NVIDIA DEEP LEARNING INSTITUTE | BROWN

Deep Learning for Science and Engineering Teaching Kit

Deep Learning for Scientists and Engineers

Lecture 7: Machine Learning using Multi-Fidelity Data

Instructors: George Em Karniadakis, Khemraj Shukla, Lu Lu
Teaching Assistants: Vivek Oommen and Aniruddha Bora

Course Roadmap

Module-1 (Basics)

- Lecture 1: Introduction
- Lecture 2: A primer on Python, NumPy, SciPy and *jupyter* notebooks
- Lecture 3: Deep Learning Networks
- Lecture 4: A primer on TensorFlow and PyTorch
- Lecture 5: Training and Optimization
- Lecture 6: Neural Network Architectures

Module-2 (PDEs and Operators)

- Lecture 7: Machine Learning using Multi-Fidelity Data
- Lecture 8: Physics-Informed Neural Networks (PINNs)
- Lecture 9: PINN Extensions
- Lecture 10: Neural Operators

Module-3 (Codes & Scalability)

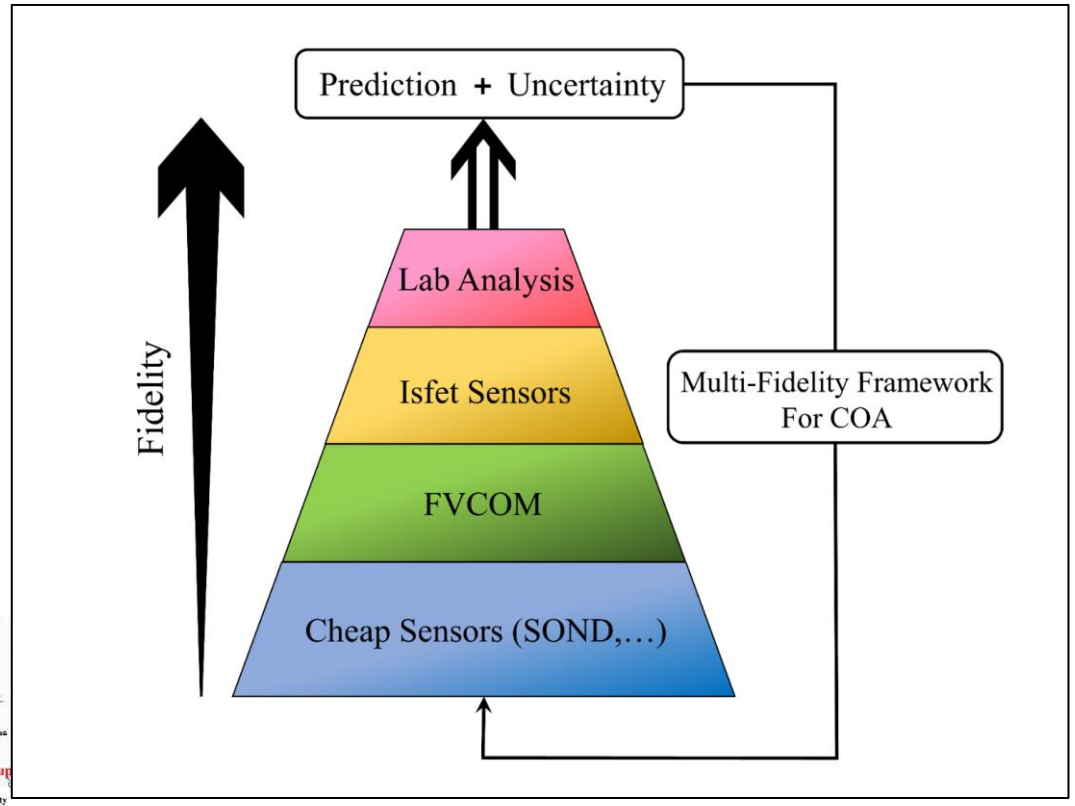
- Lecture 11: Multi-GPU Scientific Machine Learning

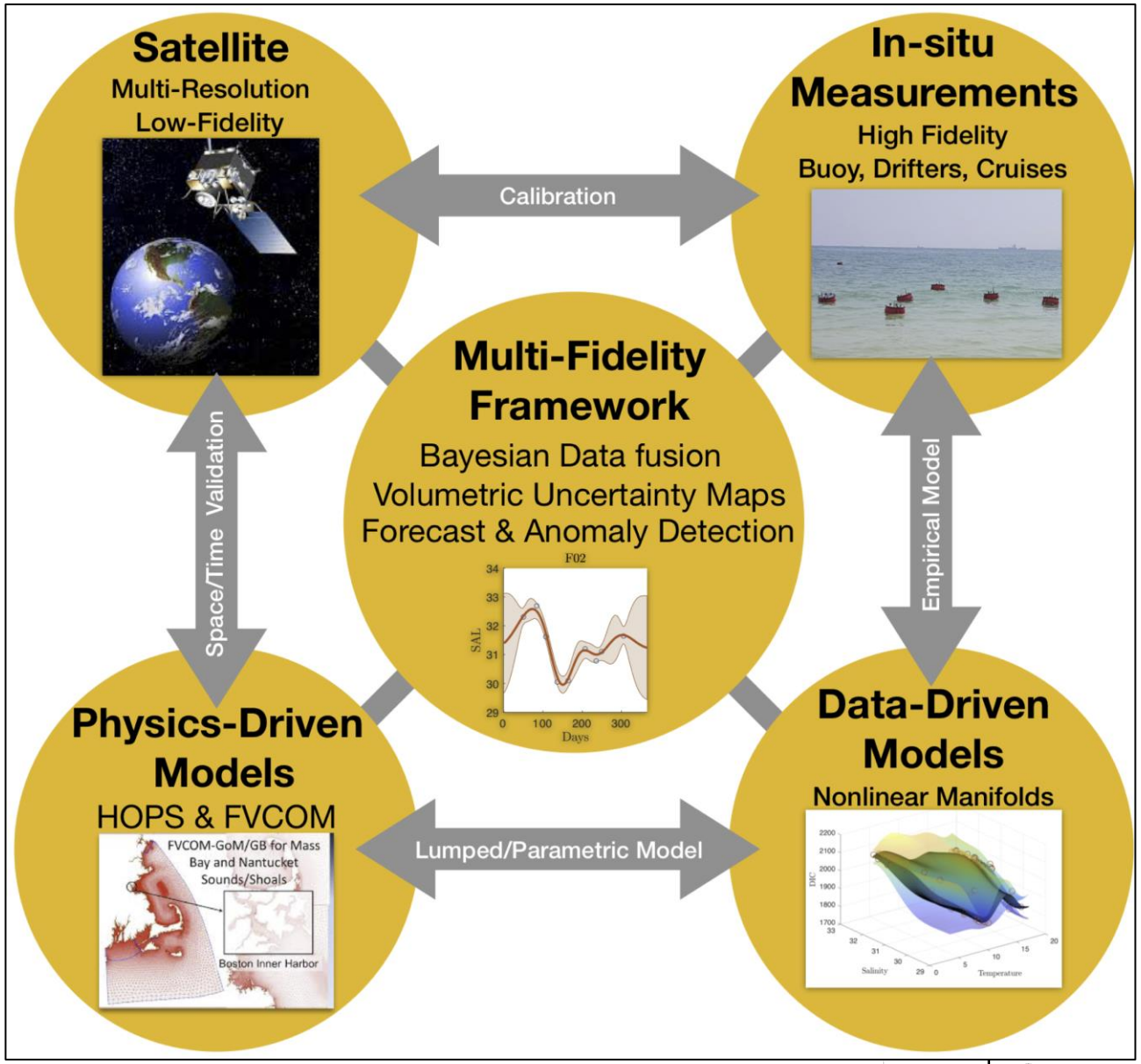
Contents

- ❑ What is multi-fidelity data: Ocean acidification example
- ❑ Gaussian process (GP) regression
- ❑ Data assimilation from noisy measurements: bathymetry and eel grass modeling
- ❑ Physics-informed kernels for GP
- ❑ Multi-fidelity GP/co-Kriging modeling
- ❑ Example: Sea Surface Temperature in Massachusetts Bay
- ❑ Nonlinear multi-fidelity GP and examples
- ❑ Acquisition functions – active learning
- ❑ NN-induced GP kernels
- ❑ Deep multi-fidelity GP
- ❑ Diffusion-manifold driven GP
- ❑ Composite multi-fidelity neural networks
- ❑ Example: Nano-indentation of 3D printed materials
- ❑ Multi-fidelity in modal space
- ❑ Example: vortex-induced vibrations of marine risers

A Motivating Example: Ocean Acidification Monitoring

- ❑ Cost-effective monitoring of OA QoIs.
- ❑ Heterogeneous measurements.
- ❑ Lack of direct measurements of QoIs.
- ❑ Reliance of physics-based models on data due to physics deficiency.



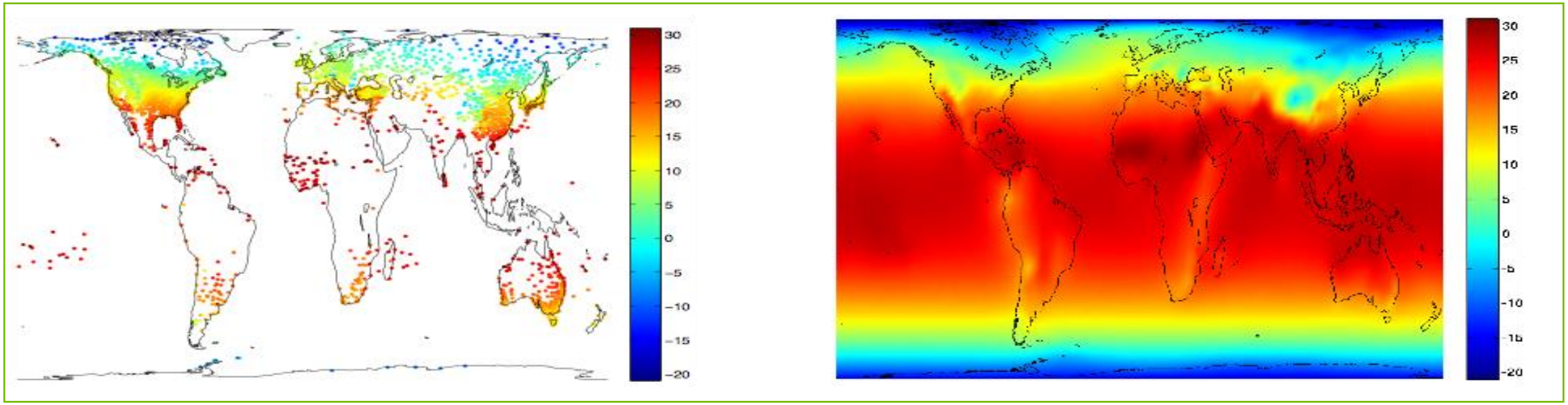


CRUNCH Group
Brown University

NVIDIA DEEP LEARNING INSTITUTE | BROWN

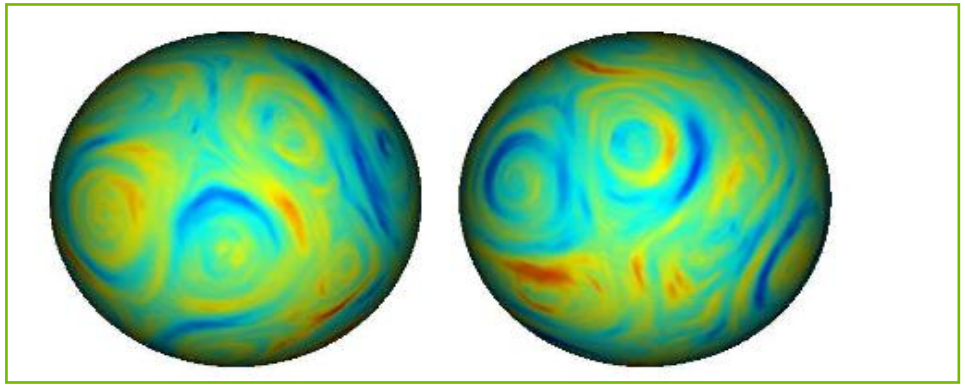
Gaussian-Markov Random Fields: A Stochastic PDE Approach

Example: Reconstruction of global temperature field from data



- GMRF representations of SPDEs can be constructed for oscillating, anisotropic, non-stationary, non-separable, spatio-temporal, and multivariate fields on manifolds.
- Also, the SPDEs can accommodate a wide variety of constraints (e.g., incompressibility, boundary conditions, etc)

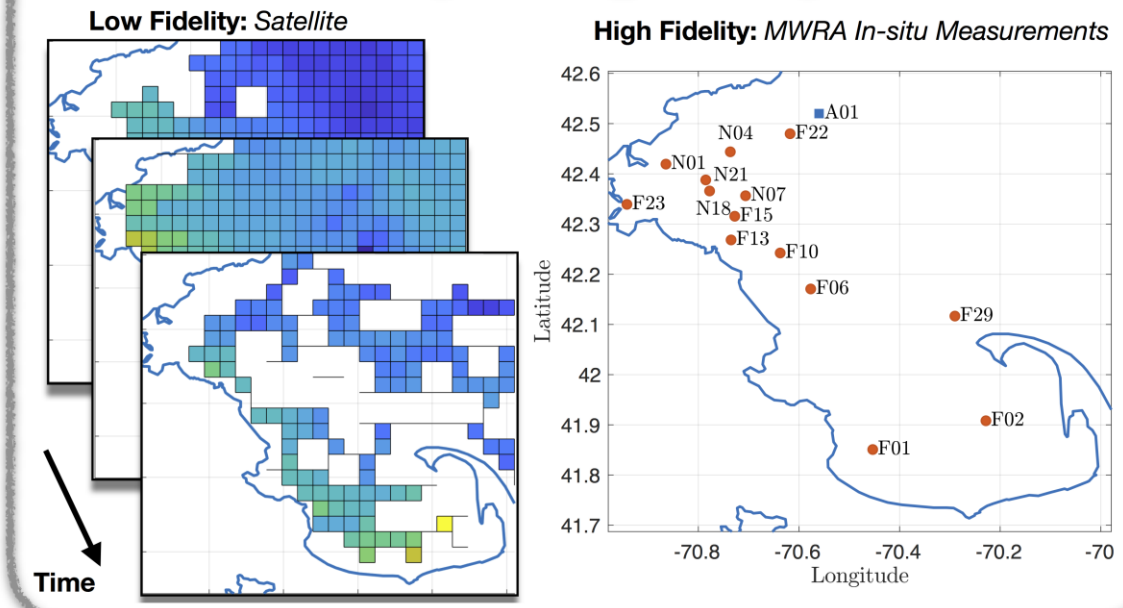
$$\left(\frac{\partial}{\partial t} + \chi_{s,t}^2 + \nabla \cdot m_{s,t} - \nabla \cdot \boldsymbol{M}_{s,t} \nabla\right)(\tau_{s,t}\chi(s,t)) = \epsilon(\boldsymbol{s},t), \qquad (\boldsymbol{s},t) \in \Omega \times \mathbb{R}$$



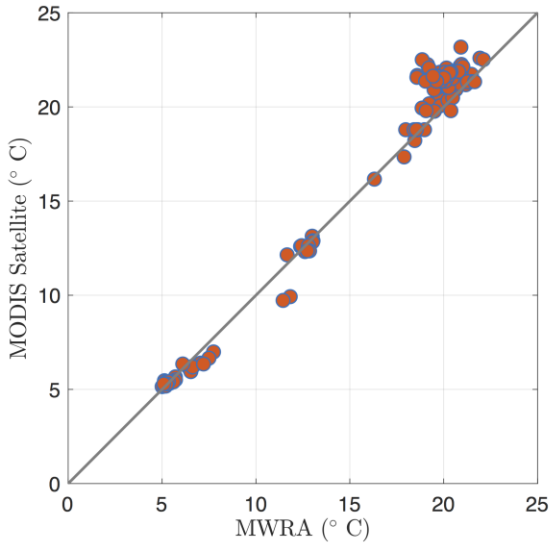
Lindgren F, Rue H, Lindström J. An explicit link between Gaussian fields and Gaussian Markov random fields: the stochastic partial differential equation approach. Journal of the Royal Statistical Society: Series B (Statistical Methodology). 2011 Sep;73(4):423-98.

