A High Order Method for Pricing of Financial Derivatives using Radial Basis Function generated Finite Differences

Slobodan Milovanović and Lina von Sydow

Department of Information Technology
Uppsala University
Sweden

Abstract

In this paper, we consider the numerical pricing of financial derivatives using Radial Basis Function generated Finite Differences in space. Such discretization methods have the advantage of not requiring Cartesian grids. Instead, the nodes can be placed with higher density in areas where there is a need for higher accuracy. Still, the discretization matrix is fairly sparse. As a model problem, we consider pricing of European options in 2D. Since such options have a discontinuity in the first derivative of the payoff function which prohibits high order convergence, we smooth this function using an established technique for Cartesian grids. Numerical experiments show that we acquire a fourth order scheme in space, both for the uniform and the nonuniform node layouts that we use. The high order method with the nonuniform node layout achieves very high accuracy with relatively few nodes. This renders the potential for solving pricing problems in higher spatial dimensions since the computational memory and time demand become much smaller with this method compared to standard techniques.

Keywords: Pricing of Financial Derivatives; Radial Basis Function generated Finite Differences; High Order Methods; Smoothing of Initial Data.

1 Introduction

In this paper, we are concerned with the numerical pricing of financial derivatives. A financial derivative is a contract whose value depends on an underlying asset such as a stock, an interest rate, or a commodity. The trading in financial

derivatives has increased tremendously during the last decades, mostly due to the possibility to hedge positions in the underlying asset. Another important feature of financial derivatives is the potential for leverage, since a small movement in the underlying asset can cause a large movement in the value of the financial derivative.

Due to the large traded volume of financial derivatives, efficient and accurate pricing of such contracts is of utmost importance. In most cases, there is no analytical formula available, and it becomes necessary to use a numerical method to compute the prices of the contracts. When a financial derivative depends on several underlying assets, the problem becomes multi-dimensional. Traditionally, the only way to price such financial derivatives is to use Monte Carlo methods for a stochastic differential equation (SDE) formulation of the problem. However, due to their arguably slow convergence, considerable efforts in the research community have been devoted to deriving efficient methods for a partial differential equation (PDE) formulation of the pricing problem. The main problem with these methods is the so-called curse of dimensionality — the number of degrees of freedom in the problem grows exponentially in the number of dimensions.

Numerical methods for the PDE formulation include adaptive Finite Differences (FD) [14, 11, 10, 15, 24], Alternating Direction Implicit (ADI) schemes [7, 5], Radial Basis Function (RBF) approximation [16, 9], Radial Basis Function Partition of Unity (RBF-PU) method [19, 21, 20], and Radial Basis Function generated Finite Differences (RBF-FD) [13, 12]. In [23] and [25] several methods for pricing of options are implemented and evaluated.

As a numerical example, we consider pricing of a European two-dimensional option. An option is a financial derivative which gives the holder the right, but not the obligation, to buy (for call options) or sell (for put options) an underlying asset at a specified strike price K at or before the time of maturity T. The method that we employ is RBF-FD. The main idea behind it is to combine the desirable features from FD (sparsity of the discretization matrices — as opposed to RBF) and RBF (meshfree — as opposed to FD). Such methods have the potential to be of high order, depending on the number of nodes used in the discretization stencil. However, for many option pricing problems, the payoff function has a discontinuity in the function itself or its derivatives, which limits the order of convergence obtained in numerical simulations. For this reason, we smooth the payoff function according to [8], before employing the numerical method. This smoothing increases the order of convergence to the expected one from the discretization that is used.

In Section 2 we define the discretization in space and time, while Section 3 is devoted to the model problems that we solve, as well as node layouts, stencils, boundary conditions, and smoothing of the initial data. Finally, the results are

presented in Section 4, and conclusions are drawn in Section 5.

2 Numerical method

We consider pricing of financial derivatives where the problem can be formulated as a PDE in D spatial dimensions and time

$$\begin{aligned} \frac{\partial u}{\partial t} + \mathcal{L}u &= 0, \\ u(s_1, \ldots, s_D, 0) &= g(s_1, \ldots, s_D), \\ s_i \geq 0,\ i = 1, \ldots, D; &\quad 0 \leq t \leq T. \end{aligned} \tag{2.1}$$

Here the solution $u(s_1, \ldots, s_D, t)$ denotes the price of the financial derivative, t denotes time, s_i, the value of the underlying asset with index i, and g the payoff function of the financial derivative. In many pricing problems the original PDE is a final value problem solved backward in time. We consider problems in forward time as in (2.1), i.e., when necessary the problem is transformed into an initial value problem.

In Sections 2.1 and 2.2 we define the spatial and temporal discretization of (2.1), respectively.

