Gaussian Process: Causal, Anticausal, Bilateral components [Matlab example]

*Executed in Matlab **R2023b**

Presentations in video:

http://personales.upv.es/asala/YT/V/gpcholEN.html

http://personales.upv.es/asala/YT/V/gpanticaEN.html

Objectives: A Gaussian Process is a "random function". Let us look at the "latent" variables from which the GP can be generated based on the different possibilities of matrix square roots of the covariance matrix.

Table of Contents

Prior mean and covariance kernel

```
Data.X=[]; Data.Y=[]; %To test "prior" model
Data.K=@K;
Xtest=-3:(1/30):5; %uniform grid to test stuff, dense enough...
LL=length(Xtest)
```

```
LL = 241
```

```
PredPrior=zeros(3,LL); %we'll predict confidence interval and mean
[Mean,CovMatrix]=predictGP(Xtest,Data);
```

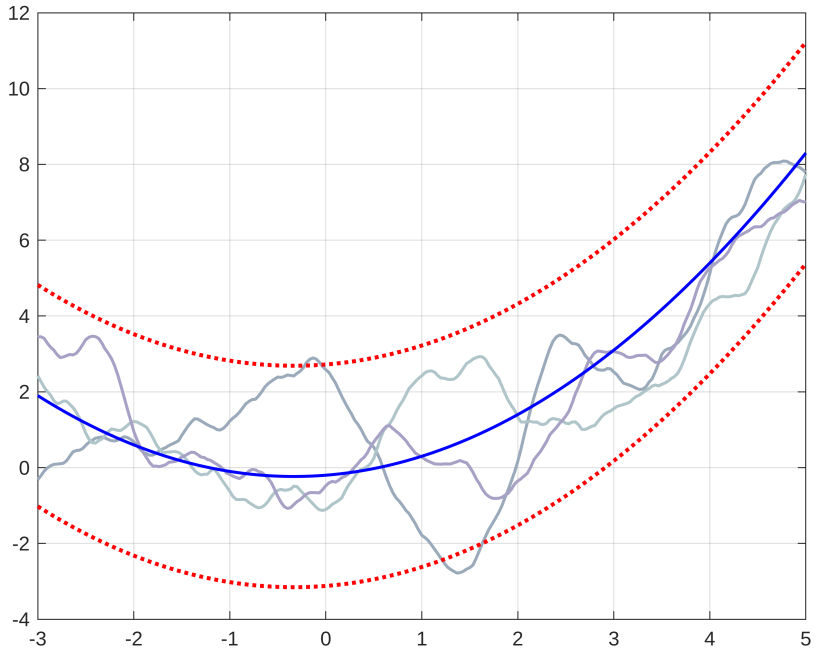
The prior is stationary (given how we defined it at the end of the file).

Confidence bounds and example samples

```
sg=sqrt(diag(CovMatrix)); %std. dev.
PredPrior=[Mean+1.96*sg Mean Mean-1.96*sg];
NTrials=3;
```

```
F1=mvnrnd(Mean,CovMatrix+1e-10*eye(LL),NTrials);

for i=1:NTrials
    if(NTrials<4)
        LW=1.5;
    else
        LW=1;
    end
    col=[.4 .55 .7]+rand()*[.3 0 0]+rand()*[0 .25 0]+rand()*[0 0 0.2];
    plot(Xtest,F1(i,:),Color=col,LineWidth=LW), hold on
end
plot(Xtest,PredPrior(:,2),'b',LineWidth=1.5), grid on
plot(Xtest,PredPrior(:,[1 3]),':r',LineWidth=2), hold off
```



Square root of covariance matrix (discretized at test points), interpretation

If we have samples of the GP, call them y_k, $1 \le k \le 241$ in our case, and the covariance matrix $K_{241\times241}$ can be factorised as $K = Q_{241\times241} \cdot Q^T_{241\times241}$, we'll say that Q is a "matrix" square root of K.

So, the GP (well, increments from its mean function) can be generated by "latent" standard normal random variables ξ, as $y = Q\xi_{241\times1}$, $\xi \sim N(0, I_{241})$ because, indeed, the covariance matrix is $E[yy^T] = E[Q\xi\xi^TQ^T] = Q \cdot E[\xi\xi^T] \cdot Q^T = Q \cdot I \cdot Q^T = K$. The internals of Matlab's `mvnrnd` work based on this fact.

*Other authors may say that B is a square root of a if $B \cdot B = A$, so be aware on which definition is used if you look up elsewhere. In here, we will use the $A = B \cdot B^T$ interpretation, only valid for symmetric matrices, of course.

