Hoja de ejercicios 2

Corredor Vegas, Guillermo
Martín Gallego, Eugenio
Montoya Manosalvas, Michell

May 16, 2018

1- Cálculo de la sensibilidad delta

(i) Determinar los órdenes de aproximación

Utilizando el desarrollo de Taylor descrito, desarrollado hasta el orden 3,

$$f(x_{\pm}) \approx f(x_0 \pm dx) \approx f(x_0) + f'(x_0)(\pm dx) + \frac{f''(x_0)}{2!}(\pm dx)^2 + \frac{f'''(x_0)}{3!}(\pm dx)^3 + \mathcal{O}(dx)^4$$

Despejando de la expresión con incremento positivo en x,

$$\frac{f(x_0 + dx) - f(x_0)}{dx} \approx f'(x_0) + \mathcal{O}(dx)$$

Por tanto el orden de aproximación para el primer método es $n_i = 1$.
De la diferencia entre el desarrollo de Taylor con incremento positivo e incremento negativo:

$$f(x_0 + dx) \approx f(x_0) + f'(x_0)(+dx) + \frac{f''(x_0)}{2!}f'(x_0)(+dx)^2 + \mathcal{O}(dx)^3$$

$$f(x_0 - dx) \approx f(x_0) + f'(x_0)(-dx) + \frac{f''(x_0)}{2!}f'(x_0)(-dx)^2 + \mathcal{O}(dx)^3$$

$$\frac{f(x_0 + dx) - f(x_0 - dx)}{2\,dx} \approx f'(x_0) + \mathcal{O}(dx)^2$$

Por lo tanto el orden de aproximación del segundo método es $n_{ii} = 2$. Numéricamente la aproximación preferible es la que tiene un orden de aproximación mayor, el segundo método.

(ii) Escribir la fórmula suponiendo que $dx = x_0 h$

$$f'(x_0) = \frac{f(x_0(1+h)) - f(x_0)}{x_0 h} + \mathcal{O}(h)$$

$$f'(x_0) = \frac{f(x_0(1+h)) - f(x_0(1-h))}{2x_0 h} + \mathcal{O}(h^2)$$

(iii) Utilizando la función que calcula el precio de una call europea, diseña una Matlab para el cálculo de la delta de una call europea utilizando diferencias divididas.

```
function delta = deltaCallEU(h,S0,K,r,T,sigma)
%% deltaCallEU:  delta of an european call (finite difference method)
%
%% SYNTAX:
%         delta = deltaCallEU_MC(h,S0,K,r,T,sigma)
%% INPUT:
%          h : Approximation error
%         S0 : Initial value of the underlying asset
%          K : Strike
%          r : Risk-free interest rate
%          T : Time to expiry
%      sigma : Volatility
%% OUTPUT:
%      delta : Value of the option delta trough finite differences
%% EJEMPLO 1:
%         S0 = 100; K = 90; r = 0.03; T = 2; sigma = 0.4;
%         h = 1e-5;
%          delta = deltaCallEU(h,S0,K,r,T,sigma)
%          blsdelta(S0,K,r,T,sigma)
%
```

```
%% EJEMPLO 2:
%       S0 = 100; K = 90; r = 0.03; T = 2; sigma = 0.4;
%       for exponente=1:16
%            h = 10^-(exponente);
%            gamma(exponente) = deltaCallEU(h,S0,K,r,T,sigma);
%            contador(exponente) = -exponente;
%       end
%       real_value = blsdelta(S0,K,r,T,sigma);
%       array2table(cat(1, contador, gamma, abs((real_value-gamma)/
    real_value))','VariableNames',{'Exponente','Valor','rel_error'})
%
%% Aplicando el metodo de diferencias finitas
fsup = priceEuropeanCall(S0*(1+h),K,r,T,sigma);
finf = priceEuropeanCall(S0*(1-h),K,r,T,sigma);
delta = (fsup - finf) ./ (2.0*S0*h);
```

Utlizando el código del ejemplo 2 del archivo *deltaCallEU* se observa que el valor de h con menor error relativo es $h = 10^{-3}$

Exponente	Valor	rel_error
-1	0.7154	0.0027977
-2	0.71739	2.8016e-05
-3	0.71741	1.1951e-06
-4	0.71734	9.4317e-05
-5	0.71743	3.2768e-05
-6	0.71743	3.2768e-05
-7	0.71743	3.2768e-05
-8	0.71743	3.2776e-05
-9	0.71743	3.2773e-05
-10	0.71743	3.3788e-05
-11	0.71744	3.8741e-05
-12	0.71749	0.00011055
-13	0.71712	0.00040943
-14	0.7141	0.0046188
-15	0.74607	0.039951
-16	0.53291	0.25718

(iv) Diseña una función Matlab para estimar la delta por MC utilizando *Common Random Numbers*

```
function [delta_MC,err_MC] = deltaCallEU_MC(M,S0,K,r,T,sigma)
%% deltaCallEU_MC:  delta of an european call through MC method
%
%% SYNTAX:
%       [delta_MC,err_MC] = deltaCallEU_MC(M,S0,K,r,T,sigma)
%% INPUT:
%       M : Number of simulations
%       S0 : Initial value of the underlying asset
%       K : Strike
%       r : Risk-free interest rate
%       T : Time to expiry
%    sigma : Volatility
%% OUTPUT:
% delta_MC : MC estimate of the price of the option in the Black-Scholes
    model
```

```
%    err_MC : MC estimate error (standard deviation)
%% EJEMPLO 1:
%         S0 = 100; K = 90; r = 0.03; T = 2; sigma = 0.4;
%         M = 1e6;
%         [delta_MC,err_MC] = deltaCallEU_MC(M,S0,K,r,T,sigma)
%         h = 1e-5;
%         deltaCallEU(h,S0,K,r,T,sigma)
%         blsdelta(S0,K,r,T,sigma)
%%
%Simulate M random trajectories
X = randn(M,1);
h=1e-6;

%Simulate M trajectories in one step
ST_plus = S0*(1+h).*exp((r-sigma^2/2)*T+sigma.*sqrt(T).*X);
ST_minus = S0*(1-h).*exp((r-sigma^2/2)*T+sigma.*sqrt(T).*X);

%Define payoff for finite difference method
payoff_plus =max(ST_plus-K,0);
payoff_minus = max(ST_minus-K,0);
delta = (payoff_plus-payoff_minus) ./(2*S0*h);

%Compute delta value and error
delta_MC = exp(-r*T)*mean(delta);
err_MC = exp(-r*T)*std(delta)/sqrt(M);
```

Lo más importante, y la base de este método, es tener en cuenta que los números aleatorios que se tienen que utilizar como base para el cálculo del método tienen que ser los mismos a la hora de calcular el precio del activo en T y los pagos (de ahí common), de otra forma se utilizarían dos bases distintas en el cálculo y el error del método de MonteCarlo sería mayor que la precisión de h (el error de aproximación) y obtendríamos valores complemtamente erróneos para la delta.
El valor obtenido para la delta por el método de MonteCarlo de *common random numbers* es

$$\delta = 0.7177 \pm 0.0008$$

---

2 - Cálculo de la sensibilidad gamma

(i,ii) Deriva la fórmula para aproximar la segunda derivada de $f(x)$ en $x = x_0$ utilizando diferencias divididas que involucren a $f(x_0), f(x_0 + dx), f(x_0 - dx)$

