PSTAT 222C Homework 2

1. Implicit vs . Crank-Nicholson Finite Difference PDE Solvers

We consider a corridor option for asset X_t with payoff of \$1 if $X_t \in [L_1, L_2]$ and \$0 otherwise. X_t is governed by the SDE

$$dX_t = rX_t dt + \sigma^2 X_t^\gamma dW_t \tag{1}$$

where γ is the elasticity of the variance. Applying Ito's Lemma to the price of the option, $V(t,x)$, we get:

$$\begin{aligned} dV &= \frac{\partial V}{\partial t}dt + \frac{\partial V}{\partial x}dX_t + \frac{1}{2}\frac{\partial^2 V}{\partial x^2}dX_t^2 && (2)\\ &= \frac{\partial V}{\partial t}dt + \frac{\partial V}{\partial x}(rX_t dt + \sigma^2 X_t^\gamma dW_t) + \frac{1}{2}\frac{\partial^2 V}{\partial x^2}(rX_t dt + \sigma^2 X_t^\gamma dW_t)^2 && (3)\\ &= \left(\frac{\partial V}{\partial t} + rX_t\frac{\partial V}{\partial x} + \frac{\sigma^2 X_t^{2\gamma}}{2}\frac{\partial^2 V}{\partial x^2}\right)dt + \sigma X_t^\gamma \frac{\partial V}{\partial x}dW_t && (4)\end{aligned}$$

Under the assumption that this is the risk-neutral measure, we must have that the expected price of the option is equal to the present value of the risk free rate of the price of the option; that is:

$$\mathbb{E}[dV] = rVdt = \left(\frac{\partial V}{\partial t} + rX_t\frac{\partial V}{\partial x} + \frac{\sigma^2 X_t^{2\gamma}}{2}\frac{\partial^2 V}{\partial x^2}\right)dt \tag{5}$$

We do a time change from $t \to T - t$, the only effect of which is that the sign of $\partial_t V$ changes. It therefore follows that the PDE for the CEV model can be written as:

$$\frac{\partial V}{\partial t} = rX_t\frac{\partial V}{\partial x} + \frac{\sigma^2 X_t^{2\gamma}}{2}\frac{\partial^2 V}{\partial x^2} - rV \tag{6}$$

Implicit Finite Difference

We can derive the implicit Finite Difference as:

$$\frac{V_n^m - V_n^{m-1}}{\Delta t} = r(n\Delta x)\frac{V_{n+1}^m - V_{n-1}^m}{2\Delta x} + \frac{\sigma^2(n\Delta x)^{2\gamma}}{2}\frac{V_{n+1}^m - 2V_n^m + V_{n-1}^m}{(\Delta x)^2} - rV_n^m \tag{7}$$

Solving for V_n^{m-1}, we find:

$$V_n^{m-1} = V_n^m - \Delta t\left(r(n\Delta x)\frac{V_{n+1}^m - V_{n-1}^m}{2\Delta x} + \frac{\sigma^2(n\Delta x)^{2\gamma}}{2}\frac{V_{n+1}^m - 2V_n^m + V_{n-1}^m}{(\Delta x)^2} - rV_n^m\right) \tag{8}$$

$$= \underbrace{\frac{\Delta t}{2}\left(rn - \frac{\sigma^2(n\Delta x)^{2\gamma}}{(\Delta x)^2}\right)}_{\tilde{a}_n} V_{n-1}^m + \left(1 + \underbrace{r\Delta t + \frac{\sigma^2(n\Delta x)^{2\gamma}\Delta t}{(\Delta x)^2}}_{\tilde{b}_n}\right) V_n^m - \underbrace{\frac{\Delta t}{2}\left(rn + \frac{\sigma^2(n\Delta x)^{2\gamma}}{(\Delta x)^2}\right)}_{\tilde{c}_n} V_{n+1}^m \tag{9}$$

$$= \tilde{a}_n V_{n-1}^m + (1 + \tilde{b}_n)V_n^m - \tilde{c}_n V_{n+1}^m \tag{10}$$

Imposing the exogenous boundary at L_1 and $L2$ such that for all $0 \leq m \leq T/\Delta t$, $V_{L_1}^m = V_{L_2}^m = 1$ and $V_{L_1-1}^m = V_{L_2+1}^m = 0$ we find that we can represent the discretized PDE as:

$$AV^m + g^{m-1} = V^{m-1} \tag{11}$$

$$V^m = A^{-1}\left(V^{m-1} - g^{m-1}\right) \tag{12}$$

To see how to apply the boundary conditions, it is helpful to visualize the matrix representation of this equation. We have:

$$\begin{pmatrix} \tilde{a}_1 & 1+\tilde{b}_1 & -\tilde{c}_1 & 0 & \cdots & 0 \\ 0 & \tilde{a}_2 & 1+\tilde{b}_2 & -\tilde{c}_2 & \cdots & 0 \\ 0 & 0 & \ddots & \ddots & \cdots & 0 \\ 0 & 0 & 0 & \tilde{a}_{N-1} & 1+\tilde{b}_{N-1} & -\tilde{c}_{N-1} \end{pmatrix} \begin{pmatrix} V_0^m \\ V_1^m \\ \vdots \\ V_{N-1}^m \\ V_{N-1}^m \end{pmatrix} = \begin{pmatrix} V_0^{m-1} \\ V_1^{m-1} \\ \vdots \\ V_{N-1}^{m-1} \\ V_N^{m-1} \end{pmatrix} \tag{13}$$

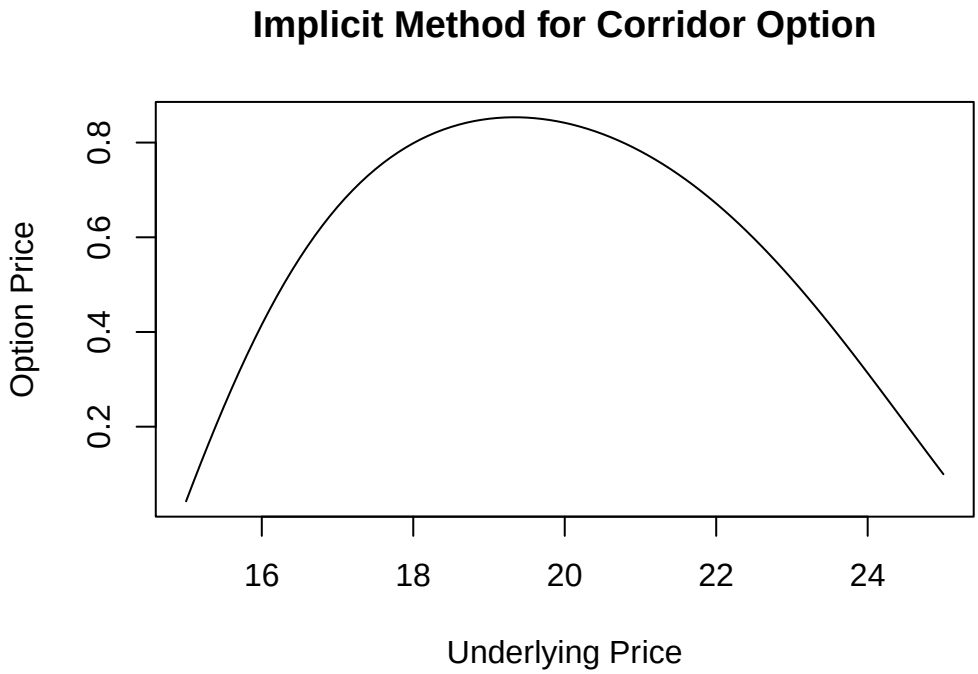
and it becomes clear that g^{m-1} is given by:

$$g^{m-1} = \begin{pmatrix} \tilde{a}_1 V_0^m \\ 0 \\ \vdots \\ 0 \\ \tilde{c}_{N-1} V_N^m \end{pmatrix} \tag{14}$$

```r
implicitfd<- function(T,dt,dx,X_min,X_max,gam,opttype,K=0,other_asset=0,x0){
  r <- 0.05
  sigma <- 0.4
  time_steps <- as.integer(T/dt)
  space_steps <- as.integer((X_max-X_min)/dx)+1
  vetS <- X_min + dx*(0:(space_steps-1))

  V = matrix(0,space_steps,time_steps) # intitialize matrix

  vetI <- X_min/dx + 0:(space_steps-1)
  a_i <- dt/2*(-sigma^2*(vetI*dx)^(2*gam)/(dx^2) + r*vetI)
  b_i <- dt*(sigma^2*(vetI*dx)^(2*gam)/(dx^2) + r)
  c_i <- dt/2*(sigma^2*(vetI*dx)^(2*gam)/(dx^2) + r*vetI)

  Amatrix <- diag(1+b_i,space_steps)
  Amatrix[(row(Amatrix) - col(Amatrix)) == 1] <- a_i[2:(space_steps)]
  Amatrix[(row(Amatrix) - col(Amatrix)) == -1] <- -c_i[1:(space_steps-1)]
  if(opttype=="cor"){
    V[,ncol(V)] <- 1 - ((vetS>L2 | vetS<L1)*1)
  }
  if(opttype=="call"){
    V[,ncol(V)] <- pmax(vetS-K,0)
    V[nrow(V),] <- X_max - K*exp(-r*seq(dt,T,dt))
    }
  if(opttype=="compound"){
      V[,ncol(V)] <- pmax(other_asset-K,0)
      V[nrow(V),] <- other_asset[length(other_asset)] - K*exp(-r*seq(dt,T,dt))
    }

  V[1,] = 0

  Bmatrix <- diag(1,space_steps)
  for (k in 1:time_steps-1){
    t = time_steps - k
    g <- c(rep(0,space_steps-1),c_i[length(c_i)]*(V[nrow(V),ncol(V)-1]))
    g[1] = a_i[1]*V[1,ncol(V)]
    V[,t-1]<-solve(Amatrix, Bmatrix %*% V[,t] + g)

  }
  return(list(V=V,price=V[which(vetS==x0)]))
}
```

```r
gam<-0.8;dt <- 0.1;T = 1/2;dx <- 0.1;L1 <- 15;L2 <- 25
```

```
val<-implicitfd(T,dt,dx,L1,L2,gam,"cor",0,0,20)
options(scipen=999)
plot(x = seq(15,25,dx), y= val$V[,1],type='l',main="Implicit Method for Corridor Option",
ylab="Option Price",xlab = "Underlying Price" )
```



The empirical convergence as Δx decreases is presented after the next solver.

Crank-Nicholson Finite Difference Solver

The Crank-Nicholson Method is essentially an average of the Explicit and Implicit Finite Difference Methods. The explicit finite difference scheme is given by:

$$\frac{V_n^m - V_n^{m-1}}{\Delta t} = r(n\Delta x)\frac{V_{n+1}^{m-1} - V_{n-1}^{m-1}}{2\Delta x} + \frac{\sigma^2(n\Delta x)^{2\gamma}}{2}\frac{V_{n+1}^{m-1} - 2V_n^{m-1} + V_{n-1}^{m-1}}{(\Delta x)^2} - rV_n^{m-1} \tag{15}$$

We solve for V_n^m:

$$V_n^m = V_n^{m-1} + \Delta t \left(r(n\Delta x) \frac{V_{n+1}^{m-1} - V_{n-1}^{m-1}}{2\Delta x} + \frac{\sigma^2 (n\Delta x)^{2\gamma}}{2} \frac{V_{n+1}^{m-1} - 2V_n^{m-1} + V_{n-1}^{m-1}}{(\Delta x)^2} - rV_n^{m-1} \right) \tag{16}$$

$$= V_n^{m-1} \left(\underbrace{1 - r\Delta t - \frac{2\sigma^2 (n\Delta x)^{2\gamma} \Delta t}{2(\Delta x)^2}}_{b_n} \right) + V_{n-1}^{m-1} \underbrace{\frac{\Delta t}{2} \left(\frac{-rn\Delta x}{2\Delta x} + \frac{\sigma^2 (n\Delta x)^{2\gamma}}{2(\Delta x)^2} \right)}_{a_n} + V_{n+1}^{m-1} \underbrace{\frac{\Delta t}{2} \left(\frac{rn\Delta x}{2\Delta x} + \frac{\sigma^2 (n\Delta x)^{2\gamma}}{2(\Delta x)^2} \right)}_{c_n} \tag{17}$$

Comparing between the explicit and implicit schemes, we see that $\tilde{a}_n = a_n$, $\tilde{b}_n = b_n$, and $\tilde{c}_n = c_n$. Crank-Nicholson solves:

$$\frac{A}{2} V^m + g^m = \frac{B}{2} V^{m-1} + g^{m-1}$$

We can now define the matrices:

$$\begin{pmatrix} -a_1^{m-1} & 1 - b_1^{m-1} & -c_1^{m-1} & 0 & \cdots & 0 \\ 0 & -a_2^{m-1} & 1 - b_2^{m-1} & -c_2^{m-1} & \cdots & 0 \\ 0 & 0 & \ddots & \ddots & \cdots & 0 \\ 0 & 0 & 0 & -a_{N-1}^{m-1} & 1 - b_{N-1}^{m-1} & -c_{N-1}^{m-1} \end{pmatrix} \begin{pmatrix} V_0^{m-1} \\ V_1^{m-1} \\ \vdots \\ V_{N-1}^{m-1} \\ V_N^{m-1} \end{pmatrix}$$

$$= \begin{pmatrix} a_1^m & 1 + b_1^m & c_1^m & 0 & \cdots & 0 \\ 0 & a_2^m & 1 + b_2^m & c_2^m & \cdots & 0 \\ 0 & 0 & \ddots & \ddots & \cdots & 0 \\ 0 & 0 & 0 & a_{N-1}^m & 1 + b_{N-1}^m & c_{N-1}^m \end{pmatrix} \begin{pmatrix} V_0^m \\ V_1^m \\ \vdots \\ V_{N-1}^m \\ V_N^m \end{pmatrix}$$

Just like in the implicit case, we can write g^{m-1} and g^m as:

$$g^m = \begin{pmatrix} a_1^m V_0^m \\ 0 \\ \vdots \\ \vdots \\ 0 \\ c_{N-1}^m V_N^m \end{pmatrix} \quad \text{and} \quad g^{m-1} = \begin{pmatrix} -a_1^{m-1} V_0^{m-1} \\ 0 \\ \vdots \\ \vdots \\ 0 \\ -c_{N-1}^{m-1} V_N^{m-1} \end{pmatrix}$$

Because we are starting with a terminal condition, we want to solve for V^{m-1}. We therefore iterate:

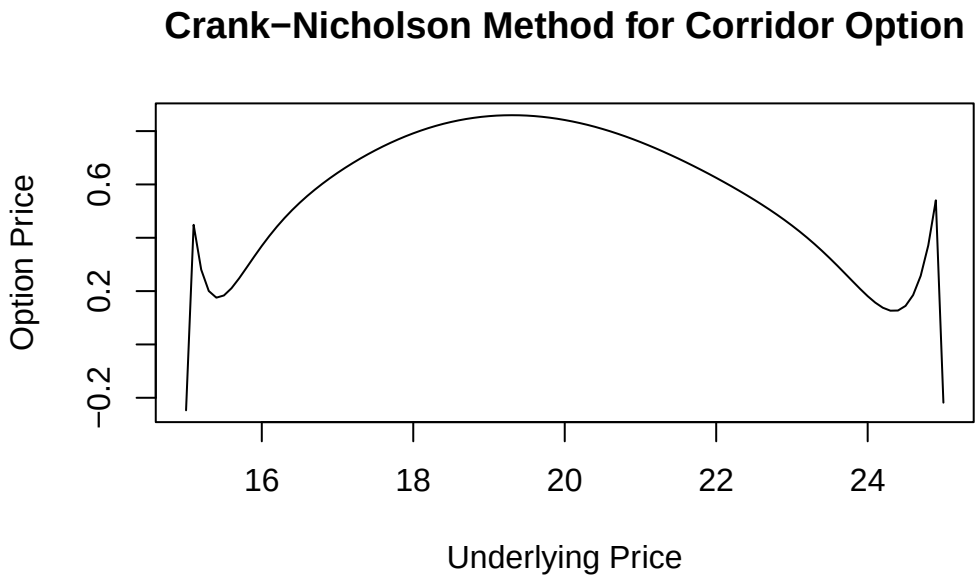
$$V^{m-1} = \left(\frac{A}{2} \right)^{-1} \left(\frac{B}{2} V^m + g^m - g^{m-1} \right)$$

```
cnFd<- function(T,dt,dx,X_min,X_max,gam,opttype,K=0,other_asset=0,x0){
  r <- 0.05
  sigma <- 0.4
```

```
time_steps <- as.integer(T/dt)
space_steps <- as.integer((X_max-X_min)/dx)+1
vetS <- X_min + dx*(0:(space_steps-1))

V = matrix(0,space_steps,time_steps) # intitialize matrix
vetI <- X_min/dx + 0:(space_steps-1)
a_i <- dt/4*(sigma^2*(vetI*dx)^(2*gam)/(dx^2) - r*vetI)
b_i <- -dt/2*(sigma^2*(vetI*dx)^(2*gam)/(dx^2) + r)
c_i <- dt/4*(sigma^2*(vetI*dx)^(2*gam)/(dx^2) + r*vetI)


Amatrix <- diag(1-b_i,space_steps)
Amatrix[(row(Amatrix) - col(Amatrix)) == 1] <- -a_i[2:(space_steps)]
Amatrix[(row(Amatrix) - col(Amatrix)) == -1] <- -c_i[1:(space_steps-1)]

if(opttype=="cor"){
  V[,ncol(V)] <- 1 - ((vetS>=L2 | vetS<=L1)*1)
}
if(opttype=="call"){
  V[,ncol(V)] <- pmax(vetS-K,0)
  V[nrow(V),] <- X_max - K*exp(-r*T)
  V[1,] = 0
  }
if(opttype=="compound"){
    V[,ncol(V)] <- pmax(other_asset-K,0)
    V[nrow(V),] <- other_asset[length(other_asset)] - K*exp(-r*T)
    V[1,] = 0
  }



Bmatrix <- diag(1+b_i,space_steps)
Bmatrix[row(Bmatrix)-col(Bmatrix)==1] <- a_i[2:(space_steps)]
Bmatrix[(row(Bmatrix) - col(Bmatrix)) == -1] <- c_i[1:(space_steps-1)]

for (k in 1:time_steps-1){
  t = time_steps - k
  g_m <- rep(0,space_steps)
  if(!opttype=="cor"){ #Boundaries are always zero with corridor
    g_m[1] = a_i[1]*(V[1,ncol(V)]+V[1,ncol(V)-1])
    g_m[length(g_m)]<- c_i[length(c_i)]*(V[nrow(V),ncol(V)] +
                V[nrow(V),ncol(V-1)])
  }
  V[,t-1]<-solve(Amatrix, Bmatrix %*% V[,t] + g_m)
```

```
  }
  return(list(V=V,price=V[which(vetS==x0)]))
}
val<-implicitfd(T,dt,dx,L1,L2,gam,"cor",0,0,20)
```

```
val<-cnFd(T,dt,dx,L1,L2,gam,"cor",0,0,20)
options(scipen=999)
plot(x = seq(15,25,dx), y= val$V[,1],type='l',main="Crank-Nicholson Method for Corridor Option",
ylab="Option Price",xlab = "Underlying Price" )
```




```
gamma <- .8
dt <- 0.01
L1 = 15
L2 = 25
x0=20
prices_cor_cn<-c()
prices_cor_imp<-c()
options(scipen = 999)
dx_list <- c(0.1, 0.05, 0.02)
for (dx in dx_list) {
  prices_cor_cn<-c(prices_cor_cn,cnFd(T,dt,dx,L1,L2,gamma,"cor",K=0,other_asset=0,x0)$price)
  prices_cor_imp<-c(prices_cor_imp,(implicitfd(T,dt,dx,L1,L2,
          gamma,"cor",K=0,other_asset=0,x0)$price))
}
```

```r
cor_data <- cbind(dx_list,prices_cor_imp,prices_cor_cn)
colnames(cor_data)<- c("dx","Implicit","Crank-Nicholson")
cor_data<-data.frame(cor_data)
```

```r
stargazer(cor_data,type = 'latex', title="Empirical Convergence of PDE Solvers for CEV Corridor Option")
```

% Table created by stargazer v.5.2.3 by Marek Hlavac, Social Policy Institute. E-mail: marek.hlavac at gmail.com
% Date and time: Fri, Jun 16, 2023 - 13:28:21

Table 1: Empirical Convergence of PDE Solvers for CEV Corridor Option

Statistic	N	Mean	St. Dev.	Min	Max
dx	3	0.057	0.040	0.020	0.100
Implicit	3	0.791	0.018	0.774	0.810
Crank.Nicholson	3	0.770	0.005	0.766	0.775

As we shrink Δx we note that the Crank-Nicholson solver has somewhat lower prices, and is closer to the Richardson-Romberg values we saw in our previous assignment, espeically for smaller Δx. Further, the Crank-Nicholson seems to be more stable for changes in Δx.

