A Gauss Implementation of Option Pricing and Hedging
The OPTION library

Thierry Roncalli

University of Evry

This version: September 5, 2008

Contents

Chapter 1

Introduction

1.1 Installation

1. The file *option.zip* is a zipped archive file. Copy this file under the root directory of Gauss, for example **D:\GAUSS60**.

2. Unzip the file. Directories will then be created and files will be copied over them:

target_path	*readme.txt*
*target_path***dlib**	DLLs
*target_path***lib**	library file
*target_path***option****examples**	example and tutorial files
*target_path***option****src**	source code files
*target_path***src**	source code files

3. If your root of Gauss is **D:\GAUSS60**, the installation is finished, otherwise you have to modify the paths of the library using notepad or the LibTool. Another way to update the library is to run Gauss, l**og on to the *option**src* directory**, delete the path with the command **lib option -n** and add the path to the library with the command **lib option -a**.

1.2 Getting started

Gauss 6.0.57+ for Windows is required to use the **OPTION** routines.

1.2.1 readme.txt file

The file *readme.txt* contains last minute information on the **OPTION** procedures. Please read it before using them.

1.2.2 Setup

In order to use these procedures, the **OPTION** library must be active. This is done by including **OPTION** in the LIBRARY statement at the top of your program:

```
library option;
```

To reset global variables in subsequent executions of the program and in order to load DLLs, the following instruction should be used:

optionSet;

1.3 What is OPTION ?

OPTION is a Gauss library for option pricing and hedging. It includes several models and concerns different option types.

OPTION contains the procedures whose list is given below.

- Black and Scholes model
  - **AmericanBS**
  - **AmericanBS_Delta**
  - **AmericanBS_Gamma**
  - **AmericanBS_ImpVol**
  - **AmericanBS_Omega**
  - **AmericanBS_Theta**
  - **AmericanBS_Vega**
  - **BinaryBS**
  - **BinaryBS_Delta**
  - **BinaryBS_Gamma**
  - **BinaryBS_Vega**
  - **CorridorBS**
  - **EuropeanBS**
  - **EuropeanBS_Delta**
  - **EuropeanBS_Gamma**
  - **EuropeanBS_impVol**
  - **EuropeanBS_Omega**
  - **EuropeanBS_Theta**
  - **EuropeanBS_Vega**
  - **EuropeanPayOff**
- Black and Scholes smiled model
  - **BinarySmiled**
  - **BinarySmiled_Delta**
  - **BinarySmiled_Gamma**
  - **BinarySmiled_Vega**
  - **CorridorSmiled**
- Merton/Bates model
  - **AmericanMerton**
  - **AmericanMerton_Delta**

  - **EuropeanMerton**
  - **EuropeanMerton_Delta**

- Rubinstein and Reiner model of Barrier options
  - **EuropeanBS_DIC**
  - **EuropeanBS_DIP**
  - **EuropeanBS_DOC**
  - **EuropeanBS_DOP**
  - **EuropeanBS_UIC**
  - **EuropeanBS_UIP**
  - **EuropeanBS_UOC**
  - **EuropeanBS_UOP**
  - **EuropeanBS_KIC**
  - **EuropeanBS_KIP**
  - **EuropeanBS_KOC**
  - **EuropeanBS_KOP**

- Cox, Ross and Rubinstein model
  - **AmericanCRR**
  - **AmericanCRR_Delta**
  - **AmericanCRR_Gamma**
  - **AmericanCRR_impVol**
  - **AmericanCRR_Omega**
  - **AmericanCRR_Theta**
  - **AmericanCRR_Vega**
  - **AsianFixedStrikeCRR**
  - **AsianFixedStrikeCRR_Delta**
  - **AsianFixedStrikeCRR_Gamma**
  - **AsianFixedStrikeCRR_impVol**
  - **AsianFixedStrikeCRR_Omega**
  - **AsianFixedStrikeCRR_Theta**
  - **AsianFixedStrikeCRR_Vega**
  - **AsianFloatingStrikeCRR**
  - **AsianFloatingStrikeCRR_Delta**
  - **AsianFloatingStrikeCRR_Gamma**
  - **AsianFloatingStrikeCRR_impVol**
  - **AsianFloatingStrikeCRR_Omega**
  - **AsianFloatingStrikeCRR_Theta**

- **AsianFloatingStrikeCRR_Vega**
- **EuropeanCRR**
- **EuropeanCRR_Delta**
- **EuropeanCRR_Gamma**
- **EuropeanCRR_impVol**
- **EuropeanCRR_Omega**
- **EuropeanCRR_Theta**
- **EuropeanCRR_Vega**
- **KnockInDownCRR**
- **KnockInDownCRR_Delta**
- **KnockInDownCRR_Gamma**
- **KnockInDownCRR_impVol**
- **KnockInDownCRR_Omega**
- **KnockInDownCRR_Theta**
- **KnockInDownCRR_Vega**
- **KnockInUpCRR**
- **KnockInUpCRR_Delta**
- **KnockInUpCRR_Gamma**
- **KnockInUpCRR_impVol**
- **KnockInUpCRR_Omega**
- **KnockInUpCRR_Theta**
- **KnockInUpCRR_Vega**
- **KnockOutDownCRR**
- **KnockOutDownCRR_Delta**
- **KnockOutDownCRR_Gamma**
- **KnockOutDownCRR_impVol**
- **KnockOutDownCRR_Omega**
- **KnockOutDownCRR_Theta**
- **KnockOutDownCRR_Vega**
- **KnockOutUpCRR**
- **KnockOutUpCRR_Delta**
- **KnockOutUpCRR_Gamma**
- **KnockOutUpCRR_impVol**
- **KnockOutUpCRR_Omega**
- **KnockOutUpCRR_Theta**
- **KnockOutUpCRR_Vega**
- **LookBackCRR**

  - **LookBackCRR_Delta**
  - **LookBackCRR_Gamma**
  - **LookBackCRR_impVol**
  - **LookBackCRR_Omega**
  - **LookBackCRR_Theta**
  - **LookBackCRR_Vega**
  - **PlotTree**
  - **PrintTree**
  - **Simulate_CRR**

- Heston model
  - **BinaryHeston**
  - **CorridorHeston**
  - **EuropeanHeston**
  - **EuropeanHeston_ImpVol**
  - **EuropeanHeston_Skew**

- Chang, Chang and Lim model
  - **AmericanCCL**
  - **EuropeanCCL**

- Dupire local volatility model
  - **ImpVol_to_LocalVol**
  - **LocalVol_Backward_PDE_Solve**
  - **LocalVol_Init**
  - **LocalVol_Forward_PDE_Solve**
  - **LocalVol_Simulate_SDE**

- Breeden and Litzenberger model to compute density from smile
  - **EuropeanBS_Density**
  - **Smile_to_Density**
  - **SplineSmile_to_Density**

- SABR model
  - **ImpVol_LN_2_Normal**
  - **ImpVol_Normal_2_LN**
  - **SABR_ATM**
  - **SABR_Calibrate**

  - **SABR_Calibrate_ATM**
  - **SABR_Delta**
  - **SABR_ImpVol**
  - **SABR_ImpVol_Black**
  - **SABR_ImpVol_Black_2**
  - **SABR_ImpVol_Normal**
  - **SABR_ImpVol_Normal_2**
  - **SABR_ImpVol_ATM**
  - **SABR_PDF**
  - **SABR_Vega**

- Durrleman model
  - **VD_Calibration**
  - **VD_HestonDensity**
  - **VD_HestonSmile**
  - **VD_Smile**
  - **VD_Smile_to_Density**

- PDM model
  - **PDM_Calibrate**
  - **PDM_ImpVol**
  - **PDM_Skew**
  - **PDM_Skew2ImpVol**

- Yield curve models
  - **BDFS**
  - **CIR**
  - **LongstaffSchwartz**
  - **Vasicek**

- Procedures for simulating diffusion processes
  - **array3d_get_x**
  - **array3d_get_y**
  - **array3d_get_z**
  - **LocalVol_Simulate_SDE**
  - **mGBM_getDate**
  - **mGBM_getSimul**
  - **mGBM_getAsset**
  - **qmc_Simulate_GBM**

  - **qmc_Simulate_GBM_av**
  - **qmc_Simulate_JDF**
  - **qmc_Simulate_JDF_av**
  - **qmc_Simulate_mSDE**
  - **qmc_Simulate_mSDE_av**
  - **qmc_Simulate_OU**
  - **qmc_Simulate_OU_av**
  - **qmc_Simulate_SDE**
  - **qmc_Simulate_SDE_av**
  - **qmc_Simulate_SDE_milstein**
  - **qmc_Simulate_SDE2**
  - **qmc_Simulate_SDE2_av**
  - **rndSet**
  - **Simulate_CRR**
  - **simulate_GBM**
  - **simulate_GBM2**
  - **simulate_mGBM**

- Procedures for solving PDE
  - **PartialBarrier_PDE**
  - **PDE**
  - **PDE_GetMesh**
  - **PDE_ReadFile**
  - **PDE_Solve**

- Procedures for computing numerical integration
  - **gaussHermite**
  - **gaussJacobi**
  - **gaussLaguerre**
  - **gaussLegendre**
  - **intCompute1D**
  - **quadHermite1**
  - **quadHermite2**
  - **quadLaguerre1**
  - **quadLegendre1**
  - **quadLegendre2**
  - **quadLegendre3**
  - **Simpson1**

1.4 Using Online Help

OPTION library supports Windows Online Help. Before using the browser, you have to verify that the **OPTION** library is activated by the `library` command.

Chapter 2

Option Pricing and Hedging

2.1 Notations

We denote by $S(t)$ the price of the asset. $S(0)$ or S_0 is the initial price or the current price. Generally, we use a geometric brownian motion for the dynamics of the asset price:

$$\begin{cases} \mathrm{d}S(t) &= \mu S(t)\,\mathrm{d}t + \sigma S(t)\,\mathrm{d}W(t) \\ S(t_0) &= S_0 \end{cases}$$

σ is called the volatility. We notice r the instantaneous interest rate.

Let K be the strike of an European option. The payout of a call option is $(S(T)-K)_+$ whereas it is $(K-S(T))_+$ for a put option. T is the settlement date of the option. We define $\tau = T - t$ as the residual maturity.

2.2 The Black and Scholes [1973] model

2.2.1 The general framework

Let $C(t, S(t))$ be the price of a call option of maturity τ (or settlemant date T) and strike K. Let $(\Omega, \mathcal{F}, \mathbb{P})$ be the probability space. We assume that the price $S(t)$ of the underlying asset is a diffusion process with the following SDE representation:

$$\begin{cases} \mathrm{d}S(t) &= \mu S(t)\,\mathrm{d}t + \sigma S(t)\ \mathrm{d}W(t) \\ S(t_0) &= S_0 \end{cases}$$

We may show that the fundamental pricing equation take the form:

$$\begin{cases} \frac{1}{2}\sigma^2 S^2 C_{SS} + (\mu - \lambda\sigma) C_S + C_t - rC = 0 \\ C(T, S) = (S-K)_+ \end{cases}$$

The function λ is interpreted as the risk price of the Wiener process $W(t)$. For an asset with a cost-of-carry equal to b, we may show that:

$$\lambda = \frac{\mu - b}{\sigma}$$

To obtain the solution, we may apply the Girsanov theorem with $\phi(t) = -\lambda$. We denote by $\mathbb{Q}$ the new probability distribution. We have:

$$\begin{cases} \mathrm{d}S(t) &= bS(t)\,\mathrm{d}t + \sigma S(t)\,\mathrm{d}W^{\mathbb{Q}}(t) \\ S(t_0) &= S_0 \end{cases}$$

with $W^{\mathbb{Q}}(t)$ a brownian motion under the probability $\mathbb{Q}$. We may then apply the Feynman-Kac theorem to obtain the martingale solution:

$$C(t_0, S_0) = \exp\left[-r(T-t_0)\right] \cdot E^{\mathbb{Q}}\left[\max\left(0, S(T) - K\right) | \mathcal{F}_{t_0}\right]$$

2.2.2 European option

The price of an European option is given by the following formulas:

$$\begin{aligned} C_{\mathrm{EU}}^{\mathrm{BS}} &= S_0 e^{(b-r)\tau}\Phi(d_1) - Ke^{-r\tau}\Phi(d_2) \\ P_{\mathrm{EU}}^{\mathrm{BS}} &= -S_0 e^{(b-r)\tau}\Phi(-d_1) + Ke^{-r\tau}\Phi(-d_2) \end{aligned}$$

with

$$\begin{aligned} d_1 &= \frac{1}{\sigma\sqrt{\tau}}\left(\ln\left(\frac{S_0}{K}\right) + b\tau\right) + \frac{1}{2}\sigma\sqrt{\tau} \\ d_2 &= d_1 - \sigma\sqrt{\tau} \end{aligned}$$

The syntax of the corresponding procedure `EuropeanBS` is

```
C/P = EuropeanBS(S0,K,tau,sigma,b,r);
```

The online help is the following:

```
/*
**> EuropeanBS
**
** Purpose: Premimum of an European option (Black and Scholes [1973]).
**
** Format:  C/P = EuropeanBS(S0,K,sigma,tau,b,r);
**
** Input:       S0 - matrix E*E, underlying prices S0
**               K - matrix E*E, Strike prices K
**           sigma - matrix E*E, volatility sigma
**             tau - matrix E*E, maturity tau
**               b - matrix E*E, cost-of-carry b
**               r - matrix E*E, interest rate r
**
** Output:   C/P - matrix E*E, Option premium
**
** Globals:  _option_type - string
**                          "call" for a call option (default)
**                          "put" for a put option
**
** Remarks:
**
*/
```

2.2.3 American option

The price of the American option is computed using the quadratic approximation of Barone-Adesi and Whaley [1987]. Let M, N and L be three scalars:

$$\begin{aligned} M &= 2r/\sigma^2 \\ N &= 2b/\sigma^2 \\ L &= 1-\exp(-r\tau) \end{aligned}$$

For the call option, we have[1]

$$C_{\text{AM}}^{\text{BS}} = \begin{cases} C_{\text{EU}}^{\text{BS}}(S_0) + A_1 \left(\frac{S_0}{S^\star}\right)^{q_1} & \text{if } S_0 < S^\star \\ S_0 - K & \text{if } S_0 \geqslant S^\star \end{cases}$$

with

$$A_1 = \left[1 - e^{(b-r)\tau}\Phi\left(d_1\left(S^\star\right)\right)\right]\frac{S^\star}{q_1}$$

and

$$q_1 = \frac{1}{2}\left(-(N-1) - \sqrt{(N-1)^2 + 4\frac{M}{L}}\right)$$

We obtain the value $S^\star$ by solving the following non-linear equation:

$$S^\star - K = C_{\text{EU}}^{\text{BS}}(S^\star) + \left[1 - e^{(b-r)\tau}\Phi\left(d_1\left(S^\star\right)\right)\right]\frac{S^\star}{q_1}$$

For the put option, we have

$$P_{\text{AM}}^{\text{BS}} = \begin{cases} P_{\text{EU}}^{\text{BS}}(S_0) + A_2 \left(\frac{S_0}{S^\star}\right)^{q_2} & \text{if } S_0 > S^\star \\ K - S_0 & \text{if } S_0 \leqslant S^\star \end{cases}$$

with

$$A_2 = -\left[1 - e^{(b-r)\tau}\Phi\left(-d_1\left(S^\star\right)\right)\right]\frac{S^\star}{q_2}$$

and

$$q_2 = \frac{1}{2}\left(-(N-1) + \sqrt{(N-1)^2 + 4\frac{M}{L}}\right)$$

We obtain the value $S^\star$ by solving the following non-linear equation:

$$K - S^\star = P_{\text{EU}}^{\text{BS}}(S^\star) - \left[1 - e^{(b-r)\tau}\Phi\left(-d_1\left(S^\star\right)\right)\right]\frac{S^\star}{q_2}$$

Remark 1 *The non-linear equations are solved by a bi-section algorithm.*

The syntax of the corresponding procedure `AmericanBS` is

```
C/P = AmericanBS(S0,K,tau,sigma,b,r);
```

The online help is the following:

[1] If $b \geqslant r$, the price of the American call option is equal to the price of the European call option.

```
/*
**> AmericannBS
**
** Purpose: Premimum of an American option (Black and Scholes [1973]).
**
** Format:  C/P = AmericanBS(S0,K,sigma,tau,b,r);
**
** Input:       S0 - matrix E*E, underlying prices S0
**               K - matrix E*E, Strike prices K
**           sigma - matrix E*E, volatility sigma
**             tau - matrix E*E, maturity tau
**               b - matrix E*E, cost-of-carry b
**               r - matrix E*E, interest rate r
**
** Output:   C/P - matrix E*E, Option premium
**
** Globals:  _option_type - string
**                          "call" for a call option (default)
**                          "put" for a put option
**
** Remarks:
**
*/
```

2.2.4 Barrier and double barrier options

We use the formulas obtained by Rubinstein and Reiner [1991] for the pricing of Barrier options. For the Down & In call **DIC** option, we have

```
C/P = EuropeanBS_DIC(S0,K,sigma,tau,b,r,tau_D,L,R);
```

We use the same syntax for Down & In put **DIP** (`EuropeanBS_DIP`), Down & Out call **DOC** (`EuropeanBS_DOC`), Down & Out put **DOP** (`EuropeanBS_DOP`), Up & In Call **UIC** (`EuropeanBS_UIC`), Up & In put **UIP** (`EuropeanBS_UIP`), Up & Out call **UOC** (`EuropeanBS_UOC`) and Up & Out put **UOP** (`EuropeanBS_UOP`). The parameters `S0`, `K`, `sigma`, `tau`, `b` and `r` are the same parameters used in the `EuropeanBS` procedure. However, you remark three additional parameters : τ_D is the settlement maturity (whereas τ is the residual strike maturity), L is the barrier level and R is the rebate.

For double barriers, we have

```
C/P = EuropeanBS_KIC(S0,K,sigma,tau,b,r,tau_D,L,H);
```

L and H are the lower and upper barrier and there is no rebate. We use the same syntax for Knock-In put **KIP** (`EuropeanBS_KIP`), Knock-Out call **KOC** (`EuropeanBS_KOC`) and Knock-Out Put (`EuropeanBS_KOP`).

2.2.5 Greeks computing

We have introduced some procedures to compute the hedging coefficients. For the Black and Scholes model, we have :

```
Delta  = EuropeanBS_Delta(S0, K, sigma, tau, b, r);
    G  = EuropeanBS_Gamma(S0, K, sigma, tau, b, r);
Omega  = EuropeanBS_Omega(S0, K, sigma, tau, b, r);
Theta  = EuropeanBS_Theta(S0, K, sigma, tau, b, r);
 Vega  = EuropeanBS_Vega(S0, K, sigma, tau, b, r);
```

Remark that we don't have implemented the greeks for all the model. However, it is very easy to compute them numerically using the procedures _Option_gradp and _Option_hessp. Let us consider an example with the LookBackCRR procedure. The first thing to do is to create a generic procedure with only one argument which may be S_0 (for the Delta and Gamma coefficients), τ (for the Theta coefficient) and σ (for the Vega coefficient).

```
proc (1) = _LookBackCRR(XXX);
  local S0,sigma,tau,b,r,N,CallPut;

  S0 = _option_S0;
  sigma = _option_sigma;
  tau = _option_tau;
  b = _option_b;
  r = _option_r;
  N = _option_N;

  if _option_greeks == 1;
    CallPut = LookBackCRR(XXX,sigma,tau,b,r,N);
  elseif _option_greeks == 2;

  elseif _option_greeks == 3;
    CallPut = LookBackCRR(S0,XXX,tau,b,r,N);
  elseif _option_greeks == 4;
    CallPut = LookBackCRR(S0,sigma,XXX,b,r,N);
  elseif _option_greeks == 5;
    CallPut = LookBackCRR(S0,sigma,tau,XXX,r,N);
  elseif _option_greeks == 6;
    CallPut = LookBackCRR(S0,sigma,tau,b,XXX,N);
  endif;

  retp(CallPut);
endp;
```

For the Delta coefficient, we have also:

```
proc (1) = LookBackCRR_Delta(S0,sigma,tau,b,r,N);
  local Delta,XXX;

  _option_S0 = S0; _option_sigma = sigma; _option_tau = tau;
  _option_b = b; _option_r = r; _option_N = N;

  _option_greeks = 1;
  XXX = S0;
  Delta = _Option_gradp(&_LookBackCRR,XXX);

  retp(Delta);
endp;
```

For the Gamma coefficient, we have:

```
proc (1) = LookBackCRR_Gamma(S0,sigma,tau,b,r,N);
  local Gamma_,XXX;

  _option_S0 = S0; _option_sigma = sigma; _option_tau = tau;
  _option_b = b; _option_r = r; _option_N = N;

  _option_greeks = 1;
  XXX = S0;
  Gamma_ = _Option_hessp(&_LookBackCRR,XXX);

  retp(Gamma_);
endp;
```

The only difference is that we use the _Option_hessp procedure instead of _Option_gradp procedure. For the Theta coefficient, we have:

```
proc (1) = LookBackCRR_Theta(S0,sigma,tau,b,r,N);
  local Theta,XXX;

  _option_S0 = S0; _option_sigma = sigma; _option_tau = tau;
  _option_b = b; _option_r = r; _option_N = N;

  _option_greeks = 4;
  XXX = tau;
  Theta = _Option_gradp(&_LookBackCRR,XXX);

  retp(Theta);
endp;
```

For the Vega coefficient, we have:

```
proc (1) = LookBackCRR_Vega(S0,sigma,tau,b,r,N);
  local Vega,XXX;

  _option_S0 = S0; _option_sigma = sigma; _option_tau = tau;
  _option_b = b; _option_r = r; _option_N = N;

  _option_greeks = 3;
  XXX = sigma;
  Vega = _Option_gradp(&_LookBackCRR,XXX);

  retp(Vega);
endp;
```

Finally, the implementation of the Omega coefficient is the following:

```
proc (1) = LookBackCRR_Omega(S0,sigma,tau,b,r,N);
  local Omega;

  Omega = S0 .* LookBackCRR_Delta(S0,sigma,tau,b,r,N) ./
          LookBackCRR(S0,sigma,tau,b,r,N);

  retp(Omega);
endp;
```

2.2.6 Volatility smile

To obtain the implied volatility from option market prices, we use the EuropeanBS_ImpVol procedure.

2.2.7 Binary and Corridor options

Let $S(t)$ be the stock price at time t. Following Rubinstein and Reiner [1991], the pay-off of the *cash-or-nothing* digital option is:

$$G(S) = \mathbf{1}_{[S>K]}$$

for a call option of strike K. The price at time t_0 of the digital option with a date of maturity T is also given by the following formula:

$$\mathbf{BC}(t_0, S(t_0)) = \mathbb{E}\left[1_{[S(T)>K]} \exp \int_{t_0}^{T} r(t)\ \mathrm{d}t \mid \mathcal{F}_{t_0}\right]$$

with $\mathcal{F}_t$ the associated filtration of the risk neutral probability measure $\mathbb{Q}$ and $r(t)$ the riskless interest rate. In the case of the BS model, we obtain:

$$\mathbf{BC}(t_0, S(t_0)) = e^{-r\tau}\Phi(d_2)$$

with

$$\begin{aligned} d_2 &= d_1 - \sigma\sqrt{\tau} \\ d_1 &= \frac{\ln\frac{S_0}{K} + b\tau}{\sigma\sqrt{\tau}} + \frac{1}{2}\sigma\sqrt{\tau} \end{aligned}$$

The online help of the `BinaryBS` is the following:

```
/*
**> BinaryBS
**
** Purpose: Pricing of Digital options (Black and Scholes [1973]).
**
** Format:  C/P = BinaryBS(S0,K,sigma,tau,b,r);
**
** Input:       S0 - matrix E*E, underlying prices S0
**               K - matrix E*E, Strike prices K
**           sigma - matrix E*E, volatility sigma
**             tau - matrix E*E, maturity tau
**               b - matrix E*E, cost-of-carry b
**               r - matrix E*E, interest rate r
**
** Output:   C/P - matrix E*E, Option premium
**
** Globals:  _option_type - string
**                          "call" for a call option (default)
**                          "put" for a put option
**
** Remarks:
**
*/
```

In the case of volatility smile, we have the following formula:

$$\mathbf{BC}_{\text{sm}}(t_0, S(t_0)) = \mathbf{BC}(t_0, S(t_0)) - p\upsilon(t_0, S(t_0))$$

with $\upsilon(t_0, S(t_0))$ the Vega of the vanilla option $\mathbf{C}(t_0, S(t_0))$ and p the skew parameter:

$$p = \frac{\partial\,\Sigma(T, K)}{\partial\,K}$$

The online help of the `BinaryBS` is the following:

```
/*
**> BinarySmiled
**
** Purpose: Pricing of Digital options (Smiled model with Vega correction).
**
** Format:  C/P = BinarySmiled(S0,K,sigma,tau,b,r,p);
**
** Input:       S0 - matrix E*E, underlying prices S0
**               K - matrix E*E, Strike prices K
**           sigma - matrix E*E, volatility sigma
**             tau - matrix E*E, maturity tau
```

```
**              b - matrix E*E, cost-of-carry b
**              r - matrix E*E, interest rate r
**              p - matrix E*E, parameter $p$ of the skew = d(ImpVol)/d(K)
**
** Output:   C/P - matrix E*E, Option premium
**
** Globals:  _option_type - string
**                          "call" for a call option (default)
**                          "put" for a put option
**
** Remarks:
**
*/
```

We define the corridor option as *a serie of double binaries options*. The pay-off is

$$G\left(S\left(T\right)\right)=\sum_{i=1}^{N}\mathbf{1}_{[S(t_i)\in[L,H]]}$$

where $(t_i)_{1\leq i\leq N}$ are the fixing dates. We may prices the corridor using the formula

$$\mathbf{CC}\left(t_0,S\left(t_0\right)\right)=e^{-r\tau}\sum_{i=1}^{N}\left[\boldsymbol{\varrho}\left(S_0,L,\sigma,\tau_i,b,r\right)-\boldsymbol{\varrho}\left(S_0,H,\sigma,\tau_i,b,r\right)\right]$$

with $\boldsymbol{\varrho}\left(S_0,K,\sigma,\tau,b,r\right)=e^{r\tau}\mathbf{BC}\left(t_0,S\left(t_0\right)\right)$ the actualized binary option price of maturity τ and with a strike equal to K. In the case of volatility smile, we have

$$\mathbf{CC}_{\mathrm{sm}}\left(t_0,S\left(t_0\right)\right)=e^{-r\tau}\sum_{i=1}^{N}\left[\boldsymbol{\xi}\left(S_0,L,\sigma_L,\tau_i,b,r,p_L\right)-\boldsymbol{\xi}\left(S_0,H,\sigma_H,\tau_i,b,r,p_H\right)\right]$$

with $\boldsymbol{\xi}\left(S_0,K,\sigma,\tau,b,r,p\right)$ the actualized smiled binary option price. We have:

```
C/P = CorridorBS(S0,L,H,sigma,tau,b,r,t);
```

and

```
C/P = CorridorSmiled(S0,L,H,sigmaL,sigmaH,tau,b,r,t,pL,pH);
```

2.3 The Cox, Ross and Rubinstein [1981] model

The CRR model is a discrete approximation of the BS model. The SDE is replaced by a Markov chain with two states:

$$S_{t+1}=\left\{\begin{array}{ll}u\times S_t & \text{with probability}\quad p\\ d\times S_t & \text{with probability}\quad 1-p\end{array}\right.$$

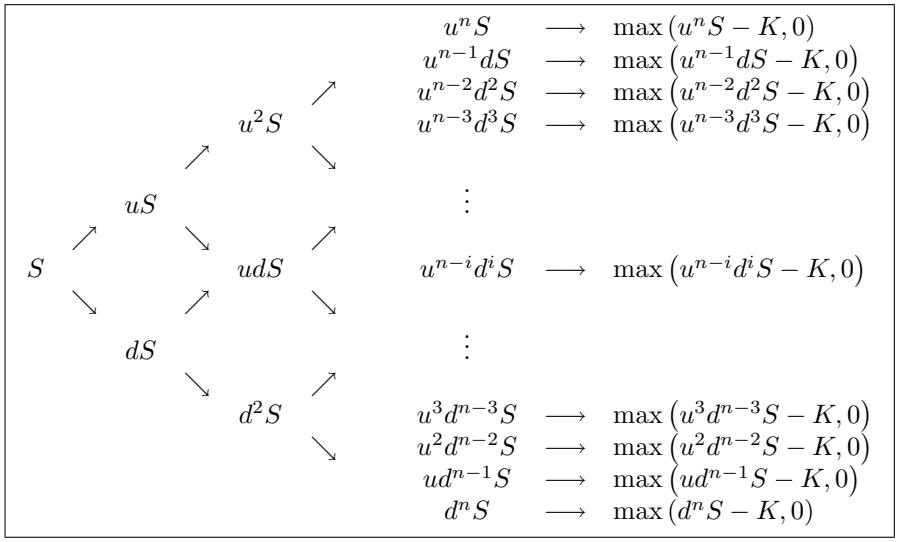
We have:

$$p=\frac{\exp\left(r\tau/n\right)-d}{u-d}$$

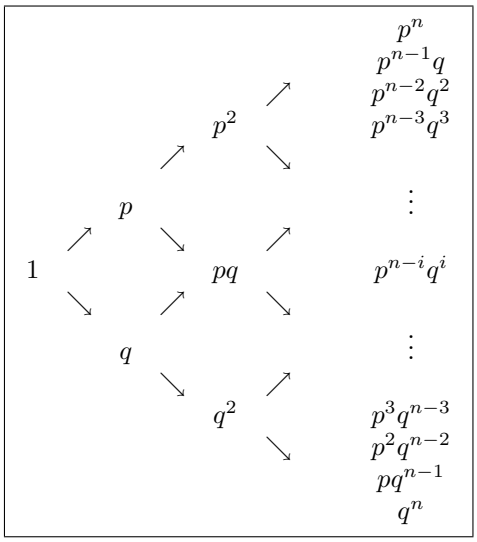
with

$$\begin{array}{rcl}u&=&\exp\left(\sigma\sqrt{\frac{\tau}{n}}\right)\\ d&=&\exp\left(-\sigma\sqrt{\frac{\tau}{n}}\right)\end{array}$$

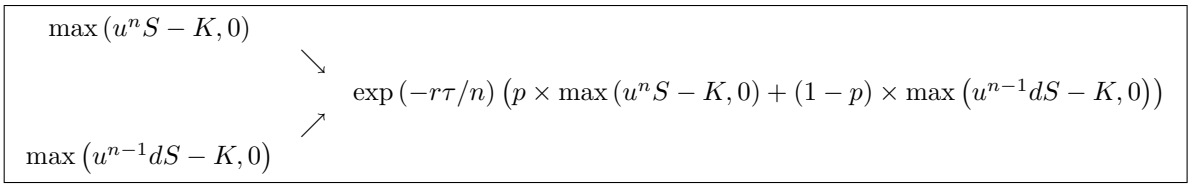
n is the number of discretized points (or the number of arbitrage possibilities). We may not build the tree of the underlying and the final values of the payoff. For the European option, we have:



The corresponding tree for the probaibilities is:



We may now build the tree of the option prices at each node. For example, we have:



The syntax of the corresponding procedure `EuropeanCRR` is

```
{C/P,Tree} = EuropeanCRR(S0,K,sigma,tau,b,r,n);
```

For the American option, we have

```
{C/P,Tree} = AmericanCRR(S0,K,sigma,tau,b,r,n);
```

In order to represent the trees, we may use the `PrintTree` or `PlotTree` procedures.

