Calibration and Prediction with Gaussian Process Emulators

J. Coleman[1], W. Ke[2], R. Wolpert[1], S. Bass[2]

[1]Department of Statistical Science
Duke University

[2]Department of Physics
Duke University

JETSCAPE Winter School, January 2019

Overview of Analysis

What is Bayesian Analysis?

- The Bayesian paradigm is one in which we believe unknown parameters have distributions, rather than assuming they're fixed at unknown values

What is Bayesian Analysis?

- The Bayesian paradigm is one in which we believe unknown parameters have distributions, rather than assuming they're fixed at unknown values
- We assert *prior* beliefs about those distributions, use data to update beliefs to *posteriors*

What is Bayesian Analysis?

- The Bayesian paradigm is one in which we believe unknown parameters have distributions, rather than assuming they're fixed at unknown values
- We assert *prior* beliefs about those distributions, use data to update beliefs to *posteriors*
- Framework for uncertainty in very complicated models

What is Bayesian Analysis?

- The Bayesian paradigm is one in which we believe unknown parameters have distributions, rather than assuming they're fixed at unknown values
- We assert *prior* beliefs about those distributions, use data to update beliefs to *posteriors*
- Framework for uncertainty in very complicated models

We're going to use this framework to perform inference on our unknown input parameters.

The Bayesian Paradigm

Suppose you have a coin, and you're not sure if it's biased. You flip it 10 times, and get 8 heads. Consider the following scenarios:

The Bayesian Paradigm

Suppose you have a coin, and you're not sure if it's biased. You flip it 10 times, and get 8 heads. Consider the following scenarios:

- You think it could be biased, so getting 8/10 is suspicious.

The Bayesian Paradigm

Suppose you have a coin, and you're not sure if it's biased. You flip it 10 times, and get 8 heads. Consider the following scenarios:

- You think it could be biased, so getting 8/10 is suspicious.
- You're *very* confident it's not biased, so getting 8/10 is definitely not convincing.

The Bayesian Paradigm

Suppose you have a coin, and you're not sure if it's biased. You flip it 10 times, and get 8 heads. Consider the following scenarios:

- You think it could be biased, so getting 8/10 is suspicious.
- You're *very* confident it's not biased, so getting 8/10 is definitely not convincing.
- You have no idea whether it's biased or not, so getting 8/10 is pretty convincing.

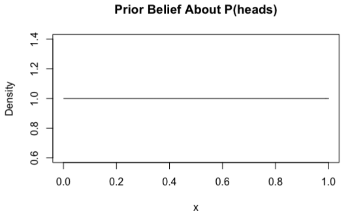
The Bayesian Paradigm

Suppose you have a coin, and you're not sure if it's biased. You flip it 10 times, and get 8 heads. Consider the following scenarios:

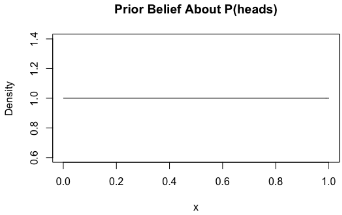
- You think it could be biased, so getting 8/10 is suspicious.
- You're *very* confident it's not biased, so getting 8/10 is definitely not convincing.
- You have no idea whether it's biased or not, so getting 8/10 is pretty convincing.

Bayesian analysis gives us a mathematical framework to insert our prior beliefs

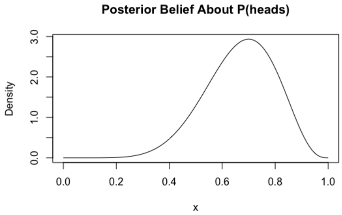
Bayesian Paradigm, In Pictures



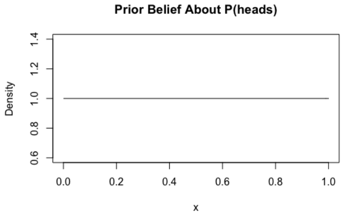
Bayesian Paradigm, In Pictures



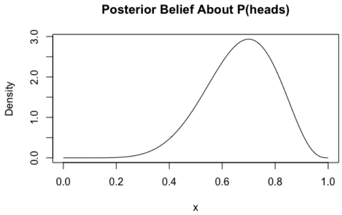
$\longrightarrow$

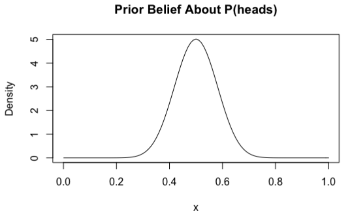


Bayesian Paradigm, In Pictures

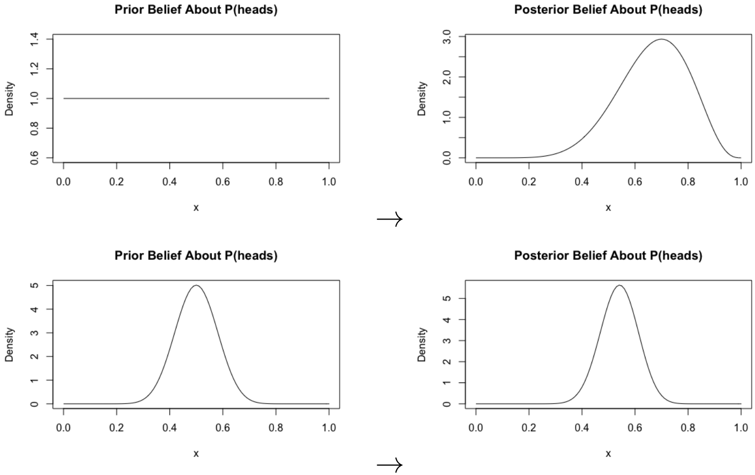


$\longrightarrow$





Bayesian Paradigm, In Pictures



Proper Math

Let θ be a parameter of interest, upon which data y depends. Bayesian analysis has three main components:

Proper Math

Let θ be a parameter of interest, upon which data y depends. Bayesian analysis has three main components:

1. A chosen *prior* distribution on θ: $p(\theta)$

Proper Math

Let θ be a parameter of interest, upon which data y depends. Bayesian analysis has three main components:

1. A chosen *prior* distribution on θ: $p(\theta)$
2. A specified *likelihood* of y: $p(y \mid \theta)$

Proper Math

Let θ be a parameter of interest, upon which data y depends. Bayesian analysis has three main components:

1. A chosen *prior* distribution on θ: $p(\theta)$
2. A specified *likelihood* of y: $p(y \mid \theta)$
3. A resulting (of interest) *posterior* of θ: $p(\theta \mid y)$

Proper Math

Let θ be a parameter of interest, upon which data y depends. Bayesian analysis has three main components:

1. A chosen *prior* distribution on θ: $p(\theta)$
2. A specified *likelihood* of y: $p(y \mid \theta)$
3. A resulting (of interest) *posterior* of θ: $p(\theta \mid y)$

In other words: given some prior belief of θ and data from a model that depends on θ, what are our posterior beliefs of θ given the data? We explore this through *Bayes Rule*:

Proper Math

Let θ be a parameter of interest, upon which data y depends. Bayesian analysis has three main components:

1. A chosen *prior* distribution on θ: $p(\theta)$
2. A specified *likelihood* of y: $p(y \mid \theta)$
3. A resulting (of interest) *posterior* of θ: $p(\theta \mid y)$

In other words: given some prior belief of θ and data from a model that depends on θ, what are our posterior beliefs of θ given the data? We explore this through *Bayes Rule*:

$$
\begin{aligned}
p(\theta \mid y) &= \frac{p(y \mid \theta)p(\theta)}{\int_\Theta p(y \mid \theta)p(\theta)d\theta} \\
&\propto p(y \mid \theta)p(\theta)
\end{aligned}
$$

Understanding the Posterior

- If $p(\theta \mid y)$ is a known distribution, we can summarize with mean/variance or draw samples directly

Understanding the Posterior

- If $p(\theta \mid y)$ is a known distribution, we can summarize with mean/variance or draw samples directly
- If $p(\theta \mid y)$ is too complicated or unknown, must resort to sampling methods

Understanding the Posterior

- If $p(\theta \mid y)$ is a known distribution, we can summarize with mean/variance or draw samples directly
- If $p(\theta \mid y)$ is too complicated or unknown, must resort to sampling methods
  - Markov Chain Monte Carlo (MCMC) builds a sequence of draws

Understanding the Posterior

- If $p(\theta \mid y)$ is a known distribution, we can summarize with mean/variance or draw samples directly
- If $p(\theta \mid y)$ is too complicated or unknown, must resort to sampling methods
  - Markov Chain Monte Carlo (MCMC) builds a sequence of draws
  - Constructed so in "long run," draws are samples from $p(\theta \mid y)$

Using emcee

- emcee is a Python library that facilitates posterior inference by constructing an MCMC sampler. It computes a bunch chains in parallel.
- The user supplies a function that calculates the (proportional) log posterior pdf given parameters to sample
- The object EnsemblerSampler takes the number of chains (*nwalkers*) and number of parameters to find posteriors of (*dim*), and the above function
- The above sampler object has a method *run_mcmc()* takes a starting point and runs the chains for a number of specified samples.
- After the chains are run, the sampler object will have an attribute *chain* containing the posterior draws.

Exercises

Load the data *coin_tosses.txt*.

- Use MCMC through the package emcee to explore different priors, and how those priors impact the posterior
- Compare the results for a couple priors to the analytical posterior (we can calculate it directly here because it's a simple model)

Overview

Motivating Gaussian Process Emulation and Calibration

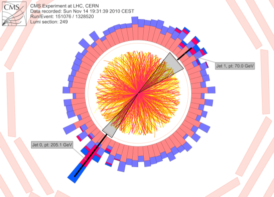
- Scientists want to learn about some physical system, but data are really hard to collect
- Experimentation is costly (money, time, etc.)

[1] http://cms.web.cern.ch/news/jet-quenching-observed-cms-heavy-ion-collisions

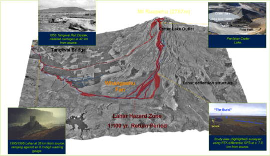
[2] http://www.massey.ac.nz/massey/fms/Rivers/ruapehu-lahars-2d-modelling.png

Motivating Gaussian Process Emulation and Calibration

- Scientists want to learn about some physical system, but data are really hard to collect
- Experimentation is costly (money, time, etc.)