Compound Option and Comparison between Implicit and Crank-Nicholson Solvers

The Compound Option is priced like a Standard Call between $t = \frac{1}{4}$ and $T = \frac{1}{2}$, with strike 20. Let S be the underlying asset, with discretization grid between 0 and S_{max}. Between these $\frac{1}{4}$ and $\frac{1}{2}$, we have the terminal condition:

$$V\left(S, \frac{1}{2}\right) = \max(S - 20, 0) \tag{18}$$

and boundary conditions for all $t \in [\frac{1}{4}, \frac{1}{2}]$:

$$V(0,t) = 0$$
$$V(S_{max}, t) = S_{max} - 20e^{-rt}$$

Denote this call option C_1. The "mid-terminal'' condition we have for the compound option at $t = \frac{1}{4}$ is:

$$V\left(S, \frac{1}{2}\right) = \max\left(C_1\left(S, \frac{1}{4}\right) - 2, 0\right)$$

and we have the same boundary condition for $S = 0$, and the upper bound now becomes:

$$V\left(S_{max}, t\right) = C_1\left(S, \frac{1}{4}\right) - 2e^{-rt} \tag{19}$$

```r
gam<-0.8
dt <- 0.01
T = 1/2
L1 <- 0
K_end = 20
L2 <- 2*K_end
K_compound<-2

options(scipen=999)
prices_imp_compound = c()
prices_cn_compound = c()

dx_size = c(0.1,0.05,0.02)
for(i in 1:length(dx_size)){
  dx = dx_size[i]
  v_call_end_imp <-implicitfd(1/4,dt,dx,L1,L2,gam,
                              opttype="call",K_end,0,20)$V
  v_total_imp <- implicitfd(1/4,dt,dx,L1,L2,gam,opttype="compound",
                              K_compound,v_call_end_imp[,1],20)$price
  prices_imp_compound<-c(prices_imp_compound,v_total_imp)
  ### Crank-Nicholson in the same loop
  v_call_end_cn <-cnFd(1/4,dt,dx,L1,L2,gam,opttype="call",K_end,0,20)$V
  v_total_cn <- cnFd(1/4,dt,dx,L1,L2,gam,opttype="compound",K_compound,v_call_end_cn[,1],20)$price
  prices_cn_compound<-c(prices_cn_compound,v_total_cn)
}

data_compound<-data.frame(cbind(dx_size,prices_imp_compound,prices_cn_compound))
colnames(data_compound)<- c("dx","Implicit","Crank-Nicholson")
```

```r
stargazer(data_compound,type = 'latex', title="Empirical Convergence of PDE Solvers for CEV Compound Opt
```

% Table created by stargazer v.5.2.3 by Marek Hlavac, Social Policy Institute. E-mail: marek.hlavac at gmail.com
% Date and time: Fri, Jun 16, 2023 - 13:28:59

Table 2: Empirical Convergence of PDE Solvers for CEV Compound Option

Statistic	N	Mean	St. Dev.	Min	Max
dx	3	0.057	0.040	0.020	0.100
Implicit	3	0.353	0.0001	0.353	0.353
Crank-Nicholson	3	0.354	0.00005	0.354	0.354

2. Explicit Scheme for Heston Model

We have the Heston Stochastic Volatility Model with:

$$dS_t = rS_t dt + S_t\sqrt{V_t}dW_t^1$$
$$dV_t = \kappa(\theta - V_t)dt + \eta\sqrt{V_t}dW_t^2$$

Applying the multivariate Ito's Lemma, to a function $f(t, V, S)$ (and excluding the $\frac{\partial f}{\partial t}$ term because it is 0) we have:

$$df(t,V,S) = \frac{\partial f}{\partial t}dt + \frac{\partial f}{\partial S}dS_t + \frac{\partial f}{\partial V}dV_t + \frac{1}{2}\frac{\partial^2 f}{\partial S^2}dS_t^2 + \frac{1}{2}\frac{\partial^2 f}{\partial V^2}dV_t^2 + \frac{\partial^2 f}{\partial S \partial V}dS_t dV_t$$

where we also have that, by the rules of Ito calculus:

$$dS_t^2 = S_t^2 V_t dt \quad dV_t^2 = \eta^2 V_t dt \quad dS_t dV_t = \rho\eta S_t V_t dt$$

and so we obtain the following differential:

$$\begin{aligned} df(t,V,S) &= \frac{\partial f}{\partial t}dt + \frac{\partial f}{\partial S}\left(rS_t dt + S_t\sqrt{V_t}dW_t^1\right) + \frac{\partial F}{\partial V}\left(\kappa(\theta - V_t)dt + \eta\sqrt{V_t}dW_t^2\right) \\ &\quad + \left(\frac{\partial^2 f}{\partial S^2}\frac{S_t^2 V_t}{2} + \frac{\partial^2 f}{\partial V^2}\frac{\eta^2 V_t}{2} + \rho\eta S_t V_t \frac{\partial f}{\partial S \partial V}\right)dt \\ &= \left(\frac{\partial f}{\partial t} + rS_t\frac{\partial f}{\partial S} + \kappa(\theta - V_t)\frac{\partial F}{\partial V} + \frac{S_t^2 V_t}{2}\frac{\partial^2 f}{\partial S^2} + \frac{\eta^2 V_t}{2}\frac{\partial^2 f}{\partial V^2} + \rho\eta S_t V_t\frac{\partial f}{\partial S \partial V}\right)dt \\ &\quad + S_t\sqrt{V_t}\frac{\partial F}{\partial S}dW_t^1 + \eta\sqrt{V_t}\frac{\partial F}{\partial V}dW_t^2 \end{aligned}$$

Taking the risk-neutral Expectation of both sides, we find that the martingale terms are 0, with the rest becoming:

$$\mathbb{E}[df(t,V,S)] = rf(t,V,S)dt = \left(\frac{\partial f}{\partial t} + rS_t\frac{\partial f}{\partial S} + \kappa(\theta - V_t)\frac{\partial F}{\partial V} + \frac{S_t^2 V_t}{2}\frac{\partial^2 f}{\partial S^2} + \frac{\eta^2 V_t}{2}\frac{\partial^2 f}{\partial V^2} + \rho\eta S_t V_t\frac{\partial f}{\partial S \partial V}\right)dt$$

After doing a time change from t to $T - t$ (which only flips the sign on $\partial_t f$ from the chain rule) we have a more familiar PDE:

$$\frac{\partial f}{\partial t} = rS_t\frac{\partial f}{\partial S} + \kappa(\theta - V_t)\frac{\partial f}{\partial V} + \frac{S_t^2 V_t}{2}\frac{\partial^2 f}{\partial S^2} + \frac{\eta^2 V_t}{2}\frac{\partial^2 f}{\partial V^2} + \rho\eta S_t V_t\frac{\partial f}{\partial S \partial V} - rf$$

Now, we have two spatial coordinates and one time coordinate, so we will use the notation that:

$f(t, V, S) = f(m\Delta t, j\Delta s, k\Delta v) = f_{j,k}^m$

We use the following approximations for the partial derivatives:

- First Order Derivatives:

$$\frac{\partial f}{\partial t} = \frac{f_{j,k}^{m+1} - f_{j,k}^{m}}{\Delta t} \qquad \frac{\partial f}{\partial V} = \frac{f_{j+1,k}^{m} - f_{j-1,k}^{m}}{2\Delta v} \qquad \frac{\partial f}{\partial S} = \frac{f_{j,k+1}^{m} - f_{j,k-1}^{m}}{2\Delta s}$$

- Second Order Standard Derivatives:

$$\frac{\partial^2 f}{\partial V^2} = \frac{f_{j+1,k}^{m} - 2f_{j,k}^{m} + f_{j-1,k}^{m}}{(\Delta v)^2} \qquad \frac{\partial^2 f}{\partial S^2} = \frac{f_{j,k+1}^{m} - 2f_{j,k}^{m} + f_{j,k-1}^{m}}{(\Delta s)^2}$$

