

# **Gaussian Mixture Models, Active Subspaces, Solving Dynamic Models on high-dimensional, irregularly-shaped state spaces**

Simon Scheidegger  
simon.scheidegger@unil.ch

Rochester, November 20<sup>th</sup>, 2019

# Today's Roadmap

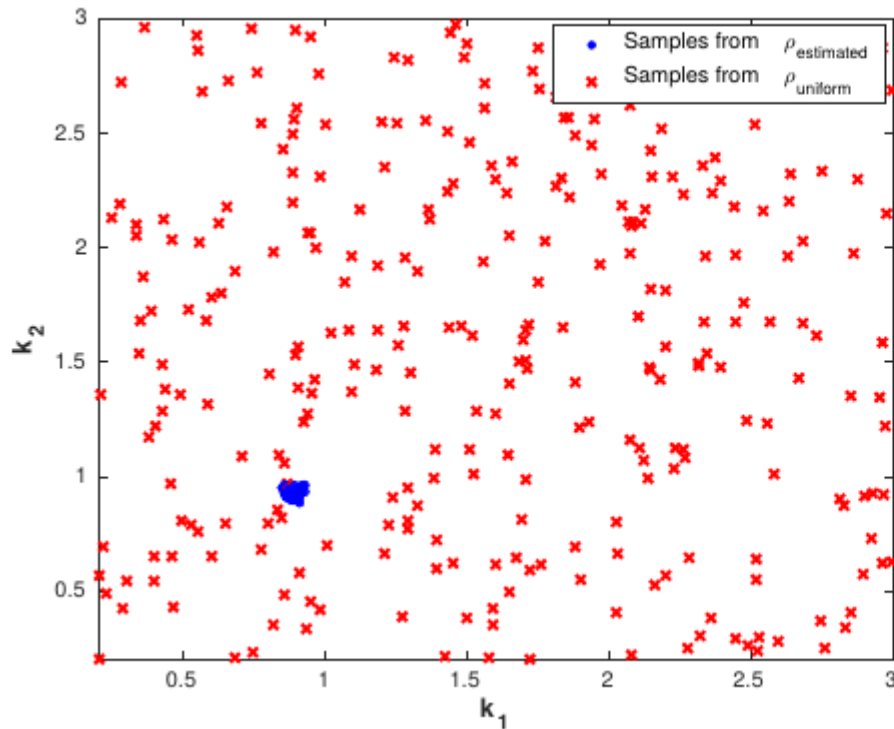
## I. Gaussian Mixture Models

- The basic idea
- The Expectation Maximization Algorithm (EM)

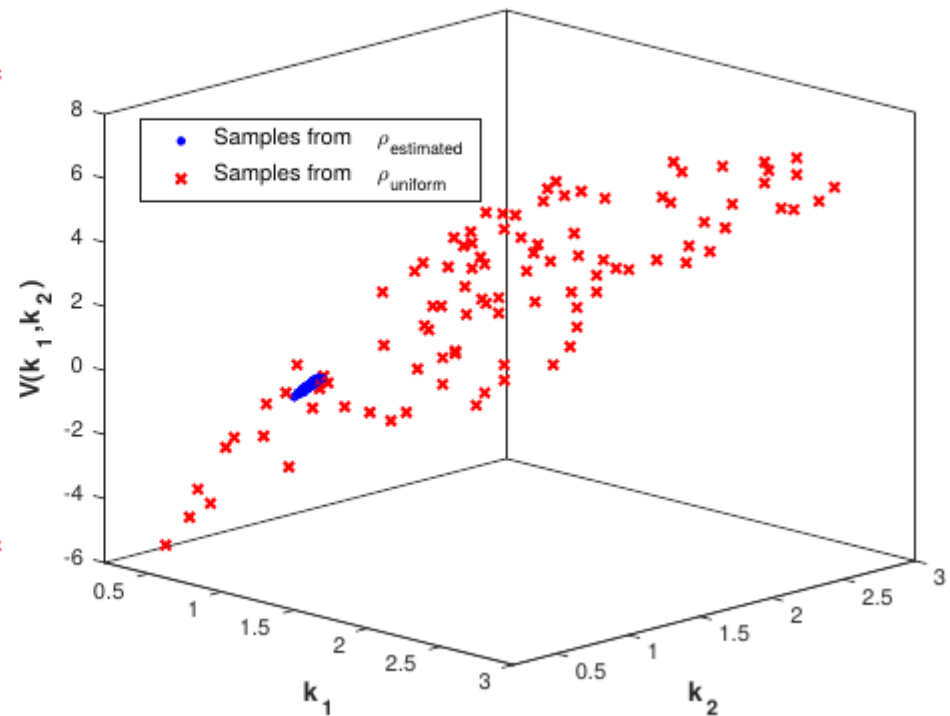
## II. "Dimension-reduction" with the active subspace method.

## III. Solving dynamic models on high-dimensional, (irregularly-shaped) state spaces.

# Recall: Want to solve Dynamic Models on high-dimensional, irregularly-shaped state-spaces

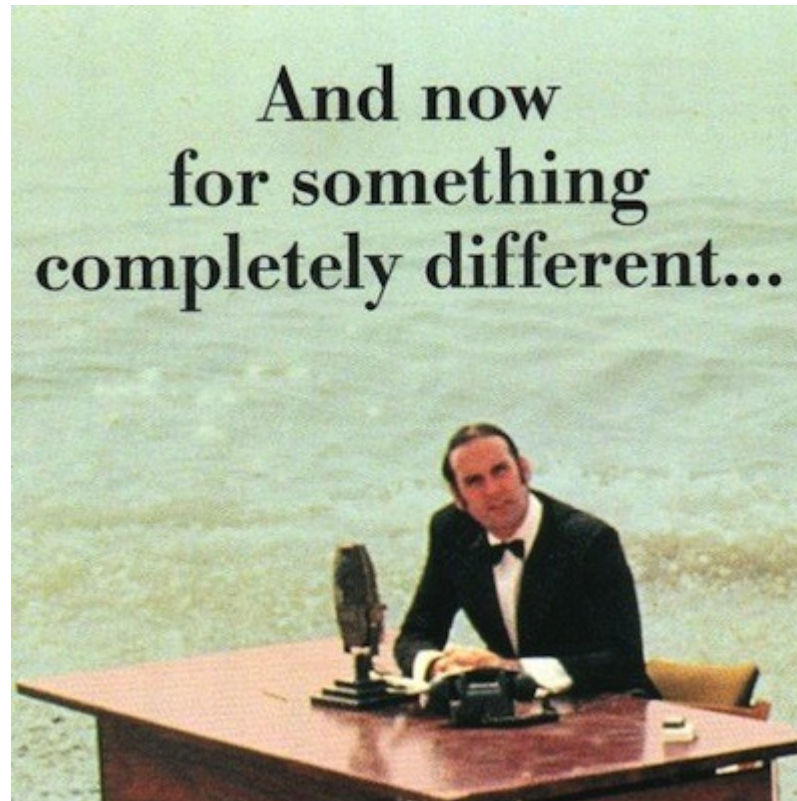


Blue: ergodic set.  
Red: Computational domain



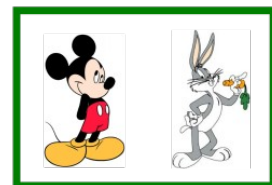
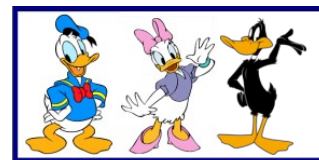
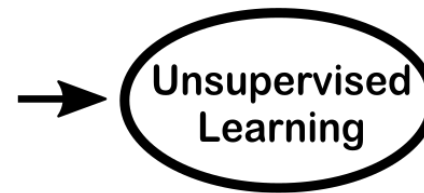
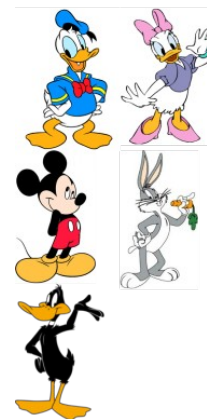
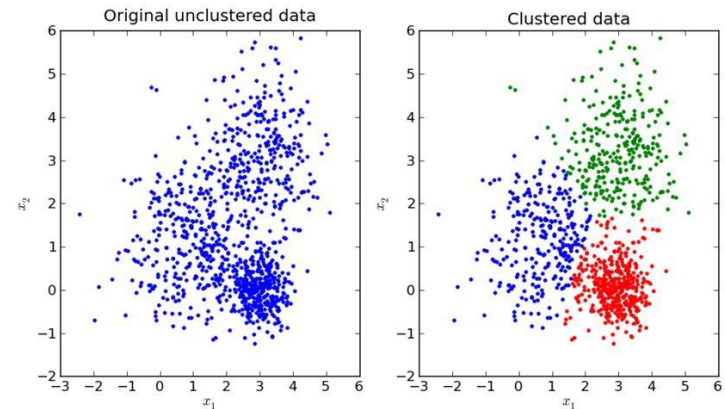
Blue: Value function, evaluated on ergodic set  
Red: Value function, evaluated on the entire comp. domain

# I. Gaussian Mixture Models (GMM)



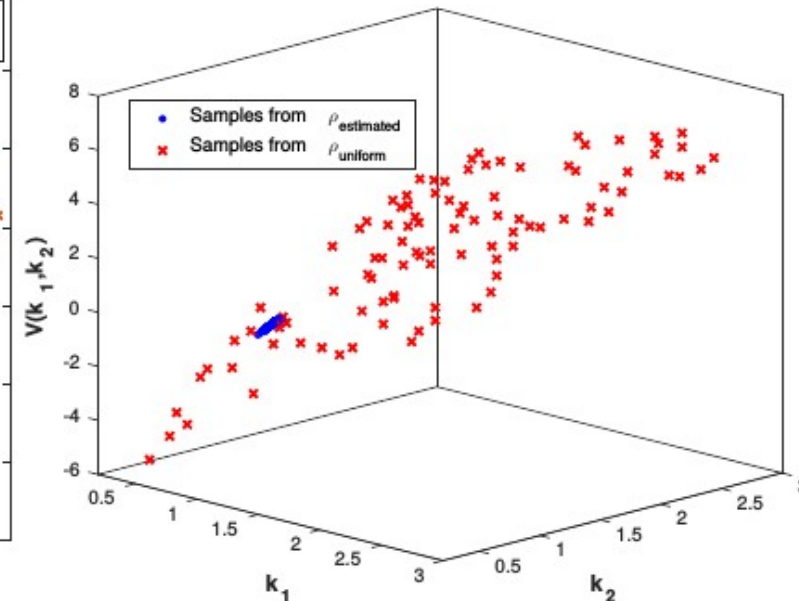
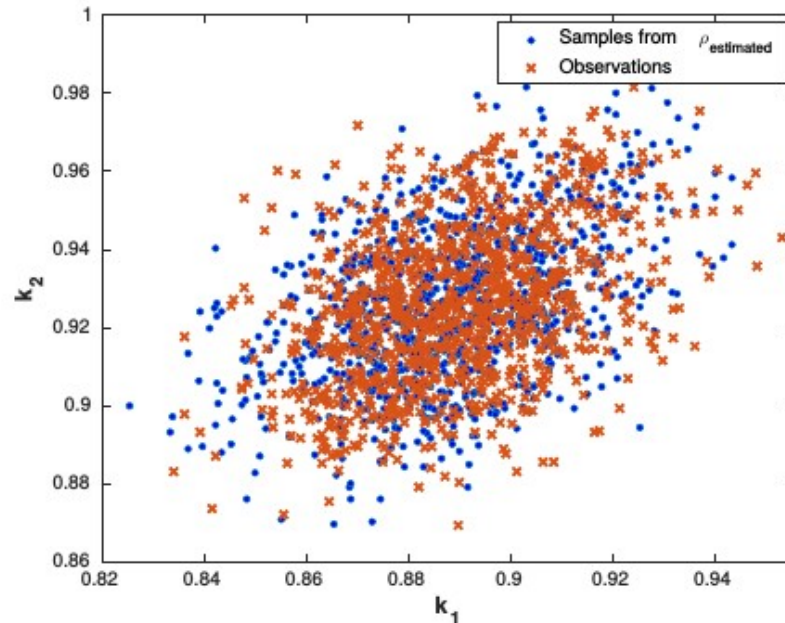
# Recall: Unsupervised Machine Learning

- Learning “what normally happens”.
- No output.
- Clustering: Grouping similar instances.
- Example applications:
  - Customer segmentation.
  - Image compression: Color quantization.
  - Bioinformatics: Learning motifs.



# Simulate the Economy

Den Haan and Marcet (1994), Judd et al. (2010, and Maliar and Maliar (2015), Scheidegger & Bilonis (2019)



$$V(\mathbf{k}) = \max_{\mathbf{l}, \mathbf{c}, \mathbf{l}} \left( u(\mathbf{c}, \mathbf{l}) + \beta \mathbb{E} \left\{ V_{next}(\mathbf{k}^+) \right\} \right),$$

*s.t.*

$$k_j^+ = (1 - \delta) \cdot k_j + I_j + \varepsilon_j, \quad j = 1, \dots, D$$

$$\Gamma_j = \frac{\zeta}{2} k_j \left( \frac{I_j}{k_j} - \delta \right)^2, \quad j = 1, \dots, D$$

$$\sum_{j=1}^D \left\{ c_j + I_j - \delta \cdot k_j \right\} = \sum_{j=1}^D \left\{ f(k_j, l_j) - \Gamma_j \right\},$$

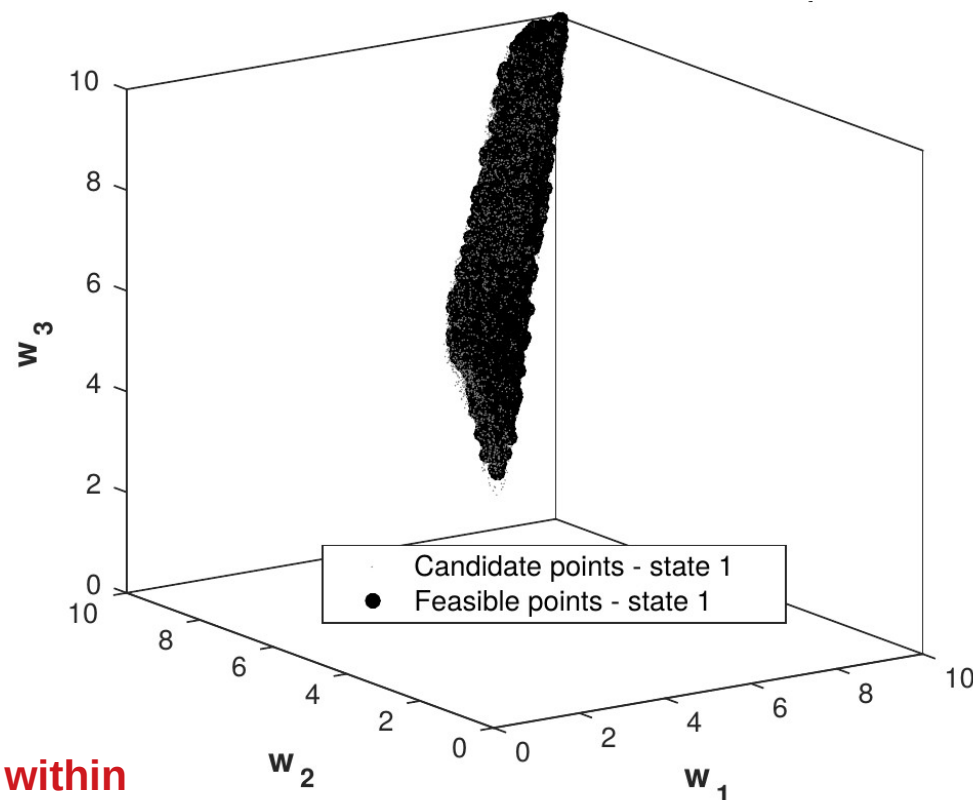
→ Simulate the Model

# Motivation – Feasible Sets

see, e.g., Abreu et al. (1986/1990), Fernandes & Phelan (2000)

See, e.g., dynamic incentive problems:

- Domain of interest is high-dimensional.
  - Irregularly-shaped.
  - You need to approximate Value- and Policy functions on such a domain.
  - In the multi-d setting, if you use grid-based methods, you spend way more than 99% of your resources in vain.
- Q: How can we represent such sets?
- Q: How can we generate “observations” from within such sets?
- One way to do so are Mixture of Gaussians (see, e.g., Bishop (2006)).
- Approximate the set in a probabilistic fashion.



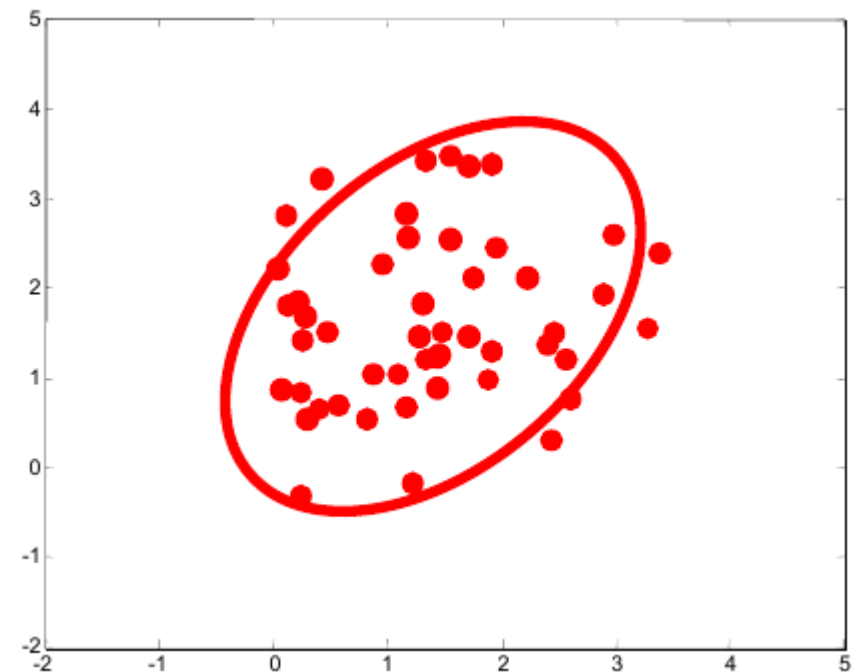
# Recall Multivariate Gaussians

$$\mathcal{N}(x; \mu, \Sigma) = \frac{1}{(2\pi)^{d/2}} |\Sigma|^{-1/2} \exp \left\{ -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right\}$$

Maximum Likelihood estimates given by:

$$\hat{\mu} = \frac{1}{N} \sum_i x^{(i)}$$

$$\hat{\Sigma} = \frac{1}{N} \sum_i (x^{(i)} - \hat{\mu})^T (x^{(i)} - \hat{\mu})$$

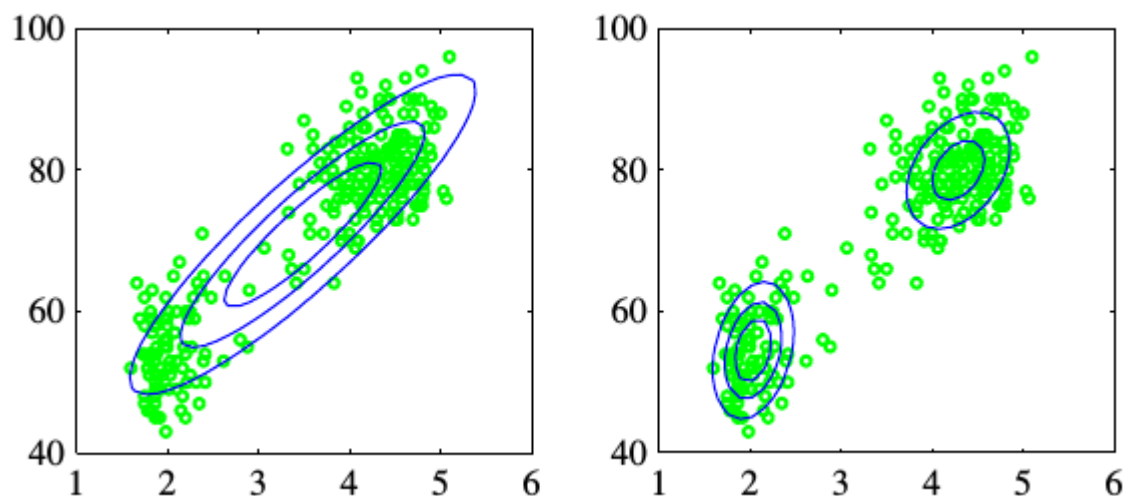




# Mixture of Gaussians – the basic idea

Consult e.g. the books by Bishop (2006) or Murphy (2012) for more details.

