Calibration and Prediction with Gaussian Process Emulators

J. Coleman, R. Wolpert, S. Bass

January 4, 2018

Motivating Gaussian Process Emulation and Calibration

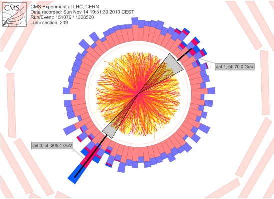
- Scientists want to learn about some physical system, but data are really hard to collect
- Experimentation is costly (money, time, etc.)

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

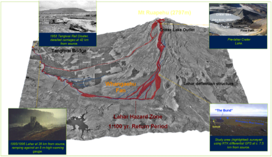
[2] http://www.massey.ac.nz/massey/fms/Rivers/ruapehu-lahars-2d-modelling.png

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.)


CMS
CMS Experiment at LHC, CERN
Data recorded: Sun Nov 14 19:31:39 2010 CEST
Run/Event: 151076 / 1328520
Lumi section: 249
Jet 1, pt: 70.0 GeV
Jet 0, pt: 205.1 GeV

[1]


Mt Ruapehu (2797m)
1953 Tangiwai Rail Disater, derailed carriages at 42 km from source.
Tangiwai Bridge
Crater Lake Outlet
Crater lake
Outlet
Flow Path
Pre-lahar Crater Lake.
Whangaehu Fan
Lahar deflection structure
Lahar Hazard Zone
1:100 yr. Return Period
1995/1996 Lahar at 28 km from source, ramping against an 8 m-high warning gauge.
"The Bund"
Vehicle
Study area (highlighted) surveyed using RTK differential GPS at c. 7.5 km from source.

[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 → 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 → 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

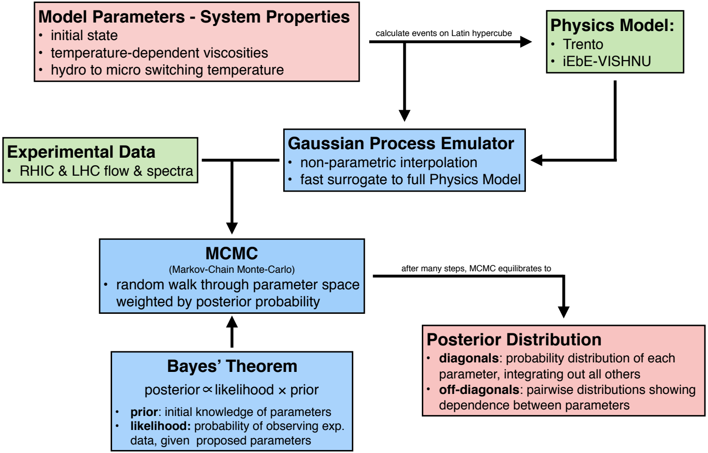
- Statisticians use GP as black box that says “close in input → close in output”
- “Emulator” of computationally expensive computer model - interpolation with uncertainty

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

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



Model Parameters - System Properties
- initial state
- temperature-dependent viscosities
- hydro to micro switching temperature

calculate events on Latin hypercube

Physics Model:
- Trento
- iEbE-VISHNU

Gaussian Process Emulator
- non-parametric interpolation
- fast surrogate to full Physics Model

Experimental Data
- RHIC & LHC flow & spectra

MCMC

(Markov-Chain Monte-Carlo)
- random walk through parameter space weighted by posterior probability

after many steps, MCMC equilibrates to

Posterior Distribution
- **diagonals**: probability distribution of each parameter, integrating out all others
- **off-diagonals**: pairwise distributions showing dependence between parameters

Bayes' Theorem

posterior $\propto$ likelihood $\times$ prior
- **prior**: initial knowledge of parameters
- **likelihood:** probability of observing exp. data, given proposed parameters

Overview of Analysis

Designing the Training Points - Latin Hypercube

Training and Validating GP Emulators

- GP Basics
- Multivariate Output - PCA

Calibration

- Intro to Bayesian Analysis
- Emulation Context

Overview

Designing the Training Points - Latin Hypercube

Training and Validating GP Emulators

GP Basics

Multivariate Output - PCA

Calibration

Intro to Bayesian Analysis

Emulation Context

Why is Design Important?

