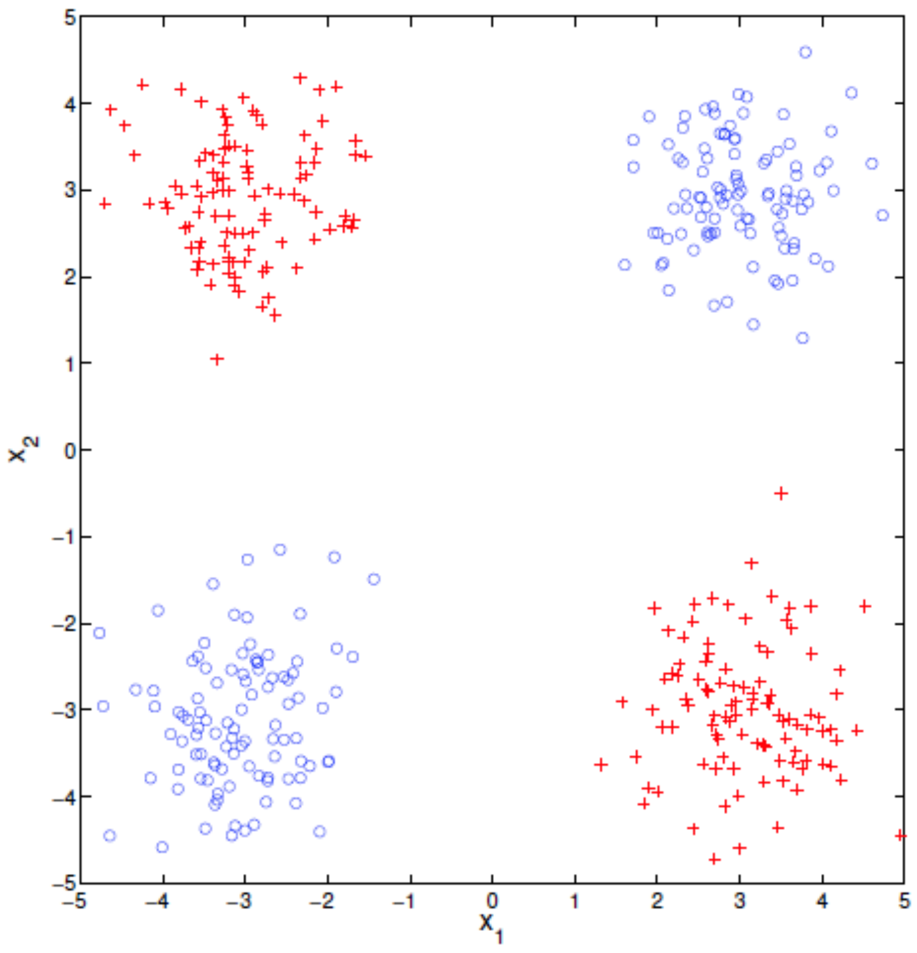
Linear kernel learning

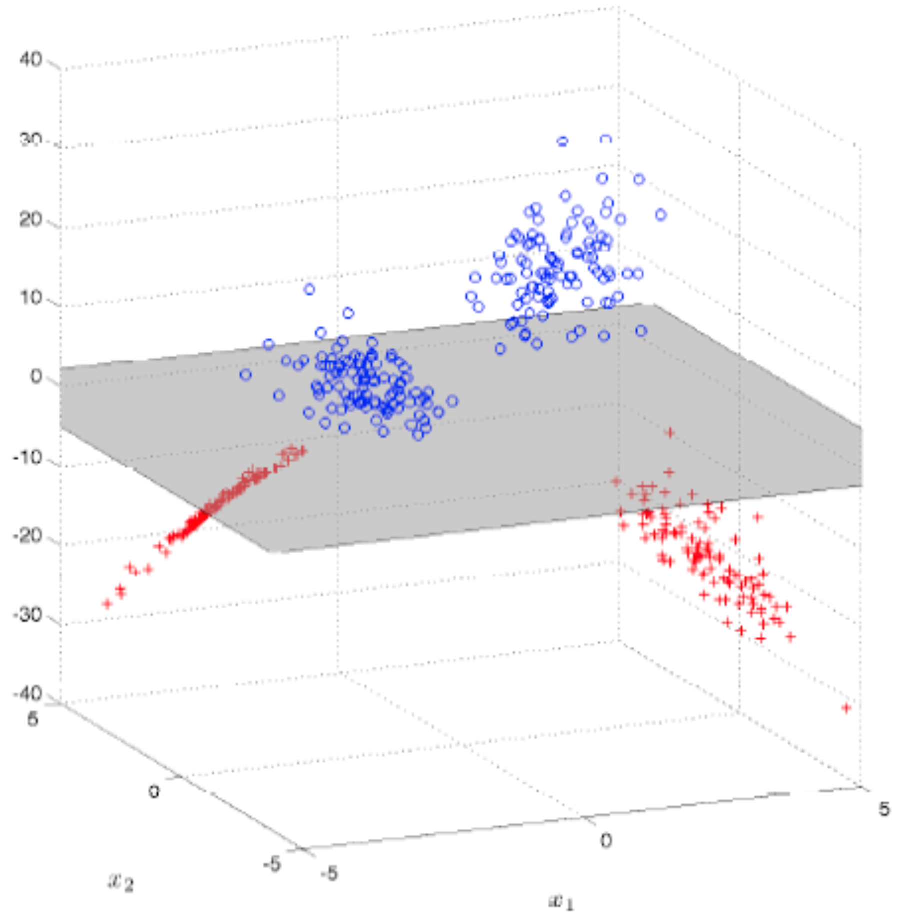
Prof. Samir Bhatt

University Of Copenhagen

Imperial College London

The feature map





No linear classifier

$$(x_1, x_2) \to (x_1, x_2, x_1, x_2)$$

But we can linearise!

How to create high dimensional spaces?

