


# From Linear Models to Gaussian Processes

**Mark van der Wilk**

Department of Computing  
Imperial College London

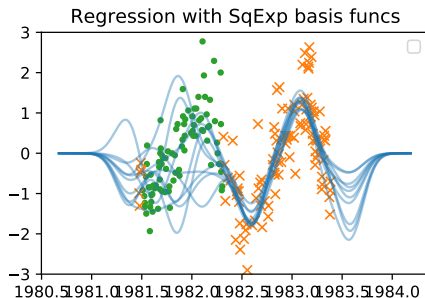
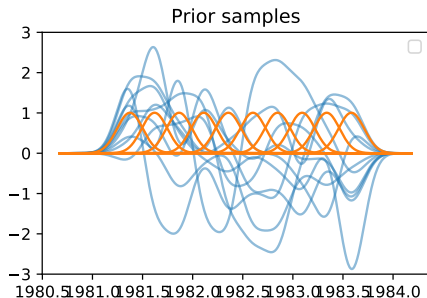
 @markvanderwilk  
m.vdwilk@imperial.ac.uk

January 23, 2023

# Recap

Last lecture we saw that:

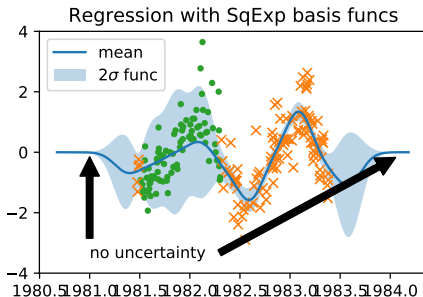
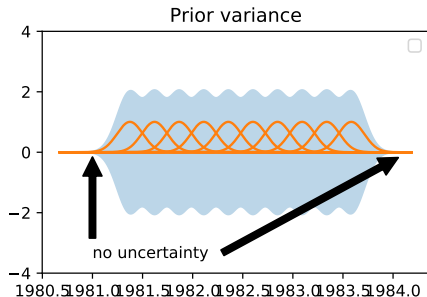
- ▶ The prior has a large effect on the predictions & we needed a sensible prior.
- ▶ Polynomial bases didn't lead to a sensible prior, squared exponential did.
- ▶ We needed many basis functions to ensure sensible uncertainty.



# Recap

Last lecture we saw that:

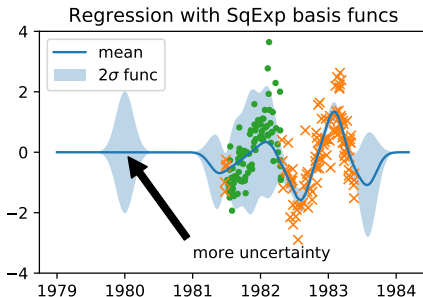
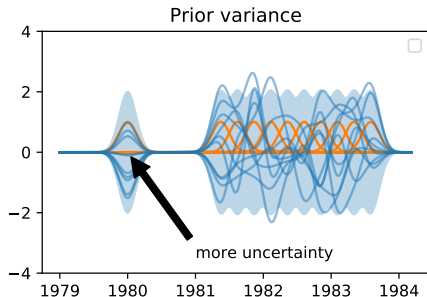
- ▶ The prior has a large effect on the predictions & we needed a sensible prior.
- ▶ Polynomial bases didn't lead to a sensible prior, squared exponential did.
- ▶ We needed many basis functions to ensure sensible uncertainty.



# Recap

Last lecture we saw that:

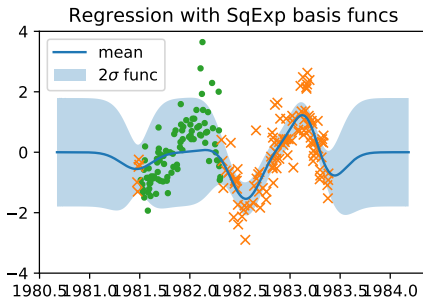
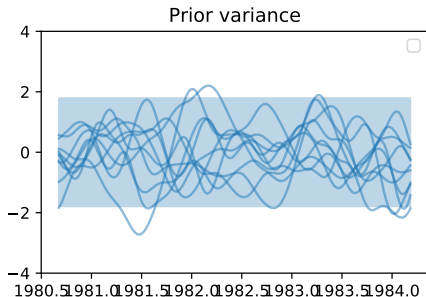
- ▶ The prior has a large effect on the predictions & we needed a sensible prior.
- ▶ Polynomial bases didn't lead to a sensible prior, squared exponential did.
- ▶ We needed many basis functions to ensure sensible uncertainty.



# Recap

Last lecture we saw that:

- ▶ The prior has a large effect on the predictions & we needed a sensible prior.
- ▶ Polynomial bases didn't lead to a sensible prior, squared exponential did.
- ▶ We needed many basis functions to ensure sensible uncertainty.

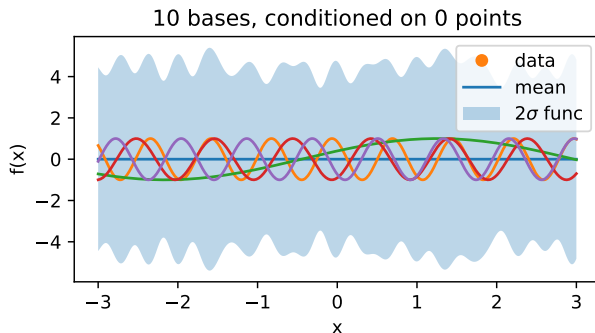


# Today

We will see:

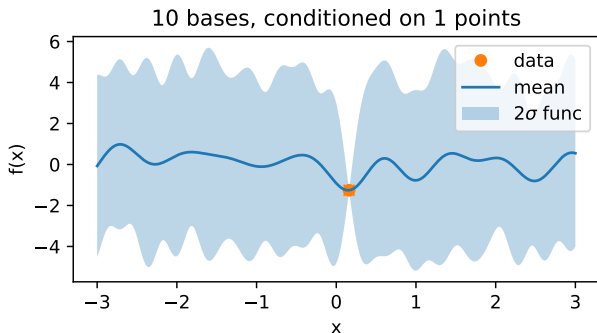
- ▶ By considering computational cost, we derive the Gaussian process view of BLR.
- ▶ This is the kernel trick!
- ▶ What is a Gaussian process.
- ▶ How to find posteriors in GP models.