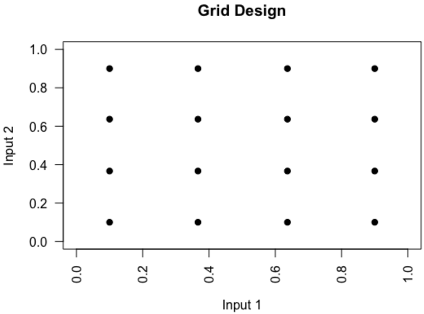
The **design points** are the points in the input parameter space at which the scientists run the expensive computer model.

- 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?

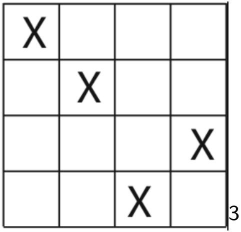
The **design points** are the points in the input parameter space at which the scientists run the expensive computer model.

- 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



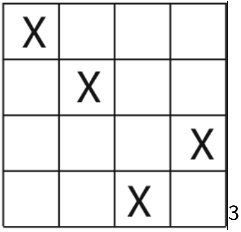
X			
	X		
			X
		X	

[3]

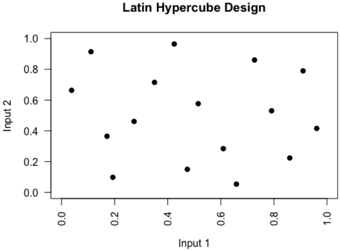
[3] https://upload.wikimedia.org/wikipedia/commons/f/fb/LHSsampling.png

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



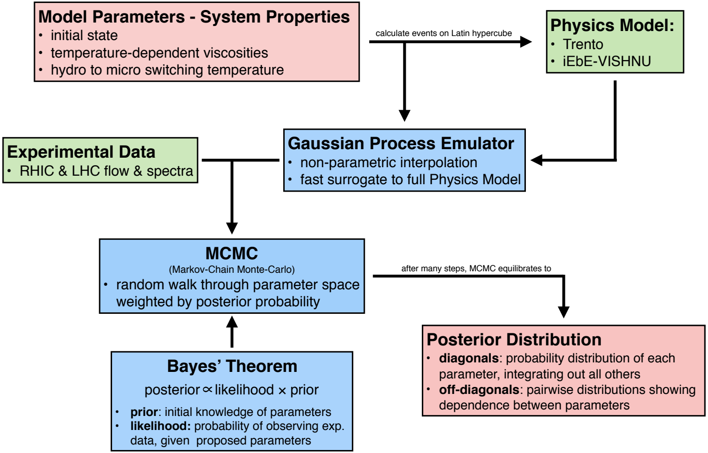
[3]



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

Flowchart of Analysis

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



Model Parameters - System Properties
- initial state
- temperature-dependent viscosities
- hydro to micro switching temperature

calculate events on Latin hypercube

Physics Model:
- Trento
- iEbE-VISHNU

Experimental Data
- RHIC & LHC flow & spectra

Gaussian Process Emulator
- non-parametric interpolation
- fast surrogate to full Physics Model

MCMC
(Markov-Chain Monte-Carlo)
- random walk through parameter space weighted by posterior probability

after many steps, MCMC equilibrates to

Posterior Distribution
- **diagonals**: probability distribution of each parameter, integrating out all others
- **off-diagonals**: pairwise distributions showing dependence between parameters

Bayes' Theorem

posterior ∝ likelihood × prior

- **prior**: initial knowledge of parameters
- **likelihood:** probability of observing exp. data, given proposed parameters

Overview

Designing the Training Points - Latin Hypercube

Training and Validating GP Emulators

- GP Basics
- Multivariate Output - PCA

Calibration

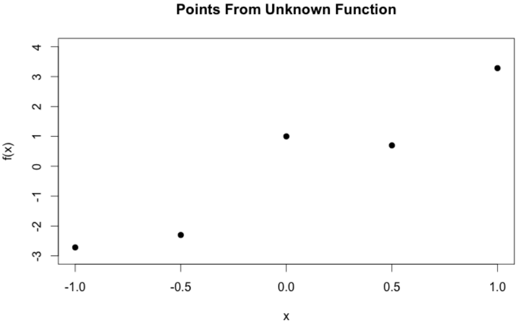
- Intro to Bayesian Analysis
- Emulation Context

