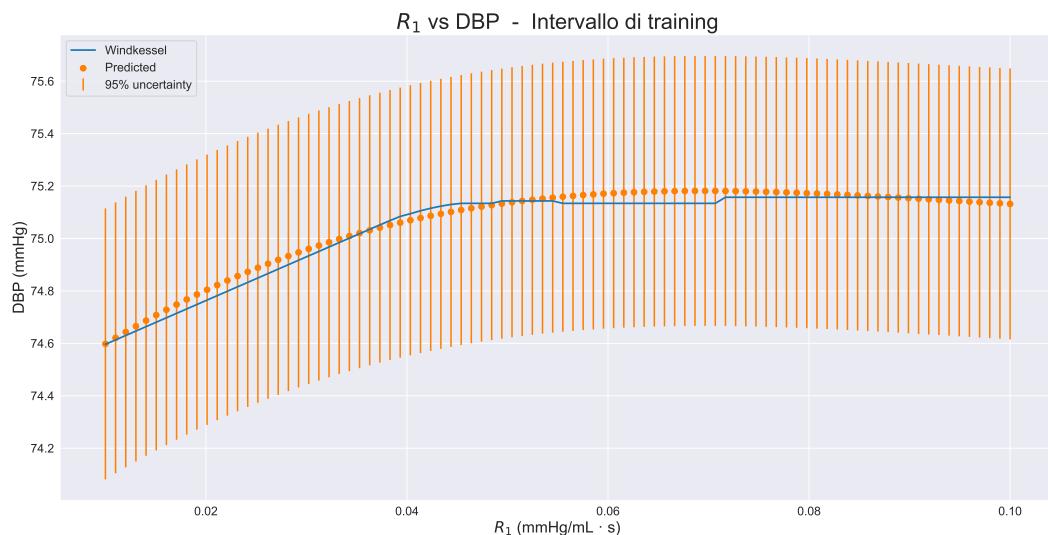


Figure VI.23: Dipendenza di DBP da R_1 sull’intervallo di training.

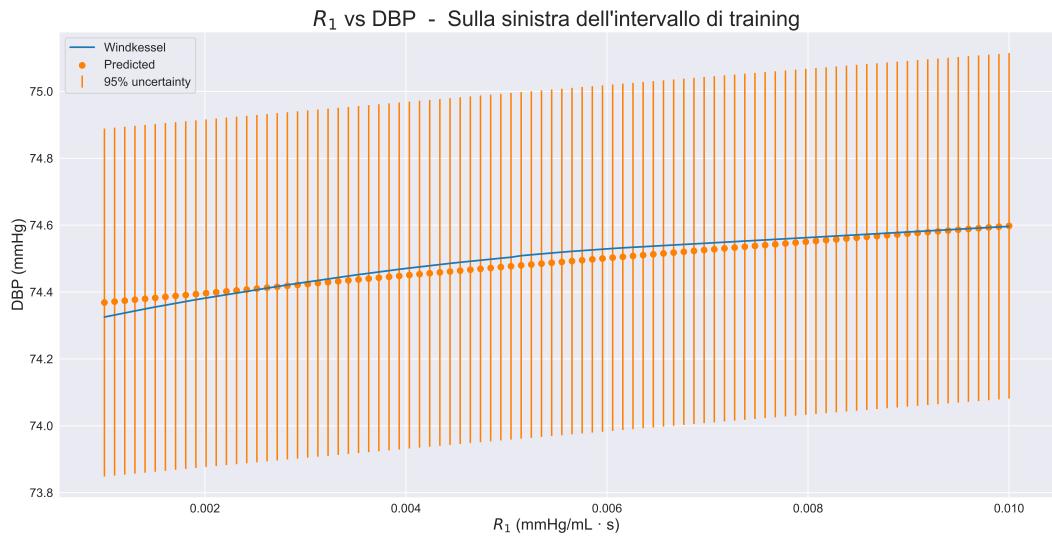


Figure VI.24: Dipendenza di DBP da R_1 sull'intervallo attiguo a sinistra dell'intervallo di training.

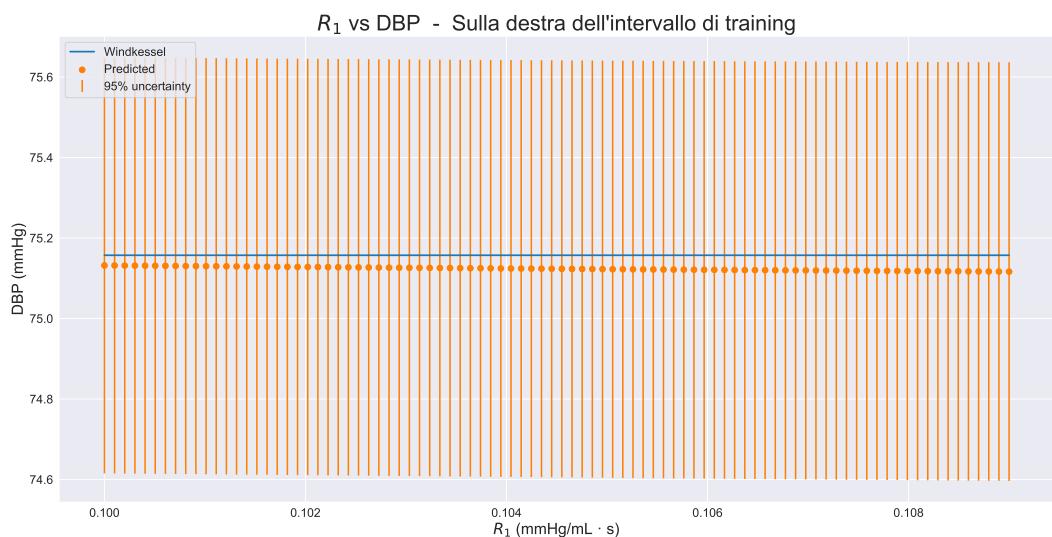


Figure VI.25: Dipendenza di DBP da R_1 sull'intervallo attiguo a destra dell'intervallo di training.

Dipendenza da R_2

Il risultato complessivo è mostrato in figura VI.26, il risultato nel solo intervallo di training in VI.27, il risultato nei singoli intervalli attigui in VI.28 e VI.29. Anche in questo caso si hanno predizioni molto precise. A prima vista sembra che le barre di errore non compaiano, in realtà sono molto brevi rispetto alla scala usata. Infatti nei grafici in cui si plottano solo gli intervalli attigui le barre di errore ricompaiono e mostrano che l'errore è minore di una unità.

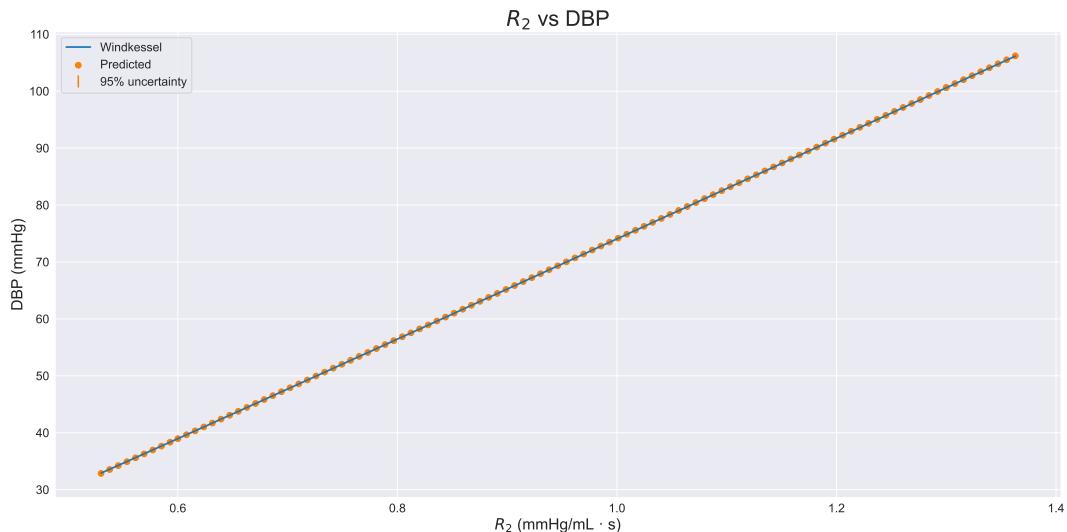


Figure VI.26: Dipendenza di DBP da R_2 sull'intervallo di training e due intervalli attigui.

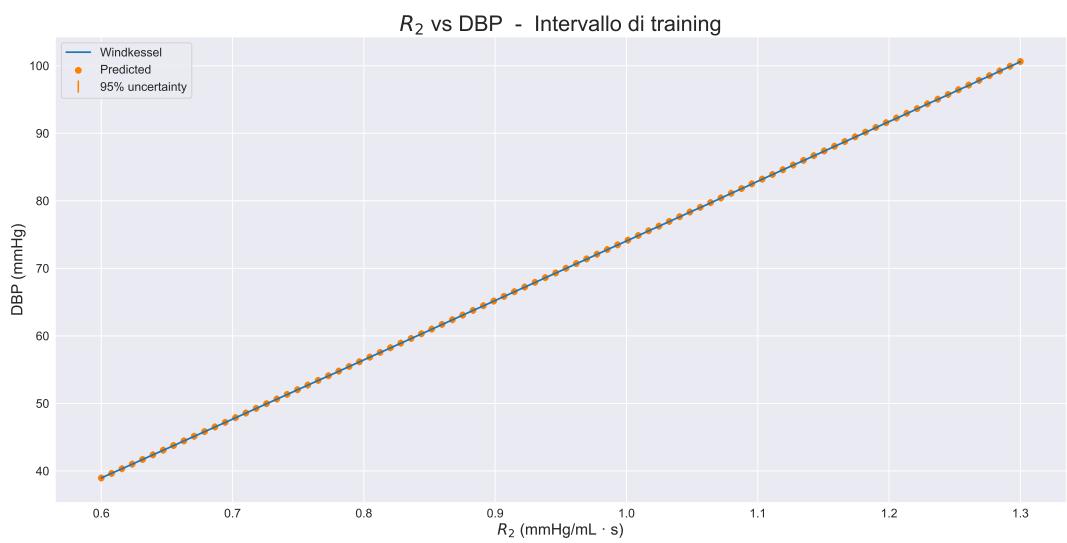


Figure VI.27: Dipendenza di DBP da R_2 sull'intervallo di training.



Figure VI.28: Dipendenza di DBP da R_2 sull’intervallo attiguo a sinistra dell’intervallo di training.

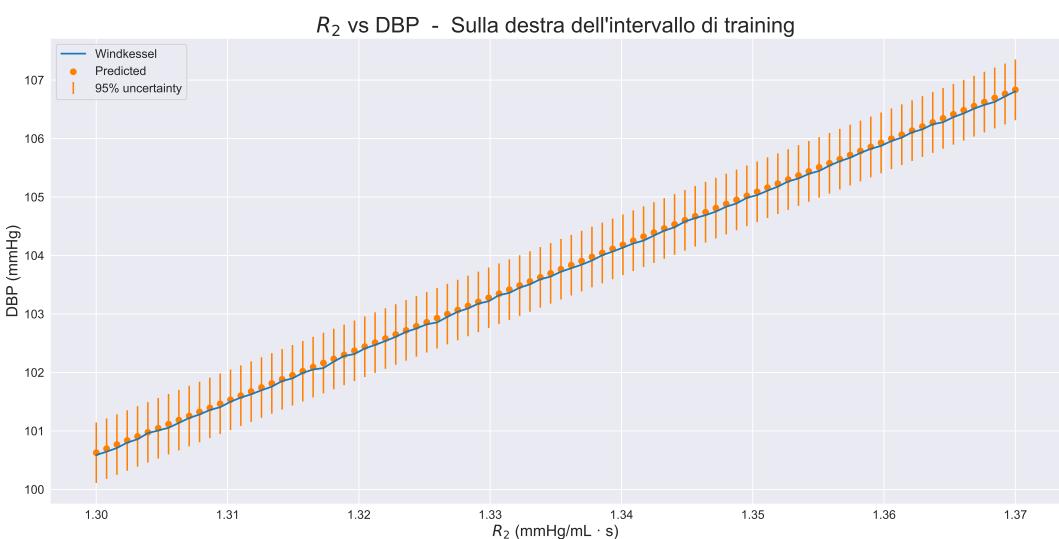


Figure VI.29: Dipendenza di DBP da R_2 sull’intervallo attiguo a destra dell’intervallo di training.

VI.3.3 Pulse pressure (PP)

Viene imposto lr = 0.05 e viene usato l'early stopper *GLEarlyStoppingCriterion* con parametri: $\alpha = 5$, patience = 3.

Training e validation loss

Il training ha necessitato novantatre EPOCHS, ha concluso con R2Score = 0.9994, MeanSquaredError = 0.0006. In figura VI.30 viene mostrato l'andamento del training e validation loss con MSE e R2Score; in verde l'andamento dell'early stopper.

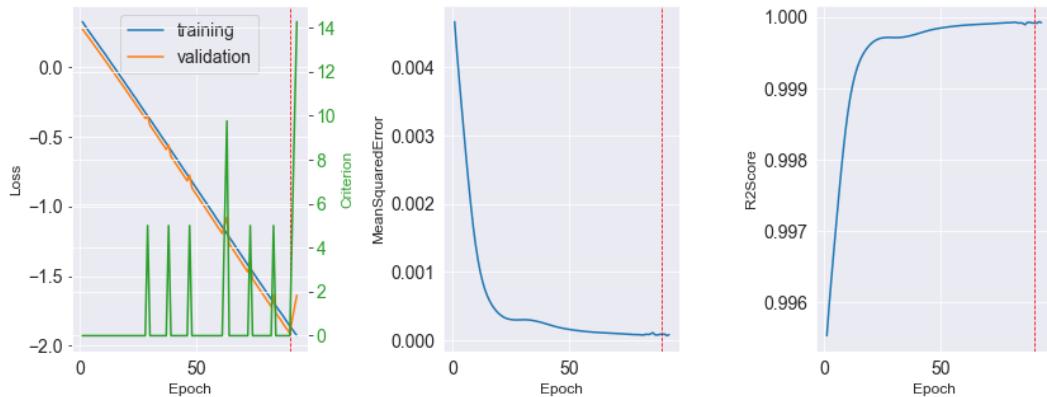


Figure VI.30: PP: andamento del training e validation loss, early stopper, R2Score e MSE.

Approssimazione dei dati di input

In figura VI.31 viene mostrato come le predizioni approssimano i dati di input. La lunghezza delle barre di errore è 0.0034.



Figure VI.31: PP: predizioni sui dati di input.

Dipendenza da C

Il risultato complessivo è mostrato in figura VI.32, il risultato nel solo intervallo di training in VI.33, il risultato nei singoli intervalli attigui in VI.34 e VI.35.