Kernel methods allow construction of non linear methods after mapping to a higher dimensional feature space $\phi : X \rightarrow \mathbb{R}^D$

Typically rely on inner products $\phi(x_i)^T \phi(x_i)$

Its implicit - explicit features need not be computed

$k(x_i, x_j) = \phi(x_i)^T \phi(x_j) = (1 + x_i^T x_j)^q$ with dimension $x \in \mathbb{R}^p$

Explicit construction would require $\binom{p+q}{q}$

A 2d quadratic example $p = 2, q = 2$ is

$$\phi(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2, x_1^2, x_2^2) \in \mathbb{R}^6$$

Kernel as an inner product between feature maps

Let $\mathscr{X}$ be a non empty set. A function $k : \mathscr{X} \times \mathscr{X} \to \mathbb{R}$ is a kernel if there exists a Hilbert space and a map $\phi : \mathscr{X} \to H$ such that for all $x, x' \in \mathscr{X}$,

$$k(x, x') := \langle \phi(x), \phi(x') \rangle_H$$

This inner product can be thought of as a similarity between the various inputs

In a vector space the standard inner product is given by $\langle x, z \rangle = \sum_{i=1}^{n} x_i z_i$

Some more intuition

If we are given a “measure of similarity” with two arguments, x, x', how can we determine if it is a valid kernel? What is the feature map? What if we want it to be infinite dimensional?

$$\Phi : \mathbf{x}_i \in \mathbb{R}^d \mapsto \Phi(\mathbf{x}_i) \in \mathbb{R}^D$$

$$\Phi : X \in \mathbb{R}^{N\times d} \mapsto \Phi(X) \in \mathbb{R}^{N\times D}$$

or

$$\Phi : \mathbf{x}_i \in \mathbb{R}^d \mapsto \Phi(\mathbf{x}_i) \in \mathbb{R}^\infty$$

$$\Phi : X \in \mathbb{R}^{N\times d} \mapsto \Phi(X) \in \mathbb{R}^{N\times\infty}$$

We can use a direct property - positive semidefiniteness (PSD)

Example of kernels

Kernel	**Kernel Function,** $k(\mathbf{x}_i, \mathbf{x}_j)$	**Power Spectral Density,** $\mathbb{P}(\boldsymbol{\omega})$
Squared Exponential†◇	$\sigma^2 \exp\left(-\frac{(\mathbf{x}_i - \mathbf{x}_j)^2}{2\ell^2}\right)$	$(2\pi) - \frac{D}{2}\sigma^D \exp\left(\frac{\sigma^D \Vert\boldsymbol{\omega}\Vert_2^2}{2}\right)$
*Matén**†‡	$\frac{2^{1-\lambda}}{\Gamma(\lambda)}\left(\frac{\sqrt{2\lambda}\Vert\mathbf{x}_i - \mathbf{x}_j\Vert_2}{\sigma}\right)^{\lambda} k_\lambda\left(\frac{\sqrt{2\lambda}\Vert\mathbf{x}_i - \mathbf{x}_j\Vert_2}{\sigma}\right)$	$\frac{2^{D+\lambda}\pi^{\frac{D}{2}}\Gamma(\lambda+\frac{D}{2})\lambda^\lambda}{\Gamma(\lambda)\sigma^{2\lambda}}\left(\frac{2\lambda}{\sigma^2} + 4\pi^2\Vert\boldsymbol{\omega}\Vert_2^2\right)^{-(\lambda+\frac{D}{2})}$
Cauchy	$\prod_{i=1}^{D} \frac{2}{1+(\mathbf{x}_i - \mathbf{x}_j)^2}$	$\exp(-\Vert\boldsymbol{\omega}\Vert_1)$
Laplacian†	$\exp\left(-\sigma\Vert\mathbf{x}_i - \mathbf{x}_j\Vert_1\right)$	$\left(\frac{2}{\pi}\right)^{\frac{D}{2}} \prod_{i=1}^{D} \frac{\sigma}{\sigma^2+\omega_i^2}$

*$\Gamma(\cdot)$ *is the gamma function and* $k_\lambda(\cdot)$ *is the modified Bessel function of the second kind.* † *Parameter* $\sigma > 0$. ‡ *Parameter* $\lambda > 0$. ◇ *Parameter* $\ell > 0$.

Plot a kernel function

```
n=100
x=seq(0,1,0.001) # distances

k1=function(x,alpha) exp(-x/alpha)
k2=function(x,alpha) exp(-x^2/alpha)

alpha=0.2
par(mfrow=c(1,2))
plot(x,k1(x,alpha),type='l')
plot(x,k2(x,alpha),type='l')
```

A PSD function

A given symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is PSD is for all $n \geq 1$ and a set of coefficients $a_1, \ldots, a_n \in \mathbb{R}^n$ and $x_1, \ldots, x_n \in \mathcal{X}^n$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j k(x_i, x_j) \geq 0 = \sum_{i=1}^{n} \sum_{j=1}^{n} \langle a_i \phi(x_i), a_j \phi(x_j) \rangle_H$$

Here the distance intuition is clear

Example of polynomial in a finite space

Our model is a linear function of x_1, x_2, x_1x_2 i.e
$f(x) = f_1x_1 + f_2x_2 + f_3x_1x_2$

Therefore our mapping is $\mathscr{X} : \mathbb{R}^2 \to \mathbb{R}$, and we say $f(\cdot) = [f_1, f_2, f_3]$

$f(\cdot)$ is a function object (vector in $\mathbb{R}^3$) and $f(x)$ is an evaluation (real number)

So $f(x) = f(\cdot)^T\phi(x) = \langle f(\cdot), \phi(x)\rangle_H$ which means evaluations of f at x are an inner product space. And $H : \mathbb{R}^2 \to \mathbb{R}$

Reproducing kernel Hilbert Spaces

H is a Hilbert space of $\mathbb{R}$ valued functions on a non empty $\mathscr{X}$. A function $k : \mathscr{X} \times \mathscr{X} \to \mathbb{R}$ is a reproducing kernel of H and H is a RKHS if

1) For all $x \in \mathscr{X}, k(\cdot, x) \in H$

2) For all $x \in \mathscr{X}$ and for all $f \in H$, $\langle f(\cdot), k(\cdot, x) \rangle_H = f(x)$

Why does this matter? Members of say L_2 spaces can't be evaluated because you can change the value at one point without changing the value of the integral (Lebesgue).

An RKHS is associated with a kernel that reproduces every function in the space in the sense that for any x in the set on which the functions are defined, and evaluation at x can be performed by taking an inner product with a function determined by the kernel.

Let simulate some data from this

We can say that we have sample from a multivariate normal with zero mean and covariance = our kernel matrix

Then $z = N(0,1)A$ where $AA^T = K$ via the Cholesky decomposition

Note this decomposition as it will be relevant later when we look at Fourier features

Let simulate some data from this

```
n=200
x=seq(0,1,length.out=n)
alpha=0.1
k1=function(x,alpha) exp(-x/alpha)
k2=function(x,alpha) exp(-x^2/alpha)
K1=k1(as.matrix(dist(x)),alpha)
K2=k2(as.matrix(dist(x)),alpha)

par(mfrow=c(2,2))
image(K1)
image(K2)

K1_ = chol(K1 + diag(1e-8,n))
K2_ = chol(K2 + diag(1e-8,n))

gamma = cbind(rnorm(n))

plot(x,K1%*%gamma,type='l',col='red',lwd=5)
plot(x,K2%*%gamma,type='l',col='blue',lwd=5)
```