Motivating Example

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

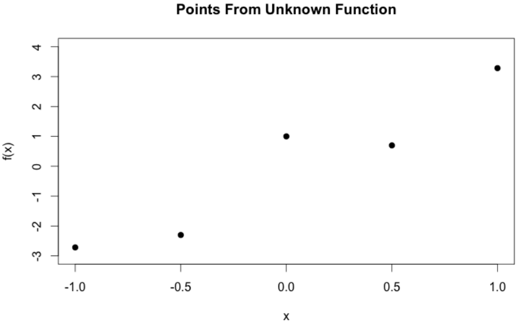
Motivating Example

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.



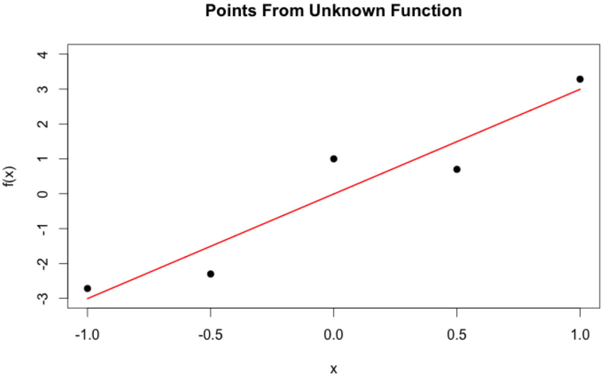
What could we use to predict new points?

Motivating Example

Looks kind of linear...?

Motivating Example

Looks kind of linear...?

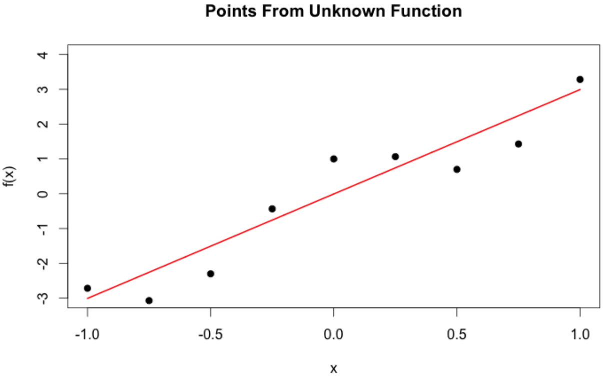


Motivating Example

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

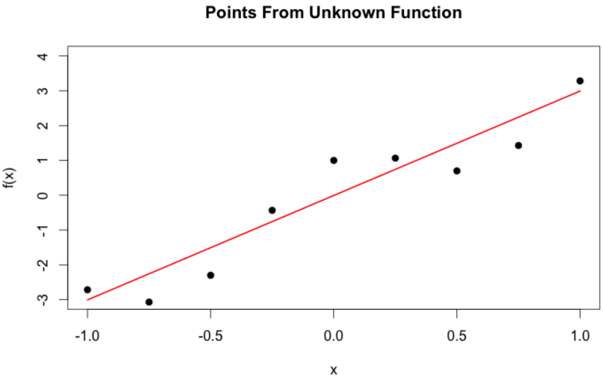
Motivating Example

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



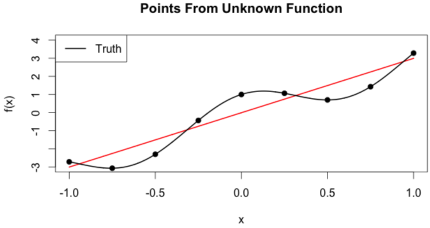
Motivating Example

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



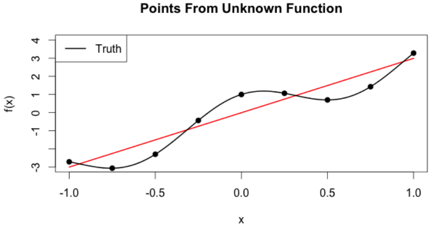
Clearly need something more flexible

Motivating Example



We want a method for interpolating that:

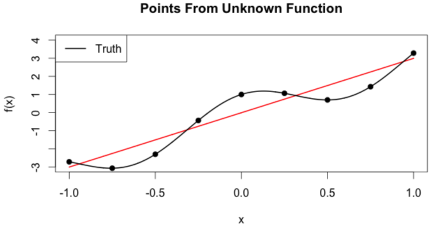
Motivating Example



We want a method for interpolating that:

- Is flexible for any shape

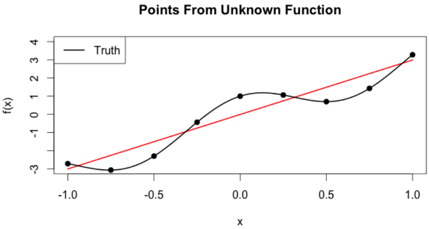
Motivating Example



We want a method for interpolating that:

- Is flexible for any shape
- Offers plausible uncertainty values

Motivating Example

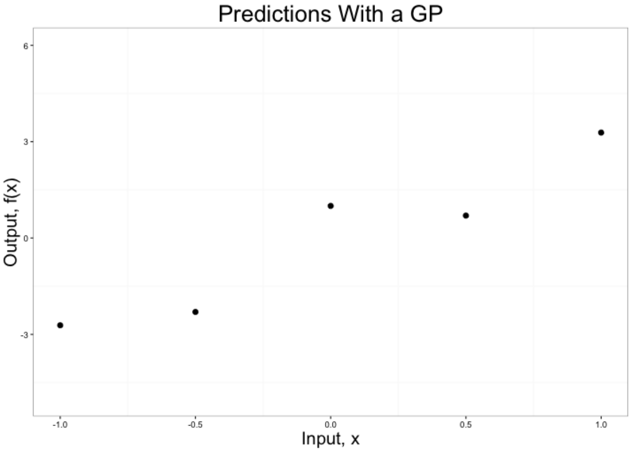


We want a method for interpolating that:

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

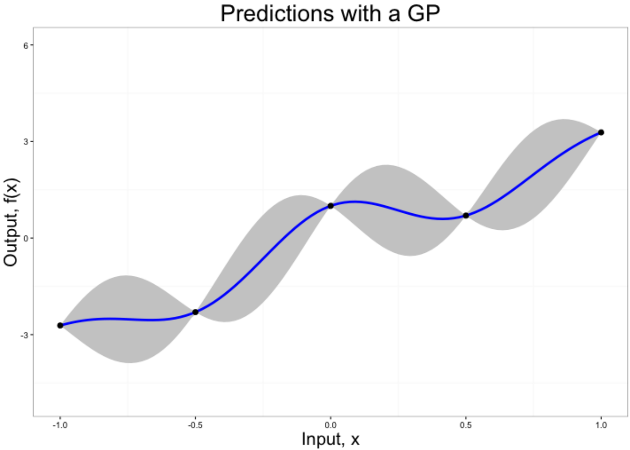
GPs In Action

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



GPs In Action

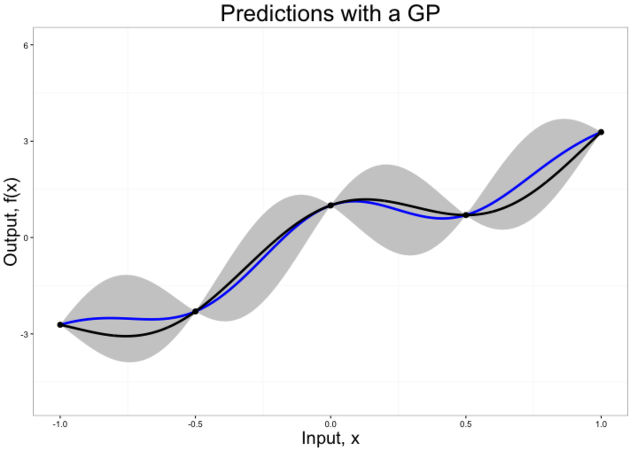
Prediction = mean + uncertainty



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

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

A Concrete example

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

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

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

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

- 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

Example Covariance Functions

The covariance function $c(\cdot,\cdot)$ is often of the form $c(\mathbf{x},\mathbf{x}') = \lambda^{-1} r(\mathbf{x}-\mathbf{x}' \mid \alpha, \ell)$. Examples of $r(\cdot \mid \alpha, \ell)$:

- Power Exponential: $r(h \mid \alpha, \ell) = e^{-|h/\ell|^{\alpha}}$, where $\alpha \in [1,2]$
    - Usually learn ℓ and fix α. Setting $\alpha = 2$ makes the function infinitely differentiable - maybe undesirable.
    - Sometimes set $\alpha = 1.9$ for computational stability
- Matérn: $r(h \mid \alpha, \ell) = \frac{2^{1-\alpha}}{\Gamma(\alpha)} \left(\frac{h}{\ell}\right)^{\alpha} K_{\alpha}\left(\frac{h}{\ell}\right)$, where K_{α} is the modified Bessel function of the second kind
    - For $\alpha = n/2$ for $n \in \mathbb{N}$, this has closed form. Most common are $\alpha = 3/2$ and $\alpha = 5/2$
        - $\alpha = 3/2 : r(h \mid \ell) = e^{-h/\ell}\left(1 + \frac{h}{\ell}\right)$
        - $\alpha = 5/2 : r(h \mid \ell) = e^{-h/\ell}\left(1 + \frac{h}{\ell} + \frac{h^2}{3\ell^2}\right)$

Usually assume **separable** covariance function. That is, if $\mathbf{x}$ has J dimensions, then $r(\mathbf{x}-\mathbf{x}' \mid \alpha_j, \ell_j)) = \prod_{j=1}^{J} r_j(x_j - x_j' \mid \alpha_j, \ell_j))$

What does this look like unconstrained?

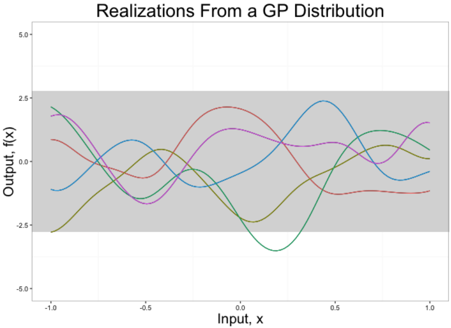


Figure: Unconstrained realizations from a mean-zero GP distribution.

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

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

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

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

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

The punchline - if we know that the joint multivariate Gaussian distribution of $Y(\mathbf{x}^*)$ and $Y(\mathbf{x_d})$, it's really easy to draw the conditional distribution of $Y(\mathbf{x}^*)$ given $Y(\mathbf{x_d})$

What does this look like?

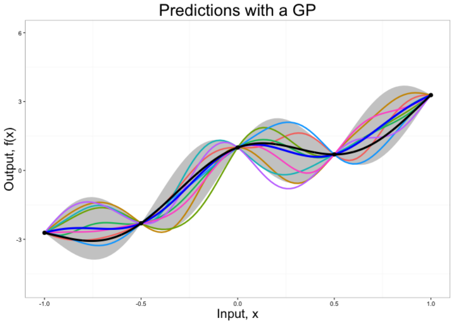


Figure: Realizations of GP conditioned on output at design points (black dots)

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.

