Reduced Basis for Vanilla and Basket Options

Olivier Pironneau *

July 19, 2012

Abstract

In [3] it was shown that by writing the solution of the Black-Scholes partial differential equation on a small set of basis functions the computing time can be dramatically reduced. In this study we explore the generalization of the technique to basket options.

Keywords  Option pricing, Galerkin basis, reduced basis

AMS  37M25, 65N99

Introduction

In banks and finance in general, derivatives like European calls on a single or compound assets are computed a very large number of times every day. When closed forms solutions are not available, an alternative to Monte-Carlo simulation is to solve the Black-Scholes partial differential equation (more details can be found in [2, 3, 4], for example).

An alternative to the Finite Difference/Element Methods for a numerical solution of the problem is to construct an appropriate basis, smaller in size but with larger support. Proper Orthogonal Decomposition is the usual tool but in [3] it was shown that a set of rescaled calls with constant volatilities form an even better reduced basis than POD.

More precisely, a call $C_{\sigma} = e^{-r\tau} K u_{\sigma}$ with volatility $\sigma$ on an asset $S$ at time to maturity $\tau$ is defined by

$$\partial_{\tau} u_{\sigma} + L_{\sigma} u_{\sigma} = 0, \quad u_{\sigma}(0) = (y - 1)^+ \text{ with } L_{\sigma} u = -\frac{\sigma^2 y^2}{2} \partial_{yy} u$$

(1)

where $K$ is the strike, $r$ the interest rate and $y = e^{r \tau} S / K$, the moneyness price.

In practice banks work with non constant volatilities $\sigma(S, \tau)$ and there are no closed form solutions for (1). So, in order to benefit from the closed form solutions
solutions we proposed in [3] to use a reference $u_\Sigma$ with $\Sigma$ constant and a set of constant volatilities $\{\Sigma_i\}$' and compute $a_i(\tau)$ such that

$$u_\sigma(y, \tau) = u_\Sigma(y, \tau) + \sum_{i=1}^{l} a_i(\tau) \mathcal{L}_\sigma u_{\Sigma_i}|_{\tau=T}\tag{2}$$

approximates (1) in the sense of Galerkin.

For any given $c > 0$, when $\Sigma_i = (2c T i)^{-\frac{1}{2}}$, $i = 1, 2, \ldots$, the method is mathematically correct (see below) and can be extended to other models like jump-diffusion and stochastic volatilities.

A rapid computation shows that $\mathcal{L}_\sigma u_{\Sigma_i}|_{\tau=T}$ is proportional to $w_i := \sqrt{y} e^{-\alpha_i \ln^2 y}$ with $\alpha_i = ci$. So here, we shall investigate the representation

$$u_\sigma(y, \tau) = u_\Sigma(y, \tau) + \sum_{i=1}^{l} a_i(\tau) \sqrt{y} e^{-\alpha_i \ln^2 y}$$

with $u_\Sigma = \frac{y}{2} \left( 1 + \text{erf}(\frac{\ln y}{\sqrt{2}\tau}) + \frac{\tau}{\sqrt{8}} - \frac{1}{2} \left( 1 + \text{erf}(\frac{\ln y}{\sqrt{2}\tau} - \frac{\tau}{\sqrt{8}}) \right) \right)$

and $\text{erf}(y) = \frac{2}{\sqrt{\pi}} \int_{0}^{y} e^{-x^2} dx\tag{3}$

and then see if it can be extended to the multidimensional case for basket options.

1 The Basis

1.1 Mathematical Result

Decomposition (3) gives a $u_\sigma$ which satisfies the initial and boundary conditions of the Black-Scholes equations (1); it has also the right exponential behavior when $y \to \infty$.