Remark 2 *The OPTION library contains other procedures to compute prices with the CRR model for various payoff functions (Asian, Barrier and Lookback). See the list in the previous chapter.*

2.4 Pricing models with volatity smile

2.4.1 The Merton [1976] jump diffusion model

The model of Merton is based on the following risk-neutral process:

$$\left\{ \begin{array}{rcl} \mathrm{d}S(t) & = & (b-\lambda k)\, S(t)\, \mathrm{d}t + \sigma S(t)\, \mathrm{d}W(t) + \tilde{k}\, \mathrm{d}N(t) \\ S(t_0) & = & S_0 \end{array} \right.$$

with $W(t)$ a Wiener process and $N(t)$ a Poisson process with intensity λ. $1+\tilde{k}$ is a lognormal random variable:

$$\ln\left(1+\tilde{k}\right) \sim N\left(\ln(1+k) - \frac{1}{2}\delta^2, \delta^2\right)$$

The price of the European option is given by:

$$\begin{aligned} C_{\mathrm{EU}}^{\mathrm{Merton}} &= \sum_{n=0}^{\infty} e^{-\lambda\tau} \frac{(\lambda\tau)^n}{n!} C_{\mathrm{EU}}^{\mathrm{BS}}(S_0, K, \sigma_n, \tau, b_n, r) \\ P_{\mathrm{EU}}^{\mathrm{Merton}} &= \sum_{n=0}^{\infty} e^{-\lambda\tau} \frac{(\lambda\tau)^n}{n!} P_{\mathrm{EU}}^{\mathrm{BS}}(S_0, K, \sigma_n, \tau, b_n, r) \end{aligned}$$

with

$$\begin{aligned} \sigma_n &= \sqrt{\sigma^2 + n\frac{\delta^2}{\tau}} \\ b_n &= b - \lambda k + n\frac{\ln(1+k)}{\tau} \end{aligned}$$

The syntax of the `EuropeanMerton` procedure is

```
C/P = EuropeanMerton(S0,K,sigma,tau,b,r,k,delta,lambda);
```

The online help is the following:

```
/*
**> EuropeanMerton
**
** Purpose: Premimum of an European option (Merton [1976]).
**
** Format:  C/P = EuropeanMerton(S0,K,sigma,tau,b,r,kbarStar,deltaStar,lambdaStar);
**
```

```
** Input:       S0 - matrix E*E, underlying prices S0
**               K - matrix E*E, Strike prices K
**           sigma - matrix E*E, volatility sigma
**             tau - matrix E*E, maturity tau
**               b - matrix E*E, cost-of-carry b
**               r - matrix E*E, interest rate r
**        kbarStar - matrix E*E, parameter $\bar{k}^{\star}$
**       deltaStar - matrix E*E, parameter $\delta^{\star}$
**      lambdaStar - matrix E*E, parameter $\lambda^{\star}$
**
** Output:   C/P - matrix E*E, Option premium
**
** Globals:  _option_type - string
**                          "call" for a call option (default)
**                          "put" for a put option
**
** Remarks:
**
*/
```

For the American option, we use the Bates [1991] method based on the quadratic approximation of Barone-Adesi and Whaley [1987]. The syntax of the `AmericanMerton` procedure is

```
C/P = AmericanMerton(S0,K,sigma,tau,b,r,k,delta,lambda);
```

2.4.2 The Heston [1993] stochastic volatility model

Heston assumes that the spot price of the underlying asset follows the SV diffusion:

$$\left\{ \begin{array}{rcl} \mathrm{d}S\left(t\right) & = & \mu S\left(t\right)\mathrm{d}t+\sqrt{V\left(t\right)}S\left(t\right)\mathrm{d}W_{1}\left(t\right) \\ \mathrm{d}V\left(t\right) & = & \kappa\left[\theta-V\left(t\right)\right]\mathrm{d}t+\sigma_{V}\sqrt{V\left(t\right)}\mathrm{d}W_{2}\left(t\right) \end{array} \right.$$

with

$$\left\{ \begin{array}{rcl} S\left(t_{0}\right) & = & S_{0} \\ V\left(t_{0}\right) & = & V_{0} \end{array} \right.$$

$W\left(t\right)=\left[\begin{array}{cc} W_{1}\left(t\right) & W_{2}\left(t\right)\end{array}\right]^{\top}$ is a two-dimensionnal Wiener process with

$$E\left[W\left(t\right)W\left(t\right)^{\top}\right]=\left[\begin{array}{cc} 1 & \rho \\ \rho & 1 \end{array}\right]\cdot t$$

Heston sets the price $\lambda_{2}\left(t,S\left(t\right),V\left(t\right)\right)$ of the risk associated to the second Wiener $W_{2}\left(t\right)$ (or the volatility $V\left(t\right)$) as follows:

$$\lambda_{2}\left(t,S\left(t\right),V\left(t\right)\right)=\frac{\lambda}{\sigma_{V}}\sqrt{V\left(t\right)}$$

In this case, we could show that the dynamics of the joint process $\left\{S\left(t\right),V\left(t\right)\right\}$ under the risk-neutral probability $\mathbb{Q}$ is:

$$\left\{ \begin{array}{rcl} \mathrm{d}S\left(t\right) & = & bS\left(t\right)\mathrm{d}t+\sqrt{V\left(t\right)}S\left(t\right)\mathrm{d}W_{1}^{\mathbb{Q}}\left(t\right) \\ \mathrm{d}V\left(t\right) & = & \left(\kappa\left[\theta-V\left(t\right)\right]-\lambda V\left(t\right)\right)\mathrm{d}t+\sigma_{V}\sqrt{V\left(t\right)}\mathrm{d}W_{2}^{\mathbb{Q}}\left(t\right) \end{array} \right.$$

with $W'\left(t\right)=\left[\begin{array}{cc} W_{1}'\left(t\right) & W_{2}'\left(t\right)\end{array}\right]^{\top}$ a two-dimensionnal Wiener process under $\mathbb{Q}$. Heston gives a closed-form solution of this problem. We have:

$$\begin{array}{rcl} C_{\mathrm{EU}}^{\mathrm{Heston}} & = & S_{0}e^{(b-r)\tau}P_{1}-Ke^{-r\tau}P_{2} \\ P_{\mathrm{EU}}^{\mathrm{Heston}} & = & S_{0}e^{(b-r)\tau}\left(P_{1}-1\right)-Ke^{-r\tau}\left(P_{2}-1\right) \end{array}$$

For $j = 1, 2$, we have:

$$P_j = \frac{1}{2} + \frac{1}{\pi} \int_0^\infty \mathrm{Re}\left[\frac{e^{-i\phi \ln K} f_j(x, V_0, \tau, \phi)}{i\phi}\right] d\phi$$

with

$$\begin{aligned}
f_j(x, V_0, \tau, \phi) &= \exp\left(C_j(\tau, \phi) + D_j(\tau, \phi) V_0 + i\phi x\right) \\
C_j(\tau, \phi) &= bi\phi\tau + \frac{a_j}{\sigma_V^2}\left[(b_j - i\rho\sigma_V\phi + d_j)\tau - 2\ln\left(\frac{1 - g_j e^{d_j\tau}}{1 - g_j}\right)\right] \\
D_j(\tau, \phi) &= \frac{1}{\sigma_V^2}(b_j - i\rho\sigma_V\phi + d_j)\left(\frac{1 - e^{d_j\tau}}{1 - g_j e^{d_j\tau}}\right) \\
g_j &= \frac{b_j - i\rho\sigma_V\phi + d_j}{b_j - i\rho\sigma_V\phi - d_j} \\
d_j &= \sqrt{(i\rho\sigma_V\phi - b_j)^2 - \sigma_V\left(2iu_j\phi - \phi^2\right)}
\end{aligned}$$

The parameters a_j, b_j et u_j are respectively:

$$\begin{aligned}
a_1 &= \kappa\theta & a_2 &= \kappa\theta \\
b_1 &= \kappa + \lambda - \rho\sigma_V & b_2 &= \kappa + \lambda \\
u_1 &= \frac{1}{2} & u_2 &= -\frac{1}{2}
\end{aligned}$$

The previous formulas are those obtained by Heston himself in its original paper. There are others formulas obtained by others authors. To compute the option prices, we use the `EuropeanHeston` procedure:

```
{C,P}  =  EuropeanHeston(S0,K,tau,b,r,sigma0,
                         kappa,theta,sigmaV,rho,lambda);
```

The online help is the following:

```
/*
**> EuropeanHeston
**
** Purpose: Premimum of an European option (Heston [1993]).
**
** Format:   {C,P} = EuropeanHeston(S0,K,tau,r,sigma0,kappa,theta,sigmaV,rho,lambda);
**
** Input:       S0 - vector N*1, underlying prices S0
**               K - vector N*1, Strike prices K
**             tau - vector N*1, maturity tau
**               r - vector N*1, interest rate r
**          sigma0 - vector N*1, volatility V0
**           kappa - vector N*1, $\kappa$ parameter
**           theta - vector N*1, $\theta$ parameter
**          sigmaV - vector N*1, $\sigma_V$ parameter
**             rho - vector N*1, $\rho$ parameter
**          lambda - vector N*1, $\lambda$ parameter
**
** Output:       C - vector N*1, Premium of the Call option
**               P - vector N*1, Premium of the Put option
**
** Globals: _quad_mtd - scalar, m\U{e9}thod to compute the knots and the weights (default = 1)
```

```
**                      1 for the Newton-Raphson algorithm
**                      2 for the SVD algorithm
**           _int_mtd - scalar, integration algorithm (default = 1)
**                      5 for simpson algorithm
**                      4 for trapezoidal algorithm
**                      3 for Hermite quadrature
**                      2 for Laguerre quadrature
**                      1 for Legendre quadrature
**     _Heston_Bounds - vector 2*1, bounds of integration (default = 1000|0)
**      _Heston_Model - scalar
**                      1 for the Heston original formulas
**                      2 for an alternative formula (I didn't remember the refernce)
**                      3 for anonther alternative formula (I didn't remember the refernce)
**
** Remarks: You may use Legendre or Laguerre quadratures and Simpson algorithm
**          (but with a large number of discretization points)
**
*/
```

Two other procedures are very useful to compute implied volatility and skew values: `EuropeanHeston_ImpVol` and `EuropeanHeston_Skew`.

The formula of the *cash-or-nothing* digital option is given by the following expression

$$\mathbf{BC}\left(t_0, S_0, V_0\right) = e^{-r\tau} P_2$$

The corresponding procedures are `BinaryHeston` and `CorridorHeston` with the following syntax:

$$\texttt{C/P} = \texttt{BinaryHeston}(\texttt{S0}, \texttt{K}, ...);$$

and

$$\texttt{C/P} = \texttt{CorridorHeston}(\texttt{S0}, \texttt{L}, \texttt{H}, ..., \texttt{t});$$

2.4.3 The Chang, Chang and Lim [1998] subordinated process model

Chang, Chang and Lim assumes that the subordinated process is a jump process given by the following equations:

$$\mathrm{d}S\left(t\right) = \mu\left(t\right)\,\mathrm{d}t + \sigma\left(t\right)\,\mathrm{d}W\left(t\right)$$

with

$$\begin{aligned} \mu\left(t\right)\,\mathrm{d}t &= \mu\,\mathrm{d}N\left(t\right) \\ \sigma^2\left(t\right)\,\mathrm{d}t &= \sigma^2\,\mathrm{d}N\left(t\right) \end{aligned}$$

$N\left(t\right)$ is a Poisson process with intensity λ.

The price of an European Option corresponds to:

$$\begin{aligned} C_{\mathrm{EU}}^{\mathrm{CCL}} &= \sum_{n=0}^{\infty} e^{-\lambda\tau}\frac{\left(\lambda\tau\right)^n}{n!} C_{\mathrm{EU}}^{\mathrm{BS}}\left(S_0, K, \sigma, n, b, r\right) \\ P_{\mathrm{EU}}^{\mathrm{CCL}} &= \sum_{n=0}^{\infty} e^{-\lambda\tau}\frac{\left(\lambda\tau\right)^n}{n!} P_{\mathrm{EU}}^{\mathrm{BS}}\left(S_0, K, \sigma, n, b, r\right) \end{aligned}$$

The syntax of the procedure `EuropeanCCL` is

```
C/P = EuropeanCCL(S0,K,tau,sigma,b,r,lambda);
```

Remark 3 *The parameters σ and n are the information-time volatility and maturity index.*

Let H be the scalar given by:

$$H = 2r/\left[\sigma^2\left(1-e^{(b-r)\tau}\right)\right]$$

.For an American call option, we have:

$$C_{\mathrm{AM}}^{\mathrm{CCL}} = \sum_{n=0}^{\infty} e^{-\lambda\tau}\frac{(\lambda\tau)^n}{n!} C_n\left(S_0, K, \sigma, n, b, r\right)$$

with

$$C_n\left(S0, K, \sigma, n, b, r\right) = \begin{cases} C_{\mathrm{EU}}^{\mathrm{BS}}\left(S_0, K, \sigma, n, b, r\right) + A_1\left(\frac{S_0}{S^\star}\right)^{q_1} & \text{if } S_0 < S^\star \\ S_0 - K & \text{if } S_0 \geqslant S^\star \end{cases}$$

We have:

$$A_1 = \left[1 - e^{(b-r)\tau}\Phi\left(d_1\left(S^\star\right)\right)\right]\frac{S^\star}{q_1}$$

and

$$q_1 = \frac{1}{2}\left(1+\sqrt{1+4H}\right)$$

We obtain the value $S^\star$ by solving the following non-linear equation:

$$S^\star - K = C_{\mathrm{EU}}^{\mathrm{BS}}\left(S_0, K, \sigma, n, b, r\right) + \left[1 - e^{(b-r)\tau}\Phi\left(d_1\left(S^\star\right)\right)\right]\frac{S^\star}{q_1}$$

For the put option, we have:

$$P_{\mathrm{AM}}^{\mathrm{CCL}} = \sum_{n=0}^{\infty} e^{-\lambda\tau}\frac{(\lambda\tau)^n}{n!} P_n\left(S_0, K, \sigma, n, b, r\right)$$

In this case, we have:

$$P_n\left(S_0, K, \sigma, n, b, r\right) = \begin{cases} P_{\mathrm{EU}}^{\mathrm{BS}}\left(S_0, K, \sigma, n, b, r\right) + A_2\left(\frac{S_0}{S^\star}\right)^{q_2} & \text{if } S_0 > S^\star \\ K - S_0 & \text{if } S_0 \leqslant S^\star \end{cases}$$

with

$$A_2 = -\left[1 - e^{(b-r)\tau}\Phi\left(-d_1\left(S^\star\right)\right)\right]\frac{S^\star}{q_2}$$

and

$$q_2 = \frac{1}{2}\left(1-\sqrt{1+4H}\right)$$

We obtain the value $S^\star$ by solving the following non-linear equation:

$$K - S^\star = P_{\mathrm{EU}}^{\mathrm{BS}}\left(S_0, K, \sigma, n, b, r\right) - \left[1 - e^{(b-r)\tau}\Phi\left(-d_1\left(S^\star\right)\right)\right]\frac{S^\star}{q_2}$$

The syntax of the procedure `AmericanCCL` is

```
C/P = AmericanCCL(S0,K,tau,sigma,b,r,lambda);
```

2.5 Calibrated pricing models

2.5.1 The Breeden and Litzenberger [1978] model

Following Breeden and Litzberger (1978), we know that the risk neutral density is related to the price of vanillas. For example, we have at time t:

$$\Pr\{S_T \leq K\} = 1 + e^{r(T-t)}\partial_K C_t(T,K)$$

where $C_t(T,K)$ denotes the price of European call option with maturity T and strike K.

In practice, we work with implied volatilities and Black-Scholes model. Let $\Sigma_t(T,K)$ be the BS implied volatility and $C^\star$ the BS price with spot volatility Σ. We have:

$$\partial_K C_t^\star(T,K,\Sigma) = -e^{-r(T-t)}\Phi(d_2)$$

where:

$$d_2 = \frac{1}{\Sigma\sqrt{T-t}}\ln\left(\frac{S_t e^{r(T-t)}}{K}\right) - \frac{1}{2\Sigma\sqrt{T-t}}$$

In the case of constant volatility, computing $\Pr\{S_T \leq K\}$ is straightforward. In othercases, we have:

$$\begin{aligned}\Pr\{S_T \leq K\} &= 1 + e^{r(T-t)}\partial_K C_t(T,K)\\ &= 1 + e^{r(T-t)}\partial_K C_t^\star(T,K,\Sigma_t(T,K))\\ &= 1 + e^{r(T-t)}\left(\partial_K C_t^\star(T,K,\Sigma_t) + \partial_\Sigma C_t^\star(T,K,\Sigma_t)\,\partial_\Sigma \Sigma_t(T,K)\right)\end{aligned}$$

with

$$\partial_\Sigma C_t^\star(T,K,\Sigma) = S_0\sqrt{T-t}\phi\left(d_2 + \frac{1}{2}\Sigma\sqrt{T-t}\right)$$

If we are interested in the density function, we have:

$$\partial_K \Pr\{S_T \leq K\} = e^{r(T-t)}\partial^2_{KK} C_t(T,K)$$

For the BS case, it comes that:

$$\partial^2_{KK} C_t^\star(T,K,\Sigma) = e^{-r(T-t)}\frac{\phi(d_2)}{K\Sigma\sqrt{T-t}}$$

In othercases, the formula is generalized as follows:

$$\begin{aligned}e^{-r(T-t)}\partial^2_{KK}\Pr\{S_T \leq K\} &= \partial^2_{KK} C_t^\star(T,K,\Sigma_t) + 2\partial^2_{K\Sigma} C_t^\star(T,K,\Sigma_t)\,\partial_\Sigma\Sigma_t(T,K) +\\ &\quad \partial_\Sigma C_t^\star(T,K,\Sigma_t)\,\partial^2_{K\Sigma}\Sigma_t(T,K) + \partial^2_{\Sigma\Sigma} C_t^\star(T,K,\Sigma_t)\left(\partial_\Sigma\Sigma_t(T,K)\right)^2\end{aligned}$$

with

$$\partial^2_{K\Sigma} C_t^\star(T,K,\Sigma) = \frac{S_0 d_1 \phi(d_1)}{\Sigma K}$$

and

$$\partial^2_{\Sigma\Sigma} C_t^\star(T,K,\Sigma) = \frac{S_0 d_1 d_2\sqrt{T-t}\phi(d_1)}{\Sigma}$$

The difficulty is then to specify the $\Sigma_t(T,K)$ function which have to be continuous for every K.

The Gauss procedure `Smile_to_Density` computes the cumulative and the probability density functions. Its syntax is

```
{Sigma,cdf,pdf} = Smile_to_Density(S0,K,smile,tau,r);
```

The argument `smile` is a procedure which computes Σ_t. Two special cases are considered : `EuropeanBS_Density` for the Black-Scholes model

```
{cdf,pdf} = EuropeanBS_Density(S0,K,sigma,tau,r);
```

and `SplineSmile_to_Density` when Σ_t is a spline function with respect to K.

```
{Sigma,cdf,pdf} = SplineSmile_to_Density(S0,K,tau,r,ImpVol_Strike,ImpVol_Volatility);
```

2.5.2 The Dupire [1994] local volatility model

We consider the local volatility model:

$$\mathrm{d}S(t) = \mu S(t)\,\mathrm{d}t + \sigma(t,S)\,S(t)\,\mathrm{d}W(t)$$

Dupire [1994] shows that we may calibrate $\sigma(t,S)$ from the implied volatility surface $\Sigma(T,K)$. For an European call option with maturity T and strike K, the price is given by the following formula:

$$C(T,K) = e^{-rT}\mathbb{E}_{\mathbb{Q}}\left[(S(T)-K)^{+}\middle|\mathcal{F}_0\right]$$

with $\mathbb{Q}$ the risk-neutral probability measure. Breeden and Litzenberger [1978] proved that the knowledge of $C(T,K)$ is equivalent to the knowledge of the probability distribution $\mathbb{Q}$. In particular, we have:

$$\mathbb{Q}(x) = \Pr\{S(T)\le x\} = 1 + e^{rT}\partial_K C(T,x)$$

and

$$\mathrm{d}\mathbb{Q}(x) = e^{rT}\partial_K^2 C(T,x)$$

The idea of Dupire [1994] is to consider the valorisation equation of $C(T,K)$ and to deduce the local volatility function:

$$\sigma(T,K) = \sqrt{2\frac{bK\partial_K C(T,K) + \partial_T C(T,K) - (b-r)\,C(T,K)}{K^2\partial_K^2 C(T,K)}}$$

with b the cost-of-carry parameter.

We notice $\Sigma(T,K)$ the implied volatility surface, that is the market prices $\hat{C}(T,K)$ verify the formula $\hat{C}(T,K) = S_0 e^{(b-r)T}\Phi(d_1) - Ke^{-rT}\Phi(d_2)$ with $d_1 = \Sigma^{-1}(T,K)\,T^{-1/2}\left(\ln\left(K^{-1}S_0\right) + bT\right) + \frac{1}{2}\Sigma(T,K)\,T^{1/2}$ and $d_2 = d_1 - \Sigma(T,K)\sqrt{T}$. Let $\mathrm{BS}(S_0,K,\Sigma,T,b,r)$ be the Black-Scholes formula. We have:

$$\begin{aligned}
\partial_K \mathrm{BS}(S_0,K,\Sigma(T,K),T,b,r) &= \partial_K\partial\,\mathrm{BS}(S_0,K,\Sigma,T,b,r) + \partial_K\Sigma(T,K)\,\partial_\Sigma\,\mathrm{BS}(S_0,K,\Sigma,T,b,r)\\
\partial_T \mathrm{BS}(S_0,K,\Sigma(T,K),T,b,r) &= \partial_T\,\mathrm{BS}(S_0,K,\Sigma,T,b,r) + \partial_T\Sigma(T,K)\,\partial_\Sigma\,\mathrm{BS}(S_0,K,\Sigma,T,b,r)
\end{aligned}$$

and

$$\begin{aligned}
\partial_K^2 \mathrm{BS}(S_0,K,\Sigma(T,K),T,b,r) &= \partial_K^2\,\mathrm{BS}(S_0,K,\Sigma,T,b,r)\\
&\quad + 2\partial_K\Sigma(T,K)\,\partial^2_{\Sigma,K}\,\mathrm{BS}(S_0,K,\Sigma,T,b,r)\\
&\quad + \partial_K^2\Sigma(T,K)\,\partial_\Sigma\,\mathrm{BS}(S_0,K,\Sigma,T,b,r)\\
&\quad + \left(\partial_K\Sigma(T,K)\right)^2\partial_\Sigma^2\,\mathrm{BS}(S_0,K,\Sigma,T,b,r)
\end{aligned}$$

The derivatives are[2]:

$$\begin{aligned}
\partial_K \text{BS} &= -e^{-rT}\Phi(d_2) \\
\partial_\Sigma \text{BS} &= S_0 e^{(b-r)T}\sqrt{T}\phi(d_1) \\
\partial_K^2 \text{BS} &= e^{-rT}\frac{\phi(d_2)}{K\Sigma\sqrt{T}} \\
\partial_\Sigma^2 \text{BS} &= e^{-rT}\frac{K\sqrt{T}\phi(d_2)\,d_1 d_2}{\Sigma} \\
\partial_{\Sigma,K}^2 \text{BS} &= e^{-rT}\frac{d_1\phi(d_2)}{\Sigma} \\
\partial_T \text{BS} &= (b-r)\,S_0 e^{(b-r)T}\Phi(d_1) + Ke^{-rT}\left(r\Phi(d_2) + \frac{\Sigma\phi(d_2)}{2\sqrt{T}}\right)
\end{aligned}$$

Using the factor $e^{-rT}\frac{K\phi(d_2)}{2\Sigma\sqrt{T}}$, it comes that the local volatility surface is equal to:

$$\sigma^2(T,K) = \frac{1 + 2bKT\Sigma^{-1}(T,K)\,\partial_K\Sigma(T,K) + 2T\Sigma^{-1}(T,K)\,\partial_T\Sigma(T,K)}{1 + 2K\sqrt{T}d_1\partial_K\Sigma(T,K) + K^2T\Sigma(T,K)\,\partial_K^2\Sigma(T,K) + K^2Td_1d_2\left(\partial_K\Sigma(T,K)\right)^2}\Sigma^2(T,K)$$

The syntax of the corresponding procedure `ImpVol_to_LocalVol` is

```
LocalVol = ImpVol_to_LocalVol(SigmaProc,dK_SigmaProc,
                              d2K_SigmaProc,dT_SigmaProc,T,K,S0,b);
```

The online help is the following:

```
/*
**> ImpVol_to_LocalVol
**
** Purpose: Computes the local volatility surface from the implied volatility surface.
**
** Format:     LocalVol = ImpVol_to_LocalVol(SigmaProc,dK_SigmaProc,d2K_SigmaProc,dT_SigmaProc,T,K,S0,b);
**
** Input:     SigmaProc - pointer to a procedure that computes the function $\Sigma \left(T,K\right)$
**         dK_SigmaProc - pointer to a procedure that computes the function d(ImpVol)/dK
**                        or scalar 0 to compute d(ImpVol)/dK by numerical derivation
**        d2K_SigmaProc - pointer to a procedure that computes the function d^2(ImpVol)/dK^2
**                        or scalar 0 to compute d(ImpVol)/dK by numerical derivation
**         dT_SigmaProc - pointer to a procedure that computes the function d(ImpVol)/dT
**                        or scalar 0 to compute d(ImpVol)/dK by numerical derivation
**                   T - matrix E*E, maturity tau
**                   K - matrix E*E, Strikes K
**                  S0 - scalar, underlying price S0
**                   b - scalar, cost-of-carry b
**
** Output:     LocalVol - matrix E*E, values of local volatility
**
** Remarks:
**
*/
```

[2]We use the fact that

$$\frac{S_0}{Ke^{-bT}}\phi(d_1) = \phi(d_2)$$

In practice, we know the implied volatilities for only some strikes and maturities. We have also to interpolate them in order to generate the function $\Sigma(T,K)$ for all strikes and maturities. They are several methods to perform this interpolation. You may use your own method. For your convenience, we have implemented one popular method which consists in spline interpolation in strike K and geometric interpolation in time T. The syntax is the following

```
LocalVol = ImpVol_to_LocalVol_CSGI(Smile_K, Smile_ImpVol, Smile_T,
                                   T, K, S0, b, r, FileName);
```

When we have computed the local volatility surface, we may price options using PDE or Monte-Carlo methods. The first thing to do is to initialize the local volatility model with the procedure `LocalVol_Init`. Then, we may price the option using Backward PDE

```
{S0,tau,C/P} = LocalVol_Backward_PDE_Solve(OptionType,K,L,H,tau,Theta);
```

or Forward PDE

```
{S0,tau,C/P} = LocalVol_Forward_PDE_Solve(OptionType,K,L,H,tau,Theta);
```

Note that `OptionType` may take the following values : c (call), p (put), doc (Down & Out call), dop (Down & Out put), uoc (Up & Out call), uop (Up & Out put), koc (Knock & Out call), kop (Knock & Out put), bc (binary call).and bp (binary put).

