University College London

Department of Computer Science

COMP0083: Advanced Topics In Machine Learning

# Kernel Assignment

*Author:*
Mr. Frederico Wieser

*Student Number:*
18018699

*Last Updated:*
November 29, 2024

# 1 Feature Spaces

## 1.1 Describing A Simple Feature Space

Looking at "Figure 1" from [Art], we instantly notice two distinct groups, red crosses and blue circles, which are defined in $\mathbf{R}^2$ with $(x_1, x_2)$. One simple feature space that could be used for error-free linear classification might be $\phi(\mathbf{x}) = (1, x_1^2 + x_2^2)$. The intuition behind recommending this kind of feature space is that balls always have fixed radii, and our data seems to be made up of two groups of circular patterns. We have also added a bias term in the model.

To further show that this is true, I have recreated the example and fit a linear classification model, trying my best to estimate the data in "Figure 1" of [Art]. Using scikit-learn's "LogisticRegression" model, we were able to fit an error-free model and present the results below in 10, using this exact feature mapping.

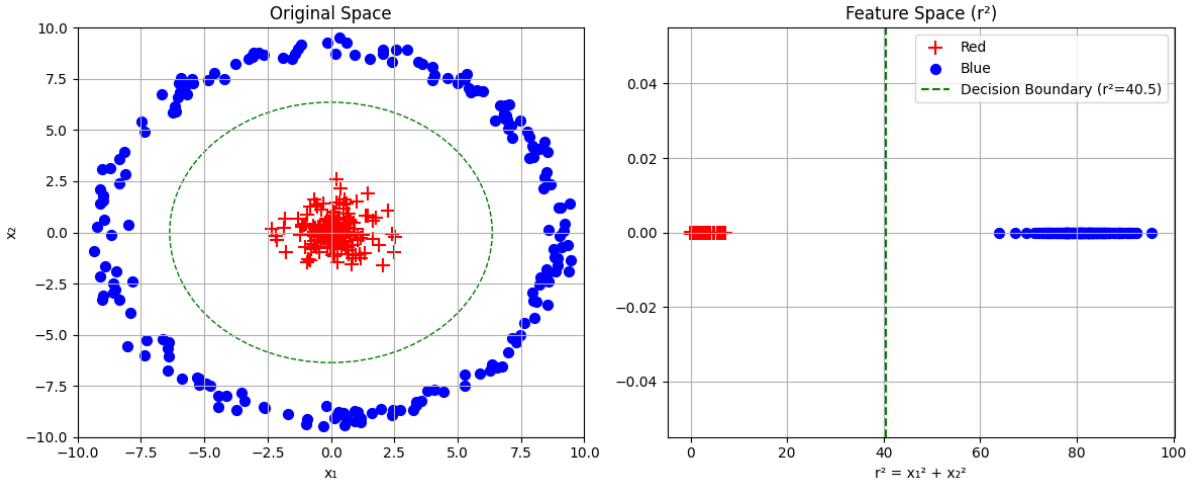


**Figure 1:** Binary classification of concentric distributions via $\phi : \mathbb{R}^2 \to \mathbb{R}^2$ where $\phi(\mathbf{x}) = (1, x_1^2 + x_2^2)$

## 1.2 Eigendecomposition of K

Let's say we have a Gram matrix:

$$K_{ij} = k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle_{\mathcal{H}}$$

Since $K$ is positive semi-definite and symmetric, we can say that the eigendecomposition is:

$$K = Q \Lambda Q^T, \quad Q = (q_1 \cdots q_m)$$

Using this form, we can equivalently say that:

$$K = Q\sqrt{\Lambda}\sqrt{\Lambda}Q^T = (Q\sqrt{\Lambda})(Q\sqrt{\Lambda})^T$$

Using this and the fact that $K = \Phi\Phi^T$, we can therefore say that $\Phi$, the feature space representation of each element $x_i \in \mathcal{X}, i \in \{1...m\}$ has the following equivalence:

$$\phi(x_i) = (q_{i1}\sqrt{\lambda_1}, \cdots, q_{im}\sqrt{\lambda_m})$$

$$\Phi = [\phi(x_1)^T, \cdots, \phi(x_m)^T] = \begin{bmatrix} q_{11}\sqrt{\lambda_1} & \cdots & q_{1m}\sqrt{\lambda_m} \\ \vdots & \ddots & \vdots \\ q_{m1}\sqrt{\lambda_1} & \cdots & q_{mm}\sqrt{\lambda_m} \end{bmatrix}$$

# 2 Kernel Dependence Detection

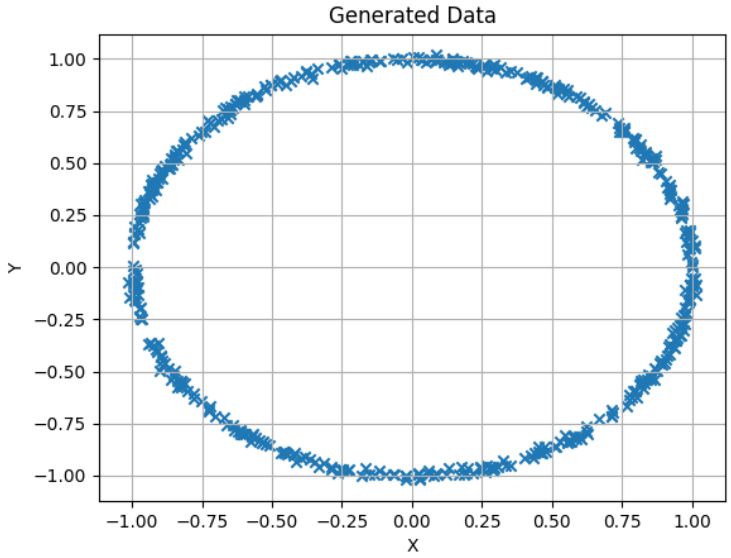


**Figure 2:** Generated data of 500 data points

Before going on, it is important to note something. Due to the numerical instabilities of the following methods we are considering, I have found it to be more numerically stable to calculate the witness functions in another way instead of $f = XH\alpha$. Instead, we use the formula, $f = \tilde{K}\alpha$ which has a short proof in the appendix A.

It is also important to note that numerically, we are going to be solving some given eigenvalue problems of the form $\mathbf{Av} = \lambda\mathbf{Bv}$. After much testing, I have found that most Python libraries seem to struggle to solve these problems due to $\mathbf{B}$ not always being positive definite. To solve this, in the exact implementations of COCO and CCA I have added a small regularisation to ensure positive definiteness in the form $\mathbf{B} \leftarrow \mathbf{B} + \varepsilon\mathbf{I}$, for all the tests we set $\varepsilon = 1 \times 10^{-10}$. This doesn't apply to the incomplete Cholesky implementation.

