THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Institut Supérieur de l'Aéronautique et de l'Espace (ISAE)*

Présentée et soutenue le *07/12/2017* **par :**

ANKIT CHIPLUNKAR

Incorporating Prior Information from Engineering Design into Gaussian Process Regression: with applications to Aeronautical Engineering

---

JURY

NICOLAS GAYTON	IFMA, Clermont-Ferrand	Rapporteur
ALEXANDER FORRESTER	University of Southampton	Rapporteur
RODOLPHE LE RICHE	CNRS, Saint-Étienne	Examinateur
NATHALIE BARTOLI	ONERA, Toulouse	Examinateur
JOSEPH MORLIER	ISAE-Supaero, Toulouse	Directeur de thèse
EMMANUEL RACHELSON	ISAE-Supaero, Toulouse	Co-directeur de thèse

---

École doctorale et spécialité :
AA : Aéronautique et Astronautique

Unité de Recherche :
Institut Clément Ader

Directeur(s) de Thèse :
Prof. Joseph MORLIER (directeur de thèse), Dr. Emmanuel RACHELSON (co-directeur de thèse) et *M. Michele COLOMBO (superviseur industriel)*

Rapporteurs :
Prof. Nicolas GAYTON et *Prof. Alexander FORRESTER*

Acknowledgements

I would, first of all, extend my gratitude to Dr. Nicolas Gayton and Dr. Alexander Forrester for accepting to review this thesis and Dr. Nathalie Bartoli and Dr. Rodolphe Le Riche for agreeing to be part of the jury.

This research work being a CIFRE thesis gave me the opportunity to work, both in an academic environment (ISAE Toulouse) and in a company (Airbus Operations S.A.S.). On the academic side, I would like to thank my tutors Dr. Joseph Morlier and Dr. Emmanuel Rachelson. They both patiently listened to my wacky ideas and nudged me into more productive directions. They have the capability to understand and explain concepts from their lowest principles, a quality which I admire and would like to replicate.

At Airbus, I am really indebted to my industrial supervisor Michele Colombo, who took part in developing the wacky ideas and attempting to make them work, working with him has been the exact opposite of a stereotypical Ph.D experience (context: http://phdcomics.com/). I would also like to thank Sebastien Blanc for encouraging and helping me integrate into the French-speaking environment. He believed in my capabilities and let me apply algorithms in the intensive flight-test campaign.

I thank the whole EGLRM team at Airbus, my colleagues during the plateau, members of the V&V IPT, and researchers at the ICA lab. I have really enjoyed my conversations with Hitul Dhoru, Simon Trapier, Alex Calder, Elisa Bosco and Sebastian Delord.

As always, I am forever grateful to my family for their love and understanding, my father for his inspiration, my mother for her unconditional love, my wife for being a great teacher and my brother for being my bro.

While all the above people have shaped me in the last 3 years by their presence, I have learned a lot virtually from the blogs of Vitalik Buterin, tweets of Naval Ravikant and books of Yuval Noah Harari, I extend my thanks to them as well.

"Stand on the shoulders of giants"

Google Scholar Motto

Abstract — Due to the high cost of performing experiments on physical systems, numerical models become an efficient means to designing them. Traditionally, these models were built by iteratively performing experiments in a controlled environment. A more cost effective method of building models is by using machine learning algorithms, which infer patterns from data and can be used to perform interpolation and extrapolation. In this thesis, we propose to build better Gaussian Process (GP) regression models by integrating the prior knowledge of Aircraft design with experimental data.

We demonstrate how to incorporate the prior information by mainly changing the covariance functions of the GP. Prior information in terms of smoothness, linearity, differentiability, etc. can be easily encoded using simple covariance functions. Using spectral mixture kernels we demonstrate how to build models for structural dynamic experiments and automatically identify dynamic parameters such as modal frequency. To the best of our knowledge, such a method has not been used in earlier literature to identify the modal parameters. We then use the change-point kernels to identify onset of non-linearity, and use a neural network kernel to interpolate shock in transonic regime. For each application we compare the proposed model with the state-of-the-art model and demonstrate the cost or performance gains achieved.

Similarly, relationships between multiple outputs can be learned or enforced by manipulating the covariance functions. We first demonstrate how the prior information of simulation models can be incorporated to perform extrapolation of aircraft loads using the flight-test data. We then incorporate the physical laws of flight-mechanics into GP regression to identify flight loads. For both use-cases, we first validate the methodology on a synthetic dataset and then demonstrate the gains on a flight-test dataset.

A limitation of GPs is that they scale very poorly with data. There are two main methods of scaling GPs. The first reduces the dataset using a limited set of inducing inputs, while the second distributes the dataset into smaller batches. We demonstrate the scaling capabilities achieved, by building a GP model for interpolating pressures on millions of nodes in a CFD mesh. To the best of our knowledge this scale of GP model has not been created before for interpolating aerodynamic pressures. A similar surrogate model was used in a recent Airbus flight test campaign to compare the pressures predicted from high-fidelity CFD computations to the pressures measured real time on the wing. We then demonstrate how to scale up Multi-Task GPs to large number of data-points, using both approximations. The proposed approach was then validated on a synthetic dataset and on a flight test dataset.

Keywords: Gaussian Process, Flight-Mechanics, Aircraft design, Structural Dynamics, Shock Interpolation.

Résumé — En raison du coût élevé des essais sur les systèmes physiques, les modèles numériques deviennent un moyen préférentiel pour les concevoir. Une méthode plus efficiente de construction de modèles consiste à utiliser des algorithmes d'apprentissage statistique, qui déduisent des modèles à partir de données et peuvent être utilisés pour interpoler et extrapoler. Dans cette thèse, nous proposons de construire de meilleurs modèles de régression de Processus Gaussiens (GPs) en intégrant la connaissance préalable de la conception d'aéronef avec des données expérimentales.

Nous démontrons comment intégrer l'information préalable en modifiant principalement les fonctions de covariance du GP. En utilisant les noyaux de mélange spectral, nous démontrons comment construire des modèles pour des expériences dynamiques structurelles et identifier automatiquement des paramètres dynamiques comme la fréquence modale. Nous utilisons ensuite le noyau de 'change-point' pour identifier l'apparition des non-linéarités et utilisons un noyau de réseau neuronal pour interpoler le choc dans le régime transsonique. Pour chaque application, nous comparons le modèle proposé avec le modèle d'état de l'art et démontrons les gains de coût ou de performance obtenus.

De même, les relations entre plusieurs sorties peuvent être apprises ou imposées en manipulant les fonctions de covariance. Nous démontrons d'abord comment l'information préalable du modèle de simulation peut être incorporée pour effectuer une extrapolation des données expérimentales. Nous incorporons ensuite les lois physiques de la mécanique de vol dans la régression du GP pour identifier les charges de vol. Pour les deux cas d'utilisation, nous validons d'abord la méthodologie sur un ensemble de données synthétiques, puis démontrons les gains sur des données d'essai en vol.

Effectuer une régression GP pour un grand nombre de données devient coûteux très vite, deux méthodes pour étendre des GPs existent. La première 'Sparse GPs' utilise un ensemble des points inductifs pour réduire le coût de calcul de la matrice de précision. La seconde est appelée 'Distributed GPs' elle divise l'ensemble de données en sous-ensembles plus petits, en divisant le modèle en plusieurs lots. Nous démontrons l'avantage de, construit un modèle GP pour interpoler les pressions sur des millions de nœuds dans un maillage CFD. À notre connaissance, tel modèle GP n'a jamais été mis en œuvre pour l'interpolation des pressions. Un modèle de substitution similaire a été utilisé pour une récente campagne d'essais en vol d'Airbus. Nous démontrons ensuite comment étendre les 'Multi-Task GPs' (MTGP) à un grand nombre de sorties, à la fois en utilisant une approximation 'Sparse GPs' et une approximation de 'Distributed GPs'. L'approche proposée a ensuite été validée sur un ensemble de données synthétiques et sur un ensemble de données d'essai en vol.

Mots clés : Processus gaussien, Mécanique de vol, Conception d'aéronefs, Dynamique structurale, Interpolation de choc.

Contents

I Introduction **1**

1 Context **3**

1.1 Introduction 4
1.2 Aircraft design cycle 5
1.3 Machine Learning 7
1.4 Bayesian linear regression 9
1.5 Layout 13
1.6 Contributions 15

2 Gaussian Process Regression **17**

2.1 Introduction 18
2.2 Prior 19
2.2.1 Hyper-parameters 19
2.2.2 Mean function 20
2.2.3 Covariance function 20
2.2.4 Sampling functions from GP priors 21
2.3 Posterior 24
2.3.1 Posterior with Noise-free observations 24
2.3.2 Posterior with Noisy observations 27
2.3.3 Interpretation of posterior 29
2.4 Choosing Hyper-parameters 31
2.5 Summary and discussion 35

3 Scaling up Gaussian Process Regression **37**

3.1 Introduction . . . 38
3.2 Sparse Approximations . . . 38
3.2.1 Nyström Approximation . . . 38
3.2.2 Variational Approximation . . . 41
3.2.3 Experiments . . . 42
3.3 Distributed Gaussian Process . . . 44
3.3.1 Combining experts . . . 46
3.3.2 Experiments . . . 49
3.4 Summary and discussion . . . 49

II Incorporating structure in Gaussian Process Regression 53

4 Basic Covariance Functions 55
4.1 Introduction . . . 56
4.2 Properties . . . 57
4.3 Non-stationary kernels . . . 58
4.3.1 Linear Kernel . . . 58
4.3.2 Neural Network Kernel . . . 59
4.3.3 Constant and Noise kernel . . . 61
4.4 Stationary kernels . . . 62
4.4.1 Squared Exponential Kernel . . . 63
4.4.2 Matérn Kernel . . . 63
4.4.3 Experiments . . . 66
4.4.4 Spectral Mixture Kernels . . . 69
4.5 Application: Identifying Structural Dynamics Parameters . . . 70
4.5.1 Results on a toy data-set . . . 73
4.5.2 Results on a HCT building data-set . . . 75
4.6 Summary and discussion . . . 75

5 Combining Basic Covariance Functions 79
5.1 Introduction . . . 80

5.2 One dimensional inputs . . . 81
5.2.1 Multiplying Kernels . . . 81
5.2.2 Adding Kernels . . . 81
5.2.3 Change-Point kernels . . . 83
5.2.4 Application: Identifying onset of non-linearity using CP kernel . . . 84
5.3 Multi-dimensional kernels . . . 86
5.3.1 Adding across dimensions . . . 86
5.3.2 Multiplying across dimensions . . . 87
5.3.3 Sensitivity analysis . . . 87
5.3.4 Low dimensional structure . . . 89
5.3.5 Application: Interpolation of aerodynamic pressures . . . 90
5.4 Summary and discussion . . . 98

III Incorporating multiple outputs in Gaussian Process Regression 101

6 Multi-task learning for extrapolation 103
6.1 Introduction . . . 104
6.2 Multi-task Gaussian Process . . . 105
6.2.1 An extra dimension . . . 106
6.2.2 Simple Multi-task kernel . . . 107
6.2.3 Linear Model of Coregionalization . . . 109
6.2.4 Multi-fidelity GP . . . 110
6.2.5 Posterior distribution . . . 112
6.3 Experiments . . . 113
6.4 Extrapolation of Flight Loads . . . 116
6.4.1 Current Approach . . . 117
6.4.2 Proposed Approach . . . 119
6.4.3 Proposed method . . . 119
6.4.4 Experiments on toy data-set . . . 121
6.5 Summary and discussion . . . 123

7 Adding relationships in GP Regression **125**
7.1 Introduction . . . 126
7.2 Encoding relationships in GP . . . 126
7.2.1 Case of Linear Operators . . . 127
7.2.2 Case of non-linear operators . . . 131
7.2.3 Calculating posterior . . . 133
7.3 Experiments . . . 134
7.3.1 Quadratic relation on Synthetic Data . . . 135
7.3.2 Flight Mechanics on Flight Test Data . . . 137
7.4 Scaling up MTGP . . . 139
7.4.1 Variational Approximation on Multi-output GP . . . 139
7.4.2 Distributed Inference on Multi-output GP . . . 141
7.5 Experiments . . . 142
7.5.1 Experiments on Theoretical Data . . . 142
7.5.2 Experiments on Flight Test Data . . . 145
7.6 Summary and discussion . . . 148

IV Conclusions **149**

8 Conclusions and perspectives **151**
8.1 Perspectives . . . 153

A Gaussian Identities **157**
A.1 Joint distribution . . . 158
A.2 Sum of Gaussians . . . 158
A.3 Affine property of Gaussians . . . 158
A.4 Marginal Distribution . . . 158
A.5 Conditional Distribution . . . 159

B Proper Orthogonal Decomposition for pressure snapshots **161**
B.1 Pressure Snapshot . . . 161
B.2 POD for aerodynamic snapshots . . . 162

B.3 Interpolation 163

C Gaussian Process Regression enforcing Non-linear operators **165**

Bibliography **169**

List of Figures

1.1 Phases of an aircraft design cycle 5

1.2 Induction vs Deduction 6

1.3 Bias vs Variance trade-off 8

1.4 Prior: Contours of probability density for a prior on parameters w as defined by equation 1.7 11

1.5 Posterior and Prediction in Bayesian linear regression 12

2.1 Covariance matrix for a SE kernel with different hyper-parameters at the input points $\boldsymbol{X}^* = \{[0 : 0.02 : 1]\}$. The SE kernel of figure 2.1(a) has a lower length-scale than figure 2.1(b). Notice how the covariance values are more spread out for figure 2.1(b). 23

2.2 The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior. We can observe that figure 2.2(a) varies faster when compared to figure 2.2(b) due to smaller length scale hyper-parameter. 25

2.3 Prediction in the case of noiseless observations. The solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent three functions drawn at random from a GP posterior. We can observe that Bayes Theorem eliminates all the functions that do not pass through the observed data-set. 28

2.4 Prediction in the case of noisy observations. The solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent three functions drawn at random from a GP posterior. The mean and the draws do not pass exactly from the observation points. 30

2.5 Posteriors for 2 different sets of hyper-parameters. Solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from mean. 32

2.6 Maximizing marginal likelihood 35

3.1 Approximate Gram matrix for a SE kernel using Nyström approximation. . 40

3.2 Results of Nyström Approximation on a toy data-set of size $N = 1000$ . . 44

3.3 Approximate Gram matrix for a SE kernel using mixture of experts. 46

3.4 Results of Distributed GPs approximation on a toy data-set of size $N = 1000$ 50

4.1 Prior and posterior from a GP prior of linear kernel. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior. 60

4.2 Maximizing Marginal Likelihood Linear kernel 60

4.3 Draws from Neural Network kernels having different hyper-parameters. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior. 61

4.4 Covariance functions and Power spectrums for SE kernel with three different length-scales 64

4.5 Covariance functions and Power spectrums for three different kernels . . . 65

4.6 Prior distribution and five random draws from 3 different covariance functions. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior. . . 66

4.7 Posterior distribution and three random draws from 3 different covariance functions. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior. . 67

4.8 Posterior distributions from three different covariance functions after maximizing the hyper-parameters. 68

4.9 Different types of measurements for estimation of Modal parameters in OMA 71

4.10 Results of Spectral Mixture kernels on a toy data-set 74

4.11 Results of Spectral Mixture kernels on real data from HTC building 76

5.1 Random draws from combining a Linear and SE kernel. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior. Random functions drawn from a linear GP are linear. 82

5.2 Random draws by having a change-point between a Linear and SE kernel. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior. Random functions drawn from a linear GP are linear. 84

5.3 Estimation of linear regimes using a change-point kernels 86

5.4 Random draw from a 2 dimensional prior. 88

5.5 Random draw from a 2 dimensional prior which encodes a **low-dimensional** structure. The hyper-parameters of the kernel are $\theta_{amplitude} = 1$ and $\theta_{lengthscale} = 1$ 89

5.6 Details of the FTF 92

5.7 Results for elsA interpolation 93

5.8 CRM shape for aerodynamic simulations 95

5.9 Results for CRM interpolation 96

5.10 Comparison of pressure interpolations for first cut $y/b = 0.105$. Here we compare the accuracy of prediction for interpolation and extrapolation cases. 97

5.11 Performance of Distributed GPs across cuts 98

5.12 Pressure reconstructions for constant $Mach = 0.845$ and sweeping $\alpha \in [1, 3]$ 99

6.1 The outputs can also be represented as an extra dimension. 106

6.2 Joint draws from a simple multi-task kernel function between outputs y^1 and y^2. The matrix $\boldsymbol{K_{output}} = [4, 0.2; 0.2, 1]$ while the covariance function between inputs $k_{input}(x_1, x_2)$ is a SE kernel with hyper-parameters ($\theta = [1, 0.2]$). The solid black line defines the mean function, shaded region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior. We can observe that figure 6.3 varies faster when compared to figure 6.2 due to smaller length scale hyper-parameter in one of the latent functions. 108

6.3 Joint draws from a 'Linear Model of Coregionalization' kernel function between outputs y^1 and y^2. We use two latent functions for figure; the kernels between output dimensions are $k^1_{output} = [4, 0.2; 0.2, 1]$ and $k^2_{output} = [1, 0.2; 0.2, 1]$, while the covariance function between inputs are SE kernel with hyper-parameters ($\theta = [1, 0.2]$) for the first latent process and ($\theta = [1, 0.1]$) for the second latent process. The solid black line defines the mean function, shaded region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior. We can observe that figure 6.3 varies faster when compared to figure 6.2 due to smaller length scale hyper-parameter in one of the latent functions. 110

6.4 Joint Posterior distribution for two outputs with missing data at $x^2 = [-1 : -0.75]$. The solid black line defines the mean function, shaded region defines 95% confidence interval (2σ) distance away from the mean. 115

6.5 10-fold RMSE box-plots for increasing number of input points. The box-plots in red are cases when outputs are assumed independent, the box-plots in green are cases when covariance is assumed to be multi-task, and the box-plots in blue are cases when the output covariance is assumed to be multi-fidelity. The error improves with increasing data-points,the multi-fidelity covariance is best fir for all the three cases. 116

6.6 Flight-loads identification 117

6.7 Current Flight-Loads theoretical model 118

6.8 Example of difference between simulated Coefficient of lift (f^s) and experimental Coefficient of lift (y^e) (coming from equation 6.36), with respect to angle of attack. 122

6.9 Boxplots of extrapolation for the four different cases; experimental set with no error (equations 6.33), with a white noise error (equations 6.35), with a sinusoidal error (equations 6.35) and with translation (equations 6.36). . . 122

7.1 Gram matrices for the joint kernel, which encodes the relation between two outputs y^1 and y^2 130

7.2 Joint draws defined by priors on related-outputs. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior. 131

7.3 Multi-Output Gaussian Process Regression Predictions. The solid black line defines the mean function, shaded region defines 95% confidence interval (2σ) distance away from the mean. Dotted lines represent the functions randomly drawn from the posterior distributions. 134

7.4 GP Regression on Quadratic relationship. The solid black line represents the predicted mean while the shaded area denotes 2σ uncertainty region. The dashed black line represents the real value of f^1 or f^2. For y^1 the data is hidden from section $x = [-0.2, 0.2]$. 136

7.5 Joint multi-output GPR on K_z and C_z relationship. The solid black line represents the predicted mean while the shaded area denotes 2σ uncertainty region. The figure demonstrates the impact of the covariance function for relationship enabled regression. 138

7.6 Approximate Gram matrix for a Joint MTGP kernel using Nyström approximation 140

7.7 Approximate Gram matrix for a SE kernel using mixture of experts. 141

7.8 Experimental results for differential relationship while using Variational approximation. Predicted mean is represented by solid black line. 2σ confidence band is represented by light red for f^1 and light blue for f^2. The dashed black line represents the true latent function values 144

7.9 Comparison of run time and RMSE between Distributed GPs and Variational Inference 145

7.10 Wing Load Diagram 146

7.11 Experimental results for aircraft flight loads 147

List of Matlab Codes

2.1 A zero mean function 20
2.2 A SE covariance function 21
2.3 Plotting the Gram Matrix 22
2.4 Sampling a random function from the prior 23
2.5 Calculating and plotting the mean, the variance and a sample of the posterior 26
2.6 Matlab code for a noisy SE covariance function 28
2.7 Code for data-set D2 32
2.8 Optimizing the Log Marginal Likelihood 34
3.1 Gram Matrix using Nyström Approximation 39
3.2 Code for toy data-set 3 42
3.3 Randomly clustering points into experts 45
3.4 log Marginal Likelihood for Distributed GPs 48
6.1 Code for data-set D4 114
7.1 Create joint covariance functions using Matlab Symbolic Toolbox 129
7.2 Create joint covariance functions for non-linear operators using Matlab Symbolic Toolbox 132

Nomenclature

$GP[m(x), k(x, x')]$ Gaussian Process

$\Pr[a \mid b]$ Conditional probability distribution of a given b

$\Pr[a, b]$ Joint probability distribution of a and b

$\Pr[a]$ Probability distribution of a

$\boldsymbol{a}$ Vector $\boldsymbol{a}$, written in bold lowercase

$\boldsymbol{a}^T$ Transpose of vector $\boldsymbol{a}$

$\mathcal{N}[\boldsymbol{\mu}, \boldsymbol{\Sigma}]$ Multivariate Gaussian Distribution

$\mathcal{N}[\mu, \sigma^2]$ Scalar Gaussian distribution

$\mathcal{R}$ Set of all real numbers

$\boldsymbol{A}$ Matrix $\boldsymbol{A}$, written in bold uppercase

a Scalar a, written in lowercase

A_i i^{th} mode shape

$Cov[x, x']$ Covariance between two random variables

D_{inputs} Number of dimensions in input data

$D_{outputs}$ Number of dimensions in the output data

M Number of inducing points

M_x Bending moment on the wing

N Number of points in training dataset

N_* Number of points in test dataset

$N_{experiments}$ Number of experiments

$N_{experts}$ Number of experts in distributed GP

N_{hyp} Number of hyper-parameters

N_{joint} Total number of points for MTGP

N_j Number of points in the j^{th} output for MTGP

N_{nodes} Number of nodes in aerodynamic mesh

P Number of points in an expert

Q Number of Gaussians in a SM kernel

T_z Shear Load on the wing

Ω Full spatial coordinates of the aerodynamic mesh

$\boldsymbol{\theta}$ Vector of hyper-parameters

$\boldsymbol{y_{joint}}$ Joint output dataset for MTGP

$\boldsymbol{y}$ Full training output dataset

$\boldsymbol{y}^j$ Full set of outputs for the j^{th} output for MTGP

δ Dirac-delta function

ϵ Independent white noise

η_{edge} Wing span at the edge of the wing

η_{root} Wing span at the root of the wing

λ_i i^{th} modal frequency

$\mathbf{E}[x]$ Expectation of the random variable x

$\mathcal{D}_i$ i^{th} Data-set

μ Mean value of a Gaussian Distribution

$\boldsymbol{C}$ Damping matrix

$\boldsymbol{H}$ Low-rank matrix

$\boldsymbol{I_N}$ Identity matrix of size N

$\boldsymbol{K(X_1, X_2)}$ Gram matrix between $\boldsymbol{X_1}$ and $\boldsymbol{X_2}$

$\boldsymbol{K_{FITC}}$ Gram matrix using FITC approximation

$\boldsymbol{K_{Nystrom}}$ Gram matrix using Nyström approximation

$\boldsymbol{K_{lower}}$ Lower Cholesky decomposition of Gram matrix

$\boldsymbol{K_{outputs}}$ Covariance matrix between output dimensions

$\boldsymbol{K_{stiffness}}$ Stiffness matrix

$\boldsymbol{M}$ Mass matrix

$\boldsymbol{P}$ Full Pressure data for the aerodynamic mesh

$\boldsymbol{X}^{\boldsymbol{M}}$ Set of inducing points

$\boldsymbol{X}^{\boldsymbol{j}}$ Full set of inputs for the j^{th} output for MTGP

$\boldsymbol{X_{joint}}$ Joint input dataset for MTGP

$\boldsymbol{X}$ Full training input dataset

$\boldsymbol{\Sigma_{Prior}}$ Covariance matrix of prior over parameters

$\boldsymbol{x_*}$ Full test input dataset

ν Degrees of freedom of student distribution

ω_i i^{th} spatial coordinate of the aerodynamic node

$\phi(x)$ Basis functions

$\sigma^{(i)}(x)$ Predicted covariance of the i^{th} expert

σ_n^2 Variance of the white noise

τ Time lag between two points

$\theta_{amplitude}$ Amplitude hyper-parameter

$\theta_{changeLocation}$ Change location hyper-parameter

$\theta_{intensity}$ Intensity hyper-parameter for k_{CP}

$\theta_{lengthScale}$ Length scale hyper-parameter

b Wing span

d Distance between two input points

$diag(\boldsymbol{K})$ Diagonal of the matrix

f Function mapping transformation from x to y

$f_{pressure}$ Pressure function mapping transformation from ω_i to p_i

$k(x, x')$ Covariance function a GP

k^{δ} Covariance function of the difference between multi-fidelity GPs

$m(x)$ Mean function a GP

$m^{(i)}(x)$ Predicted mean of the i^{th} expert

p_i Pressure at the i^{th} aerodynamic node

w Parameters of functions

x_* Single test input point

x_i i^{th} observational input point

y_i i^{th} observational output point

$ANOVA$ Analysis Of Variance

ARD Automatic Relevance Determination

BCM Bayesian Committee Machine

BIC Bayesian Information Criterion

CFD Computational Fluid Dynamics

CP Change-Point

CRM Common Research Model

CV Cross Validation

EMA Experimental Modal Analysis

FEM Finite Element Method

$FITC$ Fully Independent Training Conditional

FTF Flap Track Fairing

GP Gaussian Process

HCT Heritage Court Tower

IT Information Technology

LOO Leave One Out

MDO Multi Disciplinary Optimization

$MDOF$ Multi Degree Of Freedom

$NExT$ Natural Excitation Technique

OMA Operational Modal Analysis

PCA Principal Component Analysis

POD Proper Orthogonal Decomposition

POE Product Of Experts

PSD Positive Semi Definite

$RANS$ Reynolds Averaged Navier-Stokes

RFP Rational Fractional Polynomial

$RMSE$ Root Mean Square Error

ROM Reduced Order Models

SE Standard Exponential

SVD Singular Value Decomposition

$gPOE$ generalized Product Of Experts

$rBCM$ robust Bayesian Committee Machine

Part I

Introduction

Chapter 1

Context

Résumé

En raison du coût important des expériences réalisées sur les systèmes physiques, concevoir ces derniers purement par l'expérimentation devient un exercice coûteux. Par conséquent les modèles numériques deviennent des moyens importants dans le processus de conception des systèmes physiques. Traditionnellement, ces modèles ont été construits en effectuant des expériences dans un environnement contrôlé, itérativement. Une méthode plus rentable pour construire ces modèles est d'utiliser des algorithmes d'apprentissage automatique, qui déduisent ces modèles de données (expérimentales ou simulées) et peuvent être utilisés pour interpoler et extrapoler. Dans cette thèse, nous développons des modèles d'apprentissage automatique en combinant la connaissance acquise sur le design (conception) d'avion avec les données expérimentales.

Les modèles d'apprentissage automatique constituent un compromis entre le biais et la variance, formalisé par Wolpert dans son célèbre théorème "No free lunch" [Wolpert 1997]. Les fonctions constitutives dans un espace d'hypothèse représentent le biais (préjugé) ou les hypothèses de l'algorithme apprenant (eg. des fonctions linéaires pour la régression linéaire). En l'absence de biais suffisants, la famille de fonctions dans l'espace de recherche devient très grande, et mène à une variance élevée dans le modèle. Au contraire, un biais trop important signifie que la vraie fonction de transformation n'existe pas dans l'espace d'hypothèse. Dans ce cas, l'algorithme d'apprentissage trouve la fonction la plus proche de f dans l'espace d'hypothèse, ce qui mène à un sous-ajustement.

Une méthode pour surmonter ce compromis est l'utilisation d'un grand nombre de données. Étant donné la facilité d'accès à un nombre de plus en plus grand de données, nous pouvons progressivement réduire le biais dans l'apprentissage de modèles. C'est

aussi le concept principal de 'Apprentissage profond', où plusieurs couches de réseaux de neurones définissent un vaste espace d'hypothèse [Goodfellow 2016, LeCun 2015].

Nous proposons de construire de meilleurs modèles d'apprentissage de Processus Gaussiens (GPs) en intégrant les connaissances préalablement testées sur les systèmes des avions avec les données expérimentales. Un modèle généré par la fusion des deux méthodologies sera à la fois cohérent avec la physique du système et sera plus rapide à évaluer. Nous intégrons trois types de connaissances antérieures en répondant aux questions suivantes :

1. **Motifs**: Comment ajouter des informations *a-priori* d'un motifs dans un algorithme d'apprentissage ? (part II).
2. **Données de simulation**: Comment fusionner l'information *a-priori* des simulations avec des expériences ? (chapter 6).
3. **Relations**: Comment ajouter des informations *a-priori* des relations entre les mesures ? (chapter 7).
4. **Le passage à l'échelle**: Comment appliquer la régression GP à de grands volumes de données ? (chapter 3 et section 7.4).

1.1 Introduction

Machine learning

In the past decade due to the boom in Information Technology (IT) companies, more investment has gone into developing both computational infrastructure and methods. One of the ubiquitous methods developed due to this investment is that of Machine Learning. Learning algorithms look for patterns in data, learn from them, and make decisions. They are used in web search, optimization, spam filtering, ad placement, stock trading, healthcare, manufacturing, space exploration, particle physics, security, and lot more. The speed and adaptability of learning methods are changing everything around us one algorithm at a time [Domingos 2015]. The World Economic Forum, Davos 2016 [Schwab 2016] has dubbed this as the fourth industrial revolution; first was steam powered, second was electrically powered, third was IT powered, fourth will be powered by artificial intelligence algorithms.

Aircraft design

In comparison aircraft design is almost a century old field, the first successful flight being by the Wright brothers in 1903 [Wright 1934]. Currently, the design of an aircraft is a highly-iterative optimization process. Based on the initial target objectives and general trends, aircraft designers define the objectives for domain specific departments (eg. aerodynamics design or structural design). These objectives further trickle down to more dedicated teams and tasks (eg airfoil design or fuselage design). This gives rise to

a huge Multi-Disciplinary Optimization (MDO) problem. Teams come up with individual constraints, they iterate around their domains, interact with neighboring domains and negotiate overlapping constraints. Teams negotiate and solidify individual objectives and constraints to find the optimized design for an aircraft, while taking into account the sparse infrastructure, human and economic limitations. This is a massive MDO problem spanning almost a decade, costing billions and involving thousands of people.

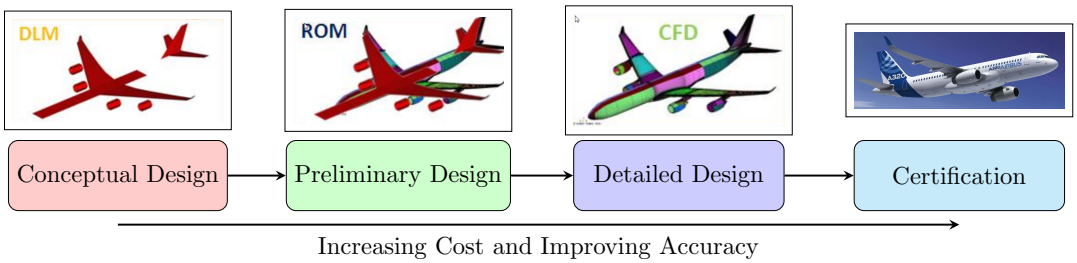


Figure 1.1: Phases of an aircraft design cycle

1.2 Aircraft design cycle

Design phases

To simplify this process an aircraft design is broken down into several design phases (figure 1.1). Each phase requires an ever increasing amount of predictions and fidelity of the target objectives (aircraft weight or lift to drag ratio). Preliminary design phase requires a few low-fidelity design trade-offs between major disciplines, whereas detailed design phase, requires intensive intra-disciplinary and inter-disciplinary optimization. Finally, during the flight-test and certification phase, capability to predict target objectives in real-time can provide significant gains. These analyses cover large parts of the flight envelope and require high-fidelity predictions. Hence the capability to accurately and quickly predict the target objectives is an integral part of an aircraft design cycle [Raymer 2012].

Need for Speed

In the last decade, high-fidelity physics-based, mathematical simulations have become central to designing an aircraft. However, high-fidelity simulations are computationally expensive, this is the case for several Computational Fluid Dynamics (CFD) and Finite Element Method (FEM) based solvers. Due to this high cost, high-fidelity simulations are launched only for a few carefully chosen design configurations. This results in inefficient exploration of the design space and thus a non-optimal design. A common strategy to speed up simulations is by reducing the physical complexity of the model to make quick predictions. As an example, linear potential flows (simpler aerodynamic model), or coarser FEM meshes (simpler structural model) are regularly used during the preliminary design phase [Cummings 2015]. While this is an acceptable practice during the preliminary design

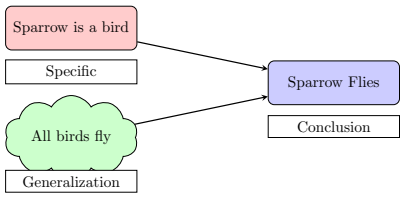
phase, during the detailed design phase, physical complexity is needed to find a robust optimum design point [Raymer 2012].

Need for accuracy

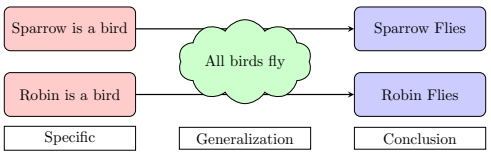
Instead of approximating physical complexity, surrogate models[1] simplify mathematical complexity [Verveld 2016]. Surrogate models learn patterns between the input and output data-set and then are used to make predictions on the desired point. This property is very useful in quickly exploring the design space and finding a robust design point [Forrester 2008]. Moreover, surrogate models are commonly passed across disciplines to perform inter-disciplinary optimizations. For example, a loads department would prefer running a quick surrogate model over the costly CFD model while performing the load's loop [Gazaix 2017, Lefebvre 2017].

Deduction vs Induction

The main difference between the engineering design and surrogate modeling can be explained by the difference between deduction and induction [Domingos 2012] (figure 1.2). Engineering design is deduction: where a very general formula is applied to a particular case (figure 1.2(a)). The basics of Newtonian physics, when applied to a particular aircraft geometry, give inertial loads. The basics of aerodynamics, when applied to a particular set of aircraft geometry and aircraft states, give out aerodynamic pressures. Engineering design takes global rules and applies them to local configurations. Whereas, surrogate modeling is induction: it looks at local features and data, tries to find similarity measures between them and gives a global formula for the process (figure 1.2(b)). For example, an algorithm to detect faces in images will look at several images with and without faces, learn a facial pattern and make predictions on new images [Marszalek 2007]. We here see a possible complementary relationship between engineering design and machine learning; where engineering design needs models to generate data, machine learning needs data to generate models.



(a) Engineering design - Deduction: The figure shows an application of a general rule to a particular case.



(b) Surrogate modelling - Induction: The figure shows how multiple examples can be used to infer underlying rules or patterns that govern the system.

Figure 1.2: Induction vs Deduction

Model building in aircraft engineering is traditionally outsourced to research institutes. Researchers perform iterative experiments in a controlled environment and discover pat-

[1]Surrogate models, learning algorithms and machine learning models will be used interchangeably throughout this manuscript

terns between the physics of the system. This is a rigorous and time-consuming method of developing models. For example, it took 200 years to iteratively develop the gas law [2], Boyle's law in 1600's found the relation between Pressure and Volume, Charle's Law in 1787 discovered the relationship between Volume and Temperature, while Gay-Lussac's Law in 1809 discovered the relationship between Pressure and Temperature [Clapeyron 1834]. Machine learning is a much more cost effective method of building models. Using data and few basic assumptions, automatic models can be built between desired inputs and outputs. For example, while the first model of a neural network was proposed in 1950's [McCulloch 1943, Rosenblatt 1958], neural networks are today used daily for tasks such as tagging cat photos on Facebook and converting speech to text[3]. In this thesis, we wish to automatically build models for aircraft design tasks primarily to be used during the detailed design phase and certification phase.

Developing faster models

1.3 Machine Learning

The core objective of supervised learning algorithms is to find a transformation function between the inputs and outputs. There are three main components in a supervised learning algorithm:

Components of learning

1. **Representation**: A learning algorithm starts with a family of functions. For example, a linear model is a family of linear functions, a trigonometric model defines a family of trigonometric functions. If an algorithm is not able to represent the actual function in its family of functions, it will find the closest function in its hypothesis space[4].

2. **Evaluation**: Some measure is needed to distinguish a good function from a bad function in the chosen hypothesis space. This measure is termed as evaluation; one example is the least squares error commonly used in many learning algorithms.

3. **Optimization**: Finally, the algorithm iteratively searches in its hypothesis space to find the best possible function explaining the data. The choice of optimizer defines the speed of learning and is also important if there are multiple minima in the evaluation criteria.

Surrogate models suffer from the bias vs variance trade-off (figure 1.3), formalized by 'Wolpert' in his famous "no free lunch theorem" [Wolpert 1997]. The constituent functions in a hypothesis space represent the bias or assumptions of the learning algorithm (eg.

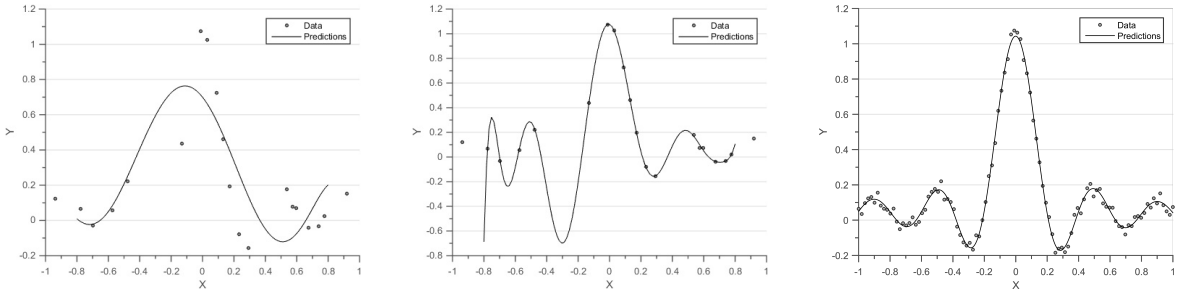
Bias vs Variance

[2] $Pressure \times Volume \propto numberOfMoles \times Temperature$

[3] I wrote a fourth of this thesis using a text to speech software

[4] The term family of functions, hypothesis space and representation will be used interchangeably throughout this manuscript

linear functions for linear regression). In the absence of sufficient assumptions, the family of functions in the search space becomes very large which leads to high variance or over-fitting in the surrogate model (figure 1.3(b)). On the contrary, stringent assumptions mean that the true transformation function (f) does not exist in the hypothesis space. In this case, the learning algorithm finds the function closest to f in its hypothesis space and leads to under-fitting (figure 1.3(a)).



(a) High-bias and low variance (b) Low bias and High variance (c) Bias replaced with lots of data

Figure 1.3: Bias vs Variance trade-off

Soft and hard constraints

One method to overcome this trade-off is by using lots and lots of data. Data acts as a hard constraint for learning algorithm, imagine a family of all possible continuous functions ($y = f(x)$) in the range of $x \in [-1, 1]$ and $y \in [-1, 1]$. What will happen, given an observation ($x = 0, y = 0$). All the functions that do not pass through this point will be eliminated, the new data-point has basically reduced the possible family of functions. Whereas, bias acts as soft constraint, we can use the bias of linearity or smoothness to reduce our hypothesis space. Therefore, both bias and data help in reducing the hypothesis space[5]. Given access to more and more data, we can progressively reduce the biases used while model building thereby relying more on true evidence. This is also the main concept behind deep learning, where several layers of neural networks define a very large hypothesis space [Goodfellow 2016, LeCun 2015].

Unfortunately in aircraft design, generating a huge amount of accurate data is a costly exercise, for example a high fidelity CFD simulation runs for weeks [Murthy 2014, Jameson 2012, Forrester 2008] and a flight-test campaign costs millions of euros [Fox 2004]. On another hand due to centuries of research and tinkering, we have a treasure trove of prior information about these physical systems. We propose to build better machine learning models by integrating the time-tested prior knowledge of physical systems with experimental data.

To integrate the prior information we propose to use the Bayesian inference for model

[5]Bias can also be looked upon as distilled knowledge or patterns gained after interpreting huge amounts data

building. Bayesian inference is a method of statistical inference in which the Bayes theorem is used to update an initial probability (prior) using evidence (data-set) to give the final probability (posterior). More specifically, we will use the Gaussian Processes (GP) Regression or Kriging framework which is a subset of the Bayesian Inference algorithms to define prior information of physical systems. This choice is driven by the fact that GPs have been shown to perform better when data is scarce or costly [Stein 1999], and GPs provide a functional method to define the hypothesis space. But, before deep diving into the details of GP let us have a look at a simple Bayesian linear regression algorithm.

1.4 Bayesian linear regression

Suppose we have access to a training set of observations (or outputs) $\boldsymbol{y} = (y(\boldsymbol{x_1}), \ldots, y(\boldsymbol{x_i}), \ldots, y(\boldsymbol{x_N}))^T$, evaluated at a set of known inputs $\boldsymbol{X} = (\boldsymbol{x_1}, \ldots, \boldsymbol{x_i}, \ldots, \boldsymbol{x_N})^T$, and we wish to predict $\boldsymbol{y(x_*)}$ at a test input $\boldsymbol{x_*}$. The input and output can be multi-dimensional; $\boldsymbol{x_i} \in \mathbb{R}^{D_{inputs}}$ and $\boldsymbol{y(x_i)} \in \mathbb{R}^{D_{outputs}}$. The process of learning the transformation function 'f' to make a prediction at a new point is called regression. In the following section we follow the formulation for Bayesian linear regression provided by [MacKay 2003].

Basis functions

A simple method to perform regression is by assuming a functional form of f (representation), then minimizing the error (evaluation) between the observed outputs (y) and the predicted outputs $f(x)$ with respect to the parameters of f (optimization). The function is written in terms of basis functions $\phi(x)$. For example when $\phi(x) = \{1, \boldsymbol{x}\}^T$ we are performing linear regression, when $\phi(x) = \{1, \boldsymbol{x}, \boldsymbol{x^2}, \ldots, \boldsymbol{x^L}\}^T$ we are performing L^{th} order polynomial regression. We will focus on linear regression in this section and hence $\phi(x) = \{1, \boldsymbol{x}\}^T$.

$$f(\boldsymbol{x}) = \begin{Bmatrix} 1 & \boldsymbol{x} \end{Bmatrix} \begin{Bmatrix} w_0 \\ \boldsymbol{w_1} \end{Bmatrix} \qquad y(\boldsymbol{x_i}) = f(\boldsymbol{x_i}) + \epsilon \tag{1.1}$$

Likelihood

Here, $\boldsymbol{w}$ are the parameters of the function, such that $w_0 \in \mathbb{R}$ denotes the intercept and $\boldsymbol{w_1} \in \mathbb{R}^{D_{inputs}}$ denotes the slope of the regression model. The measurements are corrupted by independent white noise ϵ, such that the noise is a random variable sampled from a white noise Gaussian[6] with variance σ_n^2. The above equations 1.1 can be combined to result

[6] $Probability density : \Pr[\epsilon] = \mathcal{N}(0, \sigma_n) = \frac{1}{\sqrt{2\pi\sigma_n^2}} \exp^{-\frac{\epsilon^2}{2\sigma_n^2}}$

in the likelihood $\Pr[\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{w}]$

$$\begin{aligned}\Pr[\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{w}] &= \prod \Pr[y_i \mid \boldsymbol{x_i}, \boldsymbol{w}] \\ &= \prod \mathcal{N}[[1, \boldsymbol{x_i}]\boldsymbol{w}, \sigma_n^2] \\ &= \mathcal{N}(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{w}, \sigma_n^2 \boldsymbol{I_N})\end{aligned} \tag{1.2}$$

The notation $\Pr[\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{w}]$ symbolizes the probability distribution of the observations $\boldsymbol{y}$ at the inputs $\boldsymbol{X}$ given the parameter $\boldsymbol{w}$. The notation $\mathcal{N}[\boldsymbol{\mu}, \boldsymbol{\Sigma}]$ symbolizes a multi-variate Gaussian distribution for mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. $\boldsymbol{I_N}$ is an identity matrix of size $N \times N$.

Prior

While performing Bayesian inference we specify a prior distribution to encode our assumptions on the parameters before we look at the observations. For this case, we put a zero mean Gaussian prior on our weights.

$$\Pr[\boldsymbol{w}] = \mathcal{N}(0, \boldsymbol{\Sigma_{Prior}}) \tag{1.3}$$

The prior distribution on $\boldsymbol{w}$ induces a prior distribution over functions parametrized by $\boldsymbol{w}$, effectively we are defining the family of functions $(\Pr[f(\boldsymbol{x})] = \mathcal{N}(0, \boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{\Sigma_{Prior}}\boldsymbol{x}))$ by placing a prior distribution over $\boldsymbol{w}$. Once we have a prior distribution encoding our biases, we use the Bayes rule to look at the observations and get a posterior distribution of parameters.

$$posterior = \frac{likelihood \times prior}{marginal \quad likelihood}$$

$$\Pr[\boldsymbol{w} \mid \boldsymbol{y}, \boldsymbol{X}] = \frac{\Pr[\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{w}] \times \Pr[\boldsymbol{w}]}{\Pr[\boldsymbol{y} \mid \boldsymbol{X}]} \tag{1.4}$$

Marginal Likelihood

The *marginal likelihood* $(\Pr[\boldsymbol{y} \mid \boldsymbol{X}])$ is a normalization constant, for more details please refer to section 2.4. After, using the equation 1.2, 1.3 and 1.4 we can get the posterior distribution of weights as:

$$\Pr[\boldsymbol{w} \mid \boldsymbol{y}, \boldsymbol{X}] = \mathcal{N}\left(\frac{1}{\sigma_n^2}\boldsymbol{A}^{-1}\boldsymbol{X}\boldsymbol{y}, \boldsymbol{A}^{-1}\right) \tag{1.5}$$

Here, $\boldsymbol{A} = \sigma_n^{-2}\boldsymbol{X}\boldsymbol{X}^{\boldsymbol{T}} + \boldsymbol{\Sigma}_{\boldsymbol{Prior}}^{-\boldsymbol{2}}$. Thus the posterior distribution for function f at test point $\boldsymbol{x_*}$ becomes:

$$\Pr[f \mid \boldsymbol{x}_*, \boldsymbol{X}, \boldsymbol{y}] = \mathcal{N}\left(\frac{1}{\sigma_n^2}\boldsymbol{x}_*\boldsymbol{A}^{-1}\boldsymbol{X}\boldsymbol{y}, \boldsymbol{x}_*^{\boldsymbol{T}}\boldsymbol{A}^{-1}\boldsymbol{x}_*\right) \tag{1.6}$$

The mean $\frac{1}{\sigma_n^2}\boldsymbol{x_*}\boldsymbol{A^{-1}}\boldsymbol{X}\boldsymbol{y}$ can be used as a prediction at the test point $\boldsymbol{x_*}$, while the variance is a measure of uncertainty for this prediction. We can thus obtain the prediction $f(\boldsymbol{x_*})$, using a prior set of beliefs (equation 1.3) and updating those beliefs using observations.

Posterior

Suppose we have a toy data-set $\mathcal{D}_1 = \{\boldsymbol{X} = [-0.5, 0.33, 0.66], \boldsymbol{y} = [0, 0.5, 0.5]\}$ and want to find a function that fits this data-set using Bayesian linear regression. If we assume a prior distribution of parameter $\boldsymbol{w}$ as defined by equation 1.7, then the prior probability density of $\boldsymbol{w}$ will look like figure 1.4.

Experiment

$$\Pr\left[\begin{pmatrix} w_0 \\ w_1 \end{pmatrix}\right] = \mathcal{N}\left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \boldsymbol{\Sigma_{Prior}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right] \tag{1.7}$$

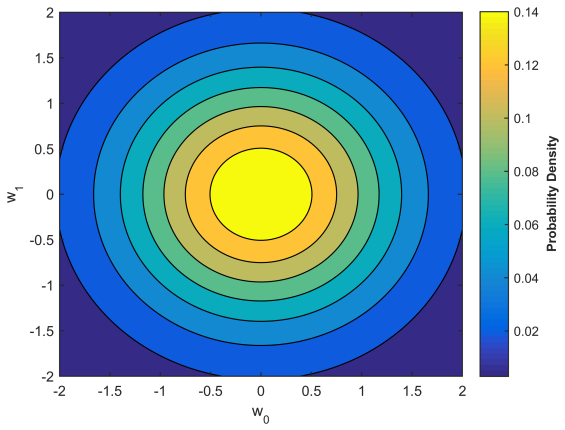
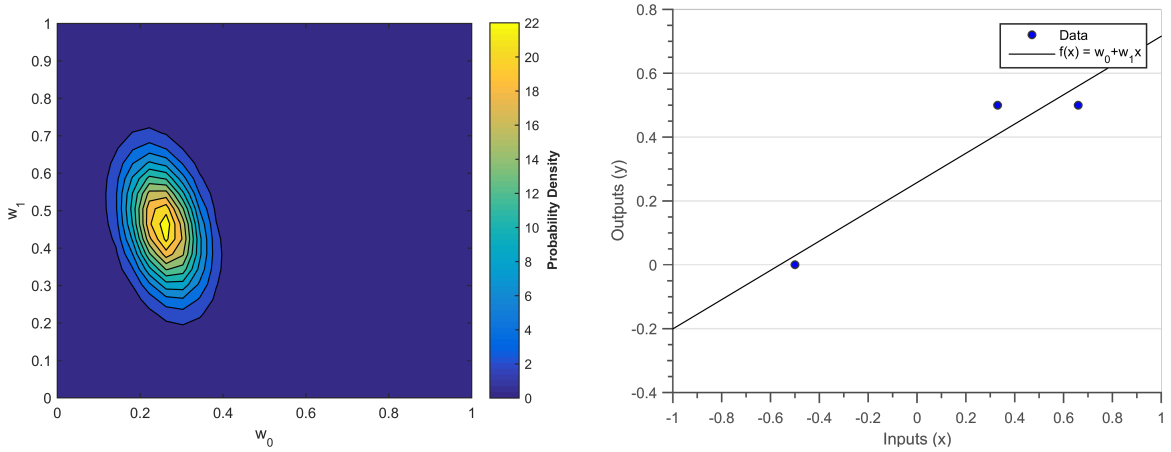


Figure 1.4: Prior: Contours of probability density for a prior on parameters w as defined by equation 1.7

Using equations 1.7 and 1.6 we can calculate the posterior distribution for the parameters. For a noise model of $\Pr[\epsilon] = \mathcal{N}(0, \sigma_n^2 = 0.1^2)$ the posterior distribution comes out as the figure 1.5(a)

$$\Pr\left[\begin{pmatrix} w_0 \\ w_1 \end{pmatrix} \mid \mathcal{D}_1, \Sigma_P, \sigma_n\right] = \mathcal{N}\left[\begin{pmatrix} 0.2576 \\ 0.4584 \end{pmatrix}, \begin{bmatrix} 0.0037 & -0.0022 \\ -0.0022 & 0.0138 \end{bmatrix}\right] \tag{1.8}$$

This simple example demonstrates how to estimate the parameters w_0 and w_1 using the prior distribution on parameters w and data-set $\mathcal{D}_1$. The estimate of the intercept is $w_0 = 0.2576$ and slope is $w_1 = 0.4584$ (equation 1.8), we can use this estimate to plot the value of linear function $f(x)$ (figure 1.5(b)).



(a) Contours of probability density for posterior distribution of parameters w using equations 1.7 and 1.5. The predicted intercept is 0.2576 and the predicted slope is 0.4584

(b) Linear Prediction based on the posterior, given the data-set $\mathcal{D}_1$ prior $\Sigma_P$and noise σ_n. The predicted linear line does not pass through the data-points but between them.

Figure 1.5: Posterior and Prediction in Bayesian linear regression

While the Bayesian linear regression framework provides an opportunity to encode prior assumptions in terms of distributions of the parameters. A much more elegant and expressive method is by using GPs to express prior assumptions. GPs are a distribution over functions and hence enable us to encode prior knowledge directly in the functional space.

Non-parametric models

Learning algorithms are mainly divided into two main types. The first is defined by parametric models which can only represent a limited hypothesis space. They use parameters to describe the function between input and output domain. For example the weight parameters w in Bayesian linear regression. The second are non-parametric models whose hypothesis space grows with the size of data [Ghahramani 2013]. Non-parametric models use data to represent functions, GP Regression is a type of non-parametric model. One can imagine a non-parametric model like a stretched rubber sheet: whenever it sees data it deforms accordingly to compensate for the new data-point. Hence the more data it sees the more it starts mimicking the actual function.

Gaussian Process

Kriging was first used in the context of Geo-statistics research by Daniel Krige [Krige 1951], this was later formalized by Matheron in his seminal work "Principals of Geo-statistics" [Matheron 1963]. It has since been a popular method in engineering design to create surrogate models of costly computer simulations. Interest in GPs grew in the machine learning community from neural networks research. It was shown that a Bayesian neural network becomes a GP as the number of neurons tends to infinity [Neal 2012]. GPs are probabilistic distributions over functions, which provide a Bayesian non-parametric approach to smoothing and interpolation. Regression performed through GPs are gener-

alizations of the Kriging algorithm.

Informally, a GP is a method to probabilistically define a family of functions. A GP can be completely parametrized by its mean and covariance function. The mean function defines the trend-line of the family of functions, while the covariance defines the spatial dependency of input-points. Throughout this thesis we will manipulate the covariance function to enforce prior information into a GP Regression framework.

1.5 Layout

This thesis is divided into three main parts, each part is then divided into individual chapters and their constituent sections. This part sets up the prerequisites required to understand the concepts introduced in the next two parts. The first chapter demonstrates the need for performing regression in aircraft design tasks and describes a very basic Bayesian linear regression algorithm.

Chapter 2

Chapter 2 shows the key processes involved in a GP regression framework. GPs as distributions over functions have a rich history in geo-statistics and machine learning. The second chapter heavily draws ideas from [Krige 1951, Matheron 1963] of the geo-statistics community and [Stein 1999, Kennedy 2000, Rasmussen 2005, MacKay 2003] of the machine-learning community, showing a work-flow of how to perform regression using GPs. The chapter describes the key constituents of a GP, how to draw random functions from them, describes how to perform prediction in presence and absence of measurement noise, and finally introduces marginal likelihood maximization as a form of evaluation method to automatically choose hyper-parameters.

Chapter 3

Chapter 3 deals with the problem of scaling GP regression to massively many points. Traditional GPs are computationally infeasible on a standard laptop if the number of data-points increases to more than $\mathcal{O}(10^4)$. There exist two main methods to scale up a GP regression, one using a reduced set of inducing points while another based on mixture of experts methodology. This chapter draws heavily from the works of [Quiñonero-Candela 2005, Seeger 2003, Snelson 2006, Titsias 2009] for the approximation method of inducing points and [Cao 2014, Tresp 2000, Chen 2009, Deisenroth 2015] for the approximation method of mixture of experts, please refer to the individual publications for more detail. We demonstrate the limitations and capabilities of both the methods on a toy data-set, giving directions to choosing optimal parameters and extracting the best possible result.

Part II

A GP prior can be fully parameterized by its mean and covariance functions. From part II on-wards we only concentrate on manipulating the covariance function to embed the desired *a-priori* information into a GP prior. Part II demonstrates how to incorporate an *a-priori* pattern into a regression framework. It consists of two chapters, chapter 4 describes

basic covariance functions, while chapter 5 explains how to make new covariance functions by combining the basic covariance functions. This part has been inspired from the works of [Bishop 2006, MacKay 2003, Duvenaud 2014, Wilson 2014, Lloyd 2014, Durrande 2001, Durrande 2013a].

Chapter 4

Chapter 4 describes different types of covariance functions and their corresponding families of functions. Patterns such as smoothness, linearity, differentiability, etc. can be easily encoded using simple covariance functions. We list the properties of basic stationary and non-stationary covariance functions, while also giving an example to demonstrate the effects of choosing different covariance functions on the final regression model. Stationary kernels have an interesting property that their Fourier transforms exist and are more interpretable. We exploit this property to learn modal parameters of a structure from their dynamic excitations.

Chapter 5

Chapter 5 provides intuition on what happens when we combine different covariance functions. We first look at the effects of simply adding or multiplying covariances in one-dimensional inputs (section 5.2), and then investigate how to create covariance functions for multi-dimensional inputs (section 5.3). We use these kind of covariance functions to detect onset of plasticity in Tensile experiments on an aluminum alloy (section 5.2.4) and interpolate pressures in the Transonic regime (section 5.3.5).

Part III

Several real-world problems often exhibit strong correlations between output variables, for example, correlations across spatial coordinates (x, y, z) in an experiment. Part III of this thesis demonstrates how to incorporate an *a-priori* information among multiple outputs, this is also called as 'Multi-Task Gaussian Process' (MTGP) in GP literature. This part is split into two chapters, chapter 6 describes the MTGP for the case of multi-fidelity simulations, while chapter 7 describes the MTGP for the case when we have *a-priori* information of relationships between outputs. This part has been inspired from the works of [Forrester 2007, Alvarez 2011, Bonilla 2008, Boyle 2005, Kennedy 2000, Le Gratiet 2013, Constantinescu 2013, Alvarez 2009, Jidling 2017, Ginsbourger 2013, Särkkä 2011]

Chapter 6

Chapter 6 describes how to build GP regression models in presence of multi-fidelity simulations. The *a-priori* information in this case is that one simulation is more accurate than another. The chapter starts by presenting MTGP models when no information of dependency between outputs exists, we then present the multi-fidelity model. Finally, we expand the multi-fidelity model to the case of extrapolating experimental data using a simulation model.

Chapter 7

Finally, chapter 7 demonstrates how to build an MTGP model if we know a relationship in the form of an operation between two outputs. We first tackle the case of linear operators and then expand the case to non-linear operators. We then give methods to scale up MTGP models to large numbers of data-points, mainly by applying techniques already seen in chapter 3.

1.6 Contributions

Before highlighting the contributions of this thesis it is important to understand the research work in context of a CIFRE thesis. A CIFRE thesis is different from a traditional PhD thesis since the doctoral candidate is an employee of the host company, while also being linked to the research university. Thus the research performed in this thesis is application oriented, mostly trying to solve some industry specific problem.

Developments

The main theme of this thesis is applying the developments from the supervised machine learning community to solving problems in engineering design. We choose GP regression as our supervised Machine Learning algorithm, this choice is driven because GP regression performs best when data is costly or sparse and the covariance functions acts as a good mechanism for encoding functional relationships. The developments proposed in this thesis can be further divided into two subgroups. First, where we have made contributions to the GP regression community. Second, where we have applied a formalism from GP regression literature to problems in aircraft design. Table 1.6 highlights the main contributions of this thesis.

