Multi-Output Gaussian Stochastic Processes

*Materials: see link in description.

Executed in Matlab **R2024a**

Objectives: illustrate the concept and representation of a 2-output GP, via an example.

Presentations in video:

https://personales.upv.es/asala/YT/V/mimoGP1EN.html (motivation)

https://personales.upv.es/asala/YT/V/mimogp2EN.html (realizations)

https://personales.upv.es/asala/YT/V/mimogp3pcaEN.html (PCA, KL components)

https://personales.upv.es/asala/YT/V/mimogp4predAEN.html (prediction)

https://personales.upv.es/asala/YT/V/mimogp5predUEN.html (latent component estimation)

Table of Contents

Basic idea, motivation

In some applications we may have two "functions" of a given argument real x, or even multidimensional input, and functions are $f_1(x)$ and $f_2(x)$ which somehow are "correlated".

Examples:

●1● Weather sensor network (input would be latitude-longitude)...

Temperature and Humidity at "same" points. **Isotopic**.

Heterotopic: Temperature at some points, humidity at other different points.

●2● Other interesting cases in engineering: multioutput " $f_1(t) =$**position**, $f_2(t) =$**velocity**", in a 2nd order mechanical system subject to white-noise acceleration, say.

If we consider a statistical GP model of that setup, we may wish to generate realizations, or maybe we have samples $f_1(x_k)$, $f_2(x_k)$ and we wish to "interpolate" them; it will be better exploiting the "correlation" between f_1 and f_2 to improve our estimations. For example, we will generalize the idea of estimating speed from position measurements, or something like that.

Representations of multi-output GP

Multi-output (say, 2 outputs) may be consider either a function $f : \mathbb{R} \mapsto \mathbb{R}^2$ or a function $f : \mathbb{R} \times \{1, 2\} \mapsto \mathbb{R}$.

We may have "independent" GPs $k_1(x, x')$, $k_2(x, x')$, or "correlated" ones $k(x, x')$ being a 2x2 matrix... independent case: diagonal matrix, we would just think in $f_1 : \mathbb{R} \mapsto \mathbb{R}$, $f_2 : \mathbb{R} \mapsto \mathbb{R}$, separately.

***Note:** The ODE or PDE/based representation (stochastic differential equations SDE, signal processing, filtering) will **NOT** be addressed here: this PDF is just introductory/motivational, and the topic of SDE has room for a whole course in itself.

Let us detail, assuming zero mean everywhere to avoid notational clutter and to keep things simple.

As an (n+1)-dimensional input , single output GP

If we consider $f : \mathbb{R} \times \{1, 2\} \mapsto \mathbb{R}$, then we need a covariance kernel $k : \{R \times \{1, 2\}\}^2 \mapsto \mathbb{R}$, expressed as, say, $k((x, i), (x', i'))$ or, well, just $k(x, i, x', i')$.

Once we think in the "(n+1)-dimensional input, single-output" GP, there is NOTHING NEW UNDER THE HOOD: everything from single-ouput GPs applies to multi-ouput GPs (prediction, principal components, etc.). So theory ends here... The rest is "plotting commands for better intuition", but nothing "fundamental" from a conceptual point of view.

***Note:** when one output is related to the other one via an ODE (position--speed, filtering, etc.) there IS more theory under the hood. These issues are NOT in the scope of the present material.

*Two ways of "stacking" when building covariance matrices of a finite set of data: first "i", then "x", or first "x" then "i"... $f:\mathbb{R}\times\{1,2\}\mapsto\mathbb{R}$ or $f:\{1,2\}\times\mathbb{R}\mapsto\mathbb{R}$. It's the same, anyway, but I just mention it to avoid confusion when interpreting the Matlab code below.

How to obtain these covariance matrices? We cannot "invent" them, as they must be positive-definite for any finite set of data... well, we may consider (x,i) as an "extended" 1+dimensional vector, and use any usual kernel, say

$k(x,i,x',j)=Me^{-\alpha\cdot(x-x')^2-\beta\cdot(i-j)^2}$. It would be OK, but there is another option, see below...

As an "independent component/factor" representation

We consider a "component" being sampled from "independent factor GPs $u_i(x)$", each one with a different kernel, if so wished. Later on, obtain $f_1(x)$ and $f_2(x)$ via a linear transformation (a matrix C) of those independent factors $u_i(x)$.

Thus, we will assume, for instance: $f_{2\times1}(x)=C_{2\times3}\cdot u_{3\times1}(x)$, being $u_i(x)$ realizations of three independent GPs.

```
U=cell(3,1);
sgu1=3; sgu2=0.75; sgu3=.3; %stationary variances
dd1=2;  dd2=0.65; dd3=0.2; %distance parameters
U{1}=@(x1,x2) sgu1^2*exp(-norm(x2-x1)^2/dd1^2);%kernel for component 1
U{2}=@(x1,x2) sgu2^2*exp(-norm(x2-x1)^2/dd2^2); %kernel 2
U{3}=@(x1,x2) sgu3^2*exp(-norm(x2-x1)^2/dd3^2);%kernel 3
C=[1 1 0;-1 0 1]
```

```
C = 2×3
     1     1     0
    -1     0     1
```

The "factors" may be considered "latent" variables, which may be worth estimating if so wished (see below), if there is any "physical" interpretation of interest for them.

Note that we are NOT discussing "how" or "why" we arrived to this statistical GP model... it comes from whoever studied the data-based or theoretical modelling phase.