- Second Order Mixed Derivatives:

$$\frac{\partial^2 f}{\partial S \partial V} = \frac{f_{j+1,k+1}^{m} + f_{j-1,k-1}^{m} - f_{j+1,k-1}^{m} - f_{j-1,k+1}^{m}}{4\Delta s \Delta v}$$

Letting $S_k = k\Delta s$ and $V_j = j\Delta v$, Our numeric simulation therefore solves:

$$\begin{aligned}
&\frac{f_{j,k}^{m+1} - f_{j,k}^{m}}{\Delta t} = rS_k \frac{f_{j,k+1}^{m} - f_{j,k-1}^{m}}{2\Delta s} + \kappa(\theta - V_j)\frac{f_{j+1,k}^{m} - f_{j-1,k}^{m}}{2\Delta v} \\
&+ \frac{1}{2}V_j S_k^2 \left(\frac{f_{j,k+1}^{m} - 2f_{j,k}^{m} + f_{j,k-1}^{m}}{(\Delta s)^2} \right) + \frac{1}{2}\eta^2 V_j \left(\frac{f_{j+1,k}^{m} - 2f_{j,k}^{m} + f_{j-1,k}^{m}}{(\Delta v)^2} \right) \\
&+ \rho\eta S_k V_j \left(\frac{f_{j+1,k+1}^{m} + f_{j-1,k-1}^{m} - f_{j+1,k-1}^{m} - f_{j-1,k+1}^{m}}{4\Delta s \Delta v} \right) - r f_{j,k}^{m}
\end{aligned}$$

Letting $S_k = k\Delta s$ and $V_j = j\Delta v$, we find:

$$\begin{aligned}
&\frac{f_{j,k}^{m+1} - f_{j,k}^{m}}{\Delta t} = rk \frac{f_{j,k+1}^{m} - f_{j,k-1}^{m}}{2} + \kappa(\theta - j\Delta v)\frac{f_{j+1,k}^{m} - f_{j-1,k}^{m}}{2\Delta v} \\
&+ \frac{1}{2}jk^2\Delta v \left(f_{j,k+1}^{m} - 2f_{j,k}^{m} + f_{j,k-1}^{m} \right) + \frac{1}{2}\eta^2 j \left(\frac{f_{j+1,k}^{m} - 2f_{j,k}^{m} + f_{j-1,k}^{m}}{\Delta v} \right) \\
&+ \rho\eta jk \left(\frac{f_{j+1,k+1}^{m} + f_{j-1,k-1}^{m} - f_{j+1,k-1}^{m} - f_{j-1,k+1}^{m}}{4} \right) - r f_{j,k}^{m}
\end{aligned}$$

Collecting terms, solving for $f_{j,k}^{m-1}$ we find:

$$\begin{aligned}
f_{j,k}^{m+1} = {} & f_{j,k}^{m} \left(1 - r\Delta t - jk^2\Delta v\Delta t - \frac{\eta^2 j \Delta t}{\Delta v} \right) + \frac{f_{j,k-1}^{m}\Delta t}{2}\left(jk^2\Delta v - rk \right) \\
& + \frac{f_{j,k+1}^{m}\Delta t}{2}\left(rk + jk^2\Delta v \right) + \frac{f_{j-1,k}^{m}\Delta t}{2\Delta v}\left(\eta^2 j - \kappa(\theta - j\Delta v) \right) + \frac{f_{j+1,k}^{m}\Delta t}{2\Delta v}\left(\eta^2 j + \kappa(\theta - j\Delta v) \right) \\
& + \rho\eta jk \left(\frac{f_{j+1,k+1}^{m} + f_{j-1,k-1}^{m} - f_{j+1,k-1}^{m} - f_{j-1,k+1}^{m}}{4} \right)
\end{aligned}$$

Explicit Solver for a European Put

With the above formulation, we need to compute the boundary conditions for a European Put Option with strike price K. For our grid, we will let there be N_s points in S and N_v points in V. We have the following boundaries:

- $S = 0$: As $S \to 0$, the value of the option is the discounted strike price, i.e. $f^t_{j,0} = Ke^{-r(T-t_m)}$
- $S \to \infty$ As $S \to \infty$, it is less and less likely that the option will retain value at expiration, and so the value should decrease. To represent this, we will set the value at the maximal grid point to 0, i.e. $f^m_{j,N_s} = 0$ for all j, m
- $V \to 0$: As V decreases, volatility goes to 0, and we are left with the value of the drift of S_t. This means that at the boundary where $V = 0$, we have that $\partial^2_V f = 0$, and we solve for $j = 1$ in our 2nd order approximation. We find $f^m_{0,k} = 2f^m_{1,k} - f^m_{2,k}$.
- $V \to \infty$: As V gets larger, the change in V does not drive changes in the price significantly. We solve similarly to the previous case, but at $j = N_v - 1$, and find $f^m_{N_v,k} = 2f^m_{N_v-1,k} - f^m_{N_v-2,k}$
- Terminal Condition: At maturity T, the option value must be equal to $(K - S_T)_+ = (K - j\Delta s)_+$ for each value in N_s and N_v.

Note that at each time point $m\Delta t$, we have a $N_s \times N_v$ matrix.

```
explicit_FD_put <- function(dt, ds, dv, K){

  # Hardcoded parameters
  r <- 0.05
  kappa <-1
  theta <- .2
  eta <- .5
  rho <- -0.4
  T <- 1
  S_max <- 2*K
  V_min <- 0.05
  V_max <- 0.6


  time_steps <- as.integer(T/dt)
  s_steps <- round(S_max/ds)+1
  v_steps <- round((V_max - V_min)/dv)+1
  S <- ds *(0:(s_steps-1))
  V <- V_min + dv *(0:(v_steps -1))
  print(ds)

  #Terminal Condition
  term_vals <- pmax(K- S,0)
  f <- matrix(rep(term_vals,v_steps),nrow = v_steps, ncol = s_steps,byrow = TRUE)
  rownames(f) <- V; colnames(f) <- S

  for(i in 1:time_steps){
```

```
    f_upd <- matrix(0,ncol = s_steps,nrow=v_steps)
    for(j_ind in 2:(v_steps-1)){ #Exclude first and last S value
      for( k_ind in 2:(s_steps-1)){  #Exclude first and last v value
        j = V[j_ind]; k = S[k_ind]
        fjk <- 1 - r*dt - j*k^2 *dt *dv - eta^2 * j *(dt/dv)
        fj_km1 <- (dt/2)*(j*k^2*dv - r*k)
        fj_kp1 <- (dt/2)*(r*k + j*k^2*dv)
        fjp1_k <- (dt/(2*dv))*(eta^2*j + kappa*(theta - j*dv))
        fjm1_k <- (dt/(2*dv))*(eta^2*j - kappa*(theta - j*dv))
        cross_same <- rho * eta * j *k *dt
        cross_different <- - cross_same
        f_upd[j_ind,k_ind] <- f[j_ind,k_ind]*fjk +f[j_ind,k_ind-1]*fj_km1 +
          f[j_ind,k_ind+1]*fj_kp1 + f[j_ind+1,k_ind]*fjp1_k +
          f[j_ind-1,k_ind]*fjm1_k +
          cross_same*(f[j_ind+1,k_ind+1]+ f[j_ind-1,k_ind-1])+
          cross_different*(f[j_ind-1,k_ind+1]+ f[j_ind+1,k_ind-1])
      }
    }

    # Enforce Boundary Conditions
    # vol upper
    f_upd[1,2:(s_steps-1)] <- 2*f_upd[2,2:(s_steps-1)]- f_upd[3,2:(s_steps-1)]
    # vol lower
    f_upd[v_steps,2:(s_steps-1)] <- 2*f_upd[v_steps-1,2:(s_steps-1)]-
          f_upd[v_steps-2,2:(s_steps-1)]

    # S lower
    f_upd[,1] <- K*exp(-r * i *dt)

    # S Upper - should be initialized to 0, but reset here to be safe
    f_upd[,s_steps]<-0
    #update F
    f<-f_upd
  }
  return(list(put_price = f,underlying_price = S, vol = V))
}
```