Note that $e^{-ic \ln^2 y} = f(y)^i$ with $f(y) = e^{-c \ln^2 y}$. By the Stone-Weirstrass theorem $\{f(y)^n\}_{n=0,1,\ldots}$ is a basis if $y \to f(y)$ is a separating function; however $e^{-c \ln^2 y}$ is a separating function only on the interval $(0, 1)$ or on $(1, \infty)$ but not on $(0, \infty)$. So we need a symmetry assumption about 1: $u(\frac{1}{y}) = \frac{1}{y} u(y)$ for (3) to work. Fortunately if $\sigma(\frac{1}{y}, \tau) = \sigma(y, \tau)$ then it is easy to show that solutions of (1) have the required symmetry. If symmetry does not hold then another set of vectors must be added in the basis, such as $\{\sqrt{y}w^i(y)\}_{i=1,2,\ldots}$. More details can be found in [3].

1.2 Implementation

Note that

$$\partial_{yy} w^i = \frac{e^{-\alpha_i \ln^2 y}}{y \sqrt{y}} \left( 4\alpha_i^2 \ln^2 y - \frac{1}{4} - 2\alpha_i \right)$$
\[ \frac{\partial^2 u_\Sigma}{2} \frac{\partial u_\Sigma}{\Sigma} = \frac{(\Sigma^2 - \sigma^2)y^2}{2} \frac{\partial u_\Sigma}{\Sigma} = e^{-\frac{\Sigma^2}{\Sigma}} \frac{(\Sigma^2 - \sigma^2)}{2\Sigma{2\pi}^T} \sqrt{y} e^{-\frac{\ln^2 y}{2\Sigma^2}} \] (4)

So (3) is

\[ \sum_{1}^{f} \dot{a}_i \sqrt{ye^{-\alpha_i \ln^2 y}} - \sum_{1}^{f} a_i e^{-\alpha_i \ln^2 y} \sigma^2 \sqrt{y}(2\alpha^2 \ln^2 y - \frac{1}{8} - \alpha_i) \]

\[ = \frac{-e^{-\frac{\Sigma^2}{\Sigma}}}{2\Sigma{2\pi}^T} \frac{(\Sigma^2 - \sigma^2)}{\sqrt{y}} e^{-\frac{\ln^2 y}{2\Sigma^2}} \] (5)

The Galerkin method requires to multiply this equation by \( y^{-2} \sqrt{ye^{-\alpha_i \ln^2 y}} \) (the extra factor \( y^{-2} \) is for convenience) and integrate over \( \mathbb{R}^+ \). We need the following:

\[ M_{ij} = \int_{0}^{\infty} e^{-(\alpha_i + \alpha_j) \ln^2 y} d\frac{1}{y} = \int_{\mathbb{R}} e^{-z^2} \frac{dz}{\sqrt{\alpha_i + \alpha_j}} = \frac{\sqrt{\pi}}{\sqrt{\alpha_i + \alpha_j}} \]

\[ B_{ij} = -\int_{0}^{\infty} e^{-(\alpha_i + \alpha_j) \ln^2 y} \sigma^2 (2\alpha^2 \ln^2 y - \frac{1}{8} - \alpha_i) d\frac{1}{y} \]

\[ F_j = -\frac{e^{-\frac{\Sigma^2}{\Sigma}}}{2\Sigma{2\pi}^T} \int_{0}^{\infty} e^{-(\alpha_i + \frac{1}{2\alpha_i}) \ln^2 y} \sigma^2 (\Sigma^2 - \sigma^2) d\frac{1}{y} \] (6)

Finally, one has to solve for \( a = (a_1(\tau), ..., a_I(\tau))^T \) the system:

\[ M\dot{a} + Ba = F \] (7)

1.2.1 The Case \( \sigma \) Constant

When \( \sigma \) is constant:

\[ M_{ij} = \sqrt{\frac{\pi}{\alpha_i + \alpha_j}} \]

\[ B_{ij} = \frac{\sqrt{\pi\sigma^2}}{(\alpha_i + \alpha_j)^2} (\frac{\alpha_i \alpha_j}{\alpha_i + \alpha_j} + \frac{1}{8}) \]

\[ F_j = \frac{e^{-\frac{\Sigma^2}{\Sigma}}}{2\Sigma{2\pi}^T} \frac{(\Sigma^2 - \sigma^2)}{2\alpha_i \Sigma^2 + 1} \] (8)