Covariance matrix computation: So, $f_{2\times1}(x)=C_{2\times3}\cdot u_{3\times1}(x)$ would entail that

$cov(f(x),f(x'))=E[f(x)\cdot f(x')^T]=C\cdot E[u(x)\cdot u(x')^T]\cdot C^T=C\cdot K(x,x')\cdot C^T$

being $K(x,x')$ a diagonal matrix with the covariance kernels of each of the "independent" components".

```
ComponentCovariance=@(x1,x2) diag([U{1}(x1,x2) U{2}(x1,x2) U{3}(x1,x2)]);
FkernelMatrix=@(x1,x2) C*ComponentCovariance(x1,x2)*C'; %MATRIX output
[UNUSED]
```

It can be easily seen, developing the matrix product expressions, that

$$cov(f_i(x), f_j(x')) = E[f(x) \cdot f(x')^T] = C_{i,:} \cdot E[u(x) \cdot u(x')^T] \cdot C^T = C \cdot K(x, x') \cdot (C_{j,:})^T$$

Let's program that as a function for later use:

```
Fkernel=@(x1,i1,x2,i2) ...
    C(i1,:)*ComponentCovariance(x1,x2)*C(i2,:)'; %Scalar output of Fkernel
```

Samples, realizations (example)

```
%Xtest=(-5:0.05:5)';
Xtest=(-4:0.05:4)';
Ntest=length(Xtest)
```

```
Ntest = 161
```

Representation 1: full covariance 2x2 matrix kernel

We will stack input samples as "x index" in rows $[Xtest\ \ 1; Xtest\ \ 2; \cdots]$:

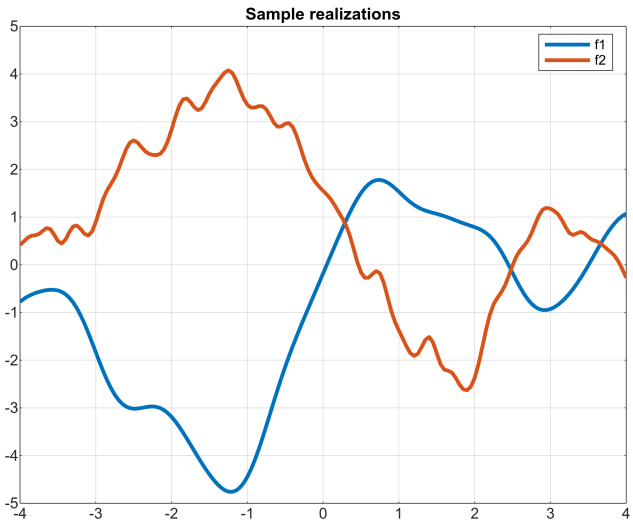
```
XtestTwice=[Xtest; Xtest];
XtestLabels=kron([1;2],ones(size(Xtest)));
```

We compute the full covariance matrix:

```
Ktest=K(XtestTwice,XtestLabels,XtestTwice,XtestLabels, Fkernel);
size(Ktest)
```

```
ans = 1×2
   322   322
```

```
Samples=mvnrnd(zeros(length(XtestTwice),1),Ktest);
plot(Xtest,[Samples(1:Ntest); Samples((Ntest+1):end)],LineWidth=2.5), grid on
legend("f1","f2",Location="best"), title("Sample realizations")
```



Representation 2: from component realizations

Of course, we might have preferred obtaining three separate SISO realizations for each of the u1(x) ,u2(x), u3(x) components and later on multiply by C. Results would be equivalent, clearly. Let's do it:

```
PriorVarU1=K_siso(Xtest,Xtest,U{1});
PriorVarU2=K_siso(Xtest,Xtest,U{2});
PriorVarU3=K_siso(Xtest,Xtest,U{3});
U1sample=mvnrnd(zeros(length(Xtest),1),PriorVarU1); size(U1sample)
```

```
ans = 1×2
     1   161
```

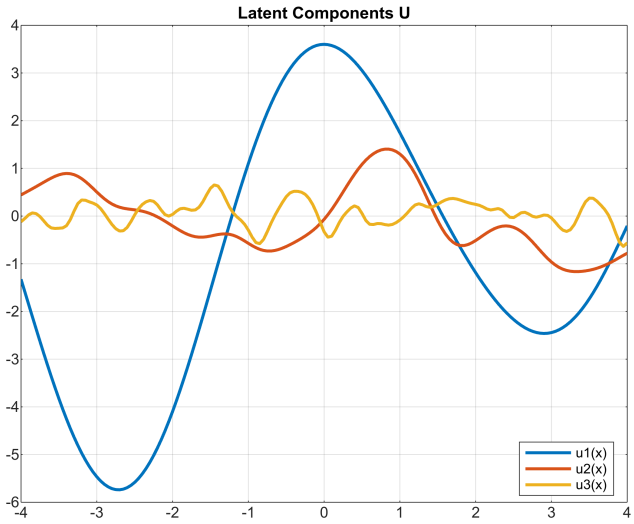
```
U2sample=mvnrnd(zeros(length(Xtest),1),PriorVarU2); size(U2sample)
```

```
ans = 1×2
     1   161
```

```
U3sample=mvnrnd(zeros(length(Xtest),1),PriorVarU3); size(U3sample)
```

```
ans = 1×2
     1   161
```

```
plot(Xtest,[U1sample;U2sample;U3sample],LineWidth=2), grid on
legend("u1(x)","u2(x)","u3(x)",Location="best"), title("Latent Components U")
```