Remark 4 *If you want to price options with Monte Carlo methods, you may use the `LocalVol_simulate_SDE` procedure.*

2.5.3 The SABR [2002] model

Hagan *et al.* [2002] suggest to use the SABR model (Stochastic $\alpha-\beta-\rho$) to take into account the smile effect. Let F be the forward price. We have:

$$\left\{\begin{array}{rcl} \mathrm{d}F(t) &=& \sigma(t)F(t)^{\beta}\,\mathrm{d}W_1(t) \\ \mathrm{d}\sigma(t) &=& \nu\sigma(t)\,\mathrm{d}W_2(t) \end{array}\right.$$

with $\mathbb{E}[W_1(t)W_2(t)]=\rho t$. The model has also 4 parameters: β, ν, ρ and α the actual value of the instantaneous volatility. One of the big interest of the SABR model is that we may have an approximate formula of the implied Black volatility:

$$\Sigma_{\mathrm{BLACK}}(T,K)\simeq\Sigma_{\mathrm{SABR}}(F_0,T,K;\alpha,\beta,\nu,\rho)$$

with Σ_{SABR} the function defined as follows:

$$\begin{array}{rcl}\Sigma(T,K) &=& \dfrac{\alpha}{(F_0K)^{(1-\beta)/2}\left(1+\frac{(1-\beta)^2}{24}\ln^2 F_0/K+\frac{(1-\beta)^4}{1920}\ln^4 F_0/K+\ldots\right)}\left(\dfrac{z}{x(z)}\right)\times \\ && \left(1+\left(\dfrac{(1-\beta)^2\alpha^2}{24(F_0K)^{1-\beta}}+\dfrac{\rho\alpha\nu\beta}{4(F_0K)^{(1-\beta)/2}}+\dfrac{2-3\rho^2}{24}\nu^2\right)T+\ldots\right)\end{array}$$

with $z=\nu\alpha^{-1}(F_0K)^{(1-\beta)/2}\ln F_0/K$ and $x(z)=\ln\left(\sqrt{1-2\rho z+z^2}+z-\rho\right)-\ln(1-\rho)$. For ATM option, we have:

$$\Sigma(T,F_0)=\frac{\alpha}{F_0^{1-\beta}}\left(1+\left(\frac{(1-\beta)^2\alpha^2}{24F_0^{2-2\beta}}+\frac{\rho\alpha\nu\beta}{4F_0^{1-\beta}}+\frac{2-3\rho^2}{24}\nu^2\right)T+\ldots\right)$$

To compute the Black implied volatility, we use the `SABR_ImpVol_Black` procedure

```
ImpVol = SABR_ImpVol_Black(F0,K,alpha,beta,nu,rho,tau);
```

In the case of the Normal (or Gaussian) model, we have

```
ImpVol = SABR_ImpVol_Normal(F0,K,alpha,beta,nu,rho,tau);
```

In order to calibrate the parameters, we consider the `SABR_Calibrate` procedure

```
{alpha,beta,nu,rho} = SABR_Calibrate(F0,K,ImpVol,beta,tau,sv);
```

If we prefer the parametrization $(\Sigma_{\text{ATM}}, \beta, \nu, \rho)$ in place of $(\alpha, \beta, \nu, \rho)$, we use the `SABR_ATM` and `SABR_Calibrate_ATM` procedures:

```
ImpVol_ATM = SABR_ATM(F0,alpha,beta,nu,rho,tau);
```

```
alpha = SABR_Calibrate_ATM(F0,ImpVol_ATM,beta,nu,rho,tau);
```

The first procedure converts $(\alpha, \beta, \nu, \rho)$ into $(\Sigma_{\text{ATM}}, \beta, \nu, \rho)$ whereas the second procedure converts $(\Sigma_{\text{ATM}}, \beta, \nu, \rho)$ into $(\alpha, \beta, \nu, \rho)$.

The Gauss procedure `sabr_pdf` computes the cumulative and the probability density functions. Its syntax is

```
{Sigma,cdf,pdf} = sabr_pdf(F0,K,alpha,beta,nu,rho,tau);
```

Finally, we provide two other procedures `sabr_delta` and `sabr_vega` to compute the greeks: Their syntax are

```
{DeltaBS,Delta1,Delta2} = sabr_delta(F0,K,alpha,beta,nu,rho,tau);
```

and

```
{VegaBS,VegaATM} = sabr_vega(F0,K,alpha,beta,nu,rho,tau);
```

The definition of the greeks corresponds to the following formulas:

$$\Delta_{\text{BS}} = \frac{\partial\, BS}{\partial\, F_0}$$

$$\Delta_1 = \frac{\partial\, BS}{\partial\, F_0} + \frac{\partial\, BS}{\partial\, \Sigma}\left(\frac{\partial\, \Sigma_{\text{SABR}}}{\partial\, F_0}\right)$$

$$\Delta_2 = \frac{\partial\, BS}{\partial\, F_0} + \frac{\partial\, BS}{\partial\, \Sigma}\left(\frac{\partial\, \Sigma_{\text{SABR}}}{\partial\, F_0} + \frac{\partial\, \Sigma_{\text{SABR}}}{\partial\, \alpha}\frac{\partial\, \alpha_{\text{SABR}}}{\partial\, F_0}\right)$$

$$\mathcal{V}_{\text{BS}} = \frac{\partial BS}{\partial\, \Sigma}$$

$$\mathcal{V}_{\text{ATM}} = \frac{\partial\, C}{\partial\, \Sigma_{\text{ATM}}} = \frac{\partial\, BS}{\partial\, \Sigma}\frac{\partial\, \Sigma_{\text{SABR}}}{\partial\, \Sigma_{\text{ATM}}} = \frac{\partial\, BS}{\partial\, \Sigma}\frac{\partial\, \Sigma_{\text{SABR}}}{\partial\, \alpha}\frac{\partial\, \alpha_{\text{SABR}}}{\partial\, \Sigma_{\text{ATM}}}$$

Remark that we use a finite difference method to compute the greeks in the Gauss procedures (the shifts are given by the external variables `_sabr_shift_Delta` and `_sabr_shift_Vega`).

2.5.4 The Durrleman [2004] model

Durrleman [2004] shows a general result in order to obtain an analytical approximation of implied volatility. Under some assumptions, we obtain the following formula:

$$\Sigma(T,K) \simeq \sqrt{\sigma_0^2 + a\ln\frac{S_0}{K} + b\frac{T}{2} + \frac{c}{2}\ln^2\frac{S_0}{K} + d\frac{T}{2}\ln\frac{S_0}{K} + \frac{e}{6}\ln^3\frac{S_0}{K}}$$

Let us assume that the dynamics of S_t is markovian with

$$S_t = S_0 \exp\left(\int_0^t \sigma_s \,\mathrm{d}W_s - \frac{1}{2}\int_0^t \sigma_s^2 \,\mathrm{d}s\right)$$

and

$$\begin{aligned}
\mathrm{d}\sigma_t^2 &= \mu_t \,\mathrm{d}t - 2\sigma_t\left(a_t \,\mathrm{d}W_t + \tilde{a}_t \,\mathrm{d}\tilde{W}_t\right)\\
\mathrm{d}\mu_t &= \ell_t \,\mathrm{d}t + \omega_t \,\mathrm{d}W_t + \tilde{\omega}_t \,\mathrm{d}\tilde{W}_t\\
\mathrm{d}a_t &= m_t \,\mathrm{d}t + u_t \,\mathrm{d}W_t + \tilde{u}_t \,\mathrm{d}\tilde{W}_t\\
\mathrm{d}\tilde{a}_t &= n_t \,\mathrm{d}t + v_t \,\mathrm{d}W_t + \tilde{v}_t \,\mathrm{d}\tilde{W}_t\\
\mathrm{d}u_t &= o_t \,\mathrm{d}t + x_t \,\mathrm{d}W_t + \tilde{x}_t \,\mathrm{d}\tilde{W}_t\\
\mathrm{d}x_t &= (.) \,\mathrm{d}t + y_t \,\mathrm{d}W_t + (.) \,\mathrm{d}\tilde{W}_t\\
\mathrm{d}v_t &= (.) \,\mathrm{d}t + z_t \,\mathrm{d}W_t + (.) \,\mathrm{d}\tilde{W}_t\\
\mathrm{d}\tilde{u}_t &= (.) \,\mathrm{d}t + s_t \,\mathrm{d}W_t + (.) \,\mathrm{d}\tilde{W}_t\\
\mathrm{d}m_t &= (.) \,\mathrm{d}t + q_t \,\mathrm{d}W_t + (.) \,\mathrm{d}\tilde{W}_t\\
\mathrm{d}\omega_t &= (.) \,\mathrm{d}t + r_t \,\mathrm{d}W_t + (.) \,\mathrm{d}\tilde{W}_t
\end{aligned}$$

Let us define:

$$\begin{aligned}
b_t &= \mu_t - \frac{a_t^2}{2} - \frac{2\tilde{a}_t^2}{3} - \sigma_t^2 a_t + \frac{2\sigma_t u_t}{3}\\
c_t &= \frac{-2u_t}{3\sigma_t} - \frac{a_t^2}{2\sigma_t^2} + \frac{2\tilde{a}_t^2}{3\sigma_t^2}\\
d_t &= \frac{2m_t}{3} - \frac{\omega_t}{3\sigma_t} - \frac{x_t}{2} - \frac{\mu_t a_t}{3\sigma_t^2} + \frac{\tilde{a}_t\tilde{u}_t}{6\sigma_t} + \frac{\tilde{a}_t v_t}{\sigma_t} + \frac{2a_t\tilde{a}_t^2}{3\sigma_t^2} + \frac{2\sigma_t u_t}{3} - \frac{a_t^2}{3}\\
e_t &= \frac{x_t}{2\sigma_t^2} + \frac{2a_t u_t}{\sigma_t^3} - \frac{3\tilde{a}_t\tilde{u}_t}{2\sigma_t^3} - \frac{\tilde{a}_t v_t}{\sigma_t^3} + \frac{3a_t^3}{2\sigma_t^4} - \frac{4a_t\tilde{a}_t^2}{\sigma_t^4}
\end{aligned}$$

$$\begin{aligned}
f_t = {} & -\frac{o_t}{3\sigma_t} - \frac{x_t}{2} - \frac{2a_t m_t}{3\sigma_t^2} - \frac{a_t^4}{8\sigma_t^4} + \frac{u_t^2}{3\sigma_t^2} - \frac{\tilde{u}_t^2}{5\sigma_t^2} +\\
& \frac{22\tilde{a}_t^4}{15\sigma_t^4} - \frac{7v_t^2}{15\sigma_t^2} + \frac{\tilde{v}_t^2}{3\sigma_t^2} - \frac{a_t\tilde{a}_t\tilde{u}_t}{\sigma_t^3} - \frac{10a_t\tilde{a}_t v_t}{3\sigma_t^3} +\\
& \frac{2\tilde{a}_t n_t}{3\sigma_t^2} + \frac{\mu_t u_t}{2\sigma_t^3} + \frac{a_t^2 u_t}{2\sigma_t^3} - \frac{11\tilde{a}_t^2 u_t}{6\sigma_t^3} + \frac{2a_t x_t}{3\sigma_t^2} -\\
& \frac{3\tilde{a}_t\tilde{x}_t}{5\sigma_t^2} + \frac{7\mu_t a_t^2}{6\sigma_t^4} - \frac{3a_t^2\tilde{a}_t^2}{\sigma_t^4} - \frac{\mu_t\tilde{a}_t^2}{\sigma_t^4} + \frac{28\tilde{a}_t^2\tilde{v}_t}{15\sigma_t^3} +\\
& \frac{\tilde{a}_t v_t}{3\sigma_t} + \frac{\tilde{a}_t\tilde{u}_t}{2\sigma_t} - \frac{a_t^3}{6\sigma_t^2} - \frac{q_t}{3\sigma_t} + \frac{r_t}{6\sigma_t^2} + \frac{2y_t}{5\sigma_t} +
\end{aligned}$$

$$\frac{2a_t w_t}{3\sigma_t^3} - \frac{8\tilde{a}_t s_t}{15\sigma_t^2} - \frac{8\tilde{u}_t v_t}{15\sigma_t^2} - \frac{4\tilde{a}_t z_t}{5\sigma_t^2} + \frac{a_t^3\tilde{a}_t^2}{6\sigma_t^4} - \frac{\tilde{a}_t\tilde{w}_t}{2\sigma_t^3} - \frac{a_t u_t}{3\sigma_t}$$

$$\begin{aligned} g_t &= \frac{15a_t\tilde{a}_t\tilde{u}_t}{\sigma_t^5} + \frac{10a_t\tilde{a}_t v_t}{\sigma_t^5} - \frac{15a_t^4}{2\sigma_t^6} - \frac{2y_t}{5\sigma_t^3} - \frac{2u_t^2}{\sigma_t^4} + \\ &\quad \frac{4v_t^2}{5\sigma_t^4} - \frac{24\tilde{a}_t^4}{5\sigma_t^6} + \frac{6\tilde{u}_t^2}{5\sigma_t^4} - \frac{16\tilde{a}_t^2\tilde{v}_t}{5\sigma_t^5} + \frac{6\tilde{u}_t v_t}{5\sigma_t^4} + \\ &\quad \frac{32a_t^2\tilde{a}_t^2}{\sigma_t^6} + \frac{4\tilde{a}_t z_t}{5\sigma_t^4} + \frac{8\tilde{a}_t^2 u_t}{\sigma_t^5} + \frac{6\tilde{a}_t s_t}{5\sigma_t^4} + \frac{8\tilde{a}_t\tilde{x}_t}{5\sigma_t^4} - \\ &\quad \frac{12a_t^2 u_t}{\sigma_t^5} - \frac{3a_t x_t}{\sigma_t^4} \end{aligned}$$

$$\begin{aligned} h_t &= -\frac{r_t}{4} + \ell_t - \frac{7\tilde{a}_t^2 u_t}{12\sigma_t} + \frac{\sigma_t o_t}{2} + \frac{v_t^2}{10} - \frac{u_t^2}{3} - \frac{\tilde{v}_t^2}{2} - \tilde{a}_t n_t - \frac{3\tilde{u}_t^2}{5} - \\ &\quad \frac{a_t x_t}{4} + \frac{\tilde{a}_t a_t v_t}{2\sigma_t} + \frac{3\sigma_t^2 x_t}{4} - \sigma_t^2 m_t - \frac{\sigma_t^2\tilde{a}_t^2}{2} + \frac{7\sigma_t^2 a_t^2}{8} - \frac{\sigma_t^3 u_t}{2} + \\ &\quad \frac{\sigma_t w_t}{2} - \frac{3a_t^4}{16\sigma_t^2} + \frac{\tilde{a}_t^4}{15\sigma_t^2} - \mu_t a_t + \frac{3\tilde{a}_t\tilde{w}_t}{4\sigma_t} + \frac{a_t^3}{2} - \frac{3\sigma_t y_t}{10} + \\ &\quad \frac{\sigma_t q_t}{2} - \frac{3a_t^2 u_t}{4\sigma_t} + a_t\tilde{a}_t^2 + \frac{\mu_t\tilde{a}_t^2}{2\sigma_t} - \frac{2\tilde{a}_t^2\tilde{v}_t}{5\sigma_t} - \frac{a_t^2\tilde{a}_t^2}{2\sigma_t^2} - \frac{5a_t\tilde{a}_t\tilde{u}_t}{4\sigma_t} + \\ &\quad \frac{\mu_t u_t}{4\sigma_t} + \frac{5\sigma_t\tilde{a}_t\tilde{u}_t}{4} - \frac{\sigma_t a_t u_t}{2} - \frac{\sigma_t\tilde{a}_t v_t}{2} - \frac{\tilde{u}_t v_t}{10} - \frac{\tilde{a}_t s_t}{10} - \\ &\quad \frac{3\tilde{a}_t z_t}{5} - \frac{3\tilde{a}_t\tilde{x}_t}{10} - a_t m_t \end{aligned}$$

One can approximate the implied volatility at date t for a call of maturity T by the computation:

$$\Sigma_t(T,K) = \sqrt{\sigma_t^2 + a_t k_t + \frac{1}{2}b_t\tau + \frac{1}{2}c_t k_t^2 + \frac{1}{2}d_t k_t\tau + \frac{1}{6}e_t k_t^3 + \frac{1}{4}f_t k_t^2\tau + \frac{1}{24}g_t k_t^4 + \frac{1}{6}h_t\tau^2}$$

where $\tau = T - t$ and $k_t = \ln\frac{S_t}{K}$.

Remark 5 *We use the results of* Dhouibi *[2005] for the higher orders $k_t^2\tau$, k_t^4 and τ^2.*

The syntax of the `VD_Smile` procedure is

```
Sigma = VD_Smile(St,K,sigma_t,at,bt,ct,dt,et,tau);
```

To obtain the distribution $\mathbb{Q}$, we use the `VD_Smile_to_Density` procedure:

```
{Sigma,cdf,pdf} = VD_Smile_to_Density(St,K,sigma_t,at,bt,ct,dt,et,tau,r);
```

In the case of the Heston model, we have

$$\begin{aligned} \mathrm{d}S_t &= \sigma_t S_t\,\mathrm{d}W_t \\ \mathrm{d}\sigma_t^2 &= \kappa\left(\mu - \sigma_t^2\right)\,\mathrm{d}t + \varepsilon\sigma_t\left(\rho\mathrm{d}W_t + \sqrt{1-\rho^2}\mathrm{d}\tilde{W}_t\right) \end{aligned}$$

with $\tilde{\rho} = \sqrt{1-\rho^2}$ (or $\tilde{\rho} = -\sqrt{1-\rho^2}$),. We obtain the following parameters:

$$\begin{aligned} a_t &= -\frac{\varepsilon\rho}{2} \\ b_t &= \kappa\left(\mu - \sigma_t^2\right) + \frac{\sigma_t^2\varepsilon\rho}{2} - \frac{\varepsilon^2}{6}\left(1 - \frac{\rho^2}{4}\right) \\ c_t &= \frac{\varepsilon^2}{6\sigma_t^2}\left(1 - \frac{7\rho^2}{4}\right) \end{aligned}$$

and

$$\begin{aligned} d_t &= \frac{\kappa\varepsilon\rho}{6}\left(\frac{\mu}{\sigma_t^2} + 1\right) - \frac{\varepsilon^2\rho}{12}\left(\rho + \frac{\varepsilon\tilde{\rho}^2}{\sigma_t^2}\right) \\ e_t &= \frac{\varepsilon^3\rho}{2\sigma_t^4}\left(1 - \frac{11\rho^2}{8}\right) \\ f_t &= -\frac{a_t^4}{8\sigma_t^4} + \frac{22\tilde{a}_t^4}{15\sigma_t^4} + \frac{7\mu_t a_t^2}{6\sigma_t^4} - \frac{3a_t^2\tilde{a}_t^2}{\sigma_t^4} - \frac{\mu_t\tilde{a}_t^2}{\sigma_t^4} - \frac{a_t^3}{6\sigma_t^2} + \\ &\quad \frac{r_t}{6\sigma_t^2}\frac{2a_t w_t}{3\sigma_t^3} + \frac{a_t^3\tilde{a}_t^2}{6\sigma_t^4} - \frac{\tilde{a}_t\tilde{w}_t}{2\sigma_t^3} \\ g_t &= -\frac{15a_t^4}{2\sigma_t^6} - \frac{24\tilde{a}_t^4}{5\sigma_t^6} + \frac{32a_t^2\tilde{a}_t^2}{\sigma_t^6} = \frac{\varepsilon^4}{10\sigma_t^6}\left(26\rho^2 - 3 - \frac{443}{16}\rho^4\right) \\ h_t &= -\frac{r_t}{4} + \ell_t - \frac{\sigma_t^2\tilde{a}_t^2}{2} + \frac{7\sigma_t^2 a_t^2}{8} - \frac{3a_t^4}{16\sigma_t^2} + \frac{\tilde{a}_t^4}{15\sigma_t^2} - \mu_t a_t + \\ &\quad \frac{a_t^3}{2} + a_t\tilde{a}_t^2 + \frac{\mu_t\tilde{a}_t^2}{2\sigma_t} - \frac{a_t^2\tilde{a}_t^2}{2\sigma_t^2} \end{aligned}$$

For the local volatility mode $\sigma_t^2 = f(t, S_t)$, we have

$$\Sigma(T,K) = \sqrt{f(0,S_0) - \frac{1}{2}S_0\partial_S f(0,S_0)\ln\frac{S_0}{K} + \ldots}$$

For the SABR model, we retreive the formula of Hagan *et al.* [2002]. Remark that the library contains three procedures `VD_Heston_Parameters`, `VD_HestonSmile` and `VD_HestonDensity` to manage the Durrleman approximation of the Heston model.

A procedure to calibrate general models is available and is called `VD_Calibration`. Let $\left\{\hat{\Sigma}_t(T_i,K_i), i=1,\ldots,n\right\}$ be a set of market implied volatilities. `VD_Calibration` solves the following problem:

$$\theta = \arg\min\sum_{i=1}^{n} w_i\left(\hat{\Sigma}_t(T_i,K_i) - \Sigma_t(T_i,K_i)\right)^2$$

where

$$\Sigma_t(T,K) = \sqrt{\sigma_t^2 + a_t\ln\frac{S_t}{K} + b_t\frac{T-t}{2} + \frac{c_t}{2}\ln^2\frac{S_t}{K} + d_t\frac{T-t}{2}\ln\frac{S_t}{K} + \frac{e_t}{6}\ln^3\frac{S_t}{K}}$$

and

$$\begin{pmatrix} \sigma_t \\ a_t \\ b_t \\ d_t \\ e_t \end{pmatrix} = g(\theta)$$

Its syntax is:

```
VD_Calibraton(S0,K,MarketSmile,tau,ModelParams,sv,Constr);
```

`ModelParams` is a pointer to a procedure that computes $g(\theta)$. `Constr` is a vector which indicates the parameters which are activated or not. To impose the positivity of $\Sigma_t(T,K)$, you have to set `_VD_positivity = 1`.

Remark 6 *It may be interested in some cases to use the parametrization* $\left(\Sigma_t^{ATM}(T,S_t), a_t, b_t, d_t, e_t\right)$ *instead of the parametrization* $(\sigma_t, a_t, b_t, d_t, e_t)$*. It could be done by using the command* `_VD_ATM = 1`*.*

2.5.5 The PDM model

The works of Durrleman suggest to use a function of the following form to compute implied volatility

$$\Sigma(T,K) = \sqrt{\alpha^2 + \beta \ln \frac{(S_0+m)}{K} + \gamma \ln^2 \frac{(S_0+m)}{K} + \delta \ln^3 \frac{(S_0+m)}{K}}$$

The online help of the `PDM_ImpVol` is the following:

```
/*
**> PDM_ImpVol
**
** Purpose: Computes the Implied Volatility of an European option (PDM).
**
** Format:  ImpVol = PDM_ImpVol(S0,K,alpha,beta,gamma_,delta,m);
**
** Input:       S0 - matrix E*E, underlying prices S0
**               K - matrix E*E, Strike prices K
**           alpha - matrix E*E, parameter $\alpha$
**            beta - matrix E*E, parameter $\beta$
**          gamma_ - matrix E*E, parameter $\gamma$
**           delta - matrix E*E, parameter $\delta$
**               m - matrix E*E, parameter $m$
**
** Output:  ImpVol - matrix E*E, Implied volatiliy values
**
** Remarks:
**
*/
```

The model is useful to manage the skew, because we have:

$$\mathrm{sk}(T,K) = \frac{1}{2\Sigma(T,K)}\left(-\beta + 2\gamma \ln \frac{(S_0+m)}{K} + 3\delta \ln^2 \frac{(S_0+m)}{K}\right)$$

The online help of the `PDM_Skew` is the following:

```
/*
**> PDM_Skew
**
** Purpose: Computes the Skew of an European option (PDM).
**
** Format:  Skew = PDM_Skew(S0,K,alpha,beta,gamma_,delta,m);
**
** Input:       S0 - matrix E*E, underlying prices S0
**               K - matrix E*E, Strike prices K
```

```
**           alpha - matrix E*E, parameter $\alpha$
**            beta - matrix E*E, parameter $\beta$
**          gamma_ - matrix E*E, parameter $\gamma$
**           delta - matrix E*E, parameter $\delta$
**               m - matrix E*E, parameter $m$
**
** Output:    Skew - matrix E*E, Skew
**
** Remarks:
**
*/
```

To convert skew to omplied volatilities, we consider the `PDM_Skew2ImpVol` procedure:

```
ImpVol = PDM_Skew2ImpVol(S0,K,Skew,beta,gamma_,delta,m);
```

We may now describre the calibration procedure. We distinguish two cases:

1. If we have n ($n \geq 5$) implied volatilities, we consider the following least squares problem:

$$\mathrm{RSS}(\alpha,\beta,\gamma,\delta,m) = \sum_{i=1}^{n} w_i \left(\Sigma(T,K_i) - \hat{\Sigma}(T,K_i)\right)^2$$

   where $\hat{\Sigma}(T,K_i)$ is the market volatility for the strike K_i. w_i is the weight corresponding to the i^{th} strike.

2. If we have only two implied volatilities and some informations on the skew, the optimization program becomes:

$$\begin{aligned} & \min w_- \left(\mathrm{sk}(T,K_-) - \widehat{\mathrm{sk}}(T,K_-)\right)^2 + w_+ \left(\mathrm{sk}(T,K_+) - \widehat{\mathrm{sk}}(T,K_+)\right)^2 \\ \text{u.c.} \quad & \Sigma(T,K_i) = \hat{\Sigma}(T,K_i) \text{ for } i = 1,2 \end{aligned}$$

   For some numerical reasons, we prefer to consider the following program:

$$\min \sum_{i=1}^{2} \left(\Sigma(T,K_i) - \hat{\Sigma}(T,K_i)\right)^2 + w_- \left(\Sigma(T,K_-) - \Sigma^\star(T,K_-)\right)^2 + w_+ \left(\Sigma(T,K_+) - \Sigma^\star(T,K_+)\right)^2$$

   with

$$\Sigma^\star(T,K) = -\frac{1}{2\widehat{\mathrm{sk}}(T,K)} \left(\frac{\beta}{K} + 2\frac{\gamma}{K}\ln\frac{(S_0+m)}{K} + 3\frac{\delta}{K}\ln^2\frac{(S_0+m)}{K}\right)$$

The syntax of the `PDM_calibrate` procedure is

```
{a,b,c,d,m} = PDM_Calibrate(S0,Strike_Vol,ImpVol,Strike_Skew,Skew,sv);
```

2.6 Numerical methods

2.6.1 Solving PDE with numerical methods

We consider the linear parabolic equation

$$\frac{\partial u(t,x)}{\partial t} + c(t,x)\,u(t,x) = \mathcal{A}_t u(t,x) + d(t,x) \tag{2.1}$$

where $\mathcal{A}_t$ is the elliptic differential equation

$$\mathcal{A}_t u(t,x) = a(t,x)\frac{\partial^2 u(t,x)}{\partial x^2} + b(t,x)\frac{\partial u(t,x)}{\partial x} \tag{2.2}$$

The main idea is to solve the equation (2.1) for $t \in [t^-, t^+]$ and $x \in [x^-, x^+]$. In this case, we use the method of finite difference, well-adapted for 2-order parabolic equations in x.

2.6.1.1 The finite difference method

We introduce a uniform finite-difference mesh for t and x. Let N_t and N_x be the number of discretisation points for t and x respectively. We denote by k and h the mesh spacings. We have

$$\begin{aligned} k &= \frac{t^+ - t^-}{N_t - 1} \\ h &= \frac{x^+ - x^-}{N_x - 1} \end{aligned}$$

and

$$\begin{aligned} t_m &= t^- + m \cdot k \\ x_i &= x^- + i \cdot h \end{aligned}$$

Let u_i^m be the approximate solution to (2.1) at the grid point (t_m, x_i) and $u(t_m, x_i)$ the exact solution of the partial differential equation at this point.