GP + Aircraft Design	
Chapter 4	Identifying Structural dynamic parameters [Chiplunkar 2017b]
Chapter 5	Identifying onset of non-linearity [Chiplunkar 2016b]
Chapter 5	Interpolating Shock position [Chiplunkar 2017a]
Chapter 6	Extrapolating using a simulation model
Chapter 7	Adding flight mechanics to GP [Chiplunkar 2016a]

Additions to GP Regression	
Chapter 7	Scaling multi-task GP [Chiplunkar 2016c, Chiplunkar 2017c]

Table 1.1: List of contributions

When we are applying GP regression to Aircraft design problems, we are effectively combining the prior knowledge coming from decades of experimentation and adding it to a learning algorithm. A model generated from merging of the two methodologies will be both consistent with the physics of the system and be quicker to evaluate. A major roadblock in using GP regression is that building GPs for large number of data-points is a costly task, we will also propose solutions for scaling GPs in this thesis. We answer the following questions throughout our thesis:

1. **Pattern**: How to add *a-priori* information of a pattern in a learning algorithm? For example, given that shock is a discontinuous change in pressure, how to predict the position of shock on an airfoil (part II).

2. **Simulation model**: How to merge *a-priori* information of simulations with experiments? For example, given a simulation model and experimental data how to perform extrapolations on experimental data (chapter 6)?

3. **Relationships**: How to add *a-priori* information of relationships between measurements? For example given $Loads = \int Pressures$, how to make a robust loads model when we measure both pressures and loads (chapter 7)?

4. **Scaling**: How to scale GP regression to large data-sets? How to scale single output and multiple output GPs to large number of outputs (chapter 3 and section 7.4).

The Matlab codes available in this manuscript are only for understanding. The basic toolbox used for this thesis is GPML provided with [Rasmussen 2005], all new functionalities have been grafted into this basic toolbox. The code of this manuscript is available at `https://github.com/ankitchiplunkar/thesis_isae`.

Chapter 2

Gaussian Process Regression

Résumé

Dans ce chapitre nous fournissons une introduction à la Régression avec GPs. Les GPs sont les candidats idéaux à la régression grâce à de leur propriété de marginalisation qui les rend résolubles informatiquement. Ce qui rends les réalisations des fonctions aléatoires, le calcul de distribution *a posteriori* et la sélection automatique d'hyper-paramètres faisables informatiquement aussi. Ceci fait des GPs des candidats idéaux pour définir une distribution *a-priori* dans un cadre de Régression Bayésien.

La section 2.2 détaille les composants clés du GPs. Un GP peut être complètement paramétré par sa moyenne et sa fonction de covariance. La tendance d'un GP est définie par sa fonction moyenne, tandis que la structure de ses fonctions constitutives est définie par la fonction de covariance. La moyenne d'un GP peut être supposée nulle, car un terme supplémentaire dans la fonction de covariance peut représenter la fonction moyenne. Par conséquent, le problème de l'apprentissage dans un GP consiste à trouver les propriétés appropriées de la fonction de covariance (section 2.2.3). Une fois qu'une forme de fonction de covariance est choisie, nous pouvons calculer la matrice de Gram aux points désirés et l'utiliser pour tirer des fonctions aléatoires (section 2.2.4).

La section 2.3 décrit comment calculer la distribution *a posteriori*. Celle-ci est la distribution conditionnelle ($\Pr[f(\boldsymbol{x_*}) \mid \boldsymbol{y}, \boldsymbol{X}, \boldsymbol{\theta}]$) pour une distribution *a-priori* supposée ($\Pr[f] = GP(0, k(\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{\theta})$) et les ensembles des points de données observés ($\mathcal{D} = \boldsymbol{X}, \boldsymbol{y}$). En raison de l'hypothèse Gaussienne, les probabilités conditionnelles sont résolubles informatiquement et peuvent être calculées en utilisant quelques opérations matricielles. Le calcul de la distribution *a posteriori* est facile à la fois en l'absence (section 2.3.1) ou en l'presence (section 2.3.2) de bruit dans les observations.

La section 2.4 montre l'importance de choisir les hyper-paramètres correctement. Une pratique courante dans la communauté est de maximiser la probabilité marginale pour choisir automatiquement les hyper-paramètres. La probabilité marginale est la probabilité d'une distribution *a-priori* $\Pr[f] = GP(0, k(\boldsymbol{x_1}, \boldsymbol{x_2}, \theta)$ générant les observations $\mathcal{D}$. La maximisation de la probabilité marginale donne les hyper-paramètres optimaux.

2.1 Introduction

Data-set

Suppose we perform a simulation or experiment on an input point $\boldsymbol{x_j} \in \mathbb{R}^{D_{inputs}}$ and measure an output $y_j \in \mathbb{R}$. In this chapter we assume that the input is D_{inputs}-dimensional and the output is one-dimensional. We can thus have a data-set of N observations, $\{\mathcal{D} = (\boldsymbol{x_j}, y_j) | j \in [1; N]\}$. The full input and output vectors can be denoted as $\boldsymbol{X} = (\boldsymbol{x_1}, \ldots, \boldsymbol{x_j}, \ldots, \boldsymbol{x_N})^T$ and $\boldsymbol{y} = \{\boldsymbol{y_1}, \ldots \boldsymbol{y_j}, \ldots, \boldsymbol{y_N}\}^T$ such that $\boldsymbol{X} \in \mathbb{R}^{N \times D_{inputs}}$ and $\boldsymbol{y} \in \mathbb{R}^N$. Given this data, we are interested in making predictions on a target point[1] $\boldsymbol{x_*}$ that may not be present in our series of experiments. This means that we need to use our training data and learn the true physical process $f(\boldsymbol{x})$ that generates our data-set.

As discussed in the previous chapter, the first step of a learning algorithm is defining a family of functions. Gaussian Processes (GPs) can be used to probabilistically define the family of functions. More formally, a GP is a distribution over functions such that any finite set of function values $[f(\boldsymbol{x_1}), \ldots, f(\boldsymbol{x_j}), \ldots, f(\boldsymbol{x_N})]$ have a joint Gaussian distribution [Rasmussen 2005].

Infinite dimensional random vector

While a normal distribution describes a scalar random variable (for example $x \sim \mathcal{N}(0, 1)$ defines a Gaussian variable with mean 0 and variance 1). A multi variate distribution defines a vector of random variables (for example $\boldsymbol{x} \sim \mathcal{N}(\binom{0}{0}, \left(\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right))$ defines a Gaussian vector with mean $\binom{0}{0}$ and covariance $\left(\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right)$). A GP is the extension of this concept in the functional space. We can also think of a function as an infinite dimensional vector, with each entry in the vector specifying the function value $f(\boldsymbol{x})$ at a particular point $\boldsymbol{x}$.

Mean

A GP model before conditioning on data can be completely parameterized by its mean $(m(\boldsymbol{x}))$ and its covariance function $(k(\boldsymbol{x_1}, \boldsymbol{x_2}))$ also called a kernel.

$$\mathbf{E}[f(\boldsymbol{x})] = m(\boldsymbol{x}) \tag{2.1}$$

Covariance

In the context of GPs a kernel is a measure of similarity between pairs of functional values $(f(\boldsymbol{x}))$ evaluated at input points, often involving an inner product of basis functions $\phi(\boldsymbol{x})$ [Bishop 2006]. Please refer to Part II for a more detailed insight into kernels[2].

[1] Also called as test point, prediction point or target point.

[2] The terms covariance functions, kernel and kernel functions will be used interchangeably during the reminder of this thesis

$$Cov(f(\boldsymbol{x_1}), f(\boldsymbol{x_2})) = \mathbf{E}[f(\boldsymbol{x_1}) - m(\boldsymbol{x_1}), f(\boldsymbol{x_2}) - m(\boldsymbol{x_2})] \tag{2.2}$$
$$= k(\boldsymbol{x_1}, \boldsymbol{x_2}) \tag{2.3}$$

We can formally write the probability of the function f as:

$$\Pr[f(\boldsymbol{x})] = GP(m(\boldsymbol{x}), k(\boldsymbol{x_1}, \boldsymbol{x_2})) \tag{2.4}$$

The notation $\Pr(f(\boldsymbol{x}))$ symbolizes probability distribution of function f at the input point $\boldsymbol{x}$. A function randomly drawn from a GP yields a random function around the mean function $m(\boldsymbol{x})$, and its shape is defined by the covariance function $k(\boldsymbol{x_1}, \boldsymbol{x_2})$.

Tractable

Performing inference on an infinite dimensional vector (function) can be a computationally intensive task. Thankfully, due to the marginalization property of Gaussians (appendix A.4), if we ask for properties of the function at a finite number of points, then GP will give us the same answer, if we ignore the infinitely many other points. In other words any finite set of function values $[f(\boldsymbol{x_1}), \ldots, f(\boldsymbol{x_j}), \ldots, f(\boldsymbol{x_N})]$ have a joint Gaussian distribution in GP (this is also the formal definition of GP). This property means that GP specified in equation 2.4 also specifies equation 2.5. This makes GPs computationally tractable, which is one of the major benefits of GP.

$$\Pr\begin{bmatrix} f(\boldsymbol{x_1}) \\ f(\boldsymbol{x_2}) \end{bmatrix} = \mathcal{N}\left(\begin{bmatrix} m(\boldsymbol{x_1}) \\ m(\boldsymbol{x_2}) \end{bmatrix}, \begin{bmatrix} k(\boldsymbol{x_1}, \boldsymbol{x_1}) & k(\boldsymbol{x_1}, \boldsymbol{x_2}) \\ k(\boldsymbol{x_2}, \boldsymbol{x_1}) & k(\boldsymbol{x_2}, \boldsymbol{x_2}) \end{bmatrix}\right) \tag{2.5}$$

GP regression work-flow

While performing regression in a GP framework, we first define a family of functions also called prior (section 2.2). The next step involves looking at the data-set and eliminating all the functions in our prior hypothesis space which do not pass through the data-set, this step gives us the posterior mean and variance (section 2.3). Finally, we can further improve our predictions by fine-tuning our hyper-parameters (section 2.4).

2.2 Prior

In the Bayesian framework, a prior is a probability distribution before looking at any evidence. In the context of a GP Regression, this is provided by the mean and covariance function.

2.2.1 Hyper-parameters

Both mean and covariance functions are specified by a set of hyper-parameters θ, these are the parameters of the GP. Selecting a prior in GP boils down to choosing an appropriate

functional form of the mean and covariance matrix and then choosing the hyper-parameters of the prior [Rasmussen 2005]. We will look at how to automatically fine-tune hyper-parameters in section 2.4, whereas part II will describe how to choose the functional form of covariance functions.

2.2.2 Mean function

The mean function $m(\boldsymbol{x})$ of a GP represents its trend. In Universal Kriging, we usually choose a mean function of the form $m(\boldsymbol{x}) = \phi(\boldsymbol{x})^T\boldsymbol{\theta}$, with $\phi(\boldsymbol{x}) = (\phi_1(\boldsymbol{x}), \ldots, \phi_p(\boldsymbol{x}))$ being a vector of basis functions and $\boldsymbol{\theta} \in \mathbb{R}^p$ is a vector of hyper-parameters [Matheron 1963]. In Simple Kriging, we assume a constant mean function $m(\boldsymbol{x}) = constant$.

Zero mean

Without loss of generality, we can assume the mean function to be zero everywhere, since uncertainty about the mean function can be taken into account by adding an extra term to the covariance function (more details in section 5.2.2).

We assume a zero mean prior throughout this thesis. After accounting for the zero mean, the GP model can be completely parametrized by the kernel. Hence the problem of learning in a GP is exactly the problem of finding suitable properties of the covariance function [Rasmussen 2005] (equation 2.6).

$$\Pr[f(\boldsymbol{x})] = GP(0, k(\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{\theta})) \tag{2.6}$$

The Matlab Code 2.1 below is a sample of zero mean function.

```
% zero mean function
meanFunction = @(x) 0*x;
```

Matlab Code 2.1: A zero mean function

2.2.3 Covariance function

The covariance function is a positive definite kernel, such that for any $a_i \in \mathbb{R}$ equation 2.7 is valid [Stein 1999].

$$\sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j k(\boldsymbol{x_1}, \boldsymbol{x_2}) \geq 0 \tag{2.7}$$

A popular choice of covariance function is a Standard Exponential (SE) function (equation 2.8), because it defines a family of highly smooth (infinitely differentiable) non-linear

functions as shown in figure 2.2.

$$k_{SE}(\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{\theta}) = \theta^2_{amplitude} exp\left[-\frac{\boldsymbol{d^2}}{2\theta^2_{lengthScale}}\right] \tag{2.8}$$

SE kernel

For the case of the SE kernel the hyper-parameters ($\boldsymbol{\theta} = [\theta_{amplitude}, \theta_{lengthScale}]$) are amplitude ($\theta_{amplitude}$), which defines average distance from mean, and the length scale ($\theta_{lengthScale}$), which defines the smoothness of functions. Here, $\boldsymbol{d}$ defines the absolute distance between points ($\boldsymbol{d} = |\boldsymbol{x_1} - \boldsymbol{x_2}|$). Covariance functions which are purely a function of distance $\boldsymbol{d}$ are called isotropic stationary functions. These covariance functions remain unchanged if the points $\boldsymbol{x_1}, \boldsymbol{x_2}$ are rotated or translated. Hence a family of functions defined by stationary kernels will have similar local features throughout the input domain.

Length-Scale

When $\boldsymbol{x_1}$ tends to $\boldsymbol{x_2}$, then $k(\boldsymbol{x_1}, \boldsymbol{x_2})$ approaches $\theta^2_{amplitude}$ this means that $f(\boldsymbol{x_1})$ is highly correlated with $f(\boldsymbol{x_2})$. This is a good characteristic for smooth functions since points in the neighbourhood must be alike. If $\boldsymbol{x_1}$ is far away from $\boldsymbol{x_2}$, then $k(\boldsymbol{x_1}, \boldsymbol{x_2})$ tends to zero this means that far away points are loosely correlated. Hence, far off observations will have negligible effect while performing interpolations. How fast or slow the covariance decreases with distance depends on the length scale parameter $\theta_{lengthScale}$, smaller length-scale means a faster moving function. In general we cannot extrapolate more than $\theta_{lengthScale}$ units from the closest data-point [Duvenaud 2014].

The Matlab Code 2.2 below is a sample of SE covariance function, theta(1) is the amplitude hyper-parameter, while theta(2) is the length-scale hyper-parameter.

```
% Standard exponential covariance function
SEKernel = @(theta, x1, x2)(theta(1).^2*exp(-(x1 - x2).^2/(2*
   theta(2).^2)));
% theta(1): is the amplitude hyper-parameter
% theta(2): is the length-scale hyper-parameter
```

Matlab Code 2.2: A SE covariance function

2.2.4 Sampling functions from GP priors

Gram matrix

To have a look at the constituent functions in a prior we can randomly sample functions from the GP. To draw random functions from a GP we choose $N*$ input points $\boldsymbol{X_*} = \{\boldsymbol{x_{1*}}, \boldsymbol{x_{2*}}, \ldots, \boldsymbol{x_{N*}}\}^T$ and write corresponding mean vector $\boldsymbol{m(X_*)}$ and covariance matrix $\boldsymbol{K(X_*, X_*)}$ using equation 2.5 and 2.8, the covariance matrix (equation 2.9) is also called the Gram matrix. We then generate a random Gaussian vector $\boldsymbol{f(X_*)}$ for this multi-variate

Gaussian (equation 2.6) and plot the generated values as a function of inputs $\boldsymbol{X_*}$.

$$\boldsymbol{K(X_*, X_*)} = \begin{bmatrix} k(\boldsymbol{x_{1*}}, \boldsymbol{x_{1*}}) & k(\boldsymbol{x_{1*}}, \boldsymbol{x_{2*}}) & \dots & k(\boldsymbol{x_{1*}}, \boldsymbol{x_{N*}}) \\ k(\boldsymbol{x_{2*}}, \boldsymbol{x_{1*}}) & k(\boldsymbol{x_{2*}}, \boldsymbol{x_{2*}}) & \dots & k(\boldsymbol{x_{2*}}, \boldsymbol{x_{N*}}) \\ \vdots & \vdots & \ddots & \vdots \\ k(\boldsymbol{x_{N*}}, \boldsymbol{x_{1*}}) & k(\boldsymbol{x_{N*}}, \boldsymbol{x_{2*}}) & \dots & k(\boldsymbol{x_{N*}}, \boldsymbol{x_{N*}}) \end{bmatrix} \tag{2.9}$$

The Matlab Code 2.3 below is a sample function to evaluate the Gram Matrix. The function 'evaluateGramMatrix' will later be used regularly to calculate the posterior mean, posterior variance (code 2.5) and choosing hyper-parameters (code 2.8).

```
% Function to evaluate the gram matrix
function gramMatrix = evaluateGramMatrix(covarianceFunction,
   theta, x1, x2)

    [X1, X2] = meshgrid(x1, x2);
    % evaluating element wise gram matrix
    gramMatrix = covarianceFunction(theta, X1, X2);

end

% Initial parameters
N = 100; % Number of testing points
xStar = linspace(-1, 1, N)'; % Input training points

theta(1) = 1; % Amplitude Hyper-parameter
theta(2) = 0.2; % Length Scale Hyper-parameter

% Visualizing the Gram matrix
gramMatrix = evaluateGramMatrix(SEKernel, theta, xStar, xStar);
imagesc(gramMatrix); % Plotting the Gram Matrix
```

Matlab Code 2.3: Plotting the Gram Matrix

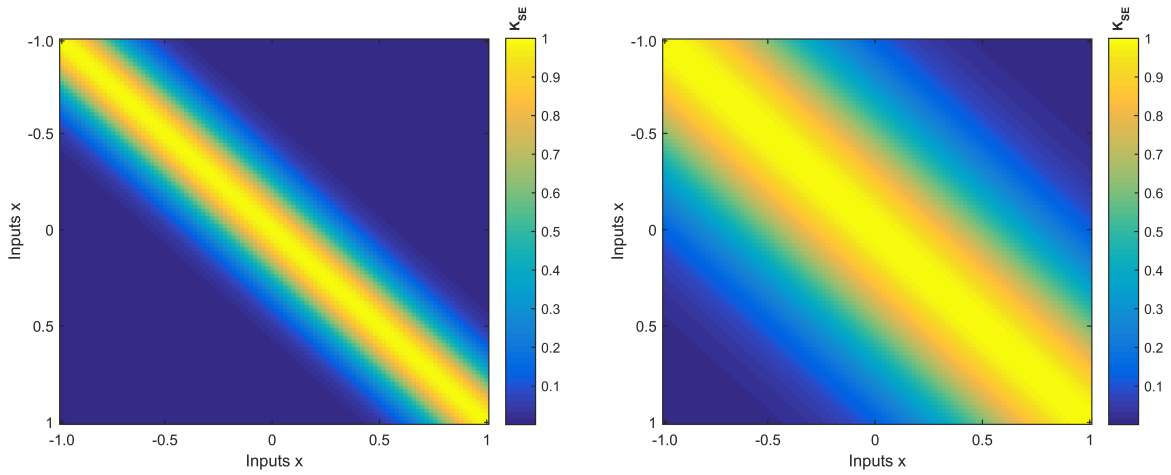
Figure 2.1

Figure 2.1 shows the covariance matrix for SE kernel with different hyper-parameters at the input points $\boldsymbol{X^*} = \{[0 : 0.02 : 1]\}$. The SE kernel of figure 2.1(a) has a lower length-scale than figure 2.1(b). Notice how the covariance values are more spread out for figure 2.1(b).

Random Draw

To generate a random Gaussian vector $\boldsymbol{f(X_*)}$ of length N_*, we first calculate the Cholesky decomposition[3] of the covariance matrix $\boldsymbol{K(X_*, X_*)} = \boldsymbol{K_{Lower} K_{Lower}^T}$, where

[3] Cholesky Decomposition is also called the square-root of matrix and is defined for positive definite matrices



(a) Covariance matrix for a SE Kernel with ($\boldsymbol{\theta} = [1, 0.2]$) at the input points $\boldsymbol{X}^* = \{[0 : 0.02 : 1]\}$.

(b) Covariance matrix for a SE kernel with ($\boldsymbol{\theta} = [1, 0.5]$) at the input points $\boldsymbol{X}^* = \{[0 : 0.02 : 1]\}$

Figure 2.1: Covariance matrix for a SE kernel with different hyper-parameters at the input points $\boldsymbol{X}^* = \{[0 : 0.02 : 1]\}$. The SE kernel of figure 2.1(a) has a lower length-scale than figure 2.1(b). Notice how the covariance values are more spread out for figure 2.1(b).

$\boldsymbol{K_{Lower}}$ is a lower triangular matrix. We then generate a random vector $\boldsymbol{u}$, such that $\boldsymbol{u} = \mathcal{N}(0, \boldsymbol{I_{N_*}})$ and $\boldsymbol{I_{N_*}}$ is an identity matrix of size $N_* \times N_*$. The random vector can then be computed as $\boldsymbol{f}(\boldsymbol{X_*}) = \boldsymbol{m}(\boldsymbol{X_*}) + \boldsymbol{K_{Lower}}\boldsymbol{u}$, and when plotted with the inputs $\boldsymbol{X_*}$, gives a randomly drawn function. The Matlab Code 2.4 shows how we can draw functions randomly from a GP prior.

```
function [randomFunction] = drawRandomFunctions(meanVector,
   gramMatrix)

N = length(meanVector);

% Tip: add a jitter term to the gram matrix so that matrix
   inversion is numerically stable
jitter = 10^(-6);
% The cholesky decomposition
L = chol(gramMatrix + eye(N)*jitter);

randomFunction = meanVector + L'*rand(N, 1);

end

% Initial parameters
```

```
N = 100; % Number of testing points
xStar = linspace(-1, 1, N)'; % Input training points

theta(1) = 1; % Amplitude Hyper-parameter
theta(2) = 0.2; % Length Scale Hyper-parameter

% Plotting the mean of prior
meanVector = meanFunction(xStar);
hold on; plot(xStar, meanVector, 'k');

% Plotting the variance of Prior
gramMatrix = evaluateGramMatrix(SEKernel, theta, xStar, xStar);
f = [meanVector+2*sqrt(diag(gramMatrix)); flipdim(meanVector-2*
   sqrt(diag(gramMatrix)),1)];
hold on; fill([xStar; flipdim(xStar,1)], f, [7 7 7]/8);

% Plotting the randomly drawn sample function
[randomFunction] = drawRandomFunctions(meanVector, gramMatrix);
hold on; plot(xStar, randomFunction, 'b');
```

Matlab Code 2.4: Sampling a random function from the prior

Figure 2.2

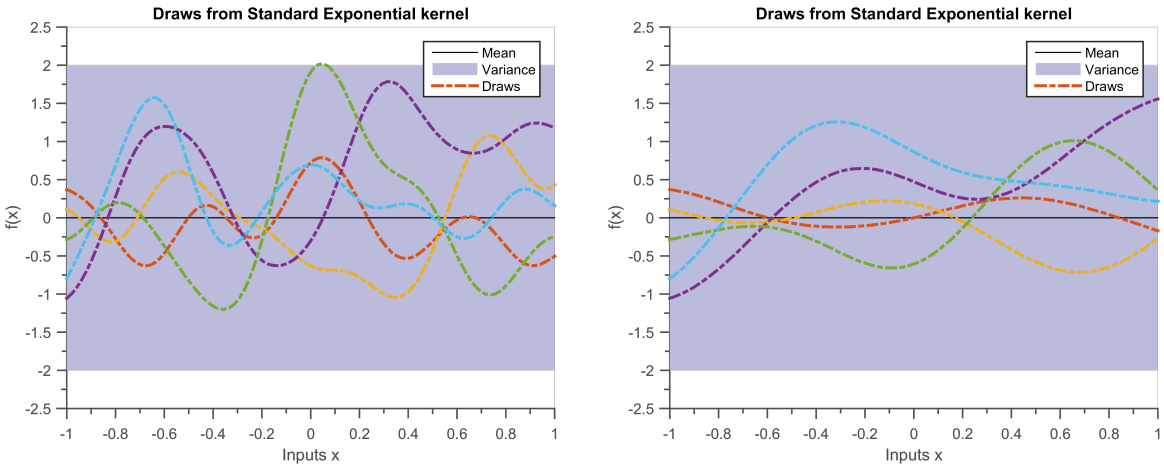
Figure 2.2 shows 5 random functions drawn for a zero mean GP with the covariance matrices of figure 2.1. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior. We can observe that functions in figure 2.2(a) vary faster when compared to functions in figure 2.2(b) due to the smaller length scale hyper-parameter.

2.3 Posterior

Once we have defined an appropriate prior we wish to incorporate the information of the training data-set into the probabilistic framework. In the Bayesian framework, a posterior is the probability distribution after updating the information of evidence into prior distribution.

2.3.1 Posterior with Noise-free observations

We first consider the case of noise-free observations, that is $\{y(x_i) = f_i \ \forall i \in [1; N]\}$. This is the case while creating surrogate models of computer simulations since their outputs can



(a) Draws from a GP prior with mean zero and SE Kernel with ($\boldsymbol{\theta} = [1, 0.2]$).

(b) Draws from a GP prior with mean zero and SE kernel with ($\boldsymbol{\theta} = [1, 0.5]$)

Figure 2.2: The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior. We can observe that figure 2.2(a) varies faster when compared to figure 2.2(b) due to smaller length scale hyper-parameter.

be treated as having no noise [Sacks 1989]. If we desire to interpolate at test points $\boldsymbol{X}_*$, then the joint distribution of the training outputs $\boldsymbol{f}(\boldsymbol{X})$ and test outputs $\boldsymbol{f}(\boldsymbol{X}_*)$ is given by equation 2.10.

$$\Pr\begin{bmatrix}\boldsymbol{f}(\boldsymbol{X})\\ \boldsymbol{f}(\boldsymbol{X}_*)\end{bmatrix} = \mathcal{N}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) & \boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}_*)\\ \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X}) & \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X}_*)\end{bmatrix}\right) \tag{2.10}$$

$\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}_*)$ is $N \times N_*$ covariance matrix between the training points $\boldsymbol{X}$ and test points $\boldsymbol{X}_*$ (equation 2.9). The other covariance matrices $\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X})$, $\boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X})$ and $\boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X}_*)$ can be computed similarly.

Conditioned distribution

The posterior will be the conditional probability of $\boldsymbol{f}(\boldsymbol{X}_*)$ given the prior and data-set. For a multi-variate Gaussian prior the conditional posterior distribution is also a multi-variate Gaussian and can be calculated tractably. For a more detailed derivation refer to appendix A.5. Graphically, we can imagine that the Bayes theorem is removing all the functions from our prior family of functions that do not pass through the data-set (figure 2.3). The predicted distribution after adding the information of data-set into the prior can

be written as:

$$\Pr(\boldsymbol{f}(\boldsymbol{X}_*) \mid \boldsymbol{X}_*, \boldsymbol{X}, \boldsymbol{f}(\boldsymbol{X})) = GP(\boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}))^{-1}\boldsymbol{f}(\boldsymbol{X}), \\ \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}_*) - \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}))^{-1}\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}_*)) \tag{2.11}$$

The term $\boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}))^{-1}\boldsymbol{f}(\boldsymbol{X})$ is the predicted mean of the posterior at the test points $\boldsymbol{X}_*$. The term $\boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X}_*) - \boldsymbol{K}(\boldsymbol{X}_*, \boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}))^{-1}\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}_*)$ is the predicted covariance.

We again use the data-set $\mathcal{D}_1$ (earlier used in section 1.4) to calculate the posterior distribution. The Matlab Code 2.5 shows how to calculate the posterior mean and variance. Notice that calculating the posterior mean and variance require only a few matrix operations.

.

```
function [meanPosterior, gramMatrixPosterior] =
    evaluatePosterior(meanFunction, covarianceFunction, theta, x,
     xStar)

% Evaluating the mean vector
meanVector = meanFunction(xStar);

% Evaluating the neccesary matrices
gramMatrixXstarX = evaluateGramMatrix(covarianceFunction, theta,
     xStar, x);
gramMatrixXX = evaluateGramMatrix(covarianceFunction, theta, x,
    x);
gramMatrixXstarXstar = evaluateGramMatrix(covarianceFunction,
    theta, xStar, xStar);

+% Calculating the posterior mean and covariance
meanPosterior = meanVector + gramMatrixXstarX*(gramMatrixXX\y);
gramMatrixPosterior = gramMatrixXstarXstar - gramMatrixXstarX*(
    gramMatrixXX\gramMatrixXstarX');

end

%% Dataset D_1
x = [-0.5, 1/3, 2/3]';
y = [0, 0.5, 0.5]';
hold on; plot(x, y, '*');
```

```
[meanPosterior, gramMatrixPosterior] = evaluatePosterior(
   meanFunction, SEKernel, theta, x, y, xStar);
% Plotting the variance and mean of Posterior
hold on; plot(xStar, meanPosterior, 'k');
f = [meanPosterior+2*sqrt(diag(gramMatrixPosterior)); flipdim(
   meanPosterior-2*sqrt(diag(gramMatrixPosterior)),1)];
hold on; fill([xStar; flipdim(xStar,1)], f, [7 7 7]/8)

% Plotting a randomly drawn sample
[randomFunction] = drawRandomFunctions(meanPosterior,
   gramMatrixPosterior);
hold on; plot(xStar, randomFunction, 'b');
```

Matlab Code 2.5: Calculating and plotting the mean, the variance and a sample of the posterior

Figure 2.3 shows the posterior GP after adding observed data into the initial prior. We can see that thanks to the Bayes theorem all the functions that do not pass through the data, are eliminated from the hypothesis space. The solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from the mean. The mean of the posterior distribution is also used as a point estimate for interpolation. The dashed lines represent three functions drawn at random from a GP posterior. Random functions can be sampled from the posterior distribution as described in the earlier section.

Figure 2.3

2.3.2 Posterior with Noisy observations

If we assume a more general case of noisy observations, then the measured outputs can be written as:

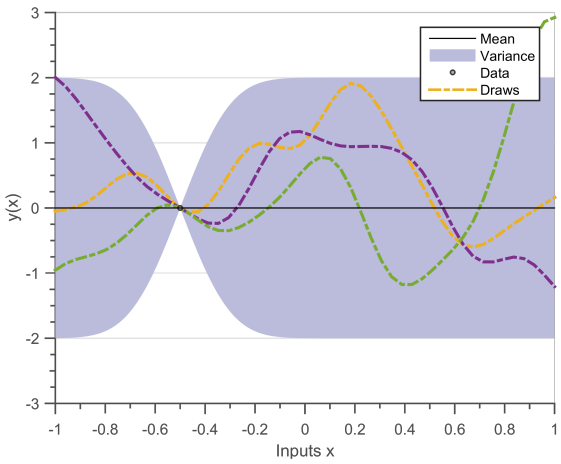
$$y(\boldsymbol{x}) = f(\boldsymbol{x}) + \epsilon \tag{2.12}$$

Such that ϵ is an independent random noise sampled from a white noise Gaussian $\mathcal{N}(0, \sigma_n^2)$. We can thus write the prior GP of the noisy case as:

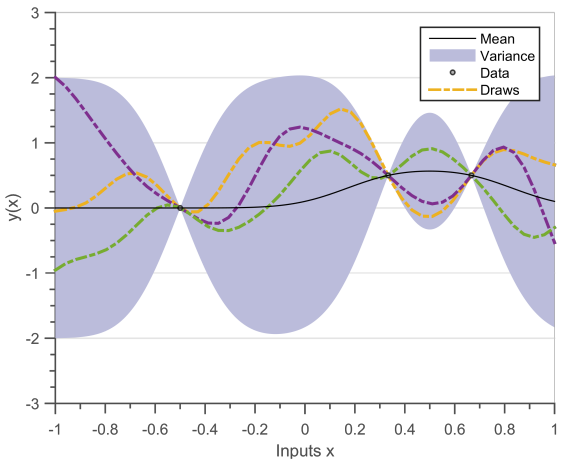
$$\Pr[y(\boldsymbol{x})] = GP(0, k(\boldsymbol{x_1}, \boldsymbol{x_2}) + \sigma_n^2 \delta_{x_1 x_2}) \tag{2.13}$$

Here, $\delta_{x_1 x_2}$ is a Kronecker delta function, which is one if $x_1 = x_2$ and zero otherwise. Since the noise is independent for each observation, there is no noise term for covariances across inputs. The Matlab code 2.6 shows a method to add independent noise into a SE covariance function.

Noise model



(a) Posterior distribution for the case of noiseless observations. Prior is a GP with mean zero and covariance as SE Kernel with ($\boldsymbol{\theta} = [1, 0.2]$), data-set is $\{x = -0.5; f = 0\}$.



(b) Posterior distribution for the case of noiseless observations. Prior is a GP with mean zero and covariance as SE Kernel with ($\boldsymbol{\theta} = [1, 0.2]$), data-set is $\{x = [-0.5, 0.33, 0.66]; f = [0, 0.5, 0.5]\}$.

Figure 2.3: Prediction in the case of noiseless observations. The solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent three functions drawn at random from a GP posterior. We can observe that Bayes Theorem eliminates all the functions that do not pass through the observed data-set.

```
function covariance = noisySEKernel(theta, x1, x2)
% theta(1): is the amplitude hyperparameter
% theta(2): is the length-scale hyperparameter
% theta(3): is the noise hyperparameter

if x1 == x2 % addition of white noise
    covariance = SEKernel(theta(1, 2), x1, x2) + theta(3)^2;
else % normal SE kernel
    covariance = SEKernel(theta(1, 2), x1, x2);
end

end
```

Matlab Code 2.6: Matlab code for a noisy SE covariance function

The joint distribution of the training outputs $\boldsymbol{y}(\boldsymbol{X})$ and true physical process $\boldsymbol{f}(\boldsymbol{X}_*)$

according to the above prior becomes equation 2.14.

$$\Pr\begin{bmatrix}\boldsymbol{y}(\boldsymbol{X})\\ \boldsymbol{f}(\boldsymbol{X}_*)\end{bmatrix} = \mathcal{N}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma_n^2\boldsymbol{I}_{\boldsymbol{N}} & \boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}_*)\\ \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X}) & \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X}_*)\end{bmatrix}\right) \tag{2.14}$$

The difference between equation 2.14 and 2.10 is the addition of noise term $\sigma_n^2\boldsymbol{I}_{\boldsymbol{N}}$. Since the noise at each measurement point is assumed to be independent, it is multiplied to an identity matrix. To know how to add more complex noise models please refer to section 5.2.2. The posterior distribution of $\boldsymbol{f}(\boldsymbol{X}_*)$ can be calculated as follows:

$$\begin{aligned}\Pr(\boldsymbol{f}(\boldsymbol{X}_*) \mid \boldsymbol{X}_*, \boldsymbol{X}, \boldsymbol{y}(\boldsymbol{X})) = GP(&\boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma_n^2\boldsymbol{I}_{\boldsymbol{N}})^{-1}\boldsymbol{f}(\boldsymbol{X}),\\ &\boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X}_*) - \boldsymbol{K}(\boldsymbol{X}_*,\boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}) + \sigma_n^2\boldsymbol{I}_{\boldsymbol{N}})^{-1}\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}_*))\end{aligned} \tag{2.15}$$

Figure 2.4 shows the posterior GP after adding observed data into the initial prior. The solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent three functions drawn at random from a GP posterior. Random functions can be sampled from the posterior distribution as described in the section 2.2. Due to the inclusion of noise in the prior, we see that the draws from posterior are not necessarily passing through the observed point.

Figure 2.4

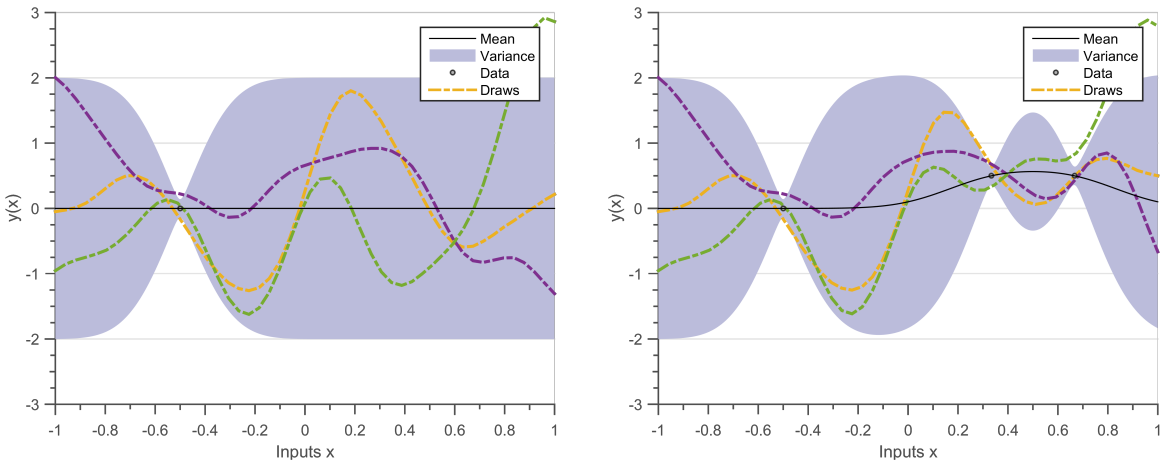
2.3.3 Interpretation of posterior

We will now introduce a short hand notation and replace the lengthy notation $\boldsymbol{K}(\boldsymbol{X_1},\boldsymbol{X_2})$ with $\boldsymbol{K}_{\boldsymbol{X_1X_2}}$ if it is a matrix, $\boldsymbol{k}_{\boldsymbol{X_1X_2}}$ if it is a vector, and $k_{X_1X_2}$ if it is a scalar. For the case where we have only one test point $\boldsymbol{x_*}$, we can write the predictive mean and variance in short-hand as:

$$\mathbf{E}[f(\boldsymbol{x}_*)] = \boldsymbol{k}_{\boldsymbol{Xx}_*}^T(\boldsymbol{K}_{\boldsymbol{XX}} + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{y} \tag{2.16}$$

$$Cov[f(\boldsymbol{x}_*)] = k_{x_*x_*} - \boldsymbol{k}_{\boldsymbol{Xx}_*}^T(\boldsymbol{K}_{\boldsymbol{XX}} + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{k}_{\boldsymbol{Xx}_*} \tag{2.17}$$

Precision Matrix Both the predictive mean (equation 2.16) and predictive covariance (equation 2.17) need the inverse of the covariance matrix $(\boldsymbol{K}_{\boldsymbol{XX}} + \sigma_n^2\boldsymbol{I})^{-1}$. The inverse of a covariance matrix is also known as the precision matrix. While the elements of a covariance matrix capture the variance and correlation information, a precision matrix contains the conditional dependence information [MacKay 2003]. Thus, if the $(i,j)^{th}$ element of a precision matrix is zero, the i^{th} and j^{th} random variables are conditionally independent.



(a) Posterior distribution for the case of noisy observations. Prior is a GP with mean zero, covariance as SE Kernel with ($\boldsymbol{\theta} = [1, 0.2]$) and noise as $\sigma_n = [0.02]$), , data-set is $\{x = -0.5; f = 0\}$.

(b) Posterior distribution for the case of noisy observations. Prior is a GP with mean zero, covariance as SE Kernel with ($\boldsymbol{\theta} = [1, 0.2]$) and noise as $\sigma_n = [0.02]$), , data-set is $\{\boldsymbol{x} = [-0.5, 0.33, 0.66]; \boldsymbol{f} = [0, 0.5, 0.5]\}$.

Figure 2.4: Prediction in the case of noisy observations. The solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent three functions drawn at random from a GP posterior. The mean and the draws do not pass exactly from the observation points.

$\mathcal{O}(N^3)$

Calculating the precision matrix is an $\mathcal{O}(N^3)$ operation for a covariance matrix of size N. After $N \sim 10,000$ a normal computer runs out of RAM, and thus cannot perform the inversion. Fortunately, there exist several approximations to efficiently invert the covariance matrix and perform predictions, more details are available in chapter 3.

Predicted mean The predictive mean is a linear combination of the observations y_i, and factors of $\boldsymbol{k}_{\boldsymbol{X}\boldsymbol{x}_*}^T(\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2\boldsymbol{I})^{-1}$. Since a SE kernel $\boldsymbol{k}_{\boldsymbol{X}\boldsymbol{x}_*}^T$ decreases exponentially with distance, observations closer to x_* have more impact on the final prediction (equation 2.18).

$$\mathbf{E}[f(\boldsymbol{x}_*)] = \sum_{i=1}^{N} \boldsymbol{k}_{\boldsymbol{X}\boldsymbol{x}_*}^T(\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2\boldsymbol{I})^{-1} y_i \tag{2.18}$$

The predictive mean can also be interpreted as a linear combination of the basis functions $k_{x_i x_*} \in \mathbb{R}$, and participation factors $(\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2\boldsymbol{I})^{-1}Y$ (equation 2.19).

$$\mathbf{E}[f(\boldsymbol{x}_*)] = \sum_{i=1}^{N} k_{x_i x_*}(\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{y} \tag{2.19}$$

This means that even though a GP represents an infinite-dimensional vector (function), to predict the mean we only care about the N dimensional multivariate Gaussian (section 2.14). If the precision matrix is cached, then calculating the mean is an $\mathcal{O}(N)$ operation. $\mathcal{O}(N)$

Predicted variance The predicted variance is a combination of two terms, $k_{x_*x_*}$ which is the variance due to prior assumptions, and $-\boldsymbol{k}_{\boldsymbol{X}\boldsymbol{x}_*}^T(\boldsymbol{K}_{\boldsymbol{XX}} + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{k}_{\boldsymbol{X}\boldsymbol{x}_*}$, which denotes the decrease in variance due to the observations. The predictive covariance of test targets $y(x_*)$ can be calculated by adding a noise term σ_n^2 in predictive covariance equation 2.17.

$$Cov[y(\boldsymbol{x}_*)] = k_{x_*x_*} - \boldsymbol{k}_{\boldsymbol{X}\boldsymbol{x}_*}^T(\boldsymbol{K}_{\boldsymbol{XX}} + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{k}_{\boldsymbol{X}\boldsymbol{x}_*} + \sigma_n^2 \tag{2.20}$$

We observe that the predicted variance in not dependent on the observations y, this is one of the flaws in GP regression. Since the assumption that the data-set ($\mathcal{D}$) comes from a GP might not necessarily be true, the predicted variance can poorly represent the model error. Hence, predicted variance is not necessarily a measure of model error but an efficient method to track uncertainties arising from the prior assumption and non-continuous observations [Shah 2014].

The mean and variance are highly dependent on the hyper-parameters. In order to automatically learn the hyper-parameters, we must perform model selection. Section 2.4 details how to fine-tune hyper-parameters to find an optimal prediction.

2.4 Choosing Hyper-parameters

Since the properties of functions under a GP are controlled by the functional form of the covariance kernel and its hyper-parameters, model selection amounts to choosing a functional form and learning the hyper-parameters $\boldsymbol{\theta}$ from data. In this section we discuss how to select an optimal model by tuning hyper-parameters for a given covariance function. Please refer to part II for discussion on how to choose covariance functions.

We define a new data-set $\mathcal{D}_2$ which will be used to compare predictions using different hyper-parameters. The function $f(x)$ (equation 2.21) is evaluated at 20 equidistant points between $x \in [-1, 1]$ and is corrupted by an independent white noise having variance $\sigma_{noise}^2 = 0.1^2$. Matlab code 2.7 is a sample code to generate the data-set $\mathcal{D}_2$. *data-set* $\mathcal{D}_2$

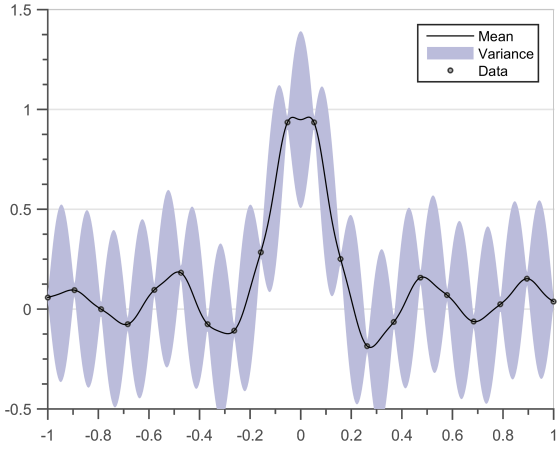
$$f(x) = \frac{sin(5\pi x)}{5\pi x} \tag{2.21}$$

```
nData = 20; % number of data-points

f = @(x)sin(5*pi*x)./(5*pi*x); % Function

noise = 0.1; % Noise in dataset
xData = linspace(-1, 1, nData)';
yData = f(xData) + noise^2*rand(nData, 1);
```

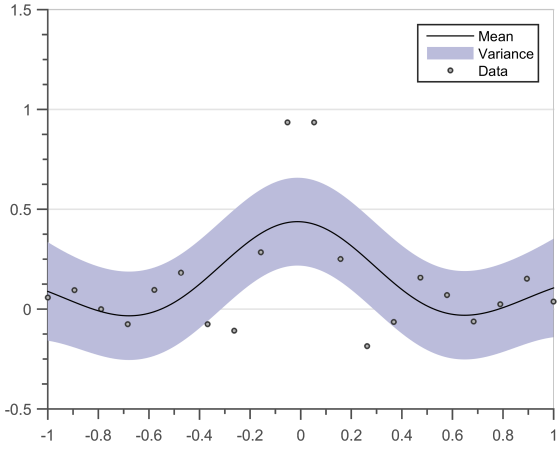
Matlab Code 2.7: Code for data-set D2

Figure 2.5

Figure 2.5 demonstrates that choosing optimal hyper-parameters is very vital for accurate prediction. It compares the posterior distributions obtained for SE priors with two different hyper-parameters. We observe that the mean of figure 2.5(a) passes through all the observed data-points but is more complex. The mean in figure 2.5(b) is a smooth function but does not fit the data properly.



(a) Posterior between SE prior with hyper-parameters ($\boldsymbol{\theta} = [0.35, 0.05]; \sigma_{noise} = 0.01$) and data.



(b) Posterior between SE prior with hyper-parameters ($\boldsymbol{\theta} = [0.35, 0.5]; \sigma_{noise} = 0.01$) and data.

Figure 2.5: Posteriors for 2 different sets of hyper-parameters. Solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from mean.

To estimate the hyper-parameters in a pure Bayesian framework, we should put a prior over our hyper-parameters $\Pr[\boldsymbol{\theta}]$ and use Bayes Rule to estimate the posterior $\Pr[\boldsymbol{\theta} \mid \boldsymbol{X}, \boldsymbol{y}]$ over our data-set (just like we did in section 1.4).

$$\Pr[\boldsymbol{\theta} \mid \boldsymbol{X}, \boldsymbol{y}] = \frac{\Pr[\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}] \times \Pr[\boldsymbol{\theta}]}{\Pr[\boldsymbol{y} \mid \boldsymbol{X}]} \tag{2.22}$$

The normalizing constant in the denominator is given by the following integral.

$$\Pr[\boldsymbol{y} \mid \boldsymbol{X}] = \int \Pr[\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}] \Pr[\boldsymbol{\theta}] d\boldsymbol{\theta} \tag{2.23}$$

Depending on the functional form of the covariance function the normalizing constant may or may not be analytically integrable. We are effectively integrating (marginalizing) out the hyper-parameters from our probability distribution (appendix A.4). Hence, this approach becomes intractable and several sampling schemes have been proposed to calculate the posterior of hyper-parameters [Osborne 2010, Neal 2011].

LOO-CV

Another method to find the optimal hyper-parameters is by performing Cross-Validation (CV). CV procedure is to split the experimental design set into two disjoint sets, one is used for training and the other one is used to monitor the performance of the surrogate model. A particular case of CV is the Leave-One-Out (LOO) where test sets are obtained by removing one observation at-a-time [Rasmussen 2005, Dubrule 1983, Le Gratiet 2013].

In this manuscript we neither put a prior over our hyper-parameters nor use LOO-CV for choosing hyper-parameters. We integrate out the latent functions f to calculate the marginal likelihood, and maximize it to find optimal hyper-parameters [MacKay 2003]. The probability of generating the observations ($\boldsymbol{y}$) at the points ($\boldsymbol{X}$) from a prior (defined by $k(\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{\theta})$) is called the marginal likelihood $\Pr[\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}]$. In other words, marginal likelihood is the probability that our data-set $\mathcal{D}$ was generated from a particular prior. Hence, when we maximize a marginal likelihood we are finding the best prior that could generate our data-set. Using equation 2.13 and 2.14 we get:

$$\begin{aligned} \Pr[\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}] &= \mathcal{N}(0, \boldsymbol{K(X, X')} + \sigma_n^2 \boldsymbol{I}) \\ &= \frac{1}{\sqrt{(2\pi)^N \boldsymbol{K_{XX}}}} exp^{-\frac{1}{2}\boldsymbol{y}^T \boldsymbol{K_{XX}} \boldsymbol{y}} \end{aligned} \tag{2.24}$$

Directly maximizing the marginal likelihood with respect to the hyper-parameters can be inefficient. This is because the marginal likelihood does not vary significantly with the hyper-parameters. Hence to speed up the optimization process we generally maximize the log of marginal likelihood [Rasmussen 2005].

$$\log(\Pr[\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}]) = -\frac{1}{2}\boldsymbol{y}^T[\boldsymbol{K_{XX}} + \sigma_n^2 \boldsymbol{I}]^{-1}\boldsymbol{y} - \log\left|\boldsymbol{K_{XX}} + \sigma_n^2 \boldsymbol{I}\right| - \frac{N}{2}\log(2\pi) \tag{2.25}$$

Trade-off

The log marginal likelihood is composed of three terms a data-fit term ($-\frac{1}{2}\boldsymbol{y}^T[\boldsymbol{K_{XX}} + \sigma_n^2 \boldsymbol{I}]^{-1}\boldsymbol{y}$), a model complexity term ($\log|\boldsymbol{K_{XX}} + \sigma_n^2 \boldsymbol{I}|$), and a normalization term ($\frac{N}{2}\log(2\pi)$). It performs a trade-off between the data-fit term and the complexity

term. The optimization of log marginal likelihood provides the best compromise in terms of explaining the existing data-set $\{(x_i, y_i)\}$ and the initial assumptions encoded in the prior. The Matlab code 2.8 defines the function 'logMarginalLikelihood' (equation 2.25), which is maximized later in the same code.

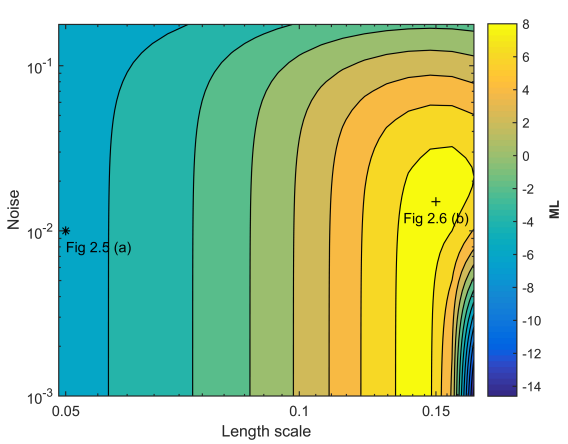
```
function logMarginalLikelihood = logMarginalLikelihood(theta,
    covarianceFunction, x, y)
N = length(x);

% Gram Matrix
gramMatrixXX = evaluateGramMatrix(covarianceFunction, theta, x,
    x);

% The data fit term
dataFitTerm = -1*(0.5)*y'*inv(gramMatrixXX)*y;

% The complexity term
jitter = 10^(-6);
L = chol(gramMatrixXX + eye(N)*jitter);
complexityTerm = -1*sum(log(diag(L)));

% Normalization term
normalizationTerm = -1*N*log(2*pi)/2;
logMarginalLikelihood = dataFitTerm + complexityTerm +
    normalizationTerm;

end

% Optimizing the log marginal likelihood
theta = [1, 0.2];
% Amplitude hyp = 1
% Length Scale = 0.2

options = optimoptions('fminunc','GradObj','off', 'MaxIter',
    100); % indicate gradient is provided
optimizedTheta = fminunc(@(x) -1*logMarginalLikelihood(x,
    SEKernel, xData, yData), theta, options);
```

Matlab Code 2.8: Optimizing the Log Marginal Likelihood

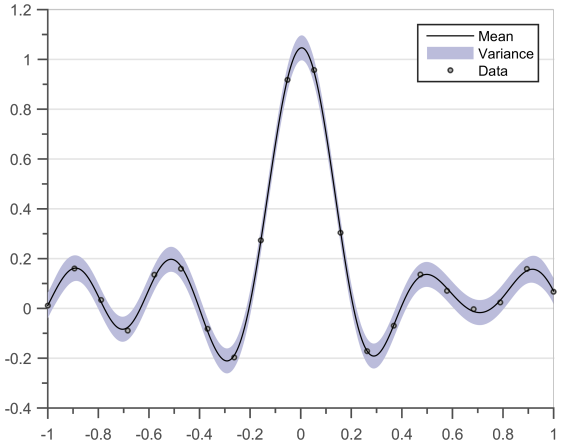
Figure 2.6(a)

Figure 2.6(a) shows the contours of the marginal likelihood with respect to length-scale $\theta_{lengthScale}$ and noise σ_n hyper-parameters. The data-set ($\mathcal{D}_2$) is same as used in figure 2.5 and the prior is a zero mean with SE kernel. Figure 2.6(b) shows the posterior for the

same data-set as used in figure 2.5 but for the hyper-parameters where marginal likelihood is maximum.



(a) Marginal likelihood contours for varying noise and length-scale parameter. The amplitude hyper-parameter is ($\theta_{amplitude} = [0.35]$). Also shown on the figure are locations of hyper-parameters for figures 2.5(a) and 2.6(b).



(b) Posterior between SE prior with optimized hyper-parameters ($\theta = [0.35, 0.15]; \sigma_{noise} = 0.015$) and data. $\log(ML) = 8.04$. Solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from mean.

Figure 2.6: Maximizing marginal likelihood

The marginal likelihood could have multiple maxima in the space of hyper-parameters. Since we do not know *a-priori* the multi-modality of log marginal likelihood, proper care should be taken while optimizing the hyper-parameters. [Forrester 2008] propose the use of a global optimizer (genetic algorithm) to find the global optimum, while [Le Gratiet 2013, Bouhlel 2016a] propose the use of cross validation to find the global optimum.

Multi modality

2.5 Summary and discussion

In this chapter we provide a brief introduction on how to perform Regression with GPs. GPs are the ideal candidate for regression due to their marginalization property which makes them computationally tractable. Even if GPs define an infinite dimensional random vector, inference on a few points does not require the presence of infinitely other points. This makes drawing functions, calculating posterior distribution, and automating selection of hyper-parameters computationally feasible.

Section 2.2 details the key components of the GPs. A GP can be completely parametrized by its mean and covariance function. While the trend of a GP is defined by its mean function, the structure of its constituent functions is defined by the covariance

Section 2.2

function. The mean of a GP can be assumed to be zero, since an extra term in the covariance function can represent the mean function. Hence the problem of learning in a GP is exactly the problem of finding suitable properties of the covariance function (subsection 2.2.3). Once a function form of covariance is chosen, we can calculate the Gram matrix at desired points and use it to draw random functions from our prior (subsection 2.2.4).

Section 2.3

Section 2.3 describes how to calculate the posterior distribution. The posterior is the conditional distribution ($\Pr[f(\boldsymbol{x_*}) \mid \boldsymbol{y}, \boldsymbol{X}, \boldsymbol{\theta}]$) for an assumed Prior distribution ($\Pr[f] = GP(0, k(\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{\theta}))$ and a set of observed data-points ($\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{y}\}$). Due to the Gaussian assumption, the conditional probabilities are all computationally tractable and can be calculated using a few matrix operations, side-stepping the computational burden of performing iterative sampling. Calculating the posterior is easy both in the absence (subsection 2.3.1)and presence (subsection 2.3.2) of noise in observations.

Section 2.4

Given a functional form of the covariance, section 2.4 shows the importance of choosing the correct hyper-parameters. In a pure Bayesian framework the posterior distribution of the hyper-parameters should be calculated ($\Pr[\boldsymbol{\theta} \mid \boldsymbol{X}, \boldsymbol{y}]$), but this is formally non-computable, needing several iterations for calculation of integrals. A common practice in the community is maximizing the marginal likelihood to automatically choose the hyper-parameters. Marginal likelihood is the probability of a prior distribution $\Pr[f] = GP(0, k(\boldsymbol{x_1}, \boldsymbol{x_2}, \theta)$ generating the observations $\mathcal{D}$. Hence maximizing the marginal likelihood gives the optimal set of hyper-parameters for a functional form of covariance function (figure 2.6).

Calculating the precision matrix $[\boldsymbol{K_{XX}} + \sigma^2_{noise}\boldsymbol{I}]^{-1}$ is an important task in calculating the marginal likelihood, posterior mean and posterior covariance. Unfortunately, this task has a computational complexity of $\mathcal{O}(N^3)$ and memory footprint of $\mathcal{O}(N^2)$. This puts an upper limit of $N \sim 10^4$ on the number of data-points, a standard laptop cannot store such a big matrix for inversion [4].

In the certification phase of aircraft design, we have access to millions ($N \sim 10^6$) of training data-points which depend on several different parameters. [Bouhlel 2016b] tackle the problem of building models across high-dimensional input spaces. The next chapter describes few methods of performing approximate inference which scales GPs to $N \sim 10^6$ or more data-points.

[4] The computer runs out of memory before we run out of patience :p

Chapter 3

Scaling up Gaussian Process Regression

Résumé

Le calcul de lois a posteriori dans les GPs devient difficile pour les ensembles de données de grande taille. Le calcul de la matrice de précision est une opération de complexité $\mathcal{O}(N^3)$, mettant une limite de $N \sim 10^4$ points de données pour la construction de modèles. Ce chapitre décrit l'état de l'art pour l'étendue des GPs pour les tâches de régression. Il existe deux méthodes pour étendre des GPs, la première méthode 'Sparse GPs' utilisent un ensemble des points inductifs pour réduire le coût de calcul de la matrice de précision. La seconde est appelée 'Distributed GPs' elle divise l'ensemble de données en sous-ensembles plus petits, en distribuant le modèle en plusieurs lots.

Les méthodes 'Sparse GPs' utilisent l'approximation de Nyström, récrivant la matrice de Gram, diminuant ainsi la complexité de calcul à $\mathcal{O}(NM^2)$ (M $\ll$ N) , M étant le nombre de points inductifs. Grâce à des expériences sur les données synthétiques, on peut montrer que on peut fixer $M \sim N/10$ quand les points inductifs sont distribués aléatoirement et $M \sim N/50$ quand les emplacements des points inductifs sont optimisés. Cette approximation pousse la limite de régression GP à $N \sim 10^6$ points de données.

"Distributed GPs" distribue les tâches de régression GP dans plusieurs lots, diminuant ainsi la complexité informatique à $\mathcal{O}(NP^3)$ (P $\ll$ N), P étant le nombre des points dans un lot. Grâce à des expériences sur les données synthétiques, nous démontrons que $P \sim N/100$ n'affecte pas beaucoup la précision de la régression. En fait, nous pouvons réduire davantage P si nous permettons la répétition de points entre les lots. Cela permet d'étendre le GP à n'importe quel nombre de points de données.