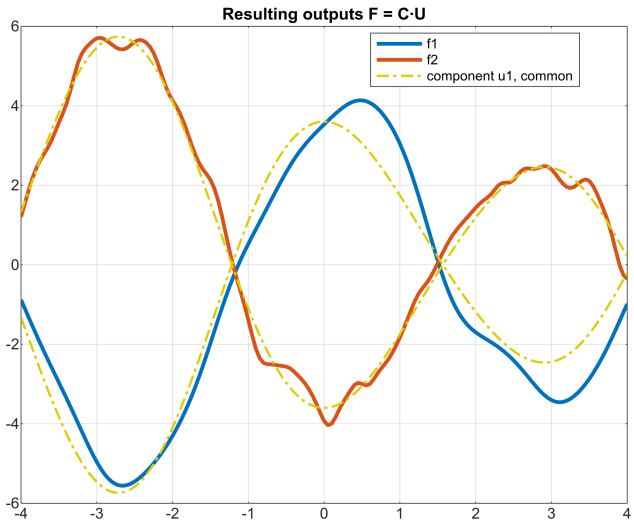



```
Fsample=C*[U1sample;U2sample;U3sample];
size(Fsample)
```

```
ans = 1×2
     2   161
```

```
plot(Xtest,Fsample,LineWidth=2.5), grid on,
hold on
plot(Xtest,[U1sample;-U1sample],"-.",LineWidth=1.5,Color=[.87 .8 0])
hold off, legend("f1","f2","component u1, common",Location="best")
```

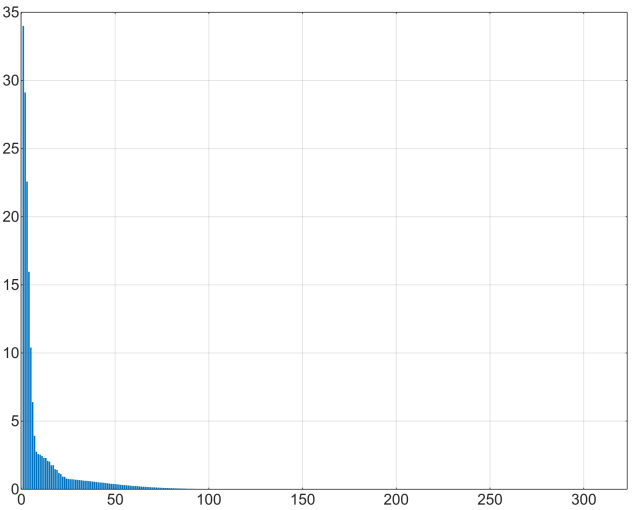
```
title("Resulting outputs F = C·U")
```



Karhunen-Loeve eigenfunctions (principal component analysis if finite number of samples)

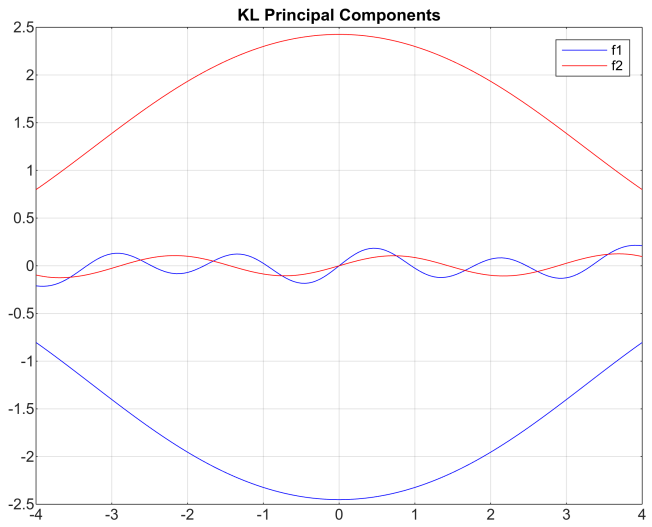
Let's analyze eigenvectors and eigenvalues of the covariance matrix, taking into account the 2-output structure:

```
[V,D]=eig(Ktest);
PrincStd=sqrt(diag(D)+3e-14);
[PrincStdSorted,idx]=sort(PrincStd,1,"descend");
bar(PrincStdSorted), grid on
```




```
PrincCom=V(:,idx)*diag(PrincStdSorted); %eigenvectors, reordered
for CompNumber=[1 16]
    plot(Xtest,PrincCom(1:Ntest,CompNumber),'b' )
    hold on
    plot(Xtest,PrincCom((Ntest+1):end,CompNumber),'r')
```

```
end
hold off, grid on, legend("f1","f2"), title("KL Principal Components")
```




```
%return %change C to 1 -1... or not?
```

Prediction

Well, nothing too new once we have the covariance matrix, understood as $k(x, i, x', i')$, whatever the modelling steps we thought to build it: from some samples (x_k, i_k) we can estimate, in mean and variance, a "posterior" at any position and index.

Zero mean assumed, for simplicity.

