Precision and Speed for Option Pricing

University of Paris VI, Laboratoire J.-L. Lions

Olivier Pironneau[1]

with Y. Achdou, R. Cont (UP-VII), N. Lantos, (Natixis), G. Loeper (BNP)
LJLL-University of Paris VI

October 15, 2009

Outline

Aim: Be fast and accurate. Be adaptive.

- See if Partial Differential Equations do better than Monte-Carlo.
- Stay away from modeling
- mesh adaptivity
- Parallel computing
- Stream Computing

1.
   - Scientific Computing in Finance
   - A Mix of Monte-Carlo and PDE
   - A Reduced Basis
   - Back to PDE, especially American Option

Computational finance

Includes

- Market data analysis
- Portfolio Optimization
- Risk assessment
- **Option pricing**

The Black-Scholes Model for an asset with tendency μ and volatility σ

$$dS_t = S_t(\mu dt + \sigma dW_t), \qquad S_0 \text{ known}$$

- $\mu = r(t)$ the interest rate under the risk-neutral probability law
- (W_t) : a standard Brownian motion.
  - A European (resp. American) option on S is a contract giving the right to buy (call) or sell (put) this asset at (resp. before) time T (**maturity**) at price K (**strike**).
  - Its value at t is the expected profit at T discounted to t:

$$C_t = e^{-r(T-t)}\mathbf{E}(S_T - K)^+, \quad P_t = e^{-r(T-t)}\mathbf{E}(K - S_T)^+$$

Pricing Options: 3 numerical methods

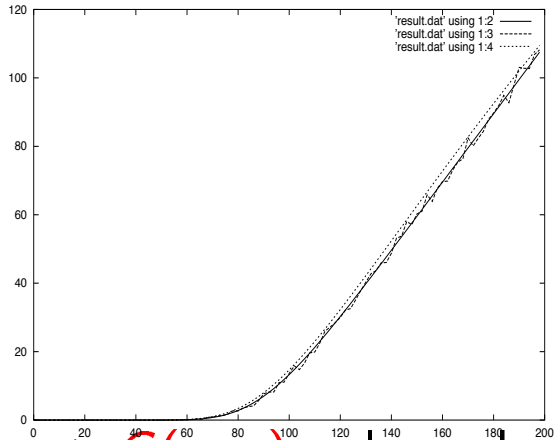
- Tree Methods and Monte-Carlo Methods:

$$\frac{S_{k+1}^n - S_k^n}{\delta t} = S_k^n(\mu\delta t + \sigma\sqrt{\delta t}\mathcal{N}_{01}),\ S_0^n = S_0.$$

$$C_t = e^{-r(T-t)}\frac{1}{N}\sum_1^N \left((S_T^n - K)^+\right)\ \mathcal{N}_{01} \approx f(\texttt{rand()})$$

- Ito Calculus : $C_t = C(S_t, t)$ solution of

$$\partial_t C + \frac{\sigma^2 x^2}{2}\partial_{xx}^2 C + rx\partial_x C - rC = 0$$
$$C(T, x) = (x - K)^+ \quad C(0, t) = 0$$
$$C(x, t) \sim x - Ke^{-r(T-t)} \text{ when } x \to \infty$$



The time reversed put $P(x, T-t) = Ke^{-r(T-t)} - x + C(x, t)$ solves the forward heat equation with $P(+\infty, t) = 0$ and diffusion $= \frac{1}{2}x^2\sigma^2$.

A posteriori estimates (Y.Achdou)

Proposition

Let $u_{h,\delta t}$ be the solution of a Black-Scholes problem by FEM of order 1:

$$[[u - u_{h,\delta t}]](t_n) \leq c(u_0)\delta t + \frac{\mu}{\sigma_{\min}^2}\left(\sum_{m=1}^{n} \eta_m{}^2 + \frac{\delta t_m}{\sigma_{\min}^2} g(\rho_{\delta t}) \prod_{i=1}^{m-1}(1 - 2\lambda\delta t_i) \sum_{\omega\in\mathcal{T}_{mh}} \eta_{m,\omega}{}^2\right)^{\frac{1}{2}}$$

where L, μ are the time-continuity constants of σ^2, $r, x\sigma\partial_x\sigma$ in L^∞,
$c(u_0) = (\|u_0\|^2 + \delta t\|\nabla u_0\|^2)^{1/2}$, $\quad g(\rho_{\delta t}) = (1+\rho_{\delta t})^2 \max(2, 1+\rho_{\delta t})$

$$\eta_m^2 = \delta t_m e^{-2\lambda t_{m-1}} \frac{\sigma_{\min}^2}{2} |u_h^m - u_h^{m-1}|_V^2,$$

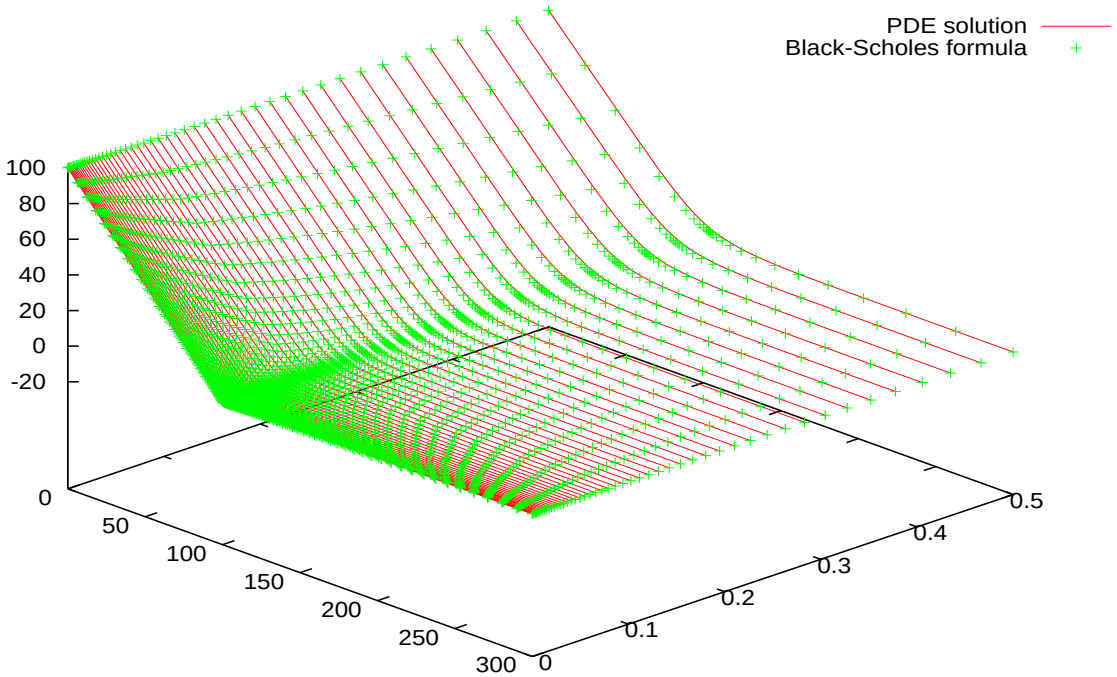
$$\eta_{m,\omega} = \frac{h_\omega}{x_{\max}(\omega)} \left\| \frac{u_h^m - u_h^{m-1}}{\delta t_m} - rx\frac{\partial u_h^m}{\partial x} + ru_h^m \right\|_{L^2(\omega)}$$