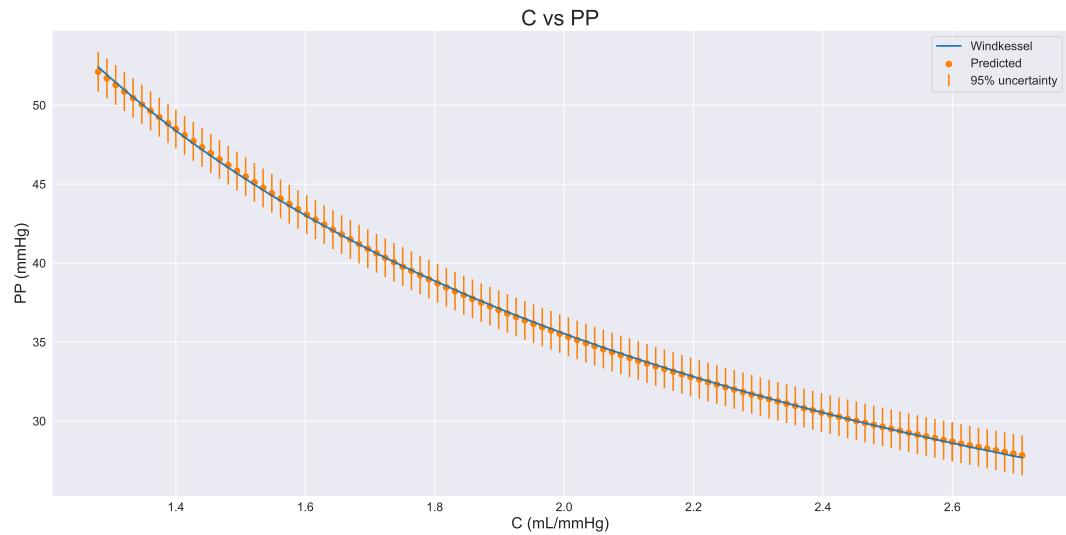


Figure VI.32: Dipendenza di PP da C sull'intervallo di training e due intervalli attigui.

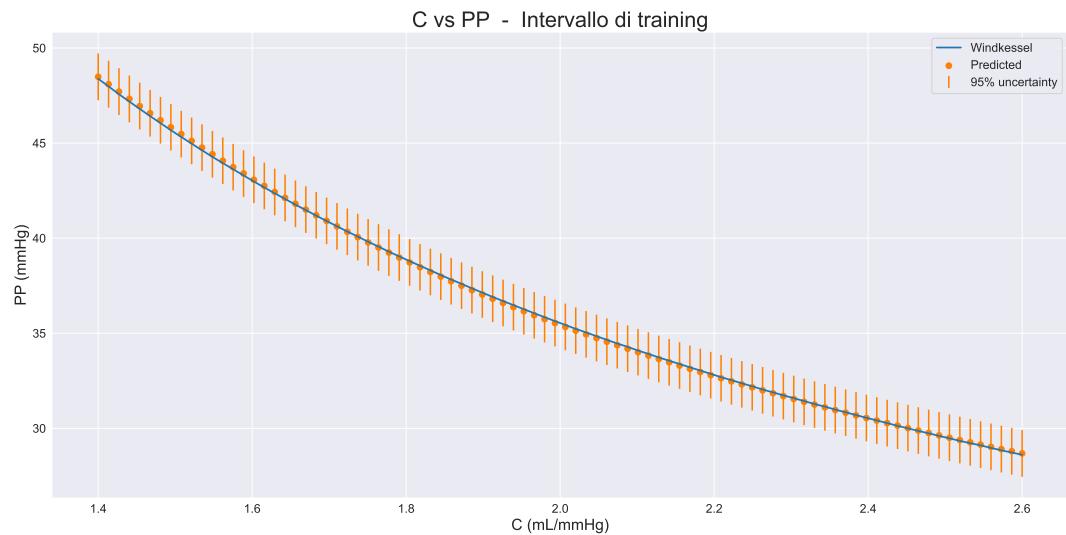


Figure VI.33: Dipendenza di PP da C sull'intervallo di training.



Figure VI.34: Dipendenza di PP da C sull'intervallo attiguo a sinistra dell'intervallo di training.

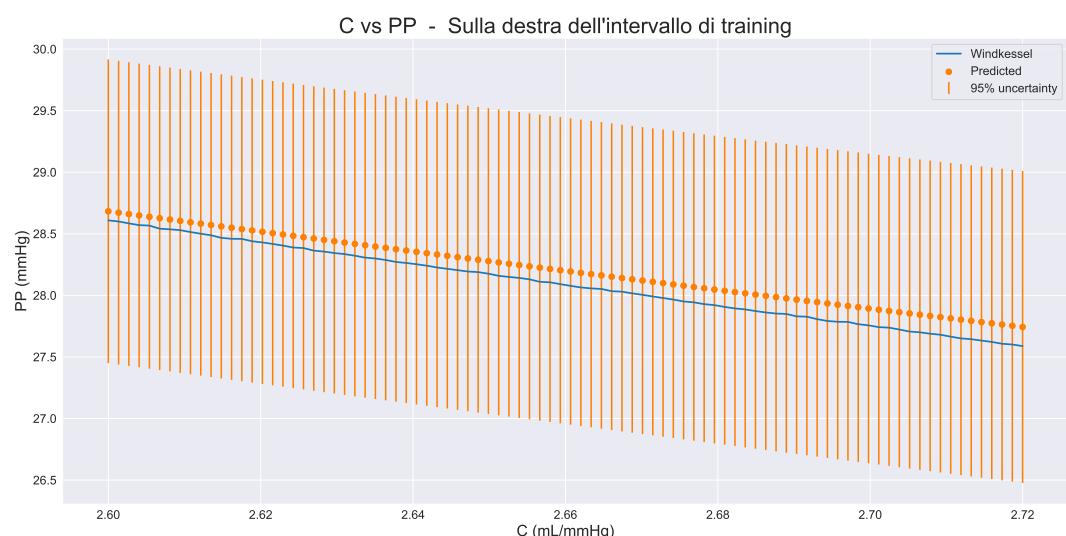


Figure VI.35: Dipendenza di PP da C sull'intervallo attiguo a destra dell'intervallo di training.

Dipendenza da R_1

Il risultato complessivo è mostrato in figura VI.36, il risultato nel solo intervallo di training in VI.37, il risultato nei singoli intervalli attigui in VI.38 e VI.39.

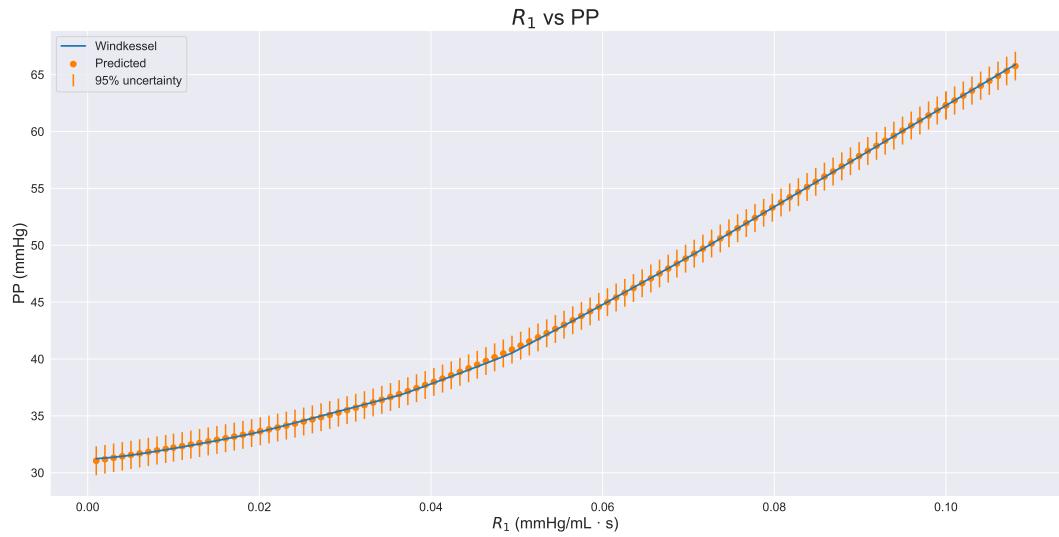


Figure VI.36: Dipendenza di PP da R_1 sull’intervallo di training e due intervalli attigui.

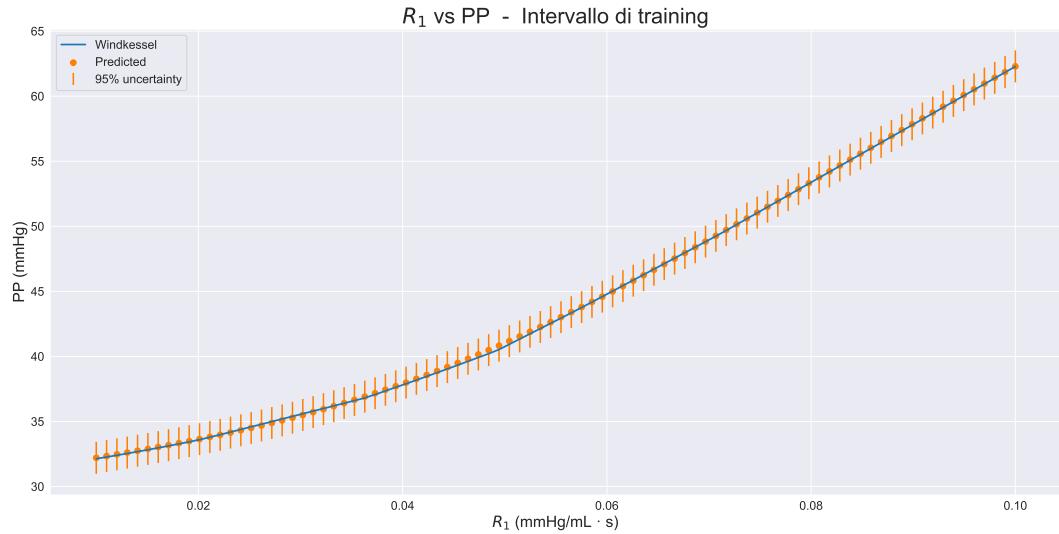


Figure VI.37: Dipendenza di PP da R_1 sull’intervallo di training.

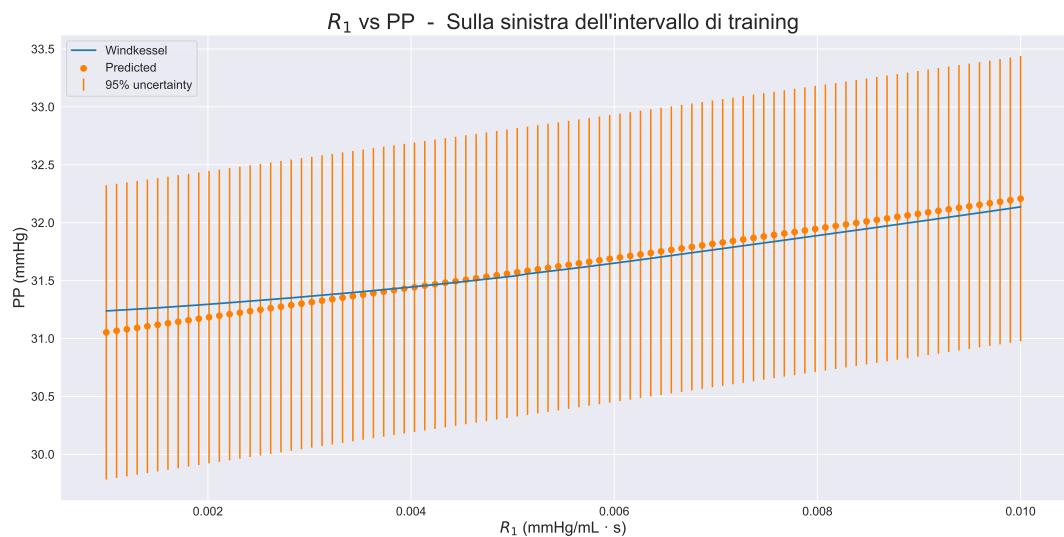


Figure VI.38: Dipendenza di PP da R_1 sull'intervallo attiguo a sinistra dell'intervallo di training.

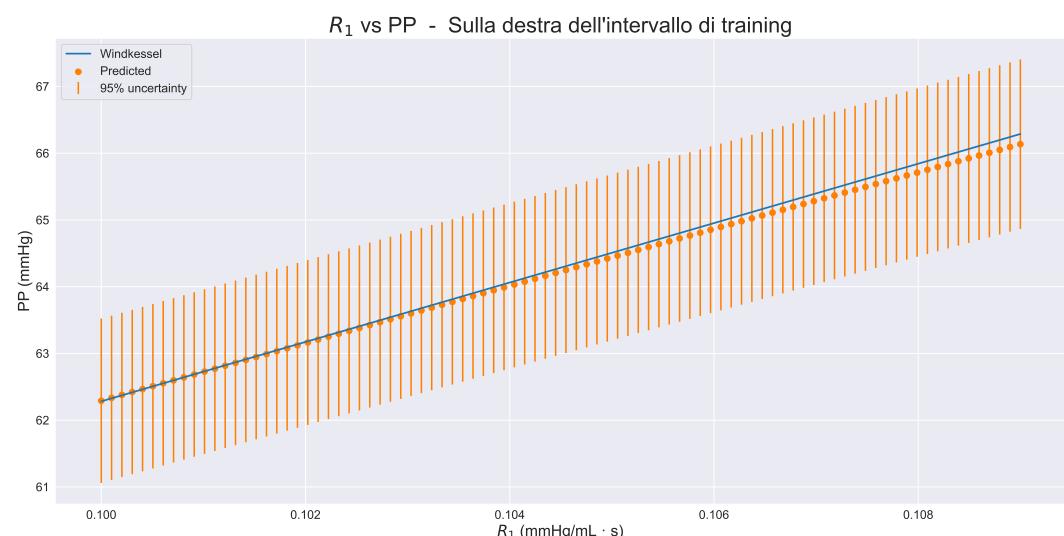


Figure VI.39: Dipendenza di PP da R_1 sull'intervallo attiguo a destra dell'intervallo di training.

Dipendenza da R_2

Il risultato complessivo è mostrato in figura VI.40, il risultato nel solo intervallo di training in VI.41, il risultato nei singoli intervalli attigui in VI.42 e VI.43.

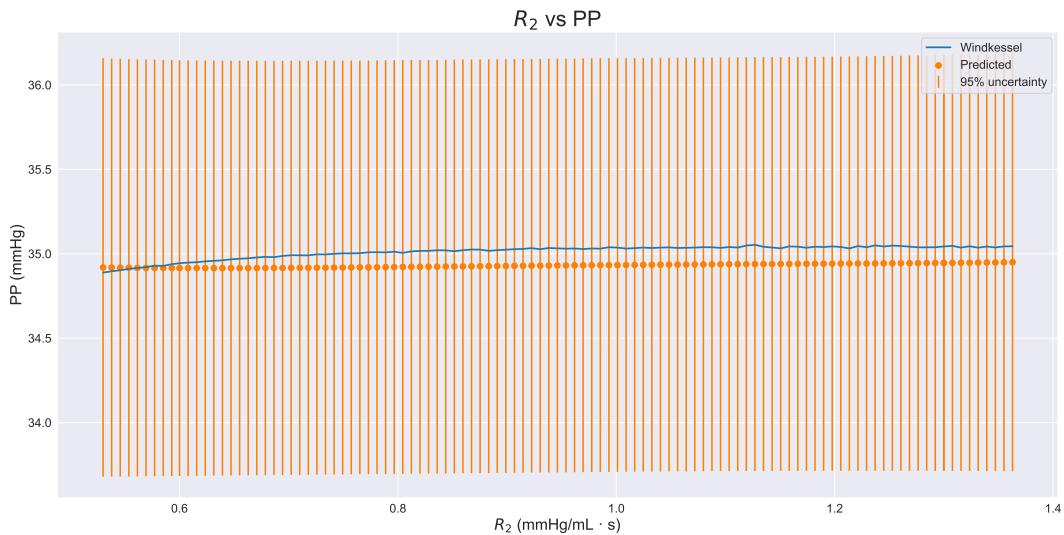


Figure VI.40: Dipendenza di PP da R_2 sull’intervallo di training e due intervalli attigui.