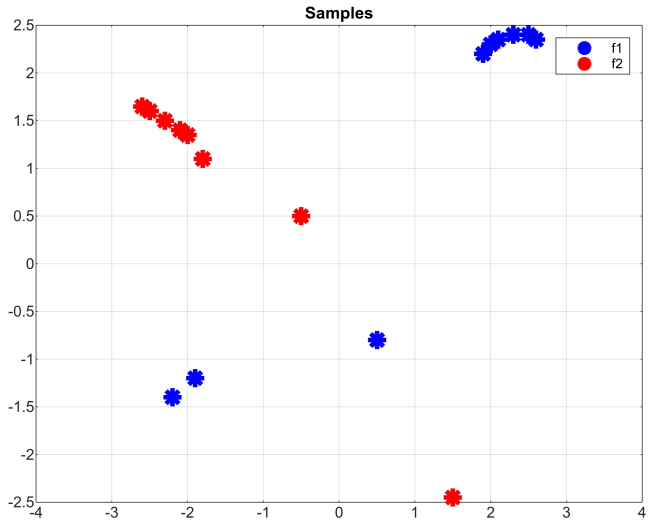
```
example=3;

%position and sensor number data
if example==1
    %Example 1: combination of data
    DataXI=[2 1;
        1.5 2;
        0.5  1;
        -0.5  2;
        -1.9 1;
        -1.8 2;
        -2 2];
    %observed sensor outputs
    DataY=[2.3; -2.45; -0.8; 0.5;-1.2; 1.1; 1.35]


elseif example==3
    %Example 1: combination of data
    DataXI=[2 1;
        1.5 2;
```

```
        0.5  1;
        -0.5  2;
        -1.9 1;
        -1.8 2;
        -2 2;
        1.9 1;
        2.1 1;
        2.3 1;
        2.5 1;
        2.6 1;
        -2.1 2;
        -2.3 2;
        -2.5 2;
        -2.6 2;
        -2.2 1]
    %observed sensor outputs
    DataY=[2.3; -2.45; -0.8; 0.5;-1.2; 1.1; 1.35; ...
        2.2; 2.35;2.4; 2.4; 2.35; 1.4;1.5;1.6;1.65;-1.4];


else
    %Example 2: use just one to predict another:
    DataXI=[2 1;
        1.5 1;
        0.5  1;
        -0.5  1;
        -1.9 1;
        -1.8 1;
        -2 1 ];
    %observed sensor outputs
    DataY=[2.3; 2.45; 0.8; 0.5;1.2; 1.1; 1.35];
end
```

```
DataXI = 17×2
    2.0000    1.0000
    1.5000    2.0000
    0.5000    1.0000
   -0.5000    2.0000
   -1.9000    1.0000
   -1.8000    2.0000
   -2.0000    2.0000
    1.9000    1.0000
    2.1000    1.0000
    2.3000    1.0000
     ⋮
```

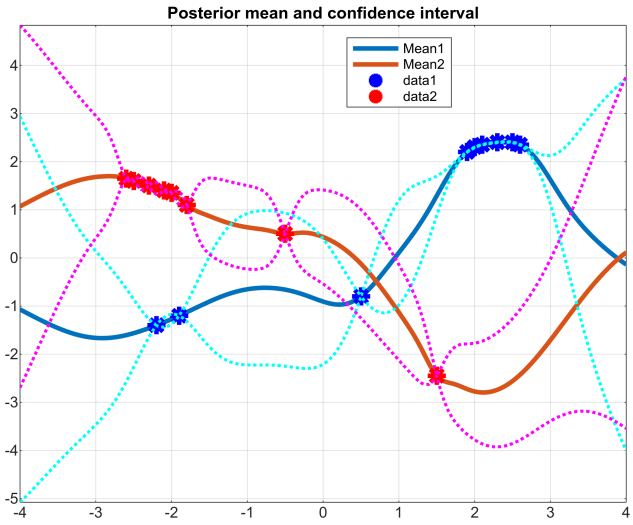
```
 Nsamples=length(DataY);

idx1=find(DataXI(:,2)==1); Nsamples1=length(idx1);
idx2=find(DataXI(:,2)==2); Nsamples2=length(idx2);
%observed sensor outputs
```

```
plot(DataXI(idx1,1),DataY(idx1),'b*',LineWidth=3,MarkerSize=12)
hold on
plot(DataXI(idx2,1),DataY(idx2),'r*',LineWidth=3,MarkerSize=12)
hold off, grid on, title("Samples"), legend("f1","f2"), xlim([min(Xtest)
max(Xtest)])
```




```
noisevar=1e-5; %Measurement noise variance

PriorVar1=K(Xtest,ones(size(Xtest)),Xtest,ones(size(Xtest)),Fkernel);
PriorVar2=K(Xtest,2*ones(size(Xtest)),Xtest,2*ones(size(Xtest)),Fkernel);

Kinfo=K(DataXI(:,1),DataXI(:,2),DataXI(:,1),DataXI(:,2),Fkernel) ...
    +noisevar*eye(Nsamples);
Ktest1info=K(Xtest,ones(size(Xtest)),DataXI(:,1),DataXI(:,2),Fkernel);
Ktest2info=K(Xtest,2*ones(size(Xtest)),DataXI(:,1),DataXI(:,2),Fkernel);

PosteriorMean1=0+(Ktest1info/Kinfo)*DataY;
PosteriorMean2=0+(Ktest2info/Kinfo)*DataY;
PosteriorVar1=PriorVar1-(Ktest1info/Kinfo)*Ktest1info';
std1=sqrt(diag(PosteriorVar1));
PosteriorVar2=PriorVar2-(Ktest2info/Kinfo)*Ktest2info';
std2=sqrt(diag(PosteriorVar2));
%we are disregarding correlation between f1 and f2 in
%the posterior... joint realizations/conf. ellipsoids would need it, but
%marginal confidence intervals do not need it.

plot(Xtest,[PosteriorMean1 PosteriorMean2],LineWidth=3), grid on
hold on
plot(DataXI(idx1,1),DataY(idx1),'b*',LineWidth=3,MarkerSize=12)
plot(DataXI(idx2,1),DataY(idx2),'r*',LineWidth=3,MarkerSize=12)
plot(Xtest,[PosteriorMean1-2*std1 PosteriorMean1+2*std1],'c:',LineWidth=2)
plot(Xtest,[PosteriorMean2-2*std2 PosteriorMean2+2*std2],'m:',LineWidth=2)
hold off
```