- Figure: plots of the ‘old faithful’ data.
- The blue curves show contours of **constant probability density**.
- On the left is a **single Gaussian distribution** which has been fitted to the data using maximum likelihood.

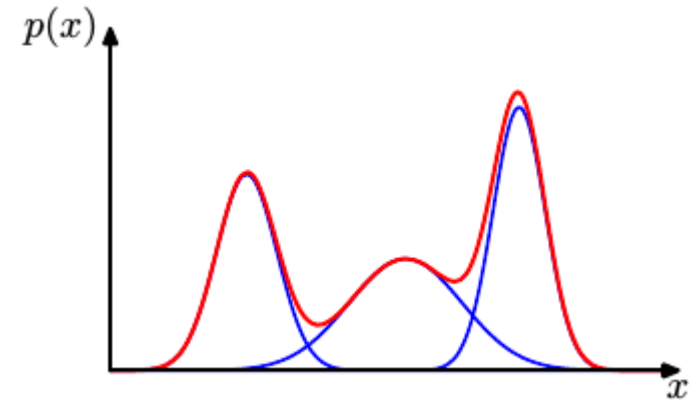


Waiting time between eruptions and the duration of the eruption for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA.

- Note that **this distribution fails to capture the two clumps in the data** and indeed places much of its probability mass in the central region between the clumps where the data are relatively sparse.
- On the right the distribution is given by a **linear combination of two Gaussians** which has been fitted to the data, and which gives a better representation of the data.

# GMM basics

- Example of a Gaussian mixture distribution  $p(x)$  in one dimension showing three Gaussians (each scaled by a coefficient) in blue and their sum in red.



- Linear combinations of Gaussians can give rise to very complex densities.
- By using a sufficient number of Gaussians and adjusting their means covariances as well as the coefficients in the linear combination, almost any density can be approximated with arbitrary accuracy.
- $k$  latent variables.

# Superposition of Gaussians

- Formally, a GMM is:  $p(\mathbf{x}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$  **(A)**
- Each Gaussian density  $\mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$  is called **a component of the mixture**.
- It has its own **mean  $\boldsymbol{\mu}_k$**  and **covariance  $\boldsymbol{\Sigma}_k$** .
- $\pi_k$  : mixing coefficients.**

→ If we integrate both sides of **(A)** with respect to  $\mathbf{x}$ , and note that both  $p(\mathbf{x})$  and the individual Gaussian components are normalized, one obtains:

$$\sum_{k=1}^K \pi_k = 1. \quad 0 \leq \pi_k \leq 1.$$

$$\left[ p(\mathbf{x}) \geq 0 \quad \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \geq 0 \right]$$

# Examplein 1d

Observations  $x_1 \dots x_n$

- $K=2$  Gaussians with unknown  $\mu, \sigma^2$
- Estimation trivial if we know the source of each observation

$$\mu_b = \frac{x_1 + x_2 + \dots + x_{n_b}}{n_b}$$
$$\sigma_b^2 = \frac{(x_1 - \mu_1)^2 + \dots + (x_n - \mu_n)^2}{n_b}$$



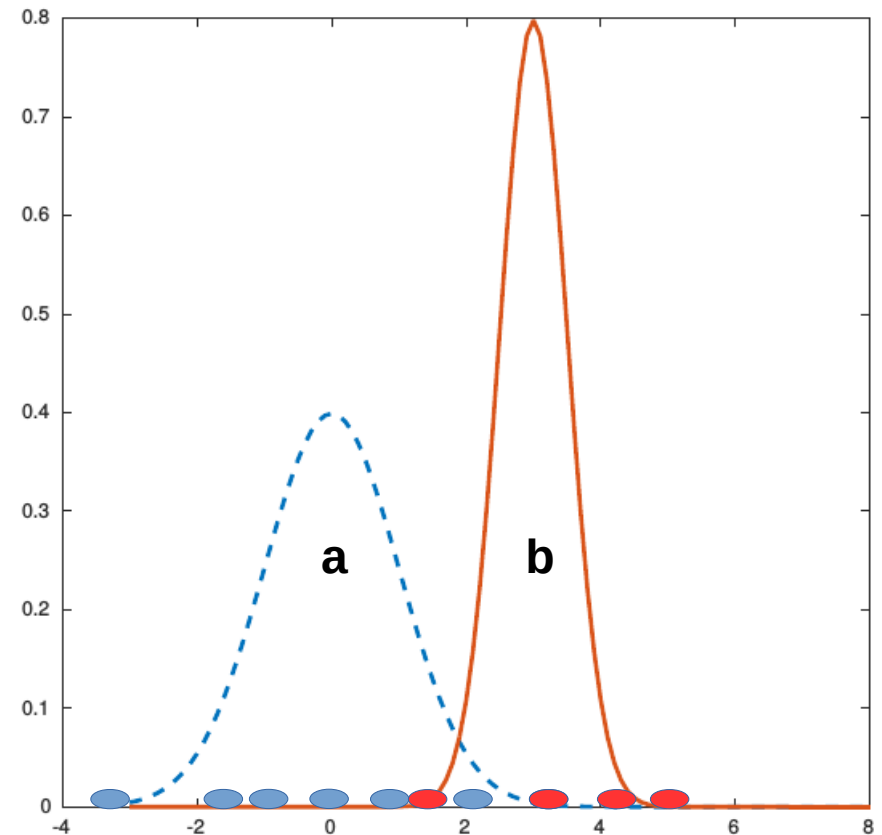
# Example: Expectation Maximization in 1d

see, e.g., Dempster et al. (1977)

Observations  $x_1 \dots x_n$

- K=2 Gaussians with unknown  $\mu$ ,  $\sigma^2$
- Estimation trivial if we know the source of each observation

$$\mu_b = \frac{x_1 + x_2 + \dots + x_{n_b}}{n_b}$$
$$\sigma_b^2 = \frac{(x_1 - \mu_b)^2 + \dots + (x_{n_b} - \mu_b)^2}{n_b}$$



# Example: Expectation Maximization in 1d (II)

- What if we don't know the source?
- If we knew parameters of the Gaussians ( $\mu$ ,  $\sigma^2$ )

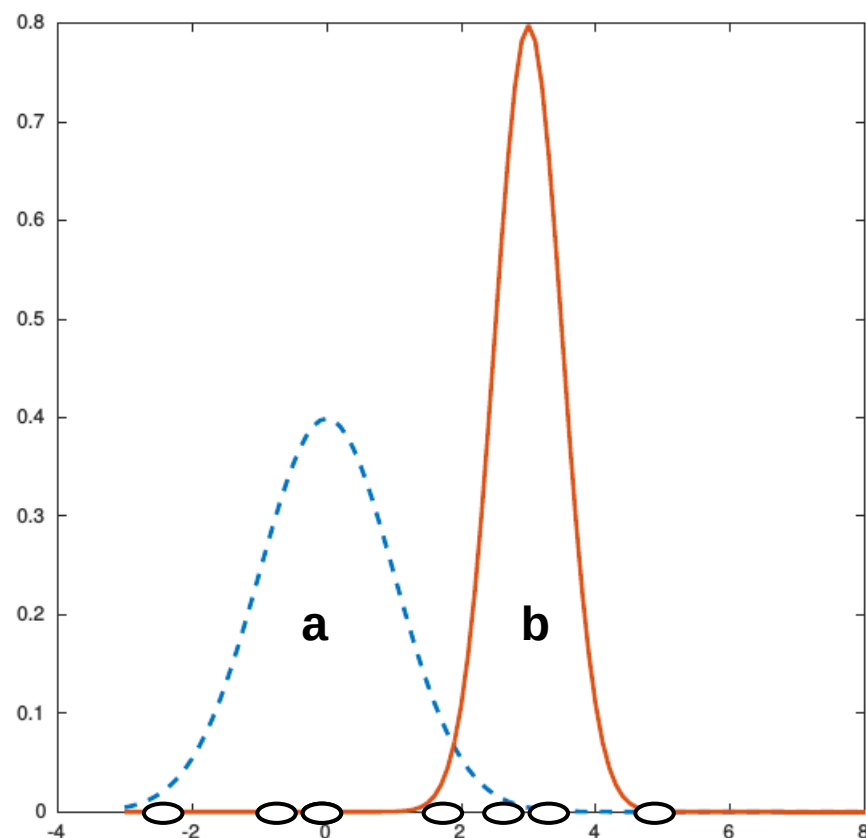


# Example: Expectation Maximization in 1d (II)

- What if we don't know the source?
  - If we knew parameters of the Gaussians ( $\mu$ ,  $\sigma^2$ )
- can guess whether point is more likely to be a or b.

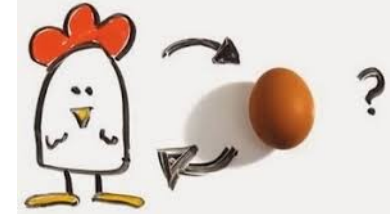
$$P(b | x_i) = \frac{P(x_i | b)P(b)}{P(x_i | b)P(b) + P(x_i | a)P(a)}$$

$$P(x_i | b) = \frac{1}{\sqrt{2\pi\sigma_b^2}} \exp\left(-\frac{(x_i - \mu_b)^2}{2\sigma_b^2}\right)$$



# EM Algorithm (in 1d)

see, e.g., Dempster et al. (1977), Bishop (2006), Murphy (2012) and references therein for details.



A fundamental problem:

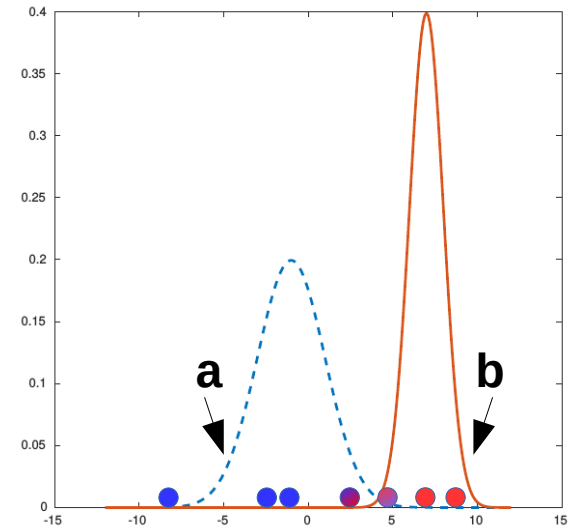
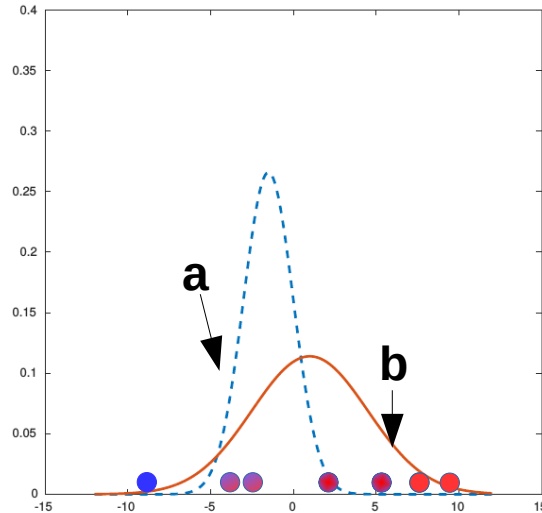
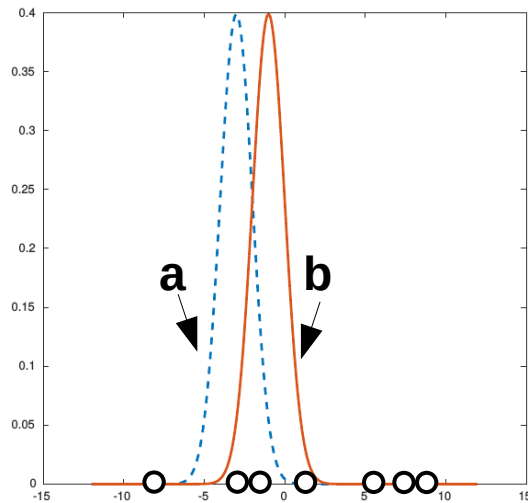
- we need  $(\mu_a, \sigma_a^2)$  and  $(\mu_b, \sigma_b^2)$  to guess the source of the points.
- we need to know the source to estimate  $(\mu_a, \sigma_a^2)$  and  $(\mu_b, \sigma_b^2)$ .

## EM algorithm

1. **start** with two randomly placed Gaussians  $(\mu_a, \sigma_a^2)$  and  $(\mu_b, \sigma_b^2)$ .
2. **E(xpectation)-step:**
  - for each point:  $P(b|x_i)$  = does it look like it came from b?
3. **M(maximization)-step:**
  - adjust  $(\mu_a, \sigma_a^2)$  and  $(\mu_b, \sigma_b^2)$  to fit points assigned to them.
4. **Iterate until convergence.**



# EM in 1d



$$P(x_i | b) = \frac{1}{\sqrt{2\pi\sigma_b^2}} \exp\left(-\frac{(x_i - \mu_b)^2}{2\sigma_b^2}\right)$$

$$b_i = P(b | x_i) = \frac{P(x_i | b)P(b)}{P(x_i | b)P(b) + P(x_i | a)P(a)}$$

$$a_i = P(a | x_i) = 1 - b_i$$

$$\mu_b = \frac{b_1 x_1 + b_2 x_2 + \dots + b_n x_{n_b}}{b_1 + b_2 + \dots + b_n}$$

$$\sigma_b^2 = \frac{b_1 (x_1 - \mu_b)^2 + \dots + b_n (x_n - \mu_b)^2}{b_1 + b_2 + \dots + b_n}$$

$$\mu_a = \frac{a_1 x_1 + a_2 x_2 + \dots + a_n x_{n_b}}{a_1 + a_2 + \dots + a_n}$$

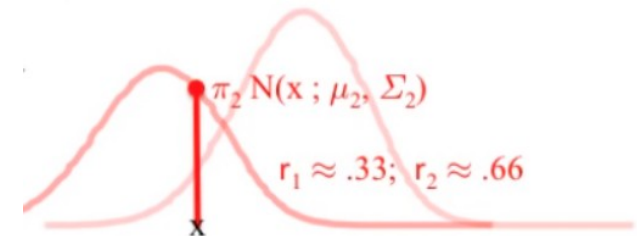
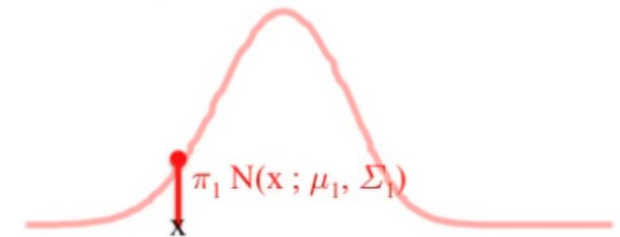
$$\sigma_a^2 = \frac{a_1 (x_1 - \mu_a)^2 + \dots + a_n (x_n - \mu_a)^2}{a_1 + a_2 + \dots + a_n}$$

→ We could also estimate priors:  
 $P(b) = (b_1 + b_2 + \dots + b_n) / n$   
 $P(a) = 1 - P(b)$

# EM in the multidimensional case

- Start with parameters describing each cluster
- Mean  $\mu_c$ , Covariance  $\Sigma_c$ , “size”  $\pi_c$
- **E-step (“Expectation”):**
  - For **each observation/point**  $x_i$
  - Compute “ $r_{ic}$ ”, the probability that it belongs to cluster  $c$ .
    - Compute its probability under model  $c$ .
    - Normalize to sum to one (over clusters  $c$ ).

$$r_{ic} = \frac{\pi_c \mathcal{N}(x_i ; \mu_c, \Sigma_c)}{\sum_{c'} \pi_{c'} \mathcal{N}(x_i ; \mu_{c'}, \Sigma_{c'})}$$



- If  $x_i$  is very likely under the  $c$ -th Gaussian, it gets high weight.
- Denominator just makes  $r$ 's sum to one.

# EM in the multidimensional case

- **M-step (“Maximization step”):**
  - For each cluster (Gaussian)  $z=c$
  - Update its parameters using the (weighted) data points

$$N_c = \sum_i r_{ic}$$

Total responsibility allocated to cluster  $c$

$$\pi_c = \frac{N_c}{N}$$

Fraction of total assigned to cluster  $c$

$$\mu_c = \frac{1}{N_c} \sum_i r_{ic} x_i$$

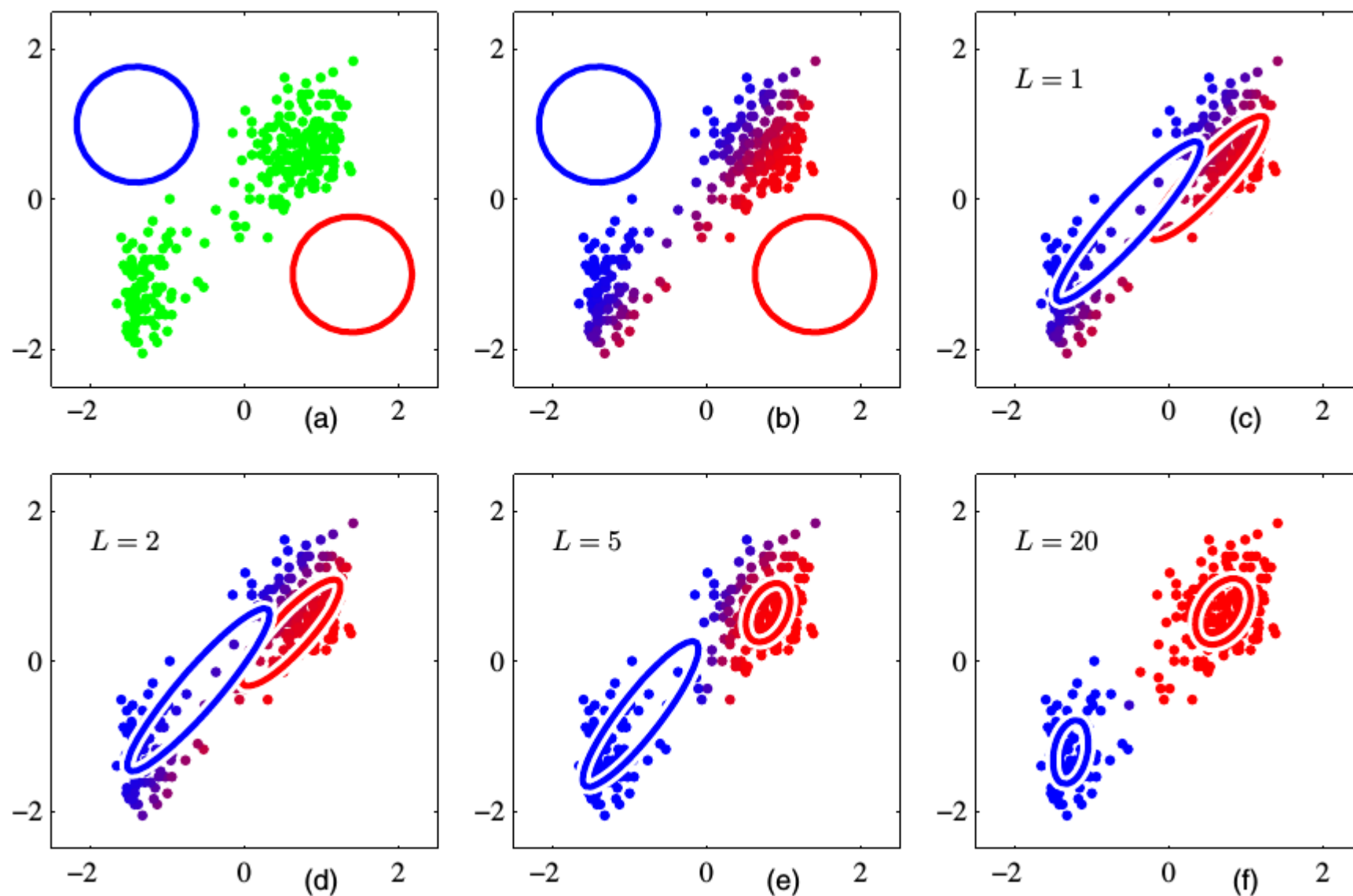
Weighted mean of assigned data

$$\Sigma_c = \frac{1}{N_c} \sum_i r_{ic} (x_i - \mu_c)^T (x_i - \mu_c)$$

Weighted covariance of assigned data  
(use new weighted means here)

