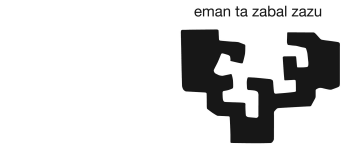


Ejercicios 2

Cálculo Numérico en Finanzas

Máster en Banca y Finanzas Cuantitativas

Autores: Edoardo Busetti y Pablo Muñoz

Bilbao, a 15 de Mayo de 2019.

ÍNDICE

Ejercicio 1: Cálculo de la Sensibilidad Delta

EJERCICIO 1

· APARTADO (i)

· FÓRMULA DESARROLLO DE TAYLOR: $\sum_{m=0}^{\infty} \frac{f^{(m)}(a)}{m!} \cdot (x-a)^m$

EN NUESTRO CASO: $a = X_0 \quad x = X_0 + dx$

· $f(X_0 + dx) \simeq f(X_0) + dx\, f'(X_0) + \mathcal{O}(dx^2)$

↳ DESPEJAMOS: $f'(X_0) = \frac{f(X_0+dx) - f(X_0) + \mathcal{O}(dx^2)}{dx} = \boxed{\frac{f(X_0+dx) - f(X_0)}{dx} + \mathcal{O}(dx)}$

· $f(X_0 \oplus dx) \simeq f(X_0) + dx\, f'(X_0) + \frac{1}{2!} \cdot dx^2 \cdot f''(X_0) + \mathcal{O}(dx^3)$

· $f(X_0 \ominus dx) \simeq f(X_0) + (-dx) \cdot f'(X_0) + \frac{1}{2!}(-dx^2) \cdot f''(X_0) + \mathcal{O}(dx^3)$

· $f(X_0 \oplus dx) - f(X_0 \ominus dx) \simeq 2dx \cdot f'(X_0) + 0 + \mathcal{O}(dx^3)$

↳ DESPEJAMOS: $f'(X_0) = \frac{f(X_0+dx) - f(X_0-dx) + \mathcal{O}(dx^3)}{2dx} = \boxed{\frac{f(X_0+dx) - f(X_0-dx)}{2dx} + \mathcal{O}(dx^2)}$

· LA APROXIMACIÓN A LA DERIVADA PREFERIBLE ES LA SEGUNDA YA QUE TIENE UN ERROR DE ORDEN MÁS PEQUEÑO.

· APARTADO (ii)

$X_0 \neq 0$ → ESCRIBIR LA FÓRMULA SUPONIENDO QUE $dx = X_0 \cdot h$

· FÓRMULA ① $f'(X_0) \simeq \frac{f(X_0 + X_0 \cdot h) - f(X_0)}{X_0 \cdot h} + \mathcal{O}(h) \rightarrow f'(X_0) \simeq \frac{f(X_0(1+h)) - f(X_0)}{X_0 \cdot h} + \mathcal{O}(h)$

· FÓRMULA ② $f'(X_0) \simeq \frac{f(X_0 + X_0 \cdot h) - f(X_0 - X_0 \cdot h)}{2 \cdot X_0 \cdot h} + \mathcal{O}(h^2) \rightarrow f'(X_0) \simeq \frac{f(X_0(1+h)) - f(X_0(1-h))}{2 \cdot X_0 \cdot h} + \mathcal{O}(h^2)$

Apartado III

En este apartado hemos diseñado una Función en Matlab para que nos calcula la Delta de una Opción Call Europea utilizando diferencias divididas. La función se llama "deltaCallEU" e incorporará en ella 2 funciones más:

- numericalDerivative.m
- priceEuropeanCall.m

Código (deltaCallEU):

```
function delta = deltaCallEU(h,S0,K,r,T,sigma)
%% deltaCallEU: delta de una call europea
%
%% SYNTAX:
%       delta = deltaCallEU(h,S0,K,r,T,sigma)
%
%% INPUT:
%       S0 : Initial value of the underlying asset
%        K : Strike price of the option
%        r : Risk free intrest rate
%        T : Time to expiry
%        sigma : Volatility
%        h : Sensibility of the numerical derivative
%% OUTPUT:
%    delta : Delta of the European Call Option
%
%%  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)

%Define f as the price of an european option in function of price
%while keeping all the other values constant.
f = @(price_undelying)priceEuropeanCall(price_undelying,K,r,T,sigma);

%Delta calculated as the derivative of the call price with respect to price.
delta = numericalDerivative(f,S0,h);
```

Código (priceEuropeanCall):

```
function price = priceEuropeanCall(S0,K,r,T,sigma)
%% priceEuropeanCall: Price a European Call using the Black Scholes Formula
%
%% SYNTAX:
%        price = priceEuropeanCall(S0,K,r,T,sigma)
%
%% INPUT:
%       S0 : Initial value of the underlying asset
%        K : Strike price of the option
%        r : Risk free intrest rate
%        T : Time to expiry
%    sigma : Volatility
%% OUTPUT:
%    price : Price of the European Call Option
%Using the Black Scholes formula for the price of a Call Option

discountedK = K.*exp(-r.*T);
totalVolatility = sigma.*sqrt(T);
d_plus = log(S0./discountedK)./totalVolatility + 0.5*totalVolatility;
d_minus = d_plus - totalVolatility;

price = S0.*normcdf(d_plus) - discountedK.*normcdf(d_minus);
end
```

Código (numericalDerivative):

```
function  derivative = numericalDerivative(f,x0,h)
% numericDerivative: Numerical estimate of the derivative of f at x0
%
% SYNTAX:
%        derivative = numericalDerivative(f,x0,h)
%
% INPUT:
%            f : Handle to the function whose derivative is being calculated
%           x0 : Point at which the derivative is calculated
%            h : Parameter for divided differences (1e-5 - 1e-6)
%
% OUTPUT:
%   derivative : Value of the derivative of f at x0
%
if x0 == 0
    derivative = (f(h) - f(-h))./(2*h);
else
    %We use this formula beacuse this way the relative error is constant,
    %since h is always summed to 1 and not to x0 like it would be in a
    %different implementation of the numerical derivative.
    derivative = (f(x0*(1+h))-f(x0*(1-h)))./(2*h*x0);
end
```

Una vez ejecutado el código, obtenemos el siguiente valor de la Delta de la Opción Call Europea:

Delta	0.7174

De cara a obtener el valor de h que nos da el resultado de la Delta con menor error hemos elaborado el siguiente código:

Código

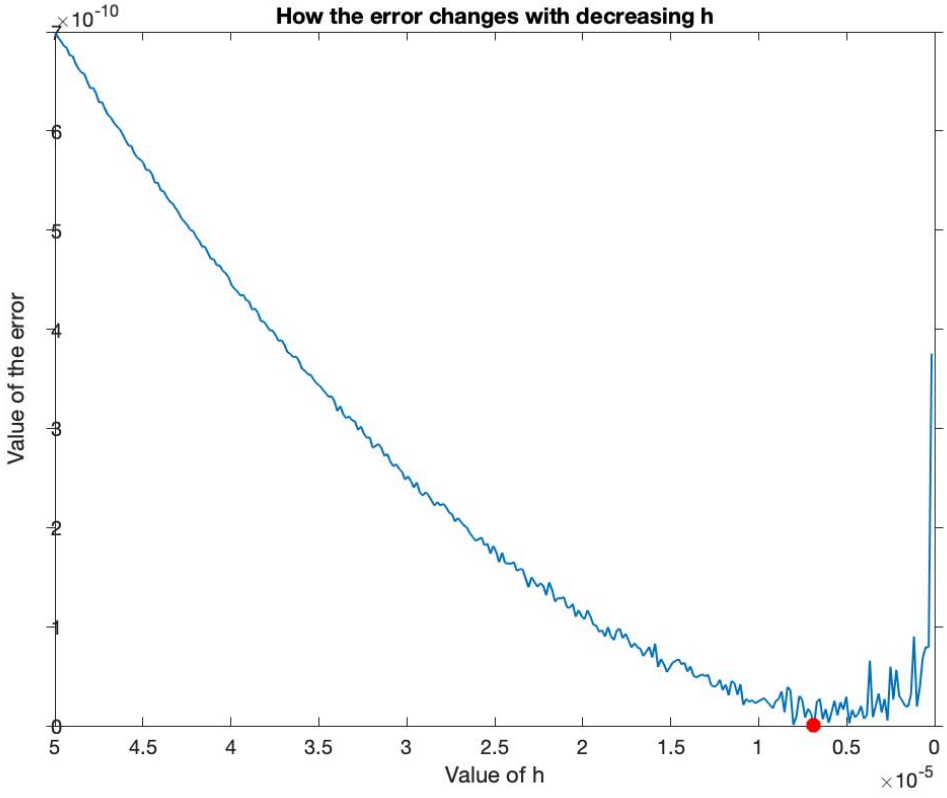
```
% Test program used to find the value of the sensibility "h" of the
% numerical derivative that minimizes the error of estimation of the delta
% of a call.

S0 = 100; K = 90; r = 0.03; T = 2; sigma = 0.4; N = 15;

h_vec = linspace(0,5e-5,300);

error = zeros(length(h_vec),1);
for j=1:length(h_vec)
    error(j)  = abs(deltaCallEU(h_vec(j),S0,K,r,T,sigma) -
blsdelta(S0,K,r,T,sigma))/blsdelta(S0,K,r,T,sigma);
end
figure(1)
plot(h_vec,error,'LineWidth',1)
%loglog(h_vec,error,'LineWidth',1)
set(gca, 'XDir','reverse')
xlabel('Value of h')
ylabel('Value of the error')
title('How the error changes with decreasing h')
hold on

[value,position] = min(error);
plot(h_vec(position),error(position),'r*','LineWidth',3)
min_error = value;
h_minerror = h_vec(position);

disp([newline,newline,'The value of h for the minimum error is:
',num2str(h_minerror),newline])

disp(['The minimum error found is:              ',num2str(min_error)])
```

Nota: Este código ha sido modificado respecto del original para conseguir una representación gráfica más sencilla de entender en el proceso de búsqueda del error mínimo.

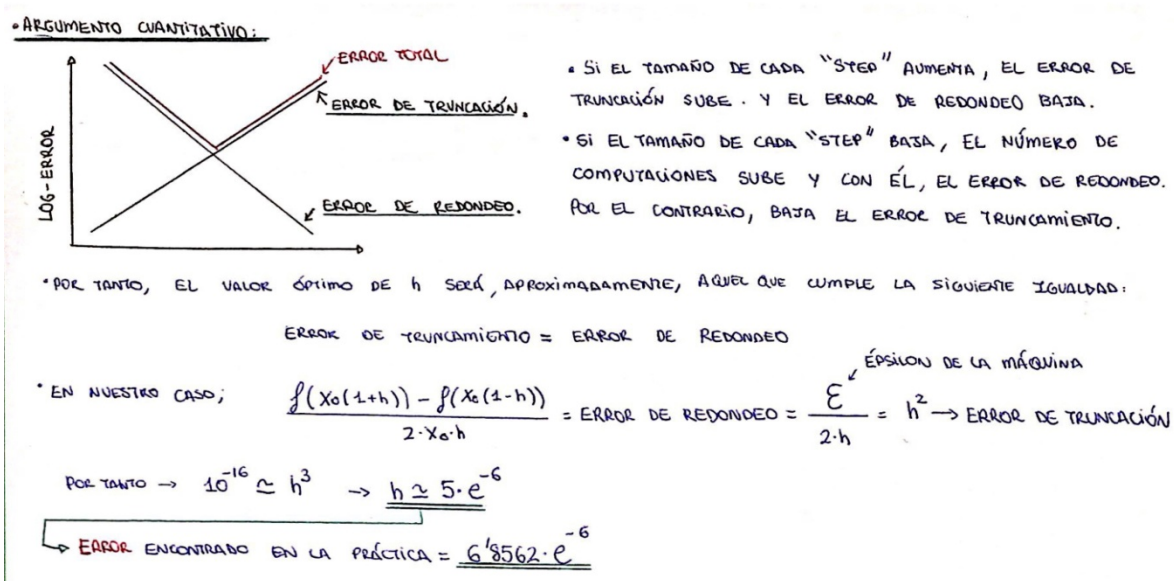
En el código original, para buscar el valor de h que obtiene el error mínimo, el vector h_vec estaba definido como: h_vec = 10.^-(1:-1:-16) para encontrar el tamaño de grandeza de h. Después de haber utilizado este código y haber encontrado que el mínimo error estaba cerca de 1e-6 hemos programado un gráfico tomando como h_vec 300 valores desde 0 hasta 5e-5.

Una vez ejecutado este segundo código para obtener el valor de H que minimiza el error hemos obtenido los siguientes outputs:

The value of h for the minimum error	6.8562e-06
The minimum error found	2.0304e-13



De cara a justificar el valor óptimo de h encontrado vamos a proporcionar un argumento cuantitativo (basado en el valor de épsilon y en el orden del error de truncación de la aproximación de la derivada por diferencias divididas).



Apartado IV

En este apartado vamos a diseñar una función en Matlab para estimar la Delta por MonteCarlo utilizando "*Common Random Numbers*".

Código

```
function [delta_MC,err_MC] = deltaCallEU_MC(M,S0,K,r,T,sigma)

%% deltaCallEU_MC: delta de una call europea mediante MC
%
%% SYNTAX:
%       [delta_MC,err_MC] = deltaCallEU_MC(M,S0,K,r,T,sigma)
%
%% INPUT:
%       S0 : Initial value of the underlying asset
%        K : Strike price of the option
%        r : Risk free intrest rate
%        T : Time to expiry
%        sigma : Volatility
%        h : Sensibility of the numerical derivative
%        M : Number of simulations
%
%% OUTPUT:
%    delta_MC : Delta of the European Call Option estimated via Monte Carlo
%    err_MC : Error of the Monte Carlo estimation
%
% 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)

%% generate M samples from N(0,1)
X = randn(M,1);

%% Set the value for Delta_x
Delta_x = 1e-4;

%% simulate M minus / plus trajectories in one step
S0_plus = S0+Delta_x;
S0_minus = S0-Delta_x;

%We simulate the paths that the asset would take in the case the underlying
%price was S0_plus and in the case it was S0_minus so that we can then
%apply the Divided Differences method for each simulated scenario

%% Calculate the M trajectories for S0_plus and S0_minus
```

```
TotalVolatitily = sigma*sqrt(T);
%Randomizing the Total Volatility to create M independent paths
TotalVolatitily = TotalVolatitily*X;

ST_plus = (S0_plus)*exp((r-(sigma^2)/2)*T+TotalVolatitily);
ST_minus = (S0_minus)*exp((r-(sigma^2)/2)*T+TotalVolatitily);
%% define minus / plus payoffs
payoffs_plus = max(ST_plus-K,0);
payoffs_minus = max(ST_minus-K,0);

%% compute Delta along each trajectory
discountFactor = exp(-r.*T);

%Applying the divided difference formula on the discounted payoffs (i.e. the prices)
delta_vector = (payoffs_plus*discountFactor - payoffs_minus*discountFactor)/(2*Delta_x);

%The variance is given by (â^'(g(X_m)-âŸ¨g(X)âŸ©_M )^2)/(M-1)
Standard_dev = std(delta_vector);

%% MC estimate

%The Monte Carlo Delta is the mean of the delta in each simulated scenario
delta_MC = mean(delta_vector);

%The error of the estimation is given by the std divided by sqrt(M)
err_MC = Standard_dev/sqrt(M);
```

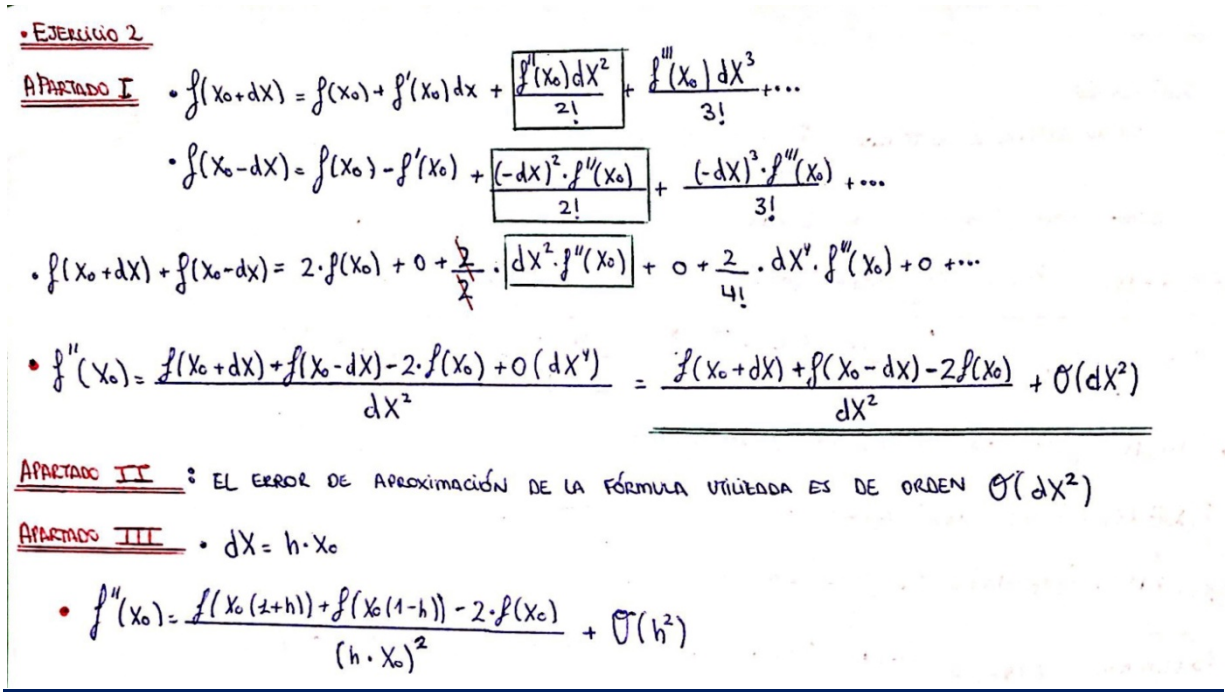
Este código se basa en la fórmula de diferencias divididas, donde se calculan dos diferentes series de payoffs, una a partir de S0+DeltaX y otra a partir de S0-DeltaX. Después, se aplica la fórmula de las diferencias divididas entre los payoffs descontados (es decir, los precios simulados) en cada estado y se hace una media.