2.1 Radial Basis Function generated Finite Differences

In RBF-FD the spatial operator $\mathcal{L}u$ in (2.1) at a location $\mathbf{s}^c = (s_1^c, s_2^c, \ldots, s_D^c)$, is approximated as a linear combination of the solution at the m closest nodes $\mathbf{s}^k$ (possibly including $\mathbf{s}^c$), $k = 1, \ldots, m$

$$\mathcal{L}u|_{\mathbf{s}^c} \approx \sum_{k=1}^{m} w_k u|_{\mathbf{s}^k}. \tag{2.2}$$

The weights w_k are calculated by enforcing (2.2) to be exact for an RBF $\phi(r)$

$$\begin{bmatrix} \phi(\|\mathbf{s}^1 - \mathbf{s}^1\|) & \ldots & \phi(\|\mathbf{s}^1 - \mathbf{s}^m\|) \\ \vdots & \ddots & \vdots \\ \phi(\|\mathbf{s}^m - \mathbf{s}^1\|) & \ldots & \phi(\|\mathbf{s}^m - \mathbf{s}^m\|) \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix} = \begin{bmatrix} \mathcal{L}\phi(\|\mathbf{s} - \mathbf{s}^1\|)|_{\mathbf{s}^c} \\ \vdots \\ \mathcal{L}\phi(\|\mathbf{s} - \mathbf{s}^m\|)|_{\mathbf{s}^c} \end{bmatrix}. \tag{2.3}$$

It is given from RBF interpolation that (2.3) is a nonsingular system, and hence a unique set of weights w_k, $k = 1, \ldots, m$ can be computed.

Typical choices of RBFs are listed in Table 1. For the first four examples, the parameter $\varepsilon \in \mathbb{R}$ is the shape parameter of the RBF. For polyharmonic splines (PHSs), the parameter $q \in \mathbb{N}$.

In this paper, we follow [4], [1], and [12] and use PHSs as basis functions together with polynomials of degree p in the interpolation. With that approach, the polynomial degree (instead of the RBF) controls the rate of convergence,

while the RBFs contribute to reduction of approximation errors and are necessary in order to have a stable approximation.

$\phi(r)$	
Gaussian	$e^{-(\varepsilon r)^2}$
Inverse quadratic	$1/(1+(\varepsilon r)^2)$
Multiquadric	$\sqrt{1+(\varepsilon r)^2}$
Inverse multiquadric	$1/\sqrt{1+(\varepsilon r)^2}$
Polyharmonic splines	r^{2q-1}

Table 1: A list of commonly used RBFs $\phi(r)$.

In (2.4), we augment (2.3) with monomials of degree one

$$\begin{bmatrix} & & & 1 & s_1^1 & \dots & s_D^1 \\ & B & & 1 & \vdots & & \vdots \\ & & & 1 & s_1^m & \dots & s_D^m \\ 1 & \dots & 1 & 0 & 0 & \dots & 0 \\ s_1^1 & \dots & s_1^m & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ s_D^1 & \dots & s_D^m & 0 & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_m \\ \gamma_0 \\ \gamma_1 \\ \vdots \\ \gamma_D \end{bmatrix} = \begin{bmatrix} \mathcal{L}\phi(\|\mathbf{s}-\mathbf{s}^1\|)|_{\mathbf{s}^c} \\ \vdots \\ \mathcal{L}\phi(\|\mathbf{s}-\mathbf{s}^m\|)|_{\mathbf{s}^c} \\ \mathcal{L}1|_{\mathbf{s}^c} \\ \mathcal{L}s_1|_{\mathbf{s}^c} \\ \vdots \\ \mathcal{L}s_D|_{\mathbf{s}^c} \end{bmatrix}, \tag{2.4}$$

where B is the coefficient-matrix in (2.3).

Now, we place N computational nodes $\mathbf{s}_i^c$, $i = 1, \dots, N$ at the locations where we want to approximate the solution. The weights for each computational node from solving (2.4) are assembled row-wise into the sparse differentiation matrix $W \in \mathbb{R}^{N \times N}$, with m nonzero elements per row. This leads to the following semi-discretization of (2.1)

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\bar{u}(t) + W\bar{u}(t) &= \bar{0}, \\ \bar{u}(0) &= \bar{g}, \end{aligned} \tag{2.5}$$

where $\bar{u}(t) \in \mathbb{R}^{N \times 1}$ is the vector of unknowns at time t, with approximations of u in the computational nodes $\mathbf{s}_i^c$, $i = 1, \dots, N$, $\bar{0} \in \mathbb{R}^{N \times 1}$ is a vector with only zeros, and $\bar{g} \in \mathbb{R}^{N \times 1}$ is the vector with the function g evaluated in the computational nodes $\mathbf{s}_i^c$, $i = 1, \dots, N$.

Equation (2.5) forms a system of linear ordinary differential equations (ODEs) in time. In the next section, we describe how to solve it.

2.2 Temporal discretization

For the time discretization of (2.5), we use the Backward Differentiation Formula of order two (BDF2) [6]. This time-stepping scheme requires the solution at two

previous time steps, and we therefore employ Backward Euler (BDF1) for the first time step. It is convenient to have the same coefficient matrix in all time steps, so we use non-equidistant time steps as described in [9] and later used in e.g., [13, 12]. This is accomplished by discretizing the time interval with M steps of length $\Delta t^\ell = t^\ell - t^{\ell-1}$, where $\ell = 1, \ldots, M$. We define $\omega_\ell = \Delta t^\ell / \Delta t^{\ell-1}$ for $\ell = 2, \ldots, M$ and arrive at

$$\bar{u}^1 - \bar{u}^0 = \Delta t^1 W \bar{u}^1, \tag{2.6}$$

$$\bar{u}^\ell - \beta_1^\ell \bar{u}^{\ell-1} + \beta_2^\ell \bar{u}^{\ell-2} = \beta_0^\ell W \bar{u}^\ell, \quad \ell = 2, \ldots, M, \tag{2.7}$$

where

$$\beta_0^\ell = \Delta t^\ell \frac{1+\omega_\ell}{1+2\omega_\ell}, \quad \beta_1^\ell = \frac{(1+\omega_\ell)^2}{1+2\omega_\ell}, \quad \beta_2^\ell = \frac{\omega_\ell^2}{1+2\omega_\ell}. \tag{2.8}$$