# Infinite Basis Functions: Another Reason



- ▶ Non-local basis functions give uncertainty everywhere (like polynomials).
- ▶ But uncertainty decreases non-locally!
- ▶ For  $M$  bases, when we condition on  $M$  points: full certainty!

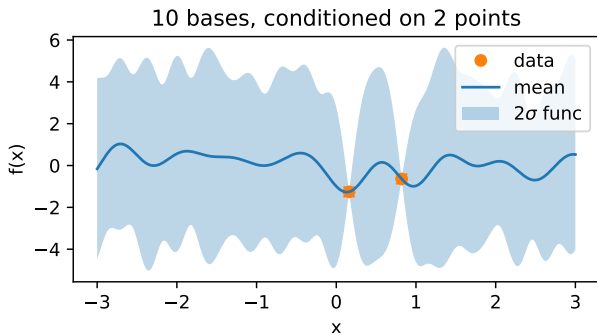
# Infinite Basis Functions: Another Reason



- ▶ Non-local basis functions give uncertainty everywhere (like polynomials).
- ▶ But uncertainty decreases non-locally!
- ▶ For  $M$  bases, when we condition on  $M$  points: full certainty!

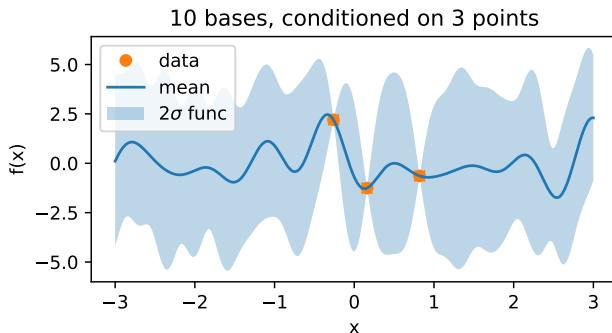


# Infinite Basis Functions: Another Reason



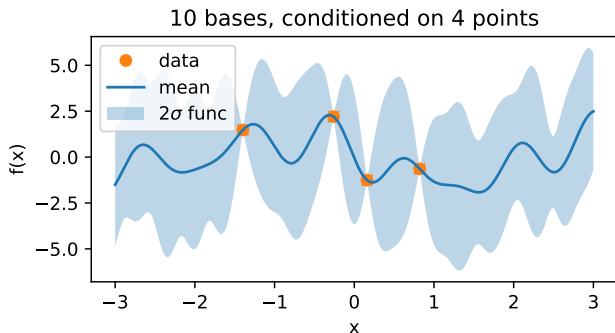
- ▶ Non-local basis functions give uncertainty everywhere (like polynomials).
- ▶ But uncertainty decreases non-locally!
- ▶ For  $M$  bases, when we condition on  $M$  points: full certainty!

# Infinite Basis Functions: Another Reason



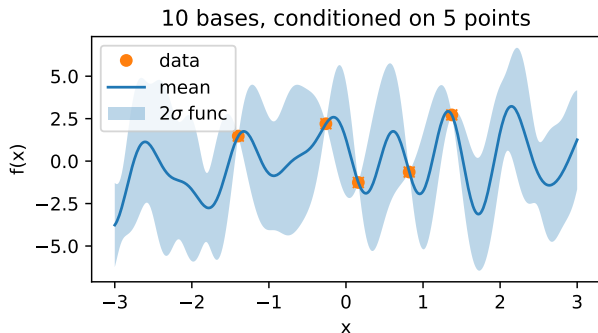
- ▶ Non-local basis functions give uncertainty everywhere (like polynomials).
- ▶ But uncertainty decreases non-locally!
- ▶ For  $M$  bases, when we condition on  $M$  points: full certainty!

# Infinite Basis Functions: Another Reason



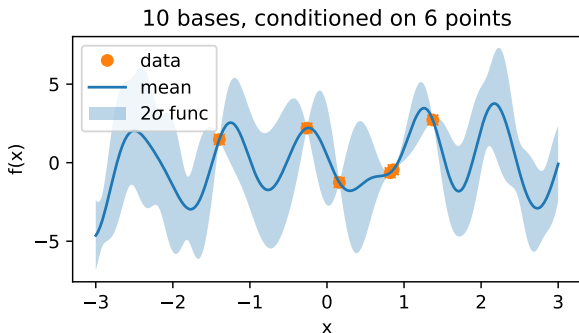
- ▶ Non-local basis functions give uncertainty everywhere (like polynomials).
- ▶ But uncertainty decreases non-locally!
- ▶ For  $M$  bases, when we condition on  $M$  points: full certainty!

# Infinite Basis Functions: Another Reason



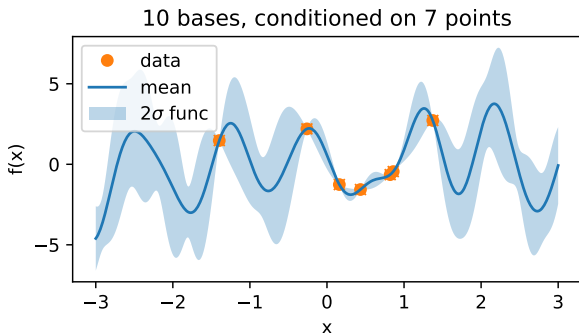
- ▶ Non-local basis functions give uncertainty everywhere (like polynomials).
- ▶ But uncertainty decreases non-locally!
- ▶ For  $M$  bases, when we condition on  $M$  points: full certainty!