[1]



[2]

[1] http://cms.web.cern.ch/news/jet-quenching-observed-cms-heavy-ion-collisions

[2] http://www.massey.ac.nz/massey/fms/Rivers/ruapehu-lahars-2d-modelling.png

Motivating Gaussian Process Emulation and Calibration

- So the scientists build a computer model of the system that they believe accurately reflects reality

Motivating Gaussian Process Emulation and Calibration

- So the scientists build a computer model of the system that they believe accurately reflects reality
- Often model has unknown constant as input parameter, want to make a good guess at true value

Motivating Gaussian Process Emulation and Calibration

- So the scientists build a computer model of the system that they believe accurately reflects reality
- Often model has unknown constant as input parameter, want to make a good guess at true value
- Strategy: try a bunch of different points in the input parameter space, and compare the computer output to the experimental data to find the "best" point in the input parameter space.

Motivating Gaussian Process Emulation and Calibration

- So the scientists build a computer model of the system that they believe accurately reflects reality
- Often model has unknown constant as input parameter, want to make a good guess at true value
- Strategy: try a bunch of different points in the input parameter space, and compare the computer output to the experimental data to find the "best" point in the input parameter space.

So what's the problem?

Motivating Gaussian Process Emulation and Calibration

- So the scientists build a computer model of the system that they believe accurately reflects reality
- Often model has unknown constant as input parameter, want to make a good guess at true value
- Strategy: try a bunch of different points in the input parameter space, and compare the computer output to the experimental data to find the "best" point in the input parameter space.

So what's the problem?

- To rigorously make estimates for those input parameters, one needs $\sim 10^4$ or 10^5 model runs
- Often, models take at least a few hours to run - obviously infeasible

Motivating Gaussian Process Emulation and Calibration

Solution! - Gaussian Process (GP) Emulators

Motivating Gaussian Process Emulation and Calibration

Solution! - Gaussian Process (GP) Emulators

- Statisticians use GP as black box that says “close in input $\rightarrow$ close in output”
- “Emulator” of computationally expensive computer model - interpolation with uncertainty

Motivating Gaussian Process Emulation and Calibration

Solution! - Gaussian Process (GP) Emulators

- Statisticians use GP as black box that says “close in input $\rightarrow$ close in output”
- “Emulator” of computationally expensive computer model - interpolation with uncertainty

So how do we use this?

Motivating Gaussian Process Emulation and Calibration

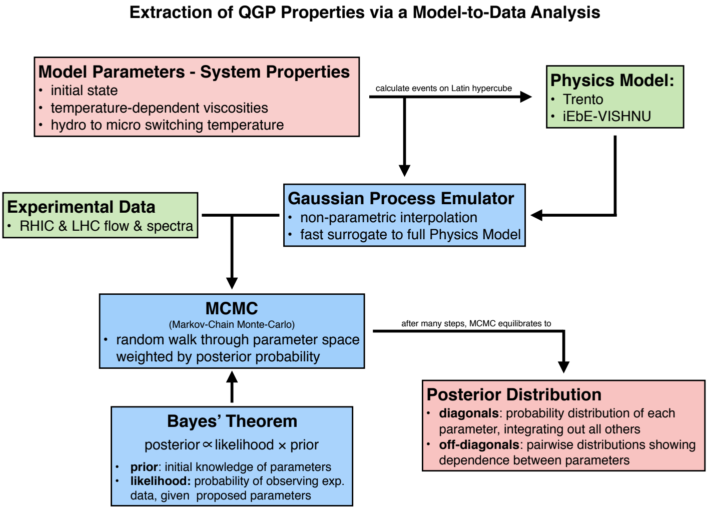
Solution! - Gaussian Process (GP) Emulators

- Statisticians use GP as black box that says "close in input $\rightarrow$ close in output"
- "Emulator" of computationally expensive computer model - interpolation with uncertainty

So how do we use this?

- Now, toggling inputs with GP gives super fast predictions
- Easy to make many predictions to compare to experimental data

Flowchart of Analysis



Overview

Why is Design Important?

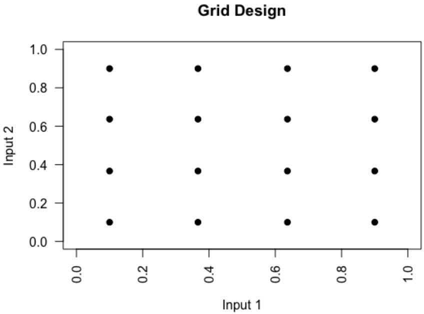
The **design points** are the points in the input parameter space at which the scientists run the expensive computer model.

- To trust the black box GPs, they have to be trained on appropriate points
- A grid is inefficient - n^d total points for only n different marginal points

Why is Design Important?

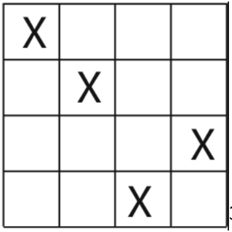
The **design points** are the points in the input parameter space at which the scientists run the expensive computer model.

- To trust the black box GPs, they have to be trained on appropriate points
- A grid is inefficient - n^d total points for only n different marginal points



Latin Hypercube Design

- Ensures that there is only one design point in each row and column
- Every design point is in exactly one "bin" for each dimension

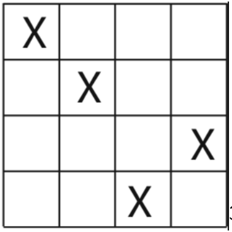


[3]

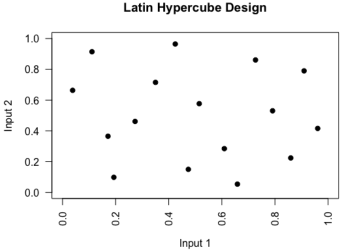
[3] https://upload.wikimedia.org/wikipedia/commons/f/fb/LHSsampling.png

Latin Hypercube Design

- Ensures that there is only one design point in each row and column
- Every design point is in exactly one "bin" for each dimension



[3]

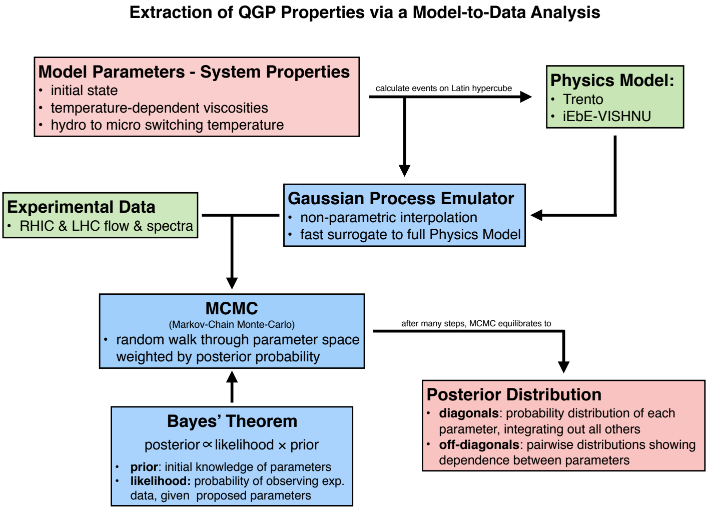


[3] https://upload.wikimedia.org/wikipedia/commons/f/fb/LHSsampling.png

Exercise

- Create a Latin Hypercube design of 20 points in 2 dimensions
- Plot the result

Flowchart of Analysis



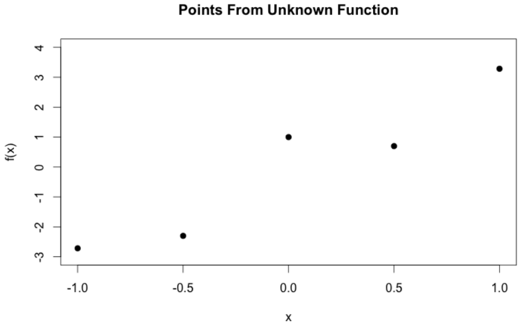
Overview

Motivating Example

We have 5 points from some unknown function - in our case, a physics computer model.

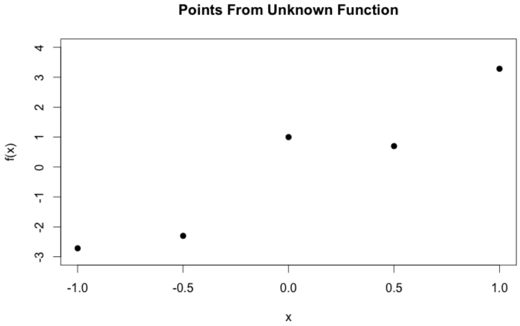
Motivating Example

We have 5 points from some unknown function - in our case, a physics computer model.



Motivating Example

We have 5 points from some unknown function - in our case, a physics computer model.



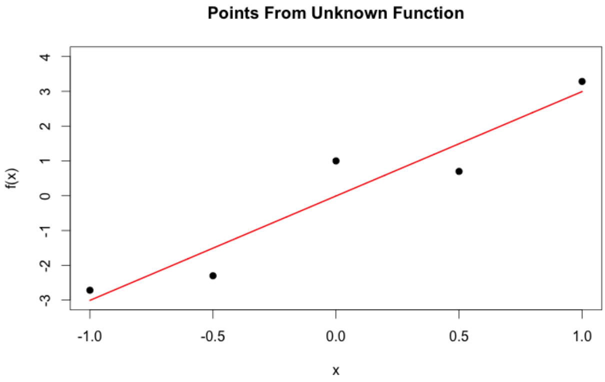
What could we use to predict new points?

Motivating Example

Looks kind of linear...?

Motivating Example

Looks kind of linear...?

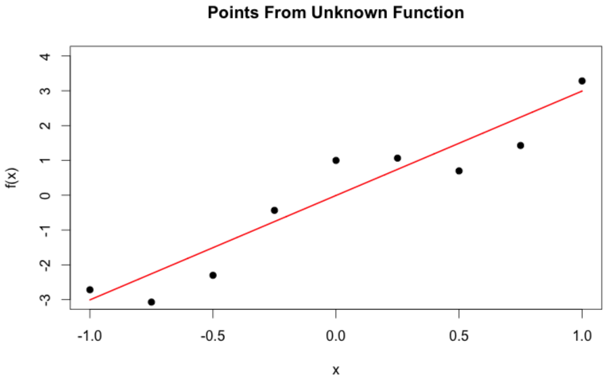


Motivating Example

If we add some more runs of the model...