Matrix square root is not unique, as any orthogonal matrix U, i.e., fulfilling $UU^T = I$ generates another square root QU because $K = (QU) \cdot (U^T Q^T)$.

Different matrix square roots have different interpretations of the "latent" random variables.

There will be several of them with interest in discrete-time stochastic processes: the "orthogonal" (diagonalisation, Karhunen-Loeve), the "triangular" (Cholesly, causal or anticausal representations) and the "symmetric" one, to be discussed in several videos in the collection.

Of course, there is theory generalising to the "discrete-time" going from $-\infty$ to $+\infty$ and to the full "continuous-time" case (spectral factor, all-pass functions, Fourier features) but it is more involved, out of the scope here...

Diagonalization of covariance matrix (discretized at test points), orthogonal square root

Further detail in previous video.

```
[V,D]=eig(CovMatrix); %eigenvalues + eigenvectors
[eevv,idx]=sort(diag(D),1,"descend");
V=V(:,idx); D=D(:,idx); %we sort eigenvalues and eigenvectors to match
minev=min(eevv); %eevv are positive... except for numerical tolerance
issues...
eevv=eevv+max(0,-minev); %now minimum is non-negative
```

Note: we carried out principal component analysis (meaning of eigenvalues and eigenvectors of covariance matrix)... If we have "many" points, we do approach the continuous GP eigenfunctions and eigenvalues, the Karhunen-Loeve ones:

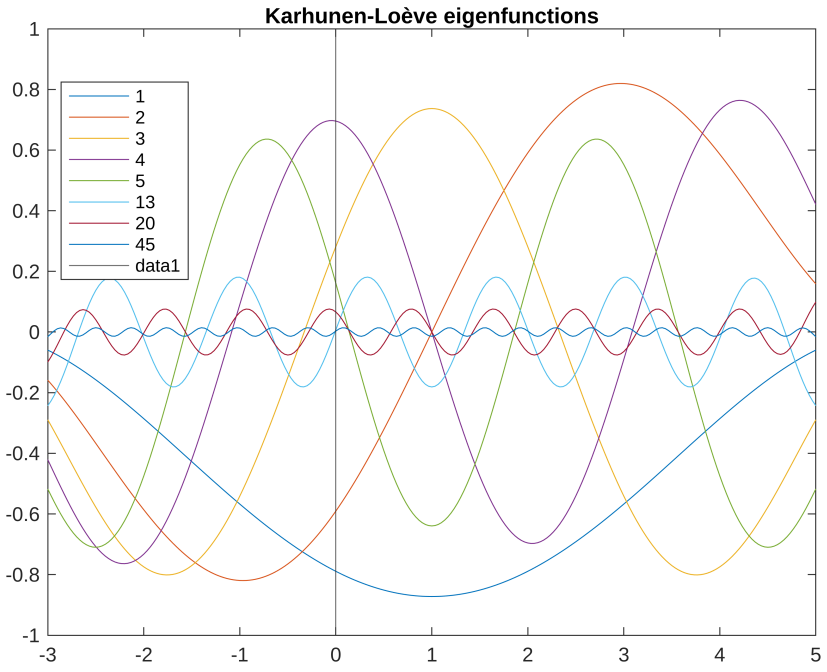
$Kv_i = \lambda_i v_i$ (matrix eigenvectors) get converted in the limit case to $\Rightarrow \quad \int_{-2}^{2} \kappa(x,s) \cdot \phi_i(s)\, ds = \lambda_i \cdot \phi_i(x)$

covariance kernel eigenfunctions

Let us have a look at the eigenvectors (i.e., eigenfunctions):

```
PrincCom=V*diag(sqrt(abs(eevv)));
plot(Xtest,PrincCom(:,[1 2 3 4 5 13 20 45])),
legend("1","2","3","4","5","13","20","45", Location="best")
xline(0)
```

```
title("Karhunen-Loève eigenfunctions")
```




```
Q=PrincCom; %matrix square root for animation
%return
```

*Execute animPCA.m script now.

Cholesky square root factorization (triangular) of covariance matrix, CONVOLUTION representation

Let us now discuss the "lower triangular" square root.