We compute the values for ω_ℓ using the recursive condition $\beta_0^\ell = \beta_0^{\ell-1}$, which keeps the coefficient matrix constant throughout all time steps. Since our time interval has the length T, we chose the initial time step length Δt^1 from

$$\sum_{\ell=1}^{M} \Delta t^\ell = T = \Delta t^1 (1 + \sum_{\ell=2}^{M} \prod_{\ell'=2}^{\ell} \omega^\ell). \tag{2.9}$$

Finally, we start the time integration by setting $\bar{u}^0 = \bar{g}$.

From the temporal discretization we get the following linear system of equations to solve in each time step

$$A\bar{u}^\ell = \bar{b}^\ell, \tag{2.10}$$

where $A = I - \Delta t^1 W$, Δt^1 is given by (2.9), $\bar{b}^\ell = \beta_1^\ell \bar{u}^{\ell-1} - \beta_2^\ell \bar{u}^{\ell-2}$, $\ell = 2, \ldots, M$, and $\bar{b}^1 = \bar{u}^0 = \bar{g}$.

3 Model problem

As a model problem we consider a European call option issued on two underlying assets s_1 and s_2

$$\frac{\partial u}{\partial t} + \mathcal{L}u = 0, \tag{3.11}$$
$$s_1 \geq 0, \ s_2 \geq 0, \quad 0 \leq t \leq T,$$

with

$$\mathcal{L}u = \frac{1}{2}\Big(\sigma_1^2 s_1^2 \frac{\partial^2 u}{\partial s_1^2} + \sigma_2^2 s_2^2 \frac{\partial^2 u}{\partial s_2^2}\Big) + \rho\sigma_1\sigma_2 s_1 s_2 \frac{\partial^2 u}{\partial s_1 \partial s_2} + r\Big(s_1 \frac{\partial u}{\partial s_1} + s_2 \frac{\partial u}{\partial s_2}\Big) - ru, \tag{3.12}$$

and

$$u(s_1, s_2, T) = g(s_1, s_2) = \left(\frac{1}{2}(s_1 + s_2) - K\right)^+. \tag{3.13}$$

Here $(f(x))^+ = \max(f(x), 0)$, r denotes the risk-free interest rate in the market, σ_i the volatility of asset i, and ρ the correlation between the assets. As a close-field boundary condition in $s_1 = s_2 = 0$ we set

$$u(0, 0, t) = 0, \quad 0 \leq t \leq T, \tag{3.14}$$

and as a far-field boundary condition we set

$$u(s_1, s_2, t) = \left(\frac{1}{2}(s_1 + s_2) - Ke^{-rt}\right), \quad 0 \leq t \leq T, \tag{3.15}$$

for s_1 and s_2 large enough. The parameters used are given in Table 2.

r	0.03
σ_1	0.15
σ_2	0.15
$\rho_{1,2}$	0.5
K	1
T	0.2

Table 2: Parameters used in the model problem

Equation (3.11) is a PDE that should be solved backward in time. To apply the time-stepping scheme in Section 2.2, we therefore transform (3.11) into a problem that is solved forward in time.

3.1 Node layout, stencils, and boundary conditions

We consider both a uniform and a nonuniform node layout, presented in Figure 1. Unlike classical grid-based methods (e.g., standard FD methods) we do not need to use a rectangular domain. Instead, we only use the lower-triangular half of the rectangle which reduces the number of computational nodes by a factor of two, and hence the computational complexity significantly.

The reason for introducing a nonuniform node layout is that we can cluster nodes where we are most interested in having an accurate solution. In general, we are most interested in having an accurate solution in the neighborhood of $s_1 + s_2 = 2K$, which is also where the truncation error is largest due to large derivatives in the solution from the discontinuity in the first derivative of the payoff function.

We start to present a nonuniform node distribution in 1D that is generated as introduced in [7] and later used for RBF-FD and option pricing in [13].

Consider N_1 equidistant nodes $x_1^{(1)} < \ldots < x_1^{(i)} < \ldots < x_1^{(N_1)}$ constructed by

$$x_1^{(i)} = \operatorname{arcsinh}\left(-\frac{K}{c}\right) + (i-1)\Delta x, \quad i = 1, \ldots, N_1, \tag{3.16}$$