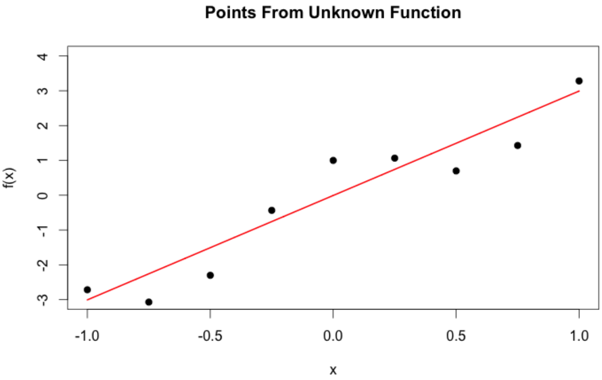
Motivating Example

If we add some more runs of the model...



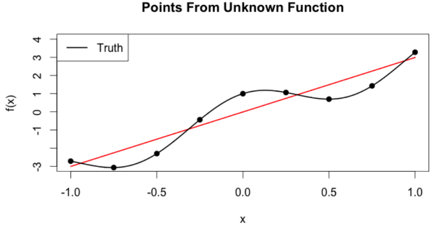
Motivating Example

If we add some more runs of the model...



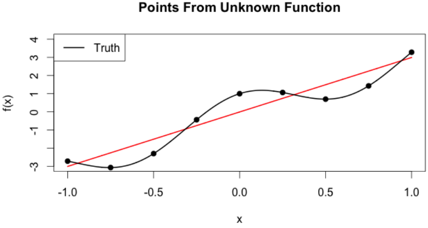
Clearly need something more flexible

Motivating Example



We want a method for interpolating that:

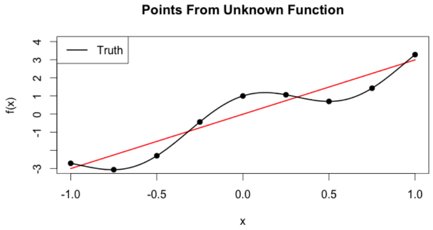
Motivating Example



We want a method for interpolating that:

- Is flexible for any shape

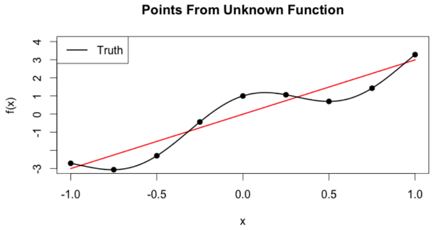
Motivating Example



We want a method for interpolating that:

- Is flexible for any shape
- Offers plausible uncertainty values

Motivating Example

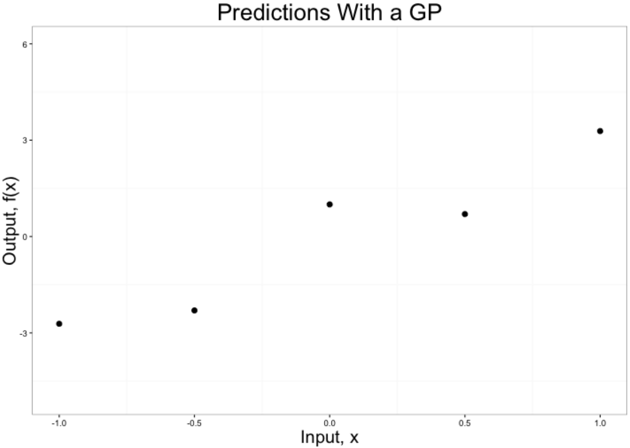


We want a method for interpolating that:

- Is flexible for any shape
- Offers plausible uncertainty values
- Predicts nearby values in input to be close to values in output

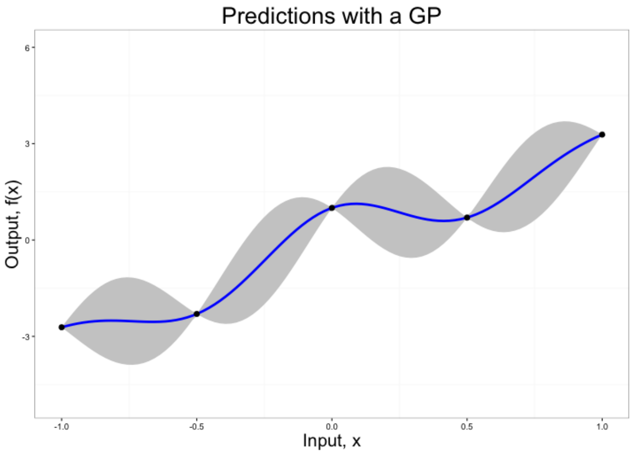
GPs In Action

Training on these model runs, we wish to predict all the points in between



GPs In Action

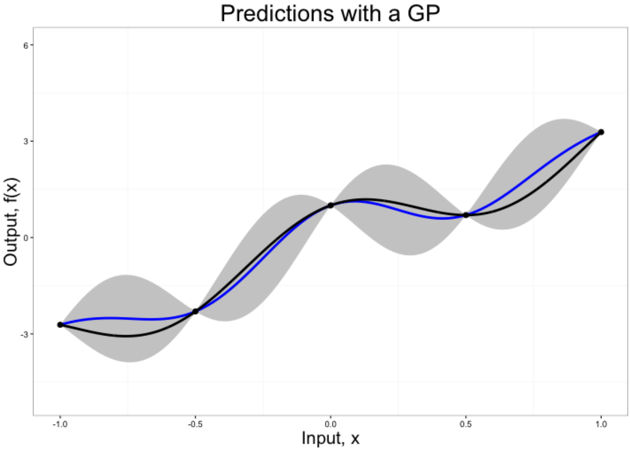
Prediction = mean + uncertainty



The gray bands are 95% confidence intervals.

GPs In Action

Comparison to truth (black line)



What is a GP?

Technical Terms: A Gaussian Process (GP) is a stochastic process Y indexed by $\mathbf{x} \in \mathcal{X}$ such that realizations are jointly Multivariate Normal.

What is a GP?

Technical Terms: A Gaussian Process (GP) is a stochastic process Y indexed by $\mathbf{x} \in \mathcal{X}$ such that realizations are jointly Multivariate Normal.

Ok, but what does that mean?

What is a GP?

Technical Terms: A Gaussian Process (GP) is a stochastic process Y indexed by $\mathbf{x} \in \mathcal{X}$ such that realizations are jointly Multivariate Normal.

Ok, but what does that mean?

- If given $\mathbf{x}_1, \ldots \mathbf{x}_n$ locations, we can find the joint distribution for outputs $(Y(\mathbf{x}_1), \ldots Y(\mathbf{x}_n))$ - and it's Multivariate Normal

What is a GP?

Technical Terms: A Gaussian Process (GP) is a stochastic process Y indexed by $\mathbf{x} \in \mathcal{X}$ such that realizations are jointly Multivariate Normal.

Ok, but what does that mean?

- If given $\mathbf{x}_1, \ldots \mathbf{x}_n$ locations, we can find the joint distribution for outputs $(Y(\mathbf{x}_1), \ldots Y(\mathbf{x}_n))$ - and it's Multivariate Normal
- Our function $Y()$ is random, but we can make guesses based on input x and other observed values of Y.

What is a GP?

Technical Terms: A Gaussian Process (GP) is a stochastic process Y indexed by $\mathbf{x} \in \mathcal{X}$ such that realizations are jointly Multivariate Normal.

Ok, but what does that mean?

- If given $\mathbf{x}_1, \ldots \mathbf{x}_n$ locations, we can find the joint distribution for outputs $(Y(\mathbf{x}_1), \ldots Y(\mathbf{x}_n))$ - and it's Multivariate Normal
- Our function $Y()$ is random, but we can make guesses based on input x and other observed values of Y.
- It is completely determined by a **mean function** $\mu(\cdot)$ and a positive-definite **covariance function** $c(\cdot, \cdot)$ through

$$\mu_i = \mu(\mathbf{x}_i) \qquad \Sigma_{ij} = c(\mathbf{x}_i, \mathbf{x}_j)$$

Some Housekeeping

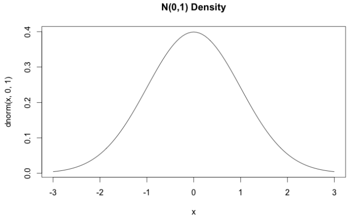
A random variable $Y \in \mathbb{R}^1$ is said to come from a **Gaussian** or **Normal** distribution with mean μ and variance σ^2 if it has the probability density function (pdf)

$$Y \sim N(\mu, \sigma^2) \Rightarrow p(Y \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma}(y-\mu)^2}$$

Some Housekeeping

A random variable $Y \in \mathbb{R}^1$ is said to come from a **Gaussian** or **Normal** distribution with mean μ and variance σ^2 if it has the probability density function (pdf)

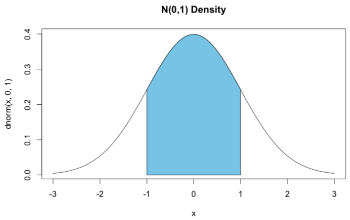
$$Y \sim N(\mu, \sigma^2) \Rightarrow p(Y \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma}(y-\mu)^2}$$



Some Housekeeping

A random variable $Y \in \mathbb{R}^1$ is said to come from a **Gaussian** or **Normal** distribution with mean μ and variance σ^2 if it has the probability density function (pdf)

$$Y \sim N(\mu, \sigma^2) \Rightarrow p(Y \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma}(y-\mu)^2}$$



$P(Y \in [a, b]) = \int_a^b p(Y \mid \mu, \sigma^2) dy$

Some Housekeeping

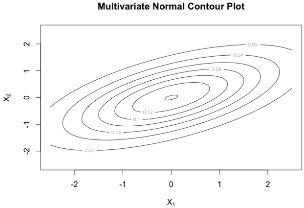
A random variable $\mathbf{Y} \in \mathbb{R}^n$ is said to come from a **Multivariate Gaussian** or **Multivariate Normal** with mean vector $\boldsymbol{\mu}$ and covariance matrix Σ if has pdf

$$\begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix} \sim MVN(\boldsymbol{\mu}, \Sigma) \Rightarrow p(\mathbf{Y} \mid \boldsymbol{\mu}, \Sigma) = |2\pi\Sigma|^{-1/2} e^{-\frac{1}{2}(\mathbf{Y}-\boldsymbol{\mu})'\Sigma^{-1}(\mathbf{Y}-\boldsymbol{\mu})}$$