Further more because there can be some ambiguity in what people mean by a Gaussian kernel in the literature, for clarity we will be using this kernel function:

$$k(x, x') := \exp\left(-\gamma^{-2}\|x - x'\|^2\right)$$

## 2.1 Incomplete Cholesky for Efficient COCO

### 2.1.1 Deriving Computationally Efficient COCO

Incomplete Cholesky decomposition provides a low-rank approximation of symmetric positive semi-definite matrices by computing a partial triangular factorization based on column residual norms [Joh]

Our task is to use this method to compare the implementation of COCO computed exactly against

an approximation via incomplete Cholesky. We will compare number of operations and runtimes. First, we will mathematically formulate how to apply incomplete Cholesky and will later apply this in a Python3 program. From the extract on incomplete Cholesky provided [Joh] we know the following decomposition of the kernel matrix:

$$K = XX' = R'Q'QR = R'R$$

From this and using the fact that $H = H'$ we can make the jump to say that:

$$\tilde{K} = HR'RH = (RH)'(RH)$$

Going further with this idea, we can rewrite our original eigenvalue problem for COCO:

$$\begin{bmatrix} 0 & \frac{1}{n}\tilde{K}\tilde{L} \\ \frac{1}{n}\tilde{L}\tilde{K} & 0 \end{bmatrix}\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \lambda \begin{bmatrix} \tilde{K} & 0 \\ 0 & \tilde{L} \end{bmatrix}\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

Using the following formulas $\tilde{R_x} = R_x H$, $\tilde{R_y} = R_y H$, $\tilde{K} = \tilde{R_x}^T \tilde{R_x}$, and $\tilde{L} = \tilde{R_y}^T \tilde{R_y}$, we can now write the eigenvalue problem as:

$$\begin{bmatrix} 0 & \frac{1}{n}\tilde{R_x}^T \tilde{R_x} \tilde{R_y}^T \tilde{R_y} \\ \frac{1}{n}\tilde{R_y}^T \tilde{R_y} \tilde{R_x}^T \tilde{R_x} & 0 \end{bmatrix}\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \lambda \begin{bmatrix} \tilde{R_x}^T \tilde{R_x} & 0 \\ 0 & \tilde{R_y}^T \tilde{R_y} \end{bmatrix}\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

Now using a result from [Gre+05] we know that we can rewrite this in the form:

$$\begin{bmatrix} 0 & \frac{1}{n}\tilde{R}_x \tilde{R}_y^T \\ \frac{1}{n}\tilde{R}_y \tilde{R}_x^T & 0 \end{bmatrix}\begin{bmatrix} \tilde{R}_x^T \alpha \\ \tilde{R}_y^T \beta \end{bmatrix} = \lambda \begin{bmatrix} \tilde{R}_x^T \alpha \\ \tilde{R}_y^T \beta \end{bmatrix}$$

This greatly simplifies our problem and reduces the number of operations. Since we now are solving an eigenvalue problem for a much smaller matrix. For solving COCO optimization problems, the standard approach requires $O(n^3)$ computational complexity due to direct matrix operations, which can be prohibitively expensive for large-scale applications. Cholesky decomposition reduces this to $O(tn^2)$ where $t < n$, providing significant computational savings and maintaining accuracy.

### 2.1.2 Implementation

In order to compare both the exact and incomplete Cholesky implementations, I have produced some graphs using the data generated from the following equations.

$$\begin{aligned} x &= \sin(t) + n_1 \\ y &= \cos(t) + n_2 \\ n_1, n_2 &\sim \mathcal{N}(0, 0.01^2) \\ t &\sim \mathcal{U}([0, 2\pi]) \end{aligned}$$

I have chosen to only use 500 samples when creating our dataset for the following graphs. As we will see later in the assignment, the orientation of $f$ and $g$ can change from dataset to dataset and run to run, for both the exact COCO and Cholesky COCO implementations. This impacts the plots that show both projections $f$ and $g$.

By doing both implementations, though, it has been much simpler to tell how good the incomplete Cholesky method is. For a given bandwidth $\gamma = 1$ on 500 samples, we get $\textbf{COCO}_{\text{EXACT}} = 0.090304672$ and $\textbf{COCO}_{\text{CHOLESKY}} = 0.090304672$. We also saw that our correlation for $f$ and $g$, was much higher than our correlation for $x$ and $y$. Where our original correlation was $-0.049$ and our new correlation for both the exact and incomplete Cholesky methods was around $0.943$ and $-0.943$, respectively. The sign change coming from these random flips in orientation.