El error de esta estimación es la deviación típica de los deltas calculados en cada estado simulado dividido por la raíz del número de simulaciones, es decir, disminuye de forma menor que la forma lineal al aumentar del número de simulaciones.

Una vez ejecutado el código obtenemos los siguientes valores:

Delta	0.7174
Delta_MC	0.7180
err_MC	8.2966e-04

Ejercicio 2: Cálculo de la sensibilidad gamma



Apartado IV

En este apartado hemos diseñado una función en Matlab para el cálculo de la Gamma de una Opción Call Europea utilizando la fórmula de diferencias divididas calculada en el primer apartado de este ejercicio.

Código:

```
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:
%       S0 : Initial value of the underlying asset
%        K : Strike price of the option
%        r : Risk free intrest rate
%        T : Time to expiry
%        sigma : Volatility
%        h : Sensibility of the numerical derivative
%% OUTPUT:
%    gamma : gamma of the European Call Option
%
%% EJEMPLO 1:
%  S0 = 100; K = 90; r = 0.03; T = 2; sigma = 0.4;
%  h = 1e-4; Value of h that minimizes the error
%  gamma = gammaCallEU(h,S0,K,r,T,sigma)
%  blsgamma(S0,K,r,T,sigma)


%Define f as the price of an european option in function of price
%while keeping all the other values constant.
f = @(price_undelying)priceEuropeanCall(price_undelying,K,r,T,sigma);

%gamma calculated as the second order derivative of the call price with respect to
price.
% using the formula derivated in the fist part of this exercise.
gamma = (f(S0*(1+h)) + f(S0*(1-h)) - 2*f(S0))/((h*S0)^2);
```

Una vez ejecutado el código obtenemos el siguiente valor para la Gamma de la opción:

Gamma	0.0060

El valor de h que nos da el resultado con menor error lo hemos hallado ejecutando el siguiente código que calcula el error de la Gamma por cada valor de h de tamaño desde 1 hasta 1e-16 (que es el épsilon de la máquina):

Código

```
%% (IV)¿Cuál es el valor de h que nos da el resultado con menor error?
%Ilustra la repuesta con evaluaciones de la funcion para distintos valores de h.

% Test program to find out the minimum value of h that minimizes the
% error of the estimation of the gamma of a call using the divided
% differences approach

S0 = 100; K = 90; r = 0.03; T = 2; sigma = 0.4; N = 16;

h_vec = 10.^(0:-1:-N);

error = zeros(1,N);
for j=1:length(h_vec)
    h = h_vec(j);
    error(j)  = abs(gammaCallEU(h,S0,K,r,T,sigma) - blsgamma(S0,K,r,T,sigma));
end

%Minimum value found for the error and the h that generates that min error
[min_error,position] = min(error);
h_min = h_vec(position);

disp([newline,newline,'The value of h for the minimum error is:
',num2str(h_min),newline])
disp(['The minimum error found is:               ',num2str(min_error)])
```

Una vez ejecutado el código, obtenemos este valor de h óptimo:

The value of h for the minimum error	1e-4
The minimum error found	8.2366e-14

Apartado V

En este apartado vamos a diseñar una función en Matlab para estimar la Gamma por MonteCarlo utilizando "*Common Random Numbers*".

El método utilizado es el mismo que hemos hecho para el calculo de la deltaMC, pero ahora usamos la fórmula de diferencias divididas que hemos obtenido en el apartado (i) de este ejercicio.

A esta función la hemos añadido un parámetro adicional “Delta_x” que, si no especificamos ningún valor será 10 por defecto. Esto se debe a que hemos utilizado esta misma función para buscar el valor óptimo de Delta_x, es decir, aquel valor de Delta_x que minimiza el error.

Código

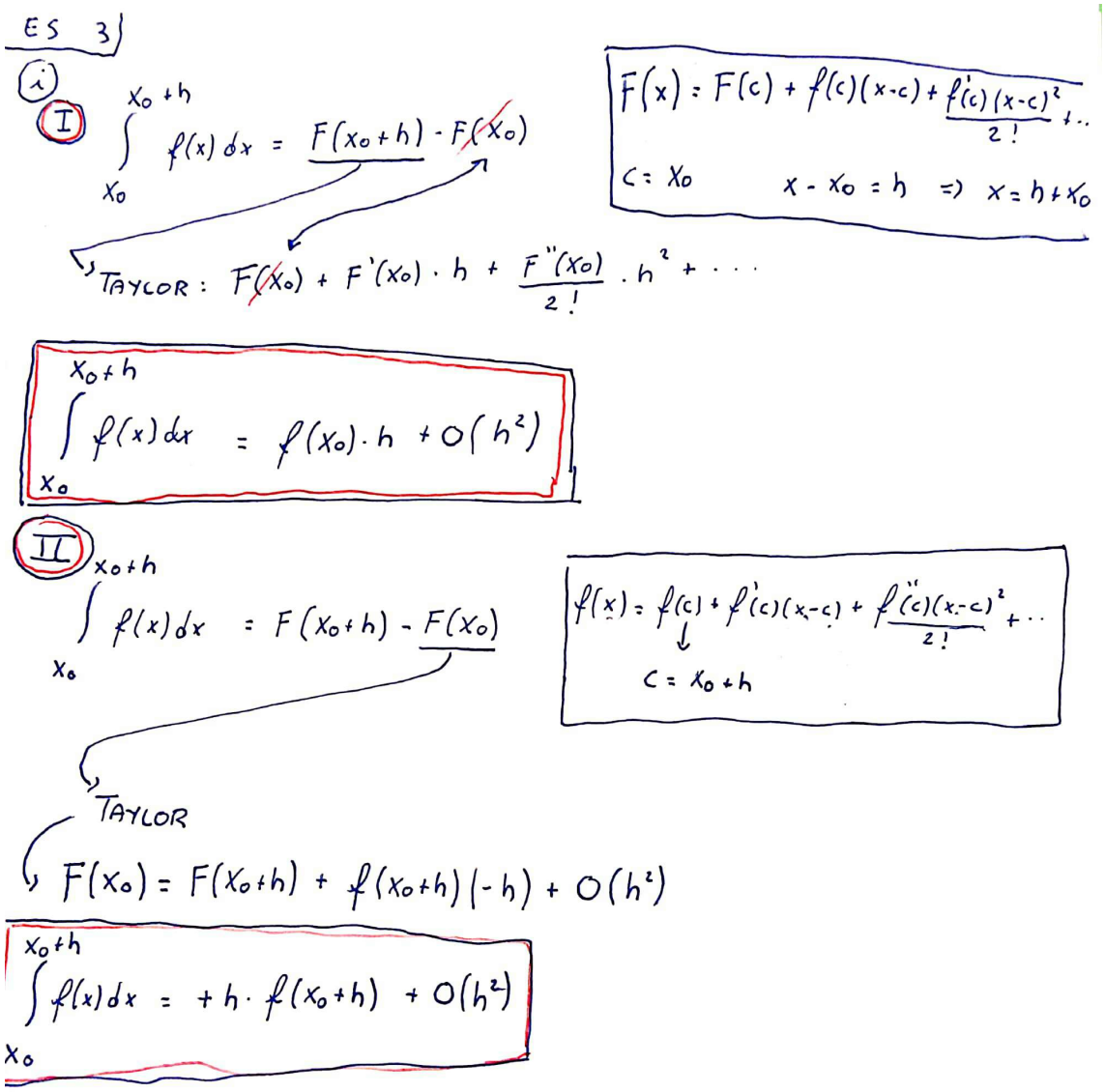
```
function [gamma_MC,err_MC] = gammaCallEU_MC(M,S0,K,r,T,sigma,Delta_x)

% deltaCallEU_MC: delta de una call europea mediante MC
%
%% SYNTAX:
%       [gamma_MC,err_MC] = gammaCallEU_MC(M,S0,K,r,T,sigma,Delta_x)
%
%% INPUT:
%       S0 : Initial value of the underlying asset
%        K : Strike price of the option
%        r : Risk free intrest rate
%        T : Time to expiry
%        sigma : Volatility
%        Delta_x : Sensibility of the numerical derivative
%        M : Number of simulations
%
%% OUTPUT:
%    gamma_MC : gamma of the European Call Option estimated via MC
%    err_MC : Error of the Monte Carlo estimation
%
%% EJEMPLO 1:
%  S0 = 100; K = 90; r = 0.03; T = 2; sigma = 0.4;
%  M = 1e6;
%  [gamma_MC,err_MC] = gammaCallEU_MC(M,S0,K,r,T,sigma)
%  h = 1e-4;
%  gammaCallEU(h,S0,K,r,T,sigma)
%  blsgamma(S0,K,r,T,sigma)

if ~exist('Delta_x','var')

     % third parameter does not exist, so default it to something
     %Value of Delta_x that minimizes the error

      Delta_x = 10;
 end

%% generate M samples from N(0,1)
X = randn(M,1);

%% Set the value for Delta_x -> Not needed here
%Delta_x = 1;

%% simulate M minus / plus trajectories in one step
S0_plus = S0+Delta_x;
S0_minus = S0-Delta_x;

%We simulate the paths that the asset would take in the case the underlying
%price was S0_plus and in the case it was S0_minus so that we can then
%apply the Divided Differences method for each simulated scenario


%% Calculate the M trajectories for S0_plus and S0_minus
TotalVolatitily = sigma*sqrt(T);

%Randomizing the Total Volatility to create M independent paths
TotalVolatitily = TotalVolatitily*X;

ST_plus = (S0_plus)*exp((r-(sigma^2)/2)*T+TotalVolatitily);
ST_minus = (S0_minus)*exp((r-(sigma^2)/2)*T+TotalVolatitily);
ST = (S0)*exp((r-(sigma^2)/2)*T+TotalVolatitily);

%% define minus / plus payoffs
payoffs_plus = max(ST_plus-K,0);
payoffs_minus = max(ST_minus-K,0);
payoffs = max(ST-K,0);

%% compute Delta along each trajectory
discountFactor = exp(-r.*T);
%Applying the divided difference formula on the discounted payoffs
Prices_plus = payoffs_plus*discountFactor;
Prices_minus = payoffs_minus*discountFactor;
Prices = payoffs*discountFactor;

gamma_vector = (Prices_plus + Prices_minus - 2*Prices)/((Delta_x)^2);
std_gamma_vec = std(gamma_vector);

%% MC estimate
```

```
%The Monte Carlo Gamma is the mean of the Gamma in each simulated scenario
gamma_MC = mean(gamma_vector);
%err_MC = std(gamma_vector)/sqrt(M);
err_MC = std_gamma_vec/sqrt(M);
```

Una vez ejecutado el código obtenemos los siguientes valores:

Gamma	0.0060
Gamma_MC	0.0060
err_MC	1.7420e-05

Ejercicio 3: Ejercicio de derivación en papel



EJERCICIO (III)

$$\int_{x_0}^{x_0+h} f(x)dx \approx \frac{f(x_0)+f(x_0+h)}{2}\cdot h + \mathcal{O}(h^{n_{III}})$$

- DADO QUE $f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \dots$

  - $x = x_0$
  - $a = x_0+h$
  - $x-a = h$

- $F(x_0) = F(x_0+h) + F'(x_0+h)\cdot(-h) + \frac{(-h)^2\cdot F''(x_0+h)}{2!} + \frac{(-h)^3\cdot F'''(x_0+h)}{3!}$ (↑ TAYLOR)

(A) $\boxed{F(x_0+h) - F(x_0)} = h\cdot F'(x_0+h) - \frac{h^2 F''(x_0+h)}{2!} + \frac{h^3 F'''(x_0+h)}{3!}$

- $F(x_0+h) \overset{\text{TAYLOR}}{=} F(x_0) + F'(x_0)\cdot(h) + \frac{F''(x_0)\,h^2}{2!} + \dots$

(B) $\boxed{F(x_0+h) - F(x_0)} = F'(x_0)\cdot(h) + \frac{F''(x_0)\cdot h^2}{2!} + \dots$

$$A + B = \not{2}\cdot\int_{x_0}^{x_0+h} f(x) = \frac{(f(x_0+h)+f(x_0))\cdot h}{2} + \frac{h^2\left(F''(x_0) - F''(x_0+h)\right)}{2!} + \dots$$

↳ PARECE QUE ES DE ORDEN 2 PERO LA REGLA DE NEWTON-COTES DICE QUE DEBERÍA SER DE ORDEN 3.

IV

QUIERO

$$x - c = \frac{h}{2} \qquad c = x_0 + \frac{h}{2}$$

$$x = \frac{h}{2} + c \qquad x = x_0 + h$$

$$F(x_0+h) = F\left(x_0+\frac{h}{2}\right) + F'\left(x_0+\frac{h}{2}\right)\cdot\frac{h}{2} + \frac{F''\left(x_0+\frac{h}{2}\right)\left(\frac{h}{2}\right)^2}{2!} + \dots$$

QUIERO $\quad x - c = -\frac{h}{2} \qquad c = x_0 + \frac{h}{2}$

$$x = c - \frac{h}{2} = x_0$$

$$F(x_0) = F\left(x_0+\frac{h}{2}\right) - F'\left(x_0+\frac{h}{2}\right)\cdot\frac{h}{2} + \frac{F''\left(x_0+\frac{h}{2}\right)\cdot\left(\frac{h}{2}\right)^2}{2!} + \dots$$

$$\int_{x_0}^{x_0+h} f(x)\,dx = F(x_0+h) - F(x_0) = 2F'\left(x_0+\frac{h}{2}\right)\cdot\frac{h}{2} + 0 + \frac{F'''\left(x_0+\frac{h}{2}\right)\left(\frac{h}{2}\right)^3}{3!}$$

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

APARTADO (ii)

- SE HA PROPUESTO EL SIGUIENTE ALGORITMO PARA CALCULAR UNA INTEGRAL DEFINIDA:

$$I = \int_a^b f(x)\,dx = \left(\frac{1}{2}\cdot f(a) + \sum_{i=1}^{N} f(a+ih) + \frac{1}{2}\cdot f(b)\right)\cdot h$$

¿CUÁL ES EL ORDEN DE CONVERGENCIA, h, DEL ALGORITMO?

SIN RESPONDER.

¿QUÉ OCURRE SI N ES DEMASIADO PEQUEÑO?

SI N ES PEQUEÑO → h ES GRANDE → $O(h^n)$ ES GRANDE

└ ERROR DE TRUNCACIÓN.

¿QUÉ OCURRE SI N ES DEMASIADO GRANDE?

SI N ES GRANDE → h ES PEQUEÑO → $O(h^n)$ ES PEQUEÑO

└ ERROR DE TRUNCACIÓN PEQUEÑO, PERO EL ERROR DE REDONDEO ES GRANDE.

Apartado III

El código utilizado en este apartado es:

1) La función que calcula el integral:

```
function [Int] = IntegralDefinida(f,a,b,N)

h = (b-a)/N;

Int = (f(a)/2 + sum(f(a+(1:N-1)*h)) + f(b)/2)*h;
end
```

2) La function que determina el numero de iteraciones necesario para encontrar la integral definida de una function con una tolerancia absoluta de "TOL_ABS" donde el valor mas pequeno que puedo dar a "TOL_ABS" es 1e-16.

```
function [Int,N] = fFindN(f,a,b,TOL_ABS)
realInt = integral(f,a,b,'AbsTol',1e-16);

N = 1;
error = 1;
while error > TOL_ABS
    N = N+1;
    Int = IntegralDefinida(f,a,b,N);
    error = abs(Int - realInt);
end
```

3) El programa que comprueba la function.

```
f = @(x)exp(x); a = 0; b = 3 ; TOL_ABS = 1e-8;

[Int_Value,NumIter] = fFindN(f,a,b,TOL_ABS);

disp(['The value of the integral is : ',num2str(Int_Value),newline,'The Absolute
Tolerance is: ',num2str(TOL_ABS),newline,'The number of iterations needed is:
',num2str(NumIter)])
```

Los resultados de este programa son:

The value of the integral is : 19.0855

The Absolute Tolerance is: 1e-08

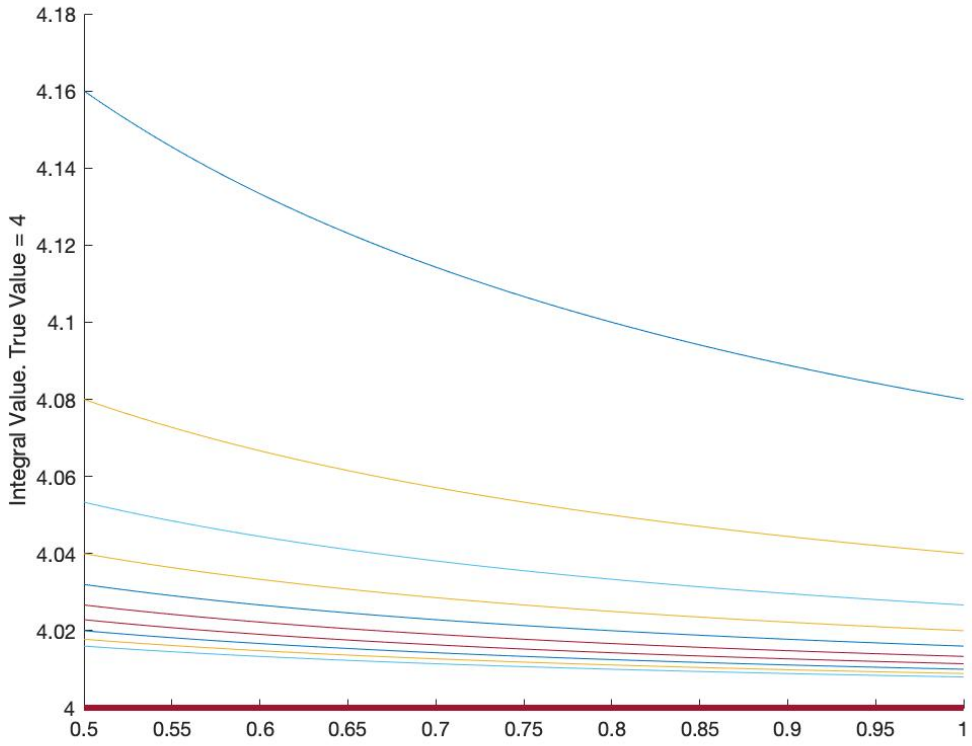
The number of iterations needed is: 37835

Tambien hemos hecho un programa que nos aiuda a visualizar La convergencia de la integral a el valor verdadero, al aumentar de N.