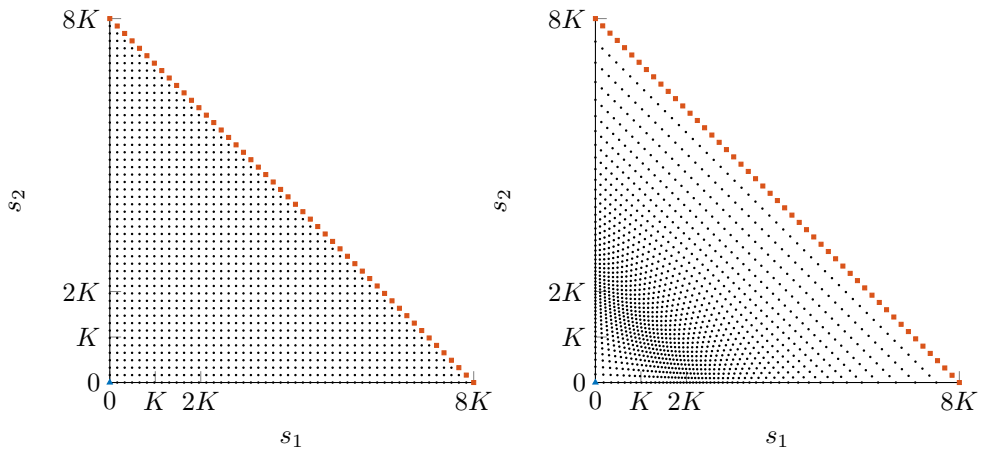
where c is a positive real constant which specifies how dense the node distribution becomes around the strike price K,

$$\Delta x = \frac{1}{N_1}\left[\operatorname{arcsinh}\left(\frac{s_{\max} - K}{c}\right) - \operatorname{arcsinh}\left(-\frac{K}{c}\right)\right],$$

and $s_{\max}$ denotes the far-field boundary. Then, the nonuniform node distribution s_1 is generated pointwise as

$$s_1^{(i)} = K + c \cdot \sinh(x^{(i)}), \quad i = 1, \ldots, N_1. \tag{3.17}$$

The nonuniform node layout is generated by using the one-dimensional node layouts from (3.16) and (3.17), along the axes s_1 and s_2, and then uniformly placing the internal points in the diagonal direction. The number of nodes along each diagonal is increased by one for each diagonal. The far-field boundary is located at $s_1 + s_2 = s_{\max} = 8K$. The density tuning parameter used in Figure 1 for the nonuniform node layout and in the numerical experiments presented in Section 4, is $c = 0.8$. It should be noted that a too small value of c eventually leads to an ill-conditioned problem.



(a) Uniform node layout.

(b) Nonuniform node layout.

Figure 1: Uniform and nonuniform computational node layouts in 2D. The boundary conditions are employed in the blue triangle node (the close-field boundary condition) and in the red square nodes (the far-field boundary condition).

We also introduce the notation N_s for the number of nodes along one of the

axes, i.e.,

$$\frac{N_s(N_s+1)}{2} = N. \tag{3.18}$$

The nearest neighbors for constructing the stencils are efficiently determined using the k-D tree algorithm, [2]. In Figure 2 we show examples of stencils at different locations in the domain. The polynomial space is of size

$$\nu = \begin{pmatrix} p+D \\ p \end{pmatrix},$$

which we use to set the size of the stencils to $m = 5\nu$, following [4, 1, 12]. We are aiming for a fourth order scheme and use $p = 4$ and $q = 5$ which gives

$$\nu = \begin{pmatrix} 6 \\ 4 \end{pmatrix} = 15.$$

Hence, we use a stencil size that is $m = 5 \cdot 15 = 75$.

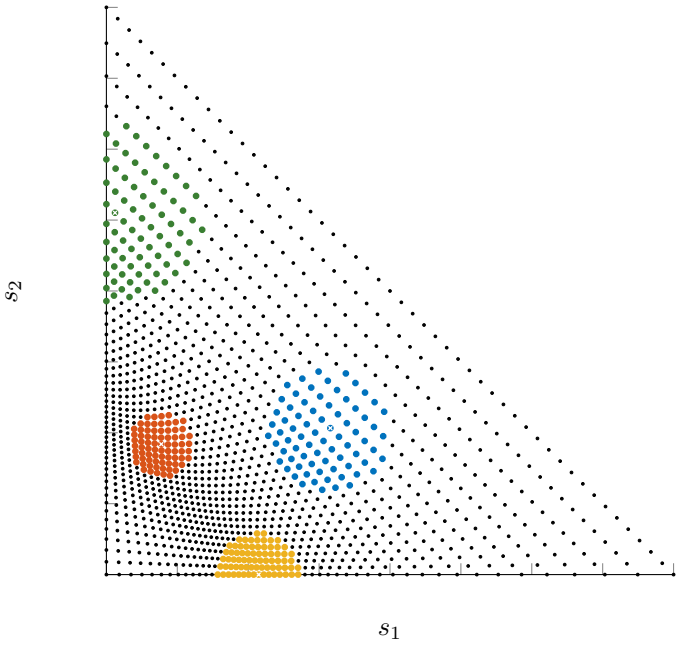


Figure 2: Examples of nearest neighbor based stencils used for approximating the differential operator on a nonuniform node layout. The central node of each displayed stencil is denoted by a white cross mark. All stencils are of the same size $m = 75$.

For the boundary nodes we use different treatments depending on where the node is located. For the node $s_1 = s_2 = 0$ (the blue triangle in Figure 1), we set the close-field boundary condition from (3.14). For the nodes $s_1 + s_2 = 8K$ (the

red squares in Figure 1), we set the far-field boundary conditions from (3.15). For the boundary nodes along the axes, i.e., $s_1 = 0$, $s_2 > 0$ and $s_2 = 0$, $s_1 > 0$ we solve (3.11) using the discretization scheme defined in Section 2. The k-D tree algorithm generates one-sided stencils for those nodes.