3.1 Introduction

The GP regression approach, as mentioned in earlier chapter, is intractable for large data-sets. For a data-set of size N the covariance matrix $\boldsymbol{K_{XX}}$ is of size $N \times N$ and $\mathcal{O}(N^3)$ time is needed for calculating the precision matrix and $\mathcal{O}(N^2)$ memory for storage. Since inverting the covariance matrix takes considerable amount of time and memory, almost all techniques to scale up GP regression try to approximate the inversion of Gram matrix $\boldsymbol{K_{XX}}$.

Low-rank approximation

Let us take the example of a SE kernel, for a high value of length-scale, the Gram matrix ($\boldsymbol{K_{XX}}$) is spread out and has a rank lower than N (figure 2.2(b)). Due to this characteristic, the Gram matrix can be approximated using low-rank approximations, reducing the cost of inverting the Gram matrix. In the GP literature, sparse approximations (section 3.2) use a set of inducing points to compress the information of the several observations through the low-rank approximation.

Block-diagonal approximation

For the same SE kernel, if the length-scale tends to a low value, the Gram matrix is not of low-rank but tends to a diagonal matrix (figure 2.2(a)). In the GP literature the mixture of experts (section 3.3) methodology exploits the block diagonal nature of the Gram matrix by distributing data-points into a subset of experts, assuming independence across experts and distributing the calculations into several batches. The first regime suggests global (numerical) low-rank approximations while the second regime suggests local block-diagonal approximations [March 2015, Chenhan 2016].

The remaining chapter unfolds as follows, section 3.2 describes the Sparse Approximations detailing several methods of choosing inducing points and then performing experiments on a toy data-set. Section 3.3 describes the Distributed GPs methodology detailing several methods for merging of experts and then performing experiments on the same toy data-set.

3.2 Sparse Approximations

Sparse methods use a small subset of input points as support or inducing points to approximate the Gram matrix. Suppose we use M inducing points $\boldsymbol{X}^{\boldsymbol{m}} = \{\boldsymbol{x_1^m}, \boldsymbol{x_2^m}, \ldots, \boldsymbol{x_M^m}\}^T$, such that $M < N$. The points $\boldsymbol{X}^{\boldsymbol{m}}$ can be a subset of training inputs in the input space.

3.2.1 Nyström Approximation

Using Nyström approximation the Gram matrix can be approximated as equation 3.1 [Quiñonero-Candela 2005, Seeger 2003], for more detailed derivation refer to

[Williams 2001].

$$\boldsymbol{K}_{Nystorm}(\boldsymbol{X},\boldsymbol{X}) = \boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}^m)\boldsymbol{K}(\boldsymbol{X}^m,\boldsymbol{X}^m)^{-1}\boldsymbol{K}(\boldsymbol{X}^m,\boldsymbol{X}) \tag{3.1}$$

Here, $\boldsymbol{K}(\boldsymbol{X}^m,\boldsymbol{X}^m)$ is a $M \times M$ Gram matrix evaluated at inducing points $\boldsymbol{X}^m$, $\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X}^m)$ is an $N \times M$ Gram matrix between training points and inducing points. The inversion of approximate Gram matrix takes $\mathcal{O}(NM^2)$ time to compute. The code 3.1 defines a function 'evaluateNystromGramMatrix' which is used to evaluate the Nström approximation of the Gram matrix.

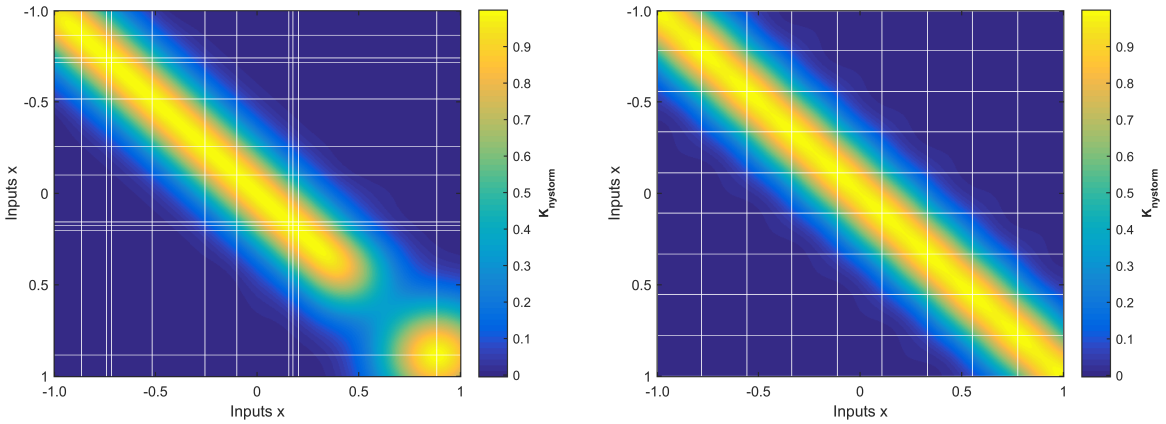
```
% Function to evaluate the approximate gram matrix
function gramMatrix = evaluateNystromGramMatrix(
   covarianceFunction, theta, xm, x1, x2)

        gramMatrixX1M = evaluateGramMatrix(covarianceFunction,
           theta, x1, xm);
        gramMatrixMM = evaluateGramMatrix(covarianceFunction,
           theta, xm, xm);
        gramMatrixMX2 = evaluateGramMatrix(covarianceFunction,
           theta, xm, x2);

        gramMatrix = gramMatrixX1M*inv(gramMatrixMM)*
           gramMatrixMX2;
end
% Test points
nStar = 1000;
xStar = linspace(-1, 1, nStar);

% Inducing points
nInducing = 20;
inducingPoints = linspace(-1, 1, nInducing);

approximateGramMatrix = evaluateNystromGramMatrix(
   covarianceFunction, theta, inducingPoints, xStar, xStar);
```

Matlab Code 3.1: Gram Matrix using Nyström Approximation

Figure 3.1(a) is an approximate Gram matrix computation using Nyström approximation of the matrix in figure 2.1(a) at the input points $\boldsymbol{X}^* = \{[0 : 0.02 : 1]\}$. The inducing points $\boldsymbol{X}^m$ are chosen randomly from the set of input points and their location is denoted by white lines. We can observe that if the gap between inducing points increases, the accuracy of the Gram matrix degrades (eg. at $x \sim 0.5$). Figure 3.1(b) is again an

Figure 3.1(a)

approximated Gram matrix using Nyström approximation of the matrix in figure 2.1(a). This time the equally spaced inducing points are chosen in the range of $\boldsymbol{X}^*$. Notice the significant improvement in the Gram matrix upon different set of inducing inputs.



(a) Approximated Gram matrix using Nyström approximation for a SE Kernel with ($\boldsymbol{\theta} = [1, 0.2]$) (figure 2.1(a)) at the input points $\boldsymbol{X}^* = \{[0 : 0.02 : 1]\}$. The inducing points were chosen randomly, the white lines denote the location of inducing points.

(b) Approximated Gram matrix using Nyström approximation for a SE Kernel with ($\boldsymbol{\theta} = [1, 0.2]$) (figure 2.1(a)) at the input points $\boldsymbol{X}^* = \{[0 : 0.02 : 1]\}$. The white lines denote the location of inducing points, the inducing points are uniformly distributed. Notice the significant improvement in Gram matrix due to different inducing inputs

Figure 3.1: Approximate Gram matrix for a SE kernel using Nyström approximation.

Later, [Snelson 2006] proposed the Fully Independent Training Conditional (FITC) approach which corrects the diagonal terms of the Gram matrix and improves the prediction capabilities (equation 3.2).

$$\boldsymbol{K_{FITC}}(\boldsymbol{X}, \boldsymbol{X}) = diag[\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) - \boldsymbol{K_{Nystorm}}(\boldsymbol{X}, \boldsymbol{X})] + \boldsymbol{K_{Nystorm}}(\boldsymbol{X}, \boldsymbol{X}) \tag{3.2}$$

Note that calculating $diag(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}))$ is an $\mathcal{O}(N)$ operation and thus does not significantly impact the time taken.

Posterior

The posterior distribution for the approximate prior can be derived similarly to section 2.3 [Williams 2001] and is a Gaussian. The predictive mean, and predictive variance are written as equation 3.3 and equation 3.4. Here, $\boldsymbol{K_{Approximate}}(\boldsymbol{X}, \boldsymbol{X'})$ can be the approximated Gram matrix either from the Nyström approximation (equation 3.1) or the FITC approximation (equation 3.2).

$$\mathbf{E}[f_{approximate}(\boldsymbol{x_*})] = \boldsymbol{k}_{\boldsymbol{X}\boldsymbol{x_*}}^T(\boldsymbol{K_{Approximate}}(\boldsymbol{X}, \boldsymbol{X'}) + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{y} \tag{3.3}$$

$$Cov[f_{approximate}(\boldsymbol{x_*})] = k_{x_*x_*} - \boldsymbol{k}_{\boldsymbol{X}\boldsymbol{x_*}}^T(\boldsymbol{K_{Approximate}}(\boldsymbol{X}, \boldsymbol{X}) + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{k}_{\boldsymbol{X}\boldsymbol{x_*}} \tag{3.4}$$

By approximating the $\boldsymbol{K(X, X)}$ using inducing points, we have effectively changed the GP prior. This means that $\boldsymbol{X^m}$ have also become the hyper-parameters of our GP. Hence we should fine-tune locations of $\boldsymbol{X^m}$ and the hyper-parameters $\boldsymbol{\theta}$ to obtain a good prediction of our data. The marginal likelihood for the approximate prior (equation 3.5) can be written similarly as equation 2.24.

$$\Pr[\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{X^m}, \boldsymbol{\theta}] = \mathcal{N}(0, \boldsymbol{K_{Approximate}(X, X')} + \sigma_n^2 \boldsymbol{I}) \tag{3.5}$$

The approximate marginal likelihood (equation 3.5) has more parameters ($\boldsymbol{X^m}$ and $\boldsymbol{\theta}$) to fine-tune when compared to the old, exact marginal likelihood (equation 2.25). Hence, the maximization of the marginal likelihood in equation 3.5 is prone to over-fitting especially when the number of inducing inputs is large. This means that if we keep on increasing the number of inducing points, a time will come when we will tend to decrease the accuracy of our predictions on the test data-set, due to over-fitting. The variational approximation (detailed next) approach overcomes this issue of over-fitting by adding a regularization term penalizing over-fitting.

Over-fitting

3.2.2 Variational Approximation

The variational approximation does not attempt to approximate the Gram matrix. Instead, it assumes a probability distribution $q(f)$ of the true posterior distribution $p(f \mid y)$ and minimizes the distance between the two [Titsias 2009].

The $q(f)$ is written in terms of inducing points ($\boldsymbol{X^m}$) and the KL divergence $KL(q||p)$[1] is minimized between the variational distribution q and true distribution p. When we minimize the KL divergence, we are making the assumed distribution closer to true distribution and hence improving the values of ($\boldsymbol{X^m}$) and $\boldsymbol{\theta}$. This minimization of KL divergence is equivalently expressed as the maximization of equation 3.6

KL divergence

$$F_V = log(\mathcal{N}[0, \boldsymbol{K_{Nystorm}(X, X)} + \sigma_n^2 \boldsymbol{I}]) - \frac{1}{2\sigma_n^2} Tr(\boldsymbol{K_{XX}} - \boldsymbol{K_{Nystorm}(X, X)}) \tag{3.6}$$

Notice the similarity between equation 3.6 and 3.5. The novelty of the above objective function is that it contains a regularization term: $\frac{1}{2\sigma_n^2} Tr(\boldsymbol{K_{XX}} - \boldsymbol{K_{Nystorm}(X, X)})$. Thus, F_V attempts to maximize the marginal likelihood as derived for Nyström approximation and simultaneously minimizes the trace. When the regularization term tends to zero ($\boldsymbol{K_{XX}} = \boldsymbol{K_{Nystorm}(X, X)}$), it means that the inducing variables can exactly reproduce the true Gram Matrix.

[1] KL divergence is a measure of distance between two probability distributions

Posterior

The posterior distribution for variational inference approximation is the same as the one derived for Nyström approximation (equations 3.3 and 3.4), for a detailed derivation refer to [Titsias 2009]. The difference between Nyström approximation and variational approximation is the addition of regularization parameter ($\frac{1}{2\sigma_n^2} Tr(\boldsymbol{K_{XX}} - \boldsymbol{K_{Nystorm}(X, X)})$) while optimizing $\boldsymbol{X^m}$ and $\boldsymbol{\theta}$, this additional term reduces over-fitting.

3.2.3 Experiments

In this section, we conduct experiments on a toy data-set to observe the accuracy of Nyström approximation for varying number and location of inducing points. The basic toolbox used for this section is GPML provided with [Rasmussen 2005] on MATLAB 2014b. All experiments were performed on an Intel quad-core processor with 4Gb RAM.

10 fold CV

10-fold Cross Validation (CV) will be used to assess the performance of the prediction. CV is a technique where the data-set is partitioned as the test set and training set. A model is learned using the training set and Root Mean Square Error (RMSE) is calculated between the prediction and test set as a measure of accuracy. In the 10-fold version of CV, the data-set will be randomly partitioned into 10 subsets containing an equal number of points. Of the 10 subsets, a single subset is retained as the test data-set, and the remaining 9 (10 - 1) subsets are used as training data. The cross-validation process is then repeated 10 times (the folds), with each of the k subsets used exactly once as the validation data.

The toy data-set ($\mathcal{D}_3$) was generated at 1000 input points $\boldsymbol{X} = \{[-1 : 0.002 : 1]\}$ by sampling a random function from a GP[2] with zero mean, SE covariance function ($\boldsymbol{\theta} = [1, 0.1]$) and noise $\sigma_n = 0.3$. Code 3.2 generates the data-set $\mathcal{D}_3$.

```
%% Generating the toy Dataset 3
theta = [1, 0.1];
noiseHyp = 0.3;

nData = 1000;
xData = linspace(-1, 1, nStar)';

kSENoiseFree = evaluateGramMatrix(covarianceFunction, theta,
   xStar, xStar);
kSENoisy = kSENoiseFree + exp(2*noiseHyp)*eye(nStar);

L = chol(kSENoisy);
yData = L'*rand(nStar, 1);%
```

[2] ($\Pr[\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}, \sigma_n] = GP(0, k_{SE}(\boldsymbol{x}, \boldsymbol{x'}, \theta = [1, 0.1]) + (0.3)^2 \boldsymbol{I})$

```
plot(xData, yData, '.')
```

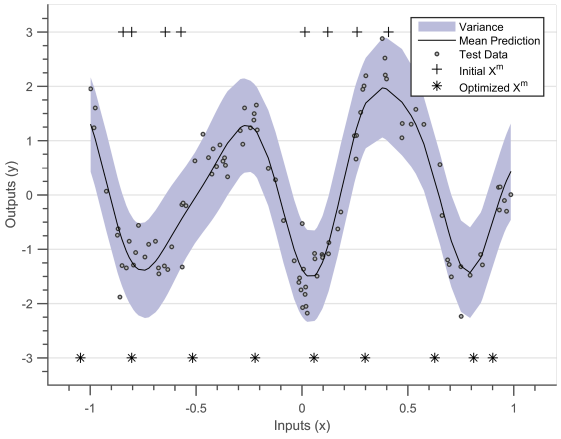
Matlab Code 3.2: Code for toy data-set 3

Figure 3.2(a) is the prediction of the GP obtained after Nyström approximation using 10 inducing points. The solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from the mean. The points denoted by '+' sign are initial locations of inducing points, while the points denoted by '*' sign are locations of inducing points after optimization. The points denoted by '.' are the test points for this fold of the 10-fold CV. *Figure 3.2(a)*

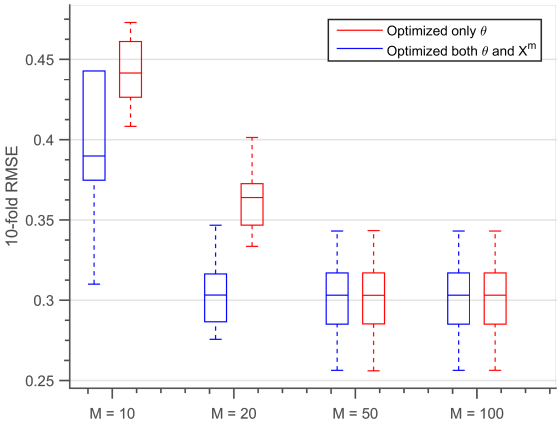
Notice the inducing points, while initially randomly distributed are later uniformly distributed due to optimization of marginal likelihood. In the case of dataset $\mathcal{D}_3$, since the training points are randomly distributed, uniformly distributed inducing inputs are better approximations of the Gram matrix. In cases where training data-set tends to be dense in one region and sparse in another region, lesser inducing points get allocated at the dense region and more get allocated at the sparse region. This happens because the information contained in a dense cluster of data-points can be approximated by a fewer data-points (points in a neighbourhood are similar section 2.2.3) [Snelson 2006].

Figure 3.2(b) are 10-fold RMSE box-plots for varying number of inducing points. The box-plots in red are cases when only the hyper-parameters were optimized while inducing inputs were distributed randomly. The box-plots in blue are the cases when both locations of inducing points and hyper-parameters are optimized. The accuracy of prediction improves with increasing number of inducing points. Accuracy is better when both locations of inducing points and hyper-parameters are optimized. Since the noise in the generated toy data-set is $\sigma_n = 0.3$, it is the best achievable RMSE value. Models constructed when optimizing both $\boldsymbol{\theta}, \boldsymbol{X^m}$ reach this RMSE limit for $M = 20$. After $M = 50$, accuracy is similar for both the optimization routines. As a thumb rule, if $M = \frac{N}{10}$, then randomly distributing the inducing points and optimizing $\boldsymbol{\theta}$ will be sufficient to give a good prediction [Cao 2013], while if $M = \frac{N}{50}$ then both inducing points and hyper-parameters should be optimized. *Figure 3.2(b)*

Global low-rank approximations are best suited for the case of spread out Gram matrices (example high length-scale SE priors). We have seen three types of low-rank approximation algorithms in this section. While Nyström and FITC approximations are the simplest method to approximate Gram matrix, finding optimal locations of the inducing points can often lead to over-fitting. We then look at variational approximation procedure which adds a regularization term while finding inducing points, thereby, penalizing over-fitting. The lower computational cost due to sparse approximations, scales Sparse GPs to the training set of sizes $N \sim \mathcal{O}(5 \times 10^5)$. [Gal 2014] propose a distributed architecture for scaling variational Sparse GPs. In the next section we look at how to approximate Gram matrices using mixture of experts, this enables us to massively scale GPs to sizes more



(a) Posterior between a Nyström approximated SE prior with 10 inducing inputs and training data. The solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from the mean. The points denoted by '+' sign are initial locations of inducing points, while the points denoted by '*' sign are locations of inducing points after optimization.



(b) 10-fold RMSE box-plots for varying number of inducing points. The box-plots in red are cases when only the hyper-parameters $\boldsymbol{\theta}$ were optimized while inducing inputs were distributed randomly. The box-plots in blue are the cases when both locations of inducing points X^m and hyper-parameters $\boldsymbol{\theta}$ are optimized.

Figure 3.2: Results of Nyström Approximation on a toy data-set of size $N = 1000$

than $N > \mathcal{O}(10^6)$ by exploiting distributed architecture.

3.3 Distributed Gaussian Process

Netflix prize

In the year 2006 Netflix organized a machine learning competition, to create a recommendation algorithm for its users. Teams from all over the world competed in this competition to make the best video recommendation algorithm. As the competition progressed, teams started figuring out that the accuracy increases upon combining algorithms developed by multiple teams. The winner and the runner-up were model ensembles of hundreds of algorithms. Creating model ensembles (also called mixture of experts) is now a standard practice in many learning competitions [Bauer 1998].

Mixture of experts in GPs use bagging, where subsets of data are generated, individual GPs are trained on these subsets, and their results are finally combined [Chen 2009]. If the data-set is partitioned into $N_{experts}$ subsets, such as $\mathcal{D}^{(i)} = \boldsymbol{X}^{(i)}, \boldsymbol{y}^{(i)}, i \in 1, \ldots N_{experts}$. Then each subset of data $\mathcal{D}^{(i)}$ learns an individual GP model, which are to be combined together to give the final predictions . Due to this independent learning, choosing hyper-parameters, and calculating predictions become easily parallelizable and indifferent to the computational infrastructure.

Initially, these model types were used to highlight local features in the data [Rasmussen 2002]. Later, [Ng 2014] proposed to use the parallel feature to speed up predictions. Instead of learning a different GP for each subset, we can also tie all the different experts using one single set of hyper-parameters. This is equivalent to assuming one single GP for the whole data-set, thus the experts are same but they each have access to different data subsets. This tying of experts greatly reduces the number of hyper-parameters, acts as a regularization, and inhibits over-fitting. Equation 3.7 denotes an independent GP prior for each expert $\mathcal{D}^{(i)}$ such that the hyper-parameters $\boldsymbol{\theta}$ and σ_n are same for all experts.

$$\Pr[\boldsymbol{y}^{(i)} \mid \boldsymbol{X}^{(i)}, \boldsymbol{\theta}, \sigma_n] = GP(0, \boldsymbol{K}(\boldsymbol{X}^{(i)}, \boldsymbol{X}^{(i)'}, \boldsymbol{\theta}) + \sigma_n^2 \boldsymbol{I}) \tag{3.7}$$

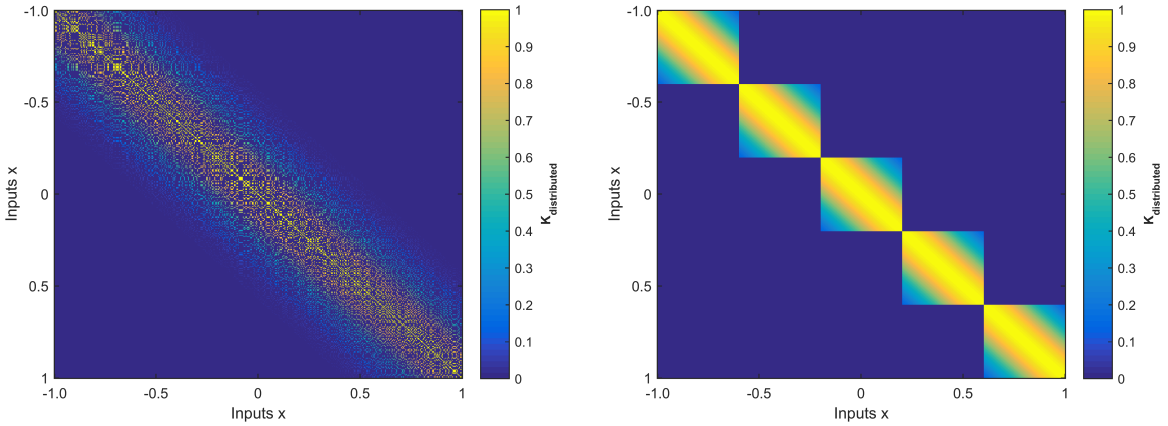
The sample code 3.3 presents a method to randomly clusterize data-set across 6 experts.

```
nStar = 500;
numberOfExperts = 6; % number of experts
sizeOfExperts = floor(nStar/numberOfExperts);

idxRandom = randperm(nStar); % distributing the integers 1:500
   randomly

% Clustering points randomly into experts
for i=1:numberOfExperts
    if i<numberOfExperts
        idxExpert{i} = idxRandom((i-1)*sizeOfExperts+1 : i*
           sizeOfExperts);
    elseif i==numberOfExperts
        idxExpert{i} = idxRandom((i-1)*sizeOfExperts+1 : nStar);
    end
end

% Points in the ith expert
xIthExpert = xData(idxExpert{i});
yIthExpert = yData(idxExpert{i});
```

Matlab Code 3.3: Randomly clustering points into experts

Figure 3.3(a) is an approximate Gram matrix using the above approximation, for a SE Kernel with ($\boldsymbol{\theta} = [1, 0.2]$) at 500 input points $\boldsymbol{X}^* = \{[0 : 0.02 : 1]\}$. 5 experts each having 100 points are chosen, points in the experts are distributed randomly, this gives the approximate Gram matrix the scattered shape. Figure 3.3(b) is an approximate Gram matrix using same approximation, and uniformly distributed experts. 5 experts each

having 100 points are chosen, The first expert has first set of 100 points, the second expert has the second set of 100 points and so on. Both the figures are trying to approximate the Gram matrix of the matrix in figure 2.1(a). The Gram matrix with randomly chosen experts has a more global nature but lacks many high variance regions. The Gram matrix for uniformly chosen experts retains more local features. Inversion of this Gram matrix is an operation of complexity $\mathcal{O}(N_{experts}P^3)$, where P is the number of points in an expert.



(a) Approximated Gram matrix using Distributed GPs approximation for a SE Kernel with ($\boldsymbol{\theta} = [1, 0.2]$) (figure 2.1(a)) at the input points $\boldsymbol{X}^* = \{[0 : 0.02 : 1]\}$. Points in the experts are distributed randomly, this gives the approximate Gram matrix scattered shape.

(b) Approximated Gram matrix using Distributed GPs approximation for a SE Kernel with ($\boldsymbol{\theta} = [1, 0.2]$) (figure 2.1(a)) at the input points $\boldsymbol{X}^* = \{[0 : 0.02 : 1]\}$. Points in the experts are distributed uniformly. We can observe that covariance across experts goes to zero.

Figure 3.3: Approximate Gram matrix for a SE kernel using mixture of experts.

Since the experts have different data subsets, we can construct a posterior distribution for each expert (equations 3.8 and 3.9). In the following equations $m^{(i)}(\boldsymbol{x}_*)$ and $\sigma^{(i)}(\boldsymbol{x}_*)$ are the mean and covariance predictions from expert i at point $\boldsymbol{x}_*$.

$$m^{(i)}(y(\boldsymbol{x}_*)) = \boldsymbol{k}^T_{\boldsymbol{X}^{(i)}\boldsymbol{x}_*}(\boldsymbol{K}_{\boldsymbol{X}^{(i)},\boldsymbol{X}^{(i)\prime}} + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{y}^{(i)} \tag{3.8}$$

$$\sigma^{(i)}(y(\boldsymbol{x}_*)) = k_{x_*x_*} - \boldsymbol{k}^T_{\boldsymbol{X}^{(i)}\boldsymbol{x}_*}(\boldsymbol{K}_{\boldsymbol{X}^{(i)},\boldsymbol{X}^{(i)\prime}} + \sigma_n^2\boldsymbol{I})^{-1}\boldsymbol{k}_{\boldsymbol{X}^{(i)}\boldsymbol{x}_*} \tag{3.9}$$

3.3.1 Combining experts

There are several methods in the literature, which combine the posterior predictions of individual experts, and give a global estimate. The Product of Experts (POE) model, uses the independence assumption between experts, and multiplies the individual posterior

distribution[3], but these predictions tend to be overconfident. Another method called the generalized Product of Experts (gPOE), assigns a participation factor to each expert, this is based on the amount of uncertainty in prediction (more confident experts have higher say in prediction)[Cao 2014]. The Bayesian Committee Machine (BCM) imposes the independence assumption between each expert pair using Bayes Rule, but can result in bad predictions when leaving data regime [Tresp 2000].

rBCM

This thesis will use robust Bayesian Committee Machine (rBCM) model to combine the posterior distributions of experts [Deisenroth 2015]. The rBCM model is an aggregation of all the above three mentioned methods, it combines the confidence weighting parameter of gPOE, with the Bayesian formulation in BCM technique, to generate the following posterior distributions.

$$m(y(\boldsymbol{x_*})) = (Cov(y(\boldsymbol{x_*})))^{-2} \sum_i \beta_i (\sigma^{(i)})^{-2} m^{(i)}(\boldsymbol{x_*}) \tag{3.10}$$

$$Cov(y(\boldsymbol{x_*}))^{-2} = \sum_i \beta_i \sigma^{(i)}(y(\boldsymbol{x_*})) + (1 - \sum_i \beta_i)(k_{x_* x_*})^{-2} \tag{3.11}$$

$k_{x_* x_*}$ is the auto-covariance of the prior at prediction point $\boldsymbol{x_*}$. β_k determines the influence of experts on the final predictions [Cao 2014] and is given as $\beta_i = \frac{1}{2}(\log k^2_{x_* x_*} - \log(\sigma^{(i)}(y(\boldsymbol{x_*})))^2)$. Experts which are very confident in their predictions at $\boldsymbol{x_*}$ will tend to have low $\sigma^{(i)}$ thereby leading to a higher influence factor β_i.

Marginal Likelihood

Due to the independence assumption, the marginal likelihood can be written as a sum of individual likelihoods and then can be optimized to find the best-fit hyper-parameters. By approximating the $\boldsymbol{K(X, X)}$ in terms of $\mathcal{D}^{(i)}$ and $N_{experts}$ we have again changed the GP prior. This means that the number of experts $N_{experts}$ and clustering of points also impact the prediction capabilities of GP. The below equation 3.12 describes the formulation for marginal likelihood.

$$\log \Pr[\boldsymbol{y} \mid \boldsymbol{X}, \mathcal{D}, \boldsymbol{\theta}] \approx \sum_{k=1}^{N_{experts}} \log \Pr[\boldsymbol{y^{(i)}} \mid \boldsymbol{X^{(i)}}, \boldsymbol{\theta}] \tag{3.12}$$

The sample code 3.4 defines the function 'logMarginalLikelihoodDGP' which calculates the new log marginal likelihood (equation 3.12) using the experts as defined in code 3.3. The optimal hyper-parameters can be obtained after maximizing this function with respect to hyper-parameters.

[3] $\Pr[y(\boldsymbol{x_*}) \mid \boldsymbol{x_*}, \mathcal{D}, \boldsymbol{\theta}] \propto \prod \Pr[\boldsymbol{y^{(i)}} \mid \boldsymbol{X^{(i)}}, \mathcal{D}^{(i)}, \boldsymbol{\theta}]$

```
function logMarginalLikelihoodDGP = logMarginalLikelihoodDGP(
   theta, covarianceFunction, x, y, idxExpert)

        numberOfExperts = length(idxExpert);

        % Adding log marginal likelihoods of independent experts
        logMarginalLikelihoodDGP = 0;
        for i =1:numberOfExperts
                % Points in the iTH expert
                xOfIthExpert = x(idxExpert{i});
                yOfIthExpert = y(idxExpert{i});

                logMarginalLikelihoodDGP =
                   logMarginalLikelihoodDGP + ...
                            logMarginalLikelihood(theta,
                               covarianceFunction, xOfIthExpert,
                                yOfIthExpert);
        end

end

% Hyper parameters
noiseHyp = 0.1;
theta = [1, 0.2, noiseTerm];

logMarginalLikelihoodDGP(theta, covarianceFunction, xData, yData
   , idxExpert)
```

Matlab Code 3.4: log Marginal Likelihood for Distributed GPs

k-means

Maximizing the above log-marginal likelihood will give the optimal values of hyper-parameters. Points in the experts can be distributed either randomly, or using a clustering scheme (eg. k-means clustering[4]) for stationary kernels k-means clustering should be preferred. The k-means algorithm clusters points based on a measure of distance, points in separate clusters are far away from each other when compared to points in the same cluster. This means that the covariance (for stationary kernels) between separate clusters is significantly lower when compared to points in same cluster. Hence, the cross-covariance across separate clusters can be more easily assumed to be zero.

[4]k-means algorithm clusters close by points in one cluster. The notion of closeness is defined by some measure of distance

3.3.2 Experiments

We again conduct experiments on a toy data-set, this time to observe the accuracy of the Distributed GPs approximation for a varying number of experts.

Again the 10-fold Cross Validation (CV) will be used to assess the performance of the prediction. The same toy data-set ($\mathcal{D}_3$) as used in section 3.2.3 was used to perform the experiments in this section (1000 data-points from $\Pr[\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}, \sigma_n] = GP(0, \boldsymbol{K_{SE}}(\boldsymbol{X}, \boldsymbol{X'}, \boldsymbol{\theta} = [\boldsymbol{1}, \boldsymbol{0.1}]) + (0.3)^2\boldsymbol{I})$.

Figure 3.4(a) is the prediction of the GP obtained after distributed approximation using 9 experts and k-means clustering. The solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from the mean. The colored points denoted by '*' at the bottom show how different points are distributed across experts, similar colored points belong to one expert. The data denoted by '.' is the test data for one fold of the 10-fold CV.

Figure 3.4(a)

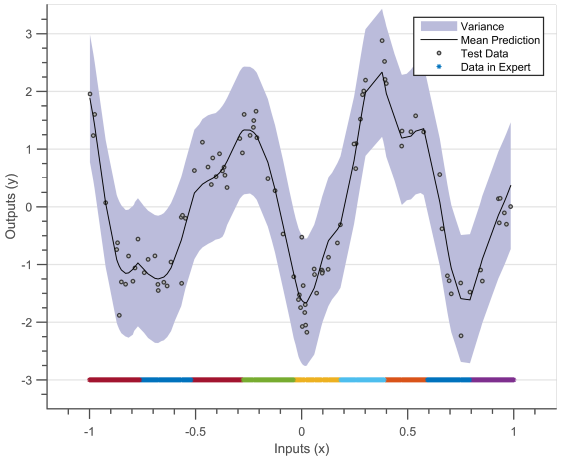
The points across experts are uniformly distributed, as can be observed by the coloring scheme. Since the training points are almost uniformly distributed, the k-means algorithm will cluster the points uniformly. The training and the test data-set for figures 3.4(a) and 3.2(a) are kept intentionally same. We can thus compare the interpolating properties of both Sparse GPs approximation (figure 3.4(a)) and Distributed GPs (figure 3.2(a)). The Nyström approximation has a global smooth shape while the Distributed GPs approximation retains the local features of the data-set.

Figure 3.4(b) are 10-fold RMSE box-plots for different values of P. The box-plots in red are cases when the points are distributed using k-means clustering. The box-plots in blue are the cases when points are distributed randomly. The accuracy of prediction improves with increasing number of points in an expert. Note, the noise in the generated toy data-set is $\sigma_n = 0.3$, it is the best achievable RMSE value. Accuracy is slightly better when experts are distributed using k-means clustering, both being very close to the 0.3 RMSE limit. As a thumb-rule, setting $P = \frac{N}{100}$, and optimizing the hyper-parameters ($\boldsymbol{\theta}$) is a good enough approximation.

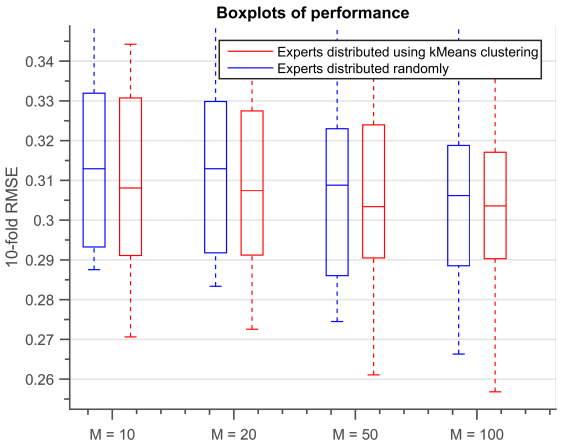
Figure 3.4(b)

3.4 Summary and discussion

The calculation of posterior in GPs becomes computationally intractable for large data-sets. Calculating the precision matrix is an operation of computational complexity $\mathcal{O}(N^3)$, putting a limit of $N \sim 10^4$ data-points for model building. This chapter describes the state-of-the-art for scaling up GPs for regression tasks. There exist two methods to scale up GPs, the first called Sparse GPs, they use a set of inducing inputs to reduce the computational cost of calculating the precision matrix. The second called Distributed GPs they divide the



(a) Prediction of the GP obtained after distributed approximation using 9 experts and k-means clustering. The solid black line defines the mean function, blue region defines 95% confidence interval (2σ) distance away from the mean. The colored points in the points denoted by '*' at the bottom show how different points are distributed across experts, similar colored points belong to one expert. The data denoted by '.' is the test data for one fold of the 10-fold CV.



(b) 10-fold RMSE box-plots for different number of points across. The box-plots in red are cases when only the hyper-parameters when the points are distributed using k-means clustering. The box-plots in blue are the cases when points are distributed randomly. The accuracy of prediction improves with increasing number of points in an expert

Figure 3.4: Results of Distributed GPs approximation on a toy data-set of size $N = 1000$

data-set into smaller subsets called experts, distributing the model building into several batches.

Conclusions

Sparse methods use Nyström approximation rewriting the Gram matrix, thereby reducing the computational complexity to $\mathcal{O}(NM^2)$ (M $\ll$ N), M being the number of inducing points. Through experiments on a toy data-set, it can be shown that we can set $M \sim N/10$ when inducing points are randomly distributed, and $M \sim N/50$ when the locations of inducing points are optimized. This approximation pushes the limit of GP Regression to $N \sim 10^6$ data-points. Distributed GPs distribute the GP Regression tasks into several batches, thereby reducing the computational complexity to $\mathcal{O}(NP^3)$ (P $\ll$ N), P being the number of points in an expert. Through experiments on a toy data-set we demonstrate that $P \sim N/100$ does not effects the regression task significantly. In fact we can further reduce P if we enable repetition of points between experts. This enables to scale GPs to any number of data-points.

There are several reasons why GPs should be preferred to perform regression tasks. GPs provide a probabilistic framework to define a family of functions, while the covariance functions allows to incorporate a wide range of assumptions (part II). GPs are computationally tractable, given a covariance function and observations, the predictive distribution

can be calculated exactly. By providing a closed form expression of marginal likelihood GPs provide a powerful method to automatically select hyper-parameters. Although GPs suffer in presence of large data-sets, there exist several approximate methods to scale GPs to millions of data-points.

During the detailed design phase of an aircraft design cycle, high-fidelity code is used to explore the design space. This code is costly, mostly due to fine granularity of the meshes used. If we wish to use a surrogate model to explore the design space, then that surrogate model should be able to handle the large number of mesh points. Thankfully, due to the methods described in the current chapter we can scale GP regression to $N \sim 10^6$. We will demonstrate this capability by running a GP regression on millions of data-points in a CFD mesh (section 5.3.5). This surrogate model was used in a recent Airbus flight test campaign to compare pressures predicted from a high-fidelity CFD computation to pressures measured on the wing, in real time.

Part II

Incorporating structure in Gaussian Process Regression

Chapter 4

Basic Covariance Functions

Résumé

La partie II de la thèse montre comment incorporer les informations *a-priori* dans la construction de modèles GP, en choisissant les différents types de fonctions de covariance. Bien que les formes fonctionnelles de covariance discutées ici soient généralement utilisées dans la communauté d'apprentissage automatique, elles ne sont pas souvent utilisées pour construire des modèles d'ingénierie. La contribution originale du chapitre 4 et du chapitre 5 est l'application de ces noyaux pour construire des modèles d'ingénierie (en mécanique des fluides et en structure). Cette partie est fortement inspirée des travaux de [Duvenaud 2014, Wilson 2014, Lloyd 2014, Durrande 2001].

La section 4.2 définit quelques propriétés importantes des fonctions de covariance. La section 4.3 détaille quelques propriétés de noyaux non-stationnaires et discute dans quelle mesure la Régression Linéaire Bayésienne peut être efficacement vue comme une régression GP avec la fonction de covariance linéaire. La section 4.4 décrit les noyaux stationnaires, utilisant le *théorème de Bochner* nous pouvons représenter un noyau stationnaire par sa transformée de Fourier, qui augmente l'interprétation des fonctions constitutive de la famille. Pour chaque fonction de covariance, nous essayons de donner une visualisation de la forme des fonctions constitutives.

La contribution principale de ce chapitre consiste à démontrer comment utiliser la régression GP pour détecter automatiquement les paramètres modaux de la dynamique structurelle. À notre connaissance, une telle méthode n'a pas été utilisée dans la littérature existante pour identifier les paramètres modaux. Par l'utilisation des noyaux de 'Spectral Mixture' nous démontrons comment construire des modèles pour des expériences dynamiques structurelles et identifier automatiquement des paramètres dynamiques comme la fréquence modale (section 4.5) [Chiplunkar 2017b]. Il s'agit

d'une étape très préliminaire de l'application de noyaux de 'Spectral Mixture' pour l'identification de systèmes, et des problèmes comme l'identification de la forme du mode ou le taux d'amortissement demeurent dans cet algorithme. Nous souhaitons approfondir cette application dans le futur.

4.1 Introduction

If we assume the mean function as equal to zero, then a GP prior can be completely parametrized by its covariance function. Hence the problem of learning in a GP regression is exactly the problem of finding suitable properties of the covariance function [Rasmussen 2005]. The covariance function consists of two parts: a functional form, (which specifies the shape of functions in the hypothesis space) and a set of hyper-parameters (which define the probability of a function in the hypothesis space). In section 2.4 we have seen how to automatically choose hyper-parameters, part II of this thesis will detail how to choose the functional form.

Part II of the thesis shows how to incorporate prior information of patterns into building GP models, by choosing different types of covariance functions. Chapter 4 shows a few basic kernel types, while chapter 5 shows how to combine these basic kernels together and incorporate more complex patterns. This part is heavily inspired from prior works of [Duvenaud 2014, Wilson 2014, Lloyd 2014, Durrande 2001].

Contribution

Although the functional forms of covariance discussed here are commonly used in the machine learning community, unfortunately they are not often used to build engineering models. The original contribution of chapter 4 and chapter 5 is application of these kernels to build meaningful engineering models[1]. In chapter 4 we build a GP model to automatically identify structural dynamics parameters (section 4.5) [Chiplunkar 2017b], while in chapter 5 we use a combination of basic kernels to identify the onset of non-linear behaviour in physical systems. For example we identify the initiation of flow separation in NACA 0012 airfoil and initiation of plasticity in AL6061 alloy using a statistical criteria for automatic detection of non-linearity (section 5.2.4) [Chiplunkar 2016b]. We finally build a GP model to predict the position of aerodynamic shock in the transonic regime (section 5.3.5) [Chiplunkar 2017a].

The current chapter unfolds as follows; section 4.2 details a few important properties of covariance functions. Section 4.3 gives some insights into non-stationary covariance functions. Section 4.4 describes stationary kernels and their Fourier domain. We then leverage this knowledge to automatically identify the dynamic behaviour of structural experiments (section 4.5).

[1] both in structural and fluid mechanics

4.2 Properties

A kernel is a function that maps any pair of inputs ($\boldsymbol{x_1} \in \mathcal{X}$ and $\boldsymbol{x_2} \in \mathcal{X}$) into a scalar $\mathbb{R}$, the inputs can be scalars, vectors, categorical variables [Villegas García 2013] or even images. The covariance function of a GP is a special type of kernel, which specifies covariance of a pair of random functions $f(\boldsymbol{x_1})$ and $f(\boldsymbol{x_2})$ situated at points $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ (mostly written as a function of $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$).

Measure of distance

Most of the learning algorithms work on distance measures, i.e. if two points are closer then their observations (y) will also tend to be similar. Covariance functions specify this measure of distance in a GP Regression. If two points have a high value of covariance, then they are assumed to be close, and hence will have similar value of outputs (y). Therefore, by defining a covariance function we encode which type of input points will be termed as close, this effectively encodes biases into our family of functions. Biases based on smoothness (section 4.4.1), linearity (section 4.3.1), differentiability (section 4.4.2), etc. can be easily encoded using simple covariance functions.

A covariance function between $f(\boldsymbol{x_1})$ and $f(\boldsymbol{x_2})$ can be written as equation 4.1.

$$k(\boldsymbol{x_1}, \boldsymbol{x_2}) = cov(f(\boldsymbol{x_1}), f(\boldsymbol{x_2})) \tag{4.1}$$

A covariance function $k(\boldsymbol{x_1}, \boldsymbol{x_2})$ is always symmetric, since:

$$\begin{aligned} k(\boldsymbol{x_1}, \boldsymbol{x_2}) &= cov(f(\boldsymbol{x_1}), f(\boldsymbol{x_2})) && (4.2)\\ &= \mathbf{E}[f(\boldsymbol{x_1}) - m(\boldsymbol{x_1}), f(\boldsymbol{x_2}) - m(\boldsymbol{x_2})] && (4.3)\\ &= cov(f(\boldsymbol{x_2}), f(\boldsymbol{x_1})) && (4.4)\\ &= k(\boldsymbol{x_2}, \boldsymbol{x_1}) && (4.5) \end{aligned}$$

$k(\boldsymbol{x_1}, \boldsymbol{x_2})$ corresponds to a covariance function if it is a symmetric Positive Semi Definite (PSD) function [Mercer 1909, Loeve 1978, Durrande 2001]. Consider the variance of a new random vector $T = \sum \alpha_i f(\boldsymbol{x_i})$:

$$\begin{aligned} var(T) &= cov\left(\sum_i \alpha_i f(\boldsymbol{x_i}), \sum_j \alpha_j f(\boldsymbol{x_j})\right) = \sum_i \sum_j \alpha_i \alpha_j cov(f(\boldsymbol{x_i}), f(\boldsymbol{x_j}))\\ &= \sum_i \sum_j \alpha_i \alpha_j k(\boldsymbol{x_i}, \boldsymbol{x_j}) \end{aligned} \tag{4.6}$$

Since a variance is always non-negative, hence:

$$\sum_i \sum_j \alpha_i \alpha_j k(\boldsymbol{x_i}, \boldsymbol{x_j}) \geq 0 \tag{4.7}$$

According to the Mercer's theorem [Mercer 1909] equation 4.7 is a sufficient condition to prove that $k(\boldsymbol{x_i}, \boldsymbol{x_j})$ is a PSD function. The positive definite requirement means that the covariance kernel corresponds to an inner product in some basis space [Bishop 2006]. It is generally difficult to prove if a function is PSD, hence creating new covariance functions is a tough task. Fortunately there already exist a wide variety of covariance functions, this chapter will describe a few basic covariances.

4.3 Non-stationary kernels

Earlier in section 2.2 we have seen the SE kernel which is an example of stationary kernel. Stationary kernels are covariance functions which are purely a function of $\boldsymbol{d} = \boldsymbol{x_i} - \boldsymbol{x_j}$. In this section we list some non-stationary kernels and their properties.

4.3.1 Linear Kernel

The Bayesian linear regression described during section 1.4 can also be seen as a form of GP Regression but with a Linear covariance function. In the Bayesian linear regression we first assume a functional form of the function, in terms of its basis functions ($f(\boldsymbol{x}) = \phi(\boldsymbol{x})^T \boldsymbol{w}$). We then assume a prior distribution of parameters, this is equivalent to assuming a prior distribution over functions. For a function and its prior as defined by equation 4.8.

$$f(\boldsymbol{x_i}) = \phi(\boldsymbol{x_i})^T \boldsymbol{w} \qquad \Pr[\boldsymbol{w}] = \mathcal{N}(0, \boldsymbol{\Sigma_{Prior}}) \tag{4.8}$$

The equivalent prior over functions[2] f can be written as equation 4.9.

$$\Pr[f(\boldsymbol{x})] = GP(0, \phi(\boldsymbol{x})^T \boldsymbol{\Sigma_{Prior}} \phi(\boldsymbol{x'})) \tag{4.9}$$

The above covariance function ($k(\boldsymbol{x_1}, \boldsymbol{x_2}) = \phi(\boldsymbol{x_1})^T \boldsymbol{\Sigma_{Prior}} \phi(\boldsymbol{x_2})^T$) describes a family of functions which are linear combinations of the basis functions ($\phi(\boldsymbol{x})$) [Bishop 2006]. The matrix $\boldsymbol{\Sigma_{Prior}}$, and noise variance σ_n^2, are the hyper-parameters of this GP prior while the $\phi(\boldsymbol{x})$ represents its functional form. Hence a linear basis ($\phi(\boldsymbol{x}) = \{1, \boldsymbol{x}\}^T$) describes a family of linear functions (equation 4.10), while a polynomial basis ($\phi(\boldsymbol{x}) = \{1, \boldsymbol{x}, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^P\}^T$) encodes a family of P^{th} order polynomial basis functions.

$$k_{Lin}(\boldsymbol{x_1}, \boldsymbol{x_2}) = w_0 + w_1 \boldsymbol{x_1}^T \boldsymbol{x_2} \tag{4.10}$$

[2] By the affine property (appendix A.3) of Multivariate Gaussian random variables we have that:

$$X = \mathcal{N}(\mu, \Sigma) \implies AX = \mathcal{N}(A\mu, A\Sigma A')$$

The above equation is the covariance function for a Linear kernel, where the w_0 is the hyper-parameter for the intercept while w_1 is the hyper-parameter for slope. If we assume an independent noise ϵ on the observations, then based on the discussion on noisy GPs (section 2.3.2) the GP prior becomes:

$$\Pr[y(\boldsymbol{x})] = GP(0, w_0 + w_1 \boldsymbol{x_1}^T \boldsymbol{x_2} + \sigma_n^2 \delta_{x_1 x_2}) \tag{4.11}$$

Hence, the intercept (w_0), the slope (w_1) and the noise (σ_n) are the hyper-parameters of the above prior. The above equation is equivalent to performing Bayesian linear regression as discussed in section 1.4. The hyper-parameters can be chosen using marginal likelihood and posterior prediction can be performed based on the discussion of section 2.4.

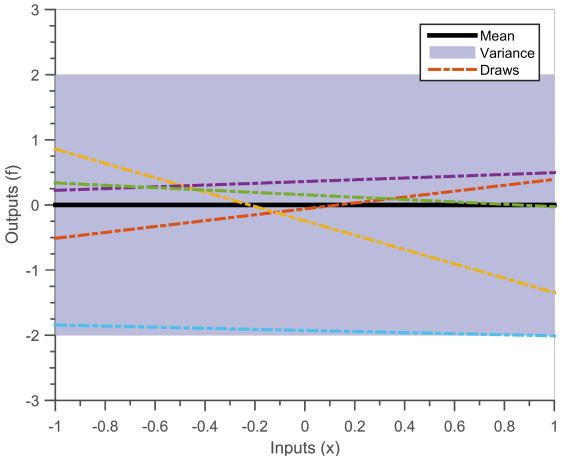
Revisiting Bayesian linear regression Let us revisit the experiment performed in section 1.4 but this time using GP regression and a linear kernel. The toy data-set $\mathcal{D}_1 = \{\boldsymbol{X} = [-0.5, 0.33, 0.66], \boldsymbol{y} = [0, 0.5, 0.5]\}$ will be used again. The prior distribution on parameters $\boldsymbol{\Sigma_{Prior}} = \left(\begin{smallmatrix} w_0 & 0 \\ 0 & w_1 \end{smallmatrix}\right)$ with $w_0 = 1, w_1 = 1$ and prior on noise $\sigma_n = 0.1$ will be the same as used in the earlier experiment.

Figure 4.1(a) shows draws from a GP prior with mean zero and Linear kernel as defined in the above paragraph. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior. Random functions drawn from a linear GP are linear. Figure 4.1(b) show draws from a GP posterior with mean zero and Linear kernel as defined in above paragraph and conditioned on the first data-point $X = -0.5, Y = 0$. The posterior mean passes from the data-point, random functions drawn from a linear GP are linear.

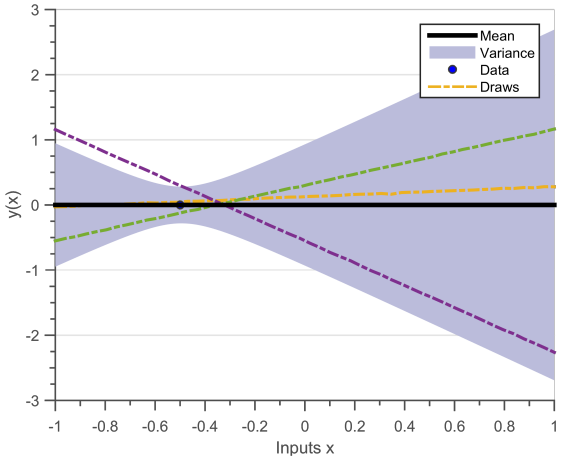
Figure 4.2(a) shows the contours of marginal likelihood with respect to intercept (w_0) and slope (w_1). The marginal likelihood is maximum for $w_0 = 0.2576, w_1 = 0.4584$, this is same as the posterior mean predicted in equation 1.8. This means that the data $\mathcal{D}_1$ has the highest possibility of coming from a data-set defined by a prior having these hyper-parameters. Figure 4.2(b) shows the posterior for same data-set but for the hyper-parameters where marginal likelihood is maximum.

4.3.2 Neural Network Kernel

It can be shown that a Neural Network with infinitely many hidden units and an error activation function ($erf(z)$) tends to a GP with a Neural Network kernel [Neal 2012] (equation

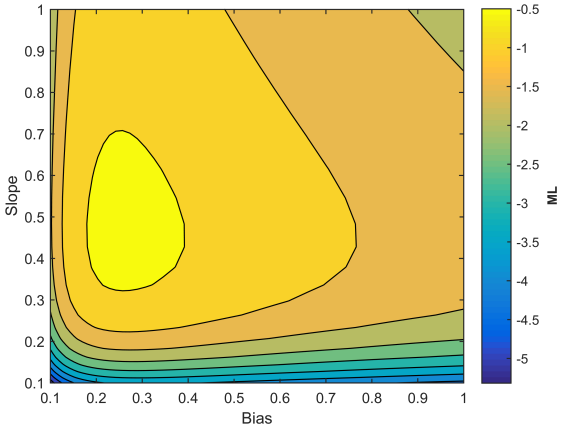


(a) Draws from a GP prior with mean zero and Linear kernel ($k_{Lin}(\boldsymbol{x_1}, \boldsymbol{x_2}) = \phi(\boldsymbol{x_1})^T \boldsymbol{\Sigma_{Prior}} \phi(\boldsymbol{x_2})^T + \sigma_n^2 \delta_{x_1 x_2}$) with $w_0 = 1, w_1 = 1$ and $\sigma_n = 0.1$. Random functions drawn from a linear GP are linear.

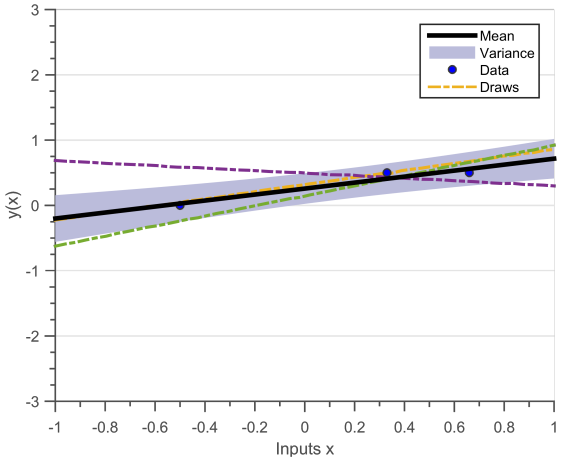


(b) Draws from a GP posterior with mean zero and Linear kernel (figure 4.1(a)) conditioned on the data $x = -0.5, y = 0$. The posterior mean passes from the data-point, random functions drawn from a linear GP are linear

Figure 4.1: Prior and posterior from a GP prior of linear kernel. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior.



(a) Marginal likelihood contours for varying bias and slope parameter. The noise hyper-parameter is ($\sigma_1 = [0.1]$). Marginal likelihood is maximum for $w_0 = 0.2576, w_1 = 0.4584$.



(b) Draws from a GP posterior, conditioned on the data-set $\mathcal{D}_1$ with mean zero and Linear kernel with hyper-parameters that maximize the marginal likelihood.

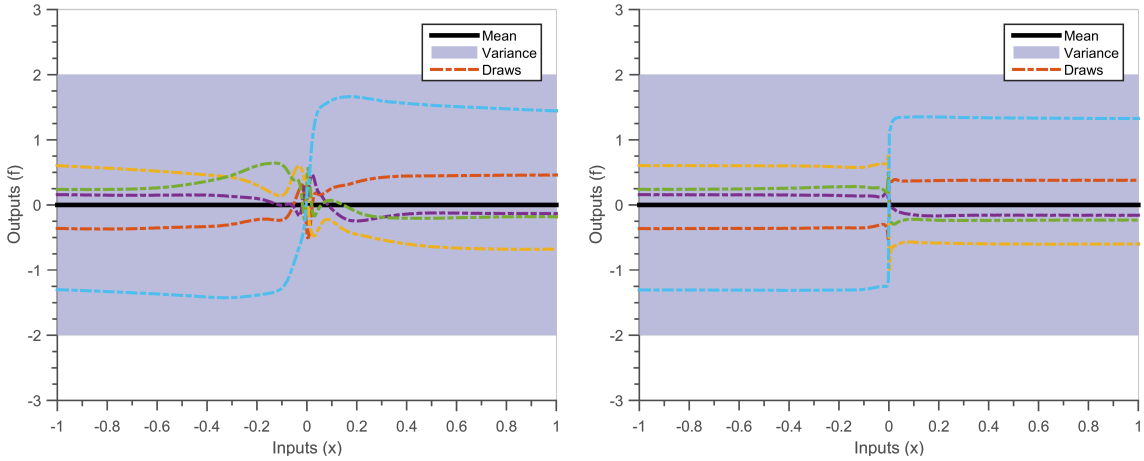
Figure 4.2: Maximizing Marginal Likelihood Linear kernel

4.12).

$$K_{NN}(x_1, x_2, \theta) = \theta_1^2 \frac{2}{\pi} sin^{-1}\left(\frac{2x_1\theta_2 x_2}{\sqrt{(1+2x_1^T\theta_2 x_1)(1+2x_2^T\theta_2 x_2)}}\right) \tag{4.12}$$

The hyper-parameters ($\theta = [\theta_1, \theta_2]$) are; amplitude θ_1 which defines average distance from mean and the length scale θ_2 which define the smoothness of functions. GPs with this covariance function define a space of superimposed sigmoidal functions. Figure 4.3 shows random draws from a Neural Network kernel but with varying value of hyper-parameters. Figure 4.3(b) has a higher value of smoothness hyper-parameter (θ_2) than figure 4.3(a), which makes the constituent functions have stronger slope. Hence, we can use this kernel to approximate the presence of discontinuity in our family of functions.

Hyper-parameters



(a) Draws from a GP prior with mean zero and Neural Network kernel (equation 4.12) with $\theta = [1, 10]$.

(b) Draws from a GP prior with mean zero and Neural Network kernel (equation 4.12) with $\theta = [1, 100]$. Higher value of θ_2 signifies increasing slope

Figure 4.3: Draws from Neural Network kernels having different hyper-parameters. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior.

4.3.3 Constant and Noise kernel

A constant covariance function (equation 4.13) defines a constant function. Here, $\sigma_{constant}$ defines the possible amplitude of the constant function.

$$k_{constant} = \sigma^2_{constant} \tag{4.13}$$

$$k_{noise}(x_1, x_2) = \sigma^2_{noise}\delta_{x_1 x_2} \tag{4.14}$$

On the other hand, equation 4.14 defines a white noise kernel, the σ_{noise} defines the amplitude of noise. $\delta_{x_1 x_2}$ is a Kronecker delta function which is 1 if $x_1 = x_2$ and zero otherwise, this means that observations at two inputs are independent of each other.