Can we learn? The representer theorem

Standard supervised learning set up
$(x_1, y_1), \ldots, (x_n, y_n), x \in \mathbb{R}^d, y \in \mathbb{R}$

We want to find a function f^* in our RKHS H that solves this minimisation of risk

$$min_{f \in H} \sum_{i=1}^{n} \mathscr{L}(y_i, f(x_i)) + \Omega(\|f\|_h^2)$$

Obvious example for regression is $\mathscr{L}(y, f(x)) = (y - f(x))^2$

The solution is $f^* = \sum_{i=1}^{n} \alpha_i k(\cdot\,, x_i)$

Least squares!

by setting $X = K(X, X)$ we can find f^* for a squared loss by least squares and identify coefficients γ

$$\mathscr{L}(y - \beta X^T) = \frac{1}{n}(\beta X^T - y)^2$$

$$\nabla \mathscr{L}(y - \beta X^T) = \frac{2}{n} X^T(\beta X^T - y) = 0$$

$$X^T X \beta = X^T y$$

$$\beta = (X^T X)^{-1} X^T y$$

Fitting on simulated data

```
n=200
x=seq(0,1,length.out=n)
alpha=0.1
k1=function(x,alpha) exp(-x/alpha)
K1=k1(as.matrix(dist(x)),alpha)
K1_ = chol(K1 + diag(1e-8,n))
gamma = cbind(rnorm(n))

z=K1%*%gamma
f=lm.fit(x=K1,y=z)

par(mfrow=c(2,1))
plot(z,f$fitted.values,pch=16)
abline(0,1,col='red')
plot(gamma,f$coefficients,pch=16)
abline(0,1,col='red')
```

Problems

- Not full rank (collinearity)

$$(X^T X)^{-1}$$ can be singular & non invertible

- How do we prevent overfitting?

Ridge regression

$$\frac{1}{2}(y - \gamma X^T)^2 + \frac{1}{2}C \parallel \gamma \parallel^2$$

$$\parallel \gamma \parallel^2 := \sqrt{\gamma_1^2 + \gamma_2^2 + \ldots + \gamma_d^2}$$

Squared loss function

Penalty

Favour small norms (preventing over complex functions)

Adding some bias (to reduce variance)

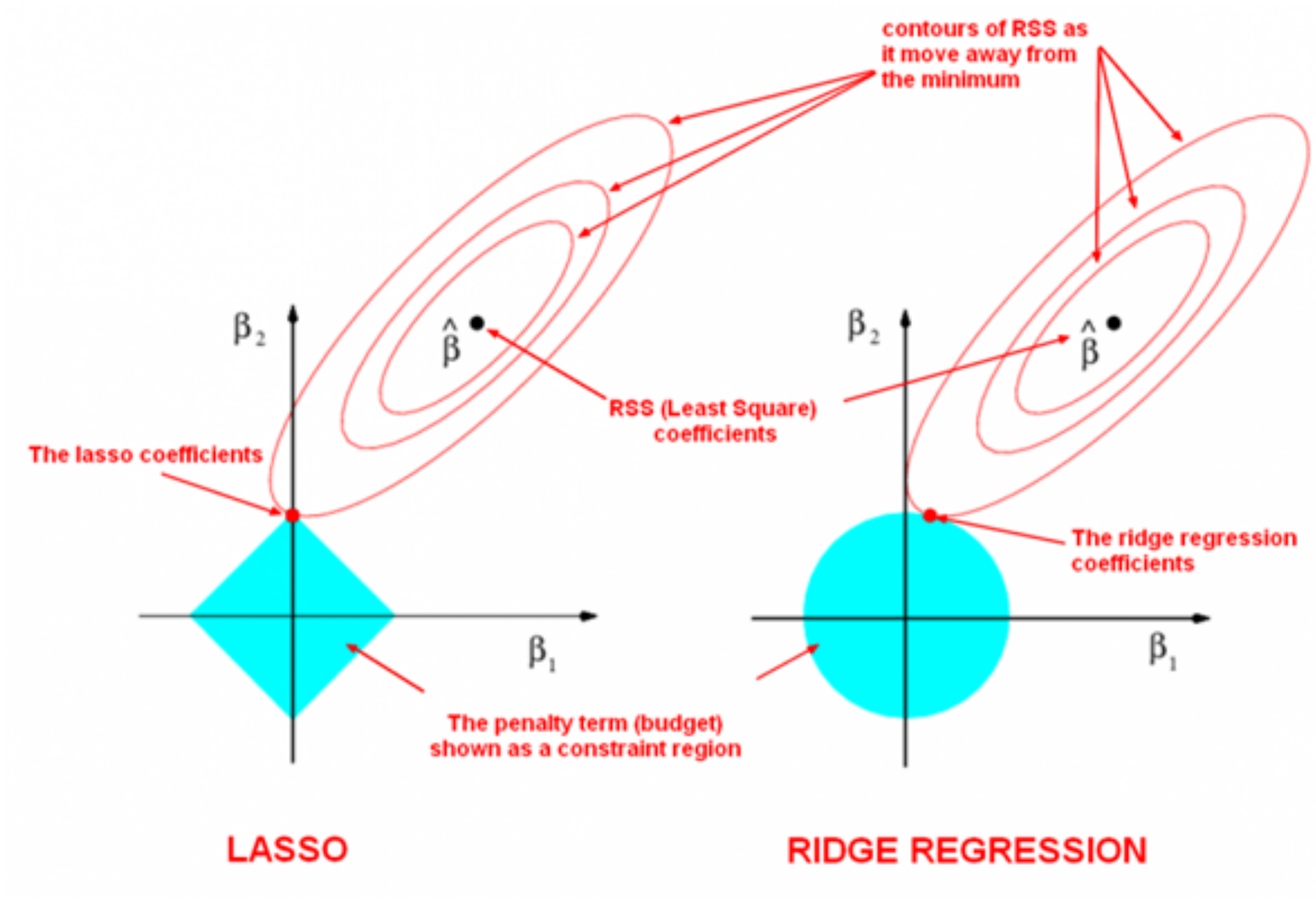
So least squares becomes

$$\nabla \mathscr{L}(y - \beta X^T) = X^T(\beta X^T - y) + \lambda\beta = 0$$

$$\beta = (X^T X + \lambda I_n)^{-1} X^T y$$

Basically the same but with this term added on the diagonal

What this is doing geometrically


contours of RSS as it move away from the minimum
β_2
$\hat{\beta}$
RSS (Least Square) coefficients
The lasso coefficients
The ridge regression coefficients
β_1
The penalty term (budget) shown as a constraint region
LASSO
RIDGE REGRESSION