Best Numerical Method

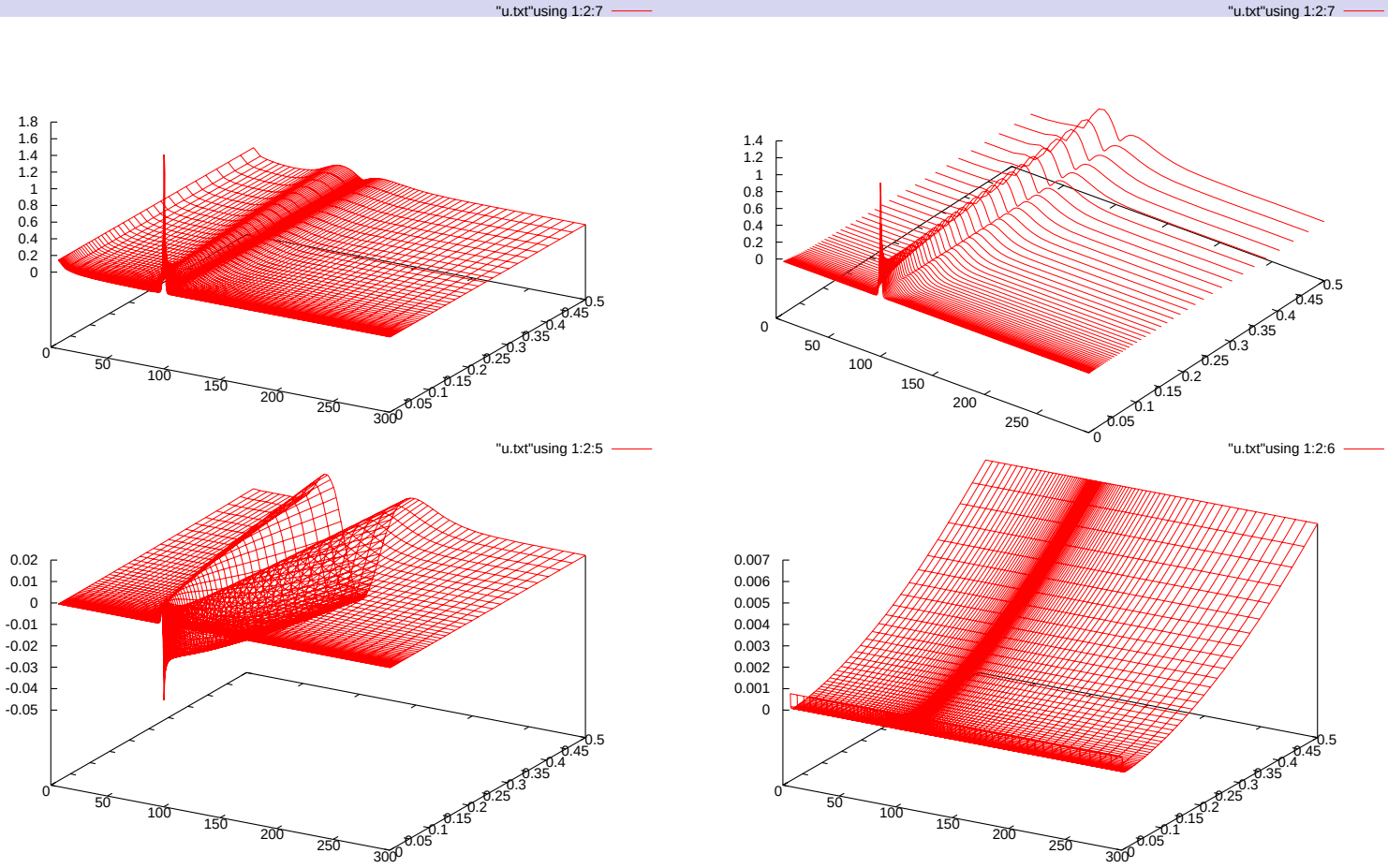
- Implicit in time, centered in space (upwind usually not necessary):

$$\frac{P^{n+1} - P^n}{\delta t} - \frac{x^2\sigma^2}{2}\partial^2_{xx}P^{n+\frac{1}{2}} + rP^{n+\frac{1}{2}} - xr\partial_x P^{n+\frac{1}{2}} = 0$$

  - FEM-P^1 + LU factorization
- mesh adaptivity + a posteriori estimates
- Banks need 0.1% precision in split seconds ...within Excel



Numerical Example



TOP: Indicator 1 with a fixed number of nodes (left) and varying (right)
BOTTOM: Actual error (left). Second indicator (right).

Source Code in C++

/Users/pironneau/Desktop/07fall/tex/NatIxis/optionNixis.cpp Page: 1
Sunday 18 November 2007 / 16:28

```
#include <math.h>
#include <iostream>
#include <fstream>
using namespace std;

typedef double ddouble;

class Option{
   public:
          const int nT, nX;   // nb of time steps and mesh points
          const double  r,S,K,T;// rate spot price strike and maturi
          ddouble **sigma;// volatility(function of x and t)
          ddouble *u, *uold, *am, *bm, *cm; // working arrays
          double dx, dt;

          ddouble phi(const double strike, double s1); // payoff
          void factLU();
          void solveLU(ddouble* z);
          void calc();

          Option(const int nT1, const int nX1,
            const double r1, const double S1, const double K1, const

          ~Option() { delete [] am; delete [] bm; delete []cm; delet

};

 Option::Option(const int nT1, const int nX1, const double r1, const
    const double K1, const double T1) : nT(nT1), nX(nX1), r(r1), S(S
 {
   const double xmax=3*S;
   dx = xmax/(nX-1);
   dt = T/(nT-1);
   u = new ddouble[nX];
   sigma = new ddouble*[nT];
   for(int i = 0; i < nT; i++) sigma[i] = new ddouble[nX];
   am       = new ddouble[nX];
   bm       = new ddouble[nX];
   cm       = new ddouble[nX];
   uold     = new ddouble[nX];
 }


ddouble Option::phi(const double strike, double x) {
   return x<strike?strike-x:0;
 }
```

/Users/pironneau/Desktop/07fall/tex/NatIxis/optionNixis.cpp
Sunday 18 November 2007 / 16:28

```
         cm[i] /= bm[i];
     }
 }

 void Option::solveLU(ddouble *z){
     z[1] /= bm[1];
     for(int i=2;i<nX-1;i++)
         z[i] = (z[i] - am[i]*z[i-1])/bm[i];
     for(int i=nX-2;i>0;i--)
         z[i] -= cm[i]*z[i+1];
 }

 void Option::calc( )
 {
   for(int i=0;i<nX;i++) u[i] = phi(K,i*dx);
   int j1=0;

   for(int j=0;j<nT;j++) { // time loop
      for(int i=1;i<nX-1;i++)  // rhs of PDE
        uold[i] = u[i] + dt*r*i*(u[i+1]-u[i-1])/2;

   u[nX-1]=0;  u[0] = K*exp(-r*(j+0.5)*dt);  // B.C. of PDE
   uold[1] += u[0]*sigma[j][1]*sigma[j][1]*dt/2;

   for(int i=1;i<nX-1;i++) { // build matrix
         ddouble aux=i*sigma[j][i]*i*sigma[j][i]*dt/2;
         bm[i] = (1+ r*dt + 2*aux);
         am[i] = -aux;
         cm[i] = -aux;
      }
   factLU();
   for(int i=1;i<nX-1;i++) u[i]=uold[i];
   solveLU(u);
    }
 }

 int main(){
 Option p(100,150,0.03, 100,110,4);// nT, nX, r, S, K, T
   for(int j=0;j<p.nT;j++) // time loop
      for(int i=0;i<p.nX;i++)  p.sigma[j][i]=0.3;
 p.calc();
 for(int i=0;i<p.nX;i++) cout<<p.u[i]<<endl;
 return 0;
 }
```

Monte-Carlo: Goods and bads

- Near to the modelisation
- Gives upper and lower bounds on the error
- Converges in $1/\sqrt{N}$ $\Rightarrow$ Quasi-Monte Carlo ?
- Can compute several derivatives with $O(\sqrt{N})$ operations
- Easy to parallelize.
- Complexity grows in $O(d)$ with the dimension d.
- Greeks by Malliavin calculus.
- Calibration is very hard.

