ADI methods for option pricing in the Heston model

Ludovic Goudenège

Janvier 2013

Premia 18

We present here two methods for option pricing in the Heston model. The both methods use an alternating directional finite difference scheme. First is a componentwise splitting method developped by Ikonen and Toivanen in [2], the second is an ADI finite difference scheme developed by In 't Hout and Foulon in [1] and [3] .

1 Componentwise splitting methods for pricing american options under stochastic volatility

1.1 Model

Let $u(s,v,t)$ denote the price of a European option if at time $T-t$ the underlying asset price equals s and its variance equals v, where T is the given maturity time of the option. Heston's stochastic volatility model implies that u satisfies the parabolic PDE

$$\frac{\partial u}{\partial t} = \frac{1}{2}s^2 v\frac{\partial^2 u}{\partial s^2} + \rho\gamma sv\frac{\partial^2 u}{\partial s\partial v} + \frac{1}{2}\gamma^2 v\frac{\partial^2 u}{\partial v^2} + (r-d)s\frac{\partial u}{\partial s} + \alpha(\beta - v)\frac{\partial u}{\partial v} - ru, \tag{1}$$

for $0 \leq t \leq T$, $s > 0$ and $v > 0$. The parameter $\alpha > 0$ is the mean-reversion rate, β is the long term mean, $\gamma > 0$ is the volatility-of-variance, $\rho \in [-1,1]$ is the correlation between the two underlying Brownian motions, r denote the interest rate and d the dividend. For this numerical method we always assume that $2\alpha\beta > \gamma^2$, which is the Feller condition.

The spatial domain is restricted to a bounded set $[0, S_{\max}]\times[0, V_{\max}]$ with fixed sufficiently large values $S_{\max}$ and $V_{\max}$. The grids $0 = y_0 < y_1 < \cdots < y_N = V_{\max}$ and $0 = x_0 < x_1 < \cdots < x_M$ are defined as described in [2]. We use parameters `h`, `kappa` and `coeff_for_x_1` defined as

$$h = \frac{V_{\max}}{N}, \quad \kappa = \frac{h_r(S_{\max})}{h_r(\texttt{strike})} \quad \text{and} \quad x_1 = \frac{h}{\texttt{coeff\_for\_x\_1}}.$$

For a European call or put option, we have the boundary conditions

$$\begin{aligned}
u(s,v,0) &= \max(0, \phi(s-K)),\\
u(0,v,t) &= \frac{1-\phi}{2}K\exp(-rt),\\
\frac{\partial u}{\partial s}(S_{\max},v,t) &= \frac{1+\phi}{2}\exp(-dt),\\
u(s,V_{\max},t) &= s\exp(-dt)\mathbb{1}_{+1}(\phi) + K\exp(-rt)\mathbb{1}_{-1}(\phi).
\end{aligned}$$

where $K > 0$ is the given strike price of the option and ϕ denotes the binary variable taking the values $+1$ for a call and -1 for a put. But we have also implemented other boundary conditions.

For a American call or put option, Ikonen and Toivanen recommend

$$\begin{aligned} u(s,v,0) &= \max(0,\phi(s-K)), \\ u(0,v,t) &= \frac{1-\phi}{2}K, \\ u(s,0,t) &= \max(0,\phi(s-K)), \end{aligned}$$

where $K > 0$ is the given strike price of the option. On the boundaries $s = S_{\max}$ and $v = V_{\max}$, they recommend a Neumann condition.