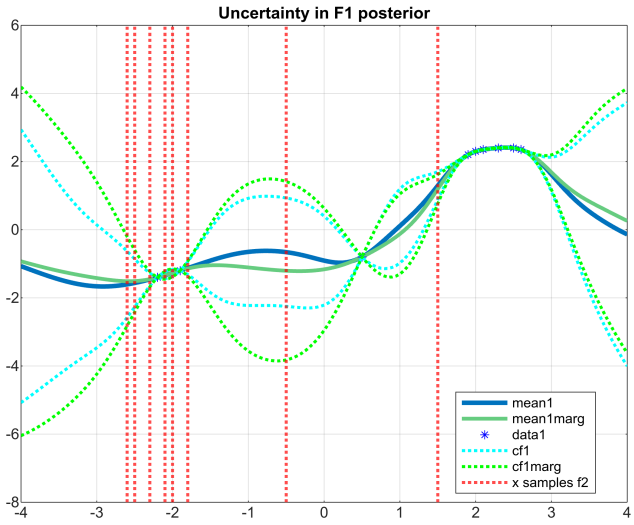
```
legend("Mean1","Mean2","data1","data2", Location="best"), axis tight
title("Posterior mean and confidence interval")
```



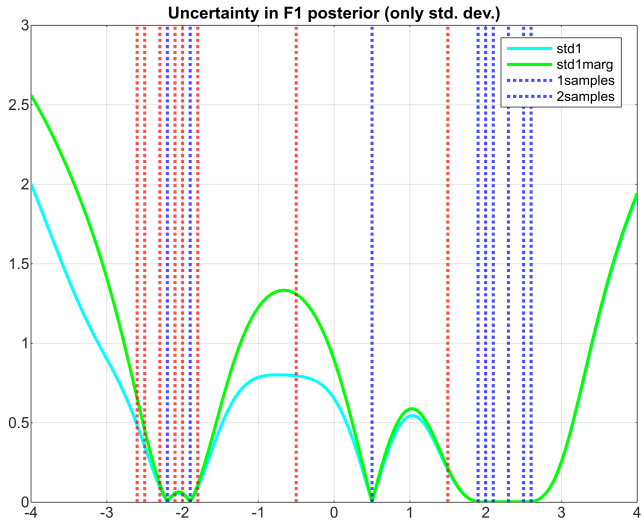
Let us compare the prediction of one of the functions in a "marginal" way, without info about the other, to see whether it is worthwhile the "joint" analysis:

```
Kinfo1=K(DataXI(idx1,1),DataXI(idx1,2),DataXI(idx1,1),DataXI(idx1,2),Fkernel)
+noisevar*eye(Nsamples1);
Kinfo2=K(DataXI(idx2,1),DataXI(idx2,2),DataXI(idx2,1),DataXI(idx2,2),Fkernel)
+noisevar*eye(Nsamples2);
Ktest1info=K(Xtest,ones(size(Xtest)),DataXI(idx1,1),DataXI(idx1,2),Fkernel);
Ktest2info=K(Xtest,2*ones(size(Xtest)),DataXI(idx2,1),DataXI(idx2,2),Fkernel)
;

PosteriorMean1Marg=0+(Ktest1info/Kinfo1)*DataY(idx1);
PosteriorMean2Marg=0+(Ktest2info/Kinfo2)*DataY(idx2);
PosteriorVar1Marg=PriorVar1-(Ktest1info/Kinfo1)*Ktest1info';
std1Marg=sqrt(diag(PosteriorVar1Marg));
PosteriorVar2Marg=PriorVar2-(Ktest2info/Kinfo2)*Ktest2info';
std2Marg=sqrt(diag(PosteriorVar2Marg));

plot(Xtest,PosteriorMean1,LineWidth=3), grid on
hold on
plot(Xtest,PosteriorMean1Marg,Color=[0.4 .8 .5],LineWidth=2.5), grid on
plot(DataXI(idx1,1),DataY(idx1),'b*')
plot(Xtest,[PosteriorMean1-2*std1 PosteriorMean1+2*std1],'c:',LineWidth=2)
plot(Xtest,[PosteriorMean1Marg-2*std1Marg
PosteriorMean1Marg+2*std1Marg],'g:',LineWidth=2)
hold off
if ~isempty(idx2)
    xline(DataXI(idx2,1),'r:',LineWidth=2)
end
```

```
legend("mean1","mean1marg","data1","","cf1","cf1marg","","x samples
f2",Location="best")
title("Uncertainty in F1 posterior")
```

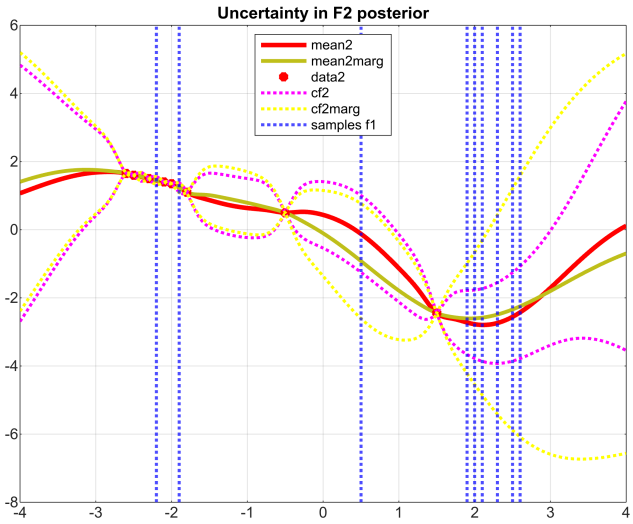



```
plot(Xtest,std1,'c',Xtest, std1Marg,'g',LineWidth=2)
xline(DataXI(idx1,1),'b:',LineWidth=2)
if ~isempty(idx2)
    xline(DataXI(idx2,1),'r:',LineWidth=2)
end
grid on
legend("std1","std1marg","1samples","","","2samples",Location="best")
title("Uncertainty in F1 posterior (only std. dev.)")
```