Lets fit ridge regression

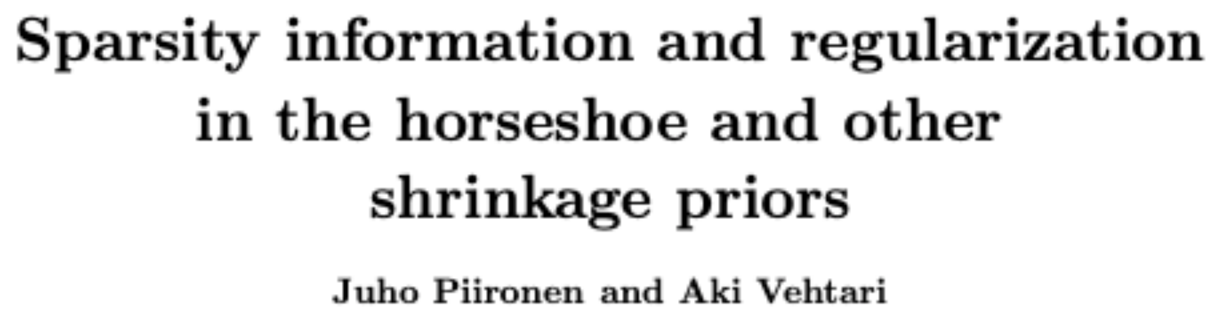
```r
n=200
x=seq(0,1,length.out=n)
alpha=0.1
k1=function(x,alpha) exp(-x/alpha)
K1=k1(as.matrix(dist(x)),alpha)
K1_ = chol(K1 + diag(1e-8,n))
gamma = cbind(rnorm(n))

z=K1%*%gamma
z_= z+ rnorm(n,0,1)

f=lm.fit(x=K1,y=z_)

plot(x,z_,pch=16)
lines(x,f$fitted.values,col='red')
lines(x,z,col='blue')

library(glmnet)
lambdas <- 10^seq(5, -5, length.out=100)
cv_fit <- cv.glmnet(K1, z_, alpha = 0, lambda = lambdas)
opt_lambda <- cv_fit$lambda.min
fit <- glmnet(K1, z_, alpha = 0, lambda = opt_lambda)

f2 <- predict(fit, s = opt_lambda, newx = K1)
lines(x,f2,col='green',lwd=2)
```

What about Bayesian models?

Ridge regression penalty = $N(0, C)$ prior

$$y \sim N(\mu, \sigma^2)$$
$$\mu = \gamma X^T$$
$$\gamma \sim N(0, C\mathbb{I})$$
$$C \sim \pi()$$
$$\lambda \sim \pi()$$

Or something better….


Sparsity information and regularization in the horseshoe and other shrinkage priors

Juho Piironen and Aki Vehtari

A Bayesian example

```r
set.seed(123)
n=50
x=seq(0,1,length.out=n)
alpha=0.1
k1=function(x,alpha) exp(-x/alpha)
K1=k1(as.matrix(dist(x)),alpha)
K1_ = chol(K1 + diag(1e-8,n))
gamma = cbind(rnorm(n))
z=K1%*%gamma
z_= z+ rnorm(n,0,1)
f=lm.fit(x=K1,y=z_)
plot(x,z_,pch=16)
lines(x,f$fitted.values,col='red')
lines(x,z,col='blue')
library(glmnet)
lambdas <- 10^seq(5, -5, length.out=100)
cv_fit <- cv.glmnet(K1, z_, alpha = 0, lambda = lambdas)
opt_lambda <- cv_fit$lambda.min
fit <- glmnet(K1, z_, alpha = 0, lambda = opt_lambda)
f2 <- predict(fit, s = opt_lambda, newx = K1)
lines(x,f2,col='green',lwd=2)
```

A Bayesian example

```
library(rstanarm)
prior <- normal( location=0, scale=0.1)
post1 <- stan_glm(as.formula(paste0('y ~
-1+',paste0('x.',1:nrow(K1),collapse="+"))), data = data.frame(y=z_,x=K1),
                  family = gaussian(link = "identity"),iter =
5000,chains=1,prior = prior,thin=10)

lines(x, post1$fitted.values,col='brown',lwd=2)

draws=as.matrix(post1)
preds = matrix(nrow=nrow(draws),ncol=n)
for(i in 1:nrow(draws)){
   preds[i,] =as.vector(K1%*%cbind(draws[i,1:n]))
   lines(x, preds[i,],col=rgb(165,42,42,alpha=10,maxColorValue=255),lwd=2)
}
```

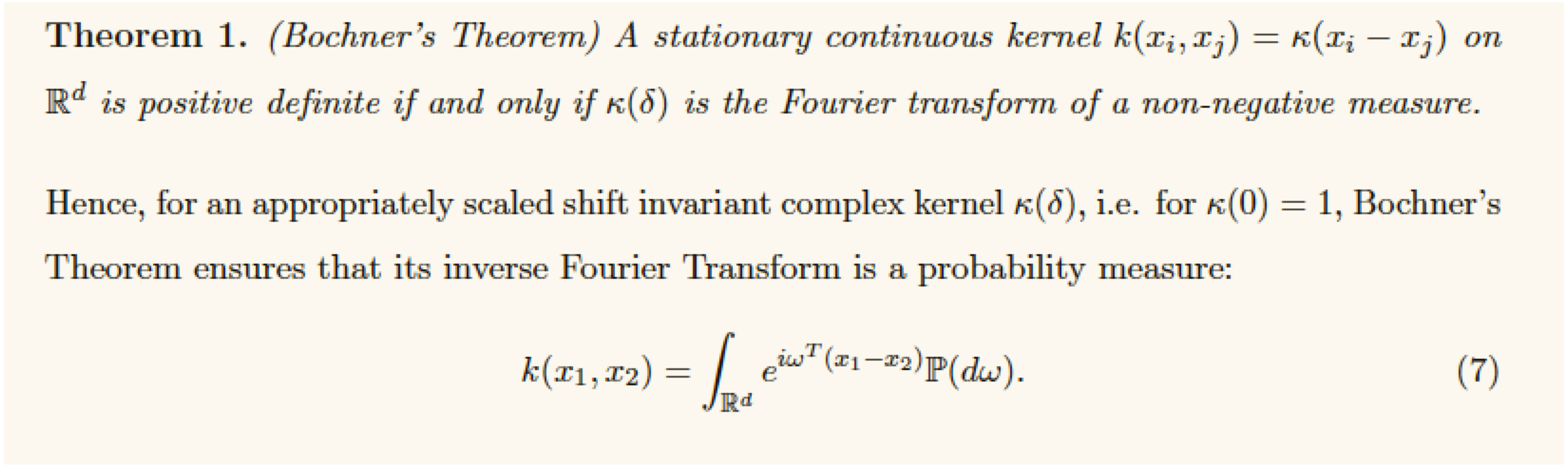
Lets take stock

- We have learned what kernels are and why they are amazing
- We can solve the risk minimisation problem via the representer theorem
- We realise that overfitting is a possibility and control for that via the ridge penalty

Whats the problem then?