# Gaussian mixture models: $d > 1$

See Bishop (2006) for details



# Bayesian Information Criterion (BIC)

See, e.g., Alpaydin (2014), MIT Press

- **How to pick k?**

- Probabilistic model: 
$$L = \ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

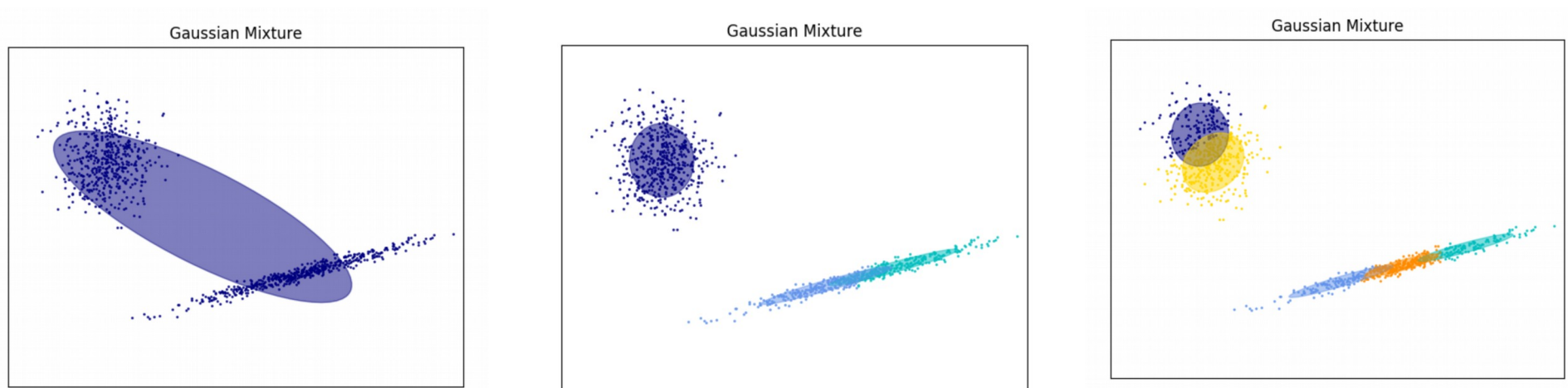
- Tries to “fit” the data (maximize likelihood)
- Choose k that makes L as large as possible?
  - K = n: each data point has its own “source”
  - may not work well for new data points
- Split points into training set **T** and validation set **V**
  - for each k: fit parameters of **T**
  - measure likelihood of **V**
  - sometimes still best when k = n
- **“Occam’s razor”**:
  - Pick the “simplest” of all models that fits the data.
  - Assess, e.g., via Bayes Information Criterion (BIC):  $\max_p \{ L - 1/2 p \log(n) \}$
  - **L**: Likelihood; **p**: # Parameters in the model – how simple is the model.

# Hands-on example 1

<https://scikit-learn.org/stable/modules/mixture.html>

Lecture\_3/code/GMM\_scikit\_example.py

- Plot the confidence ellipsoids of a mixture of two Gaussians obtained with Expectation Maximization (GaussianMixture class)
- The model has access to 1, 3, and 5 components with which to fit the data. Note that the Expectation Maximization model will necessarily use ALL components
- In the 5-component example, we can see that the Expectation Maximization model splits some components arbitrarily, because it is trying to fit too many components.



# Hands-on example 2

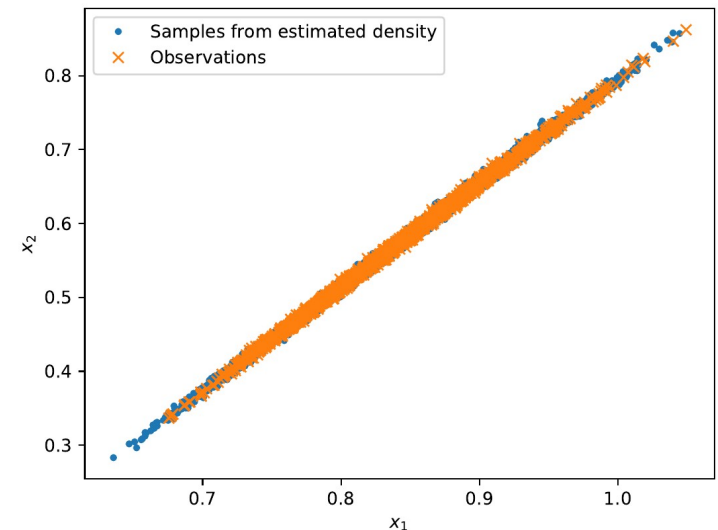
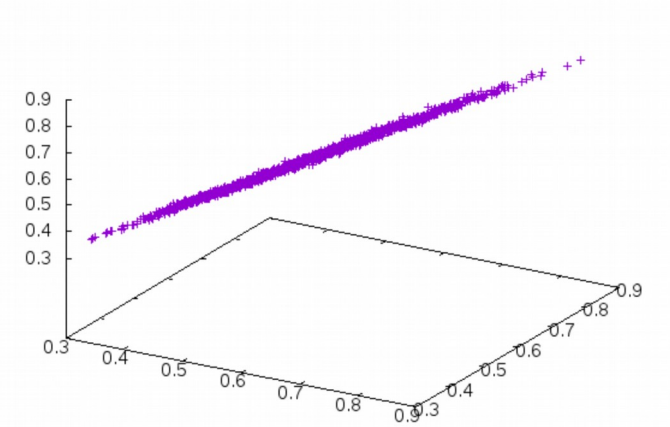
Lecture\_3/code/code/BGMM\_data

cf. [https://papers.ssrn.com/sol3/papers.cfm?abstract\\_id=3282487](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3282487)

- We simulate a bunch of data (e.g., an ergodic set).
  - Its in a text file (**ergodic\_data.txt** – 3 dimensions)

- We apply GMM (**build\_density.py**)

- We can sample data from the fitted GMM model (**sample.py**)



# Value function iteration (on irregularly-shaped domains)

e.g. Bellman (1961), Stokey, Lucas & Prescott (1989), Judd (1998), ...

The solution is approached in the limit as  $j \rightarrow \infty$  by iterations on:

$$V_{j+1}(\underline{x}) = \max_u \{r(x, u) + \beta \underline{V_j}(\underline{\tilde{x}})\}$$

s.t.

$$\tilde{x} = g(x, u)$$

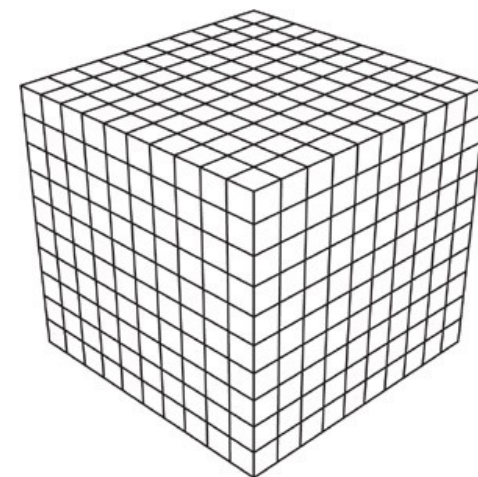
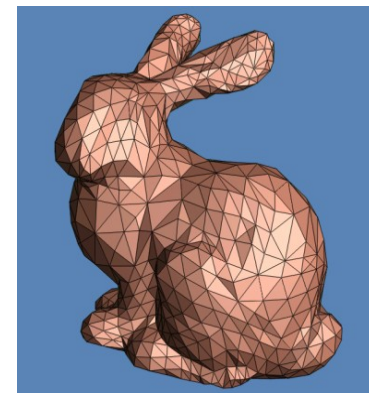
**x**: point in state space; describes your system.  
State-space potentially **irregularly-shaped**  
and **high-dimensional**.

`old solution':  
high-dimensional function on which **we interpolate**.

→ **N<sup>d</sup>** points in ordinary discretization schemes.

→ **“Curse of dimensionality”**

→ **Use-case for Gaussian process regression & active subspaces.**





# How many is dimensions is high dimensions?

Number of parameters (the dimension)	Number of model runs (at 10 points per dimension)	Time for parameter study (at 1 second per run)
1	10	10 sec
2	100	~ 1.6 min
3	1,000	~ 16 min
4	10,000	~ 2.7 hours
5	100,000	~ 1.1 days
6	1,000,000	~ 1.6 weeks
...	...	...
20	1e20	3 trillion years (240x age of the universe)

**Dimension reduction**

*Exploit symmetries, e.g., via the active subspace method*

**Deal with #Points**

*Adaptive Sparse Grids*

**High-performance computing**

*Reduces time to solution, but not the problem size*

# The Curse of Dimensionality

- **Why is it hard** (impossible) to learn functions in **high dimensions**?
- Methods relying on **local similarity measures** (e.g., kernels in the context of GPR) have severe problems.
- “The curse of highly variable functions for local kernel machines”, Bengio et al. (2006).
  - The reason is that the **Euclidean distance** is **not a good “closeness” measure in high-dimensions**.

*Most of the volume of a high-dimensional orange is in the skin, not in the pulp. If a constant number of examples is distributed uniformly in a high-dimensional hypercube, beyond some dimensionality most examples are closer to a face of the hypercube than to their nearest neighbors\*.*

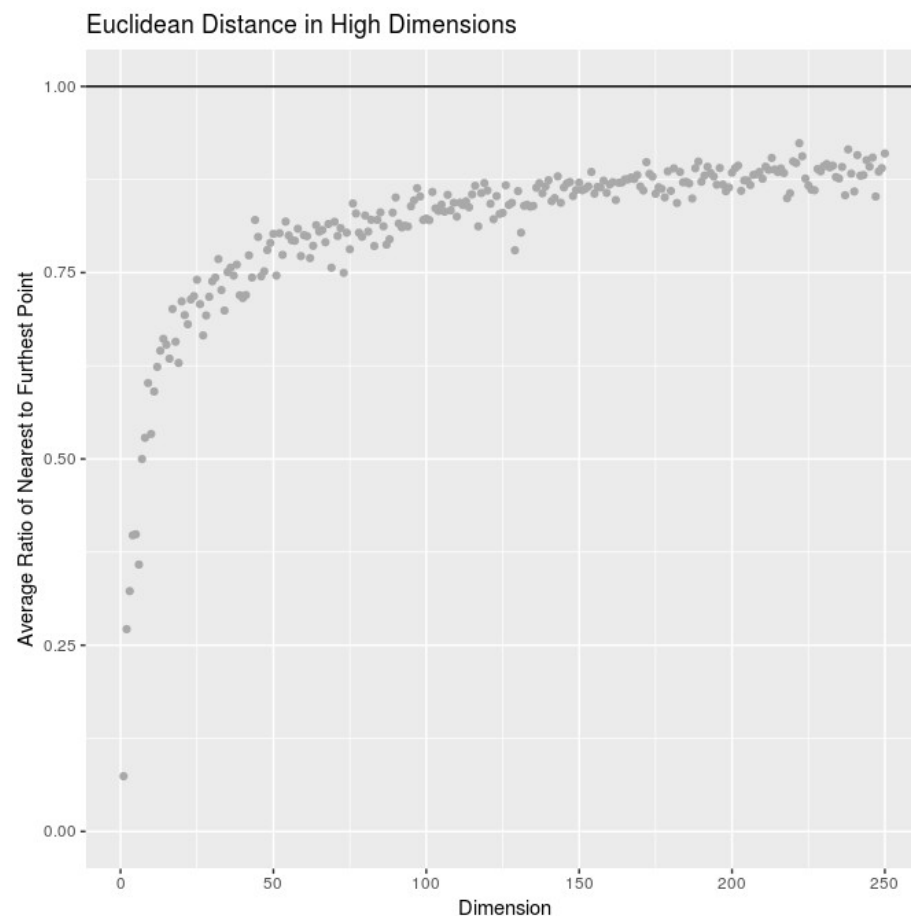
\* “A few useful things to know about machine learning”, Pedro Domingos, (2012)

# Volumes in high dimensions (II)

- Consider the following:  
Let's simulate a bunch of random normal points in many dimensions and **plot the average ratio of the distance to nearest point to the distance to farthest point**.

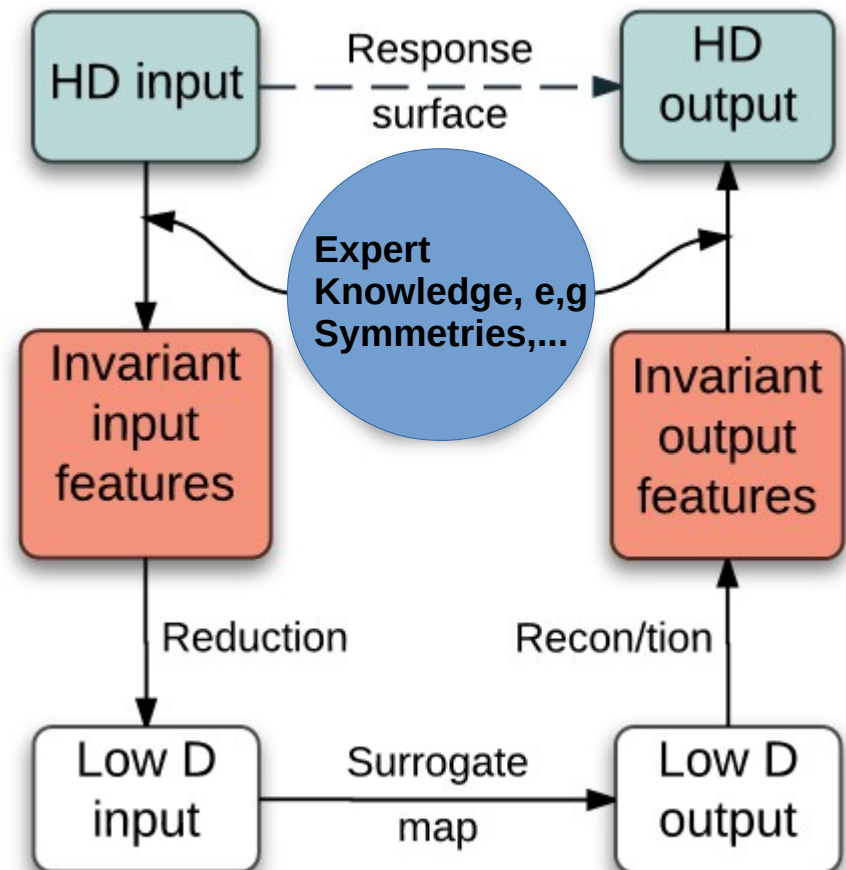
$$D = \sqrt{\sum_{i=1}^d (x_i)^2}$$

- This ratio tends to one, all points “look alike”.
  - **No good notion of “locality”**.
  - Problems, e.g., for GPR.



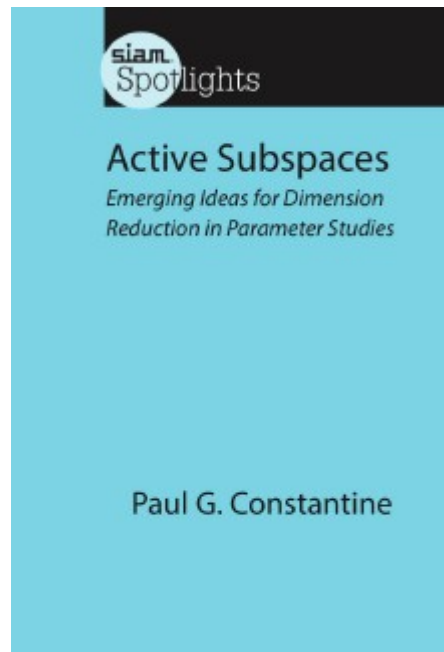
# Dealing with high dimensions

- Generic term approach:
- **Exploit invariances and symmetries**  
Induced by knowledge about the economics problem.
- Discover and exploit (non)-linear manifolds over which the **response exhibits maximal variability**.
- Here, we focus on:
  - HD input (already satisfying symmetries).
  - Discover and exploit linear manifold of maximal variability.