1.2.2 The Non Constant Case

If \( \sigma \) is a function of \( y \) and \( \tau \), with \( \sigma(\frac{1}{y}, \tau) = \sigma(y, \tau) \) it is best to express it on an exponential basis as:

\[ \sigma(x, t) = \sigma_0 + \sum_{1}^{j} \sigma_j(t)e^{-\alpha_j \ln^2 y} \] (9)

because then all integrals can be computed analytically.
1.3 Numerical Tests

The method was thoroughly tested in [3] but since we are concerned with a minor variation of the original we retested the method for the computation of a vanilla call of volatility $\sigma = \sqrt{0.9}$ using $\Sigma = \sqrt{0.5}$ and $I$ ranging from 5 to 30. The maturity is $T = 2$ and 10 time steps are used to integrate (7); with 20 time steps the precision is not significantly improved.

Figure 1 shows that the precision of the method is well within the 3 digits required by banks.

1.4 Multiprecision Arithmetics

The linear systems derived from implicit finite difference time schemes for (7), such as Euler’s or Crank-Nicolson, are very ill conditioned and very hard to solve even by SVD (here svdcmp of the numerical recipe in C [5] was used); it is not possible to go beyond $I = 20$ with a C program using double precision, so the question of the validity of the answers is raised. To check the precision of the standard C program we have replaced double by the class of reals defined in the GNU multi-precision library gmp [7]. We wrote also a script for the stand alone, well optimized for such task, Pari/GP software [6] so as to obtain a reference solution. To kill any error due to time stepping we have used 400 time steps.

The following tables give the coefficients $\{a_i\}$ for several values of $I$ computed by the double precision C program (table 1), by the multi-precision C program using gmp with 100 digits (table 2) and Pari/GP with 2000 digits (table 3).

Table 1: Solution of (7) by a C-program using double precision

<table>
<thead>
<tr>
<th>(a_i)</th>
<th>13</th>
<th>11</th>
<th>9</th>
<th>7</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_1)</td>
<td>-0.0217749</td>
<td>-0.0213967</td>
<td>-0.0232162</td>
<td>-0.0253716</td>
<td>-0.028902</td>
</tr>
<tr>
<td>(a_2)</td>
<td>-0.240335</td>
<td>-0.248527</td>
<td>-0.198266</td>
<td>-0.150316</td>
<td>-0.107021</td>
</tr>
<tr>
<td>(a_3)</td>
<td>1.18192</td>
<td>1.26419</td>
<td>0.69126</td>
<td>0.273059</td>
<td>0.0664526</td>
</tr>
<tr>
<td>(a_4)</td>
<td>-5.38426</td>
<td>-5.93775</td>
<td>-2.42334</td>
<td>-0.54813</td>
<td>-0.0573113</td>
</tr>
<tr>
<td>(a_5)</td>
<td>15.5795</td>
<td>18.3422</td>
<td>5.43691</td>
<td>0.634395</td>
<td>0.0183907</td>
</tr>
<tr>
<td>(a_6)</td>
<td>-27.7013</td>
<td>-37.5786</td>
<td>-7.69616</td>
<td>-0.390463</td>
<td></td>
</tr>
<tr>
<td>(a_7)</td>
<td>27.287</td>
<td>51.1317</td>
<td>6.62803</td>
<td>0.0984288</td>
<td></td>
</tr>
<tr>
<td>(a_8)</td>
<td>-8.66529</td>
<td>-45.4532</td>
<td>-3.16675</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(a_9)</td>
<td>-7.74419</td>
<td>25.2024</td>
<td>0.643141</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(a_{10})</td>
<td>3.92338</td>
<td>-7.84081</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(a_{11})</td>
<td>6.50512</td>
<td>1.03153</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(a_{12})</td>
<td>-6.60973</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(a_{13})</td>
<td>1.78158</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

