Métodos Computacionais em Finanças / IMPA(2021) *Soluções*

Francis Araujo

January 3, 2022

1 Modelo de volatilidade local

$$dS(t) = \mu S(t)dt + \sigma(S(t), t)S(t)dW(t) \tag{1}$$

Defina:

$$\sigma(S, t) = \sigma_0 + \sigma_1 \cos\left(\frac{2\pi S}{K}\right) \sin\left(\frac{2\pi t}{K}\right)$$

Note que esta equação é similar à de Black-Scholes clássica, mas a volatilidade é uma função do tempo e o preço da ação $S(t)$. Neste caso não existe uma fórmula para a solução, portanto é necessário o uso de métodos numéricos.

1.1 Método de Milstein

Implemente um código computacional que simule, usando o método de Milstein, M trajetórias do preço da ação no intervalo $[0, T]$.

1.1.1 Código Referência

Nome do arquivo: milstein_algorithm
Solution:
Note que a equação (1) pode ser escrita da seguinte forma:

$$dS(t) = a(S(t), t)dt + b(S(t), t)dW(t)$$

Sendo assim, $a(S(t), t)$ representa a parcela do drift, enquanto $b(S(t), t)$ representa a parcela de difusão.

O método de Milstein é dado por:

$$\begin{aligned} S_{n+1} &= S_n + a\left(t_n, S_n\right) h + b\left(t_n, S_n\right) \Delta W_n + \frac{1}{2} b b_s \left(\Delta W_n^2 - h\right) \\ &= S_n + a\left(t_n, S_n\right) h + b\left(t_n, S_n\right) \sqrt{\Delta h} \cdot W_n + \frac{1}{2} b b_s \sqrt{\Delta h} \left(W_n^2 - 1\right) \end{aligned}$$

$h = t_{n+1} - t_n$; $b_s = \frac{\partial b}{\partial S}$

Aplicando este método, a dinâmica de movimento para cada instante de tempo será:

$$S_{n+1} = S_n + \mu S_n h + [\sigma_0 + \sigma_1 \cos\left(\frac{2\pi S_n}{K}\right) \sin\left(\frac{2\pi t_n}{K}\right)] S_n \sqrt{h} W_n + \frac{1}{2} b b_s \sqrt{\Delta h}\left(W_n^2 - 1\right)$$

e

$$\begin{aligned} b_s &= 1 \cdot \sigma(t_n, S_n) + S_n \cdot \sigma_s(t_n, S_n) \\ &= 1 \cdot \sigma(t_n, S_n) + S_n[-\sigma_1 sin\left(\frac{2\pi S_n}{K}\right) \cdot \sin\left(\frac{2\pi t_n}{K}\right) \cdot \frac{2\pi}{K}] \end{aligned}$$

A figura abaixo destaca a trajetória do ativo subjacente, utilizando o método proposto. Parâmetros considerados: $T = 10$; $S_0 = 1$; $M = 12$;$\mu = 0.01$; $\sigma_0 = 0.05$; $\sigma_1 = 0.03$; $K = 1.1$; Time _steps=50