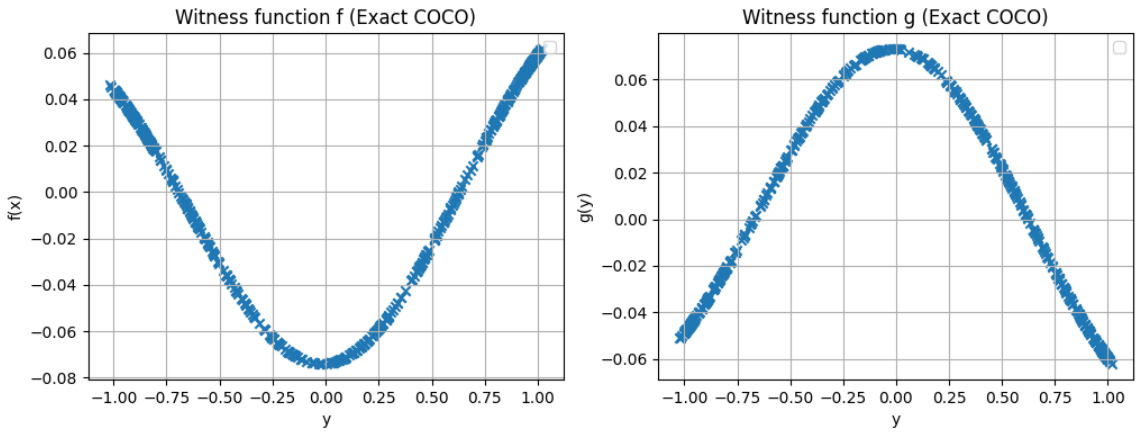


**Figure 3:** Exact COCO witness functions

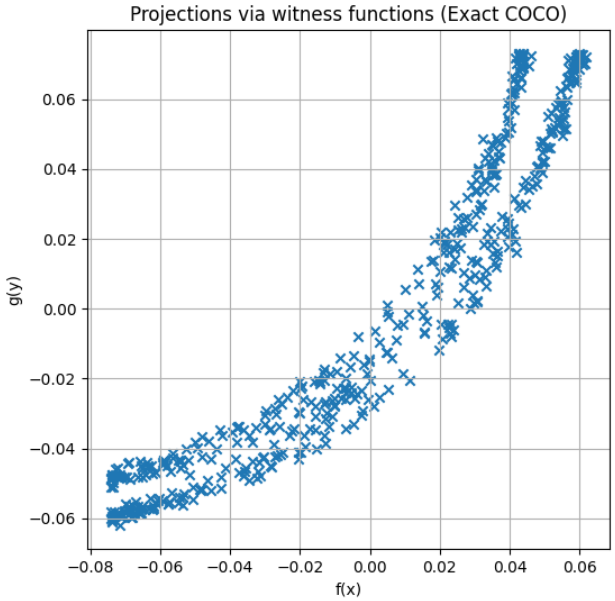


**Figure 4:** Exact COCO witness function projection

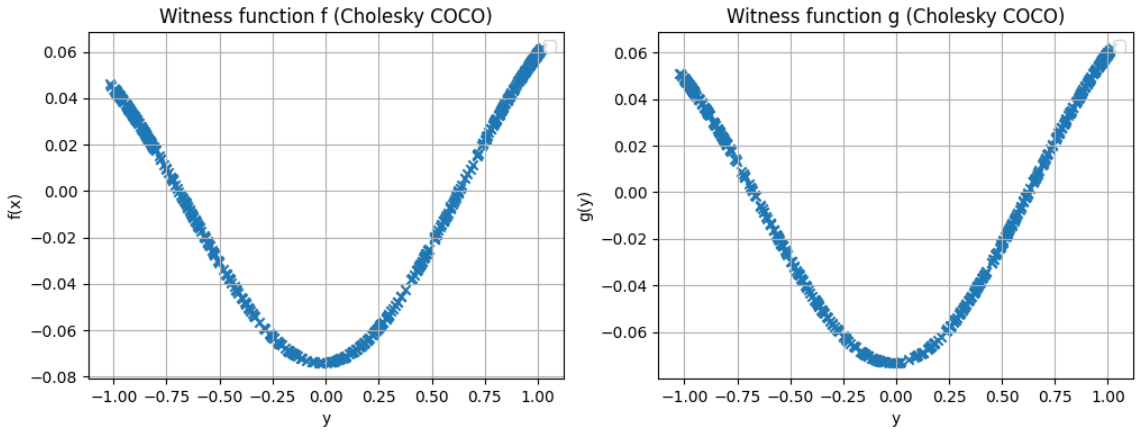


**Figure 5:** Incomplete Cholesky COCO witness functions

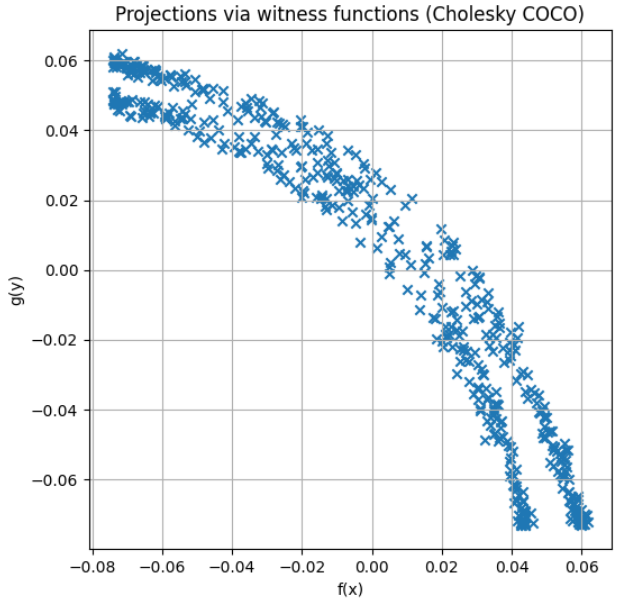


**Figure 6:** Incomplete Cholesky COCO witness function projection

## 2.2 Kernel CCA

### 2.2.1 Deriving The Eigenvalue Problem

From [Art] we know that CCA is defined as the solution to the following optimisation problem:

$$\max_{f,g} \langle f, \hat{C}_{XY} g \rangle_{\mathcal{F}}$$

Subject to:

$$\text{var}(f(x)) = \langle f, \hat{C}_{XX} f \rangle_{\mathcal{F}} = 1, \quad \text{var}(g(y)) = \langle g, \hat{C}_{YY} g \rangle_{\mathcal{G}} = 1$$

Using the method of Lagrangian multipliers, we get the following minimisation problem:

$$L(f,g,\lambda,\gamma) = -\langle f, \hat{C}_{XY} g\rangle_{\mathcal{F}} + \frac{\lambda}{2}(\langle f, \hat{C}_{XX} f\rangle_{\mathcal{F}} - 1) + \frac{\gamma}{2}(\langle g, \hat{C}_{YY} g\rangle_{\mathcal{G}} - 1)$$

We know that covariances are of the following form:

$$\hat{C}_{XY} = \frac{1}{n} XHY^T, \hat{C}_{XX} = \frac{1}{n} XHX^T, \hat{C}_{YY} = \frac{1}{n} YHY^T$$

Therefore, we can write $f$ and $g$ in the form:

$$\langle f, \hat{C}_{XX} f\rangle_{\mathcal{F}} = \frac{1}{n}\alpha^T HX^T XHX^T XH\alpha = \alpha^T \tilde{K}^2 \alpha = 1$$

$$\langle g, \hat{C}_{YY} g\rangle_{\mathcal{G}} = \frac{1}{n}\beta^T HY^T YHY^T YH\beta = \beta^T \tilde{L}^2 \beta = 1$$

Therefore, we can rewrite the Lagrangian as:

$$L(\alpha,\beta,\lambda,\gamma) = -\frac{1}{n}\alpha^T \tilde{K}\tilde{L}\beta + \frac{\lambda}{2}(\alpha^T \tilde{K}^2\alpha - 1) + \frac{\gamma}{2}(\beta^T \tilde{L}^2\beta - 1)$$

Taking the derivative of this with respect to $\alpha$ and separately to $\beta$, we get that:

$$\frac{\partial L}{\partial \alpha} = -\frac{1}{n}\tilde{K}\tilde{L}\beta + \frac{\lambda}{n}\tilde{K}^2\alpha = 0 \implies \tilde{K}\tilde{L}\beta = \lambda\tilde{K}^2\alpha$$

$$\frac{\partial L}{\partial \beta} = -\frac{1}{n}\tilde{L}\tilde{K}\alpha + \frac{\gamma}{n}\tilde{L}^2\beta = 0 \implies \tilde{L}\tilde{K}\alpha = \gamma\tilde{L}^2\beta$$

Using this, we know that, by multiplying these equations by $\alpha^T$ and $\beta^T$ respectively, we get:

$$\alpha^T \tilde{K}\tilde{L}\beta = \lambda\alpha^T \tilde{K}^2\alpha = \lambda$$

$$\beta^T \tilde{L}\tilde{K}\alpha = \gamma\beta^T \tilde{L}^2\beta = \gamma$$

Furthermore, since $\alpha^T \tilde{K}\tilde{L}\beta = \beta^T \tilde{L}\tilde{K}\alpha$ we can then say that $\lambda = \gamma$. This means that our problem can then be rewritten as:

$$\begin{bmatrix} 0 & \frac{1}{n}\tilde{K}\tilde{L} \\ \frac{1}{n}\tilde{L}\tilde{K} & 0 \end{bmatrix}\begin{bmatrix}\alpha \\ \beta\end{bmatrix} = \lambda\begin{bmatrix}\tilde{K}^2 & 0 \\ 0 & \tilde{L}^2\end{bmatrix}\begin{bmatrix}\alpha \\ \beta\end{bmatrix}$$