Partiendo del desarrollo de Taylor:

$$f(x_0 + dx) \approx f(x_0) + f'(x_0)(+dx) + \frac{f''(x_0)}{2!}(+dx)^2 + \mathcal{O}(dx)^3$$

$$f(x_0 - dx) \approx f(x_0) + f'(x_0)(-dx) + \frac{f''(x_0)}{2!}(-dx)^2 + \mathcal{O}(dx)^3$$

Reordenando ambas expresiones (hasta orden 2) se puede llegar a las expresiones en diferencias adelantadas y atrasadas:

$$\frac{f(x_0 + dx) - f(x_0)}{dx} \approx f'(x_0) + \frac{f''(x_0)}{2!}(+dx) + \frac{f'''(x_0)}{6}(dx)^2 + \mathcal{O}(dx)^3$$

$$\frac{f(x_0) - f(x_0 - dx)}{dx} \approx f'(x_0) + \frac{f''(x_0)}{2!}(-dx) + \frac{f'''(x_0)}{6}(-dx)^2 + \mathcal{O}(dx)^3$$

y restando ambas ecuaciones se elimina el término de orden 2

$$f''(x_0)dx \approx \frac{f(x_0+dx)-f(x_0)}{dx} - \frac{f(x_0)+f(x_0-dx)}{dx} + \mathcal{O}(dx)^3$$

Dividiendo entre dx el error final es de orden 2 o mayor:

$$f''(x_0) \approx \frac{f(x_0+dx)-2f(x_0)+f(x0-dx)}{dx^2} + \mathcal{O}(dx)^2$$

(iii) Utilizando la función que calcula el precio de una call europea, diseña una Matlab para el cálculo de la delta de una call europea utilizando diferencias divididas.

$$f''(x_0) \approx \frac{f(x_0(1+h))-2f(x_0)+f(x_0(1-h))}{(x_0 \cdot h)^2}$$

(iv) Diseña una función Matlab para estimar la delta por MC utilizando *Common Random Numbers*

```
function gamma = gammaCallEU(h,S0,K,r,T,sigma)
%% gammaCallEU: gamma de una call europea
%
%% SYNTAX:
%         gamma = gammaCallEU(h,S0,K,r,T,sigma)
%
%% INPUT:
%          h : Approximation error
%         S0 : Initial value of the underlying asset
%          K : Strike
%          r : Risk-free interest rate
%          T : Time to expiry
%      sigma : Volatility
%
%% OUTPUT:
%      gamma : Value of the option gamma sensitivity
%
%% EXAMPLE 1:
% S0 = 100; K = 90; r = 0.03; T = 2; sigma = 0.4;
% h = 1e-4
% gamma = gammaCallEU(h,S0,K,r,T,sigma)
% blsgamma(S0,K,r,T,sigma)
%
%% EXAMPLE 2:
% S0 = 100; K = 90; r = 0.03; T = 2; sigma = 0.4;
% for exponente=1:16
%       h = 10^-(exponente);
%       gamma(exponente) = gammaCallEU(h,S0,K,r,T,sigma);
%       contador(exponente) = -exponente;
% end
% real_value = blsgamma(S0,K,r,T,sigma);
% table(contador', gamma', abs(real_value-gamma)','VariableNames',{'
    Exponente','Valor','ABS_error'})
% %
payoff = @(ST) max(ST-K,0);

C = @(S0) priceEuropeanOption(S0,r,T,sigma,payoff);

```

```
38 gamma = (C(S0*(1+h))-2*C(S0)+C(S0*(1-h)))/(S0*h)^2;
```

Utilizando el código expuesto en el ejemplo 2 de este código se puede observar que el valor de h que da menor error $h = 1e-3$. Esto se debe a que estamos llamando en *priceEuropeanOption*, dentro de la cual el método para calcular la integral se ha definido que tenga una tolerancia de $1e-6$, por lo que ya arrastra un error interior además del inherente a las operaciones realizadas en la función principal.

```
    Exponente       Valor       ABS_error
---------    -----------    ----------

  -1         0.0059919    1.4618e-05
  -2         0.0059774    1.4886e-07
  -3         0.0059772    2.8865e-09
  -4         0.0060112    3.3949e-05
  -5         0.0086357     0.0026585
  -6         0.0029416     0.0030356
  -7        0.00010658     0.0058707
  -8        -0.0071054      0.013083
  -9            1.0658        1.0598
 -10                 0     0.0059772
 -11           -3552.7        3552.7
 -12         1.4211e+06    1.4211e+06
 -13        -1.0658e+08    1.0658e+08
 -14        -3.5527e+09    3.5527e+09
 -15         1.4211e+12    1.4211e+12
 -16        -1.0658e+14    1.0658e+14
```

(v) Diseña una función Matlab para estimar la gamma por MC utilizando Common Random Numbers

```
1  function [gamma, err] = gammaCallEU_MC(M,S0,K,r,T,sigma)
2  % gammaCallEU_MC: gamma de una call europea mediante MC
3  %
4  % EJEMPLO 1:
5  % S0 = 100; K = 90; r = 0.03; T = 2; sigma = 0.4;
6  % M = 1e6;
7  % [gamma, err] = gammaCallEU_MC(M,S0,K,r,T,sigma)
8  % h = 1e-5;
9  % gammaCallEU(h,S0,K,r,T,sigma)
10 % blsgamma(S0,K,r,T,sigma)
11 %
12 X = randn(M,1); h=1e-2;
13 %
14 ST_plus = S0*(1+h).*exp((r-sigma^2/2)*T+sigma.*sqrt(T).*X);
15 ST = S0.*exp((r-sigma^2/2)*T+sigma.*sqrt(T).*X);
16 ST_minus = S0*(1-h).*exp((r-sigma^2/2)*T+sigma.*sqrt(T).*X);
17 %
18 payoff_plus = max(ST_plus-K,0);
19 payoff = max(ST-K,0);
20 payoff_minus = max(ST_minus-K,0);
21 %
22 derivative = (payoff_plus - 2*payoff + payoff_plus)/(S0*h)^2;
23
24 gamma = exp(-r*T)*mean(derivative);
25 err = exp(-r*T)*std(derivative)/sqrt(M);
```

3 - Integración numérica

(i) ¿Cuál de las aproximaciones es preferible numéricamente? Justifica la respuesta

La aproximación preferible es aquella cuyo error sea de orden menor. Desarrollando por serie de Taylor la función a integrar se puede calcular el error de las primeras aproximaciones:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) + \mathcal{O}^2$$

Integrando esta expresión entre x_0 y $x_0 + h$:

$$\int_{x_0}^{x_0+h} f(x_0) + f'(x_0)(x - x_0) = f(x_0)h + f'(x_0)(\frac{x^2}{2} - x \cdot x_0)\Big|_{x_0}^{x_0+h}$$