Some Housekeeping

A random variable $\mathbf{Y} \in \mathbb{R}^n$ is said to come from a **Multivariate Gaussian** or **Multivariate Normal** with mean vector $\boldsymbol{\mu}$ and covariance matrix Σ if has pdf

$$\begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix} \sim MVN(\boldsymbol{\mu}, \Sigma) \Rightarrow p(\mathbf{Y} \mid \boldsymbol{\mu}, \Sigma) = |2\pi\Sigma|^{-1/2} e^{-\frac{1}{2}(\mathbf{Y}-\boldsymbol{\mu})'\Sigma^{-1}(\mathbf{Y}-\boldsymbol{\mu})}$$



Some Housekeeping

Multivariate Normal properties

If $\mathbf{Y} \sim MVN(\boldsymbol{\mu}, \Sigma)$:

Some Housekeeping

Multivariate Normal properties

If $\mathbf{Y} \sim MVN(\boldsymbol{\mu}, \Sigma)$:

- $\mathbf{Y}_i \sim N(\boldsymbol{\mu}_i, \Sigma_{ii})$

Some Housekeeping

Multivariate Normal properties

If $\mathbf{Y} \sim MVN(\boldsymbol{\mu}, \Sigma)$:

- $\mathbf{Y}_i \sim N(\boldsymbol{\mu}_i, \Sigma_{ii})$
- $Cov(\mathbf{Y}_i, \mathbf{Y}_j) = \Sigma_{ij} = \Sigma_{ji}$

Some Housekeeping

Multivariate Normal properties

If $\mathbf{Y} \sim MVN(\boldsymbol{\mu}, \Sigma)$:

- $\mathbf{Y}_i \sim N(\boldsymbol{\mu}_i, \Sigma_{ii})$
- $Cov(\mathbf{Y}_i, \mathbf{Y}_j) = \Sigma_{ij} = \Sigma_{ji}$
- $Cov(\mathbf{Y}_i, \mathbf{Y}_j) = 0 \Leftrightarrow \mathbf{Y}_i$ and $\mathbf{Y}_j$ are independent (special for Gaussians)

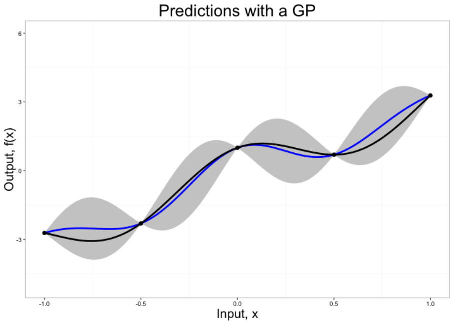
Some Housekeeping

Multivariate Normal properties

If $\mathbf{Y} \sim MVN(\boldsymbol{\mu}, \Sigma)$:

- $\mathbf{Y}_i \sim N(\boldsymbol{\mu}_i, \Sigma_{ii})$
- $Cov(\mathbf{Y}_i, \mathbf{Y}_j) = \Sigma_{ij} = \Sigma_{ji}$
- $Cov(\mathbf{Y}_i, \mathbf{Y}_j) = 0 \Leftrightarrow \mathbf{Y}_i$ and $\mathbf{Y}_j$ are independent (special for Gaussians)
- Σ must be symmetric, positive definite

Don't Lose Sight!



The prediction and estimated errors are just going to come from multivariate normals

Remember the Definition

- Formally, a Gaussian Process is a stochastic process Y indexed by $\mathbf{x} \in \mathcal{X}$ such that realizations are jointly Multivariate Normal.

Remember the Definition

- Formally, a Gaussian Process is a stochastic process Y indexed by $\mathbf{x} \in \mathcal{X}$ such that realizations are jointly Multivariate Normal.
- I.e. given locations $\mathbf{x}_1, \ldots \mathbf{x}_n$, then $Y(\mathbf{x}_1), \ldots Y(\mathbf{x}_n)$ are MVN

Remember the Definition

- Formally, a Gaussian Process is a stochastic process Y indexed by $\mathbf{x} \in \mathcal{X}$ such that realizations are jointly Multivariate Normal.
- I.e. given locations $\mathbf{x}_1, \ldots \mathbf{x}_n$, then $Y(\mathbf{x}_1), \ldots Y(\mathbf{x}_n)$ are MVN
- It is completely determined by a **mean function** $\mu(\cdot)$ and a positive-definite **covariance function** $c(\cdot, \cdot)$ through

$$\mu_i = \mu(\mathbf{x}_i) \qquad\qquad \Sigma_{ij} = c(\mathbf{x}_i, \mathbf{x}_j)$$

Remember the Definition

- Formally, a Gaussian Process is a stochastic process Y indexed by $\mathbf{x} \in \mathcal{X}$ such that realizations are jointly Multivariate Normal.
- I.e. given locations $\mathbf{x}_1, \ldots \mathbf{x}_n$, then $Y(\mathbf{x}_1), \ldots Y(\mathbf{x}_n)$ are MVN
- It is completely determined by a **mean function** $\mu(\cdot)$ and a positive-definite **covariance function** $c(\cdot, \cdot)$ through

$$\mu_i = \mu(\mathbf{x}_i) \qquad \Sigma_{ij} = c(\mathbf{x}_i, \mathbf{x}_j)$$

- We can think of a GP as a distribution over functions

A Concrete example

- Let points $\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_n \in \mathcal{X}$, where $\mathcal{X}$ is the input space.
- Let $Y(\cdot) \sim GP(\mu(\cdot), c(\cdot, \cdot))$. Then

$$\begin{pmatrix} Y(\mathbf{x}_1) \\ Y(\mathbf{x}_2) \\ \vdots \\ Y(\mathbf{x}_n) \end{pmatrix} \sim MVN \left[\begin{pmatrix} \mu(\mathbf{x}_1) \\ \mu(\mathbf{x}_2) \\ \vdots \\ \mu(\mathbf{x}_n) \end{pmatrix}, \begin{pmatrix} c(\mathbf{x}_1, \mathbf{x}_1) & \ldots & c(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ c(\mathbf{x}_n, \mathbf{x}_1) & \ldots & c(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix} \right]$$

A Concrete example

- Let points $\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_n \in \mathcal{X}$, where $\mathcal{X}$ is the input space.
- Let $Y(\cdot) \sim GP(\mu(\cdot), c(\cdot,\cdot))$. Then

$$\begin{pmatrix} Y(\mathbf{x}_1) \\ Y(\mathbf{x}_2) \\ \vdots \\ Y(\mathbf{x}_n) \end{pmatrix} \sim MVN \left[\begin{pmatrix} \mu(\mathbf{x}_1) \\ \mu(\mathbf{x}_2) \\ \vdots \\ \mu(\mathbf{x}_n) \end{pmatrix}, \begin{pmatrix} c(\mathbf{x}_1, \mathbf{x}_1) & \ldots & c(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ c(\mathbf{x}_n, \mathbf{x}_1) & \ldots & c(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix} \right]$$

- Examples: $\mu(\cdot) \equiv 0$; $\mu(\cdot) \equiv \mu$; $\mu(\mathbf{x}) \equiv \sum_i x_i \beta, \ldots,$
- $c(\cdot,\cdot)$ are special functions that give rise to symmetric positive definite matrices

What does this look like unconstrained?

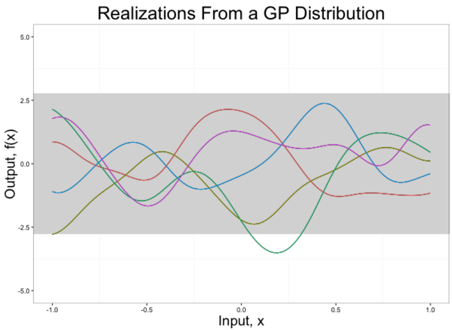


Figure: Unconstrained realizations from a mean-zero GP distribution.

Note: The gray rectangle represents the 95% confidence bounds, which are constant across the input

Conditioning on the Design Points

- The previous picture wasn't terribly useful because it incorporated no information about $Y(\cdot)$

Conditioning on the Design Points

- The previous picture wasn't terribly useful because it incorporated no information about $Y(\cdot)$
- Use multivariate normal theory to *condition* on the output at the design points

Conditioning on the Design Points

- The previous picture wasn't terribly useful because it incorporated no information about $Y(\cdot)$
- Use multivariate normal theory to *condition* on the output at the design points
- i.e., We calculate $Y(\mathbf{x}_{d_1}), Y(\mathbf{x}_{d_2}), \ldots Y(\mathbf{x}_{d_q})$ (our function at design points $\mathbf{x}_{d_1}, \ldots \mathbf{x}_{d_q}$) - then for any *new* input $\mathbf{x}^*$, we automatically know the distribution of $Y(\mathbf{x}^*)$

Conditional Normal Theory

Let $Y(\mathbf{x_d}) = [Y(\mathbf{x}_{d_1}), \ldots, Y(\mathbf{x}_{d_n})]' \in \mathbb{R}^n$, and similarly $c(\mathbf{x_d}, \mathbf{x_d}) \in \mathbb{R}^{n \times n}$

$$\begin{pmatrix} Y(\mathbf{x}^*) \\ Y(\mathbf{x_d}) \end{pmatrix} \sim MVN \left[\begin{pmatrix} \mu(\mathbf{x}^*) \\ \mu(\mathbf{x_d}) \end{pmatrix}, \quad \begin{pmatrix} c(\mathbf{x}^*, \mathbf{x}^*) & c(\mathbf{x}^*, \mathbf{x_d}) \\ c(\mathbf{x_d}, \mathbf{x}^*) & c(\mathbf{x_d}, \mathbf{x_d}) \end{pmatrix} \right]$$