3.2 Smoothing of initial data

Since the initial data $g(s_1, s_2)$ in (3.13) has a discontinuity in the first derivative, the obtained spatial order of convergence for a finite difference scheme is limited to two, regardless of the formal order of the scheme. The formal spatial order of the scheme in (2.4) is p, i.e., for $p > 2$ the obtained convergence order is limited by the lack of smoothness in the final condition. In [8], a smoothing of the initial data that recovers the order of convergence to the formal order of the scheme is introduced. This approach has been successfully used for option pricing problems in e.g., [17] and [3].

Since we are aiming for a fourth order scheme, we use a fourth order smoothing operator Φ_4 defined by its Fourier transform

$$\hat{\Phi}_4(\omega) = \left(\frac{\sin(\omega/2)}{\omega/2}\right)^4 \left(1 + \frac{2}{3}\sin^2(\omega/2)\right). \tag{3.19}$$

Using Wolfram Mathematica to compute the inverse Fourier transform of (3.19) gives

$$\begin{aligned}\Phi_4(s) = \frac{1}{72}\Big(&- (s-3)^3 \cdot \operatorname{sgn}(s-3) - (s+3)^3 \cdot \operatorname{sgn}(s+3) \\ &+ 12(s-2)^3 \cdot \operatorname{sgn}(s-2) + 12(s+2)^3 \cdot \operatorname{sgn}(s+2) \\ &- 39(s-1)^3 \cdot \operatorname{sgn}(s-1) - 39(s+1)^3 \cdot \operatorname{sgn}(s+1) \\ &+ 56s^3 \cdot \operatorname{sgn}(s)\Big),\end{aligned} \tag{3.20}$$

where

$$\operatorname{sgn}(x) = \frac{|x|}{x}.$$

Following [8], [17], and [3], we get the smoothed final condition on a uniform node layout as

$$\tilde{g}(s_1, s_2) = \frac{1}{\Delta s^2} \int_{-3\Delta s}^{3\Delta s} \int_{-3\Delta s}^{3\Delta s} \Phi_4\left(\frac{\tilde{s}_1}{\Delta s}\right) \Phi_4\left(\frac{\tilde{s}_2}{\Delta s}\right) g(s_1 - \tilde{s}_1, s_2 - \tilde{s}_2)\, \mathrm{d}\tilde{s}_1\, \mathrm{d}\tilde{s}_2. \tag{3.21}$$

Since $g(s_1, s_2)$ is smooth in a large part of the computational domain, we only need to compute (3.21) in the nodes that are close enough to $s_1 + s_2 = 2K$ to be affected from the smoothing. Also, since the nodes along a diagonal all have the same distance to $s_1 + s_2 = 2K$, we only need to compute one value of $\tilde{g}(s_1, s_2)$ for each diagonal and use that value for all nodes on that diagonal.

The theory in [8] shows that replacing the final condition $g(s_1, s_2)$ with $\tilde{g}(s_1, s_2)$ defined in (3.21) gives a fourth order scheme for Cartesian grids, i.e., the node layout that we here refer to as uniform. Here, we want to use this smoothing also for our nonuniform node layout defined in Section 3.1. This layout can be seen as a slightly skewed Cartesian grid and the nodes are equidistantly distributed along the diagonals. For this node layout we replace Δs in (3.21) with

$$\Delta s_i = \min_{k=1,\ldots,m}^{k\neq c} \|\mathbf{s}_i^c - \mathbf{s}_i^k\|, \quad i = 1, \ldots, N.$$

4 Numerical results

The numerical method described in Section 2 applied to the model problems described in Section 3 is implemented in MATLAB. In all experiments, we start by scaling the original problem such that $s_{\max} = 1$ and time runs forward in the PDE. After the integration, the solution is transformed back to the original problem.

The linear system defined in (2.10) is solved using GMRES [18], with an incomplete LU factorization as the preconditioner using `nofill`. The convergence tolerance for the iterations is set to 10^{-8}, and as the initial condition for each iteration we use the computed solution from the previous time step.

The numerical experiments are performed on a laptop equipped with a 2.3 GHz Intel Core i7 CPU and 16 GB of RAM. The computation of the RBF-FD weights is performed in parallel using the parallel toolbox command `parfor` with four workers.

In Figure 3 we plot the error $\Delta u_{\max}$ as a function of $1/\sqrt{N}$ as well as of CPU-time for the model problem. The error is defined as

$$\Delta u(s_1, s_2) = |u^c(s_1, s_2, 0) - u^*(s_1, s_2, 0)|, \tag{4.22}$$

where u^c is the computed solution and u^* is a reference solution computed with a second order finite difference method on a very fine grid. We use (4.22) to define

$$\Delta u_{\max} = \max_{[s_1,s_2]\in\hat{\Omega}} \Delta u(s_1, s_2), \tag{4.23}$$

where $\hat{\Omega} = \left[\frac{1}{3}K, \frac{5}{3}K\right] \times \left[\frac{1}{3}K, \frac{5}{3}K\right]$. We denote standard second order finite differences [22] by FD. RBF-FD-GS is an RBF-FD method with Gaussian RBFs, stencil size $m = 25$ and a node density dependent shape parameter that is presented in detail in [13]. Abbreviation RBF-FD-PHS is used for the method that is presented in this paper. Moreover, we use designation `smoothed` in the superscript for the computations performed with the smoothing of the initial data, and `uniform` and `nonuniform` to specify the node layouts.