1.2 Finite difference schemes

We follow [2] for the finite difference schemes on the intermediate form

$$\begin{aligned} \frac{\partial u}{\partial t} &+ \left[-\frac{1}{2}vs^2 + \omega\rho\gamma vs\frac{h_l}{2h} + (1-\omega)\rho\gamma vs\frac{h_r}{2h}\right]\frac{\partial^2 u}{\partial s^2} \\ &+ \left[-\frac{1}{2}\gamma^2 v + \omega\rho\gamma vs\frac{h}{2h_l} + (1-\omega)\rho\gamma vs\frac{h}{2h_r}\right]\frac{\partial^2 u}{\partial v^2} \\ &+ \left[-rs - \omega\rho\gamma vs\frac{1}{h} + (1-\omega)\rho\gamma vs\frac{1}{h}\right]\frac{\partial u}{\partial s} \\ &+ \left[-\alpha(\beta - v) - \omega\rho\gamma vs\frac{1}{h_l} + (1-\omega)\rho\gamma vs\frac{1}{h_r}\right]\frac{\partial u}{\partial v} \\ &+ \left[r + \omega\rho\gamma vs\frac{1}{h_l h} + (1-\omega)\rho\gamma vs\frac{1}{h_r h}\right]u \\ &- \omega\rho\gamma vs\frac{1}{h_l h}u(s-h_l, v-h) - (1-\omega)\rho\gamma vs\frac{1}{h_r h}u(s+h_r, v+h). \end{aligned}$$

On the strict interior of the grid, each spatial derivative is replaced by the finite difference scheme described in [2] in section 6.

For the European options, at the boundary $s = 0$ or $v = V_{\max}$, the solution is given by the Dirichlet conditions. The boundary $v = 0$ is treated as an outflow boundary, so the derivative $\partial u/\partial v$ is approximated using the upwind scheme and all other derivatives vanish. At the boundary $s = S_{\max}$, we have a Neumann condition treated in the procedure `construction_splitting_matrix_neumann`.

For the American options, at the boundary $s = 0$ or $v = 0$, the solution is given by the Dirichlet conditions. At the boundary $s = S_{\max}$ or $v = V_{\max}$, we have a Neumann condition treated in the procedure `construction_splitting_matrix_neumann`.

We use the proposed scheme with a fixed time step Δt and with temporal grid points given by $t_n = n\Delta t$ for $n = 0, 1, 2, \ldots, L$. Moreover, we can used two backward Euler steps with an half time step $\Delta t/2$ as suggested in the article (Rannacher's idea). But it slows down the computation when the grids are too large.

1.3 Variables

We give here a list of the "global" variables used in the scheme.

```
double x = 100.0; // Asset price
double y = 0.01; // Volatility^2 = Variance
double t = 1.0; // Maturity
double r = log(1+10.0/100.0); // Rate = 10% -> r = log(1+Rate) = log(1.1)
double divid = 0.0; // Dividend
double alpha = 2.0; // Mean-reversion rate
double beta = 0.01; // Long term mean
double gamma = 0.2; // Volatility-of-variance
double rho = 0.5; // Correlation
double strike = 100.0; // Strike
```

```
int call_or_put = -1; // Binary variable

// Parameters for the grid and the computation.
int mt = 100; // Number of points for the spot grid
int N = 50; // Number of points for the variance grid
int L = 1000; // Number of points for the temporal grid
double X = 800.0; // Variable Smax
double Y = 5.0; // Variable Vmax
double kappa = 5.0; // Parameter kappa for the grid computation
double coeff_for_x_1 = 20.0; // Parameter for the grid computation -> x_1 = h/coeff
double h = Y/((double)(N)); // Size of the step of the uniform grid of volatility.
double omega = 0.5;  // Omega parameter for the scheme. Should be between 0 and 1.
```

We recommend `mt` $= 101$, `N` $= 50$ and `L` $= 1000$. Moreover `X` should be chosen as `X` $= 8$`x` in order to ensure a sufficiently large grid. In all cases `Y` $= 1$ is a good parameter.

1.4 Procedures and functions

`size_grid_generation_IT` : Procedure to compute the size of the grid of the spot variable using the Ikonen and Toivanen algorithm.

`grid_generation_IT` : Procedure to compute the grid of the spot variable using the Ikonen and Toivanen algorithm.

`grid_generation_uniform` : Procedure to compute the uniform grid of the variance variable.

`lower_index(double *grid, int size, double value)` :
Function to find (in the tabular `grid` of size `size`) the index of a given value (`value`) or the biggest index of an element in `grid` which is inferior to the given value.

`reorder_unknowns_x_to_xy(PnlVect *Unknowns, int M, int N, PnlVect *Sortie)` :
Procedure to reorder a tabular `Unknows`. The output tabular is `Sortie`.