Empirical Convergence Results

```
dv <- 0.05
K <- 100
s0 = 100
v0 = 0.25
dx_list <- c(20,10,5)
```

```r
dt_list <- c(0.0004,0.0002,0.0001)
M = length(dx_list)*length(dt)
pb = txtProgressBar(min=0,M, initial=0)
data_heston<-matrix(0,nrow=length(dt_list),ncol=length(dx_list))
colnames(data_heston)<-dx_list; rownames(data_heston)<-dt_list
ctr<-0
for(i in 1:length(dx_list)){
  for(j in 1:length(dt_list)){
    ctr<- ctr+1
    dx = dx_list[i]
    dt = dt_list[j]
    out<-explicit_FD_put(dt,dx,dv,K)
    s0_ind <- which(out$underlying_price == s0)
    v0_ind <- which(out$vol == v0)
    data_heston[j,i]<-out$put_price[v0_ind,s0_ind]
    setTxtProgressBar(pb,ctr)
  }
}
```

```
[1] 20
==========================[1] 20
==========================[1] 20
==========================[1] 10
[1] 10
[1] 10
[1] 5
[1] 5
[1] 5
```

```r
stargazer(data_heston, type = 'latex', title="Convergence of Explicit PDE Solvers for Heston Model")
```

% Table created by stargazer v.5.2.3 by Marek Hlavac, Social Policy Institute. E-mail: marek.hlavac at gmail.com
% Date and time: Fri, Jun 16, 2023 - 13:29:06

Table 3: Convergence of Explicit PDE Solvers for Heston Model

	20	10	5
0.0004	36.172	27.136	16.127
0.0002	36.172	27.135	16.126
0.0001	36.171	27.134	16.126

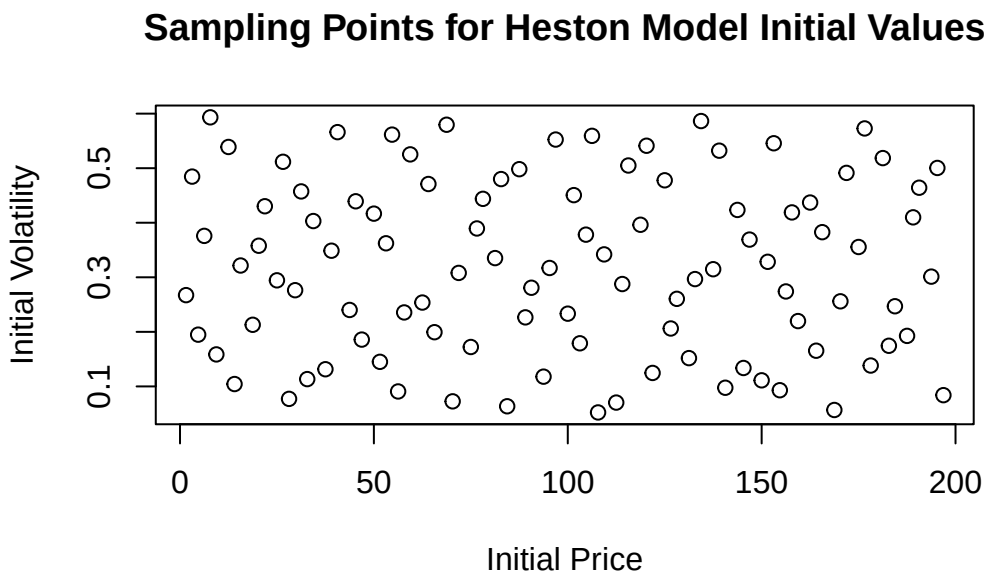
Overall, we find that the the Explicit Model is surprisingly stable for all values of Δt and Δx.

3 Gaussian Process Pricing for Heston Model

```
# Parameters
T <- 1
K <- 100
M <- 10^3
r <- 0.05
kappa <- 0.2
eta <- 0.5
rho <- -0.4
theta <- 0.2

pairs <- 100
sample <- halton(pairs,2)
V_min <- 0.05
V_max <-0.6
S_min = 0
S_max = 2*K

# Scale the samples to the desired range
S0 <- sample[, 1] * (S_max - S_min) + S_min
V0 <- sample[, 2] * (V_max - V_min) + V_min

# Time step
dt <- 0.01
periods <- T/dt

put <- rep(0,pairs)
for(p in 1:pairs){
  S <- matrix(0, nrow = M, ncol = periods + 1)
  V <- matrix(0, nrow = M, ncol = periods + 1)
  S[,1] <- S0[p]
  V[,1] <- V0[p]

  for (m in 1:M) {
    sum <- 0
    for (i in 1:periods) {
      # Generate correlated random variables
      Z1 <- rnorm(n = 1,mean = 0,sd = sqrt(dt))
      Z2 <-rho * Z1 + sqrt(1 - rho ^ 2) * rnorm(n = 1,mean = 0,sd = sqrt(dt))

      # Update V_t (Milstein scheme)
      V[m, i + 1] <-
        V[m, i] + kappa * (theta - V[m, i]) * dt + eta * sqrt(V[m, i]) * Z2 +
        eta ^ 2 / 4 * (Z2 ^ 2 - dt)
```

```
      V[m, i + 1] <-
        max(V[m, i + 1], 0)  # Ensure positivity of V[k, i + 1]

      # Update S_t (Euler scheme)
      S[m, i + 1] <-
        S[m, i] + r * S[m, i] * dt + S[m, i] * sqrt(V[m, i]) * Z1
      S[m, i + 1] <-
        max(S[m, i + 1], 0)  # Ensure positivity of S[k, i + 1]
    }
  }
  put[p] <- mean(exp(-r*T)*pmax(K-S[,periods+1],0))
}
```

```
input <- data.frame(S0=S0,V0=V0)
output <- data.frame(PutMC = put)
plot(input, main = "Sampling Points for Heston Model Initial Values",
      xlab = "Initial Price", ylab="Initial Volatility")
```



GP Surrogate with Squared-Exponential Kernel

We fit a Gaussian Process surrogate model with a Squared-Exponential kernel with $\ell_s = 10$, $\ell_V = 0.1$, and process stadnard deviation of $\eta = 10$.

```
eta <- 10
l_s <- 10
l_v <- 0.1

surfaceSK <- expand.grid(S0 = seq(S_min,S_max, 5), V0 = seq(V_min,V_max,0.05))
gpSK <- km(design=input, response=output, coef.cov=c(l_s,l_v),
          noise.var=rep(1e-7,pairs), coef.trend=eta, control=list(trace=F))

priceSK <- predict.km(gpSK, newdata = surfaceSK, type="SK")
surfaceSK$put <- priceSK$mean

zSK <- matrix(surfaceSK$put, nrow = length(unique(surfaceSK$S0)),
              ncol = length(unique(surfaceSK$V0)),byrow = F)
```