Codigo:

```
%The function that we want to calculate the integral of
f = @(x)x*2;
a = 0;
b = 2;
Real_Int = 4;


%Values of N that we will test the algorithm on
N_test = (100:100:1000);


hold all
for j = 1:length(N_test)
    N = N_test(j);
    for i = 1:N
        N_vec = 1:i;
```

```
        Int(i,j) = IntegralDefinida(f,a,b,i);
        points_X = (1:N)/N;
    end
    Int_ToPlot = Int((round(N/2):end),:);
    Int_ToPlot(Int_ToPlot==0) = nan;
    plot(points_X(round(N/2):end),Int_ToPlot);
end
ylabel('Integral Value. True Value = 4')

plot([0.5 1],[4 4],'LineWidth',3)
hold off
```



Los valores de N utilizados por el grafico son 100:100:1000 y la barra roja donde y=4 es el valor verdarero de la integral.

Ejercicio 4: Opción Call Digital

En este ejercicio vamos obtener una estimación de la Volatilidad Implícita de una Opción Call Digital. Para ello, vamos a emplear tres métodos que nos permitirán hallarla:

1. El Método de la Bisección.
2. El Método de Newton-Raphson.
3. La función "fzero" que no proporciona Matlab.

El método de la Bisección y de Newton-Raphson se basan en que la búsqueda del valor de la volatilidad implícita se puede expresar como la búsqueda de aquel valor de sigma que hace que la diferencia entre el precio calculado con este sigma y el precio real de la opción es cero. Es decir, buscamos el cero de la siguiente función: `:@(sigma)(priceEuropeanOption (S0,r,T,sigma,payoff_Digital)- price);`

Finalmente, veremos cuál de los métodos es más adecuado evaluando factores como el error de la estimación, el número de iteraciones que se han necesitado realizar y, por último, el tiempo que ha necesitado Matlab para obtener los outputs.

Código (Bisección)

```
function [Midpoint, Estimation_Error, NumerOfIterations] = Zero_BisectionMethod (f,a,b)

% Zero_BisectionMethod: Finds the zero af a function given that f(a)*f(b)<0
%
% SYNTAX:
%       [Midpoint, Estimation_Error, NumerOfIterations] = Zero_BisectionMethod (f,a,b)
%
% INPUT:
%      f : Handle to the function that we want the zero of
%     a : lower bound of the interval
%     b : higher bound of the interval
%
% OUTPUT:
%     Midpoint : Implied volatility
%     NumerOfIterations : The number of iterations needed for the convergence
%     Estimation_Error : The error of the estimation
%

% Check if the function gives changes sign at the extremes of the given
% interval

    if f(a) * f(b) > 0
        warning('The function might not have a zero')
        return
    end

Estimation_Error = b-a;
Midpoint = (a+b)/2;
NumerOfIterations = 0;
TOL_ABS = 1e-8;

%Finds a point closer than TOL_ABS to the zero of the given function
while Estimation_Error > TOL_ABS
    NumerOfIterations = NumerOfIterations +1;
    Midpoint = (a+b)/2;
    Estimation_Error = Estimation_Error/2;
    if f(a)*f(Midpoint) < 0
        b = Midpoint;
    else
        a = Midpoint;
    end
end
```

Código (Vega Digital con Derivada Numérica)

```
function vega = vegaDigital(S0,K,r,T,sigma,A)
% vegaDigital: vega de of a Digital europea
%% SYNTAX:
%       vega = vegaDigital(S0,K,r,T,sigma,A)
%
%% INPUT:
%       S0 : Initial value of the underlying asset
%        K : Strike price of the option
%        r : Risk free intrest rate
%        T : Time to expiry
%        sigma : Volatility
%        A : Payout
%% OUTPUT:
%    vega : vega of the Digital Option
%% Example:
% S0 = 100; r = 0.05; K = 90; T = 2; sigma = 0.4; A = 10;
% vegaDigital(S0,K,r,T,sigma,A)


payoff = @(ST)A*(ST>K); % payoff of a Digital option

%Define f as the price of an european option in function of sig (sigma)
%while keeping all the other values constant.
f = @(sig)priceEuropeanOption(S0,r,T,sig,payoff);

% h is the precision of the numerical derivative
% h is set bigger than usual to smooth out the final curve estimated
h = 1e-3;

%Vega calculated as the derivative of the call price with respect to sigma.
vega = numericalDerivative(f,sigma,h);
```

Código (Volatilidad Implícita)

```
function [sigm, nIter, error] = impliedVolatility_WithIterations(fPrice,fVega,price)

% impliedVolatility: Implied volatility of a derivative
%
% SYNTAX:
%       sigma = impliedVolatility_WithIterations(fPrice,fVega,price)
%
% INPUT:
%    fPrice : Handle to the function that gives the price of the derivative
%     fVega : Handle to the function that gives the vega of the derivative
%     price : The price of the derivative
%
% OUTPUT:
%     sigma : Implied volatility
%     nIter : The number of iterations needed for the convergence
%     error : The error of the estimation, as the difference between the
%     real price and the price at the calculated Implied Volatility
%
% EXAMPLE:
%        S0 = 100; K = 90; r = 0.05; T = 2;
%        price = 19.8701;
%        fPrice = @(sigma)(priceEuropeanCall(S0,K,r,T,sigma));
%        fVega  = @(sigma)(vegaEuropeanCall(S0,K,r,T,sigma));
%        ImpliedSigma  = impliedVolatility(fPrice,fVega,price)
%        Price = priceEuropeanCall(S0,K,r,T,impliedSigma)  % should be equal to price
%        Error = Price-price

%Setting the absolute tollerance to power(10,-8)
TOLABS  = 1e-8;

%The maximum number of iterations -> To prevent stuck loop.
MAXITER = 100;

sigm  = 0.3;         % intial estimate    %Seed Value
dSigma = 10*TOLABS;   % enter loop for the first time % the difference between sigma t-1
and sigma t
nIter  = 0;           %Counter of the number of iterations.

%Newton - Raphson implementation
while (nIter < MAXITER && abs(dSigma) > TOLABS) %Always set a max number of iterations.
    nIter = nIter + 1; %Always count the number of iterations.
    dSigma = (fPrice(sigm) - price)/fVega(sigm);
```

```
    sigm = (sigm - dSigma);
end

error = abs(dSigma);
%If the loop reaches the max number of iterations then it sends a warning
%telling that it failed to converge.
if (nIter == MAXITER)
    warning('Newton-Raphson not converged')
end
```

Código (Test Volatilidad Implícita con los tres métodos descritos)

```
%% Estimating the implied volatility of  a Digital Optiin by finding the zero of a
function in an interval
%We try 3 different methods of finding the implied volatility to check
%which one is more efficient and more precise (Bisecion Method - Newton Raphson - Matlab
fzero)
%% Data for the example
S0 = 100; r = 0.1; price = 7.81; K = 90; A = 10; T = 2;
%% Functions needed for the Bisection MEthod implementation
%Payoff function of a digital
payoff_Digital = @(ST)A*(ST > K);
%Function Handle: Price of an digital option as a function of Sigma - the price of that
option.
fPrice_minus_price = @(sigma)(priceEuropeanOption(S0,r,T,sigma,payoff_Digital)- price);
%% Functions needed for the Newton-Raphson implementation
fVega = @(sig)vegaDigital(S0,K,r,T,sig,A);
fPrice_Sigma = @(sigm)priceEuropeanOption(S0,r,T,sigm,payoff_Digital);
%% Bisection method Results
tic;
%Doing the test 50 times for each method to reduce the variance of the Time to
convergence results
for i=1:50
[IV_BM, Error_BM, NIter_BM] = Zero_BisectionMethod(fPrice_minus_price,0,10);
end
time = toc;
disp (['The estimated implied Volatility using the Bisection Method is:
',num2str(IV_BM),newline,'The Error of the estimation is: ',
num2str(Error_BM),newline,'The number of iterations needed until convergence is: ',
num2str(NIter_BM),newline,'The time to convergence is:
',num2str(time),newline,newline]);
%% Newton-Raphson Results
tic;
for i=1:50
[IV_NR , Niter_NR , Error_NR] =
impliedVolatility_WithIterations(fPrice_Sigma,fVega,price);
end
time = toc;
disp(['The estimated implied Volatility using Newton-Raphson is:
',num2str(abs(IV_NR)),newline,'The Error of the estimation is: ',
num2str(Error_NR),newline,'The number of iterations needed until convergence is: ',
num2str(Niter_NR),newline,'The time to convergence is:
',num2str(time),newline,newline]);
%% fzero implementation
tic;
for i=1:50
[IV_fzero,Error_NR] = fzero(fPrice_minus_price,0);
end
time = toc;
disp(['The estimated implied Volatility using fzero is:
',num2str(abs(IV_fzero)),newline,'The Error of the estimation is: ',
num2str(Error_NR),newline,'The time to convergence is: ',num2str(time)]);

%The result obteined with the function fzero is not only more efficient but
%also it returns a smaller error than both the Bisection Method and the
%Newton-Raphson method.
```

En este código se ejecuta 50 veces cada método de búsqueda de la volatilidad implícita para obtener resultados de tiempos de búsqueda menos variables y así poder hacer una comparación más precisa entre ellos.

El Output de este código es el siguiente:

	Bisection Method	Newton-Raphson	Fzero
Estimated Implied Volatility	0.12197	0.12197	0.12197
Error of the estimation	9.3132e-09	2.3166e-10	-4.7384e-12
Number of iterations needed until convergence	30	10	-
Time to convergence	11,5805	70.911	66.965

Por tanto, observamos cómo los tres métodos obtienen el mismo valor para la Volatilidad Implícita obteniendo errores bastante pequeños, siendo el menor error el de la función "fzero" que proporciona Matlab.

En cuanto a eficiencia, los métodos más eficientes son la función "fzero" y el Método de Newton-Raphson ya que obtienen tiempos para la convergencia .

Si comparamos el Método de la Bisección con el Método de Newton-Raphson, nos quedaríamos con el Método de Newton-Raphson debido a que este es más eficiente ya que tarda menos en converger porque su orden de convergencia es cuadrático y, por el contrario, el orden de convergencia del Método de la Bisección es lineal. Además, se observa cómo el Método de la Bisección necesita realizar un mayor número de iteraciones provocando lentitud en la ejecución.

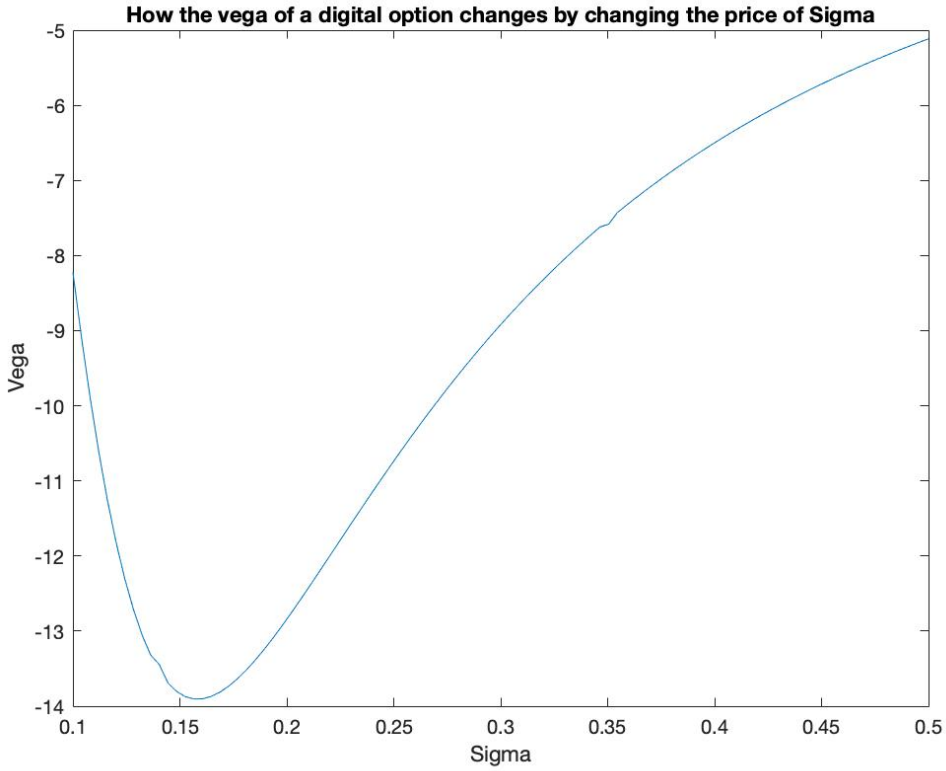
Ejercicio 5: Vega de una Opción

1) Estudio sobre la relación entre Vega y volatilidad del Subyaciente

En este apartado vamos a realizar un código de cara a obtener un gráfico de la Vega de una opción, es decir, de la sensibilidad de una opción ante cambios en la volatilidad del activo subyacente.

Código

```
%% Testing how the Vega changes b changing the volatility in the interval [0.1 0.5]
%When the digital option is in the money volatility is bad, as it increases
%the change of the option getting out of the money --> That's why in this
%example we see that the vega is negative.

S0 = 100; K = 90; r = 0.1; T = 2; A = 10;

Sigmas = linspace(0.1,0.5);

Vegas_2D = vegaDigital(S0,K,r,T,Sigmas,A);
plot(Sigmas,Vegas_2D)
xlabel('Sigma')
ylabel('Vega')
title('How the vega of a digital option changes by changing the price of Sigma')

%% (Extra) See how the vega changes also as a function of the underlying asset price

Prices = linspace(60,110);

Vegas_3D = zeros(100,100);

cumulatedmean = 0;

for price_index=1:100
    tic;
    price = Prices(price_index);
    Vegas_3D(price_index,:) = vegaDigital(price,K,r,T,Sigmas,A);
    disp(['Loading: ',num2str(price_index),'%'])
    time=toc;

    cumulatedmean = (cumulatedmean + time);
    timeleft = (cumulatedmean/price_index)*(100-price_index);
    disp([num2str(timeleft),' Seconds left',newline])
end


figure(2)
plot(Prices,Vegas_3D(:,20))
xlabel('Prices')
ylabel('Vega')
title('How the vega of a digital option changes by changing the price of the underlying
asset')

figure(3)
[x,y] = meshgrid(Sigmas,Prices);
surf(x,y,Vegas_3D)
xlabel('Sigma')
ylabel('Prices')
zlabel('Vegas')

figure(4)
mesh(x,y,Vegas_3D)
xlabel('Sigma')
ylabel('Prices')
zlabel('Vegas')
```



Cuando la Opcion Digital es "*in the money*" la volatilidad es perjudicial ya que un aumento de la volatilidad aumenta la probabilidad de que la opcion pase a encontrarse "*out of the money*". Este es el motivo por el cual en nuestro ejemplo (S0 = 100 , K = 90 → Opción "*in the money"*), la vega es siempre negativa.

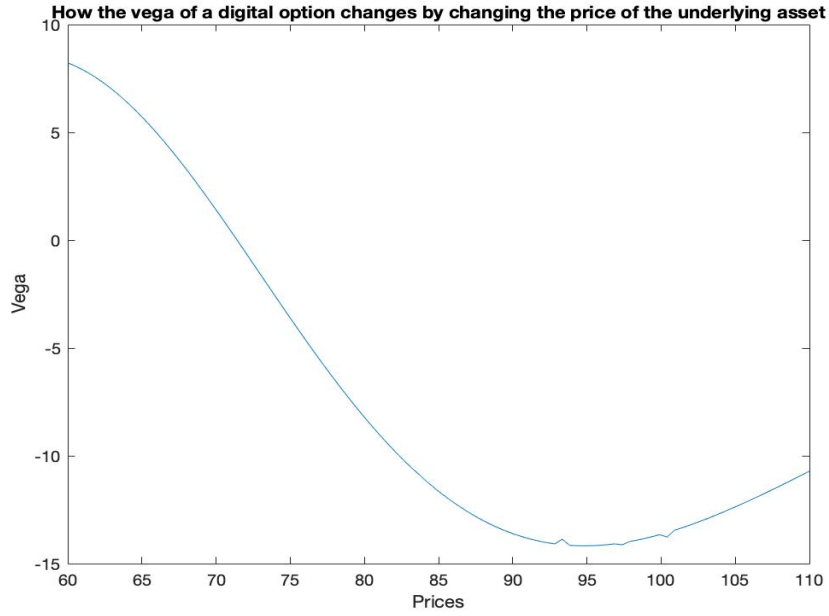
En este gráfico podemos apreciar cómo al disminuir la volatilidad del activo subyacente, la vega pasa a ser más negativa hasta llegar a un valor de aproximadamente 0,15. A partir, de este valor, la Vega comienza a subir pero siempre siendo negativa.