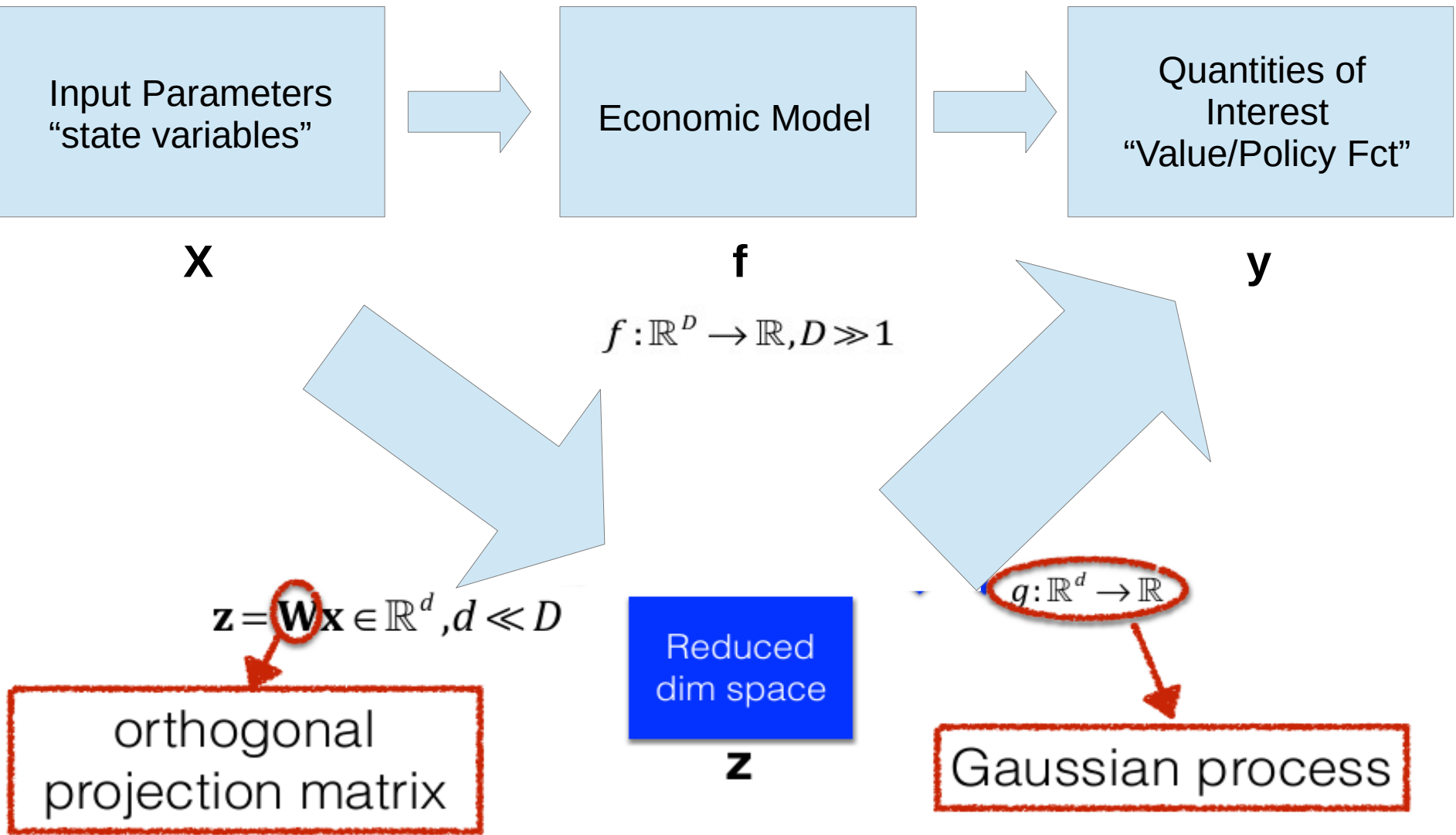
# Active Subspaces

<http://bookstore.siam.org/sl02/>

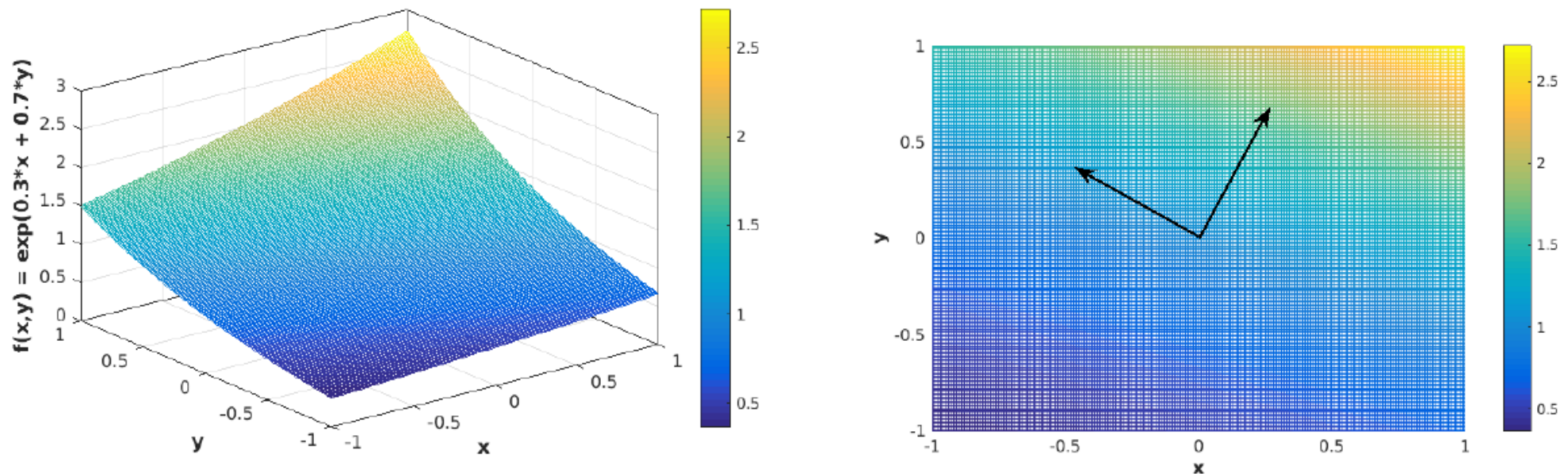


# High Dimensions → Active Subspaces

(see, e.g., Constantine et. al (2013); Constantine (2015), with references therein)



# Active Subspaces – intuitive example



Function varies most along  $[0.3, 0.7]$ , and is constant in the orthogonal direction.



# Discover active subspaces

(see, e.g., Constantine (2015), with references therein)

Step 1: Find  $\mathbf{W}$

“Mean-square directional derivative”

Note: there are also derivative-free ways of constructing  $\mathbf{W}$ .

$$\mathbf{C} = \mathbb{E}[\nabla_{\mathbf{x}} f(\mathbf{x}) \nabla_{\mathbf{x}} f(\mathbf{x})^T] \approx \frac{1}{N} \sum_{i=1}^N \nabla_{\mathbf{x}} f(\mathbf{x}^{(i)}) \nabla_{\mathbf{x}} f(\mathbf{x}^{(i)})^T$$

$$\mathbf{C} = \mathbf{W} \mathbf{\Lambda} \mathbf{W}^T$$

Gradient info

Partition the eigendecomposition

→ Compute Eigenvalues, order them.

→ Look for “gaps.”

$$\mathbf{\Lambda} = \begin{bmatrix} \mathbf{\Lambda}_1 & \\ & \mathbf{\Lambda}_2 \end{bmatrix}, \quad \mathbf{W} = [\mathbf{W}_1 \quad \mathbf{W}_2], \quad \mathbf{W}_1 \in \mathbb{R}^{m \times n}$$

Create a rotated coordinate system

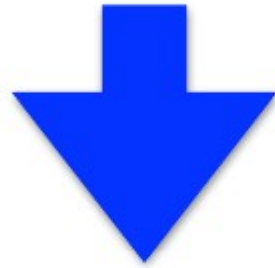
$$\mathbf{x} = \mathbf{W} \mathbf{W}^T \mathbf{x} = \mathbf{W}_1 \mathbf{W}_1^T \mathbf{x} + \mathbf{W}_2 \mathbf{W}_2^T \mathbf{x} = \mathbf{W}_1 \mathbf{q} + \cancel{\mathbf{W}_2 \mathbf{y}}$$



# GPs in active subspace

Step 2: Regression  $\longrightarrow$   $f(\mathbf{x}) \approx g(\mathbf{W}_1^T \mathbf{x}).$

$$\mathcal{D}_{\text{projected}} = \left\{ \left( \mathbf{z}^{(i)} = \mathbf{W} \mathbf{x}^{(i)}, y^{(i)} = f(\mathbf{x}^{(i)}) \right) \right\}_{i=1}^N$$

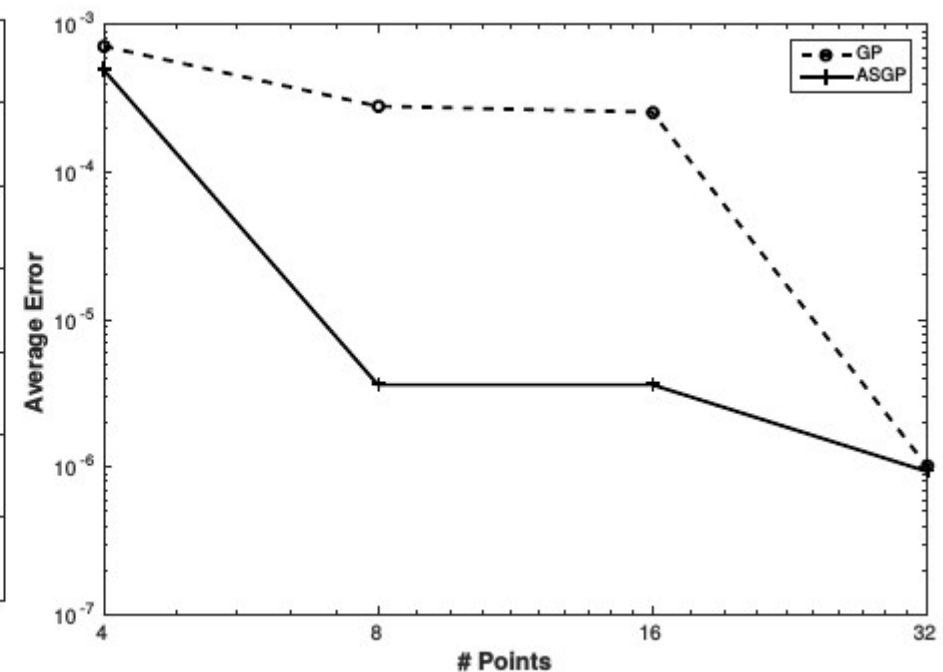
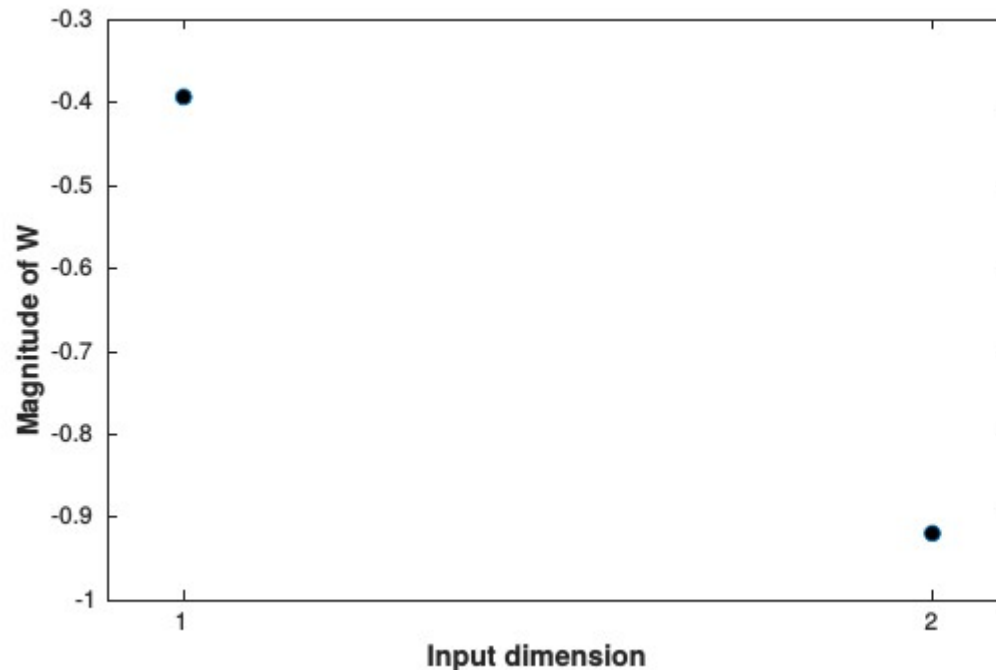


$$g(\cdot) \sim \text{GP}\left(g(\cdot) | m(\cdot), k_0(\cdot, \cdot)\right).$$

$$g(\cdot) | \mathcal{D}_{\text{projected}} \sim \text{GP}\left(g(\cdot) | m^*(\cdot), k_0^*(\cdot, \cdot)\right)$$

# Analytical example in 2d

cf. [Lecture\\_3/code/AS\\_ex1.py](#)



$$f(x, y) = \exp(0.3x + 0.7y)$$

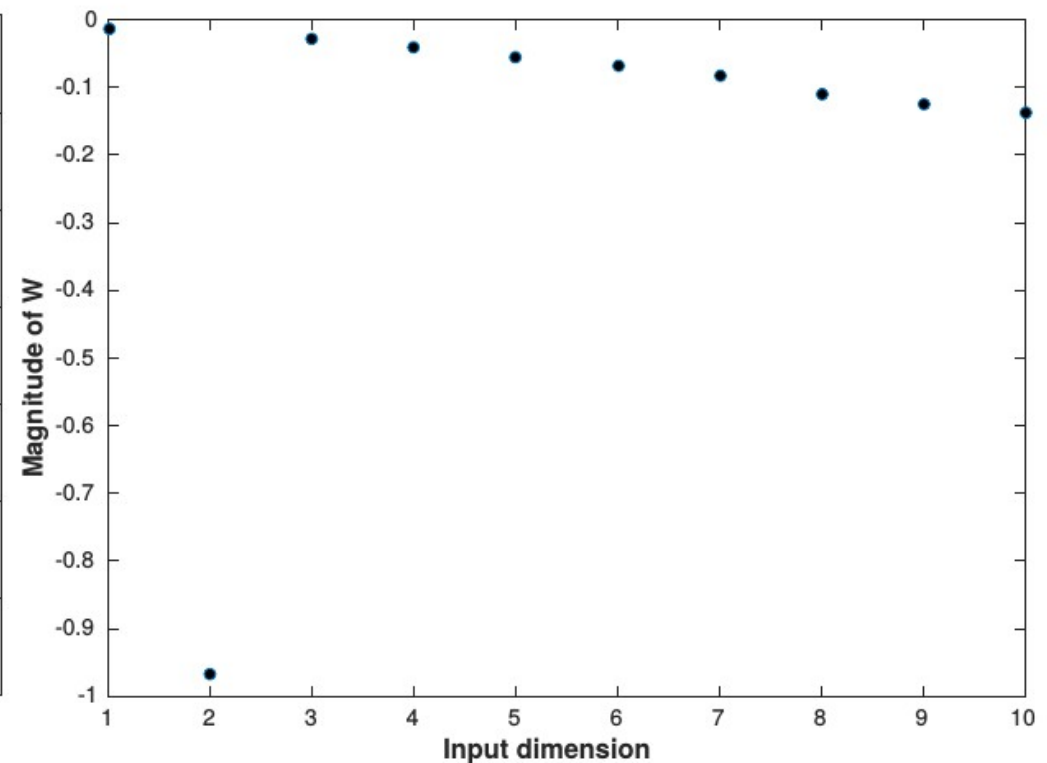
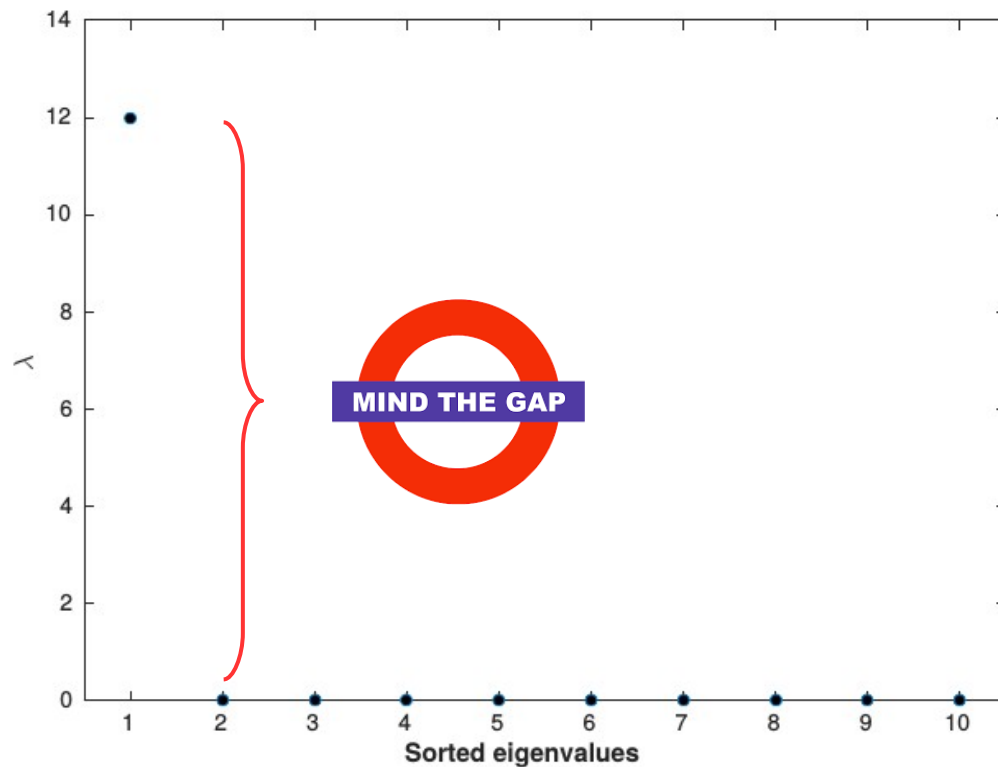
**The left panel** shows the projection matrix  $W$  of the **1-dimensional active subspace**.  
**The right panel** displays a comparison of the interpolation error for 2-dimensional GPs and an 1-dimensional AS of varying resolution, respectively.

# 10d analytical example

cf. [Lecture\\_3/code/AS\\_ex2.py](#)

$$f : [-1, 1]^{10} \rightarrow \mathbb{R}$$

$$f(x_1, \dots, x_{10}) = \exp(0.01x_1 + 0.7x_2 + 0.02x_3 + 0.03x_4 + 0.04x_5 \\ + 0.05x_6 + 0.06x_7 + 0.08x_8 + 0.09x_9 + 0.1x_{10})$$



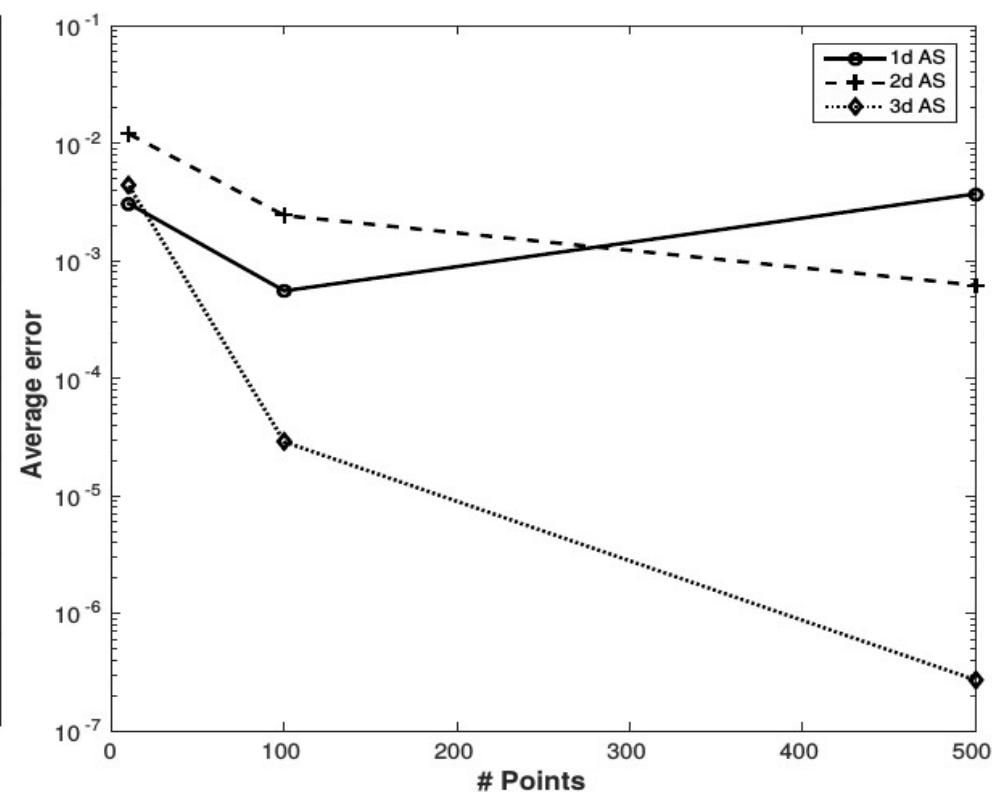
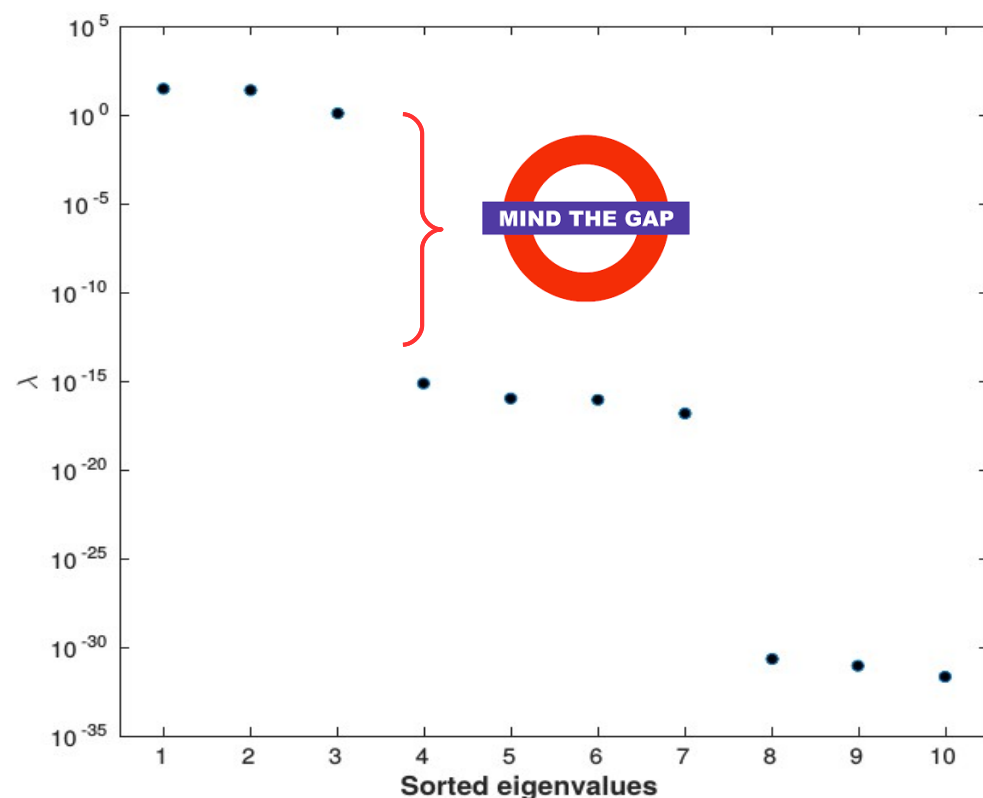
# 10d analytical example: Gap

cf. [Lecture\\_3/code/AS\\_ex3.py](#)

$$f : [-1, 1]^{10} \rightarrow \mathbb{R}$$

$$f(x_1, \dots, x_{10}) = x_2 \cdot x_3 \cdot \exp(0.01x_1 + 0.7x_2 + 0.02x_3 + 0.03x_4 + 0.04x_5 \\ + 0.05x_6 + 0.06x_7 + 0.08x_8 + 0.09x_9 + 0.1x_{10}).$$