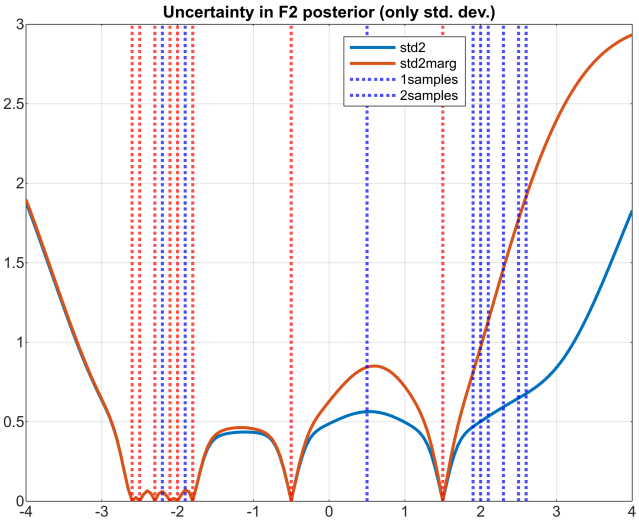



```
plot(Xtest,PosteriorMean2,'r',LineWidth=3), grid on
hold on
plot(Xtest,PosteriorMean2Marg,'r',Color=[.75 .75 .1],LineWidth=2.5), grid on
plot(DataXI(idx2,1),DataY(idx2),'r*',LineWidth=3)
```

```
plot(Xtest,[PosteriorMean2-2*std2 PosteriorMean2+2*std2],'m:',LineWidth=2)
plot(Xtest,[PosteriorMean2Marg-2*std2Marg
PosteriorMean2Marg+2*std2Marg],'y:',LineWidth=2)
hold off
xline(DataXI(idx1,1),'b:',LineWidth=2)
legend("mean2","mean2marg","data2","","cf2","cf2marg","","samples
f1",Location="best")
title("Uncertainty in F2 posterior")
```




```
plot(Xtest,[std2 std2Marg],LineWidth=2), grid on
xline(DataXI(idx1,1),'b:',LineWidth=2)
if ~isempty(idx2)
    xline(DataXI(idx2,1),'r:',LineWidth=2)
end
legend("std2","std2marg","1samples","","","","2samples",Location="best")
title("Uncertainty in F2 posterior (only std. dev.)")
```



Estimating the latent factors u_i(x)

OK, sort of Kalman filter: predict the "u_i" from the "y_i"... we would name in control x_i instead of u_i? $y = Cx$ we say in control, but it's just notation.

But instead of functions of "time" can be functions of "anything".

If $f_{2\times1}(x) = C_{2\times3} \cdot u_{3\times1}(x)$, we can say that $cov(u(x), f(x')) = E[u(x)f(x')^T] = E[u(x)u(x')^T \cdot C^T] = K(x,x') \cdot C^T$.

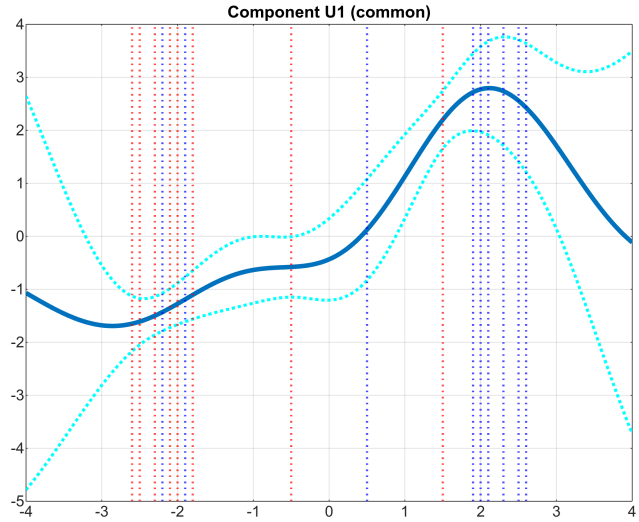
As components are independent (diagonal K), $cov(u_i(x), f_j(x'))$ ends up being $k_i(x,x') \cdot c_{ji}$, which is the "ij" element of covariance matrix $K(x,x') \cdot C^T$. Indeed:

$$K \cdot C^T = \begin{pmatrix} k_1 & 0 & 0 \\ 0 & k_2 & 0 \\ 0 & 0 & k_3 \end{pmatrix} \begin{pmatrix} c_{11} & c_{21} \\ c_{12} & c_{22} \\ c_{13} & c_{23} \end{pmatrix} = \begin{pmatrix} k_1c_{11} & k_1c_{21} \\ k_2c_{12} & k_2c_{22} \\ k_3c_{13} & k_3c_{23} \end{pmatrix}$$