GP Surrogate with a Matern-5/2 Kernel

```
surfaceMat52 <- expand.grid(S0 = seq(S_min, S_max, 5),
                            V0 = seq(V_min, V_max, 0.05))

gpMat52 <- km(
                  design =input,  response=output,
                  # alternatively: use the estimated simulation noise
                  noise.var=rep(1e-7,pairs),
                  covtype="matern5_2",   # can also try "gauss" or "matern3_2"
                  optim.method="gen",
                  estim.method = "MLE",
                  control=list(max.generations=100,pop.size=100,
                               wait.generations=8,
                               solution.tolerance=1e-5,
                               maxit = 1000, trace=F
                  ))

priceMat52 <- predict(gpMat52,newdata = surfaceMat52, type="UK",cov=TRUE)
surfaceMat52$put <- priceMat52$mean

zMat52 <- matrix(surfaceMat52$put, nrow=length(unique(surfaceMat52$S0)), ncol =length(unique(surfaceMat5
```

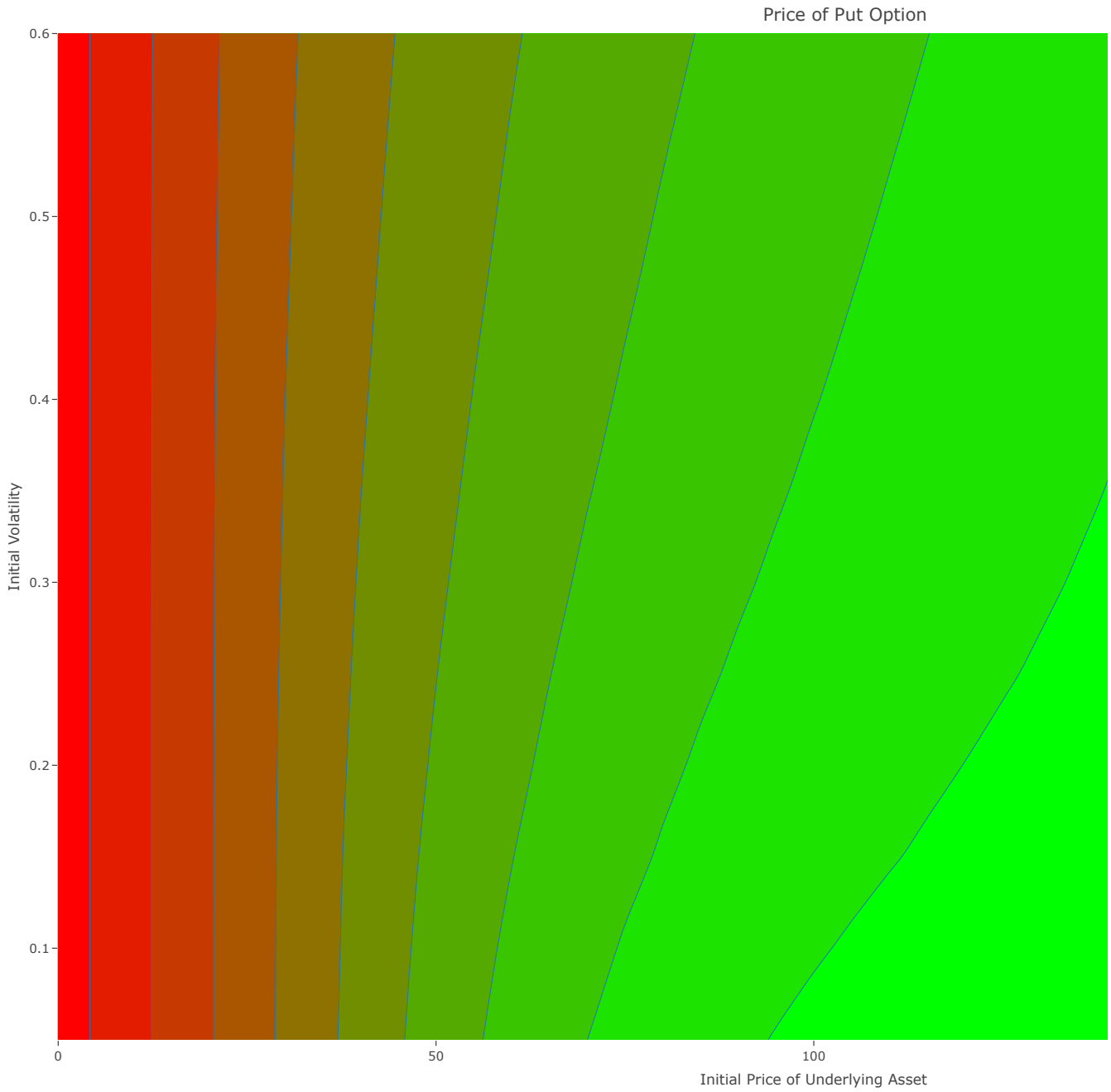
Comparison of Heston PDE and GP pricing Schemes with Square-Exponential and Matern-5/2 Kernels

Taking a small Δt and small Δx, we get the following plot of options prices as a function of S_0 and V_0:

```
# Choose Smallest dt and dx and plot the prices grid
out_FD <- explicit_FD_put(min(dt_list),min(dx_list),dv,K)
```