ASGP surrogates of dimension  $d = \{1, 2, 3\}$



# Active Subspaces versus PCA

- Some comment on the similarities and discrepancies between PCA and AS:
- PCA identifies a projection of the input space.
- The goal of this projection, however, is not the same as in AS.
- PCA picks the projection that minimizes the mean square reconstruction error of the input  $\mathbf{x}$  that is, it minimizes  $E[\|\mathbf{x} - \mathbf{W}(\mathbf{W}^T \mathbf{x})\|^2]$ , where the expectation is with respect to the input-generating distribution.
- AS has an objective that is very different to that of PCA:  
AS focuses on finding a  $\mathbf{W}$  that allows us to approximate  $f(\mathbf{x})$  with a function of the form  $h(\mathbf{W}\mathbf{x})$  as well as possible.
- Even though AS has not (yet) been formulated to directly minimize the mean square error  $E[(f(\mathbf{x}) - h(\mathbf{W}\mathbf{x}))^2]$ , it was shown by Constantine that the mean square error is bounded by a term proportional to the sum of the neglected eigenvalues of  $\mathbf{C}$ .
  - PCA focuses on finding the best linear projection that allows the reconstruction of the input, whereas **AS focuses on the search for the best linear projection that enables the reconstruction of the response surface  $f(\mathbf{x})$ .**

# III. Switching gears: Dynamic Programming



# Test model – growth model

To demonstrate the capabilities of our algorithm, we choose an **infinite-horizon discrete-time multi-dimensional stochastic optimal growth model** (see, e.g, Cai & Judd (2014), and references therein).

Workhorse for studying methods for solving high-dimensional economic models.

The model has few parameters and is relatively easy to explain, whereas the **dimensionality of the problem can be scaled up** in a straightforward but meaningful way

→ state-space depends linearly on the number of  **$D$  sectors (=dim)** considered.

→ there are  $D$  sectors with **capital**  $\mathbf{k}_t = (k_{t,1}, \dots, k_{t,D})$

and elastic **labour supply**  $\mathbf{l}_t = (l_{t,1}, \dots, l_{t,D})$

## Growth model (II)

The production function of sector  $i$  at time  $t$  is  $f(k_{t,i}, l_{t,i})$ , for  $i = 1, \dots, D$ .

Consumption:  $\mathbf{c}_t = (c_{t,1}, \dots, c_{t,D})$

Investment of the sectors at time  $t$ :  $\mathbf{I}_t = (I_{t,1}, \dots, I_{t,D})$

→ The goal now is to find **optimal consumption** and **labour supply decisions** such that **expected total utility over an infinite time horizon is maximized**.



# The formal multi-*d* Growth Model

See, e.g. Cai & Judd (2014) and references therein

$$V(\mathbf{k}) = \max_{\mathbf{l}, \mathbf{c}, \mathbf{l}} \left( u(\mathbf{c}, \mathbf{l}) + \beta \mathbb{E} \left\{ V_{next}(\mathbf{k}^+) \right\} \right),$$

*s.t.*

$$k_j^+ = (1 - \delta) \cdot k_j + I_j + \epsilon_j, \quad j = 1, \dots, D$$
$$\Gamma_j = \frac{\zeta}{2} k_j \left( \frac{I_j}{k_j} - \delta \right)^2, \quad j = 1, \dots, D$$
$$\sum_{j=1}^D (c_j + I_j - \delta \cdot k_j) = \sum_{j=1}^D (f(k_j, l_j) - \Gamma_j),$$

Parameter	Value
$\beta$	0.8
$\delta$	0.025
$\zeta$	0.5
$[\underline{\mathbf{k}}, \overline{\mathbf{k}}]^D$	$[0.2, 3.0]^D$
$\psi$	0.36
$A$	$(1 - \beta) / (\psi \cdot \beta)$
$\gamma$	2
$\eta$	1
$\sigma$	0.01
$D$	$\{1, \dots, 500\}$

**k: continuous states**

$$u(c, l) = \sum_{i=1}^d \left[ \frac{(c_i/A)^{1-\gamma} - 1}{1-\gamma} - (1-\psi) \frac{l_i^{1+\eta} - 1}{1+\eta} \right]$$

$$f(k_i, l_i) = A \cdot k_i^\psi \cdot l_i^{1-\psi}$$

$$V^\infty(\mathbf{k}) = u(f(k, \mathbf{e}, \mathbf{e}), \mathbf{e}) / (1 - \beta)$$

$$\epsilon_{t,j} \sim \mathcal{N}(0, \sigma^2)$$

# Value function iteration with GPR

**Data:** Initial guess  $V_{next}$  for the next period's value function. Approximation accuracy  $\bar{\eta}$ .

**Result:** The (approximate) equilibrium 3D policy functions  $\xi^* = \{\xi_1^*, \dots, \xi_{3D}^*\}$  and the corresponding value function  $V^*$ .

Set iteration step  $s = 1$ .

**while**  $\eta > \bar{\eta}$  **do**

Generate  $n$  training inputs  $\mathbf{X} = \{\mathbf{k}_i^s : 1 \leq i \leq n\} \in [\underline{\mathbf{k}}, \bar{\mathbf{k}}]^D$

**for**  $\mathbf{k}_i^s \in \mathbf{X}$  **do**

Compute the maximization problem

$$V(\mathbf{k}_i^s) = \max_{\mathbf{l}, \mathbf{c}, \mathbf{l}} \left( u(\mathbf{c}, \mathbf{l}) + \beta \mathbb{E} \left\{ V_{next}(\mathbf{k}_i^+) \right\} \right), \text{ s.t.}$$

$$k_j^+ = (1 - \delta) \cdot k_j + I_j + \epsilon_j, \quad j = 1, \dots, D$$

$$\Gamma_j = \frac{\zeta}{2} k_j \left( \frac{I_j}{k_j} - \delta \right)^2, \quad j = 1, \dots, D$$

$$\sum_{j=1}^D (c_j + I_j - \delta \cdot k_j) = \sum_{j=1}^D (f(k_j, l_j) - \Gamma_j)$$

given the next period's value function  $V_{next}$ .

Set the training targets for the value function:  $t_i = V(\mathbf{k}_i^s)$ .

If required, set the training targets to learn the policy function

$$\xi_{ji}(\mathbf{k}_i^s) \in \arg \max_{p_j} V(\mathbf{k}_i^s).$$

**end**

Set  $\mathbf{t} = \{t_i : 1 \leq i \leq n\}$ .

Given  $\{\mathbf{X}, \mathbf{t}\}$ , learn a surrogate  $V_{surrogate}$  of  $V$  with ASGP (or GP).

Set  $\xi_j = \{\xi_{ji} : 1 \leq i \leq n\}$ .

Given  $\{\mathbf{X}, \xi_j\}$ , learn a surrogate of the policy  $\xi_j$  with ASGP (or GP).

Calculate (an approximation for) the error, e.g.,  $\eta = \|V_{surrogate} - V_{next}\|_\infty$ .

Set  $V_{next} = V_{surrogate}$ .

Set  $s = s + 1$ .

**end**

$V^* = V_{surrogate}$ .

$\xi^* = \{\xi_1, \dots, \xi_{3D}\}$ .

**“Hybrid parallel”**

Implementation

(Shared & distributed memory parallelism)

# Algorithmic Complexity of GPR-DP

**$N$** : number of observations.

The computational cost of standard GPR is dominated by the need to construct the **Cholesky decomposition** of the  $N \times N$  covariance matrix at each step of the likelihood optimization – that is,  **$O(N^3)$** .

The computational cost of **AS** arises from **a single SVD of an  $N \times N$  matrix**, plus the cost of a standard GP regression that is, it is also  **$O(N^3)$** .

In both cases, the right number of observations  $N$  depends on the function smoothness and the input dimensionality (rule of thumb:  **$N \sim 2\text{-}10D$  observations**):

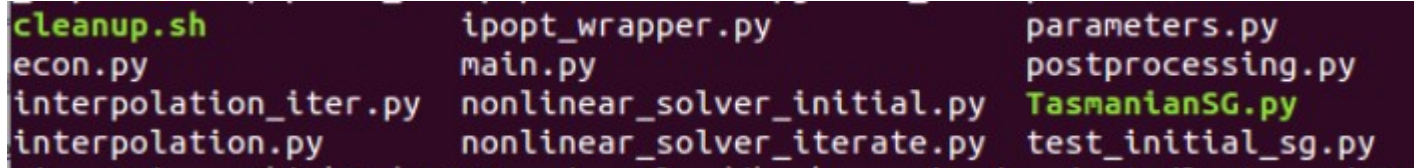
**$D$**  for the GP regression and  **$d$**  for the ASGP regression.

The complete details of this **relationship are not entirely understood theoretically** and are well beyond the scope of the present lecture. However, we observe in numerical experiments that the number of samples required by GP regression, is GPR much larger than the number of samples required by AS-GPR,

$$N_{\text{GPR}} \gg N_{\text{AS-GPR}} \text{ when } D \gg d.$$

# Setup of Code

Lecture\_3/code/growth\_model\_GPR



```
cleanup.sh      ipopt_wrapper.py  parameters.py
econ.py         main.py           postprocessing.py
interpolation_iter.py  nonlinear_solver_initial.py  TasmanianSG.py
interpolation.py  nonlinear_solver_iterate.py  test_initial_sg.py
```

**main.py:** driver routine

**econ.py:** contains production function, utility,...

**nonlinear\_solver\_initial/iterate.py:** interface GPR  $\leftrightarrow$  IPOPT.

**ipopt\_wrapper.py:** specifies the optimization problem  
(objective function,...).

**interpolation.py:** interface value function iteration  $\leftrightarrow$  sparse grid.

**postprocessing.py:** auxiliary routines, e.g., to compute the error.

# Code snippet – main.py

```

import nonlinear_solver_initial as solver      #solves opt. problems for terminal VF
import nonlinear_solver_iterate as solviter    #solves opt. problems during VFI
from parameters import *                     #parameters of model
import interpolation as interpol              #interface to sparse grid library/terminal VF
import interpolation_iter as interpol_iter     #interface to sparse grid library/iteration
import postprocessing as post                #computes the L2 and Linfinity error of the model
import numpy as np

#=====
# Start with Value Function Iteration

for i in range(numstart, numits):
    # terminal value function
    if (i==1):
        print "start with Value Function Iteration"
        interpol.GPR_init(i)

    else:
        print "Now, we are in Value Function Iteration step", i
        interpol_iter.GPR_iter(i)

#=====
print "=====
print " "
print " Computation of a growth model of dimension ", n_agents , " finished after ", numits, " steps"
print " "
print "=====
#=====

# compute errors
avg_err=post.ls_error(n_agents, numstart, numits, No_samples_postprocess)

#=====
print "=====
print " "
#print " Errors are computed -- see error.txt"
print " "
print "=====
#=====

```

# Code snippet – parameters.py

```
#=====
# How many training points for GPR
n_agents= 1 # number of continuous dimensions of the model
No_samples = 10*n_agents

# control of iterations
numstart = 1 # which is iteration to start (numstart = 1: start from scratch, number=/0: restart)
numits = 7 # which is the iteration to end

filename = "restart/restart_file_step_" #folder with the restart/result files

#=====

# Model Paramters

beta=0.8
rho=0.95
zeta=0.5
psi=0.36
gamma=2.0
delta=0.025
eta=1
big_A=(1.0-beta)/(psi*beta)

# Ranges For States
k_bar=0.2
k_up=3.0
range_cube = k_up - k_bar # range of  $[0..1]^d$  in 1D

# Ranges for Controls
c_bar=1e-2
c_up=10.0

l_bar=1e-2
l_up=10.0

inv_bar=1e-2
inv_up=10.0

#=====

# Number of test points to compute the error in the postprocessing
No_samples_postprocess = 20
```

# Code snippet – ipopt\_wrapper.py

```

=====
# Objective Function to start VFI (in our case, the value function)

def EV_F(X, k_init, n_agents):

    # Extract Variables
    cons=X[0:n_agents]
    lab=X[n_agents:2*n_agents]
    inv=X[2*n_agents:3*n_agents]

    knext= (1-delta)*k_init + inv

    # Compute Value Function
    VT_sum=utility(cons, lab) + beta*V_INFINITY(knext)

    return VT_sum

# V infinity
def V_INFINITY(k=[]):
    e=np.ones(len(k))
    c=output_f(k,e)
    v_infinity=utility(c,e)/(1-beta)
    return v_infinity

=====
# Objective Function during VFI (note - we need to interpolate on an "old" GPR)

def EV_F_ITER(X, k_init, n_agents, gp_old):

    # Extract Variables
    cons=X[0:n_agents]
    lab=X[n_agents:2*n_agents]
    inv=X[2*n_agents:3*n_agents]

    knext= (1-delta)*k_init + inv

    #transform to comp. domain of the model
    knext_cube = box_to_cube(knext)

    # initialize correct data format for training point
    s = (1,n_agents)
    Xtest = np.zeros(s)
    Xtest[0,:] = knext_cube

    # interpolate the function, and get the point-wise std.
    V_old, sigma_test = gp_old.predict(Xtest, return_std=True)

    VT_sum = utility(cons, lab) + beta*V_old

    return VT_sum

```



# Code snippet – interpolate\_iter.py

```
def GPR_iter(iteration):

    # Load the model from the previous iteration step
    restart_data = filename + str(iteration-1) + ".pcl"
    with open(restart_data, 'rb') as fd_old:
        gp_old = pickle.load(fd_old)
        print "data from iteration step ", iteration -1 , "loaded from disk"
    fd_old.close()

    ##generate sample aPoints
    np.random.seed(666) #fix seed
    dim = n_agents
    Xtraining = np.random.uniform(k_bar, k_up, (No_samples, dim))
    y = np.zeros(No_samples, float) # training targets

    # solve bellman equations at training points
    for iI in range(len(Xtraining)):
        y[iI] = solver.iterate(Xtraining[iI], n_agents, gp_old)[0]

    #for iI in range(len(Xtraining)):
        #print Xtraining[iI], y[iI]

    # Instantiate a Gaussian Process model
    #kernel = 1.0 * RBF(length_scale=1.0, length_scale_bounds=(1e-1, 10.0))

    | kernel = 1.0 * RBF(length_scale=100.0, length_scale_bounds=(1e-1, 2e2)) \
      + WhiteKernel(noise_level=1, noise_level_bounds=(1e-3, 1e+0))

    #kernel = 1.0 * RBF(length_scale=100.0, length_scale_bounds=(1e-1, 2e2))
    #kernel = 1.0 * Matern(length_scale=1.0, length_scale_bounds=(1e-1, 10.0), nu=1.5)

    gp = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=10)

    # Fit to data using Maximum Likelihood Estimation of the parameters
    gp.fit(Xtraining, y)

    ##save the model to a file
    output_file = filename + str(iteration) + ".pcl"
    print output_file
    with open(output_file, 'wb') as fd:
        pickle.dump(gp, fd, protocol=pickle.HIGHEST_PROTOCOL)
        print "data of step ", iteration , " written to disk"
        print " -----"
    fd.close()
```

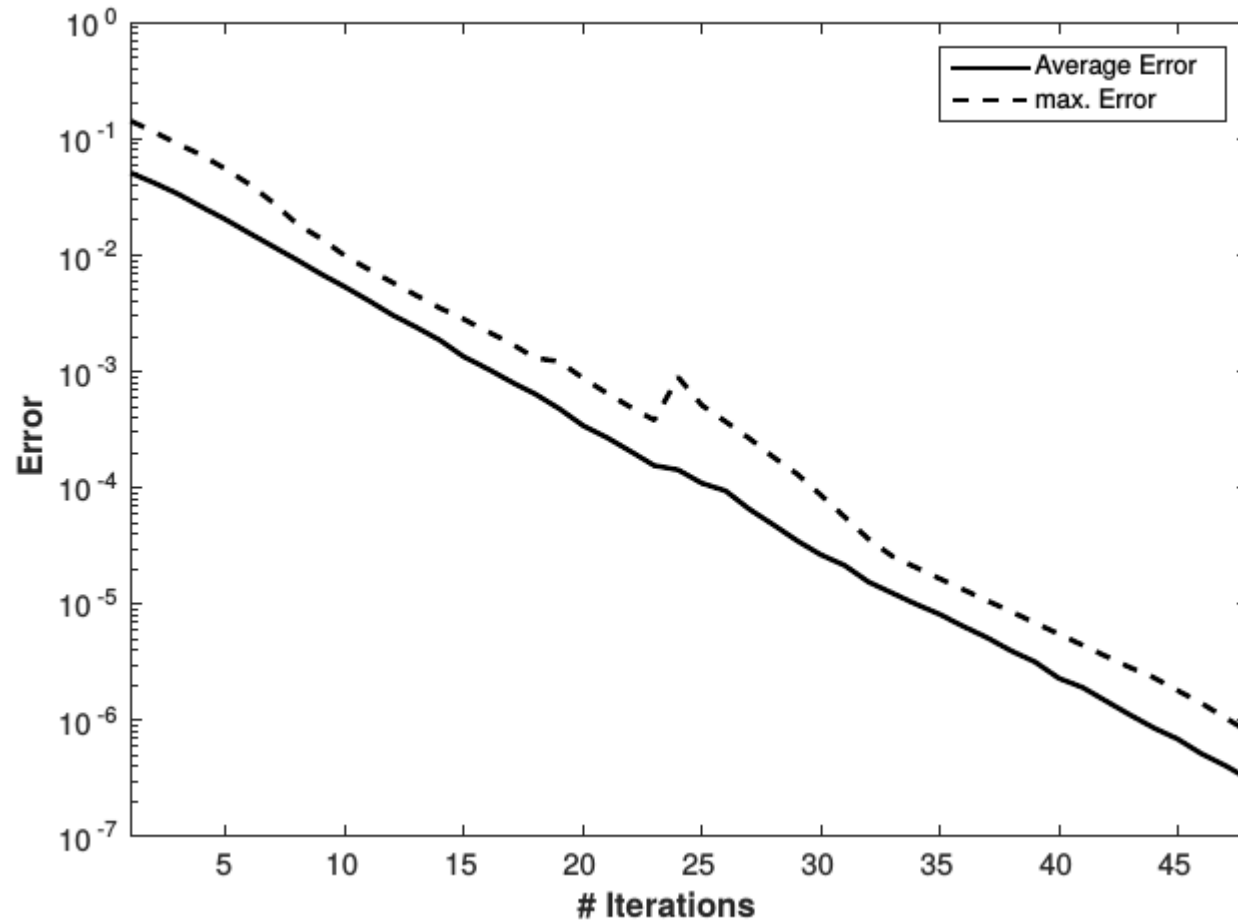


# Run the Growth model code

- Model implemented in Python (scikit-learn)
- Optimizer used: IPOPT & PYIPOPT (python interface)
- Lecture\_3/code/growth\_model\_GPR
- run with

