Generación de muestras de un proceso estocástico gaussiano de función de covarianza conocida

El presente código funcionó con Matlab **R2020b**

Objetivo: comprender la forma y significado de las realizaciones de un proceso estocástico a partir de su función de covarianza.

Tabla de Contenidos

Variable de entrada

Nuestro objetivo será "simular" un proceso gaussiano estacionario de covarianza conocida $\kappa(x_1, x_2)$.

Usaremos muestras de ese proceso cada 0.1 segundos (los procesos estocásticos en tiempo contínuos son imposibles de "simular" numéricamente). Llamaremos X al eje de tiempos (o de lo que sea, los procesos gaussianos son "funciones aleatorias" de la variable X).

```
Tf=36;%Tiempo final.
X=(0:0.1:Tf)'; N=size(X,1); %puntos de tiempo para simulación
```

Si la función de covarianza es conocida, todo se reducirá a la generación de muestras de una distribución normal N-dimensional.

Kernel (función de covarianza)

El parámetro de escala (constante de tiempo) será:

```
tau=1.5; %constante de tiempo
syms w real %frecuencia
```

Probaremos dos casos.

Caso 1. Filtro de primer orden (estacionario) $\frac{\sqrt{2\tau}}{\tau s+1}$. Su densidad espectral de potencia es:

```
G1=@(s) sqrt(2*tau)/(tau*s+1);
PSD=@(G) simplify(G(1j*w)*G(1j*w)'); %G * G transp.conjugada
PSD1=PSD(G1)
```

```
PSD1 =
```

$$\frac{12}{9\,w^2+4}$$

Y la autocorrelación es la transformada inversa de Fourier de la misma:

```
k_r_simb1=ifourier(PSD1)
```

```
k_r_simb1 =
```

$$e^{-\frac{2\,|x|}{3}}$$

Ese filtro origina el Kernel de covarianza exponencial.

Pasemos de simbólico a numérico y generemos la función de covarianza basada en la distancia entre muestras:

```
k_r1=matlabFunction(k_r_simb1);
kappa_1=@(x1,x2)  k_r1(norm(x2-x1));
```

Caso 2. Filtro de segundo orden (estacionario) $\frac{\sqrt{4\tau}}{(\tau s+1)^2}$.

La densidad espectral de potencia es:

```
G2=@(s) sqrt(4*tau)/(tau*s+1)^2;
PSD2=PSD(G2)
```

```
PSD2 =
```

$$\frac{96}{(9\,w^2+4)^2}$$

Y la autocorrelación es la transformada inversa de Fourier de la misma:

```
k_r_simb2=ifourier(PSD2)
```

```
k_r_simb2 =
```

$$\frac{2\,e^{-\frac{2\,|x|}{3}}\left(|x|+\frac{3}{2}\right)}{3}$$

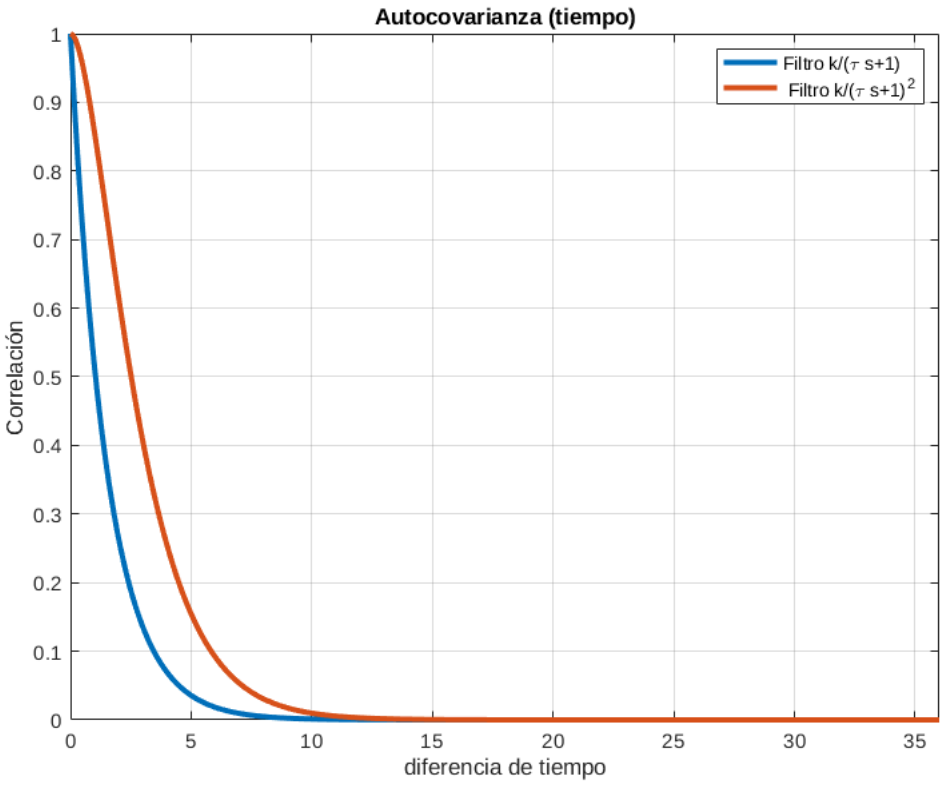
Pasando a numérico y generando la función de covarianza a partir de la distancia, tenemos:

```
k_r2=matlabFunction(k_r_simb2);
kappa_2=@(x1,x2) k_r2(norm(x2-x1));
```