# Infinite Basis Functions: Another Reason



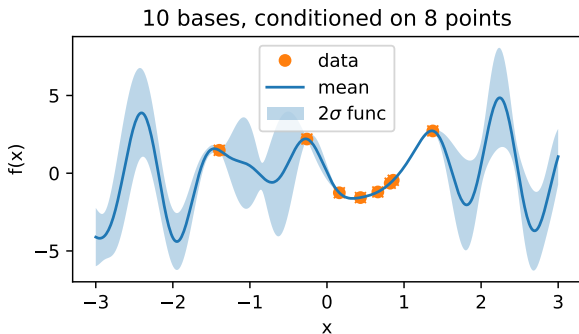
- ▶ Non-local basis functions give uncertainty everywhere (like polynomials).
- ▶ But uncertainty decreases non-locally!
- ▶ For  $M$  bases, when we condition on  $M$  points: full certainty!

# Infinite Basis Functions: Another Reason



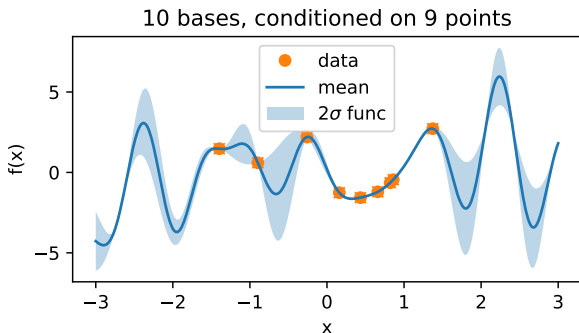
- ▶ Non-local basis functions give uncertainty everywhere (like polynomials).
- ▶ But uncertainty decreases non-locally!
- ▶ For  $M$  bases, when we condition on  $M$  points: full certainty!

# Infinite Basis Functions: Another Reason



- ▶ Non-local basis functions give uncertainty everywhere (like polynomials).
- ▶ But uncertainty decreases non-locally!
- ▶ For  $M$  bases, when we condition on  $M$  points: full certainty!

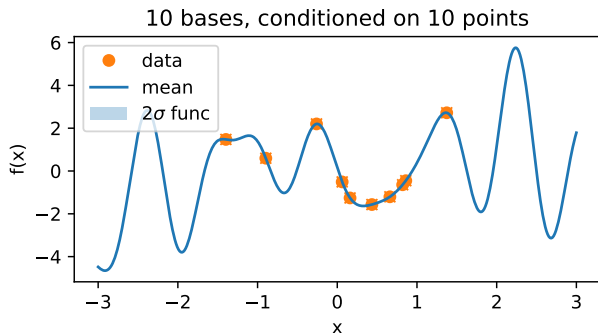
# Infinite Basis Functions: Another Reason



- ▶ Non-local basis functions give uncertainty everywhere (like polynomials).
- ▶ But uncertainty decreases non-locally!
- ▶ For  $M$  bases, when we condition on  $M$  points: full certainty!

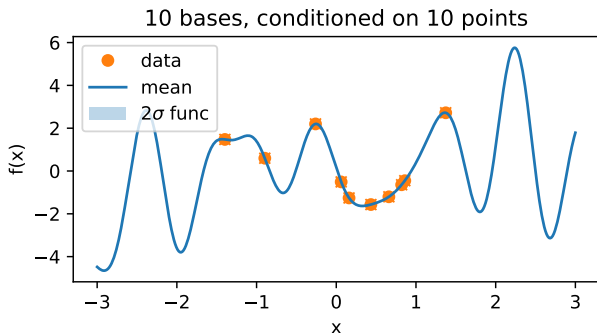


# Infinite Basis Functions: Another Reason



- ▶ Non-local basis functions give uncertainty everywhere (like polynomials).
- ▶ But uncertainty decreases non-locally!
- ▶ For  $M$  bases, when we condition on  $M$  points: full certainty!

# Infinite Basis Functions: Another Reason



- ▶ Non-local basis functions give uncertainty everywhere (like polynomials).
- ▶ But uncertainty decreases non-locally!
- ▶ For  $M$  bases, when we condition on  $M$  points: full certainty!

**Exercise:** For a model with  $M$  bases, show that after conditioning on  $M$  points (zero variance likelihood) leads to 1) completely certain predictions, 2) certain posterior.

# BLR: Computing the Posterior

For Gaussian models, finding conditionals can easily be done by finding the **joint**, and then applying the **Gaussian conditioning rule**.

$$\boldsymbol{\theta} \sim \mathcal{N}(0, \mathbf{I}_M), \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}_N \sigma^2), \quad [\Phi(\mathbf{X})]_{nm} = \phi_m(\mathbf{x}_n). \quad (1)$$

$$\boldsymbol{\theta} \in \mathbb{R}^M, \quad \mathbf{y} \in \mathbb{R}^N, \quad \boldsymbol{\epsilon} \in \mathbb{R}^N, \quad \Phi(\mathbf{X}) \in \mathbb{R}^{N \times M}, \quad \mathbf{X} \in \mathbb{R}^{N \times D}. \quad (2)$$

$$\begin{bmatrix} \boldsymbol{\theta} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathbf{I}_M & 0 \\ \Phi(\mathbf{X}) & \mathbf{I}_N \end{bmatrix} \begin{bmatrix} \boldsymbol{\theta} \\ \boldsymbol{\epsilon} \end{bmatrix} \quad (3)$$

$$\implies p\left(\begin{bmatrix} \boldsymbol{\theta} \\ \mathbf{y} \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\theta} \\ \mathbf{y} \end{bmatrix}; 0, \begin{bmatrix} \mathbf{I}_M & \Phi(\mathbf{X})^\top \\ \Phi(\mathbf{X}) & \Phi(\mathbf{X})\Phi(\mathbf{X})^\top + \sigma^2 \mathbf{I}_N \end{bmatrix}\right) \quad (4)$$

Using:

- ▶ Linear relationships between Gaussian RVs gives Gaussian joint.
  - ▶ Write joint Gaussian as a linear transformation of RVs **with known independent distributions**.
- ▶  $\mathbb{E}_{\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})}[\mathbf{A}\mathbf{x}] = \mathbf{A}\boldsymbol{\mu}$ , and  $\mathbb{V}_{\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})}[\mathbf{A}\mathbf{x}] = \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^\top$ .