- Complexity for linear regression $\mathcal{O}(nm^2)$ and therefore for kernels it becomes $\mathcal{O}(nm^2) \rightarrow \mathcal{O}(n^3)$
- Can we go back to $\mathcal{O}(nm^3)$ while still having an infinitely implied space?
- The answer amazingly is yes, but also kind of….

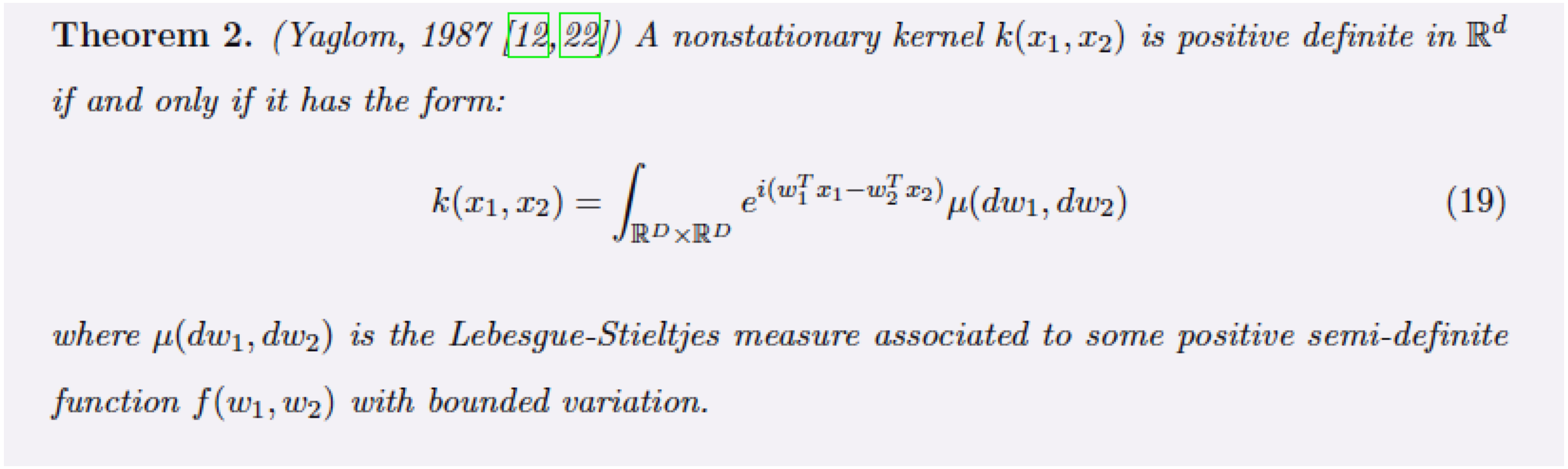
Random Fourier features (Rahimi and Recht 2009)



Theorem 1. *(Bochner's Theorem) A stationary continuous kernel $k(x_i, x_j) = \kappa(x_i - x_j)$ on $\mathbb{R}^d$ is positive definite if and only if $\kappa(\delta)$ is the Fourier transform of a non-negative measure.*

Hence, for an appropriately scaled shift invariant complex kernel $\kappa(\delta)$, i.e. for $\kappa(0) = 1$, Bochner's Theorem ensures that its inverse Fourier Transform is a probability measure:

$$k(x_1, x_2) = \int_{\mathbb{R}^d} e^{i\omega^T(x_1 - x_2)} \mathbb{P}(d\omega). \tag{7}$$



Theorem 2. *(Yaglom, 1987 [12, 22]) A nonstationary kernel $k(x_1, x_2)$ is positive definite in $\mathbb{R}^d$ if and only if it has the form:*

$$k(x_1, x_2) = \int_{\mathbb{R}^D \times \mathbb{R}^D} e^{i(w_1^T x_1 - w_2^T x_2)} \mu(dw_1, dw_2) \tag{19}$$

where $\mu(dw_1, dw_2)$ is the Lebesgue-Stieltjes measure associated to some positive semi-definite function $f(w_1, w_2)$ with bounded variation.

What is $\mathbb{P}(d\omega)$

Kernel	**Kernel Function,** $k(\mathbf{x}_i, \mathbf{x}_j)$	**Power Spectral Density,** $\mathbb{P}(\boldsymbol{\omega})$
Squared Exponential[†◇]	$\sigma^2 \exp\left(-\frac{(\mathbf{x}_i-\mathbf{x}_j)^2}{2\ell^2}\right)$	$(2\pi) - \frac{D}{2}\sigma^D \exp\left(\frac{\sigma^D \lVert\boldsymbol{\omega}\rVert_2^2}{2}\right)$
Matén[*†‡]	$\frac{2^{1-\lambda}}{\Gamma(\lambda)}\left(\frac{\sqrt{2\lambda}\lVert\mathbf{x}_i-\mathbf{x}_j\rVert_2}{\sigma}\right)^{\lambda} k_\lambda\left(\frac{\sqrt{2\lambda}\lVert\mathbf{x}_i-\mathbf{x}_j\rVert_2}{\sigma}\right)$	$\frac{2^{D+\lambda}\pi^{\frac{D}{2}}\Gamma(\lambda+\frac{D}{2})\lambda^\lambda}{\Gamma(\lambda)\sigma^{2\lambda}}\left(\frac{2\lambda}{\sigma^2}+4\pi^2\lVert\boldsymbol{\omega}\rVert_2^2\right)^{-(\lambda+\frac{D}{2})}$
Cauchy	$\prod_{i=1}^{D}\frac{2}{1+(\mathbf{x}_i-\mathbf{x}_j)^2}$	$\exp(-\lVert\boldsymbol{\omega}\rVert_1)$
Laplacian[†]	$\exp\left(-\sigma\lVert\mathbf{x}_i-\mathbf{x}_j\rVert_1\right)$	$\left(\frac{2}{\pi}\right)^{\frac{D}{2}}\prod_{i=1}^{D}\frac{\sigma}{\sigma^2+\omega_i^2}$

[*] *$\Gamma(\cdot)$ is the gamma function and $k_\lambda(\cdot)$ is the modified Bessel function of the second kind.* [†] *Parameter $\sigma > 0$.* [‡] *Parameter $\lambda > 0$.* [◇] *Parameter $\ell > 0$.*