Example: Monitoring Sea Surface Temperature (SST) in Massachusetts Bay

Low-fidelity and High-fidelity Data



Correlation
Low-Fidelity vs. High-Fidelity



Auto-regressive Gaussian Processes

$$y_L(x) = u_L(x) + \epsilon_L \quad \text{and} \quad y_H(x) = \rho u_L(x) + \delta(x) + \epsilon_H$$

$$u_L(x) \sim \mathcal{GP}(0, k(x, x'; \theta_L)) \quad \text{and} \quad \delta(x) \sim \mathcal{GP}(0, k(x, x'; \theta_\delta))$$

$$k(x, x') = \sigma^2 \exp\left(-\sum_{i=1}^{D} (x_i - x_i')^2 / \theta_i^2 \right)$$

Gaussian Processes

Starting point: The multivariate Gaussian Distribution

$$p(\underbrace{f_1, f_2, ..., f_s}_{f_A}, \underbrace{f_{s+1}, ..., f_N}_{f_b}) \sim \mathcal{N}(\mu, K) \qquad \mu = \begin{bmatrix} \mu_A \\ \mu_B \end{bmatrix} \text{ and } K = \begin{bmatrix} K_{AA} & K_{AB} \\ K_{BA} & K_{BB} \end{bmatrix}$$

Generalization: The Gaussian process

$$\mu_\infty = \underbrace{\begin{bmatrix} \mu_f \\ ... \\ ... \end{bmatrix}}_{\text{mean function}} \text{ and } K_\infty = \underbrace{\begin{bmatrix} K_{ff} & ... \\ ... & ... \end{bmatrix}}_{\text{covariance function}} \qquad K_{i,j} = k(x_i, x_j)$$

Priors over functions:

$$f \sim \mathcal{GP}(\mu(x), K(x, x'; \theta))$$

Infinite dimensional model, but finitely many observations: The marginalization property

$$p(f_A, f_B) \sim \mathcal{N}(\mu, K) \text{ Then:}$$

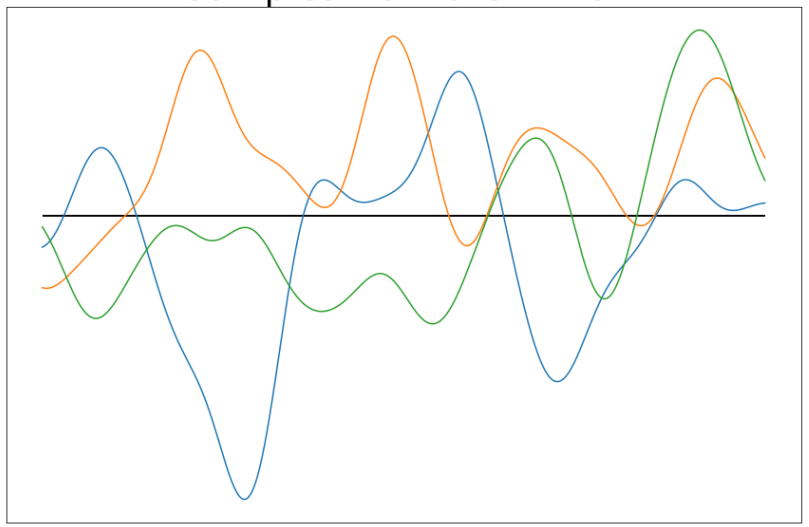
$$p(f_A) = \int_{f_B} p(f_A, f_B) df_B = \mathcal{N}(\mu_A, K_{AA})$$

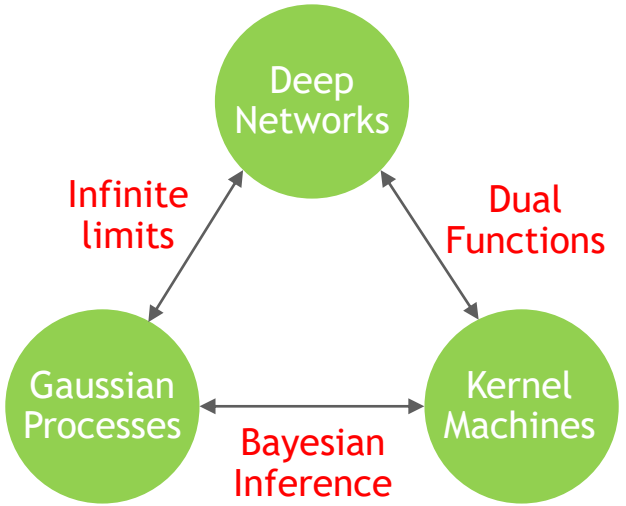
Posterior is also Gaussian:

$$p(f_A, f_B) \sim \mathcal{N}(\mu, K) \text{ Then:}$$

$$p(f_A | f_B) = \mathcal{N}(\mu_A + K_{AB} K_{BB}^{-1}(f_B - \mu_B), K_{AA} - K_{AB} K_{BB}^{-1} K_{BA})$$

Samples from a GP Prior





Kriging/Gaussian Process Regression/BLUP

- We model scattered observations y as a realization of a Gaussian process $Z(\boldsymbol{x})$ (up to zero-mean Gaussian measurement error $\epsilon(\boldsymbol{x})$), according to an observation model

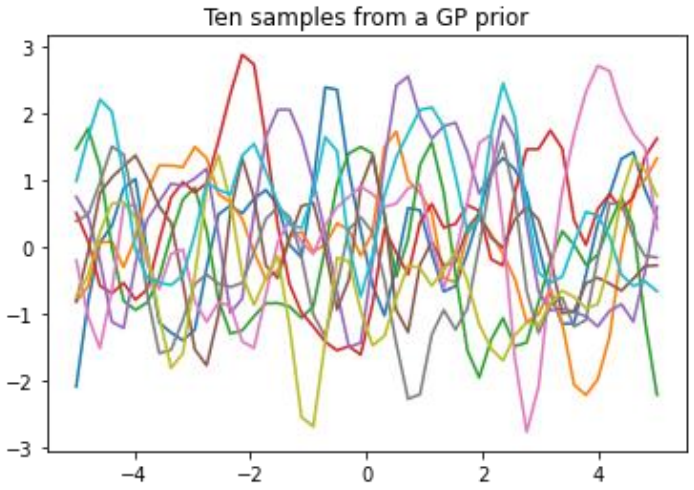
$$Y(\boldsymbol{x}) = Z(\boldsymbol{x}) + \epsilon(\boldsymbol{x})$$

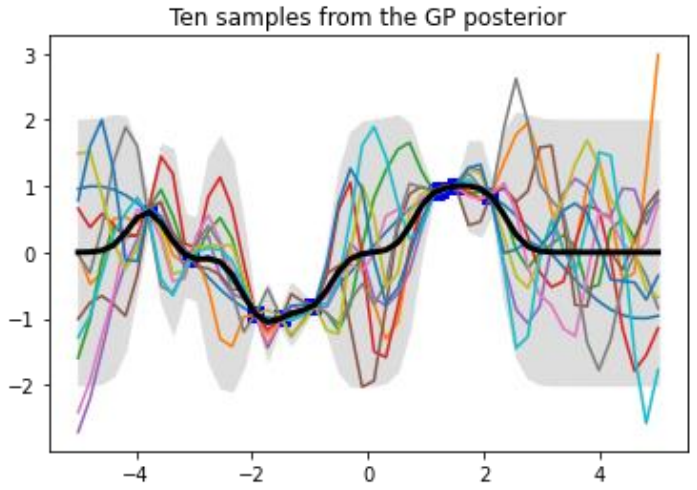
- $[Z(\boldsymbol{x}), Y(\boldsymbol{x})]^\top$ has a multivariate normal distribution,

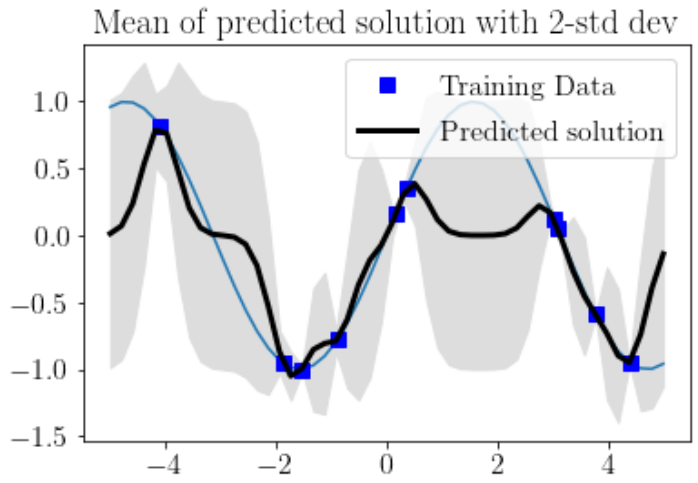
$$\begin{bmatrix} Z(\boldsymbol{x}) \\ Y(\boldsymbol{x}) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu \\ A\mu \end{bmatrix}, \begin{bmatrix} \Sigma & \Sigma A^\top \\ A^\top \Sigma & A\Sigma A^\top + \Sigma_\epsilon \end{bmatrix} \right),$$

  where Σ and Σ_ϵ are parametrized by $\{\theta, \theta_\epsilon\}$, that can be learned from the data y by maximizing the posterior $\pi(\hat{y}|y, \theta, \theta_\epsilon)$.

- Assign a Gaussian prior over functions and caliberate it in view of the data.







C.E. Rasmussen and C.K.L. Williams. Gaussian Process for Machine Learning, 2006

Kriging/Gaussian Process Regression/BLUP

- Once $Z(x)$ has been trained on N observations, we can perform predictions at new points x^* and quantify the variance as

$$\widehat{y}(\boldsymbol{x}^*) = \widehat{\mu} + r^\top (R + \widehat{\sigma}_\epsilon^2 I)^{-1}(y - \mathbf{1}\widehat{\mu})$$

$$s^2(\boldsymbol{x}^*) = \widehat{\sigma}^2 \left[1 - r^\top (R + \widehat{\sigma}_\epsilon^2 \boldsymbol{I})^{-1} r + \frac{[1 - r^\top (R + \widehat{\sigma}_\epsilon^2 \boldsymbol{I})^{-1} r]^2}{\mathbf{1}^\top (R + \widehat{\sigma}_\epsilon^2 \boldsymbol{I})^{-1} \mathbf{1}}\right],$$

  where $R = \kappa(\boldsymbol{x}, \boldsymbol{x}', \widehat{\theta})$ is the $N \times N$ correlation matrix of $Z(x)$, $r = \kappa(\boldsymbol{x}, \boldsymbol{x}^*, \widehat{\theta})$ is a $1 \times N$ correlation vector between the prediction and N training points, and $\mathbf{1}$ is a $1 \times N$ vector of ones. For $\sigma_\epsilon^2 = 0$ the predictor exactly interpolate the training data y, returning zero variance at these locations.

- **Learning**: Given y find the optimal $\{\widehat{\mu}, \widehat{\sigma}^2, \widehat{\sigma}_\epsilon^2, \widehat{\theta}\}$

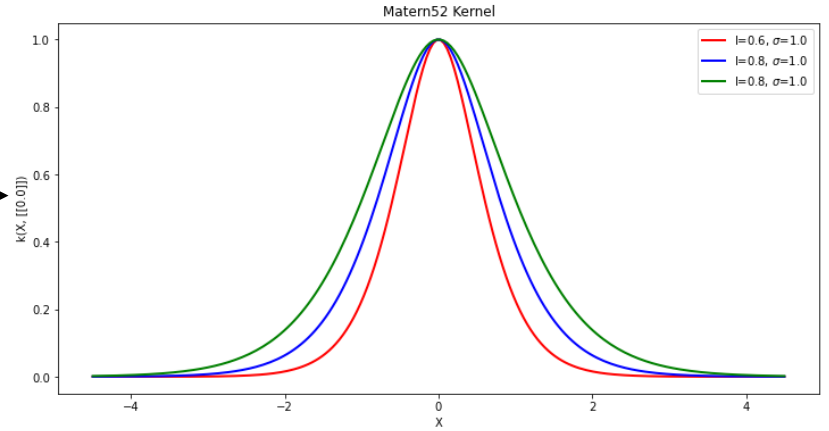
Covariance function	Expression
constant	σ_0^2
linear	$\sum_{d=1}^{D} \sigma_d^2 x_d x'_d$
polynomial	$(\boldsymbol{x} \cdot \boldsymbol{x}' + \sigma_0^2)^p$
squared exponential	$\exp(-\frac{r^2}{2l^2})$
Matérn	$\frac{1}{2^{\nu-1}\Gamma(\nu)} \left(\frac{\sqrt{2\nu}}{l} r\right)^\nu K_\nu \frac{1}{2^{\nu-1}\Gamma(\nu)} \left(\frac{\sqrt{2\nu}}{l} r\right)$
exponential	$\exp(-(\frac{r}{l}))$
$\gamma-$exponential	$\exp(-(\frac{r}{l})^\gamma)$
rational quadratic	$(1 + \frac{r^2}{2\alpha l^2})^{-\alpha}$
neural network	$\sin^{-1}\left(\frac{2\widehat{\boldsymbol{x}}^\top \Sigma \widehat{\boldsymbol{x}'}}{\sqrt{(1+2\widehat{\boldsymbol{x}}^\top \Sigma \widehat{\boldsymbol{x}})(1+2\widehat{\boldsymbol{x}'}^\top \Sigma \widehat{\boldsymbol{x}'})}}\right)$