So, "What went wrong?" [Art]. In this kernelised version of the CCA optimisation problem, we are essentially trying to find the cosine between two subspaces generated by kernel matrices [Fra02]. These spaces become identical when using invertible kernels, and makes the method produce $f$ and $g$ where our correlation will be 1. This is a problem which we can solve by adding some regularisation though, leading to the original eigenvalue problem seen in the problem sheet [Art]:

$$\begin{bmatrix} 0 & \frac{1}{n}\tilde{K}\tilde{L} \\ \frac{1}{n}\tilde{L}\tilde{K} & 0 \end{bmatrix}\begin{bmatrix}\alpha \\ \beta\end{bmatrix} = \lambda\begin{bmatrix}\tilde{K}^2 + \kappa\tilde{K} & 0 \\ 0 & \tilde{L}^2 + \kappa\tilde{L}\end{bmatrix}\begin{bmatrix}\alpha \\ \beta\end{bmatrix}$$

### 2.2.2 Implementation

For a given bandwidth $\gamma = 2$ on 500 samples, we get $\mathbf{CCA}_{\text{EXACT}} = 0.001996764$. We also saw that our correlation for $f$ and $g$, was much higher than our correlation for $x$ and $y$. Where our original correlation was $-0.049$ and our new correlation was around $0.998$

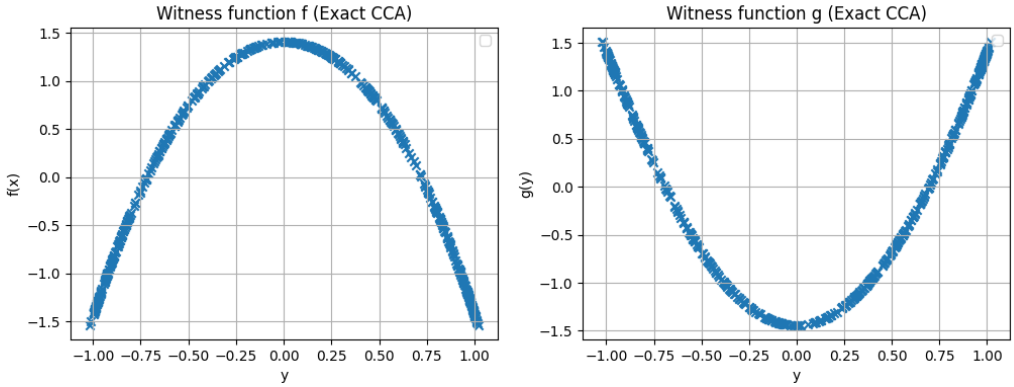


**Figure 7:** Exact CCA witness functions

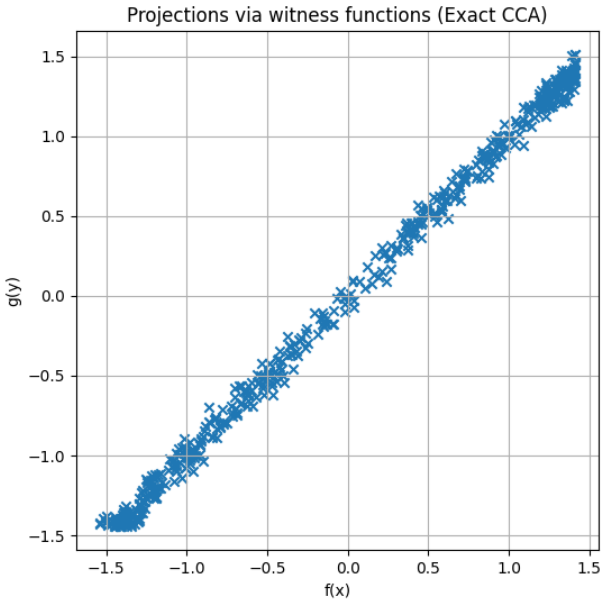


**Figure 8:** Exact CCA witness function projection

Below, we have also made plots to try to estimate how our CCA implementation has been effected by this regularisation term $\kappa$. We notice that as the regularisation term decreases, we seem to be asymptotically approaching a fixed value for both our CCA value and new correlation. This signals to us that perhaps this regularisation term really isn't making such a big difference, which is great for us when thinking about using this method on other datasets. It's hard to say if the behaviour is a quirk of CCA or if it's only happened to do our dataset, being quite simple, would be worth further investigation.