Gaussian
Student T
exponential
Cauchy

Random Fourier Features (Bochner)

$$\begin{aligned}
k(x_1, x_2) &= \int_{\mathbb{R}^D} e^{i\omega^T(x_1 - x_2)} \mathbb{P}(d\omega) \\
&= \mathbb{E}_{\omega \sim \mathbb{P}} \left[e^{i\omega^T(x_1 - x_2)} \right], \\
&= \mathbb{E}_{\omega \sim \mathbb{P}} \left[\cos(\omega^T(x_1 - x_2)) + i \sin(\omega^T(x_1 - x_2)) \right] \\
&= \mathbb{E}_{\omega \sim \mathbb{P}} \left[\cos(\omega^T(x_1 - x_2)) \right] \\
&= \mathbb{E}_{\omega \sim \mathbb{P}} \left[\cos(\omega^T x_1) \cos(\omega^T x_2) + \sin(\omega^T x_1) \sin(\omega^T x_2) \right] \\
&\approx \frac{1}{m} \sum_{k=1}^{m} \left(\cos(\omega_k^T x_1) \cos(\omega_k^T x_2) + \sin(\omega_k^T x_1) \sin(\omega_k^T x_2) \right) \\
&= \frac{1}{m} \sum_{k=1}^{m} \Phi_k(x_1)^T \Phi_k(x_2)
\end{aligned}$$

where $\{\omega_k\}_{k=1}^m \overset{i.i.d.}{\sim} \mathbb{P}$ and we denoted

$$\Phi_k(x_l) = \begin{pmatrix} \cos(\omega_k^T x_l) \\ \sin(\omega_k^T x_l) \end{pmatrix}.$$

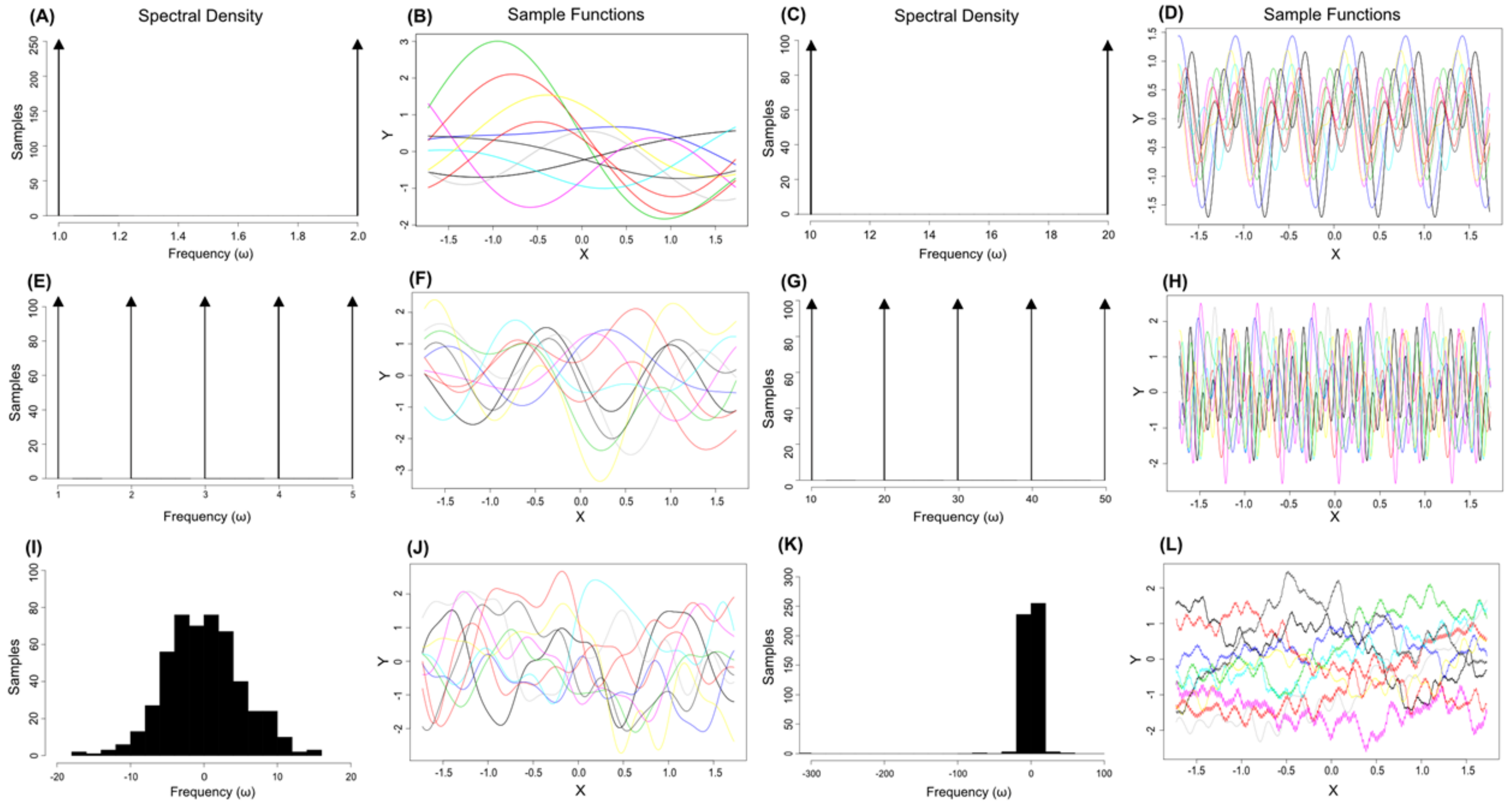
Non-stationary kernels

$$
\begin{aligned}
k(x_1, x_2) &= \frac{1}{4}\int_{\mathbb{R}^D\times\mathbb{R}^D} \left(e^{i(\omega_1^T x_1 - \omega_2^T x_2)} + e^{i(\omega_2^T x_1 - \omega_1^T x_2)} + e^{i(\omega_1^T x_1 - \omega_1^T x_2)} + e^{i(\omega_2^T x_1 - \omega_2^T x_2)}\right)\mu(d\omega_1 d\omega_2) \\
&= \frac{1}{4}\mathbb{E}_\mu\left(e^{i(\omega_1^T x_1 - \omega_2^T x_2)} + e^{i(\omega_2^T x_1 - \omega_1^T x_2)} + e^{i(\omega_1^T x_1 - \omega_1^T x_2)} + e^{i(\omega_2^T x_1 - \omega_2^T x_2)}\right) \\
&\approx \frac{1}{4m}\sum_{k=1}^m \left(e^{i(x_1^T\omega_k^1 - x_2^T\omega_k^2)} + e^{i(x_1^T\omega_k^2 - x_2^T\omega_k^1)} + e^{i(x_1^T\omega_k^1 - x_2^T\omega_k^1)} + e^{i(x_1^T\omega_k^2 - x_2^T\omega_k^2)}\right) \\
&= \frac{1}{4m}\sum_{k=1}^m \Big\{\cos(x_1^T\omega_k^1)\cos(x_2^T\omega_k^1) + \cos(x_1^T\omega_k^1)\cos(x_2^T\omega_k^2) \\
&\qquad + \cos(x_1^T\omega_k^2)\cos(x_2^T\omega_k^1) + \cos(x_1^T\omega_k^2)\cos(x_2^T\omega_k^2) \\
&\qquad + \sin(x_1^T\omega_k^1)\sin(x_2^T\omega_k^1) + \sin(x_1^T\omega_k^1)\sin(x_2^T\omega_k^2) \\
&\qquad + \sin(x_1^T\omega_k^2)\sin(x_2^T\omega_k^1) + \sin(x_1^T\omega_k^2)\sin(x_2^T\omega_k^2)\Big\} \quad \textit{(taking the real part)} \\
&= \frac{1}{4m}\sum_{k=1}^m \Phi_k(x_1)^T\Phi_k(x_2)
\end{aligned}
$$