`reorder_unknowns_xy_to_y(PnlVect *Unknowns, int M, int N, PnlVect *Sortie)` :
Procedure to reorder a tabular `Unknows`. The output tabular is `Sortie`.

`reorder_unknowns_y_to_x(PnlVect *Unknowns, int M, int N, PnlVect *Sortie)` :
Procedure to reorder a tabular `Unknows`. The output tabular is `Sortie`.

`print_unknowns(PnlVect *Unknowns, int M, int N)` :
Procedure to print the vector Unknowns. This procedure is not used in the algorithm, but it is for debugging.

`print_unknowns_cross(PnlVect *Unknowns, int M, int N)` :
Procedure to print the vector Unknowns if the order is `xy` (for instance after using procedure `reorder_unknowns_x_to_xy`). This procedure is not used in the algorithm, but it is for debugging.

```
construction_Ax_coefficients(
double r, double divid,
double alpha, double beta, double gamma, double rho,
double X, int M, double *pointsx, int index_pointx,
double Y, int N, double *pointsy, int index_pointy,
double omega, double *lower_d, double *diagonal, double *upper_d) :
```
Procedure to compute the full matrix A (output is the `PnlMat` object `MatrixTotal`) and the mixted matrix A_0 (output is the `PnlMat` object `MatrixMixted`) which contains only the mixted terms of the PDE. Remark : The Douglas scheme does not used this mixted matrix, but other schemes do.

This procedure uses the variables of the PDE with the same name (r, divid, alpha, beta, gamma and rho) and the grids (pointsx and pointsy) with their sizes (M and N).

construction_Ay_coefficients(
double r, double divid,
double alpha, double beta, double gamma, double rho,
double X, int M, double *pointsx, int index_pointx,
double Y, int N, double *pointsy, int index_pointy,
double omega, double *lower_d, double *diagonal, double *upper_d) :
Procedure to compute the full matrix A (output is the PnlMat object MatrixTotal) and the mixted matrix A_0 (output is the PnlMat object MatrixMixted) which contains only the mixted terms of the PDE. Remark : The Douglas scheme does not used this mixted matrix, but other schemes do. This procedure uses the variables of the PDE with the same name (r, divid, alpha, beta, gamma and rho) and the grids (pointsx and pointsy) with their sizes (M and N).

construction_Axy_coefficients(
double r, double divid,
double alpha, double beta, double gamma, double rho,
double X, int M, double *pointsx, int index_pointx,
double Y, int N, double *pointsy, int index_pointy,
double omega, double *lower_d, double *diagonal, double *upper_d) :
Procedure to compute the full matrix A (output is the PnlMat object MatrixTotal) and the mixted matrix A_0 (output is the PnlMat object MatrixMixted) which contains only the mixted terms of the PDE. Remark : The Douglas scheme does not used this mixted matrix, but other schemes do. This procedure uses the variables of the PDE with the same name (r, divid, alpha, beta, gamma and rho) and the grids (pointsx and pointsy) with their sizes (M and N).

modify_solution_with_neumann_boundary_condition(
double r, double divid, double time,
int M, double *pointsx, int N, double *pointsy, double coeff,
PnlVect *VectOne, PnlVect *Sortie) :
Procedure to modify the boundary coefficients of a vector VectOne in order to take account of the Neumann condition. The output is in Sortie.

do_rannacher_iteration(double r, double divid, double time,
double alpha, double beta, double gamma, double rho,
int M, double *pointsx, int N, double *pointsy,
double coeff, PnlVect *VectTwo, PnlVect *Sortie) :
Procedure to compute tow backward Euler time steps with $\Delta t/2$. The output is in Sortie.

ComSplitEu(
double x, double y,
double t, double r, double divid, double alpha, double beta, double gamma, double rho,
double E, double X, double Y, int mt, int N, int L, double kappa, double coeff_for_x_1,
double omega_global,int boundary_conditions_method, int call_or_put, double *ptprice,double
*ptdelta) :
Procedure to compute the temporal loop in the European case.
ComSplitAm(
double x, double y,
double t, double r, double divid, double alpha, double beta, double gamma, double rho,
double E, double X, double Y, int mt, int N, int L, double kappa, double coeff_for_x_1,
double omega_global,int boundary_conditions_method, int call_or_put, double *ptprice,double
*ptdelta) :
Procedure to compute the temporal loop in the American case.