Discretisation scheme for the space If we consider the central difference method to approximate the derivatives, we have

$$\frac{\partial u(t,x)}{\partial x} \simeq \frac{u_{i+1}^m - u_{i-1}^m}{2h}$$

and

$$\frac{\partial^2 u(t,x)}{\partial x^2} \simeq \frac{u_{i+1}^m - 2u_i^m + u_{i-1}^m}{h^2}$$

The equation (2.1) becomes

$$\frac{\partial u(t,x)}{\partial t} + c_i^m u_i^m = \mathbf{A}_i^m + d_i^m \tag{2.3}$$

with

$$\mathbf{A}_i^m = a_i^m \frac{u_{i+1}^m - 2u_i^m + u_{i-1}^m}{h^2} + b_i^m \frac{u_{i+1}^m - u_{i-1}^m}{2h}$$

We obtain finally

$$\frac{\partial u(t,x)}{\partial t} = \mathbf{B}_i^m \tag{2.4}$$

with

$$\mathbf{B}_i^m = \mathbf{A}_i^m + d_i^m - c_i^m u_i^m$$

Discretisation scheme for the time The most classical method to solve the equation (2.1) is to use the Euler scheme. We have

$$\frac{\partial u(t,x)}{\partial t} \simeq \frac{u_i^m - u_i^{m-1}}{k}$$

We remark also that the equation (2.1) becomes

$$\frac{u_i^m - u_i^{m-1}}{k} + c_i^m u_i^m = \mathcal{A}_t u(t,x) + d_i^m$$

However, the function $\mathcal{A}_t u(t,x)$ depends both on the time t and the space x. That's why we could not employ the traditional Euler algorithm

$$u_i^m = u_i^{m-1} + k\left[\mathcal{A}_t u(t,x) + d_i^m - c_i^m u_i^m\right]$$

In this case, we then replace the function $\mathcal{A}_t u(t,x)$ by its numerical approximation $\mathbf{A}_i^m$. So, we have

$$\begin{aligned} u_i^m &= u_i^{m-1} + k\left[\mathbf{A}_i^m + d_i^m - c_i^m u_i^m\right] \\ &= u_i^{m-1} + k\mathbf{B}_i^m \end{aligned}$$

The θ-scheme method In the previous paragraph, we have used the single-sided forward difference to approximate the derivatives $\frac{\partial u(t,x)}{\partial t}$. There exists another algorithm, like the Richardson extrapolation. The θ-scheme method is a combination of left-sided and right-sided differences. Let $\theta \in [0,1]$. We have

$$u_i^m = u_i^{m-1} + k\left[(1-\theta)\mathbf{B}_i^{m-1} + \theta\mathbf{B}_i^m\right]$$

Using the expression of $\mathbf{B}_i^m$, we obtain

$$\begin{aligned} +u_{i-1}^{m-1}\left[a_i^{m-1}(1-\theta)\frac{k}{h^2} - b_i^{m-1}(1-\theta)\frac{k}{2h}\right] \\ +u_i^{m-1}\left[1 - 2a_i^{m-1}(1-\theta)\frac{k}{h^2} - c_i^{m-1}(1-\theta)k\right] \\ u_{i+1}^{m-1}\left[a_i^{m-1}(1-\theta)\frac{k}{h^2} + b_i^{m-1}(1-\theta)\frac{k}{2h}\right] \\ +u_{i-1}^m\left[a_i^m\theta\frac{k}{h^2} - b_i^m\theta\frac{k}{2h}\right] \\ +u_i^m\left[-1 - 2a_i^m\theta\frac{k}{h^2} - c_i^m\theta k\right] \\ +u_{i+1}^m\left[a_i^m\theta\frac{k}{h^2} + b_i^m\theta\frac{k}{2h}\right] \quad = \quad -\left[d_i^{m-1}(1-\theta)k + d_i^m\theta k\right] \end{aligned}$$

2.6.1.2 The different numerical algorithms

We note now

$$\begin{aligned} \alpha_i^m &= a_i^m\frac{k}{h^2} - b_i^m\frac{k}{2h} \\ \beta_i^m &= 1 - 2a_i^m\frac{k}{h^2} - c_i^m k \\ \gamma_i^m &= a_i^m\frac{k}{h^2} + b_i^m\frac{k}{2h} \end{aligned}$$

The explicit scheme This scheme corresponds to $\theta = 0$. We have then

$$u_i^m = \alpha_i^{m-1} u_{i-1}^{m-1} + \beta_i^{m-1} u_i^{m-1} + \gamma_i^{m-1} u_{i+1}^{m-1} + d_i^{m-1} k \tag{2.5}$$

We obtain the numerical solution by iterations from the initial condition and by using Dirichlet conditions.

The implicit scheme This scheme corresponds to $\theta = 1$. We have then

$$\alpha_i^m u_{i-1}^m + \left(\beta_i^m - 2\right) u_i^m + \gamma_i^m u_{i+1}^m = -\left(u_i^{m-1} + d_i^m k\right) \tag{2.6}$$

We obtain the numerical solution by solving the linear system (2.6) and using Neumann conditions.

The mixed schemes We have then $\theta \in]0,1[$. For example, the well-famous Crank-Nicholson scheme corresponds to $\theta = \frac{1}{2}$. We introduce the following notations

$$\begin{aligned}
\varsigma_i^m &= (1-\theta)\,\alpha_i^m \\
\tau_i^m &= 1 + (1-\theta)\,(\beta_i^m - 1) \\
\upsilon_i^m &= (1-\theta)\,\gamma_i^m \\
\phi_i^m &= \theta \alpha_i^m \\
\varphi_i^m &= -1 + \theta\,(\beta_i^m - 1) \\
\chi_i^m &= \theta \gamma_i^m \\
\psi_i^m &= (1-\theta)\, d_i^{m-1} k + \theta d_i^m k
\end{aligned}$$

To obtain the numerical solution, we have to solve the linear system

$$\phi_i^m u_{i-1}^m + \varphi_i^m u_i^m + \chi_i^m u_{i+1}^m = -\left[\varsigma_i^{m-1} u_{i-1}^{m-1} + \tau_i^{m-1} u_i^{m-1} + \upsilon_i^{m-1} u_{i+1}^{m-1} + \psi_i^m\right] \tag{2.7}$$

The corresponding matrix form is

$$\Lambda_m \mathbf{u}_m = -\left[\Xi_{m-1} \mathbf{u}_{m-1} + \Psi_m\right] + \varepsilon_m \tag{2.8}$$

with

$$\mathbf{u}_m = \begin{bmatrix} u_1^m \\ u_2^m \\ \vdots \\ u_i^m \\ \vdots \\ u_{N_x-3}^m \\ u_{N_x-2}^m \end{bmatrix}$$

The Υ_m and Φ_m matrices are defined in the following manner

$$\Lambda_m = \begin{bmatrix}
\varphi_1^m & \chi_1^m & 0 & & & \\
\phi_2^m & \varphi_2^m & \chi_2^m & 0 & & \\
\ddots & \ddots & \ddots & \ddots & \ddots & \\
 & 0 & \phi_i^m & \varphi_i^m & \chi_i^m & 0 \\
 & & \ddots & \ddots & \ddots & \ddots \\
 & & & 0 & \phi_{N_x-2}^m & \varphi_{N_x-2}^m
\end{bmatrix}$$

$$\Xi_m = \begin{bmatrix} \tau_1^m & \upsilon_1^m & 0 & & & \\ \varsigma_2^m & \tau_2^m & \upsilon_2^m & 0 & & \\ \ddots & \ddots & \ddots & \ddots & \ddots & \\ & 0 & \varsigma_i^m & \tau_i^m & \upsilon_i^m & 0 \\ & & \ddots & \ddots & \ddots & \ddots \\ & & & 0 & \varsigma_{N_x-2}^m & \tau_{N_x-2}^m \end{bmatrix}$$

ε_m is the *residual absortion vector* (Kurpiel and Roncalli [1999])

$$\varepsilon_m = \begin{bmatrix} -\left(\phi_1^m u_0^m + \varsigma_1^{m-1} u_0^{m-1}\right) \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \\ -\left(\chi_{N_x-2}^m u_{N_x-1}^m + \upsilon_{N_x-2}^{m-1} u_{N_x-1}^{m-1}\right) \end{bmatrix}$$

2.6.1.3 Integrating the boundary conditions

A new form of the system of equations (2.8) is

$$\Lambda_m \mathbf{u}_m = \mathbf{v}_m + \varepsilon_m$$

with

$$\mathbf{v}_m = -\left[\Xi_{m-1}\mathbf{u}_{m-1} + \Psi_m\right]$$

The use of boundary conditions (Dirichlet or/and Neumann) leads us to modify this equation:

$$\Lambda_m^\star \mathbf{u}_m = \mathbf{v}_m^\star \tag{2.9}$$

with

$$\begin{aligned} \Lambda_m^\star &\longleftarrow \Lambda_m \\ \mathbf{v}_m^\star &\longleftarrow \mathbf{v}_m \\ \left(\mathbf{v}_m^\star\right)_1 &\longleftarrow -\varsigma_1^{m-1} u_0^{m-1} \\ \left(\mathbf{v}_m^\star\right)_{N_x-1} &\longleftarrow -\upsilon_{N_x-2}^{m-1} u_{N_x-1}^{m-1} \end{aligned}$$

- Conditions on x^-
  - Dirichlet: $u\left(t, x^-\right) = u_{x^-}\left(t\right)$

$$\left(\mathbf{v}_m^\star\right)_1 \longleftarrow -\phi_1^m u_{x^-}\left(t_m\right)$$

  - Neumann: $\left.\frac{\partial\, u(t,x)}{\partial\, x}\right|_{x=x^-} = u'_{x^-}\left(t\right)$

$$\begin{aligned} \left(\Lambda_m^\star\right)_{1,1} &\longleftarrow \phi_1^m \\ \left(\mathbf{v}_m^\star\right)_1 &\longleftarrow \phi_1^m u'_{x^-}\left(t_m\right) h \end{aligned}$$

- Conditions on x^+

– Dirichlet: $u\left(t,x^{+}\right)=u_{x^{+}}\left(t\right)$

$$\left(\mathbf{v}_{m}^{\star}\right)_{N_x-2}\longleftarrow-\chi_{N_x-2}^{m}u_{x^{+}}\left(t_{m}\right)$$

– Neumann: $\left.\frac{\partial u(t,x)}{\partial x}\right|_{x=x^{+}}=u_{x^{+}}^{\prime}\left(t\right)$

$$\begin{aligned}\left(\Lambda_{m}^{\star}\right)_{N_x-2,N_x-2} &\longleftarrow \chi_{N_x-2}^{m}\\ \left(\mathbf{v}_{m}^{\star}\right)_{N_x-2} &\longleftarrow -\chi_{N_x-2}^{m}u_{x^{+}}^{\prime}\left(t_{m}\right)h\end{aligned}$$

2.6.1.4 Stability of the numerical algortihms

We could show that the algorithm is stable if $\theta\geq\frac{1}{2}$. In the general case, the stability assumption is verified if

$$k\rightarrow 0\bigwedge h\rightarrow 0\bigwedge\frac{k}{h^{2}}\rightarrow 0$$

2.6.1.5 The Gauss implementation

To initialize the PDE problem, we use the `PDE` procedure with the following syntax:

```
call PDE(aProc,bProc,cProc,dProc,eProc,
        tminBound,xminBound,xmaxBound,DxminBound,DxmaxBound);
```

The online help of `PDE` is the following:

```
/*
**> PDE
**
** Purpose: Initializes the PDE problem
**
** Format: call PDE(aProc,bProc,cProc,dProc,eProc,
**                  tminBound,xminBound,xmaxBound,DxminBound,DxmaxBound);
**
** Input:    aProc - scalar, pointer to a procedure which computes a(t,x)
**           bProc - scalar, pointer to a procedure which computes b(t,x)
**           cProc - scalar, pointer to a procedure which computes c(t,x)
**           dProc - scalar, pointer to a procedure which computes d(t,x)
**           eProc - scalar, pointer to a procedure which computes e(t,x,U)
**                   -- or --
**                   scalar 0
**       tminBound - scalar, pointer to a procedure which computes U(tmin,x)
**       xminBound - scalar, pointer to a procedure which computes U(t,xmin)
**       xmaxBound - scalar, pointer to a procedure which computes U(t,xmax)
**      DxminBound - scalar, pointer to a procedure which computes dU(t,xmin)/dx
**      DxmaxBound - scalar, pointer to a procedure which computes dU(t,xmax)/dx
**
** Output:
**
** Globals:  __output - scalar (defaut = 1)
**                      1 - print information about the PDE problem
**                      0 - no printing
**
**
** Remarks:
**
*/
```

We may then solve the PDE with the PDE_Solve procedure. Its syntax is

```
call PDE_Solve(tmin,tmax,Nt,xmin,xmax,Nx,NameFile,theta);
```

The online help of PDE_Solve is the following:

```
/*
**> PDE_Solve
**
** Purpose: Solves the PDE problem with the finite difference method.
**
** Format: call PDE_Solve(tmin,tmax,Nt,xmin,xmax,Nx,NameFile,theta);
**
** Input:     tmin - scalar, value of $t^{-}$
**            tmax - scalar, value of $t^{+}$
**              Nt - scalar, value of $N_t$
**            xmin - scalar, value of $x^{-}$
**            xmax - scalar, value of $x^{+}$
**              Nx - scalar, value of $N_x$
**        NameFile - string, name of the solution dataset file
**           theta - scalar, value of $\theta$
**
** Output:
**
** Globals :  _pde_Elliptic - scalar (d\U{e9}faut = 0)
**                            0 if the PDE problem is a parabolic problem
**                            1 if the PDE problem is an elliptic problem
**
**           _pde_PrintIters - scalar (d\U{e9}faut = 0)
**                             0 - does not print iterations
**                             1 - printing after each I iterations
**
**         _pde_SaveLastIter - scalar (d\U{e9}faut = 0)
**                             0 for saving the solution of all the iterations
**                             1 for saving only the last solution
**
**                  __output - scalar (d\U{e9}faut = 1)
**                             1 - print information about the algorithm,
**                                 the mesh ratios and the convergence
**                             0 - no printing
**
** Remarks:
**
*/
```

The solution of the PDE is stored in a dataset file. In order to extract some values, we use the following command line:

```
z = PDE_readFile(NameFile,cn);
```

The online help of PDE_readFile is the following:

```
/*
**> PDE_readFile
**
** Purpose: Reads the solution dataset file of the PDE problem.
**
** Format: z = PDE_readFile(NameFile,cn);
**
** Input: NameFile - string, name of the solution dataset file
**              cn - scalar 0 to obtain the numeric solution $u_i^m$
```

```
**                    -- or --
**                    string "t" to obtain the values $t_m$
**                    -- or --
**                    string "x" to obtain the vector $x_i$
**                    -- or --
**                    "x"|xi to obtain the numeric solution $u(t,xi)$
**                    "t"|tm to obtain the numeric solution $u(tm,x)$
**
** Output:        z - matrice E*E
**
** Globals:
**
** Remarks:
**
*/
```

2.6.1.6 Link between Backward PDE and Forward PDE in option pricing

Suppose that the dynamics of the underlying price under the probability measure $\mathbb{Q}$ is given by the following SDE:

$$\left\{ \begin{array}{rcl} \mathrm{d}S(t) & = & bS(t)\,\mathrm{d}t + \sigma(t, S(t))\, S(t)\,\mathrm{d}W^{\mathbb{Q}}(t) \\ S(t_0) & = & S_0 \end{array} \right.$$

The Partial Differential Equation of pricing a call option $C(t, S)$ is:

$$\left\{ \begin{array}{l} \frac{1}{2}\sigma^2(t, S)\, S^2\, \partial_S^2 C(t, S) + b\, \partial_S C(t, S) + \partial_t C(t, S) - rC(t, S) = 0 \\ C(T, S) = (S - K)_+ \end{array} \right.$$

To solve this problem, we may write it as a Cauchy problem by considering the change of variable $\tau = T - t_0$. We also have:

$$\left\{ \begin{array}{l} \frac{1}{2}\sigma^2(\tau, S)\, S^2\, \partial_S^2 C(\tau, S) + b\, \partial_S C(\tau, S) - \partial_\tau C(\tau, S) - rC(\tau, S) = 0 \\ C(0, S) = (S - K)_+ \\ C(\tau, 0) = 0 \\ \partial_S C(\tau, +\infty) = 1 \end{array} \right. \tag{2.10}$$

The equation (2.10) is called the **Backward PDE Equation.**

Using Breeden et Litzenberger [1978], we may consider another pricing equation which does not give $C(t, S)$ but $C(t, K)$:

$$\left\{ \begin{array}{l} \frac{1}{2}\sigma^2(\tau, K)\, K^2\, \partial_K^2 C(\tau, K) - b\, \partial_K C(\tau, K) - \partial_\tau C(\tau, K) - (r - b)\, C(\tau, K) = 0 \\ C(0, K) = (S_0 - K)_+ \\ \partial_K C(\tau, +\infty) = -1 \\ C(\tau, 0) = 0 \end{array} \right. \tag{2.11}$$

The equation (2.11) is called the **Forward PDE Equation.**

The main differences between the equations (2.10) and (2.11) are:

- In the backward problem, the state variable is the price of the underlying asset whereas in the forward problem, the state variable is the strike of the option.
- In the first problem, the strike is constant whereas it is the spot price in the second problem. So, with the second problem, we may price options with several strikes in one time.

Let us consider the pricing problem of barrier options. The *backward* PDE takes the following form

$$\begin{cases} \frac{1}{2}\sigma^2(t,S)\,S^2\,\partial_S^2 U(t,S) + bS\,\partial_S U(t,S) + \partial_t U(t,S) - rU(t,S) = 0 \\ U(T,S(T)) = G(T,S(T)) \end{cases}$$

with $G(T,S(T))$ the payoff functional. In the case of a call option, $G(T,S(T))$ is equal to $(S-K)^+$ whereas we have $G(T,S(T)) = (K-S)^+$ for a put option. To solve numerically the PDE, we use the following boundary conditions and additional restrictions (we always assume that $S^- \leq L \leq H \leq S^+$ and we notice $U^- = U(t,S^-)$, $U^+ = U(t,S^+)$, $\partial_S^- = \partial_S U(t,S^-)$ and $\partial_S^+ = \partial_S U(t,S^+)$):

Type of option	C	DOC	UOC	KOC
Boundary conditions	$U^- = 0$ $\partial_S^+ = 1$	$U^- = 0$ $\partial_S^+ = 1$	$U^- = 0$ $U^+ = 0$	$U^- = 0$ $U^+ = 0$
restrictions	$\surd$	$L \leq S$	$S \leq H$	$L \leq S \leq H$

Type of option	P	DOP	UOP	KOP
Boundary conditions	$\partial_S^- = -1$ $U^+ = 0$	$U^- = 0$ $U^+ = 0$	$\partial_S^- = -1$ $U^+ = 0$	$U^- = 0$ $U^+ = 0$
restrictions	$\surd$	$L \leq S$	$S \leq H$	$L \leq S \leq H$

The *forward* PDE takes the following form

$$\begin{cases} \frac{1}{2}\sigma^2(\tau,K)\,K^2\,\partial_K^2 C(\tau,K) - bK\,\partial_K C(\tau,K) - \partial_\tau C(\tau,K) - (r-b)\,C(\tau,K) = 0 \\ C(0,K) = G(0,K) \end{cases}$$

with $G(0,K)$ the payoff functional. In the case of a call option, $G(0,K)$ is equal to $(S_0-K)^+$ whereas we have $G(0,K) = (K-S_0)^+$ for a put option. To solve numerically the PDE, we use the following boundary conditions and additional restrictions (we always assume that $K^- \leq L \leq H \leq K^+$):

Type of option	C	P
Boundary conditions	$\partial_K U(t,K^-) = -1$ $U(t,K^+) = 0$	$U(t,K^-) = 0$ $\partial_K U(t,K^+) = 1$
restrictions	$\surd$	$\surd$

The procedures `LocalVol_Backward_PDE_Solve` and `LocalVol_Forward_PDE_Solve` to price barrier options using the Dupire [1994] model use the previous framework.

Remark that in the case of vanilla options, we have:

$$\frac{1}{2}\sigma^2(T,K)\,K^2\,\partial_K^2 \mathbf{C}(T,K) - bK\,\partial_K \mathbf{C}(T,K) - \partial_T \mathbf{C}(T,K) - (r-b)\,\mathbf{C}(T,K) = 0$$

It comes that the *forward* PDE for the binary option in the smiled model is:

$$\begin{aligned} \frac{1}{2}\sigma^2(T,K)\,K^2\,\partial_K^2 \mathbf{BC}(T,K) + \left[\sigma^2(T,K)\,K + \sigma(T,K)\,p(T,K)\,K^2 - bK\right]\partial_K \mathbf{BC}(T,K) & \\ -\partial_T \mathbf{BC}(T,K) - r\mathbf{BC}(T,K) &= 0 \end{aligned}$$

with

$$p(T,K) = \partial_K \sigma(T,K)$$

2.6.2 Numerical integration

Sometimes, the price of an option is expressed as the expected value of the payoff function $G(S(T))$ under the risk-neutral probability measure $\mathbb{Q}$:

$$\begin{aligned} C &= e^{-r\tau}\mathbb{E}^{\mathbb{Q}}\left[G(S(T))|\mathcal{F}_{t_0}\right] \\ &= e^{-r\tau}\int G(x)\,\mathrm{d}\mathbb{Q}(x) \end{aligned}$$

In this case, we provide several procedures for numerical integration. Gauss quadratures (Hermite, Jacobi, Laguerre, Legendre) are implemented in 1, 2 or 3 dimensions. For example, we have:

```
I = quadLegendre3(&f,xl,yl,zl,N);
```

With this procedure, we may compute the following integral:

$$I = \int_{x^-}^{x^+}\int_{y^-}^{y^+}\int_{z^-}^{z^+} f(x,y,z)\,\mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z$$

N is the quadrature order of the algorithm.

Two other procedures are very interesting. The first one `Simpson1` is an implementation of Simpson algorithm:

```
/*
**> Simpson1
**
** Purpose: 1D Integration (Simpson).
**
** Format:    {I,retcode} = Simpson1(&f,xl,tol,NumIters);
**
** Input:    &f - pointer to the procedure containing the function to be integrated.
**           xl - matrix 2*1, the limits of x.
**                The first row is the upper limit and the second row is the lower limit.
**          tol - scalar, tolerance of the algorithm
**     NumIters - scalar, number of iterations allowed
**
** Output:    I - vector M*1
**      retcode - scalar
**                0 if convergence, otherwise 1
**
** Globals: _quad_mtd - scalar, m\U{e9}thod to compute the knots and the weights (default = 1)
**                      1 for the Newton-Raphson algorithm
**                      2 for the SVD algorithm
**
** Remarks:
**
*/
```

The second procedure is `intCompute1D`:

```
/*
**> intCompute1D
**
** Purpose: 1D Integration.
**
** Format:    I = intCompute1D(&f,xl);
**
```

```
** Input:    &f - pointer to the procedure containing the function to be integrated.
**           xl - matrix 2*1, the limits of x.
**                The first row is the upper limit and the second row is the lower limit.
**
** Output:    I - scalar
**
** Globals: _quad_mtd - scalar, m\U{e9}thod to compute the knots and the weights (default = 1)
**                      1 for the Newton-Raphson algorithm
**                      2 for the SVD algorithm
**           _int_mtd - scalar, integration algorithm (default = 1)
**                      5 for simpson algorithm
**                      4 for trapezoidal algorithm
**                      3 for Hermite quadrature
**                      2 for Laguerre quadrature
**                      1 for Legendre quadrature
**
** Remarks:
**
*/
```

2.6.3 Monte Carlo and quasi-Monte Carlo methods

2.6.3.1 The Black and Scholes model

The dynamics of the underlying price in the univariate BS model is a Geometric Brownian motion. The SDE representation is:

$$\left\{ \begin{array}{rcl} \mathrm{d}x\left(t\right) & = & \mu x\left(t\right)\,\mathrm{d}t + \sigma x\left(t\right)\,\mathrm{d}W\left(t\right) \\ x\left(0\right) & = & x_0 \end{array} \right.$$

The solution is:

$$x\left(t\right) = x_0 e^{\left(\mu - \frac{1}{2}\sigma^2\right)t + \sigma W(t)}$$

To simulate a GBM, we use the following simulation scheme:

$$\ln x\left(t_i\right) - \ln x\left(t_{i-1}\right) = \left(\mu - \frac{1}{2}\sigma^2\right)\left(t_i - t_{i-1}\right) + \sigma\sqrt{t_i - t_{i-1}}\varepsilon_i$$

with $\varepsilon_i \sim \mathcal{N}\left(0,1\right)$. The corresponding Gauss procedure is `simulate_GBM` and its syntax is:

```
x = simulate_GBM(x0,mu,sigma,t,Ns);
```

The online help is

```
/*
**> simulate_GBM
**
** Purpose: Simulates geometric brownian motion processes (exact scheme).
**
** Format:   x = simulate_GBM(x0,mu,sigma,t,Ns);
**
** Input:        x0 - vector 1*Ns ou scalar, initial position $x_0$
**               mu - vector 1*Ns ou scalar, parameter $\mu$
**            sigma - vector 1*Ns ou scalar, parameter $\sigma$
**                t - vector Nt*1, fixing dates $t_i$
**               Ns - scalar, number of simulations $N_s$
**
** Output:        x - matrix Nt*Ns, matrix of the simulated paths
**
```

```
** Globale:
**
** Remarks:
**
*/
```

In the multivariate case, we have:

$$\begin{cases} \mathrm{d}x_k(t) &= \mu_k x_k(t)\,\mathrm{d}t + \sigma_k x_k(t)\,\mathrm{d}W_k(t) \\ x_k(0) &= x_{k,0} \end{cases}$$

with

$$\mathbb{E}\left[W_k(t)\,W_l(t)\right] = \rho_{k,l}t$$

The solution is:

$$x_k(t) = x_{k,0}e^{\left(\mu_k - \frac{1}{2}\sigma_k^2\right)t + \sigma_k W_k(t)}$$

To simulate a mGBM, we use the following simulation scheme:

$$\ln x_k(t_i) - \ln x_k(t_{i-1}) = \left(\mu_k - \frac{1}{2}\sigma_k^2\right)(t_i - t_{i-1}) + \sigma_k\sqrt{t_i - t_{i-1}}\varepsilon_{k,i}$$

with $(\varepsilon_{1,i}, \varepsilon_{2,i}, \cdots)$ a Gaussian vector with correlation matrix $\left(\rho_{k,l}\right)$. The syntax of the `simulate_mGBM` procedure is:

```
x = simulate_mGBM(x0,mu,sigma,rho,t,Ns);
```

The procedure returns a 3D array with dimensions $N_t \times N_s \times N_x$. In order to manipulate the array in an easy way, one may use the procedures `y = mGBM_getDate(t,x,t0)`, `y = mGBM_getSimul(x,indx)` and `y = mGBM_getAsset(x,indx)`. Remark that in the 2D case, a more convenient way to simulate mGBM is:

```
{x1,x2} = simulate_GBM2(x01,x02,mu1,mu2,sigma1,sigma2,rho,t,Ns);
```

2.6.3.2 Other diffusion processes

Both MC and quasi MC methods are implented for the procedures described below. The first thing is to initialize the random number generator:

```
call rndset(algr,dim,coord);
```

The online help of the `rndset` is:

```
/*
**> rndset
**
** Purpose: Initializes the random number generator.
**
** Format:    call rndset(algr,dim,coord);
**
** Input:       algr - string
**                     "rnd" or "gauss" for the LCG algorithm of Gauss
**                     "rnd1" for the Park & Miller generator
**                     "rnd2" for the L'ecuyer generator
**                     "Sobol" for the Sobol generator
**                     "Faure" for the Faure generator
```

```
**               dim - dimension of the Sobol or Faure generator
**                     0 to initialize the generator
**             coord - vector c*1, coordinates
**
** Output:
**
** Globals:  _rnd_algr - scalaire
**
*/
```

For the GBM process, the syntax of the procedure is:

```
x = qmc_simulate_GBM(x0,mu,sigma,t,Ns);
```

The Ornstein-Uhlenbeck process is defined by the following SDE representation:

$$\mathrm{d}x\left(t\right) = a\left(b - x\left(t\right)\right)\,\mathrm{d}t + \sigma\,\mathrm{d}W\left(t\right)$$

The solution is:

$$x\left(t\right) = x_0 e^{-a(t-t_0)} + b\left(1 - e^{-a(t-t_0)}\right) + \sigma\int_{t_0}^{t} e^{a(\theta-t)}\,\mathrm{d}W\left(\theta\right)$$

The exact discretization scheme is then:

$$x\left(t_{i+1}\right) = e^{-\alpha(t_{i+1}-t_i)}x\left(t_i\right) + \mu\left(1 - e^{-\alpha(t_{i+1}-t_i)}\right) + \sigma\sqrt{\frac{1 - e^{-2\alpha(t_{i+1}-t_i)}}{2a}}\varepsilon_i$$

with $\varepsilon_i \sim \mathcal{N}\left(0,1\right)$. The corresponding Gauss procedure is `qmc_simulate_OU` and its syntax is:

```
x = qmc_simulate_OU(x0,a,b,sigma,t,Ns);
```

In the case of a general SDE:

$$\mathrm{d}x\left(t\right) = \mu\left(t, x\left(t\right)\right)\,\mathrm{d}t + \sigma\left(t, x\left(t\right)\right)\,\mathrm{d}W\left(t\right)$$

or a Jump-Diffusion process:

$$\mathrm{d}x\left(t\right) = \mu\left(t, x\left(t\right)\right)\,\mathrm{d}t + \sigma\left(t, x\left(t\right)\right)\,\mathrm{d}W\left(t\right) + \kappa\left(t, x\left(t\right)\right)\,\mathrm{d}N\left(t\right)$$

where $N\left(t\right)$ a Poisson process with intensity $\lambda\left(t, x\left(t\right)\right)$, we use the Euler-Maruyama scheme and we have:

```
x = qmc_simulate_SDE(x0,&procMu,&procSigma,t,Ns);
```

and

```
x = qmc_simulate_JDP(x0,procMu,procSigma,procKappa,procLambda,t,Ns);
```

Two other procedures are provided for bivariate and multivaraite SDE. Their names are `qmc_simulate_SDE2` and `qmc_simulate_mSDE`.