# BLR: Computing the Posterior

Gaussian conditioning formula (will be provided in exam):

$$p\left(\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}; \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}\right) \quad (5)$$

$$p(\mathbf{x}_2|\mathbf{x}_1) = \mathcal{N}\left(\mathbf{x}_2; \boldsymbol{\mu}_2 + \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}(\mathbf{x}_1 - \boldsymbol{\mu}_1), \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12}\right) \quad (6)$$

$p(\mathbf{x}_1|\mathbf{x}_2)$  is similar, and can be obtained by reordering the vector to  $\begin{bmatrix} \mathbf{x}_2 & \mathbf{x}_1 \end{bmatrix}^\top$ . You can find the covariance matrix for this ordering in terms of the covariance blocks that are given above.

# BLR: Computing the Posterior

Gaussian conditioning formula (will be provided in exam):

$$p\left(\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}; \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}\right) \quad (5)$$

$$p(\mathbf{x}_2|\mathbf{x}_1) = \mathcal{N}\left(\mathbf{x}_2; \boldsymbol{\mu}_2 + \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}(\mathbf{x}_1 - \boldsymbol{\mu}_1), \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12}\right) \quad (6)$$

$p(\mathbf{x}_1|\mathbf{x}_2)$  is similar, and can be obtained by reordering the vector to  $\begin{bmatrix} \mathbf{x}_2 & \mathbf{x}_1 \end{bmatrix}^\top$ . You can find the covariance matrix for this ordering in terms of the covariance blocks that are given above.

$$p\left(\begin{bmatrix} \boldsymbol{\theta} \\ \mathbf{y} \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\theta} \\ \mathbf{y} \end{bmatrix}; 0, \begin{bmatrix} \mathbf{I}_M & \Phi(\mathbf{X})^\top \\ \Phi(\mathbf{X}) & \Phi(\mathbf{X})\Phi(\mathbf{X})^\top + \sigma^2\mathbf{I}_N \end{bmatrix}\right) \quad (7)$$

$$p(\boldsymbol{\theta}|\mathbf{y}) = \mathcal{N}\left(\boldsymbol{\theta}; \Phi(\mathbf{X})^\top [\Phi(\mathbf{X})\Phi(\mathbf{X})^\top + \sigma^2\mathbf{I}_N]^{-1} \mathbf{y} \right. \\ \left. \mathbf{I}_M - \Phi(\mathbf{X})^\top [\Phi(\mathbf{X})\Phi(\mathbf{X})^\top + \sigma^2\mathbf{I}_N]^{-1} \Phi(\mathbf{X})\right) \quad (8)$$

# BLR: Computing the Posterior

Looks complicated. But we can compute it!

$$p(\boldsymbol{\theta}|\mathbf{y}) = \mathcal{N}\left(\boldsymbol{\theta}; \Phi(\mathbf{X})^\top [\Phi(\mathbf{X})\Phi(\mathbf{X})^\top + \sigma^2 \mathbf{I}_N]^{-1} \mathbf{y} \right. \\ \left. \mathbf{I}_M - \Phi(\mathbf{X})^\top [\Phi(\mathbf{X})\Phi(\mathbf{X})^\top + \sigma^2 \mathbf{I}_N]^{-1} \Phi(\mathbf{X}) \right) \quad (9)$$

What is the computational cost?

---

<sup>1</sup> $N \times N$  matrix multiplication and matrix inversion can both be  $O(N^{2.373})$ , but we assume  $O(N^3)$ . Most important is that we distinguish these expensive operations from cheaper ones that are  $O(N^2)$ .

# BLR: Computing the Posterior

Looks complicated. But we can compute it!

$$p(\boldsymbol{\theta}|\mathbf{y}) = \mathcal{N}\left(\boldsymbol{\theta}; \Phi(\mathbf{X})^\top [\Phi(\mathbf{X})\Phi(\mathbf{X})^\top + \sigma^2 \mathbf{I}_N]^{-1} \mathbf{y} \right. \\ \left. \mathbf{I}_M - \Phi(\mathbf{X})^\top [\Phi(\mathbf{X})\Phi(\mathbf{X})^\top + \sigma^2 \mathbf{I}_N]^{-1} \Phi(\mathbf{X}) \right) \quad (9)$$

What is the computational cost? We assume costs of simple linear algebra algorithms, even though more efficient algorithms exist<sup>1</sup>.

- ▶  $\Phi(\mathbf{X})$ :  $O(NMD)$  — Assume linear time cost for each dimension of input, then need to compute each basis function for each data point.

---

<sup>1</sup> $N \times N$  matrix multiplication and matrix inversion can both be  $O(N^{2.373})$ , but we assume  $O(N^3)$ . Most important is that we distinguish these expensive operations from cheaper ones that are  $O(N^2)$ .

# BLR: Computing the Posterior

Looks complicated. But we can compute it!

$$p(\boldsymbol{\theta}|\mathbf{y}) = \mathcal{N}\left(\boldsymbol{\theta}; \Phi(\mathbf{X})^\top [\Phi(\mathbf{X})\Phi(\mathbf{X})^\top + \sigma^2 \mathbf{I}_N]^{-1} \mathbf{y} \right. \\ \left. \mathbf{I}_M - \Phi(\mathbf{X})^\top [\Phi(\mathbf{X})\Phi(\mathbf{X})^\top + \sigma^2 \mathbf{I}_N]^{-1} \Phi(\mathbf{X}) \right) \quad (9)$$

What is the computational cost? We assume costs of simple linear algebra algorithms, even though more efficient algorithms exist<sup>1</sup>.