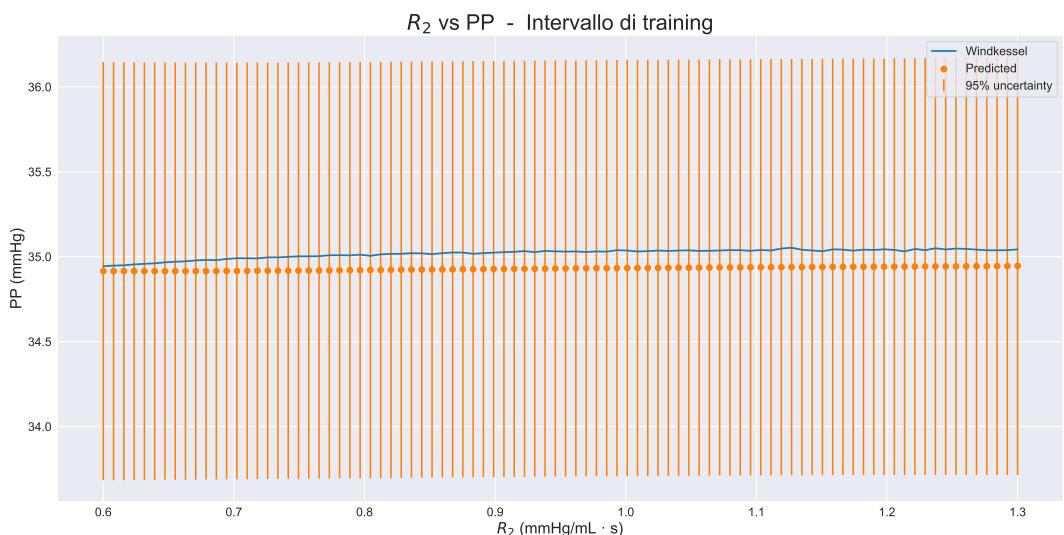


Figure VI.41: Dipendenza di PP da R_2 sull’intervallo di training.

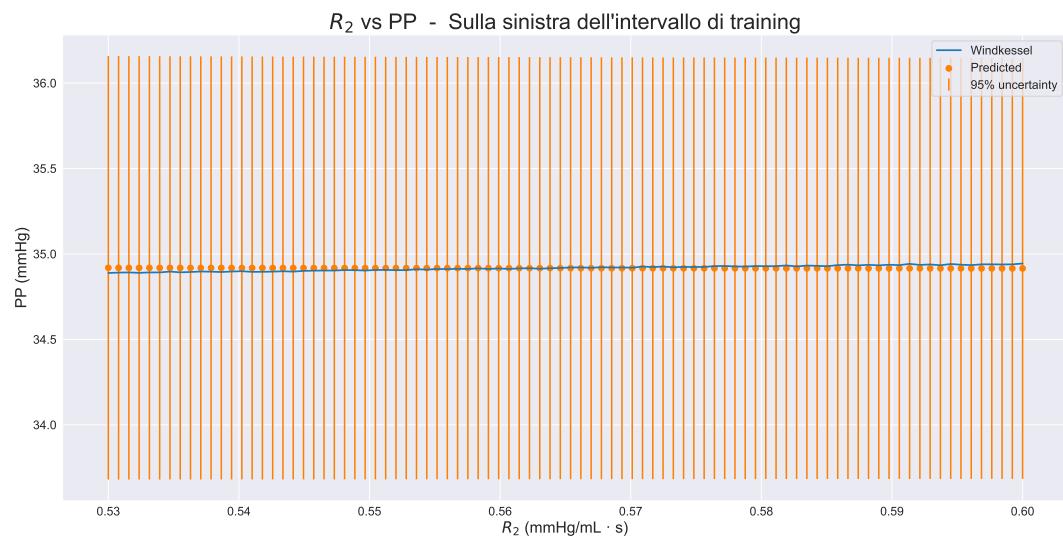


Figure VI.42: Dipendenza di PP da R_2 sull'intervallo attiguo a sinistra dell'intervallo di training.

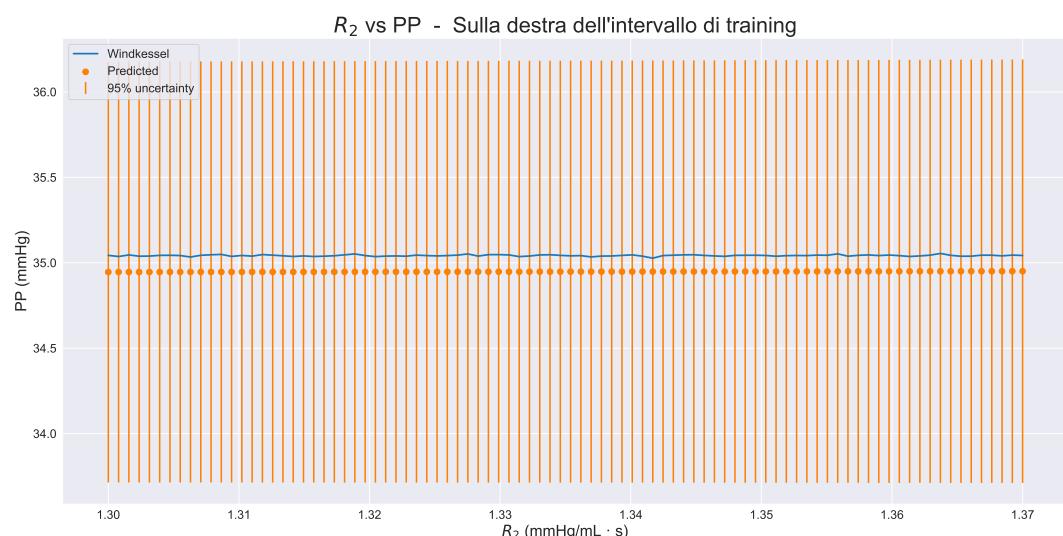


Figure VI.43: Dipendenza di PP da R_2 sull'intervallo attiguo a destra dell'intervallo di training.

VI.3.4 Systolic blood pressure (SBP)

Viene imposto $lr = 0.07$ e viene usato l'early stopper *GLEarlyStoppingCriterion* con parametri: $\alpha = 5$, patience = 1.

Training e validation loss

Il training ha necessitato centotredici EPOCHS, ha concluso con $R2Score = 0.9999$, $MeanSquaredError = 0.0052$. In figura VI.44 viene mostrato l'andamento del training e validation loss con MSE e R2Score; in verde l'andamento dell'early stopper.

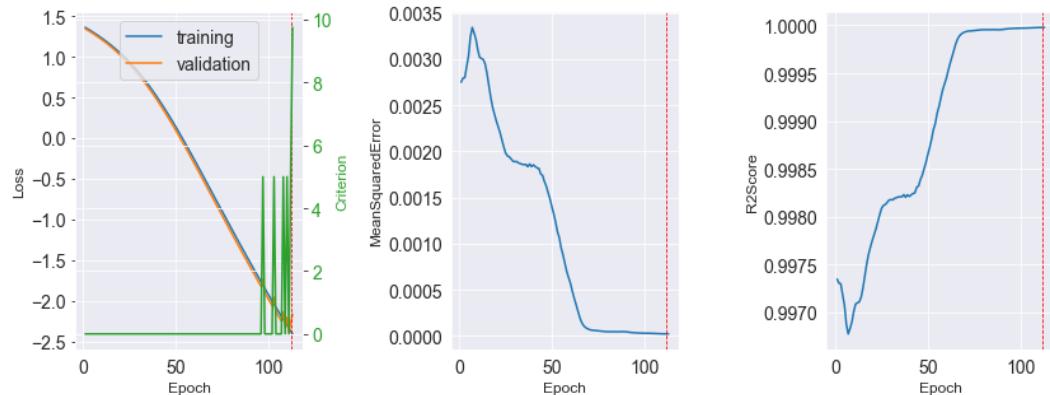


Figure VI.44: SBP: andamento del training e validation loss, early stopper, $R2Score$ e MSE.

Approssimazione dei dati di input

In figura VI.45 viene mostrato come le predizioni approssimano i dati di input. Le barre di errore sono lunghe 0.0012, dunque non si notano.



Figure VI.45: SBP: predizioni sui dati di input.

Dipendenza da C

Il risultato complessivo è mostrato in figura VI.46, il risultato nel solo intervallo di training in VI.47, il risultato nei singoli intervalli attigui in VI.48 e VI.49.

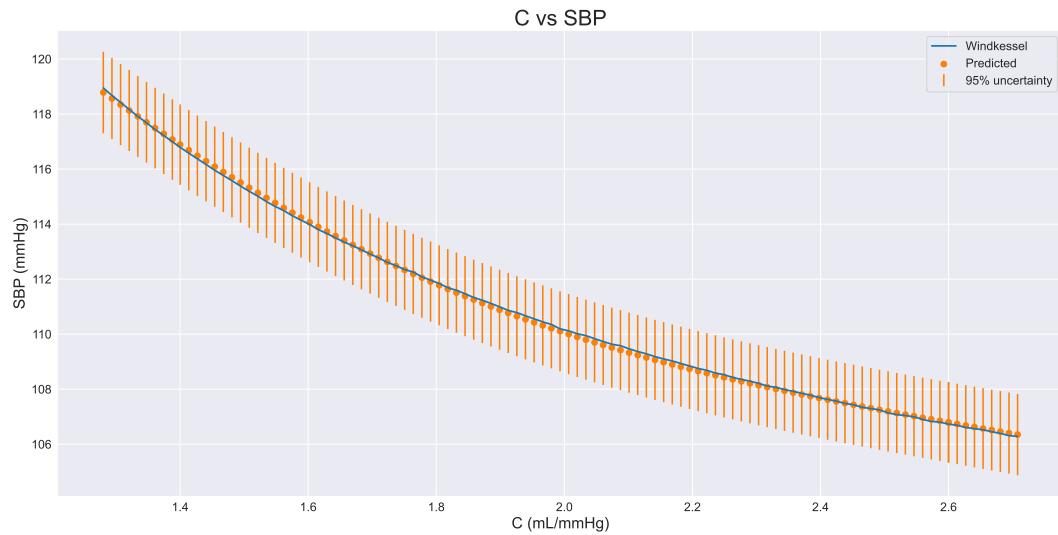


Figure VI.46: Dipendenza di SBP da C sull’intervallo di training e due intervalli attigui.

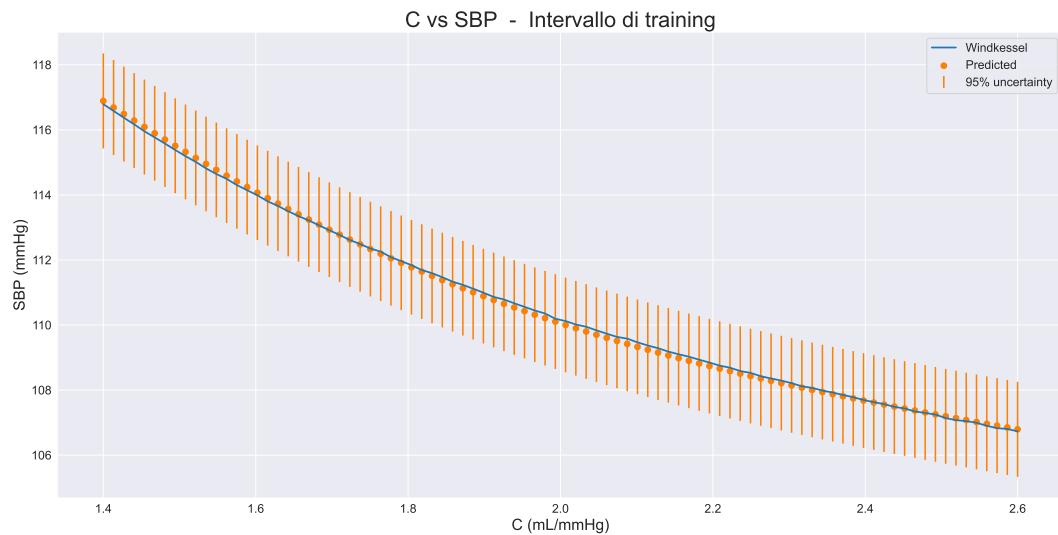


Figure VI.47: Dipendenza di SBP da C sull’intervallo di training.

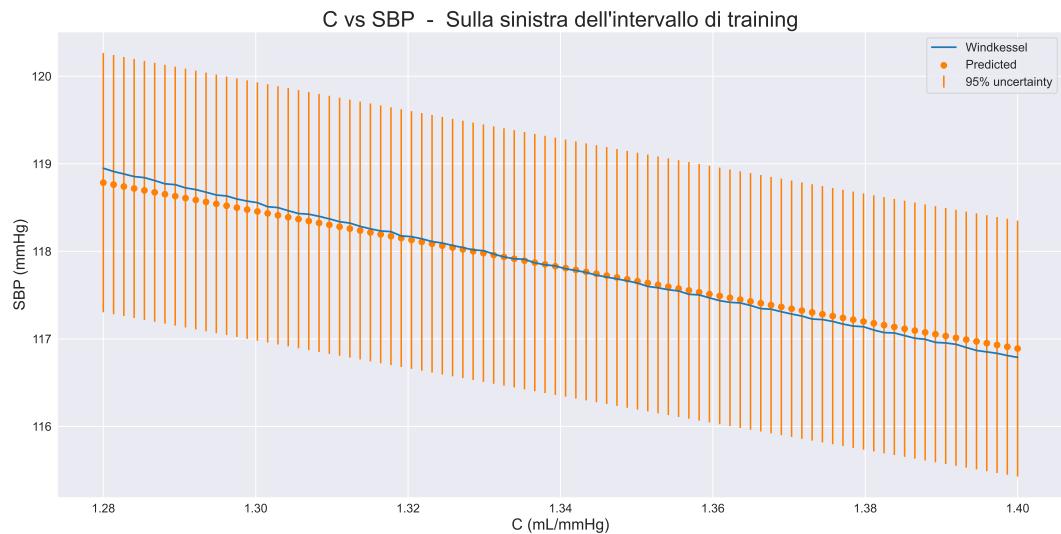


Figure VI.48: Dipendenza di SBP da C sull'intervallo attiguo a sinistra dell'intervallo di training.

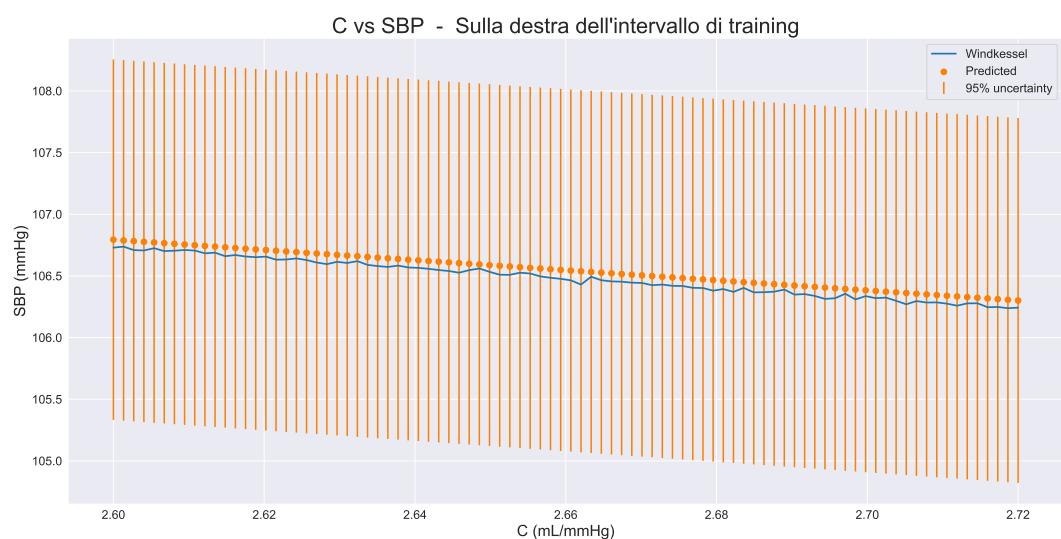


Figure VI.49: Dipendenza di SBP da C sull'intervallo attiguo a destra dell'intervallo di training.