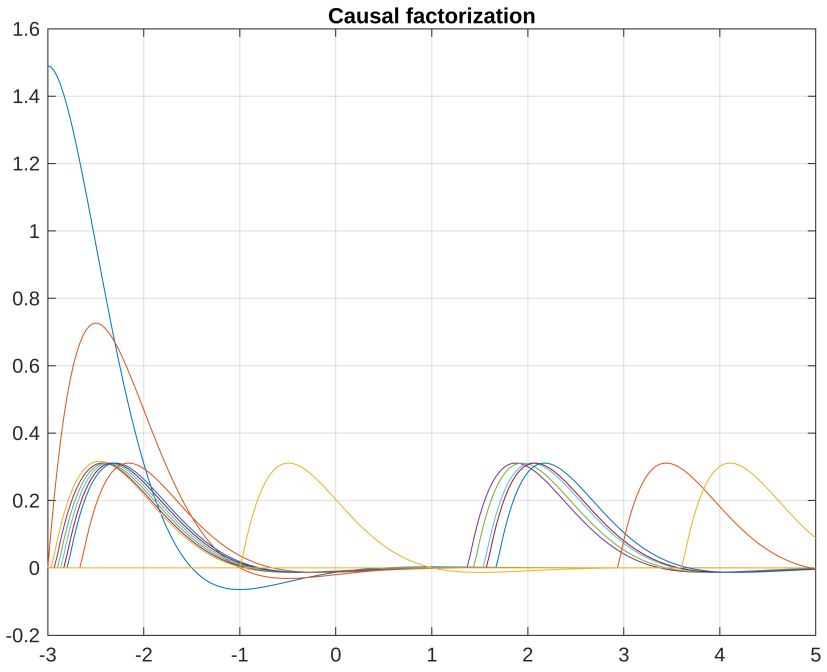
```
Q=chol(CovMatrix+1e-15*eye(LL),'lower');
Q
```

```
Q = 241×241
    1.4905         0         0         0         0         0         0         0 ···
    1.4865    0.1084         0         0         0         0         0         0
    1.4752    0.2090    0.0390         0         0         0         0         0
    1.4574    0.2985    0.0837    0.0379         0         0         0         0
    1.4337    0.3776    0.1236    0.0819    0.0378         0         0         0
    1.4049    0.4468    0.1588    0.1212    0.0818    0.0378         0         0
    1.3717    0.5068    0.1898    0.1559    0.1210    0.0818    0.0378         0
    1.3346    0.5582    0.2166    0.1863    0.1557    0.1210    0.0818    0.0378
    1.2943    0.6016    0.2397    0.2128    0.1861    0.1557    0.1210    0.0818
    1.2512    0.6375    0.2592    0.2355    0.2125    0.1861    0.1557    0.1210
     ⋮
```

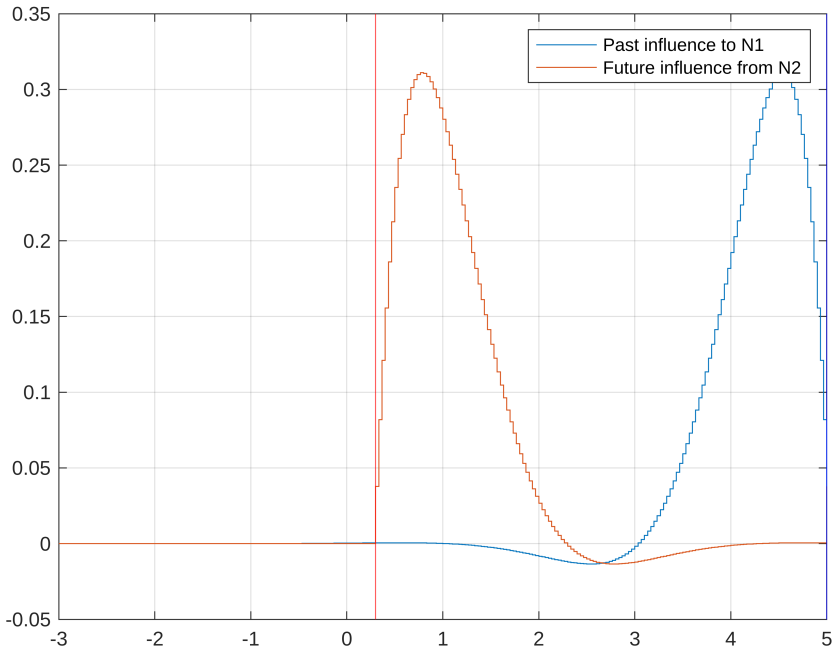
```
norm(Q*Q'-CovMatrix) %it is indeed a matrix square root
```

```
ans = 7.3271e-15
```