**\$ python main.py**

# 1d Growth Model – converges



# Pin-down a 1d VF by GPs

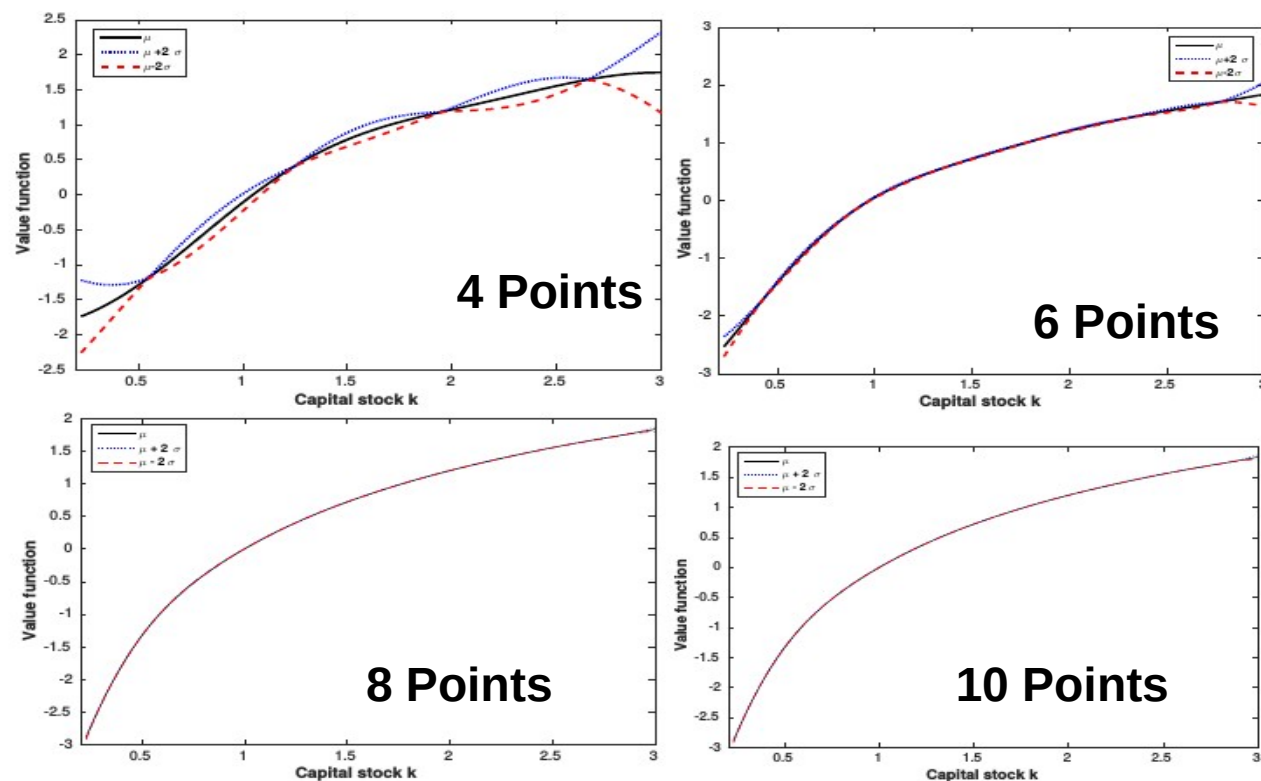
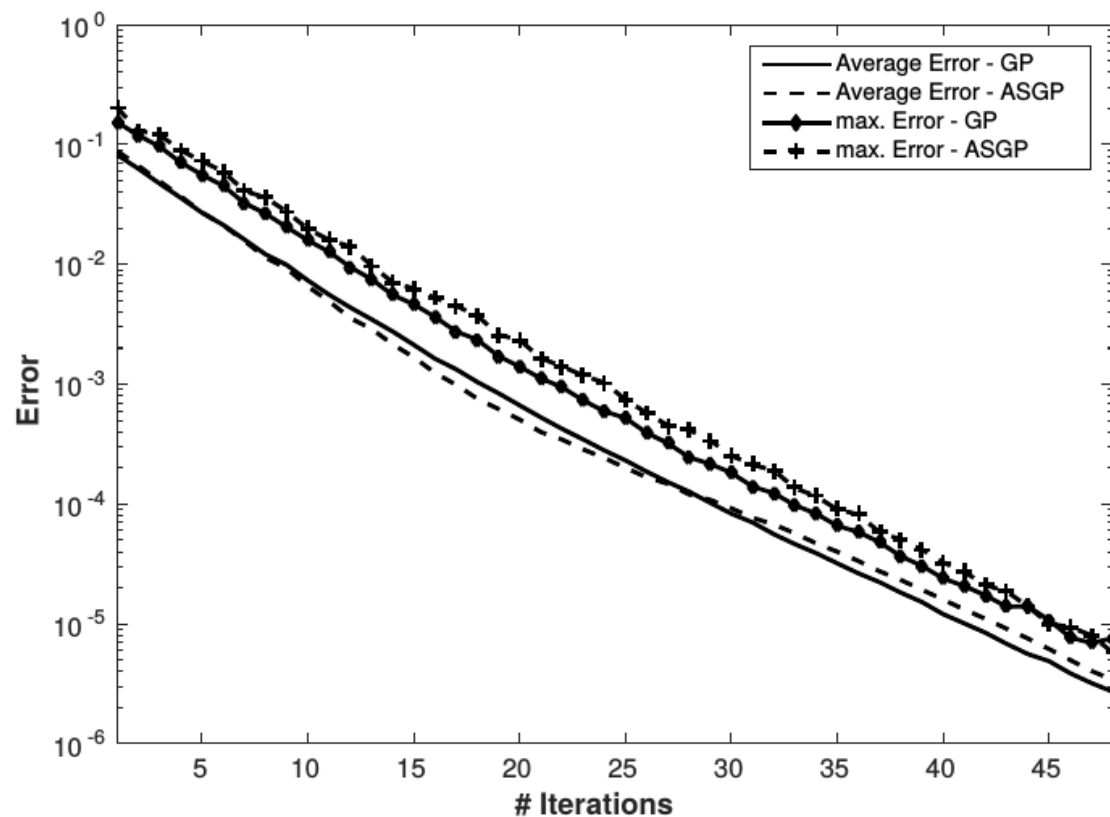
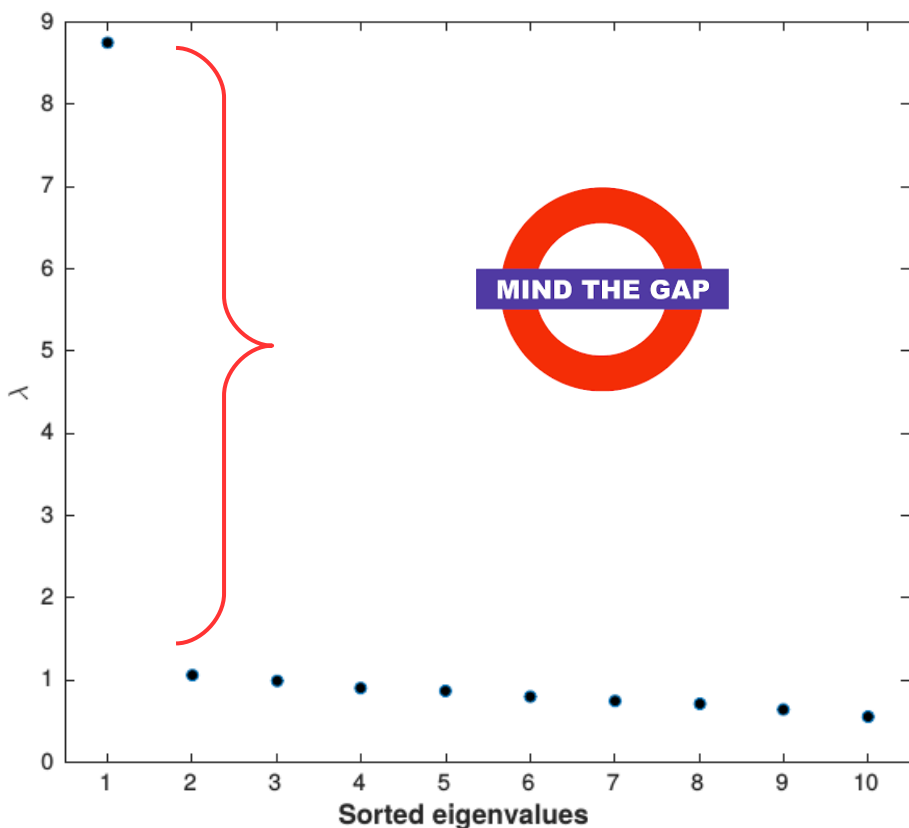


Figure 5: Above's four panels display the predictive mean value function from a 1-dimensional growth model at convergence, and the 95% confidence interval. The upper left figure was constructed based on only 4 sample points in the state space, the upper right on 6 sample points. The lower left was a result of 8 sample points, and the lower right is based on 10 sample points that pin down the function.

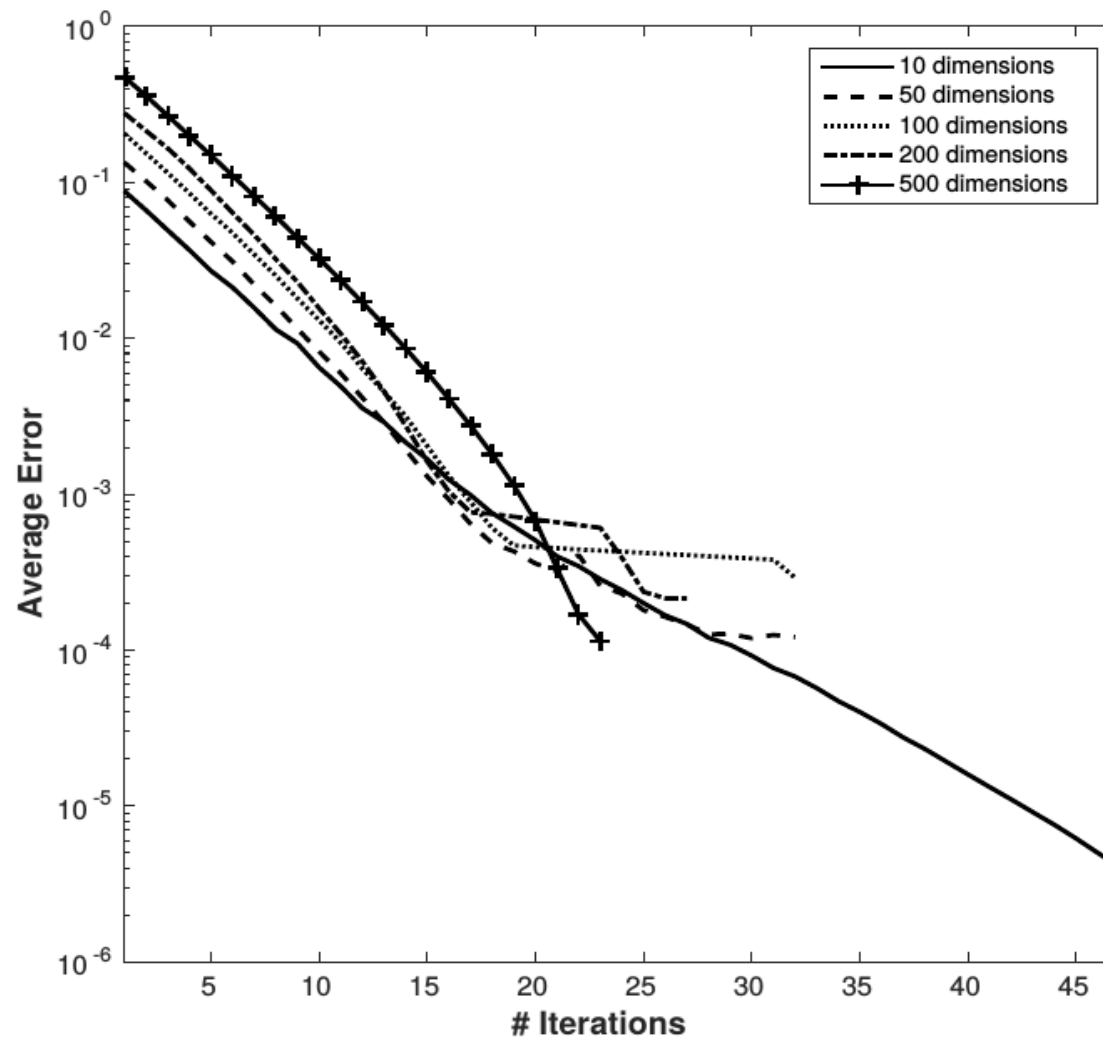
# 10d Growth Model – converges



**Left panel:** sorted eigenvalues of the 10-dimensional OG model.

**Right panel:** decreasing maximum and average error for 10-D OG model, computed either by GPs or ASGP with an AS of dimension 1, respectively.

# Growth model – 1 active subspace



# Context

- **500d growth model**: every individual observation used to train the ASGP consists of solving an optimization problem with **500 continuous states** and **1,500 controls**.
- **far beyond what has been done so far in the literature** in the context of computing global solutions to economic problems. Cai & Judd (2014), for example or up to **4 continuous states**.
- Note that **adaptive sparse grid-based solution algorithms** (e.g. Brumm & Scheidegger (2017)) as well as algorithms that are based on Smolyak's method (Judd et. al. (2014), Kruger & Kubler (2004)) **would not be able solve models of this size**.
- Brumm & Scheidegger (2017) were able to compute global solutions for models up to **100 dimensions** with adaptive sparse grids and a massively parallelized code, whereas Kruger & Kubler (2004) deal with up to **20 continuous states** when employing Smolyak's method.
- Their underlying **data structures of sparse grids become so complex** and too slow to operate on that they in practice become **un - operational** for problems of this size.
- the very high dimensionality required, e.g., in large-scale multi-country OLG models.

## Dynamic Programming on irregularly-shaped geometries

- Natural modelling geometry of economic models often not a hypercube, but rather **a simplex** (e.g., Brumm & Kubler (2014)), or a **hyper-ball** kind of ergodic set (Judd et. al. (2014), Maliar & Maliar (2015)).
  - We need to be able to perform value function iteration or time iteration on “arbitrary/irregularly-shaped” geometries.
  - Want to focus on the region of interest (**limited computational budget**).

# Recall – GPs

(e.g. Murphy (2012), Rasmussen & Williams (2006), with references therein)

GPs: grid-free way of constructing interpolators

→ I can add geometry-free observations to  $D^*$ !

Training set:  $D = \{(\mathbf{x}_i, y_i) | i = 1, \dots, n\}$

$$\begin{pmatrix} \mathbf{f} \\ \mathbf{f}_* \end{pmatrix} \sim \mathcal{N} \left( \begin{pmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}_* \end{pmatrix}, \begin{pmatrix} \mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{pmatrix} \right) \quad \begin{aligned} p(\mathbf{f}_* | \mathbf{X}_*, \mathbf{X}, \mathbf{f}) &= \mathcal{N}(\mathbf{f}_* | \boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*) \\ \boldsymbol{\mu}_* &= \boldsymbol{\mu}(\mathbf{X}_*) + \mathbf{K}_*^T \mathbf{K}^{-1} (\mathbf{f} - \boldsymbol{\mu}(\mathbf{X})) \\ \boldsymbol{\Sigma}_* &= \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{K}_* \end{aligned}$$

Test point = interpolation at  $\mathbf{X}^*$

\*this feature is also useful in case the solver does not converge at some point!



# Simulate the economy

Judd et. al. (2014), Maliar & Maliar (2015)

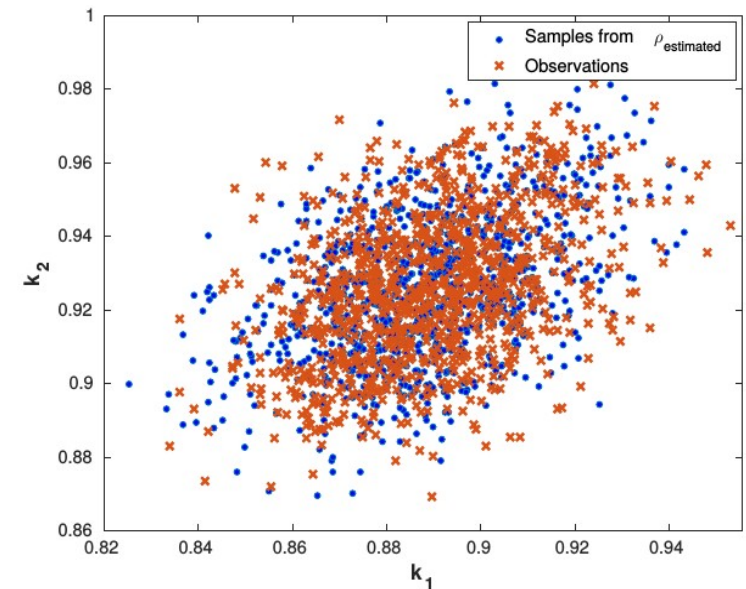
## 1. Simulate economy at few points, learn the policy

$$k_j^+ = (1 - \delta) \cdot k_j + I_j + \epsilon_j, \quad j = 1, \dots, D$$

## 2. Approximate density with mixture of Gaussians (e.g. Rasmussen (2000), Blei & Jordan (2005))

$$\rho_{estimated}(\mathbf{x}) = \sum_{m=1}^M \pi_m \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)$$

→ generate training data; plug them into VFI



# DP on ergodic sets

**Data:** Initial guess  $V_{next}$  for the next period's value function. Approximation accuracy  $\bar{\eta}$ .

**Result:** The  $m$  (approximate) equilibrium policy functions  $\xi^*$  and the corresponding value function  $V^*$  on  $\Omega_{ergodic}$ .

Set iteration step  $s = 1$ .

**while**  $\eta > \bar{\eta}$  **do**

    Determine  $\Omega_{ergodic}$  by using (64) and (65).

    Generate  $n$  training inputs  $\mathbf{X}_{ergodic} = \{\mathbf{k}_i^s : 1 \leq i \leq n\} \in \Omega_{ergodic}$

**for**  $\mathbf{k}_i^s \in \mathbf{X}_{ergodic}$  **do**

        Evaluate the Bellman operator  $TV^s(\mathbf{k}_i^s)$  (see Eq. (5))

        given next period's value function  $V_{next}$ .

        Set the training targets for the value function:  $t_i = TV(\mathbf{k}_i^s)$ .

        If required, set the training targets to learn the  $j$ -th policy function:

$\xi_{j_i}(\mathbf{k}_i^s) \in \arg \max_{p_j} TV(\mathbf{k}_i^s)$ .

**end**

    Set  $\mathbf{t}_{ergodic} = \{t_i : 1 \leq i \leq n\}$ .

    Given  $\{\mathbf{X}_{ergodic}, \mathbf{t}_{ergodic}\}$ , learn a surrogate  $V_{surrogate}$  of  $V$ .

    Set  $\xi_j = \{\xi_{j_i} : 1 \leq i \leq n\}$ .

    Given  $\{\mathbf{X}_{ergodic}, \xi_j\}$ , learn a surrogate of the policy  $\xi_j$ .

    Calculate (an approximation for) the error, e.g.:  $\eta = \|V_{surrogate} - V_{next}\|_\infty$ .

    Set  $V_{next} = V_{surrogate}$ .

    Set  $s = s + 1$ .

**end**

$V^* = V_{surrogate}$ .