Operando aparte el último término del producto:

$$(\frac{x^2}{2} - x \cdot x_0)\Big|_{x_0}^{x_0+h} = \frac{(x_0+h)^2}{2} - x_0(x_0+h) - \frac{x_0^2}{2} + x_0^2 = \frac{(x_0^2 + hx_0 + h^2)}{2} - x_0(x_0+h) + \frac{x^2}{2} = \frac{h^2}{2}$$

Para la primera aproximación, desarrollando alrededor de x_0:

$$\int_{x_0}^{x_0+h} f(x_0) + \mathcal{O}(h)dx \approx f(x_0)(h) + \mathcal{O}(h) \cdot h = f(x_0)h + \mathcal{O}(h^2)$$

Para la segunda, de forma similar,

$$\int_{x_0}^{x_0+h} f(x_0+h) + \mathcal{O}(h)dx \approx f(x_0+h)(h) + \mathcal{O}(h) \cdot h = f(x_0+h)h + \mathcal{O}(h^2)$$

En el tercero introduzco la aproximación por diferencias adelantadas de $f'(x_0)$

$$\int_{x_0}^{x_0+h} f(x_0)+f'(x_0)(x-x_0)+\mathcal{O}(h^2)dx = f(x_0)(h)+f'(x_0)\frac{h^2}{2}+\mathcal{O}(h^3) = f(x_0)h+\left(\frac{f(x_0+h)-f(x_0)}{h}+\mathcal{O}(h)\right)\frac{h^2}{2}$$

Por lo tanto, para la tercera aproximación:

$$\int_{x_0}^{x_0+h} f(x_0) + f'(x_0)(x - x_0) + \mathcal{O}(h^2)dx = \frac{f(x_0) + f(x_0+h)}{2}h + \mathcal{O}(h^3)$$

En la cuarta aproximación se realiza el desarollo de Taylor a partir del punto medio entre x_0 y $x_0 + h$

$$\int_{x_0}^{x_0+h} f(x_0 + \frac{h}{2}) + f'(x_0 + \frac{h}{2})(x - x_0 - \frac{h}{2}) + \mathcal{O}(h^2)dx = f(x_0 + \frac{h}{2}) + \mathcal{O}(h^3)$$

$$\left(\frac{x^2}{2} - x\left(x_0 + \frac{h}{2}\right)\right)\Big|_{x_0}^{x_0+h} = 0$$

Como el orden 2 se anula, el orden de aproximación de esta integral es 3. Además, si se calculara el valor de estos órdenes, se vería que este último es la mitad que el tercero, por lo tanto este sería el favorito.

(ii) Se ha propuesto el siguiente algoritmo para calcular una integral definida ¿Qué ocurre si N es demasiado pequeño? ¿Qué ocurre si N es demasiado grande?

En el caso de que N sea demasiado pequeño, como tenemos que el error de aproximación se define en este método como $h = (b - a)N^{-1}$ entonces el valor de h en cuanto la diferencia entre los límites no sea despreciable respecto a sus imágenes será bastante grande, y la aproximación realizada por este método de la integral definida no será precisa, ya que los segmentos de la base tomados serán demasiado grandes (matemáticamente infinitesimales, numéricamente lo más pequeños posibles). Este error es conocido

como error de truncamiento, y aparece cuando se intenta analizar numéricamente una operación que analíticamente tiene infinitas interacciones.
En el caso de que N sea demasiado grande estaremos dividiendo la base en secciones demasiado pequeñas, y habrá un momento en que aparezca error de redondeo en los cálculos, ya que exigirán un nivel de aproximación mayor que el que el software puede permitir debido a la forma en que almacena la información.

(iii) Proponed un algoritmo para determinar N, suponiendo que necesito conocer el valor de I con error absoluto máximo $TOL\ ABS$.

```
N=10; err = 10 (>TOL_ABS)
while err>TOL_ABS
N = 10*N;
valor_exacto = quadl(f,a,b) ;% tolerancia por defecto
trapecio_compuesto = (implementacion metodo)
err = abs(valor_exacto - trapecio_compuesto)
end
```

---

4 - Call digital

$$payoff(S(t_0+T)) = \begin{cases} A\ si\ S(t_0+T) > K \\ 0\ si\ S(t_0+T) \leq K \end{cases} \tag{1}$$

Implementa el método de la bisección para calcular la volatilidad implícita del producto dados S0,r,T,K y el precio de la call.

```
function [x_med, err] = BiseccionZero(f,a,b)
%% SYNTAX:
%        [x_med, err] = BiseccionZero(f,a,b)
%
%% ENTRADA:
%            f: funcion
%        [a,b]: Intervalo que acota al cero
%
%% SALIDA:
%        x_med: estimacin del cero de la funcin
%          err: error de la estimacin
%
%% PROCESAMIENTO:
% Inicialmente, el error de estimacion es: err = (b-a)
% Repetir hasta convergencia
% 1. Calculamos el punto medio del intervalo: x_med = (a+b)/2.0;
% 2. Actualizamos el error de estimacion: err = err/2
% 3. Si f(a)*f(x_med) < 0, buscamos el cero en [a,x_med]
% 4. Si f(b)*f(x_med) < 0, buscamos el cero en [x_med,b]

%% EJEMPLO 1:
%        S0 = 100; K = 90; r = 0.1; T = 2; precio = 7.81; A = 10;
%        a = 0; b = 1;
%        payoff = @(ST) A .* (ST>K); %Opcion digital que paga A si ST>K
%        f = @(sigma)(priceEuropeanOption(S0,r,T,sigma,payoff)- precio);
%        [x_med, err] = BiseccionZero(f,a,b)
%        x_med_newton = fzero(f,0.5)
%%
TOL_ABS = 1e-6;          % error absoluto
```

```
TOL_REL = 1e-16;          % error relativo
TOL_f = 1e-10;            % valor minimo positivo que puede distinguir f
err= b-a;
x_med = (a+b)/2;
while err>TOL_ABS || abs(err/x_med) < TOL_REL || abs(f(x_med)) < TOL_f
    x_med = (a+b) / 2;
    if f(a).*f(x_med) < 0
        b = x_med;
    else a = x_med;
    end
    err = err / 2;
end
```

Para obtener una estimación de la volatilidad implícita de la call digital, se va buscar el cero de la función mediante el método de la bisección. Esta técnica se basa en el teorema del valor intermedio y parte del supuesto de que f(a) y f(b) tienen signos opuestos. Consiste en dividir a la mitad repetidamente los subintervalos de [a, b] y en cada paso, localizar la mitad que contiene a la solución. Evaluarla, si es igual a cero se obtiene la solución buscada, si no lo es se redefine el intervalo, como$[a, x_med]$ ó $[x_med, b]$ según se haya determinado en cuál de estos intervalos ocurre un cambio de signo. Con este nuevo intervalo se continúa sucesivamente encerrando la solución en un intervalo cada vez más pequeño, hasta alcanzar la precisión deseada, en nuestro caso, hasta que se supere uno de los tres posibles criterios de convergencia. Si comparamos nuestra solución obtenida con la proporcionada f_zero, vemos que ambas coinciden hasta el sexto decimal, que es cuando nuestra implementación supera la tolerancia absoluta.