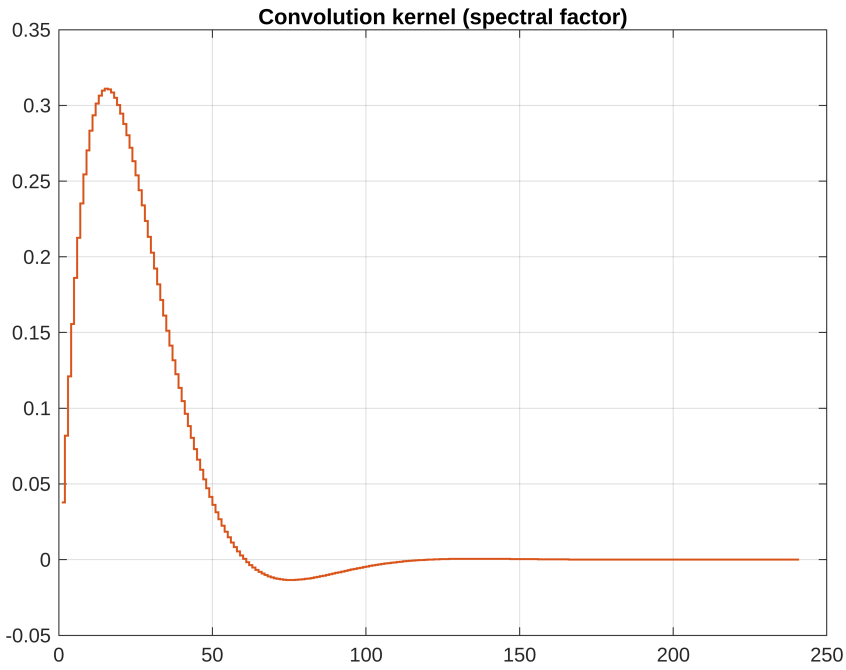
```
plot(Xtest,Q(:,[1 2 3 4 5 6 7 8 12 62 133 135 138 139 142 180 200])), grid on
title("Causal factorization")
```




```
N1=241;
stairs(Xtest,Q(N1,:)), grid on
hold on
N2=100;
stairs(Xtest,Q(:,N2)), hold off, xline(Xtest(N2),'r'), xline(Xtest(N1),'b')
legend("Past influence to N1", "Future influence from N2")
```




```
stairs(Q((N2):end,N2),LineWidth=1), hold on,
stairs(fliplr(Q(N1,:)),LineWidth=1), hold off, grid on, title("Convolution
kernel (spectral factor)")
```



Triangular structure: the GP can be expressed by the formula

$$y_k = \sum_{j \le k} q_{kj} \xi_j$$

because, if it is lower triangular $q_{ij} = 0$ for $j > k$, so no need to appear in the matrix product expression.

And, well, except for the first samples, columns are the same as rows.

If the nonzero elements of a column are denoted as the sequence $\{h_0, h_1, \dots\}$ then, when initial condition (or "finite window processing") effects have vanished (we need to close a column "far away from the left"), the row is the column "flipped", and the response is:

$$y_k = \sum_{j \le k} h_{k-j} \xi_j$$

which is called the "convolution formula... only truly valid at least in formal discussion if "past" extends indefinitely so sum is from $j = -\infty$ so indeed there is no effect of initial conditions or "finite window" signal processing. Of course, the idea can be extended to continuous-time convolution kernel $y(t) = \int_{-\infty}^{t} h(t-s)\xi(s)\, ds$, with formal difficulties as $\xi(s)$ is a white-noise with infinite variance, etc. (out of the scope of this material).

Execute **`animChol.m`** NOW.

```
```

Let's see the anticausal option: correlation does not imply causality and everything can be "repeated" if we consider right-to-left progression of random variables.