With Pari/GP the precision reached at $y = 1$ as a function of $n$ is given in table 4; see Appendix for details. Since Pari/GP gives a reference solution,
Figure 1: Top figure: Here $\Sigma = \sqrt{0.5}$ and the call for this volatility is the middle curve. The call computed by the reduced basis method overlaps perfectly - even with $I = 5$ here - the call computed analytically in both cases $\sigma = \sqrt{0.9}$ (the highest curve and the plus marks) and $\sigma = \sqrt{0.3}$ (the lowest curve and the cross marks). Bottom left figure: errors with $I = 10$ (difference between the analytical Black-Scholes formula and this method) in the interval $y \in (0.5, 1.5)$ shown to be below the 0.1% accepted standard. Bottom right figure: The highest curve is $L^1$ error versus ten times the time to maturity $\tau$. The lowest curve is the $L^1$ error at $\tau = T$ versus the number of basis functions $I$. 
Table 2: Solution of (7) with C-gmp program with 100 digits

<table>
<thead>
<tr>
<th>$a_i \backslash I$</th>
<th>13</th>
<th>11</th>
<th>9</th>
<th>7</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>-0.0219085</td>
<td>-0.0213871</td>
<td>-0.0232148</td>
<td>-0.0253716</td>
<td>-0.028902</td>
</tr>
<tr>
<td>$a_2$</td>
<td>-0.236885</td>
<td>-0.247962</td>
<td>-0.198283</td>
<td>-0.150316</td>
<td>-0.107021</td>
</tr>
<tr>
<td>$a_3$</td>
<td>1.14373</td>
<td>1.25403</td>
<td>0.691328</td>
<td>0.273059</td>
<td>0.0664526</td>
</tr>
<tr>
<td>$a_4$</td>
<td>-5.15269</td>
<td>-5.87781</td>
<td>-2.42339</td>
<td>-0.548131</td>
<td>-0.0573113</td>
</tr>
<tr>
<td>$a_5$</td>
<td>14.7611</td>
<td>18.1975</td>
<td>5.43653</td>
<td>0.634395</td>
<td>0.0183907</td>
</tr>
<tr>
<td>$a_6$</td>
<td>-26.1075</td>
<td>-37.5513</td>
<td>-7.69479</td>
<td>-0.390463</td>
<td></td>
</tr>
<tr>
<td>$a_7$</td>
<td>26.3364</td>
<td>51.7575</td>
<td>6.6247</td>
<td>0.0984288</td>
<td></td>
</tr>
<tr>
<td>$a_8$</td>
<td>-11.74</td>
<td>-46.9311</td>
<td>-3.16578</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_9$</td>
<td>1.22566</td>
<td>26.7835</td>
<td>0.642903</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_{10}$</td>
<td>-7.48962</td>
<td>-8.68536</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_{11}$</td>
<td>14.6487</td>
<td>1.21389</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_{12}$</td>
<td>-9.77278</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_{13}$</td>
<td>2.30447</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Solution of (7) with Pari/GP with 400 digits

<table>
<thead>
<tr>
<th>$a_i \backslash I$</th>
<th>13</th>
<th>11</th>
<th>9</th>
<th>7</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>-0.02066914</td>
<td>-0.021770946</td>
<td>-0.023255599</td>
<td>-0.025412096</td>
<td>-0.028946425</td>
</tr>
<tr>
<td>$a_2$</td>
<td>-0.31070357</td>
<td>-0.25354825</td>
<td>-0.200472013</td>
<td>-0.151746073</td>
<td>-0.107848484</td>
</tr>
<tr>
<td>$a_3$</td>
<td>2.57881850</td>
<td>1.441042637</td>
<td>0.707556845</td>
<td>0.27952402</td>
<td>0.068508564</td>
</tr>
<tr>
<td>$a_4$</td>
<td>-19.5656666</td>
<td>-7.745999051</td>
<td>-2.488138433</td>
<td>-0.562211076</td>
<td>-0.058919414</td>
</tr>
<tr>
<td>$a_5$</td>
<td>101.295195</td>
<td>27.78591074</td>
<td>5.593182295</td>
<td>0.651966407</td>
<td>0.018968232</td>
</tr>
<tr>
<td>$a_6$</td>
<td>-361.806032</td>
<td>-66.75122227</td>
<td>-7.925926655</td>
<td>-0.401672718</td>
<td></td>
</tr>
<tr>
<td>$a_7$</td>
<td>904.938023</td>
<td>107.5659839</td>
<td>6.830376803</td>
<td>0.101306325</td>
<td></td>
</tr>
<tr>
<td>$a_8$</td>
<td>-1595.62124</td>
<td>-114.635667</td>
<td>-3.264830896</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_9$</td>
<td>1972.61340</td>
<td>77.44780718</td>
<td>0.663260035</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_{10}$</td>
<td>-1672.4445</td>
<td>-30.02794459</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_{11}$</td>
<td>925.886937</td>
<td>5.087458879</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_{12}$</td>
<td>-301.393122</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_{13}$</td>
<td>43.7413842</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Precision $\epsilon$ versus $n$, the number of functions in the basis