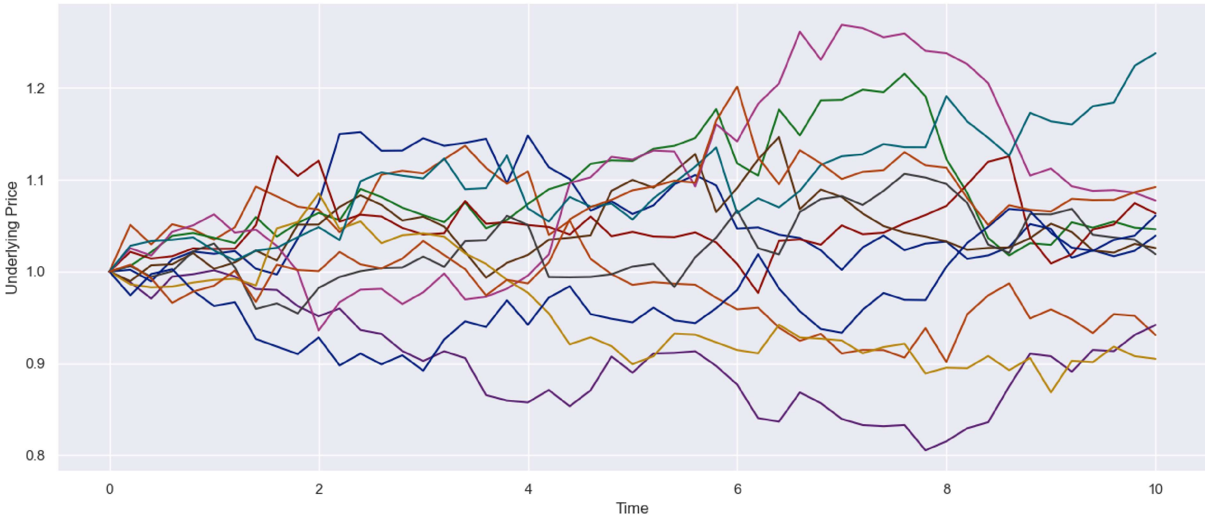


Figure I: Trajetória de Preços (Milstein)

Além disso, interessante mencionar a distribuição dos preços do ativo subjacente na maturidade(S_T), para esta simulação. O gráfico II destaca que 40% dos preços finais são menores ou iguais do que a média amostral, representado pelo valor 0.4 atribuído a função de distribuição acumulada (eixo y) para este determinado momento. É possível interpretar, também, que aproximadamente 60% da distribuição de preços se localiza um desvio-padrão de distância da média amostral.

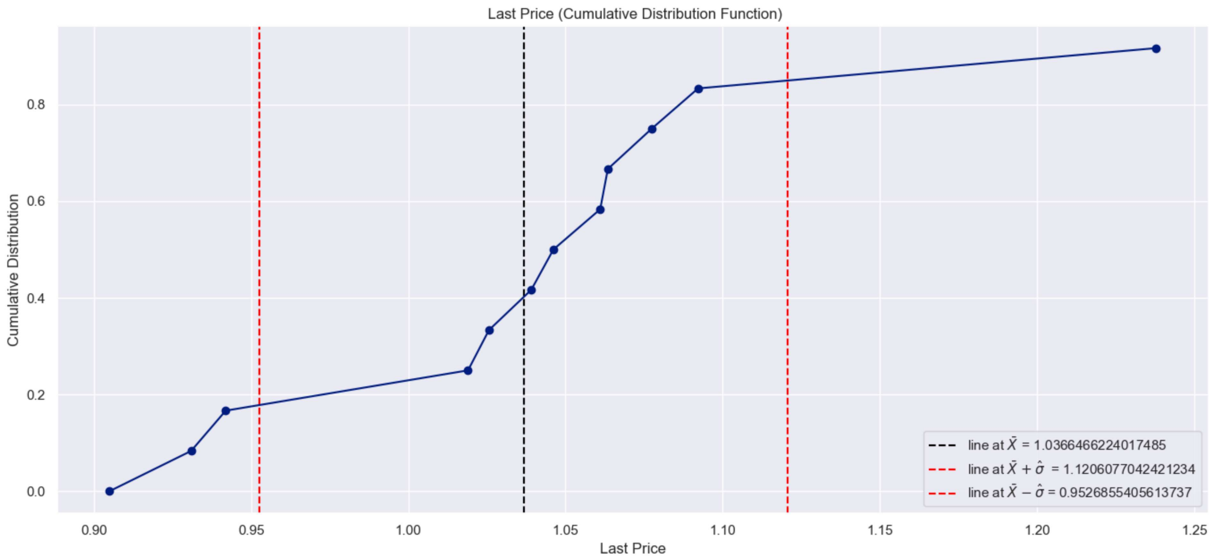


Figure II: Distribuição Acumulada (Preço Final)