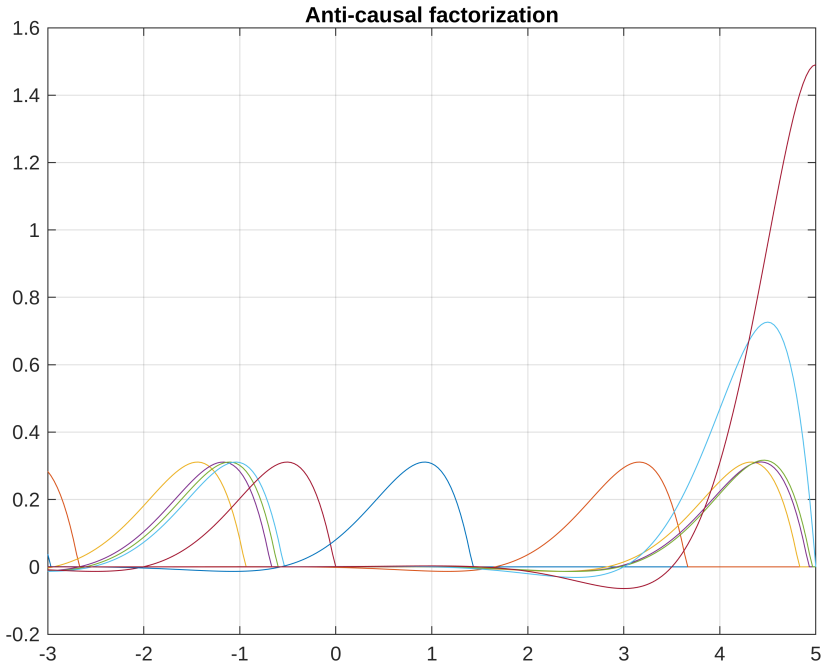
```
Qflip=fliplr(flipud(Q))
```

```
Qflip = 241×241
    0.0378    0.0818    0.1210    0.1557    0.1861    0.2125    0.2352    0.2544 ···
         0    0.0378    0.0818    0.1210    0.1557    0.1861    0.2125    0.2352
         0         0    0.0378    0.0818    0.1210    0.1557    0.1861    0.2125
         0         0         0    0.0378    0.0818    0.1210    0.1557    0.1861
         0         0         0         0    0.0378    0.0818    0.1210    0.1557
         0         0         0         0         0    0.0378    0.0818    0.1210
         0         0         0         0         0         0    0.0378    0.0818
         0         0         0         0         0         0         0    0.0378
         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0
    ⋮
```

```
norm(Qflip*Qflip'-CovMatrix)
```

```
ans = 2.2911e-14
```

```
plot(Xtest,Qflip(:,[1 10 62 70 72 74 90 133 200 235 238 239 240 241])), grid
on
title("Anti-causal factorization")
```




```
Q=Qflip;% for animation
```

```

```

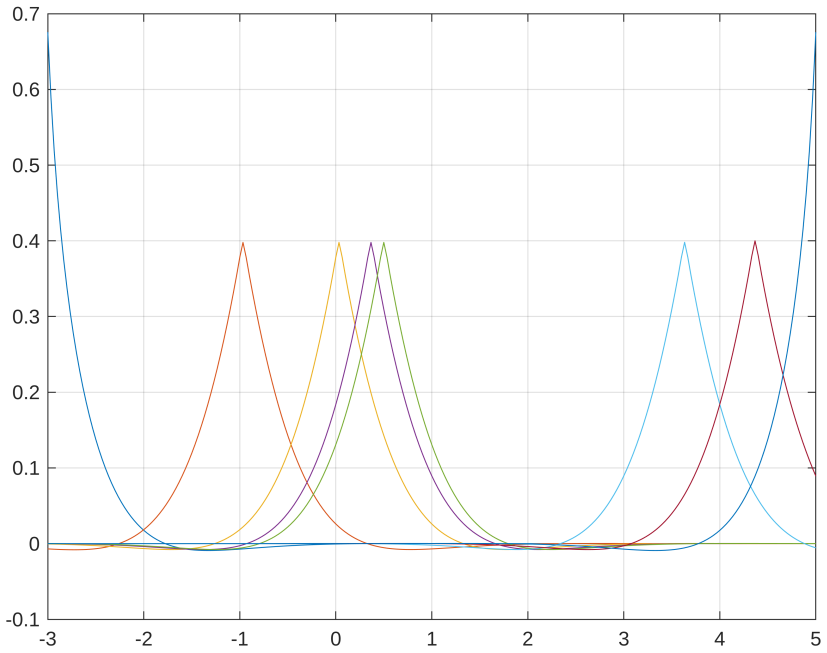
Symmetric square root of covariance matrix

Note that in this case $Q = Q^T = V \cdot \sqrt{D} \cdot V^T$, so this is a "true" square root as $K = Q^2$, no transpose involved.

```
figure()
Q=sqrtm(CovMatrix+1e-13*eye(LL));
norm(Q*Q'-CovMatrix)
```

```
ans = 2.7156e-13
```

```
plot(Xtest,Q(:,[1 62 92 102 106 200 222 241])), grid on
```