1.5 Complete algorithm

- First we create the variables.
- We define the terminal value of the option.
- We build the matrix A_x, A_y and A_{xy}.
- We start a loop on time.
  - If it is the first time step, we (can) do two backward Euler iterations with $\Delta t/2$.
  - Else
    - We do the first step with matrix A_x using Crank-Nicolson scheme.
    - We reorder the vector in order xy.
    - We do the second step with matrix A_{xy} using Crank-Nicolson scheme.
    - We reorder the vector in order y.
    - We do the third step with matrix A_y using Crank-Nicolson scheme.
    - We reorder the vector in order x.
- End of loop on time.
- Finally, we do an interpolation to find the value of the option at point (x, y).

In the American case, at each time iteration, we have added a step which computes the maximum between the current value and the payoff function.

1.6 Conclusion

The scheme proposed by Ikonen and Toivanen is easy to understand. But the matrix are difficult to build, and it is not really clear how to treat the boundary conditions. Indeed the unknowns $u^{(k+1/3)}$ is not a "consistent" unknowns (it is only a temporary unknown), so it should not verify the boundary conditions of $u^{(k)}$. In the article, there is no explanation on this difficulty.

Moreover the grid for the spot variable needs some additional requirements with undesired effects. For instance the grid on the variance variable is uniform, so the grid is not really precise near the interest points. Furthermore the conditions for the non positiveness of the off-diagonal elements seem wrong in the article (we have changed it in the algorithm). If $\rho < 0$, the algorithm should be changed (the conditions for the non positiveness are changed and it has not been implemented).

We have not implemented the Strang symmetrized splitting method or Euler algorithm, but all the procedure have been written.

In the American case, we have not implemented the Brennan-Schwartz algorithm, but we have compared the solution with the payoff at each time iteration. It gives good results without difficulties.

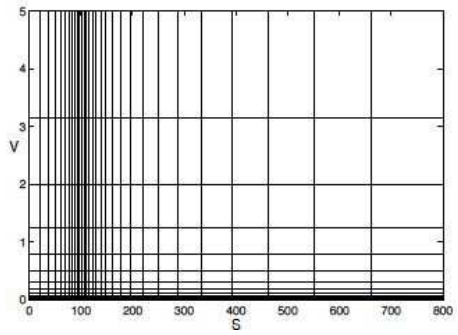
2 ADI finite difference schemes for option pricing in the Heston model with correlation

2.1 Model

Let $u(s, v, t)$ denote the price of a European option if at time $T-t$ the underlying asset price equals s and its variance equals v, where T is the given maturity time of the option. Heston's stochastic volatility model implies that u satisfies the parabolic PDE

$$\frac{\partial u}{\partial t} = \frac{1}{2}s^2 v\frac{\partial^2 u}{\partial s^2} + \rho\gamma s v\frac{\partial^2 u}{\partial s \partial v} + \frac{1}{2}\gamma^2 v\frac{\partial^2 u}{\partial v^2} + (r-d)s\frac{\partial u}{\partial s} + \alpha(\beta - v)\frac{\partial u}{\partial v} - ru, \tag{2}$$

for $0 \leq t \leq T$, $s > 0$ and $v > 0$. The parameter $\alpha > 0$ is the mean-reversion rate, β is the long term mean, $\gamma > 0$ is the volatility-of-variance, $\rho \in [-1, 1]$ is the correlation between the two underlying Brownian motions, r denote the interest rate and d the dividend. For this numerical method we always assume that $2\alpha\beta > \gamma^2$, which is the Feller condition.