2.6.3.3 Reduction variance techniques based on antithetic variables

Antithetic variables are available for the previous algorithms. For example, we have:

```
{x,xtilde} = qmc_simulate_GBM_av(x0,mu,sigma,t,Ns);
```

or

```
{x,xtilde} = qmc_simulate_OU_av(x0,a,b,sigma,t,Ns);
```

2.6.3.4 Simulating binomial trees

We have provided a procedure to simulate binomial trees:

```
{x,xtilde} = Simulate_CRR_av(x0,sigma,tau,b,r,n,nS);
```

Chapter 3

Some examples

In this chapter, we present some examples of using the option library. You will find more examples in the **[gauss root]\option\examples** directory

3.1 Pricing vanilla options with closed-form formulas

3.1.1 Black-Scholes examples

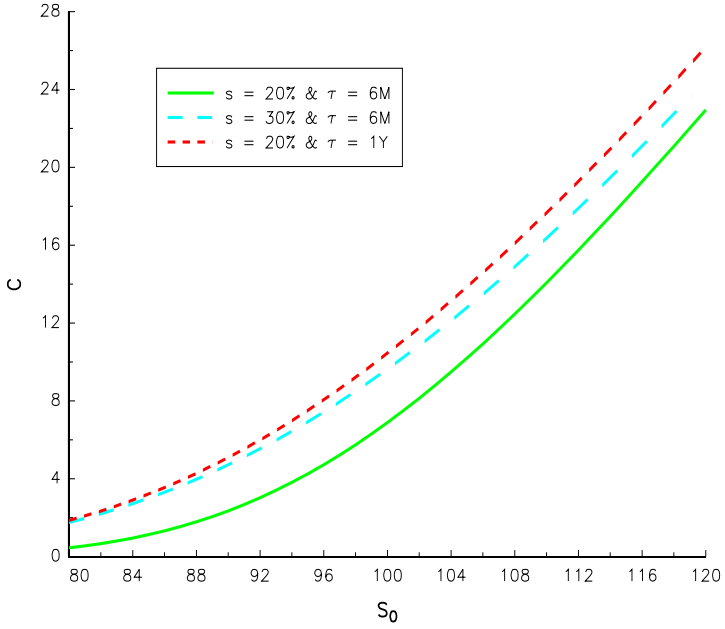
In the following program, we want to price an European option with $K = 100$, $b = r = 5\%$, a 6M maturity and a volatility of 20%. We compar these prices with those obtained with a 6M maturity and a volatility of 30%, and a 1Y maturity and a volatility of 20%. The results are reported in Figure 3.1.

```
new;
library option,pgraph;

S0 = seqa(80,1,41);
K = 100;
sigma = 0.20~0.30~0.20;
r = 0.05;
b = r;
tau = 0.5~0.5~1.0;

C = EuropeanBS(S0,K,sigma,tau,b,r);

graphset;
  fonts("simplex simgrma");
  _pdate = ""; _pnum = 2; _pframe = 0;
  _pltype = 6|1|3; _plwidth = 10;
  xlabel("\216S]0["); ylabel("\216C");
  _plegstr = "\216s\201 = 20% & \202t\201 = 6M\000\216s\201 = 30% & \202t\201 = 6M\000\216s\201 = 20% & \202t\201 = 1Y";
  _plegctl = {2 5 2 5};
  graphprt("-c=1 -cf=bs1.eps");
  xy(S0,C);
```

The next program is an example of computing Gamma hedging coefficients (Figure 3.2).

```
new;
library option,pgraph;

S0 = seqa(80,1,41);
K = 100;
sigma = 0.20;
r = 0.05;
b = r;
tau = seqa(0.02,0.01,50)';
```

Figure 3.1: bs1.eps



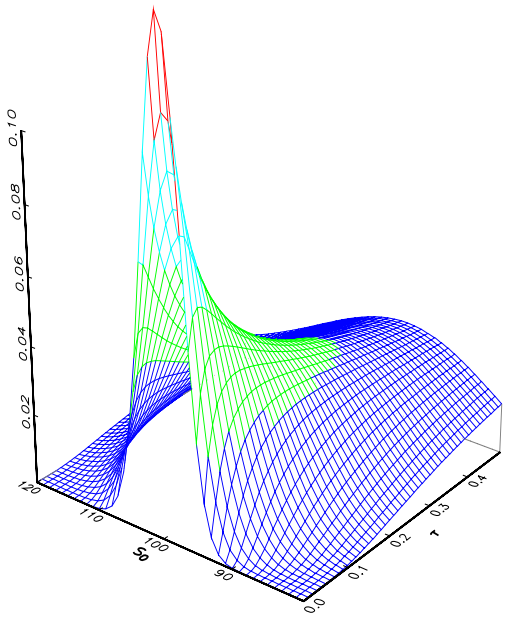

```
G = EuropeanBS_Gamma(S0,K,sigma,tau,b,r);

graphset;
  fonts("simplex simgrma");
  _pdate = ""; _pnum = 2; _pframe = 0;
  _pltype = 6|1|3; _plwidth = 10;
  _pzclr = 9|10|11|12;
   xlabel("\216\202t\201"); ylabel("\216S]0[");
  xtics(0,0.5,0.1,2); ytics(80,120,10,2); ztics(0,0.10,0.02,2);
  graphprt("-c=1 -cf=bs2.eps");
  surface(tau,S0,G);
```

3.1.2 Cox-Ross-Rubinstein examples

Let us consider the pricing of an american option using the CRR model. The parameters are $S_0 = 100$, $K = 100$, $\sigma = 20\%$, $\tau = 1$ (1Y maturity) and $b = r = 5\%$. The number of time steps is $N = 7$. We print the tree of the dynamics of the underlyings and the tree of the option prices using the `PrintTree` procedure. Then, we plot the tree of the option prices in Figure 3.3 in the case $N = 20$.

```
new;
library option,gWizard,pgraph;

cls;

S0 = 100;
K = 100;
sigma = 0.20;
tau = 1;
r = 0.05;
b = r;
N = 7;
```

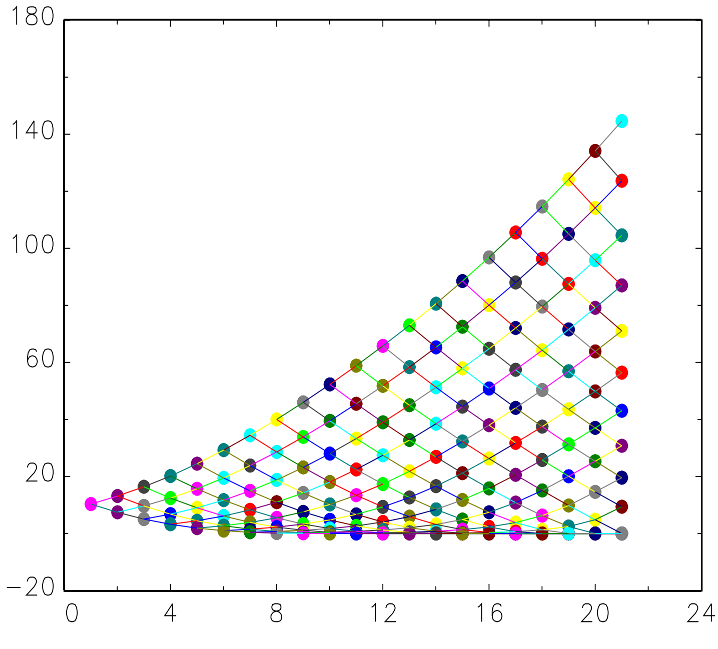
Figure 3.2: bs2.eps




```
{C,tree} = AmericanCRR(S0,K,sigma,tau,b,r,N);

treeC = vread(tree,"CallPut");
treeS = vread(tree,"S");

Tree_S = vread(tree,"S");
Tree_CallPut = vread(tree,"callput");

print "Tree of the underlying asset price";
call PrintTree(Tree_S,0);

print "Tree of the option price";
call PrintTree(Tree_CallPut,0);

print ftos(C,"Option price = %lf",4,4);

N = 20;
{C,tree} = AmericanCRR(S0,K,sigma,tau,b,r,N);
tree = vread(tree,"CallPut");

graphset;
  gBuffer = PlotTree(tree,4);
  call gWizardInfo(gBuffer);
  _paxht = 0.20; _pnumht = 0.20; _ptitlht = 0.20;
  gWizardSet(gBuffer,1);
  _psym[.,4] = 4*ones(rows(_psym),1);
  _psym[.,5] = ceil(rndu(rows(_psym),1)*14);
  _pline[.,8] = ceil(rndu(rows(_pline),1)*14);
  graphprt("-c=1 -cf=crr4.eps");
  gWizardDraw(gBuffer,1);


Tree of the underlying asset price

                                                                169.74893
                                                    157.39012
                                        145.93111                145.93111
```

```
                                                                    135.30639                          135.30639
                                                   125.45522                          125.45522                          125.45522
                                  116.32128                         116.32128                          116.32128
                 107.85234                         107.85234                          107.85234                          107.85234
100.00000                         100.00000                         100.00000                          100.00000
                 92.719360                         92.719360                          92.719360                          92.719360
                                  85.968796                         85.968796                          85.968796
                                                   79.709717                          79.709717                          79.709717
                                                                    73.906339                          73.906339
                                                                                      68.525485                          68.525485
                                                                                                       63.536390
                                                                                                                         58.910534
Tree of the option price

                                                                                                                         69.748927
                                                                                                       58.101859
                                                                                      47.349525                          45.931110
                                                                    37.426451                          36.018131
                                                   28.621683                          26.873634                          25.455218
                                  21.203616                         19.188409                          17.033016
                 15.260032                         13.211784                          10.866507                          7.8523412
10.703790                         8.8304444                         6.7140249                          4.1202721
                 5.7598702                         4.0540769                          2.1619847                          0.00000000
                                  2.4059296                         1.1344343                          0.00000000
                                                   0.59525916                         0.00000000                         0.00000000
                                                                    0.00000000                         0.00000000
                                                                                      0.00000000                         0.00000000
                                                                                                       0.00000000
                                                                                                                         0.00000000

Option price = 10.7038
```

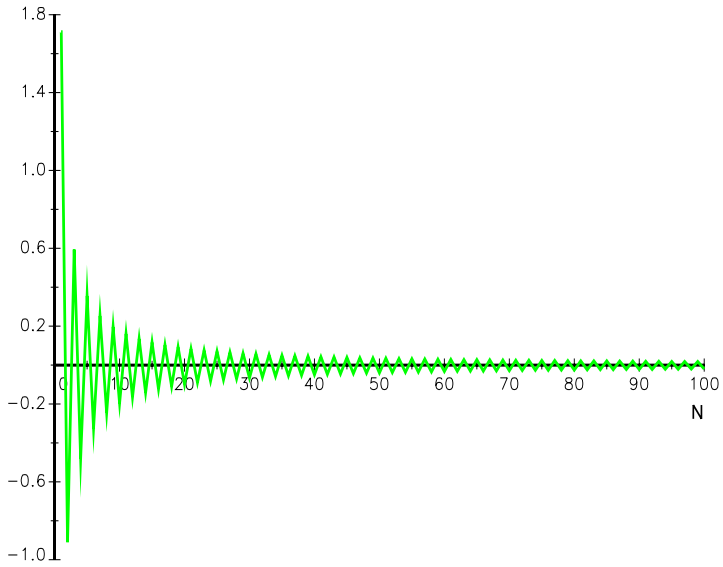
Figure 3.3: crr4.eps



The following example illustrates the convergence of the CRR price to the BS price when N tends to infinity, that is when the discrete model is closed to the continuous model. In Figure 3.4, we have reported the difference between the CRR price and the BS price. We verify that the difference becomes smaller when N tends to infinity.

```
new;
library option,gWizard,pgraph;

cls;

S0 = 100;
K = 100;
sigma = 0.20;
tau = 1;
r = 0.05;
b = r;

nMax = 100;

C = zeros(nMax,1);
N = 1;
do until N > nMax;
  {C[N],tree} = EuropeanCRR(S0,K,sigma,tau,b,r,N);
  N = N + 1;
endo;

BS = EuropeanBS(S0,K,sigma,tau,b,r);

graphset;
  _pdate = ""; _pnum = 2; _pframe = 0;
  _pltype = 6; _plwidth = 10;
  xlabel("\216N"); _pcross = 1;
  graphprt("-c=1 -cf=crr5.eps");
  xy(0,C-BS);
```

Figure 3.4: crr5.eps



3.1.3 American options

The next program reproduces the results of Barone-Adesi and Whaley [1987] published in the *Journal of Finance*.

```
/*
** Barone-Adesi, G. and R. E. Whaley [1987], Efficient analytic
** approximation of american option values, Journal of Finance,
** 42, pages 301-320
*/

new;
library option;

K = 100;
e = ones(5,1);
r = (0.08|0.12|0.08|0.08).*.e;
sigma = (0.20|0.20|0.40|0.20).*.e;
tau = (0.25|0.25|0.25|0.50).*.e;
S0 = ones(4,1).*.seqa(80,10,5);

begin_ct = hsec;

/* Table I */

b = -0.04*ones(20,1);

_Option_Type = "call";
C = AmericanBS(S0,K,sigma,tau,b,r);
_Option_Type = "put";
P = AmericanBS(S0,K,sigma,tau,b,r);

Print "                          Table I       ";
call Print_Table(b,r,sigma,tau,S0,C,P);


/* Table II */

b = 0.04*ones(20,1);

_Option_Type = "call";
C = AmericanBS(S0,K,sigma,tau,b,r);
_Option_Type = "put";
P = AmericanBS(S0,K,sigma,tau,b,r);

Print "                          Table II       ";
call Print_Table(b,r,sigma,tau,S0,C,P);


/* Table III */

b = 0.00*ones(20,1);

_Option_Type = "call";
C = AmericanBS(S0,K,sigma,tau,b,r);
_Option_Type = "put";
P = AmericanBS(S0,K,sigma,tau,b,r);

Print "                          Table III       ";
call Print_Table(b,r,sigma,tau,S0,C,P);


/* Table IV */

b = r;
_Option_Type = "call";
C = AmericanBS(S0,K,sigma,tau,b,r);
_Option_Type = "put";
P = AmericanBS(S0,K,sigma,tau,b,r);

Print "                          Table IV       ";
call Print_Table(b,r,sigma,tau,S0,C,P);

/* Table V */

b = (-0.04|0.00|0.04|0.08).*.e;
r = (0.08|0.08|0.08|0.08).*.e;
sigma = (0.20|0.20|0.20|0.20).*.e;
tau = (3|3|3|3).*.e;
```

```
_Option_Type = "call";
C = AmericanBS(S0,K,sigma,tau,b,r);
_Option_Type = "put";
P = AmericanBS(S0,K,sigma,tau,b,r);

Print "                          Table V       ";
call Print_Table(b,r,sigma,tau,S0,C,P);

end_ct = hsec;

print ftos((end_ct-begin_ct)/100,"Computational Time: %lf seconds",5,3);


proc (0) = Print_Table(b,r,sigma,tau,S0,C,P);
  local mask,fmt,omat;

  mask = 1;
  let fmt[7,3]=   "*.*lf" 8 2 "*.*lf" 8 2  "*.*lf" 8 2
                  "*.*lf" 8 2 "*.*lf" 8 0 "*.*lf" 15 3  "*.*lf" 15 3;

  omat = b~r~sigma~tau~S0~C~P;
  print "     b       r     sigma    tau       S0          Call           Put";
  print "-------------------------------------------------------------------"\
        "-----";
  call printfm(omat,mask,fmt);
  print;  print;
  retp;
endp;
```

Table I

b	r	sigma	tau	S0	call	Put
-0.04	0.08	0.20	0.25	80	0.032	20.419
-0.04	0.08	0.20	0.25	90	0.590	11.251
-0.04	0.08	0.20	0.25	100	3.525	4.397
-0.04	0.08	0.20	0.25	110	10.315	1.118
-0.04	0.08	0.20	0.25	120	20.000	0.184
-0.04	0.12	0.20	0.25	80	0.032	20.248
-0.04	0.12	0.20	0.25	90	0.587	11.146
-0.04	0.12	0.20	0.25	100	3.506	4.355
-0.04	0.12	0.20	0.25	110	10.288	1.107
-0.04	0.12	0.20	0.25	120	20.000	0.183
-0.04	0.08	0.40	0.25	80	1.067	21.463
-0.04	0.08	0.40	0.25	90	3.284	13.927
-0.04	0.08	0.40	0.25	100	7.411	8.274
-0.04	0.08	0.40	0.25	110	13.502	4.523
-0.04	0.08	0.40	0.25	120	21.233	2.297
-0.04	0.08	0.20	0.50	80	0.229	20.982
-0.04	0.08	0.20	0.50	90	1.387	12.645
-0.04	0.08	0.20	0.50	100	4.724	6.372
-0.04	0.08	0.20	0.50	110	10.955	2.650
-0.04	0.08	0.20	0.50	120	20.000	0.919

Table II

b	r	sigma	tau	S0	call	Put
0.04	0.08	0.20	0.25	80	0.052	20.000
0.04	0.08	0.20	0.25	90	0.849	10.183
0.04	0.08	0.20	0.25	100	4.441	3.544
0.04	0.08	0.20	0.25	110	11.662	0.798
0.04	0.08	0.20	0.25	120	20.898	0.118
0.04	0.12	0.20	0.25	80	0.052	20.000
0.04	0.12	0.20	0.25	90	0.841	10.161
0.04	0.12	0.20	0.25	100	4.397	3.525
0.04	0.12	0.20	0.25	110	11.547	0.794
0.04	0.12	0.20	0.25	120	20.694	0.118
0.04	0.08	0.40	0.25	80	1.289	20.528
0.04	0.08	0.40	0.25	90	3.823	12.927
0.04	0.08	0.40	0.25	100	8.350	7.456
0.04	0.08	0.40	0.25	110	14.797	3.958
0.04	0.08	0.40	0.25	120	22.716	1.954

0.04	0.08	0.20	0.50	80	0.414	20.000
0.04	0.08	0.20	0.50	90	2.180	10.706
0.04	0.08	0.20	0.50	100	6.496	4.772
0.04	0.08	0.20	0.50	110	13.425	1.760
0.04	0.08	0.20	0.50	120	22.060	0.546

Table III

b	r	sigma	tau	S0	call	Put
0.00	0.08	0.20	0.25	80	0.040	20.000
0.00	0.08	0.20	0.25	90	0.702	10.578
0.00	0.08	0.20	0.25	100	3.927	3.927
0.00	0.08	0.20	0.25	110	10.811	0.940
0.00	0.08	0.20	0.25	120	20.016	0.146
0.00	0.12	0.20	0.25	80	0.040	20.000
0.00	0.12	0.20	0.25	90	0.698	10.526
0.00	0.12	0.20	0.25	100	3.900	3.900
0.00	0.12	0.20	0.25	110	10.754	0.934
0.00	0.12	0.20	0.25	120	20.001	0.145
0.00	0.08	0.40	0.25	80	1.169	20.927
0.00	0.08	0.40	0.25	90	3.534	13.394
0.00	0.08	0.40	0.25	100	7.844	7.844
0.00	0.08	0.40	0.25	110	14.085	4.226
0.00	0.08	0.40	0.25	120	21.856	2.116
0.00	0.08	0.20	0.50	80	0.303	20.040
0.00	0.08	0.20	0.50	90	1.723	11.481
0.00	0.08	0.20	0.50	100	5.478	5.478
0.00	0.08	0.20	0.50	110	11.902	2.149
0.00	0.08	0.20	0.50	120	20.335	0.703

Table IV

b	r	sigma	tau	S0	call	Put
0.08	0.08	0.20	0.25	80	0.069	20.000
0.08	0.08	0.20	0.25	90	1.025	10.013
0.08	0.08	0.20	0.25	100	5.017	3.220
0.08	0.08	0.20	0.25	110	12.620	0.681
0.08	0.08	0.20	0.25	120	22.067	0.097
0.12	0.12	0.20	0.25	80	0.090	20.000
0.12	0.12	0.20	0.25	90	1.217	10.000
0.12	0.12	0.20	0.25	100	5.582	2.925
0.12	0.12	0.20	0.25	110	13.474	0.578
0.12	0.12	0.20	0.25	120	23.021	0.079
0.08	0.08	0.40	0.25	80	1.427	20.248
0.08	0.08	0.40	0.25	90	4.148	12.514
0.08	0.08	0.40	0.25	100	8.916	7.100
0.08	0.08	0.40	0.25	110	15.609	3.712
0.08	0.08	0.40	0.25	120	23.742	1.807
0.08	0.08	0.20	0.50	80	0.570	20.000
0.08	0.08	0.20	0.50	90	2.755	10.235
0.08	0.08	0.20	0.50	100	7.706	4.193
0.08	0.08	0.20	0.50	110	15.233	1.446
0.08	0.08	0.20	0.50	120	24.297	0.424

Table V

b	r	sigma	tau	S0	call	Put
-0.04	0.08	0.20	3.00	80	2.524	26.245
-0.04	0.08	0.20	3.00	90	4.966	20.641
-0.04	0.08	0.20	3.00	100	8.670	15.990
-0.04	0.08	0.20	3.00	110	13.882	12.221
-0.04	0.08	0.20	3.00	120	20.882	9.235
0.00	0.08	0.20	3.00	80	4.203	22.395
0.00	0.08	0.20	3.00	90	7.537	16.498
0.00	0.08	0.20	3.00	100	12.030	12.030
0.00	0.08	0.20	3.00	110	17.641	8.687
0.00	0.08	0.20	3.00	120	24.297	6.222
0.04	0.08	0.20	3.00	80	6.965	20.325
0.04	0.08	0.20	3.00	90	11.621	13.563
0.04	0.08	0.20	3.00	100	17.399	9.108
0.04	0.08	0.20	3.00	110	24.092	6.122

```
   0.04    0.08    0.20    3.00     120         31.491          4.115
   0.08    0.08    0.20    3.00      80         11.590         20.000
   0.08    0.08    0.20    3.00      90         18.120         11.634
   0.08    0.08    0.20    3.00     100         25.743          6.962
   0.08    0.08    0.20    3.00     110         34.163          4.257
   0.08    0.08    0.20    3.00     120         43.134          2.640


Computational Time: 0.031 seconds
```

3.2 Implied volatilities, volatility skew and volatility smile

3.2.1 Computing the smile

To compute the smile, we have to define the option prices and to invert the BS formula using the `EuropeanBS_ImpVol` procedure. In the next program, we first compute the implied volatilities for BS prices and for two other sets of option prices:

K	90	95	98	99	100	101	102	105	110
Set #1	17.0	13.5	11.6	11.0	10.4	9.9	9.5	8.3	7.0
Set #2	18.0	14.25	12.0	11.0	10.4	10.1	9.7	8.6	7.8

In the case of BS, we verify that the smile curve is flat (see Figure 3.5).

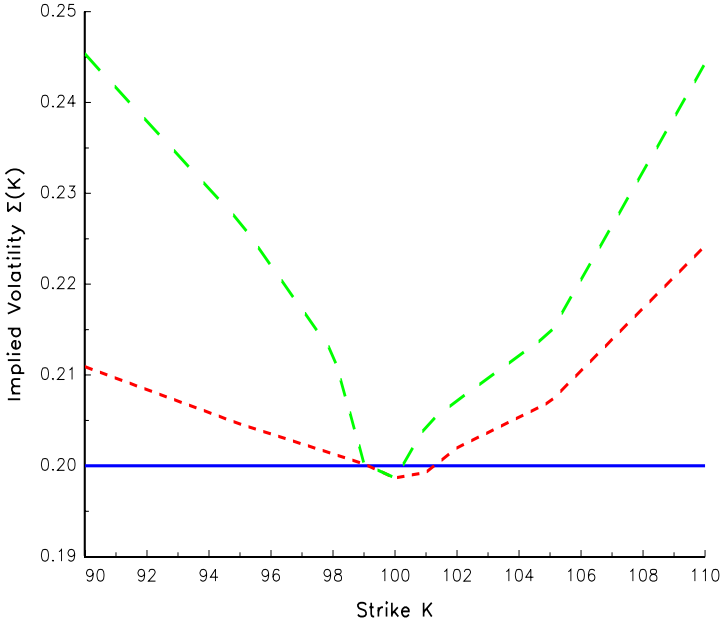
```
new;
library option,pgraph;

S0 = 100;
sigma = 0.20;
r = 0.05;
b = r;
tau = 1.0;

let K = 90 95 98 99 100 101 102 105 110;

C1 = EuropeanBS(S0,K,0.20,tau,b,r);
ImpVol1 = EuropeanBS_ImpVol(S0,K,tau,b,r,C1);

let C2 = 17.0 13.5 11.6 11.0 10.4 9.9 9.5 8.3 7.0;
ImpVol2 = EuropeanBS_ImpVol(S0,K,tau,b,r,C2);

let C3 = 18.0 14.25 12.0 11.0 10.4 10.1 9.7 8.6 7.8;
ImpVol3 = EuropeanBS_ImpVol(S0,K,tau,b,r,C3);

graphset;
  fonts("simplex simgrma");
  _pdate = ""; _pnum = 2; _pframe = 0;
  _pltype = 6|3|1; _plwidth = 10; _pcolor = 9|12|10;
  xlabel("\216Strike K");
  ylabel("\216Implied Volatility \202S\201(K)");
  graphprt("-c=1 -cf=smile1.eps");
  xy(K,ImpVol1~ImpVol2~ImpVol3);
```

3.2.2 Smile in the Heston model

In the following program, we compute the smile in the Heston model. Remark that the `EuropeanHeston_ImpVol` procedure is just a combination of the `EuropeanHeston` and `EuropeanBS_ImpVol` procedures.

```
proc (1) = EuropeanHeston_ImpVol(S0,K,tau,r,Sigma0,kappa,theta,sigmaV,rho,lambda);
  local C,P,ImpVol;

  {C,P} = EuropeanHeston(S0,K,tau,r,sigma0,kappa,theta,sigmaV,rho,lambda);
```

Figure 3.5: smile1.eps



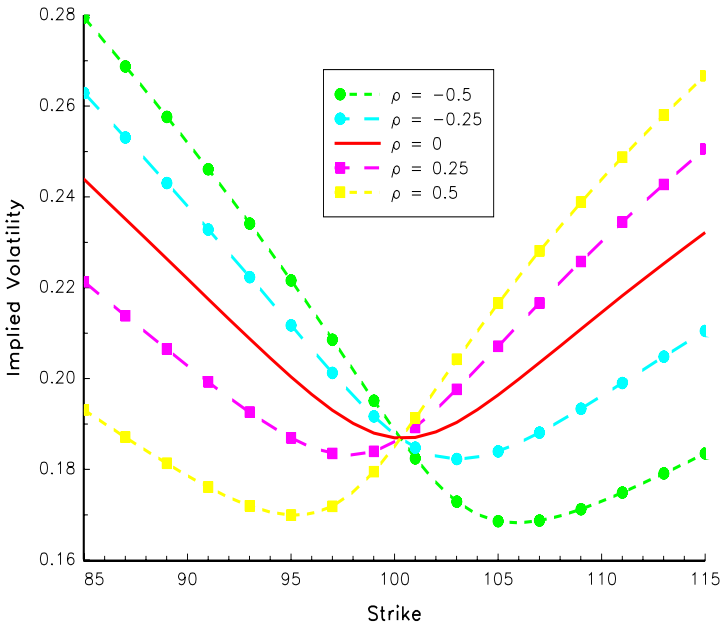

```
  if _option_type == "call";
    ImpVol = EuropeanBS_ImpVol(S0,K,tau,r,r,C);
  elseif  _option_type == "put";
    ImpVol = EuropeanBS_ImpVol(S0,K,tau,r,r,C);
  endif;

  retp(ImpVol);
endp;
```

We consider the following set of parameters: $S_0 = 100$, $V_0 = \theta = (20\%)^2$, $\kappa = 0.5$, $\sigma_V = 0.9$ and $\lambda = 0$. We compute the implied volatilities for a 1M maturity and strikes in the range $[85; 115]$. In Figure 3.6, we report the smile for various values of ρ.

```
new;
library option,pgraph;
optionSet;

cls;

S0 = 100;
tau = 1/12;
r = 0.05;
sigma =  0.20;

V0 = 0.2^2;

theta = V0;
Kappa = 0.5;
sigmaV = 0.9;
lambda = 0;

K = seqa(85,1,31);
nK = rows(K);

rho = seqa(-0.5,0.25,5);
```

```
nRho = rows(rho);

ImpVol = zeros(nK,nRho);

_int_order = 128;
_int_mtd = 2;

i = 1;
do until i > nRho;
  ImpVol[.,i] = EuropeanHeston_ImpVol(S0,K,tau,r,V0,kappa,theta,sigmaV,rho[i],lambda);
  i = i + 1;
endo;

graphset;
  _pdate = ""; _pnum = 2; _pframe = 0; _plwidth = 10;
  _pltype = 3|1|6|1|3;
  _pstype = 8|8|10|9|9;
  _plctrl = 2|2|0|2|2;
  fonts("simplex simgrma");
  xtics(85,115,5,5);
  xlabel("\216Strike");
  ylabel("\216Implied Volatility");
  _plegstr = "\202r\201 = -0.5\000\202r\201 = -0.25\000\202r\201 = 0"\
             "\000\202r\201 = 0.25\000\202r\201 = 0.5";
  _plegctl = {2 5 4 4.5};
  graphprt("-c=1 -cf=heston2.eps");
  xy(K,ImpVol);
```