■

1.2 Método Euler-Maruyama Simplificado

Implemente um código computacional para a precificação de uma opção Call Europeia. Isto é, calcule:

$$C(0, S_0) = e^{-rT} E(S(T) - K)^+$$

1.2.1 Código Referência

Nome do arquivo: Euler_M _Simplificado

Solution:

O esquema numérico é dado por:

$$\begin{aligned} S_{n+1} &= S_n + a(t_n, S_n) h + b(t_n, S_n) \sqrt{h} \cdot \operatorname{sign}(\zeta_n - \frac{1}{2}) \\ &= S_n + [\mu S_n] \cdot h + [\sigma_0 + \sigma_1 \cos\left(\frac{2\pi S_n}{K}\right) \sin\left(\frac{2\pi t_n}{K}\right)] S_n \cdot \sqrt{h} \cdot \operatorname{sign}(\zeta_n - \frac{1}{2}) \end{aligned}$$

Tais que:

[1] $\zeta_n \sim U([0,1])$

[2] sign(.) representa a função sinal. Assim, se o valor obtido da variável aleatória uniforme for menor de $\frac{1}{2}$, este assumirá valor -1. Caso contrário, o valor a ser assumido será 1.

Uma vez que a dinâmica de S_n está bem estabelecida, será necessário implementar o processo para calcular o preço da opcão de compra Européia. Este consiste obter o valor presente do payoff estimado no vencimento.

■

1.3 Strong Euler-Maruyama

Implemente um código computacional para a precificação de uma opção do tipo Asian-European Call.Isto é, calcule:

$$C\left(0, S_0\right) = e^{-rT} E\left(\frac{1}{T}\int_0^T S(u)du - K\right)^+$$

1.3.1 Código Referência

Nome do arquivo: Strong_Euler
Solution:

Note que a precificação da opção asiática é um problema 'path-dependent', uma vez que o valor da opção no vencimento depende da média dos preços do ativo subjacente entre $[0, T]$. Sendo assim, é necessário um método numérico que não só aproxime bem os valores no vencimento como também toda trajetória de preços. Dito isso, será aplicado o método **forte** de Euler-Maruyama.

Defina $y(t) = \int_0^t S(u)du$. Então, o problema se resume em calcular:

$$C\left(0, S_0\right) = e^{-rT} E\left(\frac{1}{T}y(T) - K\right)^+$$

Além disso, note que:

$$\begin{aligned}
y(t) &= \int_0^t S(u)du \\
\Rightarrow \frac{dy(t)}{dt} &= S(t) \\
\Rightarrow dy(t) &= S(t) \cdot dt
\end{aligned}$$

Portanto, o sistema de EDEs formado pela dinâmica do ativo subjacente e de y(t):

$$\begin{aligned}
dS(t) &= \mu S(t)dt + \sigma(S(t), t)S(t)dW(t) \\
dy(t) &= S(t) \cdot dt
\end{aligned}$$

Realizando o processo de discretização descrito anterior para a equação diferencial do ativo subjacente:

$$\begin{aligned} S_{n+1} &= S_n + a\left(t_n, S_n\right)h + b\left(t_n, S_n\right)\sqrt{h}\cdot Z \\ &= S_n + [\mu S_n]\cdot h + [\sigma_0 + \sigma_1 \cos\left(\frac{2\pi S_n}{K}\right)\sin\left(\frac{2\pi t_n}{K}\right)]S_n \cdot \sqrt{h}\cdot Z \end{aligned}$$

Realizando o processo de discretização para y(t):

$$\begin{aligned} y(t_{n+1}) &= y(t_n) + S_n \cdot h \\ y(t_0) &= 0 \end{aligned}$$

Uma vez que o processo de discretização está estabelecido, usaremos para calcular o valor da opção, definido pelo valor presente do payoff estimado por Monte Carlo.