The spatial domain is restricted to a bounded set $[0, S_{\max}] \times [0, V_{\max}]$ with fixed sufficiently large values $S_{\max}$ and $V_{\max}$. We take $S_{\max} = 8K$ and $V_{\max} = 5$. For a European call or put option, we have the boundary conditions

$$\begin{aligned} u(s, v, 0) &= \max(0, \phi(s - K)), \\ u(0, v, t) &= \frac{1 - \phi}{2} K \exp(-rt), \\ \frac{\partial u}{\partial s}(S_{\max}, v, t) &= \frac{1 + \phi}{2} \exp(-dt), \\ u(s, V_{\max}, t) &= s \exp(-dt) \mathbb{1}_{+1}(\phi) + K \exp(-rt) \mathbb{1}_{-1}(\phi). \end{aligned}$$

where $K > 0$ is the given strike price of the option and ϕ denotes the binary variable taking the values $+1$ for a call and -1 for a put.

For a American call or put option, (following [2]) we recommend

$$\begin{aligned} u(s, v, 0) &= \max(0, \phi(s - K)), \\ u(0, v, t) &= \frac{1 - \phi}{2} K, \\ u(s, 0, t) &= \max(0, \phi(s - K)), \end{aligned}$$

where $K > 0$ is the given strike price of the option. On the boundaries $s = S_{\max}$ and $v = V_{\max}$, we recommend a Neumann condition.

Let $M \leq 1$ be an integer and $c > 0$ a constant, let equidistant points $\xi_0 < \xi_1 < \cdots < \xi_M$ given by

$$\xi_i = \sinh^{-1}(-K/c) + i\Delta\xi, \quad 0 \leq i \leq M,$$

with

$$\Delta\xi = \frac{1}{M} \left(\sinh^{-1}((S_{\max} - K)/c) - \sinh^{-1}(-K/c) \right).$$

Then a non-uniform mesh $0 < s_0 < s_1 < \cdots < s_M = S_{\max}$ is defined through the transformation

$$s_i = K + c \sinh(\xi_i), \quad 0 \leq i \leq M.$$

For the v-direction, we choose $N \leq 1$ and a constant $d > 0$. Consider equidistant points given by $\eta_j = j\Delta\eta$ for $0 \leq j \leq N$ with

$$\Delta\eta = \frac{1}{N} \sinh^{-1}(V_{\max}/d).$$

Then we define a mesh $0 = v_0 < v_1 < \cdots < v_N = V_{\max}$ through

$$v_j = d \sinh(\eta_j), \quad 0 \leq j \leq N.$$

Figure 1 displays the spatial grid obtained for $S_{\max} = 800$, $V_{\max} = 5$, $K = 100$, $c = K/5$, $d = V_{\max}/500$, $M = 30$ and $N = 15$.

2.2 Finite difference schemes

We follow [1] for the finite difference schemes. On the strict interior of the grid, each spatial derivative is replaced by its corresponding central finite difference scheme. Except in the region $v > 1$ where we apply upwind scheme for $\partial u/\partial v$ whenever the flow in the v-direction is towards $v = V_{\max}$. At the boundary $s = 0$ or $v = V_{\max}$, the solution is given by the Dirichlet condition. The boundary $v = 0$ is treated as an outflow boundary, so the derivative $\partial u/\partial v$ is approximated using the upwind scheme and all other derivatives vanish. At the boundary $s = S_{\max}$, we use the Neumann condition to simplify the scheme. The derivative $\partial u/\partial s$ is directly given and we approximate $\partial^2 u/\partial s^2$ using a virtual point $\tilde{s} = 2s_M - s_{M-1}$ where the value at this point is given by

$$u(\tilde{s}, v, t) = u(s_M, v, t) + \frac{1+\phi}{2}\exp(-dt)(\tilde{s} - s_M) = u(s_M, v, t) + \frac{1+\phi}{2}\exp(-dt)(s_M - s_{M-1}).$$