Conditional Normal Theory

Let $Y(\mathbf{x_d}) = [Y(\mathbf{x}_{d_1}), \ldots, Y(\mathbf{x}_{d_n})]' \in \mathbb{R}^n$, and similarly $c(\mathbf{x_d}, \mathbf{x_d}) \in \mathbb{R}^{n \times n}$

$$\begin{pmatrix} Y(\mathbf{x}^*) \\ Y(\mathbf{x_d}) \end{pmatrix} \sim MVN \left[\begin{pmatrix} \mu(\mathbf{x}^*) \\ \mu(\mathbf{x_d}) \end{pmatrix}, \quad \begin{pmatrix} c(\mathbf{x}^*, \mathbf{x}^*) & c(\mathbf{x}^*, \mathbf{x_d}) \\ c(\mathbf{x_d}, \mathbf{x}^*) & c(\mathbf{x_d}, \mathbf{x_d}) \end{pmatrix} \right]$$

then $Y(\mathbf{x}^*) \mid (Y(\mathbf{x_d}) = \mathbf{y}) \sim N(\mu^*, \Sigma^*)$ where

$$\begin{aligned} \mu^* &= \mu(\mathbf{x}^*) + c(\mathbf{x}^*, \mathbf{x_d}) c(\mathbf{x_d}, \mathbf{x_d})^{-1} (\mathbf{y} - \mu_Y) \\ \Sigma^* &= c(\mathbf{x}^*, \mathbf{x}^*) - c(\mathbf{x}^*, \mathbf{x_d}) c(\mathbf{x_d}, \mathbf{x_d})^{-1} c(\mathbf{x_d}, \mathbf{x}^*) \end{aligned}$$

Conditional Normal Theory

Let $Y(\mathbf{x_d}) = [Y(\mathbf{x}_{d_1}), \ldots, Y(\mathbf{x}_{d_n})]' \in \mathbb{R}^n$, and similarly $c(\mathbf{x_d}, \mathbf{x_d}) \in \mathbb{R}^{n \times n}$

$$\begin{pmatrix} Y(\mathbf{x}^*) \\ Y(\mathbf{x_d}) \end{pmatrix} \sim MVN\left[\begin{pmatrix} \mu(\mathbf{x}^*) \\ \mu(\mathbf{x_d}) \end{pmatrix}, \quad \begin{pmatrix} c(\mathbf{x}^*, \mathbf{x}^*) & c(\mathbf{x}^*, \mathbf{x_d}) \\ c(\mathbf{x_d}, \mathbf{x}^*) & c(\mathbf{x_d}, \mathbf{x_d}) \end{pmatrix}\right]$$

then $Y(\mathbf{x}^*) \mid (Y(\mathbf{x_d}) = \mathbf{y}) \sim N(\mu^*, \Sigma^*)$ where

$$\mu^* = \mu(\mathbf{x}^*) + c(\mathbf{x}^*, \mathbf{x_d}) c(\mathbf{x_d}, \mathbf{x_d})^{-1} (\mathbf{y} - \mu_Y)$$
$$\Sigma^* = c(\mathbf{x}^*, \mathbf{x}^*) - c(\mathbf{x}^*, \mathbf{x_d}) c(\mathbf{x_d}, \mathbf{x_d})^{-1} c(\mathbf{x_d}, \mathbf{x}^*)$$

Conditional Normal Theory

Let $Y(\mathbf{x_d}) = [Y(\mathbf{x}_{d_1}), \ldots, Y(\mathbf{x}_{d_n})]' \in \mathbb{R}^n$, and similarly $c(\mathbf{x_d}, \mathbf{x_d}) \in \mathbb{R}^{n \times n}$

$$\begin{pmatrix} Y(\mathbf{x}^*) \\ Y(\mathbf{x_d}) \end{pmatrix} \sim MVN \left[\begin{pmatrix} \mu(\mathbf{x}^*) \\ \mu(\mathbf{x_d}) \end{pmatrix}, \quad \begin{pmatrix} c(\mathbf{x}^*, \mathbf{x}^*) & c(\mathbf{x}^*, \mathbf{x_d}) \\ c(\mathbf{x_d}, \mathbf{x}^*) & c(\mathbf{x_d}, \mathbf{x_d}) \end{pmatrix} \right]$$

then $Y(\mathbf{x}^*) \mid (Y(\mathbf{x_d}) = \mathbf{y}) \sim N(\mu^*, \Sigma^*)$ where

$$\mu^* = \mu(\mathbf{x}^*) + c(\mathbf{x}^*, \mathbf{x_d}) c(\mathbf{x_d}, \mathbf{x_d})^{-1} (\mathbf{y} - \mu_Y)$$
$$\Sigma^* = c(\mathbf{x}^*, \mathbf{x}^*) - c(\mathbf{x}^*, \mathbf{x_d}) c(\mathbf{x_d}, \mathbf{x_d})^{-1} c(\mathbf{x_d}, \mathbf{x}^*)$$

Important!: The diagonal of Σ^* gives the marginal variance of the predicted points. A Gaussian random variable has 95% probability of falling within $\pm$ 1.96 standard deviations from the mean.

What does this look like?

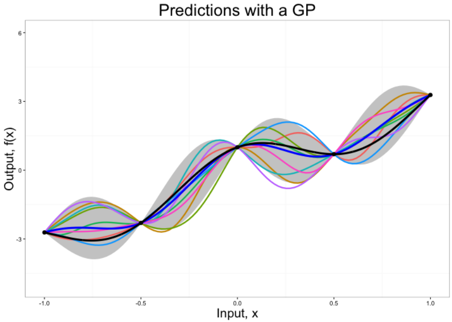


Figure: Realizations of GP conditioned on output at design points (black dots)

This is the same picture as before - the extra lines are just *draws* from the multivariate normal with the conditional mean of the blue line and the conditional covariance matrix as described.

GP Exercises

Assume you want to estimate the (unknown) function $y(x) = 3x + \cos(5x)$, but you only know y at $x = \{-1, -0.5, 0, 0.5, 1\}$

- Find and plot the mean and variance at all points $x^* = \{-1, -0.99, \ldots, 0.99, 1\}$. Compare to the truth
- Draw five possible sample paths using the above mean and variance

Short Recap of Using Gaussian Processes

- Pick a set of design points $\{\mathbf{x}_{d_1}, \ldots, \mathbf{x}_{d_q}\}$, calculate output $\{Y(\mathbf{x}_{d_1}), \ldots, Y(\mathbf{x}_{d_q})\}$

Short Recap of Using Gaussian Processes

- Pick a set of design points $\{\mathbf{x}_{d_1}, \ldots, \mathbf{x}_{d_q}\}$, calculate output $\{Y(\mathbf{x}_{d_1}), \ldots, Y(\mathbf{x}_{d_q})\}$
- Choose mean function $\mu(\cdot)$ and covariance function $c(\cdot, \cdot)$

Short Recap of Using Gaussian Processes

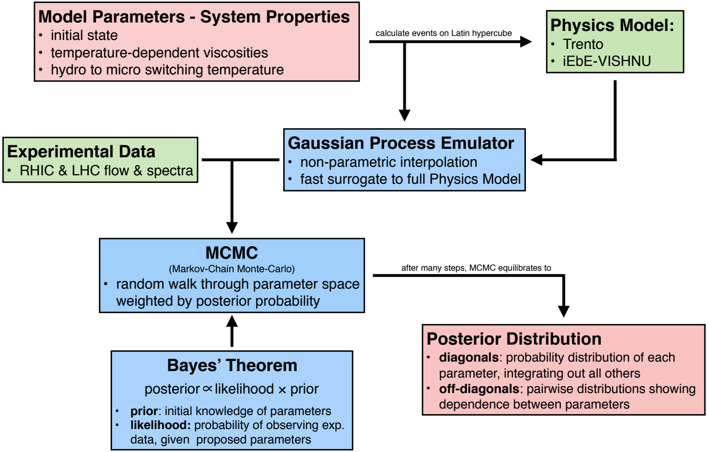
- Pick a set of design points $\{\mathbf{x}_{d_1}, \ldots, \mathbf{x}_{d_q}\}$, calculate output $\{Y(\mathbf{x}_{d_1}), \ldots, Y(\mathbf{x}_{d_q})\}$
- Choose mean function $\mu(\cdot)$ and covariance function $c(\cdot, \cdot)$
- Train the GP on the design points and model output to find appropriate hyperparameters for $c(\cdot, \cdot)$

Short Recap of Using Gaussian Processes

- Pick a set of design points $\{\mathbf{x}_{d_1}, \ldots, \mathbf{x}_{d_q}\}$, calculate output $\{Y(\mathbf{x}_{d_1}), \ldots, Y(\mathbf{x}_{d_q})\}$
- Choose mean function $\mu(\cdot)$ and covariance function $c(\cdot, \cdot)$
- Train the GP on the design points and model output to find appropriate hyperparameters for $c(\cdot, \cdot)$
- For any set of unknown point $\mathbf{x}^*$, find the mean and covariance of $Y(\mathbf{x}^*)$ by following the conditional normal distribution rules

Flowchart of Analysis

Extraction of QGP Properties via a Model-to-Data Analysis



Overview

Common Issue - Multivariate Output

- The above theory works great...assuming your function $Y()$ only has one output

Common Issue - Multivariate Output

- The above theory works great...assuming your function $Y()$ only has one output
- But this is rarely the case - usually have multiple observables for each model run

Common Issue - Multivariate Output

- The above theory works great...assuming your function $Y()$ only has one output
- But this is rarely the case - usually have multiple observables for each model run
- Can't just train independent GPs for each observable, because the observables probably aren't independent

Common Issue - Multivariate Output

- The above theory works great...assuming your function $Y()$ only has one output
- But this is rarely the case - usually have multiple observables for each model run
- Can't just train independent GPs for each observable, because the observables probably aren't independent
- With many observables, probably desire dimension reduction as well

Solution - Principal Components Analysis

- Both problems (dependent columns and high dimensionality) can be solved with PCA

Solution - Principal Components Analysis

- Both problems (dependent columns and high dimensionality) can be solved with PCA
- PCA rotates (centered and scaled) output matrix $\mathbf{Y} \in \mathbb{R}^{n \times p}$ onto an orthogonal space

Solution - Principal Components Analysis

- Both problems (dependent columns and high dimensionality) can be solved with PCA
- PCA rotates (centered and scaled) output matrix $\mathbf{Y} \in \mathbb{R}^{n \times p}$ onto an orthogonal space
  - Because new columns are orthogonal, they are independent (MVN property)

Solution - Principal Components Analysis

- Both problems (dependent columns and high dimensionality) can be solved with PCA
- PCA rotates (centered and scaled) output matrix $\mathbf{Y} \in \mathbb{R}^{n \times p}$ onto an orthogonal space
  - Because new columns are orthogonal, they are independent (MVN property)
- Uses Singular Value Decomposition to find directions of highest variation of data