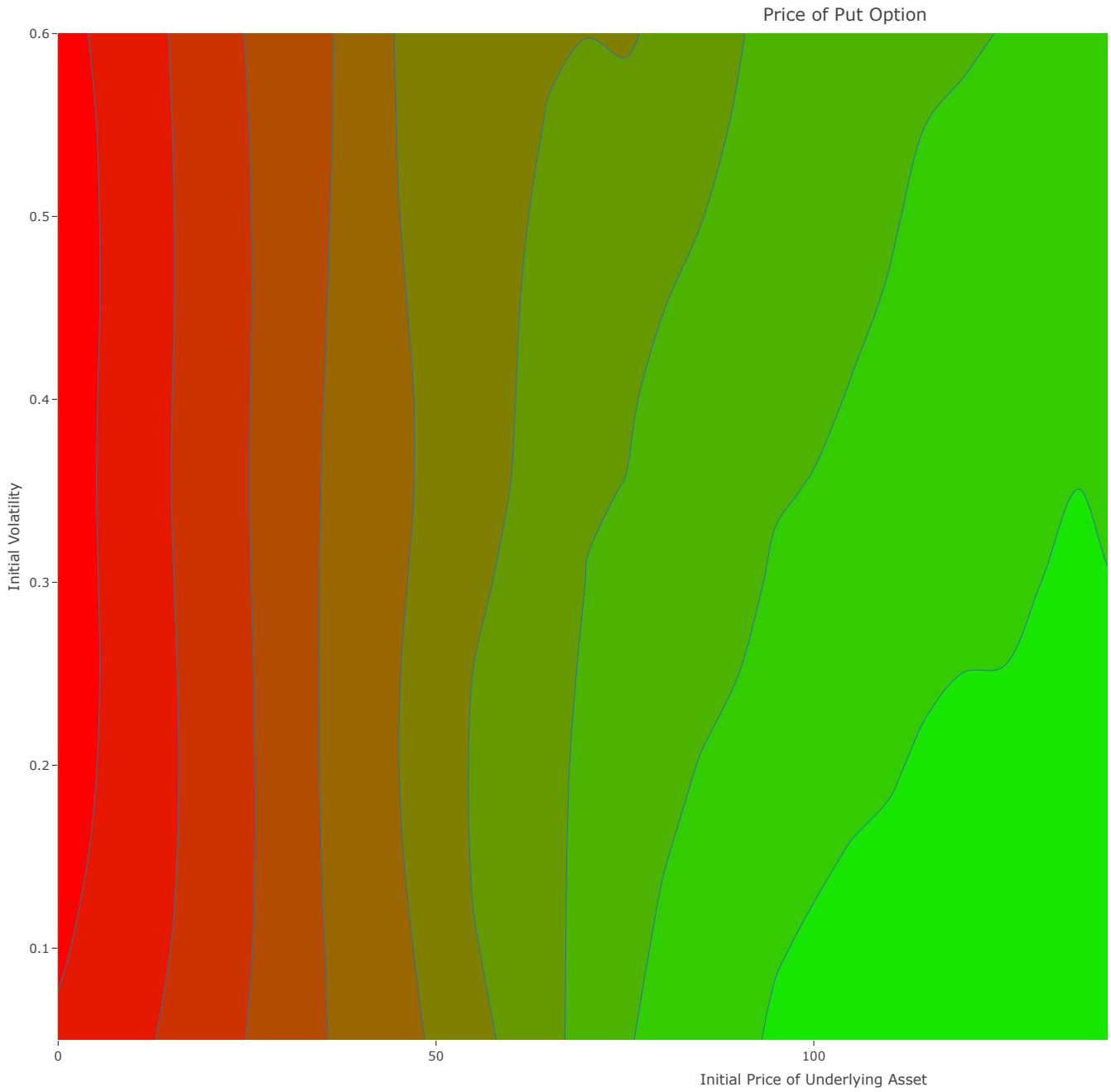
```
[1] 5
```

```
plot_ly(
            x=unique(out_FD$underlying_price),
            y=unique(out_FD$vol),
            z=out_FD$put_price,
            type='contour',
            autocontour = F,
            colors = colorRamp(c("green","red"))

            ) %>%
layout(title = 'Price of Put Option',
            xaxis =list(title = "Initial Price of Underlying Asset"),
            yaxis = list(title = "Initial Volatility")
            ) %>% colorbar(title = "Put Price")
```



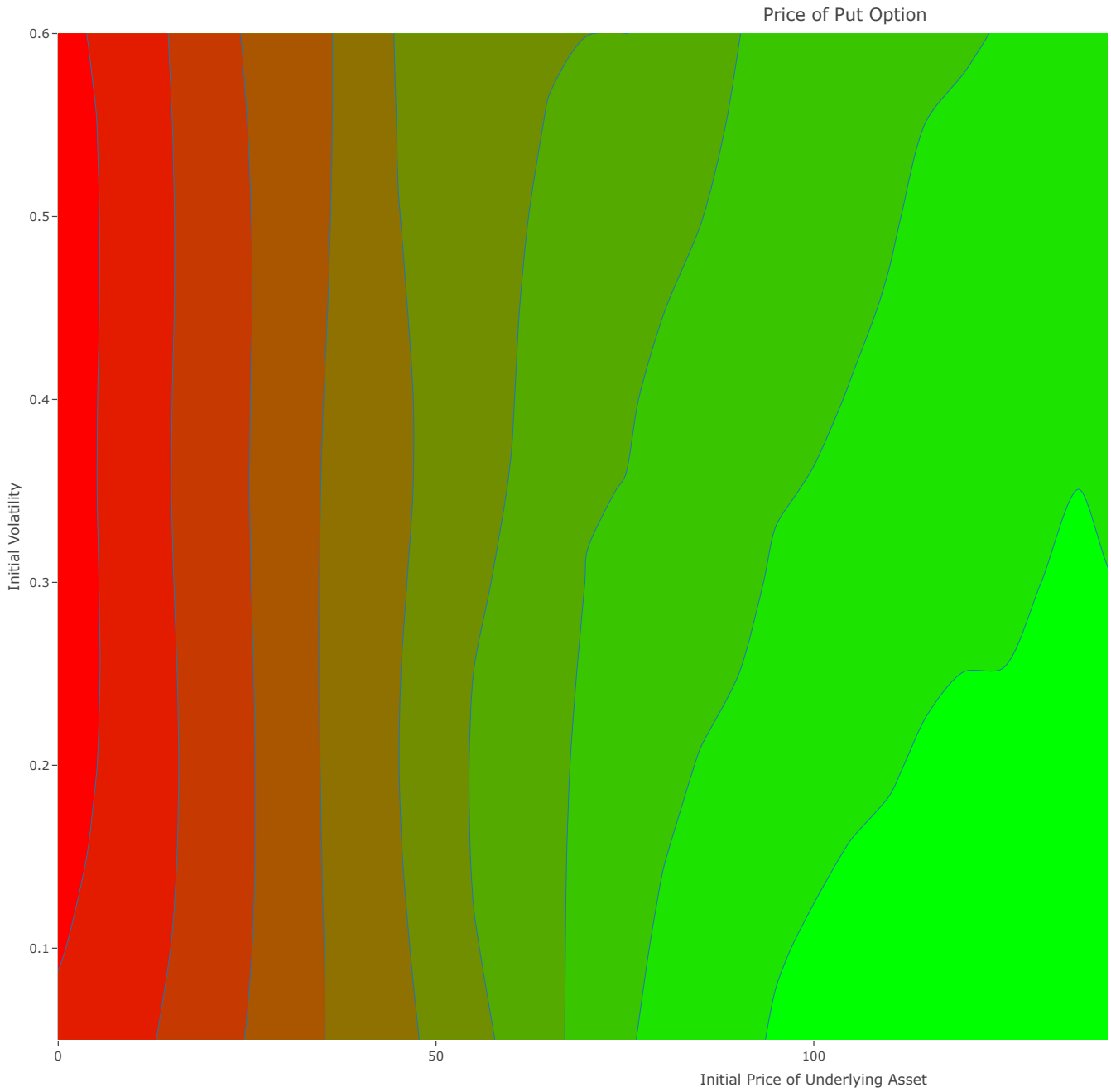

```
plot_ly(
            x=unique(surfaceSK$S0),
```

```
            y=unique(surfaceSK$V0),
            z=t(zSK),
            type='contour',
            autocontour = F,
            colors = colorRamp(c("green","red"))

            ) %>%
layout(title = 'Price of Put Option',
            xaxis =list(title = "Initial Price of Underlying Asset"),
            yaxis = list(title = "Initial Volatility")
            ) %>% colorbar(title = "Put Price")
```




```
plot_ly(
            x=unique(surfaceMat52$S0),
```

```
            y=unique(surfaceMat52$V0),
            z=t(zMat52),
            type='contour',
            autocontour = TRUE,
            contours = list(
              end = 26,
              size = 2,
              start = 2),
            line = list(smoothing = 0.85),
            colors = colorRamp(c("green","red"))

            ) %>%
layout(title = 'Price of Put Option',
            xaxis =list(title = "Initial Price of Underlying Asset"),
            yaxis = list(title = "Initial Volatility")
            ) %>% colorbar(title = "Put Price")
```


Price of Put Option
0.6
0.5
0.4
0.3
0.2
0.1
Initial Volatility
0
50
100
Initial Price of Underlying Asset