4.4 Stationary kernels

Covariance functions which are purely a function of distance $d = (x_1 - x_2)$ are called as stationary functions, Whereas covariance functions which are functions of absolute value of distance $d = |x_1 - x_2|$ are called as isotropic covariance functions. Stationary covariance functions remain unchanged if the points x_1, x_2 are translated. Hence a family of functions defined by stationary kernels will have similar local features throughout the input domain.

Bochner's theorem

The *Bochner's theorem* defines a relationship between a stationary covariance function and its Fourier transform. It states that the Fourier transform of a stationary covariance function exists and is a positive finite measure. If $k(d)$ is a stationary covariance function (equation 4.15) then $S(s)$ is its Fourier transform (equation 4.16), also called the power spectrum or spectral density [Bochner 1959, Stein 1999, Cox 1977]. A positive finite measure in this context means that $S(s)$ is non-negative for all frequencies s, and integral in equation 4.15 is finite.

$$k(d) = \int S(s) e^{2\pi i s^T d} ds \tag{4.15}$$

$$S(s) = \int k(d) e^{-2\pi i s^T s} dd \tag{4.16}$$

Power spectrum

The power spectrum is a more interpretable method of understanding constituent functions in a hypothesis space. A covariance function probabilistically defines a hypothesis space, the power spectrum corresponding to this covariance function tells us the power of the frequencies in this hypothesis space. If a power spectrum has more power at lower frequencies, then the constituent functions in its hypothesis space will be more smooth. Whereas, if the power spectrum has more power at higher frequencies, then the constituent functions in its hypothesis space will be less smooth [Wilson 2014].

4.4.1 Squared Exponential Kernel

We have already encountered the SE kernel in section 2.2.3. It is one of the most widely used kernel because it defines a hypothesis space of infinitely differentiable (infinitely smooth) functions. The SE kernel is Gaussian in shape (equation 4.17), which makes its Fourier transform also Gaussian in shape (equation 4.18).

$$k_{SE}(d,\theta) = \theta^2_{amplitude} exp\left[-\frac{d^2}{2\theta^2_{lengthScale}}\right] \tag{4.17}$$

$$S_{SE}(s,\theta) = \theta^2_{amplitude} exp\left[-2\pi\theta^2_{lengthScale}s^2\right] \tag{4.18}$$

Hyper-parameters

The two hyper-parameters of the SE kernel are the amplitude hyper-parameter ($\theta_{amplitude}$), which defines the amplitude of functions and the length-scale hyper-parameter ($\theta_{lengthScale}$), which defines the smoothness of functions.

Interpretation

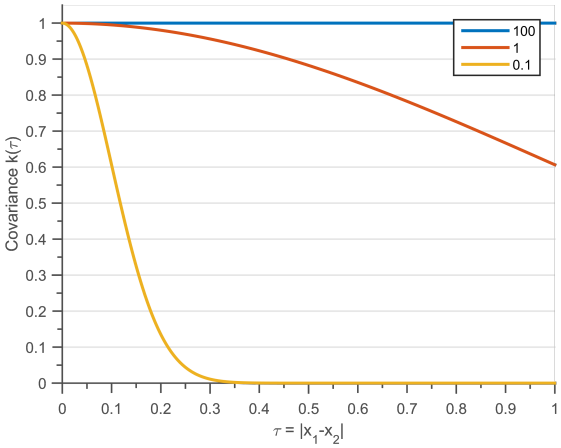
As discussed earlier, the covariance function (k_{SE}) defines the measure of similarity between two input points, if two points have high value of covariance then their output values (y) will be similar. If we increase $\theta_{lengthScale}$ then k_{SE} will also increase, for the same value of d and $\theta_{amplitude}$. This means that, the points for same value of d will become more similar and hence the constituent functions in the hypothesis space will become more smooth. As length-scale increases the constituent functions tend to become smoother. Another way to interpret the effect of length-scale is by observing the power spectrum (S_{SE}). If we increase $\theta_{lengthScale}$ then S_{SE} will start decreasing, for the same values of s and $\theta_{amplitude}$. This means that less power will be allocated to higher frequencies and hence the constituent functions in the hypothesis space will become more smooth (Figure 2.2).

Figure 4.4(a) plots the covariance values of SE kernel (($\theta_{amplitude} = 1$) for various values of length-scales ($\theta_{lengthScale} = [0.1, 1, 100]$)) whereas figure 4.4(b) plots the power spectrum for same kernels. We can see that the power spectrum ($S(s)$) for higher length-scales has less power at higher frequencies, meaning that their corresponding hypothesis spaces will not have faster moving functions.

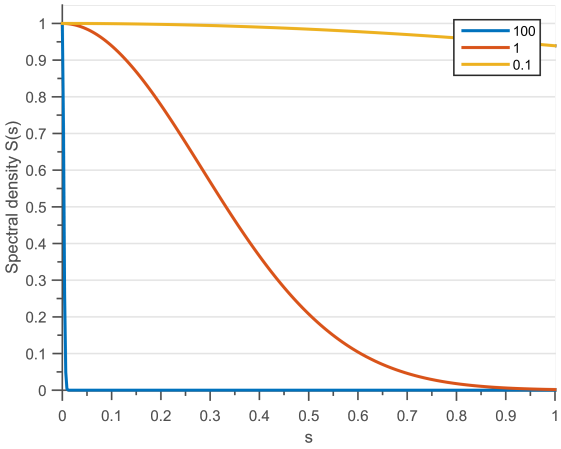
Note, as $\theta_{lengthScale}$ tends to infinity, an SE kernel tends to a constant covariance function (Fourier transform of a constant function is a delta function). Whereas, if $\theta_{lengthScale}$ tends to zero, an SE kernel tends to a white noise kernel (Fourier transform of a delta function is a constant function).

4.4.2 Matérn Kernel

The Matérn kernel is the second most popular kernel after the squared exponential. This kernel is derived using a t-distribution as the power spectrum ($S(s)$) and calculating its



(a) Kernel density for SE kernel with three different length-scales. The hyper-parameters are amplitude ($\theta_{amplitude} = 1$) and length-scale ($\theta_{lengthScale} = [0.1, 1, 100]$). $k(d)$ for $\theta_{lengthScale} = 100$ resembles a constant function.



(b) Power spectrum for SE kernel with three different length-scales. The hyper-parameters are amplitude ($\theta_{amplitude} = 1$) and length-scale ($\theta_{lengthScale} = [0.1, 1, 100]$). $S(s)$ for $\theta_{lengthScale} = 0.1$ resembles a constant function, while for $\theta_{lengthScale} = 100$ $S(s)$ resembles a dirac-delta function.

Figure 4.4: Covariance functions and Power spectrums for SE kernel with three different length-scales

inverse Fourier transform. A t-distribution has more weight on higher-frequencies (when compared to a Gaussian distribution) and hence gives rise to more non-smooth functions.

$$k_{Matern}(\nu, d, \theta) = \theta^2_{amplitude} \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu(d))}}{\theta_{lengthScale}} \right)^{\nu} K_{\nu} \left(\frac{\sqrt{2\nu(d))}}{\theta_{lengthScale}} \right) \tag{4.19}$$

Differentiability

k_{Matern} is the covariance function for a Matérn kernel, ν is a positive parameter which signifies the degrees of freedom in the t-distribution of the power spectrum. $\Gamma(\nu)$ is a Gamma function, while K_ν is a modified Bessel function. The Matérn kernel provides the flexibility to define a hypothesis space of functions with varying degree of differentiability, due to this property they are often used to build machine learning models [Minasny 2005, Cornford 2002]. The degree of differentiability of the functions in the hypothesis space can be set as $[\nu - 1/2]$, i.e. $\nu = 1/2$ (equation 4.20) defines family of non-differentiable but continuous functions, $\nu = 3/2$ (equation 4.21) defines a family of functions differentiable only once and $\nu = 5/2$ (equation 4.22) defines a family of twice differentiable functions. Note, as ν tends to ∞ a t-distribution tends to a Gaussian distribution, similarly as ν

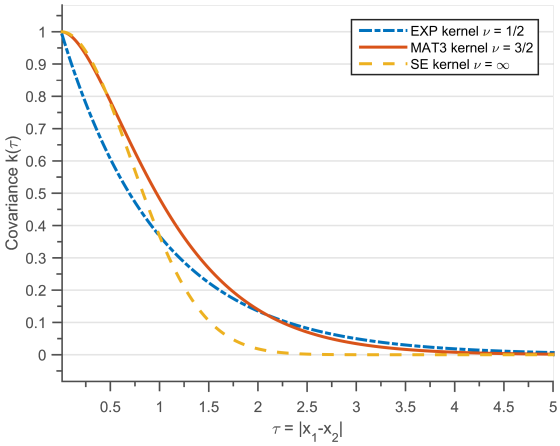
tends to ∞ the Matérn kernel tends to a SE kernel.

$$k_{Matern}(\nu = 1/2, d, \theta) = \theta^2_{amplitude} exp[-\frac{d}{\theta_{lengthScale}}] \tag{4.20}$$

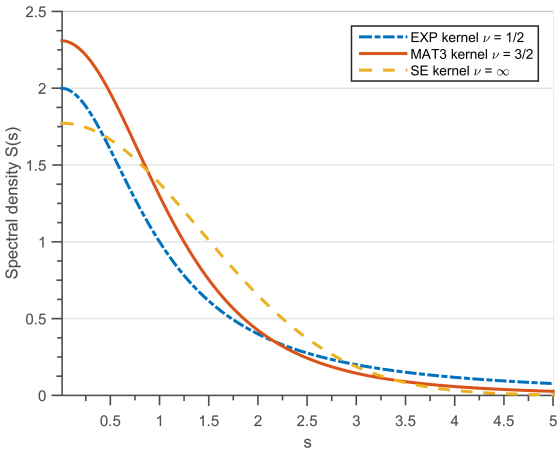
$$k_{Matern}(\nu = 3/2, d, \theta) = \theta^2_{amplitude}(1 + \frac{\sqrt{3}d}{\theta_{lengthScale}})exp[-\frac{\sqrt{3}d}{\theta_{lengthScale}}] \tag{4.21}$$

$$k_{Matern}(\nu = 5/2, d, \theta) = \theta^2_{amplitude}(1 + \frac{\sqrt{5}d}{\theta_{lengthScale}} + \frac{5d^2}{3\theta^2_{lengthScale}})exp[-\frac{\sqrt{5}d}{\theta_{lengthScale}}] \tag{4.22}$$

When $\nu = 1/2$ the Matérn kernel is also called the Ornstein-Uhlenbeck or Exponential kernel, this creates a hypothesis space of non-differentiable continuous functions and was used to explain the Brownian motion [Uhlenbeck 1930]. Figure 4.5(a) plots the covariance values of Exponential ($\nu = 1/2$), Matérn/ ($\nu = 3/2$) and SE kernel ($\theta_{amplitude} = 1$ and $\theta_{lengthScale} = 1$) whereas figure 4.5(b) plots the power spectrum for the same kernels. We can see that the power spectrum ($S(s)$) of the SE kernel has lowest power for higher frequencies followed by Matérn ($\nu = 3/2$) and Exponential kernel, meaning that the SE kernel has more smooth functions in its hypothesis space followed by Matérn ($\nu = 3/2$) and Exponential kernel.



(a) Kernel density for exponential, Matérn ($\nu = 3/2$)and SE kernel. The hyper-parameters are amplitude ($\theta_{amplitude} = 1$) and length-scale ($\theta_{lengthScale} = 1$)



(b) Power spectrum for Exponential, Matérn ($\nu = 3/2$)and SE kernel. The hyper-parameters are amplitude ($\theta_{amplitude} = 1$) and length-scale ($\theta_{lengthScale} = 1$). Exponential and Matérn have more power at higher frequencies when compared to SE kernel.

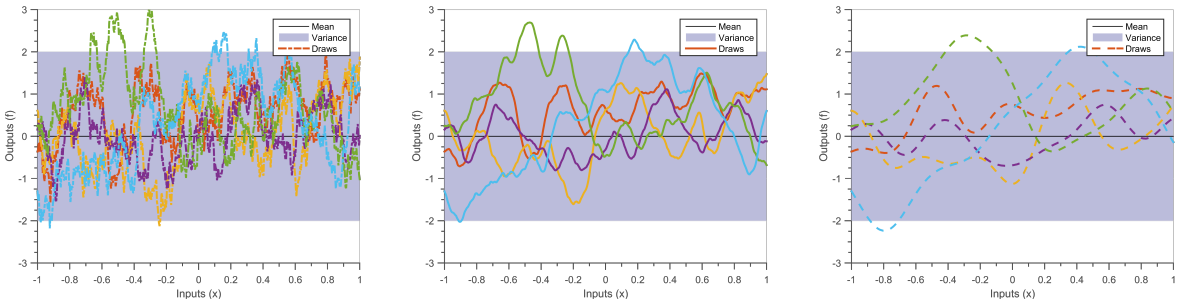
Figure 4.5: Covariance functions and Power spectrums for three different kernels

4.4.3 Experiments

Let us interpolate the data-set $\mathcal{D}_2$ used to choose the hyper-parameters in section 2.3, but this time using three different covariance functions. We will use Exponential kernel, Matérn ($\nu = 1/2$) and SE kernel and compare their interpolations.

We follow the standard framework of GP regression; we first draw random functions from the covariance function to judge the hypothesis space, we then calculate the posterior distribution conditioned on data-set $\mathcal{D}_2$, and finally optimize the marginal likelihood to compare the final predictions of the three covariance functions.

Figure 4.6 shows 5 random functions drawn for a zero mean GP with all the three covariance functions having the same values of hyper-parameters ($\theta_{lengthscale} = 1, \theta_{amplitude} = 1$), their corresponding covariance functions (figure 4.5(a)) and power spectrums (figure 4.5(b)) are shown in figure 4.5. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The colored (non-black) lines represent five functions drawn at random from a GP prior. We can observe that figure 4.6(a) draws non-differentiable functions while 4.6(b) and 4.6(c) draw smoother functions. For the same value of the length-scale, the Matérn ($\nu = 3/2$) kernel has faster variations in its functions, when compared to the SE kernel.



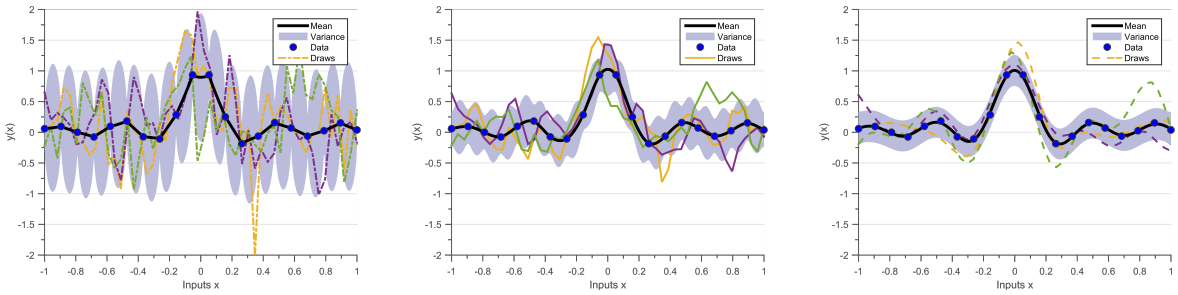
(a) Draws from a GP prior with mean zero, Exponential kernel (Matérn with $\nu = 1/2$) and hyper-parameters $\theta_{lengthscale} = 1, \theta_{amplitude} = 1$. Functions drawn from this kernel are non-differentiable.

(b) Draws from a GP prior with mean zero, Matérn kernel $\nu = 3/2$ and hyper-parameters $\theta_{lengthscale} = 1, \theta_{amplitude} = 1$. Functions drawn from this kernel are differentiable only once

(c) Draws from a GP prior with mean zero, SE kernel (Matérn with $\nu = \infty$) and hyper-parameters $\theta_{lengthscale} = 1, \theta_{amplitude} = 1$. Functions drawn from this kernel are infinitely differentiable

Figure 4.6: Prior distribution and five random draws from 3 different covariance functions. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior.

Figure 4.7 shows the posterior GP conditioned on the data-set $\mathcal{D}_2$ for three different covariance functions with the same hyper-parameters, here the marginal likelihood is not optimized. For similar values of hyper-parameters and data-set, Exponential kernel has highest variance followed by Matérn and SE kernel. This is a consequence of the bias vs variance trade-off since the SE kernel has the most restrictive hypothesis space (infinite differentiability is a stricter assumption), followed by Matérn $\nu = 3/2$ and Exponential kernel.



(a) Draws from a GP posterior with mean zero and Exopnential kernel (figure 4.6(a)) conditioned on the data $\mathcal{D}_2$. The posterior mean passes through the data-points, random functions drawn from exponential kernel are non-differentiable

(b) Draws from a GP posterior with mean zero and Matérn kernel $\nu = 3/2$ (figure 4.6(b)) conditioned on the data $\mathcal{D}_2$. The posterior mean passes through the data-points, random functions drawn from this kernel are differentiable only once

(c) Draws from a GP posterior with mean zero and SE kernel $\nu = \infty$ (figure 4.6(b)) conditioned on the data $\mathcal{D}_2$. The posterior mean passes through the data-points, random functions drawn from exponential kernel are infinitely differentiable

Figure 4.7: Posterior distribution and three random draws from 3 different covariance functions. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior.

Figure 4.8 shows the posterior GP conditioned on the data-set $\mathcal{D}_2$, for three different covariance functions with optimized hyper-parameters. All the three covariance functions estimate the same value of amplitude hyper-parameter, while having significantly different value of noise hyper-parameter (lowest noise estimated for Exponential kernel followed by Matérn ($\nu = 3/2$) and SE kernel).

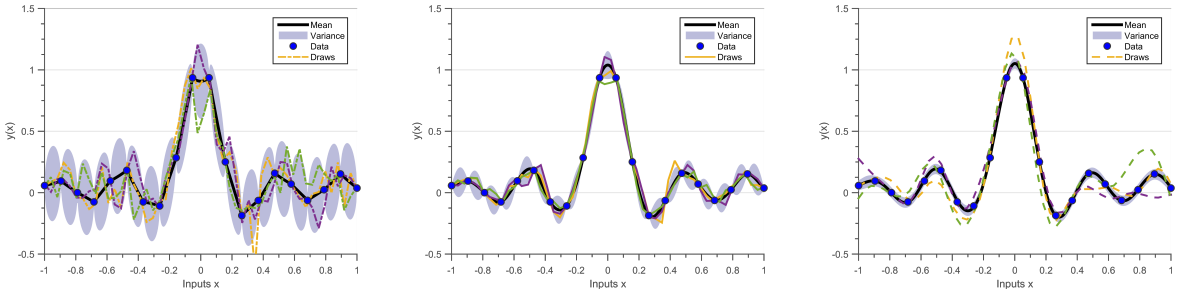
Bias vs Variance

The Exponential kernel defines a family of non-differentiable functions, hence functions in its hypothesis space have the flexibility to pass through all the data-points. Whereas, the SE kernel has less flexibility in the functions, due to its strict bias (infinitely differentiable). If the data-set does not have the same level of smoothness the SE kernel will find the closest function in its hypothesis space, and associate the difference to noise. Hence the Exponential kernel will almost always have a low noise

estimate than the SE kernel. The infinitely smooth assumption of the squared exponential function is unrealistic in several cases, making the Matérn kernels ($\nu = 5/2$) second most popular choice of kernels [Stein 1999, Cornford 2002].

Engineering judgment

This experiment only shows the different posteriors obtained for the same observational data and different functional forms of the covariance function. All three predictions can be the correct interpolations depending on the type of experiment, this is where engineering judgment is required. For example, if the data-set $\mathcal{D}_2$ was measured from a Brownian motion experiment then figure 4.8(a) would be the best fit, since the exponential kernel encodes prior information of Brownian motion (section 4.4.2).



(a) Draws from a GP posterior, conditioned on the data-set $\mathcal{D}_1$ with mean zero and Exponential kernel with hyper-parameters ($\theta_{lengthscale} = 0.215$, $\theta_{amplitude} = 0.312$ and $\sigma_n = 2.8e^{-}5$) that maximize the marginal likelihood ($max(ML) = -1$).

(b) Draws from a GP posterior, conditioned on the data-set $\mathcal{D}_1$ with mean zero and Matérn ($\nu = 3/2$) kernel with hyper-parameters ($\theta_{lengthscale} = 0.2$, $\theta_{amplitude} = 0.347$ and $\sigma_n = 1.7e^{-}5$) that maximize the marginal likelihood ($max(ML) = 2$).

(c) Draws from a GP posterior, conditioned on the data-set $\mathcal{D}_1$ with mean zero and se kernel with hyper-parameters ($\theta_{lengthscale} = 0.151$, $\theta_{amplitude} = 0.358$ and $\sigma_n = 0.01$) that maximize the marginal likelihood ($max(ML) = 8$).

Figure 4.8: Posterior distributions from three different covariance functions after maximizing the hyper-parameters.

Choosing functional form

If nothing is known about the data or type of experiment, and the number of hyper-parameters are same, then the covariance function with the maximum optimized marginal likelihood should be preferred. By this logic the SE kernel should be the preferred functional form of the covariance for the data-set $\mathcal{D}_2$. If the number of hyper-parameters are not the same, then marginal likelihood tends to be higher for covariance functions with greater number of hyper-parameters. [Duvenaud 2014, Lloyd 2014] propose to use the Bayesian Information Criterion (BIC)[3] to choose optimal covariance functions if the number of hyper-parameters is different.

[3]The BIC again performs a trade-off between data-fit and model complexity, for more details refer to section 4.5

4.4.4 Spectral Mixture Kernels

Spectral Mixture kernels define a more general class of stationary kernels exploiting the *Bochner's theorem* [Bochner 1959]. They define the power spectrum ($S(s)$) as a scale location mixture of Gaussians [Wilson 2013]. This has two benefits: firstly, with enough Gaussian components, scale location mixtures of Gaussians can approximate a curve up to arbitrary precision [Kostantinos 2000, Bishop 2006], secondly, the inverse Fourier transform of a scale location mixture of Gaussians can be evaluated analytically and is also a mixture of Gaussians.

$$S_{SM}(s,\mu,\sigma,w) = \sum_{q=1}^{Q} \frac{w_q}{\sqrt{2\pi\sigma_q^2}} \left(\exp\left[-\frac{(s-\mu_q)^2}{2\sigma_q^2} \right] + \exp\left[-\frac{(-s-\mu_q)^2}{2\sigma_q^2} \right] \right) \tag{4.23}$$

Here, S_{SM} is the power spectrum of the Spectral Mixture kernel. Each Gaussian component q has a mean μ_q, variance σ_q and weight w_q , there are total Q such components. The second term $\exp\left[-\frac{(-s-\mu_q)^2}{2\sigma_q^2}\right]$ is needed because a power spectrum of valid kernels should be symmetric around $s = 0$. The inverse Fourier transform of such a power spectrum will be a valid kernel and can be written as equation 4.24. For a detailed derivation refer to [Wilson 2014].

$$k_{SM}(d,\mu,\sigma,w) = \sum_{q=1}^{Q} w_q cos(2\pi\mu_q) exp[-2\pi^2 d^2 \sigma_q^2] \tag{4.24}$$

Here, k_{SM} is the covariance function of the above power spectrum (equation 4.23). The parameter w_q is the weight of the Gaussian component q, the mean of the Gaussian component μ_q defines the period of kernel while the variance σ_q of the Gaussian component denotes inverse of the length scale .

Pattern recognition

[Wilson 2014] demonstrate that the Spectral Mixture kernels can be used as a replacement for many available kernels, since they define a general class of stationary kernels. Due to their expansive hypothesis space, they can be used as a means to perform automatic pattern discovery. Their Fourier transform can provide more understanding of the observation data-set (this is very similar to interpreting the Fourier transform of dynamic data) [Wilson 2013]. One of the major hindrances of SE kernels is that they cannot be used for extrapolation, we cannot extrapolate an SE kernel a $\theta_{lengthScale}$ distance away from the last observation (figure 2.3). Since Spectral Mixture kernels can detect patterns in the data they can also be used to perform extrapolation[4].

[4]A detailed code can be found : https://people.orie.cornell.edu/andrew/code/#spectral

We propose to use this relationship between the covariance function and its Fourier transform to automatically identify parameters of a dynamic system. Dynamic engineering systems are generally parametrized by their modal frequencies and participation factors. In structural engineering, identification of modal frequencies is an important step for certification, while minor change in modal frequencies can help in fast discovery of failure. In the next section we apply the Spectral Mixture kernel to automatically identify the modal frequencies of a structural system, parts of the following work have been published in [Chiplunkar 2017b].

4.5 Application: Identifying Structural Dynamics Parameters

Modal analysis has been widely used as a means of identifying dynamic properties such as modal frequencies, damping ratios and mode shapes of a structural system. Traditionally, the system is subjected to artificial input excitations and the output deformations (displacements, velocities or accelerations) are measured. These later help in identifying the modal parameters of the system, this process is called Experimental Modal Analysis (EMA).

OMA

Since the last decade Operational Modal Analysis (OMA) has gained considerable interest in the structural dynamics community. OMA identifies the modal parameters only from the output measurements while assuming ambient excitations as random noise. OMA is cheaper because it does not require expensive experimental setup and can be used in real time operational use cases such as health monitoring [Peeters 2005, Shahdin 2010, Rainieri 2007].

MDOF

In the last few decades several algorithms primarily using the assumption of second order differential, Multi Degree Of Freedom (MDOF) system (equation 4.25) have been developed to find the modal parameters [Guillaume 2003, Richardson 1982].

$$\boldsymbol{M}\{\ddot{x}(t)\} + \boldsymbol{C}\{\dot{x}(t)\} + \boldsymbol{K_{Stiffness}}\{x(t)\} = \{\boldsymbol{f}(t)\} \tag{4.25}$$

Here, $\boldsymbol{M}$, $\boldsymbol{C}$ and $\boldsymbol{K_{Stiffness}}$ denote the mass, damping and stiffness matrices respectively. $\{x(t)\}$ and $\{\boldsymbol{f}(t)\}$ denote the displacement and force vectors at the time t, while $\ddot{x}(t)$ and $\dot{x}(t)$ denote the second and first time derivative of displacement respectively. Figure 4.9(a) shows an example of ambient measurements $x(t)$ on a structure. In almost all OMA algorithms the measurement $x(t)$ is assumed to be generated from a random force excitation.

Earlier Work The Natural Excitation Technique (NExT) [James III 1995] proves that the auto-correlation function $k(\tau)$ can be written as sum of decaying sinusoid's [Spitznogle 1970, Ibrahim 1977, Guillaume 2003]. The auto-correlation describes the similarity between measurements as a function of time lag τ between them (figure 4.9(b)). *NExT*

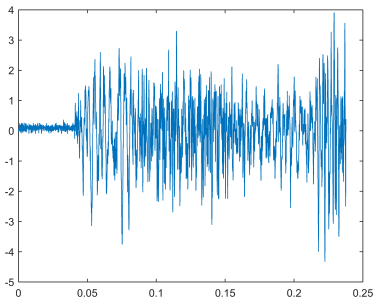
$$k(\tau) = \int x(t)x(t-\tau)dt \quad k(\tau) = \sum A_i exp(-\lambda_i \tau) sin(B_i \tau) \tag{4.26}$$

Here, $k(\tau)$ denotes the auto-correlation for random vector $x(t)$ as a function of time lag τ. While, λ_i and A_i denote the modal frequency and mode shapes for the i^{th} mode. The above coefficients are found by minimizing the least square error between the measured $k(\tau)$ and the predicted $k(\tau)$ from equation 4.26.

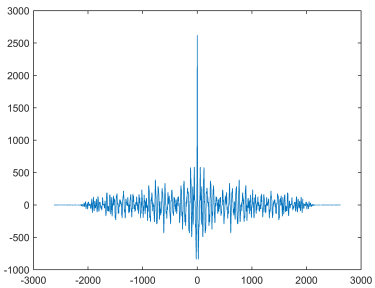
If we assume the measurement $x(t)$ to be a stationary random process, then according to *Bochner's theorem*, the Fourier transform of $k(\tau)$ (power spectrum $S(s)$) exists [Bochner 2016]. Figure 4.9(c) shows the power spectrum calculated for the measurement $x(t)$ shown in figure 4.9(a). Using the above mentioned second order differential assumption a Rational Fractional Polynomial (RFP) (equation 4.27) can be used to fit a power spectrum [Richardson 1982, Allemang 1998, Chauhan 2007]. *RFP*

$$S(s) = \int k(\tau) e^{-2\pi i s^T s} d\tau \quad S(s) = \frac{\sum a_k(s)^k}{\sum b_l(s)^l} \tag{4.27}$$

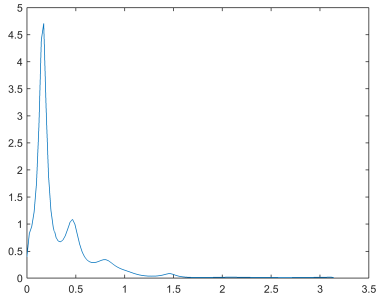
Here, the poles of the polynomial denote the modal frequencies, while other modal parameters can be derived from the coefficients a_k and b_l. The coefficients of the polynomial can be found by minimizing the least squared error. RFP based algorithms face problems of numerical stability as the value of number of modes (l) increases.



(a) Measured output on accelerometers $x(t)$



(b) Auto-correlation $k(\tau)$ of the measured output fig: 4.9(a)



(c) Power spectrum density $S(s)$ of the output fig: 4.9(a)

Figure 4.9: Different types of measurements for estimation of Modal parameters in OMA

Measurement	Auto-correlation	Power Spectrum
$x(t)$	$k(\tau) = \int x(t)x(t-\tau)dt$	$S(s) = \int k(\tau)exp(-2\pi i s^T \tau)d\tau$
Assumption: Second Order Differential		
	$\sum A_i exp(-\lambda_i \tau)sin(B_i \tau)$	$\frac{\sum a_k(s)^k}{\sum b_l(s)^l}$
Assumption: Gaussian Mixture Model		
$GP(0, k_{SM})$	$\sum w_i cos(2\pi\mu_i\tau)exp\{-2\pi^2\sigma_i^2\tau^2\}$	$\sum w_i \frac{1}{\sqrt{2\pi\sigma_i^2}} exp\{\frac{1}{2\sigma_i^2}(s-\mu_i)^2\}$

Table 4.1: Comparison of fitting functions

Contribution The above mentioned OMA algorithms NExT in the time lag (τ) domain and RFP in the frequency domain (s), assume the dynamic behaviour to be a second order differential system. This assumption fails for non-linear systems and for cases where modal frequencies are very close. Instead we propose to use the Spectral Mixture kernel to fit the measurement $x(t)$.

Using the same hypothesis used in section 4.4.4, we can say that a scale-location mixture of Gaussian components can approximate any distribution. We thus place a scale-location mixture of Gaussians on the power spectrum ($S(s)$), this means that we are assuming a prior distribution of $x(t)$ as a GP with a Spectral Mixture kernel (equation 4.28). The hyper-parameters of the system are: the mean μ_q defines the modal frequency, the variance σ_q which is a measure of damping, the weight w_q which defines the participation factor of component q and the total number of components Q.

$$\Pr[x(t)] = GP(0, k_{SM} = \sum_{q=1}^{Q} w_q cos(2\pi\mu_q)exp[-2\pi^2\tau^2\sigma_q^2) \tag{4.28}$$

Table 4.1

We would like to emphasize that keeping the computational complexities aside, fitting a Spectral Mixture GP in time-domain (equation 4.28), fitting Spectral Mixture (equation 4.24) on covariance values and fitting a scale-location mixture of Gaussians (equation 4.23) on the power spectrum are equivalent. In the publication [Chiplunkar 2017b] we fit a scale-location mixture of Gaussians on the power spectrum, here we propose to fit the GP on the measurements $x(t)$. Refer to table 4.1 for a more comprehensive view at various fitting functions.

Choosing Q

While the modal parameters can be chosen by minimizing the least square error, how to choose the number of modes is a recurring question in several OMA algorithms. This problem is partially resolved by using stabilization diagrams or mode identifica-

tion functions [Allemang 1998, Williams 1985, Shih 1988]. But in practical situations, engineering judgment is often required to estimate the optimal modal order. To automatically choose Q, we use the Bayesian Information Criteria (BIC) [Findley 1991] which penalizes more complex models to estimate the parameter Q. The BIC estimate has been earlier used to find the optimal functional form of covariance function [Duvenaud 2013]

$$BIC(Q) = N \ln(ML) + N_{hyp} \ln(N) \tag{4.29}$$

Here, N denotes the number of data-points to fit, ML denotes the marginal likelihood of the fit and N_{hyp} denote the number of hyper-parameters to fit. The BIC performs a trade-off between the data-fit term $N \ln(ML)$ and the complexity penalty term $N_{hyp} \ln(N)$, basically penalizing for over-fitting. Lowest value of BIC is preferred.

BIC

Generally, identification of modal frequencies requires engineering judgment, i.e. engineers have to look at the stabilization diagrams and figure out the optimal value of modal order. Using the Spectral Mixture kernel we have encoded the knowledge of peaks in the power spectrum which in combination with BIC has eliminated the need to perform costly engineering judgment exercises and has made the process automatic. We now compare the accuracy of finding modal frequencies using a Spectral Mixture kernel vs NExT [James III 1995] methodology for a toy data-set, and then later for a real data-set of HCT building [Brincker 2000].

4.5.1 Results on a toy data-set

In this section we conduct experiments, applying our approach on a highly damped 3 degree of freedom system. As stated earlier we fit a GP model having a Spectral Mixture covariance on the measurements $x(t)$ for varying number of Gaussian components Q. We then evaluate the BIC to find the optimal value of Q for this measurement. The results of automatic identification are then compared to that of NeXT identification method.

All experiments were performed on an Intel quad-core processor with 4Gb RAM. The Spectral Mixture technique suffers from multiple minimas and thus care should be taken while initializing the hyper-parameters[5].

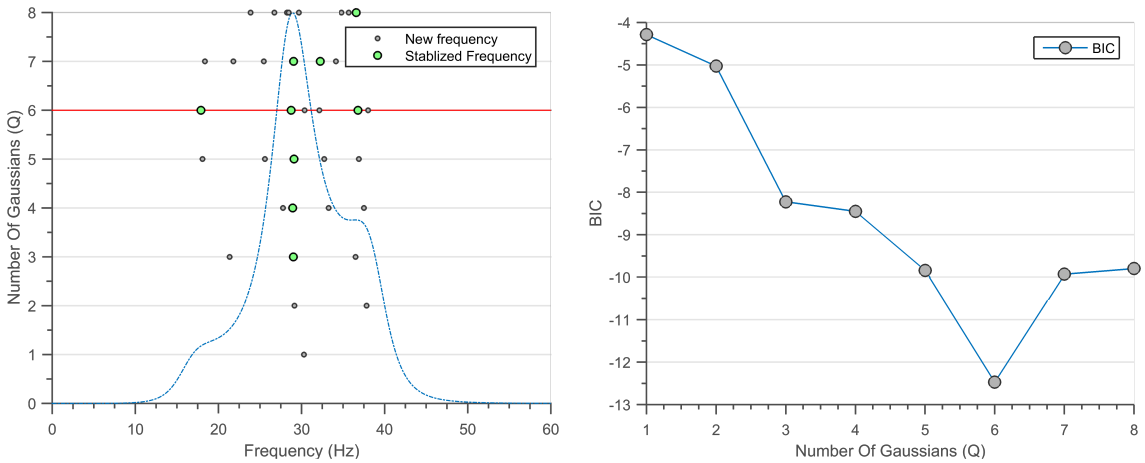
Table 4.2 shows the comparison of frequencies for Modal frequencies predicted using NeXT algorithm and Spectral Mixture algorithm. We can observe that the NeXT method cannot predict the third frequency with enough accuracy. This is because the third frequency has the hignest value of damping.

[5]For fast optimization we first fit a Gaussian Mixture model on the power spectrum to initialize the hyper-parameters and then optimize the marginal likelihood.

	Real Frequency	Error NeXT Frequency	Error Spectral Mixture
First Frequency (Hz)	17.5	2.7 %	0.28 %
Second Frequency (Hz)	30	3.4 %	4.1 %
Third Frequency (Hz)	35	12.34%	5.1%

Table 4.2: Comparison of Modal frequencies for toy data-set

Figure 4.10(a) shows the stabilization diagram with increasing number of Gaussians Q. As we progressively increase the number of components we start getting more and more modal frequencies. We call a frequency stabilized if the difference between modal frequencies of two subsequent Q is less than 1%. The green points are stabilized frequencies. Figure 4.10(b) shows the BIC criterion with increasing number of Gaussian's Q. We can see that that the BIC is minimum for $Q = 6$ and hence if we add any more Gaussians for our data-set we will be over-fitting. The red line in figure 4.10(a) shows the Q with minimum BIC and hence the stabilized frequencies for this automatically become our modal frequency predictions.



(a) Stabilization diagram with increasing number of Gaussians Q, the dots denote the stabilized frequencies. We can observe that as the number of Q increases the algorithm starts finding better and better modes.

(b) The BIC criterion with increasing number of Gaussian's Q. We can see that that the BIC is minimum for $Q = 6$ and hence if we add anymore Gaussian's for our data-set we will be performing over-fitting

Figure 4.10: Results of Spectral Mixture kernels on a toy data-set

4.5.2 Results on a HCT building data-set

We now apply our methodology on real ambient response of Heritage Court Tower (HCT) Building[6]. The responses are measured at 6 different points on the building resulting in 6 different power spectrums. We compare the modal frequencies obtained using Spectral Mixture kernel with the modal frequencies obtained in the original paper [Brincker 2000]. To compare the performance we automatically identify modal frequencies present in each of the 6 power spectrums and take the mean of the stabilized frequencies.

Table 4.2 shows the comparison of Modal frequencies predicted in [Brincker 2000] with the Spectral Mixture algorithm. The results of the two methods are very similar, although the frequencies obtained by Spectral Mixture GP are completely automatic.

	Spectral Mixture	[Brincker 2000]
First Frequency (Hz)	1.23	1.23
Second Frequency (Hz)	1.29	1.27
Third Frequency (Hz)	1.43	1.45
Fourth Frequency (Hz)	3.87	3.86
Fifth Frequency (Hz)	4.28	4.25

Table 4.3: Comparison of Modal frequencies for HTC data-set

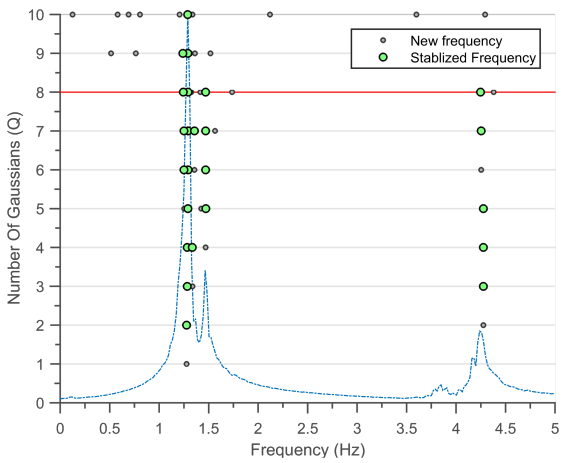
Figure 4.11(a) shows the stabilization diagram with increasing number of Gaussians Q. We can observe that as the number of Q increases the algorithm starts finding better and better modes, three modes start stabilizing from $Q = 5$. Figure 4.11(b) shows the BIC criterion with increasing number of Gaussian's Q. We can see that that the BIC is minimum for $Q = 8$ and hence if we add any more Gaussians for our data-set we will be over-fitting.

In the current setting of the Spectral Mixture model we propose an automatic way to identify the most important frequencies of a structural system. Neither the mode shapes nor the damping ratios are estimated in the current format. In the future we would like to derive a method to estimate mode-shape and damping ratio such that the contributions of neighbouring Gaussians are also taken into account.

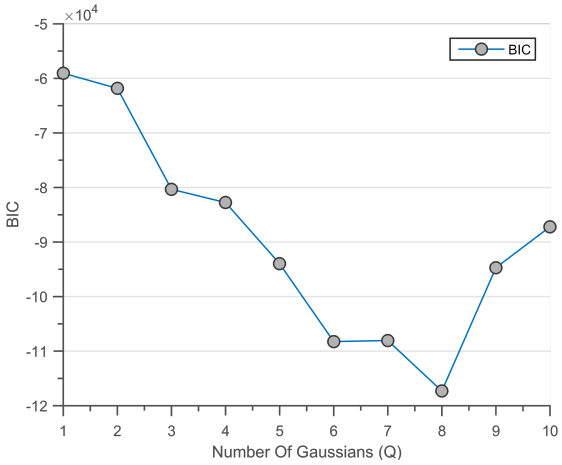
4.6 Summary and discussion

This chapter discussed only a small number of possible covariance functions, the table below provides a list of basic covariance functions that we have encountered thus far. Several other other functional forms of the covariance function exist in the literature, for example

[6]The data is available at : http://www.brinckerdynamics.com/oma-toolbox



(a) Stabilization diagram for one of the 6 power spectrums. The green dots denote the stabilized frequencies, the red line denotes minimum BIC. We can observe that as the number of Q increases the algorithm starts finding better and better modes.



(b) The BIC criterion with increasing number of Gaussian's Q. We can see that that the BIC is minimum for $Q = 8$ and hence if we add any-more Gaussian's for our data-set we will be per-forming over-fitting

Figure 4.11: Results of Spectral Mixture kernels on real data from HTC building

Gibbs kernel, Rational Quadratic kernel, Periodic kernel etc for more detailed list please refer to works of [Rasmussen 2005, Duvenaud 2013, Wilson 2014]. .

Kernel Name	Expression $k(x_i, x_j)$	Constituent functions
Constant	σ_c^2	Constant functions
Noise	$\sigma_n^2 \delta_{x_i x_j}$	White noise
Linear	$\phi(x_i)\Sigma\phi(x_j)$	Linear combination of ϕ
Neural Network	$\theta_1^2 \frac{2}{\pi} sin^{-1}\left(\frac{2x\theta_2 x'}{\sqrt{(1+2x^T\theta_2 x)(1+2x'^T\theta_2 x')}}\right)$	Linear combinations of Sigmoids
Standard Exponential	$\theta^2_{amplitude} exp[-\frac{\tau^2}{2\theta^2_{lengthScale}}]$	Infinitely Differentiable functions
Matérn	$\theta^2_{amplitude} \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu(\tau))}}{\theta_{lengthScale}}\right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu(\tau))}}{\theta_{lengthScale}}\right)$	Adjustable differentiability
Spectral Mixture	$\sum_{q=1}^{Q} w_q cos(2\pi\mu_q) exp[-2\pi^2\tau^2\sigma_q^2]$	Summation of periodic kernels

Table 4.4: List of covariance functions

We start off this chapter by defining a few important properties (section 4.2) of the covariance functions. Section 4.3 details a few non-stationary kernels and discusses how Bayesian linear regression can be effectively seen as a GP regression with linear covariance function. Section 4.4 describes stationary kernels, Using the *Bochner's theorem* we can effectively represent a stationary kernel into its Fourier transform, this gives rise to another interpretation of the constituent family of functions. Section 4.4.3 demonstrates the effects on posterior prediction for different functional forms of covariance functions. For each covariance function we try to give a visualization of the shape of constituent functions by randomly drawing functions from their priors.

Modal parameter estimation

The main contribution of this chapter is demonstration on how to use GP regression to automatically detect modal parameters of structural dynamics. To the best of our knowledge, such a method has not been used in the earlier literature to identify modal parameters. Using the Spectral Mixture kernels we demonstrate how to build models for structural dynamic experiments and automatically identify dynamic parameters such as modal frequency. This is a very early stage of application of Spectral Mixture Kernel for system identification, and there remain problems such as identification of mode-shape, and damping ratio in this algorithm. We wish to tackle these problems in the future.

As mentioned earlier the core aim of machine learning was to identify patterns and extrapolate on data automatically. There are two main approaches in pattern discovery for GPs; one is based on increasing the hypothesis space by defining a process over all stationary kernels [Wilson 2012]. The second which defines a language of kernels and iteratively adds basic kernels to come up with an explanation to the pattern in the data [Lloyd 2014]. In the next chapter we will look in detail at how to build more complex kernels using basic kernels. Which approach will be retained in the long term is difficult to predict but automatic pattern detection is a highly active research subject in GPs.

Chapter 5

Combining Basic Covariance Functions

Résumé

Souvent, le type d'information initiale que nous voulons intégrer n'est pas disponible sous la forme de fonctions de covariance basiques. Heureusement, plusieurs nouveaux noyaux peuvent être créés en fusionnant les noyaux de base que nous avons vus dans le chapitre précédent. Ce chapitre est inspiré par les travaux de [Bishop 2006, MacKay 2003, Durrande 2001, Durrande 2013a].

La section 5.2 décrit comment combiner les noyaux pour créer des noyaux unidimensionnels. Nous fournissons une explication intuitive de ce qui se produit lorsque nous ajoutons ou multiplions des noyaux, ces types de noyaux peuvent être utilisés pour détecter et séparer les modèles de données. La section 5.2.4 décrit comment créer des noyaux pour des dimensions plus élevées. Ce type de noyaux peut être utilisé pour effectuer une analyse de sensibilité ou projeter les données en dimension plus faibles.

Dans ce chapitre, nous utilisons une combinaison de noyaux de base pour identifier l'apparition de non linéarités dans les systèmes physiques. Par exemple, nous identifions l'initiation de la séparation de l'écoulement pour le profil d'aile NACA 0012 et l'initiation de la plasticité pour l'alliage AL6061 en utilisant un critère statistique pour la détection automatique de la non-linéarité (section 5.2.4) [Chiplunkar 2016b]. Nous construisons enfin un modèle de GP pour prédire la position du choc aérodynamique en régime transsonique (section 5.3.5) [Chiplunkar 2017a].

5.1 Introduction

What if the kind of patterns that we wish to encode are not possible by using the basic kernels described in chapter 4? What if we wish to encode several different patterns in our hypothesis space? Or what if we want to build models for data-sets with more than one input dimension? Thankfully, many new kernels can be constructed by merging a few basic kernels, this chapter answers the above questions.

The original contribution of this chapter is applying the newly created covariance functions to create engineering design models. The set of equations below are a few simple methods to create valid covariance functions [Bishop 2006, MacKay 2003, Durrande 2001, Durrande 2013a].

$$k(x_1, x_2) = k_1(x_1, x_2) + k_2(x_1, x_2) \tag{5.1}$$
$$k(x_1, x_2) = k_1(x_1, x_2) \times k_2(x_1, x_2) \tag{5.2}$$
$$k(x_1, x_2) = q(x_1)k_1(x_1, x_2)q(x_2) \tag{5.3}$$
$$k(x_1, x_2) = k_1(h(x_1), h(x_2)) \tag{5.4}$$
$$k(x_1, x_2) = g(g(k_1(x_1, x_2), x_1), x_2) \tag{5.5}$$
$$k(x_1, x_2) = \int k_1(x_1, u)k_2(u, x_2)du \tag{5.6}$$

Here, $k_1(x_1, x_2)$ and $k_2(x_1, x_2)$ are valid covariance function, while $q(x)$ and $h(x)$ are any continuous functions. Equation 5.6 defines a covariance function through convolution of two kernels, while equation 5.5 defines transformation through a linear operator $g(.) \in \mathcal{C}^2$, for more discussion on this equation refer to chapter 7.

Let us take the case of a 2-dimensional input vector[1] $\boldsymbol{x}$ such that $\boldsymbol{x} = \{x^1, x^2\}$ (x^1, x^2 are coordinates of $\boldsymbol{x}$ in the first and second dimension respectively). Then the following covariance functions are also valid.

$$k(\boldsymbol{x_1}, \boldsymbol{x_2}) = k_1(x_1^1, x_2^1) + k_2(x_1^2, x_2^2) \tag{5.7}$$
$$k(\boldsymbol{x_1}, \boldsymbol{x_2}) = k_1(x_1^1, x_2^1) \times k_2(x_1^2, x_2^2) \tag{5.8}$$

The current chapter is written to provide intuition on what happens when we combine covariance functions. This chapter unfolds as follows; section 5.2 provides intuition on combining kernels for one-dimensional inputs, while section 5.3 details how to create covariance functions for higher-dimensions (equation 5.7 and 5.8) inputs. We then apply the newly constructed kernels, to first detect onset of non-linearity in experimental data, and later interpolate the shock pressures on a CRM wing in transonic regime.

[1] The superscript is used to denote the number of dimension and does not denote the power.

5.2 One dimensional inputs

Combining kernels can give rise to interesting features, in this section we provide intuition on combining kernels in one dimension. Section 5.2.1 details effects of multiplying kernels (equation 5.2) while section 5.2.2 describes effects of adding kernels (equation 5.1). Section 5.2.4 describes the Change-Point (CP) kernel. We use the CP kernel to detect start of plasticity in elastic structures such as beams and start of flow separation on an airfoil [Chiplunkar 2016b].

5.2.1 Multiplying Kernels

Multiplying two covariance functions acts as an AND operator, the resulting kernel has high value only if we have high value on both the kernels. Multiplying a Linear kernel L times, will result in a L^{th} order polynomial regression (equation 5.9).

$$k_{Lin} = w_0 + w_1 x_1^T x_2 \quad k_{Poly} = \prod_{i=1}^{L} \left(w_0^i + w_1^i x_1^T x_2\right) \tag{5.9}$$

Equation 5.10 shows the prior obtained after multiplying a Linear and a SE kernel. This covariance function resembles a SE covariance function but with the amplitude hyper-parameter ($\theta_{amplitude} = x_1 x_2$) proportional to the distance.

$$k_{Multi} = x_1 x_2 exp[-\frac{d^2}{2}] \tag{5.10}$$

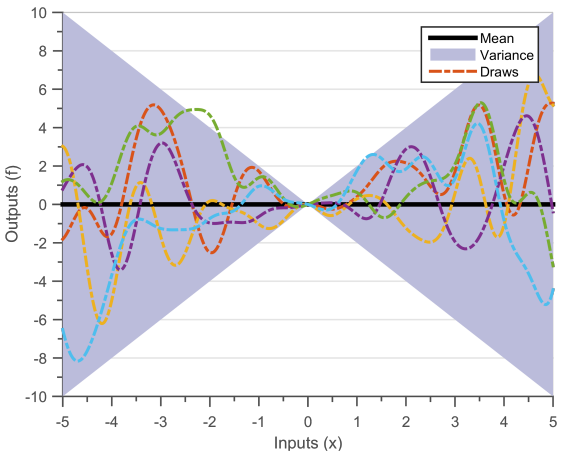
Figure 5.1(a) shows random draws obtained using k_{Multi} covariance, the hyper-parameters of the linear kernel are $w_0 = 0$ and $w_1 = 1$, this means that there is no intercept and $k_{Lin}(x_1, x_2) = x_1 x_2$. The hyper-parameters of the SE part are $\theta_{amplitude} = 1$ and $\theta_{lengthScale} = 1$, similar to figure 4.6(c). Since multiplying two kernels is an AND operation, k_{Multi} tends to zero at $x = 0$ since k_{Lin} is zero at $x = 0$.

Figure 5.1(a)

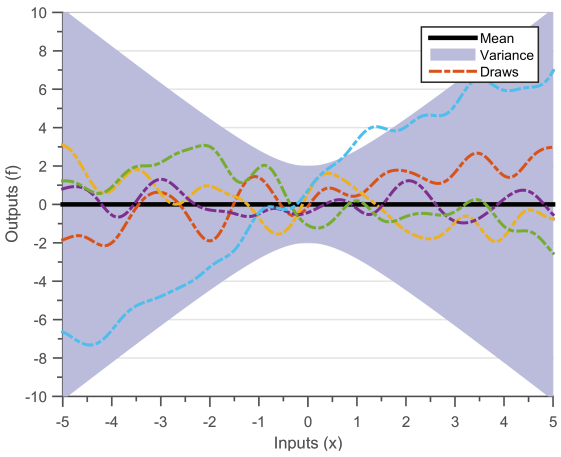
5.2.2 Adding Kernels

Adding two kernels acts as an OR operator, this means that the resulting kernel will have high value if either of the two kernels have high value [Durrande 2011].

Figure 5.1(b) shows the prior obtained after adding a Linear and a SE kernel. The hyper-parameters of the linear kernel are $w_0 = 0$ and $w_1 = 1$, this means that there is no intercept and $k_{Lin}(x_1, x_2) = x_1^T x_2$. The hyper-parameters of the SE part are $\theta_{amplitude} = 1$ and $\theta_{lengthScale} = 1$, similar to figure 4.6(c).



(a) Draws from a GP prior with mean zero and kernel obtained by **multiplying** a Linear kernel with SE kernel. The hyper-parameters of the linear kernel are $w_0 = 0$ and $w_1 = 1$ while the hyper-parameters of the SE part are $\theta_{amplitude} = 1$ and $\theta_{lengthScale} = 1$. We see that the variance at $x = 0$ goes to zero, since multiplication is an AND operation



(b) Draws from a GP prior with mean zero and kernel obtained by **adding** a Linear kernel with SE kernel. The hyper-parameters of the linear kernel are $w_0 = 0$ and $w_1 = 1$ while the hyper-parameters of the SE part are $\theta_{amplitude} = 1$ and $\theta_{lengthScale} = 1$. We see that the variance at $x = 0$ goes to variance of SE kernel, since multiplication is an OR operation

Figure 5.1: Random draws from combining a Linear and SE kernel. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior. Random functions drawn from a linear GP are linear.

Adding Noise

The Linear kernel discussed in section 4.3.1 is a sum of three covariance functions; a constant covariance (w_0), a linear covariance ($w_1 x_1^T x_2$) and a white noise covariance function ($\sigma_n^2 \delta_{xx'}$) (equation 5.11). Similarly, the noisy posterior case discussed in section 2.3 is a case of adding a SE kernel with a white noise kernel, a heteroscedastic[2] noise model can be similarly created by adding several kernels together.

$$k(x_1, x_2) = w_0 + w_1 x_1^T x_2 + \sigma_n^2 \delta_{xx'} \tag{5.11}$$

An interesting consequence of adding kernels is that now we can decompose the result into additive parts. Suppose k_{Sum} is a covariance function by adding n covariance functions $k_1, k_2, \ldots, k_n$ (equation 5.1), then the posterior mean and covariance can be written as equation 5.12 and 5.13.

[2]Noise depending on the inputs

$$\mathbf{E}[f(x) \mid \boldsymbol{X}, \boldsymbol{y}, k_{Sum}] = \sum_{i=1}^{n} \boldsymbol{k_i}(\boldsymbol{x_* X})(\boldsymbol{K_{Sum}}(\boldsymbol{X}, \boldsymbol{X}))^{-1}\boldsymbol{y} \tag{5.12}$$

$$Cov[f(x) \mid \boldsymbol{X}, \boldsymbol{y}, k_{Sum}] = \sum_{i=1}^{n} \left[k_i(x_* x_*) - \boldsymbol{k_i}(\boldsymbol{x_* X})(\boldsymbol{K_{Sum}}(\boldsymbol{X}, \boldsymbol{X}))^{-1}\boldsymbol{k_i}(\boldsymbol{X x_*})\right] \tag{5.13}$$

Note that the precision matrix $(\boldsymbol{K_{Sum}}(\boldsymbol{X}, \boldsymbol{X}))^{-1}$ cannot be decomposed into its constituent covariance matrices. This is not an issue since this strategy is used to separate/discover individual effects in the data-set. For example, if we make a covariance function by adding a Linear kernel and an SE kernel, we can seperate the Linear part of the data-set from the non-linear part.

Discovering pattern

This comes very handy while iteratively discovering structure in the data. One method to make an optimal covariance structure by hand is to iteratively add new kernels until the posterior error variance represents a white noise. [Rasmussen 2005] use a sum of several kernels to interpolate CO_2 content in the atmosphere through the years. [Durrande 2013b] use a sum of periodic kernels to detect which genes are responsible for the 24-hour cycle (*circadian rhythm*) in *arabidopsis* plant. [Duvenaud 2013, Lloyd 2014] propose to automatically detect pattern by adding new kernels, and using the BIC measure to find the optimal covariance structure.

5.2.3 Change-Point kernels

The CP kernel was introduced to recognize changes in regimes, and adapt the covariance function accordingly. They were initially introduced to identify change points in time-series modelling [Osborne 2010, Saatçi 2010]. These kernels can be defined through addition and multiplication with sigmoidal functions (equation 5.14).

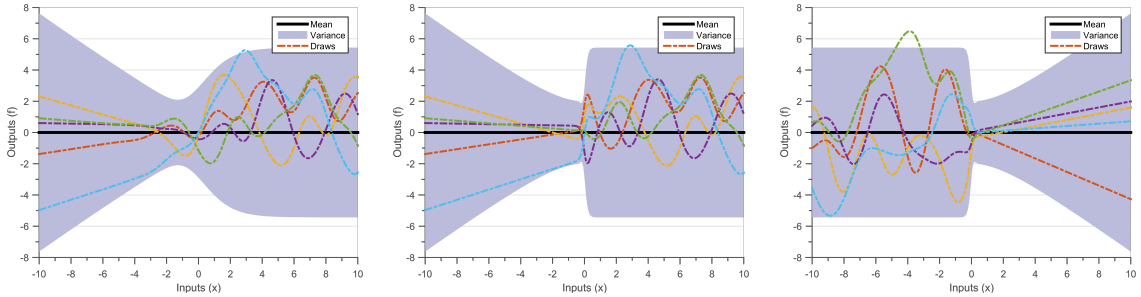
$$sigm(x, \theta) = \frac{1}{\theta_{intensity} + e^{\theta_{changeLocation} - x}} \tag{5.14}$$

$$k_{CP}(k_1, k_2, x_1, x_2) = sigm(x_1)k_1 sigm(x_2) + (1 - sigm(x_1))k_2(1 - sigm(x_2)) \tag{5.15}$$

Hyper-parameters

The hyper-parameters of this kernel are the $\theta_{changeLocation}$ which determines the location of the change point, $\theta_{intensity}$ determine the intensity of change between the two patterns, and the hyper-parameters of covariance functions k_1 and k_2. Figure 5.2 shows the randomly drawn functions from a CP kernel for varying values of $\theta_{intensity}$, where the regime changes from a Linear kernel to a SE kernel. The hyper-parameters of the linear kernel are $w_0 = 0$

and $w_1 = 1$, this means that there is no intercept and $k_{Lin}(x_1, x_2) = x_1^T x_2$. The hyper-parameters of the SE part are $\theta_{amplitude} = 1$ and $\theta_{lengthScale} = 1$, similar to figure 4.6(c). We see that as the value of $\theta_{intensity}$ increases the regime change happens more rapidly, while if $\theta_{intensity}$ changes sign then the order of regime reverses.



(a) Draws from a GP prior with mean zero and CP kernel (equation 5.15) between a Linear and a SE kernel with $[\theta_{intensity}, \theta_{changeLocation}] = [1, 0]$.

(b) Draws from a GP prior with mean zero and CP kernel (equation 5.15) between a Linear and a SE kernel with $[\theta_{intensity}, \theta_{changeLocation}] = [10, 0]$. Notice if the value of $\theta_{intensity}$ increases then the change between two patterns becomes more significant.