Figure 3.6: heston2.eps



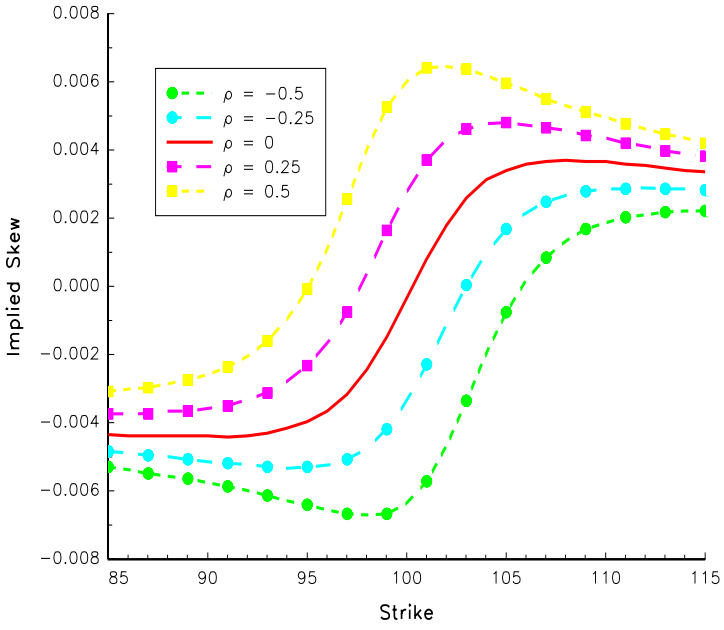
3.2.3 Computing the skew

This is the same program as previously, but we have replaced the `EuropeanHeston_ImpVol` procedure by the `EuropeanHeston_Skew` procedure (see Figure 3.7).

```
Skew = zeros(nK,nRho);
```

```
_int_order = 128;
_int_mtd = 2;

dK = 0.25;

i = 1;
do until i > nRho;
  Skew[.,i] = EuropeanHeston_Skew(S0,K,tau,r,V0,kappa,theta,sigmaV,rho[i],lambda,dK);
  i = i + 1;
endo;

graphset;
  _pdate = ""; _pnum = 2; _pframe = 0; _plwidth = 10;
  _pltype = 3|1|6|1|3;
  _pstype = 8|8|10|9|9;
  _plctrl = 2|2|0|2|2;
  fonts("simplex simgrma");
  xtics(85,115,5,5);
  xlabel("\216Strike");
  ylabel("\216Implied Skew");
  _plegstr = "\202r\201 = -0.5\000\202r\201 = -0.25\000\202r\201 = 0"\
             "\000\202r\201 = 0.25\000\202r\201 = 0.5";
  _plegctl = {2 5 2 4.5};
  graphprt("-c=1 -cf=heston3.eps");
  xy(K,Skew);
```

Figure 3.7: heston3.eps



3.2.4 An example with binary options

In the following examples, we compare the binary call prices for the BS model (BC0), the Heston model (BC2) and the smiled model (BC1) when we use the Skew computed by the `EuropeanHeston_Skew` procedure. Remark that we may use numerical derivatives to find the Heston prices (BC4). If we compare the columns 3 and 4 of the printed results, we obtain the same

values. It means that the true formula implemented in the `BinaryHeston` procedure is equivalent to the first derivative of the European Heston price with respect to the strike. We also remark that the Heston prices are closer to the smiled prices than the BS prices.

```
new;
library option,pgraph;
optionSet;

cls;

K = seqa(80,10,5);
nK = rows(K);

sigma = 0.20;
tau = 1/12;
r = 0.05;
S0 = 100;
V0 = 0.2^2;
theta = V0;
Kappa = 0.5;
sigmaV = 0.25;
lambda = 0;

rho = 0.5;

_int_order = 128;
_int_mtd = 2;
dK = 0.10;
ImpVol = EuropeanHeston_ImpVol(S0,K,tau,r,V0,kappa,theta,sigmaV,rho,lambda);
Skew = EuropeanHeston_Skew(S0,K,tau,r,V0,kappa,theta,sigmaV,rho,lambda,dK);

proc (1) = BCC_K(K);
  local C,P;
  {C,P} = EuropeanHeston(S0,K,tau,r,V0,kappa,theta,sigmaV,rho,lambda);
  retp(-C);
endp;

BC0 = BinaryBS(S0,K,ImpVol,tau,r,r);
BC1 = BinarySmiled(S0,K,ImpVol,tau,r,r,Skew);
BC2 = BinaryHeston(S0,K,tau,r,V0,kappa,theta,sigmaV,rho,lambda);
BC3 = _option_gradp(&BCC_K,K,0);
print $ftocv(100*(BC0~BC1~BC2~BC3),2,3);


         99.584          99.584          99.584          99.584
         97.457          97.318          97.314          97.314
         51.540          49.792          49.765          49.765
          6.574           5.993           5.969           5.969
          0.307           0.267           0.264           0.264
```

3.3 Computing risk-neutral probability density function

3.3.1 The breeden-Litzenberger method

We consider the following smile:

$$\begin{aligned}\Sigma(K) &= 0.25 + 5 \times 10^{-6} \times (K-90)^2 \times \mathbf{1}\{K \leq 90\} + \\ &\quad 5 \times 10^{-6} \times (K-90)^{1.5} \times \mathbf{1}\{K > 90\}\end{aligned}$$

Then we compute the cdf and pdf using the `Smile_to_Density` procedure and we compare them with BS cdf and pdf (see Figure 3.8). Remark that you may change the maturity and you will find very different results.

```
new;
library option,pgraph;
```

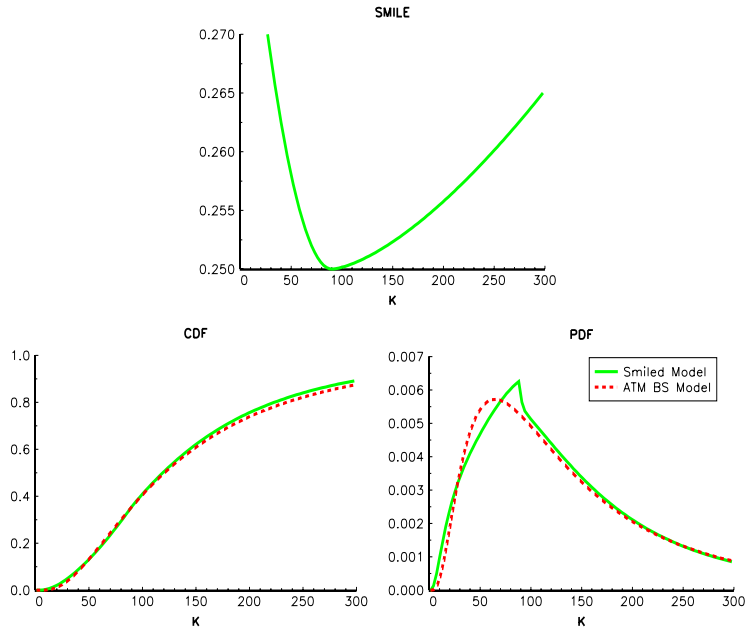
```
S0 = 100;
tau = 10;
r = 0.05;

Sk = 90;
fn smile(K) = 0.25 + (5e-6*(K-Sk)^2) .* (K .<= Sk) + (5e-6*((K-Sk)^2)^0.75) .* (K .> Sk);

K = seqa(1,3,100);

{Sigma,CDF,PDF} = Smile_to_Density(S0,K,&smile,tau,r);

{cdf_ln,pdf_ln} = EuropeanBS_Density(S0,K,0.25,tau,r);

graphset;
  begwind;
  makewind(9/2,6.855/2,9/4,6.855/2,1);
  makewind(9/2,6.855/2,0,0,1);
  makewind(9/2,6.855/2,9/2,0,1);

  _pltype = 6|3; _pframe = 0; _pnum = 2;
  _paxht = 0.20; _pnumht = 0.20; _ptitlht = 0.20;
  _pcolor = 10|12; _plwidth = 10; _pltype = 6|3;
  xtics(0,300,50,5);
  xlabel("\216K");

setwind(1);
  title("\216SMILE");
  ytics(0.25,0.27,0.005,2);
  xy(K,sigma);

setwind(2);
  title("\216CDF");
  ytics(0,1,0.2,2);
  xy(K,cdf~cdf_ln);

setwind(3);
  title("\216PDF");
  ytics(0,7e-3,1e-3,2);
  _plegstr = "Smiled Model\000ATM BS Model";
  _plegctl = {2 5 5 5};
  xy(K,pdf~pdf_ln);

  graphprt("-c=1 -cf=smile_to_density1.eps");

endwind;
```

3.3.2 Computing the density function in the Heston model using the Durrleman approximation

The next program shows how to compute the RND using the Durrleman model. We apply the method to the Heston model and we compare the obtained RND with the RND in the BS model using the ATM implied volatility (see Figure 3.9).

```
new;
library option,pgraph;

// #define VD_OLD_VERSION

_VD_order = 4;

S0 = 100;
tau = 1/2;
r = 0.0;

kappa = 1;
sigma = 0.20;
mu = sigma^2;
epsilon = 0.2;
rho = -0.75;
```

Figure 3.8: smile_to_density1.eps



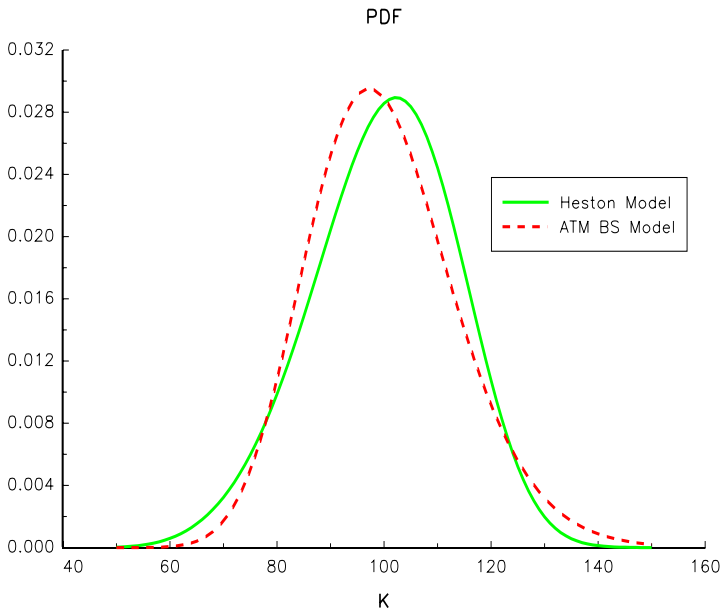

```
K = seqa(50,1,101);
{smile,cdf,pdf} = VD_HestonDensity(S0,K,sigma,kappa,mu,epsilon,rho,tau,r);
{cdf_ln,pdf_ln} = EuropeanBS_Density(S0,K,_VD_ATM_volatility,tau,r);

graphset;
  _pdate = "";
  _pltype = 6|3|1; _pframe = 0; _pnum = 2;
  _pcolor = 10|12|13; _plwidth = 10;
  title("\216PDF");
  xlabel("\216K");
  _plegstr = "Heston Model\000ATM BS Model";
  _plegctl = {2 5 6 4};
  graphprt("-c=1 -cf=smile_to_density3.eps");
  xy(K,pdf~pdf_ln);
```

3.3.3 The pdf in the SABR model

The following Gauss code is an illustration of the `sabr_pdf` procedure. We consider a smile curve with three points: $(K, \Sigma(K))$ takes respectively the values $(4\%, 11\%)$, $(5, 9\%)$ and $(8\%, 10\%)$. The maturity is one month. We calibrate the parameters $(\alpha, \beta, \nu, \rho)$ for two different cases: $\beta = 1$ and $\beta = 0$. In the third case, we measure the impact of a change in the ATM implied volatility. We proceed as follows:

1. First, we first estimate the parameters $(\alpha, \beta, \nu, \rho)$ using the `SABR_Calibrate` procedure.
2. Second, we compute the new value of α when the ATM implied volatility moves from 9% to 12% using the `SABR_Calibrate_ATM` procedure.
3. Finally, we compute the pdf using the `SABR_pdf` procedure.

Figure 3.9: smile_to_density3.eps

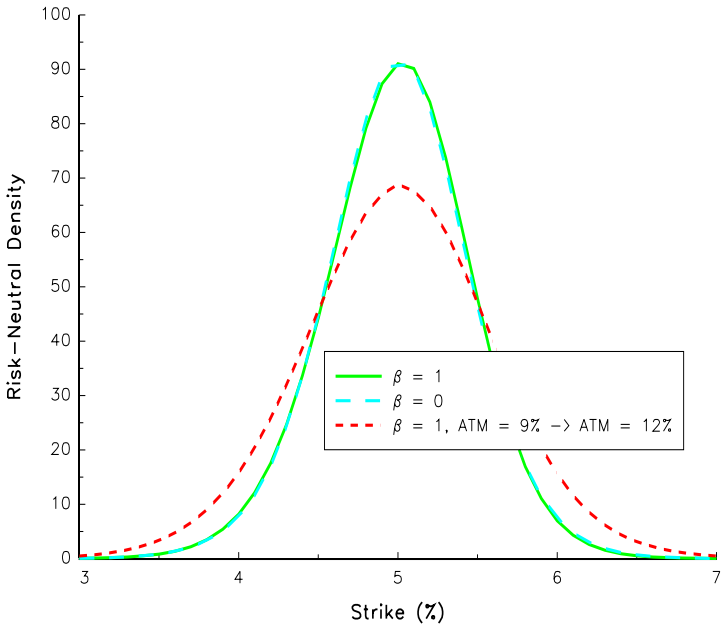


The results are reported in Figure 3.10.

```
new;

#ifdef USE_OPTMUM
  library option,pgraph,optmum;
#else
  library option,pgraph;
#endif

optionSet;

cls;

F0 = 0.05;
tau = 1;

let Strike = 0.04 0.05 0.08;
let ImpVol = 0.11 0.09 0.10;

K = seqa(0.02,0.001,81);

beta = 1;
{alpha,beta,nu,rho} = SABR_Calibrate(F0,Strike,ImpVol,beta,tau,0);
{ImpVol1,cdf1,pdf1} = sabr_pdf(F0,K,alpha,beta,nu,rho,tau);

beta = 0.0;
{alpha,beta,nu,rho} = SABR_Calibrate(F0,Strike,ImpVol,beta,tau,0);
{ImpVol2,cdf2,pdf2} = sabr_pdf(F0,K,alpha,beta,nu,rho,tau);

beta = 1;
{alpha,beta,nu,rho} = SABR_Calibrate(F0,Strike,ImpVol,beta,tau,0);
ATM = 0.12;
alpha = SABR_Calibrate_ATM(F0,ATM,beta,nu,rho,tau);
{ImpVol3,cdf3,pdf3} = sabr_pdf(F0,K,alpha,beta,nu,rho,tau);

graphset;
  fonts("simplex simgrma");
```

```
_pdate = ""; _pnum = 2; _pframe = 0;
_pltype = 6|1|3; _plwidth = 10;
xlabel("\216Strike (%)");
xtics(3,7,1,2);
ylabel("\216Risk-Neutral Density");
_plegstr = "\202b\201 = 1\000\202b\201 = 0\000\202b\201 = 1, ATM = 9% -> ATM = 12%";
_plegctl = {2 5 4 2};
graphprt("-c=1 -cf=sabr6.eps");
xy(100*K,pdf1~pdf2~pdf3);
```

Figure 3.10: sabr6.eps



3.4 Calibrating the parameters

3.4.1 The Durrleman model

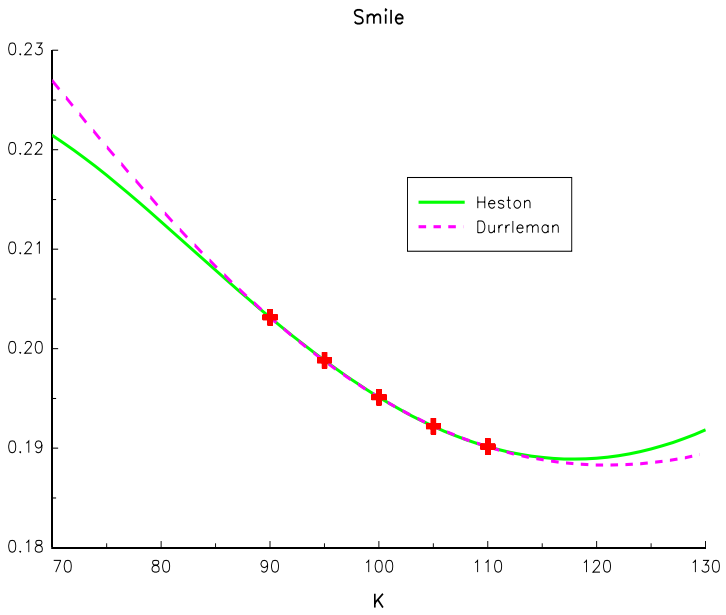
In this example, we calibrate the Durrlemean parameters from Heston option prices. We proceed as follows:

1. We consider 5 strikes $K = 90$, $K = 95$, $K = 100$, $K = 100$, $K = 105$ and $K = 110$.
2. For all these strikes, we compute the implied volatility for the Heston model.
3. Then, we estimate the parameters $(\sigma_t, a_t, b_t, d_t, e_t)$.
4. Finally, we compute the smile curve using the `VD_HestonSmile` procedure.

```
new;

#define USE_QNEWTON

#ifdef USE_CO
```

```
  library option,co,pgraph;
#endif

#ifdef USE_OPTMUM
  library option,optmum,pgraph;
#endif

#ifdef USE_QNEWTON
  library option,pgraph;
#endif

// #define VD_OLD_VERSION 1.0

cls;

_VD_order = 4;

S0 = 100;
tau = 1/2;
r = 0.0;

kappa = 1;
sigma = 0.20;
mu = sigma^2;
epsilon = 0.2;
rho = -0.3;

fn SmileProc(K) = VD_HestonSmile(S0,K,sigma,kappa,mu,epsilon,rho,tau,r);
K = 90|95|100|105|110;

MarketSmile = SmileProc(K);

#ifdef VD_OLD_VERSION

proc (6) = ModelParams(theta);
  theta[1:4] = sqrt(theta[1:4]^2);
  retp(VD_Heston_Parameters(theta[1],theta[2],theta[3],theta[4],theta[5]));
endp;

#else

proc (9) = ModelParams(theta);
  theta[1:4] = sqrt(theta[1:4]^2);
  retp(VD_Heston_Parameters(theta[1],theta[2],theta[3],theta[4],theta[5]));
endp;

#endif


_co_Algorithm = 1;
__output = 1;

sv = 0.22|1|0.1^2|0.2|-0.1;
{theta,fmin,grd,retcode} = VD_Calibraton(S0,K,MarketSmile,tau,&ModelParams,sv,0);
theta[1:4] = sqrt(theta[1:4]^2);

x = seqa(70,1,61);

Smile1 = VD_HestonSmile(S0,x,sigma,kappa,mu,epsilon,rho,tau,r);
Smile2 = VD_HestonSmile(S0,x,theta[1],theta[2],theta[3],theta[4],theta[5],tau,r);

graphset;
  _pdate = "";
  _pltype = 6|3|1; _pframe = 0; _pnum = 2; _plwidth = 10;
  _pcolor = 10|13;
  xlabel("\216K");
  ytics(0.18,0.23,0.01,2);
  title("\216Smile");
  e = ones(rows(K),1);
  _psym = K~MarketSmile~e.*(11~5~12~1~10);
  _plegstr = "Heston\000Durrleman";
  _plegctl = {2 5 5 4};
  graphprt("-c=1 -cf=smile_to_density4.eps");
  xy(x,smile1~smile2);
```

Figure 3.11: smile_to_density4.eps



3.4.2 The PDM model

3.4.2.1 The case of three volatilities

Using the implied volatilities for the strikes $K_1 = 90$, $K_2 = 100$ and $K_3 = 120$, we estimate the parameters of the PDM model. We report the corresponding smile curves in Figure 3.12.

```
new;
library option,pgraph;
OptionSet;

Strike = seqa(70,1,61);

let S0 = 100;
let sv = 0.50 0.1 0.1;

let K   =  90   100  120;
let Vol =  0.30 0.25 0.28;
{a,b,c,d,m} = PDM_Calibrate(S0,K,Vol,0,0,sv);
Smile = PDM_ImpVol(S0,Strike,a,b,c,d,m);

K1 = K; Vol1 = Vol; Smile1 = Smile;

let K   =  90   100  120;
let Vol =  0.30 0.25 0.25;
{a,b,c,d,m} = PDM_Calibrate(S0,K,Vol,0,0,sv);
Smile = PDM_ImpVol(S0,Strike,a,b,c,d,m);

K2 = K; Vol2 = Vol; Smile2 = Smile;

let K   =  90   100  130;
let Vol =  0.30 0.25 0.24;
{a,b,c,d,m} = PDM_Calibrate(S0,K,Vol,0,0,sv);
Smile = PDM_ImpVol(S0,Strike,a,b,c,d,m);
```

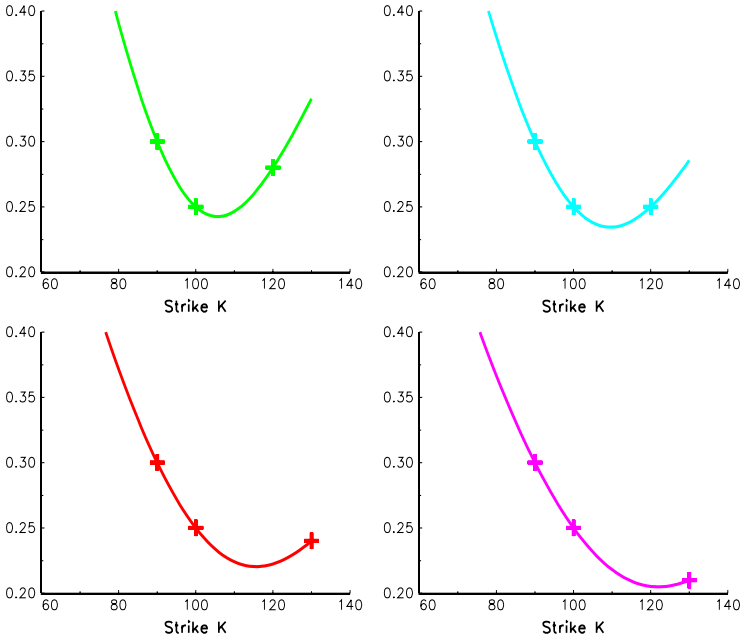
```
K3 = K; Vol3 = Vol; Smile3 = Smile;

let K   =  90   100  130;
let Vol =  0.30 0.25 0.21;
{a,b,c,d,m} = PDM_Calibrate(S0,K,Vol,0,0,sv);
Smile = PDM_ImpVol(S0,Strike,a,b,c,d,m);

K4 = K; Vol4 = Vol; Smile4 = Smile;

graphset;
  begwind;
  window(2,2,0);

    fonts("simplex simgrma");
    _pdate = ""; _pnum = 2; _pframe = 0;
    _paxht = 0.25; _pnumht = 0.22; _ptiltht = 0.25;
    _pltype = 6|1|3;
    _plwidth = 10;
    xlabel("\216Strike K");
    xtics(60,140,20,2);
    ytics(0.20,0.40,0.05,2);

  i = 1;
  setwind(i);
    _pcolor = 9 + i;
    _psym = K1~Vol1~ones(3,1).*(11~10~9+i~1~0);
    xy(Strike,Smile1);

  i = 2;
  setwind(i);
    _pcolor = 9 + i;
    _psym = K2~Vol2~ones(3,1).*(11~10~9+i~1~0);
    xy(Strike,Smile2);

  i = 3;
  setwind(i);
    _pcolor = 9 + i;
    _psym = K3~Vol3~ones(3,1).*(11~10~9+i~1~0);
    xy(Strike,Smile3);

  i = 4;
  setwind(i);
    _pcolor = 9 + i;
    _psym = K4~Vol4~ones(3,1).*(11~10~9+i~1~0);
    xy(Strike,Smile4);

    graphprt("-c=1 -cf=pdm3.eps");

  endwind;
```

3.4.2.2 Taking into account the skew

In the following example, we try to calibrate two implied volatilities Σ_1 and Σ_2 for the strikes K_1 and K_2 and one skew Sk_3 for the strike K_3. We remark that we couldn't calibrate the PDM model for the following parameters $(K_1, \Sigma_1) = (90, 30\%)$, $(K_2, \Sigma_2) = (100, 25\%)$, and $(K_3, \mathrm{Sk}_3) = (70, -0.1\%)$ (see the third graphic in Figure 3.13).

```
new;
library option,pgraph;
OptionSet;

Strike = seqa(70,1,61);

let S0 = 100;

Kmin = 70;
SkewMin = -0.01;
Kmax = 0;
SkewMax = 0.0;
if Kmax == 0;
```

Figure 3.12: pdm3.eps

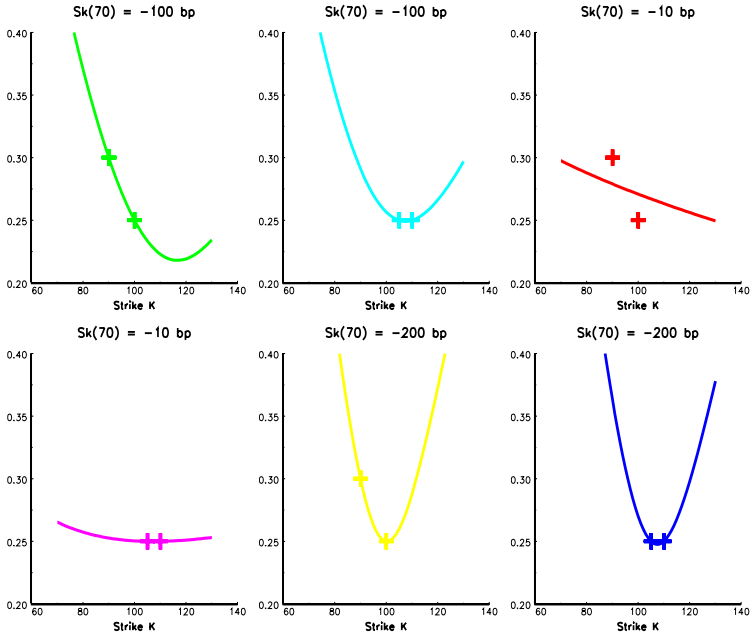



```
  Skew = Kmin~SkewMin;
else;
  Skew = Kmin~SkewMin |
         Kmax~SkewMax ;
endif;

let K   =  90   100;
let Vol =  0.30 0.25;
let sv = 0.5 0.1 0.1;
{a,b,c,d,m} = PDM_Calibrate(S0,K,Vol,Skew[.,1],Skew[.,2],sv);
Smile = PDM_ImpVol(S0,Strike,a,b,c,d,m);
K1 = K; Vol1 = Vol; Smile1 = Smile;

let K   =  105   110;
let Vol =  0.25 0.25;
let sv = 0.5 0.1 0.1;
{a,b,c,d,m} = PDM_Calibrate(S0,K,Vol,Skew[.,1],Skew[.,2],sv);
Smile = PDM_ImpVol(S0,Strike,a,b,c,d,m);
K2 = K; Vol2 = Vol; Smile2 = Smile;

Kmin = 70;
SkewMin = -0.001;
Kmax = 0;
SkewMax = 0.0;
if Kmax == 0;
  Skew = Kmin~SkewMin;
else;
  Skew = Kmin~SkewMin |
         Kmax~SkewMax ;
endif;

let K   =  90   100;
let Vol =  0.30 0.25;
let sv = 0.5 0.1 0.1;
{a,b,c,d,m} = PDM_Calibrate(S0,K,Vol,Skew[.,1],Skew[.,2],sv);
Smile = PDM_ImpVol(S0,Strike,a,b,c,d,m);
K3 = K; Vol3 = Vol; Smile3 = Smile;
```

```
let K   =  105   110;
let Vol =  0.25 0.25;
let sv = 0.5 0.1 0.1;
{a,b,c,d,m} = PDM_Calibrate(S0,K,Vol,Skew[.,1],Skew[.,2],sv);
Smile = PDM_ImpVol(S0,Strike,a,b,c,d,m);
K4 = K; Vol4 = Vol; Smile4 = Smile;

Kmin = 70;
SkewMin = -0.02;
Kmax = 0;
SkewMax = 0.0;
if Kmax == 0;
  Skew = Kmin~SkewMin;
else;
  Skew = Kmin~SkewMin |
         Kmax~SkewMax ;
endif;

let K   =  90   100;
let Vol =  0.30 0.25;
let sv = 0.5 0.1 0.1;
{a,b,c,d,m} = PDM_Calibrate(S0,K,Vol,Skew[.,1],Skew[.,2],sv);
Smile = PDM_ImpVol(S0,Strike,a,b,c,d,m);
K5 = K; Vol5 = Vol; Smile5 = Smile;

let K   =  105   110;
let Vol =  0.25 0.25;
let sv = 0.5 0.1 0.1;
{a,b,c,d,m} = PDM_Calibrate(S0,K,Vol,Skew[.,1],Skew[.,2],sv);
Smile = PDM_ImpVol(S0,Strike,a,b,c,d,m);
K6 = K; Vol6 = Vol; Smile6 = Smile;

graphset;
  begwind;
  window(2,3,0);

    fonts("simplex simgrma");
    _pdate = ""; _pnum = 2; _pframe = 0;
    _paxht = 0.25; _pnumht = 0.22; _ptitlht = 0.30;
    _pltype = 6|1|3;
    _plwidth = 10;
    xlabel("\216Strike K");
    xtics(60,140,20,2);
    ytics(0.20,0.40,0.05,2);

  i = 1;
  setwind(i);
    _pcolor = 9 + i;
    title("\216Sk(70) = -100 bp");
    _psym = K1~Vol1~ones(2,1).*(11~15~9+i~1~0);
    xy(Strike,Smile1);

  i = 2;
  setwind(i);
    _pcolor = 9 + i;
    _psym = K2~Vol2~ones(2,1).*(11~15~9+i~1~0);
    xy(Strike,Smile2);

  i = 3;
  setwind(i);
    title("\216Sk(70) = -10 bp");
    _pcolor = 9 + i;
    _psym = K3~Vol3~ones(2,1).*(11~15~9+i~1~0);
    xy(Strike,Smile3);

  i = 4;
  setwind(i);
    _pcolor = 9 + i;
    _psym = K4~Vol4~ones(2,1).*(11~15~9+i~1~0);
    xy(Strike,Smile4);

  i = 5;
  setwind(i);
```

```
  title("\216Sk(70) = -200 bp");
  _pcolor = 9 + i;
  _psym = K5~Vol5~ones(2,1).*(11~15~9+i~1~0);
  xy(Strike,Smile5);

i = 6;
setwind(i);
  _pcolor = 9;
  _psym = K6~Vol6~ones(2,1).*(11~15~9~1~0);
  xy(Strike,Smile6);

  graphprt("-c=1 -cf=pdm5.eps");

endwind;
```