- ▶  $\Phi(\mathbf{X})$ :  $O(NMD)$  — Assume linear time cost for each dimension of input, then need to compute each basis function for each data point.
- ▶  $\Phi(\mathbf{X})\Phi(\mathbf{X})^\top$ :  $O(N^2M)$  — Matrix multiplication

---

<sup>1</sup> $N \times N$  matrix multiplication and matrix inversion can both be  $O(N^{2.373})$ , but we assume  $O(N^3)$ . Most important is that we distinguish these expensive operations from cheaper ones that are  $O(N^2)$ .



# BLR: Computing the Posterior

Looks complicated. But we can compute it!

$$p(\boldsymbol{\theta}|\mathbf{y}) = \mathcal{N}\left(\boldsymbol{\theta}; \Phi(\mathbf{X})^\top [\Phi(\mathbf{X})\Phi(\mathbf{X})^\top + \sigma^2 \mathbf{I}_N]^{-1} \mathbf{y} \right. \\ \left. \mathbf{I}_M - \Phi(\mathbf{X})^\top [\Phi(\mathbf{X})\Phi(\mathbf{X})^\top + \sigma^2 \mathbf{I}_N]^{-1} \Phi(\mathbf{X}) \right) \quad (9)$$

What is the computational cost? We assume costs of simple linear algebra algorithms, even though more efficient algorithms exist<sup>1</sup>.

- ▶  $\Phi(\mathbf{X})$ :  $O(NMD)$  — Assume linear time cost for each dimension of input, then need to compute each basis function for each data point.
- ▶  $\Phi(\mathbf{X})\Phi(\mathbf{X})^\top$ :  $O(N^2M)$  — Matrix multiplication
- ▶  $[\Phi(\mathbf{X})\Phi(\mathbf{X})^\top + \sigma^2 \mathbf{I}_N]^{-1}$  —  $O(N^3)$  Matrix inversion (or Cholesky)

<sup>1</sup> $N \times N$  matrix multiplication and matrix inversion can both be  $O(N^{2.373})$ , but we assume  $O(N^3)$ . Most important is that we distinguish these expensive operations from cheaper ones that are  $O(N^2)$ .

# Woodbury Identity (exam skill)

Usually  $M \ll N$ , so bottleneck:  $[\Phi(\mathbf{X})\Phi(\mathbf{X})^\top + \sigma^2 \mathbf{I}_N]^{-1} \sim O(N^3)$

- ▶ Annoying that we have to compute an  $O(N^3)$  cost inverse when the matrix we want is only  $\mathbb{R}^{M \times M}$ .
- ▶ Also, note that  $\Phi(\mathbf{X})\Phi(\mathbf{X})^\top$  is at most rank  $M$ ! **Low rank** matrices are usually cheaper to deal with!

---

<sup>2</sup>Matrix cookbook recipe 156

# Woodbury Identity (exam skill)

Usually  $M \ll N$ , so bottleneck:  $[\Phi(\mathbf{X})\Phi(\mathbf{X})^\top + \sigma^2 \mathbf{I}_N]^{-1} \sim O(N^3)$

- ▶ Annoying that we have to compute an  $O(N^3)$  cost inverse when the matrix we want is only  $\mathbb{R}^{M \times M}$ .
- ▶ Also, note that  $\Phi(\mathbf{X})\Phi(\mathbf{X})^\top$  is at most rank  $M$ ! **Low rank** matrices are usually cheaper to deal with!

Woodbury Identity<sup>2</sup>:

$$\underbrace{(\mathbf{A} + \mathbf{UBV})^{-1}}_{N \times N} = \mathbf{A}^{-1} - \mathbf{A}^{-1} \mathbf{U} \underbrace{(\mathbf{B}^{-1} + \mathbf{VA}^{-1} \mathbf{U})^{-1}}_{M \times M} \mathbf{VA}^{-1} \quad (10)$$

$$\mathbf{A} \in \mathbb{R}^{N \times N}, \quad \mathbf{U} \in \mathbb{R}^{N \times M}, \quad \mathbf{V} \in \mathbb{R}^{M \times N}, \quad \mathbf{B} \in \mathbb{R}^{M \times M} \quad (11)$$

---

<sup>2</sup>Matrix cookbook recipe 156

# BLR: Cheap Posterior Mean

Let's start with the mean:

$$\mu_{\theta} = \Phi(X)^{\top} [\Phi(X)\Phi(X)^{\top} + \sigma^2 I_N]^{-1} \mathbf{y} \quad (12)$$

and take  $A = \sigma^2 I_N$ ,  $U = \Phi(X)$ ,  $B = I_M$ ,  $V = \Phi(X)^{\top}$ :

$$\begin{aligned} [\Phi(X)\Phi(X)^{\top} + \sigma^2 I_N]^{-1} &= \frac{I_N}{\sigma^2} - \frac{\Phi(X)}{\sigma^2} \left[ I_M + \frac{\Phi(X)^{\top} \Phi(X)}{\sigma^2} \right]^{-1} \frac{\Phi(X)^{\top}}{\sigma^2} \\ \therefore \mu_{\theta} &= \sigma^{-2} \Phi(X)^{\top} \left[ I_N - \frac{\Phi(X)}{\sigma^2} [I_M + \sigma^{-2} \Phi(X)^{\top} \Phi(X)]^{-1} \Phi(X)^{\top} \right] \mathbf{y} \\ &= \left[ I_M - \sigma^{-2} \Phi(X)^{\top} \Phi(X) [I_M + \sigma^{-2} \Phi(X)^{\top} \Phi(X)]^{-1} \right] \sigma^{-2} \Phi(X)^{\top} \mathbf{y} \\ &= \left[ \left[ I_M + \cancel{\sigma^{-2} \Phi(X)^{\top} \Phi(X)} \right] - \cancel{\sigma^{-2} \Phi(X)^{\top} \Phi(X)} \right] \\ &\quad [I_M + \sigma^{-2} \Phi(X)^{\top} \Phi(X)]^{-1} \sigma^{-2} \Phi(X)^{\top} \mathbf{y} \end{aligned} \quad (13)$$

# BLR: Cheap Posterior Mean

$$\mu_{\theta} = [I_M + \sigma^{-2}\Phi(\mathbf{X})^{\top}\Phi(\mathbf{X})]^{-1}\sigma^{-2}\Phi(\mathbf{X})^{\top}\mathbf{y} \quad (14)$$

Now we can compute in:

- ▶  $\Phi(\mathbf{X})$ :  $O(NMD)$  — As earlier.
- ▶  $\Phi(\mathbf{X})^{\top}\Phi(\mathbf{X})$ :  $O(M^2N)$  — Matrix multiplication
- ▶  $[I_M + \Phi(\mathbf{X})^{\top}\Phi(\mathbf{X})]^{-1}$  —  $O(M^3)$  Matrix inversion (or Cholesky)

So when  $M \ll N$ , we now have  $O(NM^2)$ .