PDEs: Goods and bads

- Nearer to the analytical formulas
- Upper and lower error bounds with *a posteriori* estimates
- Converges in $O(N^{-p})$ with order of approximation p
- Compute several derivatives in $O(N \log N)$ operations (Dupire).
- Can be parallelized.
- Cursed by dimension d in $O(N^d)$ except with *sparse grid*.
- Greeks are very easy
- Calibration is reasonably easy

Outline

Aim: Be fast and accurate. Be adaptive.

- See if Partial Differential Equations do better than Monte-Carlo.
- Stay away from modeling
- mesh adaptivity
- Parallel computing
- Stream Computing

1
   - Scientific Computing in Finance
   - A Mix of Monte-Carlo and PDE
   - A Reduced Basis
   - Back to PDE, especially American Option

A bit of both worlds: mixed Monte-Carlo + PDE methods

- Heston Model: with $v_t = \sigma_t^2$

$$
\begin{aligned}
dS_t &= S_t r \mathrm{d}t + S_t \sigma_t (\rho dW_t^1 + \sqrt{1-\rho^2} dW^2), \\
dv_t &= k(\theta - v_t)\mathrm{d}t + \delta \sqrt{v_t} dW_t^2, \\
& S_{i+1} = S_i (1 + r\delta t + \sigma_i \sqrt{\delta t}(N_{0,1}^1 \rho + N_{0,1}^2 \sqrt{1-\rho^2})) \\
& v_{i+1} = v_i + k(\theta - v_i)\delta t + \sigma_i \sqrt{\delta t} N_{0,1}^2 \delta \text{ with } \sigma_i = \sqrt{v_i} \\
& P_M = \frac{1}{M} \sum (K - S_N^m)^+.
\end{aligned}
$$

A bit of both worlds: mixed Monte-Carlo + PDE methods

- Heston Model: with $v_t = \sigma_t^2$

$$
\begin{aligned}
dS_t &= S_t r \mathrm{d}t + S_t \sigma_t (\rho dW_t^1 + \sqrt{1-\rho^2} dW^2), \\
dv_t &= k(\theta - v_t)\mathrm{d}t + \delta \sqrt{v_t} dW_t^2, \\
& S_{i+1} = S_i (1 + r\delta t + \sigma_i \sqrt{\delta t}(N_{0,1}^1 \rho + N_{0,1}^2 \sqrt{1-\rho^2})) \\
& v_{i+1} = v_i + k(\theta - v_i)\delta t + \sigma_i \sqrt{\delta t} N_{0,1}^2 \delta \text{ with } \sigma_i = \sqrt{v_i} \\
& P_M = \frac{1}{M} \sum (K - S_N^m)^+.
\end{aligned}
$$

The method is slow, so keep MC for v_{i+1} but use a PDE for P_M, result of an Itô calculus on the equation for S_t for each realization $\sigma^m = \{\sqrt{v_i^m}\}_{i=1}^N$. So for each m, solve analytically or numerically the PDE with respect to the variables t and S *conditionally to the volatility realization* σ^m.. Intuitively $r \longrightarrow r + \frac{\sqrt{1-\rho^2}}{\delta t} \delta W^2$

Proposition *As $\delta t \to 0$, the process S_t converges to the solution of the stochastic differential equation of Heston's model. The option is given by*

$$\partial_t u + rS\partial_S u + \frac{1}{2}\sqrt{1-\rho^2}\sigma_t^2 S^2 \partial_{SS} u + \rho\sigma_t\mu_t S\partial_S u = ru$$

With constant coefficients there is analytical solution:

$$\begin{aligned}
\bar{\sigma}^2 &= \frac{1}{T}\int_0^T \sigma_t^2 \mathrm{d}t, \quad M = \frac{\rho}{T}\sum_i \sigma_{t_i}(W^2_{t_{i+1}} - W^2_{t_i}), \\
S(x) &= S_0 \exp\left((r+M)T - \frac{1}{2}\bar{\sigma}^2 T + \sqrt{1-\rho^2}\bar{\sigma}x\right), \\
u(0,S_0) &= e^{-rT}\int_{\mathcal{R}^+} \phi(S(x))\frac{\exp(-x^2/2T)}{\sqrt{2\pi T}}\mathrm{d}x
\end{aligned}$$

ρ	-0.5	0	0.5	0.9
Heston MC	11.135	10.399	9.587	8.960
Heston MC+BS	11.102	10.391	9.718	8.977
Speed-up	42	44	42	42

$S0 = 100$, $K = 90$, $r = 0.05$, $\sigma_0 = 0.6$,$\theta = 0.36$, $k = 5$, $\delta = 0.2$, $T = 0.5$.
$M = 3\ 10^5$, $M' = 10^4$, $\frac{T}{\delta t} = 300$, $\frac{S_{max}}{\delta S} = 400$.

Proof : Consider the following process:

$$\begin{aligned} dS_t &= rS_t dt + \rho\sigma_t S_t \mu_t dt + \sigma_t \sqrt{1-\rho^2} d\tilde{W}_t^1, \\ \mu_t &= \frac{W^2(t_{i+1}) - W^2(t_i)}{\delta t} - \frac{1}{2}\rho\sigma_t S_t \text{ for } t \in [t_i, t_{i+1}). \end{aligned}$$

By Ito's formula we have

$$d\log(S) = r dt + \rho\sigma_t\mu_t dt + \sqrt{1-\rho^2}\sigma_t d\tilde{W}_t^1 - \frac{1}{2}(1-\rho^2)\sigma_t^2 dt.$$

Thus we get

$$\begin{aligned} S(t_{i+1}) &= S(t_i)\exp\left(r\delta t + \sigma_{t_i}\left(\sqrt{1-\rho^2}(\tilde{W}^1_{t_{i+1}} - W^1_{t_i}) + \rho(W^2_{t_{i+1}} - W^2_{t_i})\right) - \right. \\ &= S(t_i)\exp\left(r\delta t + \sigma_{t_i}\left(W^1_{t_{i+1}} - W^1_{t_i}\right) - \frac{1}{2}\sigma^2_{t_i}\delta t\right) \end{aligned}$$

We recognize here a discretization of the stochastic integral

$$\exp\left(rt + \int_0^t \sigma_t dW_t^1 - \frac{1}{2}\int_0^t \sigma^2 dt\right).$$

where σ_t solves the second equation of Heston's model

Outline

Aim: Be fast and accurate. Be adaptive.