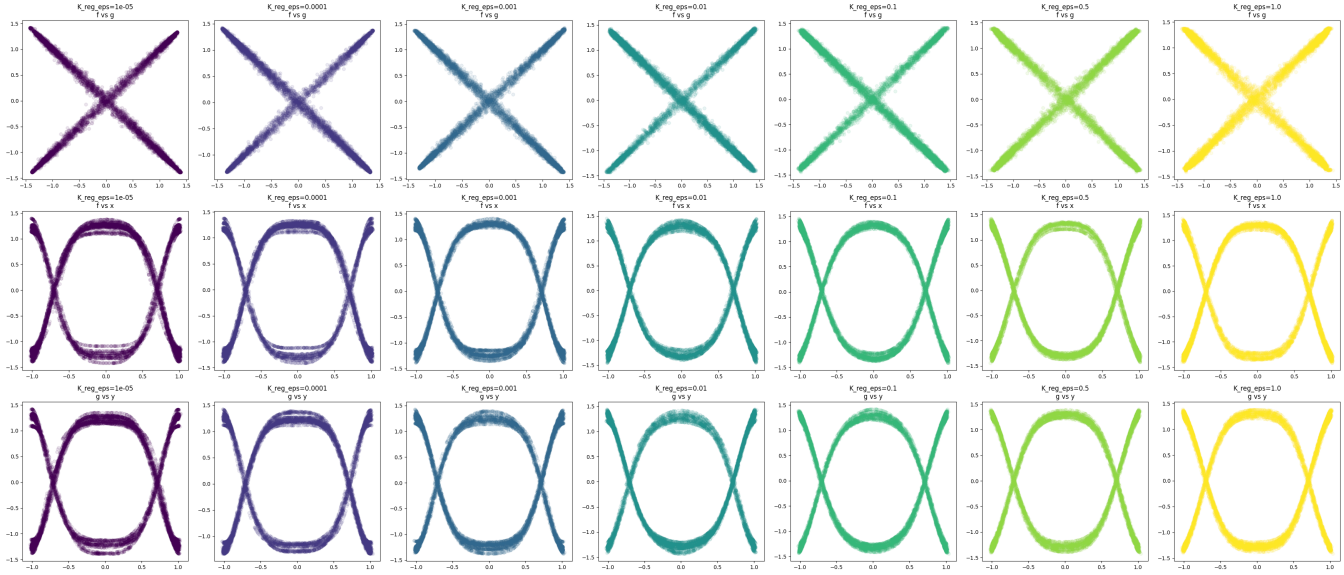

**Figure 9**

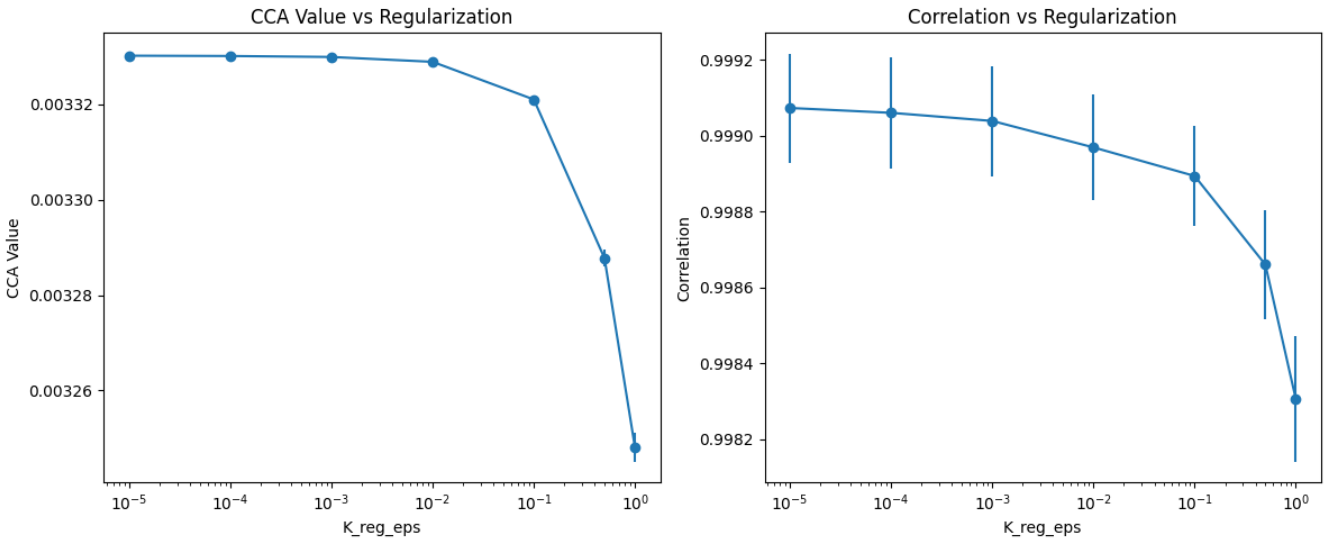


**Figure 10**

## 2.3 Comparing COCO to CCA

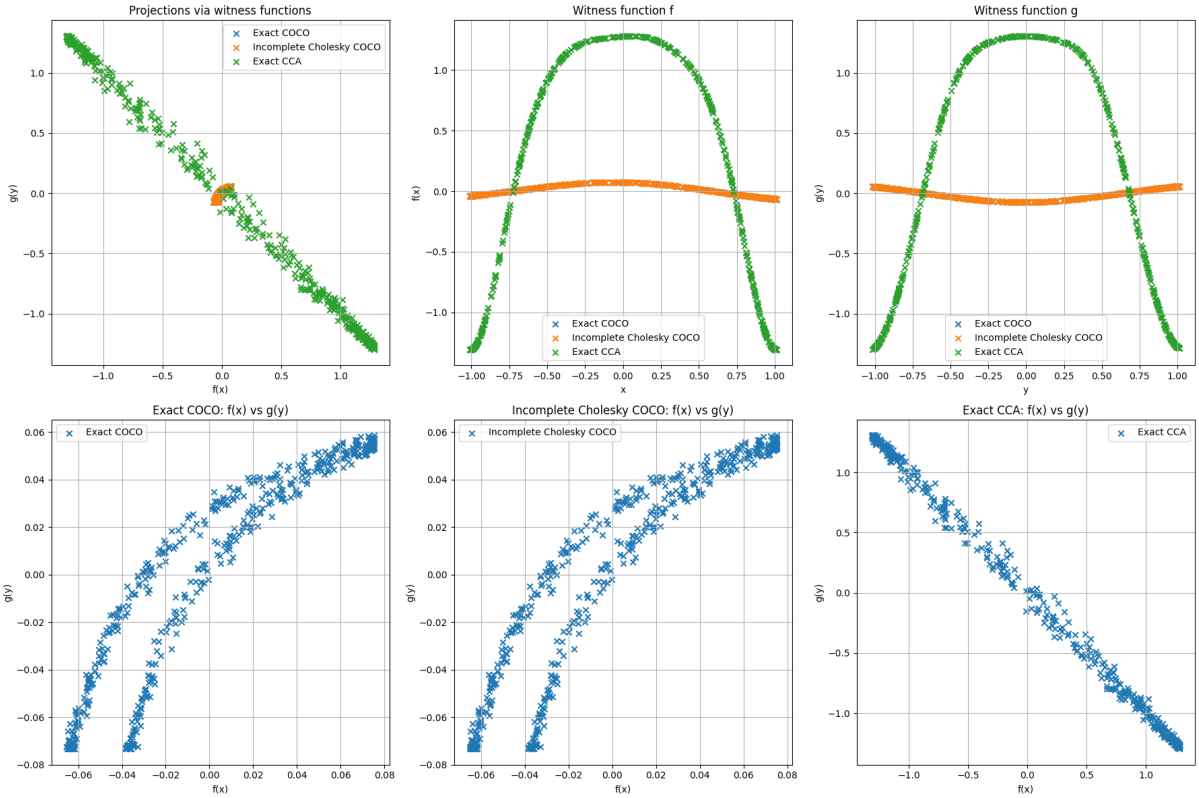


**Figure 11:** Witness function plots for exact COCO, incomplete Cholesky COCO, and exact CCA all on the same data, where the bandwidths are $\gamma_{\text{COCO}} = 1$ and $\gamma_{\text{CCA}} = 2$

Looking at these plots of the witness functions in 11, we instantly can tell that COCO based methods seem to have a much lower amplitude and scale in the witness functions against the input data plots and the projection plots ($f$ vs. $g$), respectively. More importantly, we also notice that the COCO based methods have projection plots with smaller absolute correlation values and with non-linear behaviour, against CCA where projections have only linear behaviour. What we can infer from this data is that perhaps CCA is better at capturing non-linear relationships in our original input space, compared to COCO based methods.
When comparing to the witness functions $f$ and $g$ of CCA to COCO, we can see that the original optimisation problems differ in what the constraints are, for CCA are:

$$\text{var}(f(x)) = \left\langle f, \hat{C}_{XX} f \right\rangle_{\mathcal{F}} = 1,$$
$$\text{var}(g(y)) = \left\langle g, \hat{C}_{YY} g \right\rangle_{\mathcal{G}} = 1,$$