$u^* = \{\xi_1, \dots, \xi_m\}$ .

► Probably only every 5 to 10 steps...

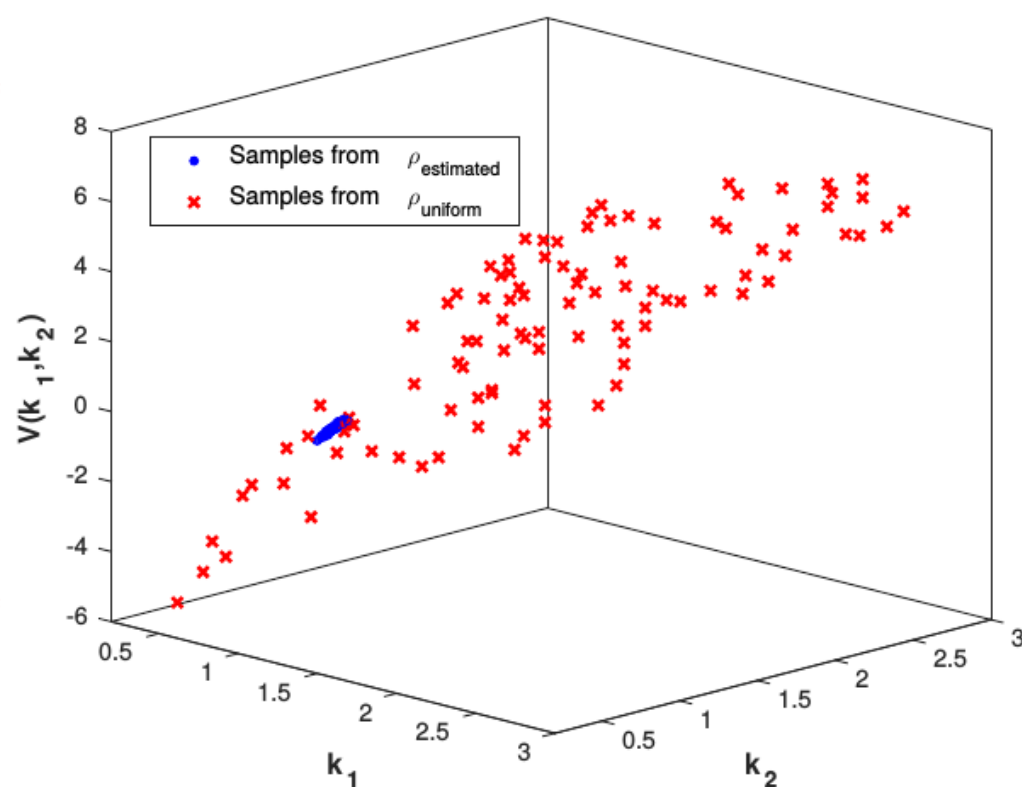
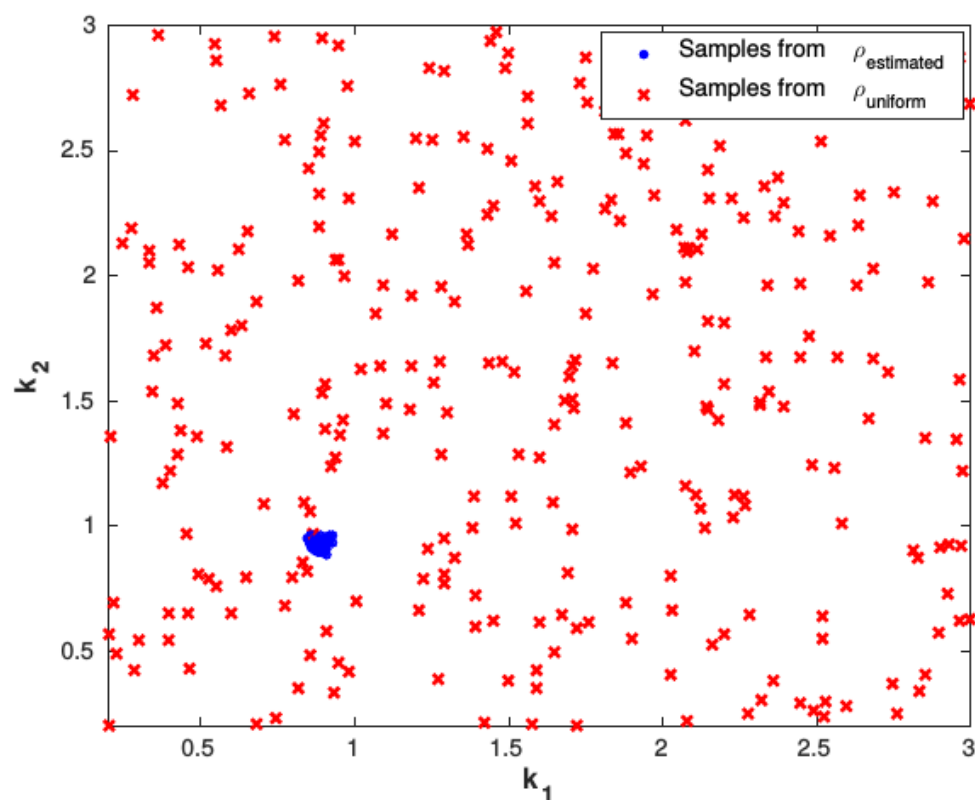
$$\mathbf{X}_{ergodic} = \left\{ \mathbf{x}_{ergodic}^{(1)}, \dots, \mathbf{x}_{ergodic}^{(N)} \right\}$$

$$\mathbf{t}_{ergodic} = \left\{ t_{ergodic}^{(1)}, \dots, t_{ergodic}^{(N)} \right\}$$

$$\{\mathbf{X}_{ergodic}, \mathbf{t}_{ergodic}\} \in \Omega_{ergodic}$$

**Algorithm 2:** Overview of the critical steps of the VFI algorithm that operates on the ergodic set  $\Omega_{ergodic}$ .

# Focus resources where needed



# Uncertainty Quantification (UQ)

- Uncertainty quantification (UQ) is a field of applied mathematics concerned with the **quantification and propagation of uncertainties** through computational models.
- In the quantitative economics community, UQ should be of paramount interest, as it can help to address questions such as **which parameters are driving the conclusions derived from an economic model**.
- Answering them, in turn, can inform the researcher for example **on which parts of the model she needs to focus** on when calibrating it.

There are various sources of uncertainty that can enter economic models, including:

- i) **parameter uncertainty** (robustness of results)
- ii) **interpolation uncertainty** (more data reduce uncertainty)

# Enlarge the state space

- Standard approaches require a **large number of model evaluations**.
- **We get away with a single model evaluation!**
- Since we can deal with very high-dimensional problems, we simply **enlarge the state space** by the parameters of interest, e.g.:

$$\tilde{S} = (\mathbf{k}_t, \gamma)$$

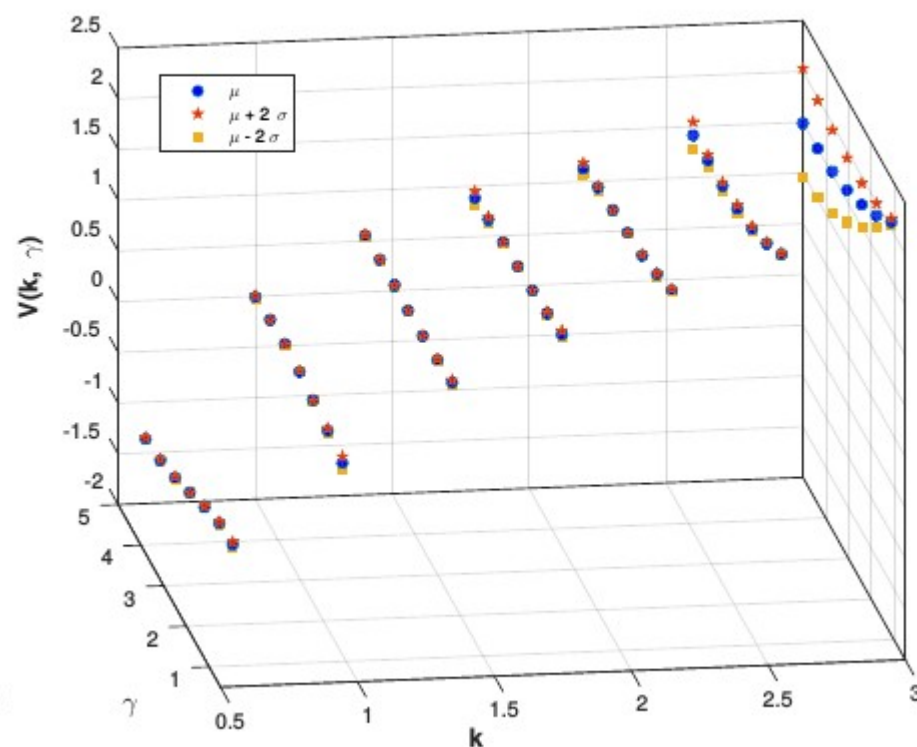
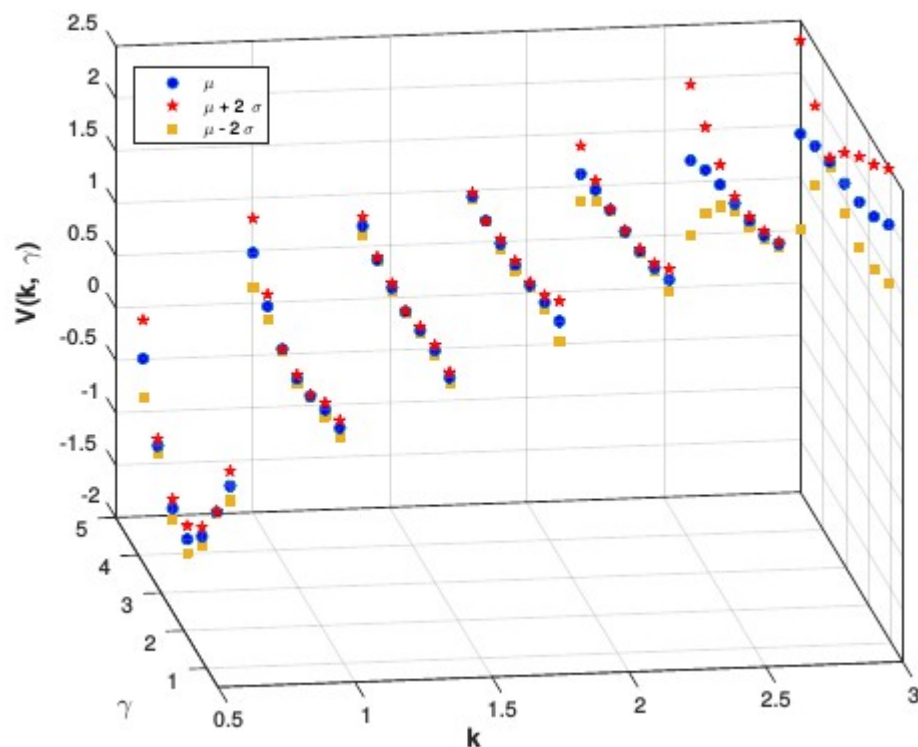
In the growth model, we simply set it to:

$$[\underline{\mathbf{k}}, \overline{\mathbf{k}}]^D \times [\underline{\gamma}, \overline{\gamma}] = [0.2, 3]^D \times [0.5, 5]$$

- no calibration exercise needed!
- evaluate, e.g., **univariate impact of a parameter on the model conclusion**.
- **Bring the model and the data together.**

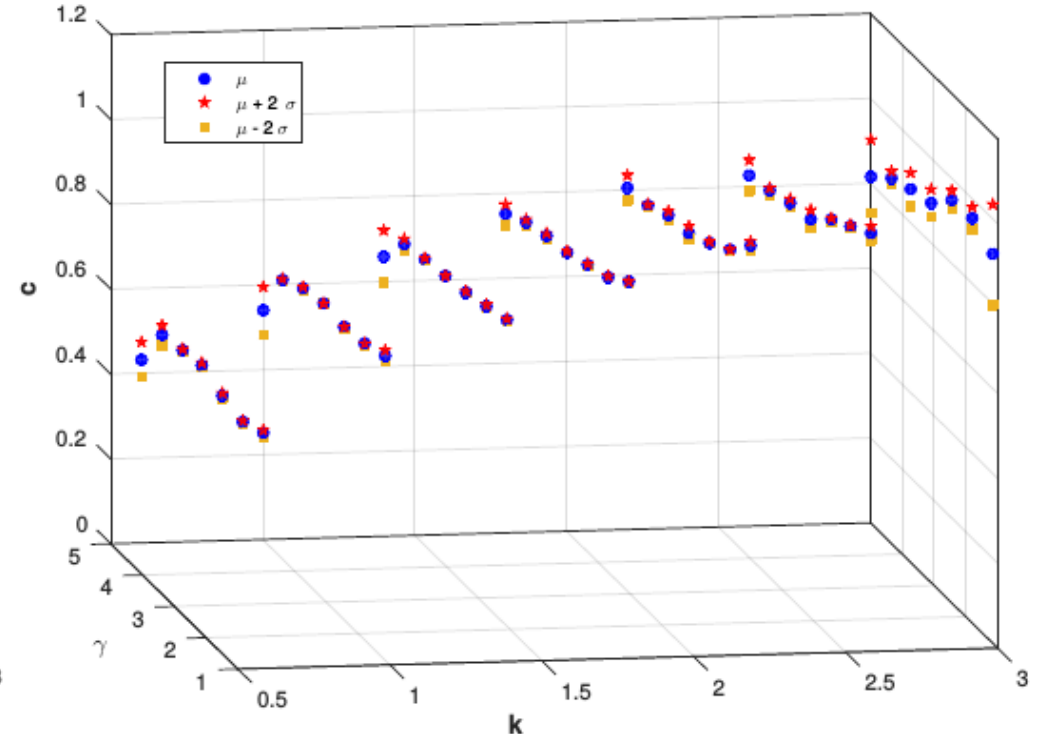
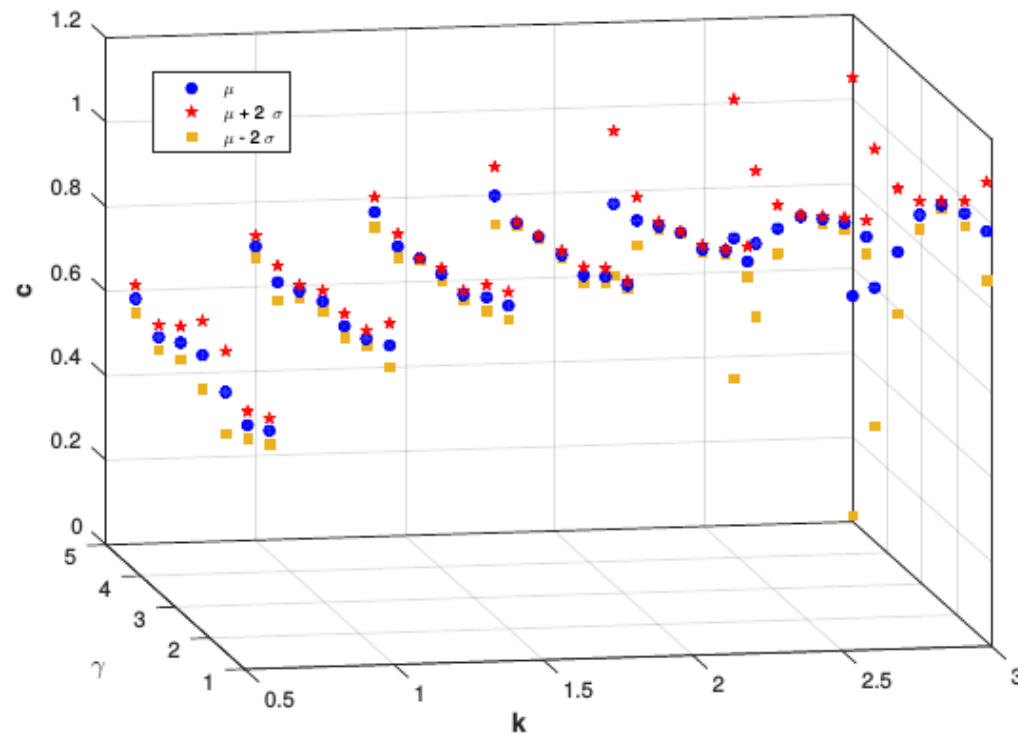
# Value function $V(k, \gamma)$

Solve a DP problem with  $\gamma$  (risk aversion) uniformly sampled from [0.5: 5]  
 Note: other distributions for parameters possible!



**Left:** predictive mean and 95% quantile of the value function (20 sample points)  
**Right:** predictive mean and 95% quantile of the value function (40 sample points)

# Consumption $c(k, \gamma)$



**Left:** predictive mean and 95% quantile of the value function (20 sample points)  
**Right:** predictive mean and 95% quantile of the value function (40 sample points)

# Convergence of ASGDP

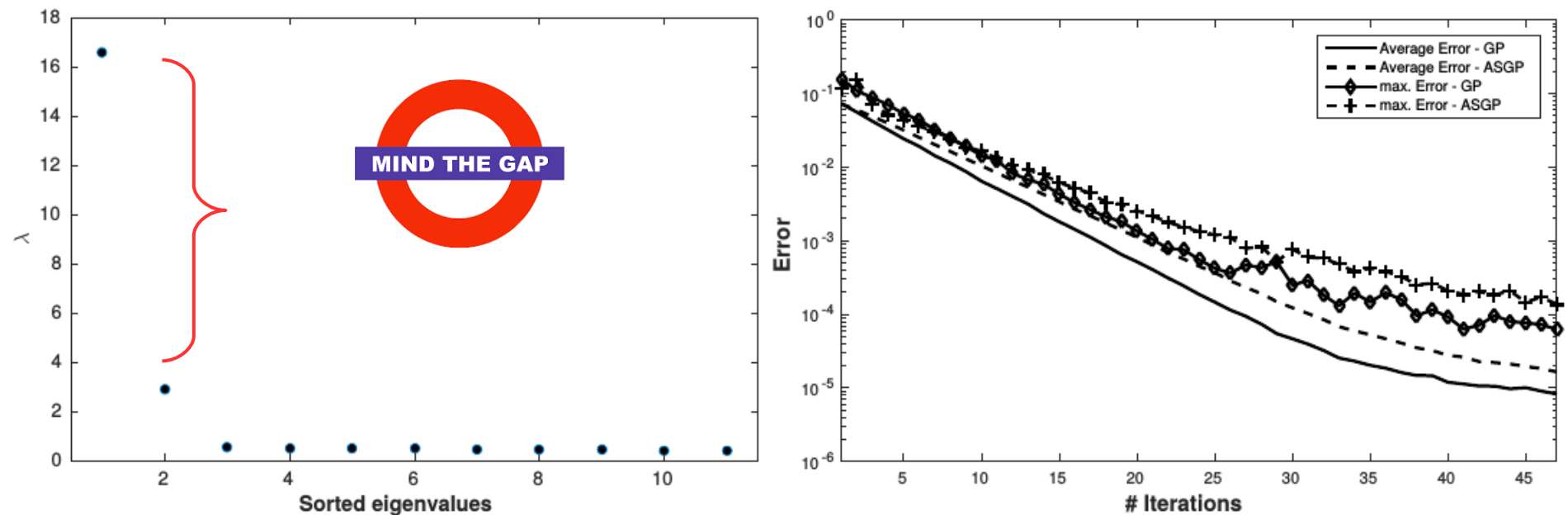


Figure 13: Left panel—Sorted eigenvalues of  $\mathbf{C}_N$  (see Eq. (42)) for the 11-dimensional OG problem with continuous states  $\tilde{S} = (k_1, \dots, k_{10}, \gamma)$ . Right panel—Decreasing maximum and average error for the two 11-dimensional OG models, computed either by GPs or by ASGP with an AS of dimension 1, respectively.)