- See if Partial Differential Equations do better than Monte-Carlo.
- Stay away from modeling
- mesh adaptivity
- Parallel computing
- Stream Computing

1
   - Scientific Computing in Finance
   - A Mix of Monte-Carlo and PDE
   - A Reduced Basis
   - Back to PDE, especially American Option

A Basis for PIDE and Calibration (+ R. Cont & N. Lantos)

Consider the Black-Scholes equation:

$$\partial_t u + rS\partial_S u + \frac{\sigma^2(S,t)}{2}\partial_{SS} u - ru = 0$$

Proposition

Assume that $\sigma(e^{r(T-t)}\frac{S}{K}, t) = \sigma(e^{-r(T-t)}\frac{K}{S}, t)$ Choose c; let u^i be the solution with constant vol $\sigma = \frac{c}{\sqrt{i}}$ and let

$w^i(S) = \mathcal{L}_\sigma u^i := \{rS\partial_S u^i + \frac{\sigma^2(S,t)}{2}\partial_{SS} u^i - ru^i\}|_{t=0}$. Then $\{w^i\}_{i=1,2...}$ is a spacial basis for u in the sense that

$$u(S,t) = u_\Sigma(S,t) + \sum_{i=1}^{\infty} \alpha_i(t) w^i(S)$$

Proof: Set $\tau = T - t$, $y = e^{r(T-t)}\frac{S}{K}$, the forward moneyness price.

Let $v(y,\tau) = \frac{e^{r(T-t)}}{K} C(S,t)$, then

$$\partial_\tau v - \frac{1}{2}\sigma^2 y^2 \partial_{yy} v = 0 \quad \text{in } \mathcal{R}^+ \times (0,T), \quad v(y,0) = v_0(y)$$

$$\partial_\tau v - \frac{1}{2}\sigma^2 y^2 \partial_{yy} v = 0 \quad \text{in } \mathcal{R}^+ \times (0,T), \quad v(y,0) = v_0(y)$$

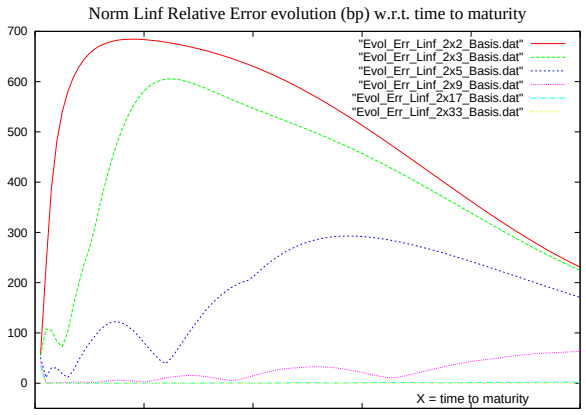
- Let $x = \ln y$. When $\sigma^2 = i$, v is such that $\mathcal{L}_\sigma v = C\sqrt{y}e^{-i\frac{x^2}{2c^2 T}}$.
- For a given $\alpha \neq 0$ the set $\{\exp(-n\alpha x^2)\}_{n\in N}$ is an algebra, so by the Stone-Weiestrass theorem it is a basis for the continuous *even* functions on $\mathcal{R}^+$ which decay exponentially fast at $\pm\infty$ because $\exp(-\alpha x^2)$ is a separating function on $\mathcal{R}^+$ (i.e. $f(x) \neq f(x')$ for all $x \neq x' \geq 0,\ x \geq 0$).
- Given a function $y \to f(y)$ we can decompose $g(\ln y) := f(y)/\sqrt{y}$ on the basis w^i only if $g(x)$ is even,i.e. only if $yf(\frac{1}{y}) = f(y)$

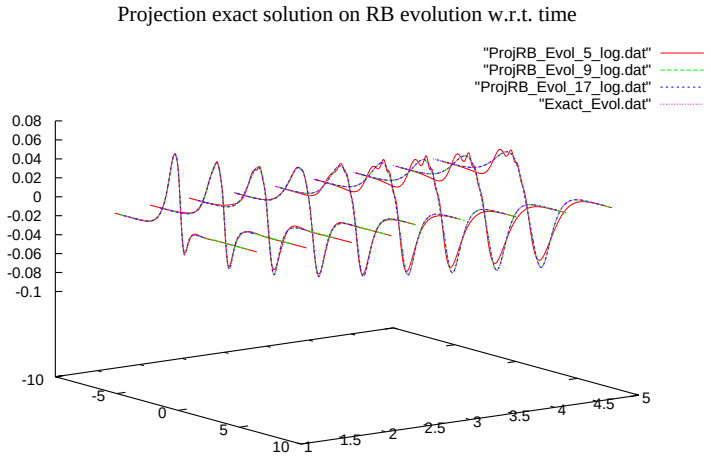
Proposition If $\sigma(y,\tau) = \sigma(\frac{1}{y},\tau)$ and $f(y,\tau) = yf(\frac{1}{y},\tau)\ \forall y > 0$ then v

$$\partial_\tau v - \frac{\sigma^2 y^2}{2}\partial_{yy} v = f, \quad v(y,0) = 0, \text{ in } \mathcal{R}^+ \times [0,T] \qquad (1)$$

is invariant under the transformation $v(y) \to yv(\frac{1}{y})$.

Extension to Option based on Jump Processes





Proposition For "most" kernels J with $v(y,0) = v_0(y)$ and

$$\partial_\tau v - \frac{1}{2}\sigma^2 y^2 \partial_{yy} v + \int_{\mathbb{R}} v(ye^z)J(z)dz - v(y)\int_{\mathbb{R}} J(z)dz - y\partial_y v(y)\int_{\mathbb{R}}(e^z - 1)J(z) = 0$$

v can be decomposed on $\{\mathcal{L}_\sigma v^i\} \cup \{\mathcal{L}^J v^i\}$ where $\mathcal{L}_\sigma = -\frac{1}{2}\sigma^2 y^2 \partial_{yy}$, $\mathcal{L}^J$ the Levy integral op of the PIDE and v^i is the BS sol. with vol $c/\sqrt{i}$.

Exact Merton vs the projections for $n = 5, 9, 17$ *at* $\tau \in [1; 5]$

Application to Calibration

one may solve

$$\sigma = \arg\,\min_\sigma \sum_{j=1}^{J} |u_\sigma(K_j, T_j) - u_j|^2 \qquad (2)$$

With a similar decomposition but with K, T and Dupire's equation, then (2) is a sum of independent problems at each time T_j: for each T^* one solves

$$\alpha(T^*) = \arg\min_\alpha \sum_{j:T_j=T^*} |u(K_j, T^*; \alpha) - u_j|^2 \ :$$

$$u(K_j, T^*; \alpha) = u_\Sigma(K_j, T^*) + \sum_{i=1}^{I} \alpha_i \left[u_{\sigma_i}(K_j, T_M) - u_\Sigma(K_j, T_M)\right]$$

where $T_M = \max T_j$ is the reference time chosen to build the basis. The volatility surface is recovered from Dupire's equation and $u_\sigma(K, T) = u(K, T; \alpha)$; the derivatives with respect to K are computed analytically.

Observation Data

Strike	1 Month	2 Months	6 Months	12 Months	24 Months	36 Months
700						733
800						650.6
900						569.8
1000					467.8	
...	...	...	...	...	...	...
1215				253.4		
1225			219	245		
1250			196.6	224.2	269.2	
1275			174.5	203.9	251	
1300			152.9	184.1	233.2	
1325			131.9	164.9	215.8	
1350			111.7	146.3	198.9	
1365			100			
1375	50.6	60	92.5	139	182.6	
1380	46.1	55.8				
1385	41.8	51.8				
...	...	...	...	...	...	...
1700					32.7	75,7
1800					15.5	
1900					5.2	

index SPX on 21.12.2006) at spot price 1418.3, r=3/100

Results I

Here there are 6 times T^*. The basis is made of Black-scholes calls with volatility $0.3/\sqrt{i}$, i=2..9. The Black-Scholes solution used for the translation corresponds to $\Sigma = 0.3$.

- At each T^* a set of 8 α_i are computed by solving (3) by a conjugate gradient method with a maximum of 300 iterations.
- We found the optimization more efficient if α_i is replaced by $10 \sin \alpha_i$.