COCO constraints are:

$$\|f\|_{\mathcal{F}} = 1,$$
$$\|g\|_{\mathcal{G}} = 1,$$

Delving deeper into the COCO conditions, one can say that $\|f\|_{\mathcal{F}} = \sqrt{n * (\mathbf{E}(f^2))} = 1$, where $n$ is the number of data points, in our code this is the formula we use to normalise $f$ and $g$. In comparison, we know that for CCA the condition is $\text{var}(f(x)) = 1$ the also implies that $\text{St.Dev}(f(x)) = \sqrt{\mathbf{E}(f^2) - \mathbf{E}(f)^2}$, where in our code we use the standard deviation to normalise $f$ and $g$, to satisfy the conditions. Thinking about this, we can see that, in comparison to CCA, COCO has a dependence on the number of data points $n$. When directly comparing these two methods, then it is very much expected that we are going to have a much smaller scale to our witness functions in the COCO setting than our CCA setting, which is shown in our figure above 11. There is also a minor contribution from the quadratic mean of $f$ under the square root in CCA which makes the standard deviation even smaller and scales up the witness functions in CCA, when thinking about the comparison to COCO.

Comparing runtimes on all models for the same dataset, we consistently see that the incomplete Cholesky COCO is much faster than the exact COCO and exact CCA implementations. For a singular run that was used to generate the data in 11, we saw the following timings:

- Exact COCO time: 0.754s
- Incomplete Cholesky COCO time: 0.010s
- Exact CCA time: 0.257s

To further show this phenomenon of flipping and to indicate that our results are reproducible, we have generated a plot below which combines the results from 20 random runs and uses 300 data points 12. This plot clearly illustrates how our results and insights, at least in the scope of this relatively simple example, apply across all runs. Furthermore, we also notice that the witness function plots for CCA tend to have flatter peaks than the COCO variants. Why don't we explore this further.

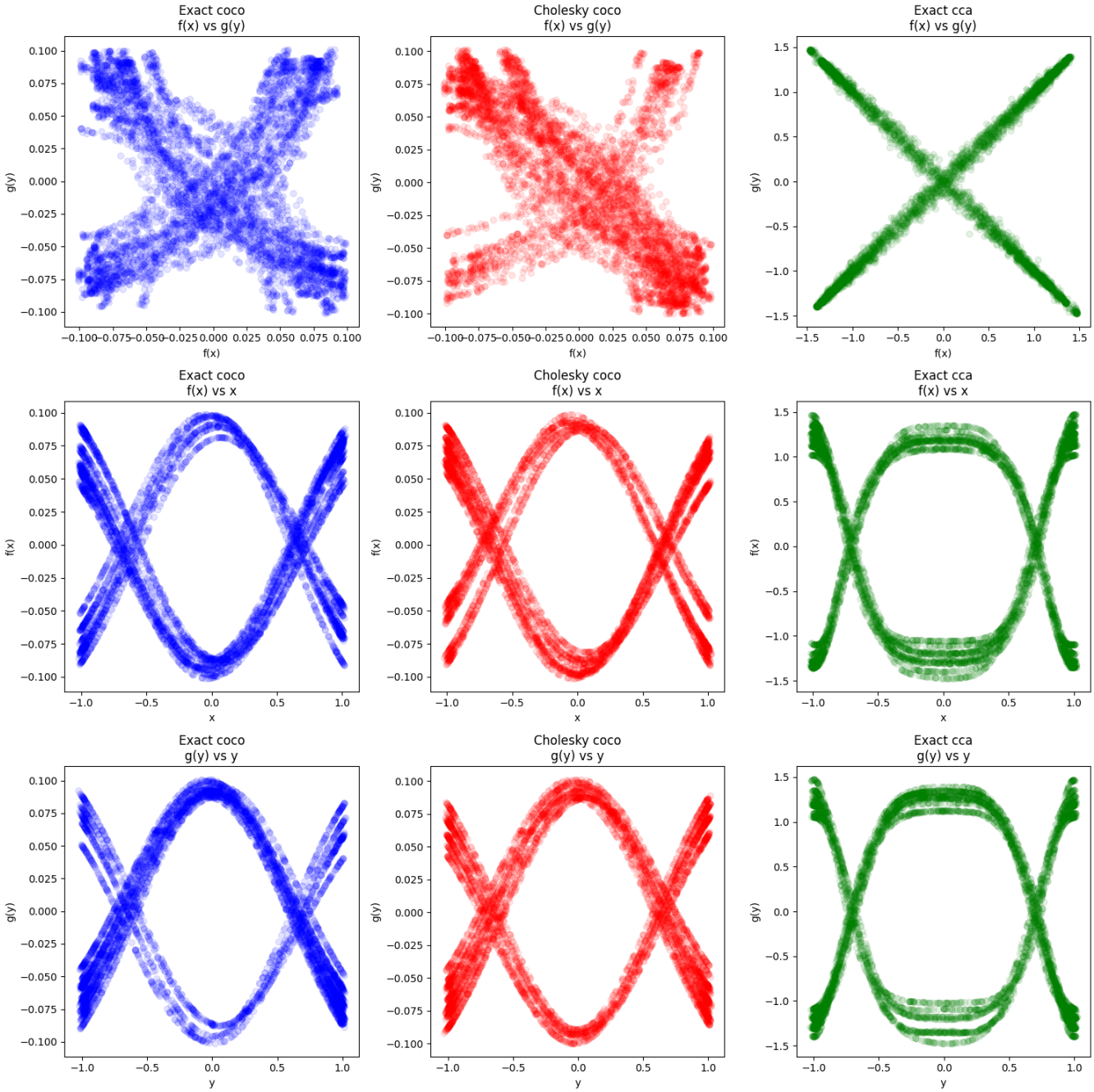


**Figure 12:** 20 random runs for all methods with 300 data points each run

We will now compare how changing the values of gamma makes a difference to COCO/CCA values for the incomplete Cholesky COCO and exact CCA implementations. Below you will find these graphs which similar to 11 shows different runs layered on top of each other but this time for 30 runs on each different $\gamma$ value. The main takeaway to take from these graphs is that CCA seems to be more robust than COCO when we aren't sure about what the appropriate bandwidth is, at least in this example. We also notice that as opposed to CCA which seems to asymptote as $\gamma$ decreases, but still also has a degeneration in the witness functions, COCO has a defined peak for the values itself and the correlation. This is interesting and may be worth further consideration.