Kernel and Kernel Parameters

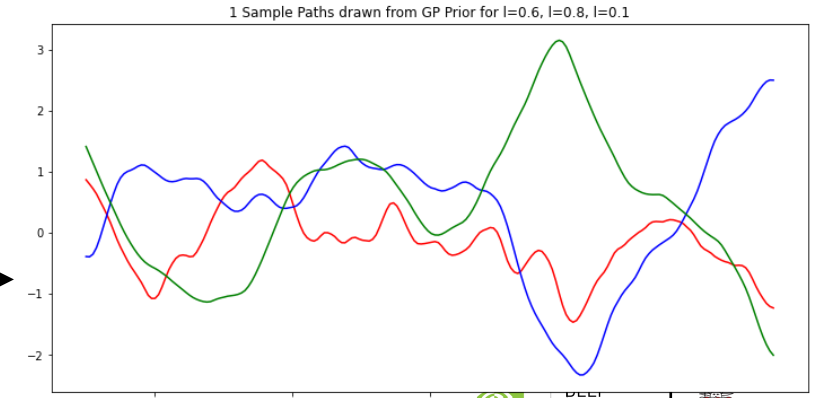
Covariance function	Expression
Matérn	$\frac{1}{2^{\nu-1}\Gamma(\nu)}\left(\frac{\sqrt{2\nu}}{l}r\right)^{\nu} K_{\nu}\frac{1}{2^{\nu-1}\Gamma(\nu)}\left(\frac{\sqrt{2\nu}}{l}r\right)$

```
import GPy
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12, 6)
```

```
# Define Kernel Using GPy
ker1 = GPy.kern.Matern52(input_dim=1, variance = 1.0, lengthscale=0.6)
ker2 = GPy.kern.Matern52(input_dim=1, variance = 1.0, lengthscale=0.8)
ker3 = GPy.kern.Matern52(input_dim=1, variance = 1.0, lengthscale=1.0)

xmin=-5
xmax=5
N=200
X = np.linspace(xmin, xmax, N)[:, None]
mu = np.zeros((N))
C1 = ker1.K(X,X)
C2 = ker2.K(X,X)
C3 = ker3.K(X,X)
fig, ax = plt.subplots()
ker1.plot(ax=ax, linewidth=2, color="r", label="l=0.6, $\sigma$=1.0")
ker2.plot(ax=ax, linewidth=2, color="b", label="l=0.8, $\sigma$=1.0")
ker3.plot(ax=ax, linewidth=2, color="g", label="l=0.8, $\sigma$=1.0")
ax.legend()
ax.set_title("Matern52 Kernel")
```

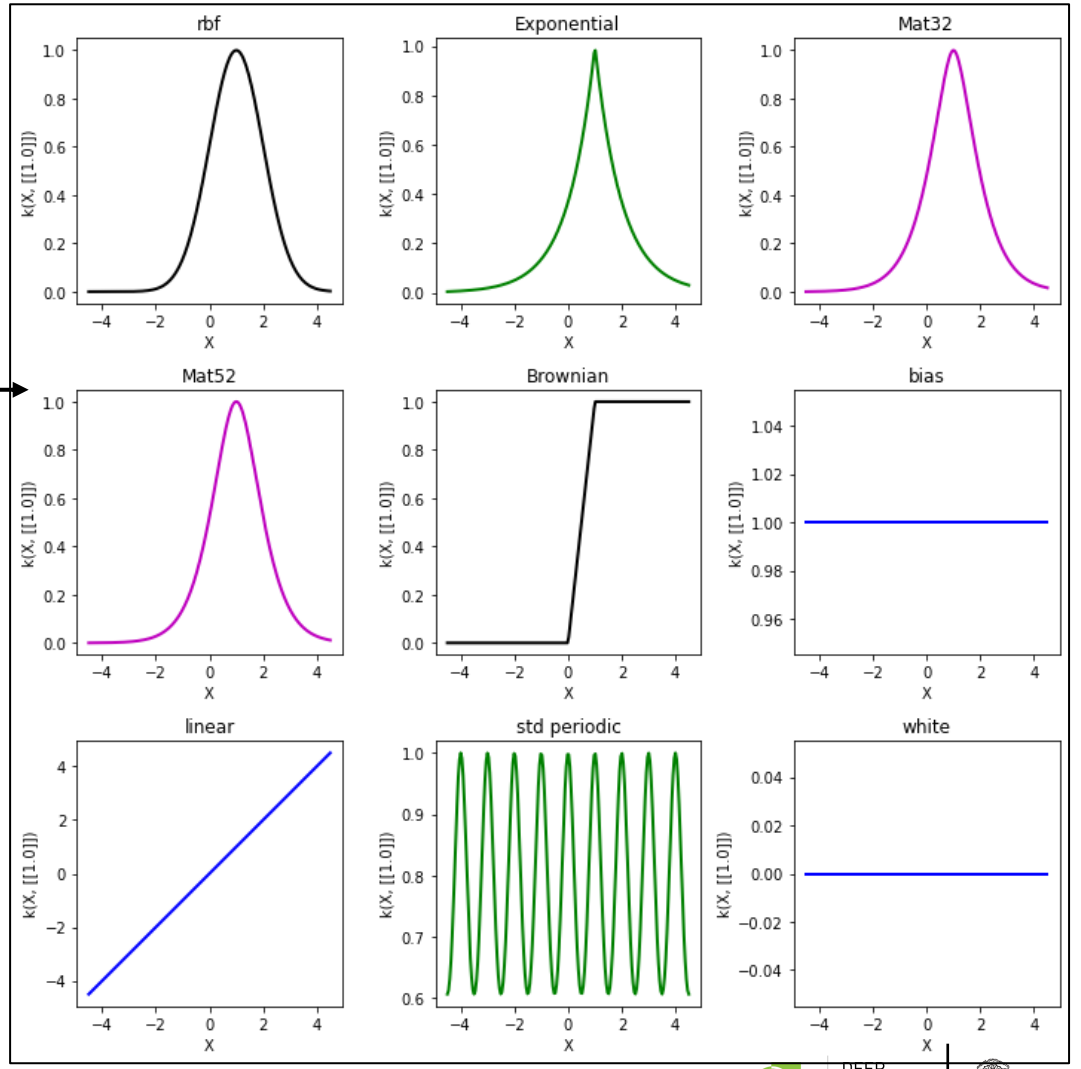



```
Z1 = np.random.multivariate_normal(mu,C1,1)
Z2 = np.random.multivariate_normal(mu,C2,1)
Z3 = np.random.multivariate_normal(mu,C3,1)
fig, ax = plt.subplots()
plt.plot(X[:],Z1[0,:], "-r")
plt.plot(X[:],Z2[0,:], "-b")
plt.plot(X[:],Z3[0,:], "-g")

ax.set_title("1 Sample Paths drawn from GP Prior for l=0.6, l=0.8, l=0.1")
plt.show()
```



Kernels for Gaussian Process

```
from random import randrange
import GPy
import matplotlib.pyplot as plt
import numpy as np

figure, axes = plt.subplots(3,3, figsize=(10,10), tight_layout=True)
kerns = [GPy.kern.RBF(1), GPy.kern.Exponential(1), GPy.kern.Matern32(1), \
         GPy.kern.Matern52(1), GPy.kern.Brownian(1), GPy.kern.Bias(1), \
         GPy.kern.Linear(1), GPy.kern.StdPeriodic(1), GPy.kern.White(1)]

col = ['r', 'b', 'g', 'b', 'm', 'k']

for k,a in zip(kerns, axes.flatten()):
    j = np.random.randint(0, high=6, size=1, dtype=int)
    k.plot(ax=a, x=1, color=col[j[0]])
    a.set_title(k.name.replace('_', ' '))
```



Gaussian Process Regression: Implementation

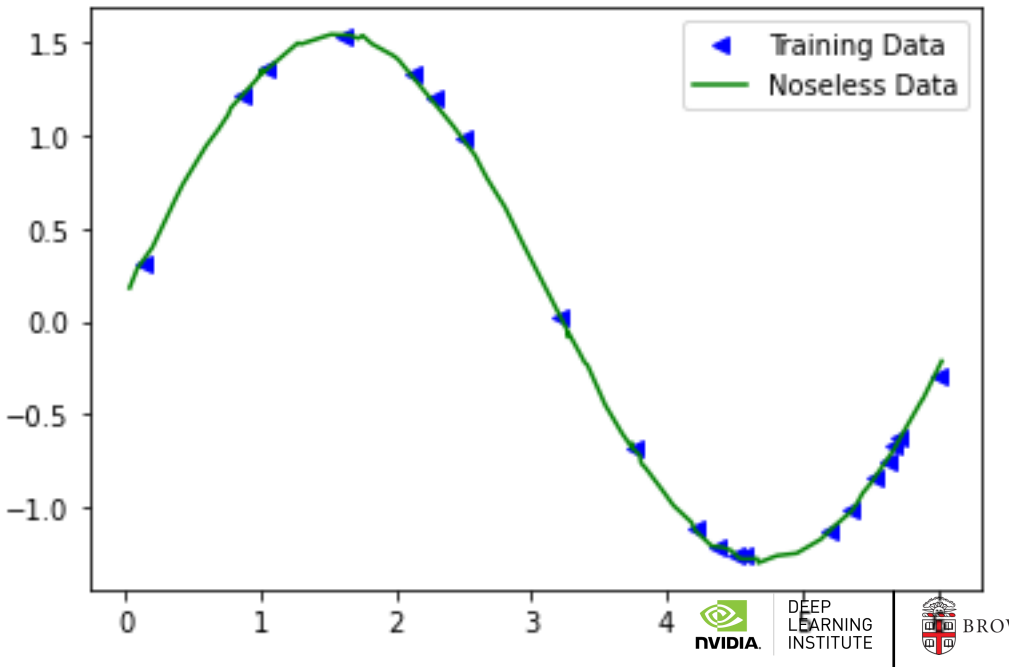
1. Import modules and Helper Routine for Plot

```
import GPy
import numpy as np
import matplotlib.pyplot as plt
```

2. Generate and Normalize Training and Testing Data

```
N=20 # Number of training points
Nstar = 70 # number of test points

X = np.sort(np.random.rand(N+Nstar, 1)*6.14, axis=0)
Y = np.sin(X) + np.random.randn(*X.shape) * 0.008

perm = np.random.permutation(X.shape[0])
x_tr = X[np.sort(perm[0:N],axis=0),:]
Y_tr = Y[np.sort(perm[0:N],axis=0),]
x_star = X[np.sort(perm[N:N+Nstar],axis=0),:]
Y_star = Y[np.sort(perm[N:N+Nstar],axis=0),:]



### Normalize Data
Ymean = Y.mean()
Ystd = Y.std()
Y_tr-=Ymean
Y_tr/=Ystd


Y_star -= Ymean
Y_star /= Ystd

# plot data
plt.plot(x_tr,Y_tr, 'b<',label='Training Data')
plt.plot(x_star,Y_star, '-g',label='Noseless Data')
plt.legend()
plt.show()
```

Training data



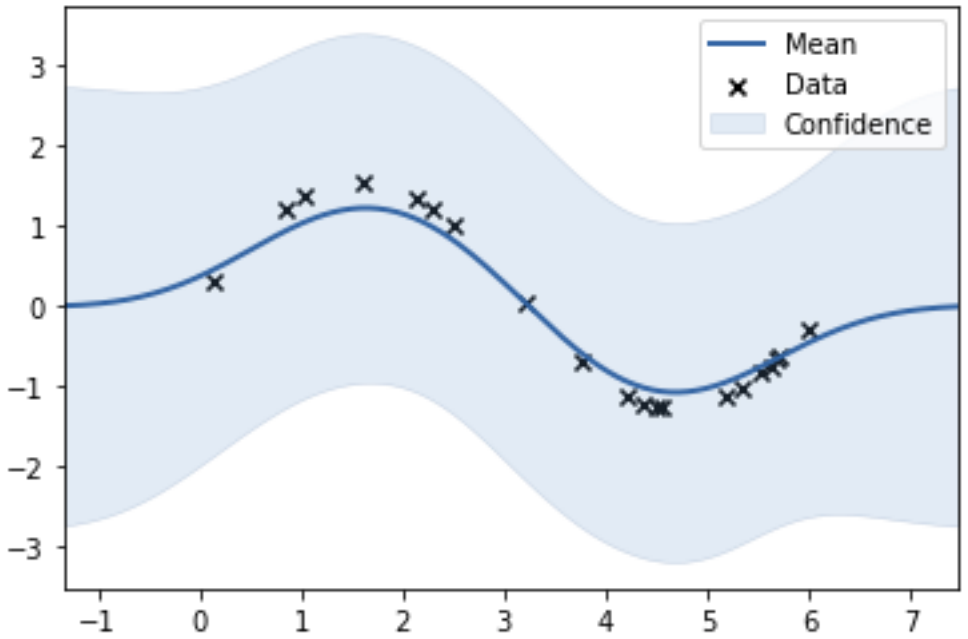
Gaussian Process Regression: Implementation

3. Define Kernel and optimize GP model for model parameters

```
dim_1 = x_tr.shape[1]
# Define RBF Kernel also known as Squared Exponential kernel
k = GPy.kern.RBF(dim_1)

#Define Gaussian Process Regression Model
m = GPy.models.GPRegression(X=x_tr, Y=Y_tr, kernel=k)
print(m)
```

```
Name : GP regression
Objective : 23.812870975912027
Number of Parameters : 3
Number of Optimization Parameters : 3
Updates : True
Parameters:
  GP regression.          |  value  |  constraints  |
  rbf.variance            |    1.0  |      +ve      |
  rbf.lengthscale         |    1.0  |      +ve      |
  Gaussian_noise.variance |    1.0  |      +ve      |
```

Prior



RBF kernel:

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

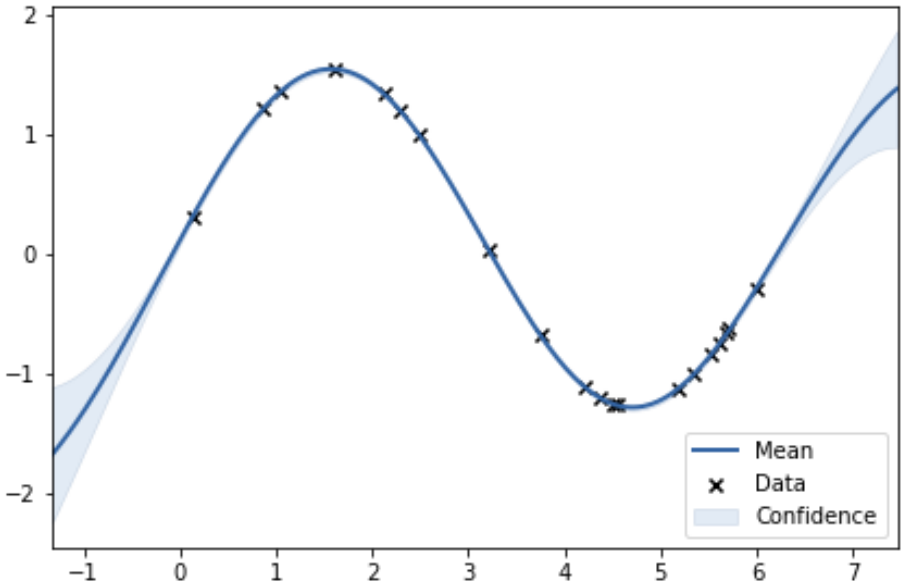
- Gaussian Noise variance acts like a regularization in GP models.
- Larger values of noise variance lead to more smoother model.