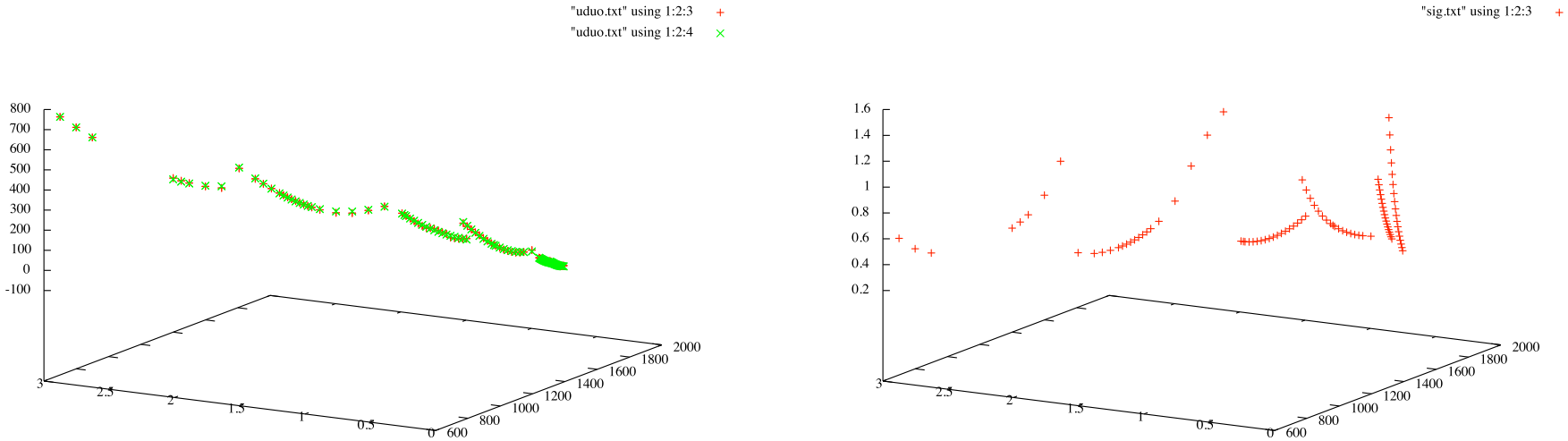


Figure: Top: Difference between observations and model predictions (left). Bottom: the local volatility. The method is very fast, even faster than fitting with an implied volatility, but it gives the local volatility only at all T^*. .

American Put Option

$$P_0 = \sup_{\tau \in (0,T)} \mathbf{E}[e^{-r\tau}(K - S_\tau)^+]$$

$$= \sup_{\tau} \mathbf{E}[e^{-r\tau}(K - S_0 e^{(r-\frac{\sigma^2}{2})\tau + \sigma W_\tau})^+] \text{ when } r, \sigma \text{ are constant}$$

$$\approx \sup_{\tau} \frac{1}{K} \sum_k [e^{-r\tau}(K - S_0 e^{(r-\frac{\sigma^2}{2})\tau + \sigma\sqrt{\tau} N_{01}^k})^+]$$

By comparison the same European option is valued as

$$\frac{1}{K} \sum_k [e^{-r\tau}(K - S_0 e^{(r-\frac{\sigma^2}{2})T + \sigma\sqrt{T} N_{01}^k})^+]$$

Longstaff-Schwartz: an example (P.Glasserman)

Stock K=110, r=6%, $h(S)=(K-S)^+$ $\quad E[Y|X] = -107 + 2.983X - 0.01813X^2$

Path	t=0	t = 1	t = 2	t = 3	$h(s_3)$	Y=$h(s_3)$(1-r)	X=S_2	h(X)	E(X|Y)
1	100	109	108	134	0	0	108	2	3.69
2	100	116	126	154	0	0	126	0	0
3	100	122	107	103	7	7 $\times$.94	107	3	4.61
4	100	93	97	92	18	18 $\times$.94	97	13	11.76
5	100	111	156	152	0	0	156	0	0
6	100	76	77	90	20	20 $\times$.94	77	33	15.2
7	100	92	84	101	9	9 $\times$.94	84	26	15.7
8	100	88	122	134	0	0	122	0	0

$E[Y|X] = 203.8 - 3.335X + 0.01356X^2$ $\quad$ Stopping rule

Path	t = 2	t = 3	Y	X=S_1	h(X)	E(Y|X)	Path	Stop	P
1	0	0	0	109	1	1.39	1		0
2	0	0	0	0	0	0	2		0
3	3	7	4.61 $\times$.94	0	0	-2	3	t_3	7
4	13	0	13 $\times$.94	93	17	10.92	4	t_1	17
5	0	0	0	0	0	0	5		0
6	33	0	33 $\times$.94	76	34	28.66	6	t_1	34
7	26	0	26 $\times$.94	92	18	11.75	7	t_1	18
8	0	0	0	88	22	15.33	8	t_1	22

Implementation in C++(I)


```cpp
// Pricing an American Put Option by Longstaff & Schwartz's Least-Squares Monte-Carlo
// written by Tobias Lipp (LJLL-UPMC), Feb 2009

#include <iostream>
#include <cmath>
#include <cstdlib>
using namespace std;

class lsmc {  public:
    lsmc(double, double, double, double, double, int, int, int);
    ~lsmc();
    void Cal_Price();
    void Display_Results();
private:
    double payoff(double S) { return K-S>0. ? K-S : 0.; }
    void Generate_Trajs();
    void Init_cf();
    void Cal_ExVal(int&, double*, int);
    void Build_Ab(double**, double*, int, double*);
    void factQR(double**, double*, double*, int);
    void solveQR(double**, double*, double*, double*);
    void Update_cf(double*, double*, int);

    double T, K, S0, r, sig;
    int I, M, N;
    double dt, sdt,
    **S, **cf;
    double PA;
};

lsmc::lsmc(double nT, double nK, double nS0, double nr, double nsig, int nI, int nM, int nN)
: T(nT), K(nK), S0(nS0), r(nr), sig(nsig), I(nI), M(nM), N(nN) {
    dt = T/M;
    sdt = sqrt(dt);
    // matrix: stock prices
    S = new double*[I];
    for(int i=0; i<I; i++)  S[i] = new double[M+1];
    // matrix: cash flows
    cf = new double*[I];
    for( int i=0; i<I; i++)  cf[i] = new double[2];
    PA=0.;
}

lsmc::~lsmc() {
    for(int i=0; i<I; i++) {  delete[] S[i];  delete[] cf[i]; }
    delete[] S;  delete[] cf;
}
// -------------------------------------------------------
void lsmc::Cal_Price() {
    Generate_Trajs();
    Init_cf();

    for(int t=M-1; t>0; t--) {
        // Immediate exercise values and nb of 'in the money pathes' at time t
        double* const ex = new double[I];
        int nmp=0;
        Cal_ExVal(nmp,ex,t);
        // Least Square Pb.: min|Ax-b|^2,  b = cashflow * discountfactor
        double** const A = new double*[nmp];
        for( int i=0; i<nmp; i++)
            A[i] = new double[N];
```



```cpp
        double* const b = new double[nmp];
        Build_Ab(A,b,t,ex);
        // QR-Decomposition via Householder,  Transformations: A = QR
        double* diagR = new double[N];
        factQR(A,b,diagR,nmp);
        // Solve R x = b1 (b=(b1,b2))
        double* const x = new double[N];
        solveQR(A,b,x,diagR);
        // Update cashflow mx
        Update_cf(ex,x,t);

        delete[] ex;
        for(int i=0; i<nmp; i++)  delete[] A[i];
        delete[] A;  delete[] b;  delete[] diagR;  delete[] x;
    }
    // PA: Price American (Put)
    for( int i=0; i<I; i++)  PA += exp(-r*cf[i][0]*dt)*cf[i][1];
    PA /= I;

}
// -------------------------------------------------------
double gauss() {
    double x=double(1.+rand())/double(1.+RAND_MAX);
    double y=double(1.+rand())/double(1.+RAND_MAX);
    return sqrt(-2*log(x))*cos(2*M_PI*y);
}
// -------------------------------------------------------
void lsmc::Generate_Trajs() {
    for(int i=0; i<I; i++) {
        S[i][0] = S0;
        for(int j=1; j<=M; j++)
            S[i][j] = S[i][j-1]*( 1 + r*dt + sig*sdt*gauss() );
    }
}

void lsmc::Init_cf() {
    for( int i=0; i<I; i++) {
        cf[i][0] = M; // cash flows at time cf[*][0]
        cf[i][1] = payoff(S[i][M]); // cf[*][1] flowing amount
    }
}
// -------------------------------------------------------
void lsmc::Cal_ExVal(int& nmp, double* ex, int t) {
    for(int i=0; i<I; i++) {
        ex[i] = payoff(S[i][t]);
        if( ex[i] > 0. )  nmp++;
    }
}

void lsmc::Build_Ab(double** A, double* b, int t, double* ex) {
    int ii=0;
    for (int i=0; i<I; i++) {
        if( ex[i] == 0. )
            continue;
        for(int j=0; j<N; j++)
            A[ii][j] = pow(S[i][t],j);
        b[ii] = exp(-r*(cf[i][0]-t)*dt) * cf[i][1];
        ii++;
    }
}
// -------------------------------------------------------
```