where $\{(\omega_k^1, \omega_k^2)\}_{k=1}^m \overset{i.i.d.}{\sim} \mu$ and

$$
\Phi_k(x_l) = \begin{pmatrix} \cos(x_l^T\omega_k^1) + \cos(x_l^T\omega_k^2) \\ \sin(x_l^T\omega_k^1) + \sin(x_l^T\omega_k^2) \end{pmatrix}.
$$

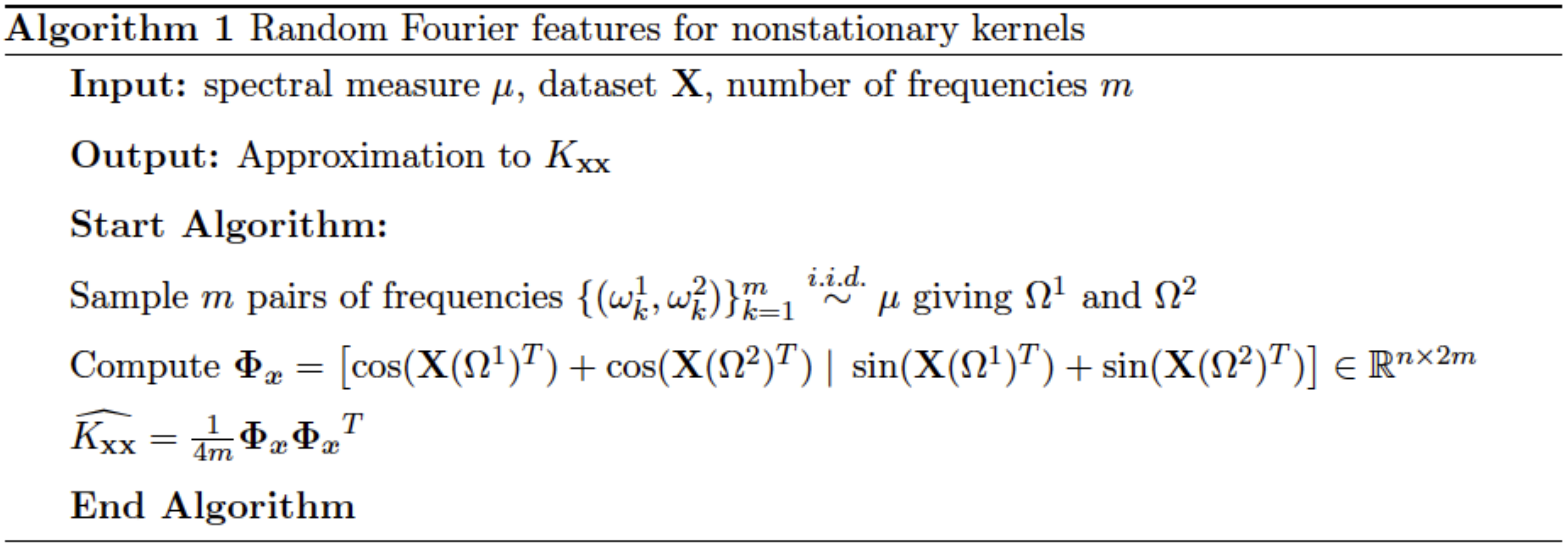
What do the functions look like?



Lets have a look

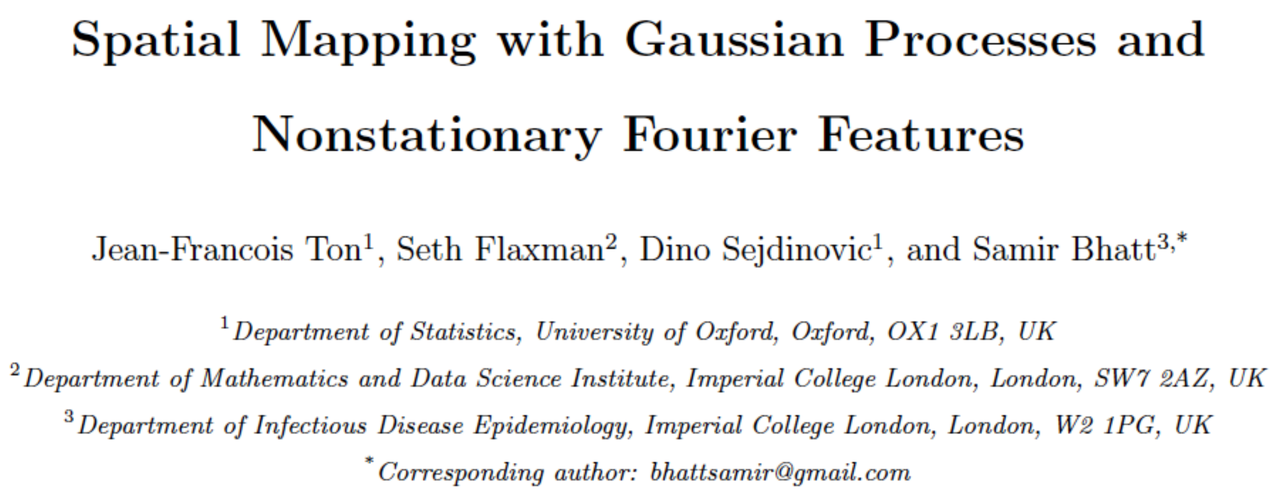
```r
set.seed(1234)
n=500
x=as.matrix(seq(-2,2,length.out=n))
x=scale(x) # remember to always scale
k=200
# change this (the spectral density)
omega1<-t(cbind(c(rep(3,k))))
omega2<-t(rnorm(k,0,5))
omega3<-t(rcauchy(k,0,2))
par(mfrow=c(3,2))
nsamp<-10
for(i in 1:4){
    eval(parse(text=paste0("omega<-omega",i)))
    xxprojected <- x%*%(omega) # project
    f<- sqrt(1/k)*cbind(cos(xxprojected),sin(xxprojected)) # monte carlo bit
    K<-(f)%*%t(f) # recreate kernel matrix
    cK=chol(K+diag(1e-6,n))
    samp<- matrix(rnorm(n*nsamp),nrow=nsamp,ncol=n) %*% cK
    samp<-t(samp)
    hist(omega,20,col='black',main="Spectral Density")
    plot(x,samp[,1],type='l',ylim=c(min(samp),max(samp)),main="Samples")
    for(j in 1:nsamp){
        lines(x,samp[,j],type='l',col=j)
    }
}
```