(c) Draws from a GP prior with mean zero and CP kernel (equation 5.15) between a Linear and a SE kernel with $[\theta_{intensity}, \theta_{changeLocation}] = [-10, 0]$. Notice if the sign of $\theta_{intensity}$ changes then the order of kernels gets reversed.

Figure 5.2: Random draws by having a change-point between a Linear and SE kernel. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior. Random functions drawn from a linear GP are linear.

5.2.4 Application: Identifying onset of non-linearity using CP kernel

Several physical processes can be represented using a linear approximation in some part of their regime. This approximation is possible because the linear effects dominate in that part of the regime, but they eventually wear off and second and third order effects start becoming more powerful.

Linear assumption

This basic assumption is used in making simple models in several domains; for example in a material during the elastic regime $Stress \propto Strain$ is a basic linear approximation. The proportionality constant between Stress and Strain is called Young's Modulus, which is unique for each material. When the elastic regime starts wearing off, non-linear behaviour called plasticity starts taking over and the approximation $Stress \propto Strain$ is not valid anymore. Similarly in aerodynamics, when characterizing a flow over an airfoil during the

Linear regime $Lift \propto AngleOfAttack$, the airflow is attached on the airfoil during this regime. When the airflow starts separating from the airfoil the non-linear effects start dominating.

Contribution

The value of these basic physical parameters such as slope (eg. Young's Modulus, coefficient of lift) and location of change in regime (eg. start of plasticity and separation of flow) are progressively fed into further simulations. Generally, the slope and location of change points are evaluated using engineering judgment, this is a slow and costly process. We propose to estimate the slope and location of change-point automatically using a GP with CP kernel. Using the CP kernel which transitions from linear domain (linear kernel) to non-linear domain (SE kernel), prior information of the transition is encoded in the kernel structure.

We perform our experiments on openly available Stress and Strain data of Aluminum Alloy 6061 [Kaufman 1999] and Lift and Angle data of NACA 0012 airfoil[a]. To estimate the CP hyper-parameters, we again perform a 10-fold cross validation. The data-set will be randomly partitioned into 10 subsets containing an equal number of points. Of the 10 subsets, a single subset is retained as the test data-set, and the remaining 9 (10 - 1) subsets are used as training data. The cross-validation process is then repeated 10 folds, with each of the k subsets used exactly once as the validation data. The marginal-likelihood is optimized for each of the training data-set and location of change-point and slope of the linear regime is noted. We then compare their average with the values available in the literature.

[a]Airfoil data from: http://airfoiltools.com/airfoil/details?airfoil=n0012-il

Table 5.1 shows the results of physical parameters for AL 6061 when calculated using change-point kernel vs that available in the literature. We can observe that the change-point automatically predicts the correct values of Young's Modulus and start of non-linearity.

	Change-point	Literature
Young's Modulus (GPa)	68.5	68.9
Start of plasticity	0.94%	0.95%

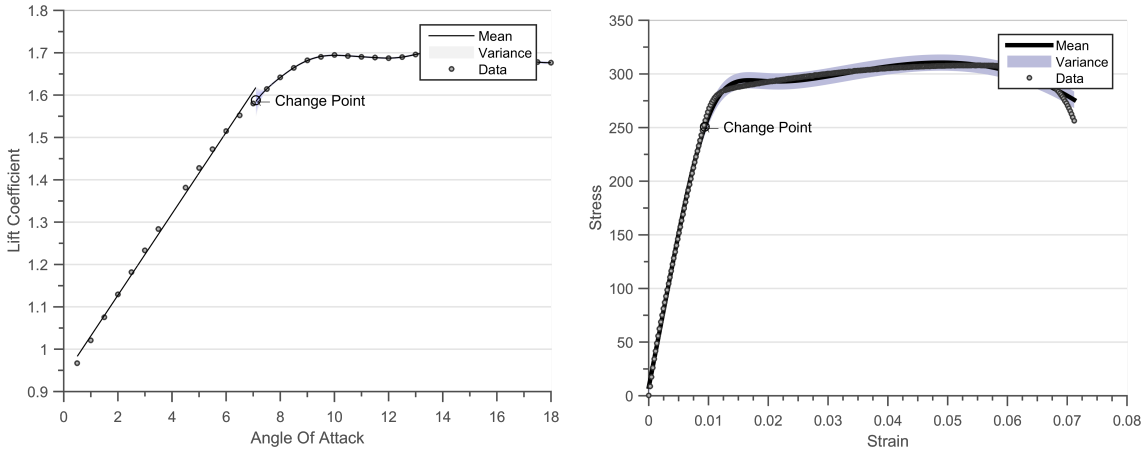
Table 5.1: Comparison of Young's Modulus for Al 6061 data-set

Figure 5.3

Figure 5.3 shows the posterior predictions when using the CP kernel. Figure 5.3(a) is the posterior distribution for the case of NACA 0012 airfoil, the Linear and SE regimes are plotted independently. Figure 5.3(b) shows the posterior distribution for the case of AL 6061. We can observe that the algorithm predicts a CP for the data-set, this is the point where the non-linear effects start dominating.

Multi-modality

The marginal likelihood of a change-point kernel has many local minimas, there is a local minimum at every observation point. This is because the kernel puts a non-linear regime at every observation point, hence a global optimizer should be used for optimization. The results of this study were be presented in the SIAM Uncertainty Quantification 2016 Conference [Chiplunkar 2016b].



(a) Posterior distribution for the case of NACA 0012 airfoil, the Linear and SE regimes are plotted independently.

(b) Posterior distribution for the case of AL 6061 alloy, the Linear and SE regimes are plotted independently.

Figure 5.3: Estimation of linear regimes using a change-point kernels

5.3 Multi-dimensional kernels

In this section we develop intuitions on how to build kernels for higher-dimensional inputs. This section demonstrates what happens when we add or multiply kernels across dimensions (equation 5.7 and 5.8), while also providing kernels on how to perform sensitivity analysis or to encode lower-dimensional structure (equation 5.4). We then apply the multi-dimensional covariance function to interpolate aerodynamic pressure (section 5.3.5), comparing the accuracy of GP interpolation with common POD technique [Chiplunkar 2017a].

5.3.1 Adding across dimensions

Consider an input data-set which is multi-dimensional $\boldsymbol{x} \in \mathbb{R}^{D_{inputs}}$. A simple additive kernel can be constructed by adding the kernels for individual dimensions [Hastie 1990]. This operation encodes the information that added dimensions are independent of each

other (equation 5.16).

$$k(\boldsymbol{d}, \boldsymbol{\theta}) = \sum_{i=1}^{D_{inputs}} (\theta^i_{amplitude})^2 exp\left[-\frac{(d^i)^2}{2(\theta^i_{lengthScale})^2}\right] \tag{5.16}$$

Here, $d^i = x^i_1 - x^i_2$ is the distance between two input points at the i^{th} dimension. Figure 5.4(b) is a randomly drawn function after adding two SE kernels, the hyper-parameters of both the SE kernels are $\theta_{amplitude} = 1$ and $\theta_{lengthscale} = 0.2$.

5.3.2 Multiplying across dimensions

If we want to include interactions between two dimensions then their kernels can be multiplied together (equation 5.8). A kernel which allows for interaction between all the possible D_{inputs}-dimensions can be constructed by multiplying all kernels for all the dimensions. The multi-dimensional Automatic Relevance Determination (ARD) kernel (equation 5.17) can be seen as a multiplication of several one-dimensional kernels with different length-scales [Rasmussen 2005]. It is called ARD because the value of length-scale determines which dimensions are more relevant.

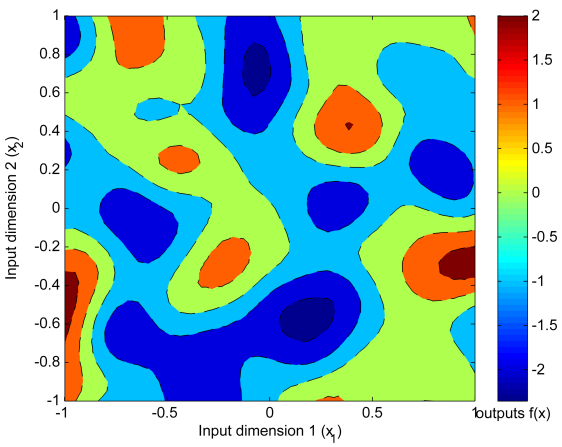
$$k(\boldsymbol{d}, \boldsymbol{\theta}) = (\theta_{amplitude})^2 \prod_{i=1}^{D_{inputs}} exp\left[-\frac{(d^i)^2}{2(\theta^i_{lengthScale})^2}\right] \tag{5.17}$$

Figure 5.4(a) is obtained after multiplying 2 SE kernels, the hyper-parameters of both the SE kernels are $\theta_{amplitude} = 1$ and $\theta_{lengthscale} = 0.2$

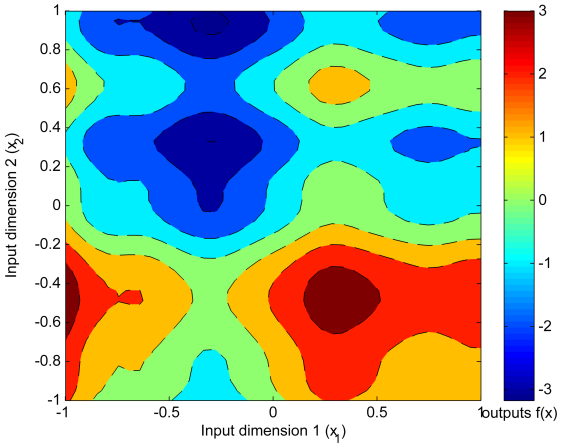
Matérn kernels have been found to have superior performance when compared to SE kernels on data-sets with high-dimensions [Le 2013]. It is argued that the Matérn kernel accounts for the concentration of measure effect in higher-dimensions. Since SE kernel is inverse Fourier of Gaussian density, high-dimensional samples of an SE kernel will be constrained to surface of the D_{inputs}-dimensional ellipse. This problem is less severe for a heavy tailed t-distribution power spectrum of Matérn kernel [Wilson 2014].

5.3.3 Sensitivity analysis

[Duvenaud 2011, Durrande 2013a, Chastaing 2015] define a class of additive kernels which are formed upon adding several low-dimensional interactions. Equation 5.18 is the basic form of covariance function which can be used to perform sensitivity analysis.



(a) Random draw from a 2 dimensional prior obtained after **multiplying** two SE kernels. The hyper-parameters of both the SE kernels are $\theta_{amplitude} = 1$ and $\theta_{lengthscale} = 0.2$



(b) Random draw from a 2 dimensional prior obtained after **adding** two SE kernels. The hyper-parameters of both the SE kernels are $\theta_{amplitude} = 1$ and $\theta_{lengthscale} = 1$

Figure 5.4: Random draw from a 2 dimensional prior.

$$k(\boldsymbol{x_1}, \boldsymbol{x_2}) = (\theta)^2 \prod_{i=1}^{D_{inputs}} \left(1 + k^i(x_1^i, x_2^i)\right) \tag{5.18}$$

The above kernel will include all the possible interactions across dimensions. For example for a 3-dimensional input space, the above kernel will include all the first order terms (equation 5.19), all the second order terms (equation 5.21), and the third order term (equation 5.22).

$$\begin{aligned}
k_{first-order}(\boldsymbol{x_1}, \boldsymbol{x_2}) =& k^1(x_1^1, x_2^1) + k^2(x_1^2, x_2^2) + k^3(x_1^3, x_2^3) && (5.19)\\
k_{second-order}(\boldsymbol{x_1}, \boldsymbol{x_2}) =& k^1(x_1^1, x_2^1) \times k^2(x_1^2, x_2^2) + k^2(x_1^2, x_2^2) \times k^3(x_1^3, x_2^3) && (5.20)\\
&+ k^3(x_1^3, x_2^3) \times k^1(x_1^1, x_2^1) && (5.21)\\
k_{third-order}(\boldsymbol{x_1}, \boldsymbol{x_2}) =& k^1(x_1^1, x_2^1) \times k^2(x_1^2, x_2^2) \times k^3(x_1^3, x_2^3) && (5.22)\\
& && (5.23)
\end{aligned}$$

These kinds of kernels can be used to analyze the sensitivity of interactions between various dimensions (also called as ANalysis Of VAriance, ANOVA).

5.3.4 Low dimensional structure

We can also encode a low dimensional structure into family of functions by specifying the kernel as $k_{low} = k(\boldsymbol{x_1 H}, \boldsymbol{x_2 H})$ (equation 5.4), here $\boldsymbol{H}$ is a low rank matrix.

$$k_{low}(\boldsymbol{d}, \boldsymbol{\theta}) = (\theta_{amplitude})^2 exp\left[-\frac{1}{2}\boldsymbol{d\Sigma d}^T\right] \tag{5.24}$$

Here, $\boldsymbol{\Sigma}$ is $\boldsymbol{HH}^T/(\theta^2_{lengthscale})$ which encodes the low dimensional structure. Figure 5.3.4 is obtained after encoding a low-dimensional into SE kernels, the hyper-parameters of the SE kernel are $\theta_{amplitude} = 1$ and $\theta_{lengthscale} = 0.2$ while $\boldsymbol{\Sigma} = \left(\begin{smallmatrix} 1 & 0 \\ -1 & 0 \end{smallmatrix}\right)$.

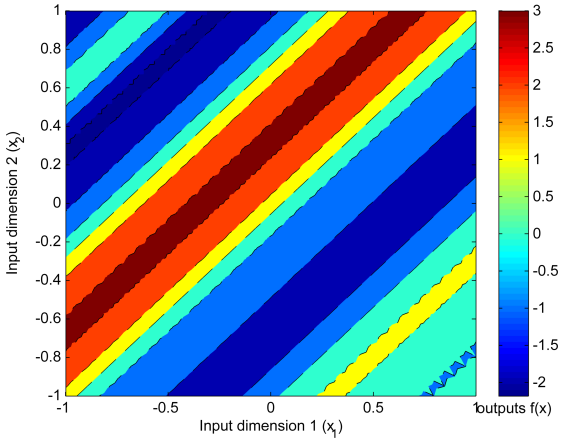


Figure 5.5: Random draw from a 2 dimensional prior which encodes a **low-dimensional** structure. The hyper-parameters of the kernel are $\theta_{amplitude} = 1$ and $\theta_{lengthscale} = 1$

When $\boldsymbol{\Sigma}$ is a diagonal matrix we get an ARD kernel

$$\begin{aligned} k(\boldsymbol{d}, \boldsymbol{\theta}) = (\theta_{amplitude})^2 exp\left[-\frac{1}{2}\boldsymbol{d\Sigma d}^T\right] &= (\theta_{amplitude})^2 exp\left[\sum_{i=1}^{D_{inputs}} -\frac{(d^i)^2}{2(\theta^i_{lengthScale})^2}\right] \\ &= (\theta_{amplitude})^2 \prod_{i=1}^{D_{inputs}} exp\left[-\frac{(d^i)^2}{2(\theta^i_{lengthScale})^2}\right] \end{aligned} \tag{5.25}$$

When the number of dimension increases significantly, the number of hyper-parameters also increases this makes the optimization of marginal likelihood inefficient. [Bouhlel 2016b] first reduce the dimensionality of the data-set and then perform interpolation thereby circumventing the problem of higher dimensions. [Garnett 2013, Tripathy 2016] use this covariance to reduce the dimensionality of input domain. *KPLS*

In the current section we have seen how to build covariance functions for multi-dimensional inputs. We now apply multi-dimensional kernels to build a surrogate model for Aerodynamic pressures. We validate our method on 2 test cases: the first in subsonic regime on a Flap Track Fairing (FTF) and the second in transonic regime on NASA's Common Research Model (CRM) Wing. The reader can also note that the results of these experiments were used during a recent Airbus Flight test campaign.

5.3.5 Application: Interpolation of aerodynamic pressures

Accurate prediction of aerodynamic pressures at a flight configuration is computationally expensive. Hence, it becomes advantageous to use surrogate models as approximations of high-fidelity aerodynamic models. A popular method of surrogate modelling in the aerodynamics community is by interpolating Reduced Order Models (ROM). A set of aerodynamic pressure snapshots is generated by performing CFD simulations for different aerodynamic parameters (eg. angle of attack α, Mach). Then orthogonal basis vectors are found in the parameter space for the set of pressures snapshots. Generally, Proper Orthogonal Decomposition (POD) [Tan 2003, Rosenbaum 2013, Braconnier 2011] (also called as Principal Component Analysis (PCA) or Singular Value Decomposition (SVD)) is used to find the linear subspace. Finally, the reduced models are interpolated at the desired point in the parameter-space [Beckert 2001, Barrault 2004].

Pressure snapshot

Let us first start by defining a pressure snapshot. There exists a 3 dimensional spatial vector $\boldsymbol{\omega_i} \in \mathbb{R}^3$ such that $\boldsymbol{\omega_i} = \{(\omega_i^1, \omega_i^2, \omega_i^3)\}$. Here, $i \in [1, N_{nodes}]$ are the spatial coordinates of the i^{th} pressure node in a CFD mesh containing N_{nodes} pressure nodes. Similarly there exists a D dimensional parameter vector $\boldsymbol{d_j} \in \mathbb{R}^D$, for $\boldsymbol{d_j} = \{(d_j^1, d_j^2, \ldots, d_j^D)\}$. Here, $j \in [1, N_{experiment}]$ correspond to the j^{th} parameter set, while D corresponds to the number of parameters. The parameters can be any parameters which are desired to be interpolated, some common examples include Mach, Angle of Attack for steady aerodynamics and time or frequency for unsteady aerodynamics. We will only concentrate on interpolating steady aerodynamics in this section.

The pressure measured on the i^{th} pressure node for the j^{th} parameter set will be denoted as $p_j(\boldsymbol{\omega_i})$ defined by equation 5.26. We next define the matrix $\boldsymbol{\Omega} = \{\boldsymbol{\omega_1}; \boldsymbol{\omega_2}; \ldots; \boldsymbol{\omega_{N_{nodes}}}\}$ for $\boldsymbol{\Omega} \in \mathbb{R}^{N_{nodes} \times 3}$ containing the full spatial information of the aerodynamic mesh. Finally, the pressure snapshot for the CFD/experiment run j will be denoted as $\boldsymbol{P_j(\Omega)} = \{p_j(\boldsymbol{\omega_1}); p_j(\boldsymbol{\omega_2}); \ldots; p_j(\boldsymbol{\omega_{N_{nodes}}})\}$ for $\boldsymbol{P_j(\Omega)} \in \mathbb{R}^{N_{nodes}}$ defined by the equation 5.27.

$$p_j(\boldsymbol{\omega_i}) = f_{pressure}(\boldsymbol{\omega_i}, \boldsymbol{d_j}) \tag{5.26}$$

$$\boldsymbol{P_j(\Omega)} = f_{pressure}(\boldsymbol{\Omega}, \boldsymbol{d_j}) \tag{5.27}$$

Here, $f_{pressure}$ symbolizes the aerodynamic process which when applied to a spatial location ($\boldsymbol{\omega}$) and an aerodynamic parameter ($\boldsymbol{d}$) gives us the pressures, we eventually wish to approximate this process ($f_{pressure}$). The POD methodology decomposes the set of pressure snapshots ($\boldsymbol{P_j(\Omega)}$) into their eigen vectors ($\boldsymbol{\phi^l(\Omega)}$) and participation factors ($a^l(\boldsymbol{d_j})$). To reconstruct the pressure snapshot at a new point $\boldsymbol{d_{new}}$, the participation factors are interpolated and then linearly combined to give the new pressure snapshot (equation 5.28). For more details please refer to appendix B

POD

$$\boldsymbol{P_{new}(\Omega)} = \sum_{l=1}^{p} a^l(\boldsymbol{d_{new}})\boldsymbol{\phi}^l(\boldsymbol{\Omega}) \tag{5.28}$$

Due to the assumption of linear subspace, interpolation through ROM is highly efficient both in terms of cost and performance in the subsonic regime [Verveld 2016]. Unfortunately in the transonic regime, the shock creates a highly non-linear, almost discontinuous pressure distribution and the assumption of linear subspace does not hold [Li 2016]. Although the performance of ROM interpolators can be improved with larger number of samples [Franz 2014, Forrester 2008], we propose to improve the accuracy of prediction using the Distributed GPs regression for the same number of samples.

Contribution

We interpolate the pressure $p_j(\omega_i)$ by simply multiplying the kernels across dimensions (equation 5.29). To interpolate in subsonic regime we multiply Matérn $\nu = 5/2$ across dimensions, a Matérn kernel is chosen since it does not suffer from concentration of measure effect in higher dimensions. We are not interested in linearly separating the effects of individual dimensions, hence a simple multiplicative kernel will suffice for this problem.

$$\Pr[p_j(\omega_i)] = GP\left(0, k(x_1, x_2) = \prod_{i=1}^{D_{inputs}} k_{Mat(\nu=5/2)}(x_1^i, x_2^i)\right) \tag{5.29}$$

The transonic regime often contains a shock on the surface of the wing. This shock almost creates a discontinuity in the pressure field. We know from section 4.3.2 that Neural Network kernels can represent these type of almost discontinuous functions in its hypothesis space. Hence to interpolate in transonic regime we use the Neural Network kernel in the dimension of shock. For example, if we know that the shock appears on the chord-wise direction, then we can represent the pressure as equation

5.30.

$$\Pr[p_j(\boldsymbol{\omega_i})] = GP\left(0, k(\boldsymbol{x_1}, \boldsymbol{x_2}) = k_{NN}(x_1^{chord}, x_2^{chord}) \times \prod_{i=1}^{D_{inputs}-1} k_{Mat(\nu=5/2)}(x_1^i, x_2^i)\right) \tag{5.30}$$

Here, x_i^{chord} represents the chord wise dimension of the i^{th} input, while k_{NN} and $k_{Mat\nu=5/2}$ represent the Neural Network and Matérn ($\nu = 5/2$) kernels respectively.

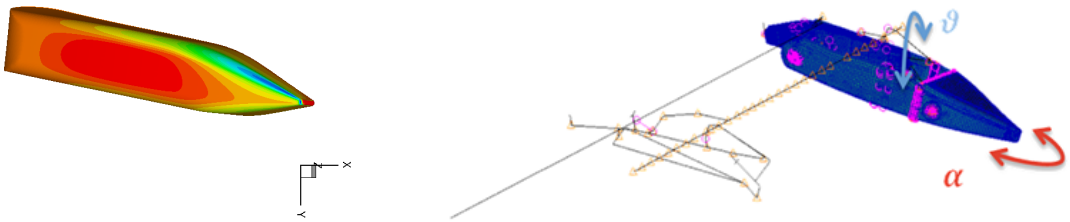
We now test the performance of Distributed GPs and POD+I on two sets of numerical experiments. Firstly, we test the accuracy on a detailed Flap Track Fairing (FTF) design [Bosco 2016] in subsonic regime (section 5.3.5) based on RANS simulation from the elsA solver [Cambier 2008]. Finally, we compare the accuracy on CRM wing in the transonic regime using the elsA solver and a k-Omega-SST turbulence model [Vassberg 2014]. elsA® [Cambier 2008] is a CFD simulation platform that allows representation of both internal and external aerodynamics from the low subsonic to the high supersonic flow regime.

Interpolation in subsonic regime

FTF

A FTF is situated below the wing and is used to deploy flaps for landing and take-off configurations. A FTF experiences heavy dynamic excitation due to the exhaust coming from engine. The dynamic nature makes the design of FTF a challenging task, where each simulation can last for 2 days. If we can effectively interpolate pressure snapshots then a 2 day dynamic simulation can be reduced to a few hours.

Interpolation of FTF pressure snapshots has been earlier studied using POD methodology [Bosco 2016]. In this section we use this data-set to validate the interpolation capabilities of a Distributed GPs algorithm.



(a) FTF Pressure snapshot (b) FTF parameters $\boldsymbol{d_i} = (\theta_i, \alpha_i)$

Figure 5.6: Details of the FTF

The FTF parameters chosen for this analysis are θ, the rotation around the longitudinal axis of the FTF, and α, the rotation around the *spigot* axis, main connection between the

track and the wing structure figure 5.6(b). The surface mesh of the CFD skin contains almost 36 thousand nodes (more precisely $N_{nodes} = 36802$). We run the simulation for 9 different values of α and 9 different values of θ ($N_{experiment} = 9 \times 9 = 81$). In this particular case Reynolds Averaged Navier-Stokes (RANS), equations are used with Spalart-Allmaras turbulence model to simulate the flow around the FTF. Figure 5.6(a) shows one particular pressure snapshot.

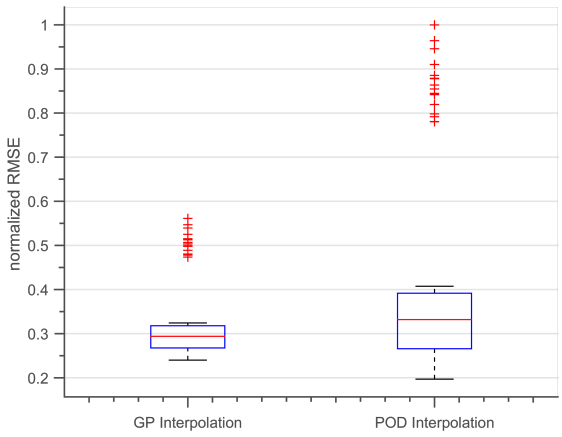
$\boldsymbol{d_i} = (\theta_i, \alpha_i)$

We use the Leave One Out (LOO) Cross Validation method to quantify the performance of the two methodologies. $[\alpha, \theta]$ pairs are removed one by one from the database to create a new training set. The new training set is used to perform interpolation according to POD+I and Distributed GPs. Pressure snapshot is reconstructed for the missing $[\alpha, \theta]$ pairs. The methods are finally compared by evaluating the Root Mean Square Error (RMSE) and time of prediction for each pair case.

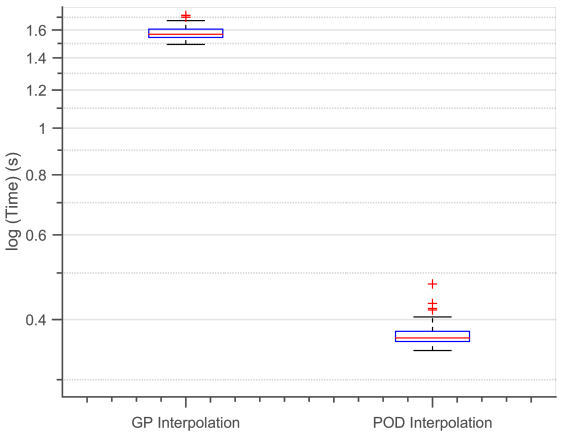
LOO-CV

While performing Distributed GPs regression, we learn a model between the input vector $\boldsymbol{x_{ij}} = [\omega_i^1, \omega_i^2, \omega_i^3, \alpha_j, \theta_j]$ and the pressures ($y_{ij} = p_{ij}$). We build a multi-dimensional kernel by multiplying 5 Matérn kernels, different length-scale for each dimension. As discussed earlier we propose to use Matérn kernel since the SE kernel has a very restrictive hypothesis space for high-dimensional regression. Effectively we are learning a GP model for $N = 36802 \times 80 = 2.9$ million data-points, there are 1000 points in each expert.

Distributed GPs



(a) Box plots of normalized RMSE for the two different model types. POD has a RMSE of 0.48 ± 0.27 whereas Distributed GPs has a RMSE of 0.37 ± 0.1. The mean Distributed GPs prediction is 20% better than POD. Outliers are extrapolation cases



(b) Time taken to perform prediction for the two different model types. POD takes $2.35s \pm 0.11s$ whereas Distributed GPs takes $37.84s \pm 4.35s$ to perform the interpolation. The average POD method is 19 times faster than Distributed GPs.

Figure 5.7: Results for elsA interpolation

Figures 5.7(a) denote the RMSE estimates for different pairs. For RMSE, the Distributed GPs performs marginally better than POD. The outliers in the box-plots denote cases where the removed pairs are on the edges of the domain. Since the removed pairs are

on the edges of the domain, there are no CFD snapshots surrounding these pairs. Hence we are extrapolating during the edge cases. POD has a RMSE of 0.48 ± 0.27 whereas Distributed GPs has a RMSE of 0.37 ± 0.1. The average Distributed GPs prediction is 20% better than POD. Figure 5.7(b) shows the time taken to perform prediction for the two model types. We are only comparing the time to perform interpolation, time to learn the models will be much longer. POD takes $2.35s \pm 0.11s$ whereas Distributed GPs takes $37.84s \pm 4.35s$ to perform the interpolation.

Figure 5.7

Although the GP technique can more efficiently capture non-linear behavior we see a relatively low improvement in performance for the amount of time invested. Deciding between the simple and time-tested POD method or costly and accurate Distributed GPs interpolation in the subsonic regime can be a delicate task and mostly depends on the preferences of the final user.

Interpolating in transonic regime

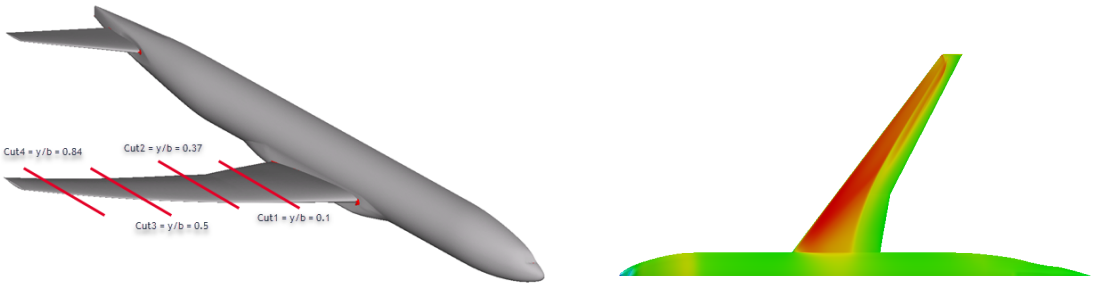
We now proceed to compare the accuracy of the two methods in transonic regime on the CRM wing model proposed by NASA. Since the introduction of the CRM for the 4th Drag Prediction Workshop [Vassberg 2014], the CRM has been become a very widely used test case for applied computational aerodynamics. Due to the widespread experience and availability of wind-tunnel test results for the CRM configuration, it is a natural case to benchmark interpolations.

Shocks

Due to the shape of an airfoil, airflow is accelerated on the upper surface of the wing. This causes shocks to appear on the upper surface of the wing in the transonic regime. Shocks are sudden changes (almost discontinuous) in pressure and are important for estimating the performance of the aircraft [Jameson 1974, Cole 2012]. Moreover, an aircraft flies in the transonic regime for 80% of its journey (cruise). Hence accurate prediction of location and strength of a shock is very important during design. Since POD is a linear subspace reduction method it has difficulties in interpolating a discontinuous shock regime [Verveld 2016].

Again we use the elsA® solver to perform simulations on the design. We use the $\kappa - \omega$ SST turbulence model to perform predictions since it has good performance in the fuselage wing interaction regions [Menter 2003, Vassberg 2014]. The CFD was run for a combination of 21 values of $\alpha \in [1 : 0.1 : 3]$ and 5 values of $Mach \in [0.84 : 0.005 : 0.86]$, hence $N_{experiment} = 21 \times 5 = 105$. Figure, 5.8(b) shows one of the pressure snapshots for $\alpha = 2$ and $Mach = 0.85$. We then cut the wing at four distinct locations $y/b = [0.105, 0.37, 0.5, 0.84]$ (figure 5.8(a)) to clearly observe different types of aerodynamic behavior. Here, y denotes the y-distance from aircraft axis and b denotes the span of one wing.

As detailed in the earlier section we again use LOO-CV for comparing the performance of the two methods. The POD+I method has been run as described in ap-



(a) CRM. The four red lines are the four cuts at $y/b = [0.105, 0.37, 0.5, 0.84]$. Here, y denotes the y-distance from aircraft axis and b denotes the span of one wing.

(b) Pressure snapshot on the CRM for $\alpha = 2$ and $Mach = 0.85$, using elsa solver and $\kappa - \omega$ SST turbulence model. We can observe double shock pattern appearing on the outer sections of the wing

Figure 5.8: CRM shape for aerodynamic simulations

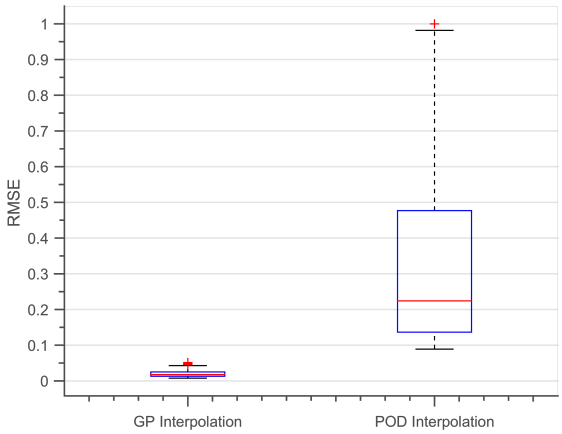
pendix B. For GP regression, we learn a GP model between $y_{ij} = p_{ij}$ and input vector $\boldsymbol{x_{ij}} = [x_i^{chord}, \alpha_j, Mach_j]$. We build a multi-dimensional kernel by multiplying 2 Matérn kernels (for α and $Mach$ dimensions) and one Neural Network kernel (for x^{chord} dimension). The shock will appear in the spatial dimension and hence using a Neural Network kernel in that dimension lets us capture the discontinuity more accurately.

Figure 5.9(a) denote the RMSE estimates for different pairs. POD has a RMSE of 0.32 ± 0.23 whereas Distributed GPs has a RMSE of 0.02 ± 0.01. The average Distributed GPs prediction is 16 times better than POD in transonic regime. The outliers in the box-plots denote cases where the removed pairs are on the edges of the domain. Since the removed pairs are on the edges of the domain, there are no CFD snapshots surrounding these pairs. Hence we are extrapolating on the outliers. Figure 5.9(b) shows the time taken to perform prediction for the three different model types. POD takes $0.5s \pm 0.03s$ whereas Distributed GPs takes $40.6s \pm 2.3s$ to perform the interpolation. In transonic regime GP has a significantly better error performance and becomes the obvious choice for interpolation.

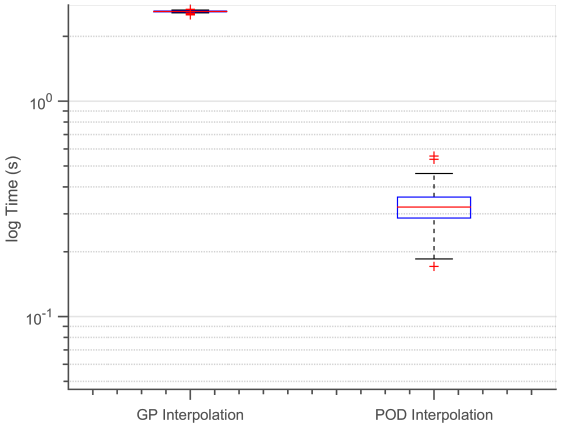
Figure 5.9

Figure 5.10(a) shows the comparison between predicted pressures using the POD method and the Distributed GPs for case of interpolation. Reconstruction is performed on the pressure snapshot at $\alpha = 2$ and $Mach = 0.85$ for the $y/b = 0.105$. We can observe that the shape of shock has been smoothed out by the POD method. Figure 5.10(b) shows comparison between POD method and Distributed GPs for extrapolation, the extrapolation is being performed for the aerodynamic parameter ($Mach$) and not the spatial parameter. Reconstruction is performed on the hidden pressure snapshot of $\alpha = 2$ and $Mach = 0.84$ which is an extrapolation case. We can observe that POD introduces errors both for the intensity of shock, and location of shock for the extrapolation case, this explains the high amount of error in figure 5.9(a).

Figure 5.10



(a) Normalized RMSE for the two different model types. POD has a RMSE of 0.32 ± 0.23 whereas Distributed GPs has a RMSE of 0.02 ± 0.01. The average Distributed GPs prediction is 16 times better than POD in transonic regime. The outliers in the box-plots denote cases where the removed pairs are on the edges of database. Since the removed pairs are on the edges of database matrix, there are no CFD snapshots surrounding these pairs. Hence extrapolation is performed during the edge cases.



(b) Time taken to perform prediction for the two different model types. POD takes $0.5s \pm 0.03s$ whereas Distributed GPs takes $40.6s \pm 2.3s$ to perform the interpolation. This is the time to perform prediction and not learning the model. The average POD method is 80 times faster than Distributed GPs.

Figure 5.9: Results for CRM interpolation

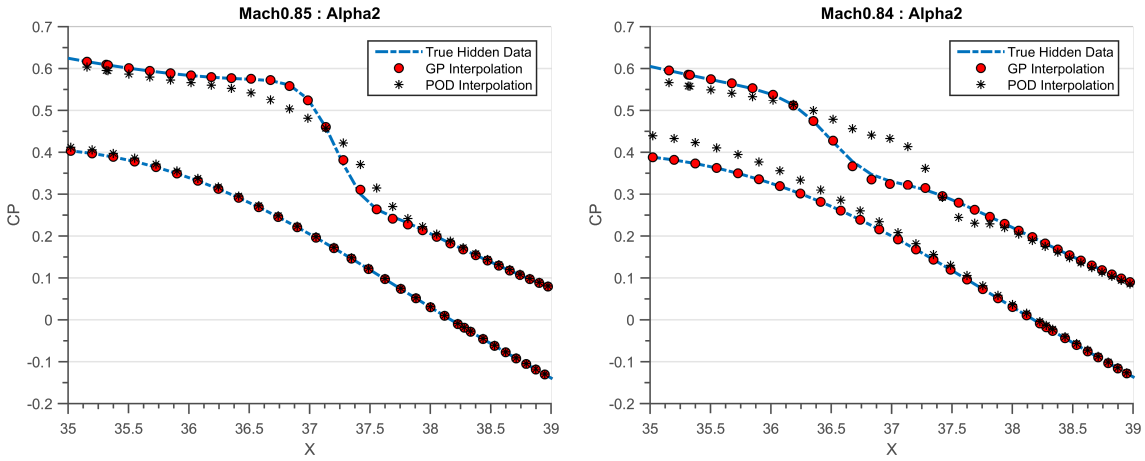
Comparison across cuts

We next study the accuracy of Distributed GPs for different airfoils on the wing. Using the methodology described earlier we build a Distributed GPs model for each airfoil and measure the accuracy of interpolation performed for each cut using the LOO-CV methodology.

Figure 5.11

Figure 5.11(a) shows the RMSE performance across cuts. The performance of interpolation deteriorates as we go further away from the fuselage. This is primarily because as we go further away from the fuselage double shocks start appearing on the airfoil as observable from figure 5.8(b). Figure 5.11(b) shows interpolation performed by the POD method and Distributed GPs methods at $\alpha = 2$ and $Mach = 0.85$, for the last cut ($y/b = 0.84$). While, POD smooths out the double shock pattern, Distributed GPs lacks the accuracy observed in figure 5.10(a).

Figures 5.12(a) and 5.12(b) show the evaluation of pressures upon varying $\alpha \in [1, 3]$ at locations $y/b = 0.105$ and $y/b = 0.84$ respectively. The color coding denotes coefficient of pressure for the upper side of airfoil, The x-axis denotes chord-wise location and y-axis denotes α. White lines denote presence of a pressure snapshot due to CFD run, everything



(a) Comparison between POD method and Distributed GPs for **interpolation**. The X-axis denotes chord dimension, only showing chord-section near shock for clarity. The Y-axis denotes the coefficient of pressure. Reconstruction is performed on the pressure snapshot at $\alpha = 2$ and $Mach = 0.85$ for the $y/b = 0.105$. We can observe that the intensity of shock has been smoothed out by POD method

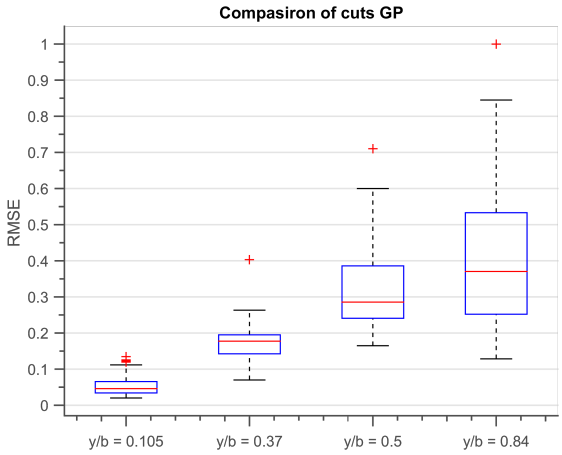
(b) Comparison between POD method and Distributed GPs for **extrapolation** (the extrapolation is being performed for the aerodynamic parameter ($Mach$) and not the spatial parameter). The X-axis denotes chord dimension, only showing chord-section near shock for clarity. The Y-axis denotes the coefficient of pressure. Reconstruction is performed on the pressure snapshot at $\alpha = 2$ and $Mach = 0.84$ for the $y/b = 0.105$. We can observe that POD introduces errors both for the intensity of shock and location of shock for this case

Figure 5.10: Comparison of pressure interpolations for first cut $y/b = 0.105$. Here we compare the accuracy of prediction for interpolation and extrapolation cases.

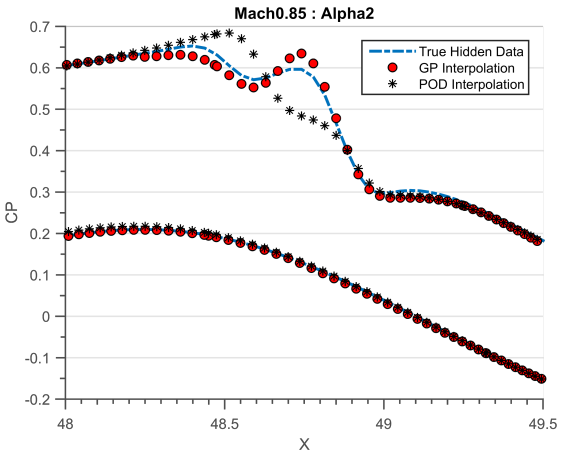
in between is interpolation. Dashed black lines denote constant pressure contours Color between two contours has been smoothed for clarity. For figure 5.12(a) we observe a strong shock near $\alpha = 3$ which slowly gets converted to a weak shock near $\alpha = 1$. The presence of a single shock is also the reason why Distributed GPs performs better at this cut location. For figure 5.12(b) we observe a single shock near $\alpha = 3$ which slowly gets converted to a double shock pattern. The zone from single to double shock is a very interesting point for performance, since the wing drag is minimum during this transition phase. Distributed GPs starts performing badly near the transition phases, this can be observed by the small pools of $C_P = 0.5$ at the transition phase from single to double shock.

Double shocks

The current section presents a comparison between two different surrogate model building methods: time-tested surrogate modelling methods such as POD coupled with spline interpolation and upcoming machine learning methods such as Distributed GPs for subsonic and transonic regimes. The Distributed GPs performs marginally better than POD technique in subsonic regime but is many times slower. On the contrary for the case of transonic regime Distributed GPs clearly outperforms POD+I method. This is mainly due



(a) Normalized RMSE for different airfoils based on Distributed GPs. The mean RMSE of different cuts from $y/b = [0.105, 0.37, 0.5, 0.84]$ is $[0.053, 0.17, 0.31, 0.40]$ respectively. The accuracy of interpolation deteriorates as we go farther away from the fuselage. This is due to appearance of double shock on the outer section of wing.



(b) Comparison between POD method and Distributed GPs for interpolation. The X-axis denotes chord dimension, only showing chord-section near shock for clarity. The Y-axis denotes the coefficient of pressure. Reconstruction is performed on the pressure snapshot at $\alpha = 2$ and $Mach = 0.85$ for the location $y/b = 0.84$. While POD smooths out the double shock pattern Distributed GPs also lacks the accuracy observed in figure 5.10(a).

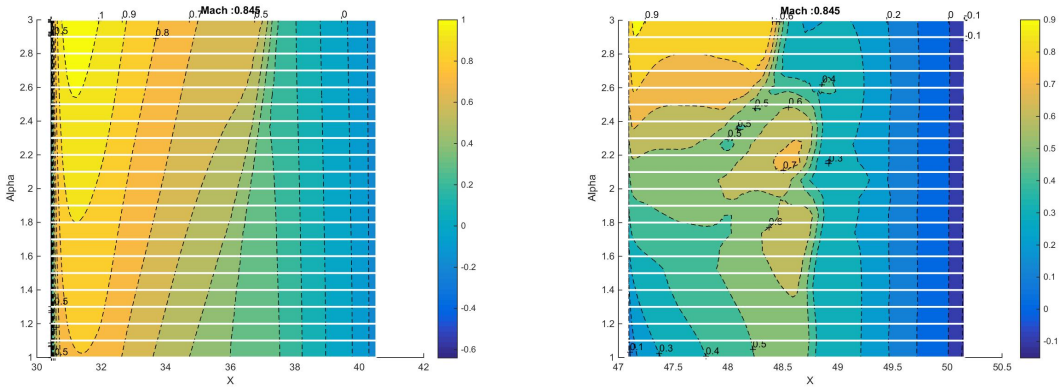
Figure 5.11: Performance of Distributed GPs across cuts

to the presence of shock on the airfoil.

Plots like figure 5.12(b) give a quick understanding of the flight regime for very few simulation runs. These plots can also be used to find transition regimes from single shock to double shock, which are very interesting performance points. In the future we wish to improve reconstruction of double shock patterns by improving the length scale and improve the choice of experts for Distributed GPs.

5.4 Summary and discussion

In the last two chapters we have tried to answer the question: How to add *a-priori* information of a pattern in a learning algorithm? We present a sample of the wide variety of available covariance functions. Due to the ability to create new kernels, encoding prior information into the structure can be performed easily. If we have an *a-priori* information about the pattern of function to be learned, then embedding this information into the GP algorithm greatly improves accuracy and cost of prediction. Table 5.2 lists a few commonly known combinations of covariance functions in the literature.



(a) Interpolation performed at constant $Mach = 0.845$ and $\alpha = [1, 3]$ for the location $y/b = 0.105$. The color coding denotes coefficient of pressure for upper side of airfoil, The x-axis denotes chord-wise location and y-axis denotes α. White lines denote presence of a pressure snapshot due to CFD run, everything in between is interpolation. Dashed black lines denote constant pressure contours color between two contours has been smoothed for clarity. We observe a strong shock near $\alpha = 3$ which slowly gets converted to a weak shock near $\alpha = 1$.

(b) Interpolation performed at constant $Mach = 0.845$ and $\alpha = [1, 3]$ for the location $y/b = 0.84$. The color coding denotes coefficient of pressure for upper half of airfoil, The x-axis denotes chord-wise location and y-axis denotes α. White lines denote presence of a pressure snapshot due to CFD run, everything in between is interpolation. Dashed black lines denote constant pressure contours color between two contours has been smoothed for clarity. We observe a single shock near $\alpha = 3$ which slowly gets converted to a double shock pattern.

Figure 5.12: Pressure reconstructions for constant $Mach = 0.845$ and sweeping $\alpha \in [1, 3]$

Model	Structure	Citation
Linear Regression	$k_{constant} + k_{linear} + k_{noise}$	
Polynomial	$k_{constant} + \prod k_{linear} + k_{noise}$	
Ordinary kriging	$k_{SE} + k_{noise}$	[Krige 1951]
Simple kriging	$k_{constant} + k_{SE} + k_{noise}$	
Universal Kriging	$\prod k_{linear} + k_{SE} + k_{noise}$	[Matheron 1963]
Multiple Kernel	$\sum k_{SE} + k_{noise}$	
Spectral Mixture	$\sum cosk_{SE} + k_{noise}]$	[Wilson 2013]
Change point	$\sum CP(k_{Lin}, k_{SE}) + k_{noise}]$	[Osborne 2010]
Additive GPs	$\prod_i (1 + k^i_{SE})]$	[Duvenaud 2011]

Table 5.2: Combination of covariance functions available in literature

Section 5.2 describes how to combine kernels to create one-dimensional kernels. While section 5.2.4 describes how to build kernels for higher dimensions. There are two main contributions of this chapter, first we develop a novel methodology to detect the start of

a non-linear regime in an engineering design problem. This is thanks to the change-point kernel which lets us define change from one regime to another [Chiplunkar 2016b]. Second, we encode the information of shock to interpolate pressures in the transonic regime. This strategy gives significant gains in accuracy when compared to the standard POD+I for aerodynamic interpolation [Chiplunkar 2017a].

In the next two chapters we will tackle the remaining questions posed in section 1.6 of the thesis. Chapter 6 discusses how to perform extrapolation given a computer simulation of experiments, while chapter 7 discusses how to encode prior information of relationships between measurements into a GP regression.

Part III

Incorporating multiple outputs in Gaussian Process Regression

Chapter 6

Multi-task learning for extrapolation

Résumé

Dans la partie III, nous nous intéressons à l'apprentissage de relations entre plusieurs sorties. Une régression sur les sorties multiples peut être divisé en deux catégories, lorsque nous souhaitons découvrir des relations sur plusieurs sorties, et lorsque nous voulons appliquer une relation connue entre les sorties (en tant que biais) dans notre algorithme d'apprentissage.

Dans ce chapitre, nous verrons comment effectuer la régression GP en présence de sorties multiples. Une méthode simple d'apprentissage d'un modèle de régression pour les sorties multiples est de supposer le nombre de sorties comme une dimension supplémentaire (cette dimension supplémentaire est une variable catégorielle). Cette hypothèse est bénéfique pour deux raisons, nous avons effectivement augmenté le nombre de points de données (figure 6.2.1), puis des fonctions de covariances pour plusieurs dimensions (section 5.3) peuvent être également appliquées au cas des sorties multiples (section 6.2.1).

Si aucune information n'est disponible sur la relation entre les sorties, nous pouvons apprendre la relation entre les sorties en utilisant une covariance 'simple MTGP' (section 6.2.2), un 'Linear Model Of Coregionalization' (section 6.2.3), ou un 'convoluted GP' [Alvarez 2011]. Si nous savons *a-priori* que les sorties proviennent de modèles de fidélité multiples, nous pouvons utiliser la fonction de covariance jointe proposée par [Kennedy 2000] (section 6.2.4).

Dans la section 6.4, nous avons montré comment les modèles GP multi-fidélité peuvent être utilisés pour effectuer une interpolation ou une extrapolation des données expérimentales du modèle. Nous incluons un terme d'erreur et un terme de translation

dans un modèle multi-fidélité normal pour tenir compte des différences entre le modèle de simulation et les données expérimentales. Dans des travaux futurs, nous souhaitons quantifier la performance des différentes méthodes d'extrapolation et décider de la méthode à choisir dans un scénario particulier.

6.1 Introduction

In the current part (part III) we are interested in learning relationships between multiple outputs. Performing regression on multiple-outputs can be split into two categories, first when we wish to discover relationships across multiple outputs, and second when we wish to enforce a known relationship between outputs (as a bias) into our learning algorithm.

Multi-Task Learning

When no relationship is known between outputs, we are interested in discovering these relationships. Several real-world problems often exhibit strong correlations between output variables, for example, correlations across spatial coordinates (x, y, z) in an experiment. This setting is also called 'Multi-Task Learning' in the machine learning literature. Learning relationships across outputs can improve the efficiency and prediction accuracy of the models when compared to training the outputs individually. This is because learning together outputs which are related, effectively increases the number of data-points, thereby providing more data for the learning algorithm [Caruana 1998]. In the GP literature these models are called 'Multi-Task GP' (MTGP) [Alvarez 2011, Bonilla 2008, Boyle 2005], while in the kriging literature they are called as Cokriging [Helterbrand 1994, Chilès 1999, Ver Hoef 1998].

Enforcing relationships

When a relationship between the outputs is known, enforcing such a relationship in a learning algorithm will mean that we have implemented a correct bias. The outputs can be results from computer codes of varying accuracy [Kennedy 2000, Forrester 2007, Le Gratiet 2013], they can be related through a known equation[1] [Ginsbourger 2013, Särkkä 2011], or they can be related through a known computer code[2] [Constantinescu 2013].

We split the part III of this thesis into two chapters. The current chapter (chapter 6) will describe the basics of MTGP, present a model to interpolate a series of multi-fidelity simulations, and then propose a method to extrapolate experimental data in the presence of simulation models. The next chapter (7) will describe how to encode known relationships between outputs, the most popular form of this model is called as Gradient Enhanced Kriging (GEK) in the literature.

The main contribution of this chapter is two-fold we first compare the accuracy of

[1] For example if we know that $acceleration = d(velocity)/d(time)$ and we measure both velocity and acceleration. How can we enforce the known equation into our model?

[2] For example stress and forces in a CSM code

prediction of different MTGP models. Secondly, we expand the multi-fidelity formulation provided by [Kennedy 2000] to the case of extrapolating experimental data using a simulation model. This type of model is useful during aircraft certification when there is a need to extrapolate experimental data.

The current chapter unfolds as follows, section 6.2 describes the various methods available in the literature to perform Multi-Task GP (MTGP). Section 6.2.4 describes MTGP model in presence of outputs coming from simulations with different accuracy. Section 6.3 then compares the prediction capabilities of a normal MTGP with multi-fidelity GP. Finally, section 6.4 describes our model of extrapolation for certification. This chapter is influenced from the works of [Forrester 2007, Alvarez 2011, Bonilla 2008, Boyle 2005, Kennedy 2000, Le Gratiet 2013].

6.2 Multi-task Gaussian Process

Suppose we have a D_{inputs}-dimensional input space and a $D_{outputs}$-dimensional ($D_{outputs} > 1$) output space.

$$\boldsymbol{x_i^j} = [x_{i1}^j, x_{i2}^j, \ldots, x_{iD_{inputs}}^j] \tag{6.1}$$

Notations

Here, x_i^j is the i^{th} input point of the j^{th} output, which is a horizontal vector with D_{inputs} components, each component representing a scalar value. The training data-set for the j^{th} output[3] can be then written as $\{(x_i^j, y_i^j)\}$ for $i \in [1; N_j]$. Here N_j is the number of measurement points for the j^{th} output, while $x_i^j \in \mathbb{R}^{D_{inputs}}$ and $y_i^j \in \mathbb{R}$.

$$\boldsymbol{X^j} = \begin{bmatrix} \boldsymbol{x_1^j} \\ \boldsymbol{x_2^j} \\ \vdots \\ \boldsymbol{x_{N_j}^j} \end{bmatrix} ; \quad \boldsymbol{y^j} = \begin{bmatrix} y_1^j \\ y_2^j \\ \vdots \\ y_{N_j}^j \end{bmatrix} \tag{6.2}$$

Here, $\boldsymbol{X^j}$ is the matrix containing all input points of the j^{th} output such that $\boldsymbol{X^j} \in \mathbb{R}^{N_j \times D_{inputs}}$. While $\boldsymbol{y^j}$ is the vector containing all the output values for the j^{th} output such that $\boldsymbol{y^j} \in \mathbb{R}^{N_j}$. If $\Sigma N_j = N_{joint}$ for $\forall j \in [1, D_{outputs}]$. Then N_{joint} represents the total number of training points for all the outputs combined

[3]The superscript is used to denote the 'number of output' and does not denote the power.

6.2.1 An extra dimension

One simple way of building a joint model for multiple outputs is by treating them as a single-output GP but with an extra dimension, the extra dimension denoting the output number (equation 6.3) [Osborne 2010].

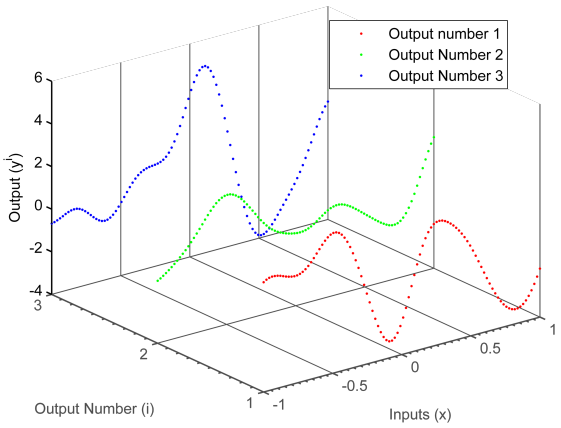


Figure 6.1: The outputs can also be represented as an extra dimension.

The figure 6.2.1 demonstrates how the different outputs can be treated as a single output coming from different dimensions. We can thus write the new input point as equation 6.3.

$$\boldsymbol{x_i^j} = [x_{i1}^j, x_{i2}^j, \ldots, x_{iD_{inputs}}^j, j] \tag{6.3}$$

Categorical variable

Note, the new input point has an extra dimension which denotes its output number. The extra dimension (j) is a categorical variable and hence has no measure of distance. Henceforth we define the joint output vector $\boldsymbol{y_{joint}}$ such that all the output values are stacked one after the other, while $\boldsymbol{y_{joint}} \in \mathbb{R}^{N_{joint}}$. Similarly, we define the joint input matrix as $\boldsymbol{X_{joint}}$, having one extra dimension representing the output number, such that $\boldsymbol{X_{joint}} \in \mathbb{R}^{N_{joint} \times (D_{inputs}+1)}$.

$$\boldsymbol{X_{joint}} = \begin{bmatrix} \boldsymbol{X^1}, \mathbb{1}_{\boldsymbol{N_1}} \times 1 \\ \boldsymbol{X^2}, \mathbb{1}_{\boldsymbol{N_2}} \times 2 \\ \vdots \\ \boldsymbol{X^{D_{outputs}}}, \mathbb{1}_{\boldsymbol{N_{D_{outputs}}}} \times \boldsymbol{D_{outputs}} \end{bmatrix}; \quad \boldsymbol{y_{joint}} = \begin{bmatrix} \boldsymbol{y^1} \\ \boldsymbol{y^2} \\ \vdots \\ \boldsymbol{y^{D_{outputs}}} \end{bmatrix} \tag{6.4}$$

Here, $\mathbb{1}_{\boldsymbol{N_j}}$ denotes a vector of ones to be multiplied with the 'number of output'. The size of $\mathbb{1}_{\boldsymbol{N_j}}$ is $1 \times N_j$, N_j is the number of data-points for the j^{th} output. Since the multi-output GP can be represented as a single output GP with an extra dimension, we can use all the kernel making techniques discussed in section 5.3 to the current problem. For a

case of 2 outputs we can write a zero mean GP prior for the full set of inputs and outputs $\{\boldsymbol{X_{joint}}, \boldsymbol{y_{joint}}\}$, as equation 6.5.

$$\begin{aligned}\Pr[\boldsymbol{y_{joint}}] &= \Pr\begin{bmatrix}\boldsymbol{y_{joint}}(x,1)\\ \boldsymbol{y_{joint}}(x,2)\end{bmatrix}\\ &= GP\left[\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}Cov(\boldsymbol{y_{joint}}(x,1),\boldsymbol{y_{joint}}(x,1)) & Cov(\boldsymbol{y_{joint}}(x,1),\boldsymbol{y_{joint}}(x,2))\\ Cov(\boldsymbol{y_{joint}}(x,2),\boldsymbol{y_{joint}}(x,1)) & Cov(\boldsymbol{y_{joint}}(x,2),\boldsymbol{y_{joint}}(x,2))\end{pmatrix}\right]\end{aligned} \tag{6.5}$$

Here, $Cov(\boldsymbol{y_{joint}}(x,j),\boldsymbol{y_{joint}}(x,j))$ is called the auto-covariance between observations of the j^{th} output, while $Cov(\boldsymbol{y_{joint}}(x,2),\boldsymbol{y_{joint}}(x,1))$ is called cross-covariance between the 2^{nd} and the 1^{st} output. Once we have the functional forms of the covariance functions, we can equivalently calculate the Gram matrix ($\boldsymbol{K_{XX}}$), by replacing the values of individual input points. The upcoming sections describe different forms of covariance functions between outputs and their corresponding family of functions.