La volatilidad implícita calculada por el método de la bisección, junto a su error, es:

```
x_med = 0.1220
err = 9.5367e-07
```

Comparado con la volatilidad implícita obtenida por la función implementada en Matlab *fzero*, ambas coinciden con una precisión de 6 cifras decimales (ya que la precisión que se le ha pedido al método a través de TOL_ABS es de 6 cifras decimales).

---

5 - Vega de una call digital

Hacer una gráfica que muestre cómo varía la vega de este producto (sensibilidad de este producto frente a variaciones en la volatilidad) para volatilidades entre 10% y 50%. Hacer un ajuste polinómico (utilizar la función polyfit de Matlab, con un grado a determinar) para aproximar la curva de dependencia con un error relativo menor de 0.01. El grado del polinomio que aproxima dicha curva debe ser lo más pequeño posible.

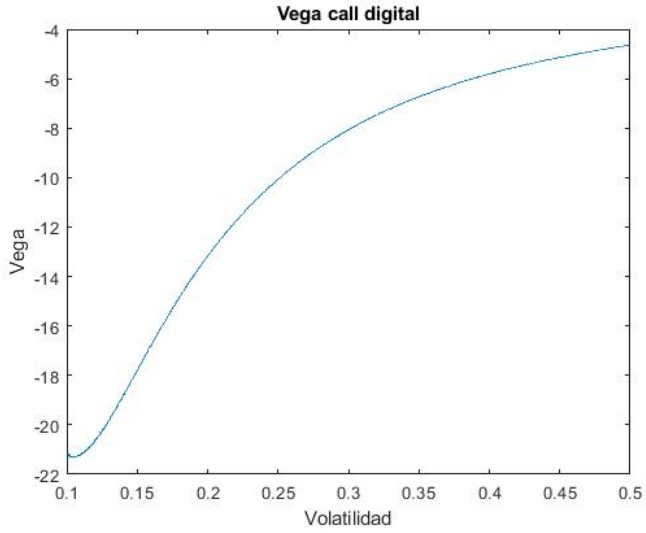
```
function vega = vegaCall(h,S0,K,r,T,sigma,payoff)
%% vegaCall: vega de una call (en funcin del payoff, metodo diferencias
    finitas)
%
%% SYNTAX:
%       vega = vegaCall(S0,K,r,T,sigma,payoff)
%
%% INPUT:
%        h : Approximation error
%       S0 : Initial value of the underlying asset
%        K : Strike
%        r : Risk-free interest rate
```

```
%          T : Time to expiry
%      sigma : Volatility
%    payoff : Handle to the function of ST that specifies the payoff
%
%% OUTPUT:
%      gamma : Value of the option gamma sensitivity
%
%% EXAMPLE 1:
% S0 = 100; K = 90; r = 0.05; T = 2; sigma = 0.1; A = 10; h = 1e-6;
% payoff = @(ST)(A.*(ST>K)); % payoff of a digital call option
% vega = vegaCall(h,S0,K,r,T,sigma,payoff);
%
% % EXAMPLE 2:
% S0 = 100; K = 90; r = 0.05; T = 2; A = 10; h = 1e-6;
% payoff = @(ST)(A.*(ST>K)); % payoff of a digital call option
% sigma_ini = 0.1; sigma_final = 0.5; n = 1000;
% for i=1:n
%      sigma = sigma_ini + (sigma_final-sigma_ini)*i/n;
%      yPlot(i) = vegaCall(h,S0,K,r,T,sigma,payoff);
% end
% xPlot=linspace(sigma_ini,sigma_final,n);
% plot(xPlot,yPlot);  %Plots vega for different volatilities
% title('Vega call digital'); xlabel('Volatilidad'); ylabel('Vega');
% orden = 0; err_rel = 1; %Initialize variables
% while err_rel>0.01   %Stopping condition
%      orden = orden+1;
%      polinomio = polyfit(xPlot,yPlot, orden); %fit polynomial of order
    orden
%      for i = 1:n
%          sigma = xPlot(i);
%          base = flip(sigma.^linspace(0,orden,orden+1));
%          est_value(i) = sum(polinomio.*base);
%          error(i) = abs((est_value(i)-yPlot(i))/yPlot(i));      %error
    vector
%      end
%      err_rel = max(error); %stores the greatest value of the error vector
% end
% orden
%% CODE
price = @(sigma)(priceEuropeanOption(S0,r,T,sigma,payoff));
vega = numericalDerivative(price,sigma,h);
```

La vega de una opción es la derivada de la función de valor de la opción respecto a su volatilidad, y mide la sensibilidad a la volatilidad.
El objetivo es realizar un gráfico donde se aprecie cómo varía la vega de una call digital para diferentes volatilidades. El programa implementando calcula la vega de una call utilizando diferencias finitas para obtener una aproximación de la derivada primera, llamando a las funciones previamente definidas *numericalDerivative* y *priceEuropeanOption*. Para realizar el gráfico se ha calculado la vega para 1000 puntos en el intervalo $\sigma \in [0.1, 0.5]$.

Además, se pide ajustar esta curva mediante un ajuste polinómico de orden n a través de la función *polyfit* de tal forma que el grado sea el menor posible. El código para esta parte se encuentra en el ejemplo 2. Primero se rellena un vector con las vegas y las volatilidades utilizadas para calcularlas. El objetivo es que el error relativo para todos los puntos entre el valor calculado de la vega y el aproximado por el polinomio sea menor que 0.01, por lo que se utiliza un bucle *while* que incluya esta condición. Si en alguno de los puntos calculados el error relativo es mayor que 0.01 entonces se calcula el siguiente ajuste polinómico de orden mayor, hasta que se cumpla la condición. El resultado es:

```
orden =  8
```