Gaussian Process Regression: Implementation

4. Optimize

```
m.optimize(messages=1)
m.plot()
```

Posterior




```
onstraints   |  priors
   +ve       |
   +ve       |
   +ve       |
```

Model: GP regression

Objective: -34.58282342238019

Number of Parameters: 3

Number of Optimization Parameters: 3

Updates: True

GP_regression.	value	constraints	priors
rbf.variance	4.758688562299606	+ve	
rbf.lengthscale	2.385712681113764	+ve	
Gaussian_noise.variance	0.00010550018745290958	+ve	

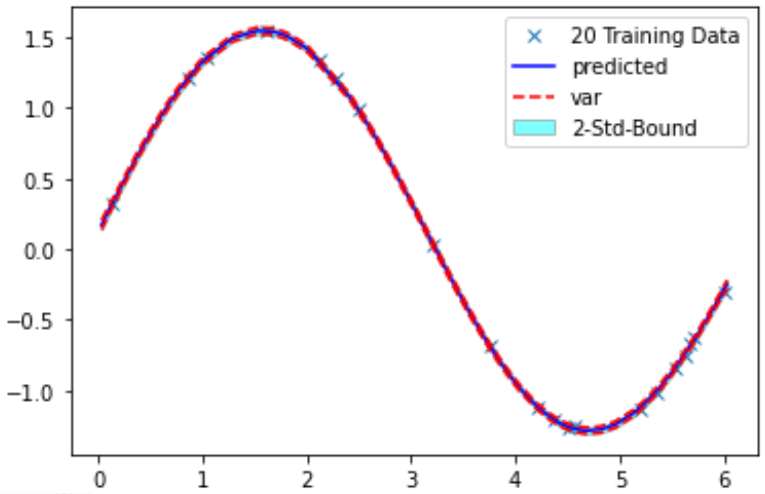
Gaussian Process Regression: Implementation

5. Prediction

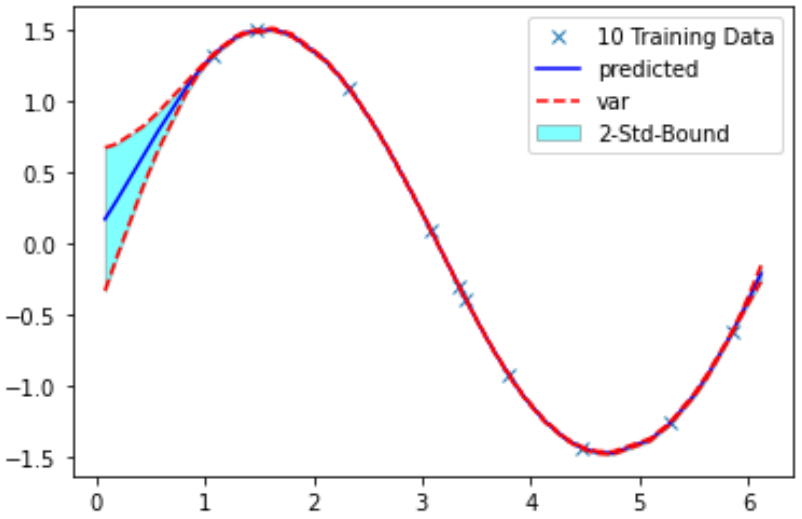
GPy

```
mu,var = m.predict(x_star)
plt.plot(x_tr, Y_tr, "x", label="Training Data")
plt.plot(x_star, mu, '-b',label='predicted')
plt.plot(x_star, mu+2*np.sqrt(var), 'r--',label='var')
plt.plot(x_star, mu-2*np.sqrt(var), 'r--')
plt.legend()
vv = 2*np.sqrt(var)
plt.fill_between(x_star[:,0], (mu-vv)[:,0], (mu+vv)[:,0], alpha=0.5, edgecolor='gray', facecolor='cyan')
plt.legend()
plt.ylim((-2.5, 2.5))
plt.show()
```

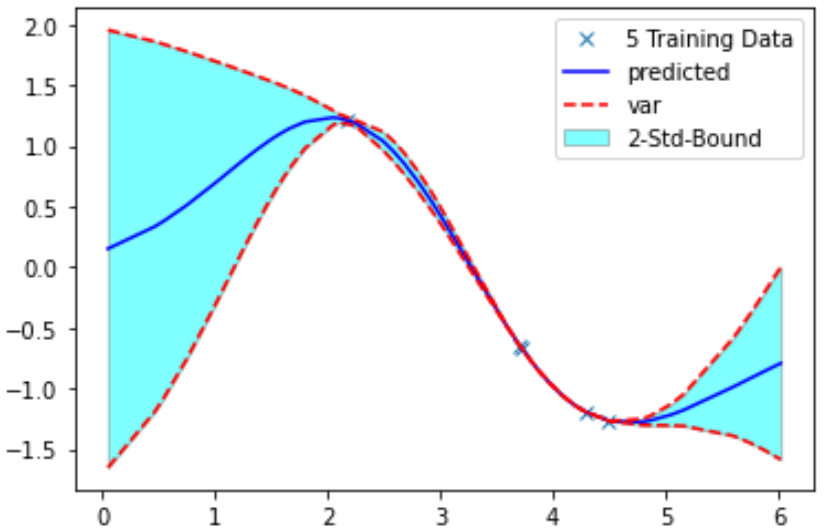
20 Training Datapoints



10 Training Datapoints



5 Training Datapoints



Data Assimilation from Noisy Field Measurements

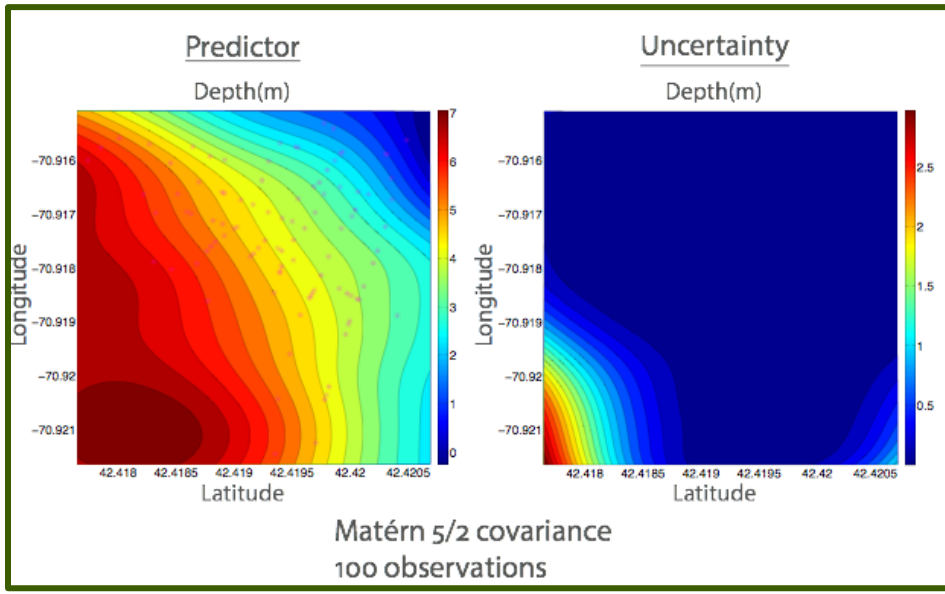
- Pirate's Cove data-set: 1,328 noisy measurements of bathymetry and plant height variables

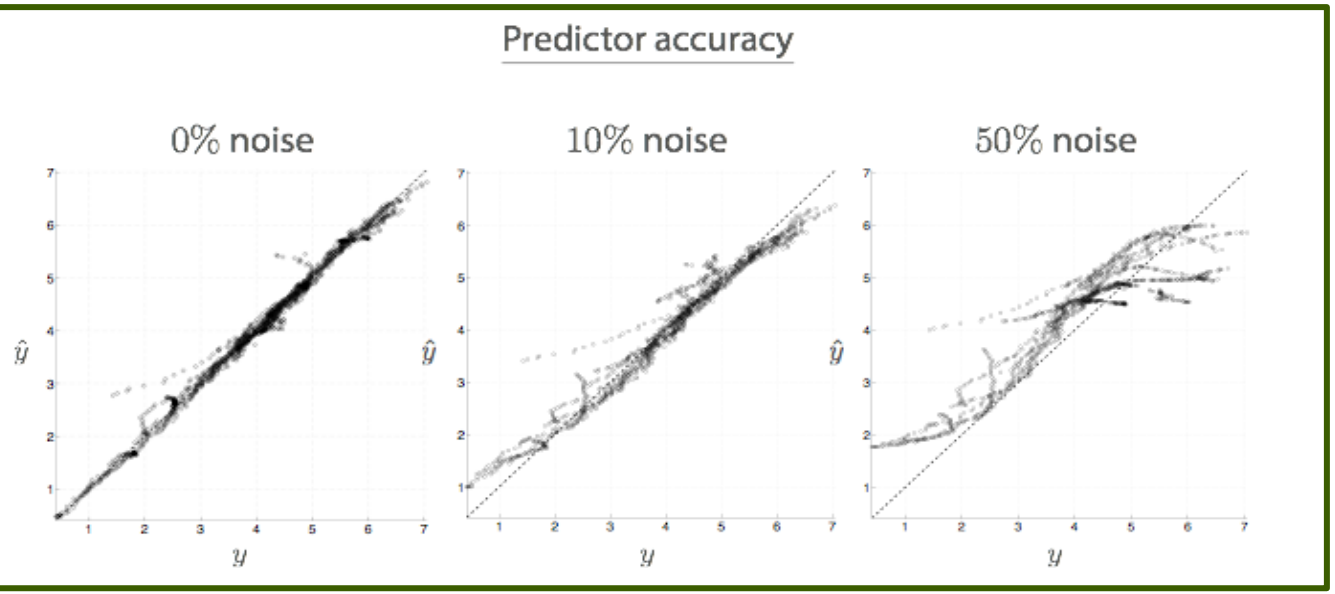
- **Measurements:**
  Mike Sacarny (MIT Sea Grant)

<u>Objective:</u> Reconstruct the bathymetry and plant height maps with quantified uncertainty

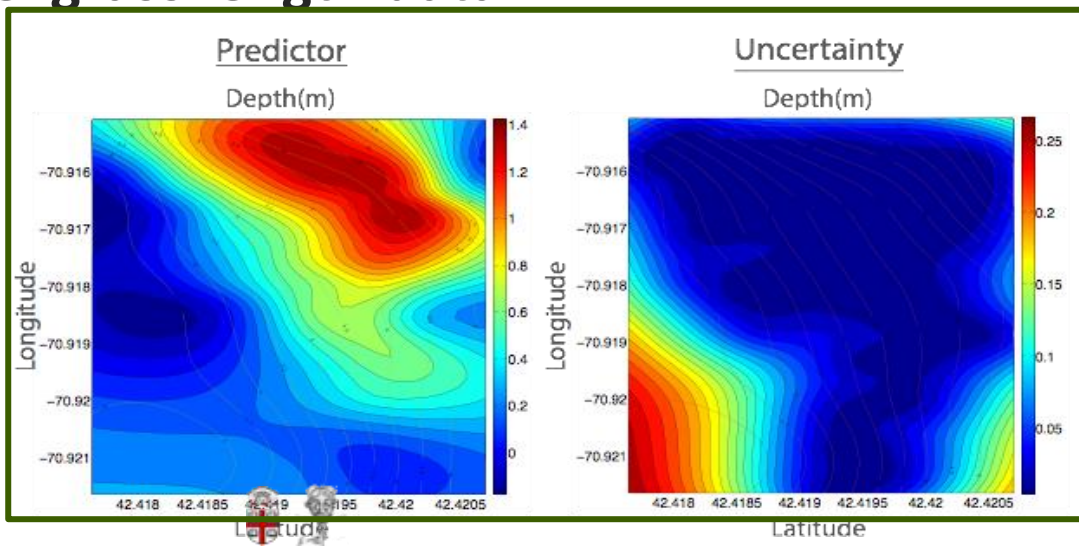
Data Assimilation from Noisy Field Measurements