Independent of spatial discretization that is used, we employ BDF2 with $M = N_s$ time steps in all experiments. For the RBF-FD methods N_s is defined in (3.18) and for FD it is defined by $N_s = \sqrt{N}$. With this number of time steps, the temporal discretization error is not visible in the plots.

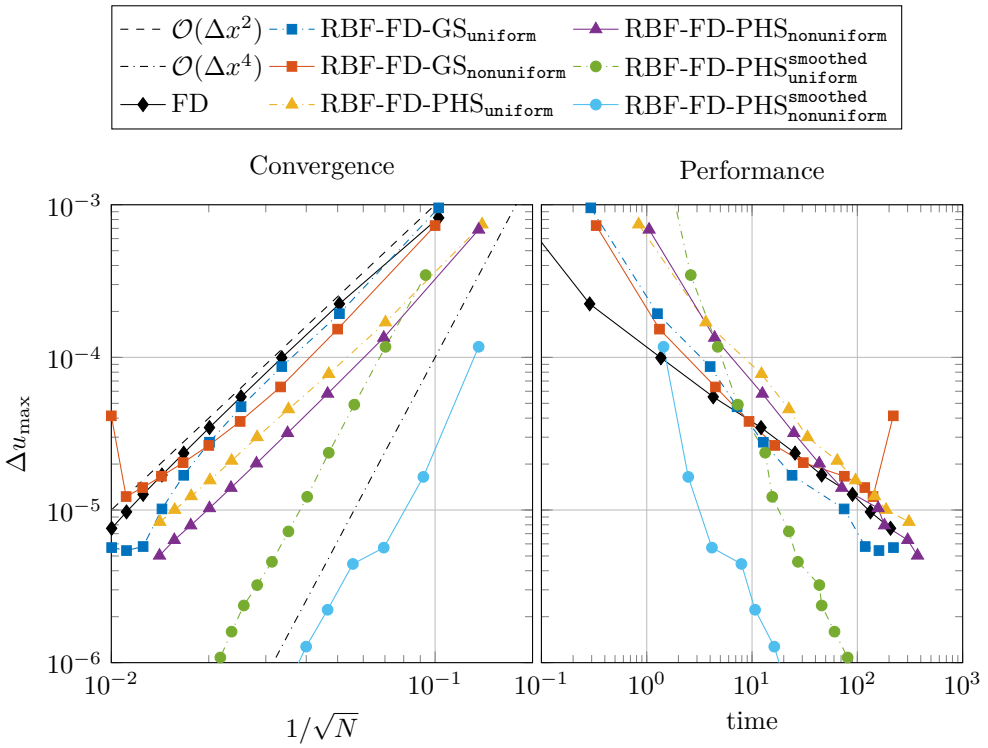


Figure 3: $\Delta u_{\max}$ as a function of $1/\sqrt{N}$ and CPU-time in seconds for the European call option.

In Figure 3 we see that all methods, but the two that are using a smoothed final condition, exhibit second order convergence. Among those five second order methods, RBF-FD with PHS exhibits the smallest error for a given N, and a nonuniform node layout gives a smaller error than the uniform one using the same N. RBF-FD-PHS with a smoothed final condition exhibits fourth order spatial convergence whether we are using a uniform or nonuniform node layout (apart from a small deviation for the nonuniform node layout). When it comes to computational time to reach a certain $\Delta u_{\max}$, FD is competitive for the larger errors displayed. This makes sense since the RBF-FD methods all have to compute the weights w_k, $k = 1, \ldots, m$ before the time-stepping. Moreover, our model problem has a fairly short time to maturity $T = 0.2$. For longer times to maturity, FD does not perform equally well compared to the RBF methods, see [13, 12]. We also establish that the fourth order methods

quickly become superior when it comes to CPU-time to reach a certain $\Delta u_{\max}$. That is especially true for RBF-FD-PHS$^{\text{smoothed}}_{\text{nonuniform}}$. Even though this method has a computational prephase that includes both computation of weights w_k, $k = 1, \ldots, m$ and smoothing of the final condition, the method requires a much smaller CPU-time than the other methods for $\Delta u_{\max} < 10^{-4}$.