# Quantity of interest (QoI) – an example

Jaynes (1982); Harenberg (2017), with references therein

Parameters:  $\chi = \{\gamma, \delta, \psi, \zeta, \eta\}$

Parameter	Baseline value	lower bound for $\chi_i$	upper bound for $\bar{\chi}_i$
$\gamma$	2.0	0.5	5.0
$\delta$	0.025	0.02	0.03
$\psi$	0.36	0.32	0.4
$\zeta$	0.5	0.0	1.0
$\eta$	1.0	0.0	2.0

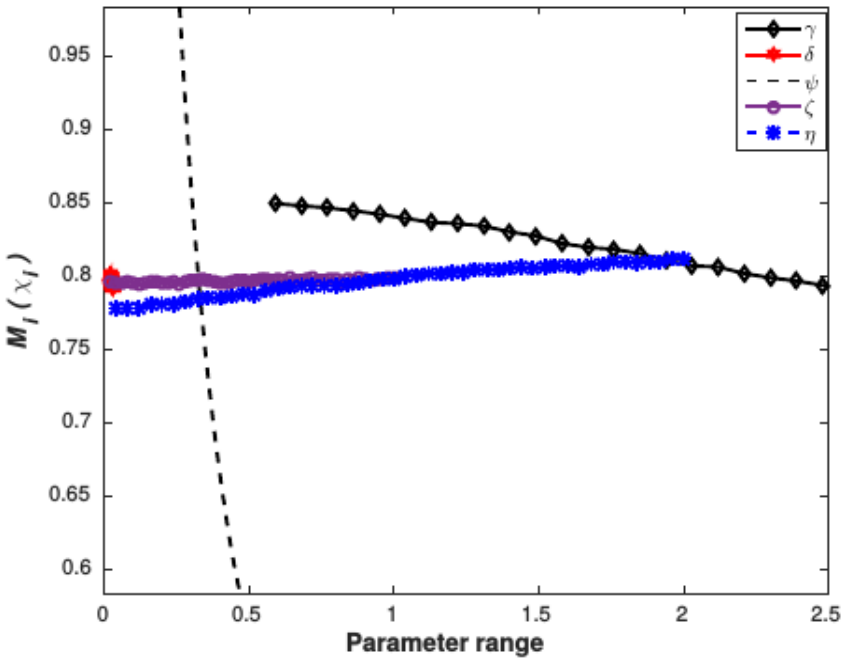
Extended state space:  $\tilde{S} = (\mathbf{k}, \chi) \rightarrow [\underline{\mathbf{k}}, \bar{\mathbf{k}}] \times [\underline{\gamma}, \bar{\gamma}] \times [\underline{\delta}, \bar{\delta}] \times [\underline{\psi}, \bar{\psi}] \times [\underline{\zeta}, \bar{\zeta}] \times [\underline{\eta}, \bar{\eta}]$ .

QoI: **aggregate production**:  $\mathcal{M}(\chi) = \mathbb{E}[f]$

Univariate effects:

$$\mathcal{M}_i(\chi_i) = \mathbb{E}[\mathcal{M}(\Theta | \Theta_i = \chi_i)]$$

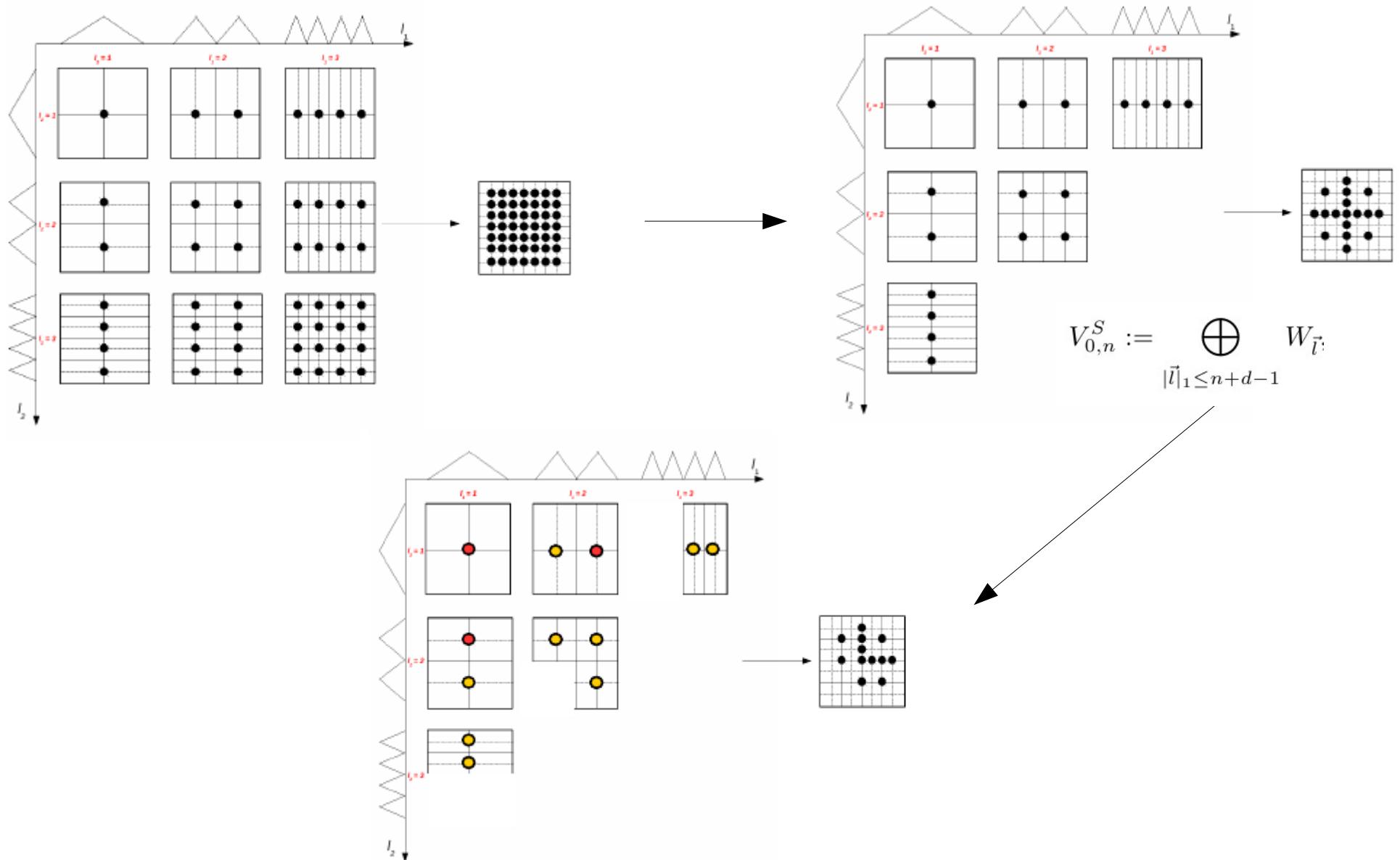
- commonly interpreted as a robust relationship between an input parameter and the QoI.
- **A global solution method required here.**



# Wrap-up of the lecture-suite



# Introduction to Sparse Grids and Adaptive Sparse Grids



# Dynamic Programming and Time Iteration with Sparse Grids

$$V(\underline{\mathbf{k}}) = \max_{\mathbf{I}, \mathbf{c}, \mathbf{l}} \left( u(c, l) + \beta \left\{ \underline{V_{next}(k^+)} \right\} \right),$$

*s.t.*

$$k_j^+ = (1 - \delta) \cdot k_j + I_j, \quad j = 1, \dots, D$$

$$\Gamma_j = \frac{\zeta}{2} k_j \left( \frac{I_j}{k_j} - \delta \right)^2, \quad j = 1, \dots, D$$

$$\sum_{j=1}^D (c_j + I_j - \delta \cdot k_j) = \sum_{j=1}^D (f(k_j, l_j) - \Gamma_j)$$

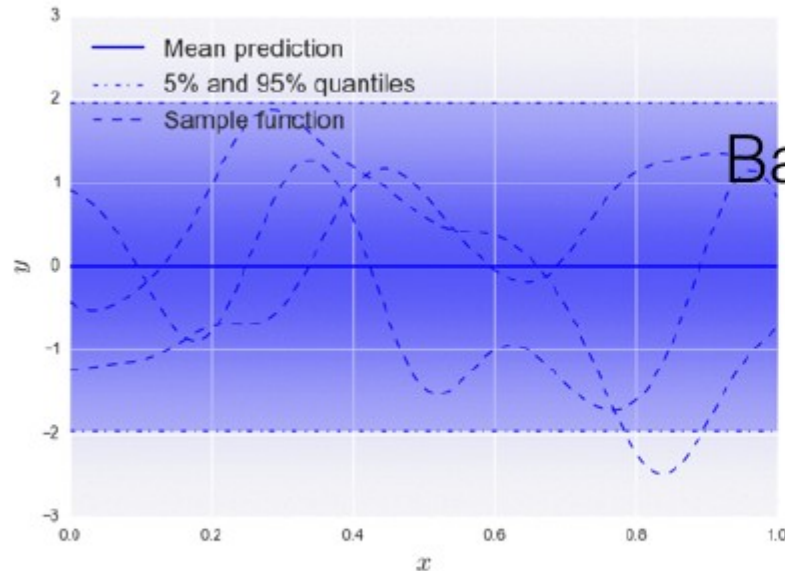
**State  $\mathbf{k}$ :** sparse grid coordinates

$V_{\text{next}}$  : sparse grid interpolator from the previous iteration step

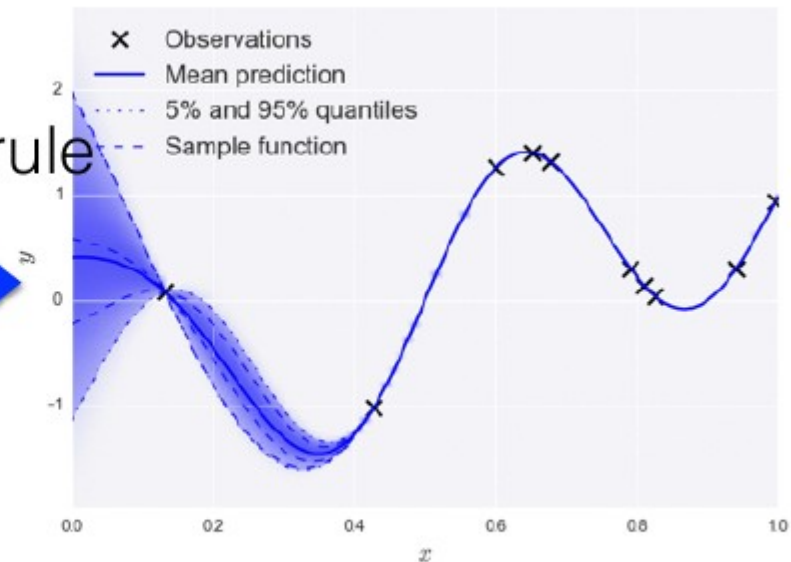
**Solve this optimization problem at every point in the sparse grid!**

**Attention: Take care of the econ domain/ sparse grid domain**

# Intro to ML and Basics on Gaussian Process Regression



Prior GP



Posterior GP

Training set:  $D = \{(\mathbf{x}_i, y_i) | i = 1, \dots, n\}$

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

$$\boldsymbol{\mu}_* = \boldsymbol{\mu}(\mathbf{X}_*) + \mathbf{K}_*^T \mathbf{K}^{-1} (\mathbf{f} - \boldsymbol{\mu}(\mathbf{X}))$$

$$\boldsymbol{\Sigma}_* = \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{K}_*$$

Test point = interpolation at  $\mathbf{X}^*$

→ **predictive mean**  $\mu_* = \mathbb{E}(f_*)$

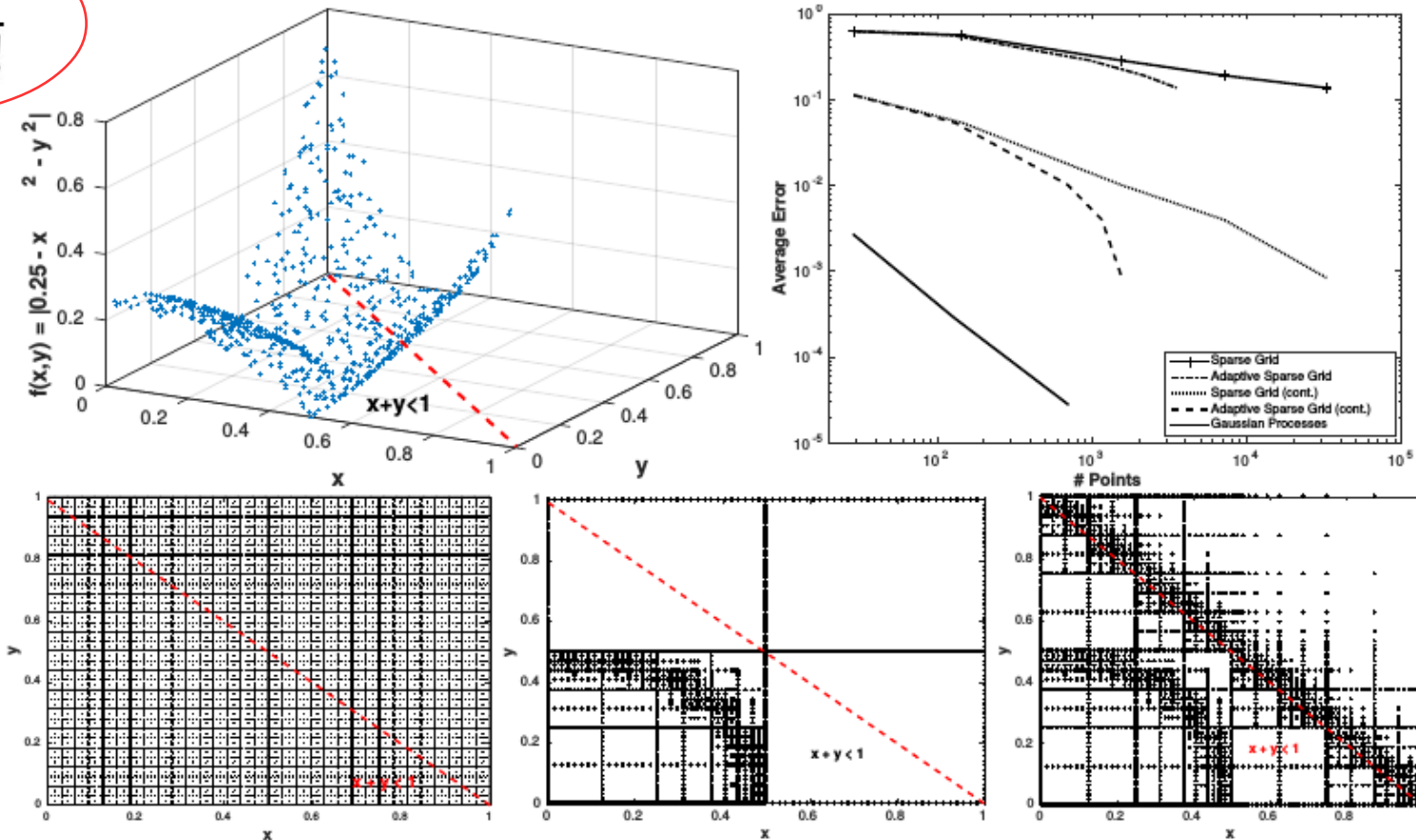
→ **Confidence Intervals!**

Where we have data, we have high confidence in our predictions.

Where we do not have data, we cannot be too confident about our predictions.

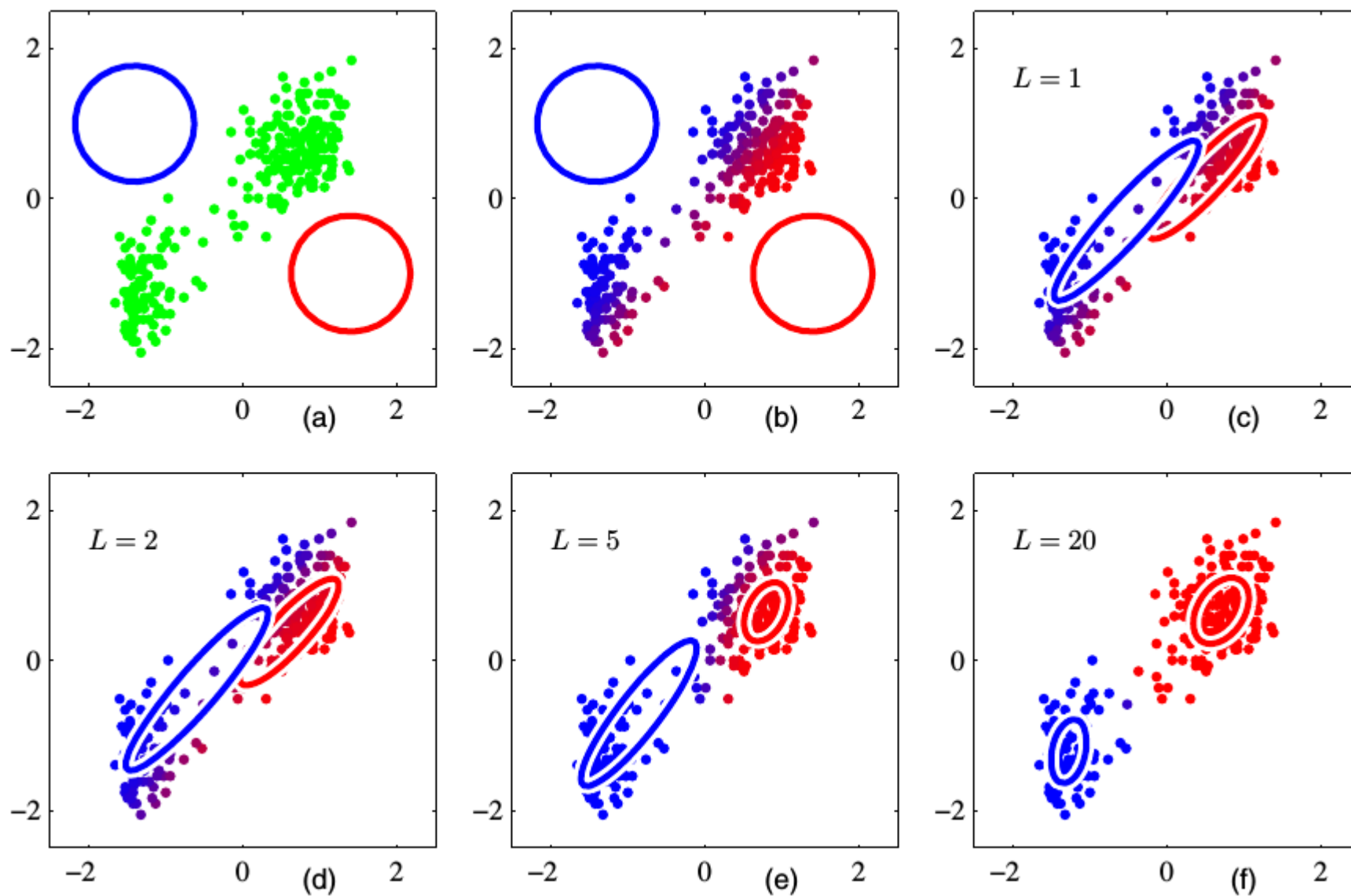
# Non-hypercubic domain: GPR versus ASG

$$\text{Vol}_{\Delta} = \frac{1}{D!}$$



**Figure:** The upper left panel shows the analytical test function evaluated at random test points on the simplex. The upper right panel displays a comparison of the interpolation error for GPs, sparse grids, and adaptive sparse grids of varying resolution and constructed under the assumption that a continuation value exists, (denoted by "cont"), or that there is no continuation value. The lower left panel displays a sparse grid consisting of 32,769 points. The lower middle panel shows an adaptive sparse grid (cont) that consists of 1,563 points, whereas the lower right panel shows an adaptive sparse grid, constructed with 3,524 points and under the assumption that the function outside  $\Delta$  is not known.

# “noisy” GPR and GMM



# Introduction to Active Subspaces and DP on irregularly-shaped domains