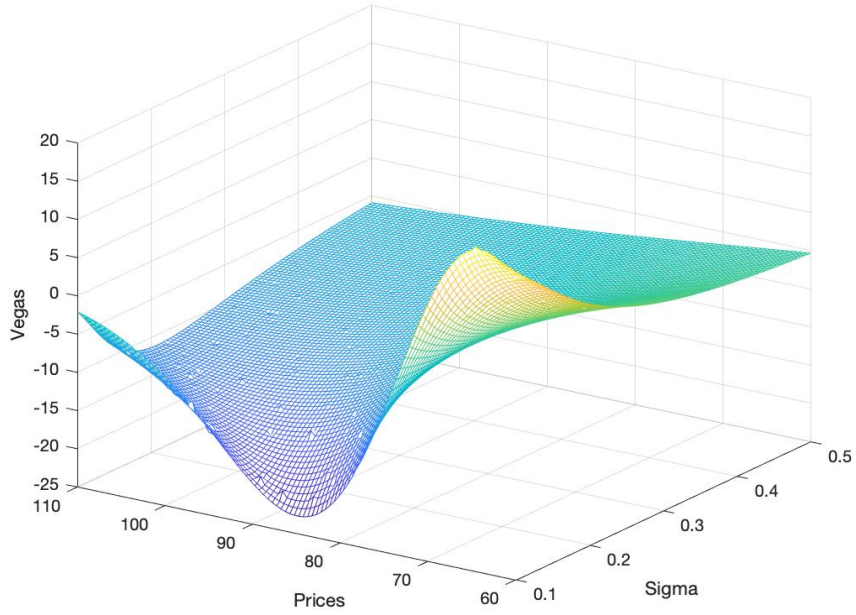
Esto se debe a que variaciones en la volatilidad son más significativas cuando la sigma se hace más pequeña hasta un valor límite (que en este caso es de 0.15), donde la volatilidad es tan pequeña que siguientes disminuciones en la volatilidad no van a ser tan significativas como las precedentes para variar el precio de la opción.

2) (Extra) Cómo varía la Vega al variar del Precio del Subyacente

Como curiosidad vamos a visualizar cómo cambia la Vega de una Opción Digital no sólo ante cambios de la sigma sino también al cambiar del precio del activo subyacente.

A continuación se muestran dos gráficos en dos y tres dimensiones. En el gráfico de dos dimensiones fijamos el valor de la volatilidad y estudiamos cómo cambia la Vega al cambiar del precio del subyacente. En el grafico en 3D observamos cómo varía la vega de la opción al variar la volatilidad pero también al variar el precio del activo subyacente.





Cómo se puede apreciar en estos dos gráficos, cuando la Opción Digital se encuentra muy "*out of the money*", cambios en la volatilidad nos vendrán bien porque subirá la probabilidad de que la opción llegue a estar "*in the money*".

Esto se ve claramente en ambos los gráficos ya que la Vega es positiva cuando la opción está muy "*out of the money*" y es muy negativa cuando la opcion está "*in the money*".

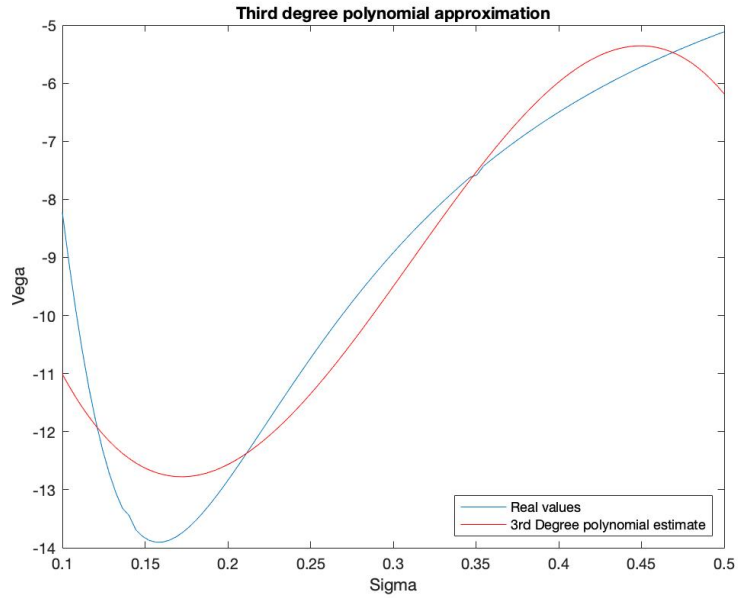
3) Ajuste polinómico para estimar la curva de relación entre vega y sigma.

Vamos a hacer un ajuste polinómico utilizando la función ***polyfit*** de Matlab para aproximar la curva de dependencia con un error relativo menor de 0.01 en el intervalo considerado (es decir, el error relativo máximo será inferiro a ese umbral).

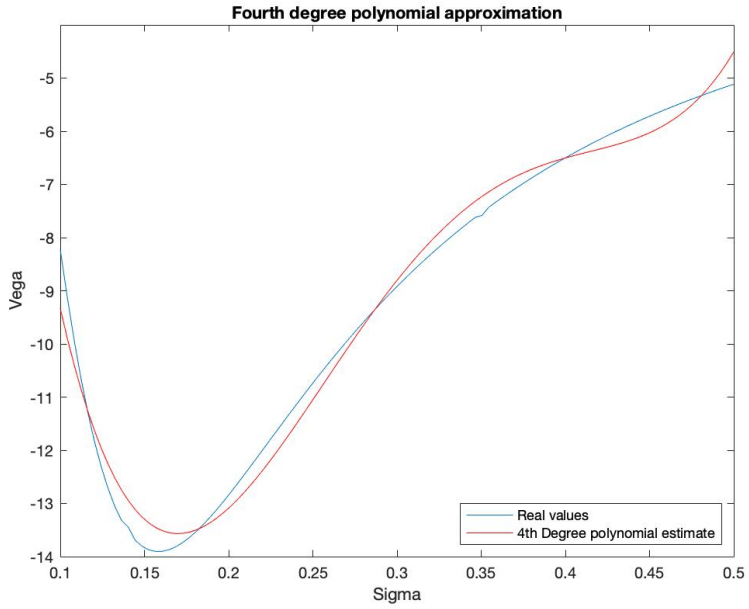
Código

```
%% In this test program we want to find out which degree polinomial can fit the curve
previuosly calculated while keeping an error lower than 0.01
S0 = 100; K = 90; r = 0.1; T = 2; A = 10;
Sigmas = linspace(0.1,0.5);
Vegas_2D = vegaDigital(S0,K,r,T,Sigmas,A);
%The lowest degree polinomial fit that gives back a relative error lower
%than 0.01 is 3, with an error of 0.0051616 (2 degree polinomial gives an
%error of 0.013244).
Polynomial_3 = polyfit(Sigmas,Vegas_2D,3);
Estimate_3 = polyval(Polynomial_3,Sigmas);
Error_3 = sum(abs((Estimate_3-Vegas_2D)/Vegas_2D));
figure(1)
plot(Sigmas,Vegas_2D)
xlabel('Sigma')
ylabel('Vega')
hold on
plot(Sigmas,Estimate_3,'r')
legend('Real values','3rd Degree polynomial estimate','Location','southeast')
title('Third degree polynomial approximation')
disp(['The relative error of the 3rd degree polinomial estimate is:
',num2str(Error_3)])
%% (EXTRA)
%The fourth degree polynomial looks like it fits better the data without
%giving signs of overfitting
figure(2)
plot(Sigmas,Vegas_2D)
xlabel('Sigma')
ylabel('Vega')
hold on
Polynomial_4 = polyfit(Sigmas,Vegas_2D,4);
Estimate_4 = polyval(Polynomial_4,Sigmas);
Error_4 = sum(abs((Estimate_4-Vegas_2D)/Vegas_2D));
plot(Sigmas,Estimate_4,'r')
legend('Real values','4th Degree polynomial estimate','Location','southeast')
title('Fourth degree polynomial approximation')
disp(['The relative error of the 4rd degree polinomial estimate is:
',num2str(Error_4)])
```

Una vez ejecutado el código podemos apreciar que sólo con un polinomio de tercer grado el error relativo es mas pequeño que 0.01. Por tanto, ya de entrada sabemos que el menor grado posible para aproximar esta curva con un error inferior a 0.01 es 3.



Por curiosidad miramos también una aproximaciónde 4 grado a dicha curva:



La aproximación claramente tiene un error inferior al de 3º grado, pero parece que arriba a la derecha la curva aproximada suba de forma más que lineal cuando en realidad parece que la tendencia real (de la línea en azul) posea una tendencia con menos pendiente al alza. Se podría estudiar más esta cuestión investigando si al poner un 4º grado de aproximación en el polinomio encontráramos una situación de "*overfitting*".

El resultado tras ejecutar el código es:

The relative error of the 3rd degree polinomial estimate is: **0.0051616**

The relative error of the 4rd degree polinomial estimate is: **0.0010572**

Ejercicio 6: Movimiento Browniano Aritmético

Apartado I

De cara a simular 500 trayectorias del Proceso Browniano Aritmético, hemos elaborado el siguiente código definiendo cada parámetro y las acciones a realizar.

Código

```
%% (I) Simulate 500 trayectories of a Arithmetic Brownian Motion
%Number of simulations
nsimul=500;

%Number of steps(nsteps)
nsteps=300;

%Parameters of the simulation
mu = 5;
sigma = 0.7;
T = 5;
t0 = 2;
B0 = 100;

%Maturity
Maturity = T-t0;

% Time interval (dt) Vector of timesteps (timestep)
dt=Maturity/nsteps; timestep=(t0:dt:T);

%Brownian motion increases
X = randn(nsimul,nsteps);
dB = mu * dt + sigma * sqrt(dt) * X;

%Set the starting value for all simulations as zero, then set the other
%steps as the cumulative sum of dB.
B=cumsum([B0*ones(nsimul,1) (dB)],2);
```

Apartados II y III

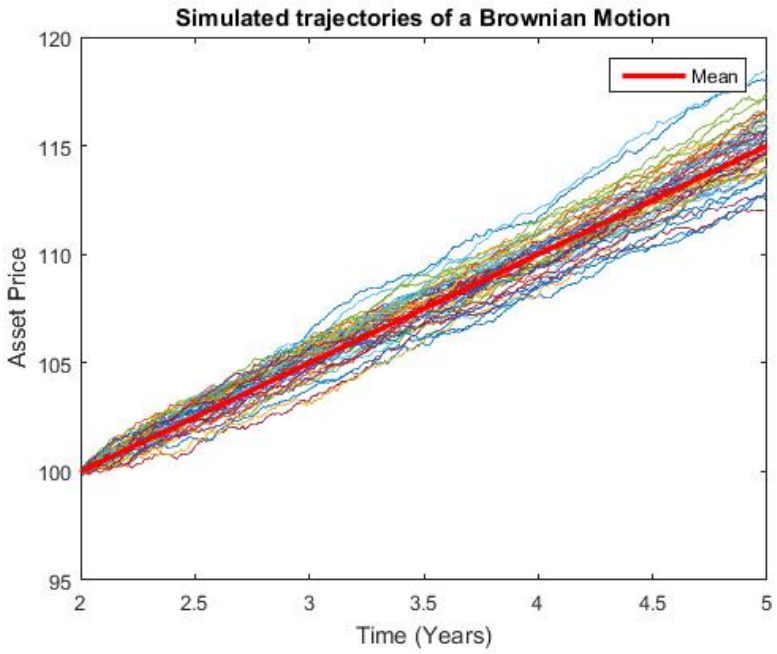
Para mostrar en un mismo gráfico las primeras 50 trayectorias simuladas del Browniano junto con una media teórica que crece linealmente con el tiempo de acuerdo a la siguiente expresión, hemos realizado el siguiente código:

$$\mathbb{E}[B(t)] = B0 + \mu(t - t0)$$

Código:

```
%% (II) Make a Graph simulating 500 trayectories
%Plot simulated paths:
figure(1)
plot(timestep, B(1:50,:)','HandleVisibility','off')
title('Simulated trajectories of a Brownian Motion');
xlabel('Time (Years)')
ylabel('Asset Price')
%% (III) Adding the Mean
cumulate_dmean = cumsum([B0 ones(1,300)*mu*dt]);
hold on
plot(timestep,cumulate_dmean,'r','LineWidth',2.5)
legend('Mean')
```

Una vez ejecutado el código, hemos obtenido el gráfico con las 50 trayectorias deseadas junto con la media descrita previamente.



Apartado IV

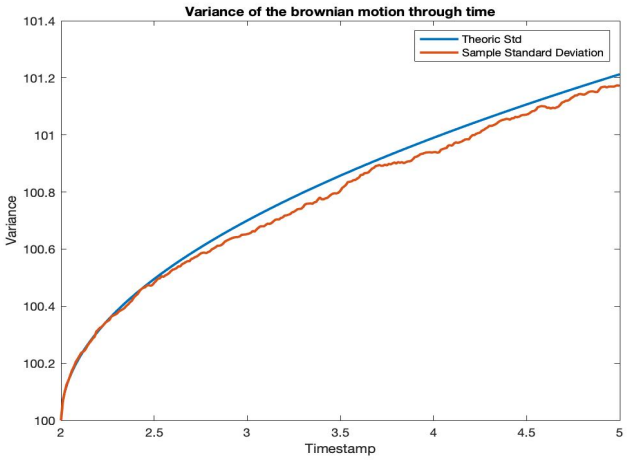
En este apartado buscamos comparar la Desviación Típica Muestral de las trayectorias simuladas con la Desviación Típica Teórica descrita a continuación:

$$std[B(t)] = \sigma\sqrt{t - t0}$$

Código:

```
%% (IV) Plotting the standard deviation
figure(2)
hold on
Variance_vec = sigma*(timestep-2).^0.5;
plot(timestep, Variance_vec +B0,'LineWidth',2 )
title('Variance of the brownian motion trough time')
xlabel('Timestamp')
ylabel('Variance')
plot(timestep, var(B)+ B0,'LineWidth',2)
legend('Theoric Variance','Sample Variance')
hold off
```

Una vez ejecutado el código, hemos obtenido un gráfico en el que aparece la Desviación Típica Muestral (en rojo) de las trayectorias simuladas y la Desviación Típica Teórica (en azul). Como es lógico, ambas desviaciones típicas crecen con el paso del tiempo ya que la desviación del proceso es acumulativa con el avance del tiempo. Se puede apreciar cómo la Desviación Típica de las simulaciones se adapta bien a la Desviación Típica Teórica.



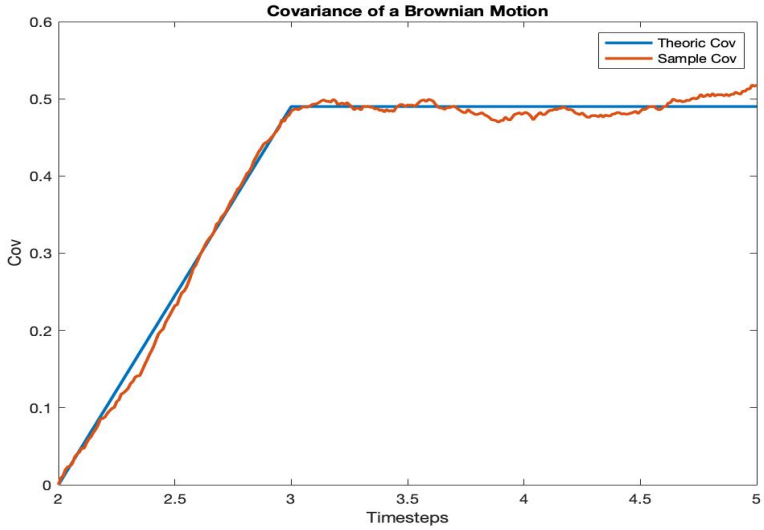
Apartado V

En este apartado vamos a ilustrar que la autocovarianza del proceso posee la siguiente forma:

$$cov[B(t), B(t')] = \sigma\, 2\, \min(t - t0, t' - t0)$$

Código

```
%% (V) Plotting the Autocovariance
t1 = 3;
cov = (sigma.^2)*min(timestep-t0,t1-t0);
figure(3)
plot(timestep,cov,'LineWidth',2)
title('Covariance of a Brownian Motion')
xlabel('Timesteps')
ylabel('Cov')
hold on
%Covariance formula for t1 = 3
B_t1 = B(:,101); %the value of B where t = 3
% Covariance Formula: Sum((X_i - E(X)(Y_i - E(Y))/N
Cov_vec = ((B - mean(B,1))'*(B_t1 - mean(B_t1)))/nsimul;
plot(timestep,Cov_vec,'LineWidth',2)
legend('Theoric Cov','Sample Cov')
```



Una vez ejecutado el código obtenemos este gráfico como Output. En él se puede observar cómo la Autocovarianza Teórica y la Autocovarianza Muestral del proceso son bastante similares.