Figure 3.13: pdm5.eps



3.5 The Dupire and the SABR models

3.5.1 Computing the local volatility surface

In the following example, we build an implied volatility surface using spline approximations. Then, we deduce the local volatility surface (see Figure 3.14).

```
new;
library option,pgraph;
OptionSet;

cls;

let Smile_K      = 80   85   90   95   100  105  110   115  120;
let Smile_ImpVol = 0.23 0.22 0.21 0.20 0.19 0.20 0.21 0.22 0.23;

S0 = 100;
```

```
r = 0.05;
b = 0.05;

declare matrix p_smoothing;

proc SigmaSurface(T,K);
 local r,c,d,smile,i;
 r = _max_(rows(T),rows(K));
 c = _max_(cols(T),cols(K));
 smile = zeros(r,c);
 i =  1;
 do until i > c;
   d = 0.15*T[i];
   Smile[.,i] = fspline(csspline(Smile_K,Smile_ImpVol + d,1,p_smoothing),K);
   i = i + 1;
 endo;
 retp(Smile);   /* Cubic Spline Smoothing */
endp;

p_smoothing = 0.25;
K = seqa(70,1,61);

T = seqa(0,0.01,101)';
ImpVol = SigmaSurface(t,K);
LocalVol = ImpVol_to_LocalVol(&SigmaSurface,0,0,0,T,K,S0,b);

pqgwin many;

graphset;
  _pdate = ""; _pnum = 2; _pframe = 0;
  xtics(70,130,10,2);
  ytics(0,1,0.20,2);
  xlabel("\216K");
  ylabel("\206t\201");
  title("\216Implied Volatiliry (in %)");
  surface(K',T',100*ImpVol');

graphset;
  _pdate = ""; _pnum = 2; _pframe = 0;
  xtics(70,130,10,2);
  ytics(0,1,0.20,2);
  xlabel("\216K");
  ylabel("\206t\201");
  title("\216Local Volatiliry (in %)");
  graphprt("-c=1 -cf=dupire8.eps");
  surface(K',T',100*LocalVol');
```

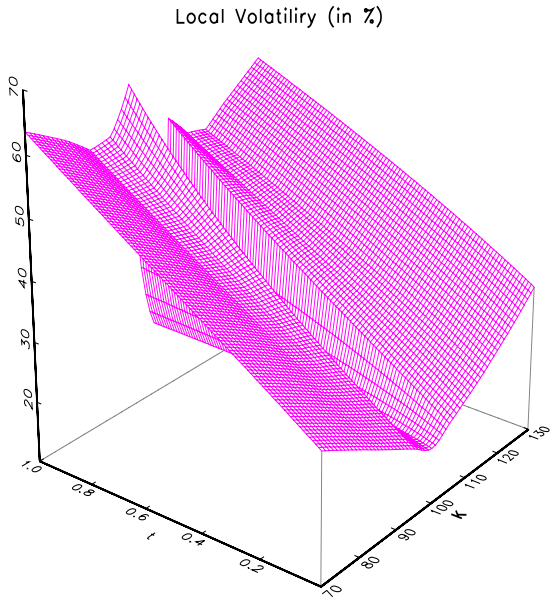
3.5.2 Pricing with the local volatility model

3.5.2.1 With the Backward PDE solver

In the next program, we price several barrier options ($K = 100$, $L = 90$ and $H = 115$) using the Dupire model with the Backward PDE solver.

```
new;
library option,pgraph;
OptionSet;

cls;

loadm databuf = Dupire-ImpVol;

Smile_K = vread(databuf,"K");
Smile_T = vread(databuf,"T");
Smile_ImpVol = vread(databuf,"ImpVol");

S0 = 100;
b = 0.05;
r = 0.05;
```

Figure 3.14: dupire8.eps




```
FileName = "Dupire11";
call LocalVol_Init(Smile_K,Smile_ImpVol,Smile_T,0,200,201,1,201,S0,b,r,FileName);

Tau = 1.0;
K = 100;
L = 90;
H = 115;

let string OptionType[10,1] = "call" "put" "doc" "dop" "uoc" "uop" "koc" "kop" "bc" "bp";
nIters = rows(OptionType);

__output = 0;

i = 1;
do until i > nIters;

  {tmp_S0,tmp_Tau,C} = LocalVol_Backward_PDE_Solve(OptionType[i],K,L,H,tau,0.5);
  str = upper(OptionType[i]) $+ ftos(C," = %lf",5,4);
  print /flush str;

  i = i + 1;
endo;

CALL = 8.8422
PUT = 3.9652
DOC = 8.0141
DOP = 0.2669
UOC = 1.0009
UOP = 3.8056
KOC = 0.6746
KOP = 0.2047
BC = 0.5644
BP = 0.3597
```

3.5.2.2 With the MC solver

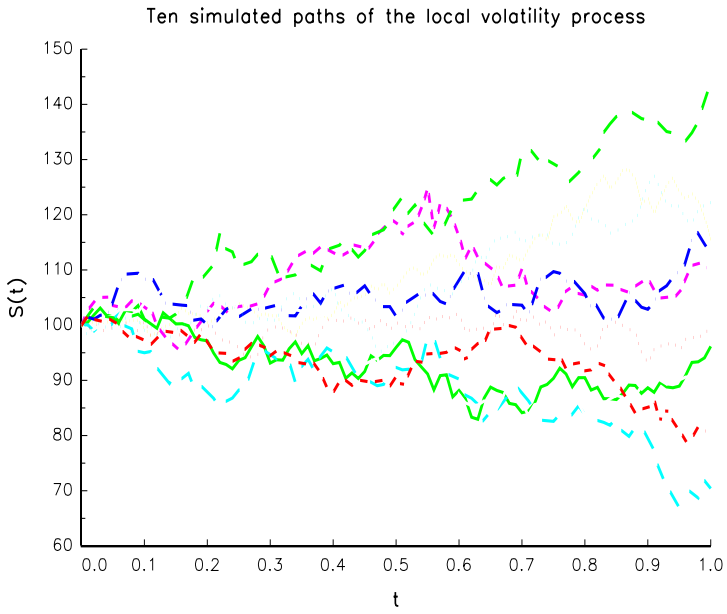
In what follows, we compare the option prices obtained by the PDE solver with the prices obtained by monte-carlo simulations.

```
new;
library option,pgraph;
OptionSet;

cls;

loadm databuf = Dupire-ImpVol;

Smile_K = vread(databuf,"K");
Smile_T = vread(databuf,"T");
Smile_ImpVol = vread(databuf,"ImpVol");

S0 = 100;
b = 0.05;
r = 0.05;

FileName = "Dupire11";
call LocalVol_Init(Smile_K,Smile_ImpVol,Smile_T,0,200,201,1,201,S0,b,r,FileName);

t = seqa(0,0.01,101);
Ns = 10;
x = LocalVol_simulate_SDE(t,Ns);

graphset;
  _pdate = ""; _pnum = 2; _pframe = 0;
  _plwidth = 10;
  xlabel("\216t");
  ylabel("\216S(t)");
  xtics(0,1,0.10,1);
  title("\216Ten simulated paths of the local volatility process");
  graphprt("-c=1 -cf=dupire13.eps");
  xy(t,x);

Ns = 10000;  // Use a larger value to check the convergence
Tau = 1.0;
K = 100;
L = 90;
H = 115;

t = seqa(0,Tau/100,101); nT = rows(t);

let string OptionType[10,1] = "call" "put" "doc" "dop" "uoc" "uop" "koc" "kop" "bc" "bp";
nIters = rows(OptionType);

__output = 0;

i = 1;
do until i > nIters;

  print /flush "-----------------------------------------------------------";

  {tmp_S0,tmp_Tau,C} = LocalVol_Backward_PDE_Solve(OptionType[i],K,L,H,tau,0.5);
  str = upper(OptionType[i]) $+ "(PDE)" $+ ftos(C," = %lf",5,4);
  print /flush str;

  S = LocalVol_simulate_SDE(t,Ns);
  e = ones(Ns,1);
  j = 1;
  do until j > nT;
    e = e .and _LocalVol_restrictions(0,S[j,.]');
    j = j + 1;
  endo;

  ST = S[Nt,.]';
  payout = e .* _LocalVol_payoff(ST,K);

  C = exp(-r*tau)*meanc(payout);
  str = upper(OptionType[i]) $+ "(MC) " $+ ftos(C," = %lf",5,4);
```

```
  print /flush str;

  i = i + 1;
endo;
```

```
------------------------------------------------------------
CALL(PDE) = 8.8422
CALL(MC)  = 8.9718
------------------------------------------------------------
PUT(PDE) = 3.9652
PUT(MC)  = 4.0795
------------------------------------------------------------
DOC(PDE) = 8.0141
DOC(MC)  = 8.1508
------------------------------------------------------------
DOP(PDE) = 0.2669
DOP(MC)  = 0.3090
------------------------------------------------------------
UOC(PDE) = 1.0009
UOC(MC)  = 1.1590
------------------------------------------------------------
UOP(PDE) = 3.8056
UOP(MC)  = 3.8080
------------------------------------------------------------
KOC(PDE) = 0.6746
KOC(MC)  = 0.9126
------------------------------------------------------------
KOP(PDE) = 0.2047
KOP(MC)  = 0.2462
------------------------------------------------------------
BC(PDE) = 0.5644
BC(MC)  = 0.5881
------------------------------------------------------------
BP(PDE) = 0.3597
BP(MC)  = 0.3657
```

Figure 3.15: dupire13.eps



3.5.3 Understanding the SABR dynamics

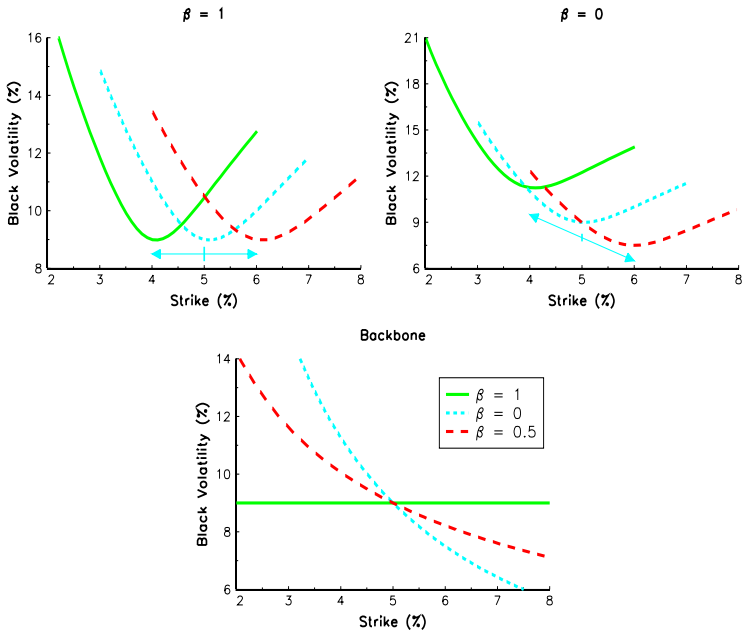
3.5.3.1 With respect to the β parameter

In Figure 3.16, we show the dynamics of the smile with respect to the β parameter in the SABR model. The underlying idea is to calibrate the sabr smile when the forward F_0 is equal to 5%. Then, we compute the smile when F_0 moves to 4% or 6%. Finally, we build the backbone, that is the curve $\Sigma = \Sigma(K = F_0)$ which indicates the value of the implied volatility when the strike is equal to the forward.

```
new;

#ifdef USE_OPTMUM
  library option,pgraph,optmum;
#else
  library option,pgraph;
#endif

optionSet;

cls;

F0 = 0.05;
tau = 1;

N = 41;
K = seqa(0.02,0.001,N) ~seqa(0.03,0.001,N) ~seqa(0.04,0.001,N);
K = K + 0.0001;
K_BB = seqa(0.02,0.001,N+30);

let Strike = 0.04 0.05 0.06;
let ImpVol = 0.11 0.09 0.10;

{alpha1,beta1,nu1,rho1} = SABR_Calibrate(F0,Strike,ImpVol,1,tau,0);
{alpha2,beta2,nu2,rho2} = SABR_Calibrate(F0,Strike,ImpVol,0,tau,0);
{alpha3,beta3,nu3,rho3} = SABR_Calibrate(F0,Strike,ImpVol,0.5,tau,0);

let F0 = 0.04 0.05 0.06;

ImpVol1 = SABR_ImpVol(F0[1],K[.,1],alpha1,beta1,nu1,rho1,tau) ~
          SABR_ImpVol(F0[2],K[.,2],alpha1,beta1,nu1,rho1,tau) ~
          SABR_ImpVol(F0[3],K[.,3],alpha1,beta1,nu1,rho1,tau) ;

BackBone1 = SABR_ImpVol(K_BB,K_BB,alpha1,beta1,nu1,rho1,tau);

ImpVol2 = SABR_ImpVol(F0[1],K[.,1],alpha2,beta2,nu2,rho2,tau) ~
          SABR_ImpVol(F0[2],K[.,2],alpha2,beta2,nu2,rho2,tau) ~
          SABR_ImpVol(F0[3],K[.,3],alpha2,beta2,nu2,rho2,tau) ;

BackBone2 = SABR_ImpVol(K_BB,K_BB,alpha2,beta2,nu2,rho2,tau);
BackBone3 = SABR_ImpVol(K_BB,K_BB,alpha3,beta3,nu3,rho3,tau);

graphset;
  begwind;
  makewind(9/2,6.855/2,0,6.855/2,1);
  makewind(9/2,6.855/2,9/2,6.855/2,1);
  makewind(9/2,6.855/2,9/4,0,1);

  fonts("simplex simgrma");
  _pdate = ""; _pnum = 2; _pframe = 0;
  _pltype = 6|3|1; _plwidth = 10;
  _paxht = 0.22; _pnumht = 0.20; _ptitlht = 0.22;
  ylabel("\216Black Volatility (%)");
  xlabel("\216Strike (%)");
  xtics(2,8,1,2);

  setwind(1);
    title("\216\202b\201 = 1");
    _parrow = 5~8.5~4~8.5~2~0.20~01~11~1~6~0 |
              5~8.5~6~8.5~2~0.20~01~11~1~6~0;
    _pline = 1~6~5~8.25~5~8.75~1~11~6;
```

```
  ytics(8,16,2,2);
  xy(100*K,100*ImpVol1);

setwind(2);
  title("\216\202b\201 = 0");
  _parrow = 5~8.0~4~9.5~2~0.20~01~11~1~6~0 |
            5~8.0~6~6.5~2~0.20~01~11~1~6~0;
  _pline = 1~6~5~7.75~5~8.25~1~11~6;
  ytics(6,21,3,2);
  xy(100*K,100*ImpVol2);

setwind(3);
  title("\216Backbone");
  ytics(6,14,2,2);
  _parrow = 0; _pline = 0;
  _plegstr = "\202b\201 = 1\000\202b\201 = 0\000\202b\201 = 0.5";
  _plegctl = {2 6 6 4};
  xy(100*K_BB,100*(BackBone1~BackBone2~BackBone3));

  graphprt("-c=1 -cf=sabr1.eps");

endwind;
```

Figure 3.16: sabr1.eps



3.5.3.2 With respect to the ATM implied volatility

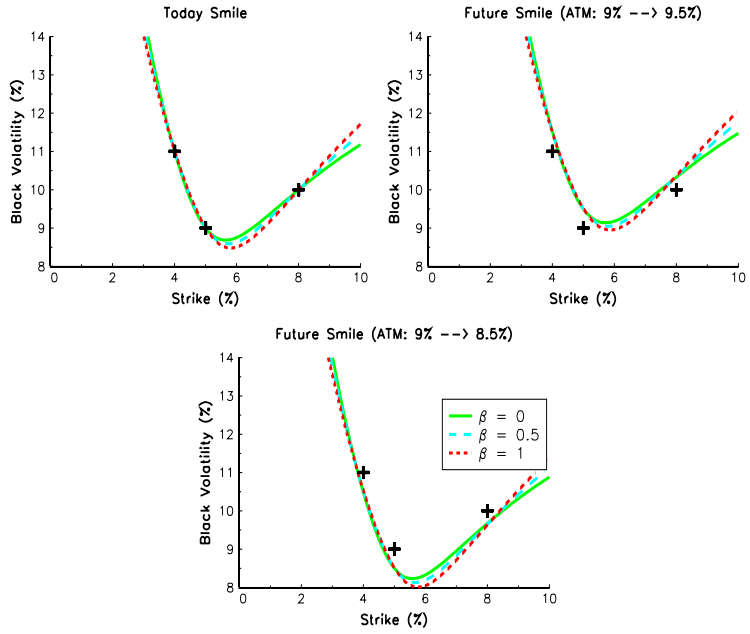
In Figure 3.17, we show how the actual smile moves with respect to the ATM volatility.

```
new;

#ifdef USE_OPTMUM
  library option,pgraph,optmum;
#else
  library option,pgraph;
#endif
```

```
optionSet;

cls;

F0 = 0.05;
tau = 1;

let Strike = 0.04 0.05 0.08;
let ImpVol = 0.11 0.09 0.10;

ATM = ImpVol[2];

beta = 0;
{alpha1,beta1,nu1,rho1} = SABR_Calibrate(F0,Strike,ImpVol,beta,tau,0);

beta = 0.5;
{alpha2,beta2,nu2,rho2} = SABR_Calibrate(F0,Strike,ImpVol,beta,tau,0);

beta = 1;
{alpha3,beta3,nu3,rho3} = SABR_Calibrate(F0,Strike,ImpVol,beta,tau,0);

N = 81;
K = seqa(0.02,0.001,N);
Smile1 = SABR_ImpVol_ATM(F0,K,ATM,beta1,nu1,rho1,tau);
Smile2 = SABR_ImpVol_ATM(F0,K,ATM,beta2,nu2,rho2,tau);
Smile3 = SABR_ImpVol_ATM(F0,K,ATM,beta3,nu3,rho3,tau);
Smile = Smile1~Smile2~Smile3;

// F0 = 0.051;

ATM = 9.5/100;
Smile1 = SABR_ImpVol_ATM(F0,K,ATM,beta1,nu1,rho1,tau);
Smile2 = SABR_ImpVol_ATM(F0,K,ATM,beta2,nu2,rho2,tau);
Smile3 = SABR_ImpVol_ATM(F0,K,ATM,beta3,nu3,rho3,tau);
Smile_Future_1 = Smile1~Smile2~Smile3;

ATM = 8.5/100;
Smile1 = SABR_ImpVol_ATM(F0,K,ATM,beta1,nu1,rho1,tau);
Smile2 = SABR_ImpVol_ATM(F0,K,ATM,beta2,nu2,rho2,tau);
Smile3 = SABR_ImpVol_ATM(F0,K,ATM,beta3,nu3,rho3,tau);
Smile_Future_2 = Smile1~Smile2~Smile3;

graphset;
  begwind;
  makewind(9/2,6.855/2,0,6.855/2,1);
  makewind(9/2,6.855/2,9/2,6.855/2,1);
  makewind(9/2,6.855/2,9/4,0,1);

  fonts("simplex simgrma");
  _pdate = ""; _pnum = 2; _pframe = 0;
  _paxht = 0.22; _pnumht = 0.20; _ptitlht = 0.22;
  _pltype = 6|1|3;
  _plwidth = 10;
  ylabel("\216Black Volatility (%)");
  xlabel("\216Strike (%)");
  xtics(0,10,2,2);
  ytics(8,14,1,2);
  _psym = 100*Strike~100*ImpVol~ones(rows(ImpVol),1).*(11~8~0~1~0);
  _plegstr = "\202b\201 = 0\000\202b\201 = 0.5\000\202b\201 = 1";

  setwind(1);
    title("\216Today Smile");
    xy(100*K,100*Smile);

  setwind(2);
    title("\216Future Smile (ATM: 9% --> 9.5%)");
    xy(100*K,100*Smile_Future_1);

  setwind(3);
    title("\216Future Smile (ATM: 9% --> 8.5%)");
    _plegctl = {2 6 6 3.5};
    xy(100*K,100*Smile_Future_2);
```

```
graphprt("-c=1 -cf=sabr4.eps");

endwind;
```

Figure 3.17: sabr4.eps



3.6 Solving PDE

3.6.1 The influence of the $\theta-$scheme

We consider the Vasicek model. We remind that the dynamics of the instantaneous interest rate is the following:

$$\mathrm{d}r(t) = a(b - r(t))\,\mathrm{d}t + \sigma \mathrm{d}W(t)$$

In the case of a Bond (with terminal value $B(T,r) = 1$), the pricing equation becomes:

$$\begin{cases} \frac{1}{2}\sigma^2 \partial_r^2 B(t,r) + (a(b-r) - \lambda\sigma)\,\partial_r B(t,r) + \partial_t B(t,r) - rB(t,r) = 0 \\ B(T,r) = 1 \end{cases}$$

let us define $\tau = T - t$. We have:

$$\begin{cases} \frac{1}{2}\sigma^2 \partial_r^2 B(\tau,r) + (a(b-r) - \lambda\sigma)\,\partial_r B(\tau,r) = \partial_\tau B(\tau,r) + rB(\tau,r) \\ B(0,r) = 1 \end{cases}$$

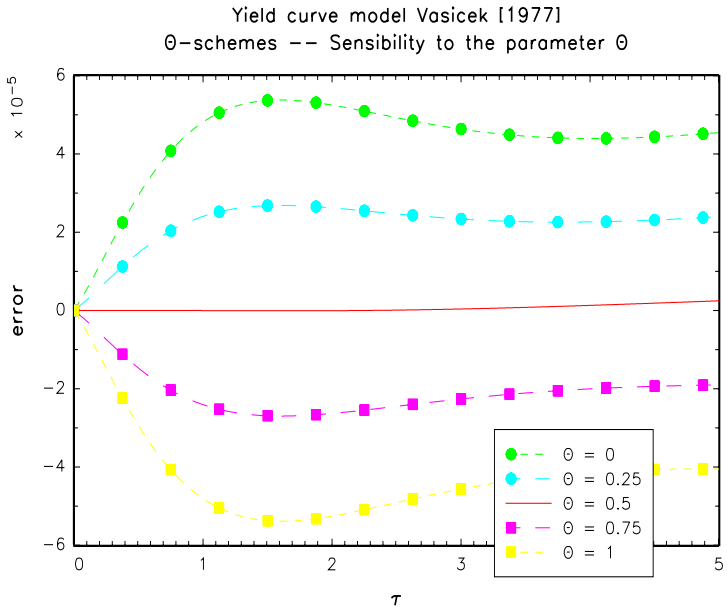
With this form, we may directly use the `PDE` and `PDE_solve` procedures. We consider several $\theta-$schemes in order to measure the impact of θ on the numerical errors (see Figure 3.18).

```
new;
library option,pgraph;
```

```
dlibrary option_tridiag; // Don't miss to download the tridiag solver

cls;

a = 0.95; b = 0.10; sigma = 0.2; lambda = 0.05;
bprime =  b - lambda*sigma/a;

proc aProc(t,x);
  retp( 0.5*(sigma^2) * ones(rows(x),1) );
endp;


proc bProc(t,x);
  retp( a*(bprime-x) );
endp;

proc cProc(t,x);
  retp( x );
endp;

proc dProc(t,x);
  retp( zeros(rows(x),1) );
endp;

proc tminBound(t,x);
  retp( ones(rows(x),1) );
endp;

call PDE(&aProc,&bProc,&cProc,&dProc,0,
         &tminBound,0,0,0,0);

tmin = 0;
tmax = 5;
Nt = 1001;
xmin = -1;
xmax = 1;
Nx = 101;

_pde_PrintIters = 0;
_pde_SaveLastIter = 0;

U = zeros(Nt,5);
theta = seqa(0,0.25,5);

i = 1;
do until i > 5;

  call PDE_solve(tmin,tmax,Nt,xmin,xmax,Nx,"pde",theta[i]);

  t = PDE_readFile("pde","t");
  x = PDE_readFile("pde","x");
  U[.,i] = PDE_readFile("pde","x"|x[56]);

  i = i + 1;
endo;

tau = t;
{CouponZero,TauxZero} = Vasicek(x[56],a,b,sigma,lambda,tau);
diff = CouponZero - U;

call DeleteFile("pde.dat");

graphset;
  _pdate = ""; _pnum = 2; _pltype = 3|1|6|1|3; _pstype = 8|8|1|9|9; _plctrl = 75|75|0|75|75;
  fonts("simplex simgrma");
  title("\216Yield curve model Vasicek @[1977@]\L"\
        "\216\202Q\201-schemes -- Sensibility to the parameter \202Q\201");
  xlabel("\216\202t\201");
  xtics(0,5,1,10);
  ylabel("\216error");
  _plegstr="\202Q\201 = 0\000\202Q\201 = 0.25\000\202Q\201 = 0.5\000"\
           "\202Q\201 = 0.75\000\202Q\201 = 1";
  _plegctl={ 2 5 6 0.5};
```

```
_pysci = 2;
graphprt("-c=1 -cf=pde2.eps");
xy(t,diff);
```

Figure 3.18: pde2.eps



3.6.2 Pricing an American option

In the case of an European option with the BS model, we have:

$$\begin{cases} \frac{1}{2}\sigma^2 \partial_S^2 C(\tau, S) + rS\,\partial_S C(\tau, S) = \partial_\tau C(\tau, S) + rC(\tau, S) \\ C(0, S) = (S - K)^+ \end{cases}$$