---

6 - Movimiento Browniano aritmético

```
function B = ArithmeticBrownian(t0,T,M,N,mu,sigma,B0)
%% ArithmeticBrownian: Simulates M arithmetic Brownian movements
%
%% SYNTAX:
%          [B] = ArithmeticBrownian(t0,T,M,N,mu,sigma,B0)
%
%% INPUT:
%         t0 : Initial simulation time
%          T : Final simulation time
%          M : Number of simulations
%          N : Numbers of step (from t0 to t0+T)
%         mu : Expected value
%      sigma : Variance
%         B0 : Initial value
%% OUTPUT:
%         B :  matrix representing M trajectories at N steps
%
%% EXAMPLE1:
%         t0 = 2.0; T = 3.0; N = 300; M = 500;
%         mu = 5.0; sigma = 0.7; B0 = 100;
%         B = ArithmeticBrownian(t0,T,M,N,mu,sigma,B0);
%%
dT = T/N; % longitud del paso de simulacin
X = randn(M,N); % X ~ N(0,1)
%% Simulacin
t = t0:dT:(t0+T);% equivalente a: t = linspace(t0,t0+T,N+1)
factor = mu*dT+sigma*sqrt(dT)*X;
B = cumsum([B0*ones(M,1) factor], 2);
%%
figure(1); hold on
     plot(t,B(1:50,:)')
     plot(t, B0 + mu.*(t-t0),'r','LineWidth',2.5)
hold off
```

```
title ('Grfico 1')
%%
figure(2); hold on
    for i=1:N+1
       desvt(i) = std(B(:,i));
    end
    plot(t,desvt)
    plot(t,sigma.*sqrt(t-t0))
hold off
title ('Desviacin estandar')
%%
figure(3); hold on
    t1 = 3.0;
    for i=1:N+1
        covmatrix = cov(B(:,i),B(:,(t1-t0)/dT));
        autocov(i) = covmatrix(1,2);
    end
    plot(t,autocov)
    plot(t,sigma^2.*min(t-t0,t1-t0))
hold off
title ('Autocovarianza')
scale = 1;
t_refTot = [2.0 3.0 4.0 5.0];
%%
for i=1:4
    t_ref = t_refTot(i);
    modelPdf = @(b)(normpdf(b,B0+mu*(t_ref-t0),sigma*sqrt(t_ref-t0)));
    centro = mu*(t_ref-t0);
    radio = 4*sigma*sqrt(t_ref-t0);
    B_min = B0 + min(centro - radio);
    B_max = B0 + max(centro + radio);
    figure(3+i)
    graphicalComparisonPdf(B(:,100*(i-1)+1),modelPdf,scale,B_min,B_max)
    title([ 'Histograma del proceso para t = ',num2str(i+1)])
end
```

(i) Simulad 500 **trayectorias del proceso Browniano con** $\mu = 5.0, \quad \sigma = 0.7$; **en** 300 **pasos de tiempo, en el intervalo** $[2, 5]$**, partiendo de**$_0 = 100$ **y representa las 50 primeras trayectorias, demostrando que la media crece linealmente con el tiempo.**

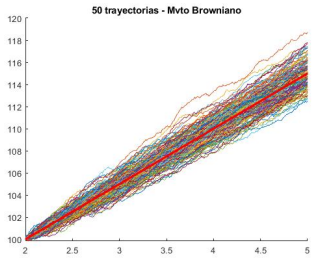
Si comparamos los estadísticos del proceso estimados a partir de todas las simulaciones con los valores teóricos, podemos comprobar que tanto la media, la desviación estándar como la autocovarianza del proceso se parecen, en general, a sus valores teóricos. Y si, además, realizamos los histogramas del proceso y los comparamos con la distribución teórica, vemos que esta sigue una districión normal de media $B0 + \mu * (t - t0)$ y desv típica $\sigma * sqrt(t - t0)$

(iv) En una segunda gráfica, ilustrad que la desviación estándar del proceso sigue la ecuación

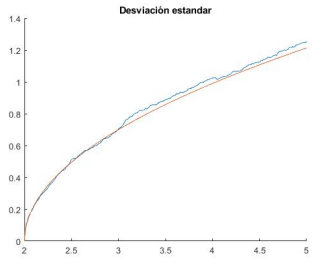
$$std[B(t)] = \sigma\sqrt{t - t_{0)}}$$

(vi) Finalmente, mostrad histogramas del proceso para $t = 2$**,**$t = 3$**,** $t = 4$ **y para** $t = 5$ **que ilustren que el Browniano aritmético sigue una normal**

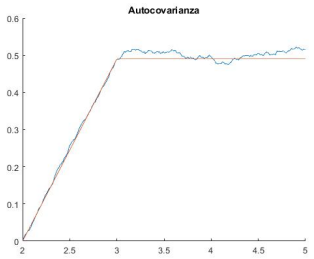
$$B(t) \backsim \ N(B_0 \ + \ \mu(t - t_0), \sigma\sqrt{t - t_0})$$



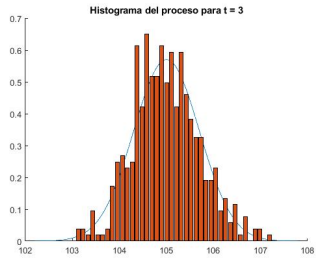
(a)



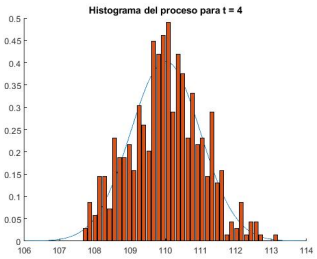
(b)



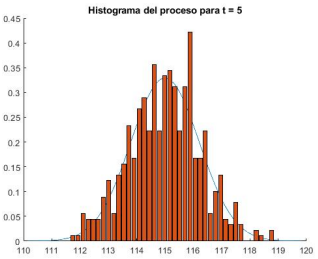
(c)



(d)



(e)



(f)

(vii) ¿A qué distribución tiende la solución del Browniano aritmético cuando $t \to t_0$? Es decir, ¿cuál es la distribución límite de una normal cuya desviación estándar tiende a cero?

Cuando la desviación estándar tiende a cero entonces la función de densidad y de probabilidad no están definidas, ya que tomando límites en sus formas explícitas se observa que cuando $t \to t_0$ la función diverge. De forma cualitativa, se puede observar que cuando esto ocurre la función de densidad tiende a infinito en t_0 mientras que tiende a 0 sino. Este es el caso de la distribución delta de Dirac, que se puede definir en forma de gaussiana como:

$$\int_{-\infty}^{+\infty} \delta(x) = 1 \qquad \delta(x) = \lim_{\sigma \to 0} \frac{\exp \frac{-x^2}{2\sigma^2}}{\sqrt{2\pi}\sigma}$$

Esta distribución presenta asímismo la propiedad de traslación, en este caso:

$$\int_{-\infty}^{+\infty} f(t|\mu, 0)dt = \int_{-\infty}^{+\infty} \delta(t-\mu)dt = B_0$$

7 - Movimiento browniano geométrico

(i,ii,iii,iv) Simula, ilustra la media y representa los histogramas

```
function S = GeometricBrownian(t0,T,M,N,mu,sigma,S0)
%% ArithmeticBrownian: Simulates M geometric Brownian movements
%
%% SYNTAX:
%          [B] = GeometricBrownian(t0,T,M,N,mu,sigma,B0)
%
%% INPUT:
%         t0 : Initial simulation time
%          T : Final simulation time
%          M : Number of simulations
%          N : Numbers of step (from t0 to t0+T)
```