■

2 Método FTCS.

No caso em que σ depende de S e t, a equação de Black-Scholes para o preço de uma opção Call Europeia com vencimento em $t = T$ e strike K, toma a forma:

$$\frac{\partial C}{\partial t} - \frac{1}{2}\sigma(S(t),t)S^2\frac{\partial^2 C}{\partial S^2} - rS\frac{\partial C}{\partial S} + rC = 0, \quad S > 0, t < T \tag{2}$$

Com as condições iniciais e de contorno:

$$\begin{aligned} C(S,0) &= (S(0) - K)^+ \\ C(0,t) &= 0 \\ C(L,t) &= L \end{aligned}$$

Implemente o cálculo da opção Call Europeia, usando o método FTCS. Considere

$$\sigma(S(t),t) = \sigma_0 + \sigma_1 \cos\left(\frac{2\pi t}{T}\right)\exp\left(-\left(\frac{S}{K} - 1\right)^2\right)$$

2.1 Código Referência

Nome do arquivo: price _surface

Solution:

Segundo a EDE fornecida, temos que:

$$\begin{aligned} \frac{\partial C}{\partial t} - \frac{1}{2}\sigma(S(t),t)S^2\frac{\partial^2 C}{\partial S^2} - rS\frac{\partial C}{\partial S} + rC &= 0 \\ \Rightarrow \frac{\partial C}{\partial t} &= \frac{1}{2}\sigma(S(t),t)S^2\frac{\partial^2 C}{\partial S^2} + rS\frac{\partial C}{\partial S} - rC \\ &= a(S(t),t)\frac{\partial^2 C}{\partial S^2} + b(S(t),t)\frac{\partial C}{\partial S} + \gamma(S(t),t)C \end{aligned}$$

Adicionalmente, pelo método FTCS, podemos determinar as seguintes aproximações:

$$C(s_i, t_j) \approx C_{i,j}$$

$$\frac{\partial}{\partial t}C(s_i, t_{j+1}) \approx \frac{C_{i,j+1} - C_{i,j}}{\Delta t}$$

$$\frac{\partial}{\partial s}C(s_i, t_j) \approx \frac{C_{i+1,j} - C_{i-1,j}}{2\Delta s}$$

$$\frac{\partial^2}{\partial s^2}C(s_i, t_j) \approx \frac{C_{i+1,j} - 2C_{i,j} + C_{i-1,j}}{\Delta s^2}$$

Além disso, sabemos que:

$$s_i = s_0 + i\Delta s \Rightarrow s_i = i\Delta s, \text{ se } s_0 = 0 \tag{3}$$

$$t_j = t_0 + j\Delta t \Rightarrow t_j = j\Delta t, \text{ se } t_0 = 0 \tag{4}$$

Sendo assim, realizando as devidas substituições na equação diferencial, é possível obter:

$$\begin{aligned}
\frac{C_{i,j+1} - C_{i,j}}{\Delta t} &= a(S(t),t)[\frac{C_{i+1,j} - 2C_{i,j} + C_{i-1,j}}{\Delta s^2}] + b(S(t),t)[\frac{C_{i+1,j} - C_{i-1,j}}{2\Delta s}] + \gamma(S(t),t)C_{i,j} + O(S(t),t) \\
C_{i,j+1} - C_{i,j} &= \Delta t[a(S(t),t)[\frac{C_{i+1,j} - 2C_{i,j} + C_{i-1,j}}{\Delta s^2}] + b(S(t),t)[\frac{C_{i+1,j} - C_{i-1,j}}{2\Delta s}] + \gamma(S(t),t)C_{i,j} + O(S(t),t)] \\
C_{i,j+1} &= \Delta t[a(S(t),t)[\frac{C_{i+1,j} - 2C_{i,j} + C_{i-1,j}}{\Delta s^2}] + b(S(t),t)[\frac{C_{i+1,j} - C_{i-1,j}}{2\Delta s}] + \gamma(S(t),t)C_{i,j}] + C_{i,j}
\end{aligned}$$