Representemos gráficamente, la covarianza entre muestras en distintos instantes (estacionario: sólo dependen de la "diferencia de tiempo").

```
autocov=@(kappa) KernelMatr(0,X,kappa);
plot(X,[autocov(kappa_1);autocov(kappa_2)],'LineWidth',3),
```

```
grid on, xlabel('diferencia de tiempo'),ylabel('Correlación'),xlim([0,Tf])
legend('Filtro k/(\tau s+1)',' Filtro k/(\tau s+1)^2')
title('Autocovarianza (tiempo)')
```



Simulación del proceso Gaussiano

Diagonalización de la matriz de covarianza (comp. principales)

```
kappa=kappa_2;
```

La matriz de VC de las muestras es num.muestras*num.muestras:

```
K=KernelMatr(X,X,kappa);
size(K)
```

```
ans = 1×2
   361   361
```

La vamos a factorizar mediante la diagonalización: $Q = V \cdot D^{1/2}$, $K = QQ^T$.

```
tic
[V, D]=eig(K);
desvtip=sqrt(D+1e-13);%tolerancias en valores propios...
Q=V*desvtip;
toc
```

```
Elapsed time is 0.013342 seconds.
```

```
norm(K-Q*Q')
```

```
ans = 1.4174e-04
```

```
desvtip(353:360,353:360)
```

```
ans = 8×8
    3.5646    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    4.0323    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    4.5546    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    5.1244    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    5.7244    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    6.3243    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    6.8800    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    7.3370
```

En efecto, si $y = Q\phi$, siendo ϕ un vector de variables ruido blanco (randn), de varianza 1, esto es $E[\phi\phi^T] = I$, entonces $E[yy^T] = E[Q\phi \cdot \phi^T Q^T] = Q \cdot E[\phi\phi^T] \cdot Q^T = Q \cdot I \cdot Q^T = K$

Nota: La descomposición de Choleski también puede factorizar $K = QQ^T$ (la factorización no es única), y es más eficiente computacionalmente.