## BLR: Cheap Posterior Variance

We can similarly apply Woodbury to the posterior variance, just slightly differently.

Always **remember the goal!** From large inverse, to small inverse.

$$\Sigma_{\theta} = \mathbf{I}_M - \Phi(\mathbf{X})^{\top} [\Phi(\mathbf{X})\Phi(\mathbf{X})^{\top} + \sigma^2 \mathbf{I}_N]^{-1} \Phi(\mathbf{X}) \quad (15)$$

We take  $\mathbf{A}^{-1} = \mathbf{I}_M$ ,  $\mathbf{U} = \Phi(\mathbf{X})^{\top}$ ,  $\mathbf{B}^{-1} = \sigma^2 \mathbf{I}_N$ ,  $\mathbf{V} = \Phi(\mathbf{X})^{\top}$  to obtain:

$$\Sigma_{\theta} = [\mathbf{I}_M + \sigma^{-2} \Phi(\mathbf{X})^{\top} \Phi(\mathbf{X})]^{-1} \quad (16)$$

Also computable in  $O(NM^2)$ !

# Two Ways to Compute

Method 1, cost  $O(N^3 + N^2M + NMD)$ :

$$p(\boldsymbol{\theta}|\mathbf{y}) = \mathcal{N}\left(\boldsymbol{\theta}; \Phi(\mathbf{X})^\top [\Phi(\mathbf{X})\Phi(\mathbf{X})^\top + \sigma^2 \mathbf{I}_N]^{-1} \mathbf{y} \right. \\ \left. \mathbf{I}_M - \Phi(\mathbf{X})^\top [\Phi(\mathbf{X})\Phi(\mathbf{X})^\top + \sigma^2 \mathbf{I}_N]^{-1} \Phi(\mathbf{X}) \right) \quad (17)$$

Method 2, cost  $O(NM^2 + M^3 + NMD)$ :

$$p(\boldsymbol{\theta}|\mathbf{y}) = \mathcal{N}\left(\boldsymbol{\theta}; [\mathbf{I}_M + \sigma^{-2} \Phi(\mathbf{X})^\top \Phi(\mathbf{X})]^{-1} \sigma^{-2} \Phi(\mathbf{X})^\top \mathbf{y} \right. \\ \left. [\mathbf{I}_M + \sigma^{-2} \Phi(\mathbf{X})^\top \Phi(\mathbf{X})]^{-1} \right) \quad (18)$$

# Predictive Distribution

Compute predictive distribution from mean and variance of  $p(\boldsymbol{\theta}|\mathbf{y})$  was an exercise (q&a\_video\_07 notes).

1. We find the posterior parameters in some way.
2. We apply Woodbury to ensure we take a small matrix inverse.
3. We get predictions at a cost of  $O(NM^2 + M^3 + NMD)$ .

Using the parameters found by method 2:

$$p(\mathbf{y}_*|\mathbf{y}) = \mathcal{N}\left(\boldsymbol{\theta}; \quad \boldsymbol{\phi}(\mathbf{x}_*)^\top [\mathbf{I}_M + \sigma^{-2}\Phi(\mathbf{X})^\top \Phi(\mathbf{X})]^{-1} \sigma^{-2}\Phi(\mathbf{X})^\top \mathbf{y} \right. \\ \left. \boldsymbol{\phi}(\mathbf{x}_*)^\top [\mathbf{I}_M + \sigma^{-2}\Phi(\mathbf{X})^\top \Phi(\mathbf{X})]^{-1} \boldsymbol{\phi}(\mathbf{x}_*) + \sigma^2 \mathbf{I}_N \right) \quad (19)$$



## Predictive Distribution — Exercises

We can also find a different form of the predictive distribution, *without* finding the posterior over parameters first.

1. Using the method of transforming Gaussian RVs, show that the joint  $p(\mathbf{y}, y_*)$  is

$$p(\mathbf{y}, y_*) = \mathcal{N}\left(\begin{bmatrix} \mathbf{y} \\ y_* \end{bmatrix}; 0, \begin{bmatrix} \Phi(\mathbf{X})\Phi(\mathbf{X})^\top + \sigma^2 \mathbf{I}_N & \Phi(\mathbf{X})\boldsymbol{\phi}(\mathbf{x}_*) \\ \boldsymbol{\phi}(\mathbf{x}_*)^\top \Phi(\mathbf{X})^\top & \boldsymbol{\phi}(\mathbf{x}_*)^\top \boldsymbol{\phi}(\mathbf{x}_*) + \sigma^2 \end{bmatrix}\right) \quad (20)$$

2. Show that

$$\begin{aligned} p(y_* | \mathbf{y}) = \mathcal{N}\Big(y_*; & \boldsymbol{\phi}(\mathbf{x}_*)^\top \Phi(\mathbf{X})^\top [\Phi(\mathbf{X})\Phi(\mathbf{X})^\top + \sigma^2 \mathbf{I}_N]^{-1} \mathbf{y}, \\ & \boldsymbol{\phi}(\mathbf{x}_*)^\top \boldsymbol{\phi}(\mathbf{x}_*) + \sigma^2 \\ & - \boldsymbol{\phi}(\mathbf{x}_*)^\top \Phi(\mathbf{X})^\top [\Phi(\mathbf{X})\Phi(\mathbf{X})^\top + \sigma^2 \mathbf{I}_N]^{-1} \Phi(\mathbf{X})\boldsymbol{\phi}(\mathbf{x}_*) \Big) \end{aligned} \quad (21)$$

The cost of computing the predictive in this way is  $O(N^3 + N^2M + NMD)$  (like the earlier posterior).