So the scheme (in the s-direction) is given by

$$\begin{aligned}
\frac{\partial u}{\partial t} &= \frac{1}{2}s^2 v\left(\frac{1}{(s_M - s_{M-1})^2}u(s_{M-1}, v, t) - \frac{2}{(s_M - s_{M-1})^2}u(s_M, v, t) + \frac{1}{(s_M - s_{M-1})^2}u(\tilde{s}, v, t)\right) \\
&\quad + (r-d)s\frac{1+\phi}{2}\exp(-dt) + \frac{1}{2}\gamma^2 v\frac{\partial^2 u}{\partial v^2} + \alpha(\beta - v)\frac{\partial u}{\partial v} - ru \\
&= \frac{1}{2}s^2 v\left(\frac{1}{(s_M - s_{M-1})^2}u(s_{M-1}, v, t) - \frac{1}{(s_M - s_{M-1})^2}u(s_M, v, t)\right) \\
&\quad + \left(\frac{s^2 v}{2(s_M - s_{M-1})} + (r-d)s\right)\frac{1+\phi}{2}\exp(-dt) \\
&\quad + \frac{1}{2}\gamma^2 v\frac{\partial^2 u}{\partial v^2} + \alpha(\beta - v)\frac{\partial u}{\partial v} - ru.
\end{aligned}$$

The term $\left(\frac{s^2 v}{2(s_M - s_{M-1})} + (r-d)s\right)\frac{1+\phi}{2}\exp(-dt)$ explicitly appears in the procedures which solve linear systems.

We use ADI schemes with a fixed time step Δt and with temporal grid points given by $t_n = n\Delta t$ for $n = 0, 1, 2, \ldots, L$. We have implemented the Douglas scheme (with parameter $\theta = 1/2$), but the Craig-Sneyd, modified Craig-Sneyd and Hundsdorfer-Verwer schemes can also be treated in the same way. Moreover, we have used two backward Euler steps with an half time step $\Delta t/2$ as suggested in the article (Rannacher's idea).

In the American case, we have implemented the dynamics programming algorithm and not the algorithm implemented in [3].

2.3 Variables

We give here a list of the "global" variables used in the scheme.

```
double x = 100.0; // Asset price
double y = 0.01; // Volatility^2 = Variance
double t = 1.0; // Maturity
double r = log(1+10.0/100.0); // Rate = 10% -> r = log(1+Rate) = log(1.1)
double divid = 0.0; // Dividend
double alpha = 2.0; // Mean-reversion rate
double beta = 0.01; // Long term mean
double gamma = 0.2; // Volatility-of-variance
double rho = 0.5; // Correlation
double strike = 100.0; // Strike
int call_or_put = -1; // Binary variable

// Parameters for the grid and the computation.
```

```
int M = 80; // Number of points for the spot grid
int N = 20; // Number of points for the variance grid
int L = 20000; // Number of points for the temporal grid
double X = 800.0; // Variable Smax
double Y = 5.0; // Variable Vmax
double theta = 0.5;  // Theta parameter for the Douglas scheme. Should be between 0 and 1.
int computation_lu = 1; // Make the computation of the LU decomposition only once.
int scheme = 0; // The ADI scheme. 0 is for Douglas scheme.
int am = 0; // Flag for the american options (European = 0 and American = 1).
```

We recommend `M` $= 80$, `N` $= 20$ and `L` $= 20000$. Moreover `X` should be chosen as `X` $= 8$`x` in order to ensure a sufficiently large grid. In all cases `Y` $= 5$ is a good parameter. The computation is accelerated as the LU decomposition is pre-calculated, so we recommend `computation_lu` $= 1$. The only implemented scheme is the Douglas scheme (i.e. `scheme` $= 0$). Finally the parameter `theta` should be $\geq 1/2$.

2.4 Procedures and functions

`asinh1` : Function to compute the hyperbolic arcsinus.

`grid_generation_HF_spot` : Procedure to compute the grid of the spot variable using the t'Hout and Foulon algorithm.

`grid_generation_HF_variance` : Procedure to compute the grid of the variance variable using the t'Hout and Foulon algorithm.

`lower_index(double *grid, int size, double value)` :
Function to find (in the tabular `grid` of size `size`) the index of a given value (`value`) or the biggest index of an element in `grid` which is inferior to the given value.

`reorder_unknowns_x_to_y(PnlVect *Unknowns, int M, int N, PnlVect *Sortie)` :
Procedure to reorder a tabular `Unknows` ordered in increasing spot then increasing variance. The output tabular is `Sortie`.