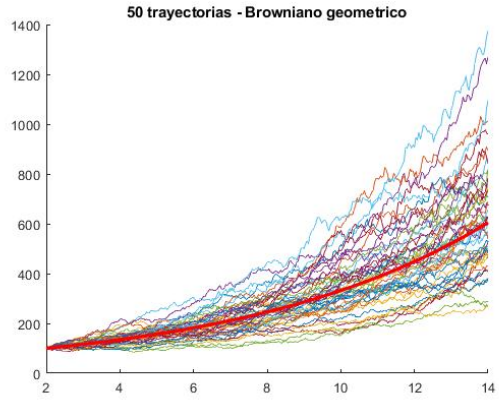
```
%        mu : Expected value
%     sigma : Variance
%        S0 : Initial value
%% OUTPUT:
%        S :  matrix representing M trajectories at N steps
%
%% EXAMPLE1:
%        t0 = 2.0; T = 12.0; N = 300; M = 500;
%        mu = 0.15; sigma = 0.1; S0 = 100;
%        gbrown = GeometricBrownian(t0,T,M,N,mu,sigma,S0);
%%
dT = T/N; % longitud del paso de simulacin
X = randn(M,N); % X ~ N(0,1)
%% Simulacin
t = t0:dT:(t0+T);% equivalente a: t = linspace(t0,t0+T,N+1)
factor = exp((mu-0.5*sigma^2)*dT+sigma*sqrt(dT)*X);
S = cumprod([S0*ones(M,1) factor], 2);
figure(1); hold on
    plot(t,S(1:50,:)')
    plot(t, S0*exp(mu.*(t-t0)),'r','LineWidth',2.5)
hold off
title('50 trayectorias - Browniano geometrico')

%%
t_refTot = [2.0 6.0 10.0 14.0];
scale = 0;
for i=1:4
    t_ref = t_refTot(i);
    modelPdf = @(b) (lognpdf(b,log(S0)+(mu-0.5*sigma^2)*(t_ref-t0),sigma*
        sqrt(t_ref-t0)));
    centro = (mu-0.5*sigma^2)*(t_ref-t0);
    radio = 4.0*sigma*sqrt(t_ref-t0);
    S_min = S0*exp(min(centro - radio));
    S_max = S0*exp(max(centro + radio));
    figure (1+i)
    graphicalComparisonPdf(S(:,100*(i-1)+1),modelPdf,scale,S_min,S_max)
    title([ 'Histograma del proceso para t = ',num2str(4*(i-1)+2)])
end
```

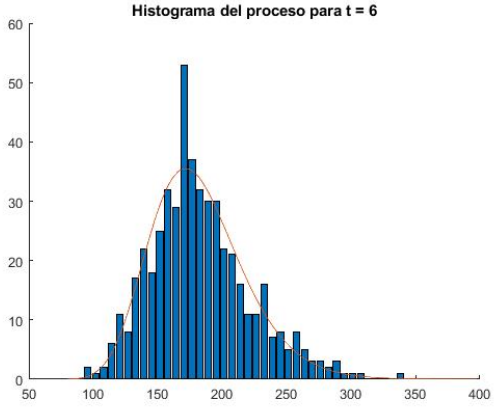
Al igual que el ejercicio anterior, vemos que la media crece linealmente con el tiempo de acuerdo de acuerdo con la expresión del apartado tres y si realizamos los histogramas del proceso y los comparamos con la distribución teórica, vemos que esta sigue una districión lognormal de media $(log(S0)+(\mu\frac{\sigma^2}{(t-t0)})$, y desviación típica $\sigma(t-t0)$. Para $t=t0$ evidentemente el histograma no mostrará ninguna distribución sino todos los valores concentrados en un punto, 100, ya que es donde se encuentran en el instante inicial todas las trayectorias.

(v) ¿A qué distribución tiende la solución del Browniano aritmético cuando $t \to t_0$? Es decir, ¿cuál es la distribución límite de una lognormal cuya desviación estándar tiende a cero?

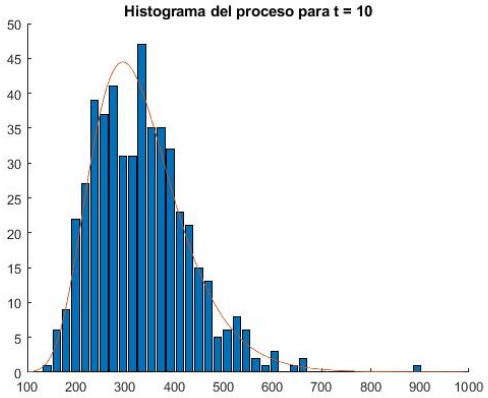
Dada la forma generalizada que tiene la distribución lognormal, al igual que la anterior el límite se puede expresar a través de la delta de Dirac, ya que puede ser compuesta con una función suavemente comportada, como el logaritmo natural.



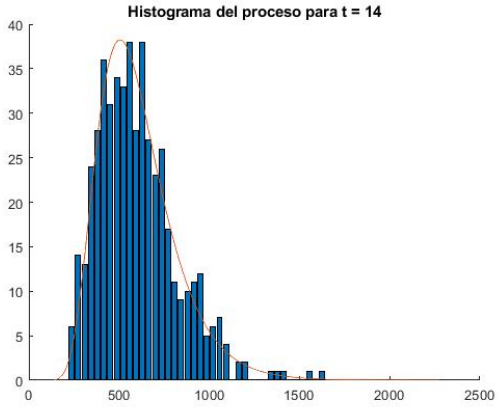
(a)



(b)



(c)



(d)

---

8 - Serie temporal AR(1)

El objetivo es escribir una función Matlab que devuelva una matriz con M filas y T columnas. Cada una de las filas corresponde a una serie temporal generada a partir de las condiciones iniciales mediante el procedimiento descrito.

```
1     function [X,u] = simAR1(M,T, phi0,phi1,sigma,X0)
2  %% simAR1 : Simulacion de un proceso AR(1)
3  %
4  %% SYNTAX:
5  %              X = simAR1(M,T,phi0,phi1,sigma,X0)
6  %
7  %% INPUT:
8  %                  X : Matriz (M,T) que contiene los M trayectorias simuladas
9  %                  M : Numero de trayectorias simuladas
10 %                  T : Tiempos para la simulacin
11 %                 X0 : Valor inicial de la serie temporal
12 % phi0,phi1,sigma : Parmetros que caracterizan el proceso AR(1)
13 %
14 % Ejemplo 1:
15 % M = 3; T = 1000; X0 = 0; phi0 = 0.0; phi1 = 0.7; sigma = 0.25;
```

```
% [X,u] = simAR1(M,T,phi0,phi1,sigma,X0);
% figure(1); plot(1:30,X(:,1:30));
% figure(2); autocorr(X(1,:),30);
% figure(3); subplot(2,1,1); autocorr(u(1,:),30);
% subplot(2,1,2); autocorr(abs(u(1,:)),30);
%
% Ejemplo 2: Simulacin aprox. estacionaria
% M = 1; T = 1000; X0 = 0; phi0 = 0.0; phi1 = -0.7; sigma = 0.25;
% [X,u] = simAR1(M,T,phi0,phi1,sigma,X0);
% figure(1); plot(1:T,X);
% figure(2); autocorr(X(1,:),30);
% figure(3); subplot(2,1,1); autocorr(u(1,:),30);
% subplot(2,1,2); autocorr(abs(u(1,:)),30);
%
% Ejemplo 3: Simulacin no estacionaria
% M = 1; T = 500; X0 = 10; phi0 = 0.0; phi1 = 0.95; sigma = 0.25;
% [X,u] = simAR1(M,T,phi0,phi1,sigma,X0);
% figure(1); plot(1:T,X);
%%
3X = zeros(M,T);
u = sigma*randn(M,T); % X ~ N(0,1)
X(:,1) = X0;
for i = 2:T
    X(:,i) = phi0+phi1*X(:,i-1)+u(:,i);
end
```