Implementation in C++(II)


```
inline double sqr(double x) {  return x*x;}
inline double sgn(double a) {  return a>0. ? 1. : -1.;}
double norm2(double** A, int I, int j) {
    double norm2=0.;
    for(int i=j; i<I; i++)  norm2 += sqr(A[i][j]);
    return norm2;
}
// -----------------------------------------------------
void lsmc::factQR(double** A, double* b, double* diagR, int nmp) {
    for(int j=0; j<N; j++) {
        diagR[j] = -sgn(A[j][j])*sqrt(norm2(A,nmp,j));
        A[j][j] -= diagR[j];

        double v2 = norm2(A,nmp,j);

        for(int jj=j+1; jj<N; jj++) {
            double va=0.;
            for(int i=j; i<nmp; i++)
                va += A[i][j]*A[i][jj];
            for(int i=j; i<nmp; i++)
                A[i][jj] -= 2*va/v2*A[i][j];
        }

        double vb=0.;
        for(int i=j; i<nmp; i++)
            vb += A[i][j]*b[i];
        for(int i=j; i<nmp; i++)
            b[i] -= 2*vb/v2*A[i][j];
    }
}
// -----------------------------------------------------
void lsmc::solveQR(double** A, double* b, double* x, double* diagR) {
    for(int i=N-1; i>-1; i--) {
        x[i] = b[i];
        for(int j=i+1; j<N; j++)
            x[i] -= A[i][j]*x[j];
        x[i] /= diagR[i];
    }
}
// -----------------------------------------------------
void lsmc::Update_cf(double* ex, double* x, int t) {
    for(int i=0; i<I; i++) {
        if( ex[i] <= 0. )
            continue;
        double con=0.;
        for(int k=0; k<N; k++)
            con += x[k]*pow(S[i][t],k);
        if( con <= ex[i] ) {
            cf[i][0] = t;
            cf[i][1] = ex[i];
        }
    }
}
// -----------------------------------------------------
void lsmc::Display_Results() {
    std::cout << " Price American Put: P = " << PA << std::endl;
}

int main() {

  srand(time(0));
```



```
  int ela = clock();

  const double T=1., K=100., S0=100., r=0.05, sig=0.3;
  const int I=100000, // nb. of trajectories (muss >= N sein)
            M=20, // nb. of time steps
        N=4;  // nb. of basis fcts.

  lsmc p(T,K,S0,r,sig,I,M,N);

  p.Cal_Price();
  p.Display_Results();

  ela = clock() - ela;
  cout << " Elapsed time: " << ela/1e6 << " sec" << endl;
  return 0;
}
```

Outline

Aim: Be fast and accurate. Be adaptive.

- See if Partial Differential Equations do better than Monte-Carlo.
- Stay away from modeling
- mesh adaptivity
- Parallel computing
- Stream Computing

1.
   - Scientific Computing in Finance
   - A Mix of Monte-Carlo and PDE
   - A Reduced Basis
   - Back to PDE, especially American Option

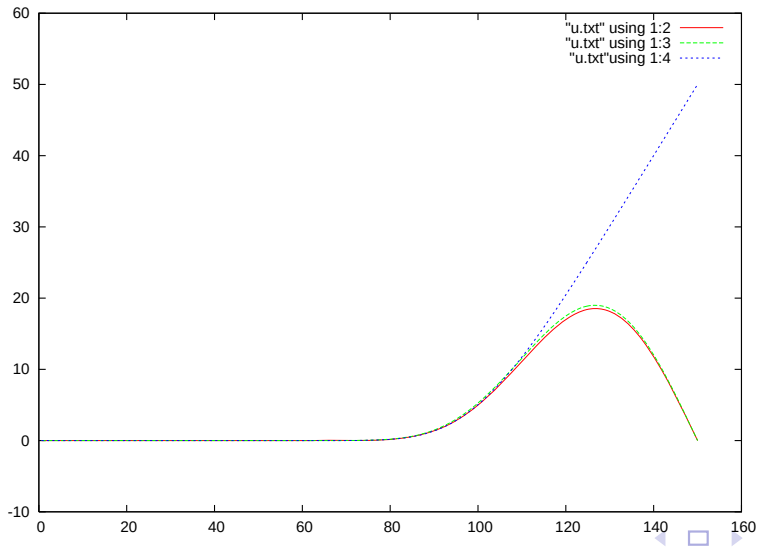
Back to PDE: Barrier Options

If u stops to exist when $S_t > S_M$ and when $S_t < S_M$ then just add $u(S_M, t) = 0$ for all t. The theory is the same but with

$$V = \left\{ v \in L^2(]S_m, S_M[) : \; x\frac{dv}{dx} \in L^2(]S_m, S_M[), v(S_m) = v(S_M) = 0 \right\}$$

$$u_h(x,t) = \sum_{j=2}^{N} u_j(t) w^j(x) \quad \text{(do not take the first and last hat function)}$$

$$\Leftrightarrow \; B\frac{dU}{dt} + A(t)U = 0, \text{ where } B_{ij} = (w^j, w^i), \; A_{ij}(t) = a_t(w^j, w^i).$$



American Options

In American options one of the two must be also an equality

$$\begin{aligned} \frac{\partial u}{\partial t} - \frac{\sigma^2 x^2}{2}\frac{\partial^2 u}{\partial x^2} - rx\frac{\partial u}{\partial x} + ru &\geq 0 \\ u &\geq u_\circ := (K-x)^+ \end{aligned}$$

$$V = \left\{ v \in L^2(\mathcal{R}^+), x\frac{\partial v}{\partial x} \in L^2(\mathcal{R}^+) \right\},$$

$$\mathcal{K} = \{ \mathbf{v} \in \mathbf{L^2(0,T;V)}, \mathbf{v} \geq \mathbf{u}_\circ \text{ a.e. in } \mathbf{(0,T)} \times \mathcal{R}^+ \}.$$

Find $\mathbf{u} \in \mathcal{K} \cap C^0([0,T]; L^2(\mathcal{R}^+)), \qquad \frac{\partial u}{\partial t} \in L^2(0,T;V'),$

$$\begin{aligned} \textbf{s.t.} \qquad (\frac{\partial \mathbf{u}}{\partial \mathbf{t}}, \mathbf{v}-\mathbf{u}) + \mathbf{a_t(u, v-u)} &\geq 0, \qquad \forall v \in \mathcal{K}, \\ \mathbf{u(t=0)} &= \mathbf{u}_\circ \end{aligned}$$

This is similar to the ice-water problem in engineering.