# Infinite Basis Functions

So we said that to *properly* model uncertainty, and have a flexible enough model, we needed *many*, or even an **infinite** number of basis functions.

- ▶ For the  $O(NM^2 + M^3 + NMD)$  method, all terms contain  $M \rightarrow \infty$  because each matrix we compute grows with the features.
- ▶ For the  $O(N^3 + N^2M + NMD)$  method, the matrices we need are all of finite size...:

# Infinite Basis Functions

So we said that to *properly* model uncertainty, and have a flexible enough model, we needed *many*, or even an **infinite** number of basis functions.

- ▶ For the  $O(NM^2 + M^3 + NMD)$  method, all terms contain  $M \rightarrow \infty$  because each matrix we compute grows with the features.
- ▶ For the  $O(N^3 + N^2M + NMD)$  method, the matrices we need are all of finite size...:

$$\Phi(X)\Phi(X)^T \in \mathbb{R}^{N \times N}, \quad \Phi(X)\phi(\mathbf{x}_*) \in \mathbb{R}^{N \times 1} \quad (22)$$

# Infinite Basis Functions

So we said that to *properly* model uncertainty, and have a flexible enough model, we needed *many*, or even an **infinite** number of basis functions.

- ▶ For the  $O(NM^2 + M^3 + NMD)$  method, all terms contain  $M \rightarrow \infty$  because each matrix we compute grows with the features.
- ▶ For the  $O(N^3 + N^2M + NMD)$  method, the matrices we need are all of finite size...:

$$\Phi(X)\Phi(X)^T \in \mathbb{R}^{N \times N}, \quad \Phi(X)\phi(\mathbf{x}_*) \in \mathbb{R}^{N \times 1} \quad (22)$$

Notice that we only need **inner products** between feature vectors:

$$[\Phi(X)\Phi(X)^T]_{ij} = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j). \quad (23)$$

# Infinite Basis Functions

So we said that to *properly* model uncertainty, and have a flexible enough model, we needed *many*, or even an **infinite** number of basis functions.

- ▶ For the  $O(NM^2 + M^3 + NMD)$  method, all terms contain  $M \rightarrow \infty$  because each matrix we compute grows with the features.
- ▶ For the  $O(N^3 + N^2M + NMD)$  method, the matrices we need are all of finite size...:

$$\Phi(X)\Phi(X)^T \in \mathbb{R}^{N \times N}, \quad \Phi(X)\phi(\mathbf{x}_*) \in \mathbb{R}^{N \times 1} \quad (22)$$

Notice that we only need **inner products** between feature vectors:

$$[\Phi(X)\Phi(X)^T]_{ij} = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j). \quad (23)$$

What if I told you... there were functions that computed inner products... without computing the vector itself?

# Infinite Basis Functions

So we said that to *properly* model uncertainty, and have a flexible enough model, we needed *many*, or even an **infinite** number of basis functions.

- ▶ For the  $O(NM^2 + M^3 + NMD)$  method, all terms contain  $M \rightarrow \infty$  because each matrix we compute grows with the features.
- ▶ For the  $O(N^3 + N^2M + NMD)$  method, the matrices we need are all of finite size...:

$$\Phi(X)\Phi(X)^T \in \mathbb{R}^{N \times N}, \quad \Phi(X)\phi(\mathbf{x}_*) \in \mathbb{R}^{N \times 1} \quad (22)$$

Notice that we only need **inner products** between feature vectors:

$$[\Phi(X)\Phi(X)^T]_{ij} = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j). \quad (23)$$

What if I told you... there were functions that computed inner products... without computing the vector itself? **Kernel trick**.<sup>3</sup>

<sup>3</sup><http://oneweirdkerneltrick.com>

# Kernels: Polynomial kernel

If we can compute the matrices  $\Phi(X)\Phi(X)^\top$  and  $\Phi(X)\phi(\mathbf{x}_*)$  directly, without first computing the features, we could do computations without incurring the cost for large features!

# Kernels: Polynomial kernel

If we can compute the matrices  $\Phi(X)\Phi(X)^\top$  and  $\Phi(X)\phi(\mathbf{x}_*)$  directly, without first computing the features, we could do computations without incurring the cost for large features!

Example: Polynomial kernel

$$k(x, y) = (xy + 1)^{M-1} = \sum_{m=0}^{M-1} \binom{M-1}{m} x^m y^m = \phi(x)^\top \phi(y) \quad (24)$$

$$\text{for } M = 3, \quad \phi(x) = [1 \quad \sqrt{2}x \quad x^2]^\top \quad (25)$$



# Kernels: Polynomial kernel

If we can compute the matrices  $\Phi(X)\Phi(X)^\top$  and  $\Phi(X)\phi(\mathbf{x}_*)$  directly, without first computing the features, we could do computations without incurring the cost for large features!

Example: Polynomial kernel

$$k(x, y) = (xy + 1)^{M-1} = \sum_{m=0}^{M-1} \binom{M-1}{m} x^m y^m = \phi(x)^\top \phi(y) \quad (24)$$

$$\text{for } M = 3, \quad \phi(x) = [1 \quad \sqrt{2}x \quad x^2]^\top \quad (25)$$

We can compute very large inner products for very cheap!