Bathymetry data

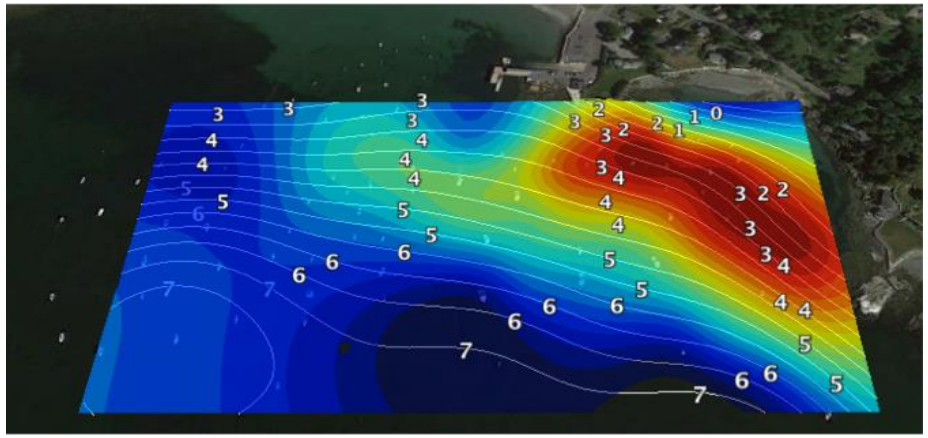




Eel grass length data

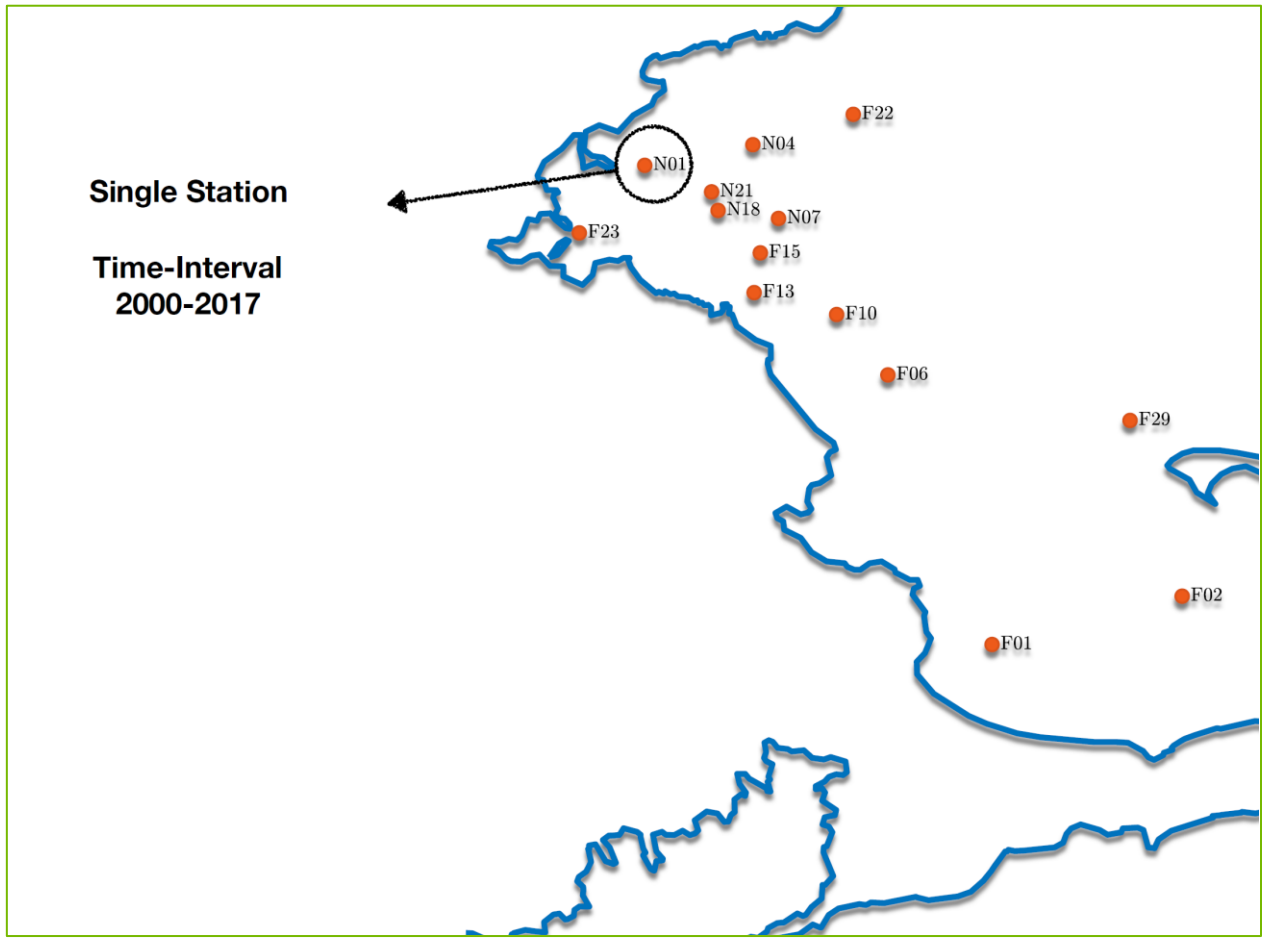


Predictions using 100 training points

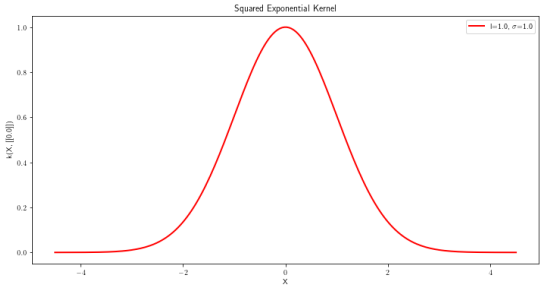


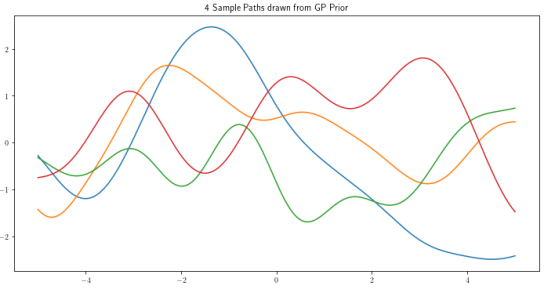
Physics-Informed Kernels

MWRA Forecast



Seasonal Dynamics

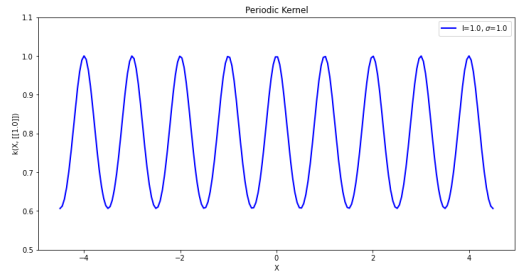


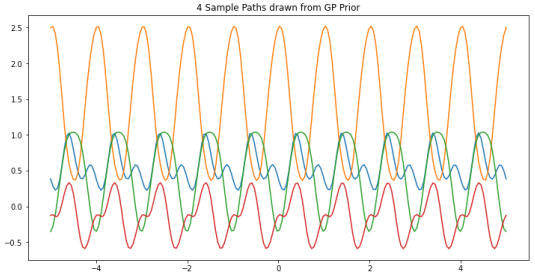


Squared Exponential Kernel:

$$k_{se}(\boldsymbol{x}, \boldsymbol{x}') = \sigma^2 \exp\left(-\frac{(\boldsymbol{x} - x')^2}{2l^2}\right)$$

Hyperparameters: σ and l



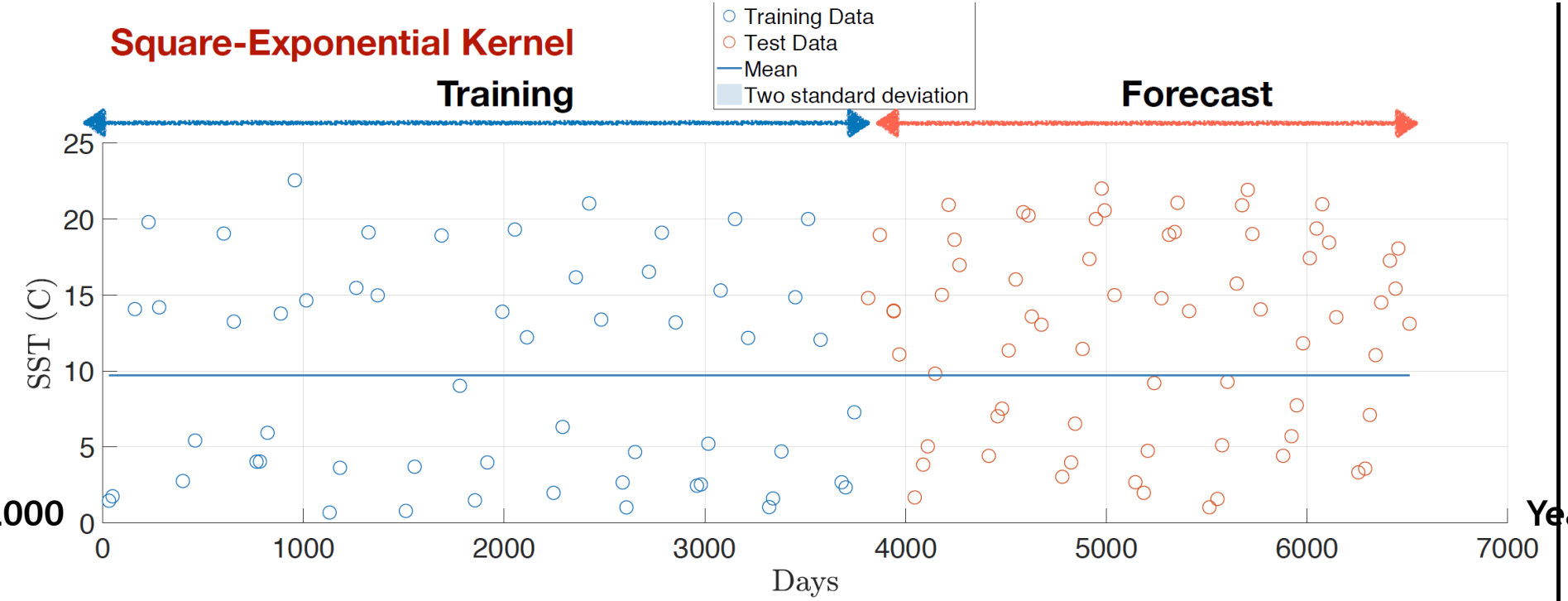


Periodic Kernel:

$$k_{per}(\boldsymbol{x}, \boldsymbol{x}') = \sigma^2 \exp\left(-\frac{2\sin^2(\pi|\boldsymbol{x} - \boldsymbol{x}'|/p)}{l^2}\right)$$

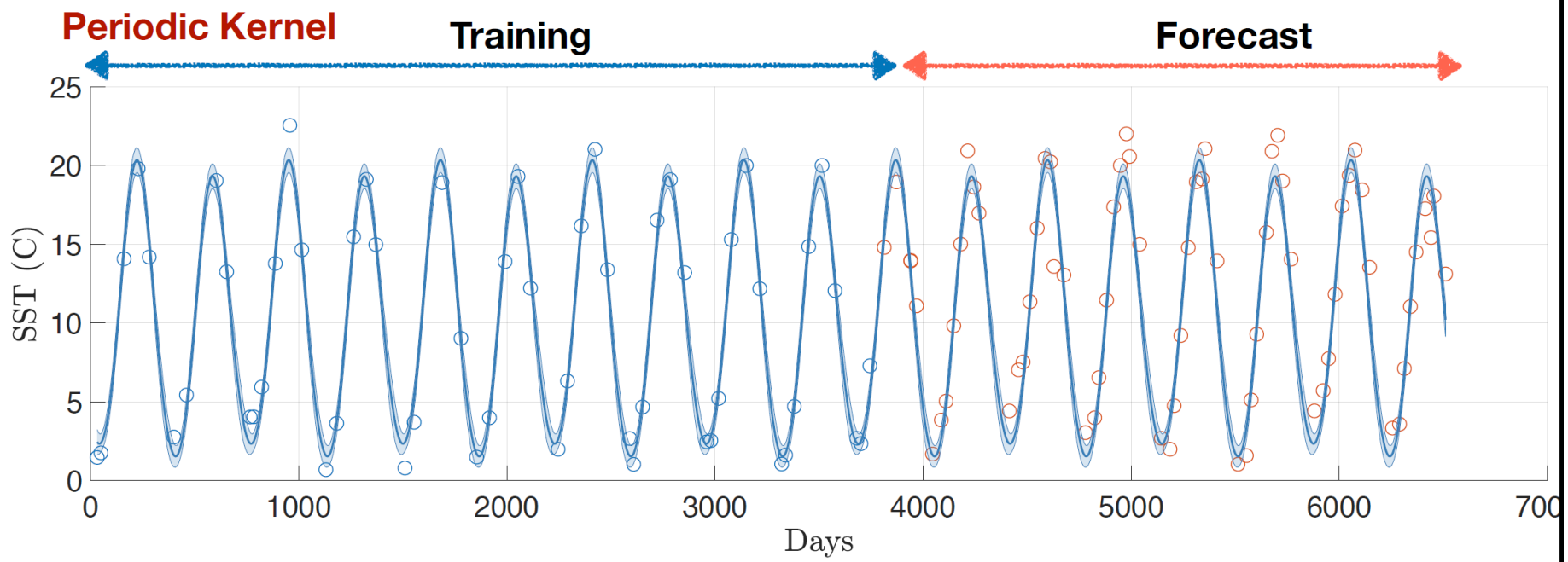
Hyperparameters: σ, l, and p

Square-Exponential Kernel



Year 2000 — Year 2007

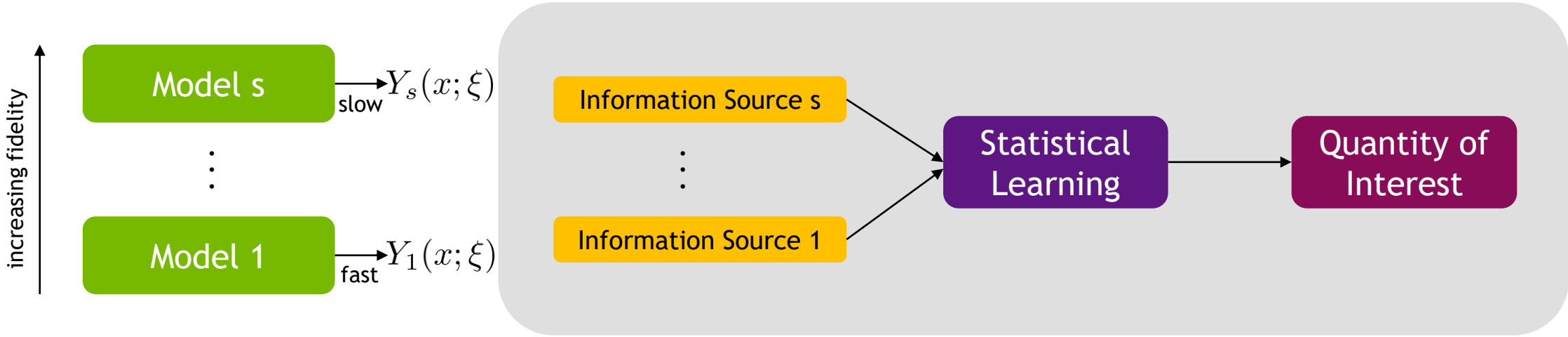
Periodic Kernel



DEEP LEARNING INSTITUTE

BROWN

Multi-Fidelity Modeling



- Number of runs is limited by time and computational resources
- We cannot compute at all $(x;\xi)$
- Prediction of $Z_i(x) = \mathbb{E}[f(Y_i(x;\xi))]$ is a problem of statistical inference

Multi-Fidelity Modeling with GPs

Auto-regressive model:

$$f_t(x) = \rho_{t-1}(x) f_{t-1}(x) + \delta_t(x) \qquad t = 1, ..., s$$

Theorem (LeGratiet, 2014):

The predictive posterior of a recursive scheme has exactly the same distribution with the fully coupled model given a nested experimental design.