Regularity Results (Achdou)

Assume that there exists $M > 0$ s.t.

$$|x^2 \frac{\partial^2 \sigma}{\partial x^2}| + |\frac{\partial \sigma}{\partial t}| + |x \frac{\partial^2 \sigma}{\partial x \partial t}| \leq M, \quad \text{a.e.}$$

- The free boundary (exercise prize) is a continuous curve $t \longrightarrow \gamma(t)$
- $x \longrightarrow u(t,x)$ is convex
- If $t \longrightarrow \gamma(t)$ is Lipschitz in $[\tau, T]$ then

$$\|(\gamma(\sigma') - \gamma(\sigma))^+\|^{\mathbf{3}}_{\mathbf{L}^3(\tau,\mathbf{T})} \leq \mathbf{c}_\tau \|\sigma - \sigma'\|^{\mathbf{2}}_{\mathbf{L}^\infty((\tau,\mathbf{T})\times\mathcal{R}^+)} .$$

which implies differentiability in σ away from zero.

Semi-Smooth Newton Method (K.Kunisch)

After time discretization reformulate the problem as

$$a(u,v) - (\lambda, v) = (f, v) \quad \forall v \in H^1(\mathcal{R}^+), \quad i.e. Au - \lambda = f$$
$$\lambda - \min\{0, \lambda + c(u - \phi)\} = 0,$$

The last eq. is equivalent to $\lambda \leq 0,\ \lambda \leq \lambda + c(u - \phi)$ i.e. $u \geq \phi,\ \lambda \leq 0$, with equality on one of them for each x. This problem is equivalent for any real constant $c > 0$ because λ is the Lagrange multiplier of the constraint.

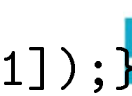
Newton's algorithm gives

- Choose $c > 0,\ , u_0,\ \lambda_0$, set $k = 0$.
- Determine $A_k := \{x\ :\ \lambda_k(x) + c(u_k(x) - \phi(x)) < 0\}$
- Set $u_{k+1} = \arg\min_{u \in H^1(\mathcal{R}^+)}\{a(u,u) - 2(f,u)\ :\ u = \phi \text{ on } A_k\}$
- Set $\lambda_{k+1} = f - Au_{k+1}$

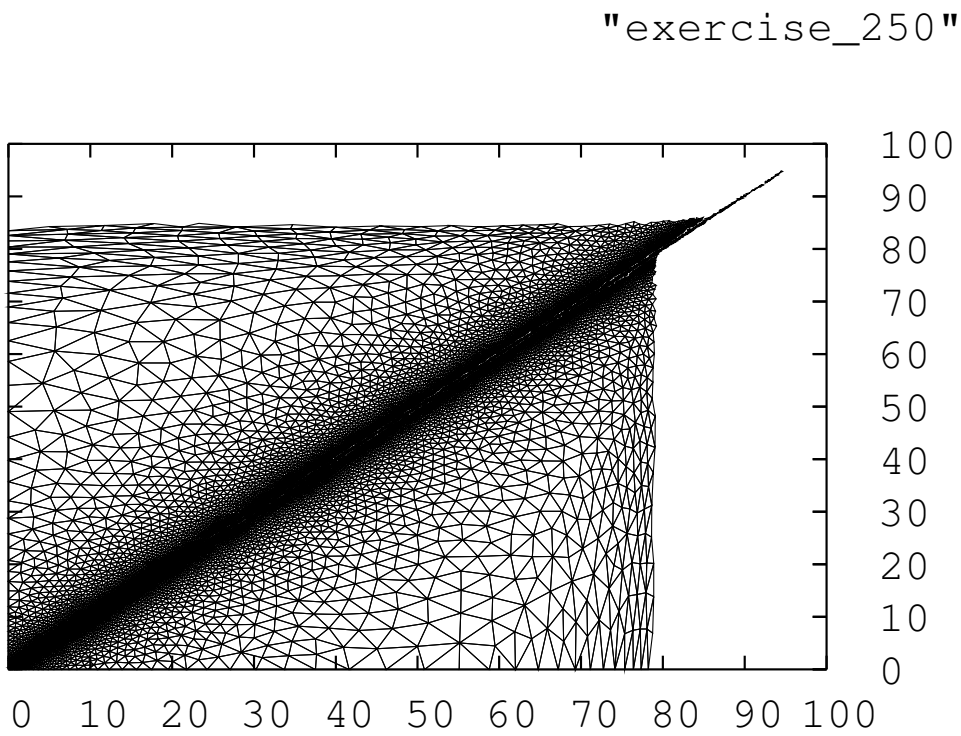
Theorem For any $c > 0$ $u^k \longrightarrow u$ solution of the problem.

Simplicity

```
void Option::calc( const bool AMERICAN){
  double c=10, tgv = 1e30;  int kmax = AMERICAN*3+1;
  for(int i=0;i<nX;i++) u[i] = max(K-i*dx,0);
  for(int j=0;j<nT;j++){ int jT=(nT-1-j)*T;
      for(int i=1;i<nX-1;i++)  // rhs of EDP
        {  uold[i] = u[i] + dt*r*i*(u[i+1]-u[i-1])/2; lam[i] = 0; }
      u[nX-1]=0; u[0] = exp(-r*(j+0.5)*dt);
/**/ for(int k=0;k<kmax;k++){
        for(int i=1;i<nX-1;i++){
        double aux=i*sigma[jT][i]*i*sigma[jT][i]*dt/2;
            bm[i] = (1+ r*dt + 2*aux); am[i] = -aux; cm[i] =-aux;
/**/       if(AMERICAN && lam[i]+c*(u[i]-max(K-i*dx,0))<0)
              { indic[i]=1; bm[i] = tgv;} else indic[i]=0;
/**/    }       factLU();  for(int i=1;i<nX-1;i++)
/**/    if(indic[i]) u[i] = tgv*max(K-i*dx,0); else u[i]=uold[i];
 solveLU(u);    for(int i=1;i<nX-1;i++){
  double aux=i*sigma[jT][i]*i*sigma[jT][i]*dt/2;
 /**/   lam[i]=uold[i]-(1+r*dt+ 2*aux)*u[i]+aux*(u[i+1]+u[i-1]);}}}
```



Results (Achdou)



Best of put basket option, $\sigma_1 = 0.2,\ \sigma_2 = 0.1, \rho = -0.8$

Multidimensional problems

1. Basket Option $\mathrm{d} < W_i W_j >= \sigma_{ij} \mathrm{d}t$

$$\mathrm{d}S_i = S_i(r\mathrm{d}t + \mathrm{d}W_i),\ i = 1..d, \quad u = e^{-(T-t)r}\mathbf{E}(\sum S_i - K)^+$$