Apartado VI

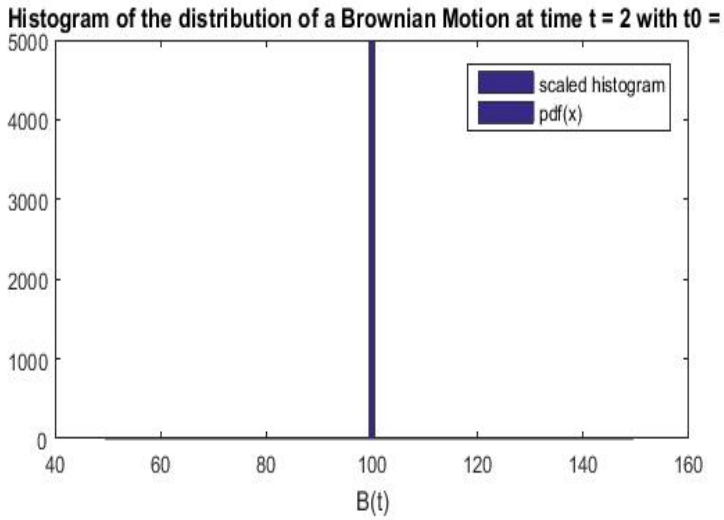
Finalmente, vamos a mostrar histogramas del proceso para t = 2, t = 3, t = 4 y para t = 5 que ilustren que el Browniano aritmético sigue la siguiente Distribución Normal:

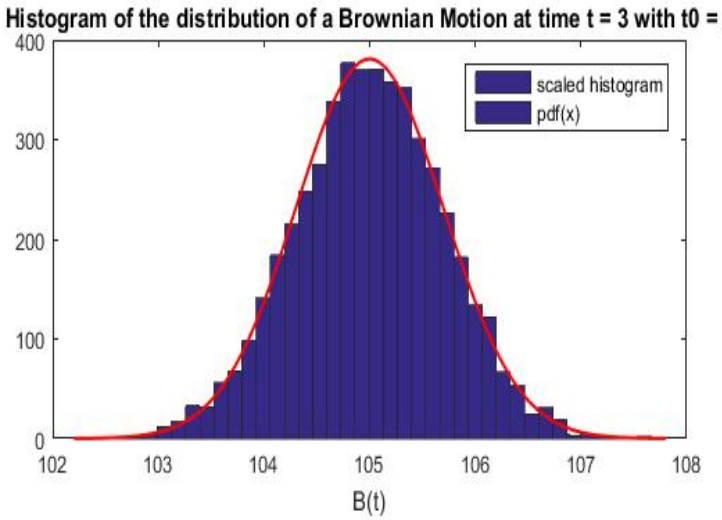
$$B(t) \sim N(B0 + \mu(t - t0\ ), \sigma\sqrt{t - t0})$$

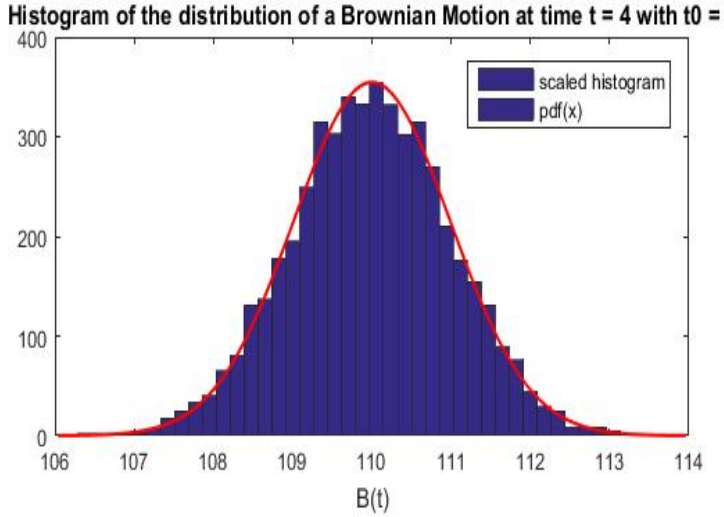
Código:

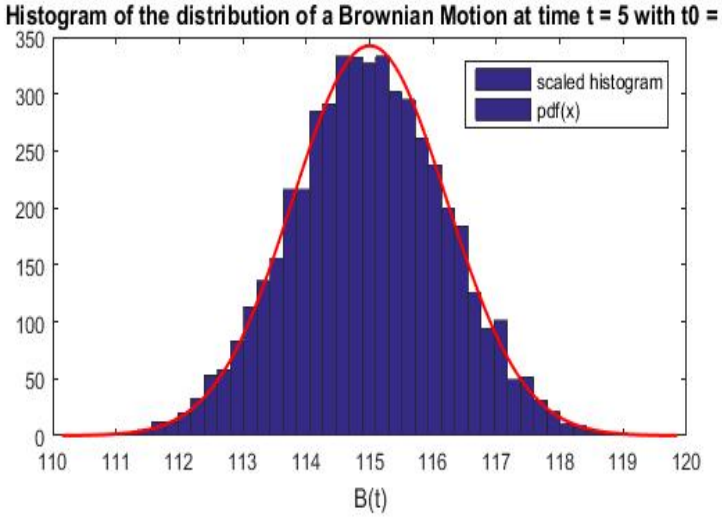
```
%% (VI)  Normal Distribution Brownian Motion
%Changing the number of simulations so that the histogram is more precise
nsimul=5000;
X = randn(nsimul,nsteps);
dB = mu * dt + sigma * sqrt(dt) * X;
B=cumsum([B0*ones(nsimul,1) (dB)],2);
%Parameters Histogram
t_ref = 2:5;
bins = 100;
BValues_vec = 1:100:301;

% Parameters Grafical Comparison
scale = 0;

figure(4)

for i=1:length(t_ref)
    t_var = t_ref(i);
    BrownianValues =  B(:,BValues_vec(i));
    subplot(2,2,i)
    hist(BrownianValues,bins);
    str = sprintf('Histogram of the distribution of a Brownian Motion at time t = %d
with t0 = 2', t_var);

    title(str)
    xlabel('B(t)')


    %Adding the theorical normal distribution over the plot

    hold on
    center = mu*(t_ref(i)-t0);
    radius = 4*sigma*sqrt(t_ref(i)-t0);
    B_min = B0 + min(center - radius) ;
    B_max = B0 + max(center + radius);
    points = B_min:0.01:B_max;
    mean_B = B0 + mu*(t_ref(i)-t0);
    std_B = sigma * sqrt(t_ref(i) - t0);
    modelPdf = @(x)normpdf(x,mean_B,std_B);
    graphicalComparisonPdf(B(:,BValues_vec(i)),modelPdf,scale,B_min,B_max)
    legend('scaled histogram','pdf(x)')
end
```

Una vez ejecutado el código, estos son los histogramas que hemos obtenido que nos permiten ver gráficamente cómo el Browniano efectivamente sigue una Distribución Normal ya que el histograma se ajusta de manera satisfactoria a la Función de Densidad de una Normal.


Histogram of the distribution of a Brownian Motion at time t = 2 with t0 = 2
scaled histogram
pdf(x)
B(t)


Histogram of the distribution of a Brownian Motion at time t = 3 with t0 = 2
scaled histogram
pdf(x)
B(t)


Histogram of the distribution of a Brownian Motion at time t = 4 with t0 = 2
scaled histogram
pdf(x)
B(t)


Histogram of the distribution of a Brownian Motion at time t = 5 with t0 = 2
scaled histogram
pdf(x)
B(t)

Apartado VII

Cuando t tiende a t0 observamos cómo la Varianza tiende a 0, ya que en t0 el Browniano no habrá avanzado en ninguna dirección y mantendrá su posición inicial (En nuestro caso, B(t0) = 100). Por tanto, toda la densidad de probabilidad se encontrará en un sólo punto, en B(t0) = 100 que coincidirá con el valor de la media del proceso en ese instante. A este fenómeno se le conoce como Delta de Dirac.

Como se puede observar en el primer gráfico del Apartado VI, hemos fijado t = 2 como el instante t0, es decir, como posición inicial del Browniano.

Ejercicio 7 : Movimiento Browniano Geométrico

Apartado I

De cara a simular 500 trayectorias del Proceso Browniano Geométrico, hemos elaborado el siguiente código definiendo cada parámetro y las acciones a realizar.

Código:

```
%% (I) Simulate 500 trayectories of a Geometric Brownian Motion
%Number of simulations (nsimul)
nsimul=500;

% Number of steps(nsteps)
nsteps=300;

%Parameters of the simulation
mu = 0.15;
sigma = 0.1;
T = 14;
t0 = 2;
S0 = 100;

%Maturity
Maturity = T-t0;

% Time interval (dt) Vector of timesteps (timestep)
dt=Maturity/nsteps; timestep=(t0:dt:T);

%Brownian motion increases
X = randn(nsimul,nsteps);
dB = exp((mu-0.5*sigma^2)*dt+sigma*sqrt(dt)*X);

%Set the starting value for all simulations as zero, then set the other
%steps as the cumulative sum of dB.

B=cumprod([S0*ones(nsimul,1) dB],2);
```

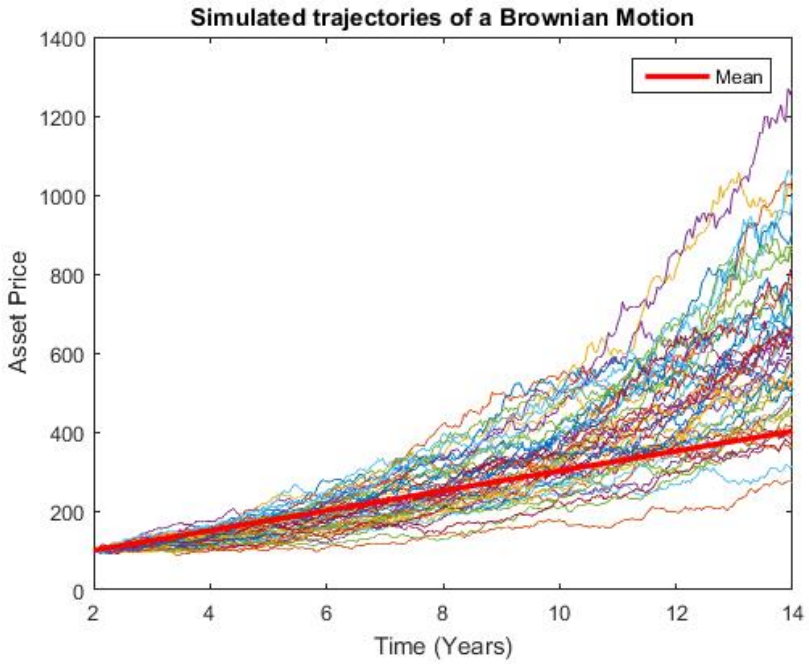
Apartados II y III

Para mostrar en un mismo gráfico las primeras 50 trayectorias simuladas del Browniano junto con una media teórica que crece exponencialmente con el tiempo de acuerdo a la siguiente expresión, hemos realizado el siguiente código:

$$\mathbb{E}[S(t)] = S0e\ [\mu(t-t0)]$$

Código:

```
%% (II) Make a Graph simulating 50 trayectories
%Plot simulated paths:
figure(1)
plot(timestep, B(1:50,:)','HandleVisibility','off')
title('Simulated trajectories of a Brownian Motion');
xlabel('Time (Years)')
ylabel('Asset Price')

%% (III) Adding the Mean
cumulate_dmean = cumsum([S0 ones(1,300)*exp(mu*dt)]);
hold on
plot(timestep,cumulate_dmean,'r','LineWidth',2.5)
legend('Mean')
```



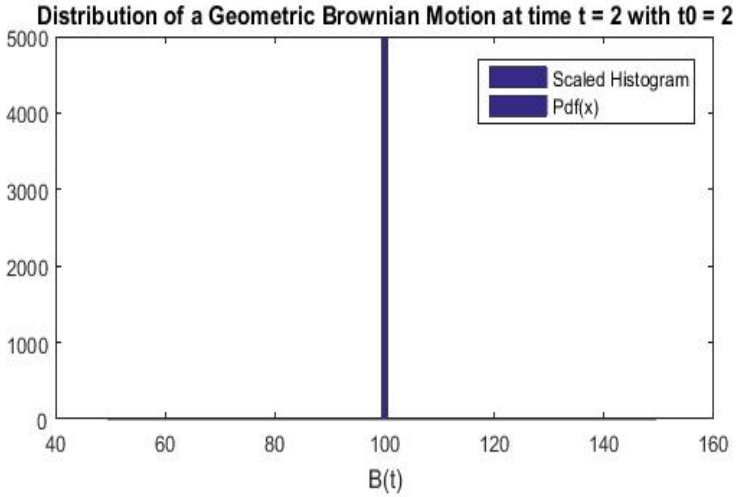
Apartado IV

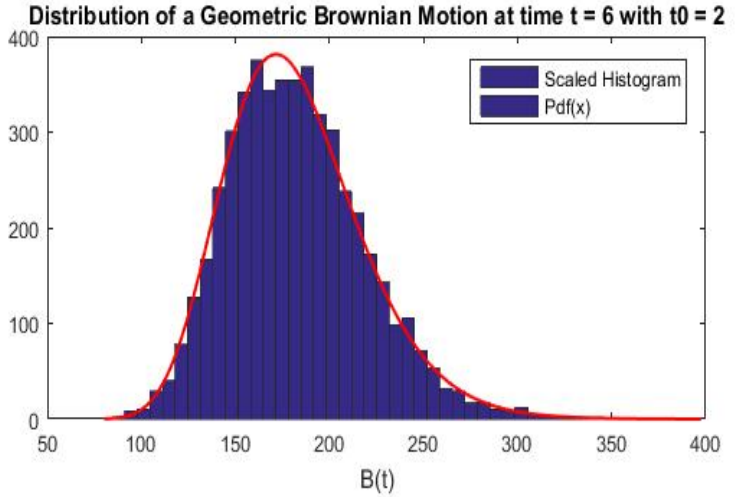
Finalmente, vamos a mostrar histogramas del proceso estocástico para $t = 2$, $t = 6$, $t = 10$ y para $t = 14$ que ilustren que el Movimiento Browniano Geométrico sigue una Distribución LogNormal, es decir:

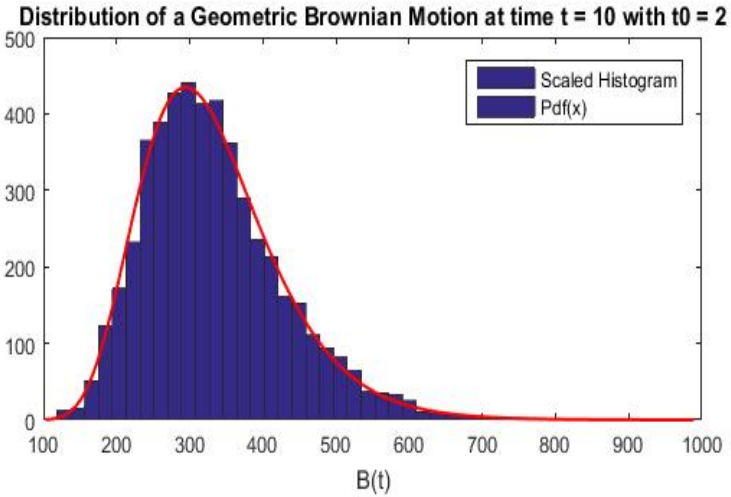
$$S(t) \sim LN\left(\log(S0) + (\mu - \sigma^2/2)(t - t0), \sigma\sqrt{t - t0}\right)$$

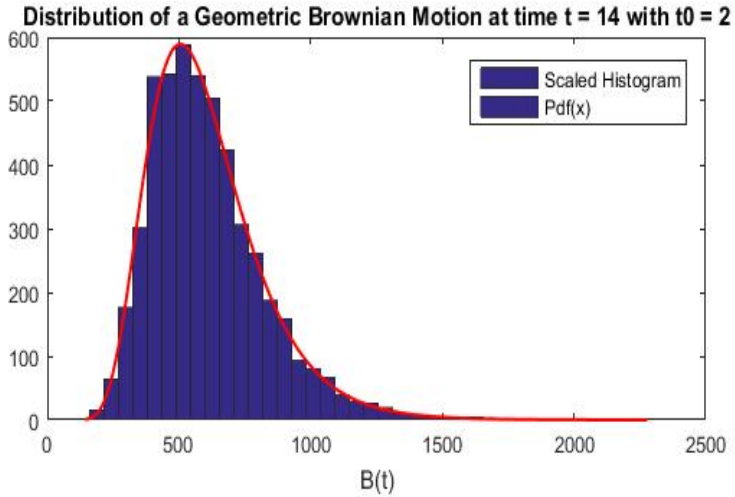
Código:

```
%% (IV)  Normal Distribution Brownian Motion
%Changing the number of simulations so that the histogram is more precise
nsimul=5000;
X = randn(nsimul,nsteps);
dB = exp((mu-0.5*sigma^2)*dt+sigma*sqrt(dt)*X);
B=cumprod([S0*ones(nsimul,1) dB],2);

%Parameters Histogram
t_ref = [2 6 10 14];
bins = 100;
BValues_vec = 1:100:301;

% Parameters Grafical comparison
scale = 0;

figure(4)
for i=1:length(t_ref)
    t_var = t_ref(i);
    BrownianValues =  B(:,BValues_vec(i));
    subplot(2,2,i)
    hist(BrownianValues,bins);
    str = sprintf('Distribution of a Geometric Brownian Motion at time t = %d with t0 =
2', t_var);
    title(str)
    xlabel('B(t)')

    %Adding the theorical normal distribution over the plot
    hold on
    center = (mu-0.5*sigma^2)*(t_ref(i)-t0);
    radius = 4.0*sigma*sqrt(t_ref(i)-t0);
    S_min = S0*exp(min(center - radius));
    S_max = S0*exp(max(center + radius));
    points = S_min:0.01:S_max;
    mean_B = log(S0) + (mu - 0.5*sigma^2)*(t_ref(i)-t0);
    std_B = sigma * sqrt(t_ref(i) - t0);
    modelPdf = @(x)lognpdf(x,mean_B,std_B);
    graphicalComparisonPdf(B(:,BValues_vec(i)),modelPdf,scale,S_min,S_max)
    legend('Scaled Histogram','Pdf(x)')
end
```


Distribution of a Geometric Brownian Motion at time t = 2 with t0 = 2
Scaled Histogram
Pdf(x)
B(t)


Distribution of a Geometric Brownian Motion at time t = 6 with t0 = 2
Scaled Histogram
Pdf(x)
B(t)


Distribution of a Geometric Brownian Motion at time t = 10 with t0 = 2
Scaled Histogram
Pdf(x)
B(t)


Distribution of a Geometric Brownian Motion at time t = 14 with t0 = 2
Scaled Histogram
Pdf(x)
B(t)

Apartado V

Cuando t tiende a t0 observamos cómo la Varianza tiende a 0, ya que en t0 el Browniano no habrá avanzado en ninguna dirección y mantendrá su posición inicial (En nuestro caso, B(t0) = 100). Por tanto, toda la densidad de probabilidad se encontrará en un sólo punto, en B(t0) = 100 que coincidirá con el valor de la media del proceso en ese instante. A este fenómeno se le conoce como Delta de Dirac.