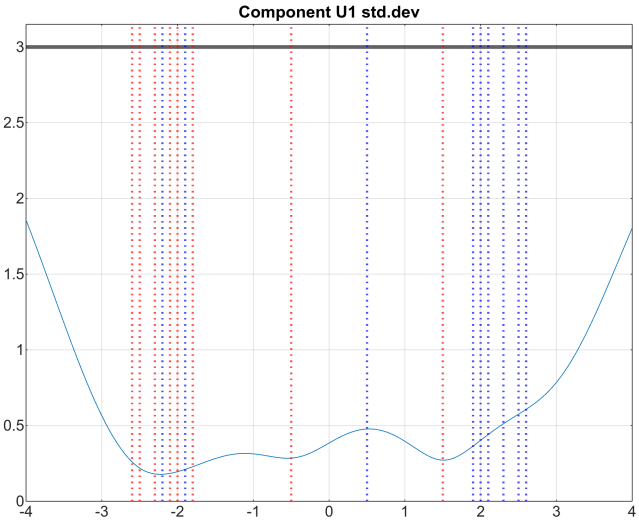
So we can use that covariance to estimate the latent factors from the available information.

```
%KernelUF=@(x1,x2) ComponentCovariance(x1,x2)*C'; %UNUSED
KernelUF=@(x1,component,x2,outputlabel) ...
    U{component}(x1,x2)*C(outputlabel,component);



KtestU1info=K_UF(Xtest,1,DataXI(:,1),DataXI(:,2),KernelUF);
KtestU2info=K_UF(Xtest,2,DataXI(:,1),DataXI(:,2),KernelUF);
KtestU3info=K_UF(Xtest,3,DataXI(:,1),DataXI(:,2),KernelUF);

PosteriorMeanU1=0+(KtestU1info/Kinfo)*DataY;
PosteriorMeanU2=0+(KtestU2info/Kinfo)*DataY;
PosteriorMeanU3=0+(KtestU3info/Kinfo)*DataY;

PosteriorVarU1=PriorVarU1-(KtestU1info/Kinfo)*KtestU1info';
stdU1=sqrt(diag(PosteriorVarU1));
PosteriorVarU2=PriorVarU2-(KtestU2info/Kinfo)*KtestU2info';
stdU2=sqrt(diag(PosteriorVarU2));
PosteriorVarU3=PriorVarU3-(KtestU3info/Kinfo)*KtestU3info';
stdU3=sqrt(diag(PosteriorVarU3));



plot(Xtest,PosteriorMeanU1,LineWidth=3)
hold on
```

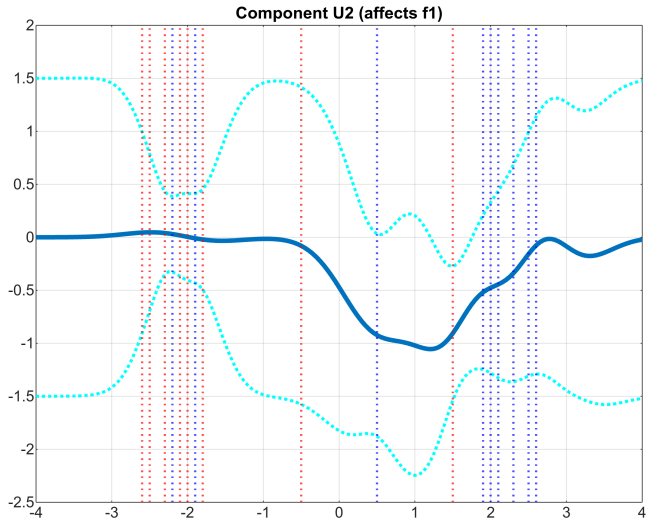
```
plot(Xtest,[PosteriorMeanU1-2*stdU1
PosteriorMeanU1+2*stdU1],'c:',LineWidth=2)
hold off, grid on, title("Component U1 (common)")
xline(DataXI(idx1,1),'b:',LineWidth=1.5)
if ~isempty(idx2)
    xline(DataXI(idx2,1),'r:',LineWidth=1.5)
end
```



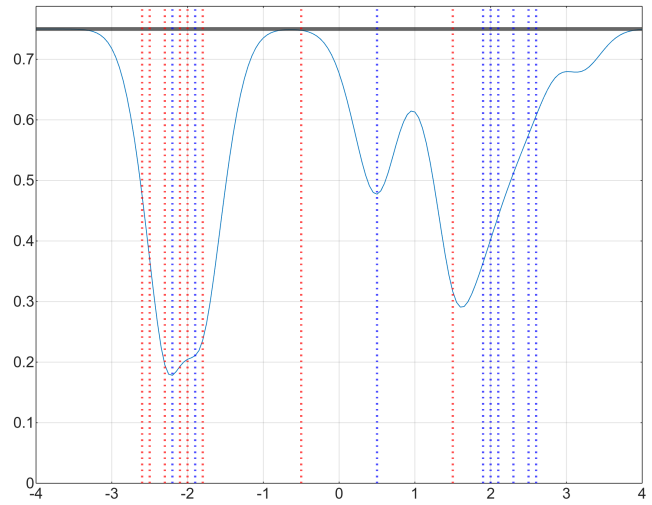

```
plot(Xtest,stdU1), yline(sgu1,LineWidth=2.5)
xline(DataXI(idx1,1),'b:',LineWidth=1.5)
if ~isempty(idx2)
    xline(DataXI(idx2,1),'r:',LineWidth=1.5)
end
grid on, ylim([0 sgu1*1.05])
title("Component U1 std.dev")
```