Predictive Posterior

$$\begin{aligned} p(f_*|y, X, x_*) &= \mathcal{N}(f_*|\mu_*, \sigma_*^2) \\ \mu_*(x_*) &= k_{*N}(\boldsymbol{K} + \sigma_\epsilon^2 I)^{-1} y \\ \sigma_*^2(x_*) &= k_{**} - k_{*N}(\boldsymbol{K} + \sigma_\epsilon^2 I)^{-1} k_{N*} \end{aligned}$$

$\boldsymbol{K}$ becomes a block covariance matrix

$$\boldsymbol{K} = \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix}$$

Key Idea: Replace f_{t-1} with GP posterior of the previous level $\tilde{f}_{t-1}$

$$f_t(x) = \rho_{t-1}(x) f_{t-1}(x) + \delta_t(x) \qquad \tilde{f}_{t-1} \sim f_{t-1}|\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_{t-1}$$

- Le Gratiet L, Garnier J. Recursive co-kriging model for design of computer experiments with multiple levels of fidelity. International Journal for Uncertainty Quantification. 2014;4(5).
- Kennedy MC, O'Hagan A. Predicting the output from a complex computer code when fast approximations are available. Biometrika. 2000 Mar 1;87(1):1-3.

Information Fusion via Recursive Co-Kriging

- We have s levels of information sources producing outputs $y_t(x_t)$, at locations $\boldsymbol{x}_t \in D_t \subseteq \mathbb{R}^d$, sorted by increasing order of fidelity and modeled by Gaussian processes $Z_t(\boldsymbol{x}),\ t = 1, \ldots, s$.

$$Z_t(\boldsymbol{x}) = \rho_{t-1}(\boldsymbol{x}) Z_{t-1}(\boldsymbol{x}) + \delta_t(\boldsymbol{x}), \qquad t = 2, \ldots s,$$

  where $\delta_t(x)$ is a Gaussian field independent $\{Z_{t-1}, \ldots, Z_1\}$ and distributed as $\delta_t \sim \mathcal{N}(\mu_{\delta_t}, \sigma_t^2 R_t)$.

- $\rho(\boldsymbol{x})$ is a scaling factor that quantifies the correlation between $\{Z_t(\boldsymbol{x}), Z_{t-1}(\boldsymbol{x})\}$.

- Once $Z_t(\boldsymbol{x})$ has been trained on N_t observation, we can perform predictions at new points $\boldsymbol{x}_t^*$ and quantify the variance as

$$\widehat{y}_t(\boldsymbol{x}_t^*) = \widehat{\mu}_t + \widehat{\rho}_{t-1}\widehat{y}_{t-1}(\boldsymbol{x}_t^*) + r_t^\top (R_t + \widehat{\sigma}_{\epsilon_t}^2 \boldsymbol{I})^{-1}[y_t(\boldsymbol{x}_t) - \mathbf{1}\widehat{\mu}_t - \widehat{\rho}_{t-1}\widehat{y}_{t-1}\boldsymbol{x}_t)],$$

$$s_t^2(\boldsymbol{x}_t^*) = \widehat{\rho}_{t-1}^2 s_{t-1}^2(\boldsymbol{x}_t^*) + \widehat{\sigma}_t^2 \left[1 - r_t^T (R_t + \widehat{\sigma}_{\epsilon_t}^2)^{-1} r_t + \frac{[1 - r_t^\top (R_t + \widehat{\sigma}_{\epsilon_t}^2 \boldsymbol{I})^{-1} r_t]^2}{\mathbf{1}_t^T (R_t + \widehat{\sigma}_{\epsilon_t}^2 I)^{-1} \mathbf{1}_t}\right]$$

- $R_t = \kappa_t(\boldsymbol{x}_t, \boldsymbol{x}_t'; \widehat{\theta}_t)$ is the $N_t \times N_t$ correlation matrix of $Z_t(\boldsymbol{x}), r_t = \kappa_t(\boldsymbol{x}_t, \boldsymbol{x}_t^*; \widehat{\theta}_t)$ is a $1 \times N_t$ correlation vector between the prediction and N_t training points, and $\mathbf{1}_t$ is a $1 \times N_t$ vector of ones.

Learning: Given y_t find the optimal $\{\widehat{\mu}_t, \widehat{\sigma}_t^2, \widehat{\sigma}_{\epsilon_t}^2, \widehat{\theta}_t, \widehat{\rho}_{t-1}\}$

The object of inference is $[Z_s(\boldsymbol{x}) | \boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_{s-1}]$
(Output of the high-fidelity model conditional on all model data)

Multi-fidelity GPR Implementation

- **Problem Setup**:
  High Fidelity Function: $f_h(x) = (6x - 2.0)^2 \sin(12x - 4)$
  Low Fidelity Function: $f_l(x) = \frac{1}{2} f_h(x) + 10\left(x - \frac{1}{2}\right) - 5$

- We model the low-fidelity function by $f_l(x) = u_1(x)$ and high fidelity function by $f_h = \rho u_1(x) + u_2(x)$, where

$$u_1(x) \sim \mathcal{GP}(0, k_1(x, x'; \theta_1)),$$

  and

$$u_2(x) \sim \mathcal{GP}(0, k_2(x, x'; \theta_2)),$$

  are Gaussian Processes.

- This will result into following multi-output Gaussian process

$$\begin{bmatrix} f_l(x) \\ f_h(x) \end{bmatrix} \sim \mathcal{GP}\left(0, \begin{bmatrix} k_{ll}(x, x'; \theta_1) & k_{lh}(x, x'; \theta_1, \rho) \\ k_{hl}(x, x'; \theta_1, \rho) & k_{hh}(x, x'; \theta_1, \theta_2, \rho), \end{bmatrix}\right)$$

  where

$$\begin{aligned} k_{ll}(x, x'; \theta_1) &= k_1(x, x'; \theta_1) \\ k_{hl}(x, x'; \theta_1, \rho) &= k_{lh}(x, x'; \theta_1, \rho) = \rho k_1(x, x'; \theta_1) \\ k_{hh}(x, x'; \theta_1, \theta_2, \rho) &= \rho^2 k_1(x, x'; \theta_1) + k_2(x, x'; \theta_2) \end{aligned}$$

Multi-fidelity GPR Implementation

- **Training**: Training data- LF= $\{\boldsymbol{x}_l, \boldsymbol{y}_l\}$, HF= $\{\boldsymbol{x}_h, \boldsymbol{y}_h\}$
- We assume that

$$\boldsymbol{y}_l = f_l(\boldsymbol{x_l}) + \epsilon_l, \qquad \epsilon_l \sim \mathcal{N}(\mathbf{0}, \sigma_l^2 \boldsymbol{I})$$

and

$$\boldsymbol{y}_h = f_h(\boldsymbol{x_h}) + \epsilon_h, \qquad \epsilon_h \sim \mathcal{N}(\mathbf{0}, \sigma_h^2 \boldsymbol{I})$$

- We obtain

$$\boldsymbol{y} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{K}),$$

where

$$y = \begin{bmatrix} \boldsymbol{y}_l \\ \boldsymbol{y}_h \end{bmatrix}$$

and

$$K = \begin{bmatrix} k_{ll}(\boldsymbol{x}_l, \boldsymbol{x}_l; \theta_1) + \sigma_l^2 \boldsymbol{I} & k_{lh}(\boldsymbol{x}_l, \boldsymbol{x}_h; \theta_1, \rho) \\ k_{hl}(\boldsymbol{x}_h, \boldsymbol{x}_l; \theta_1) & k_{hh}(\boldsymbol{x}_h, \boldsymbol{x}_h; \theta_1, \theta_2, \rho) + \sigma_h^2 \boldsymbol{I} \end{bmatrix}$$

- The hyper-parameters θ_1, θ_2, ρ and the noise variance parameters σ_l^2, σ_h^2 can be trained by minimizing the resulting negative log marginal likelihood.

$$\mathcal{NLML} = -\frac{1}{2}\boldsymbol{y}^\top \boldsymbol{K}^{-1}\boldsymbol{y} - \frac{1}{2}\log|\boldsymbol{K}| - \frac{N}{2}\log(2\pi)$$

Multi-fidelity GPR Implementation

- **Prediction**: Prediction at test point $\boldsymbol{x}^*$ can be made by first writing the distribution

$$\begin{bmatrix} f_h(x^*) \\ \boldsymbol{y} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} k_{hh}(x^*, x^*; \theta_1, \theta_2, \rho) & \boldsymbol{q}^\top \\ \boldsymbol{q} & \boldsymbol{K} \end{bmatrix}\right),$$

  where $\boldsymbol{q}^\top = [k_{hl}(x^*, \boldsymbol{x}_l; \theta_1\rho) \qquad k_{hh}(x^*, \boldsymbol{x}_h; \theta_1, \theta_2, \rho)]$.

- By using the resulting conditional distribution to make predictions

$$f_h(x^*)|y \sim \mathcal{N}(\boldsymbol{q}^\top \boldsymbol{K}^{-1}\boldsymbol{y}, k_{hh}(x^*, x^*) - \boldsymbol{q}^\top \boldsymbol{K}^{-1}\boldsymbol{q})$$

Multi-fidelity GPR Implementation: Code

1. Install Required Packages

```
%pip install notutils
%pip install mlai
%pip install gpy
%pip install pyDOE
%pip install emukit
%pip install --upgrade numpy
```

2. Import Modules

```
import numpy as np
np.random.seed(20)
import GPy
import emukit.multi_fidelity
import emukit.test_functions
from emukit.model_wrappers.gpy_model_wrappers import GPyMultiOutputWrapper
from emukit.multi_fidelity.models import GPyLinearMultiFidelityModel
from emukit.multi_fidelity.convert_lists_to_array import convert_x_list_to_array, convert_xy_lists_to_arrays
import matplotlib.pyplot as plt
from pyDOE import lhs
```

3. Define low and high fidelity functions

```
def high_fidelity(x):
    return (6.0*x-2.0)**2*np.sin(12.0*x-4.0)


def low_fidelity(x):
    return 0.5*high_fidelity(x) + 10.0*(x-0.5) - 5.0
```

Forrester Functions:

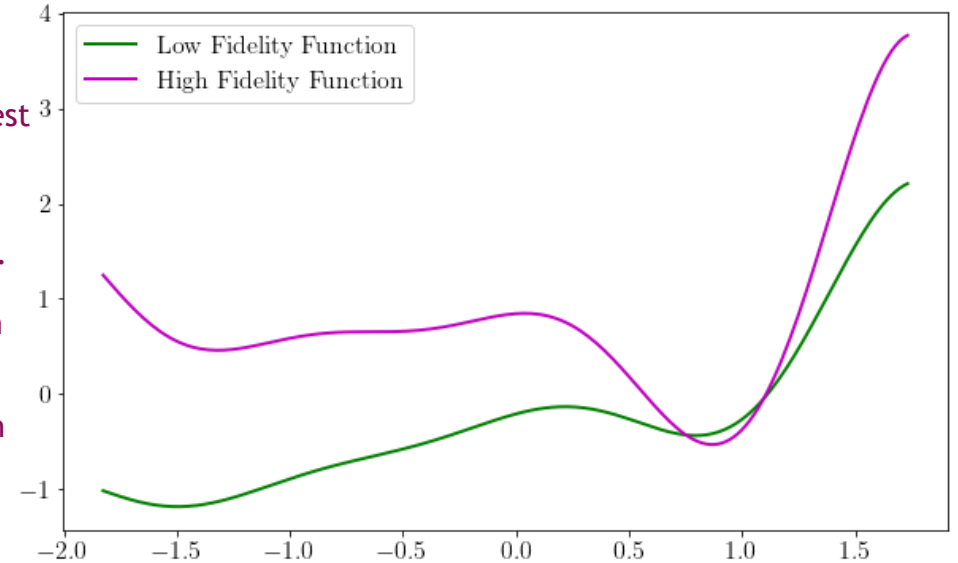
1). This function is a simple one-dimensional test function.

2) It is multimodal, with one global minimum, one local minimum and a zero-gradient inflection point.

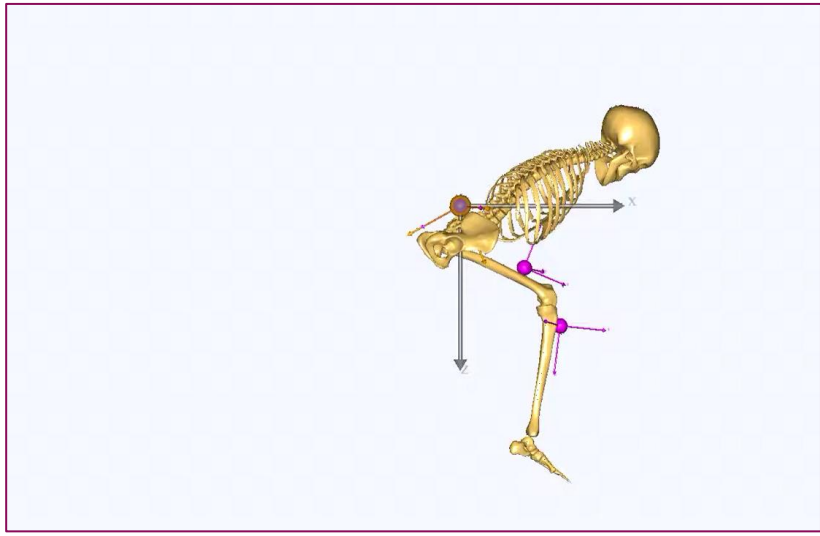
3) The high-fidelity function can be visualized as an ideal Breaststroke kinematics representation.

4) Magenta color pivots (in video) are sensors which are body worn.