Dipendenza da R_1

Il risultato complessivo è mostrato in figura VI.50, il risultato nel solo intervallo di training in VI.51, il risultato nei singoli intervalli attigui in VI.52 e VI.53.

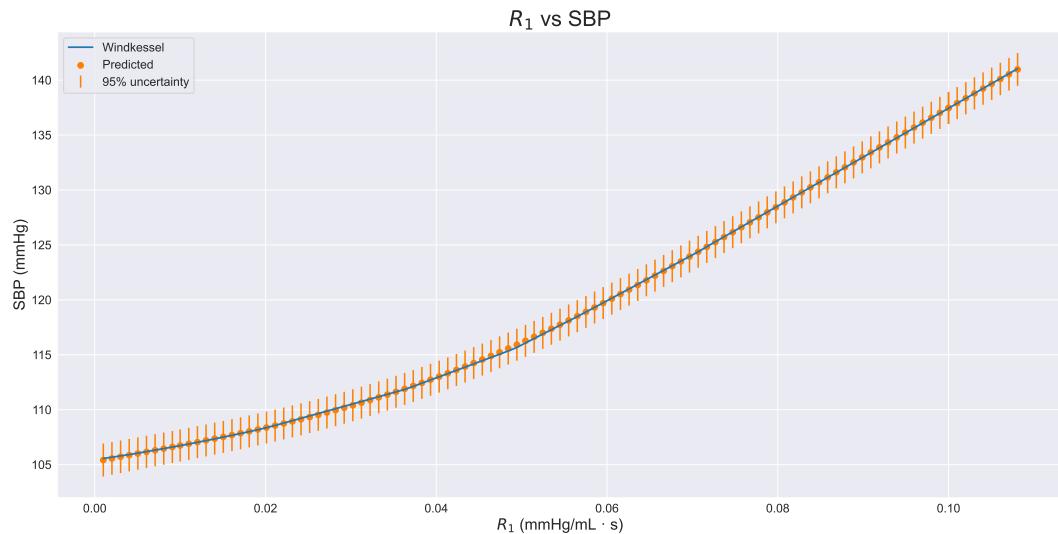


Figure VI.50: Dipendenza di SBP da R_1 sull’intervallo di training e due intervalli attigui.

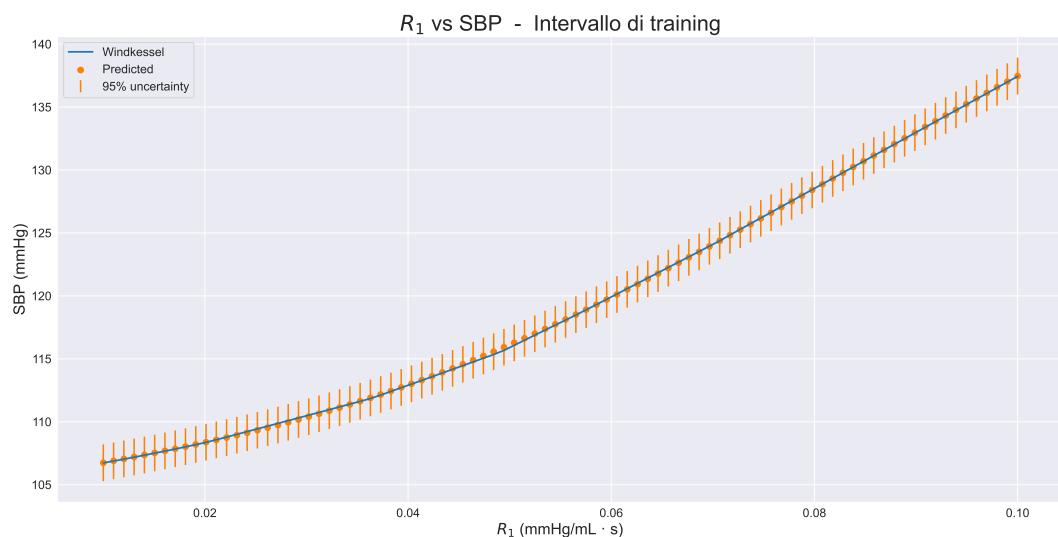


Figure VI.51: Dipendenza di SBP da R_1 sull’intervallo di training.

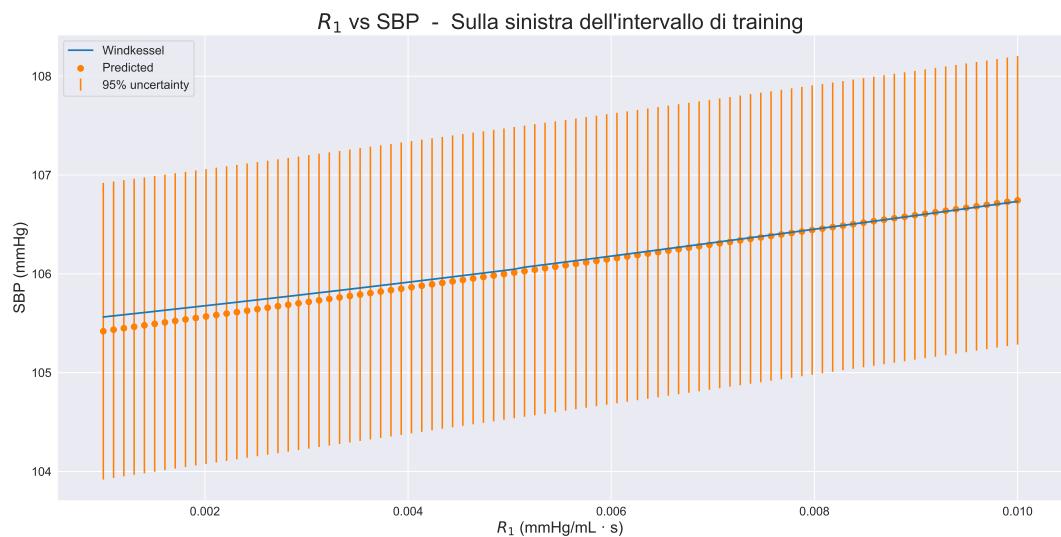


Figure VI.52: Dipendenza di SBP da R_1 sull'intervallo attiguo a sinistra dell'intervallo di training.



Figure VI.53: Dipendenza di SBP da R_1 sull'intervallo attiguo a destra dell'intervallo di training.

Dipendenza da R_2

Il risultato complessivo è mostrato in figura VI.54, il risultato nel solo intervallo di training in VI.55, il risultato nei singoli intervalli attigui in VI.56 e VI.57.

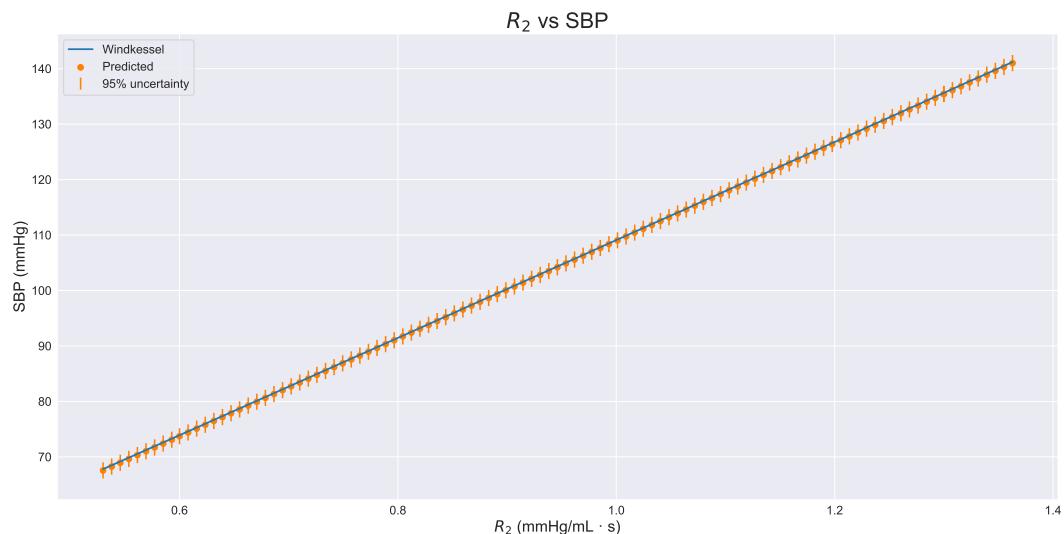


Figure VI.54: Dipendenza di SBP da R_2 sull’intervallo di training e due intervalli attigui.



Figure VI.55: Dipendenza di SBP da R_2 sull’intervallo di training.

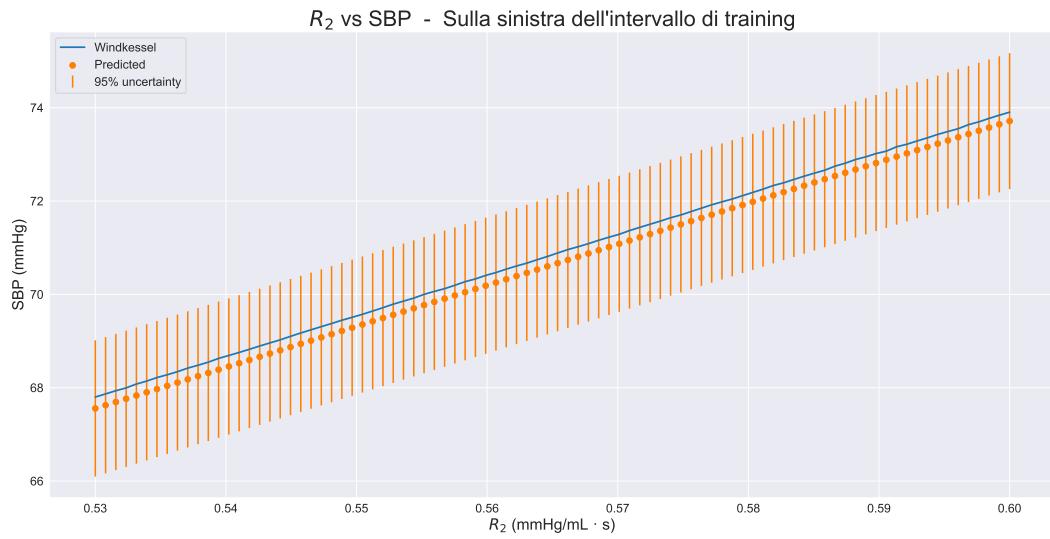


Figure VI.56: Dipendenza di SBP da R_2 sull'intervallo attiguo a sinistra dell'intervallo di training.



Figure VI.57: Dipendenza di SBP da R_2 sull'intervallo attiguo a destra dell'intervallo di training.

VI.4 Tempo di esecuzione: approssimazione di MAP, DBP, SBP, PP

Come fatto in V.4.2, si calcola ora il tempo di esecuzione per trovare il valore di MAP, DBP, SBP, PP a partire dai valori di C, R_1, R_2 . Poichè viene eseguito un training per ogni parametro, vengono riportati i tempi di stima dei parametri per ogni modello addestrato.

Parametro stimato	Tempo esecuzione (media + dev. std.)
DBP	5.89 ms ± 1.63 ms
SBP	5.23 ms ± 1.75 ms
MAP	5.6 ms ± 1.64 ms
PP	4.03 ms ± 1.19 ms

Table VI.1: Tempo di esecuzione per trovare l'approssimazione dei parametri usando i processi gaussiani addestrati.

Poichè per ogni parametro si addestra un modello indipendente, per ottenere i valori di MAP, DBP, SBP e PP è necessario girare i quattro processi gaussiani addestrati. Dunque il tempo di esecuzione necessario per ottenere le approssimazioni dei quattro parametri è la somma dei tempi necessari per ottenerli uno a uno. Dunque si può dire che in tutto sono necessari 20.75 ms in media per ottenere i quattro valori. Comparando questa quantità alla tabella V.2 è evidente che l'uso dei processi gaussiani richiede molto meno tempo rispetto all'approssimazione della soluzione del modello Windkessel già dopo soli dieci cicli cardiaci (un numero piuttosto basso se l'obiettivo è trovare la soluzione a convergenza).

Evidentemente, dunque, i processi gaussiani risolvono il problema del tempo di esecuzione nel modello Windkessel, fornendo una buona approssimazione del valore di MAP, DBP, SBP, PP.

VII

Conclusions and future directions

This chapter summarizes the main results obtained in the course of the work, and proposes *future directions* that could improve what has been studied in both theoretical and practical terms.

A conclusion is the place where you get tired of thinking.

Arthur Bloch

VII.1 Conclusions

It was seen throughout the elaborate what Gaussian processes are and, in particular, explained in what sense they generalize the multivariate Gaussian distribution. The main kernel functions were illustrated, explaining the meaning of the parameters of the functions (called in supervised learning *hyperparameters*). It was then explained how to generalize covariance functions in multiple dimensions, and then apply this generalization to the context of supervised learning.

Then it was explained how to use Gaussian processes for the prediction of observations without noise (or interpolation) and with noise (*noisy* observations), showing how simple in theoretical terms it is to obtain remarkable results.

Despite what has been seen, i.e., the simplicity and power of Gaussian processes, it is worth explaining why they are not widely used in the scientific landscape: only recently, in fact, there are some research groups applying them to hemodynamic contexts, such as [Lon+20] and [Yuh+22]. The main reasons that [RW06] identifies to motivate the low scientific interest are two: the first is that the application of Gaussian processes requires the handling of large matrices and in particular the inversion of them, something that has become computationally addressable only in recent decades; the second is that most of the theoretical study has been done using the same covariance functions, with little awareness about them and thus without exploiting the power of this technology. Research like [Duv14] and books like [RW06] have certainly helped the scientific community in this regard.

The Windkessel model was then introduced in the paper, leaving extensive coverage of the application part. Although the model is rather simple, being formed only by a differential equation, it provides results of remarkable accuracy. Within the same chapter, the local sensitivity of the variables with respect to the parameters was studied, which allowed to understand how little influence the distal pressure P_d has on MAP, DBP, SBP, PP; for this reason, it was discarded from the supervised learning parameters because, leaving it, one risked worsening the performance of the statistical model both in terms of learning time and accuracy.

Finally, the results of supervised learning performed with the GPErks library were reported after explaining its operation from a statistical point of view. These results allow us to conclude that indeed Gaussian processes provide a shortcut to the approximation of MAP, DBP, SBP and PP compared to the use of the Windkessel model. With the right parameters, supervised learning has been seen to provide an acceptable standard deviation, hence error,¹ with fairly short training. Moreover, from the input parameters of the Wind-

¹The term *acceptable* refers to the precision that can be accepted in a clinical setting where measurement errors of physiological parameters can be very high.

kessel model, the trained Gaussian process is able to return the value of MAP, SBP, DBP and PP much faster and with less computational cost than using the Windkessel model, then solving a differential equation to convergence. For these reasons, Gaussian processes have proven to be a tool with high potential and wide uses in applied mathematics.

VII.2 Future directions

The in-depth and systematic study of Gaussian processes, the Bayesian theoretical basis on which supervised learning rests and the Windkessel model make it easy to identify, in retrospect, how the approach used can be improved.