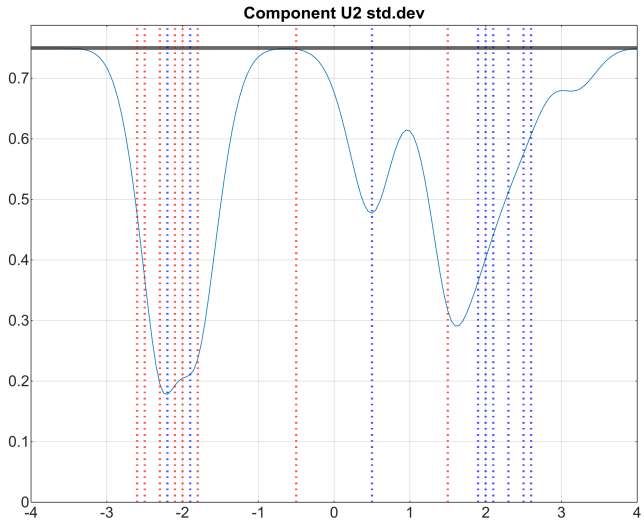
```
plot(Xtest,PosteriorMeanU2,LineWidth=3)
hold on
plot(Xtest,[PosteriorMeanU2-2*stdU2
PosteriorMeanU2+2*stdU2],'c:',LineWidth=2)
hold off, grid on, title("Component U2 (affects f1)")
xline(DataXI(idx1,1),'b:',LineWidth=1.5)
if ~isempty(idx2)
    xline(DataXI(idx2,1),'r:',LineWidth=1.5)
end
```



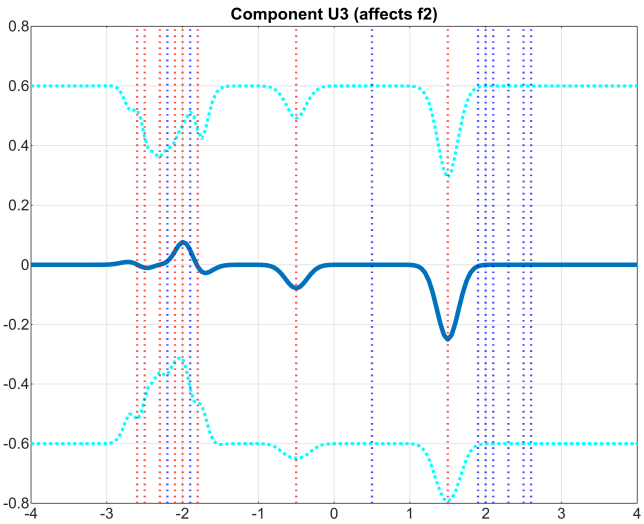

```
plot(Xtest,stdU2), yline(sgu2,LineWidth=2.5), grid on, ylim([0 sgu2*1.05])
xline(DataXI(idx1,1),'b:',LineWidth=1.5)
if ~isempty(idx2)
    xline(DataXI(idx2,1),'r:',LineWidth=1.5)
end
```



```
title("Component U2 std.dev")
```

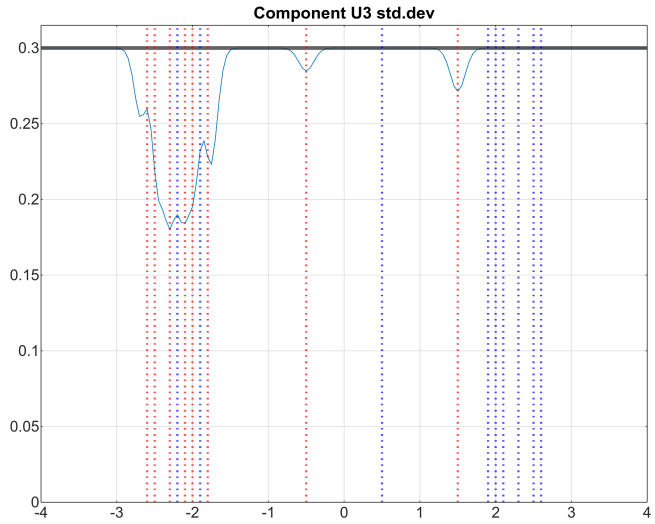



```
plot(Xtest,PosteriorMeanU3,LineWidth=3)
hold on
plot(Xtest,[PosteriorMeanU3-2*stdU3
PosteriorMeanU3+2*stdU3],'c:',LineWidth=2)
hold off, grid on, title("Component U3 (affects f2)")
xline(DataXI(idx1,1),'b:',LineWidth=1.5)
if ~isempty(idx2)
    xline(DataXI(idx2,1),'r:',LineWidth=1.5)
end
```




```
plot(Xtest,stdU3), yline(sgu3,LineWidth=2.5), grid on, ylim([0 sgu3*1.05])
```

```
xline(DataXI(idx1,1),'b:',LineWidth=1.5)
if ~isempty(idx2)
    xline(DataXI(idx2,1),'r:',LineWidth=1.5)
end
title("Component U3 std.dev")
```




```
function M=K(x1,i1,x2,i2,KernelFunction)
%Kernelfunction returns a SCALAR
%data samples in x1 and x2 are ROWS
N1=size(x1,1);
N2=size(x2,1);
M=zeros(N1,N2);
for i=1:N1
    for j=1:N2
        M(i,j)=KernelFunction(x1(i,:), i1(i), x2(j,:), i2(j));
    end
end
end

function M=K_siso(x1,x2,KernelFunction)
%Kernelfunction returns a SCALAR
%data samples in x1 and x2 are ROWS
N1=size(x1,1);
N2=size(x2,1);
M=zeros(N1,N2);
for i=1:N1
    for j=1:N2
        M(i,j)=KernelFunction(x1(i,:), x2(j,:));
    end
```

```
end
end


function M=K_UF(x1,u,x2,i2,KernelUF)
%Kernelfunction returns a SCALAR
%data samples in x1 and x2 are ROWS
% u is the "component number", i2 is the measurement available at x2.
N1=size(x1,1);
N2=size(x2,1);
M=zeros(N1,N2);
for i=1:N1
    for j=1:N2 %TODO
        M(i,j)=KernelUF(x1(i,:),u, x2(j,:),i2(j));
    end
end
end
```