La implementación obtenida del código para el ejemplo 1: La matriz realizada contiene M series tempo-

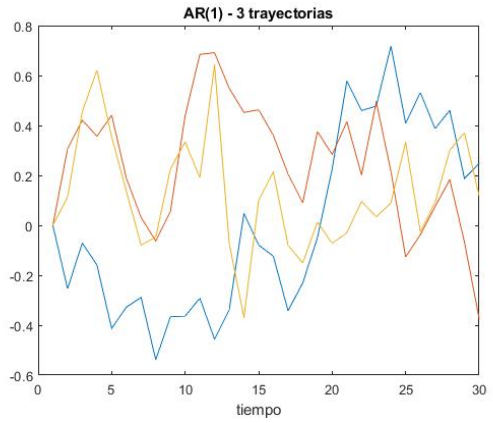


Figure 1: 3 trayectorias AR(1) simuladas

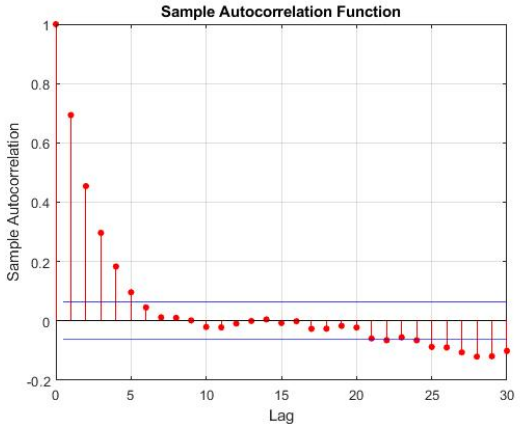


Figure 2: Función de autocorrelación con 30 lags

rales AR (1), cada una de las cuales describe un proceso en que las observaciones en un momento dado son predecibles a partir de las observaciones previas del proceso más una innovación del proceso u, es decir, cada observación en un momento dado es modelada en función de valores anteriores. Si las raíces del polinomio caen fuera del círculo de la unidad el proceso es estacionario, es decir, la distribución del proceso en un instante del tiempo fijo o un posición fija es la misma para todos los instantes del tiempo o posiciones. Por lo que, de existir, la media y la varianza no varían a lo largo del tiempo o la posición.

9 - Modelos ARMA(p,q)+GARCH(r,s)

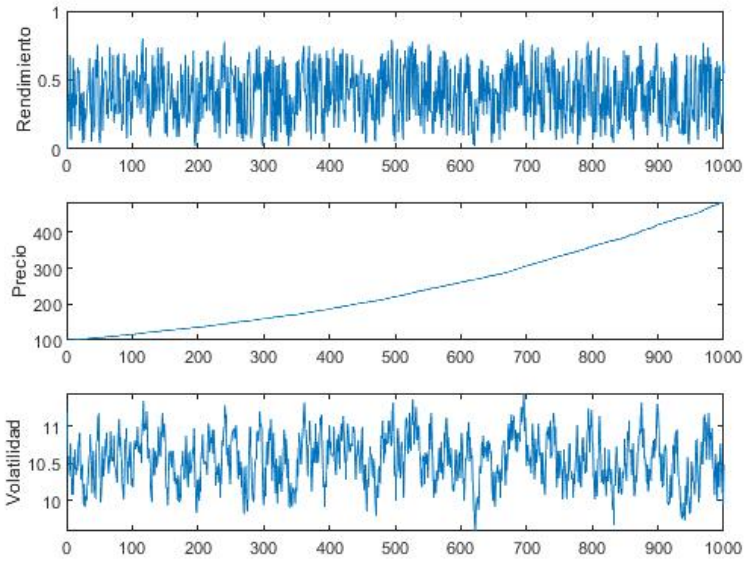
```
function [X,u,h] = simARMAGARCH(M,T,phi0,phi1,theta,kappa,alpha,beta,X0,u0,
    h0)
% simARMAGARCH : Simulacin de un proceso ARMA(p,q) + GARCH(r,s)
```

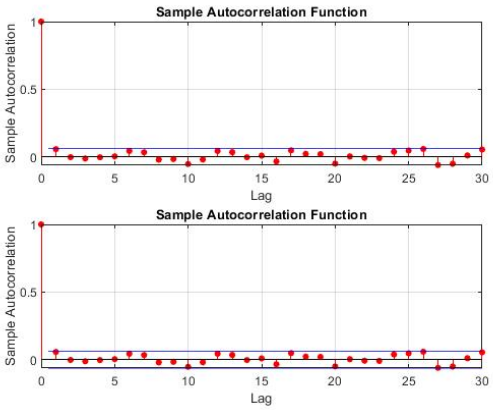
```
%
%% Sintaxis:
% X = simARMAGARCH(M,T,phi0,phi1,theta,kappa,alpha,beta,X0,u0,h0)
%
%% INPUT:
%                       M : Numero de trayectorias simuladas
%                       T : Pasos para la simulacin
%         phi0,phi1,theta : Parmetros que caracterizan el proceso ARMA(p,q)
%       kappa,alpha,beta  : Parmetros que caracterizan el proceso GARCH(r,s)
%                     X0  : Valores iniciales de la serie temporal
%                     u0  : Valores iniciales de las innovaciones
%                     h0  : Valores iniciales de la volatilidad
%
%% OUTPUT:
%                       X : Matriz (M,T) que contiene las M trayectorias
    simuladas
%                       u : Matriz (M,T) que contiene la matriz de errores
    sqrt(ht)*error_t
%                       h : Matriz (M,T) que contiene el modelo GARCH(r,s)
%
%% Ejemplo 1: Simulacin ARMA(2,3) + GARCH(1,2)
% M = 3; T = 1000; d = 3;
% phi0 = 0.0; phi1 = [0.4 -0.2]; theta = [0.3 -0.6 0.1];
% kappa = 0.1; alpha = 0.1; beta = [0.6 0.2];
% X0 = [0 0 0]; u0 = [0 0 0]; h0 =kappa/(1-sum(alpha)-sum(beta))*ones(1,d);
% [X,u,h]=simARMAGARCH(M,T,phi0,phi1,theta,kappa,alpha,beta,X0,u0,h0);
% diasEnAnyo = 250;
% S0 = 100; dT = 1/diasEnAnyo; S = cumprod([S0*ones(M,1) exp(X*dT)],2);
% figure(1); subplot(3,1,1); plot(1:T,X(:,1:T)); ylabel('Rendimiento');
% subplot(3,1,2); plot(1:T,S(:,1:T)); ylabel('Precio');
% subplot(3,1,3);
% plot(1:T,sqrt(diasEnAnyo*h(1,1:T))); ylabel('Volatilidad');
% figure(2); autocorr(X(1,:),30);
% figure(3); subplot(2,1,1); autocorr(u(1,:),30);
% subplot(2,1,2); autocorr(abs(u(1,:)),30);
%
%% Ejemplo 2: Simulacin AR(1)
% M = 1; T = 1000; phi0 = 0.0; phi1 = -0.7; sigma = 0.25;
% theta = []; kappa = sigma; alpha = []; beta = []; % simula AR(1)
% X0 = 0; u0 = 0; h0 = sigma;
% [X,u,h]=simARMAGARCH(M,T,phi0,phi1,theta,kappa,alpha,beta,X0,u0,h0);
% figure(1); plot(1:T,X(:,1:T));
% figure(2); T1 = 50; plot(1:T1,X(:,1:T1));
% figure(3); autocorr(X(1,:),30);
% figure(4); subplot(2,1,1); autocorr(u(1,:),30);
% subplot(2,1,2); autocorr(abs(u(1,:)),30);
% figure(5); plot(1:T,h(1,:));
%
%% Ejemplo 3: Simulacin AR(1) + GARCH(1,1)
% M = 1; T = 1000;
% phi0 = 0.0; phi1 = 0.15;
% theta = [];
% kappa = 0.1; alpha = 0.10; beta = 0.7;
% X0 = 0; u0 = 0; h0 = kappa/(1-alpha-beta);
% [X,u,h]=simARMAGARCH(M,T,phi0,phi1,theta,kappa,alpha,beta,X0,u0,h0);
% diasEnAnyo = 250;
% S0 = 100; dT = 1/diasEnAnyo; S = cumprod([S0*ones(M,1) exp(X*dT)],2);
```