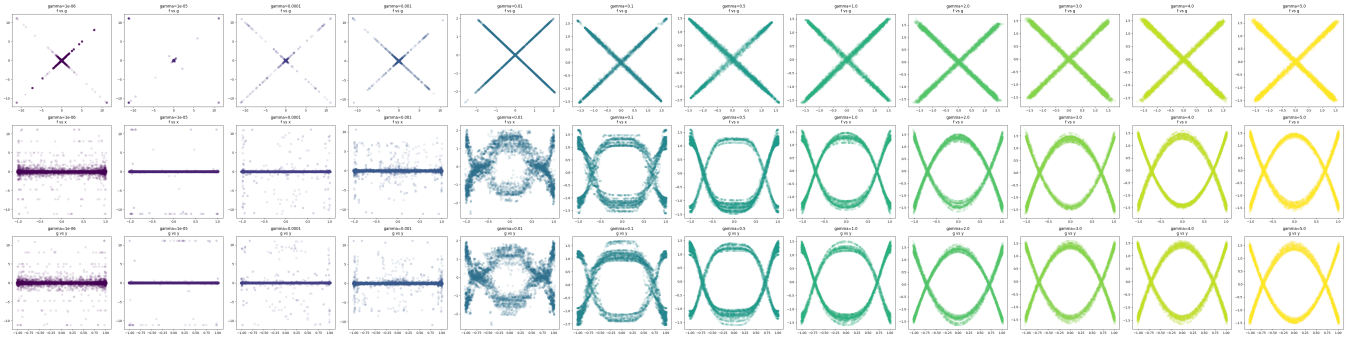

**Figure 13**

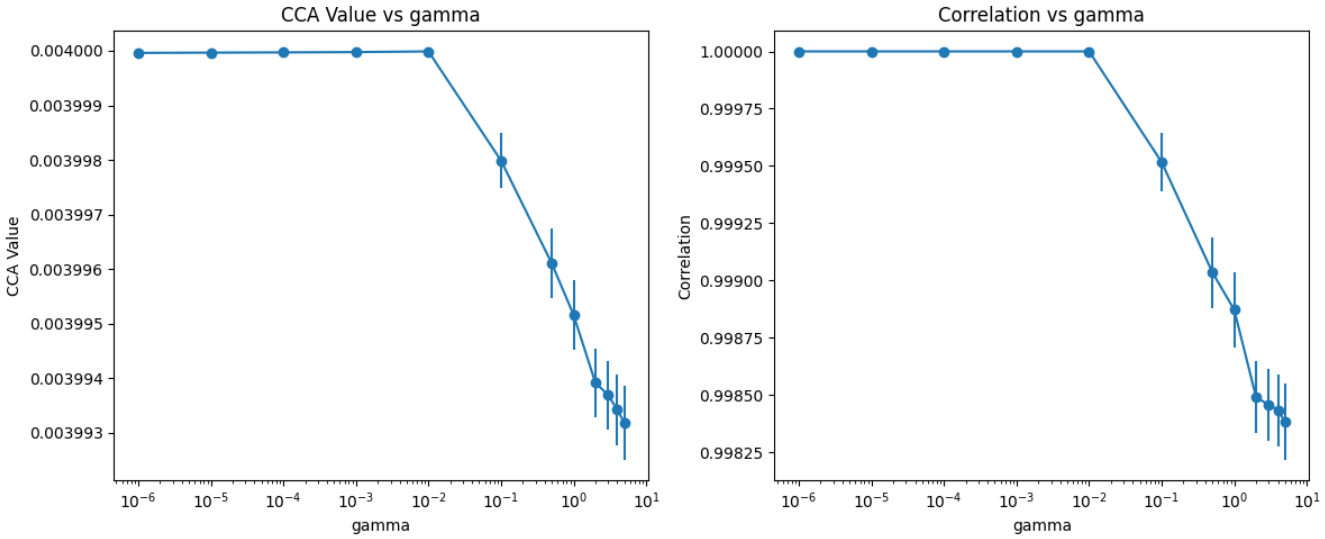


**Figure 14**

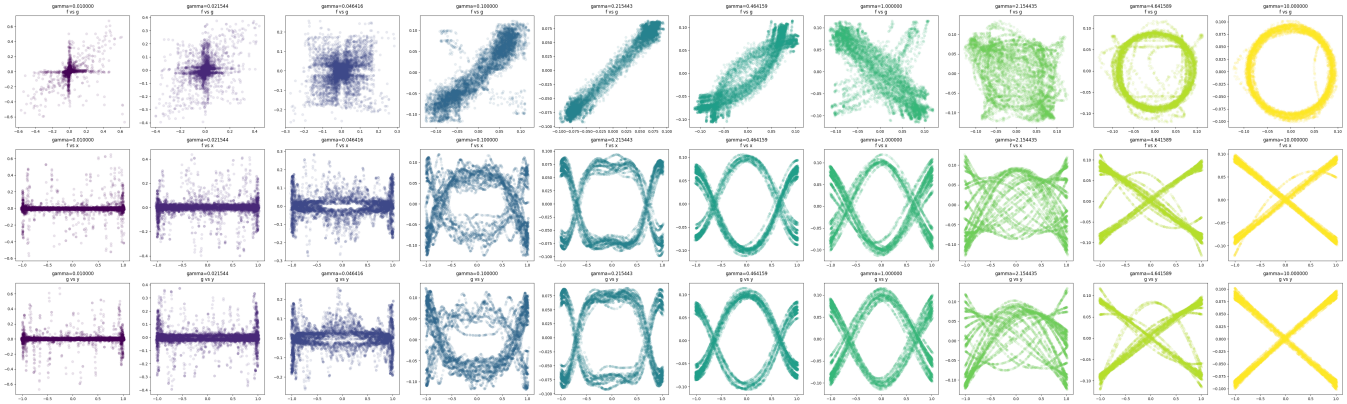

**Figure 15**

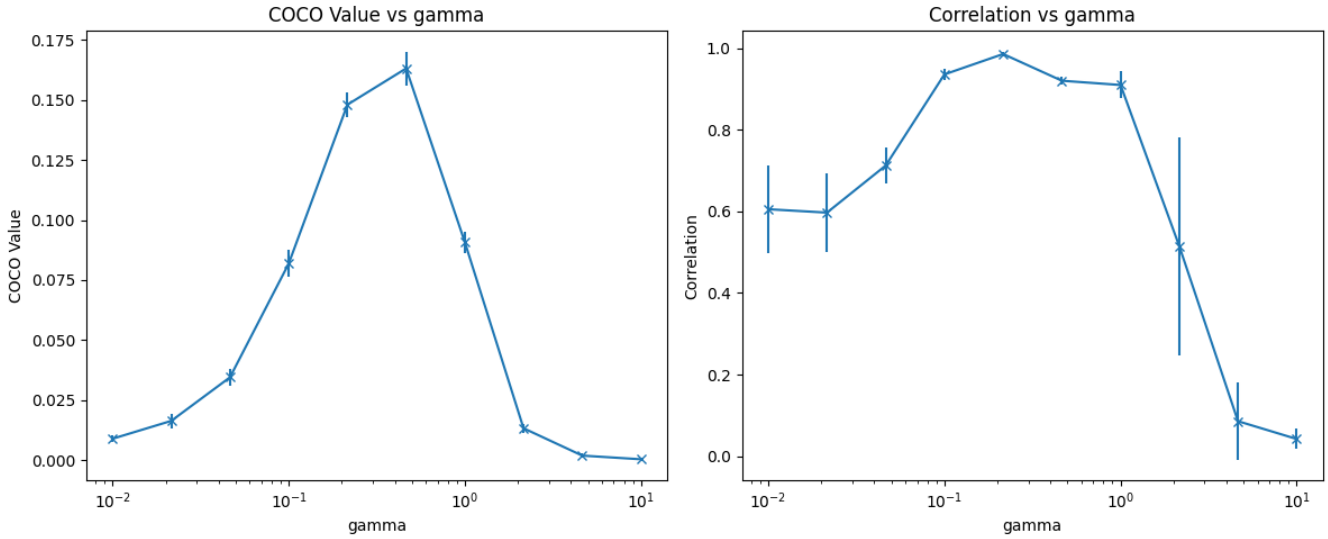


**Figure 16**

# References

[Fra02] Michael I. Jordan Francis R. Bach. **Kernel Independent Component Analysis**. https://www.jmlr.org/papers/volume3/bach02a/bach02a.pdf. (Accessed on 11/3/2024). Feb. 2002.

[Gre+05] Arthur Gretton et al. "Kernel Methods for Measuring Independence". In: **Journal of Machine Learning Research** 6.70 (2005). (Accessed on 11/3/2024), pp. 2075–2129. URL: http://jmlr.org/papers/v6/gretton05a.html.

[Art] Hugh Dance Arthur Gretton. **Kernel assignment: advanced topics in machine learning**. https://www.gatsby.ucl.ac.uk/~gretton/coursefiles/questions_24_CCA.pdf. (Accessed on 11/1/2024).

[Joh] Nello Cristianini John Shawe-Taylor. **Kernel Methods for Pattern Analysis**. https://www.gatsby.ucl.ac.uk/~gretton/coursefiles/incompleteCholesky.pdf. (Accessed on 11/1/2024).

# A Proof $f = \tilde{\mathbf{K}}\alpha$

We begin by considering an RKHS $\mathcal{F}$ with reproducing kernel, $k(x,y) = \langle \phi(x), \phi(y) \rangle_{\mathcal{F}}$ where $\phi : \mathcal{X} \to \mathcal{F}$ is the feature map.

By the Representer Theorem, we know that any solution $f$ can be written as a linear combination of centred feature mappings:

$$f = \sum_{i=1}^{n} \alpha_i [\phi(x_i) - \hat{\mu}_x]$$

where $\hat{\mu}_x = \frac{1}{n}\sum_{i=1}^{n} \phi(x_i)$ is the empirical mean in feature space. For any point $x \in \mathcal{X}$, the reproducing property gives us:

$$f(x) = \langle f, \phi(x)\rangle_{\mathcal{F}} = \left\langle \sum_{i=1}^{n} \alpha_i [\phi(x_i) - \hat{\mu}_x], \phi(x) \right\rangle_{\mathcal{F}}$$

Expanding this inner product and applying the reproducing property again:

$$\begin{aligned}
\left\langle \sum_{i=1}^{n} \alpha_i [\phi(x_i) - \hat{\mu}_x], \phi(x) \right\rangle_{\mathcal{F}} &= \sum_{i=1}^{n} \alpha_i \langle \phi(x_i), \phi(x)\rangle_{\mathcal{F}} - \sum_{i=1}^{n} \alpha_i \langle \hat{\mu}_x, \phi(x)\rangle_{\mathcal{F}} \\