The PDE of the American option is the same, but it contains a variational inequalities (see Crepey [2007] for further details). The idea is that option may be exercised at any time. So at each iterations of the PDE, the solution is the maximum between the PDE solution and the payoff:

$$u_t \leftarrow \max\left(u_t, (S - K)^+\right)$$

with u_t be the solution of the PDE at time t. That's why we have to define the `eProc` procedure.

```
new;
library option,pgraph;

dlibrary option_tridiag; // Don't miss to download the tridiag solver

cls;

declare matrix b,r,sigma,tau;

K = 100;
```

```
S0 = 80 | 90 | 100 | 110 | 120;
bVector = -0.04 | -0.04 | -0.04 | -0.04 ;
rVector =  0.08 | 0.12 | 0.08 | 0.08 ;
sigmaVector = 0.20 | 0.20 | 0.40 | 0.20 ;
tauVector = 0.25 | 0.25 | 0.25 | 0.50 ;
NtVector = floor(1825 * tauVector);
let string OptionType[2,1] = "call" "put";

proc aProc(t,x);
  retp( 0.5*(sigma^2) .* (x^2) );
endp;

proc bProc(t,x);
  retp( b .* x );
endp;

proc cProc(t,x);
  retp( r .* ones(rows(x),1) );
endp;

proc dProc(t,x);
  retp( zeros(rows(x),1) );
endp;

/*
**> variational inequality
*/

proc eProc(t,x,U);
  local PayOff,cnd;

  if _Option_Type $== "call";
    PayOff = x - K;
    PayOff = PayOff .* (PayOff .> 0);
  else;
    PayOff = K - x;
    PayOff = PayOff .* (PayOff .> 0);
  endif;

  cnd = PayOff .> U;

  retp( cnd .* PayOff + (1-cnd) .* U );
endp;

proc tminBound(t,x);
  local PayOff;

  if _Option_Type $== "call";
    PayOff = x - K;
  else;
    PayOff = K - x;
  endif;

  PayOff = PayOff .* (PayOff .> 0);
  retp( PayOff );
endp;

proc xminBound(t,x);
  retp( zeros(rows(x),1) );
endp;

proc DxminBound(t,x);
  retp( -ones(rows(x),1) );
endp;

proc xmaxBound(t,x);
  retp( zeros(rows(x),1) );
endp;

proc DxmaxBound(t,x);
  retp( ones(rows(x),1) );
endp;
```

```
_PDE_SaveLastIter = 1;
__output = 0;

i = 1;
do until i > 2;

  _Option_Type = OptionType[i];

  j = 1;
  do until j > 4;

    xmin = 50; xmax = 150; Nx = 501;
    tmin = 0; tmax = tauVector[j]; Nt = NtVector[j];

    b = bVector[j];
    r = rVector[j];
    sigma = sigmaVector[j];
    tau = tauVector[j];

    if _Option_Type $== "call";
      call PDE(&aProc,&bProc,&cProc,&dProc,&eProc,
               &tminBound,&xminBound,0,0,&DxmaxBound);
    else;
      call PDE(&aProc,&bProc,&cProc,&dProc,&eProc,
               &tminBound,0,&xmaxBound,&DxminBound,0);
    endif;

    call PDE_solve(tmin,tmax,Nt,xmin,xmax,Nx,"pde",0.5);

    tau = PDE_readFile("pde","t");
    x = PDE_readFile("pde","x");
    U = PDE_readFile("pde","t"|tau);

    _fcmptol = 1e-12;  // Modify this value if you want that the procedure FindIndex works correctly

    indx = FindIndex(x,S0);
    x = x[indx];
    primeFD = U[indx];

    primeEU = EuropeanBS(x,K,sigma,tau,b,r);
    primeBAW = AmericanBS(x,K,sigma,tau,b,r);

    call PrintTable(x,K,sigma,tau,b,r,primeEU,primeBAW,primeFD,j);

    j = j + 1;

  endo;

  i = i + 1;
endo;

call DeleteFile("pde.dat");

proc (0) = PrintTable(x,K,sigma,tau,b,r,primeEU,primeBAW,primeFD,cn);
  local Nobs,e,mask,fmt,omat;

  Nobs = rows(x);
  e = ones(Nobs,1);
  K = K .* e;
  sigma = sigma .* e;
  tau = tau .* e;
  b = b .* e;
  r = r .* e;

  mask = 1;
  let fmt[9,3]=   "*.*lf" 7 1 "*.*lf" 7 1  "*.*lf" 7 2
                  "*.*lf" 7 2 "*.*lf" 7 2  "*.*lf" 8 2
                  "*.*lf" 11 3  "*.*lf" 11 3 "*.*lf" 11 3;

  omat = x~K~sigma~tau~b~r~primeEU~primeBAW~primeFD;

  if cn == 1;
    print /flush "    ";
```

```
    if _option_type $== "call";
      print "    S0     K     sigma   tau    b      r         Call       Call       Call     ";
    else;
      print "    S0     K     sigma   tau    b      r         Put        Put        Put      ";
    endif;
      print "                                                 European   American   American ";
      print "                                                            BAW        FD       ";
      print "==========================================================================================";
  endif;

  call printfm(omat,mask,fmt);
  retp;
endp;
```

S0	K	sigma	tau	b	r	Call European	Call American BAW	Call American FD
80.0	100.0	0.20	0.25	-0.04	0.08	0.029	0.032	0.029
90.0	100.0	0.20	0.25	-0.04	0.08	0.570	0.590	0.580
100.0	100.0	0.20	0.25	-0.04	0.08	3.421	3.525	3.524
110.0	100.0	0.20	0.25	-0.04	0.08	9.847	10.315	10.356
120.0	100.0	0.20	0.25	-0.04	0.08	18.618	20.000	20.000
80.0	100.0	0.20	0.25	-0.04	0.12	0.029	0.032	0.029
90.0	100.0	0.20	0.25	-0.04	0.12	0.564	0.587	0.575
100.0	100.0	0.20	0.25	-0.04	0.12	3.387	3.506	3.502
110.0	100.0	0.20	0.25	-0.04	0.12	9.749	10.288	10.326
120.0	100.0	0.20	0.25	-0.04	0.12	18.433	20.000	20.000
80.0	100.0	0.40	0.25	-0.04	0.08	1.046	1.067	1.055
90.0	100.0	0.40	0.25	-0.04	0.08	3.232	3.284	3.270
100.0	100.0	0.40	0.25	-0.04	0.08	7.291	7.411	7.407
110.0	100.0	0.40	0.25	-0.04	0.08	13.248	13.502	13.526
120.0	100.0	0.40	0.25	-0.04	0.08	20.728	21.233	21.292
80.0	100.0	0.20	0.50	-0.04	0.08	0.210	0.229	0.215
90.0	100.0	0.20	0.50	-0.04	0.08	1.312	1.387	1.360
100.0	100.0	0.20	0.50	-0.04	0.08	4.465	4.724	4.709
110.0	100.0	0.20	0.50	-0.04	0.08	10.163	10.955	10.998
120.0	100.0	0.20	0.50	-0.04	0.08	17.851	20.000	20.000

S0	K	sigma	tau	b	r	Put European	Put American BAW	Put American FD
80.0	100.0	0.20	0.25	-0.04	0.08	20.413	20.419	20.414
90.0	100.0	0.20	0.25	-0.04	0.08	11.250	11.251	11.250
100.0	100.0	0.20	0.25	-0.04	0.08	4.396	4.397	4.396
110.0	100.0	0.20	0.25	-0.04	0.08	1.118	1.118	1.118
120.0	100.0	0.20	0.25	-0.04	0.08	0.184	0.184	0.184
80.0	100.0	0.20	0.25	-0.04	0.12	20.210	20.248	20.230
90.0	100.0	0.20	0.25	-0.04	0.12	11.138	11.146	11.139
100.0	100.0	0.20	0.25	-0.04	0.12	4.353	4.355	4.353
110.0	100.0	0.20	0.25	-0.04	0.12	1.107	1.107	1.107
120.0	100.0	0.20	0.25	-0.04	0.12	0.183	0.183	0.183
80.0	100.0	0.40	0.25	-0.04	0.08	21.430	21.463	21.444
90.0	100.0	0.40	0.25	-0.04	0.08	13.912	13.927	13.915
100.0	100.0	0.40	0.25	-0.04	0.08	8.266	8.274	8.267
110.0	100.0	0.40	0.25	-0.04	0.08	4.519	4.523	4.518
120.0	100.0	0.40	0.25	-0.04	0.08	2.294	2.297	2.289
80.0	100.0	0.20	0.50	-0.04	0.08	20.947	20.982	20.957
90.0	100.0	0.20	0.50	-0.04	0.08	12.632	12.645	12.633
100.0	100.0	0.20	0.50	-0.04	0.08	6.367	6.372	6.367
110.0	100.0	0.20	0.50	-0.04	0.08	2.648	2.650	2.648
120.0	100.0	0.20	0.50	-0.04	0.08	0.918	0.919	0.918

3.6.3 Comparing forward and backward PDE

A simple example to show the difference between forward and backward PDE. The payoff function is an European call option with a 6M maturity. In Figure 3.17, we also report the numerical errors $u(S_0, K) - BS(S_0, K)$ between the PDE solution and the BS formula. For the backward PDE, K

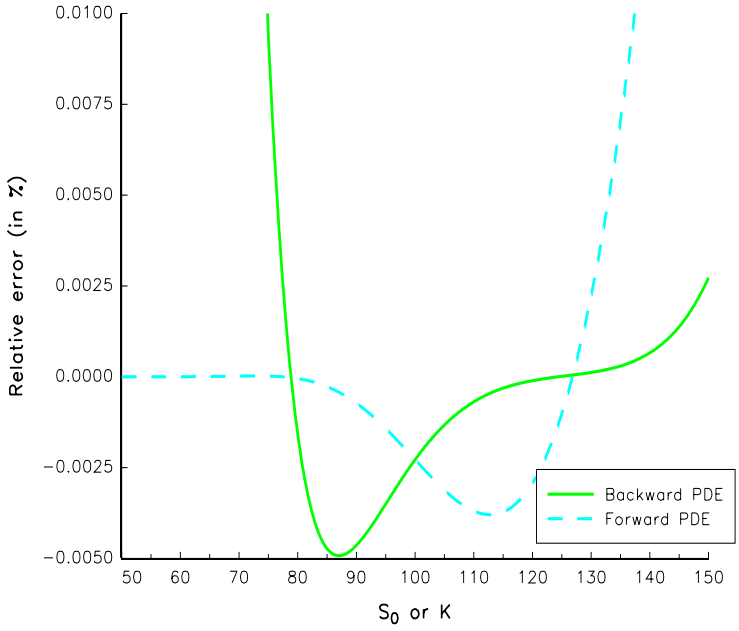
is fixed and is equal to 100 whereas S_0 is in the range $[50; 150]$. For the forward PDE, this is the contrary: S_0 is fixed and is equal to 100 whereas K is in the range $[50; 150]$.

```
new;
library option,pgraph;
dlibrary option_tridiag; // Don't miss to download the tridiag solver or use the procedure OptionSet

cls;

S0 = 100;
K = 100;
sigma = 0.20;
tau = 0.5;
r = 0.05;
b = 0.02;

proc SigmaSurface(tau,K);
  retp( sigma + 0.*tau + 0.* K );
endp;

proc Bwrd_aProc(t,x);
  retp( 0.5*(sigma^2) .* (x^2) );
endp;

proc Fwrd_aProc(t,x);
  retp( 0.5*(SigmaSurface(t,x)^2) .* (x^2) );
endp;

proc Bwrd_bProc(t,x);
  retp( b .* x );
endp;

proc Fwrd_bProc(t,x);
  retp( - b .* x );
endp;

proc Bwrd_cProc(t,x);
  retp( r .* ones(rows(x),1) );
endp;

proc Frwd_cProc(t,x);
  retp( (r-b) .* ones(rows(x),1) );
endp;

proc dProc(t,x);
  retp( zeros(rows(x),1) );
endp;

proc Bwrd_tminBound(t,x);
  local PayOff;

  PayOff = EuropeanPayOff(x,K);

  retp( PayOff );
endp;

proc Fwrd_tminBound(t,x);
  local PayOff;

  PayOff = EuropeanPayOff(S0,x);

  retp( PayOff );
endp;

proc Bwrd_xminBound(t,x);
  retp( zeros(rows(x),1) );
endp;

proc Frwd_xmaxBound(t,x);
  retp( zeros(rows(x),1) );
endp;

proc Bwrd_DxmaxBound(t,x);
  retp( ones(rows(x),1) );
```

```
endp;

proc Fwrd_DxminBound(t,x);
  retp( -ones(rows(x),1) );
endp;

// Backward PDE

_PDE_SaveLastIter = 1;
_PDE_PrintIters = 50;
__output = 1;

xmin = 0; xmax = 200; Nx = 1001;
tmin = 0; tmax = tau; Nt = 2500;

call PDE(&Bwrd_aProc,&Bwrd_bProc,&Bwrd_cProc,&dProc,0,&Bwrd_tminBound,&Bwrd_xminBound,0,0,&Bwrd_DxmaxBound);

fileName = "dupire-backward";
call PDE_solve(tmin,tmax,Nt,xmin,xmax,Nx,fileName,0.5);

// Forward PDE

_PDE_SaveLastIter = 1;
_PDE_PrintIters = 50;
__output = 1;

xmin = 0; xmax = 200; Nx = 1001;
tmin = 0; tmax = tau; Nt = 2500;

call PDE(&Fwrd_aProc,&Fwrd_bProc,&Frwd_cProc,&dProc,0,&Fwrd_tminBound,0,&Frwd_xmaxBound,&Fwrd_DxminBound,0);

fileName = "dupire-forward";
call PDE_solve(tmin,tmax,Nt,xmin,xmax,Nx,fileName,0.5);

// Comparison

fileName = "dupire-backward";

tau = PDE_readFile(fileName,"t");
x = PDE_readFile(fileName,"x");
U = PDE_readFile(fileName,"t"|tau);
e = x .> 10 .and x .< 190;
U1 = selif(U,e);
x1 = selif(x,e);
C1 = EuropeanBS(x1,K,sigma,tau,b,r);

fileName = "dupire-forward";

tau = PDE_readFile(fileName,"t");
x = PDE_readFile(fileName,"x");
U = PDE_readFile(fileName,"t"|tau);
e = x .> 10 .and x .< 160;
U2 = selif(U,e);
x2 = selif(x,e);
C2 = EuropeanBS(S0,x2,sigma,tau,b,r);

reldiff1 = 100 * (U1-C1)./C1;
reldiff2 = 100 * (U2-C2)./C2;

N = maxc(rows(x1)|rows(x2));
x = miss(zeros(N,2),0);
reldiff = miss(zeros(N,2),0);
x[1:rows(x1),1] = x1;
x[1:rows(x2),2] = x2;
reldiff[1:rows(x1),1] = reldiff1;
reldiff[1:rows(x2),2] = reldiff2;

graphset;
  _pdate = ""; _pnum = 2; _pframe = 0; _plwidth = 10;
  xlabel("\216S]0[ or K");
  ylabel("\216Relative error (in %)");
  ytics(-0.005,0.010,0.0025,0);
  xtics(50,150,10,2);
  _plegstr = "Backward PDE\000Forward PDE";
```

```
_plegctl = {2 5 6.5 1};
graphprt("-c=1 -cf=dupire1.eps");
xy(x,reldiff);
```

Figure 3.19: dupire1.eps



3.7 Monte-Carlo methods

3.7.1 Studying the MC convergence in the case of the BS model

Here is an example of computing the price of an European option using Monte Carlo methods. In Figure 3.20, we compare the MC price which is given by the following formula:

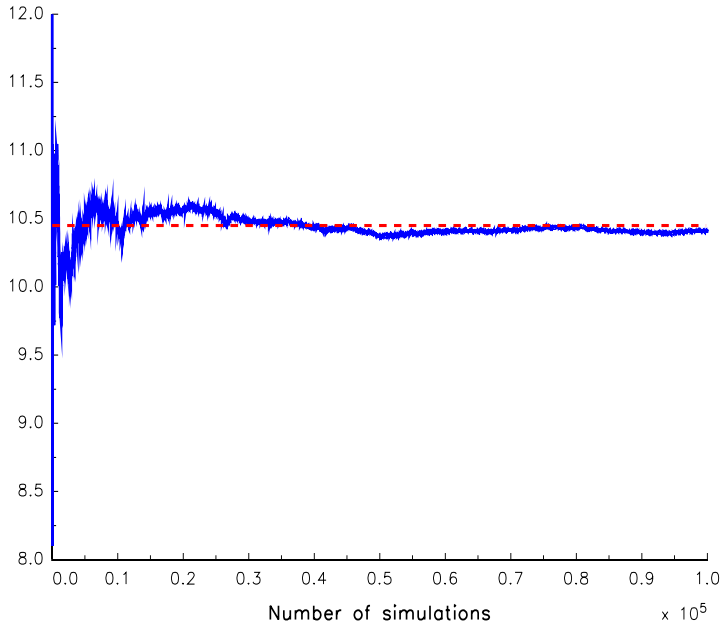
$$\frac{1}{n_s}\sum_{i=1}^{n_s}\hat{C}_i$$

where $\hat{C}_i$ is the discounted payoff for the i^{th} simulation with the price obtained with the BS formula. We remark that the MC price converges to the BS price when n_s tends to infinity.

```
new;
library option,pgraph;

S0 = 100;
K = 100;
sigma = 0.20;
r = 0.05;
b = r;
tau = 1.0;
fn call_payoff(S,K) = (S - K) .* (S .>= K);
BS = EuropeanBS(S0,K,sigma,tau,b,r);
```

```
nS = 100000;
S = simulate_GBM(S0,r,sigma,tau,nS);

ST = S';

Cs = exp(-r*tau)*cumsumc(call_payoff(St,K))./seqa(1,1,Ns);

step = 10;
indx = seqa(1,step,nS/step);
Cs = Cs[indx];

graphset;
  _pdate = ""; _pnum = 2; _pframe = 0;
  _pltype = 6; _plwidth = 10; _pcolor = 9;
  ytics(8,12,0.5,2);
  _pline = 1~3~0~BS~nS~BS~1~12~10;
  xlabel("\216Number of simulations");
  graphprt("-c=1 -cf=mc1.eps");
  xy(indx,Cs);
```

Figure 3.20: mc1.eps



3.7.2 Computing the price of Look-back options

In what follows, we show how to price the look-back option:

$$G = \max\left(0, S(T) - \min_{0 \le t \le T} S(t)\right)$$

In the first part of the example, we estimate the price with a time step corresponding to a trading day (if we suppose that there are 60 trading days in 3 months). Then, we illustrate the fact that the price is under-estimated if we use large time step because the $\min_{0 \le t \le T} S(t)$ is over-estimated (see Figure 3.21).

```
new;
library option,pgraph;

S0 = 100;
K = 100;
sigma = 0.20;
r = 0.05;
b = r;
tau = 3/12;  // 3 Months

fn lookback_payoff(ST,Smin) = (ST - Smin) .* (ST .>= Smin);

t = seqa(0,tau/60,61);
nS = 100000;
S = simulate_GBM(S0,b,sigma,t,nS);
nT = rows(t);
ST = S[nT,.]';
Smin = minc(S);

LBC = exp(-r*tau)*meanc(lookback_payoff(ST,Smin));

print ftos(LBC,"Look-back call = %lf",6,5);

nS = 100000;
nt = seqa(5,1,96);
LBC = zeros(rows(nt),1);

i = 1;
do until i > rows(nt);
  t = seqa(0,tau/(nt[i]-1),nt[i]);
  S = simulate_GBM(S0,b,sigma,t,nS);
  ST = S[nt[i],.]';
  Smin = minc(S);
  LBC[i] = exp(-r*tau)*meanc(lookback_payoff(ST,Smin));
  print /flush i;
  i = i + 1;
endo;

graphset;
  _pdate = ""; _pnum = 2; _pframe = 0;
  _pltype = 3; _plwidth = 10; _pcolor = 12;
  xlabel("\216Number of discretization points");
  xtics(0,100,10,2);
  graphprt("-c=1 -cf=mc2.eps");
  xy(nt,lbc);
```

3.7.3 An example of average of Best-Of in time across assets

Let us consider the case of three assets. We assume that the volatilities are respectively 15%, 25% and 20% and the correlation matrix is:

$$\rho = \begin{pmatrix} 1 & & \\ 0.5 & 1 & \\ 0.25 & 0.15 & 1 \end{pmatrix}$$

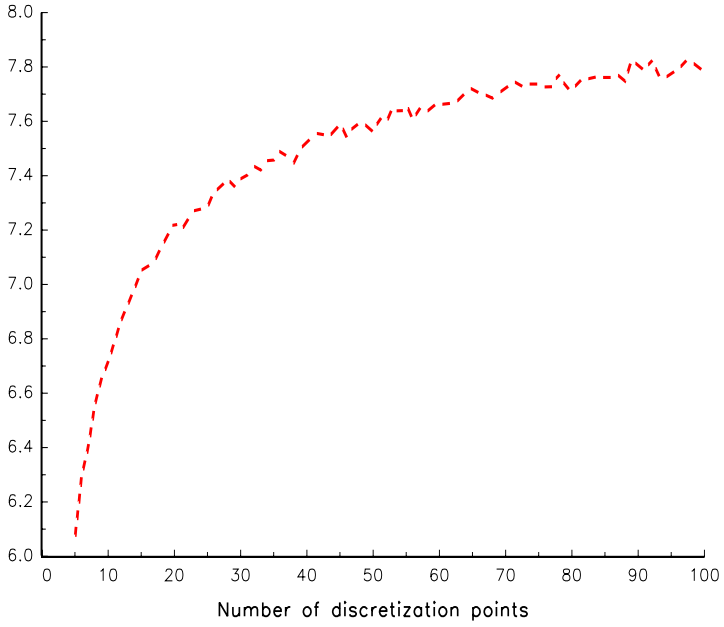
The payoff of the option is:

$$\left(\frac{1}{3} \sum_{j=1}^{3} \max_{t_i} \left(\frac{S_{t_i,j}}{S_{t_{i-1},j}} \right) - K \right)_+$$

with t_i the fixing dates of the option and $S_{t_i,j}$ the value of the asset j for the fixing date t_i.

```
new;
library option,pgraph;

cls;
```

Figure 3.21: mc2.eps




```
let x0 = 100 100 100;
let mu = 0.05 0.06 0.07;
let sigma = 0.15 0.25 0.20;                  // Implied Volatility

// Correlation Matrix between Asset returns

let rho = {1,
           0.5, 1,
           0.25, 0.15, 1};
rho = xpnd(rho);

// First, we have to correct the drift
r = 0.05;                                    // risk-free interest rate
mu = ones(rows(mu),1) * r;

// Caracteristics of the option
t = seqa(0,0.25,9);                          // 8 Fixing Dates : Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8
tau = t[9];                                  // Maturity of the option = 2 Years
Notional = 1000;                             // Notional of the option
K = 0.10;                                    // Strike of the option = 10%
fn call_payoff(S,K) = (S - K) .* (S .>= K);  // call option
fn put_payoff(S,K) = (K - S) .* (S .<= K);   // put option

Ns = 1000000;  // Number of simulations
x = array_simulate_mGBM(x0,mu,sigma,rho,t,Ns);

Nx = rows(x0);
Nt = rows(t);
BestOf = zeros(Ns,Nx);

i = 1;
do until i > Nx;                             // for each asset
  data = mGBM_getAsset(x,i);
  tmp = data[.,2:Nt] ./ data[.,1:Nt-1] - 1;  // we compute the quaterly returns for the 2 years period
  BestOf[.,i] = maxc(tmp');                  // we compute the maximum of the quaterly returns
  i = i + 1;
endo;
```

```
// BestOf is a Ns x Nx matrix where BestOf(i,j) indicates the Best-Of the quaterly returns of the asset j for the simulation i

MeanBestOf = meanc(BestOf');                // Average of Best-Of in Time Across Assets

C = exp(-r*tau) * meanc( call_payoff(MeanBestOf,K) );
P = exp(-r*tau) * meanc( put_payoff(MeanBestOf,K) );

print ftos(C,"Best-Of Call Option Price = %lf",3,2);
print ftos(P,"Best-Of Put Option Price = %lf",3,2);


Best-Of Call Option Price = 0.06
Best-Of Put Option Price = 0.00
```

3.7.4 Example with options on multi-assets

You will find different examples in the **[gauss root]\option\examples** directory:

- call_put_Basket_Spread_option.prg
- call_put_BestOf_call_asian_option.prg
- call_put_BestOf_Call_Put_option.prg
- call_put_Himalaya_option.prg
- call_put_Rainbow_option.prg
- call_put_WorstOf_Call_Put_option.prg

3.7.5 Simulating binomial trees

In the following program, we simulate binomial trees and compute the price of an European option. In Figure 3.22, we estimate the desnity of MC estimators. We remark that using antithetic variables permits to reduce the variance of the estimator.

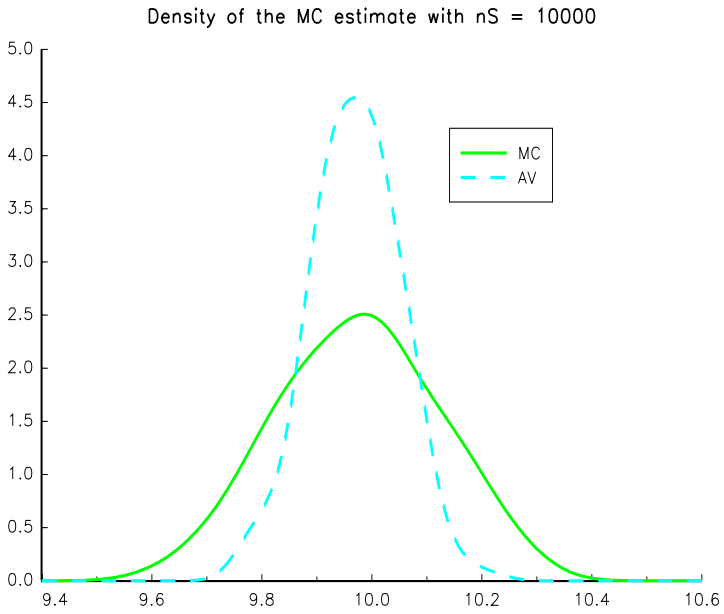
```
new;
library option,pgraph;

S0 = 100;
K = 100;
sigma = 0.20;
r = 0.05;
b = r;
tau = 1.0;

n = 4;

{CRR,Tree} = EuropeanCRR(S0,K,sigma,tau,b,r,n);
print /flush ftos(CRR,"CRR option price = %lf",5,5);

fn call_payoff(S,K) = (S - K) .* (S .>= K);

Ns = 10000;
{S1,S2} = Simulate_CRR_av(S0,sigma,tau,b,r,n,nS);

ST1 = S1[n,.]';
ST2 = S2[n,.]';

C1 = exp(-r*tau)*meanc(call_payoff(ST1,K));
C2 = exp(-r*tau)*meanc(call_payoff(ST2,K));
Cav = 0.5 * (C1 + C2);
print /flush ftos(C1,"CRR option price (MC) = %lf",5,5);
print /flush ftos(C2,"CRR option price (MC) = %lf",5,5);
print /flush ftos(Cav,"CRR option price (MC Antithetic Variables) = %lf",5,5);
```

```
CRR option price = 9.97052
CRR option price (MC) = 10.05913
CRR option price (MC) = 9.84901
CRR option price (MC Antithetic Variables) = 9.95407
```

In the last example, we compare the density of the MC estimator with and without antithetic variables.

```
new;
library option,pgraph;

S0 = 100;
K = 100;
sigma = 0.20;
r = 0.05;
b = r;
tau = 1.0;

n = 4;

{CRR,Tree} = EuropeanCRR(S0,K,sigma,tau,b,r,n);

fn call_payoff(S,K) = (S - K) .* (S .>= K);
Ns = 10000;

Nr = 500;
C1 = zeros(Nr,1);
C2 = zeros(Nr,1);
i = 1;
do until i > Nr;
  {S1,S2} = Simulate_CRR_av(S0,sigma,tau,b,r,n,nS);
  ST1 = S1[n,.]';
  ST2 = S2[n,.]';
  C1[i] = exp(-r*tau)*meanc(call_payoff(ST1,K));
  C2[i] = exp(-r*tau)*meanc(call_payoff(ST2,K));
  i = i + 1;
endo;

Cav = 0.5 * (C1 + C2);

_Kernel[1:3] = 9.4|10.6|256;
{x,d1,F,retcode} = Kernel(C1);
{x,d2,F,retcode} = Kernel(Cav);

graphset;
  _pdate = ""; _pnum = 2; _pframe = 0;
  _pltype = 6|1; _plwidth = 10;
  title("\216Density of the MC estimate with nS = 10000");
  xtics(9.4,10.4,0.2,2);
  _plegstr = "MC\000AV";
  _plegctl = {2 5 5.5 4.5};
  graphprt("-c=1 -cf=mc4.eps");
  xy(x,d1~d2);
```

Figure 3.22: mc4.eps



Bibliography

[1] Barone-Adesi, G. and R.E. Whaley [1987], Efficient analytic approximation of American option values, *Journal of Finance*, **42**, 301-320

[2] Bates, D. S. [1991], The crash of '87: was it expected? The evidence from options markets, Journal of Finance, 46, 1009-1044

[3] Black, F. [1976], The pricing of asset contracts, *Journal of Financial Economics*, **3**, 167-179

[4] Black, F. and M. Scholes [1973], The pricing of options and corporate liabilities, *Journal of Political Economy*, **81**, 637-659

[5] Breeden, D. and R. Litzenberger [1978], State contingent prices implicit in option prices, *Journal of Business*, **51**, 621-651

[6] Chang, C.W., J.S.K. Chang and K-G. Lim [1998], Information-time option pricing: theory and empirical evidence, *Journal of Financial Economics*, **48**, 211-242

[7] Crépey, S. [2007], Computational Finance, *Lecture notes*, University of Evry

[8] Dhouibi, A. [2005], Smile Approximation, Rapport de Stage, *Groupe de recherche Opérationnelle*

[9] Dupire, B. [1994], Pricing with a smile, *Risk Magazine*, **7-1**, 18-20

[10] Durrlerman, V. [2004], From Implied to Spot Volatilities, *Princeton University*, Ph.D. Thesis

[11] Hagan, P., D. Kumar, A.S. Lesniewski and D.E. Woodward [2002], Managing smile risk, *Wilmott Magazine*, September, 84-108

[12] Heston, S.L. [1993], A closed-form solution for options with stochastic volatility with applications to bond and currency options, *Review of Financial Studies*, **6**, 327-343

[13] Hull, J. and A. White [1987], The pricing of options on assets with stochastic volatilities, *Journal of Finance*, **42**, 281-300

[14] Merton, R.C. [1976], Option pricing when underlying stock returns are discontinuous, *Journal of Financial Economics*, **3**, 125-143

[15] Roncalli [1995], Introduction à la Programmation sous Gauss – Methodes Numériques en Finance et Econométrie, *Global Design*, Paris

[16] Rubinstein, M. and E. Reiner [1991], Breaking Down the Barriers, *Risk Magazine*, **4-8**, 28-35