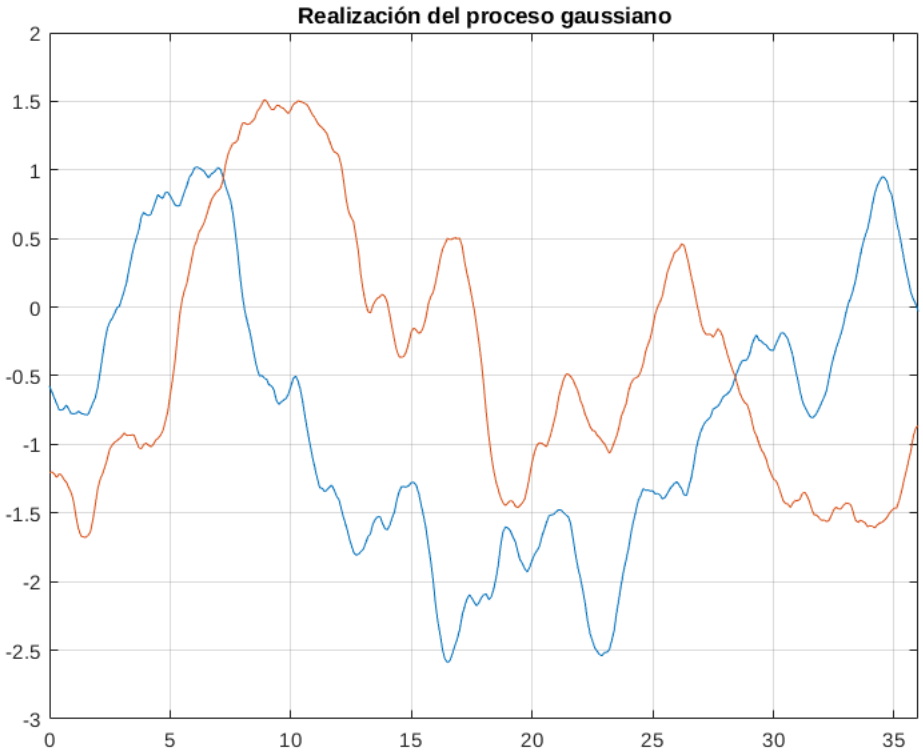
```
tic
Q=chol(K)';
toc
```

```
Elapsed time is 0.003698 seconds.
```

Realizaciones

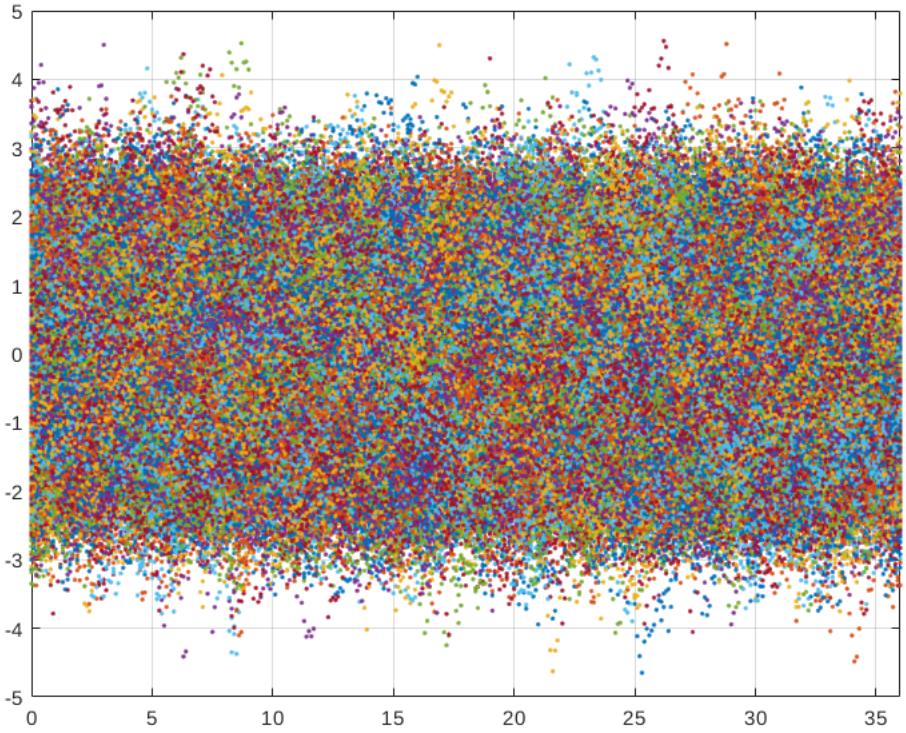
Vamos a generar realizaciones del proceso

```
numerorealizaciones=2;
valores=Q*randn(N,numerorealizaciones);
plot(X,valores), grid on, xlim([0,Tf])
title('Realización del proceso gaussiano')
```