&= \sum_{i=1}^{n} \alpha_i k(x_i, x) - \sum_{i=1}^{n} \alpha_i \left\langle \frac{1}{n} \sum_{j=1}^{n} \phi(x_j), \phi(x) \right\rangle_{\mathcal{F}} \\
&= \sum_{i=1}^{n} \alpha_i k(x_i, x) - \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i k(x_j, x)
\end{aligned}$$

Now consider the evaluation of $\tilde{K}\alpha$ at point $x$. First, we compute $H\alpha$:

$$H\alpha = \alpha - \frac{1}{n} \left( \sum_{i=1}^{n} \alpha_i \right) \mathbf{1}$$

Then for any index $i$, the $i$th component of $K(H\alpha)$ is:

$$(K(H\alpha))_i = \sum_{j=1}^{n} k(x_i, x_j) \left( \alpha_j - \frac{1}{n} \sum_{k=1}^{n} \alpha_k \right)$$

Finally, applying $H$ again yields:

$$\begin{aligned}
(HKH)_x \alpha &= \sum_{i=1}^{n} k(x, x_i) \left( \alpha_i - \frac{1}{n} \sum_{j=1}^{n} \alpha_j \right) - \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{n} k(x_j, x_i) \left( \alpha_i - \frac{1}{n} \sum_{k=1}^{n} \alpha_k \right) \\
&= \sum_{i=1}^{n} k(x, x_i)\alpha_i - \frac{1}{n} \sum_{i=1}^{n} k(x, x_i) \sum_{j=1}^{n} \alpha_j - \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{n} k(x_j, x_i)\alpha_i + \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} k(x_j, x_i) \sum_{k=1}^{n} \alpha_k \\
&= \sum_{i=1}^{n} k(x_i, x)\alpha_i - \frac{1}{n} \sum_{i=1}^{n} k(x_i, x) \sum_{j=1}^{n} \alpha_j - \frac{1}{n} \sum_{i=1}^{n} \alpha_i \sum_{j=1}^{n} k(x_j, x) + \frac{1}{n^2} \sum_{k=1}^{n} \alpha_k \sum_{i=1}^{n} \sum_{j=1}^{n} k(x_j, x_i) \\
&= \sum_{i=1}^{n} k(x_i, x)\alpha_i - \frac{1}{n} \sum_{i=1}^{n} k(x_i, x) \sum_{j=1}^{n} \alpha_j - \frac{1}{n} \sum_{i=1}^{n} \alpha_i \sum_{j=1}^{n} k(x_j, x) + \frac{1}{n} \sum_{j=1}^{n} \alpha_j \sum_{i=1}^{n} k(x_i, x) \\
&= \sum_{i=1}^{n} \alpha_i k(x_i, x) - \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i k(x_j, x)
\end{aligned}$$

Thus we have shown that $f(x) = (\tilde{K}\alpha)_x$ for all $x \in \mathcal{X}$. Since both $f$ and $\tilde{K}\alpha$ are elements of the RKHS $\mathcal{F}$, and they agree at all points, the uniqueness theorem for RKHS functions implies that $f = \tilde{K}\alpha$ as elements of $\mathcal{F}$, completing the proof.