Como se puede observar en el primer gráfico del Apartado IV, hemos fijado t = 2 como el instante t0, es decir, como posición inicial del Browniano.

Ejercicicio 8: Proceso Autorregresivo AR(1)

Consideremos el siguiente proceso que genera una serie temporal AR(1). Los parámetros que caracterizan el proceso son ϕ_1, ϕ_2 y σ. Siendo el valor inicial de la serie temporal es X_0, los valores sucesivos de la serie temporal se obtienen de la siguiente forma:

$$X_1 = \phi_0 + \phi_1 X_0 + u_1; \qquad u_1 = \sigma\,\varepsilon_1$$
$$X_2 = \phi_0 + \phi_1 X_1 + u_2; \qquad u_2 = \sigma\,\varepsilon_2$$
$$X_3 = \phi_0 + \phi_1 X_2 + u_3; \qquad u_3 = \sigma\,\varepsilon_3$$
$$\ldots$$
$$X_T = \phi_0 + \phi_1 X_{T-1} + u_T; \qquad u_T = \sigma\,\varepsilon_T$$

Donde $u1$, $u2$, $u3$, … son las innovaciones del proceso y, $\epsilon 1$, $\epsilon 2$, $\epsilon 3$, … son números aleatorios generados a partir de una Distribución Normal (0,1). En este ejercicio vamos a escribir una función en Matlab que devuelva una matriz con M trayectorias como filas y T columnas como momentos del tiempo para ver qué valores toman cada una de las trayectorias simuladas en cualquier momento del periodo considerado.

El código realizado es el siguiente:

Código

```
function [X,u] = simAR1(M,T, phi0,phi1,sigma,X0)
% simAR1 : Simulacion de un proceso AR(1)
%
% Sintaxis:
% X = simAR1(M,T,phi0,phi1,sigma,X0)
%
% X : Matriz (M,T) que contiene los M trayectorias simuladas
% M : Numero de trayectorias simuladas
% T : Tiempos para la simulacion
% X0 : Valor inicial de la serie temporal
% phi0,phi1,sigma : Parametros que caracterizan el proceso AR(1)

%We want a Matrix X=(MxT)

Xt = [X0*ones(M,1) zeros(M,T)];%We fix a dimension for X Matrix and take X0 as origin of
all the simulations.

u = sigma*randn(M,T); %u are the innovations of the AR(1) process.

for m=1:M %From the 1° trayectory to the M trayectory.
    for t= 2:T+1 %From 2 until T + 1 -> Which are in total T "times" for simulation.
        Xt(m,t) = phi0 + phi1*Xt(m,t-1)+u(m,t-1); %X value of the AR(1) process.
```

```
    end
end

% We erase the first observation of every row (Because this is X0).
X = Xt(:,2:end);
```

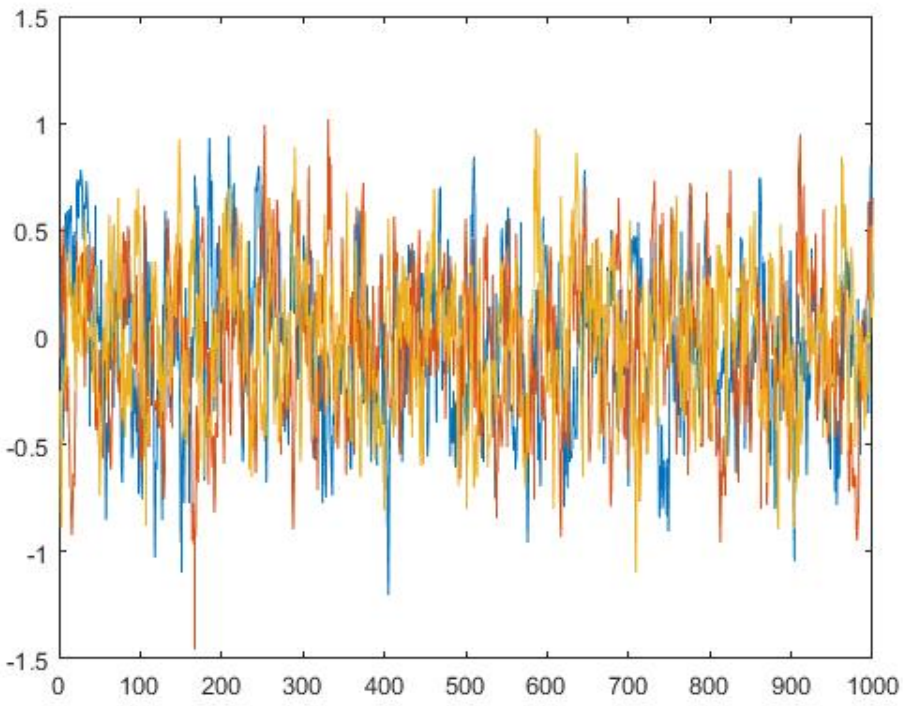
Ejemplo 1

M = 3, T = 1000, X0 = 0, phi0 = 0.0, phi1 = 0.7, sigma = 0.25

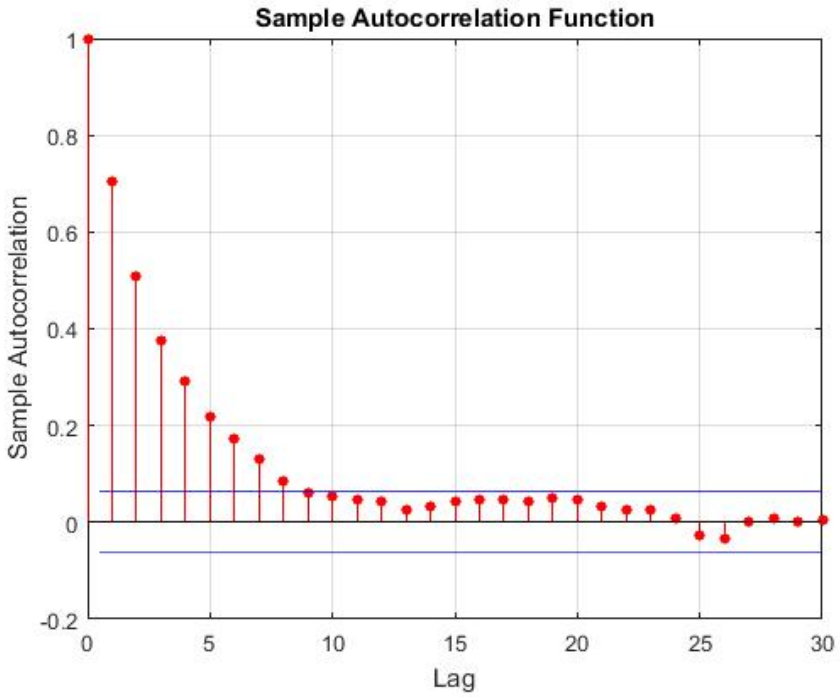
Código:

```
% [X,u] = simAR1(M,T,phi0,phi1,sigma,X0);
% figure(1); plot(1:T,X(:,1:T));
% 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);
```

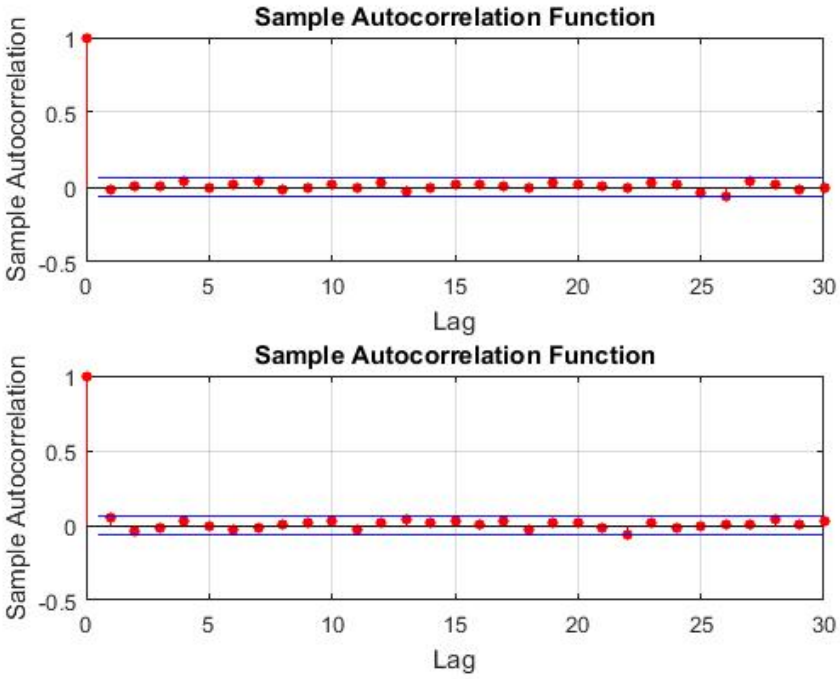
Una vez ejecutado el código para estos datos, hemos obtenido los siguientes gráficos:



En este primer gráfico observamos 3 distintas trayectorias del proceso autorregresivo con un retardo. Se puede apreciar cómo el proceso se sitúa en una banda horizontal (entre 1 y -1) y escasas veces sale de ésta.



En este segundo gráfico podemos ver cómo en el proceso AR(1) la variable aleatoria X va perdiendo autocorrelación a medida que el retardo es mayor ya que ϕ_1 es menor que 1. Además, vemos cómo a partir del *lag* 8 el proceso deja de tener autocorrelación significativa



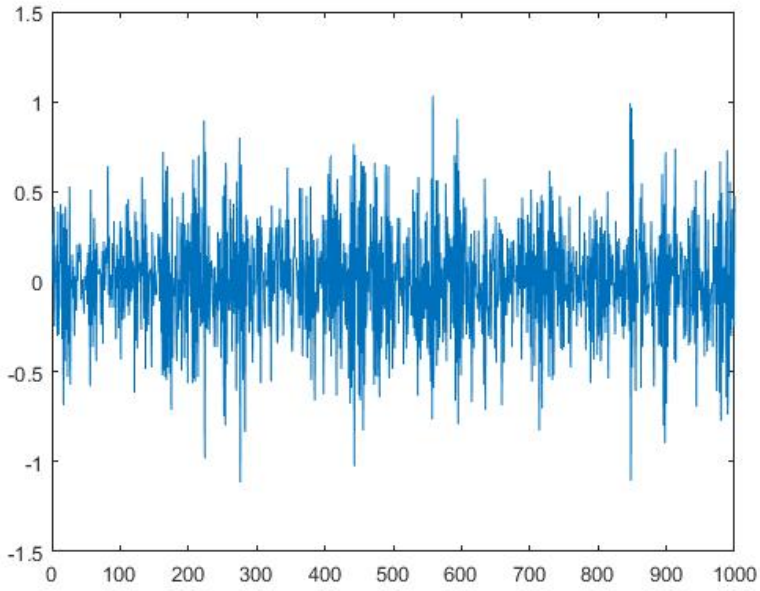
En este último gráfico podemos apreciar la autocorrelación existente en las innovaciones del proceso (u, arriba) y la autocorrelación presente en los valores absolutos de estas mismas (abs(u), abajo). Dado que las innovaciones son iguales al producto de sigma con un vector de variables independientes e idénticamente distribuidas (iid), se puede observar cómo la autocorrelación es inexistente.

Ejemplo 2

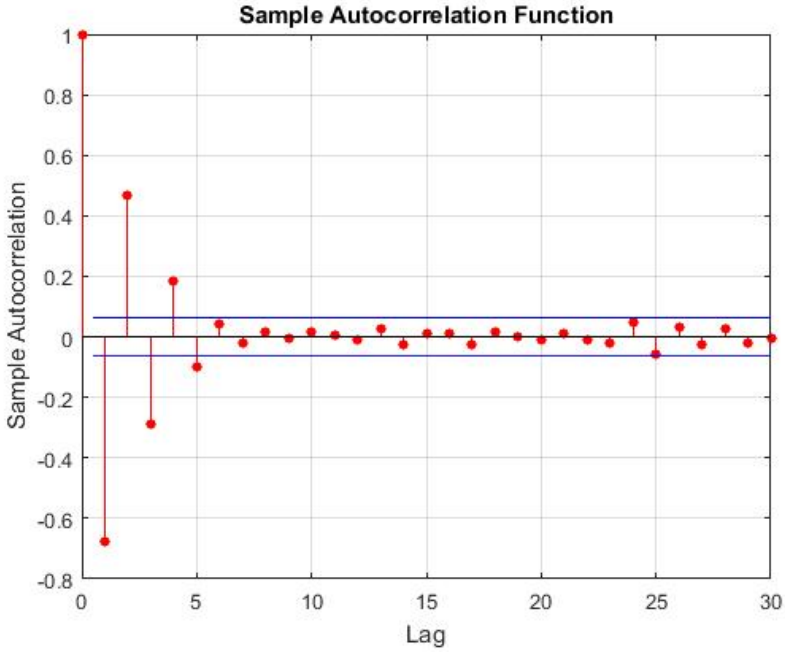
M = 1, T = 1000, X0 = 0, phi0 = 0.0, phi1 = -0.7, sigma = 0.25

Código:

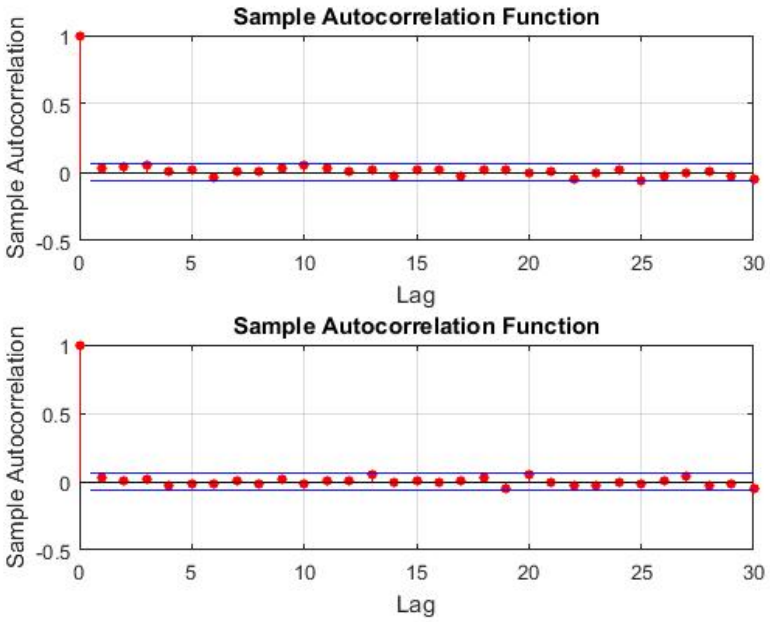
```
% [X,u] = simAR1(M,T,phi0,phi1,sigma,X0);
% figure(1); plot(1:T,X);
% figure(2); autocorr(X(1,:),30);
% figure(3); subplot(2,1,1); autocorr(u(1,:),30);
                % subplot(2,1,2); autocorr(abs(u(1,:)),30);
```



En este primer gráfico observamos una única trayectoria del proceso autorregresivo con un retardo. Se puede apreciar cómo el proceso se sitúa en una banda horizontal (entre 1 y -1) y escasas veces sale de ésta.



En este segundo gráfico podemos ver cómo en el proceso AR(1) la variable aleatoria X va perdiendo autocorrelación a medida que el retardo es mayor ya que ϕ_1 es menor que 1. Además, vemos cómo el proceso tiene autocorrelación negativa con los valores de X de orden impar debido a que el signo de ϕ_1 es negativo.



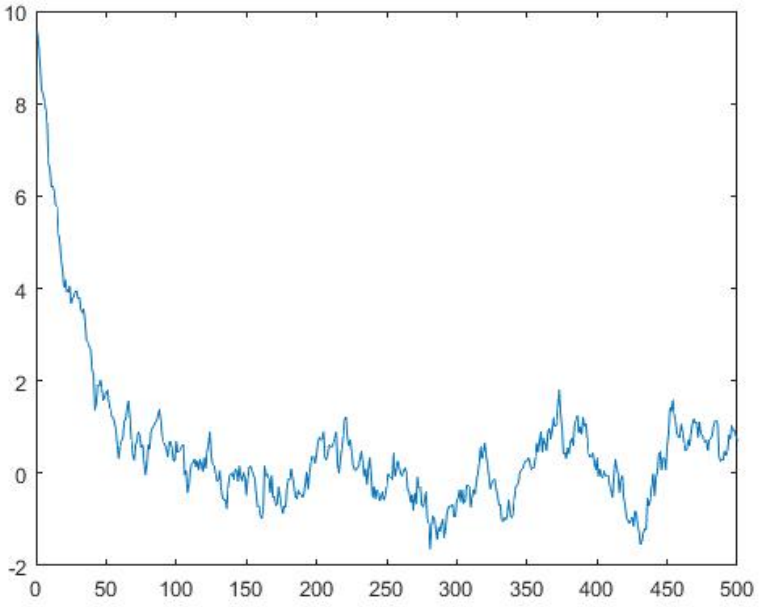
En este último gráfico podemos apreciar la autocorrelación existente en las innovaciones del proceso (u, arriba) y la autocorrelación presente en los valores absolutos de estas mismas (abs(u), abajo). Dado que las innovaciones son iguales al producto de sigma con un vector de variables independientes e idénticamente distribuidas (iid), se puede observar cómo la autocorrelación es inexistente.