<table>
<thead>
<tr>
<th>$n$</th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>7</th>
<th>9</th>
<th>11</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$</td>
<td>0.012</td>
<td>3.5 $10^{-6}$</td>
<td>1.4 $10^{-5}$</td>
<td>6.5 $10^{-6}$</td>
<td>4.2 $10^{-6}$</td>
<td>3.3 $10^{-6}$</td>
<td>2.3 $10^{-6}$</td>
</tr>
</tbody>
</table>
the conclusion is that the C-program is usable up to \( I = 9 \) but not beyond; however in Pari/GP we have inverted the matrix of the linear system and this could be unstable. If this is so then GNU-MP being the reference solution the C-program could be trusted up to \( I = 13 \). This point is important because GNU-MP is much slower that the C-program. Nevertheless for obvious stability reason, this reduced basis method cannot be used for large values of \( I \) but fortunately the precision is very high even for small values of \( I \).

1.5 Numerical Complexity

It is worth pointing out that this method is as fast if not faster than the fastest of all to compute a call for a general volatility \( \sigma \).

If (9) is used, then the \( J + 2 \) matrices are computed once and for all at the cost of \( O(I^2) \) operations each. The linear systems have a rather low condition number \( (10^{-7} \) for \( I = 8 \)) so it is best solved by SVD but it can also be solved by a conjugate gradient method preconditioned by its own diagonal. Then less than \( I/2 \) iterations seems to be sufficient. The right hand sides are computed at a cost of \( O(IJ) \) operations.

Finally if \( M \) is the number of time steps the total number of operations is \( C = (J + 2)(1 + M) + IMJ + \frac{1}{2}I^3M \). Typically \( M = 20, I = 8 \) so \( C \approx 50000 \).

Compared with a finite difference method with \( M' = 100 \) time steps and \( N = 200 \) mesh points for which the complexity is \( 3MN = 60000 \) at best.

In practice the programs confirm that this method is indeed much faster than the best finite difference/finite element solution of the PDE. All parameters are adjusted so that the precision is 0.5% or less. For instance an implicit time scheme with \( M' = 100 \) time steps and \( 200 \) \( y \)-grid points where the linear system at each iteration is solved by a Gauss factorisation – required because \( \sigma \) is time dependent – takes 0.0014" (measured by computing 1000 computations and divide the resulting CPU time by 1000 ).

On a reduced basis of 10 vectors the same computation takes 0.00018" with analytical formulae for \( B_{ij} \) and \( F_j \), a number to be multiplied by \( J \) if (9) is used.

Computations have been done on an Intel Core duo 1.86 GHz using only one of the two cores. The conclusion is that

*The method is very competitive for non constant volatilities using up to 10 terms in (9) or equivalent.*

For illustration figure 2 shows the same call computed by finite differences \( 100 \times 200 \) and on a reduced basis of 10 vectors for \( \sigma^2 = 0.7 + 0.1 \text{e}^{-2 \ln^2 y} \).
Figure 2: Comparison for $y \in (0.5, 1.5)$ between a computation of a call on a reduced basis of 10 vectors with a $y$ dependent volatility $\sigma^2 = 0.7 + 0.1e^{-2\ln^2 y}$ (upper solid line) and the same computed by a finite difference method with 100 time steps and 200 $y$-grid points (the plus symbols). The analytical results for $\sigma^2 = 0.7$ is given for comparison (lower curve).
2 The 2D case