For example, the squared-exponential kernel and a linear mean function were used in the paper. For the purposes of what was studied they worked perfectly, but the choice of kernel function and mean function deserves to be done with proper care. As seen in III.3.5, the choice of composite covariance functions may require a lengthy study starting with the knowledge (which therefore needs to be thorough) of the event to be modeled while obtaining, however, a considerable increase in accuracy in the training phase.

The early stopper should be studied extensively and possibly modified² based on the specific problem to best avoid overfitting and to precisely impose the number of EPOCHS and thus the execution time required at the training phase. In clinical settings, for example, a low execution time at the expense of lower accuracy may be preferable to provide real-time support to the clinician. Indeed, it should be noted that in clinical settings it is unnecessary to require high accuracy from training results because patient measurements may be subject to significant measurement error.

The choice of optimization method must be made based on the problem being addressed. In fact, each method has advantages and disadvantages that must be carefully studied in order to optimize the training. Furthermore, based on the method, it is also necessary to study in depth the parameters that can be set, for example the learning rate, which can modify the speed of training and its performance.

Another improvement that can be made to training is to use real data to verify the model, allowing for more reliable feedback on its functioning. There are several ways to do this, a simple way is to use a validation set of real data, so that during the training phase the model, after each training phase on the training set, validates its functioning on real data. This improvement is not always possible, since sometimes there are no real datasets of the values that interest the training. For example, for the problem addressed in the paper, an adequate database of real data could not be found: the only possibility was to use a database generated by a more complex model than the Windkessel

²Early stoppers do not follow precise rules; there is scientific research that suggests their form and provides examples of their implementation. Optimal would be to build them specifically for each problem, but this can be very time-consuming

model.

One training approach that would significantly decrease the risk of overfitting is K -fold cross validation. This technique consists of dividing the entire dataset into k subsamples of equal size. The training is executed k times where each time one of the k subsamples is used as validation test and the other $k - 1$ as training set³. Then, k results are obtained, which are averaged to generate a single estimate. Changing the validation set and the training set at each training lowers the probability of overfitting.

In a context like the one seen in the paper, the local sensitivity was sufficient to exclude P_d from the parameters to be taken into consideration since it proved to have little influence on the values of MAP, DBP, SBP, PP (low influence also confirmed by the results of training). In much larger situations, with many more input parameters (a real problem can have hundreds and potentially even thousands of parameters), a more in-depth and non-local analysis is that of global sensitivity analysis, which allows us to understand, in a global, what are the parameters that influence the outputs. This can lead to a significant decrease in the number of parameters to consider, speeding up training and increasing performance.

Since independent training is done for each variable MAP, DBP, SBP, PP, starting from the same input parameters, approximations of MAP, DBP, SBP and PP are obtained which are not correlated with each other, in the sense that, probably, there will be $PP \neq SBP - DBP$. Generally this is not a particularly serious problem, remembering, as previously mentioned, that in clinical situations there are large measurement errors. However, simultaneous training of all variables would avoid similar problems, even though it is a rather complicated training technique.

³It is possible to use one of the $k - 1$ subsamples as testing set. In reality there are different forms of k -fold cross validation and they differ in the use of k subsamples; the basic idea is to run multiple tests by partitioning the dataset.

Bibliography

- [AW20] Philip I. (Philip Irving) Aaronson and Jeremy P. T Ward. *The cardiovascular system at a glance*. eng. 5th ed. At a glance series (Oxford, England). Blackwell, 2020. ISBN: 9781405150446.
- [Bot12] Leon Bottou. “Stochastic Gradient Descent Tricks”. In: *Neural Networks, Tricks of the Trade, Reloaded*. Neural Networks, Tricks of the Trade, Reloaded. Vol. 7700. Lecture Notes in Computer Science (LNCS). Springer, Jan. 2012, pp. 430–445. URL: <https://www.microsoft.com/en-us/research/publication/stochastic-gradient-tricks/>.
- [Dam16] Andreas Damianou. *Gaussian process lecture (Jupiter python)*. en. Brown University, 2016. URL: <https://nbviewer.org/github/adamian/adamian.github.io/blob/master/talks/Brown2016.ipynb> (visited on 02/27/2022).
- [DHS11] John Duchi, Elad Hazan, and Yoram Singer. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”. In: *Journal of Machine Learning Research* 12.61 (2011), pp. 2121–2159. URL: <http://jmlr.org/papers/v12/duchi11a.html>.
- [Duv14] David Kristjanson Duvenaud. “Automatic Model Construction with Gaussian Processes”. en. Doctor of Philosophy. Pembroke College: University of Cambridge, June 2014. URL: <https://www.cs.toronto.edu/~duvenaud/thesis.pdf>.
- [Gel+95] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. New York: Chapman and Hall/CRC, June 1995. ISBN: 978-0-429-25841-1. DOI: 10.1201/9780429258411.
- [GKD19] Jochen Görtler, Rebecca Kehlbeck, and Oliver Deussen. “A Visual Exploration of Gaussian Processes”. In: *Distill* (2019). DOI: 10.23915/distill.00017. URL: <https://distill.pub/2019/visual-exploration-gaussian-processes>.
- [GTM22] Beatrice Ghitti, Eleuterio Francisco Toro, and Lucas Omar Müller. *Nonlinear lumped-parameter models for blood flow simulations in networks of vessels*. 2022. URL: <https://www.esaim-m2an.org/component/article?access=doi&doi=10.1051%5C2Fm2an%5C%2F2022052>.

- [Gut09] Allan Gut. *An Intermediate Course in Probability*. en. Seconda edizione. Springer Texts in Statistics. Springer New York, 2009. ISBN: 9781441901620. DOI: <https://doi.org/10.1007/978-1-4419-0162-0>.
- [KB17] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. Tech. rep. arXiv:1412.6980. arXiv:1412.6980 [cs] type: article. arXiv, Jan. 2017. DOI: 10.48550/arXiv.1412.6980. URL: <http://arxiv.org/abs/1412.6980> (visited on 05/18/2022).
- [KW70] George S. Kimeldorf and Grace Wahba. “A Correspondence Between Bayesian Estimation on Stochastic Processes and Smoothing by Splines”. In: *The Annals of Mathematical Statistics* 41 (Apr. 1970), pp. 495–502. DOI: 10.1214/aoms/1177697089. (Visited on 03/10/2022).
- [Lon+20] S. Longobardi, A. Lewalle, S. Coveney, I. Sjaastad, et al. “Predicting left ventricular contractile function via Gaussian process emulation in aortic-banded rats”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 378.2173 (2020), p. 20190334. DOI: 10.1098/rsta.2019.0334. URL: <https://royalsocietypublishing.org/doi/abs/10.1098/rsta.2019.0334>.
- [Mur12] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. en. Ed. by Francis Bach. Adaptive Computation and Machine Learning series. Cambridge, MA, USA: MIT Press, Aug. 2012. ISBN: 978-0-262-01802-9.
- [Mur22] Kevin P. Murphy. *Probabilistic Machine Learning: An Introduction*. en. Ed. by Francis Bach. Adaptive Computation and Machine Learning series. Cambridge, MA, USA: MIT Press, Mar. 2022. ISBN: 978-0-262-04682-4.
- [Pyt22] Pytorch authors. *R2SCORE - PyTorch Ignite*. [Online; accessed 03-May-2022]. 2022. URL: <https://pytorch.org/ignite/generated/ignite.contrib.metrics.regression.R2Score.html#r2score>.
- [Ras04] Carl Edward Rasmussen. *Gaussian Processes in Machine Learning*. en. Ed. by Olivier Bousquet, Ulrike von Luxburg, and Gunnar Ratsch. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, pp. 63–71. ISBN: 978-3-540-28650-9. DOI: 10.1007/978-3-540-28650-9_4.
- [Rud16] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2016. DOI: 10.48550/ARXIV.1609.04747. URL: <https://arxiv.org/abs/1609.04747>.
- [RW06] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. en. Adaptive computation and machine learning. Cambridge, Mass: MIT Press, 2006. ISBN: 978-0-262-18253-9.

- [SSK18] Eric Schulz, Maarten Speekenbrink, and Andreas Krause. “A tutorial on Gaussian process regression: Modelling, exploring, and exploiting functions”. en. In: *Journal of Mathematical Psychology* 85 (Aug. 2018), pp. 1–16. ISSN: 0022-2496. DOI: 10.1016/j.jmp.2018.03.001. URL: <https://www.sciencedirect.com/science/article/pii/S0022249617302158> (visited on 05/18/2022).
- [Tur16] Dr Richard E Turner. *Gaussian Processes: From the Basics to the State-of-the-Art*. en. Imperial College London, 2016. URL: <http://cbl.eng.cam.ac.uk/pub/Public/Turner/News/imperial-gp-tutorial.pdf>.
- [Wan22] Jie Wang. “An Intuitive Tutorial to Gaussian Processes Regression”. In: *arXiv:2009.10862 [cs, stat]* (Apr. 2022). arXiv: 2009.10862. URL: <http://arxiv.org/abs/2009.10862> (visited on 05/04/2022).
- [Wika] Wikipedia contributors. *Compliance (physiology) — Wikipedia, The Free Encyclopedia*. [Online; accessed 09-04-2022]. URL: [https://en.wikipedia.org/wiki/Compliance_\(physiology\)](https://en.wikipedia.org/wiki/Compliance_(physiology)).
- [Wikb] Wikipedia contributors. *Vascular resistance — Wikipedia, The Free Encyclopedia*. [Online; accessed 09-04-2022]. URL: https://en.wikipedia.org/wiki/Vascular_resistance.
- [Wik21a] Wikipedia contributors. *Wiggers diagram — Wikipedia, The Free Encyclopedia*. [Online; accessed 9-April-2022]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Wiggers_diagram&oldid=1029541282.
- [Wik21b] Wikipedia contributors. *Windkessel effect — Wikipedia, The Free Encyclopedia*. [Online; accessed 9-April-2022]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Windkessel_effect&oldid=1038550965.
- [Wik22a] Wikipedia contributors. *Cardiac cycle — Wikipedia, The Free Encyclopedia*. [Online; accessed 9-April-2022]. 2022. URL: https://en.wikipedia.org/w/index.php?title=Cardiac_cycle&oldid=1077898034.
- [Wik22b] Wikipedia contributors. *Distribuzione normale — Wikipedia, The Free Encyclopedia*. [Online; accessed 06-04-2022]. 2022. URL: https://it.wikipedia.org/wiki/Distribuzione_normale.
- [Wik22c] Wikipedia contributors. *Gradient descent*. [Online; accessed 16-May-2022]. 2022. URL: https://en.wikipedia.org/w/index.php?title=Gradient_descent&oldid=1087292976.
- [Wik22d] Wikipedia contributors. *Overfitting — Wikipedia, The Free Encyclopedia*. [Online; accessed 30-April-2022]. 2022. URL: <https://en.wikipedia.org/w/index.php?title=Overfitting&oldid=1076526991>.

- [Wik22e] Wikipedia contributors. *Training, validation, and test data sets*. [Online; accessed 30-April-2022]. 2022. URL: https://en.wikipedia.org/w/index.php?title=Training,_validation,_and_test_data_sets&oldid=1077290985.
- [Wil20] Richard Wilkinson. *An introduction to Gaussian Processes*. en. Lecture. School of Mathematical Sciences University of Nottingham, 2020. URL: https://rich-d-wilkinson.github.io/docs/talks/Wilkinson_GPSS2020.pdf.
- [Wil98] C. K. I. Williams. “Prediction with Gaussian Processes: From Linear Regression to Linear Prediction and Beyond”. en. In: *Learning in Graphical Models*. Ed. by Michael I. Jordan. Dordrecht: Springer Netherlands, 1998, pp. 599–621. ISBN: 978-94-011-5014-9. DOI: 10.1007/978-94-011-5014-9_23. URL: https://doi.org/10.1007/978-94-011-5014-9_23 (visited on 03/22/2022).
- [WLW08] Nicolaas Westerhof, Jan-Willem Lankhaar, and Berend Westerhof. “The arterial Windkessel”. In: *Medical & biological engineering & computing* 47 (July 2008), pp. 131–41. DOI: 10.1007/s11517-008-0359-2.
- [Yuh+22] Changyoung Yuhn, Marie Oshima, Yan Chen, Motoharu Hayakawa, and Shigeki Yamada. “Uncertainty quantification in cerebral circulation simulations focusing on the collateral flow: Surrogate model approach with machine learning”. In: *bioRxiv* (2022). DOI: 10.1101/2022.03.10.483573. URL: <https://www.biorxiv.org/content/early/2022/05/25/2022.03.10.483573>.

Appendice

In questo ultimo capitolo vengono inseriti tutti i codici non precedentemente menzionati per la generazione di immagini e per poter lavorare con il modello Windkessel come proposto nel capitolo V.

Tutti i codici sono in Python, per la loro creazione sono stati usati: Anaconda (versione 2021.11), conda (versione 4.12.0), python (versione 3.9.7).

Non vengono inseriti i codici completi per la creazione del training di processi gaussiani perché sono una debole modifica dei codici del dottor Stefano Longobardi, presenti su GitHub alla pagina "GPErks".

Appendix usually means "*small outgrowth from large intestine*", but in this case it means "*additional information accompanying main text*".
Or are those really the same things? Think carefully before you insult this book.

Pseudonymous Bosch

Processi gaussiani

Import e codici preliminari

Code VII.1: Import necessario per la generazione di immagini dei kernel introdotti nel capitolo III

```

1  # Imports
2  %matplotlib inline
3  %config InlineBackend.figure_format = 'svg'
4  import numpy as np
5  import scipy
6  import matplotlib
7  import matplotlib.pyplot as plt
8  from matplotlib import cm
9  from mpl_toolkits.axes_grid1 import make_axes_locatable
10 import matplotlib.gridspec as gridspec
11 import seaborn as sns
12 from matplotlib.offsetbox import AnchoredText
13
14 # Import kernels
15 from sklearn.gaussian_process.kernels import RBF
16 from sklearn.gaussian_process.kernels import DotProduct
17 from sklearn.gaussian_process.kernels import ExpSineSquared

```

Code VII.2: Import necessario per la generazione di immagini sulla predizione del capitolo III

```

1  import numpy as np
2  from matplotlib import pyplot as plt
3
4  from sklearn.gaussian_process import GaussianProcessRegressor
5  from sklearn.gaussian_process.kernels import RBF,
6      ConstantKernel as C
7  np.random.seed(1)
8
9  # Funzione da predire
10 def f(x):
11     return x * np.sin(x)

```

Code VII.3: Definizione della media cubica

```

1  def cubed_mean(x):
2      tempList = np.zeros(len(x))
3      for i in range(len(x)):
4          tempList[i] = x[i]**3
5      mean = tempList**3
6      return mean

```

Code VII.4: Codice per generare la figura III.3

```
1 def example_mean(x):
2     tempList = np.zeros(len(x))
3     for i in range(len(x)):
4         tempList[i] = x[i]
5     mean = tempList**2/4
6     return mean
7
8 nb_of_samples = 40           # Number of points in each function
9 number_of_functions = 4      # Number of functions to sample
10
11 X = np.expand_dims(np.linspace(-4, 4, nb_of_samples), 1)
12
13 # Kernel: parameters and evaluation
14 l=1
15 sigma=1
16 S = exponentiated_quadratic(X, X, l, sigma)
17 mean = example_mean(X)
18
19 # Draw samples from the prior at our data points
20 ys = np.random.multivariate_normal(mean, cov=S,
21     size=number_of_functions)
22
23 # Plot the sampled functions
24 fig, ax = plt.subplots()
25
26 for i in range(number_of_functions):
27     ax.plot(X, ys[i], linestyle='-', marker='o',
28             markersize=3, label='k(x,0)')
29 ax.set_xlim([-4, 4])
```