Ejemplo 3

M = 1, T = 500, X0 = 10, phi0 = 0.0 phi1 = 0.95, sigma = 0.25

Código:

```
% [X,u] = simAR1(M,T,phi0,phi1,sigma,X0);
% figure(1); plot(1:T,X);
```



En este gráfico podemos apreciar cómo el proceso parte de un valor de X = 10 y, rápidamente, desciende a una franja de valores compuesta entre los valores -2 y 2. Esto se debe a que $\phi_0 = 0$ y ϕ_1 es menor que 1 por lo que el proceso tenderá a variar en el entorno del 0.

Ejercicio 9: ARMA(p,q) + GARCH(r,s)

De cara a elaborar el código hemos divido éste en dos diferentes funciones:

- ARMAGARCH : Calcula un único proceso ARMA con Volatilidad GARCH.
- simARMAGARCH: tiene también un parámetro M que es el número de trayectorias que queremos simular.

```
function [X,u,h] = ARMAGARCH(T,phi0,phi,theta,kappa,alpha,beta,X0,u0,h0)

%% Defining variables and checking if the conditions are met
r = length(alpha);
s = length(beta);
p = length(phi);
q = length(theta);
if r > T
    error('r and cannot be bigger than T (Check size of alpha)')
elseif s > T
    error('s and cannot be bigger than T (Check size of beta)')
elseif p > T
    error('p and cannot be bigger than T (Check size of phi)')
elseif q > T
    error('q and cannot be bigger than T (Check size of theta)')
end

%If we are using a GARCH(1,1) check that Alpha + Beta < 1 , because if not
%the model would have zero or negative coefficient of long-term variance
if r == 1 && s==1 && alpha + beta >=1
    error('In GARCH(1,1) alpha + beta should be less than 1')
end

biggest_lag = max([r s p q]);

%% Implementation

X_vec = [X0*ones(1,biggest_lag) zeros(1,T)];
h_vec = [h0*ones(1,biggest_lag) zeros(1,T)];
u_vec = [u0*ones(1,biggest_lag) zeros(1,T)];
Eps = randn(1,T+biggest_lag);
for time=biggest_lag+1:T+biggest_lag
    %GARCH Part
    u_lags_r = u_vec(time-r:time-1); %The vector with r lags of u from t-1 to t-r
    h_lags = h_vec(time-s:time-1);
    h_vec(time) = kappa + sum((u_lags_r.^2).*alpha) + sum(h_lags.*beta);
    u_vec(time) = sqrt(h_vec(time))*Eps(time-1);


    %ARMA Part
    X_lags = X_vec(time-p:time-1);
    u_lags_q = u_vec(time-q:time-1);
    X_vec(time) = phi0 + sum(phi.*X_lags) + sum(theta.*u_lags_q) + u_vec(time);
end

u = u_vec(biggest_lag+1:end);
h = h_vec(biggest_lag+1:end);
X = X_vec(biggest_lag+1:end);



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

% Sintaxis:
% X = simARMAGARCH(M,T,phi0,phi,theta,kappa,alpha,beta,X0,u0,h0)

% X : Matriz (M,T) que contiene las M trayectorias simuladas
% M : Numero de trayectorias simuladas
% T : Pasos para la simulacion
```

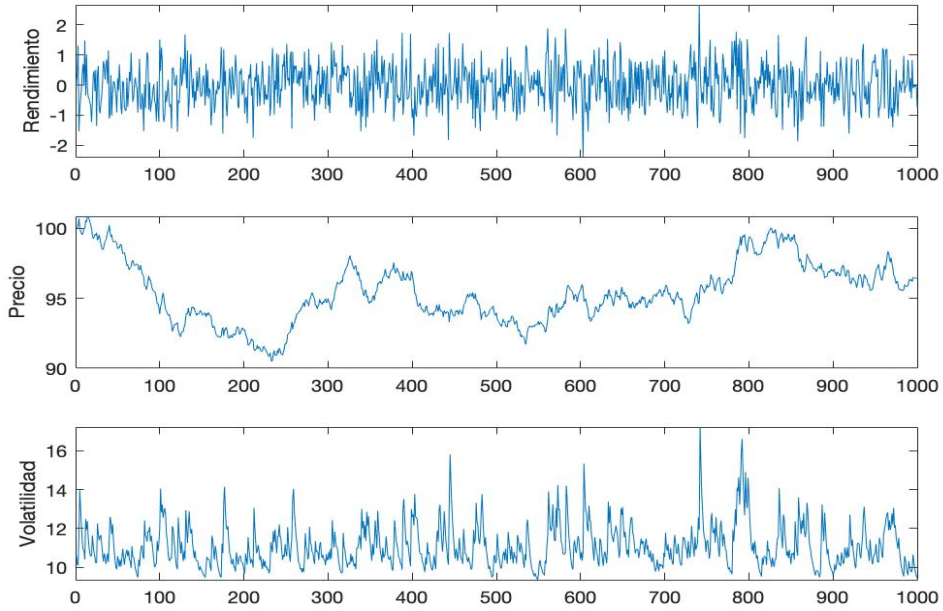
```
% phi0,phi,theta : Parametros que caracterizan el proceso ARMA(p,q)
% kappa,alpha,beta : Parametros que caracterizan el proceso GARCH(r,s)
% X0 : Valores iniciales de la serie temporal
% u0 : Valohelpres iniciales de las innovaciones
% h0 : Valores iniciales de la volatilidad

r = length(alpha);
s = length(beta);

%% En caso de que no se proporcionen condiciones iniciales, se deben generar
simulaciones a partir de condiciones iniciales razonables calculadas de manera
automatica.

% Set the intitial X0 to 0 if not specified otherwise
if ~exist('X0','var')
      X0 = zeros(1,M);
end
% Set the intitial u0 to 0 if not specified otherwise
if ~exist('u0','var')
    u0 = zeros(1,M);
end
% Set the intitial h0 to sigma if not specified otherwise and the model
% does not have GARCH volatility or to
% (kappa/(1-sum(alpha)-sum(beta)))*ones(1,M) if the model has GARCH
% volatility.
if ~exist('h0','var')
    if r == 0 && s == 0
        h0 = sigma*ones(1,M);
    else
        h0 = (kappa/(1-sum(alpha)-sum(beta)))*ones(1,M);
    end
end

%% Implementation
X = zeros(M,T);
u = zeros(M,T);
h = zeros(M,T);

for numsim=1:M
    [X(numsim,:), u(numsim,:), h(numsim,:)] =
ARMAGARCH(T,phi0,phi,theta,kappa,alpha,beta,X0(numsim),u0(numsim),h0(numsim));
end
```

Ejemplo Profesor: Simulacion AR(1) + GARCH(1,1)

M = 1, T = 1000

```
% phi0  = 0;
% phi   = 0.15; theta = [] ; % AR(1)
% kappa = 0.1;  alpha = 0.1; beta = 0.7; % GARCH(1,1)
% [X,u,h] = simARMAGARCH(M,T,phi0,phi,theta,kappa,alpha,beta);
% 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);
```

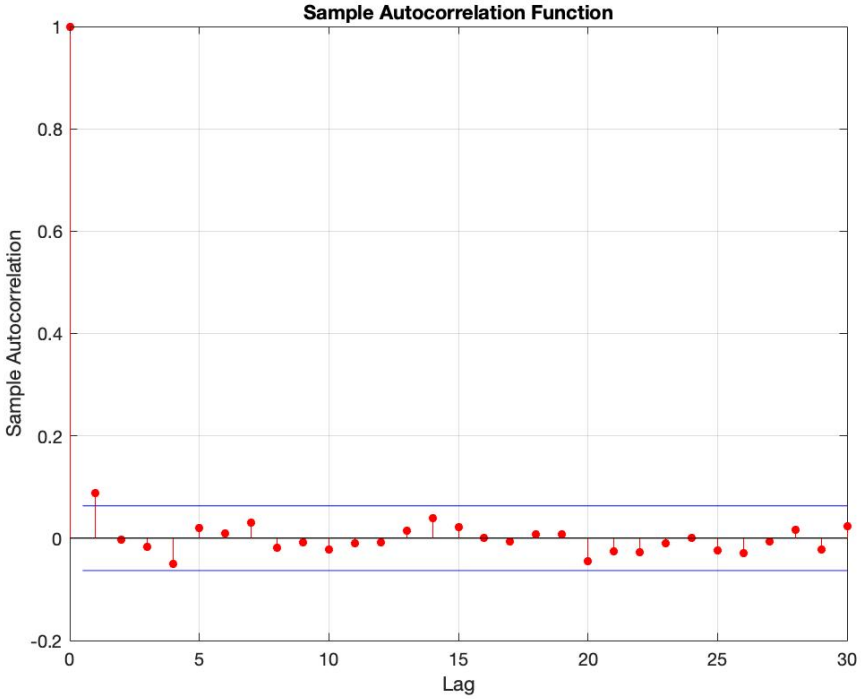
Una vez ejecutado el código para este primer ejemplo hemos obtenido 3 gráficos distintos en los que se evalúan diferentes características del proceso:



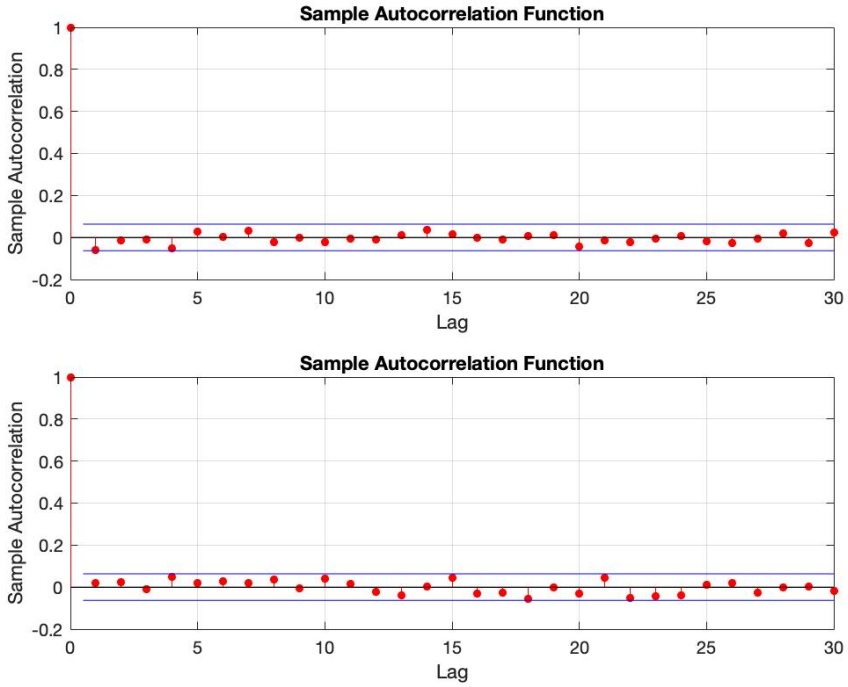
1.- Serie temporal de la evolución de 1 trayectoria de rendimientos en el tiempo.

2.- Serie temporal de la evolución de 1 trayectoria de precios en el tiempo que parten del valor 100.

3.- Serie temporal que muestra cómo evoluciona la volatilidad de la trayectoria en el tiempo. Se puede apreciar cómo la volatilidad es más volatil que los precios.



En este segundo gráfico podemos ver cómo en el proceso AR(1) la variable aleatoria X va perdiendo autocorrelación muy rápidamente dado que ϕ_1 es muy pequeño (ϕ_1= 0,15). Además, vemos cómo a partir del *lag* 2 el proceso deja de tener autocorrelación significativa.



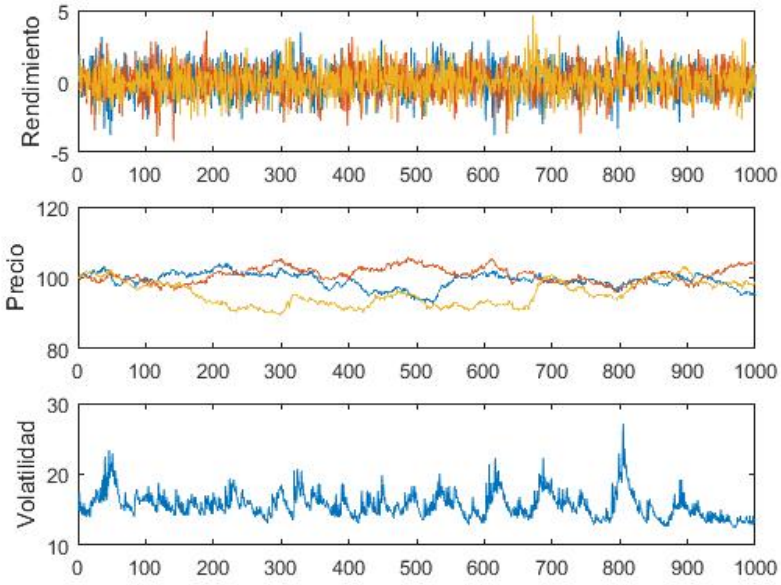
En este último gráfico podemos apreciar la autocorrelación existente en las innovaciones del proceso (u, arriba) y la autocorrelación presente en los valores absolutos de estas mismas (abs(u), abajo). Dado que las innovaciones son iguales al producto de sigma con un vector de variables independientes e idénticamente distribuidas (iid), se puede observar cómo la autocorrelación es inexistente.

Ejemplo 1: Simulacion ARMA(2,3) + GARCH(1,2)

Datos en el Código:

```
% Ejemplo 1: Simulacion 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(d,1);
% [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);
```

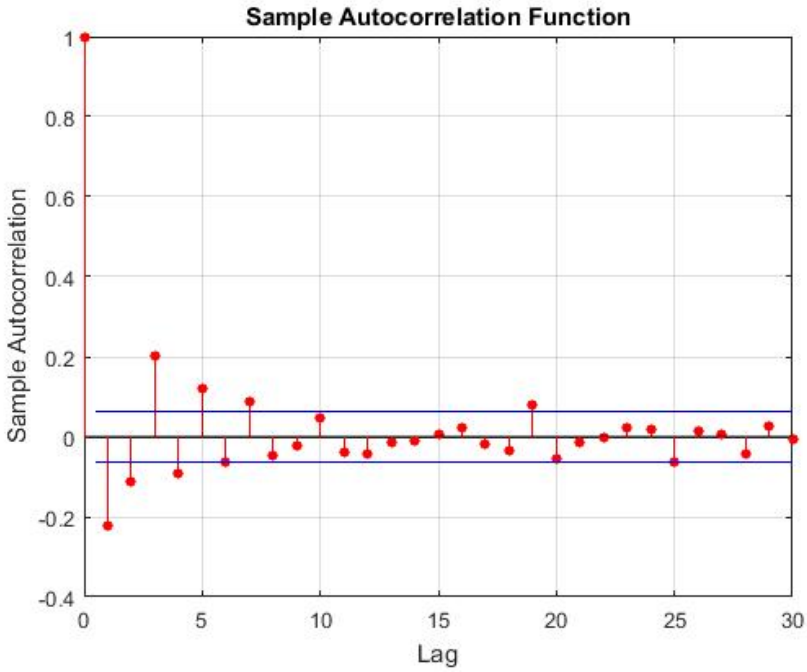
Una vez ejecutado el código para este primer ejemplo hemos obtenido 3 gráficos distintos en los que se evalúan diferentes características del proceso:



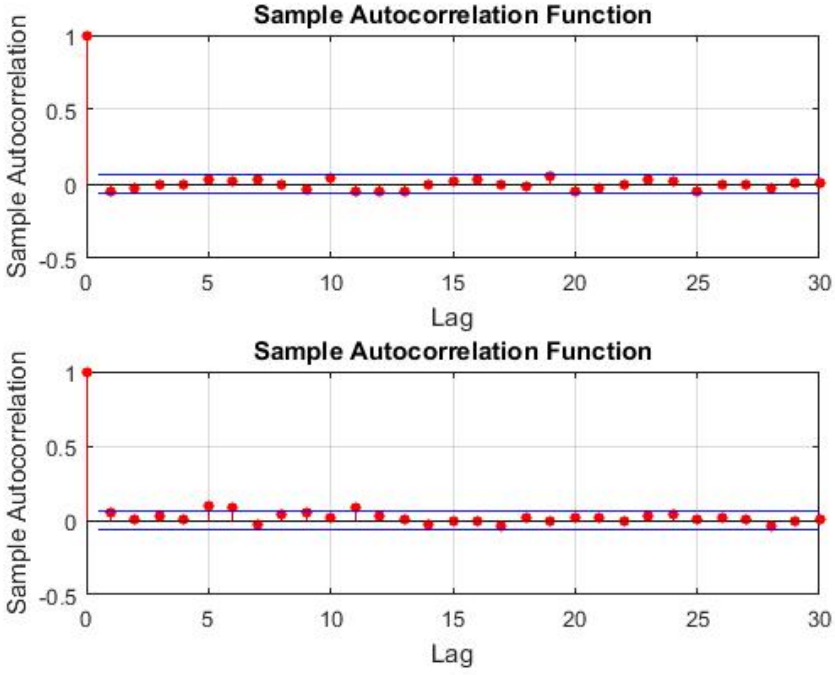
1.- Serie temporal de la evolución de 3 trayectorias de rendimientos en el tiempo.