Consider an European basket call \( C \) made on two assets \( S_1, S_2 \) with strike \( K \) and maturity \( T \):

\[
\partial_t C + \frac{1}{2} \sum_{ij=1,2} q_{ij} \sigma_i \sigma_j S_i S_j \partial S_i \partial S_j C + \mu \cdot \nabla C - rC = 0, \quad C(T) = (S_1 + S_2 - K)^+
\]

for all \((S_1, S_2, t) \in \mathbb{R}^2 \times (0, T)\); as usual \( r \) is the interest rate and \( \mu = (\mu_1 S_1, \mu_2 S_2) \) accounts also for dividends yielded by \( S_1 \) and \( S_2 \). The \( \nabla \) is the gradient with respect to \( S \), the "\( \cdot \)" is the vector product and finally \( \sigma_1, \sigma_2, q \) are the volatilities associated with \( S_1, S_2 \) and the correlation \( q_{ij} \) is 1 if \( i = j \) and \( q \) if \( i \neq j \).

2.1 The Reduced Problem

Define \( y_i = e^{\mu \tau \frac{S_i}{\mathcal{R}}} \) and \( \tau = T - t \). Let \( u = \frac{1}{K} e^{\tau C} \), then the PDE becomes

\[
\partial_t u - \Xi : \nabla \nabla u = 0, \quad u(0) = (y_1 + y_2 - 1)^+
\]

with the notation \( A : B = \sum_{i,j} A_{ij} B_{ij} \) and the volatility matrix

\[
\Xi = \frac{1}{2} \begin{pmatrix}
\sigma_1^2 & \sigma_1 \sigma_2 \\
\sigma_1 \sigma_2 & \sigma_2^2
\end{pmatrix}
\]

When \( \sigma_1 = \sigma_2 \) and \( q = 1 \) the problem reduces to a one dimensional Black-Scholes equation in the variable \( z = y_1 + y_2 \) and when \( \sigma_1 \) is constant then a closed form solution is available because (3) in \( z \) holds.

2.2 A Decomposition for the 2D case

It is convenient to introduce the vector index \( i = (i_1, i_2)^T \). Just as in the one dimensional case we approximate \( u \) by

\[
\hat{u}(y_1, y_2, \tau) := u_{\Xi}(y_1, y_2, \tau) + \sum_{i_1, i_2=1..I} a_i(\tau) w^i(y_1, y_2).
\]

with \( u_{\Xi} \) solution of (10) with volatility matrix \( \bar{\Xi} \) with parameters \( \Sigma_1, \Sigma_2, Q \) in (11) and with

\[
w^i = e^{-\alpha_{i_1} \ln y_1 - \alpha_{i_2} \ln y_2} \sqrt{y_1 y_2}
\]

Notice that

\[
\partial_{y_1} w^i = -(2\alpha_{i_1} \ln y_1 + \frac{1}{2} - \frac{\alpha_{i_2}}{y_1}), \quad \partial_{y_1 y_1} w^i = (4\alpha_{i_1}^2 \ln^2 y_1 - 2\alpha_{i_1} - \frac{1}{4}) \frac{w^i}{y_1^2}
\]

Placing \( \hat{u} \) in (10) gives

\[
\sum_i w^i \hat{a}_i - \Xi = \begin{pmatrix}
\frac{4\alpha_{i_1}^2 \ln^2 y_1 - 2\alpha_{i_1} - \frac{1}{4}}{y_1^2} & \frac{(2\alpha_{i_1} \ln y_1 + \frac{1}{2})(2\alpha_{i_2} \ln y_2 + \frac{1}{2})}{y_1 y_2} \\
\frac{2\alpha_{i_1} \ln y_1 + \frac{1}{2}}{y_1^2} & \frac{4\alpha_{i_2}^2 \ln^2 y_2 - 2\alpha_{i_2} - \frac{1}{4}}{y_2^2}
\end{pmatrix} a_i
\]
For a general constant volatility matrix we construct a

\[ (\Xi - \Xi) : \nabla \nabla u \Sigma \]  

(13)