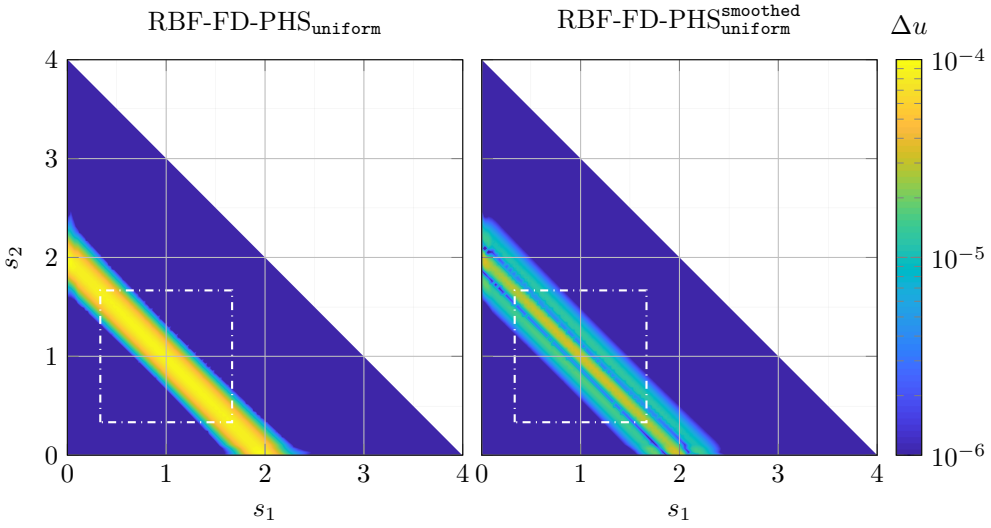


Figure 4: Heat maps of Δu for the European call basket option on uniform node layouts. The boundary of $\hat{\Omega}$ is marked with a white dash-dotted line.

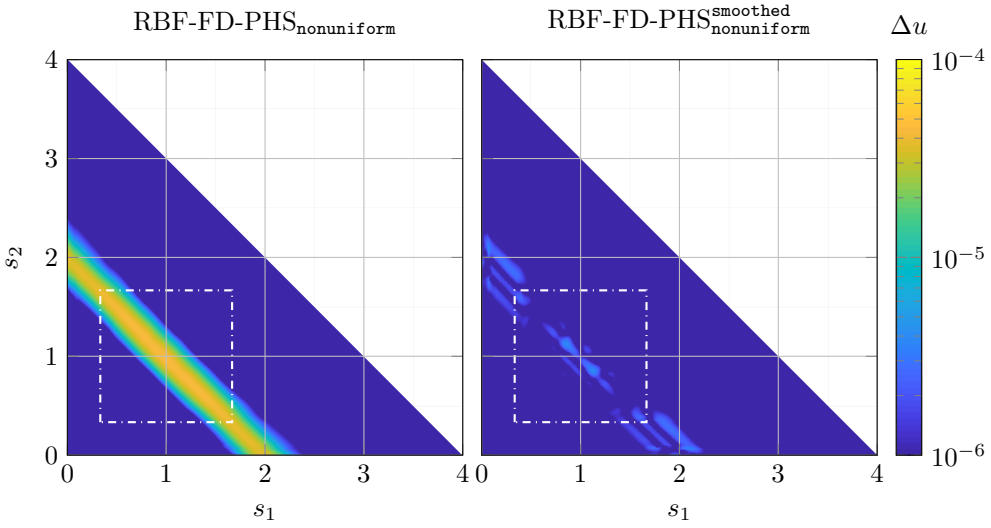


Figure 5: Heat maps of Δu for the European call basket option on nonuniform node layouts. The boundary of $\hat{\Omega}$ is marked with a white dash-dotted line.

In Figures 4–5 we show a heat map of the error Δu defined in (4.22) for the model problem using the method RBF-FD-PHS. In Figure 4 we present the error on the uniform node layout, and in Figure 5 the error on the nonuniform node layout, with $N = 6105$ for both node layouts. To the right in both figures, we have used the smoothed final condition, while the original one is used in the plots on the left. The errors in Figures 4–5 are presented for $0 \leq s_j \leq 4$, $j = 1, 2$, in order to have a better view of the error profile around the smoothed area. The color scale is the same in all four plots.

From Figures 4–5 we conclude that the smoothing of the final condition renders a Δu that has a smaller magnitude compared to the original final condition. Moreover, Δu obtained from the smoothed final condition has three maxima along a line $s_2 = s_1 + const.$ while the corresponding number of maxima is 1 for the nonsmoothed final condition. We also note that the magnitude of Δu is smaller for the nonuniform node layout compared to the uniform one. That is due to the fact that for the nonuniform node layout, the number of nodes is larger in the area where the solution has large derivatives, i.e., around the strike price. We end this section by concluding that in this particular example, $\Delta u_{\max}$ is more than one order smaller using smoothing of the final condition on a nonuniform grid than without the smoothing on a uniform grid for the same number of nodes.

5 Conclusions

In this paper, we have implemented a solver to price financial derivatives based on RBF-FD discretization in space and BDF2 in time. As RBFs we use PHSs, augmented with monomials of up to degree p. The formal order of this spatial discretization is p, however for many pricing problems the lack of smoothness of the initial data limits the actual order obtained in numerical simulations. However, by employing a smoothing technique to the initial data, the formal order of the discretization is retained.

The RBF-FD discretizations have the advantage over standard FD such that the nodes do not have to be organized in a Cartesian grid. On the other hand, the RBF-FD discretizations have the merit that they render sparse differentiation matrices as opposed to global RBF approximations that lead to full matrices. Thus, RBF-FD has the possibility to give accurate solutions on nonuniform node layouts, still yielding sparse matrices.

As a model problem, we consider pricing of a European type basket option issued on two underlying assets, resulting in a PDE in two spatial dimensions and time. By employing a nonuniform node layout that has a denser node distribution where we are most interested in having an accurate solution, together with smoothing of the final condition, the numerical experiments demonstrate

that our developed method gives a very accurate solution in a short time using fewer nodes than the methods that we compare with, for this model problem. The fact that we can solve the problem accurately with fewer nodes becomes extremely important when we want to solve problems in higher dimensions, e.g., for pricing of financial derivatives issued on several underlying assets. Since the number of degrees of freedom grows exponentially in the number of dimensions (number of underlying assets), the ability to use fewer nodes per dimension to reach a certain accuracy might lead to the possibility to solve problems that would not be possible to solve with traditional techniques.