Linear kernel

Code VII.5: Codice per generare la figura III.4

```

1  # Modify kernel to adapt to definition given
2  def linear(xa, xb, sb, sv, c):
3      xa = xa - c
4      xb = xb - c
5      kernel = sb+sv*DotProduct(sigma_0=0).__call__(xa,xb)
6      return(kernel)
7
8      fig, ax2 = plt.subplots(1, 1, figsize=(4, 4))
9      xlim = (-1, 1)
10     X = np.expand_dims(np.linspace(*xlim, num=100), 1)
11     uno = np.array([[1]])
12
13     # Kernel: parameters and evaluation
14     sb=1
15     sv=1
16     c=-1
17     S0 = linear(X, uno, sb, sv, c)
18
19     # Make the plots
20     fig.tight_layout()
21     plt.plot(X[:,0], S0[:,0], label=' $k(x, 1)$ ')
22     plt.xlabel("X", fontsize=12)
23     plt.legend(loc=1)
24     plt.xlim(xlim)
25     plt.show()

```

Code VII.6: Codice per generare la figura III.5

```

1  nb_of_samples = 50          # Number of points in each function
2  number_of_functions = 10    # Number of functions to sample
3
4  X = np.expand_dims(np.linspace(-4, 4, nb_of_samples), 1)
5
6  # Kernel: parameters and evaluation
7  sb=0
8  sv=1
9  c=0
10 S = linear(X, X, sb, sv, c)
11
12 # Draw samples from the prior at our data points.
13 # Zero mean
14 ys =
15     np.random.multivariate_normal(mean=np.zeros(nb_of_samples),
16                                     cov=S, size=number_of_functions)
17
18 # Plot the sampled functions
19 fig, ax = plt.subplots()
20 at = AnchoredText("$\sigma_b=0, \sigma_v=1, c=0$",
21                   prop=dict(size=11), frameon=True, loc='upper left')
22 at.patch.set_boxstyle("round,pad=0.,rounding_size=0.2")
23 ax.add_artist(at)
24
25 for i in range(number_of_functions):
26     ax.plot(X, ys[i], linestyle='--', marker='o',
27             markersize=3, label=' $k(x, 0)$ ')
28 ax.set_xlim([-4, 4])

```

Code VII.7: Codice per generare la figura III.6

```

1 nb_of_samples = 50          # Number of points in each function
2 number_of_functions = 10    # Number of functions to sample
3
4 X = np.expand_dims(np.linspace(-4, 4, nb_of_samples), 1)
5
6 # Kernel: parameters and evaluation
7 sv=1
8 sb=0
9 c=0 # c=-3
10 S = linear(X, X, sb, sv, c)
11
12 # Draw samples from the prior at our data points.
13 # Zero mean
14 ys =
15     np.random.multivariate_normal(mean=np.zeros(nb_of_samples),
16                                     cov=S, size=number_of_functions)
17
18 # Plot the sampled functions
19 fig, ax = plt.subplots()
20 at = AnchoredText("$\sigma_b=0, \sigma_v=1, c=0$",
21                   prop=dict(size=11), frameon=True, loc='upper left')
22 at.patch.set_boxstyle("round,pad=0.,rounding_size=0.2")
23 ax.add_artist(at)
24
25 for i in range(number_of_functions):
26     ax.plot(X, ys[i], linestyle='-', marker='o',
27              markersize=3, label='k(x,0)')
28 ax.set_xlim([-4, 4])

```

Code VII.8: Codice per generare la figura III.7

```

1 nb_of_samples = 50          # Number of points in each function
2 number_of_functions = 10    # Number of functions to sample
3
4 X = np.expand_dims(np.linspace(-4, 4, nb_of_samples), 1)
5
6 # Kernel: parameters and evaluation
7 sb=0 #sb=0.5
8 sv=1
9 c=0
10 S = linear(X, X, sb, sv, c)
11
12 # Draw samples from the prior at our data points.
13 # Zero mean
14 ys =
15     np.random.multivariate_normal(mean=np.zeros(nb_of_samples),
16                                     cov=S, size=number_of_functions)
16
17 # Plot the sampled functions
18 fig, ax = plt.subplots()
19 at = AnchoredText("$\sigma_b=0.5, \sigma_v=1, c=0$",
20                   prop=dict(size=11), frameon=True, loc='upper left')
21 at.patch.set_boxstyle("round,pad=0.,rounding_size=0.2")
22 ax.add_artist(at)
23
24 for i in range(number_of_functions):
25     ax.plot(X, ys[i], linestyle='-', marker='o',
26              markersize=3, label='k(x,0)')
27 ax.set_xlabel('$x$', fontsize=13)
28 ax.set_xlim([-4, 4])

```

Code VII.9: Codice per generare la figura III.8

```

1 nb_of_samples = 50      # Number of points in each function
2 number_of_functions = 10 # Number of functions to sample
3
4 X = np.expand_dims(np.linspace(-4, 4, nb_of_samples), 1)
5
6 # Kernel: parameters and evaluation
7 sb=0
8 sv=1 #sv = 10
9 c=0
10 S = linear(X, X, sb, sv, c)
11
12 # Draw samples from the prior at our data points.
13 # Zero mean
14 ys =
15     np.random.multivariate_normal(mean=np.zeros(nb_of_samples),
16                                     cov=S, size=number_of_functions)
17
18 # Plot the sampled functions
19 fig, ax = plt.subplots()
20 at = AnchoredText("$\sigma_b=0, \sigma_v=1, c=0$",
21                  prop=dict(size=11), frameon=True, loc='upper left')
22 at.patch.set_boxstyle("round,pad=0.,rounding_size=0.2")
23 ax.add_artist(at)
24
25 for i in range(number_of_functions):
26     ax.plot(X, ys[i], linestyle='-', marker='o',
27             markersize=3, label='k(x,0)')
28 ax.set_xlim([-4, 4])

```

Code VII.10: Codice per generare la figura III.9

```

1 nb_of_samples = 50      # Number of points in each function
2 number_of_functions = 4 # Number of functions to sample
3
4 X = np.expand_dims(np.linspace(-4, 4, nb_of_samples), 1)
5
6 # Kernel: parameters and evaluation
7 sb=1
8 sv=10
9 c=0
10 S = linear(X, X, sb, sv, c)
11 mean = cubed_mean(X)
12
13 # Draw samples from the prior at our data points.
14 # Cubed mean
15 ys = np.random.multivariate_normal(mean, cov=S,
16                                     size=number_of_functions)
17
18 # Plot the sampled functions
19 fig, ax = plt.subplots()
20 at = AnchoredText("$\sigma_b=1, \sigma_v=10, c=0$",
21                  prop=dict(size=11), frameon=True, loc='upper left')
22 at.patch.set_boxstyle("round,pad=0.,rounding_size=0.2")
23 ax.add_artist(at)
24
25 for i in range(number_of_functions):
26     ax.plot(X, ys[i], linestyle='-', marker='o',
27             markersize=3)
28 ax.plot(X, mean, 'black', linewidth=2.2, label='$x^3$')
29 ax.set_xlim([-4, 4])
30 ax.legend(loc=1)

```

Squared-exponential kernel

Code VII.11: Codice per generare la figura III.10

```

1  # Define squared-exponential kernel
2  # Modify kernel to adapt to definition given
3  def exponentiated_quadratic(xa, xb, l, sigma):
4      kernel = sigma**2 * RBF(length_scale=l).__call__(xa, xb)
5      return(kernel)
6
7  fig, ax2 = plt.subplots(1, 1, figsize=(4, 4))
8  xlim = (-4, 4)
9  X = np.expand_dims(np.linspace(*xlim, num=100), 1)
10 zero = np.array([[0]])
11
12 # Kernel: parameters and evaluation
13 sigma = 1
14 l = 1
15 Y = exponentiated_quadratic(X, zero, l, sigma)
16
17 # Make the plots
18 fig.tight_layout()
19 plt.plot(X[:,0], Y[:,0], label='$k(x,0)$')
20 plt.xlabel("X", fontsize=13)
21 plt.legend(loc=1)
22 plt.xlim(xlim)
23 plt.show()

```

Code VII.12: Codice per generare la figura III.11

```

1 nb_of_samples = 50          # Number of points in each function
2 number_of_functions = 5     # Number of functions to sample
3
4 X = np.expand_dims(np.linspace(-4, 4, nb_of_samples), 1)
5
6 # Kernel: parameters and evaluation
7 l=1
8 sigma=1
9 S = exponentiated_quadratic(X, X, l, sigma)
10
11 # Draw samples from the prior at our data points.
12 # Zero mean
13 ys = np.random.multivariate_normal(
14     mean=np.zeros(nb_of_samples), cov=S,
15     size=number_of_functions)
16
17 # Plot the sampled functions
18 fig, ax = plt.subplots()
19 at = AnchoredText("$l^2=1, \sigma^2=1$", prop=dict(size=11),
20     frameon=True, loc='upper left')
21 at.patch.set_boxstyle("round,pad=0.,rounding_size=0.2")
22 ax.add_artist(at)
23
24 for i in range(number_of_functions):
25     ax.plot(X, ys[i], linestyle='-', marker='o',
26             markersize=3, label='k(x,0)')
27 ax.set_xlim([-4, 4])

```

Code VII.13: Codice per generare la figura III.12

```

1 nb_of_samples = 150      # Number of points in each function
2 number_of_functions = 5    # Number of functions to sample
3
4 X = np.expand_dims(np.linspace(-4, 4, nb_of_samples), 1)
5
6 # Kernel: parameters and evaluation
7 l=1
8 sigma=10 # sigma=0.1
9 S = exponentiated_quadratic(X, X, l, sigma)
10
11 # Draw samples from the prior at our data points.
12 # Zero mean
13 ys = np.random.multivariate_normal(
14     mean=np.zeros(nb_of_samples), cov=S,
15     size=number_of_functions)
16
17 # Plot the sampled functions
18 fig, ax = plt.subplots()
19 at = AnchoredText("$l^2=0.3, \sigma^2=1$",
20     prop=dict(size=11), frameon=True, loc='upper left')
21 at.patch.set_boxstyle("round,pad=0.,rounding_size=0.2")
22 ax.add_artist(at)
23
24 for i in range(number_of_functions):
25     ax.plot(X, ys[i], linestyle='-', marker='o',
26             markersize=3, label='k(x,0)')
27 ax.set_xlim([-4, 4])

```

Code VII.14: Codice per generare la figura III.13

```

1 nb_of_samples = 150      # Number of points in each function
2 number_of_functions = 3    # Number of functions to sample
3
4 X = np.expand_dims(np.linspace(-4, 4, nb_of_samples), 1)
5
6 # Kernel: parameters and evaluation
7 l=0.3 # l=10
8 sigma=1
9 S = exponentiated_quadratic(X, X, l, sigma)
10
11 # Draw samples from the prior at our data points.
12 # Zero mean
13 ys = np.random.multivariate_normal(
14     mean=np.zeros(nb_of_samples), cov=S,
15     size=number_of_functions)
16
17 # Plot the sampled functions
18 fig, ax = plt.subplots()
19 at = AnchoredText("$l^2=0.3, \sigma^2=1$",
20     prop=dict(size=11), frameon=True, loc='upper left')
21 at.patch.set_boxstyle("round,pad=0.,rounding_size=0.2")
22 ax.add_artist(at)
23
24 for i in range(number_of_functions):
25     ax.plot(X, ys[i], linestyle='-', marker='o',
26             markersize=3, label='k(x,0)')
27 ax.set_xlim([-4, 4])

```

Code VII.15: Codice per generare la figura III.14

```
1 nb_of_samples = 100      # Number of points in each function
2 number_of_functions = 5   # Number of functions to sample
3
4 X = np.expand_dims(np.linspace(-4, 4, nb_of_samples), 1)
5
6 # Kernel: parameters and evaluation
7 l=0.3
8 sigma=7
9 S = exponentiated_quadratic(X, X, l, sigma)
10 mean = cubed_mean(X)
11
12 # Draw samples from the prior at our data points
13 # Cubed mean
14 ys = np.random.multivariate_normal(mean, cov=S,
15                                     size=number_of_functions)
16
17 # Plot the sampled functions
18 fig, ax = plt.subplots()
19 at = AnchoredText("$l^2=0.1, \sigma^2=10$",
20                   prop=dict(size=11), frameon=True, loc='upper left')
21 at.patch.set_boxstyle("round,pad=0.,rounding_size=0.2")
22 ax.add_artist(at)
23
24 for i in range(number_of_functions):
25     ax.plot(X, ys[i], linestyle='-', marker='o',
26             markersize=3)
27 ax.plot(X, mean, 'black', linewidth=2.2, label='$x^3$')
28 ax.set_xlim([-4, 4])
29 ax.legend(loc=1)
```

Periodic kernel

Code VII.16: Codice per generare la figura III.15

```

1  # Modify kernel to adapt to definition given
2  def periodic(xa, xb, s, l, p):
3      kernel = s**2 * ExpSineSquared(length_scale=l,
4          periodicity=p).__call__(xa, xb)
5      return kernel
6
7  # Show covariance with X=1
8  fig, ax2 = plt.subplots(1, 1, figsize=(4, 4))
9  xlim = (-4, 4)
10 X = np.expand_dims(np.linspace(*xlim, num=1000), 1)
11 zero = np.zeros((len(X), 1))
12
13 # Kernel: parameters and evaluation
14 s = 1
15 l = 1
16 p = 2
17 S0 = periodic(X, zero, s, l, p)
18
19 # Make the plots
20 fig.tight_layout()
21 plt.plot(X[:, 0], S0[:, 0], label='k(x,0)')
22 plt.xlabel("X", fontsize=13)
23 plt.legend(loc=1)
24 plt.xlim(xlim)
plt.show()

```

Code VII.17: Codice per generare la figura III.16

```

1 nb_of_samples = 200      # Number of points in each function
2 number_of_functions = 4   # Number of functions to sample
3
4 X = np.expand_dims(np.linspace(-2, 2, nb_of_samples), 1)
5
6 # Kernel: parameters and evaluation
7 l=2
8 s=1
9 p=1
10 S = periodic(X, X, s, l, p)
11
12 # Draw samples from the prior at our data points.
13 # Zero mean
14 ys =
15     np.random.multivariate_normal(mean=np.zeros(nb_of_samples),
16                                     cov=S, size=number_of_functions)
17
18 # Plot the sampled functions
19 fig, ax = plt.subplots()
20 at = AnchoredText("$l^2=2, \sigma^2=1, p=1$",
21                  prop=dict(size=11), frameon=True, loc='upper left')
22 at.patch.set_boxstyle("round, pad=0., rounding_size=0.2")
23 ax.add_artist(at)
24
25 for i in range(number_of_functions):
26     ax.plot(X, ys[i], linestyle='--', marker='o',
27             markersize=3, label='k(x,0)')
ax.set_xlim([-2, 2])

```

Code VII.18: Codice per generare la figura III.17

```

1 nb_of_samples = 250      # Number of points in each function
2 number_of_functions = 4   # Number of functions to sample
3
4 X = np.expand_dims(np.linspace(-2, 2, nb_of_samples), 1)
5
6 # Kernel: parameters and evaluation
7 l=1
8 s=0.1 # s=10
9 p=1
10 S = periodic(X, X, s, l, p)
11
12 # Draw samples from the prior at our data points.
13 # Zero mean
14 ys =
15     np.random.multivariate_normal(mean=np.zeros(nb_of_samples),
16                                     cov=S, size=number_of_functions)
17
18 # Plot the sampled functions
19 fig, ax = plt.subplots()
20 at = AnchoredText("$l^2=1, \sigma^2=0.1, p=1$",
21                  prop=dict(size=11), frameon=True, loc='upper left')
22 at.patch.set_boxstyle("round,pad=0.,rounding_size=0.2")
23 ax.add_artist(at)
24
25 for i in range(number_of_functions):
26     ax.plot(X, ys[i], linestyle='-', marker='o',
27              markersize=3, label='k(x,0)')
28 ax.set_xlim([-2, 2])