Solution - Principal Components Analysis

- Both problems (dependent columns and high dimensionality) can be solved with PCA
- PCA rotates (centered and scaled) output matrix $\mathbf{Y} \in \mathbb{R}^{n \times p}$ onto an orthogonal space
  - Because new columns are orthogonal, they are independent (MVN property)
- Uses Singular Value Decomposition to find directions of highest variation of data

$$\mathbf{Y} = \mathbf{USV}' \quad \rightarrow \quad \mathbf{Z} = \mathbf{YV}$$

Solution - Principal Components Analysis

- Both problems (dependent columns and high dimensionality) can be solved with PCA
- PCA rotates (centered and scaled) output matrix $\mathbf{Y} \in \mathbb{R}^{n \times p}$ onto an orthogonal space
  - Because new columns are orthogonal, they are independent (MVN property)
- Uses Singular Value Decomposition to find directions of highest variation of data

$$\mathbf{Y} = \mathbf{USV}' \quad \rightarrow \quad \mathbf{Z} = \mathbf{YV}$$

- Train R emulators on first R columns of $\mathbf{Z}$

PCA - Orthogonality and Dimension Reduction

$$\mathbf{Y} = \mathbf{USV}' \quad \rightarrow \quad \mathbf{Z} = \mathbf{YV}$$

- All of the columns of $\mathbf{Z}$ are orthogonal, and thus independent (if $\mathbf{Z}$ is Multivariate Normal)

PCA - Orthogonality and Dimension Reduction

$$\mathbf{Y} = \mathbf{USV}' \quad \rightarrow \quad \mathbf{Z} = \mathbf{YV}$$

- All of the columns of $\mathbf{Z}$ are orthogonal, and thus independent (if $\mathbf{Z}$ is Multivariate Normal)
- The principal components can also tell us about the percent of variance explained by each each component

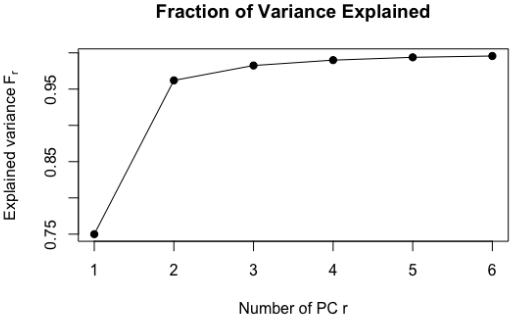
PCA - Orthogonality and Dimension Reduction

$$\mathbf{Y} = \mathbf{USV}' \quad \rightarrow \quad \mathbf{Z} = \mathbf{YV}$$

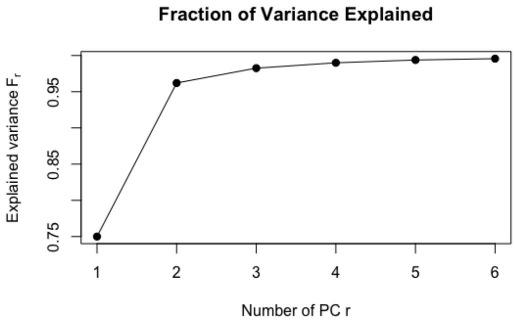
- All of the columns of **Z** are orthogonal, and thus independent (if **Z** is Multivariate Normal)
- The principal components can also tell us about the percent of variance explained by each each component
- Let $\{s_1, \ldots, s_p\} = diag(\mathbf{S})$. Then the fraction of variance explained by the first R columns of **Z** is

$$F_R = \frac{\sum_1^R s^2}{\sum_1^p s^2}$$

Look for the Elbow

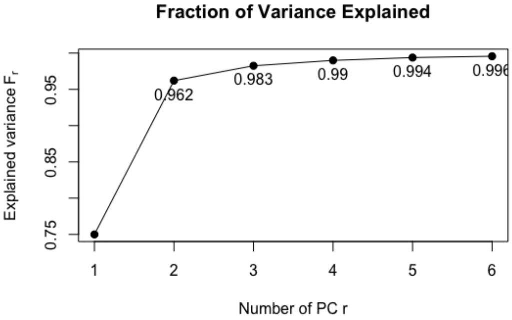


Look for the Elbow



Decide on number of components from "elbow," or some threshold of fraction of variance explained

Look for the Elbow



Decide on number of components from "elbow," or some threshold of fraction of variance explained

PCA Exercise

Load the *dev_indices* dataset

- Visualize fraction of variance explained, and choose a number of PCs
- Plot PC1 vs PC2. What is their correlation?

Overview of Analysis (So Far)

Train the Emulators

- Rotate your output data $\mathbf{Y}$ via PCA into an orthogonal space $\mathbf{Z} = \mathbf{YV}$

Overview of Analysis (So Far)

Train the Emulators

- Rotate your output data $\mathbf{Y}$ via PCA into an orthogonal space $\mathbf{Z} = \mathbf{YV}$
- Train R emulators $\{z_i(\cdot)\}$ on first R columns of $\mathbf{Z}$

Overview of Analysis (So Far)

Train the Emulators

- Rotate your output data $\mathbf{Y}$ via PCA into an orthogonal space $\mathbf{Z} = \mathbf{YV}$
- Train R emulators $\{z_i(\cdot)\}$ on first R columns of $\mathbf{Z}$

Predicting

- For new $\mathbf{x}^*$, predict $z_i(\mathbf{x}^*)$ for each of the R emulators

Overview of Analysis (So Far)

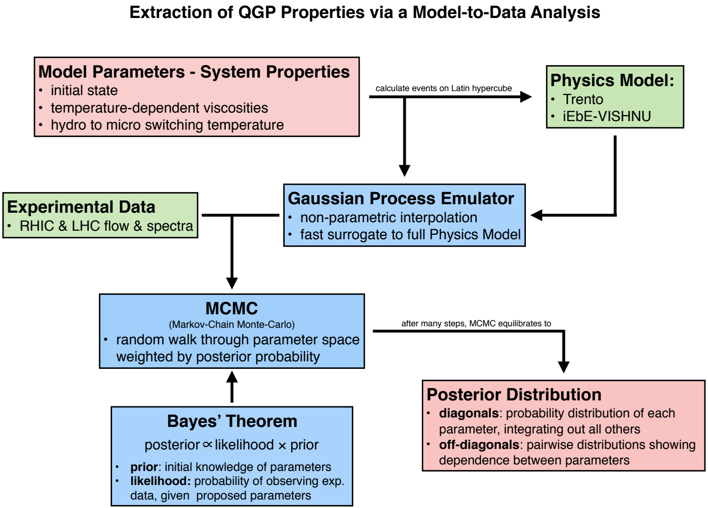
Train the Emulators

- Rotate your output data $\mathbf{Y}$ via PCA into an orthogonal space $\mathbf{Z} = \mathbf{YV}$
- Train R emulators $\{z_i(\cdot)\}$ on first R columns of $\mathbf{Z}$

Predicting

- For new $\mathbf{x}^*$, predict $z_i(\mathbf{x}^*)$ for each of the R emulators
- Let $\mathbf{z}(\mathbf{x}^*) = [z_1(\mathbf{x}^*), \ldots, z_R(\mathbf{x}^*)]$, and rotate to physical space by $\mathbf{y}(\mathbf{x}^*) = \mathbf{z}(\mathbf{x}^*)\mathbf{V}'$

Flowchart of Analysis



Overview

Calibration Setup

Let's get some notation for all the pieces.

Calibration Setup

Let's get some notation for all the pieces.

- y_{exp}: experimental result
- σ_e^2: experimental variance (specified ahead of time)
- $f_M()$: Computer function, calculated at Latin Hypercube design points
- $f_G()$: GP that will serve as surrogate for $f_M()$
- $\boldsymbol{\theta}$: Vector of input parameters to computer model (doesn't change in nature)

Calibration Setup

Let's get some notation for all the pieces.

- y_{exp}: experimental result
- σ_e^2: experimental variance (specified ahead of time)
- $f_M()$: Computer function, calculated at Latin Hypercube design points
- $f_G()$: GP that will serve as surrogate for $f_M()$
- $\boldsymbol{\theta}$: Vector of input parameters to computer model (doesn't change in nature)

Now, a model!

Calibration Setup

Let's get some notation for all the pieces.

- y_{exp}: experimental result
- σ_e^2: experimental variance (specified ahead of time)
- $f_M()$: Computer function, calculated at Latin Hypercube design points
- $f_G()$: GP that will serve as surrogate for $f_M()$
- $\boldsymbol{\theta}$: Vector of input parameters to computer model (doesn't change in nature)

Now, a model!

$$
\begin{aligned}
y_{\text{exp}} &\sim N(f_M(\boldsymbol{\theta}), \sigma_e^2) \\
&\sim N(f_G(\boldsymbol{\theta}), \sigma_e^2) \\
f_G(\boldsymbol{\theta}) &\sim N(\mu^*, \Sigma^*) \\
\boldsymbol{\theta} &\sim \text{Unif}(\theta_{\text{min}}, \theta_{\text{max}})
\end{aligned}
$$

μ^* and Σ^* calculated from conditional multivariate normal rules.

Incorporating PCA

Use PCA for when data has multiple observables:

Incorporating PCA

Use PCA for when data has multiple observables:

Let the computer output $\mathbf{Y} = \mathbf{USV}'$, so $\mathbf{Z} = \mathbf{YV}$ is a matrix of PCs

$$\begin{aligned}
\mathbf{y}_{\text{exp}} &\sim N(f_M(\boldsymbol{\theta}), \Sigma_e) \\
&\sim N(\mathbf{f}_G(\boldsymbol{\theta})\mathbf{V}'_r, \Sigma_e + \Sigma_{\text{extra}}) \\
f_G^{(i)}(\boldsymbol{\theta}) &\sim N(\mu^{(i)*}, \Sigma^{(i)*}) \\
\boldsymbol{\theta} &\sim \text{Unif}(\theta_{\min}, \theta_{\max})
\end{aligned}$$

Incorporating PCA

Use PCA for when data has multiple observables:

Let the computer output $\mathbf{Y} = \mathbf{USV}'$, so $\mathbf{Z} = \mathbf{YV}$ is a matrix of PCs

$$\begin{aligned}
\mathbf{y}_{\text{exp}} &\sim N(f_M(\boldsymbol{\theta}), \Sigma_e) \\
&\sim N(\mathbf{f}_G(\boldsymbol{\theta})\mathbf{V}_r', \Sigma_e + \Sigma_{\text{extra}}) \\
f_G^{(i)}(\boldsymbol{\theta}) &\sim N(\mu^{(i)*}, \Sigma^{(i)*}) \\
\boldsymbol{\theta} &\sim \text{Unif}(\theta_{\text{min}}, \theta_{\text{max}})
\end{aligned}$$

Here $f_G^{(i)}$ is the ith GP trained on the ith column of $\mathbf{Z}$.
Σ_{extra} is extra variation lost when transforming back and forth from PCA (see Appendix)

Recap of Whole Analysis

1. Pick design points via a Latin Hypercube, run the computer model at those design points.

Recap of Whole Analysis

1. Pick design points via a Latin Hypercube, run the computer model at those design points.
2. Transform the computer output via PCA, pick R principal components.