Equation 6.5

The sections 6.2.2 and 6.2.3 describe two simple covariance function forms when we are interested in discovering relationship between outputs, while section 6.2.4 describes how to impose prior information of multi-fidelity among outputs.

6.2.2 Simple Multi-task kernel

One simple method to calculate the auto and cross-covariance functions is by simply multiplying the covariance functions for the inputs dimensions with the covariance functions for the output dimension (equation 6.6), refer to [Bonilla 2007] for more clarity .

$$Cov(\boldsymbol{y_{joint}}(x_1,a),\boldsymbol{y_{joint}}(x_2,b)) = k_{output}(a,b) \times k_{input}(x_1,x_2) \tag{6.6}$$

There exists one issue though, the extra dimension is a categorical variable and hence has no measure of distance, i.e. the difference between two output numbers a and b is not defined (equation 6.6). [Bonilla 2007] define the covariance across outputs as equation 6.7, this makes the matrix $\boldsymbol{K_{output}}$ positive definite and hence a valid covariance matrix. $k_{output}(a,b)$ denotes the (a^{th}, b^{th}) coordinate of the matrix $\boldsymbol{K_{output}}$.

$$\boldsymbol{K_{output}} = \boldsymbol{K_{lower}}\boldsymbol{K_{lower}^T} \tag{6.7}$$

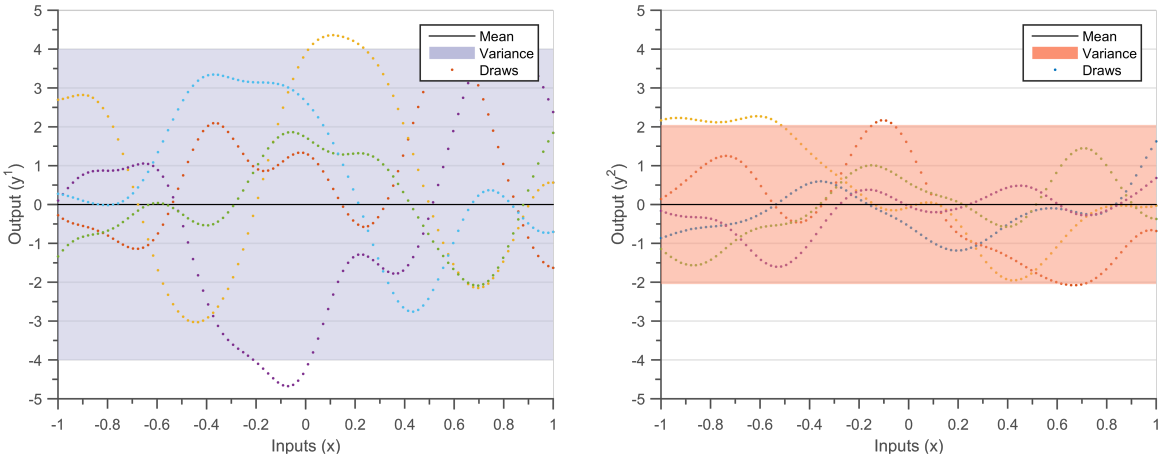
The hyper-parameters of above prior are the parameters of the lower triangular matrix L, and parameters of the input covariance function ($k_{input}(x_1^a, x_2^b)$). If $\boldsymbol{K_{output}}$ is a diagonal matrix, then there is no cross-covariance across outputs, this implies that the outputs are

independent of each other. In such a kernel design there is no transfer of information between outputs [Bonilla 2007, O'Hagan 1998].

If we measure the output 'a' at points $\boldsymbol{X^1}$ and output 'b' at not necessarily the same points $\boldsymbol{X^2}$, then we can write the Gram matrix for the above prior (equation 6.6) as equation 6.8.

$$\boldsymbol{K_{XX}} = \begin{bmatrix} k_{output}(a,a) \cdot \boldsymbol{K}_{input}(\boldsymbol{X^1}, \boldsymbol{X^1}) & k_{output}(a,b) \cdot \boldsymbol{K}_{input}(\boldsymbol{X^1}, \boldsymbol{X^2}) \\ k_{output}(b,a) \cdot \boldsymbol{K}_{input}(\boldsymbol{X^2}, \boldsymbol{X^1}) & k_{output}(b,b) \cdot \boldsymbol{K}_{input}(\boldsymbol{X^2}, \boldsymbol{X^2}) \end{bmatrix} \tag{6.8}$$

Here, '$\cdot$' denotes the multiplication between a scalar and a matrix.



(a) Five randomly drawn functions for output number 1, using the MTGP kernel proposed by [Bonilla 2007]

(b) Five randomly drawn functions for output number 2, using the MTGP kernel proposed by [Bonilla 2007]

Figure 6.2: Joint draws from a simple multi-task kernel function between outputs y^1 and y^2. The matrix $\boldsymbol{K_{output}} = [4, 0.2; 0.2, 1]$ while the covariance function between inputs $k_{input}(x_1, x_2)$ is a SE kernel with hyper-parameters ($\theta = [1, 0.2]$). The solid black line defines the mean function, shaded region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior. We can observe that figure 6.3 varies faster when compared to figure 6.2 due to smaller length scale hyper-parameter in one of the latent functions.

Figure 6.2 is a joint draw from such a kernel function between outputs y^1 and y^2. The matrix $\boldsymbol{K_{output}} = \left(\begin{smallmatrix} 4 & 0.2 \\ 0.2 & 1 \end{smallmatrix}\right)$ while the covariance function between inputs $k_{input}(x_1, x_2)$ is a SE kernel with hyper-parameters ($\theta = [1, 0.2]$). We can observe that the variance of output y^1 is twice that of output y^2, this is because $\sqrt{k_{output}(1,1)/k_{output}(2,2)} = 2$.

6.2.3 Linear Model of Coregionalization

The next method considers the output variables as a linear combination of independent latent GPs. It is called the Linear Model of Coregionalization in kriging literature [Goovaerts 1997] or Semiparametric Latent Factor Model in MTGP literature [Seeger 2005]. Latent GPs are random variables which are not directly observed.

Suppose we have a set of L latent GPs $U(x) = \{u^1(x), u^2(x), \ldots, u^L(x)\}$, where $u^i(x)$ is a GP with covariance $k_u^i(x_1, x_2)$. Any linear combination of $u^i(x) \quad \forall i \in L$ is a viable GP (section 5.2.2), therefore we can define the GP of output y^j as equation 6.9.

$$y^j(x) = \sum_{i=1}^{L} \alpha^{ij} u^i(x) \tag{6.9}$$

The covariance function can thus be written as equation 6.10.

$$\begin{aligned} Cov(\boldsymbol{y_{joint}}(x_1, a), \boldsymbol{y_{joint}}(x_2, b)) &= Cov(\sum_{i=1}^{L} \alpha^{ia} u^i(x_1), \sum_{i=1}^{L} \alpha^{ib} u^i(x_2)) \\ &= \sum_{i=1}^{L} \alpha^{ia} \alpha^{ib} Cov(u^i(x_1), u^i(x_2)) \\ &= \sum_{i=1}^{L} \alpha^{ia} \alpha^{ib} k_u^i(x_1, x_2) \\ &= \sum_{i=1}^{L} \boldsymbol{K}^{\boldsymbol{i}}_{\boldsymbol{output}}(a, b) \cdot k_u^i(x_1, x_2) \end{aligned} \tag{6.10}$$

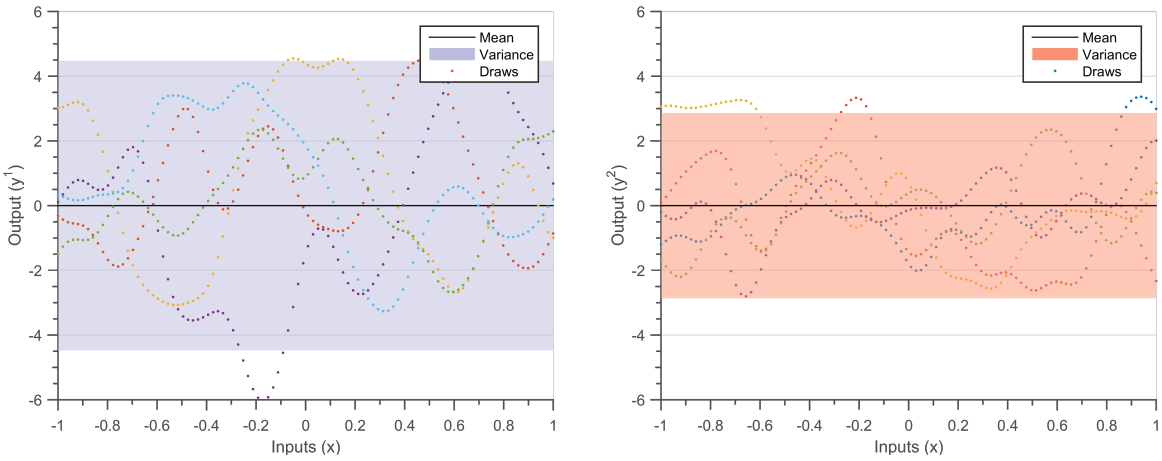
The covariance between the output dimension $\boldsymbol{K}^{\boldsymbol{i}}_{\boldsymbol{output}}$ is also called the coregionalization matrix, and is of size $D_{outputs} \times D_{outputs}$. It can be written as equation 6.11, making it a positive definite matrix [Mercer 1909].

$$\boldsymbol{K}^{\boldsymbol{i}}_{\boldsymbol{output}} = \begin{bmatrix} \alpha^{i1} \\ \alpha^{i2} \\ \vdots \\ \alpha^{iL} \end{bmatrix} \cdot \begin{bmatrix} \alpha^{i1} & \alpha^{i2} & \ldots & \alpha^{iL} \end{bmatrix} \tag{6.11}$$

Note, when $L = 1$ and $D_{outputs} > 1$ the Linear Model of Coregionalization is equivalent to the model proposed by [Bonilla 2007]. While, when $L > 1$ and $D_{outputs} = 1$ the Linear Model of Coregionalization model resembles an additive covariance of T individual covariances (section 5.2.2).

Figure 6.3

Figure 6.3 shows 3 joint draws from a 'Linear Model of Coregionalization' kernel function between outputs y^1 and y^2. We use two latent functions for this figure; the kernels between output dimensions are $\boldsymbol{K}^1_{\boldsymbol{output}} = \left(\begin{smallmatrix} 4 & 0.2 \\ 0.2 & 1 \end{smallmatrix}\right)$ and $\boldsymbol{K}^2_{\boldsymbol{output}} = \left(\begin{smallmatrix} 1 & 0.2 \\ 0.2 & 1 \end{smallmatrix}\right)$, while the covariance function between inputs are SE kernel with hyper-parameters $(\theta = [1, 0.2])$ for the first latent process and $(\theta = [1, 0.1])$ for the second latent process. The length-scale of the functions is determined by the smallest length-scale of the latent processes.



(a) Five randomly drawn functions for output number 1, using the MTGP kernel proposed by Linear Model of Coregionalization

(b) Five randomly drawn functions for output number 2, using the MTGP kernel proposed by Linear Model of Coregionalization

Figure 6.3: Joint draws from a 'Linear Model of Coregionalization' kernel function between outputs y^1 and y^2. We use two latent functions for figure; the kernels between output dimensions are $k^1_{output} = [4, 0.2; 0.2, 1]$ and $k^2_{output} = [1, 0.2; 0.2, 1]$, while the covariance function between inputs are SE kernel with hyper-parameters $(\theta = [1, 0.2])$ for the first latent process and $(\theta = [1, 0.1])$ for the second latent process. The solid black line defines the mean function, shaded region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior. We can observe that figure 6.3 varies faster when compared to figure 6.2 due to smaller length scale hyper-parameter in one of the latent functions.

6.2.4 Multi-fidelity GP

Simulation models in engineering design often come with different orders of accuracy or fidelity. The increase in accuracy is generally associated to an increase in cost. This is because capturing the higher order effects means either more non-linear models or finer meshes. This creates an opportunity to reduce the overall cost of accurate predictions by reducing the number of runs of the costly, high-fidelity computation and launching several low-fidelity computations. We can treat the low-fidelity and high-fidelity predictions as

cases with multiple outputs. The prior information that we have here is that one output is more precise than another.

[Kennedy 2000, O'Hagan 1998] propose the first Cokriging model for multi-fidelity calculations. This method has been used extensively to make cheaper surrogate models and perform cheap optimization of a costly code [Forrester 2007, March 2012]. We will describe the methods proposed by [O'Hagan 1998] in this section.

Suppose we have $D_{outputs}$ simulation models $y^i(x)\ \forall\ i \in D_{outputs}$, ordered with increasing levels of accuracy. Since these are simulations of a computer code there is no noise in the outputs. [O'Hagan 1998] use the Markov property (equation 6.12) between a higher fidelity code $y^i(x)$ and lower fidelity code $y^{i-1}(x')$ for all $x \neq x'$. This assumption means that no further information about $y^i(x)$ can be extracted from lower level models if we have access to simulation results at the same input point.

$$Cov(y^i(x), y^{i-1}(x') \mid y^{i-1}(x)) = 0 \tag{6.12}$$

The Markov property assumption means that if there are two subsequent fidelity models $y^i(x)$ and $y^{i-1}(x)$. Then the high-fidelity model $(y^i(x))$ can be represented as a linear combination of the low-fidelity model $(y^{i-1}(x))$ and a model that represents the difference between the two $(\delta^i(x))$, thereby linearly seperating the effects of the two models. It can be expressed equivalently as the set of equations below.

Linearly separable

$$y^i(x) = r^{i-1}(x)y^{i-1}(x) + \delta^i(x) \tag{6.13}$$
$$y^{i-1}(x) \perp \delta^i(x) \tag{6.14}$$

Here, $r^{i-1}(x)$ is a continuous function in $x \in \mathbb{R}$, $\perp$ signifies independence between two GPs $(Cov(y^{i-1}(x), \delta^i(x)) = 0)$. If we have 2 sets of simulation results with varying fidelity $y^2(x)$ more accurate than $y^1(x)$, the above model gives rise to the following joint-covariance structure.

$$\begin{aligned} \boldsymbol{K_{XX}} &= \begin{bmatrix} k^1(x^1, x^1) & r(x^1)k^1(x^1, x^2) \\ r(x^2)k^1(x^2, x^1) & r(x^2)^2k^1(x^2, x^2) + k^\delta(x^2, x^2) \end{bmatrix} \\ &= \begin{bmatrix} k^1(x^1, x^1) & r(x^2)k^1(x^1, x^2) \\ r(x^1)k^1(x^2, x^1) & r(x^2)^2k^1(x^2, x^2) \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & k^\delta(x^2, x^2) \end{bmatrix} \end{aligned} \tag{6.15}$$

Here, x^1 are the input points for simulation y^1, and x^2 are the simulation points for output y^2, $k^1(x^1, x^2)$ is the covariance function for output y^1. [Kennedy 2000] have used a constant function for the value of $r(x)$ and an SE kernel for the covariances $k^1(x, x')$ and $k^\delta(x, x')$.

The model of multi-fidelity GPs in equation 6.15 is equivalent to a 'Linear Model of Coregionalization' model with two latent processes. The fist process has $\boldsymbol{K}^1_{\boldsymbol{output}} = \left(\begin{smallmatrix} 1 & r \\ r & r^2 \end{smallmatrix}\right)$ as covariance between outputs and $k^1_{inputs} = k^1$ as the covariance between inputs, while the second process has $\boldsymbol{K}^2_{\boldsymbol{output}} = \left(\begin{smallmatrix} 0 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ as the covariance between outputs and $k^2_{inputs} = k^\delta$ as the covariance between inputs. We have thus linearly seperated the effects of both the models, such that the difference between the two is captured by the $k^\delta(x, x')$ covariance function.

Recursive model

Multi-fidelity GPs are expensive due to the cost of calculating the precision matrix. [Le Gratiet 2013] propose a recursive model for performing multi-fidelity, this model performs the learning for individual computer codes thereby breaking the Gram matrix into smaller sizes and reducing the computational complexity. They also extend the Cokriging model to several number of multiple outputs.

6.2.5 Posterior distribution

For simplicity let us take the case of two outputs y^1 and y^2. Suppose we measure the two outputs with some error (ϵ_{n1} and ϵ_{n2}), while the true physical process is defined by latent variables (f^1 and f^2). Then the relation between the output function, measurement error, and true physical process can be written as follows.

$$y^1 = f^1 + \epsilon_{n1} \tag{6.16}$$
$$y^2 = f^2 + \epsilon_{n2} \tag{6.17}$$

Here, ϵ_{n1} and ϵ_{n2} are measurement errors sampled from a white noise gaussian $\mathcal{N}(0, \sigma^2_{n1})$ and $\mathcal{N}(0, \sigma^2_{n2})$ respectively, then the joint error matrix can be denoted by $\boldsymbol{\Sigma}$;

$$\Sigma = \begin{bmatrix} \sigma^2_{n1} \times \mathbb{I}_{\boldsymbol{N_1}} & 0 \\ 0 & \sigma^2_{n2} \times \mathbb{I}_{\boldsymbol{N_2}} \end{bmatrix} \tag{6.18}$$

Noise model

Here, σ^2_{nj} is the variance of the measurement error sampled from a white noise Gaussian and $\mathbb{I}_{\boldsymbol{N_j}}$ is an identity matrix of size N_j (N_j are the number of data-points for j^{th} output). If we assume a zero mean GP for the above type of covariance functions (sections 6.2.2, 6.2.3 and 6.2.4). The prior for a noisy multi-task learning case can be written as equation 6.19.

$$\Pr[\boldsymbol{y_{joint}}(\boldsymbol{X_{joint}}))] = GP(0, \boldsymbol{K_{XX}} + \boldsymbol{\Sigma}) \tag{6.19}$$

If we want to make a prediction for the i^{th} output at the test point $\boldsymbol{x_*}$ ($\boldsymbol{X_*} = [\boldsymbol{x_*}, i]$).

Then the full joint prior can be expressed as equation 6.20.

$$\begin{bmatrix} \boldsymbol{y_{joint}}(\boldsymbol{X})) \\ \boldsymbol{f}(\boldsymbol{X_*})) \end{bmatrix} = GP\left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{bmatrix} \boldsymbol{K_{XX}} + \boldsymbol{\Sigma} & \boldsymbol{K_{XX_*}} \\ \boldsymbol{K_{X_*X}} & \boldsymbol{K_{X_*X_*}} \end{bmatrix}\right] \tag{6.20}$$

The posterior mean and variance, conditioned on the data-set based on the prior (equation 6.5) can then be derived as the set of equations below .

$$E[\boldsymbol{f}(\boldsymbol{X_*}))] = \boldsymbol{K_{X_*X}} \left(\boldsymbol{K_{XX}} + \boldsymbol{\Sigma}\right)^{-1} \boldsymbol{y_{joint}} \tag{6.21}$$

$$Cov(\boldsymbol{f}(\boldsymbol{X_*}))), \boldsymbol{f}(\boldsymbol{X_*}))) = \boldsymbol{K_{X_*X_*}} - \boldsymbol{K_{X_*X}} \left(\boldsymbol{K_{XX}} + \boldsymbol{\Sigma}\right)^{-1} \boldsymbol{K_{XX_*}} \tag{6.22}$$

Posterior

Here, the elements $\boldsymbol{K_{XX}}$, $\boldsymbol{K_{X_*X}}$ and $\boldsymbol{K_{X_*X_*}}$ are block covariances derived from equations 6.5. The joint-covariance matrix depends on several hyper-parameters $\boldsymbol{\theta}$. The MTGP prior has a greater number of hyper-parameters when compared to single output GP. This is due to the need to define hyper-parameters for the coregionalization matrix. We maximize the log-marginal likelihood to find a set of good hyper-parameters. This leads to an optimization problem where the objective function is given by equation 6.23

$$\log(\Pr[\boldsymbol{y_{joint}} \mid \boldsymbol{X}, \boldsymbol{\theta}]) = \log[GP(\boldsymbol{y_{joint}}|0, \boldsymbol{K_{XX}} + \boldsymbol{\Sigma})] \tag{6.23}$$

Scalability?

Calculating the posterior distribution and fine-tuning the hyper-parameters for an MTGP is similar to a Single Output GP (chapter 2). The only thing different is the assumption that the output can be expressed as an extra dimension, resulting in a change of structure of covariance matrix and increasing the number of hyper-parameters. By writing a joint-prior for multiple outputs we have effectively increased the size of the Gram matrix. This further increases the burden on the scalability of MTGP. We will discuss the scalability solutions to MTGP in chapter 7.

6.3 Experiments

Let us revisit the data-set $\mathcal{D}_2$ used to calculate the posterior in section 2.3 and 4.4.3, but here we will update the data-set to have two different outputs, let us call this data-set $\mathcal{D}_4$. We compare the accuracy of prediction on the data-set ($\mathcal{D}_4$), for independent kernels (chapter 2), simple multi-task kernels (section 6.2.2), and multi-fidelity kernels (section 6.2.4).

$$f^1(x) = \frac{sin(5\pi x)}{5\pi x} \tag{6.24}$$

$$f^2(x) = f^1(x)/2 + 0.1cos(f^1(x)) \tag{6.25}$$

The Matlab code 6.1 is how we have created the data-set $\mathcal{D}_4$ for our upcoming experiment.

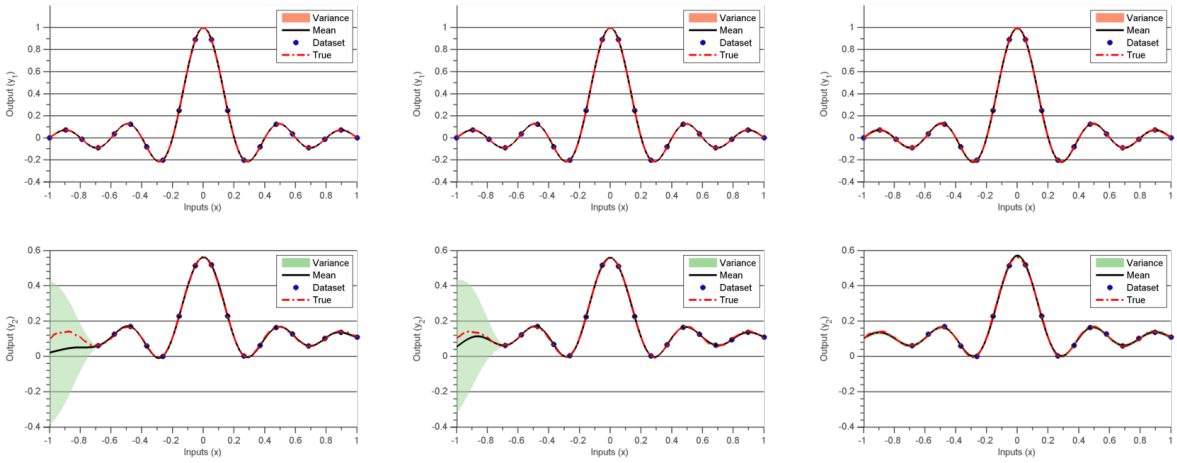
```
nData = 20; % number of data-points

f1 = @(x)sin(5*pi*x)./(5*pi*x); % Function
f2 = @(x) f1(x)/2 + 0.1*cos(f1(x));

noise = 0.1; % Noise in data-set
xData = linspace(-1, 1, nData)';
% full set of input points
xFull = [xData, ones(nData, 1); xData, 2*ones(nData, 1)];
% full set of output points
yFull = [f1(xData); f2(xData)];
```

Matlab Code 6.1: Code for data-set D4

To build the models we follow the standard work-flow of GP regression; we first define a zero mean prior using a preferred covariance function and define a hypothesis space, we then calculate the posterior distribution conditioned on data-set $\mathcal{D}_4$, and finally optimize the marginal likelihood to compare the final predictions of the three covariance functions.

Figure 6.4

Figure 6.4 shows the posterior distribution using the three covariance functions. The solid black line defines the mean function, while the shaded region defines 95% confidence interval (2σ) distance away from the mean. The output y^1 and y^2 are evaluated at $N_j = 20$ equidistant points, while the points in between $x^2 = [-1, -0.75]$ are removed from the second output data-set. Figure 6.4(a) shows the posterior when the two outputs are assumed to be independent of each other. Figure 6.4(b) shows the posterior when the two outputs are related through a multi-task covariance function, while the figure 6.4(c) shows the posterior when the outputs are related through a multi-fidelity covariance kernel. The mean prediction for the multi-fidelity covariance is best, followed by the simple multi-task kernel and then by the independent GP.



(a) GP posterior for **Independent outputs** conditioned on the data $\mathcal{D}_4$, we choose an SE kernel as the individual covariances of the individual outputs. The points where data is unavailable has high value of variance and bad mean prediction.

(b) GP posterior for a **simple multi-task** kernel conditioned on the data $\mathcal{D}_4$, we choose an SE kernel as the covariance across input points ($k_{inputs} = k_{SE}$). The points where data is unavailable has high value of variance but a better mean prediction when compared to figure 6.4(a)

(c) GP posterior for a **multi-fidelity** kernel conditioned on the data $\mathcal{D}_4$. We choose a SE kernel as the covariance across input points ($k_{inputs} = k_{SE}$) and covariance for δ process, the value of $r(x)$ is kept constant. We can observe that, the prediction at missing points is more accurate.

Figure 6.4: Joint Posterior distribution for two outputs with missing data at $x^2 = [-1 : -0.75]$. The solid black line defines the mean function, shaded region defines 95% confidence interval (2σ) distance away from the mean.

We now compare the accuracy of the three methods for an increasing number of input points. To measure the accuracy of the prediction, we again use the 10-fold Cross Validation (CV).

Figure 6.5 shows the accuracy of the three methods with an increasing number of data-points. The box-plots in red are cases when outputs are assumed independent, the box-plots in green are cases when covariance is assumed to be multi-task, and the box-plots in blue are cases when the output covariance is assumed to be multi-fidelity. As expected the error improves with the increasing number of data-points, the multi-fidelity covariance is best for all the three cases.

Linearly separable

The independence assumption cannot learn the relationship between the outputs hence has the highest value of error. The multi-task covariance has better accuracy than the independent case, although it still cannot match the performance of the multi-fidelity model. Multi-fidelity covariance model works so efficiently because the formula of the second output. The second output is a combination of a multiplication term to the first

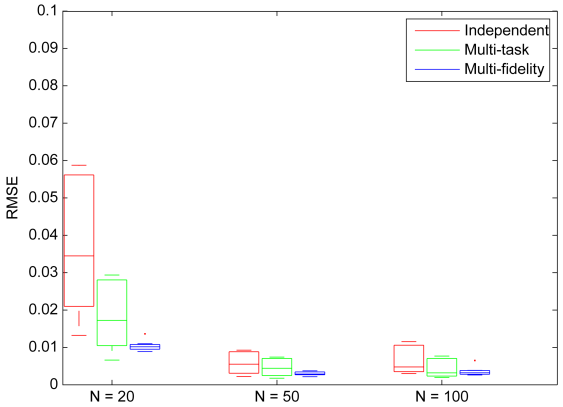


Figure 6.5: 10-fold RMSE box-plots for increasing number of input points. The box-plots in red are cases when outputs are assumed independent, the box-plots in green are cases when covariance is assumed to be multi-task, and the box-plots in blue are cases when the output covariance is assumed to be multi-fidelity. The error improves with increasing data-points,the multi-fidelity covariance is best fir for all the three cases.

output $f^2/2$, and a new term $0.1cos(f^1(x))$. This exactly matches the linearly separable assumptions in the multi-fidelity covariance function, on the other hand the multi-task model cannot effectively capture the new term $0.1cos(f^1(x))$.

6.4 Extrapolation of Flight Loads

Certification

Once an aircraft design is ready and a flight-test aircraft is manufactured, it becomes the task of the aircraft manufacturer to certify the airplane. This section is dedicated to using GP algorithms to assist in the task of flight-loads certification. For certification of flight-loads, it is necessary to show that the loads (aerodynamic forces) experienced by the aircraft are coherent with the loads predicted during the design phase. To achieve this we perform flight tests with varying aircraft parameters and get a proper understanding of how the loads are varying with respect to various parameters.

Figure 6.6

Figure 6.6 is an example of the prediction of Shear Load (Tz) on a section of the wing, for varying values of Mach and Load factor (Nz). The points in blue are the points where tests have been performed, the black line denotes the convex hull of the flight test points. The 3d surface is the prediction of a GP model built by using measurements on the input points and an SE kernel. The boundaries denoted by the red lines are the limit load cases, flying near the limit conditions will result in irreversible damage to costly flight-test aircraft. The data is normalized according to Airbus standards.

Verification and validation of the engineering model using experimental data of flight-test is a difficult exercise because:

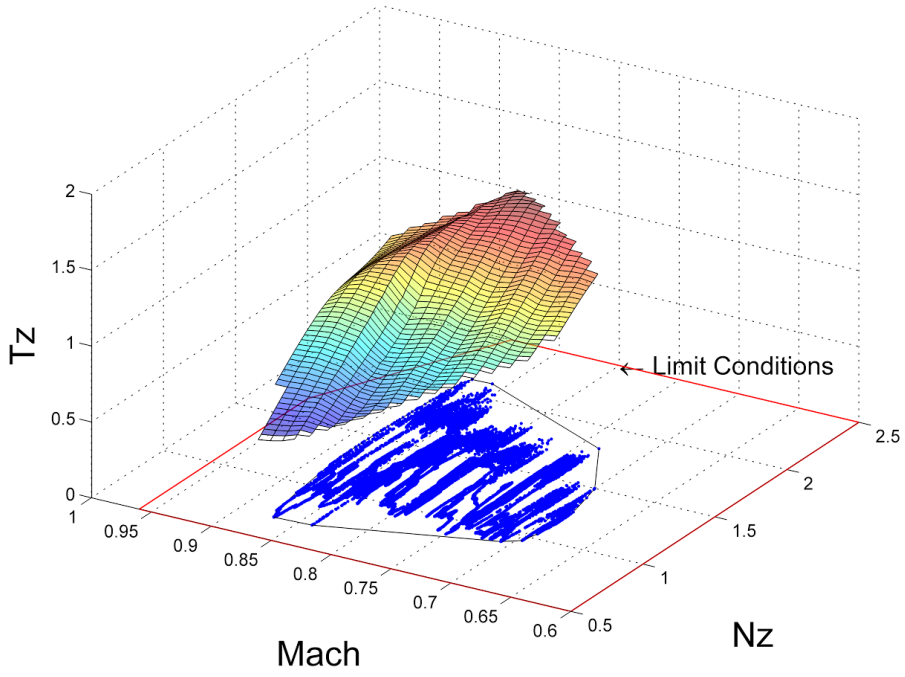


Figure 6.6: Flight-loads identification

1. Each flight-test campaign costs millions of euros, and hence flying the aircraft in all of the parameter combinations is very costly.

2. We cannot fly the aircraft at limit conditions because it will cause irreversible degradation to our single test-aircraft. But we still need to prove that the loads experienced at these limit conditions fall in the safety limit as prescribed by the certification authorities.

Hence we need to both interpolate the experimental results in the convex hull of flight-test points, while also being able to extrapolate until the red line for certifying the safety limits. In this section, we propose a method to perform extrapolation using the engineering model developed during the detailed design phase. We would like to extend the MTGP formulation for extrapolation cases.

6.4.1 Current Approach

Theoretical loads model

During the design phase of the aircraft, the flight loads department comes up with a theoretical model as shown in figure 6.7. The aerodynamic department starts from a set of inputs comprising of aircraft states (for example angle of attack, Mach etc) and aircraft geometry (for example deformation of control surfaces), and runs CFD simulations on these set of inputs. Since we cannot run CFD simulations on each and every set of parameters in

the flight domain, another aerodynamics team tries to build a parametric surrogate model which can map any set of aircraft states into rigid aerodynamic loads. This surrogate model is built by combining several types of aerodynamic computations, wind tunnel tests, and results in a lookup table model consisting of 100 aerodynamic parameters. In parallel the inertial loads department, uses the structural model to calculate the interial loads for same aircraft states. Finally, these two load models are combined to give the final load predictions.

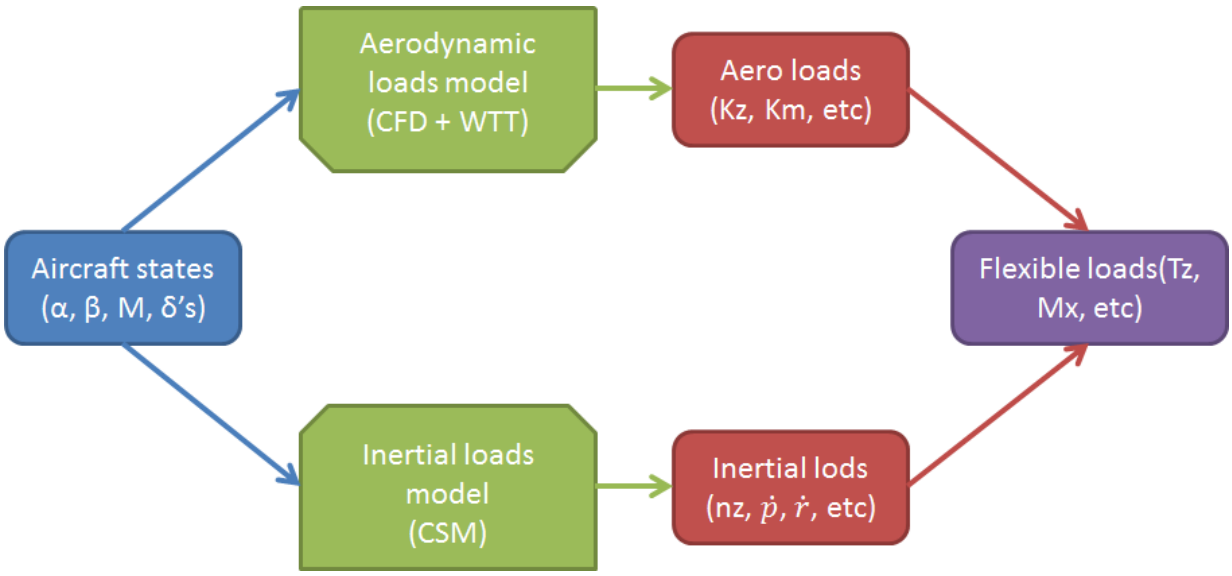


Figure 6.7: Current Flight-Loads theoretical model

Complexity

To update the simulation model, the measured data from flight-test is compared to the simulation predictions. The differences between the two models are minimized by minimizing the least square error. There are around 100 ($D_{outputs}$) load outputs (Tz, Mx, Kz etc), which are dependent on around 30 (D_{input}) flight parameters (Mach, the angle of attack etc). The parametric model has 100 variables and the flight tests are performed at millions of data-points. Updating this massively complex model is a highly complicated problem, requiring dozens of engineers and 6 months of certification time. Moreover, there are two main problems with this approach:

1. Due to the parametric nature of the surrogate model, the updated model performs perfectly in certain areas of the flight domain while giving bad results in other parts.

2. Due to the above issue many a times while updating the theoretical model, we add and remove certain parameters. CHoosing which parameters to update and which parameters to add, requires a team of 5-10 engineers working for 6 months to finally deliver the updated model.

6.4.2 Proposed Approach

Non-parametric

The main issue while using a parametric model is that, it uses parameters to express a function between input and outputs. This means to be able to express non-representable functions we iteratively add parameters in the model. We wish to replace this model updating phase with a MTGP model. We have access to both a relatively cheap simulation model produced during the design phase and costly flight-test experiments. Using the multi-fidelity formulation described in section 6.2 we can encode the simulation model as a prior information. This helps in providing the correct bias and using the years of hard work encoded into the simulation model as a prior information for learning the flight-test model.

The figure, 6.4.4 shows a common example of the difference between a simulated model and an experimental model. It shows predictions of a simulated model of the coefficient of lift, with respect to the angle of attack (in green). The points in red denote the measurements from flight-test.

6.4.3 Proposed method

There are currently three methods to perform extrapolation:

1. **Build a flight-test surrogate model**: We can build a GP model purely using data from flight-test. Unfortunately, the simple stationary covariance functions do not have any extrapolation capabilities (chapter 4). This means we would have to either use a Spectral Mixture to automatically detect patterns [Wilson 2014] or iteratively adjust the covariance function to detect patterns [Duvenaud 2014]. Unfortunately, flight-test data is very costly and thus we might not have access to a huge amount of data to learn detailed patterns. Moreover, by not using the information of a simulation model we are effectively putting years of hard-work into waste.

2. **Update the simulation model**: This is the standard methodology used by design team during the certification process. The results of flight-test are compared to the results of prediction of a simulation model. A simulation model has several parameters, which are fine-tuned to decrease the difference between flight-test data and simulation results. This is a costly and time-consuming process, requiring a team of several dozen engineers working on a tight certification schedule.

3. **Use multi-fidelity modelling**: The final option is using the multi-fidelity approach proposed in section 6.2.4. If we assume the simulation model as a low-fidelity model, and flight-test data as results from a high-fidelity model, we can use launch the simulations at limit conditions and use the multi-fidelity formulation to perform predictions. This is equivalent to encoding the simulation model as the prior information.

Suppose we have a D_{inputs}-dimensional input space, and two outputs (f^s, y^e) which represent the simulation model and flight-test data respectively. The experiments are performed at the input points x^e for $x^e \in \mathbb{R}^{N_e \times D_{inputs}}$, while the simulation model can be evaluated exactly at all points in the input domain.

$$\Pr[f^s(x)] = GP(\bar{f^s}(x), k^s = 0) \tag{6.26}$$

Here, $\bar{f^s}(x)$ signifies evaluation of the simulation model at input point x, since the simulation model is deterministic the covariance term (k^s) is to zero. We can also signify the experimental observations as:

$$y^e(x) = f^e(x) + \epsilon_{n_e} \tag{6.27}$$

Here, $f^e(x)$ is the latent physical process of the flight-test, while ϵ_{n_e} is experimental error. For the current case let's assume the experimental error is sampled from a white noise Gaussian , $\epsilon_{n_e} \sim \mathcal{N}(0, \sigma^2_{n_e})$. We can therefore, write the joint prior between the experimental measurement (y^e) and simulation model (f^s) as:

$$\Pr\begin{bmatrix} f^s \\ y^e \end{bmatrix} = GP\left(\begin{bmatrix} \bar{f^s}(x) \\ r(x^e) * \bar{f^s}(x) \end{bmatrix}, \begin{bmatrix} k^s(x^s, x^s) & r(x^s)k^s(x^s, x^e) \\ r(x^e)k^s(x^e, x^s) & r(x^e)^2 k^s(x^e, x^e) + k^\delta(x^e, x^e) + \sigma^2_{n_e} \times I_{N_e} \end{bmatrix}\right) \tag{6.28}$$

Since, the simulation model is deterministic, the covariance of the simulation model is zero ($k^s = 0$).

$$\Pr\begin{bmatrix} f^s \\ y^e \end{bmatrix} = GP\left(\begin{bmatrix} \bar{f^s}(x) \\ r(x^e) * \bar{f^s}(x) \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & k^\delta(x^e, x^e) + \sigma^2_{n_e} \times \mathbb{I}_{N_e} \end{bmatrix}\right) \tag{6.29}$$

Since the cross-covariance between simulation model and experimental data is zero, the two GPs are independent. We can thus write the GP prior for only the experimental data as equation 6.30.

$$\Pr[y^e] = GP\left(r(x^e) * \bar{f^s}(x), \quad k^\delta(x^e, x^e) + \sigma^2_{n_e} \times \mathbb{I}_{N_e}\right) \tag{6.30}$$

The above form of the GP prior means that, the modified form of the simulation model can be treated as the mean of the GP prior, while the k^δ learns the difference between the two models. This result is exactly similar to recursive multi-fidelity model [Gratiet 2012] since we have broken the joint GP formulation into smaller parts.

Latency

Often times when we are performing measurements we may have latency or translation among the measurements and predictions. This latency can be taken into account by introducing a delay hyper-parameter (Δx) [Osborne 2008]. This means that our proposed prior can be written as:

$$\Pr[y^e] = GP\left(r(x^e - \Delta x) * f^s(x \bar{-} \Delta x), \quad k^\delta(x^e, x^e) + \sigma_{n_e}^2 \times \mathbb{I}_{N_e}\right) \tag{6.31}$$

This is equivalent to assuming a Markov property as equation 6.32, $\forall x' \neq x - \Delta x$.

$$Cov(y^i(x), y^{i-1}(x') \mid y^{i-1}(x - \Delta x)) = 0 \tag{6.32}$$

If again we use a simple covariance function for k^δ then extrapolation of the covariance term will again tend to zero. This means that we still need to detect the pattern of the difference between experimental data (y^e), and simulation model ($r(x^e - \Delta x) * f^s(x \bar{-} \Delta x)$), but we have significantly reduced its need by applying the prior information of the simulation model.

6.4.4 Experiments on toy data-set

We now present a comparison of prediction accuracy between independent learning, multi-fidelity learning and proposed model in section 6.2.4. We have used a elsA solver to simulate a NACA 0012 airfoil, the model uses SST turbulence model and the Coefficient of lift (Cl) values are measured at steady points. The points in green in figure 6.4.4 represent the results of the simulation for varying angle of attack, all the other variables (Mach, Reynolds number etc) are kept constant for this exercise.

To represent the experimental data-set we make the following changes, equation 6.33 represents the case when there is no difference between experiment and simulation, equation 6.34 represents the case when there is a measurement noise, equation 6.35 represents the case when the simulation has not captured a sinusoidal component in the measurements, and equation 6.36 represents the case when there is a translation in the measurement.

$$y^e(x) = f^s(x) \tag{6.33}$$
$$y^e(x) = f^s(x) + \epsilon_n \tag{6.34}$$
$$y^e(x) = f^s(x) + sin(f^s) + \epsilon_n \tag{6.35}$$
$$y^e(x) = f^s(x + 0.2) + \epsilon_n \tag{6.36}$$

Here ϵ_n represents an independent noise, sampled from a white noise Gaussian $\epsilon_n \sim$

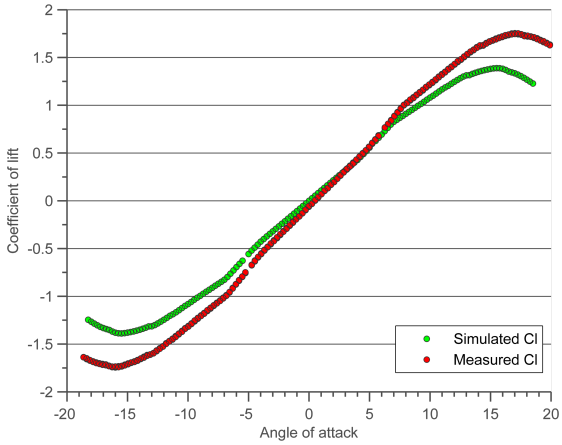


Figure 6.8: Example of difference between simulated Coefficient of lift (f^s) and experimental Coefficient of lift (y^e) (coming from equation 6.36), with respect to angle of attack.

$\mathcal{N}(0, 0.001)$. To measure the accuracy of predictions, we again launch a 10-Fold CV and calculate the RMSE. The 10-Fold CV has been performed for all the four types of changes between the measurement and experimental error. The hyper-parameters are fine-tuned for each of the three models (independent, multi-task and proposed model) by maximizing the marginal likelihood.

10-Fold CV

Figure 6.9 shows the box-plots of RMSE's for all the four cases. We see that the independent learning method always performs worst when compared to multi-task learning or proposed model with translations. When error starts appearing in the model then proposed model starts outperforming the multi-task method (there is no hyper-parameter for noise in the multi-task model). When there is a translation component present in the measurement then the proposed model clearly outperforms the two models.

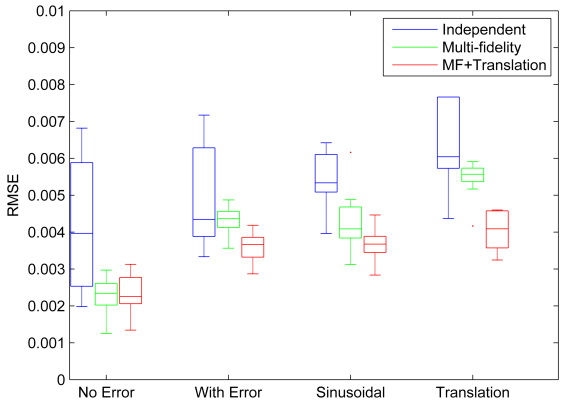


Figure 6.9: Boxplots of extrapolation for the four different cases; experimental set with no error (equations 6.33), with a white noise error (equations 6.35), with a sinusoidal error (equations 6.35) and with translation (equations 6.36).

The main aim of the current experiment is to present how the simulation model can be used for extrapolating experimental measurements. In our case, it is done by adding a translation hyper-parameter (Δx) to compensate for the lag in measurement and an error covariance to model the measurement error. We would also like to point out that the proposed model works better only because there is a translation term in the third case (equation 6.36).

Non-separable

Often times the different models cannot be linearly separable by the Markov property. This leaves us with three choices, we can try and identify the pattern in experimental data by using pattern detecting covariance functions [Wilson 2014]. Second, we can use a non-separable MTGP joint-covariance to learn the relation between the simulation model and the experimental model [Alvarez 2011], or third, we can learn the pattern of the k^{δ} automatically [Duvenaud 2013]. We wish to study the prediction properties of all the three proposed approaches and try to understand which is the better choice for tasks of model updating and extrapolation in the future.

6.5 Summary and discussion

In the current chapter, we have seen how to perform GP regression in the presence of multiple outputs. One simple method of learning a regression model for multiple outputs is by assuming the output number as an extra dimension (this extra dimension is a categorical variable). This observation is beneficial due to two reasons, first, we have effectively increased the number of data-points (figure 6.2.1). Second, all the tricks of making covariance functions for multiple dimensions can be applied to the case of multiple outputs as well (section 6.2.1).

If no information is available about the relation between outputs then we can learn the relationship between outputs by either using a simple MT covariance (section 6.2.2), a 'Linear Model of Coregionalization' (section 6.2.3), or a convoluted GP [Alvarez 2011]. If we know *a-priori* that the outputs are coming from models of multiple fidelity then we can use the joint covariance function proposed by [Kennedy 2000] (section 6.2.4).

In section 6.4 we have shown how multi-fidelity GP models can be used to perform interpolation or extrapolation of experimental data. We include an error term and a translation term into a normal multi-fidelity model to account for differences in the simulation model and experimental data. In the future, we would wish to quantify the performance of different methods of extrapolation and decide which method to choose in a particular scenario.

Chapter 7

Adding relationships in GP Regression

Résumé

Dans ce chapitre, nous considérons le problème de la régression par GPs à plusieurs sorties corrélées par les lois physiques d'un système. Ces lois physiques peuvent être représentées sous la forme d'un ensemble d'équations ou d'un code informatique (CFD ou CSM par exemple).

Un exemple usuel d'application des relations *a-priori* connues entre les sorties est appelée le 'Gradient Enhanced Kriging' dans la littérature sur le Krigeage [Chung 2002, Morris 1993, Forrester 2009] ou appelée 'GP regression with derivative observations' dans la littérature sur les GPs [Solak 2003]. La relation "dérivée" est appliquée en modifiant la fonction de covariance jointe (équation 6.5). Dans ce chapitre, nous étendons le modèle du 'Gradient Enhanced Kriging' aux intégrales, aux fonctions quadratiques ou à n'importe quelle fonctionnelle entre les sorties [Constantinescu 2013].

Le codage des lois physiques rend le modèle appris compatible avec la physique du système, ce qui constitue un avantage majeur par rapport aux modèles purement générés par les données [Jazwinski 2007]. La contribution de ce chapitre est double ; nous démontrons d'abord l'efficacité de l'ajout de relations dans un GP, à la fois sur un base des données synthétiques et sur un ensemble de données normalisées d'essais en vol. Les résultats ont été présentés lors d'une conférence AIAA en 2016 [Chiplunkar 2016a]. Après, nous démontrons comment appliquer la régression GPs avec plusieurs sorties à une grande quantité de données, en utilisant à la fois une inférence variationnelle et une 'Distributed GPs'. Les résultats de cette étude ont été publiés dans ICPRAM 2016 et LNCS 2017 [Chiplunkar 2016c, Chiplunkar 2017c].

7.1 Introduction

In this chapter, we consider the problem of modelling multiple output Gaussian Process (GP) regression correlated through physical laws of the system. Some example of physical laws can be 'Newtons Laws of Motion', 'Navier Stokes equation' or the 'Heat dissipation equation'. These physical laws have been developed after centuries of observations and iterative experiments. They can be represented in the form of a set of equations or through a computer code (CFD, CSM codes). This chapter discusses how to encode such relationships between outputs in an MTGP regression framework.

Contribution

Encoding physical laws makes the learned model consistent with the physics of the system, which is a major advantage when compared against pure data generated models [Jazwinski 2007]. The contribution of this chapter is two-fold, we first demonstrate the effectiveness of adding relationships in GP, on both a toy data-set and a normalized flight-test data-set. The results were presented in AIAA aviation conference 2016 [Chiplunkar 2016a]. Second, we demonstrate how to scale MTGP to a large amount of data, using both Variational inference and Distributed GPs. The results of this study were published in ICPRAM 2016 and LNCS 2017 [Chiplunkar 2016c, Chiplunkar 2017c].

Enhanced kriging

One popular example of enforcing the prior known relations between outputs is the "Gradient Enhanced Kriging" in Kriging literature [Chung 2002, Morris 1993, Forrester 2009] or "GP regression with derivative observations" in GP literature [Solak 2003]. The derivative relationship is enforced by changing the joint-covariance function (equation 6.5). In this chapter, we extend the framework of gradient enhanced kriging to integral enhanced kriging, quadratic enhanced kriging or any functional relationship between outputs [Constantinescu 2013].

The current chapter unfolds as follows. Section 7.2.1 describes how to enforce linear relationships between outputs, while section 7.2.2 describes how to encode non-linear relationships. In section 7.4 various methods of scaling multi-output GP are demonstrated. Finally, in section 7.5 we demonstrate the approach on both theoretical and flight-test data. This chapter has been motivated by the works of [Constantinescu 2013, Alvarez 2009, Särkkä 2011, Jidling 2017, Ginsbourger 2013, Särkkä 2011].

7.2 Encoding relationships in GP

For simplicity let us take the case of an explicit relationship between two outputs y^1 and y^2. Suppose we measure the two outputs with some error (ϵ_{n1} and ϵ_{n2}), while the true physical process is defined by latent variables (f^1 and f^2). Then the relation between the output function, measurement error, and true physical process can be written as follows[1].

[1]Reminder the superscript denotes the number of output and does not denote the power

$$y^1 = f^1 + \epsilon_{n1} \tag{7.1}$$
$$y^2 = f^2 + \epsilon_{n2} \tag{7.2}$$

Here, ϵ_{n1} and ϵ_{n2} are measurement errors sampled from a white noise Gaussian $\mathcal{N}(0, \sigma_{n1}^2)$ and $\mathcal{N}(0, \sigma_{n2}^2)$ respectively. While, the relation between the latent function can be expressed as follows:

$$f^1(z) = \mathcal{L}\left(f^2(x), z\right) \tag{7.3}$$

Here $\mathcal{L}(.) \in \mathcal{C}^2$ is an operator defining the relation between f^1 and f^2. We are thus, interested in evaluating the Gram matrix $\boldsymbol{K_{XX}}$ between the full data-set.

$$\boldsymbol{K_{XX}} = \begin{pmatrix} Cov(f^1(x), f^1(x)) & Cov(f^1(x), f^2(x)) \\ Cov(f^2(x), f^1(x)) & Cov(f^2(x), f^2(x)) \end{pmatrix} \tag{7.4}$$
$$= \begin{pmatrix} \boldsymbol{K_{11}} & \boldsymbol{K_{12}} \\ \boldsymbol{K_{21}} & \boldsymbol{K_{22}} \end{pmatrix} \tag{7.5}$$

We will denote the Gram matrices for auto-covariances as $\boldsymbol{K_{ii}}$ and cross-covariances as $\boldsymbol{K_{ij}}$ for $i \neq j$. In next two sections, we assume one output function to be an independent (f^2) random variable and evaluate the remaining auto-covariance ($Cov(f^i(x), f^i(x))$) and cross-covariance ($Cov(f^i(x), f^j(x))$ for $i \neq j$) functions exactly, if the physical relation between them is linear or use an approximate joint-covariance, if the relationship between them is non-linear[Constantinescu 2013]. $\boldsymbol{K_{XX}}$

7.2.1 Case of Linear Operators

Physical laws of the system are generally denoted by sets of differential equations which explain the basic behaviour of the system, these differential equations are linear in nature. One simple example of a linear operator is an integration operation, $f^1(z) = \int_\infty^z f^2(x)dx$.

The physical laws can be enforced by manipulating the covariance function between outputs ($Cov(f^i(x), f^j(x))$). [Alvarez 2009, Särkkä 2011] have used these forms of covariance functions to solve partial differential equations, [Jidling 2017] have used these covariances to impose linearity constraints in observations of magnetic experiments, while [Ginsbourger 2013] have used them to encode transformations in a random process. We give an overview of their work in this section.

If $\mathcal{L}(.)$ is a linear operator then it has the following property:

$$\mathcal{L}(a_1 g^1 + a_2 g^2) = a_1 \mathcal{L}(g^1) + a_2 \mathcal{L}(g^2) \tag{7.6}$$

Here, a_1 and a_2 are scalars and g^1 and g^2 are functions. If we assume a GP prior over the function f^2 such that its mean (m^2) and covariance functions (k^2) can be represented as the set of equations below. Note that m^2 is the mean of the output number 2 and not the square of its mean.

$$E[f^2(x)] = m^2(x) \tag{7.7}$$
$$Cov[f^2(x), f^2(x')] = k^2(x, x') \tag{7.8}$$

Then after applying the operator $\mathcal{L}(.)$, the mean of the function f^1 can be written as equation 7.9. From now on we will use a short-hand notation for operation $\mathcal{L}\left(f^2(x), z\right)$ as $\mathcal{L}\left(f^2, z\right)$.

$$\begin{aligned} E[f^1(z)] &= E[\mathcal{L}\left(f^2, z\right)] \\ &= \mathcal{L}(E[f^2], z) \\ &= \mathcal{L}(m^2, z) \end{aligned} \tag{7.9}$$

Similarly, the covariance of the function (f) can be written as $Cov[f^1(z), f^1(z')]$ (equation 7.10).

$$\begin{aligned} Cov[f^1(z), f^1(z')] &= E[[f^1(z) - \mathcal{L}(m^2, z)][f^1(z') - \mathcal{L}(m^2, z')] \\ &= E[[\mathcal{L}\left(f^2, z\right) - \mathcal{L}(m^2, z)][\mathcal{L}\left(f^2, z'\right) - \mathcal{L}(m^2, z')] \\ &= E[[\mathcal{L}\left(f^2 - m^2, z\right)][\mathcal{L}\left(f^2 - m^2, z'\right)]] \\ &= \mathcal{L}\left(\mathcal{L}\left(E[[f^2 - m^2][f^2 - m^2]], z'\right), z\right) \\ &= \mathcal{L}\left(\mathcal{L}\left(k^2(x, x'), z'\right), z\right) \end{aligned} \tag{7.10}$$

Since $\mathcal{L}$ is a linear operator, the resulting function is a positive semidefinite function [Ginsbourger 2013, Särkkä 2011]. We can thus write the joint-Gram matrix between the outputs f^1 and f^2 as:

$$\boldsymbol{K_{XX}} = \begin{pmatrix} \mathcal{L}(\mathcal{L}(\boldsymbol{K^2}, x^1), x^1) & \mathcal{L}(\boldsymbol{K^2}, x^1) \\ \mathcal{L}(\boldsymbol{K^2}, x^2) & \boldsymbol{K^2} \end{pmatrix} \tag{7.11}$$

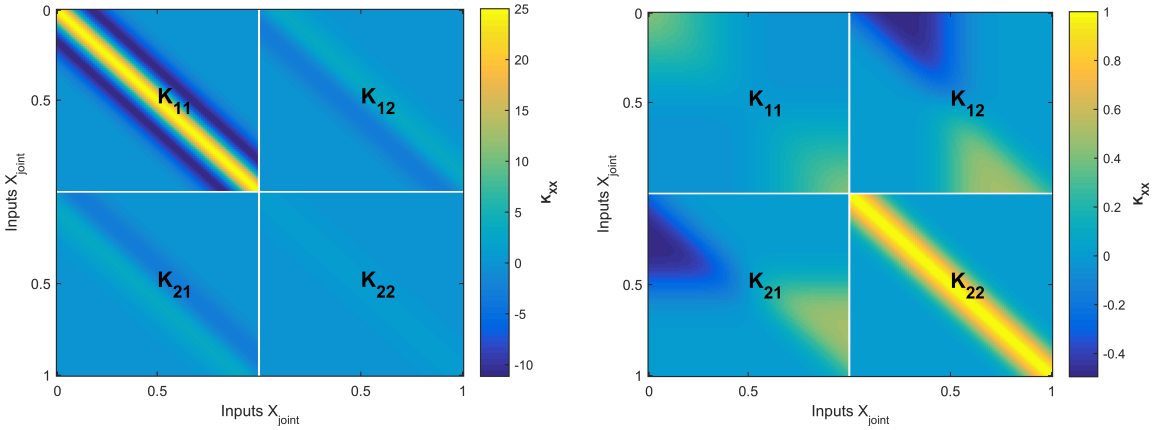
Here the matrix $\boldsymbol{K^2}$ is the Gram matrix evaluated using the covariance function $k^2(x, x')$. The hyper-parameters of the prior are $\theta = \{\theta_{amplitude}, \theta_{lengthScale}\}$. These correspond to the hyper-parameters of the independent covariance function k^2. We have effectively represented the covariance function $Cov[f^1(z), f^1(z')]$ and $Cov[f^1(z), f^2(z')]$ in terms of the covariance function $Cov[f^2(z), f^2(z')]$ using the known relation between outputs.