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

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 $\mathbf{Z}$

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)

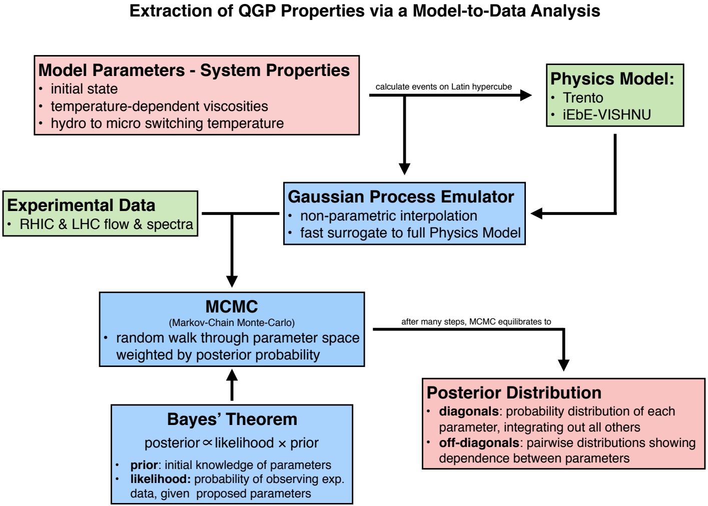
Train the Emulators

- 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



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

Model Parameters - System Properties
- initial state
- temperature-dependent viscosities
- hydro to micro switching temperature

calculate events on Latin hypercube

Physics Model:
- Trento
- iEbE-VISHNU

Gaussian Process Emulator
- non-parametric interpolation
- fast surrogate to full Physics Model

Experimental Data
- RHIC & LHC flow & spectra

MCMC
(Markov-Chain Monte-Carlo)
- random walk through parameter space weighted by posterior probability

after many steps, MCMC equilibrates to

Posterior Distribution
- **diagonals**: probability distribution of each parameter, integrating out all others
- **off-diagonals**: pairwise distributions showing dependence between parameters

Bayes' Theorem

posterior ∝ likelihood × prior

- **prior**: initial knowledge of parameters
- **likelihood:** probability of observing exp. data, given proposed parameters

Overview

Designing the Training Points - Latin Hypercube

Training and Validating GP Emulators

- GP Basics
- Multivariate Output - PCA

Calibration

- Intro to Bayesian Analysis
- Emulation Context

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.

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*:

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

Why Do Bayesian Analysis?

- Tractable inference for very complicated models

Why Do Bayesian Analysis?

- Tractable inference for very complicated models
  - If you can write down a likelihood and prior, you're good!

Why Do Bayesian Analysis?

- Tractable inference for very complicated models
    - If you can write down a likelihood and prior, you're good!
- Can incorporate expert information

Why Do Bayesian Analysis?

- Tractable inference for very complicated models
    - If you can write down a likelihood and prior, you're good!
- Can incorporate expert information

Downsides?

Why Do Bayesian Analysis?

- Tractable inference for very complicated models
  - If you can write down a likelihood and prior, you're good!
- Can incorporate expert information

Downsides?

- Often posterior is not analytically available or a known distribution, so we have to resort to sampling methods

Why Do Bayesian Analysis?

- Tractable inference for very complicated models
  - If you can write down a likelihood and prior, you're good!
- Can incorporate expert information

Downsides?

- Often posterior is not analytically available or a known distribution, so we have to resort to sampling methods
- Sampling schemes can be more computationally intensive than non-Bayesian methods

Why Do Bayesian Analysis?

- Tractable inference for very complicated models
    - If you can write down a likelihood and prior, you're good!
- Can incorporate expert information

Downsides?

- Often posterior is not analytically available or a known distribution, so we have to resort to sampling methods
- Sampling schemes can be more computationally intensive than non-Bayesian methods
- Most common is Markov Chain Monte Carlo (MCMC)
    - Basic idea is to chain together a bunch of samples in a specific way such that they eventually will be draws from the posterior

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

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

Incorporating PCA

Use PCA for when data has multiple observables:

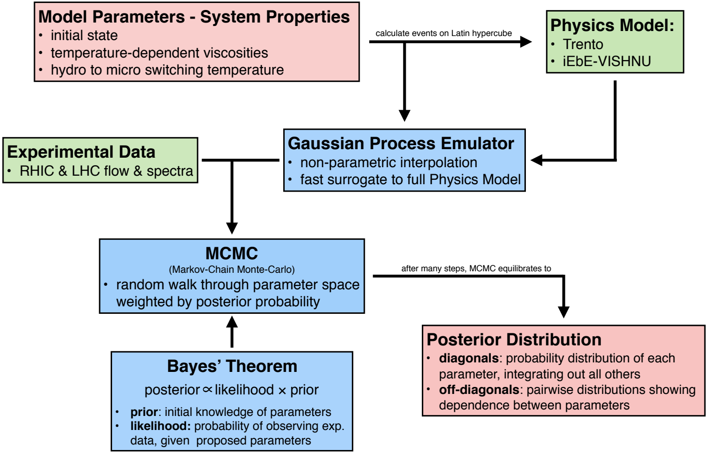
Let the computer output $\mathbf{Y} = \mathbf{USV}'$, so $\mathbf{Z} = \mathbf{YV}$ is a matrix of PCs