```

Code VII.19: Codice per generare la figura III.18

```

1 nb_of_samples = 400      # Number of points in each function
2 number_of_functions = 4   # Number of functions to sample
3
4 X = np.expand_dims(np.linspace(-2, 2, nb_of_samples), 1)
5
6 # Kernel: parameters and evaluation
7 l=1
8 s=1
9 p=0.8 # p=10
10 S = periodic(X, X, s, l, p)
11
12 # Draw samples from the prior at our data points.
13 # Zero mean
14 ys =
15     np.random.multivariate_normal(mean=np.zeros(nb_of_samples),
16                                     cov=S, size=number_of_functions)
16
17 # Plot the sampled functions
18 fig, ax = plt.subplots()
19 at = AnchoredText("$l^2=1, \sigma^2=1, p=0.8$",
20                  prop=dict(size=11), frameon=True, loc='upper left')
21 at.patch.set_boxstyle("round,pad=0.,rounding_size=0.2")
22 ax.add_artist(at)
23
24 for i in range(number_of_functions):
25     ax.plot(X, ys[i], linestyle='-', marker='o',
26              markersize=3, label='k(x,0)')
27 ax.set_xlim([-2, 2])

```

Code VII.20: Codice per generare la figura III.19

```

1 nb_of_samples = 300      # Number of points in each function
2 number_of_functions = 4   # Number of functions to sample
3
4 X = np.expand_dims(np.linspace(-2, 2, nb_of_samples), 1)
5
6 # Kernel: parameters and evaluation
7 l=0.8    # l=4
8 s=1
9 p=1
10 S = periodic(X, X, s, l, p)
11
12 # Draw samples from the prior at our data points.
13 # Zero mean
14 ys =
15     np.random.multivariate_normal(mean=np.zeros(nb_of_samples),
16                                     cov=S, size=number_of_functions)
17
18 # Plot the sampled functions
19 fig, ax = plt.subplots()
20 at = AnchoredText("$l^2=0.8, \sigma^2=1, p=1$",
21                   prop=dict(size=11), frameon=True, loc='upper left')
22 at.patch.set_boxstyle("round,pad=0.,rounding_size=0.2")
23 ax.add_artist(at)
24
25 for i in range(number_of_functions):
26     ax.plot(X, ys[i], linestyle='-', marker='o',
27              markersize=3, label='k(x,0)')
28 ax.set_xlim([-2, 2])

```

Code VII.21: Codice per generare la figura III.20

```

1 nb_of_samples = 200      # Number of points in each function
2 number_of_functions = 3   # Number of functions to sample
3
4 X = np.expand_dims(np.linspace(0, 5, nb_of_samples), 1)
5
6 # Kernel: parameters and evaluation
7 l=1
8 s=8
9 p=0.5
10 S = periodic(X, X, s, l, p)
11 mean = cubed_mean(X)
12
13 # Draw samples from the prior at our data points
14 ys = np.random.multivariate_normal(mean, cov=S,
15                                     size=number_of_functions)
16
17 # Plot the sampled functions
18 fig, ax = plt.subplots()
19 at = AnchoredText("$l^2=1, \sigma^2=8, p=0.5$",
20                   prop=dict(size=11), frameon=True, loc='upper left')
21 at.patch.set_boxstyle("round,pad=0.,rounding_size=0.2")
22 ax.add_artist(at)
23
24 for i in range(number_of_functions):
25     ax.plot(X, ys[i], linestyle='-', marker='o',
26              markersize=3)
27 ax.plot(X, mean, 'black', linewidth=2.2, label='$x^3$')
28 ax.set_xlim([0, 5])
29 ax.legend(loc=1)

```

Composizione di kernel

Code VII.22: Codice per generare la figura III.22

```

1  # Modify kernel to adapt to definition given
2  def exponentialQuadratiPlusPeriodic(xa, xb, l, sigma, p):
3      kernel = sigma**2 * RBF(length_scale=l).__call__(xa, xb)
4      kernel += sigma**2 * ExpSineSquared(length_scale=l,
5          periodicity=p).__call__(xa, xb)
6      return(kernel)
7
8  fig, ax2 = plt.subplots(1, 1, figsize=(4, 4))
9  xlim = (-4, 4)
10 X = np.expand_dims(np.linspace(*xlim, num=100), 1)
11 zero = np.array([[0]])
12
13 # Kernel: parameters and evaluation
14 sigma = 1
15 l = 1
16 p = 1
17 Y = exponentialQuadratiPlusPeriodic(X, zero, l, sigma, p)
18
19 # Make the plots
20 fig.tight_layout()
21 plt.plot(X[:, 0], Y[:, 0], label='$k(x, 0)$')
22 plt.legend(loc=1)
23 plt.xlim(xlim)
24 plt.show()

```

Code VII.23: Codice per generare la figura III.23

```

1  nb_of_samples = 200      # Number of points in each function
2  number_of_functions = 3   # Number of functions to sample
3
4  X = np.expand_dims(np.linspace(-4, 4, nb_of_samples), 1)
5
6  # Kernel: parameters and evaluation
7  sigma = 1
8  l = 1.1
9  p = 1.1
10 S = exponentialQuadratiPlusPeriodic(X, X, l, sigma, p)
11
12 # Draw samples from the prior at our data points.
13 # Zero mean
14 ys =
15     np.random.multivariate_normal(mean=np.zeros(nb_of_samples),
16                                     cov=S, size=number_of_functions)
17
18 # Plot the sampled functions
19 fig, ax = plt.subplots()
20 at = AnchoredText("$l^2=1.1, \sigma^2=1, p=1.1$",
21                   prop=dict(size=11), frameon=True, loc='upper left')
22 at.patch.set_boxstyle("round, pad=0., rounding_size=0.2")
23 ax.add_artist(at)
24
25 for i in range(number_of_functions):
26     ax.plot(X, ys[i], linestyle='-', marker='o',
27             markersize=3, label='k(x, 0)')
28 ax.set_xlim([-4, 4])

```

Code VII.24: Codice per generare la figura III.24

```

1  # Modify kernel to adapt to definition given
2  def linearTimesLinear(xa, xb, sb, sv, c):
3      xa = xa - c
4      xb = xb - c
5      kernel = sb+sv*DotProduct(sigma_0=0).__call__(xa,xb)
6      return(kernel**2)
7
8      fig, ax2 = plt.subplots(1, 1, figsize=(4, 4))
9      xlim = (-3, 3)
10     X = np.expand_dims(np.linspace(*xlim, num=100), 1)
11     zero = np.array([[1]])
12
13     # Kernel: parameters and evaluation
14     sb=0
15     sv=1
16     c=0
17     S0 = linearTimesLinear(X, zero, sb, sv, c)
18
19     # Make the plots
20     fig.tight_layout()
21     plt.plot(X[:,0], S0[:,0], label='$k(x,1)$')
22     plt.legend(loc=1)
23     plt.xlim(xlim)
24     plt.show()

```

Code VII.25: Codice per generare la figura III.25

```

1  nb_of_samples = 50          # Number of points in each function
2  number_of_functions = 10    # Number of functions to sample
3
4  X = np.expand_dims(np.linspace(-4, 4, nb_of_samples), 1)
5
6  # Kernel: parameters and evaluation
7  sb=0
8  sv=1
9  c=0
10 S = linearTimesLinear(X, X, sb, sv, c)
11
12 # Draw samples from the prior at our data points.
13 # Zero mean
14 ys =
15     np.random.multivariate_normal(mean=np.zeros(nb_of_samples),
16                                     cov=S, size=number_of_functions)
17
18 # Plot the sampled functions
19 fig, ax = plt.subplots()
20 at = AnchoredText("$\sigma_b=0, \sigma_v=1, c=0$",
21                   prop=dict(size=11), frameon=True, loc='upper left')
22 at.patch.set_boxstyle("round,pad=0.,rounding_size=0.2")
23 ax.add_artist(at)
24
25 for i in range(number_of_functions):
26     ax.plot(X, ys[i], linestyle='-', marker='o',
27             markersize=3, label='k(x,0)')
28 ax.set_xlim([-4, 4])

```

Predizione coi processi gaussiani

Code VII.26: Codice per generare la figura III.30

```

1 import numpy as np
2 from matplotlib import pyplot as plt
3 from sklearn.gaussian_process import GaussianProcessRegressor
4 from sklearn.gaussian_process.kernels import RBF, ConstantKernel as C
5
6 np.random.seed(1)
7
8 def f(x):
9     """The function to predict."""
10    return x * np.sin(x)
11
12 X = np.atleast_2d([1., 3., 5., 6., 7., 8.]).T
13
14 # Observations
15 y = f(X).ravel()
16
17 # Mesh the input space
18 x = np.atleast_2d(np.linspace(0, 10, 1000)).T
19
20 # Instantiate a Gaussian Process model
21 kernel = C(1.0, (1e-3, 1e3)) * RBF(10, (1e-2, 1e2))
22 gp = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=9)
23
24 # Fit to data
25 gp.fit(X, y)
26
27 # Make the prediction
28 y_pred, sigma = gp.predict(x, return_std=True)
29
30 plt.figure(figsize=(10, 5))
31 n_samples=15
32 y_samples = gp.sample_y(x, n_samples)
33 for idx, single_prior in enumerate(y_samples.T):
34     plt.plot(x, single_prior, linestyle=":", alpha=1, linewidth=1.3)
35     plt.plot(x, f(x), color="crimson", linestyle='--', linewidth=2.5,
36               label=u'${x}, \sin(x)$')
37     plt.plot(x, y_pred, 'b-', label=u'Media')
38     plt.plot(X, y, 'r.', markersize=13, label=u'Osservazioni')
39     plt.ylim(-5.5, 8.5)
40     plt.xlim(-0.1, 10.1)
41     plt.legend(loc='upper left')
```

Code VII.27: Codice per generare la figura III.31

```

1 import numpy as np
2 from matplotlib import pyplot as plt
3 from sklearn.gaussian_process import GaussianProcessRegressor
4 from sklearn.gaussian_process.kernels import RBF, ConstantKernel as C
5
6 np.random.seed(1)
7
8 def f(x):
9     """The function to predict."""
10    return x * np.sin(x)
11
12 X = np.atleast_2d([1., 3., 5., 6., 7., 8.]).T
13
14 # Observations
15 y = f(X).ravel()
16
17 # Mesh the input space
18 x = np.atleast_2d(np.linspace(0, 10, 1000)).T
19
20 # Instantiate a Gaussian Process model
21 kernel = C(1.0, (1e-3, 1e3)) * RBF(10, (1e-2, 1e2))
22 gp = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=9)
23
24 # Fit to data
25 gp.fit(X, y)
26
27 # Make the prediction
28 y_pred, sigma = gp.predict(x, return_std=True)
29
30 plt.figure(figsize=(10, 5))
31 n_samples=20
32 y_samples = gp.sample_y(x, n_samples)
33 for idx, single_prior in enumerate(y_samples.T):
34     plt.plot(x, single_prior, linestyle=":", alpha=1, linewidth=1.7)
35
36 plt.plot(x, y_pred, 'b-', label=u'Media')
37 plt.fill(np.concatenate([x, x[::-1]]), np.concatenate([y_pred - 1.9600 *
38     sigma, (y_pred + 1.9600 * sigma)[::-1]]), alpha=.35, fc='b', ec='None',
39     label='95% intervallo confidenza')
40 plt.plot(X, y, 'r.', markersize=13, label=u'Osservazioni')
41 plt.ylim(-6.2, 9.9)
42 plt.xlim(-0.1, 10.1)
43 plt.legend(loc='upper left')

```

Code VII.28: Codice per generare la figura III.32

```

1 import numpy as np
2 from matplotlib import pyplot as plt
3 from sklearn.gaussian_process import GaussianProcessRegressor
4 from sklearn.gaussian_process.kernels import RBF, ConstantKernel as C
5
6 np.random.seed(1)
7
8 def f(x):
9     """The function to predict."""
10    return x * np.sin(x)
11
12 X = np.atleast_2d([1., 3., 5., 6., 7., 8.]).T
13
14 # Observations
15 y = f(X).ravel()
16
17 # Mesh the input space
18 x = np.atleast_2d(np.linspace(0, 10, 1000)).T
19
20 # Instantiate a Gaussian Process model
21 kernel = C(1.0, (1e-3, 1e3)) * RBF(10, (1e-2, 1e2))
22 gp = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=9)
23
24 # Fit to data
25 gp.fit(X, y)
26
27 # Make the prediction
28 y_pred, sigma = gp.predict(x, return_std=True)
29
30 plt.figure(figsize=(10, 5))
31 n_samples=20
32 y_samples = gp.sample_y(x, n_samples)
33 for idx, single_prior in enumerate(y_samples.T):
34     plt.plot(x, single_prior, linestyle=":", alpha=1, linewidth=1.7)
35
36 plt.plot(x, y_pred, 'b-', label=u'Media')
37 plt.fill(np.concatenate([x, x[::-1]]), np.concatenate([y_pred - 1.9600 *
38     sigma, (y_pred + 1.9600 * sigma)[::-1]]), alpha=.35, fc='b', ec='None',
39     label='95% intervallo confidenza')
40 plt.plot(X, y, 'r.', markersize=13, label=u'Osservazioni')
41 plt.ylim(-6.2, 9.9)
42 plt.xlim(-0.1, 10.1)
43 plt.legend(loc='upper left')

```

Code VII.29: Codice per generare la figura III.33

```

1 import numpy as np
2 from matplotlib import pyplot as plt
3 from sklearn.gaussian_process import GaussianProcessRegressor
4 from sklearn.gaussian_process.kernels import RBF, ConstantKernel as C
5
6 np.random.seed(1)
7
8 def f(x):
9     """The function to predict."""
10    return x * np.sin(x)
11
12 X = np.linspace(0.1, 9.9, 20)
13 X = np.atleast_2d(X).T
14
15 # Observations and noise
16 y = f(X).ravel()
17 dy = 0.5 + 1.0 * np.random.random(y.shape)
18 noise = np.random.normal(0, dy)
19 y += noise
20
21 # Instantiate a Gaussian Process model
22 gp = GaussianProcessRegressor(kernel=kernel, alpha=dy ** 2,
23                               n_restarts_optimizer=10)
24
25 # Fit to data
26 gp.fit(X, y)
27
28 # Make the prediction
29 y_pred, sigma = gp.predict(x, return_std=True)
30
31 plt.figure(figsize=(10, 5))
32 n_samples=15
33 y_samples = gp.sample_y(x, n_samples)
34 for idx, single_prior in enumerate(y_samples.T):
35     plt.plot(x, single_prior, linestyle=":", alpha=1, linewidth=1.7)
36
37 plt.errorbar(X.ravel(), y, dy, fmt='r.', markersize=13,
38               label=u'Osservazioni')
39 plt.plot(x, y_pred, 'b-', label=u'Media')
40 plt.fill(np.concatenate([x, x[::-1]]), np.concatenate([y_pred - 1.9600 *
41             sigma, (y_pred + 1.9600 * sigma)[::-1]]), alpha=.25, fc='b', ec='None',
42             label='95% intervallo confidenza')
43 plt.ylim(-6, 10)
44 plt.xlim(-0.1, 10.1)
45 plt.legend(loc='upper left')
46 plt.show()