Note that

\[ g_i(y_1, y_2) := \Xi \left( \begin{array}{c} \frac{4a^{i_1}_1 \ln^2 y_1 - 2a^{i_1}_1 - \frac{1}{8}}{y_1^2} \ \frac{(2a^{i_1}_{i_2} \ln y_1 + \frac{1}{2})(2a^{i_2}_{i_1} \ln y_2 + \frac{1}{2})}{y_1 y_2} \\ \frac{4a^{i_2}_2 \ln^2 y_2 - 2a^{i_2}_2 - \frac{1}{8}}{y_2^2} \end{array} \right) \]

\[ = \sigma_1^2(2\alpha_{i_1}^2 \ln^2 y_1 - \alpha_{i_1} - \frac{1}{8}) + \sigma_1 \sigma_2 q(2\alpha_{i_1} \ln y_1 + \frac{1}{2})(2\alpha_{i_2} \ln y_2 + \frac{1}{2}) + \sigma_2^2(2\alpha_{i_2}^2 \ln^2 y_2 - \alpha_{i_2} - \frac{1}{8}) \]  

(14)

A Galerkin method amounts to multiply this equation by \( w^j \) and integrate it over \( \mathbb{R}^2 \). It gives:

\[ M_{i,j}a_j + B_{i,j}a_j = F_i \]

\[ M_{i,j} = \int_{\mathbb{R}^2} w^i w^j = \frac{\pi e^{\frac{1}{\beta_1} + \frac{1}{\beta_2}}}{\sqrt{\beta_1 \beta_2}} \]

\[ B_{i,j} = \int_{\mathbb{R}^2} g_i(y_1, y_2) w^i w^j \]

\[ F_i = \int_{\mathbb{R}^2} u_i(\Xi - \Xi) : \nabla \nabla u \Sigma \]  

(15)

2.2.1 Constant isotropic Volatilities

Let \( \beta_k = \alpha_{i_k} + \alpha_{j_k}, k = 1, 2 \); when \( \sigma_i \) are constant,

\[ B_{i,j} = \frac{\pi e^{\frac{1}{\beta_1} + \frac{1}{\beta_2}}}{\sqrt{\beta_1 \beta_2}} [\sigma_1^2 \left( \frac{2\alpha_{i_1}^2}{\beta_1} + \frac{1}{2\beta_1} \right) - \alpha_{i_1} - \frac{1}{8}] + \sigma_1 \sigma_2 q \left( \frac{\alpha_{i_1}}{\beta_1} + \frac{1}{2} \right) \left( \frac{2\alpha_{i_2}}{\beta_2} + \frac{1}{2} \right) + \sigma_2^2 \left( \frac{2\alpha_{i_2}^2}{\beta_2} + \frac{1}{2\beta_2} \right) - \alpha_{i_2} - \frac{1}{8} \]  

(16)

Let \( \sigma := \sigma_1 = \sigma_2 \); then \( u \Sigma(y_1, y_2) \) is given by (9) and

\[ F_i = dt \int_{\mathbb{R}^2} e^{\frac{\Sigma^2}{8}} \frac{\Sigma^2}{4\pi \Sigma} \]

\[ \left( (\frac{\Sigma^2}{\Sigma^2} y_1^2 + (\frac{\Sigma^2}{\Sigma^2} y_2^2) + 2y_1 y_2(\sigma_1 \sigma_2 q - \Sigma^2)) (y_1 + y_2)^{-\frac{1}{2}} \right) w^i \]  

(17)

A test was done with \( \sigma_1 = \sigma_2 = 0.3, q = 0.5, \Sigma = 0.2, T = 2 \) and with \( I = 5, \delta t = 0.1 \); \( F_i \) has been computed by quadrature. . The results are shown on figure 3

2.2.2 Constant General Volatilities

For a general constant volatility matrix we construct a \( u \Sigma \) with the same \( \sigma_i, i = 1, 2 \) but \( \bar{q} \) instead of \( q \); then the right hand side of the linear system is

\[ F_i = 2dt \int_{\mathbb{R}^2} (\bar{q} - q) \sigma_1 \sigma_2 y_1 y_2(\partial_{y_1 y_2} u \Sigma) w^i dy_1 dy_2 \]  