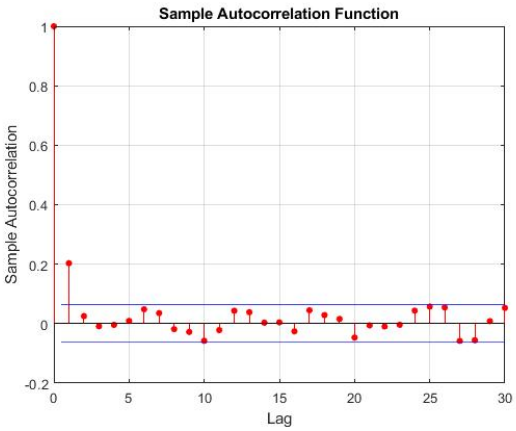
```
58 % figure(1); subplot(3,1,1); plot(1:T,X(:,1:T)); ylabel('Rendimiento');
59 % subplot(3,1,2); plot(1:T,S(:,1:T)); ylabel('Precio');
60 % subplot(3,1,3);
61 % plot(1:T,sqrt(diasEnAnyo*h(1,1:T)));ylabel('Volatilidad');
62 % figure(2); autocorr(X(1,:),30);
63 % figure(3); subplot(2,1,1); autocorr(u(1,:),30);
64 % subplot(2,1,2); autocorr(abs(u(1,:)),30);
65 %
66 %% Introducir los vectores en fila->indican columna de t=1,2...
67 %                                  1   4   X13   X14 ...
68 %     [1 2 3 ; 4 5 6]    ->       2   5   X23   X24 ...
69 %                                  3   6   X33   X34 ...
70 %% Define el valor de p,q,r,s
71 P = length(phi1);
72 Q = length(theta);
73 R = length(alpha);
74 S = length(beta);
75 %% si el vector esta vacio entonces se rellena con 0, evita errores despues
76 if isempty(phi0) == 1 phi0=0; end
77 if isempty(phi1) == 1 phi1=0;  end
78 if isempty(theta) == 1 theta=0;  end
79 if isempty(kappa) == 1 kappa=0;  end
80 if isempty(alpha) == 1 alpha=0;  end
81 if isempty(beta) == 1 beta=0;  end
82 %% Inicializa las variables y las organiza adecuadamente
83 t0 = max([size(X0,2),size(u0,2),size(h0,2)]);   %calcula el vector inicial
       de mayor longitud
84 h=zeros(M,T); X = zeros(M,T); u = zeros(M,T); perturb = rand(M,T); %Create
       all matrix
85 h(:,1:size(h0,1)) = h0'; X(:,1:size(X0,1)) = X0'; u(:,1:size(u0,1)) = u0';
       %Escribe los valores iniciales a las primeras columnas de los finales
86 %Para h0,X0,u0 se consideran 3 casos:
87 %   - que sean vectores vacios ->se rellenan con num aleat uniformes
88 %   - que tengan dim multiplo de t0->se rellenan en los vacios
89 %   - no multiplos: las columnas restantes se rellenan con n aleat
       uniformes
90 if size(h0',2) < t0
91     if isempty(h0) == 1
92         h(:,1:t0) =rand(M,t0);
93     elseif mod(t0,size(h0',2)) ~= 0
94         h(:,size(h0',2)+1:t0) = rand(M,t0-size(h0',2));
95     else
96         h(:,size(h0',2)+1:t0) = repmat(h0',1,t0-size(h0',2));
97     end
98 end
99 if size(u0',2) < t0
100     if isempty(u0) == 1
101         u(:,1:t0) =rand(M,t0);
102     elseif mod(t0,size(u0',2)) ~= 0
103         u(:,size(u0',2)+1:t0) = rand(M,t0-size(u0',2));
104     else
105         u(:,size(u0',2)+1:t0) = repmat(u0',1,t0-size(u0',2));
106     end
107 end
108 if size(X0',2) < t0
109     if isempty(X0) == 1
110         X(:,1:t0) =rand(M,t0);
```

```
111         elseif mod(t0,size(X0',2))~= 0
112             X(:,size(X0',2)+1:t0) = rand(M,t0-size(X0',2));
113         else
114             X(:,size(X0',2)+1:t0) = repmat(X0',1,t0-size(X0',2));
115         end
116 end
117 %% Implementacion de las proceso estocastico ARMA(p,q)+GARCH(r,s)
118 %   Los sumatorios se realizan seleccionando los elementos correspondientes
119 %   del vector, dandoles la vuelta (flip), multiplicarlo por los
120 %   coeficientes y sumando los elementos del vector resultante
121 for j=t0+1:T
122     h(:,j) = kappa+sum(alpha.*flip(u(:,j-R:j-1)),2)+sum(beta.*flip(h(:,j-S:
            j-1)),2);
123     u(:,j) = sqrt(h(:,j)).*perturb(:,j);
124     X(:,j) = phi0 + sum(phi1.*flip(X(:,j-P:j-1)),2) + sum(theta.*flip(u(:,j
            -Q:j-1)),2)+u(:,j);
125 end
```

La explicación de la implementación de los algoritmos, así como el código para crear condiciones iniciales en caso de que no estén dadas en el enunciado o no estén completas se encuentra en el interior de la función.
Simulación del modelo AR(1)+GARCH(1,1) con las condiciones iniciales dadas por el enunciado:




Sample Autocorrelation Function
Sample Autocorrelation
Lag
Sample Autocorrelation Function
Sample Autocorrelation
Lag


Sample Autocorrelation Function
Sample Autocorrelation
Lag