2.- Serie temporal de la evolución de 3 trayectorias de precios en el tiempo que parten del valor 100.

3.- Serie temporal que muestra cómo evoluciona la volatilidad de la primera trayectoria en el tiempo.



En este segundo gráfico podemos ver cómo en el proceso ARMA(2,3) la variable aleatoria X va perdiendo autocorrelación a medida que el retardo es mayor ya que ϕ_1, y ϕ_2 es menor que 1. A partir de el *Lag* 6, el proceso deja de poseer autocorrelación significativa.

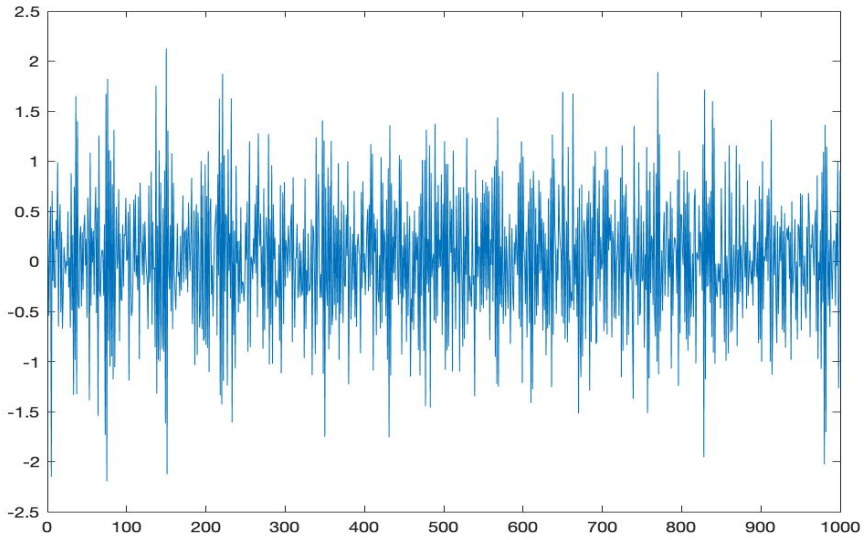


En este último gráfico podemos apreciar la autocorrelación existente en las innovaciones del proceso (u, arriba) y la autocorrelación presente en los valores absolutos de estas mismas (abs(u), abajo). Dado que las innovaciones son iguales al producto de sigma con un vector de variables independientes e idénticamente distribuidas (iid), se puede observar cómo la autocorrelación es inexistente.

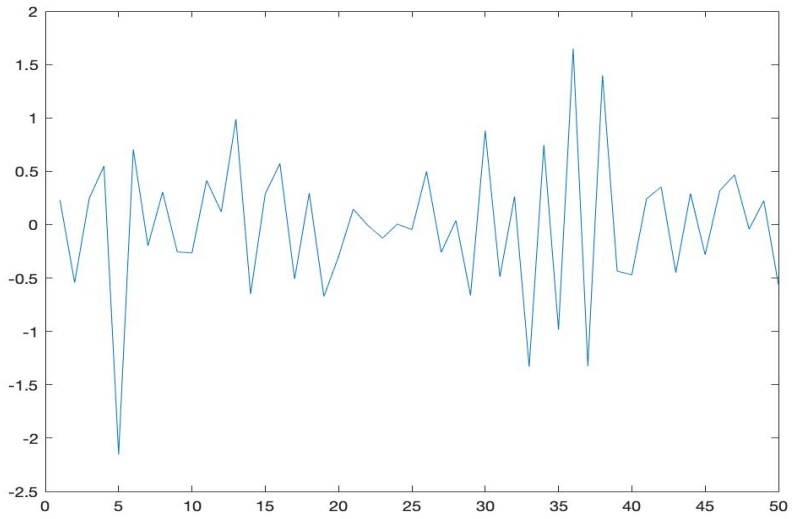
Ejemplo 2: Simulacion AR (1)

```
% Ejemplo 2: Simulacion 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,:));
```

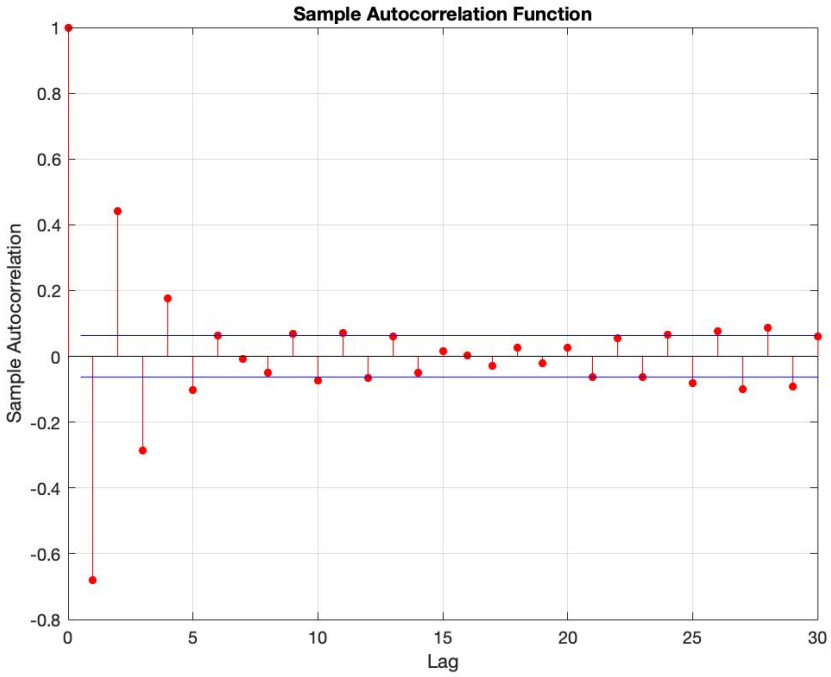
Una vez ejecutado el código para este primer ejemplo hemos obtenido 5 gráficos distintos en los que se evalúan diferentes características del proceso.



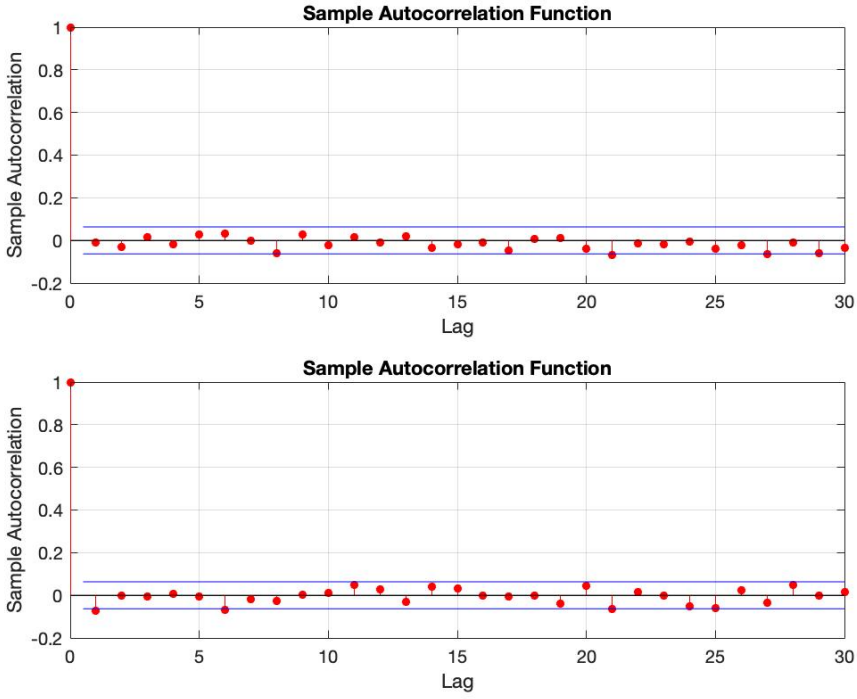
En este primer gráfico podemos observar los distintos valores que obtiene el proceso a lo largo de los 1000 *timesteps*.



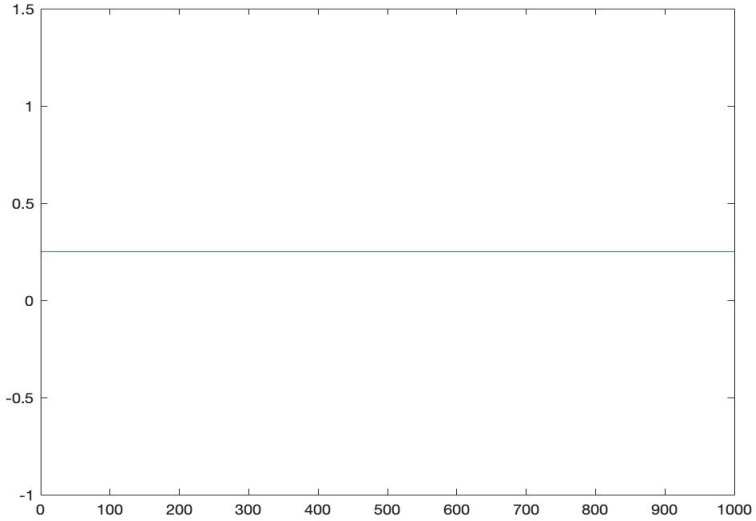
En este primer gráfico podemos observar los distintos valores que obtiene el proceso a lo largo de los 50 primeros *timesteps*.



En este tercer gráfico podemos ver cómo en el proceso AR(1) la variable aleatoria X va perdiendo autocorrelación a medida que el retardo es mayor ya que ϕ_1 es menor que 1. Además, vemos cómo el proceso tiene autocorrelación negativa con los valores de X de orden impar debido a que el signo de ϕ_1 es negativo.



En este cuarto gráfico podemos apreciar la autocorrelación existente en las innovaciones del proceso (u, arriba) y la autocorrelación presente en los valores absolutos de estas mismas (abs(u), abajo). Dado que las innovaciones son iguales al producto de sigma con un vector de variables independientes e idénticamente distribuidas (iid), se puede observar cómo la autocorrelación es inexistente.



En este último gráfico observamos que el valor de h es estable en el tiempo porque alpha y beta no tienen valores en este ejemplo. Además, h mantiene un valor de 0,25 que el valor de kappa coincide con el de sigma.

Ejemplo 3: Simulacion AR(1) + GARCH(1,1)

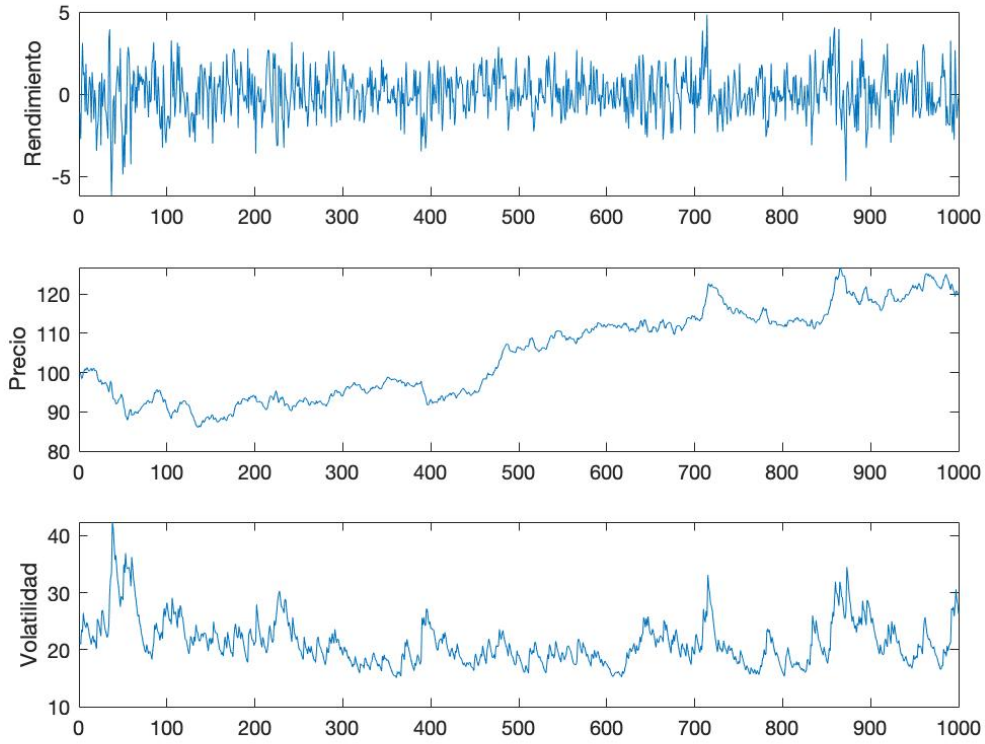
```
% Ejemplo 3: Simulacion AR(1) + GARCH(1,1)
% M = 1; T = 1000;
% phi0 = 0.0; phi1 = 0.2;
% theta = [];
% kappa = 0.1; alpha = 0.10; beta = 0.85;
% X0 = 0; u0 = 0; h0 = kappa/(1-alpha-beta);
% [X,u,h]=simARMAGARCH(M,T,phi0,phi1,theta,kappa,alpha,beta,X0,u0,h0);
% diasEnAnyo = 250;
% S0 = 100; dT = 1/diasEnAnyo; S = cumprod([S0*ones(M,1) exp(X*dT)],2);
% 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);
```

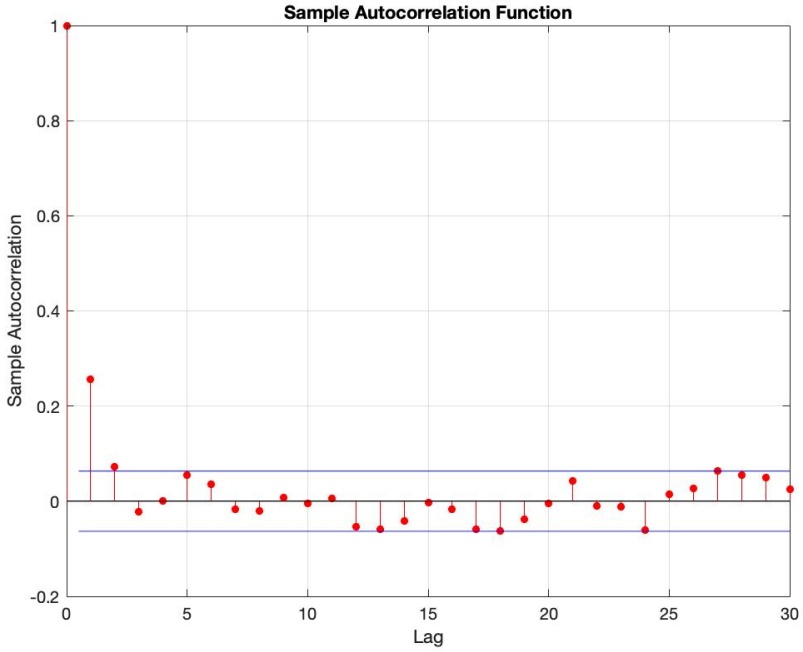
Una vez ejecutado el código para este primer ejemplo hemos obtenido 3 gráficos distintos en los que se evalúan diferentes características del proceso:

1.- Serie temporal de la evolución de 1 trayectoria de rendimientos en el tiempo.

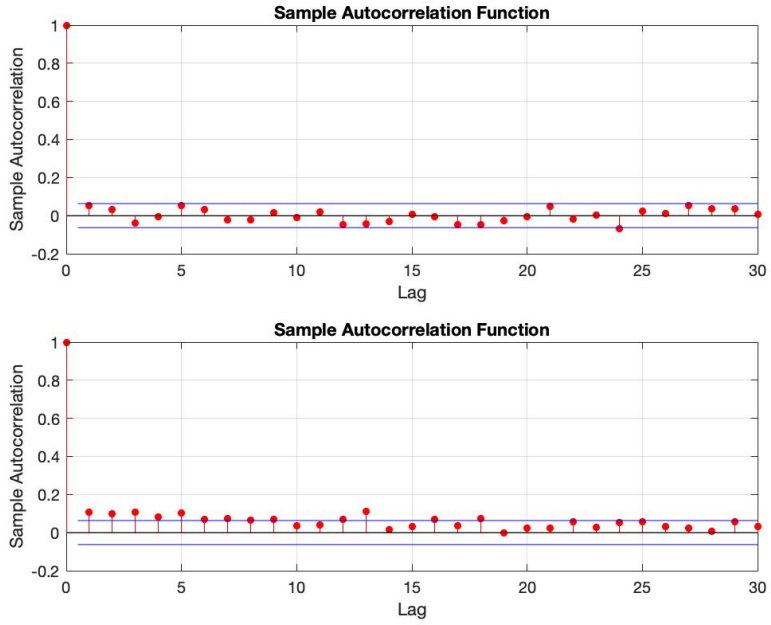
2.- Serie temporal de la evolución de 1 trayectoria de precios en el tiempo que parten del valor 100.

3.- Serie temporal que muestra cómo evoluciona la volatilidad de la trayectoria en el tiempo. Se puede apreciar cómo la volatilidad es más volatil que los precios.





En este segundo gráfico podemos ver cómo en el proceso AR (1) la variable aleatoria X va perdiendo rápidamente autocorrelación a medida que el retardo es mayor ya que $\phi_1$es bastante pequeña (ϕ_1= 0,2). A partir de el *Lag* 2, el proceso deja de poseer autocorrelación significativa.



En este último gráfico podemos apreciar la autocorrelación existente en las innovaciones del proceso (u, arriba) y la autocorrelación presente en los valores absolutos de estas mismas (abs(u), abajo). Dado que las innovaciones son iguales al producto de sigma con un vector de variables independientes e idénticamente distribuidas (iid), se puede observar cómo la autocorrelación es inexistente.