The sample code 7.1 shows one of the ways to calculate the covariance functions for the equation 7.11. For complex linear operators, the covariance functions start getting complicated pretty quickly, using the symbolic toolbox is a nice hack around manually calculating their functional forms.

```
function [covy1y1, covf1f2, covf2f1, covf2f2] =
   calculateJointCovarianceFunctions(linearOperator)

% calculating the covariance functions using symbolic toolbox
[x1, x2, sn22, sn11, sf2, ell] = deal([]);
syms x1 x2 sn22 sn11 sf2 ell sqDist dist

% ell = exp(hyp1)
% sf2 = exp(2*hyp2)

latentk22 = sf2.*(exp(-0.5.*(x1-x2)^2/ell^2));
covf1f2 = linearOperator(latentk22, x1);
covf2f1 = linearOperator(latentk22, x2);
latentk11 = g(covf1f2, x2);

covf2f2 = latentk22 + sn22;
covy1y1 = latentk11 + sn11;

end

%%
% Integral Linear operator
L = @(y, x)  int(y, x, 0, x);
[covy1y1, covf1f2, covf2f1, covf2f2] =
   calculateJointCovarianceFunctions(L);
```

Matlab Code 7.1: Create joint covariance functions using Matlab Symbolic Toolbox

Figure 7.1 represents the Gram matrix obtained by enforcing a relationship between two outputs y^1 and y^2. We use a SE kernel ($\theta_{amplitude} = 1$ and $\theta_{lengthScale} = 0.2$) as a covariance function for defining the prior distribution of the independent output y^2. The auto-covariance and cross-covariance functions are calculated using the code 7.1. The Gram matrix is evaluated at 100 equidistant points $X^j = [0 : 0.01 : 1]$ for both the outputs. Figure 7.1(a) is the Gram matrix when the outputs are related through a derivative relationship ($y^1 = \frac{\partial y^2}{\partial x}$), while figure 7.1(b) is the Gram matrix when the outputs are related through an integral relationship ($y^1 = \int_0^x y^2$). The $\boldsymbol{K_{ii}}$ represent the Gram matrices between same outputs, while the $\boldsymbol{K_{ij}}$ (for $i \neq j$) represents the Gram matrix between different outputs.



(a) Gram matrix between y^1 and y^2 such that $y^1 = \frac{\partial y^2}{\partial x}$. SE covariance function is used for the y^2 with $\theta = [1, 0.2]$, and $\boldsymbol{X}^j = [0 : 0.01 : 1]\ \forall\ j = [1, 2]$

(b) Gram matrix between y^1 and y^2 such that $y^1 = \int_0^x y^2$. SE covariance function is used for the y^2 with $\theta = [1, 0.2]$, and $\boldsymbol{X}^j = [0 : 0.01, 1]\ \forall\ j = [1, 2]$

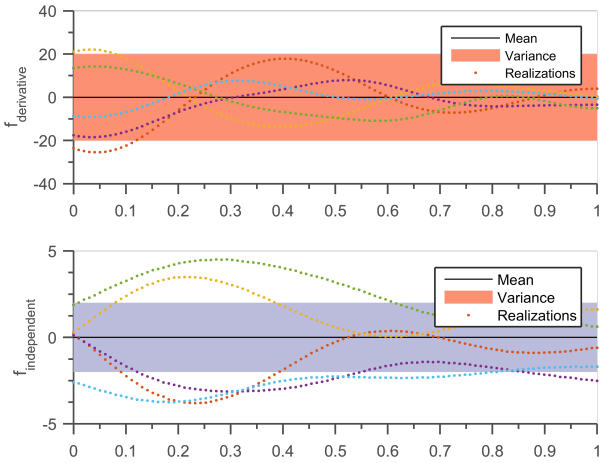
Figure 7.1: Gram matrices for the joint kernel, which encodes the relation between two outputs y^1 and y^2

The joint Gram matrix between f^1 and f^2, means that a random draw of independent function f^2 will result in a correlated draw of the function f^1. This effectively means that when we draw a random function f^2 it will result in a correlated draw of f^1, such that the two random functions will satisfy the enforced equation.

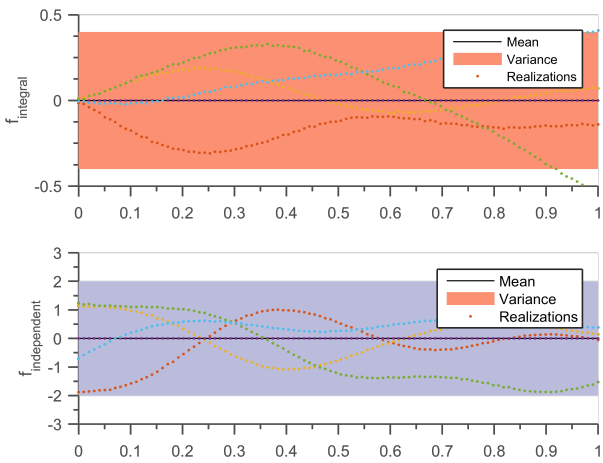
Figure 7.2

Figure 7.2 shows randomly drawn functions for two outputs related through an equation. The figure 7.2(a) shows random draws coming from a differential relationship between $f_{differential}$ (red) and $f_{independent}$ (blue) such that $f_{differential} = \frac{df_{independent}}{dx}$. We can see that the top figure is derivative of the bottom one since, $f_{derivative}$ goes to zero where $f_{independent}$ goes to maxima or minima. Similarly, the figure 7.2(b) shows random draws coming from an integral relationship between $f_{integral}$ (red) and $f_{independent}$ (blue) such that $f_{integral} = \int_0^x f_{independent}$. We can see that the top figure is integral of the bottom one since, $f_{independent}$ goes to zero where $f_{integral}$ goes to maxima or minima.

Note that, proper care should be taken before performing these operations, if an operation is not defined for the prior family of functions of f^2, then we cannot write its corresponding joint-GP. For example, if we define a GP prior for f^2 with an exponential covariance function (section 4.4.2), we cannot enforce a differential relationship on f^2. This is because the exponential covariance function defines a hypothesis space of non-differentiable functions, and hence a differential operator is not-defined for such functions.



(a) Joint draws between $f_{differential}$ (red) and $f_{independent}$ (blue) such that $f_{differential} = \frac{df_{independent}}{dx}$. We define a GP prior with zero mean and SE covariance function ($\theta = [1, 2]$) for the function f^2.



(b) Joint draws between $f_{integral}$ (red) and $f_{independent}$ (blue) such that $f_{integral} = \int_0^x f_{independent}$. We define a GP prior with zero mean and SE covariance function ($\theta = [1, 2]$) for the function f^2.

Figure 7.2: Joint draws defined by priors on related-outputs. The solid black line defines the mean function, shaded blue region defines 95% confidence interval (2σ) distance away from the mean. The dashed lines represent five functions drawn at random from a GP prior.

7.2.2 Case of non-linear operators

Simulation codes

Physical laws often do not come in form of non-linear operators, but they are difficult to solve analytically[2] for complex geometries. For example solving the structural dynamics equation is difficult for continuous complex geometries of an aircraft. One popular method to solve these differential equations is to discretize the complex geometry into simple meshes. A complex geometry of an aircraft is broken down into a simpler point, beam and plate structures, these simpler geometries are easier to solve analytically. We can then apply the physical laws on these simplistic geometries, and finally, aggregate the result of individual meshes. For example, CFD and CSM are discretized techniques to evaluate fluid and structural laws on complex geometries. These discretized operations are often non-linear and are represented in form of computer code. In this section, we wish to evaluate auto and cross-covariances across outputs related through non-linear relationships.

A non-linear operation on a GP does not result in a GP, hence for the case of non-linear $\mathcal{L}(.)$ the joint Gram matrix, as derived in equation 7.4, is not positive semi-definite [Stein 1999]. Therefore, we will use an approximate joint-covariance as developed by

[2]There is a millennium prize for solving the Navier Stokes equation

[Constantinescu 2013] for imposing non-linear relations.

$$K_{XX} = \begin{bmatrix} L\boldsymbol{K}^2 L^T & L\boldsymbol{K}^2 \\ \boldsymbol{K}^2 L^T & \boldsymbol{K}^2 \end{bmatrix} + \mathcal{O}\left(\delta_2^3\right) \tag{7.12}$$

$$L = \frac{\partial \mathcal{L}}{\partial y}\Big|_{f^2=m^2} \tag{7.13}$$

Where L is the Jacobian matrix of $\mathcal{L}(.)$ evaluated at the mean (m^2) of independent output (f^2). δ_2 is the amplitude of small variations of f^2, introduced by the Taylor series expansion of $\mathcal{L}(f^2)$ with respect to $E[f^2]$.

$\mathcal{O}\left(\delta_2^3\right)$

The above covariance function takes a parametric form that depends on the mean value process of the independent variable $(E[f^2(x)])$. Equation 7.12 is basically a Taylor series expansion for approximating related kernels. Since a Taylor series expansion is constructed from derivatives of a function, which are linear operations, the resulting approximated joint kernel is a Gaussian kernel with the non-Gaussian part $(\mathcal{O}\left(\delta_2^3\right))$ as the error. Higher-order closures can be derived with higher order derivatives of the operator $\mathcal{L}(.)$. For simplicity we will restrict ourselves to first order approximation of the auto- and cross-covariance functions leading to an error of the order $\mathcal{O}\left(\delta_2^3\right)$. A more detailed derivation can be found at appendix C.

Below is a sample code (code 7.2) for calculating the joint-covariance functions automatically. The 'jacobianOfOperator' is the jacobian of the non-linear operator $\mathcal{L}$ evaluated at the mean values of y^2.

```
function [covy1y1, covf1f2, covf2f1, covy2y2] =
   calculateNonLinearJointCovarianceFunctions(jacobianOfOperator
   )

% calculating the covariance functions using symbolic toolbox
[x1, x2, sn22, sn11, sf2, ell] = deal([]);
syms x1 x2 sn22 sn11 sf2 ell sqDist dist

latentk22 = sf2.*(exp(-0.5.*(x1-x2).^2/ell.^2));
covf1f2 = jacobianOfOperator(x1).*(latentk22);
covf2f1 = jacobianOfOperator(x2).*(latentk22);
latentk11 = jacobianOfOperator(x2).^2.*(latentk22);

% adding noise
covy2y2 = latentk22 + sn22;
covy1y1 = latentk11 + sn11;
```

```
end
```

Matlab Code 7.2: Create joint covariance functions for non-linear operators using Matlab Symbolic Toolbox

7.2.3 Calculating posterior

The posterior distribution can be calculated by conditioning the prior distribution on the observation data-set. Since a prior distribution over functions defined over the joint data-set is also a GP, we can easily derive the posterior mean, posterior covariance and marginal likelihood (the exact derivations are described in section 6.2.5). We just need to insert the appropriate Gram matrix ($\boldsymbol{K_{XX}}$) in the equations of posterior mean (equation 7.14), posterior variance (equation 7.15) and marginal likelihood (equation 7.16).

Posterior

$$E[f(\boldsymbol{X_*}))] = \boldsymbol{K_{X_*X}}\left(\boldsymbol{K_{XX}} + \boldsymbol{\Sigma}\right)^{-1}\boldsymbol{y_{joint}} \tag{7.14}$$

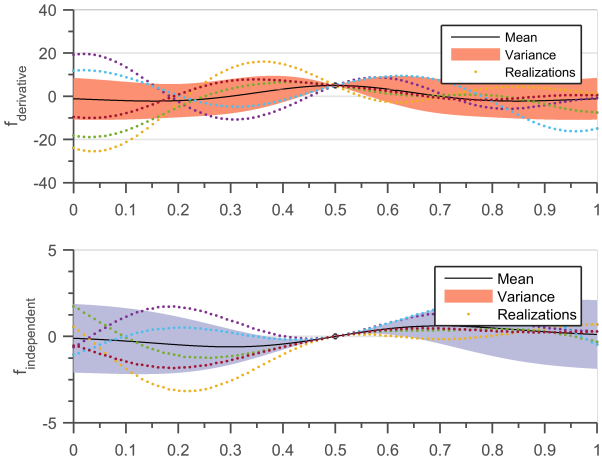
$$Cov(f(\boldsymbol{X_*})) = \boldsymbol{K_{X_*X_*}} - \boldsymbol{K_{X_*X}}\left(\boldsymbol{K_{XX}} + \boldsymbol{\Sigma}\right)^{-1}\boldsymbol{K_{XX_*}} \tag{7.15}$$

$$\log(\Pr[\boldsymbol{y_{joint}} \mid \boldsymbol{X}, \theta]) = \log[GP(\boldsymbol{y_{joint}}|0, \boldsymbol{K_{XX}} + \boldsymbol{\Sigma})] \tag{7.16}$$

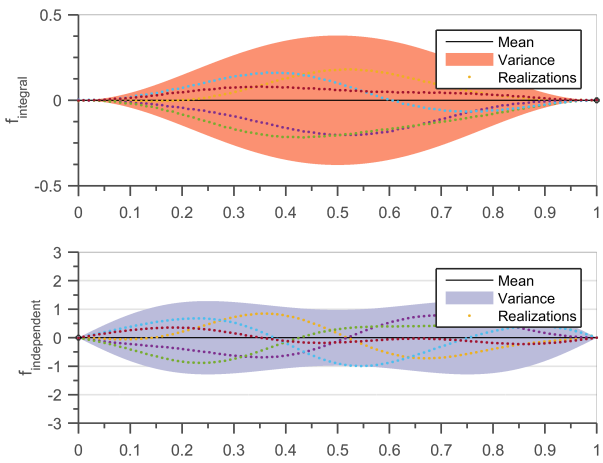
Example The figure 7.3(a) shows mean and variance for a differential relationship between independent function $f_{independent}$ (blue) and differential function $f_{derivative}$ (red), such that $f_{derivative} = \frac{\partial f_{independent}}{\partial x}$. We have conditioned the two functions such that $f_{derivative}|_{x=0} = 5$ and $f_{independent}|_{x=0} = 0$. This means that the function $f_{independent}$ passes through 0 and has a derivative equal to 5 at $x = 0$. It is easy to observe that all the functions of $f_{independent}$ that pass through $x = 0$ have the same value of derivatives. We have not maximized the marginal likelihood for this case and the hyper-parameters of k^2 are $\theta_{amplitude} = 1; \theta_{lengthScale} = 0.2$.

Figure 7.3(b) shows mean and variance for an integral relationship between independent function $f_{independent}$ (blue) and differential function $f_{integral}$ (red) and such that $f_{integral} = \int_0^x f_{independent} z.dz$. We have conditioned the two functions such that $f_{integral}|_{x=1} = 0$ and $f_{integral}|_{x=0} = 0$. This means that the function $f_{independent}$ has an integral 0 at the two points $[0, 1]$. Although it is difficult to verify from the figure, but all the randomly drawn functions of $f_{independent}$ indeed have their integral ($\int_0^1 f_{independent} = 0$) equal to zero. We have not maximized the marginal likelihood for this case and the hyper-parameters of k^2 are $\theta_{amplitude} = 1; \theta_{lengthScale} = 0.2$. All the corresponding draws from the posterior GP also follow the conditioning.

In the next section, we compare the prediction capabilities of independent GP, with that of multi-fidelity GP and joint relationship enforced GP on a synthetic data-set. We



(a) Joint draws between $f_{derivative}$ *red* and $f_{independent}$ *blue* such that $f_{derivative} = \frac{\partial f_{independent}}{\partial x}$. The posterior is conditioned such that $f_{derivative}|_{x=0} = 5$ and $f_{independent}|_{x=0} = 0$. The draws from the posterior follow the relationship between outputs and also the conditioning.



(b) Joint draws between $f_{integral}$ *red* and $f_{independent}$ *blue* such that $f_{integral} = \int f_{independent}$. The posterior is conditioned such that $f_{integral}|_{x=0} = 0$ and $f_{integral}|_{x=1} = 0$. The draws from the posterior follow the relationship between outputs and also the conditioning.

Figure 7.3: Multi-Output Gaussian Process Regression Predictions. The solid black line defines the mean function, shaded region defines 95% confidence interval (2σ) distance away from the mean. Dotted lines represent the functions randomly drawn from the posterior distributions.

then take a real flight-test data-set and study the effects of different prior k^2 on their predictions.

7.3 Experiments

In this section, we provide a numerical illustration to the theoretical derivations in the earlier sections. We start with a synthetic problem where we try to learn the model over quadratic relationships. We compare the cross-validation error values of independent GP regression with that of multi-fidelity GP and relationship enforced joint MTGP. Finally, we study the effect of choosing different priors for the independent function f^2 on a data-set for flight-loads estimation for the horizontal tail plane.

The basic toolbox used for this section is GPML provided with [Rasmussen 2005], we generate covariance functions to handle relationships as described in equations 7.4 using the “Symbolic Math Toolbox" in MATLAB 2014b (code 7.1 and 7.2).

7.3.1 Quadratic relation on Synthetic Data

We take the case of a quadratic operator $\mathcal{L}(.)$ equation 7.17.

$$f^1 = \left[f^2\right]^2 \tag{7.17}$$

Data-set

To generate the data, we randomly draw a single function of f^2 as described in equation 7.18 for 50 equally spaced inputs between [-1, 1]. The output f^1 is then calculated by applying the operation in equation 7.17. The latent functions $f's$ are then corrupted according to equation 7.18 which gives us the outputs $y's$. We finally hide the observations for y^1 in the domain $x = [-0.2, 0.2]$. We now have our data-set for testing the performance of quadratic relationship.

$$\begin{aligned} \Pr[f^2] &= GP[0, k_{SE}(0.2, 1)] \\ \Pr[\sigma_{n2}] &= \mathcal{N}[0, 0.1] \\ \Pr[\sigma_{n1}] &= \mathcal{N}[0, 1] \end{aligned} \tag{7.18}$$

$k_{SE}(1, 0.2)$ means squared exponential kernel with length scale 0.2 and variance as 1. σ_{n2} and σ_{n1} are the white noises added to the latent functions f^1 and f^2 respectively. Since the quadratic relationship $\mathcal{L}(.)$ is non-linear in nature we use equation 7.12 to calculate the auto- and cross-covariance functions as shown in equation 7.19.

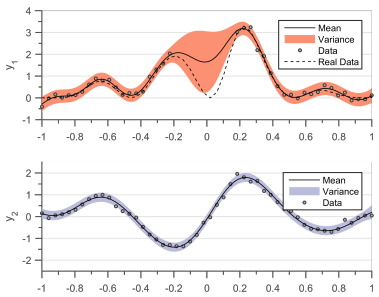
$$\begin{aligned} Cov(f^2(x), f^2(x')) &= k^2 \\ Cov(f^1(x), f^2(x')) &= 2m^2k^2 \\ Cov(f^1(x), f^1(x')) &= 4\left[m^2\right]^2 k^2 \end{aligned} \tag{7.19}$$

Here, m^2 is the mean value of function y^2. For the case of quadratic relationship the Jacobian L as described in equation 7.12 comes out to be $2{*}m^2(x)$. For non-linear operators $\mathcal{L}(.)$ the joint-covariance prior depends on the mean value of f^2 ($m^2(x)$). $Cov(f^1(x), f^1(x'))$ and $Cov(f^1(x), f^2(x'))$ are the covariance functions calculated at the input points x and x' as described in equation 7.12 and equation 6.5. We can observe that the joint-covariance has been expressed as a function of covariance k^2 using equation 7.17.

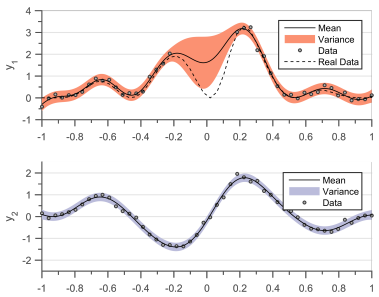
Figure 7.4

Figure 7.4(a) shows the independent fit of two GP regressions. For the case of y^1, there is a huge difference between the real data and predicted mean at points where data is unavailable. Figure 7.4(b) shows the fit using multi-fidelity GP regression model (section 6.2.4). We can observe that the multi-fidelity model cannot capture the quadratic nature of the relationship between outputs. Figure 7.4(c) shows the joint-GP regression, which gives better prediction even in the absence of data for y^1 because a transfer of information is happening from observations of y^2 present at those locations.

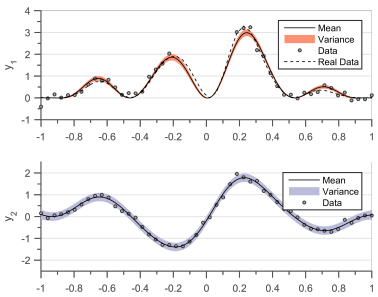
Table 7.1 shows comparison of Root Mean Square Error (RMSE). 10 sets of experiments were run for 85% of data as training set and 15% of data as the test set, test sets were



(a) Independent GP Regression for the two outputs y^1 and y^2. We can observe the huge difference between the real data and the predicted mean values at zone with no data.



(b) Multi-fidelity GP Regression for the two outputs y^1 and y^2 (section 6.2.4). We can observe the huge difference between the real data and the predicted mean values at zone with no data.



(c) Joint-GP Regression for the two outputs y^1 and y^2 related through equation 7.17. We can observe the improved prediction between zone with no data because information is being shared between the two outputs.

Figure 7.4: GP Regression on Quadratic relationship. The solid black line represents the predicted mean while the shaded area denotes 2σ uncertainty region. The dashed black line represents the real value of f^1 or f^2. For y^1 the data is hidden from section $x = [-0.2, 0.2]$.

Table 7.1: mean RMSE errors for quadratic relationship

	RMSE y^1	RMSE y^2
Independent Single-output GPR	1.74	0.14
Multi-fidelity GPR (section 6.2.4)	1.54	0.13
Joint Multi-output GPR	0.26	0.12

Table 7.1 removed uniformly from the output y^1 (similar to figure 7.4). We learn the optimal set of hyper-parameters for on training data all 10 sets of experiments. Finally, RMSE values are evaluated with respect to the test set. We are comparing here the accuracy of three model types, where the independent model gives the worst prediction, the multi-fidelity model is a little bit better but still worse than the model with the actual relationship enforced.

The current experiment demonstrates how to enforce a non-linear relationship between outputs. Multi-fidelity GP has a similar performance as that of an independent GP since the two outputs are not results of code for different fidelities. The current set of outputs do-not show the Markov property used in multi-fidelity GPs, if a similar type of information is missing in the simulation model then using the linearly separable covariance functions will obviously lead to bad extrapolation results. Enforcing the quadratic relationship between the outputs has the best accuracy of the three methods (figure 7.4(c)).

Adding inequality constraints Research on adding inequality constraints in GP models is slowly picking up pace in the GP community. [Da Veiga 2012] enforced inequality constraints by truncating probability distributions, while [Maatouk 2017] enforce inequality constraints by using functional decomposition. The quadratic relationship gives rise to an interesting property. If $f^1 = [f^2]^2$ then it means that $f^1 > 0 \; \forall x \in \mathcal{R}$, this property can also be used to indirectly enforce inequality constraints in GP models.

Suppose we have access to a N observed data-points $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{y}\}$ subject to the constraints $y_i > 0 \; \forall i \in N$. We can assume a second output $sqrtY$, such that $y = [sqrtY]^2$ and build a quadratic relationship enabled GP model between the two (y and $sqrtY$). GP models enforcing monotonocity or convexity can be similarly made by first, enforcing a derivative relationship and then a quadratic relationship.

One problem that can come up by enforcing inequality constraints using quadratic relationship, is that square root of y ($sqrtY$) can be both positive and negative. This can complicate matters when calculating the mean of the square root. It would be interesting to study the implications of choosing one sign of square roots over the another needs more investigation.

7.3.2 Flight Mechanics on Flight Test Data

In this section, we conduct experiments applying our approach on the flight loads data. We look at normalized data of a simple longitudinal maneuver. The maneuver is quasi-static which means that the airplane is in equilibrium at all times and there are no dynamic effects observed by the aircraft. The two outputs in our case are the coefficient of lift C_z on the Horizontal Tail Plane (HTP) and the spanwise lift k_z. We know that the integral of spanwise pressure will be equal to the coefficient of lift (equation 7.20).

$$C_z(\eta) = \int_{\eta_{edge}}^{\eta_{root}} k_Z(\eta) d\eta \tag{7.20}$$

.

Here, η_{edge} denotes the edge of horizontal tail span while η_{root} represents root of tail. The above equation is linear in nature and hence we will use equation 7.4 to calculate the auto- and cross-covariance functions.

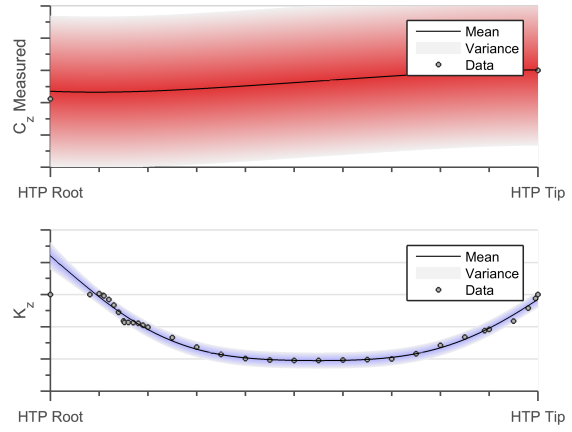
Merging domains

In practice, the coefficient of lift C_z is calculated using strain gauges on the HTP and the spanwise force is measured using pressure tappings. Chordwise pressures are measured at multiple η locations on the HTP, which are integrated chordwise to give spanwise lift distribution. We are trying to merge data coming from two different calculation chains flight mechanics (C_z) and aerodynamics (k_z), by using a basic physics based relationship between them. To the best of our knowledge this approach has not been proposed earlier in loads estimation.

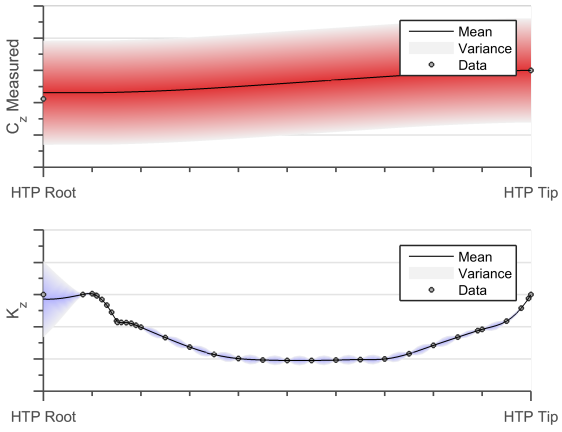
Figure 7.5

Figure 7.5 shows a comparison between two joint multi-output GP on K_z and C_z using different prior distributions. Figure 7.5(a) shows the effect of SE kernel (infinite differentiability) on the joint model. The SE kernel is a strong assumption for the type of lift distribution. To accommodate for the value of measured C_z, the error band increases and the boundary condition at HTP root is not satisfied. Moreover, we identify points in the figure 7.5(a) which fall out of the 95% confidence band. We can claim with 95% probability that if our lift distributions come from a family of infinitely differentiable functions these outliers do not satisfy the physics of the problem and can be termed as measurement errors.

Figure 7.5(b) shows the effect of twice differentiable Matérn kernel on the joint model. The twice differentiable assumption gives our family of functions enough flexibility to follow the spanwise lift data. We observe that error estimates for C_z improve considerably while there is an increased uncertainty at the HTP root for K_z. This loss in the order of differentiability can be explained due to fuselage HTP interaction near HTP root. We can conclude that even though we are working with very basic assumptions of differentiability the choice of kernel is important in performing the predictions.



(a) Joint multi-output GPR between K_z and C_z using a **SE kernel** and equation 7.20. Outlier points either do not follow the relationship between outputs or the SE kernel imposes a strict bias



(b) Joint multi-output GPR between K_z and C_z using a twice differentiable **Matérn kernel** and equation 7.20.

Figure 7.5: Joint multi-output GPR on K_z and C_z relationship. The solid black line represents the predicted mean while the shaded area denotes 2σ uncertainty region. The figure demonstrates the impact of the covariance function for relationship enabled regression.

In this section, we compare the predictive capabilities of using two different covariance functions for defining the prior of independent output f^2. For the current case, the SE kernel imposed a very strict bias for the pressure distribution K_z, this puts a few observation points outsize the 95% confidence interval. In fact, the twice differentiable Matérn kernel is a better covariance function for the problem. This is because due to the fuselage wing

interaction, the pressure distribution is not infinitely differentiable near the HTP root.

Identifying faulty sensors We also observe that the current formulation pushes observations that do not follow the physics of the system outside its confidence band, this is a very interesting property and can be used to automatically identify faulty sensors. Suppose we measure the loads on an aircraft using strain gauges, while we measure the acceleration of the aircraft using accelerometers. We can leverage the Newton's third law of motion ($\int loads = MassOfAircraft \times acceleration$) to build a robust loads model and identify faulty strain gauges automatically.

Calculating the log-marginal likelihood involves inverting the matrix $\boldsymbol{K_{XX}} + \boldsymbol{\Sigma}$. The size of the $\boldsymbol{K_{XX}} + \boldsymbol{\Sigma}$ matrix depends on a total number of input points N, hence inverting the matrix becomes intractable for a large number of input points. In the next section we describe how to solve the problem of inverting huge $\boldsymbol{K_{XX}} + \boldsymbol{\Sigma}$ matrices using approximate inference techniques.

7.4 Scaling up MTGP

As already discussed in chapter 3 on scaling GP regression, the GP approach is intractable for large data-sets. This problem is further aggravated when we are creating a joint Gram matrix between several outputs. For a multi-output GP as defined in section 6.2 the covariance matrix is of size N_{joint}, where $\mathcal{O}\left(N_{joint}^3\right)$ time is needed for inference and $\mathcal{O}\left(N_{joint}^2\right)$ memory for storage.

In this section, we extend the formulation of Variational GP approximation (section 3.2.2) and Distributed GPs approximation (section 3.3) for the case of MTGP's. The results of this study were published in ICPRAM 2016 and LNCS 2017 [Chiplunkar 2016c, Chiplunkar 2017c]. The remaining section details the two methods for approximating covariance matrix which can be later used to calculate the predictive mean, predictive covariance and the marginal likelihood (7.14, 7.15 and 7.16).

Publications

7.4.1 Variational Approximation on Multi-output GP

Sparse methods use a small set of M function points as support or inducing variables. Suppose we use M inducing variables to construct our sparse GP. The inducing variables are the latent function values evaluated at inputs $\boldsymbol{X_M}$. An approximation to the true log marginal likelihood in equation 6.23 can allow us to infer these quantities.

We try to approximate the joint-posterior distribution $p(\boldsymbol{X}|\boldsymbol{y})$ by introducing a Variational distribution $q(\boldsymbol{X})$. We use the same inducing points for all the outputs (equation

6.3) and extend the derivation of [Titsias 2009] to the multi-output case.

$$q(\boldsymbol{X}) = \mathcal{N}(\boldsymbol{X}|\boldsymbol{\mu}, \boldsymbol{A}) \tag{7.21}$$

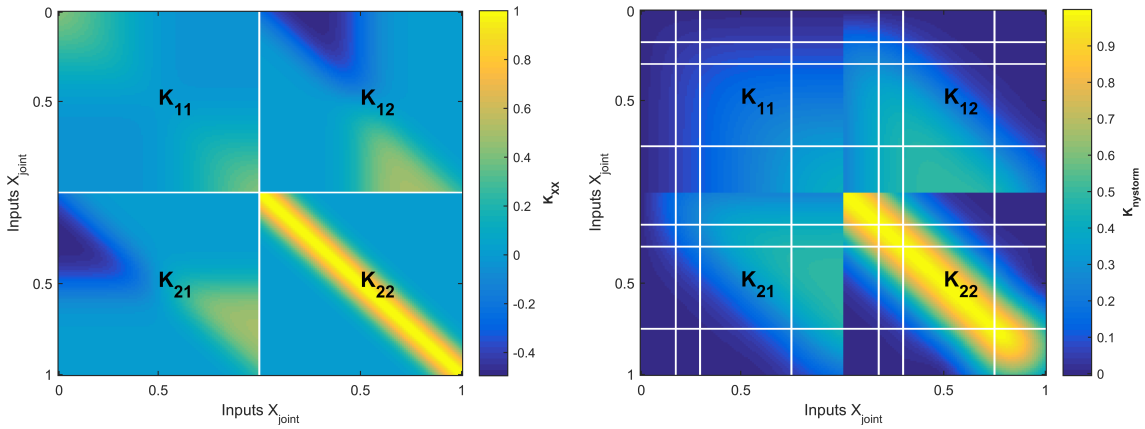
Here $\boldsymbol{\mu}$ and $\boldsymbol{A}$ are parameters of the Variational distribution. We follow the derivation provided in [Titsias 2009] and obtain the lower bound (Fv) of true marginal likelihood.

$$F_V = log(\mathcal{N}[\boldsymbol{y}|0, \sigma^2 \boldsymbol{I}_{N_{joint}} + \boldsymbol{K}_{Nystrom}]) - \frac{1}{2\sigma^2} Tr(\tilde{\boldsymbol{K}}) \tag{7.22}$$

where $\boldsymbol{K}_{Nystrom} = \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}_M} \boldsymbol{K}_{\boldsymbol{X}_M\boldsymbol{X}_M}{}^{-1} \boldsymbol{K}_{\boldsymbol{X}_M\boldsymbol{X}}$ and $\tilde{\boldsymbol{K}} = \boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} - \boldsymbol{K}_{Nystrom}$. $\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}$ is the joint-covariance matrix derived using equation 7.4 using the input vector $\boldsymbol{X}$. $\boldsymbol{K}_{\boldsymbol{X}_M\boldsymbol{X}_M}$ is the joint-covariance function on the inducing points $\boldsymbol{X}_M$, such that the inducing points are same for all the outputs. While $\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}_M}$ is the cross-covariance matrix between $\boldsymbol{X}_{joint}$ and $\boldsymbol{X}_M$.

Figure 7.6(b)

Figure 7.6(b) shows the approximate Gram matrix by using the inducing points shown by white lines for the Gram matrix described in the left figure. We see how choosing three same random inducing points for both the outputs (y^1 and y^2) impacts the Gram matrix. As analyzed in the chapter 3 we can improve the approximate Gram matrix if we increase the number of inducing points.



(a) Gram matrix between y^1 and y^2 such that $y^1 = \int_0^x y^2$. SE covariance function is used for the y^2 with $\theta = [1, 0.2]$, and $\boldsymbol{X}^j = [0 : 0.01, 1]\ \forall\ j = [1, 2]$ (figure7.1(b))

(b) Approximated Gram matrix using Nyström approximation for Gram matrix of figure in left. Three inducing points were chosen randomly, the white lines denote the location of inducing points.

Figure 7.6: Approximate Gram matrix for a Joint MTGP kernel using Nyström approximation

Lower bound

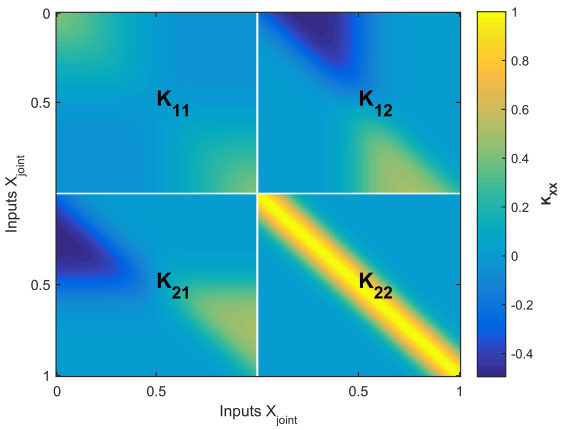
The bound (Fv) can be maximized with respect to all parameters of the covariance function; the model hyper-parameters and the Variational parameters. The optimization

parameters are the inducing inputs $\boldsymbol{X_M}$, the hyper-parameters θ of the independent covariance matrix $\boldsymbol{K_{22}}$ and the error while measuring the outputs σ. There is a trade-off between quality of the estimate and amount of time taken for the estimation process. On the one hand the number of inducing points determine the value of optimized log-marginal likelihood and hence the quality of the estimate. While, on the other hand there is a computational load of $\mathcal{O}\left(N_{joint}(MD_{outputs})^2\right)$ for inference. In upcoming experiments we increase the number of inducing points until the difference between two successive likelihoods is below a predefined quantity.

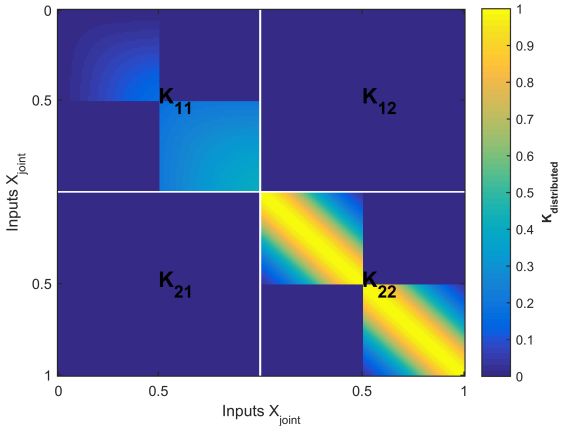
7.4.2 Distributed Inference on Multi-output GP

An alternative to sparse approximations is to learn local experts on a subset of data. We have described Distributed GPs in detail in section 3.3. We again distribute our data-set $\boldsymbol{X_{joint}}$ into smaller experts and use the independence assumption to approximate the Gram matrix.

Figure 7.7 shows the difference between an exact Gram matrix and a Gram matrix approximated using Distributed GPs. Figure 7.7(b) shows the Gram matrix after performing distributed approximation for the Gram matrix described in the left figure. We see how choosing two same uniform experts for both the outputs (y^1 and y^2) impact the Gram matrix. As analyzed in the chapter 3, we can improve the approximate Gram matrix if we increase the number of pointsin each expert.



(a) Gram matrix between y^1 and y^2 such that $y^1 = \int_0^x y^2$. SE covariance function is used for the y^2 with $\theta = [1, 0.2]$, and $\boldsymbol{X}^j = [0 : 0.01, 1]\ \forall\ j = [1, 2]$ (figure7.1(b))



(b) Approximated Gram matrix using Distributed GPs approximation for Gram matrix of figure in left. Points in the experts are distributed uniformly. We can observe that covariance across experts goes to zero

Figure 7.7: Approximate Gram matrix for a SE kernel using mixture of experts.

Equation 7.23 describes the formulation for marginal likelihood for Distributed GPs approximation. Due to the independence assumption, the marginal likelihood can be written as a sum of individual likelihoods and then can be optimized to find the best-fit hyper-parameters. After learning the hyper-parameters we can combine the predictions of local experts to give mean and variance predictions (more details in section 3.3.1).

$$\log p(\boldsymbol{y_{joint}}|\boldsymbol{X_{joint}}, \theta) \approx \sum_{k=1}^{M} \log p_k(\boldsymbol{y^{(i)}}|\boldsymbol{X^{(i)}}, \theta) \tag{7.23}$$

rBCM

The robust Bayesian Committee Machine (rBCM) model combines the various experts using their confidence on the prediction point [Deisenroth 2015]. In such manner experts which have high confidence at the prediction points get more weight when compared to experts with low confidence.

$$m(\boldsymbol{y_*}) = (Cov(\boldsymbol{X_*}))^{-2} \sum \beta_k \sigma_k^{-2} m_k(\boldsymbol{X_*}) \tag{7.24}$$

$$(Cov(\boldsymbol{y_*}))^{-2} = \sum_k \beta_k \sigma_k^{-2} + (1 - \sum_k \beta_k)\sigma_{**}^{-2} \tag{7.25}$$

In the above equations $m_k \boldsymbol{X_*}$ and σ_k are the mean and covariance predictions from expert k at point $\boldsymbol{X_*}$. σ_{**} is the auto-covariance of the prior at prediction points X_*. β_k determines the influence of experts on the final predictions [Cao 2014] and is given as $\beta_k = \frac{1}{2}(\log \sigma_{**}^{-2} - \log \sigma_k^{-2})$.

7.5 Experiments

In the current section, we first compare the predictions of independent GPs with respect to joint MTGPs both approximated through Variational approximation. We then empirically compare the performance of Distributed GPs and Variational Inference with respect to the training time and accuracy. Finally, we compare the predictions of Variational joint-GP on a real-world flight-test data-set.

7.5.1 Experiments on Theoretical Data

We consider a derivative relationship between two output functions as described in equation 7.3. Such that

$$f^1 = \frac{\partial f^2}{\partial x} \tag{7.26}$$

Data is generated from equations 7.27, a random function is drawn from GP to get f^2 whose derivative is then calculated to generate f^1. y^1 and y^2 are then calculated by adding noise according the equations 7.27. 10,000 equidistant points are generated between the locations $x \in [-1, 1]$, for both the outputs y^1 and y^2. Values of y^2 are masked in the region $x \in [0, 0.3]$, the remaining points now constitute our training data-set.

$$\begin{aligned} f^2 &\sim GP[0, k_{SE}(0.1, 1)] \\ \sigma_{n2} &\sim \mathcal{N}[0, 0.2] \\ \sigma_{n1} &\sim \mathcal{N}[0, 2] \end{aligned} \tag{7.27}$$

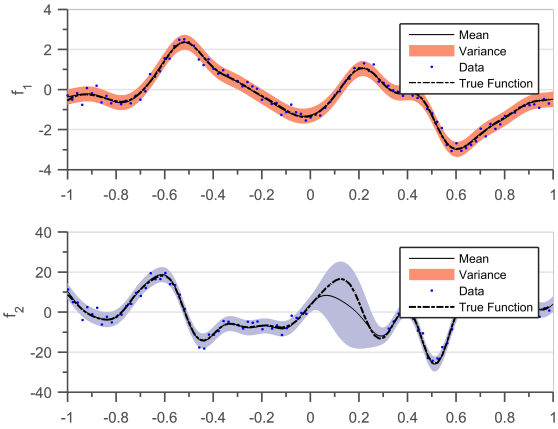
$k_{SE}(0.1, 1)$ means squared exponential kernel with $\theta_{lengthScale} = 0.1$ and $\theta_{amplitude} = 1$. Since the differential relationship ($\mathcal{L}(.)$) is linear in nature, we use the equation 7.4 to calculate the auto- and cross-covariance functions as shown in equation 7.28.

$$\begin{aligned} Cov(f^2(x), f^2(x')) &= \theta^2_{amplitude} exp\left[\frac{-1}{2}\frac{d^2}{\theta^2_{lengthScale}}\right] \\ Cov(f^1(x), f^2(x')) &= \theta^2_{amplitude}\frac{d}{\theta^2_{lengthScale}} exp\left[\frac{-1}{2}\frac{d^2}{\theta^2_{lengthScale}}\right] \\ Cov(f^1(x), f^1(x')) &= \theta^2_{amplitude}\frac{d^2 - \theta^2_{lengthScale}}{\theta^4_{lengthScale}} exp\left[\frac{-1}{2}\frac{d^2}{\theta^2_{lengthScale}}\right] \end{aligned} \tag{7.28}$$

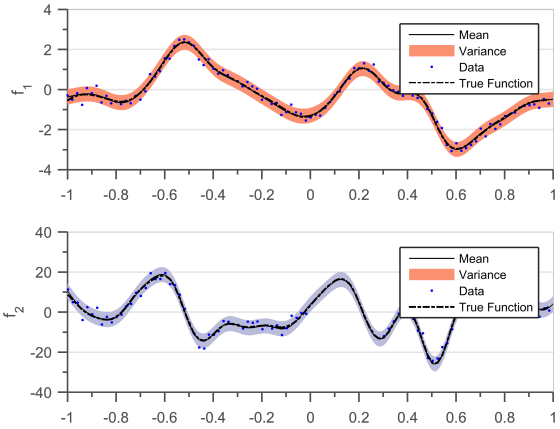
Figure 7.8 shows comparison between an independent fit GP and a joint multi-output GP whose outputs are related through a derivative relationship described in equation 7.26. For figure 7.8(a), using Variational inference algorithm, we optimize the lower bound of log marginal likelihood for independent GP's on y^1 and y^2. For figure 7.8(b), using Variational inference, we optimize the same lower bound but with a joint-covariance approach as described in section 7.4.1 using y^1, y^2 and $\mathcal{L}(.)$. We settled on using 100 equidistant inducing points for this exercise [Chiplunkar 2016c] and have only optimized the hyper-parameters to learn the model.

Figure 7.8(a) shows the independent fit of two GP for the differential relationship while, figure 7.8(b) shows the joint GP fit. The GP model with joint-covariance gives better prediction even in absence of data of y^2 for $x \in [0, 0.3]$.

Comparison between Distributed GPs and Variational GPs For the second experiment we compare the Root Mean Squared Error (RMSE) and run-times of Distributed GPs and Variational Inference algorithms while performing approximate inference. We progressively generate from 10^3 to 10^5 data-points according to the equations 7.27 and 7.26. We separated 75% of the data as the training set and 25% of the data as the test set,



(a) **Independent** fit for two GP's, experiment was run on 10,000 points but only 100 data-points are plotted to increase readability. Inference is performed using Variational inference algorithm and equidistant 100 inducing points. We can observe the huge difference between the real data and the predicted mean values at zone with no data.

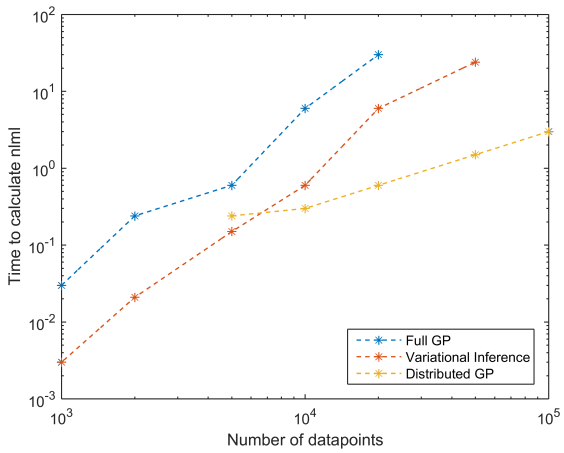


(b) **Joint multi-output GP's** for two outputs, experiment was run on 10,000 points but only 100 data-points are plotted to increase readability. Inference is performed using Variational inference algorithm and equidistant 100 inducing points. We can observe the improved prediction between zone with no data because information is being shared between the two outputs.

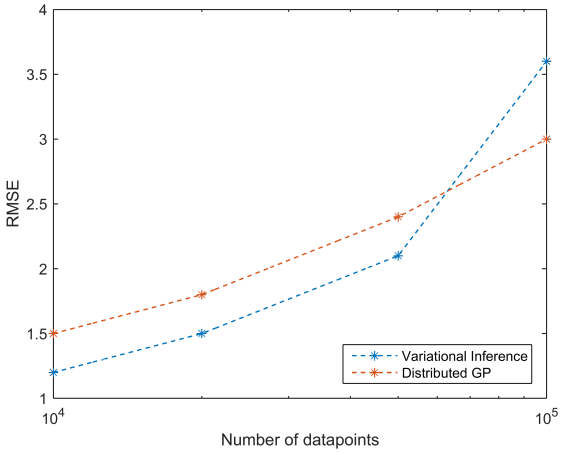
Figure 7.8: Experimental results for differential relationship while using Variational approximation. Predicted mean is represented by solid black line. 2σ confidence band is represented by light red for f^1 and light blue for f^2. The dashed black line represents the true latent function values

the training and the test sets were chosen randomly. The Variational inference relationship kernel as described in section 7.4.1 with 100 equidistant inducing points was used. We learn the optimal values of hyper-parameters for all the sets of training data. The Distributed GPs algorithm as described in section 7.4.2 was used with randomly chosen 100 points per expert. We learn the optimal values of hyper-parameters for all the sets of training data. The accuracy is plotted as RMSE values with respect to the test set. The run time is defined as time taken to calculate the approximate marginal likelihood equations 7.22 and 7.23. The RMSE values are calculated for only the dependent output y^1 and then plotted in the figure 7.9(a).

In figure 7.9(a) the time to calculate marginal likelihood with increasing number of training points is calculated. As expected the full GP takes more time when compared to Variational inference or Distributed GPs algorithms. The Variational inference algorithm has better run-time till $\mathcal{N} \sim 10^4$ data-points, after that Distributed GPs takes lesser time. In figure 7.9(b), the RMSE error with test set is compared between the Variational inference and Distributed GPs algorithm. Here too the Variational inference algorithm performs better for a lesser number of data-points, but Distributed GPs starts performing better when we reach more than $\mathcal{N} \sim 10^4$ data-points.



(a) Comparison of time to calculate log marginal likelihood for a Full GP, Variational inference and Distributed GPs with increasing number of data-points. We observe for data-points greater that 10^5 the Distributed GPs algorithm starts outperforming Variational inference



(b) Comparison of RMSE for Variational inference and Distributed GPs algorithm. We observe for data-points greater that 10^4 the Distributed GPs algorithm starts outperforming Variational inference.

Figure 7.9: Comparison of run time and RMSE between Distributed GPs and Variational Inference

As we keep on increasing the number of data-points, for a fixed set of inducing points Variational inference starts performing worse. This is because as we have already observed we need $M \sim \frac{N}{10}$ for good accuracy with inducing inputs approximation. If we keep the value of $M \sim \frac{N}{10}$ for $N \sim 10^5$ then the size of matrix $\boldsymbol{K_{MM}}$ will become very big to invert efficiently. Thus even with inducing inputs approximation we have an upper limit to the possible number of meaningful GP models. One thing to note is that we have fixed the number and position of inducing points while optimizing hyper-parameters for this experiment. While a more optimized set of inducing points will have better results, for data-sets of the order $\mathcal{N} \sim 10^5$ Distributed GPs algorithm starts outperforming Variational inference. For the case of Distributed GPs we have again fixed the number of points in an expert, but this does not have a significant impact on the trend on RMSE with increasing data-points.

7.5.2 Experiments on Flight Test Data

We perform experiments on flight loads data produced during flight test phase at Airbus. Loads are measured across the wingspan using strain gauges. Shear load T_z and bending moment M_x as described in figure 7.10 are used as two outputs for this exercise. η or point of action of forces and angle of attack α are the two inputs. The aircraft is in quasi-statis equilibrium in all conditions and there are no dynamic effects observed throughout this

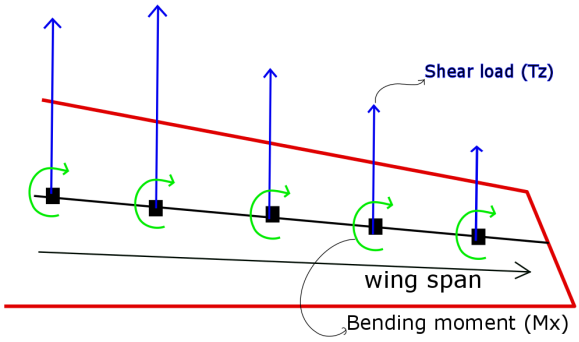


Figure 7.10: Wing Load Diagram

data-set. All data is normalized according to Airbus policy.

The relation between T_z and M_x can be written as:

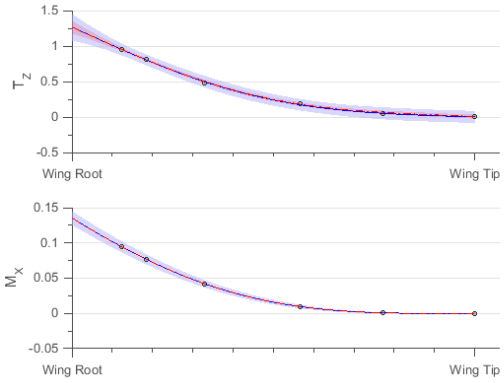
$$M_x(\eta, \alpha) = \int_{\eta}^{\eta_{edge}} T_Z(x, \alpha)(x - \eta)dx \tag{7.29}$$

.

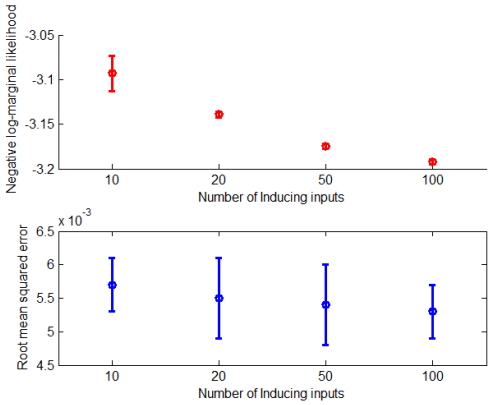
The integral operator in equation 7.29 is applied on the η dimension. Here, η_{edge} denotes the edge of the wingspan. The forces are measured at 5 points on the η dimension and at 8800 points on the α dimension. In this experiment we first compare plots of joint MTGP and independent GP, and then compare the measures of log marginal likelihood and RMSE for varying number of inducing points.

Figure 7.11(a) shows the independent (blue) and joint fit (red) of two GP. The top figure shows T_Z with the variance of dependent GP plotted in red and variance of independent GP plotted in blue. The bottom figure shows plots for M_X. 100 inducing points in the input space are used to learn and plot the figure. The variance of red is smaller than that of blue showing the improvement in confidence when imposing relationships in the GP architecture. The relationship between T_Z and M_X gives rise to better confidence during the loads prediction. This added information is very useful when identifying faulty sensor data since equation 7.29 will push data-points which do not satisfy the relationship out of the tight confidence interval.

Figure 7.11(b) shows improvement in the negative log-marginal likelihood and RMSE plots upon increasing number of inducing points. 10 sets of experiments were run on 75% of the data as training set and 25% of the data as the test set, the training and the test sets were chosen randomly. We learn the optimal values of hyper-parameters and inducing points for all the 10 sets of experiments of training data. Finally, RMSE values are evaluated with respect to the test set and log-marginal likelihood are evaluated for



(a) 2σ confidence interval and mean of the dependent GP are represented in red shade and solid red line. 2σ confidence interval and mean of the independent GP are represented in blue shade and solid blue line. Experiment was run on 8800 data-points Noisy data is denoted by circles only 1 α step is plotted. Confidence interval improves upon adding the relationship kernel.



(b) Progression of RMSE and log-likelihood upon increasing number of inducing points. Top plot shows the value of mean and variance of log-marginal likelihood. The bottom figure in blue shows the mean and variance of root mean squared error. 10 sets of experiments were run on 75% of the data as training set and 25% of the data as the test set, the training and test sets were chosen randomly.

Figure 7.11: Experimental results for aircraft flight loads

each learned model. The RMSE and log-likelihood improve upon increasing the number of inducing points.

Cost saving in MDO By enabling relationships in building models we can save significant costs while performing MDO. Earlier works have reduced the cost of optimization by incorporating Gradient Enhanced Kriging into the optimization process [Liem 2015], a similar cost saving can be obtained by using the proposed methods. Suppose we are designing a part of an aircraft by performing optimization over a fluid-structure interaction loop. The fluid part is being solved by a costly CFD code, while the structural part is being solved by a cheap CSM code. The CFD code results in *pressures* on the aerodynamic mesh, while the CSM part results in $deformations$ of the structural mesh. If we can leverage the relationship between pressure and deformation[3] then we can effectively reduce the number of required CFD runs. These type of cost savings can be made in several MDO problems.

[3] $\int pressures = Stiffness \times deformation$

7.6 Summary and discussion

The current chapter demonstrates how to incorporate physical laws into MTGPs regression framework and then eventually how to scale the framework to large data-sets. Section 7.2.1 demonstrates how to enforce physical laws, which are in the form of linear operators. Whereas section 7.2.2 demonstrates how to encode physical laws, which are in the form of non-linear operators. The improved accuracy of the model is then demonstrated both on a toy data-set and a data-set of real flight-test (section 7.3).

Section 7.4 demonstrates how to scale the MTGP to large number of data-points, both using a Variational approximation (section 7.4.1) and a Distributed GPs approximation (section 7.4.2). We then demonstrate the scalability on a toy data-set and a data-set from flight test and compare the accuracy of Distributed GPs and Variational inference on the toy-data-set.

Several future paths can be explored by leveraging this kind of relationship enabled MTGPs. Using a quadratic relationship enabled MTGP we can indirectly add inequality constraints. Leveraging the information of prior relationships can greatly reduce the cost of performing MDO. Finally, by enforcing physical laws of the system we can automatically identify faulty sensors. All these are interesting paths and should be explored further.

Part IV

Conclusions

Chapter 8

Conclusions and perspectives

In this thesis, we have built better GP regression models by integrating the prior knowledge of aircraft design with experimental or simulation data. This task is very important because of the following reasons:

1. **Cost**: Generating highly accurate data of physical systems is a costly exercise. We can use the surrogate models as a cheap substitute for accurate models.
2. **Prior knowledge**: Several types of relationships between mechanical quantities have already been discovered for aircraft design tasks. We would like to reuse them in our model building.
3. **Physically consistent**: Enforcing physical laws of the system will give rise to robust and physically consistent models.

We demonstrate how to incorporate the prior information by mainly changing the covariance functions of the GP. Prior information in terms of smoothness, linearity, differentiability, etc. can be easily encoded using simple covariance functions. Similarly, relationships between multiple outputs can be learned or enforced by treating the 'output number' as an extra dimension, this enables us again to easily create covariance functions for 'Multi-task GPs'. We now come back to the four questions that were posed in the first chapter of this thesis and highlight the contributions for each topic.

How to add *a-priori* information of a pattern in a learning algorithm?

Several types of prior pattern information can be enforced into a GP regression algorithm. We first, demonstrate how to use GP regression to automatically detect modal parameters of structural dynamics. Using the spectral mixture kernels, peaks in the frequency domain can be identified automatically. To the best of our knowledge, such a method has not

been used in the earlier literature to identify modal parameters. We then demonstrate how to identify the onset of non-linear behaviour in physical systems by using a change-point kernel. For example we identify the initiation of flow separation in NACA 0012 airfoil and initiation of plasticity in aluminum alloy using a statistical criterion for automatic detection of non-linearity [Chiplunkar 2016b]. We finally build a GP model to predict the position of aerodynamic shock in the transonic regime (section 5.3.5) [Chiplunkar 2017a]. The predicted position and accuracy of shock is better than the state-of-the-art models.

How to merge *a-priori* information of simulations with experiments?

Multi-fidelity models in GPs were first introduced by [Kennedy 2000] and have been popular in making cost effective surrogate modelling of simulation codes. We demonstrate how multi-fidelity GP models can be extended to perform model updating or extrapolation of experimental data. We include an error term and a translation term into a normal multi-fidelity model to account for differences in the simulation model and experimental data.

How to add *a-priori* information of relationships between measurements?

The physical laws or relationships have been developed after centuries of observations and iterative experiments. We extend the framework of gradient enhanced kriging to enforce any relationship between outputs. We also provide a novel graphical interpretation of this enforced multiple-output regression process. The improvement in model accuracy is then validated both on a toy data-set and a data-set of real flight-test.

How to scale GP regression to large data-sets?

Calculating the precision matrix is an important task in choosing appropriate hyper-parameters and calculating the posterior mean and variance. Unfortunately, this task puts an upper limit of $N \sim 10^4$ on the number of data-points. This problem is further aggravated when we are creating a joint Gram matrix between several outputs. We demonstrate how to scale single output GPs using 'Sparse GPs' and 'Distributed GPs' methods.

We demonstrate the capability of scaling, by building a distributed GP model for interpolating pressures on millions of nodes in a CFD mesh. This surrogate model was used in a recent Airbus flight test campaign to compare pressures predicted from a high-fidelity CFD computation to pressures measured on the wing, in real time. We then demonstrate how to scale up 'Multiple-output GPs' to large number of data-points, both using a sparse approximation and a distributed GP approximation. We then validate the scalability on a toy data-set and a data-set from flight test and compare the accuracy of distributed GP and variational inference on the toy data-set.