Ito calculus leads to a multidimensional Black-Scholes equation

$$\partial_\tau u - \sum_i (\frac{\sigma_{ij}^2 x_i x_j}{2} \partial_{x_i x_i} u - r x_i \partial_{x_i} u) + ru = 0 \quad u(0) = (\sum_i x_i - K)^+$$

Stochastic Volatility models (Stein-Stein, Orstein-Uhlenbeck, Heston)

$$\mathrm{d}S_t = S_t(r\mathrm{d}t + \sigma_t \mathrm{d}W_t), \quad \sigma_t = \sqrt{Y_t},$$
$$\mathrm{d}Y_t = \kappa(\theta - Y_t)\mathrm{d}t + \beta \mathrm{d}Z_t, \quad \mathrm{d} < W_t, Z_t >= \rho \mathrm{d}t$$
$$\partial_t U + aU + bx\partial_x U - \frac{x^2 y}{2}\partial_{xx} U - \frac{\beta^2 y}{2}\partial_{yy} U - \rho\beta y x \partial^2_{xy} U + c\partial_y U = 0$$
$$a = \mu - \kappa - \rho\beta - y, \quad b = (\mu - 2y - \rho\beta), \quad c = \kappa(\theta - y) - \rho\beta y - \beta^2$$

Basket with 3 Assets

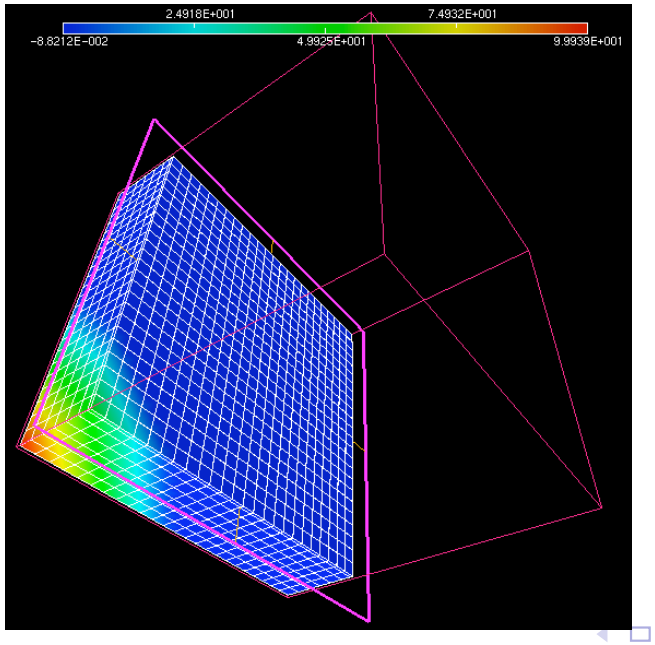
Use http://ww.freefem.org/freefem3D :

```
double N = 25;          double L=200.0;        double T=0.5;
double dt = T / 15 ; double K=100;           double r = 0.02;
double s1 = 0.3;       double s2 = 0.2;       double s3 = 0.25;
double q12 = -0.2*s1*s2;           double q13 = -0.1*s1*s3;
double q23 = -0.2*s2*s3;           double s11 = s1*s1/2;
double s22=s2*s2/2;                 double s33=s3*s3/2;

vector n = (N,N,N);
vector a = (0,0,0);
vector b = (L,L,L);
mesh M = structured(n,a,b);
femfunction uold(M) = max(K-x-y-z,0);
femfunction u(M);
```

3D Basket Option(II)

```
double t=0;  do{
    solve(u) in M  cg(maxiter=900,epsilon=1E-10)
    {
pde(u)
    (1/dt+r)*u-dx(s11*x^2*dx(u))-dy(s22*y^2*dy(u))-dz(s33*z^2*dz(u))
    - dx(q12*x*y*dy(u)) - dx(q13*x*z*dz(u)) - dy(q23*y*z*dz(u))
    - r*x*dx(u) - r*y*dy(u) - r*z*dz(u) = uold/dt;
        dnu(u)=0 on M;
     };  t = t + dt;
} while(t<T);
save(medit,"u",u,M);
save(medit,"u",M);
```



Sparse Grids (in more than 3 dimensions)

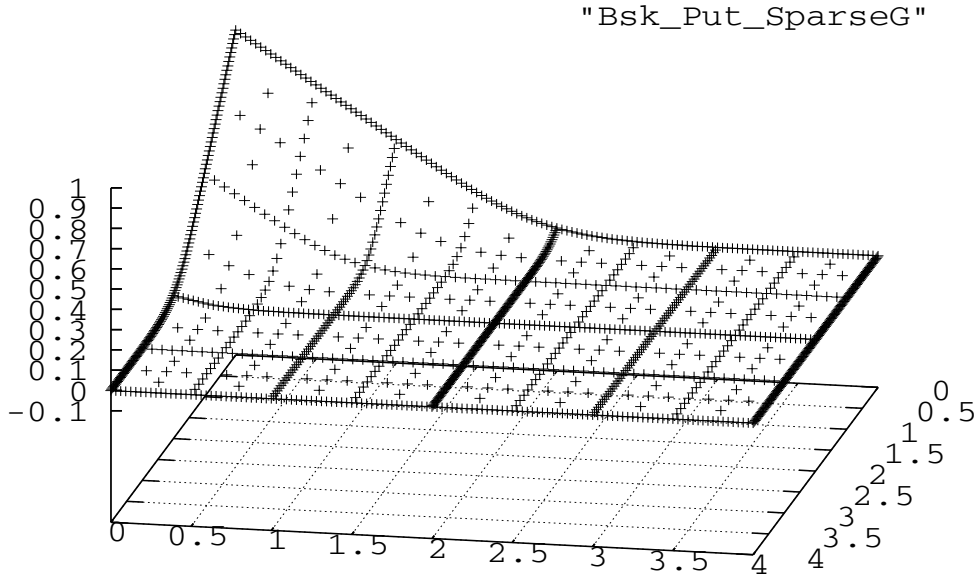
- If f is analytic then

$$\int_D f \approx \sum_i f(q^i)\omega_i$$

where most of the points are on ∂D. The argument is recursive.
S.A. Smolyak: Quadrature and interpolation formulas for tensor products of certain classes of functions Dokl. Akad. Nauk SSSR 4 pp 240-243, 1963.

- **Michael Griebel**: The combination technique for the sparse grid solution of PDEs on multiprocessor machine. Parallel Processing Letters 2 1(61-70), 1992.
In polynomial approximations of f most of the mixed terms $x_1^i x_2^j \ldots$ are not needed.

Sparse Grids (when dimension $d > 3$)



Computed by D. Pommier

See also C. Schwab et al. who can solve up to dimension 20 and a PIDE in dimension 5. **Nils Reich** built a sparse tensor product wavelet compression scheme of complexity $O(\frac{1}{h}|\log h|^{2(d-1)})$.

Summary

- Now BNP has a PDE dept (one scientist)
- Calibration and Americans are the preferred problems for PDEs
- Monte-Carlo will profit much more from GPGPU than PDEs

```
const char *KernelSource =      "\n"\
"__kernel square( __global float* input1,  __global float* input2,
"   const float rst,  const unsigned int count) {  \n"\
" const float PI =3.141592653f;\n"\
" int i = get_global_id(0);  \n"\
" float z, z1, z2; \n"\
" if(i < count){ \n"\
" z1= -2.0 * log(input1[i]); \n"\
" z2 = 2.0 * PI * input2[i]; \n"\
" z = rst*sqrt(z1)*cos(z2); \n"\
" output[i] = exp(z); \n"\
" } \n"\
"}  \n";
```

Thank you very much indeed for the invitation