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

Here $f_G^{(i)}$ is the ith GP trained on the ith column of $\mathbf{Z}$.

Flowchart of Analysis

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



Model Parameters - System Properties

- initial state
- temperature-dependent viscosities
- hydro to micro switching temperature

calculate events on Latin hypercube

Physics Model:

- Trento
- iEbE-VISHNU

Experimental Data

- RHIC & LHC flow & spectra

Gaussian Process Emulator

- non-parametric interpolation
- fast surrogate to full Physics Model

MCMC

(Markov-Chain Monte-Carlo)

- random walk through parameter space weighted by posterior probability

after many steps, MCMC equilibrates to

Bayes' Theorem

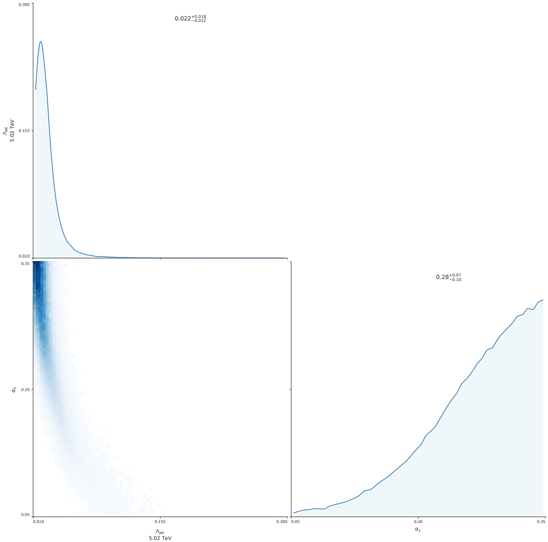
posterior ∝ likelihood × prior

- **prior**: initial knowledge of parameters
- **likelihood:** probability of observing exp. data, given proposed parameters

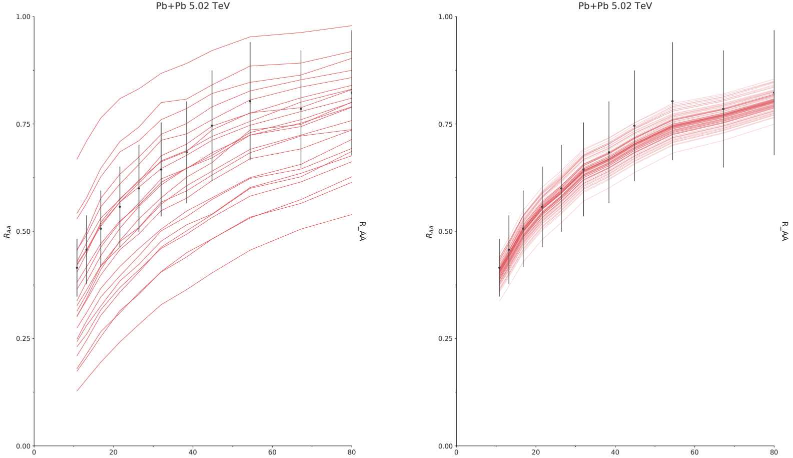
Posterior Distribution

- **diagonals**: probability distribution of each parameter, integrating out all others
- **off-diagonals**: pairwise distributions showing dependence between parameters

Calibration Results Example - Posterior Draws



Calibration Results Example - Model Output Comparison



(a) Design

(b) Posterior

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.

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

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

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

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

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

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

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

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

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

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

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

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

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

- $\sigma^{\text{sys}}_{i,k}$ 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