Assim,

$$C_{i,j+1} = \frac{\Delta t}{\Delta s^2} \cdot a(S(t),t)[C_{i+1,j} - 2C_{i,j} + C_{i-1,j}] + \frac{\Delta t}{2\Delta s} \cdot b(S(t),t)[C_{i+1,j} - C_{i-1,j}] + \Delta t \cdot \gamma(S(t),t)C_{i,j} + C_{i,j}$$

Rearranjando os termos:

$$\begin{aligned}
C_{i,j+1} &= [1 - 2\frac{\Delta t}{\Delta s^2} \cdot a(S(t),t) + \Delta t \cdot \gamma(S(t),t)]C_{i,j} + [\frac{\Delta t}{\Delta s^2} \cdot a(S(t),t) + \frac{\Delta t}{2\Delta s} \cdot b(S(t),t)]C_{i+1,j} + \\
&\quad [\frac{\Delta t}{\Delta s^2} \cdot a(S(t),t) - \frac{\Delta t}{2\Delta s} \cdot b(S(t),t)]C_{i-1,j}
\end{aligned}$$

Substituindo os coeficientes das expressões anteriores,

$$\begin{aligned}
C_{i,j+1} &= [1 - 2\frac{\Delta t}{\Delta s^2} \cdot \sigma(S(t),t)S_i^2 \cdot \frac{1}{2} - \Delta t \cdot r]C_{i,j} + \\
&\quad [\frac{\Delta t}{\Delta s^2} \cdot \sigma(S(t),t)S_i^2 \cdot \frac{1}{2} + \frac{\Delta t}{2\Delta s} \cdot rS]C_{i+1,j} + \\
&\quad [\frac{\Delta t}{\Delta s^2} \cdot \sigma(S(t),t)S_i^2 \cdot \frac{1}{2} - \frac{\Delta t}{2\Delta s} \cdot rS]C_{i-1,j}
\end{aligned}$$

Utilizando as equações (3) e (4), é possível simplificar:

$$\begin{aligned}
C_{i,j+1} &= [1 - \Delta t \cdot \sigma(S(t),t)i^2 - \Delta t \cdot r]C_{i,j} + \\
&\quad \frac{\Delta t}{2}[\sigma(S(t),t)i^2 + ri]C_{i+1,j} + \\
&\quad \frac{\Delta t}{2}[\sigma(S(t),t)i^2 - ri]C_{i-1,j}
\end{aligned}$$

Onde, $\sigma(S(t),t) = \sigma_0 + \sigma_1 \cos\left(\frac{2\pi\Delta t \cdot j}{T}\right) \exp\left(-\left(\frac{\Delta S \cdot i}{K} - 1\right)^2\right)$.

Esta equação final será utilizada no código.
Note que as expressões anteriores podem ser escritas através da forma matricial

$$C_{j+1} = M \cdot C_j + \rho$$

Onde a matriz M é tridiagonal; ρ representa o vetor das condições inicias.

2.2 Superfície de precificação C(S,t)

Utilizando as seguintes informações:

Parâmetros	r	sigma_0	sigma_1	Time Steps	Price Steps	L	T	K
Valor	0.025	0.06	0.05	16	16	20	1	10

A solução é dada por:

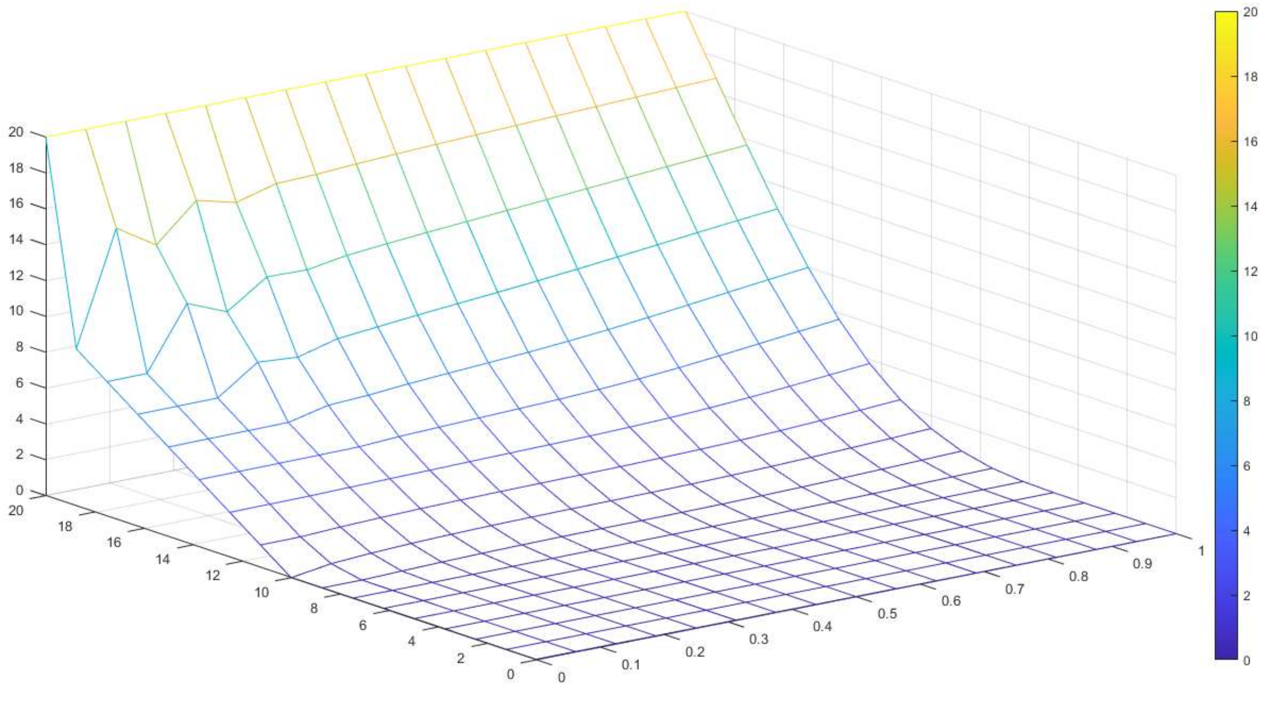
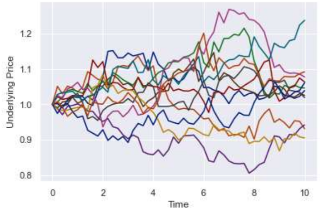


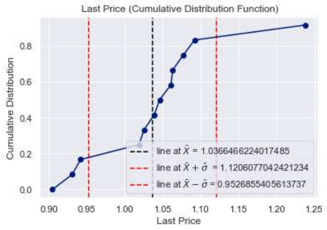
Figure III: Superfície de Precificação (Call Européia)

Constatamos, como o esperado, o preço da opção de compra é uma função crescente em relação ao preço do ativo subjacente. ∎

In [1]:

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#### SDE #####################

#### Model:  dS(t) = mu*S(t)dt +  sigma(S(t), t)*S(t)*dW(t)

#####  sigma(S(t), t) =  sigma_0 +  sigma_1 * cos(2*pi*S/K) * sin(2*pi*t/K)

def milstein_algorithm(T, Time_steps, S_0,K, M, mu, sigma_0, sigma_1):

    # Define semente aleatória
    np.random.seed(502)

    # M: Número de simulações

    # Define o intervalo de tempo
    time_delta = T / Time_steps

    # Define a matriz de preços do ativo subjacente
    S = np.zeros([M, Time_steps+1])

    # Define o preço inicial (primeira coluna da matriz)
    S[:,0] = S_0

    # Define as variáveis aleatórias do modelo
    Z = np.random.normal(size=[M, Time_steps+1]) # (Normal padrão)

    # Define a dinâmica dos preços
    for i in range(Time_steps):
        a = (mu * S[:, i])

        b = ((sigma_0 +  sigma_1 * np.cos(2*np.pi*S[:,i]/K) * np.sin(2*np.pi*i/K)) * S[:,i])

        b_s =  1*(sigma_0 +  sigma_1 * np.cos(2*np.pi*S[:,i]/K) * np.sin(2*np.pi*i/K)) + (S[:,i]*(-1*sigma_1*np.sin(
2*np.pi*i/K))*np.cos(2*np.pi*S[:,i]/K)*np.cos(2*np.pi*S[:,i]/K)*2*np.pi/K)