`reorder_unknowns_y_to_x(PnlVect *Unknowns, int M, int N, PnlVect *Sortie)` :
Procedure to reorder a tabular `Unknows` ordered in increasing variance then increasing spot. The output tabular is `Sortie`.

```
construction_matrix_A_and_A0(
double r, double divid,
double alpha, double beta, double gamma, double rho,
int M, double *pointsx, int N, double *pointsy,
PnlMat *MatrixTotal, PnlMat *MatrixMixted) :
```
Procedure to compute the full matrix A (output is the `PnlMat` object `MatrixTotal`) and the mixted matrix A_0 (output is the `PnlMat` object `MatrixMixted`) which contains only the mixted terms of the PDE. Remark : The Douglas scheme does not used this mixted matrix, but other schemes do. This procedure uses the variables of the PDE with the same name (`r`, `divid`, `alpha`, `beta`, `gamma` and `rho`) and the grids (`pointsx` and `pointsy`) with their sizes (`M` and `N`).

```
construction_matrix_A1(
double r, double divid,
double alpha, double beta, double gamma, double rho,
int M, double *pointsx, int N, double *pointsy,
PnlTridiagMat *MatrixX) :
```
Procedure to compute the matrix A_1 (output is the `PnlTridiagMat` object `MatrixX`) which contains only the terms in the s-direction (spot grid) of the PDE. This is a tridiagonal matrix because

we only use centered schemes.
This procedure uses the variables of the PDE with the same name (`r`, `divid`, `alpha`, `beta`, `gamma` and `rho`) and the grids (`pointsx` and `pointsy`) with their sizes (`M` and `N`).

`construction_matrix_A2(`
`double r, double divid,`
`double alpha, double beta, double gamma, double rho,`
`int M, double *pointsx, int N, double *pointsy,`
`PnlBandMat *MatrixY)` :
Procedure to compute the matrix A_2 (output is the `PnlBandMat` object `MatrixY`) which contains only the terms in the v-direction (variance grid) of the PDE. This is a band matrix because we use different schemes. The size of the band is less than 5.
This procedure uses the variables of the PDE with the same name (`r`, `divid`, `alpha`, `beta`, `gamma` and `rho`) and the grids (`pointsx` and `pointsy`) with their sizes (`M` and `N`).

`computation_explicit_syslin_total_matrix(`
`double r, double divid, double time, int call_or_put, double strike,`
`int M, double *pointsx, int N, double *pointsy,`
`PnlVect *VectOne, double coeff,`
`PnlMat *Matrix, PnlVect *VectTwo, PnlVect *Sortie)` :
Procedure to compute the solution `Sortie` of (Id + `coeff * Matrix`) `Sortie` = `VectTwo`. `Matrix` is a full matrix. This procedure uses the variables of the PDE with the same name (`r`, `divid`, `alpha`, `beta`, `gamma` and `rho`) and the grids (`pointsx` and `pointsy`) with their sizes (`M` and `N`).But it also uses the variables `time`, `call_or_put` and `strike` in order to compute the boundary conditions.

`computation_explicit_syslin_tridiag_matrix_without_boundary_conditions(`
`const PnlVect *VectOne, double coeff,`
`PnlTridiagMat *Matrix, const PnlVect *VectTwo, PnlVect *Sortie)` :
Procedure to compute the solution `Sortie` = `VectOne` + `coeff * Matrix * VectTwo`. `Matrix` is a tridiagonal matrix.

`computation_explicit_syslin_band_matrix(`
`const PnlVect *VectOne, double coeff,`
`PnlBandMat *Matrix, const PnlVect *VectTwo, PnlVect *Sortie)` :
Procedure to compute the solution `Sortie` = `VectOne` + `coeff * Matrix * VectTwo`. `Matrix` is a band matrix.

`modify_solution_with_neumann_boundary_condition(`
`double r, double divid, double time,`
`int M, double *pointsx, int N, double *pointsy, double coeff,`
`PnlVect *VectOne, PnlVect *Sortie)` :
Procedure to modify the boundary coefficients of a vector `VectOne` in order to take account of the Neumann condition. The output is in `Sortie`. This procedure is essentially used after the procedure `computation_explicit_syslin_tridiag_matrix_without_boundary_conditions`.