When finite window effects have vanished, this would be a "bilateral" convolution kernel that would end up in discrete time as $y(k) = \sum_{s=-\infty}^{+\infty} h(k-s)\xi(s)$, and in continuous time as $y(t) = \int_{-\infty}^{+\infty} h(t-s)\xi(s)\,ds$.

```
function [mu,Var]=predictGP(x1,Data)
arguments
    x1
    Data
end
priormean=@(x) 0.3*x.^2+0.2*x-.2;
lambda=1.5e-2^2; %measurement noise variance
K=@Data.K;
N=length(Data.X);
mu=priormean(x1)';
Var=K(x1,x1); %prior covariance among points in x1
if(N>0) %don't do this if no data available
    K1X=K(x1,Data.X);
    Gain=K1X/(K(Data.X,Data.X)+lambda*eye(N));
    mu=mu+Gain*(Data.Y-priormean(Data.X))';
    Var=Var-Gain*K1X';
    Var=0.5*(Var+Var');
end
end

function km=K(x1,x2)
```

```
M=0.6; sg=0.9;%hyperparameters stddev x and y
%kernel=@(x1,x2) M^2*exp(-norm(x2-x1)^2/2/sg^2); %stationary, depends only
on DIFFERENCE
%kernel=@(x1,x2) M^2*exp(-norm(x2-x1)/sg); %1sr-order exponential
kernel=@(x1,x2) pi*sin(pi/4 + (2^(1/2)*abs(x2-x1))/sg)*exp(-(2^(1/2)*abs(x2-
x1))/sg); %butterworth ord2
N1=length(x1); N2=length(x2); %1D case
km=zeros(N1,N2);
for i=1:N1
    for j=1:N2
        km(i,j)=kernel(x1(:,i),x2(:,j));
    end
end
end
```

```
Varacum=zeros(LL,1);
figure(100)
clf
figure(101)
clf
stddev=norm(Q(1,:));%if stationary
Amplitudes=randn(LL,1);%to build↙
realization by components
shuffle=randperm(LL); %for↙
bilateral simulation
%shuffle=LL:(-1):1;%ANTICAUSAL
%shuffle=1:LL;%not shuffled, for↙
causal or PCA (K-L).
vertflag=true;
for k=1:LL
    i=shuffle(k);
    figure(100)
    Varacum=Varacum+Q(:,i).^2;
    if(k>1)
        plot(Xtest,Q(:,shuffle(1:↙
(k-1))),LineWidth=1)
    end
    hold on
```

```
    plot(Xtest,Q(:,i),LineWidth=3.↙
5), grid on
    if(k>1)
        plot(Xtest,sqrt↙
(Varacum),':k',LineWIdth=2)
        plot(Xtest,-sqrt↙
(Varacum),':k',LineWIdth=2)
    end
    if vertflag
        xline(Xtest(i),'g',↙
LineWidth=2)
    end
    hold off
    title("Component functions↙
(features), scaled by standard↙
dev")
    ylim([-stddev stddev]*1.8)
    fontsize(16,"points") %This↙
works from version 2023a or↙
posterior
    figure(101)
    plot(Xtest,Q(:,i)/max(Q(:,↙
i))), grid on
```

```
    text(-1.9,.25,["i=" num2str(i)↙
num2str(sqrt(eevv(i)))])
    if vertflag
        xline(Xtest(i),'g',↙
LineWidth=2)
    end
    ylim([-1 1])
    title("Component functions↙
(features), scaled to unit↙
amplitude")
    fontsize(16,"points") %This↙
works from version 2023a or↙
posterior
    figure(102)
    for j=1:k %superimpose↙
previous approximations with less↙
components
        RealizTst=Q(:,shuffle(1:↙
j))*Amplitudes(shuffle(1:j));
        if j<k
            plot(Xtest,RealizTst,↙
LineWidth=1,Color=[.9 .95 .7]*((k-↙
j*0.3)/k)^2)
```

```
        else
            plot(Xtest,Q(:,i)↙
*Amplitudes(i),LineWidth=3,Color=↙
[.8 .2 .4])
            plot(Xtest,RealizTst,↙
LineWidth=5+0.04*sqrt(i),Color=[.1↙
.2 .4])
        end
        grid on
        hold on
    end
    ylim([-stddev stddev]*2.65)
     if vertflag
        xline(Xtest(i),'g',↙
LineWidth=2)
    end
    title("Building a realization↙
component by component")
    hold off
    ylabel("Increment with respect↙
to mean")
    fontsize(16,"points") %This↙
works from version 2023a or↙
```

```
posterior
    pause
end
```