        S[:,i+1] = S[:,i] + a*time_delta + b*np.sqrt(time_delta)*Z[:,i] + 0.5*b*b_s*time_delta*((Z[:,i]**2) - 1)

    # Divide o eixo x
    time_grid = np.linspace(0, T, Time_steps + 1)


    return time_grid, S

###### Simula M trajetórias de acordo com os parâmetros
store_variable = milstein_algorithm(T=10,Time_steps=50, S_0=1, K=1.1, M=12, mu=0.01, sigma_0=0.05, sigma_1=0.03)

sns.set(palette='dark')
milstein_scheme = plt.axes()
milstein_scheme.plot(store_variable[0], store_variable[1].transpose())
milstein_scheme.set_xlabel('Time')
milstein_scheme.set_ylabel('Underlying Price')
plt.show()

#### Distribuição Acumulada ####

# Armazena os últimos valores de preços
last_price = store_variable[1][:, -1]

# Computa a média para simulação de MC
mean_process = np.mean(last_price)

# Computa o desvio padrão para simulação de MC
std_process = np.std(last_price)

# Plota a CDF dos preços no vencimento
data_sorted = np.sort(last_price)

N_grid = len(data_sorted)

cdf_y = np.arange(N_grid) / float(N_grid)

plt.title(' Last Price (Cumulative Distribution Function)')

plt.axvline(x=mean_process, color='black', linestyle='--', label=r'line at $ \bar X $ = {}'.format(mean_process))

plt.axvline(x=mean_process + std_process, color='red', linestyle='--', label=r'line at $ \bar X + \hat \sigma$  = {}
'.format(mean_process + std_process))

plt.axvline(x=mean_process - std_process, color='red', linestyle='--', label=r'line at $ \bar X -  \hat \sigma$ = {}
'.format(mean_process - std_process))

plt.xlabel("Last Price")
plt.ylabel("Cumulative Distribution")

plt.plot(data_sorted, cdf_y, marker='o')

plt.legend()
plt.show()
```





In [ ]:

In [ ]:

1 FTCS Implementation (MATLAB)

```
function [C] = price_surface(r,sigma_0,sigma_1,
   Time_steps, Price_steps, L, T, K)

% Function returns the option price matrix under the
   FTCS method

% Mesh variables
delta_t = T/Time_steps;
delta_s = L/Price_steps;

% Option price matrix (Initial value)
C(1:Price_steps + 1,1:Price_steps + 1) = 0;

% Define the time and underlying grid
S = (0:Price_steps)*delta_s;
time_grid = (0:Time_steps)*delta_t;

% Initial and Boundary conditions
C(1:Price_steps + 1,1) = max(S - K,0);
C(1, 1:Time_steps + 1) = 0;
C(Price_steps + 1,1:Time_steps + 1) = L;

% Recursive function
for j = 1:Time_steps
    for i = 2:Price_steps
        sigma_st = sigma_0 + sigma_1*cos(2*pi*delta_t*
            j/T)*exp(-1*((i*delta_s/K)-1)^2);
        C(i,j+1) = 0.5*delta_t*(sigma_st*((i)^2) - r*i
            )*C(i-1,j) + (1-delta_t*(sigma_st*i*i + r))
            *C(i,j) + 0.5*delta_t*(sigma_st*((i)^2) + r
            *i)*C(i+1,j);
    end
end

% Price surface plot
mesh(time_grid, S,C);
colorbar;

end
```