Recap of Whole Analysis

1. Pick design points via a Latin Hypercube, run the computer model at those design points.
2. Transform the computer output via PCA, pick R principal components.
3. Pick a covariance function, and train R independent GPs on the first R columns of the PCA-transformed computer model output.

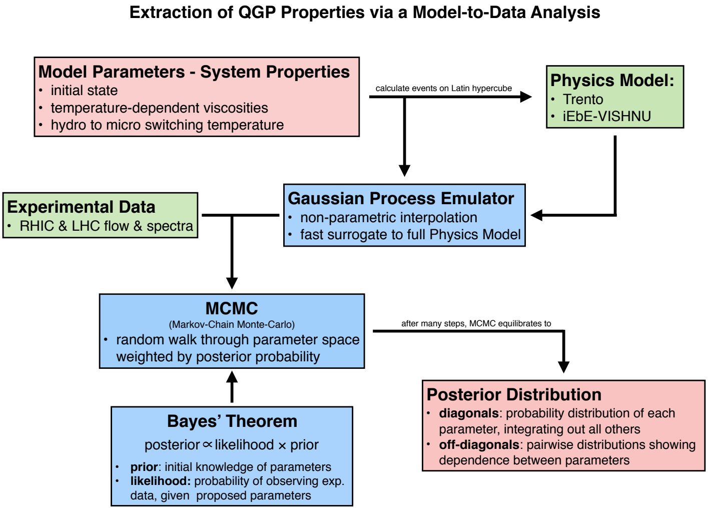
Recap of Whole Analysis

1. Pick design points via a Latin Hypercube, run the computer model at those design points.
2. Transform the computer output via PCA, pick R principal components.
3. Pick a covariance function, and train R independent GPs on the first R columns of the PCA-transformed computer model output.
4. Perform calibration, getting posterior draws for input parameters.

Recap of Whole Analysis

1. Pick design points via a Latin Hypercube, run the computer model at those design points.
2. Transform the computer output via PCA, pick R principal components.
3. Pick a covariance function, and train R independent GPs on the first R columns of the PCA-transformed computer model output.
4. Perform calibration, getting posterior draws for input parameters.
   - For each θ draw, find the GP predictions, transform them back from PCA, then put those values in the likelihood.

Flowchart of Analysis



A toy-model example

A toy-model of jet-quenching

- Initial spectrum

$$\frac{dN_0}{dp_T} \propto \frac{p_T}{\left(3^2 + p_T^2\right)^3}$$

- Energy loss ΔE follows a Γ-distribution,

$$\begin{aligned} P(\Delta E) &\propto \Delta E^{\mu^2/\sigma^2 - 1} e^{-\mu \Delta E/\sigma^2} \\ \mu &= A\sqrt{p_T}, \sigma = B\mu \end{aligned}$$

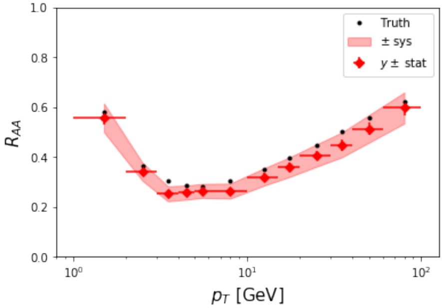
- The quenched spectrum

$$\frac{dN_1}{dp_T}(p_T) = \int \frac{dN_0}{dp_T}(p_T + x) P(x) dx$$

A toy-model example

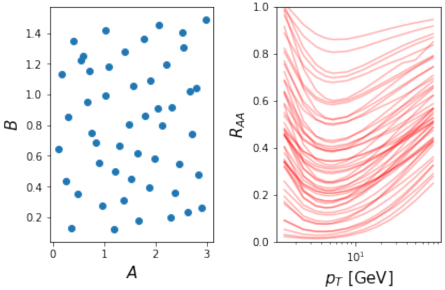
- Assume the model is perfect and the truths are: $A = 1.0, B = 0.5$.
- A measurement with finite statistics (5%) and systematic bias (10%).

$$y_{\rm exp} \approx y_{\rm true} \pm \sigma_{\rm stat} \pm \sigma_{\rm sys}$$



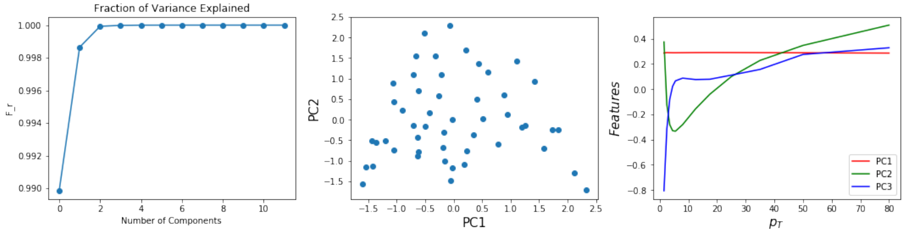
Toy model: make design

An 50-point design with $A \in [0.05, 3]$ and $B \in [0.05, 1.5]$.
Model calculations of R_{AA} widely spread between 0 and 1.



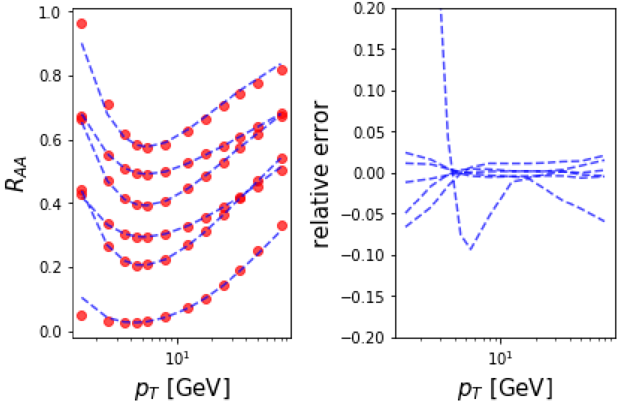
Toy model: PCA

The first two PCs account for more than 99.8% of the data variance.
PC1 is an overall shift, PC2 and PC3 capture the shapes.



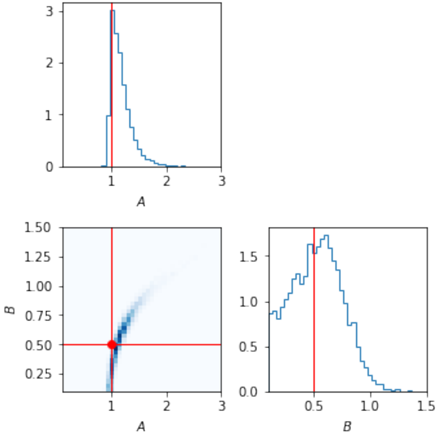
Toy model: Validation

Compare GPs' predictions to model calculations at new parameter sets. Relative error on the right.

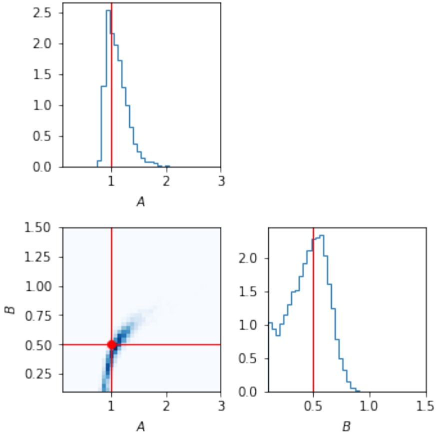


Toy model: Posterior for parameters

Using uncorrelated sys error



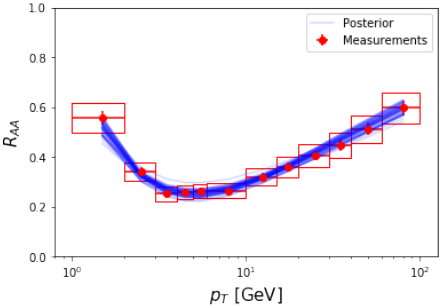
Using correlated sys error



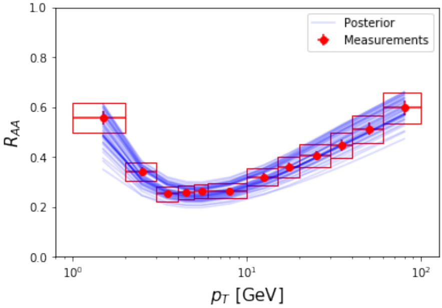
* Mistreating correlated uncertainty may bias the credible region.

Toy model: Posterior predictions

Using uncorrelated sys error



Using correlated sys error



* Mistreating correlated uncertainty may lead to overconfident prediction uncertainties.

Some References

- For more information on Gaussian Processes, see [Rasmussen and Williams, 2006]. The full book is available online.
- For more details on GP Emulation and Calibration, see [Bayarri et al., 2007] and [Higdon et al., 2008].
  - The former describes the same process in this talk of separating training the GPs and performing calibration (called *modularization*).
  - The latter describes the use of PCA in calibration.
  - Both resources describe modeling a *discrepancy function* as a way to capture the systematic departure of the computer model from the experimental data. Our model neglects this discrepancy because we assume no input parameters that varies in both nature and model.

Works Cited: I

Bayarri, M., Berger, J., Paulo, R., and Sacks, J. (2007).
A framework for validation of computer models.
Technometrics, 49(2):138–154.

Higdon, D., Gattiker, J., Williams, B., and Rightley, M. (2008).
Computer model calibration using high dimensional output.
Journal of the American Statistical Association, 103(482):570–583.

Rasmussen, C. and Williams, C. (2006).
Gaussian Processes for Machine Learning.
MIT Press.