Its really simple



Algorithm 1 Random Fourier features for nonstationary kernels

```
    **Input:** spectral measure $\mu$, dataset $\mathbf{X}$, number of frequencies $m$
    **Output:** Approximation to $K_{\mathbf{xx}}$
    **Start Algorithm:**
    Sample $m$ pairs of frequencies $\{(\omega_k^1, \omega_k^2)\}_{k=1}^m \overset{i.i.d.}{\sim} \mu$ giving $\Omega^1$ and $\Omega^2$
    Compute $\mathbf{\Phi}_x = \left[\cos(\mathbf{X}(\Omega^1)^T) + \cos(\mathbf{X}(\Omega^2)^T) \mid \sin(\mathbf{X}(\Omega^1)^T) + \sin(\mathbf{X}(\Omega^2)^T)\right] \in \mathbb{R}^{n \times 2m}$
    $\widehat{K_{\mathbf{xx}}} = \frac{1}{4m}\mathbf{\Phi}_x {\mathbf{\Phi}_x}^T$
    **End Algorithm**
```



Spatial Mapping with Gaussian Processes and Nonstationary Fourier Features

Jean-Francois Ton[1], Seth Flaxman[2], Dino Sejdinovic[1], and Samir Bhatt[3,*]

[1] *Department of Statistics, University of Oxford, Oxford, OX1 3LB, UK*

[2] *Department of Mathematics and Data Science Institute, Imperial College London, London, SW7 2AZ, UK*

[3] *Department of Infectious Disease Epidemiology, Imperial College London, London, W2 1PG, UK*

[*] *Corresponding author: bhattsamir@gmail.com*

Or in code

```
set.seed(123)
n=500
x= cbind(runif(n),runif(n))
x=scale(x)
alpha=0.5
m=100
Omega = (cbind(rnorm(m),rnorm(m))) # Squared exponential kernel
Proj = x %*% t(Omega) # Projection - combine data with sample frequencies
Phi = sqrt(1/m) * cbind(cos(Proj/alpha), sin(Proj/alpha)) # Fourier feature
matrix
K = Phi %*% t(Phi) # approximation of Kernel matrix
d=as.matrix(dist(x))
K1<-exp(-0.5*d*d/(alpha^2))
plot(K1,K)
abline(0,1,col='red')
```

Does it look great?

- The approximation of the covariance matrix is not great, and theoretical bounds suggest that this approach needs thousands of features
- Empirical evidence suggests otherwise, we need just a few hundred
- But theory has shown the Φ creates a provide a dense basis set for for an l_2 ball of l_2 functions
- Similar to uniform approximation of Neural networks

What is the generalisation bound $R(f^*) - R(\hat{f})$

- $\mathcal{O}(\frac{1}{\sqrt{n}} + \frac{1}{\sqrt{D}})$ Rahimi and Recht
- $\mathcal{O}(\frac{log(n)}{\sqrt{n}})$ Rudi (under certain conditions as good as full ridge)

We can use Φ in linear regression

```r
set.seed(123)
n=500
x=seq(0,1,length.out=n)
alpha=0.1
k1=function(x,alpha) exp(-x/alpha)
K1=k1(as.matrix(dist(x)),alpha)
K1_ = chol(K1 + diag(1e-8,n))
gamma = cbind(rnorm(n))
z=K1%*%gamma
z_= z+ rnorm(n,0,1)
f=lm.fit(x=K1,y=z_)
plot(x,z_,pch=16,cex=0.5)
lines(x,z,col='blue')
library(glmnet)
lambdas <- 10^seq(5, -5, length.out=100)
cv_fit <- cv.glmnet(K1, z_, alpha = 0, lambda = lambdas)
opt_lambda <- cv_fit$lambda.min
fit <- glmnet(K1, z_, alpha = 0, lambda = opt_lambda)
f2 <- predict(fit, s = opt_lambda, newx = K1)
lines(x,f2,col='green',lwd=2)
m=200
Omega = cbind(rnorm(m)) # Squared exponential kernel
Proj = scale(x) %*% t(Omega) # Projection - combine data with sample frequencies
Phi = sqrt(1/m) * cbind(cos(Proj/alpha), sin(Proj/alpha)) # Fourier feature matrix
lambdas <- 10^seq(5, -5, length.out=100)
cv_fit <- cv.glmnet(Phi, z_, alpha = 0, lambda = lambdas)
opt_lambda <- cv_fit$lambda.min
fit <- glmnet(Phi, z_, alpha = 0, lambda = opt_lambda)
f2 <- predict(fit, s = opt_lambda, newx = Phi)
lines(x,f2,col='brown',lwd=2)
```

Why use RFFs?

- Easy to program
- Can create high dimensional kernels without too much loss (its all about integration)
- Its approximates the whole kernel and not just a low rank one based on some data
- It casts everything as linear regression which makes it easy to customise

How can we improve things?

- Quasi-Monte Carlo integration - Halton Sequence

```r
set.seed(123)
n=500
x= cbind(runif(n),runif(n))
x=scale(x)
alpha=0.5
m=100
d=as.matrix(dist(x))
K1<-exp(-0.5*d*d/(alpha^2))
Omega = cbind(rnorm(m),rnorm(m)) # Squared exponential kernel
Proj = x %*% t(Omega) # Projection - combine data with sample frequencies
Phi = sqrt(1/m) * cbind(cos(Proj/alpha), sin(Proj/alpha)) # Fourier feature matrix
K = Phi %*% t(Phi) # approximation of Kernel matrix
par(mfrow=c(1,2))
plot(K1,K)
abline(0,1,col='red')
library(randtoolbox)
Omega_qmc<-t(qnorm(halton(m,2)))
Omega_qmc = cbind(rnorm(m),rnorm(m)) # Squared exponential kernel
Proj = x %*% t(Omega) # Projection - combine data with sample frequencies
Phi = sqrt(1/m) * cbind(cos(Proj/alpha), sin(Proj/alpha)) # Fourier feature matrix
K2 = Phi %*% t(Phi) # approximation of Kernel matrix
plot(K2,K)
abline(0,1,col='red')
```

Use GPUs and Tensorflow

- We can train these models using mini-batch gradient descent
- Dropout can be amazing
- GPUs can be very efficient