(18)
Figure 3: Pointwise errors between the analytical solution and the reduced basis method for a European call computed when $\sigma_1 = \sigma_2 = 0.3$, $I = 5$, $q = 0.5$. The maximum error is 0.0035.
Figure 4: Pointwise errors between the analytical solution and the reduced basis method for a European call computed when $\sigma_1 = \sigma_2 = 0.3$, $I = 5$, $q = 0.5$. The maximum error is 0.0035.
Conclusion

The reduced basis method studied here uses basis vectors well adapted to the problem; consequently it is very fast because a small number of vectors are sufficient to reach a high precision. However the linear systems are ill conditioned so we have done a detailed study of the numerical errors by using multiple precision arithmetic tools. We are now confident that the method can be used on baskets with very few basis vectors as well. A very simple numerical test has been presented in the unrealistic situation where the volatilities of both components of the basket are equal. When they are not equal the method works but the analytical computation of the translation $u_\Sigma$ is more complex. Finally we intend to use the method on the two dimensional integro-differential equations arising from jump diffusion models and we shall report on these cases later in the year.

References


3 Appendix: Multiprecision solver

To push the limit of the method the linear system arising from ODE was solved with the multi precision software Pari/GP. Recall that

\[ M \dot{a} + B a = F(t), \quad t \in (0, T), \quad a(0) = 0, \]

with

\[ M_{ij} = \sqrt{\frac{\pi}{i + j}}, \quad B_{ij} = \sigma^2 M_{ij} \left( \frac{i j}{i + j} + \frac{1}{8} \right), \quad F_j = -e^{-\frac{\sigma^2}{2} \left( \frac{\Sigma^2}{2} - \sigma^2 \right)} \frac{2}{2\sqrt{2j\Sigma^2t + 1}}. \]

With the Crank-Nicolson finite difference scheme the system becomes:

\[ a^0 = 0, \quad (M + \frac{\delta t}{2} B)a^{m+1} = (M - \frac{\delta t}{2} B)a^m + \delta t F, \quad m = 0, 1, \ldots, \frac{T}{\delta t}, \]

The following Pari/GP script implements the scheme with \( \sigma^2 = 0.09, \Sigma^2 = 0.25 \) 20000 time steps (other with the time-finite difference error dominates) to reach \( T=2 \). There are \( n = 7 \) basis functions in the example below and the solution is compared with the Black-Scholes formula at \( y = 1 \).

```pari
def callex(y,t,ss)= (y*(2-erfc(log(y)/sqrt(ss*2*t)+sqrt(ss*t/8)))-2+erfc(log(y)/sqrt(ss*2*t)-sqrt(ss*t/8)))/2;
S=0.25;s=0.09;m=20000;dt=2/m;n=7;
\p2000
A= matrix(n,n,i,j,(1+0.5*dt*s*(i*j/(i+j)+0.125))*sqrt(Pi/(i+j)));
M = matrix(n,n,i,j,sqrt(Pi/(i+j)));
A1=1/A;
C = A1*M;
F(t) = vector(n,i,0.5*dt*exp(-S*t/8)*(s-S)/(2*sqrt(1+2*i*S*t)))~;
G = vector(n,i,0)~;
for(k=1,m,G=-G+2*C*G+2*A1*F(2*k/m));
callex(y,t,ss)=y*(2-erfc(log(y)/sqrt(ss*2*t)+sqrt(ss*t/8)))-2+erfc(log(y)/sqrt(ss*2*t)-sqrt(ss*t/8)))/2;
y=1;t=2;
c=callex(y,t,S);
for(k=1,n,c=c+G[k]*sqrt(y)*exp(-k*log(y)*log(y)));
\p10;print(c-callex(y,t,s));
```

Computations are done with 2000 digits (at \( n = 13 \), even 1000 digits is not enough and gives a a wrong answer); it takes 4 seconds on a Mac Pro 2.66MHz. The precision reached as a function of \( n \) is given in table 4. Beyond \( n = 13 \) the computation is not possible with the required precision due to memory limitation.