`do_rannacher_iteration(` `double r, double divid, double time,`
`double alpha, double beta, double gamma, double rho,`
`int M, double *pointsx, int N, double *pointsy,`
`double coeff, PnlVect *VectTwo, PnlVect *Sortie)` :
Procedure to compute tow backward Euler time steps with $\Delta t/2$. The output is in `Sortie`.

`computation_implicit_syslin_tridiag_matrix_without_boundary_conditions(`
`int M, int N,`
`double coeff, PnlTridiagMat *Matrix, PnlVect *VectTwo, PnlVect *Sortie)` :
Procedure to compute the solution `Sortie` of (Id + `coeff * Matrix`) `Sortie` = `VectTwo`. `Matrix`

is a tridiagonal matrix.

`computation_implicit_syslin_band_matrix(`
`int M, int N,`
`double coeff, PnlBandMat *Matrix, PnlVect *VectTwo, PnlVect *Sortie)` :
Procedure to compute the solution `Sortie` of (Id + `coeff * Matrix`) `Sortie` = `VectTwo`. `Matrix` is a band matrix.

`computation_lu_implicit_band_matrix(`
`int M, int N,`
`double coeff, PnlBandMat *Matrix, PnlBandMat *Working_Matrix, PnlVectInt *p)` :
Procedure to compute the LU decomposition of (Id + `coeff * Matrix`). The output is in `Working_Matrix` (it uses a permutation vector `p`).

`computation_implicit_syslin_band_matrix_using_lu(`
`const PnlBandMat *Working_Matrix,PnlVectInt *p,PnlVect *VectTwo,PnlVect *Sortie)` :
Procedure to compute the solution of a system with an already computed LU decomposition.

`AdiHoutFoulon (`
`double x, double y, double t,`
`double r, double divid, double alpha, double beta, double gamma, double rho,`
`double strike, double X, double Y, int M, int N, int L,`
`double theta, int computation_lu, int scheme, int call_or_put,`
`double *ptprice,double *ptdelta)` :
Procedure to compute the temporal loop (see next section for the details of the algorithm).

2.5 Complete algorithm

- First we create the variables.
- We define the terminal value of the option.
- We build the matrix A, A_0, A_1 and A_2.
- We compute the LU decomposition of the matrix A_2.
- We start a loop on time.
  - If it is the first time step, we do two backward Euler iterations with $\Delta t/2$.
  - Else
    - We do the first step of the Douglas scheme :
      - Explicit linear system $U_{n-1} + \Delta t F(t_{n-1}, U_{n-1})$.
      - Dirichlet boundary conditions.
    - We do the second step of the Douglas scheme :
      - Explicit linear system $Z \leftarrow Y_0 - \theta \Delta t F_1(t_{n-1}, U_{n-1})$.
      - Implicit linear system $Y_1 - \theta \Delta t F_1(t_n, Y_1) = Z$.
      - Dirichlet boundary conditions.
    - We reorder the vector. The unknowns are now ordered with increasing variance then increasing spot.
    - We do the third step of the Douglas scheme :
      - Explicit linear system $Z \leftarrow Y_1 - \theta \Delta t F_2(t_{n-1}, U_{n-1})$.
      - Implicit linear system $Y_2 - \theta \Delta t F_2(t_n, Y_2) = Z$.
      - Dirichlet boundary conditions.
    - We reorder the vector. The unknowns are now ordered with increasing spot then increasing variance.
- End of loop on time.
- Finally, we do an interpolation to find the value of the option at point (x, y).

2.6 Conclusion

References

[1] K.J. in 't Hout and S. Foulon. Adi finite difference schemes for option pricing in the heston model with correlation. *Int. J. Numer. Anal. Mod.*, 7:303–320, 2010. 1, 7

[2] S.Ikonen J.Toivanen. Componentwise splitting methods for pricing american options under stochastic volatility. *International Journal of Theoretical and Applied Finance*, 2:331–361, 2007. 1, 2, 6

[3] K.J. in 't Hout T. Haentjens and K. Volders. Adi schemes with ikonen-toivanen splitting for pricing american put options in the heston model. *Numerical Analysis and Applied Mathematics, eds. T. E. Simos et. al., AIP Conf. Proc.*, 1281:231–234, 2010. 1, 7