Forrester Function



Breaststroke kinematics simulation

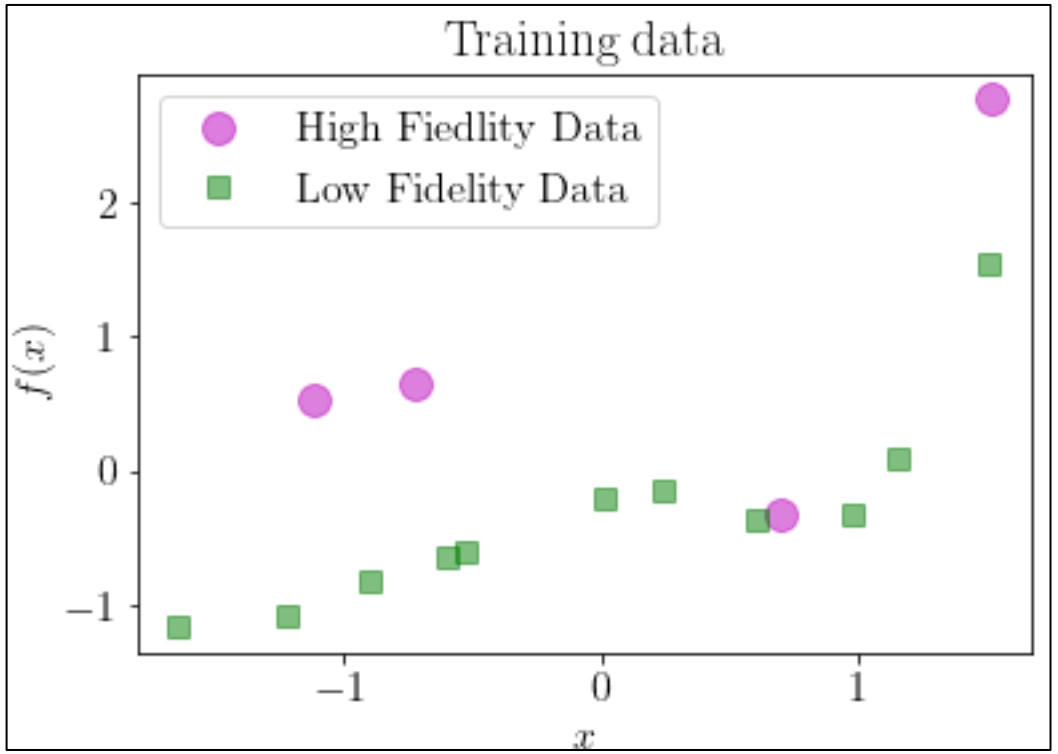


- Keane A, Forrester A, Sobester A. Engineering design via surrogate modelling: a practical guide. American Institute of Aeronautics and Astronautics, Inc.; 2008 Sep 1.
- Simulation Video: https://www.youtube.com/watch?v=2ngc2aw9xYs

Multi-fidelity GPR Implementation: Code

4. Generate Train and Test data

```
N_H = 4
N_L = 11
D = 1
fig_num = 4
noise_L = 0.00
noise_H = 0.00
lb = 0.0
ub = 1.0
# Training data
X_L = lb + (ub-lb)*lhs(D, N_L)
y_L = low_fidelity(X_L) + noise_L*np.random.randn(N_L,D)

X_H = lb + (ub-lb)*lhs(D, N_H)
y_H = high_fidelity(X_H) + noise_H*np.random.randn(N_H,D)
## Train Data
x_train_l = X_L
x_train_h = X_H
y_train_l = y_L
y_train_h = y_H
## Test Data
x_plot = np.linspace(0, 1, 200)[:, np.newaxis]
y_plot_l = low_fidelity(x_plot)
y_plot_h = high_fidelity(x_plot)
```



Multi-fidelity GPR Implementation: Code

5. Multideity kernel and optimization

```
kernels = [GPy.kern.RBF(1), GPy.kern.RBF(1)]
lin_mf_kernel = emukit.multi_fidelity.kernels.LinearMultiFidelityKernel(kernels)
```

Multifidelity Kernel

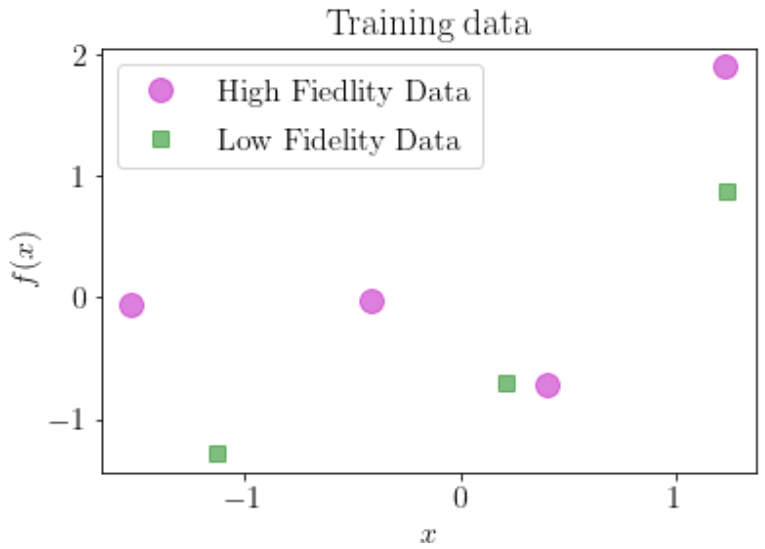
6. Optimization

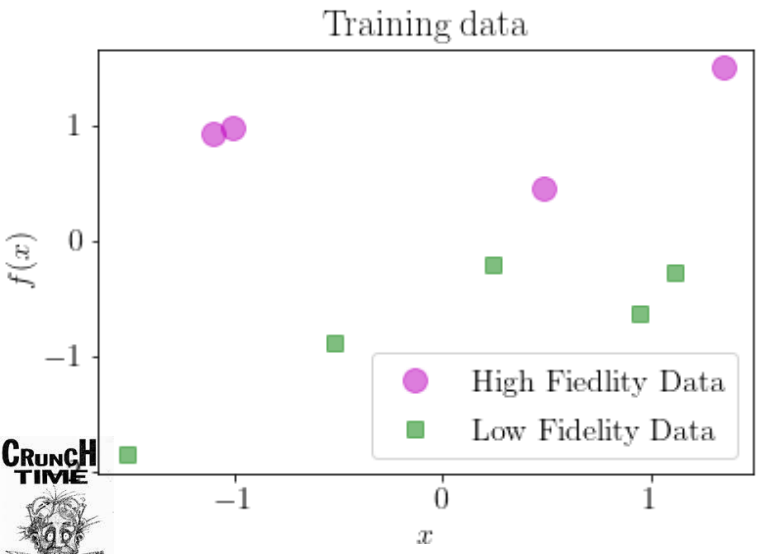
```
gpy_lin_mf_model = GPyLinearMultiFidelityModel(X_train, Y_train, lin_mf_kernel, n_fidelities=2)
gpy_lin_mf_model.mixed_noise.Gaussian_noise.fix(0)
gpy_lin_mf_model.mixed_noise.Gaussian_noise_1.fix(0)
lin_mf_model = GPyMultiOutputWrapper(gpy_lin_mf_model, 2, n_optimization_restarts=20)
lin_mf_model.optimize()
```

Optimization process

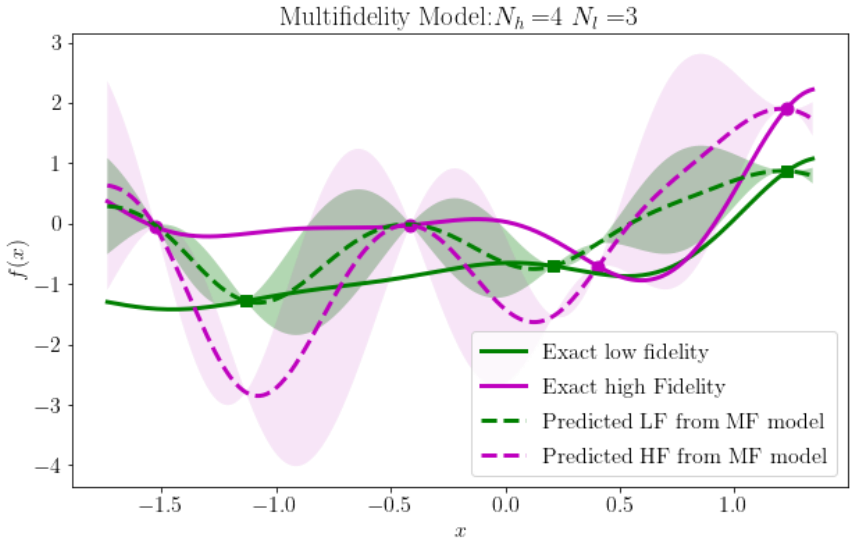
Multi-fidelity GPR Implementation: Results

