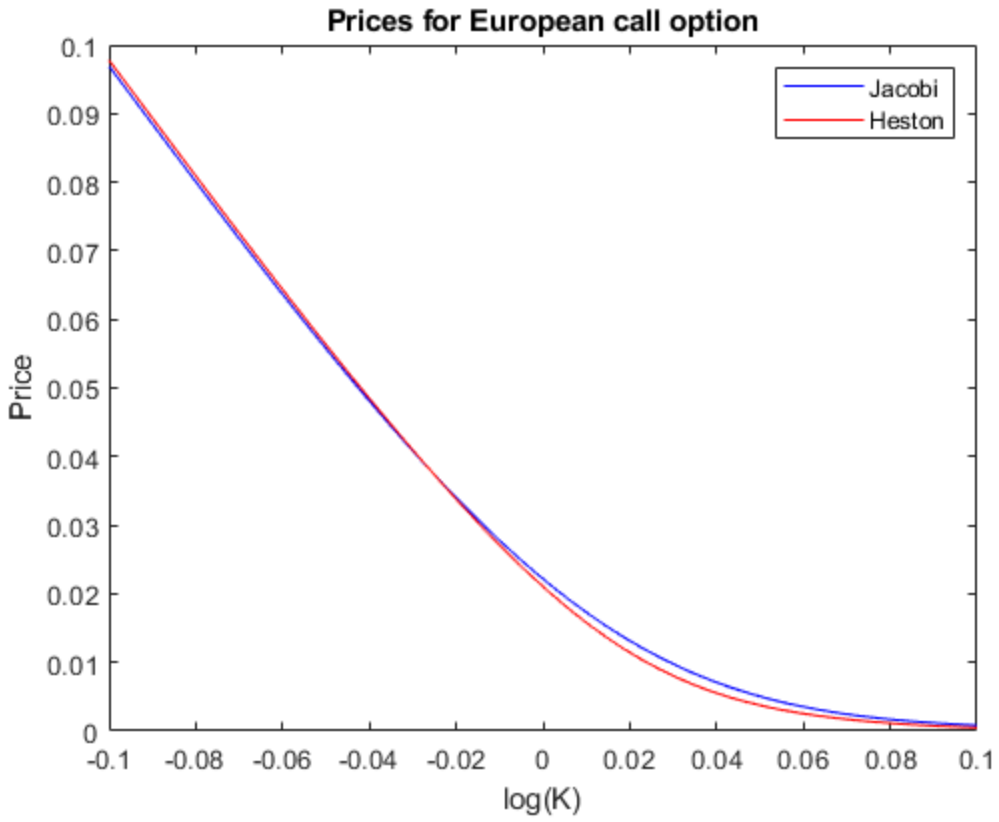
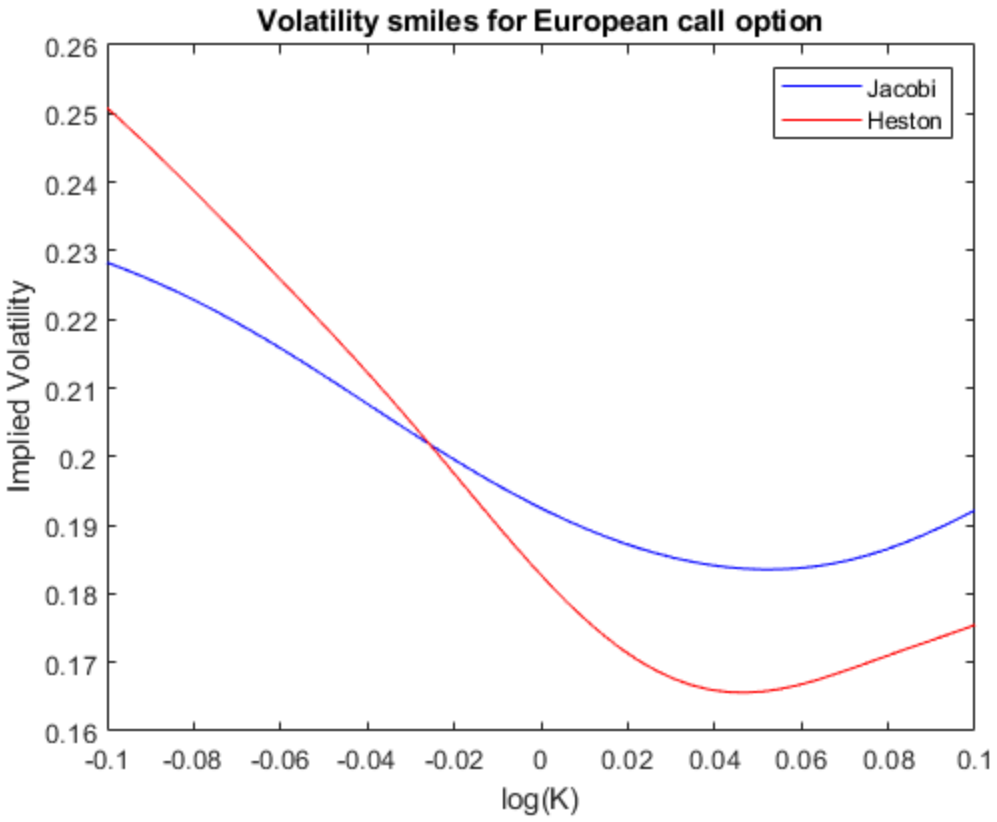
```
% Take Home Exam 3 - Part E
% Anita Mezzetti

clc
clear all
close all

% initial conditions:
X0 = 0;
V0 = 0.04;

% parameters:
kappa = 0.5;
theta = 0.04;
sigma = 1;
r = 0;
rho = -0.5;
T = 1/12;
v_min = 1e-4;
v_max = 0.08;
N = 10;          % approximation degree

% weighting gaussian distribution parameters
% GenJacobiCanonic function definded below
G = Generator_Canonic(kappa, theta, sigma, r, rho, v_min, v_max);
M = [1 V0 X0 V0^2 V0*X0 X0^2] * expm(T * G);

% mean and standard deviation:
mu_w = M(3);
sigma_w = sqrt(M(6) - M(3)^2);

k = linspace(-.1,.1,50);    % strike logprices vector

% implied volatility vectors initialization
IV_jacobi = zeros(length(k), 1);
IV_heston = zeros(length(k), 1);

% Approximated Jacobi price:
price_jacobi = PriceApprox(N, V0, X0, kappa, sigma, theta, r, rho, ...
     T, v_min, v_max, mu_w, sigma_w, k);

% Jacobi implied volatility loop
for i = 1:length(k)
    price = price_jacobi(i);
    IV_jacobi(i) = blsimpv(exp(X0), exp(k(i)), r, T, price);
end

% Heston price:
alpha = 1;
L = 100;

price_heston = zeros(length(k),1);   % initialization
```

```
for i = 1:length(k)
    price_heston(i) =
 CallHeston(X0,V0,kappa,theta,sigma,r,rho,T,k(i),alpha,L);
    max_price_heston = max(0,price_heston(i));
    % CallHeston function definded below
    IV_heston(i) = blsimpv(exp(X0), exp(k(i)), r, T,
 max_price_heston);
end

% plots
figure
plot(k,price_jacobi,'b',k, price_heston,'r')
title('Prices for European call option')
xlabel('log(K)')
ylabel('Price')
legend('Jacobi','Heston')

figure