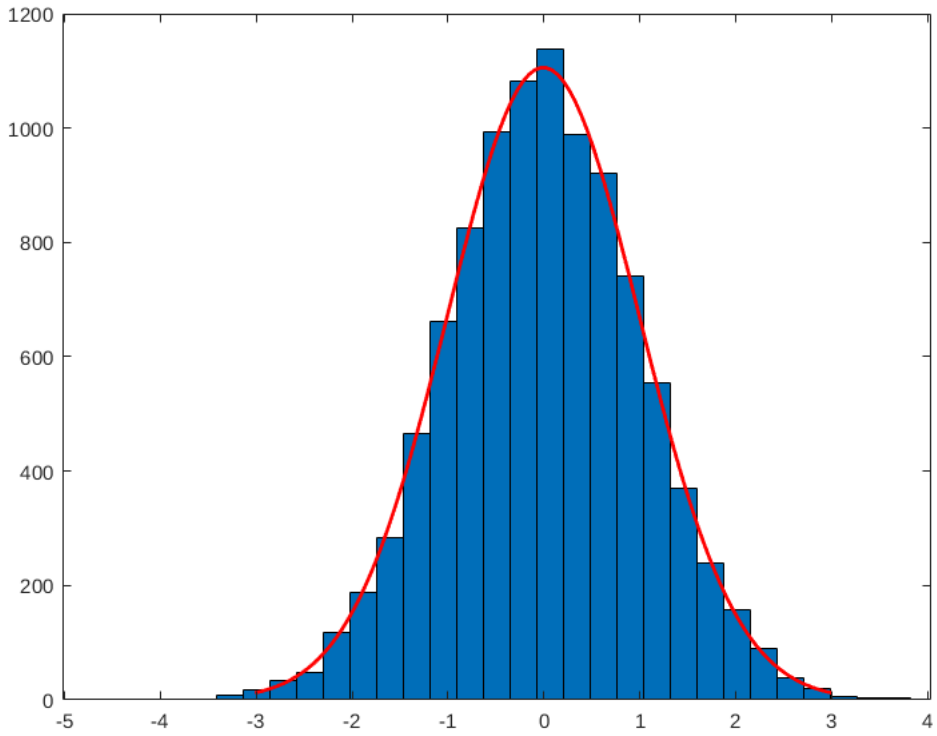
Nota: las series temporales se pueden simular como procesos "en representación interna" con entradas de ruido... No objeto de este material.

```
valores=V*desvtip*randn(N,10000);
plot(X,valores(:,1:3000),'.'), grid on, xlim([0,Tf])
```



La distribución marginal para cada instante es normal, de varianza unidad

```
histfit(valores(140,:),31,'normal')
```




```
K(140,140)
```

```
ans = 1
```

```
mean(valores(140,:)')
```

```
ans = -0.0067
```

```
cov(valores(140,:)')
```

```
ans = 1.0056
```

Las covarianzas también se ajustan (excepto aprox. muestras finitas) a la matriz especificada

```
cov(valores([1 15 30 45],:)')
```

```
ans = 4×4
    1.0032    0.3924    0.1456    0.0561
    0.3924    1.0162    0.3763    0.1402
    0.1456    0.3763    0.9957    0.3705
    0.0561    0.1402    0.3705    0.9987
```

```
K([1 15 30 45],[1 15 30 45])
```

```
ans = 4×4
    1.0000    0.3932    0.1447    0.0532
    0.3932    1.0000    0.3679    0.1353
    0.1447    0.3679    1.0000    0.3679
    0.0532    0.1353    0.3679    1.0000
```

Conclusiones

Distintas funciones de covarianza vienen generadas por distintos filtros de los componentes frecuenciales. Una vez se ha generado la matriz de varianzas-covarianzas K de un conjunto finito de muestras, la factorización $K = QQ^T$ permite obtener realizaciones del proceso (equivalentes a la "simulación" del proceso ante entradas ruido blanco, no objetivo de este ejemplo).

```
function ke=KernelMatr(x,X,kappa)
nx=size(x,1); nX=size(X,1);
ke=zeros(nx,nX);
for i=1:nx
    for j=1:nX
        ke(i,j)=kappa(x(i,:),X(j,:));
    end
end
end
```