8.1 Perspectives

Several future paths can be explored by merging prior knowledge of aircraft design with GP regression. The developments in this thesis for the field of aircraft engineering can be easily replicated to other engineering domains. Other domains also exhibit a similar problematic, where significant gains can be achieved by combining data and prior knowledge [Lookman 2017, Jidling 2017, Alvarez 2009]. The proposed GP models can then be used for **system identification** or **system control** [Kocijan 2016, Frigola 2014, Deisenroth 2011].

Similarly, by incorporating prior patterns we can create more **accurate surrogate models** when compared to state-of-the-art techniques, these surrogate models can be then used as cheap replacements to more costly simulation codes. An upcoming field of research, called Probabilistic Numerics, uses similar Bayesian principles to propagate uncertainty through engineering models [Hennig 2015].

The capability to integrate multiple outputs correlated through a known relationship opens other options for future research. Merging simulation models with experimental data is a recurring theme in several **verification and validation** problems. Multi-output GPs provide a different way to merge these two outputs. Leveraging the prior knowledge of simulation model in presence of **non-Markov** differences between outputs should be studied in detail. Moreover, by enforcing physical laws in a surrogate model, we can develop schemes to perform **cost efficient MDO**, automatically **detect faulty sensors** and enforce **inequality constraints** between outputs.

Appendix

Appendix A

Gaussian Identities

In this appendix we provide basic background on each topic to make this thesis self-contained. Let $\boldsymbol{x} = \{x_1, \ldots, x_i, \ldots, x_N\}$ be a set of N scalar Gaussian random variables $x_i \in \mathcal{R}$. Such that:

$$x_i \sim \mathcal{N}(\mu_i, \sigma_i^2) \tag{A.1}$$

Here, μ_i represents the scalar mean, while σ_i^2 represents the variance. Their probability density can thus written as:

$$\Pr[x_i] = \mathcal{N}(\mu_i, \sigma_i^2) \tag{A.2}$$

$$= \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp^{-\frac{(x_i-\mu_i)^2}{2\sigma_n^2}} \tag{A.3}$$

Let us partition the set $\boldsymbol{x}$ into two sets, $\boldsymbol{a}$ and $\boldsymbol{b}$ of size N_a and N_b respectively. Such that $\boldsymbol{a} \cup \boldsymbol{b} = \boldsymbol{x}$, while each set may contain one or more variables. We denote $\boldsymbol{x_a}$ as the ordered collection of random variables in set $\boldsymbol{a}$, thus $\boldsymbol{x_a} \in \mathcal{R}^{N_a}$ is a collection of random variables. Thereby, the probability distribution of random vector $\boldsymbol{x_a}$ can be denotes as equation A.4 and similarly for $\boldsymbol{x_b}$.

$$\Pr[\boldsymbol{x_a}] = \mathcal{N}(\boldsymbol{\mu_a}, \boldsymbol{\Sigma_a}) \tag{A.4}$$

$$= \frac{1}{\sqrt{(2\pi)^{N_a}\boldsymbol{\Sigma_a}}} exp\left[-\frac{1}{2}(\boldsymbol{x_a} - \boldsymbol{\mu_a})^T \boldsymbol{\Sigma_a}(\boldsymbol{x_a} - \boldsymbol{\mu_a})\right] \tag{A.5}$$

Here, $\boldsymbol{\mu_a}$ is the ordered collection of means in the random variable set $\boldsymbol{a}$, while $\boldsymbol{\Sigma_a}$ represents the covariance matrix. We now describe the joint, marginal and conditional distributions which are also Gaussian distributions, for a detailed treatment refer to [Bishop 2006].

A.1 Joint distribution

If the probability densities of sets $\boldsymbol{a}$ and $\boldsymbol{b}$ can be written as, $\Pr[\boldsymbol{x_a}] = \mathcal{N}(\boldsymbol{\mu_a}, \boldsymbol{\Sigma_a})$ and $\Pr[\boldsymbol{x_b}] = \mathcal{N}(\boldsymbol{\mu_b}, \boldsymbol{\Sigma_b})$ respectively. Then the joint distribution between two random vectors $\boldsymbol{x_a}$ and $\boldsymbol{x_b}$ is given by $\Pr[\boldsymbol{x_a}, \boldsymbol{x_b}]$ in equation A.6 ($\boldsymbol{a} \cup \boldsymbol{b} = \boldsymbol{x}$).

$$\Pr[\boldsymbol{x}] = \Pr[\boldsymbol{x_a}, \boldsymbol{x_b}] = \Pr\begin{bmatrix}\boldsymbol{x_a}\\ \boldsymbol{x_b}\end{bmatrix} \tag{A.6}$$

$$= \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu_a}\\ \boldsymbol{\mu_b}\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma_a} & \boldsymbol{\Sigma_{ab}}\\ \boldsymbol{\Sigma_{ba}} & \boldsymbol{\Sigma_b}\end{bmatrix}\right) \tag{A.7}$$

A.2 Sum of Gaussians

If the size of the random vectors $\boldsymbol{x_a}$ and $\boldsymbol{x_b}$ is same, then if a new vector $\boldsymbol{x_c} = \boldsymbol{x_a} + \boldsymbol{x_b}$ is sum of two Gaussians, it is also a multi-variate Gaussian with probability density given as follows:

$$\Pr[\boldsymbol{x_a}] = \mathcal{N}(\boldsymbol{\mu_a} + \boldsymbol{\mu_b}, \boldsymbol{\Sigma_a} + \boldsymbol{\Sigma_b}) \tag{A.8}$$

A.3 Affine property of Gaussians

If $\boldsymbol{x_d}$ is a linear transformation of $\boldsymbol{x_a}$ such that $\boldsymbol{x_c} = \boldsymbol{l} + \boldsymbol{M}\boldsymbol{x_a}$. Here, $\boldsymbol{l}$ and $\boldsymbol{M}$ are a vector and a matrix of constants. Then $\boldsymbol{x_d}$ it is also a multi-variate Gaussian with probability density given as follows:

$$\Pr[\boldsymbol{x_d}] = \mathcal{N}(\boldsymbol{l} + \boldsymbol{M}\boldsymbol{\mu_a}, \boldsymbol{M}\boldsymbol{\Sigma_a}\boldsymbol{M}^T) \tag{A.9}$$

In fact the affine property is valid for any linear transformation of a Gaussian distribution. This is the same property that we use to enforce relationships defined by linear operations between multiple outputs.

A.4 Marginal Distribution

The marginal distribution between two probability distributions is given as:

$$\Pr[\boldsymbol{x_a}] = \int \Pr[\boldsymbol{x_a}, \boldsymbol{x_b}] d\boldsymbol{x_b} \tag{A.10}$$

We are effectively integrating out the random variable $\boldsymbol{x_b}$. This is the same principle while calculating the 'marginal likelihood' to choose hyper-parameters, we integrate out the latent functions.

For a Gaussian random vector (multi-variate Gaussian) as described in equation A.6, the marginal distribution is a Gaussian. This is also called the marginalization property of Gaussians.

$$\Pr[\boldsymbol{x_a}] = \int \Pr[\boldsymbol{x_a}, \boldsymbol{x_b}] d\boldsymbol{x_b} \tag{A.11}$$

$$= \mathcal{N}(\boldsymbol{\mu_a}, \boldsymbol{\Sigma_a}) \tag{A.12}$$

A.5 Conditional Distribution

The probability of $\boldsymbol{x_a}$ given a value of $\boldsymbol{x_b} = \bar{\boldsymbol{x_b}}$, is called as conditional probability and is written as $\Pr[\boldsymbol{x_a} \mid \boldsymbol{x_b} = \bar{\boldsymbol{x_b}}]$.

$$\Pr[\boldsymbol{x_a} \mid \boldsymbol{x_b} = \bar{\boldsymbol{x_b}}] = \frac{\Pr[\boldsymbol{x_a}, \boldsymbol{x_b}]}{\Pr[\boldsymbol{x_b} = \bar{\boldsymbol{x_b}}]} \tag{A.13}$$

For a Gaussian random vector (multi-variate Gaussian) as described in equation A.6, the conditional distribution is a Gaussian.

$$\Pr[\boldsymbol{x_a} \mid \boldsymbol{x_b} = \bar{\boldsymbol{x_b}}] = \mathcal{N}(\boldsymbol{\mu_{a|b}}, \boldsymbol{\Sigma_{a|b}}) \tag{A.14}$$

$$\boldsymbol{\mu_{a|b}} = \boldsymbol{\mu_a} + \boldsymbol{\Sigma_{ab}} \boldsymbol{\Sigma_b}^{-1} (\bar{\boldsymbol{x_b}} - \boldsymbol{\mu_b}) \tag{A.15}$$

This is the same principle while calculating the posterior distribution of a GP, there we are conditioning a GP given an observation dataset.

Appendix B

Proper Orthogonal Decomposition for pressure snapshots

This appendix demonstrates how to perform Proper Orthogonal Interpolation (POD) based interpolation using pressure snapshots. There exist two types of POD a classical POD [Gurvich 1969] and snapshot POD [Romanowski 1996]. We will use the snapshot POD in this thesis, since it is a highly popular in aerodynamic interpolation use cases.

B.1 Pressure Snapshot

Let us first start by defining a pressure snapshot. There exists a 3 dimensional spatial vector $\omega_i \in \mathbb{R}^3$ such that $\omega_i = \{(\omega_i^1, \omega_i^2, \omega_i^3)\}$. Here, $i \in [1, N_{nodes}]$ are the spatial coordinates of the i^{th} pressure node in a mesh containing N_{nodes} pressure nodes. Similarly there exists a D dimensional parameter vector $d_j \in \mathbb{R}^D$, for $d_j = \{(d_j^1, d_j^2, \ldots, d_j^D)\}$. Here, $j \in [1, N_{parameter}]$ correspond to the j^{th} parameter set. The parameters can be any non-spatial parameter which are desired to be interpolated, some common examples include Mach, Angle of Attack for steady aerodynamics and time or frequency for unsteady aerodynamics. Since, this thesis relates to interpolating steady aerodynamics d_j will correspond to the j^{th} run in a total of $N_{parameter}$ simulations or experiments.

The pressure measured on the i^{th} pressure node for the j^{th} parameter set will be denoted as $p_j(\omega_i)$ defined by the equation B.1. We next define the matrix $\Omega = \{\omega_1; \omega_2; \ldots; \omega_{N_{nodes}}\}$ for $\Omega \in \mathbb{R}^{N_{nodes} \times 3}$ containing the full spatial information of the CFD mesh. Finally, the pressure snapshot for the CFD run j will be denoted as $P_j(\Omega) = \{p_j(\omega_1); p_j(\omega_2); \ldots; p_j(\omega_{N_{nodes}})\}$ for $P_j(\Omega) \in \mathbb{R}^{N_{nodes}}$ defined by the equation B.2.

$$p_j(\omega_i) = f_{pressure}(\omega_i, d_j) \tag{B.1}$$

$$P_j(\Omega) = f_{pressure}(\Omega, d_j) \tag{B.2}$$

B.2 POD for aerodynamic snapshots

The optimal POD basis vectors $\phi(\Omega)$ are chosen so that they maximize the cost described in equation B.3. $(\cdot,\cdot)$ is an inner product and $\langle\cdot,\cdot\rangle$ is the parameter-averaging operation [Berkooz 1993, Epureanuj 1999]. Solving this optimization problem leads to an eigen value problem, where $\phi(\Omega)$ are the eigen-vectors.

$$max_{\phi} \frac{\langle (P_j(\Omega), \phi(\Omega))^2 \rangle}{(\phi(\Omega), \phi(\Omega))} \tag{B.3}$$

The idea of snapshot POD is to write the eigen-vector $\phi(\Omega)$ in terms of the pressure snapshots (equation B.4) and not in terms of all inputs.

$$\phi^l(\Omega) = \sum_{j=1}^{N_{parameter}} \beta_j^l P_j(\Omega) \tag{B.4}$$

Here, $\phi^l(\Omega)$ is the l^{th} eigen-vector and the coefficients β_j^l can be shown to satisfy the eigen problem in equation B.5 and B.6.

$$R\beta = \Lambda\beta \tag{B.5}$$

here,

$$\beta = \begin{pmatrix} \beta^1 \\ \beta^2 \\ \dots \\ \beta^{N_{parameter}} \end{pmatrix} \quad and \quad R_{lm} = \frac{1}{N_{parameter}} (P_l(\Omega), P_m(\Omega)) \tag{B.6}$$

The matrix R is called the correlation matrix for $R \in \mathbb{R}^{N_{parameter} \times N_{parameter}}$. The eigen-vector $\beta^l \in \mathbb{R}^{1 \times N_{parameter}}$ represents the participation factors for equation B.5. We have hence calculated the eigen-vectors for our matrix of pressure snapshots.

The original pressure snapshots can be now be written as a linear combination of the eigen-vectors equation B.7.

$$P_j(\Omega) = \sum_{l=1}^{p} a^l(d_j)\phi^l(\Omega) \tag{B.7}$$

Here, $a^l(d_j)$ denotes the participation factor or amplitude for the mode l at a point d_j in the parameter space. Once we evaluate the eigen-vectors calculating the amplitudes become a task of solving equation B.8.

$$a^l(d_j) = (\phi_l(\Omega), P_j(\Omega)) \tag{B.8}$$

Note, further speed up in reconstruction of pressure snapshots can be gained by taking $p < N_{parameter}$. Only, taking into account the modes which correspond to the highest participation can significantly improve the reconstruction times [Tan 2003, Allemang 2011].

B.3 Interpolation

The above section describes how to calculate eigen-vectors from a set of pressure snapshots. Furthermore we can use equation B.7 and B.8 to reconstruct the initial snapshots. This section describes how to interpolate or predict the snapshots at an unknown point d_{new}in the parameter space.

If the participation factors $a^l(d_j)$ are smooth functions of parameters $d \in \mathbb{R}^D$, interpolation can be used to determine the participation factors at desired point d_{new}. In this thesis we use cubic spline [Bartels 1987] to interpolate the participation factors. Once the new participation factors are obtained a new snapshot can be constructed using the equation B.9. A pseudo-code is given in algorithm 1 of the full process.

$$P_{new}(\Omega) = \sum_{l=1}^{p} a^l(d_{new})\phi^l(\Omega) \tag{B.9}$$

Due to the reduced order modelling we are interpolating only the $N_{parameter}$ of the order of 10^2 parameters instead of N_{nodes} of the order of 10^4 pressure nodes. Note, for the reminder of this thesis we will use all the available modes for interpolating the snapshots i.e. $p = N_{parameter}$.

Algorithm 1: Algorithm for POD + Interpolation.

Data: $P_j(\Omega)$ (pressure snapshots),
d_j for $j \in [1 : N_{parameter}]$ (set of parameters),
d_{new} (desired point)

Result: $P_{new}(\Omega)$ (Interpolated pressure snapshot at point d_{new})

`pod()`

`// Calculating eigen-vectors using equations B.6, B.5 and B.4`

$R_{lm} = \frac{1}{N_{parameter}}(P_l, P_m^*)$

$R\beta = \Lambda\beta$

$\phi^l(\Omega) = \sum_{j=1}^{N_{parameter}} \beta_j^l P_j(\Omega)$

`// Calculating participation factors of eigen-vector` l `at point` d_j

$a_l(d_j) = (\phi_l(\Omega), P_j(\Omega))$

return $\phi^l, \quad a^l(d_j) \quad for \quad l \in [1 : N_{parameter}]$

`interpolate()`

`// Interpolating the participation factor of the` l^{th} `eigen-vector`

foreach $l \in [1 : N_{parameter}]$ **do**

  $a^l(d_{new}) = \mathbf{spline}(\alpha_l(d_j), d_{new})$

end

`// Reconstructing snapshot using the interpolated participation factors and original eigen-vectors`

$P_{new}(\Omega) = \sum_{l=1}^{p} a^l(d_{new})\phi^l()\Omega$

return $P_{new}(\Omega)$

Appendix C

Gaussian Process Regression enforcing Non-linear operators

This chapter reproduces the proof of approximation performed when enforcing non-linear operators in an MTGP framework as provided by [Constantinescu 2013].

For simplicity let us take the case of an explicit relationship between two outputs y^1 and y^2. Suppose we measure the two outputs with some error (ϵ_{n1} and ϵ_{n2}), while the true physical process is defined by latent variables (f^1 and f^2). Then the relation between the output function, measurement error, and true physical process can be written as follows.

$$y^1 = f^1 + \epsilon_{n1} \tag{C.1}$$
$$y^2 = f^2 + \epsilon_{n2} \tag{C.2}$$

Here, ϵ_{n1} and ϵ_{n2} are measurement errors sampled from a white noise Gaussian $\mathcal{N}(0, \sigma_{n1}^2)$ and $\mathcal{N}(0, \sigma_{n2}^2)$ respectively. While, the relation between the latent function can be expressed as follows:

$$f^1(z) = \mathcal{L}\left(f^2(x), z\right) \tag{C.3}$$

Here $\mathcal{L}(.) \in \mathcal{C}^2$ is an operator defining the relation between f^1 and f^2. We can write a

zero mean GP prior for the full set of inputs and outputs $\{\boldsymbol{X_{joint}}, \boldsymbol{y_{joint}}\}$, as equation 6.5.

$$\begin{aligned}\Pr[\boldsymbol{y_{joint}}] &= \Pr\begin{bmatrix}\boldsymbol{y_{joint}}(x,1)\\ \boldsymbol{y_{joint}}(x,2)\end{bmatrix}\\ &= GP\left[\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}Cov(\boldsymbol{y_{joint}}(x,1),\boldsymbol{y_{joint}}(x,1)) & Cov(\boldsymbol{y_{joint}}(x,1),\boldsymbol{y_{joint}}(x,2))\\ Cov(\boldsymbol{y_{joint}}(x,2),\boldsymbol{y_{joint}}(x,1)) & Cov(\boldsymbol{y_{joint}}(x,2),\boldsymbol{y_{joint}}(x,2))\end{pmatrix}\right]\\ &= GP\left[\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}Cov(\boldsymbol{f^1}(x),\boldsymbol{f^1}(x))+\sigma_{n1}^2 & Cov(\boldsymbol{f^1}(x),\boldsymbol{f^2}(x))\\ Cov(\boldsymbol{f^2}(x),\boldsymbol{f^1}(x)) & Cov(\boldsymbol{f^2}(x),\boldsymbol{f^2}(x))+\sigma_{n2}^2\end{pmatrix}\right]\end{aligned} \tag{C.4}$$

Here, $Cov(\boldsymbol{y_{joint}}(x,j),\boldsymbol{y_{joint}}(x,j))$ is called the auto-covariance between observations of the j^{th} output, while $Cov(\boldsymbol{y_{joint}}(x,2),\boldsymbol{y_{joint}}(x,1))$ is called cross-covariance between the 2^{nd} and the 1^{st} outputs. Similarly, $Cov(\boldsymbol{f^j}(x),\boldsymbol{f^j}(x))$ is the auto-covariance function between j^{th} latent function, while $Cov(\boldsymbol{f^1}(x),\boldsymbol{f^2}(x))$ is the cross-covariance function between the 2^{nd} and the 1^{st} latent outputs. We are thus, interested in evaluating the Gram matrix $\boldsymbol{K_{XX}}$ between the full dataset.

$$\boldsymbol{K_{XX}} = \begin{pmatrix}Cov(f^1(x),f^1(x))+\sigma_{n1}^2 & Cov(f^1(x),f^2(x))\\ Cov(f^2(x),f^1(x)) & Cov(f^2(x),f^2(x))+\sigma_{n2}^2\end{pmatrix} \tag{C.5}$$

$$= \begin{pmatrix}\boldsymbol{K_{11}}+\mathbb{I}\sigma_{n1}^2 & \boldsymbol{K_{12}}\\ \boldsymbol{K_{21}} & \boldsymbol{K_{22}}+\mathbb{I}\sigma_{n2}^2\end{pmatrix} \tag{C.6}$$

Proof The *Taylor* series expansion of $\mathcal{L}(f^2)$ around $E[f^2]$ gives:

$$\begin{aligned}\mathcal{L}\left(f^2\right) &= \mathcal{L}\left(E[f^2]\right) + L\delta_2 + \frac{1}{2}\delta_2^T H\delta_2 + \mathcal{O}(\delta_2^3)\\ L &= \frac{\partial\mathcal{L}}{\partial x}\Big|_{f^2=E[f^2]}\\ H &= \frac{\partial^2\mathcal{L}(f^2)}{\partial^2 x}\Big|_{f^2=E[f^2]}\end{aligned} \tag{C.7}$$

Where L is the Jacobian matrix of $\mathcal{L}(.)$ evaluated at the mean ($E[f^2]$) of latent output (f^2), while H is the derivative of L evaluated at the mean of f^2. δ_2 is the amplitude of small variations of f^2, introduced by the Taylor series expansion of $\mathcal{L}(f^2)$ with respect to $E[f^2]$.

On taking expectation of equation C.3:

$$\begin{aligned}E[f^1] &= E[\mathcal{L}(f^2)]\\ &= \mathcal{L}(E[f^2]) + \frac{1}{2}E[\delta_2^T H\delta_2] + E[\mathcal{O}(\delta_2^3)]\end{aligned} \tag{C.8}$$

Subtracting the above equation C.8 from equation C.3 gives:

$$f^1 - E[f^1] = \mathcal{L}(f^2) - \mathcal{L}(E[f^2]) - \frac{1}{2}E[\delta_2^T H \delta_2] - E[\mathcal{O}(\delta_2^3)] \tag{C.9}$$

Expanding $\mathcal{L}(f^2)$ in the above equation:

$$f^1 - E[f^1] = \mathcal{L}\left(E[f^2]\right) + L\delta_2 + \frac{1}{2}\delta_2^T H \delta_2 + \mathcal{O}(\delta_2^3) - \mathcal{L}(E[f^2]) - \frac{1}{2}E[\delta_2^T H \delta_2] - E[\mathcal{O}(\delta_2^3)] \tag{C.10}$$

$$\delta_1 = L\delta_2 + \left[\frac{1}{2}\delta_2^T H \delta_2 - \frac{1}{2}E[\delta_2^T H \delta_2]\right] + \left[\mathcal{O}(\delta_2^3) - E[\mathcal{O}(\delta_2^3)]\right] \tag{C.11}$$

Here, δ_1 is the amplitude of small variations of f^1. Multiplying the above equation by δ_2^T and taking the expectation gives:

$$\begin{aligned} E[\delta_1 \delta_2^T] &= LE[\delta_2 \delta_2^T] + E\left[\left[\frac{1}{2}\delta_2^T H \delta_2 - \frac{1}{2}E[\delta_2^T H \delta_2]\right]\delta_2^T\right] + E\left[\left[\mathcal{O}(\delta_2^3) - E[\mathcal{O}(\delta_2^3)]\right]\delta_2^T\right] \\ Cov(f^1, f^2) &= LCov(f^2, f^2) + E\left[\left[\frac{1}{2}\delta_2^T H \delta_2 - \frac{1}{2}E[\delta_2^T H \delta_2]\right]\delta_2^T\right] + E\left[\left[\mathcal{O}(\delta_2^3) - E[\mathcal{O}(\delta_2^3)]\right]\delta_2^T\right] \end{aligned} \tag{C.12}$$

If we eliminate the terms that are or order greater than $\mathcal{O}(\delta_2^3)$ then we get the following equation for $Cov(f^1, f^1)$:

$$Cov(f^1, f^2) = LCov(f^2, f^2) \tag{C.13}$$

Similarly by multiplying the equation C.10 by δ_1^T, taking the expectation and eliminating higher order terms will give:

$$\begin{aligned} E[\delta_1 \delta_1^T] &= LE[\delta_2 \delta_1^T] \\ Cov(f^1, f^1) &= LCov(f^2, f^1) \\ &= LCov(f^1, f^2)^T \\ &= L[LCov(f^2, f^2)]^T \\ &= LCov(f^2, f^2)L^T \end{aligned} \tag{C.14}$$

The last equation comes because $Cov(f^2, f^2)$ is symmetric, hence $Cov(f^2, f^2)^T = Cov(f^2, f^2)$. We can thus write the Gram matrix $\boldsymbol{K_{XX}}$ as:

$$\boldsymbol{K_{XX}} = \begin{pmatrix} L\boldsymbol{K_{22}}L^T + \mathbb{I}\sigma_{n1}^2 & L\boldsymbol{K_{22}} \\ \boldsymbol{K_{22}}L^T & \boldsymbol{K_{22}} + \mathbb{I}\sigma_{n2}^2 \end{pmatrix} \tag{C.15}$$

The proof for enforcing non-linear relationships in a multi-task GP regression framework is thus complete.

Bibliography

[Allemang 1998] Randall J Allemang et DL Brown. *A unified matrix polynomial approach to modal identification.* Journal of Sound and Vibration, vol. 211, no. 3, pages 301–322, 1998.

[Allemang 2011] Randall J Allemang, Allyn W Phillips et Matthew R Allemang. *Application of principal component analysis methods to experimental structural dynamics.* Structural Dynamics, Volume 3, pages 477–498, 2011.

[Alvarez 2009] Mauricio A. Alvarez, David Luengo et Neil D. Lawrence. *Latent Force Models.* In David A. Van Dyk et Max Welling, editeurs, AISTATS, volume 5 of *JMLR Proceedings*, pages 9–16. JMLR.org, 2009.

[Alvarez 2011] Mauricio A Alvarez, Lorenzo Rosasco et Neil D Lawrence. *Kernels for vector-valued functions: A review.* arXiv preprint arXiv:1106.6251, 2011.

[Barrault 2004] Maxime Barrault, Yvon Maday, Ngoc Cuong Nguyen et Anthony T Patera. *An 'empirical interpolation'method: application to efficient reduced-basis discretization of partial differential equations.* Comptes Rendus Mathematique, vol. 339, no. 9, pages 667–672, 2004.

[Bartels 1987] Richard H Bartels, John C Beatty et Brian A Barsky. An introduction to splines for use in computer graphics and geometric modeling. Morgan Kaufmann, 1987.

[Bauer 1998] Eric Bauer et Ron Kohavi. *An empirical comparison of voting classification algorithms: Bagging, boosting, and variants.* Machine learning, vol. 36, no. 1, page 2, 1998.

[Beckert 2001] Armin Beckert et Holger Wendland. *Multivariate interpolation for fluid-structure-interaction problems using radial basis functions.* Aerospace Science and Technology, vol. 5, no. 2, pages 125–134, 2001.

[Berkooz 1993] Gal Berkooz, Philip Holmes et John L Lumley. *The proper orthogonal decomposition in the analysis of turbulent flows.* Annual review of fluid mechanics, vol. 25, no. 1, pages 539–575, 1993.

[Bishop 2006] Christopher M Bishop. *Pattern Recognition*. Machine Learning, 2006.

[Bochner 1959] S. Bochner, M. Tenenbaum et H. Pollard. Lectures on fourier integrals. Annals of mathematics studies. Princeton University Press, 1959.

[Bochner 2016] Salomon Bochner. Lectures on fourier integrals.(am-42), volume 42. Princeton University Press, 2016.

[Bonilla 2007] Edwin V Bonilla, Kian M Chai et Christopher Williams. *Multi-task Gaussian process prediction*. In Advances in neural information processing systems, pages 153–160, 2007.

[Bonilla 2008] Edwin Bonilla, Kian Ming Chai et Chris Williams. *Multi-task Gaussian Process Prediction*. In J.C. Platt, D. Koller, Y. Singer et S. Roweis, editeurs, Advances in Neural Information Processing Systems 20, pages 153–160. MIT Press, Cambridge, MA, 2008.

[Bosco 2016] Elisa Bosco, Albert Lucchetti, Simon Trapier, Fausto Gill Di Vincenzo, N Gourdain et J Morlier. *Nonlinear transient Fluid/Structure interaction approach using surrogate models: Industrial application to aircraft fairing vibration excited by engine jet flow*. In 15th Dynamics Specialists Conference, page 2049, 2016.

[Bouhlel 2016a] Mohamed Amine Bouhlel. *Optimisation auto-adaptative en environnement d'analyse multidisciplinaire via les modèles de krigeage combinés à la méthode PLS*. PhD thesis, Toulouse, ISAE, 2016.

[Bouhlel 2016b] Mohamed Amine Bouhlel, Nathalie Bartoli, Abdelkader Otsmane et Joseph Morlier. *An Improved Approach for Estimating the Hyperparameters of the Kriging Model for High-Dimensional Problems through the Partial Least Squares Method*. Mathematical Problems in Engineering, vol. 2016, 2016.

[Boyle 2005] Phillip Boyle et Marcus Frean. *Dependent Gaussian processes*. In In Advances in Neural Information Processing Systems 17, pages 217–224. MIT Press, 2005.

[Braconnier 2011] T Braconnier, M Ferrier, J-C Jouhaud, M Montagnac et P Sagaut. *Towards an adaptive POD/SVD surrogate model for aeronautic design*. Computers & Fluids, vol. 40, no. 1, pages 195–209, 2011.

[Brincker 2000] Rune Brincker, Lingmi Zhang et P Andersen. *Modal identification from ambient responses using frequency domain decomposition*. In Proc. of the 18*'International Modal Analysis Conference (IMAC), San Antonio, Texas, 2000.

[Cambier 2008] L Cambier et JP Veuillot. *Status of the elsA CFD software for flow simulation and multidisciplinary applications*. AIAA paper, vol. 664, page 2008, 2008.

[Cao 2013] Yanshuai Cao, Marcus A Brubaker, David J Fleet et Aaron Hertzmann. *Efficient optimization for sparse gaussian process regression.* In Advances in Neural Information Processing Systems, pages 1097–1105, 2013.

[Cao 2014] Yanshuai Cao et David J. Fleet. *Generalized Product of Experts for Automatic and Principled Fusion of Gaussian Process Predictions.* CoRR, vol. abs/1410.7827, 2014.

[Caruana 1998] Rich Caruana. *Multitask learning.* In Learning to learn, pages 95–133. Springer, 1998.

[Chastaing 2015] Gaëlle Chastaing et Loic Le Gratiet. *ANOVA decomposition of conditional Gaussian processes for sensitivity analysis with dependent inputs.* Journal of Statistical Computation and Simulation, vol. 85, no. 11, pages 2164–2186, 2015.

[Chauhan 2007] S Chauhan, R Martell, RJ Allemang et DL Brown. *Unified matrix polynomial approach for operational modal analysis.* In Proceedings of the 25th IMAC, Orlando (FL), USA, 2007.

[Chen 2009] Tao Chen et Jianghong Ren. *Bagging for Gaussian process regression.* Neurocomputing, vol. 72, no. 7, pages 1605–1610, 2009.

[Chenhan 2016] D Yu Chenhan, William B March, Bo Xiao et George Biros. *INV-ASKIT: A Parallel Fast Direct Solver for Kernel Matrices.* In Parallel and Distributed Processing Symposium, 2016 IEEE International, pages 161–171. IEEE, 2016.

[Chilès 1999] J.P. Chilès et P. Delfiner. Geostatistics: modeling spatial uncertainty. Wiley series in probability and statistics. Wiley, 1999.

[Chiplunkar 2016a] Ankit Chiplunkar, Emmanuel Rachelson, Michele Colombo et Joseph Morlier. *Adding Flight Mechanics to Flight Loads Surrogate Model using Multi-Output Gaussian Processes.* In 17th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, AIAA AVIATION Forum, pages pp. 1–11, Washington, US, 2016. Thanks to the American Institute of Aeronautics and Astronautics editors. The publication is available at AIAA website: https://arc.aiaa.org/doi/abs/10.2514/6.2016-4000 eISBN: 978-1-62410-439-8.

[Chiplunkar 2016b] Ankit Chiplunkar, Emmanuel Rachelson, Michele Colombo et Joseph Morlier. *Identification of Physical Parameters Using Change-Point Kernels.* Society for Industrial and Applied Mathematics, Uncertainty Quantification, 2016, Avril 2016. Poster.

[Chiplunkar 2016c] Ankit Chiplunkar, Emmanuel Rachelson, Michele Colombo et Joseph Morlier. *Sparse Physics-based Gaussian Process for Multi-output Regression using*

Variational Inference. In Proceedings of the 5th International Conference on Pattern Recognition Applications and Methods, pages 437–445, 2016.

[Chiplunkar 2017a] Ankit Chiplunkar, Elisa Bosco et Joseph Morlier. *Gaussian Process for Aerodynamic Pressures Prediction in Fast Fluid Structure Interaction Simulations*. In 12th World Congress on Structural and Multidisciplinary Optimization, Braunschweig, DE, 2017.

[Chiplunkar 2017b] Ankit Chiplunkar et Joseph Morlier. *Operational Modal Analysis in Frequency Domain Using Gaussian Mixture Models*. In Topics in Modal Analysis & Testing, Volume 10, pages 47–53. Springer, 2017.

[Chiplunkar 2017c] Ankit Chiplunkar, Emmanuel Rachelson, Michele Colombo et Joseph Morlier. *Approximate inference in related multi-output Gaussian Process Regression*. Lecture Notes in Computer Science, pages pp. 88–103, 2017.

[Chung 2002] Hyoung-Seog Chung et Juan J Alonso. *Using gradients to construct cokriging approximation models for high-dimensional design optimization problems*. AIAA paper, vol. 317, page 2002, 2002.

[Clapeyron 1834] B.P.É. Clapeyron. Mémoire sur la puissance motrice de la chaleur. École Polytechnique, 1834.

[Cole 2012] Julian D Cole et L Pamela Cook. Transonic aerodynamics, volume 30. Elsevier, 2012.

[Constantinescu 2013] Emil M. Constantinescu et Mihai Anitescu. *Physics-Based Covariance Models for Gaussian Processes with Multiple Outputs*. International Journal for Uncertainty Quantification, vol. 3, 2013.

[Cornford 2002] Dan Cornford, Ian T Nabney et Christopher KI Williams. *Modelling frontal discontinuities in wind fields*. Journal of nonparametric statistics, vol. 14, no. 1-2, pages 43–58, 2002.

[Cox 1977] David Roxbee Cox et Hilton David Miller. The theory of stochastic processes, volume 134. CRC Press, 1977.

[Cummings 2015] Russell M Cummings, William H Mason, Scott A Morton et David R McDaniel. Applied computational aerodynamics: A modern engineering approach, volume 53. Cambridge University Press, 2015.

[Da Veiga 2012] Sébastien Da Veiga et Amandine Marrel. *Gaussian process modeling with inequality constraints*. In Annales de la Faculté des Sciences de Toulouse, volume 21, pages 529–555, 2012.

[Deisenroth 2011] Marc Deisenroth et Carl E Rasmussen. *PILCO: A model-based and data-efficient approach to policy search.* In Proceedings of the 28th International Conference on machine learning (ICML-11), pages 465–472, 2011.

[Deisenroth 2015] Marc Peter Deisenroth et Jun Wei Ng. *Distributed Gaussian Processes.* arXiv preprint arXiv:1502.02843, 2015.

[Domingos 2012] Pedro Domingos. *A few useful things to know about machine learning.* Communications of the ACM, vol. 55, no. 10, pages 78–87, 2012.

[Domingos 2015] P. Domingos. The master algorithm: How the quest for the ultimate learning machine will remake our world. Basic Books. Basic Books, 2015.

[Dubrule 1983] Olivier Dubrule. *Cross validation of kriging in a unique neighborhood.* Mathematical Geology, vol. 15, no. 6, pages 687–699, 1983.

[Durrande 2001] Nicolas Durrande. *Étude de classes de noyaux adaptées à la simplification et à l'interprétation des modèles d'approximation. Une approche fonctionnelle et probabiliste.* PhD thesis, Ecole Nationale Supérieure des Mines de Saint-Etienne, 2001.

[Durrande 2011] Nicolas Durrande, David Ginsbourger, Olivier Roustant et Laurent Carraro. *Additive covariance kernels for high-dimensional Gaussian process modeling.* arXiv preprint arXiv:1111.6233, 2011.

[Durrande 2013a] Nicolas Durrande, David Ginsbourger, Olivier Roustant et Laurent Carraro. *ANOVA kernels and RKHS of zero mean functions for model-based sensitivity analysis.* Journal of Multivariate Analysis, vol. 115, pages 57–67, 2013.

[Durrande 2013b] Nicolas Durrande, James Hensman, Magnus Rattray et Neil D Lawrence. *Gaussian process models for periodicity detection.* arXiv preprint arXiv:1303.7090, 2013.

[Duvenaud 2011] David K Duvenaud, Hannes Nickisch et Carl E Rasmussen. *Additive gaussian processes.* In Advances in neural information processing systems, pages 226–234, 2011.

[Duvenaud 2013] David Duvenaud, James Robert Lloyd, Roger Grosse, Joshua B Tenenbaum et Zoubin Ghahramani. *Structure discovery in nonparametric regression through compositional kernel search.* arXiv preprint arXiv:1302.4922, 2013.

[Duvenaud 2014] David Duvenaud. *Automatic Model Construction with Gaussian Processes.* PhD thesis, Computational and Biological Learning Laboratory, University of Cambridge, 2014.

[Epureanuj 1999] Bogdan I Epureanuj et Jennifer Heeg. *Reduced order models in unsteady aerodynamics.* AIAA Journal, 1999.

[Findley 1991] David F Findley. *Counterexamples to parsimony and BIC.* Annals of the Institute of Statistical Mathematics, vol. 43, no. 3, pages 505–514, 1991.

[Forrester 2007] Alexander IJ Forrester, András Sóbester et Andy J Keane. *Multi-fidelity optimization via surrogate modelling.* In Proceedings of the royal society of london a: mathematical, physical and engineering sciences, volume 463, pages 3251–3269. The Royal Society, 2007.

[Forrester 2008] Alexander Forrester, Andras Sobester et Andy Keane. Engineering design via surrogate modelling: a practical guide. John Wiley & Sons, 2008.

[Forrester 2009] Alexander IJ Forrester et Andy J Keane. *Recent advances in surrogate-based optimization.* Progress in Aerospace Sciences, vol. 45, no. 1, pages 50–79, 2009.

[Fox 2004] Bernard Fox, Michael Boito, John C Graser et Obaid Younossi. *Test and evaluation trends and costs for aircraft and guided weapons.* Rapport technique, RAND CORP SANTA MONICA CA, 2004.

[Franz 2014] Thomas Franz, Ralf Zimmermann, Stefan Görtz et Niklas Karcher. *Interpolation-based reduced-order modelling for steady transonic flows via manifold learning.* International Journal of Computational Fluid Dynamics, vol. 28, no. 3-4, pages 106–121, 2014.

[Frigola 2014] Roger Frigola, Yutian Chen et Carl Edward Rasmussen. *Variational Gaussian process state-space models.* In Advances in neural information processing systems, pages 3680–3688, 2014.

[Gal 2014] Yarin Gal, Mark van der Wilk et Carl E. Rasmussen. *Distributed Variational Inference in Sparse Gaussian Process Regression and Latent Variable Models.* arXiv:1402.1389, 2014.

[Garnett 2013] Roman Garnett, Michael A Osborne et Philipp Hennig. *Active learning of linear embeddings for Gaussian processes.* arXiv preprint arXiv:1310.6740, 2013.

[Gazaix 2017] Anne Gazaix, François Gallard, Vincent Gachelin, Thierry Druot, Stéphane Grihon, Vincent Ambert, Damien Guénot, Rémi Lafage, Charlie Vanaret, Benoît Pauwels, Nathalie Bartoli, Thierry Lefebvre, Patrick Sarouille, Nicolas Desfachelles, Joël Brézillon, Maxime Hamadi et Selime Gürol. *Towards the Industrialization of New MDO Methodologies and Tools for Aircraft Design.* AIAA Journal, 06 2017.

[Ghahramani 2013] Zoubin Ghahramani. *Bayesian non-parametrics and the probabilistic approach to modelling*. Phil. Trans. R. Soc. A, vol. 371, no. 1984, page 20110553, 2013.

[Ginsbourger 2013] David Ginsbourger, Olivier Roustant et Nicolas Durrande. *Invariances of random fields paths, with applications in Gaussian Process Regression*. arXiv preprint arXiv:1308.1359, 2013.

[Goodfellow 2016] Ian Goodfellow, Yoshua Bengio et Aaron Courville. Deep learning. MIT Press, 2016. http://www.deeplearningbook.org.

[Goovaerts 1997] P. Goovaerts. Geostatistics for natural resources evaluation. Applied geostatistics series. Oxford University Press, 1997.

[Gratiet 2012] Loic Le Gratiet. *Recursive co-kriging model for Design of Computer experiments with multiple levels of fidelity with an application to hydrodynamic*. arXiv preprint arXiv:1210.0686, 2012.

[Guillaume 2003] Patrick Guillaume, Peter Verboven, Steve Vanlanduit, Herman Van Der Auweraer et Bart Peeters. *A poly-reference implementation of the least-squares complex frequency-domain estimator*. In Proceedings of IMAC, volume 21, pages 183–192, 2003.

[Gurvich 1969] A.S. Gurvich, B.M. Koprov, L.P. Tsvang, N.L. Byzova, I.M. Bobyleva et FOREIGN TECHNOLOGY DIV WRIGHT-PATTERSON AFB OHIO. Atmospheric turbulence and radio wave propagation (selected articles). Defense Technical Information Center, 1969.

[Hastie 1990] Trevor Hastie et Robert Tibshirani. Generalized additive models. Wiley Online Library, 1990.

[Helterbrand 1994] Jeffrey D Helterbrand et Noel Cressie. *Universal cokriging under intrinsic coregionalization*. Mathematical Geology, vol. 26, no. 2, pages 205–226, 1994.

[Hennig 2015] P. Hennig. *Probabilistic Interpretation of Linear Solvers*. SIAM J on Optimization, vol. 25, Janvier 2015.

[Ibrahim 1977] SR Ibrahim et EC Mikulcik. *A method for the direct identification of vibration parameters from the free response*. The Shock and Vibration Bulletin, 1977.

[James III 1995] OH James III et TG Came. *The Natural Excitation Technique (NExT) for Modal parameter extraction from operating structures*. The International journal of analytical and experimental modal analysis, 1995.

[Jameson 1974] Antony Jameson. *Iterative solution of transonic flows over airfoils and wings, including flows at Mach 1.* Communications on pure and applied mathematics, vol. 27, no. 3, pages 283–309, 1974.

[Jameson 2012] Antony Jameson. *Computational Fluid Dynamics: Past, Present and Future*, 2012.

[Jazwinski 2007] Andrew H Jazwinski. Stochastic processes and filtering theory. Courier Corporation, 2007.

[Jidling 2017] Carl Jidling, Niklas Wahlström, Adrian Wills et Thomas B Schön. *Linearly constrained Gaussian processes.* arXiv preprint arXiv:1703.00787, 2017.

[Kaufman 1999] John Gilbert Kaufman. Properties of aluminum alloys: tensile, creep, and fatigue data at high and low temperatures. ASM international, 1999.

[Kennedy 2000] Marc C Kennedy et Anthony O'Hagan. *Predicting the output from a complex computer code when fast approximations are available.* Biometrika, vol. 87, no. 1, pages 1–13, 2000.

[Kocijan 2016] Juš Kocijan. Modelling and control of dynamic systems using gaussian process models. Springer, 2016.

[Kostantinos 2000] N Kostantinos. *Gaussian mixtures and their applications to signal processing.* Advanced signal processing handbook: theory and implementation for radar, sonar, and medical imaging real time systems, pages 3–1, 2000.

[Krige 1951] Daniel G Krige. *A statistical approach to some basic mine valuation problems on the Witwatersrand.* Journal of the Southern African Institute of Mining and Metallurgy, vol. 52, no. 6, pages 119–139, 1951.

[Le Gratiet 2013] Loic Le Gratiet. *Multi-fidelity Gaussian process regression for computer experiments.* PhD thesis, Université Paris-Diderot-Paris VII, 2013.

[Le 2013] Quoc Le, Tamás Sarlós et Alexander Smola. *Fastfood-computing hilbert space expansions in loglinear time.* In Proceedings of the 30th International Conference on Machine Learning, pages 244–252, 2013.

[LeCun 2015] Yann LeCun, Yoshua Bengio et Geoffrey Hinton. *Deep learning.* Nature, vol. 521, no. 7553, pages 436–444, 2015.

[Lefebvre 2017] Thierry Lefebvre, Nathalie Bartoli, Sylvain Dubreuil, Marco Panzeri, Riccardo Lombardi, Roberto D'Ippolito, Pierluigi Vecchia, Fabrizio Nicolosi et Pier Davide Ciampa. *Methodological enhancements in MDO process investigated in the AGILE European project.* Multidisciplinary Analysis and Optimization Conference, 06 2017.

[Li 2016] Jing Li et Weiwei Zhang. *The performance of proper orthogonal decomposition in discontinuous flows.* Theoretical and Applied Mechanics Letters, vol. 6, no. 5, pages 236–243, 2016.

[Liem 2015] Rhea P Liem, Charles A Mader et Joaquim RRA Martins. *Surrogate models and mixtures of experts in aerodynamic performance prediction for aircraft mission analysis.* Aerospace Science and Technology, vol. 43, pages 126–151, 2015.

[Lloyd 2014] James Robert Lloyd, David Duvenaud, Roger Grosse, Joshua B Tenenbaum et Zoubin Ghahramani. *Automatic construction and natural-language description of nonparametric regression models.* arXiv preprint arXiv:1402.4304, 2014.

[Loeve 1978] M. Loeve. Probability theory ii. F.W.Gehring P.r.Halmos and C.c.Moore. Springer, 1978.

[Lookman 2017] Turab Lookman, Prasanna V Balachandran, Dezhen Xue, John Hogden et James Theiler. *Statistical inference and adaptive design for materials discovery.* Current Opinion in Solid State and Materials Science, vol. 21, no. 3, pages 121–128, 2017.

[Maatouk 2017] Hassan Maatouk et Xavier Bay. *Gaussian process emulators for computer experiments with inequality constraints.* Mathematical Geosciences, vol. 49, no. 5, pages 557–582, 2017.

[MacKay 2003] David JC MacKay. Information theory, inference and learning algorithms. Cambridge university press, 2003.

[March 2012] Andrew March et Karen Willcox. *Provably convergent multifidelity optimization algorithm not requiring high-fidelity derivatives.* AIAA journal, vol. 50, no. 5, pages 1079–1089, 2012.

[March 2015] William B March, Bo Xiao et George Biros. *ASKIT: Approximate skeletonization kernel-independent treecode in high dimensions.* SIAM Journal on Scientific Computing, vol. 37, no. 2, pages A1089–A1110, 2015.

[Marszalek 2007] Marcin Marszalek et Cordelia Schmid. *Semantic hierarchies for visual object recognition.* In Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on, pages 1–7. IEEE, 2007.

[Matheron 1963] Georges Matheron. *Principles of geostatistics.* Economic geology, vol. 58, no. 8, pages 1246–1266, 1963.

[McCulloch 1943] Warren S McCulloch et Walter Pitts. *A logical calculus of the ideas immanent in nervous activity.* The bulletin of mathematical biophysics, vol. 5, no. 4, pages 115–133, 1943.

[Menter 2003] FR Menter, M Kuntz et R Langtry. *Ten years of industrial experience with the SST turbulence model.* Turbulence, heat and mass transfer, vol. 4, no. 1, pages 625–632, 2003.

[Mercer 1909] James Mercer. *Functions of positive and negative type, and their connection with the theory of integral equations.* Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character, vol. 209, pages 415–446, 1909.

[Minasny 2005] Budiman Minasny et Alex B McBratney. *The Matérn function as a general model for soil variograms.* Geoderma, vol. 128, no. 3, pages 192–207, 2005.

[Morris 1993] Max D Morris, Toby J Mitchell et Donald Ylvisaker. *Bayesian design and analysis of computer experiments: use of derivatives in surface prediction.* Technometrics, vol. 35, no. 3, pages 243–255, 1993.

[Murthy 2014] P Srinivasa Murthy. *Computational Fluid Dynamics: A Design Tool for Aircraft Industries.* In Innovative Design, Analysis and Development Practices in Aerospace and Automotive Engineering, pages 65–66. Springer, 2014.

[Neal 2011] Radford M Neal*et al. MCMC using Hamiltonian dynamics.* Handbook of Markov Chain Monte Carlo, vol. 2, no. 11, 2011.

[Neal 2012] Radford M Neal. Bayesian learning for neural networks, volume 118. Springer Science & Business Media, 2012.

[Ng 2014] Jun Wei Ng et Marc Peter Deisenroth. *Hierarchical Mixture-of-Experts Model for Large-Scale Gaussian Process Regression.* arXiv preprint arXiv:1412.3078, 2014.

[Osborne 2008] Michael A Osborne, Stephen J Roberts, Alex Rogers, Sarvapali D Ramchurn et Nicholas R Jennings. *Towards real-time information processing of sensor network data using computationally efficient multi-output Gaussian processes.* In Proceedings of the 7th international conference on Information processing in sensor networks, pages 109–120. IEEE Computer Society, 2008.

[Osborne 2010] Michael Osborne. Bayesian gaussian processes for sequential prediction, optimisation and quadrature. Oxford University New College, 2010.

[O'Hagan 1998] A O'Hagan. *A Markov property for covariance structures.* Statistics Research Report, vol. 98, page 13, 1998.

[Peeters 2005] Bart Peeters, Herman Van der Auweraer, Steven Pauwels et Jan Debille. *Industrial relevance of Operational Modal Analysis–civil, aerospace and automotive case histories.* In Proceedings of IOMAC, the 1st International Operational Modal Analysis Conference, 2005.

[Quiñonero-Candela 2005] Joaquin Quiñonero-Candela et Carl Edward Rasmussen. *A unifying view of sparse approximate Gaussian process regression.* Journal of Machine Learning Research, vol. 6, no. Dec, pages 1939–1959, 2005.

[Rainieri 2007] Carlo Rainieri, Giovanni Fabbrocino et E Cosenza. *Automated Operational Modal Analysis as structural health monitoring tool: theoretical and applicative aspects.* In Key Engineering Materials, volume 347, pages 479–484. Trans Tech Publ, 2007.

[Rasmussen 2002] Carl Edward Rasmussen et Zoubin Ghahramani. *Infinite mixtures of Gaussian process experts.* Advances in neural information processing systems, vol. 2, pages 881–888, 2002.

[Rasmussen 2005] Carl Edward Rasmussen et Christopher K. I. Williams. Gaussian processes for machine learning (adaptive computation and machine learning). The MIT Press, 2005.

[Raymer 2012] D.P. Raymer. Aircraft design: A conceptual approach. AIAA education series. American Institute of Aeronautics and Astronautics, 2012.

[Richardson 1982] Mark H Richardson et David L Formenti. *Parameter estimation from frequency response measurements using rational fraction polynomials.* In Proceedings of the 1st international modal analysis conference, volume 1, pages 167–186. Union College Schenectady, NY, 1982.

[Romanowski 1996] Michael Romanowski. *Reduced order unsteady aerodynamic and aeroelastic models using Karhunen-Loeve eigenmodes.* In 6th Symposium on Multidisciplinary Analysis and Optimization, page 3981, 1996.

[Rosenbaum 2013] Benjamin Rosenbaum et Volker Schulz. *Efficient response surface methods based on generic surrogate models.* SIAM Journal on Scientific Computing, vol. 35, no. 2, pages B529–B550, 2013.

[Rosenblatt 1958] Frank Rosenblatt. *The perceptron: A probabilistic model for information storage and organization in the brain.* Psychological review, vol. 65, no. 6, page 386, 1958.

[Saatçi 2010] Yunus Saatçi, Ryan D Turner et Carl E Rasmussen. *Gaussian process change point models.* In Proceedings of the 27th International Conference on Machine Learning (ICML-10), pages 927–934, 2010.

[Sacks 1989] Jerome Sacks, William J Welch, Toby J Mitchell et Henry P Wynn. *Design and analysis of computer experiments.* Statistical science, pages 409–423, 1989.

[Särkkä 2011] Simo Särkkä. *Linear operators and stochastic partial differential equations in Gaussian process regression.* Artificial Neural Networks and Machine Learning–ICANN 2011, pages 151–158, 2011.

[Schwab 2016] K. Schwab. The fourth industrial revolution. World Economic Forum, 2016.

[Seeger 2003] Matthias Seeger, Christopher Williams et Neil Lawrence. *Fast forward selection to speed up sparse Gaussian process regression.* In Artificial Intelligence and Statistics 9, 2003.

[Seeger 2005] Matthias Seeger, Yee-Whye Teh et Michael Jordan. *Semiparametric latent factor models.* Rapport technique 161465, EPFL-REPORT, 2005.

[Shah 2014] Amar Shah, Andrew Wilson et Zoubin Ghahramani. *Student-t processes as alternatives to Gaussian processes.* In Artificial Intelligence and Statistics, pages 877–885, 2014.

[Shahdin 2010] Amir Shahdin, Joseph Morlier, Hanno Niemann et Yves Gourinat. *Correlating low energy impact damage with changes in modal parameters: diagnosis tools and FE validation.* Structural Health Monitoring, 2010.

[Shih 1988] CY Shih, YG Tsuei, RJ Allemang et DL Brown. *Complex mode indication function and its applications to spatial domain parameter estimation.* Mechanical systems and signal processing, vol. 2, no. 4, pages 367–377, 1988.

[Snelson 2006] Edward Snelson et Zoubin Ghahramani. *Sparse Gaussian Processes using Pseudo-inputs.* In Advances in Neural Information Processing Systems, pages 1257–1264. MIT press, 2006.

[Solak 2003] E. Solak, R. Murray-smith, W. E. Leithead, D. J. Leith et Carl E. Rasmussen. *Derivative Observations in Gaussian Process Models of Dynamic Systems.* In S. Becker, S. Thrun et K. Obermayer, editeurs, Advances in Neural Information Processing Systems 15, pages 1057–1064. MIT Press, 2003.

[Spitznogle 1970] Frank R Spitznogle et Azizul H Quazi. *Representation and Analysis of Time-Limited Signals Using a Complex Exponential Algorithm.* The Journal of the Acoustical Society of America, vol. 47, no. 5A, pages 1150–1155, 1970.

[Stein 1999] M. L. Stein. Interpolation of Spatial Data: Some Theory for Kriging. Springer, New York, 1999.

[Tan 2003] Bui Thanh Tan. *Proper orthogonal decomposition extensions and their applications in steady aerodynamics.* Master of Engineering in High Performance Computation for Engineered Systems (HPCES), Singapore-MIT Alliance, 2003.

[Titsias 2009] Michalis K. Titsias. *Variational learning of inducing variables in sparse Gaussian processes.* In In Artificial Intelligence and Statistics 12, pages 567–574, 2009.

[Tresp 2000] Volker Tresp. *A Bayesian committee machine.* Neural computation, vol. 12, no. 11, pages 2719–2741, 2000.

[Tripathy 2016] Rohit Tripathy, Ilias Bilionis et Marcial Gonzalez. *Gaussian processes with built-in dimensionality reduction: Applications to high-dimensional uncertainty propagation.* Journal of Computational Physics, vol. 321, pages 191–223, 2016.

[Uhlenbeck 1930] George E Uhlenbeck et Leonard S Ornstein. *On the theory of the Brownian motion.* Physical review, vol. 36, no. 5, page 823, 1930.

[Vassberg 2014] John C Vassberg, Edward N Tinoco, Mori Mani, Ben Rider, Tom Zickuhr, David W Levy, Olaf P Brodersen, Bernhard Eisfeld, Simone Crippa, Richard A Wahls*et al. Summary of the fourth AIAA computational fluid dynamics drag prediction workshop.* Journal of Aircraft, 2014.

[Ver Hoef 1998] Jay M Ver Hoef et Ronald Paul Barry. *Constructing and fitting models for cokriging and multivariable spatial prediction.* Journal of Statistical Planning and Inference, vol. 69, no. 2, pages 275–294, 1998.

[Verveld 2016] Mark Johannes Verveld, Thiemo Kier, Niklas Karcher, Thomas Franz, Mohammad Abu-Zurayk, Matteo Ripepi et Stefan Görtz. *Reduced Order Models for Aerodynamic Applications, Loads and MDO.* Rapport technique, DLR, 2016.

[Villegas García 2013] Marco Antonio Villegas García. An investigation into new kernels for categorical variables. Master's thesis, Universitat Politècnica de Catalunya, 2013.

[Williams 1985] Roger Williams, John Crowley et Havard Vold. *The multivariate mode indicator function in modal analysis.* In International Modal Analysis Conference, pages 66–70, 1985.

[Williams 2001] Christopher KI Williams et Matthias Seeger. *Using the Nyström method to speed up kernel machines.* In Advances in neural information processing systems, pages 682–688, 2001.

[Wilson 2012] Andrew Gordon Wilson. *A Process over all Stationary Covariance Kernels.* Rapport technique, Trinity College, Cambridge, 2012.

[Wilson 2013] Andrew Gordon Wilson et Ryan Prescott Adams. *Gaussian process kernels for pattern discovery and extrapolation.* arXiv preprint arXiv:1302.4245, 2013.

[Wilson 2014] Andrew Gordon Wilson. *Covariance kernels for fast automatic pattern discovery and extrapolation with Gaussian processes*. PhD thesis, University of Cambridge, 2014.

[Wolpert 1997] David H Wolpert et William G Macready. *No free lunch theorems for optimization*. Evolutionary Computation, IEEE Transactions on, vol. 1, no. 1, pages 67–82, 1997.

[Wright 1934] Orville Wright. *How we made the first flight*. Journal of the Franklin Institute, vol. 217, no. 2, pages 239–254, 1934.

Résumé — Dans cette thèse, nous proposons de construire de meilleurs modèles Processus Gaussiens (GPs) en intégrant les connaissances antérieures avec des données expérimentales. En raison du coût élevé de l'exécution d'expériences sur les systèmes physiques, les modèles numériques deviennent un moyen évident de concevoir des systèmes physiques. Traditionnellement, ces modèles ont été construits expérimentalement et itérativement ; une méthode plus rentable de construction de modèles consiste à utiliser des algorithmes d'apprentissage automatique. Nous démontrons comment créer des modèles en intégrant une connaissance antérieure en modifiant les fonctions de covariance. Nous proposons des modèles GP pour différents phénomènes physiques en mécanique des fluides. De même, les lois physiques entre plusieurs sorties peuvent être appliquées en manipulant les fonctions de covariance. Pour chaque application, nous comparons le modèle proposé avec le modèle de l'état de l'art et démontrons les gains de coût ou de performance obtenus.

Mots clés : Processus gaussien, Mécanique de vol, Conception d'aéronefs, Dynamique structurale, Interpolation de choc.

Abstract — In this thesis, we propose to build better Gaussian Process (GP) models by integrating the prior knowledge of Aircraft design with experimental data. Due to the high cost of performing experiments on physical systems, models become an efficient means to designing physical systems. We demonstrate how to create efficient models by incorporating the prior information from engineering design, mainly by changing the covariance functions of the GP. We propose GP models to detect onset of non-linearity, detect modal parameters and interpolate position of shock in aerodynamic experiments. Similarly, physical laws between multiple outputs can be enforced by manipulating the covariance functions, we propose to integrate flight-mechanics to better identify loads using these models. For each application we compare the proposed model with the state-of-the-art model and demonstrate the cost or performance gains achieved.

Keywords: Gaussian Process, Flight-Mechanics, Aircraft design, Structural Dynamics, Shock Interpolation.

Institut Clément Ader, 3 Rue Caroline Aigle
Toulouse, France