```

Modello Windkessel

Code VII.30: Codice per importare le librerie necessarie

```

1 import numpy as np
2 import scipy as sp
3 import matplotlib.pyplot as plt
4 import matplotlib.pyplot as plt
5 from scipy.integrate import solve_ivp
6 from scipy.optimize import minimize_scalar

```

Code VII.31: Codice per il plot dei dati reali

```

1 # Pressione e flusso dai file
2 flow = np.genfromtxt('stergioFlow.dat')
3 pressure = np.genfromtxt('stergioPressure.dat')
4
5 # Time grid
6 M = 1000
7 # array tempo
8 time = np.linspace(0.,1.,M)
9
10 # Interpolo i dati nella grid
11 flow = np.interp(time,flow[:,0],flow[:,1])
12 pressure = np.interp(time,pressure[:,0],pressure[:,1])
13
14 # Plot
15 fig,ax=plt.subplots()
16 ln1 = ax.plot(time,flow,'b-',label='$Q_{in}$')
17 ax.set_xlabel("time [s]")
18 ax.set_ylabel("$Q_{in}\,,[mL/s]$")
19 ax2=ax.twinx()
20 ln2 = ax2.plot(time,pressure,'r-',label='P')
21 ax2.set_ylabel("$P\,,[mmHg]$")
22 lns = ln1+ln2
23 labs = [l.get_label() for l in lns]
24 ax.legend(lns, labs, loc=0)

```

Code VII.32: Codice per il calcolo della resistenza periferica totale

```

1 rd = np.average(pressure)/np.average(flow)
2 print("Pressione media - %.3f mmHg" % np.average(pressure))
3 print("Flusso medio - %.3f mL/s" % np.average(flow))
4 print("Resistenza - %.3f mmHg/mL*s" % rd)

```

Code VII.33: Codice per la definizione della ODE

```

1 # Definisco dP/dt
2 def dpdt(t,p,args):
3     c,r2,qFunc,pd = args
4     qin = qFunc(t)
5     return (qin-(p-pd)/r2)/c

```

Code VII.34: Codice per definire la funzione di flusso

```

1 def qFull(t):
2     # Conosco i valori del tempo e del flusso
3     q = np.interp(t,time,flow)
4     return q

```

Code VII.35: Codice per la definizione della funzione f_C

```

1 def funC(c,args):
2     """
3         Objective function for CR Windkessel with compliance
4         C unknown
5         Input arguments:
6             - c: current value for compliance
7             - args: list containing:
8                 1) time array where data is available
9                 2) pressure array where data is available (associated to time)
10                3) peripehral resistance
11                4) flow function, that is called qFunc(t)
12                5) ODE function, that is called dydt(t,y,args)
13         Output argument:
14             - errnorm: the value of objective function (7) for "c"
15     """
16
17     time, pressure, r, qFunc, dydt= args
18
19     argsIVP = [[c,r, qFunc, pd]]
20
21     # ODE
22     fun = dydt
23
24     # Tempo iniziale e finale
25     t_span = [time[0],time[-1]]
26
27     # Condizione iniziale (lista di un elemento)
28     y0=[pressure[0]]
29
30     # Metodo (stringa con il nome)
31     method='RK45'
32
33     # Punti dove valutare la soluzione
34     t_eval = time
35
36     # Tolleranza per il metodo numerico
37     tol = 1e-6
38
39     # solve_ivp
40     sol = solve_ivp(fun=fun, t_span=t_span,y0=y0,method=method,
41     t_eval=t_eval,args=argsIVP,rtol=tol)
42
43     # Vettore addendo di  $f_C$ 
44     errnorm = (sol.y[0,:] - pressure)**2
45
46     #  $f_C$ 
47     errnorm = sum(errnorm)
48     return errnorm

```

Code VII.36: Plot soluzione del modello "semplice"

```

1 pPred = flow*rd
2 plt.plot(time,pressure,label='Data')
3 plt.plot(time,pPred,label='Resistance model')
4 plt.xlabel("time [s]")
5 plt.ylabel("$P\backslash,[mmHg]\$")
6 plt.legend()

```

Code VII.37: Codice per generare la figura V.9.

```

1 # Considero i valori di C da 0.6 a 10 per minimizzare la funzione
2 M = 100
3 C = np.linspace(0.6,10,M)
4
5 function = []
6 for i in range(M):
7     args = [time, pressure, rd, qFull, dpdt]
8     # Valuto la funzione in quel C[i]
9     err = funcC(C[i],args=args)
10    function.append(err)
11
12 # Plot dei valori della f_C
13 plt.plot(C,function,'b--',label='$f\_C$')
14 plt.legend()
15 plt.ylabel("$f(C)\backslash,[mmHg]\$")
16 plt.xlabel("$C\backslash,[mL/mmHg]\$")

```

Code VII.38: Codice per la stima di C che minimizza f_C

```

1 args = [time, pressure, rd, qFull, dpdt]
2
3 # Trovo il valore di C che minimizza f_C
4 bracket = [0.6,10.]
5 # Funzione per minimizzare
6 res = minimize_scalar(funcC,bracket=bracket, args=args)
7
8 # Non funziona sempre
9 if res.success:
10     C = res.x
11     print("C: %.5f" % C)
12 else:
13     print("ATTENTION: did not succeed in finding C!")

```

Code VII.39: Codice per generare la figura V.10

```

1 # Imposto i valori
2 args = [[C, rd, qFull, pd]]
3 fun = dpdt
4 t_span = [time[0],time[-1]]
5 y0 = [p0]
6 method='RK45'
7 t_eval = time
8
9 # Uso solve_ivp per risolvere ODE
10 sol = solve_ivp(fun=fun, t_span=t_span,y0=y0,method=method,
11 t_eval=t_eval,args=args,rtol=tol)
12 pPred = sol.y[0,:]
13
14 err = pressure-pPred
15 print("Error inf-norm is: %.6f" % np.linalg.norm(err,np.inf))
16 print("Error 2-norm is: %.6f" % np.linalg.norm(err,2))
17 plt.plot(time,pressure,'g-',label='Data')
18 plt.plot(time[:10],pPred[:10],'rx',label='Model')
19 plt.xlabel("time [s]")
20 plt.ylabel("$P\backslash,[mmHg]\$")
21 plt.legend()

```

Code VII.40: Codice per la definizione della funzione $f_{C,\alpha}$

```

1 def funCR(x,args):
2 """
3 Objective function for RCR Windkessel with compliance
4 C and alpha unknown. Input arguments:
5 - x: list with current value for compliance and alpha
6 - args: list containing:
7     1) time array where data is available
8     2) pressure array where data is available (associated to time)
9     3) peripehral resistance
10    4) flow function, that is called qFunc(t)
11    5) ODE function, that is called dydt(t,y,args)
12 Output argument:
13 - errnorm: the value of objective function (7) for "c"
14 """
15 c, alpha = x
16 time, pressure, r, qFunc, dydt = args
17
18 # Definisco "r1" e "r2" usando "r" e "alpha"
19 r1= (1-alpha)*r
20 r2 = alpha*r
21
22 # *****Risolvo IVP*****
23 # Condizionale iniziale p(t0)=pressure[0]
24 args = [[c,r2,qFunc, pd]]
25 fun = dydt
26 t_span = [time[0],time[-1]]
27 y0 = [pressure[0]]
28 method='RK45'
29 t_eval = time
30 sol = solve_ivp(fun=fun, t_span=t_span,y0=y0,method=method,
31 t_eval=t_eval,args=args,rtol=tol)
32
33 # Valuto f_C,alpha
34 pin = sol.y[0,:] + r1*flow
35 errnorm = (pin - pressure)**2
36 errnorm = sum(errnorm)
37 return errnorm

```

Code VII.41: Codice per la stima di C e α

```

1 args = [time, pressure, rd, qFull, dpdt]
2 x0 = [1.7,1.]
3
4 # Scelgo il metodo
5 #method='Powell'
6 #method='CG'      NON FUNZIONA
7 #method='BFGS'    NON FUNZIONA
8 method='Nelder-Mead'
9
10 res = minimize(funCR,x0,args=args,method=method)
11 print(res)
12
13 if res.success:
14     C = res.x[0]
15     alpha = res.x[1]
16     print("C: %.5f" % C)
17     print("alpha: %.5f" % alpha)
18 else:
19     print("ATTENTION: did not succeed in finding C!")

```

Code VII.42: Codice per generare la figura V.11.

```

1 # I parametri sono stati aggiornati nei codici precedenti
2 r1= (1-alpha)*rd
3 r2 = alpha*rd
4
5 args = [[C, r2, qFull, pd]]
6 fun = dpdt
7 t_span = [time[0],time[-1]]
8 y0 = [pressure[0]]
9 method='RK45'
10 t_eval = time
11 tol = 1e-6
12
13 # solve_ivp
14 sol = solve_ivp(fun=fun, t_span=t_span,y0=y0,method=method,
15 t_eval=t_eval,args=args,rtol=tol)
pPred = sol.y[0,:] + r1*flow
16
17 # Plot
18 err = pressure-pPred
19 plt.plot(time,pressure,'g-',label='Data')
20 plt.plot(time[::10],pPred[::10],'rx',label='Model')
21 plt.xlabel("time [s]")
22 plt.ylabel("$P\backslash,[mmHg]$")
23 plt.legend()

```

Code VII.43: Codice per il metodo delle differenze finite centrate riadattato al calcolo della sensitività locale.

```

1 def centredFiniteDifference(back, forward, h):
2     """
3         ---- Parameters ----
4         back: lista Ppred con parametro Pi-h
5         forward: lista Ppred con parametro Pi+h
6         h : variazione del parametro Pi
7         ---- Returns ----
8         lista differenze finite centrate: [map', dbp', sbp', pp']
9     """
10
11     # MAP
12     mapb = np.average(back)                      # MAP(Pi-hi)
13     mapf = np.average(forward)                    # MAP(Pi+hi)
14     mapDerivative = (mapf-mapb)/(2*h)
15
16     # DBP
17     dbpb = np.min(back)                         # DBP(Pi-hi)
18     dbpf = np.min(forward)                       # DBP(Pi+hi)
19     dbpDerivative = (dbpf-dpbp)/(2*h)
20
21     # SBP
22     sbpb = np.max(back)                         # SBP(Pi-hi)
23     sbpf = np.max(forward)                       # SBP(Pi+hi)
24     sbpDerivative = (sbpf-spbp)/(2*h)
25
26     # PP
27     ppb = sbpb - dbpb                          # PP(Pi-hi)
28     ppf = sbpf - dbpf                          # PP(Pi+hi)
29     ppDerivative = (ppf-ppb)/(2*h)
30
31     return (mapDerivative, dbpDerivative, sbpDerivative, ppDerivative)

```

Code VII.44: Codice per il calcolo della sensitività di C

```

0 ****+10% ****
cost = 1.1
args = [[C*cost, r2, qFull, pd]]
sol_C_1 = solve_ivp(fun=fun, t_span=t_span, y0=y0, method=method,
t_eval=t_eval, args=args, rtol=tol)
pPred_C_1 = sol_C_1.y[0,:] + r1*flow
5
5 ****-10% ****
cost = 0.9
args = [[C*cost, r2, qFull, pd]]
sol_C_2 = solve_ivp(fun=fun, t_span=t_span, y0=y0, method=method,
t_eval=t_eval, args=args, rtol=tol)
pPred_C_2 = sol_C_2.y[0,:] + r1*flow
10
# Derivate parziali
partialDerivativeC = centredFiniteDifference(pPred_C_1, pPred_C_2,
C*(1-cost))

# Sensitività
15 S_map_C = (C/map) * partialDerivativeC[0]
S_dbp_C = (C/dbp) * partialDerivativeC[1]
S_sbp_C = (C/sbp) * partialDerivativeC[2]
S_pp_C = (C/pp) * partialDerivativeC[3]

```

Code VII.45: Codice per il calcolo della sensitività di R_1

```

0   ****+10% ****
cost = 1.1
args = [[C, r2, qFull, pd]]
sol_R1_1 = solve_ivp(fun=fun, t_span=t_span,y0=y0,method=method,
t_eval=t_eval,args=args,rtol=tol)
pPred_R1_1 = sol.y[0,:] + r1*cost*flow
5
5   ****-10% ****
cost = 0.9
args = [[C, r2, qFull, pd]]
sol_R1_2 = solve_ivp(fun=fun, t_span=t_span,y0=y0,method=method,
t_eval=t_eval,args=args,rtol=tol)
pPred_R1_2 = sol.y[0,:] + r1*cost*flow
10
# Derivate parziali
partialDerivativeR1 = centredFiniteDifference(pPred_R1_1, pPred_R1_2,
r1*(1-cost))

# Sensitività
15 S_map_R1 = (r1/map) * partialDerivativeR1[0]
S_dbp_R1 = (r1/dbp) * partialDerivativeR1[1]
S_sbp_R1 = (r1/sbp) * partialDerivativeR1[2]
S_pp_R1 = (r1/pp) * partialDerivativeR1[3]
```

Code VII.46: Codice per il calcolo della sensitività di R_2

```

0   ****+10% ****
cost = 1.1
args = [[C, r2*cost, qFull, pd]]
sol_R2_1 = solve_ivp(fun=fun, t_span=t_span,y0=y0,method=method,
t_eval=t_eval,args=args,rtol=tol)
pPred_R2_1 = sol_R2_1.y[0,:] + r1*flow
5
5   ****-10% ****
cost = 0.9
args = [[C, r2*cost, qFull, pd]]
sol_R2_2 = solve_ivp(fun=fun, t_span=t_span,y0=y0,method=method,
t_eval=t_eval,args=args,rtol=tol)
pPred_R2_2 = sol_R2_2.y[0,:] + r1*flow
10
# Derivate parziali
partialDerivativeR2 = centredFiniteDifference(pPred_R2_1, pPred_R2_2,
r2*(1-cost))

# Sensitività
15 S_map_R2 = (r2/map) * partialDerivativeR2[0]
S_dbp_R2 = (r2/dbp) * partialDerivativeR2[1]
S_sbp_R2 = (r2/sbp) * partialDerivativeR2[2]
S_pp_R2 = (r2/pp) * partialDerivativeR2[3]
```

Code VII.47: Codice per il calcolo della sensitività di P_d

```

0  ****+10% ****
cost = 1.1
args = [[C, r2, qFull, pd*cost]]
sol_Pd_1 = solve_ivp(fun=fun, t_span=t_span,y0=y0,method=method,
t_eval=t_eval,args=args,rtol=tol)
pPred_Pd_1 = sol_Pd_1.y[0,:] + r1*flow
5
5  ****-10% ****
cost = 0.9
args = [[C, r2, qFull, pd*cost]]
sol_Pd_2 = solve_ivp(fun=fun, t_span=t_span,y0=y0,method=method,
t_eval=t_eval,args=args,rtol=tol)
pPred_Pd_2 = sol_Pd_2.y[0,:] + r1*flow
10
# Derivate parziali
partialDerivativePd = centredFiniteDifference(pPred_Pd_1, pPred_Pd_2,
pd*(1-cost))

# Sensitività
15 S_map_Pd = (pd/map) * partialDerivativePd[0]
S_dbp_Pd = (pd/dbp) * partialDerivativePd[1]
S_sbp_Pd = (pd/sbp) * partialDerivativePd[2]
S_pp_Pd = (pd/pp) * partialDerivativePd[3]
```

Code VII.48: Codice per definire la funzione di flusso per il modello Windkessel ciclico

```

1  def qFull(t):
2      if type(t)==float:
3          tLoc = t-int(t)*1.
4      else:
5          tLoc = t-t.astype(int)*1.
6      q = np.interp(tLoc,time,flow)
7      return q
```