plot(k, IV_jacobi, 'b', k, IV_heston,'r')
title('Volatility smiles for European call option')
xlabel('log(K)')
ylabel('Implied Volatility')
legend('Jacobi','Heston')

% Functions:

function [price] = CallHeston(X,V,kappa,theta,sigma,r,...
    rho,T,k,alpha,L)
% call option pricing via characteristic function and Carr-Madan
 formula

% auxiliary functions
beta = @(z) kappa-1i*rho*sigma*z;
gamma = @(z) sqrt(sigma^2*(z.^2+1i*z) + beta(z).^2);

% Heston model characteristic function (slide 27 lecture 3):
num = @(z) exp(1i*z*X + 1i*z*r*T + (kappa*theta* T*beta(z)/sigma^2));
den = @(z) (cosh(0.5*gamma(z)*T) + ...
    beta(z)./gamma(z).*sinh(0.5*gamma(z)*T)).^(2*kappa*theta/sigma^2);

phi = @(z) (num(z) ./den(z)) .*...
    exp(-(z.^2 + 1i*z)*V./(gamma(z).*coth(0.5*gamma(z)*T) + beta(z)));

% integrand in Carr-Madan formula
integrand = @(nu) real(phi(nu-1i*(alpha+1))./...
    ((alpha+1i*nu).*(alpha+1+1i*nu)).*exp(-1i*nu*k));

% price via Carr-Madan formula
price = exp(-r*T-alpha*k)/pi*integral(integrand,0,L);

end

function [G] = Generator_Canonic(kappa, theta, sigma, r, rho, v_min,
 v_max)
```

```
% Build the matrix representation of the Jacobi model
%   generator with canonic polynomial basis up to second degree

% matrix initialization
G = zeros(6,6);

% vector Q
D = (sqrt(v_max)-sqrt(v_min))^(-2);
Q = D*[-v_max*v_min; v_max+v_min; 0; -1; 0; 0];

% generator matrix
G(1:2,2) = [kappa * theta; -kappa];
G(1:2,3) = [r; -0.5];
G(:,4) = [0; 2 * kappa * theta; 0; -2*kappa; 0; 0] + sigma^2 * Q;
G(:,5) = [0; r; kappa * theta; -0.5; -kappa; 0] + rho * sigma * Q;
G(:,6) = [0; 1; 2 * r; 0; -1; 0];

end
```





Published with MATLAB® R2020a