# Kernels: Infinite Dimensional Feature Spaces

We can even consider infinite dimensional feature spaces, if the limit of the inner product exists!

$$\phi_m(x) = \exp\left(-\frac{(x - c_m)^2}{2\ell^2}\right), \quad c_m = \frac{m}{M} \cdot (c_{\max} - c_{\min}) \quad (26)$$

$$k(x, x') = \frac{1}{M} \sum_{m=1}^M \phi_m(x) \phi_m(x')$$

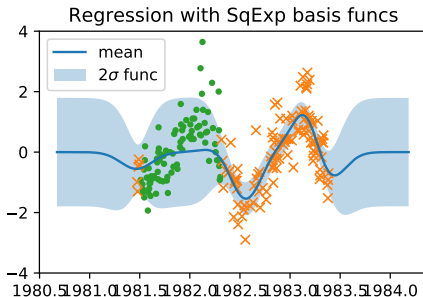
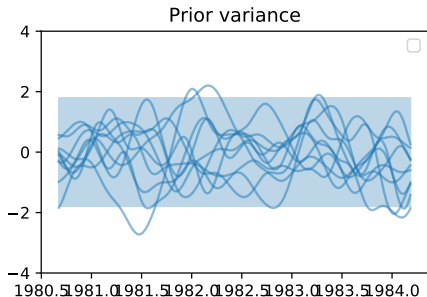
$$\begin{aligned} \lim_{M \rightarrow \infty} \frac{1}{M} \sum_{m=1}^M \phi_m(x) \phi_m(x') &= \int_{c_{\min}}^{c_{\max}} \exp\left(-\frac{(x - c)^2}{2\ell^2}\right) \exp\left(-\frac{(x' - c)^2}{2\ell^2}\right) dc \\ &= \sqrt{\pi} \ell \exp\left(-\frac{(x - x')^2}{4\ell^2}\right) \end{aligned}$$

Squared Exponential Kernel: Infinite SqExp basis functions, everywhere!

# Gaussian Process Prediction

So how do we do prediction? Just replace inner products  $\phi(\mathbf{x})^\top \phi(\mathbf{x}')$  with  $k(\mathbf{x}, \mathbf{x}')$ . Now cost is  $O(N^3 + N^2) = O(N^3)$ , down from  $\infty$  for basis funcs.

$$p(y_* | \mathbf{y}) = \mathcal{N}\left(y_*; \quad k(\mathbf{x}_*, \mathbf{X})[k(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I}_N]^{-1} \mathbf{y}, \right. \\ \left. k(\mathbf{x}_*, \mathbf{x}_*) + \sigma^2 - k(\mathbf{x}_*, \mathbf{X})[k(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I}_N]^{-1} k(\mathbf{X}, \mathbf{x}_*)\right) \quad (27)$$



# Recap

What did we do?

1. Start with a basis function model.
2. **Integrated out parameters** to directly find **predictive distribution**  $p(y_*|\mathbf{y})$ .
3. Prediction only depended on **inner products** of feature vectors.
4. We showed that we could compute inner products with a **kernel function**.
5. Computational cost down from  $\infty$  to  $O(N^3)$ .
6. Different **representation** of a basis function model.

# Recap

What did we do?

1. Start with a basis function model.
2. **Integrated out parameters** to directly find **predictive distribution**  $p(y_*|\mathbf{y})$ .
3. Prediction only depended on **inner products** of feature vectors.
4. We showed that we could compute inner products with a **kernel function**.
5. Computational cost down from  $\infty$  to  $O(N^3)$ .
6. Different **representation** of a basis function model.

... but what is a Gaussian process?

# Priors on Function Values

Another way of looking at our model:

$$p(y_n | f(\mathbf{x}_n), \mathbf{x}_n) = \mathcal{N}(y_n; f(\mathbf{x}_n), \sigma^2) \quad (28)$$

$$p(f(\mathbf{X})) = \mathcal{N}(f(\mathbf{X}); \boldsymbol{\mu}, \boldsymbol{\Sigma}) \quad (29)$$

Remember: Each parameter *implied* an entire function. So our prior placed a distribution on all the function values.

For a basis function model, find the prior on the vector of function values at each input point, denoted  $f(\mathbf{X})$ , from the prior on the weights  $p(\boldsymbol{\theta}) = \mathcal{N}(0, \mathbf{I}_M)$

$$\boldsymbol{\Sigma} = \mathbb{V}_{p(\boldsymbol{\theta})}[\Phi(\mathbf{X})\boldsymbol{\theta}] = \Phi(\mathbf{X})\Phi(\mathbf{X})^\top \quad (30)$$

# Priors on Function Values

Another way of looking at our model:

$$p(y_n | f(\mathbf{x}_n), \mathbf{x}_n) = \mathcal{N}(y_n; f(\mathbf{x}_n), \sigma^2) \quad (28)$$

$$p(f(\mathbf{X})) = \mathcal{N}(f(\mathbf{X}); \boldsymbol{\mu}, \boldsymbol{\Sigma}) \quad (29)$$

Remember: Each parameter *implied* an entire function. So our prior placed a distribution on all the function values.

For a basis function model, find the prior on the vector of function values at each input point, denoted  $f(\mathbf{X})$ , from the prior on the weights  $p(\boldsymbol{\theta}) = \mathcal{N}(0, \mathbf{I}_M)$

$$\boldsymbol{\Sigma} = \mathbb{V}_{p(\boldsymbol{\theta})}[\Phi(\mathbf{X})\boldsymbol{\theta}] = \Phi(\mathbf{X})\Phi(\mathbf{X})^\top \quad (30)$$

A Gaussian process specifies  $[\Phi(\mathbf{X})\Phi(\mathbf{X})^\top]_{ij} = \boldsymbol{\phi}(\mathbf{x}_i)^\top \boldsymbol{\phi}(\mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j)$   
**directly:**

$$p(f(\mathbf{X})) = \mathcal{N}(f(\mathbf{X}); 0, k(\mathbf{X}, \mathbf{X})) \quad (31)$$

# So what really is a Gaussian process?

See handwritten notes for:

- ▶ Definition of Gaussian process
- ▶ Gaussian processes as distributions on functions
- ▶ BLR defines a Gaussian process
- ▶ Find the posterior of a GP



# Recommended reading

- ▶ Rasmussen and Williams (2006) §2.1 + §2.2

# References I

Rasmussen, C. E. and Williams, C. K. (2006). *Gaussian processes for machine learning*. MIT press, Cambridge, MA, USA.