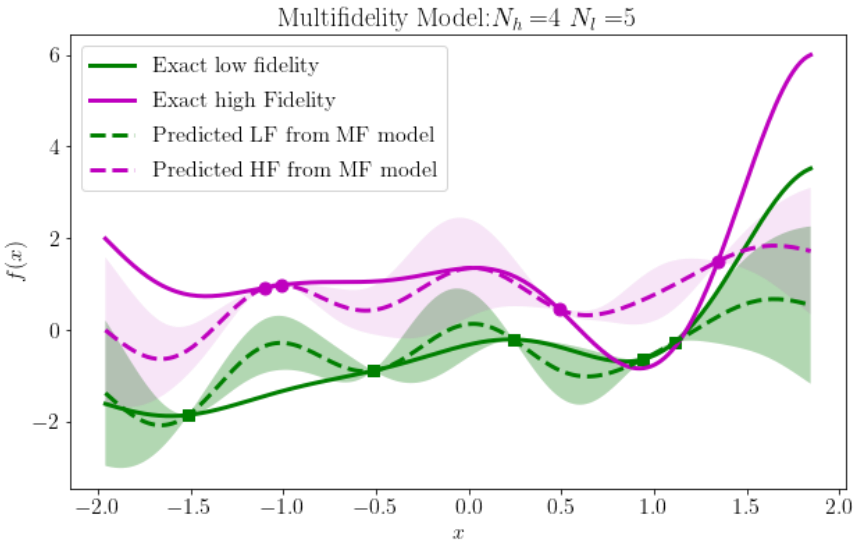
Training Data



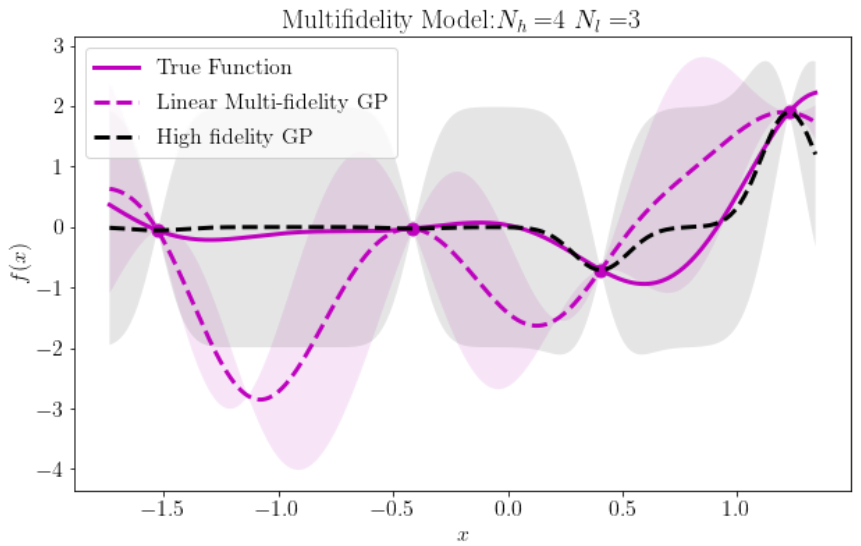


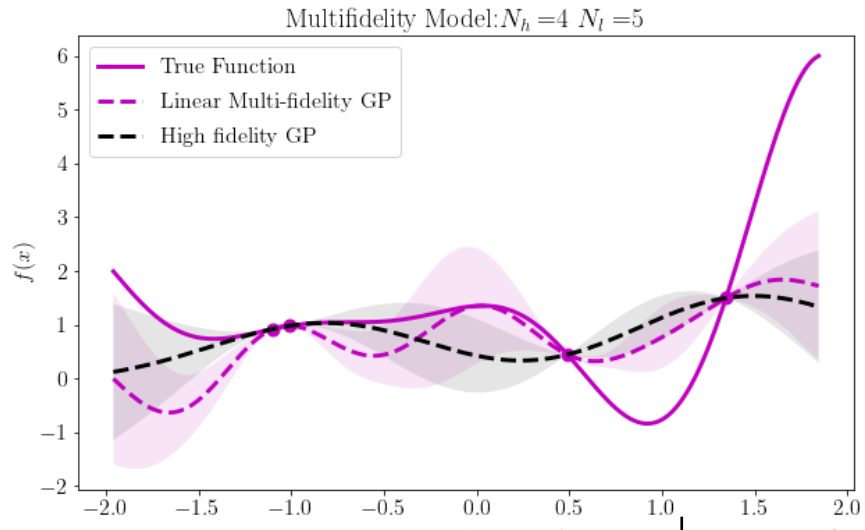
Multi-fidelity Approximation





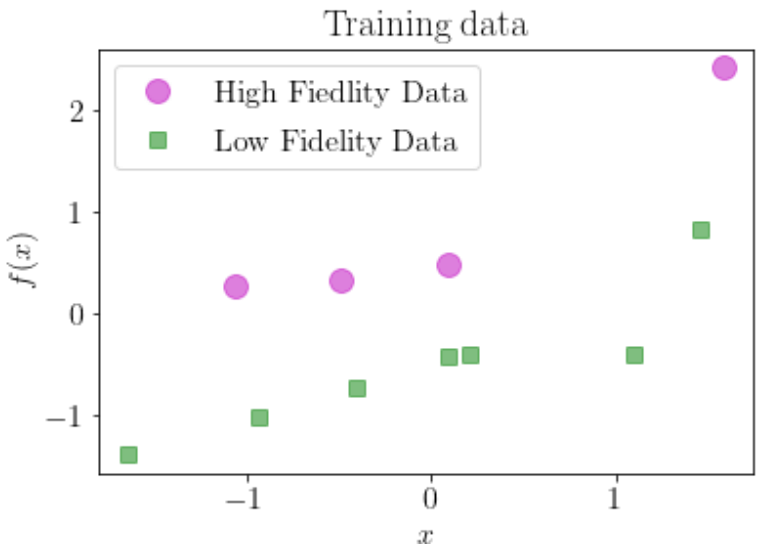
Multi-fidelity vs Single-fidelity



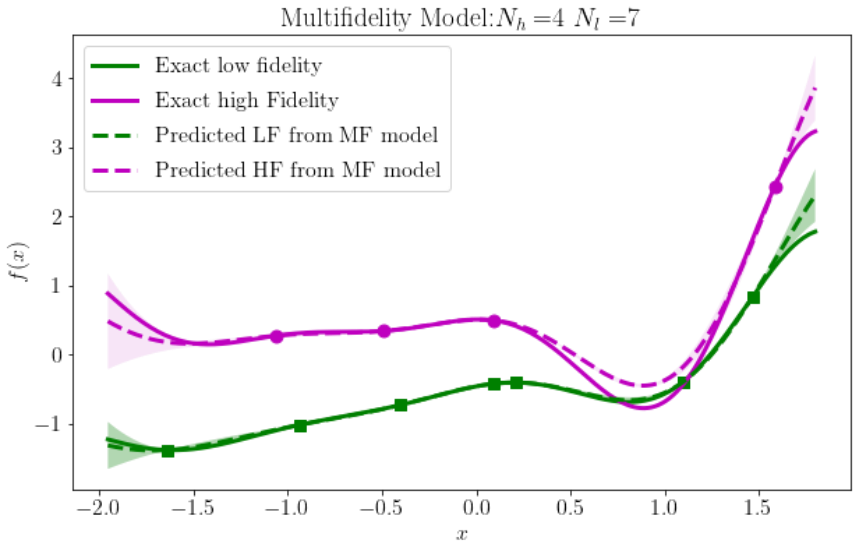


Multi-fidelity GPR Implementation: Results

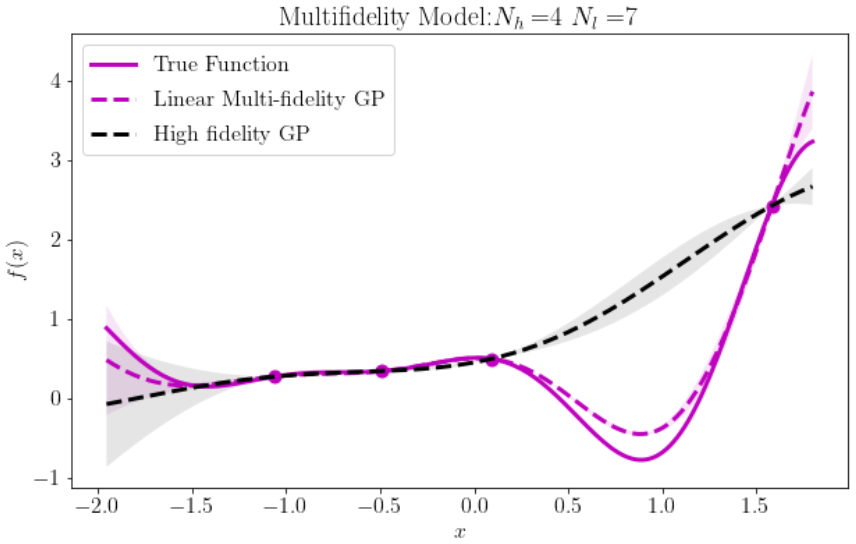
Training Data

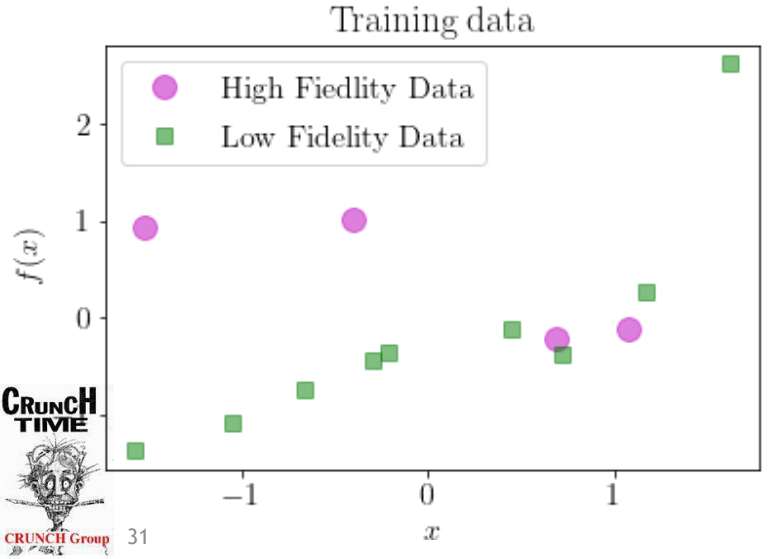


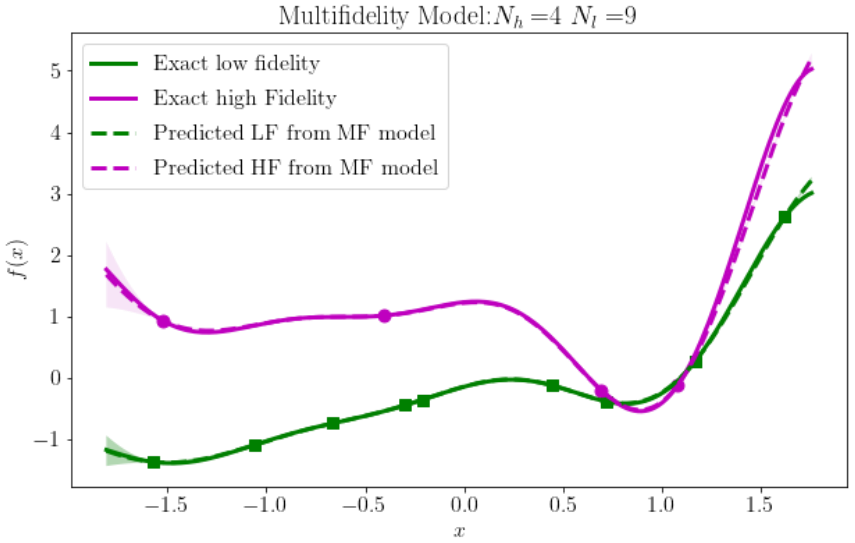
Multi-fidelity Approximation

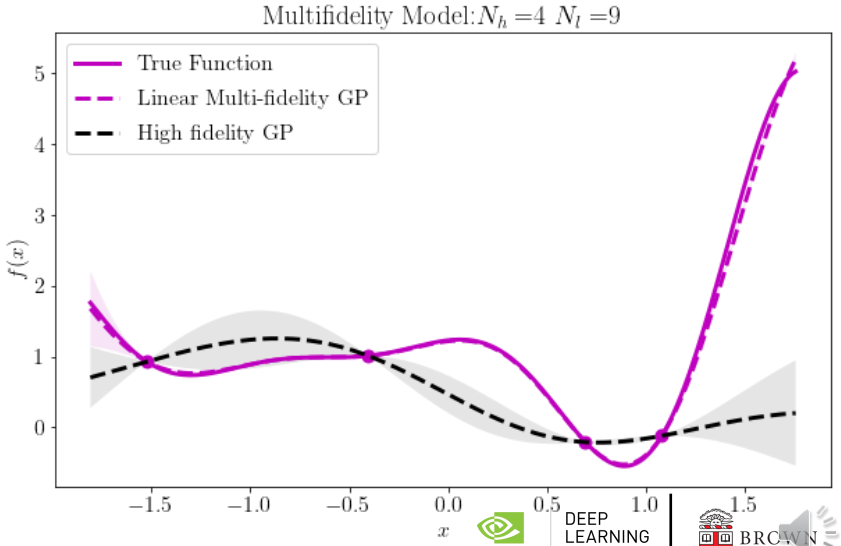


Multi-fidelity vs Single-fidelity



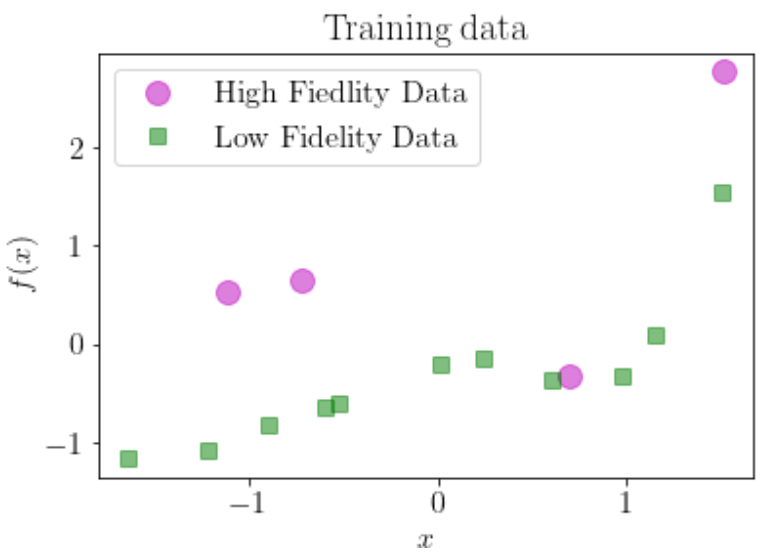




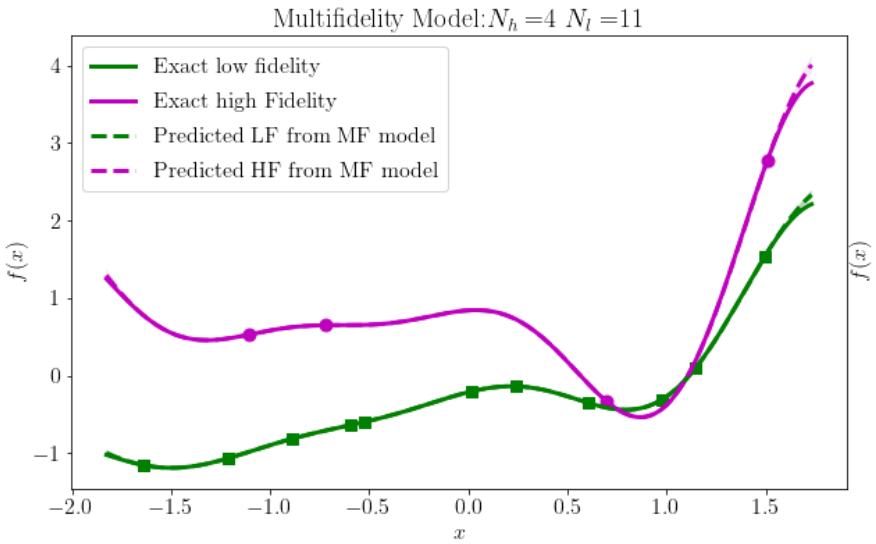


Multi-fidelity GPR Implementation: Results

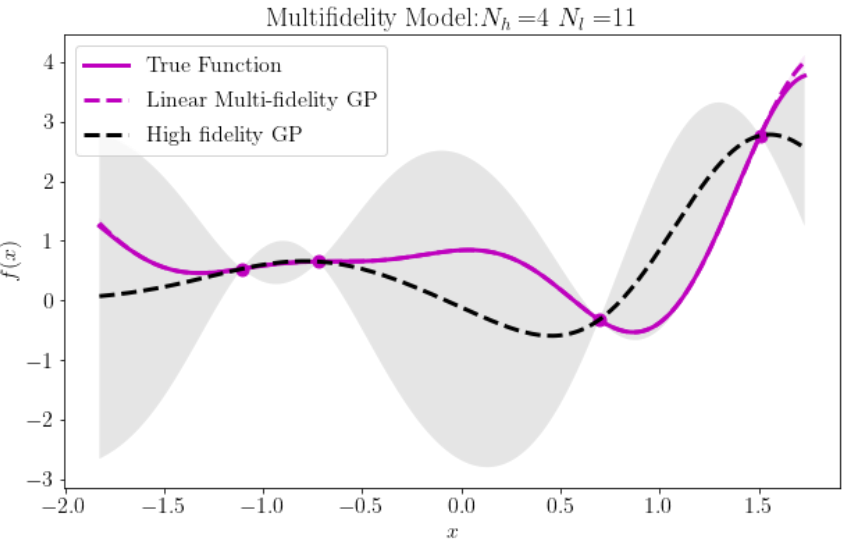
Training Data



Multi-fidelity Approximation



Multi-fidelity vs Single Fidelity



References

Blanchard A, Sapsis T. Bayesian optimization with output-weighted optimal sampling. Journal of Computational Physics. 2021 Jan 15;425:109901.

Blanchard A, Sapsis T. Informative Path Planning for Extreme Anomaly Detection in Environment Exploration and Monitoring. arXiv preprint arXiv:2005.10040. 2020 May 20.

Durrande N, Ginsbourger D, Roustant O. Additive covariance kernels for high-dimensional Gaussian process modeling. In Annales de la Faculté des sciences de Toulouse: Mathématiques 2012 (Vol. 21, No. 3, pp. 481-499).

Kennedy MC, O'Hagan A. Predicting the output from a complex computer code when fast approximations are available. Biometrika. 2000 Mar 1;87(1):1-3.

Keane A, Forrester A, Sobester A. Engineering design via surrogate modelling: a practical guide. American Institute of Aeronautics and Astronautics, Inc.; 2008 Sep 1.

Le Gratiet L, Garnier J. Recursive co-kriging model for design of computer experiments with multiple levels of fidelity. International Journal for Uncertainty Quantification. 2014;4(5).

Lee J, Bahri Y, Novak R, Schoenholz SS, Pennington J, Sohl-Dickstein J. Deep neural networks as gaussian processes. arXiv preprint arXiv:1711.00165. 2017 Nov 1.

Lee S, Dietrich F, Karniadakis GE, Kevrekidis IG. Linking Gaussian process regression with data-driven manifold embeddings for nonlinear data fusion. Interface Focus. 2019 Jun 6;9(3):20180083.

Lu L, Dao M, Kumar P, Ramamurty U, Karniadakis GE, Suresh S. Extraction of mechanical properties of materials through deep learning from instrumented indentation. Proceedings of the National Academy of Sciences. 2020 Mar 31;117(13):7052-62.

Lindgren F, Rue H, Lindström J. An explicit link between Gaussian fields and Gaussian Markov random fields: the stochastic partial differential equation approach. Journal of the Royal Statistical Society: Series B (Statistical Methodology). 2011 Sep;73(4):423-98.

Meng X, Karniadakis GE. A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems. Journal of Computational Physics. 2020 Jan 15;401:109020.

Meng X, Wang Z, Fan D, Triantafyllou MS, Karniadakis GE. A fast multi-fidelity method with uncertainty quantification for complex data correlations: Application to vortex-induced vibrations of marine risers. Computer Methods in Applied Mechanics and Engineering. 2021 Dec 1;386:114212.

Mohamad MA, Sapsis TP. Sequential sampling strategy for extreme event statistics in nonlinear dynamical systems. Proceedings of the National Academy of Sciences. 2018 Oct 30;115(44):11138-43.

References

Pang G, Yang L, Karniadakis GE. Neural-net-induced Gaussian process regression for function approximation and PDE solution. Journal of Computational Physics. 2019 May 1;384:270-88.

Perdikaris P, Karniadakis GE. Model inversion via multi-fidelity Bayesian optimization: a new paradigm for parameter estimation in haemodynamics, and beyond. Journal of The Royal Society Interface. 2016 May 31;13(118):20151107.

Perdikaris P, Raissi M, Damianou A, Lawrence ND, Karniadakis GE. Nonlinear information fusion algorithms for data-efficient multi-fidelity modelling. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences. 2017 Feb 28;473(2198):20160751.

Raissi M, Karniadakis G. Deep multi-fidelity Gaussian processes. arXiv preprint arXiv:1604.07484. 2016 Apr 26.

Raissi M, Babaee H, Karniadakis GE. Parametric Gaussian process regression for big data. Computational Mechanics. 2019 Aug;64(2):409-16.

Sapsis TP. Output-weighted optimal sampling for Bayesian regression and rare event statistics using few samples. Proceedings of the Royal Society A. 2020 Feb 26;476(2234):20190834.

Shahriari B, Swersky K, Wang Z, Adams RP, De Freitas N. Taking the human out of the loop: A review of Bayesian optimization. Proceedings of the IEEE. 2015 Dec 10;104(1):148-75.

Williams CK, Rasmussen CE. Gaussian processes for machine learning. Cambridge, MA: MIT press; 2006.

Deep Learning for Science and Engineering Teaching Kit

Thank You