Appendix - Conditional Multivariate Normal Theory

Let $\mathbf{Z} \in R^{n_z}$ and $\mathbf{Y} \in R^{n_y}$ be multivariate normal, with joint density

$$\begin{pmatrix} \mathbf{Z} \\ \mathbf{Y} \end{pmatrix} \sim MVN \left[\begin{pmatrix} \mu_Z \\ \mu_Y \end{pmatrix}, \quad \begin{pmatrix} \Sigma_{ZZ} & \Sigma_{ZY} \\ \Sigma_{YZ} & \Sigma_{YY} \end{pmatrix} \right]$$

- Remember, $\Sigma_{ZY} \neq 0 \Leftrightarrow \mathbf{Z}, \mathbf{Y}$ not independent
- I.e., if we know something about $\mathbf{Y}$, we should have more information about $\mathbf{Z}$, and vice versa
- In fact, if we know the true value of $\mathbf{Y}$ (say its known value is $\mathbf{y}$), it turns out the **conditional distribution** of $\mathbf{Z} \mid (\mathbf{Y} = \mathbf{y})$ is also multivariate normal (with adjusted mean and covariance)
  - This is somewhat special to multivariate normals

Appendix - Conditional Multivariate Normal Theory

Let $\mathbf{Z} \in R^{n_z}$ and $\mathbf{Y} \in R^{n_y}$ be multivariate normal, with joint density

$$\begin{pmatrix} \mathbf{Z} \\ \mathbf{Y} \end{pmatrix} \sim MVN\left[\begin{pmatrix} \boldsymbol{\mu}_Z \\ \boldsymbol{\mu}_Y \end{pmatrix}, \begin{pmatrix} \Sigma_{ZZ} & \Sigma_{ZY} \\ \Sigma_{YZ} & \Sigma_{YY} \end{pmatrix}\right]$$

then $\mathbf{Z} \mid (\mathbf{Y} = \mathbf{y}) \sim MVN(\boldsymbol{\mu}_{Z|Y}, \Sigma_{Z|Y})$ where

$$\boldsymbol{\mu}_{Z|Y} = \boldsymbol{\mu}_Z + \Sigma_{ZY}\Sigma_{YY}{}^{-1}(\mathbf{y} - \boldsymbol{\mu}_Y)$$
$$\Sigma_{Z|Y} = \Sigma_{ZZ} - \Sigma_{ZY}\Sigma_{YY}{}^{-1}\Sigma_{YZ}$$

Appendix - Conditional Multivariate Normal Theory

Let $\mathbf{Z} \in R^{n_z}$ and $\mathbf{Y} \in R^{n_y}$ be multivariate normal, with joint density

$$\begin{pmatrix} \mathbf{Z} \\ \mathbf{Y} \end{pmatrix} \sim MVN\left[\begin{pmatrix} \mu_Z \\ \mu_Y \end{pmatrix}, \begin{pmatrix} \Sigma_{ZZ} & \Sigma_{ZY} \\ \Sigma_{YZ} & \Sigma_{YY} \end{pmatrix}\right]$$

then $\mathbf{Z} \mid (\mathbf{Y} = \mathbf{y}) \sim MVN(\mu_{Z|Y}, \Sigma_{Z|Y})$ where

$$\mu_{Z|Y} = \mu_Z + \Sigma_{ZY}\Sigma_{YY}{}^{-1}(\mathbf{y} - \mu_Y)$$
$$\Sigma_{Z|Y} = \Sigma_{ZZ} - \Sigma_{ZY}\Sigma_{YY}{}^{-1}\Sigma_{YZ}$$

Appendix - Conditional Multivariate Normal Theory

Let $\mathbf{Z} \in R^{n_z}$ and $\mathbf{Y} \in R^{n_y}$ be multivariate normal, with joint density

$$\begin{pmatrix} \mathbf{Z} \\ \mathbf{Y} \end{pmatrix} \sim MVN \left[\begin{pmatrix} \mu_Z \\ \mu_Y \end{pmatrix}, \quad \begin{pmatrix} \Sigma_{ZZ} & \Sigma_{ZY} \\ \Sigma_{YZ} & \Sigma_{YY} \end{pmatrix} \right]$$

then $\mathbf{Z} \mid (\mathbf{Y} = \mathbf{y}) \sim MVN(\mu_{Z|Y}, \Sigma_{Z|Y})$ where

$$\mu_{Z|Y} = \mu_Z + \Sigma_{ZY} \Sigma_{YY}{}^{-1} (\mathbf{y} - \mu_Y)$$
$$\Sigma_{Z|Y} = \Sigma_{ZZ} - \Sigma_{ZY} \Sigma_{YY}{}^{-1} \Sigma_{YZ}$$

The punchline - if we know that the joint distribution of **Z** and **Y** is multivariate normal, it's really easy to draw the conditional distribution of **Z** given **Y**

Appendix - Conditional Multivariate Normal Theory

Apply the above theory to Computer Emulation

- Let $\mathbf{D} = \{\mathbf{x}\}$ be the **design** points in $\mathcal{X}$ for which we know $Y(\mathbf{x})$, of length p_D
- Let $\mathbf{U} = \{\mathbf{x}\}$ be the points in $\mathcal{X}$ for which $Y(\mathbf{x})$ is **unknown**, of length p_U
- Let $\mu(\mathbf{D})$ be the vector where $\mu(\cdot)$ is applied to each $\mathbf{x} \in \mathbf{D}$, and $\mu(\mathbf{U})$ similar
- Let $c(\mathbf{D}, \mathbf{U})$ be the matrix where $c(\{\mathbf{x}_i\}, \{\mathbf{x}_j\})$ is applied for each $\mathbf{x}_i \in \mathbf{D}$ and $\mathbf{x}_j \in \mathbf{U}$.
  - So $c(\mathbf{D}, \mathbf{U}) \in \mathbb{R}^{p_D \times p_U}$

$$\begin{pmatrix} Y(\mathbf{U}) \\ Y(\mathbf{D}) \end{pmatrix} \sim MVN \left[\begin{pmatrix} \mu(\mathbf{U}) \\ \mu(\mathbf{D}) \end{pmatrix}, \quad \begin{pmatrix} c(\mathbf{U}, \mathbf{U}) & c(\mathbf{U}, \mathbf{D}) \\ c(\mathbf{D}, \mathbf{U}) & c(\mathbf{D}, \mathbf{D}) \end{pmatrix} \right]$$

So we can estimate (with uncertainty!) $Y(\mathbf{U})$ conditioned on $Y(\mathbf{D})$ based solely conditional normal theory!

Appendix - Quick Intro to MCMC

MCMC stands for Markov Chain Monte Carlo

- We have a parameter θ that we want to learn things about (its mean, variance, etc.). If we knew the distribution of θ (say $\pi(\theta)$), we could just make a bunch of draws from that distribution, and look at the mean and variance of the draws.
    - Imagine you have a weighted coin, but you don't know the probability of heads. You could just flip the coin 1,000 times and average the number of heads to get an estimate.
    - This is the "Monte Carlo" portion - the output is random but still helps us learn about the parameter
- Often the distribution we care about is super complicated and/or high dimensional, so it's not easy to make draws from it.
    - Instead of drawing directly from $\pi(\theta)$, we use algorithms to draw a chain of $\theta^{(t)}$ that theory tells us will converge to draws from $\pi(\theta)$
    - This is the "Markov Chain" part - the draws $\theta^{(t)}$ are a chain that converge in distribution to what we care about

Appendix - Covariance Matrix Details

The specification of the experimental covariance matrix Σ_e is important for calibration. It is given in the model rather than learned.

- The Python distribution uses a block-diagonal construction, with a block for each observable.
- It also assumes the observables are indexed by some continuous variable - in our example, this is transverse momentum p_T.
  - i.e., there is a value of each observable for each p_T

$$\Sigma^{(k)} = \Sigma^{(k)}_{\text{sys}} + \Sigma^{(k)}_{\text{stat}}$$

$$\Sigma^{(k)}_{\text{stat}} = \sigma^{\text{stat}}_{i,k}\sigma^{\text{stat}}_{j,k}\delta_{ij}$$

$$\Sigma^{(k)}_{\text{sys}} = \sigma^{\text{sys}}_{i,k}\sigma^{\text{sys}}_{j,k}\exp\left[-\left(\frac{p_{i,k}-p_{j,k}}{\ell_k}\right)^2\right]$$

$$\Sigma^{(k)} = \Sigma^{(k)}_{\text{sys}} + \Sigma^{(k)}_{\text{stat}}$$
$$\Sigma^{(k)}_{\text{stat}} = \sigma^{\text{stat}}_{i,k}\sigma^{\text{stat}}_{j,k}\delta_{ij}$$
$$\Sigma^{(k)}_{\text{sys}} = \sigma^{\text{sys}}_{i,k}\sigma^{\text{sys}}_{j,k}\exp\left[-\left(\frac{p_{i,k}-p_{j,k}}{\ell_k}\right)^2\right]$$

- $\sigma^{\text{sys}}_{i,k}$ is the systematic error for the ith value of the kth observable
- $\sigma^{\text{stat}}_{i,k}$ is the statical error for the ith value of the kth observable
- $\Sigma^{(k)}_{\text{stat}}$ as above is diagonal
- $p_{i,k}$ is the ith transverse momentum of the kth observable
- $\Sigma^{(k)}_{\text{sys}}$ is scaled on the off-diagonal by a correlation function applied to the distance between the p_T values.
- ℓ_k is estimated via MLE

Appendix - Extra Variation from PCA

Let $Y = USV'$ be the SVD of Y, with V_r the matrix of the first r columns of V so that $\hat{Y} = YV_rV_r'$. Let V_b be a matrix consisting of the columns of V from $r+1$ to p (and similarly for S_b). We note that

$$VS^2V' = V_rS_r^2V_r' + V_bS_b^2V_b$$

Thus, the sample covariance matrix of $\hat{Y}$ is

$$\frac{1}{m}\hat{Y}'\hat{Y} = \frac{1}{m}V_rS_r^2V_r'$$

Thus, if Σ_Y is the sample covariance matrix of Y and $\widehat{\Sigma}_{\hat{Y}}$ is the sample covariance matrix of $\hat{Y}$, then

$$\Sigma_Y = \widehat{\Sigma}_{\hat{Y}} + \frac{1}{m}V_bS_b^2V_b$$

so that $\Sigma_{\text{extra}} = \frac{1}{m}V_bS_b^2V_b$ is the extra covariance lost when transforming from Y to $Z_r = YV_r$ then back to $\hat{Y}$.