References

[1] V. Bayona, N. Flyer, B. Fornberg, and G. A. Barnett. On the role of polynomials in RBF–FD approximations: II. Numerical solution of elliptic pdes. *Journal of Computational Physics*, 332:257–273, 2017.

[2] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[3] B. Düring and C. Heuer. High-order compact schemes for parabolic problems with mixed derivatives in multiple space dimensions. *SIAM Journal on Numerical Analysis*, 53(5):2113–2134, 2015.

[4] N. Flyer, B. Fornberg, V. Bayona, and G. A. Barnett. On the role of polynomials in RBF–FD approximations: I. Interpolation and accuracy. *Journal of Computational Physics*, 321:21–38, 2016.

[5] T. Haentjens and K. J. In 't Hout. Alternating direction implicit finite difference schemes for the Heston–Hull–White partial differential equation. *Journal of Computational Finance*, 16:83–110, 2012.

[6] E. Hairer, S. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Solving Ordinary Differential Equations. Springer, 2008. ISBN 9783540788621.

[7] K. In't Hout and S. Foulon. ADI finite difference schemes for option pricing in the Heston model with correlation. *Int. J. Numer. Anal. Model*, 7(2): 303–320, 2010.

[8] H.-O. Kreiss, V. Thom e, and O. Widlund. Smoothing of initial data and rates of convergence for parabolic difference equations. *Communications on Pure and Applied Mathematics*, 23(2):241–259, 1970.

[9] E. Larsson, K. Åhlander, and A. Hall. Multi-dimensional option pricing using radial basis functions and the generalized Fourier transform. *Journal of Computational and Applied Mathematics*, 222(1):175–192, 2008.

[10] G. Linde, J. Persson, and L. von Sydow. A highly accurate adaptive finite difference solver for the Black–Scholes equation. *International Journal of Computer Mathematics*, 86(12):2104–2121, 2009.

[11] P. Lötstedt, J. Persson, L. von Sydow, and J. Tysk. Space–time adaptive finite difference method for European multi-asset options. *Computers & Mathematics with Applications*, 53(8):1159–1180, 2007.

[12] S. Milovanović. Pricing financial derivatives using radial basis function generated finite differences with polyharmonic splines on smoothly varying node layouts. Draft report, 2018.

[13] S. Milovanović and L. von Sydow. Radial basis function generated finite differences for option pricing problems. *Computers & Mathematics with Applications*, 75(4):1462–1481, 2018.

[14] J. Persson and L. von Sydow. Pricing European multi-asset options using a space-time adaptive FD-method. *Computing and Visualization in Science*, 10(4):173–183, 2007.

[15] J. Persson and L. von Sydow. Pricing American options using a space-time adaptive finite difference method. *Mathematics and Computers in Simulation*, 80(9):1922–1935, 2010.

[16] U. Pettersson, E. Larsson, G. Marcusson, and J. Persson. Improved radial basis function methods for multi-dimensional option pricing. *Journal of Computational and Applied Mathematics*, 222(1):82–93, 2008.

[17] D. M. Pooley, K. R. Vetzal, and P. A. Forsyth. Convergence remedies for non-smooth payoffs in option pricing. *Journal of Computational Finance*, 6(4):25–40, 2003.

[18] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

[19] A. Safdari-Vaighani, A. Heryudono, and E. Larsson. A radial basis function partition of unity collocation method for convection–diffusion equations arising in financial applications. *Journal of Scientific Computing*, 64(2): 341–367, 2015.

[20] V. Shcherbakov. Radial basis function partition of unity operator splitting method for pricing multi-asset american options. *BIT Numerical Mathematics*, 56(4):1401–1423, 2016.

[21] V. Shcherbakov and E. Larsson. Radial basis function partition of unity methods for pricing vanilla basket options. *Computers & Mathematics with Applications*, 71(1):185–200, 2016.

[22] D. Tavella and C. Randall. *Pricing Financial Instruments: The Finite Difference Method (Wiley Series in Financial Engineering)*. Wiley New York, 2000.

[23] L. von Sydow, L. Josef Höök, E. Larsson, E. Lindström, S. Milovanović, J. Persson, V. Shcherbakov, Y. Shpolyanskiy, S. Sirén, J. Toivanen, et al. BENCHOP–the BENCHmarking project in Option Pricing. *International Journal of Computer Mathematics*, 92(12):2361–2379, 2015.

[24] L. von Sydow, J. Toivanen, and C. Zhang. Adaptive finite differences and IMEX time-stepping to price options under Bates model. *International Journal of Computer Mathematics*, 92(12):2515–2529, 2015.

[25] L. von Sydow, S. Milovanović, E. Larsson, K. In 't Hout, M. Wiktorsson, C. W. Oosterlee, V. Shcherbakov, M. Wyns, A. Leitao, S. Jain, T. Haentjens, and J. Waldén. BENCHOP–SLV: The BENCHmarking project in Option Pricing – Stochastic and Local Volatility problems. Submitted for publication, 2018.