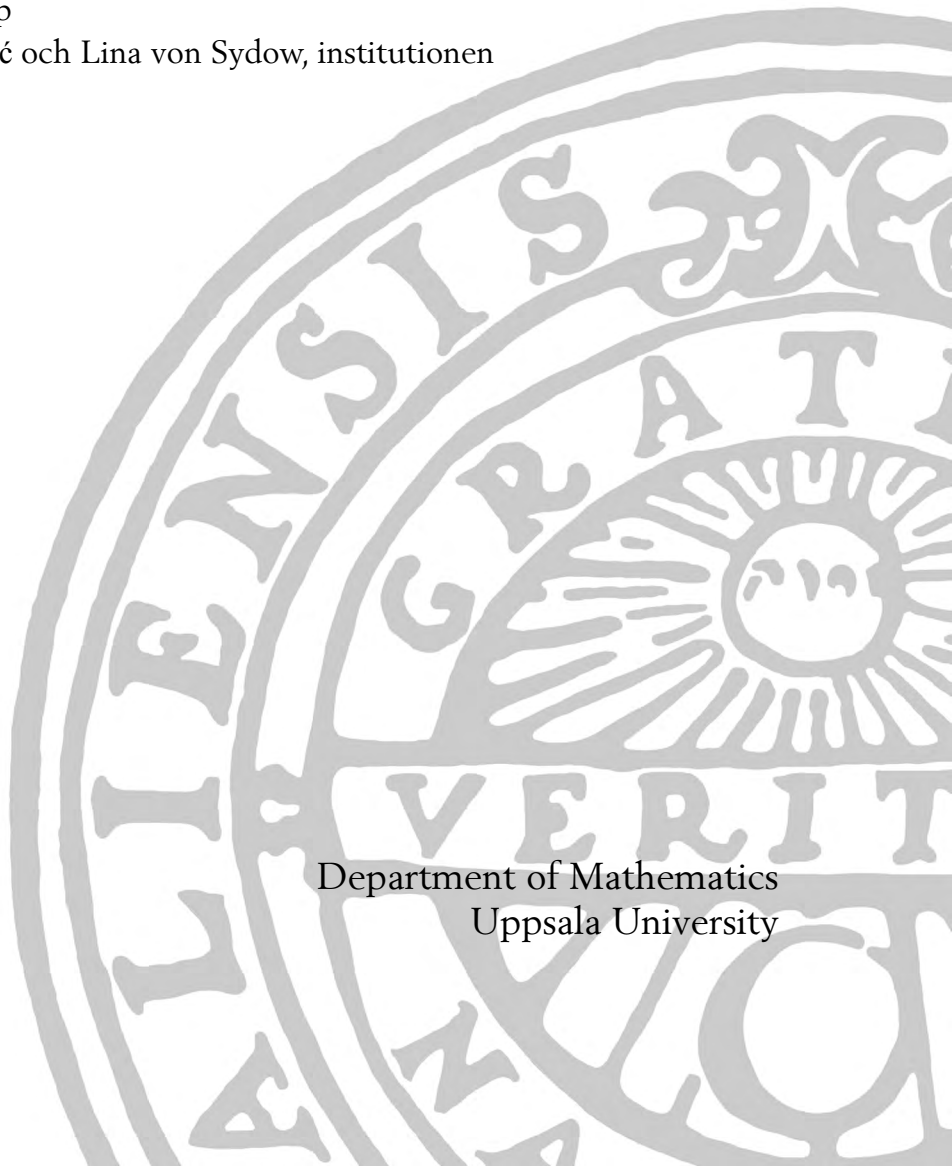
UPPSALA
UNIVERSITET

U.U.D.M. Project Report 2016:51

# Least Squares Radial Basis Function generated Finite Differences for Option Pricing

Stephane Dumanois

Examensarbete i matematik, 30 hp
Handledare: Slobodan Milovanović och Lina von Sydow, institutionen för informationsteknologi
Ämnesgranskare: Erik Ekström
Examinator: Magnus Jacobsson
December 2016



Department of Mathematics
Uppsala University

Uppsala University

# Least Squares Radial Basis Function generated Finite Differences for Option Pricing

*Author:*
Stephane Dumanois

*Supervisors:*
Slobodan Milovanović
Lina von Sydow

A thesis submitted in fulfillment of the requirements for the
Master of Science in Financial Mathematics
in the department of Information Technology and Mathematics
at Uppsala University

UPPSALA
UNIVERSITET

December 21, 2016

## Acknowledgements

I would like to thank my supervisors Slobodan Milovanović and Lina von Sydow who recommended me a subject of great interest. I also thank them for their guidance, kindness, and patience through the realization of this thesis work.

# Table of Contents

# 1 Introduction

Every day financial institutions resort to options trading in order to develop strategies to reduce their risk due to potential motions on a variable market. Therefore, institutions need to be able to price these financial products accurately. An option is defined as a financial derivative (or contract) that gives its buyer the right, but not the obligation, to buy (in case of a call option) or sell (in case of a put option) a financial security or asset at a pre-determined price (strike price) at a specific date in the case of European options, at different discrete dates chosen in advance in the case of Bermudan options, or at any moment during a certain period of time in the case of American options. These options are referred to as vanilla options in opposition to exotic options that have different kinds of properties. Renowned exotic options include barrier options, Asian options, chooser options, and compound options. See [11] for more details and examples on exotic options. A typical way of pricing these financial securities is to use Monte-Carlo (MC) methods to solve a stochastic differential equation (SDE) based on the Black-Scholes-Merton (BSM) model, a model developed independently in 1973 by M. S. Scholes and F. Black, and R. C. Merton. However, MC methods are known to be slowly converging, and this is why other ways have been developed to compute option prices. One of them is to solve the equivalent pricing problem by using a partial differential equation (PDE) instead of the SDE. Indeed, with the help of the Feynman-Kac formula, which links the solution of a PDE to the solution of its corresponding SDE. The BSM model can be solved using the BSM equation instead of the SDE, this formulation was developed in [1] and [3]. A classical and widely used PDE approach in low dimensions is the Finite Differences (FD) method, which discretizes the BSM equation both in space and time to get an approximate solution. Nevertheless, when options depend on several underlying assets, i.e. basket options, the PDE is then multi-dimensional and the numerical methods used to solve these problems frequently endure the so-called "curse of dimensionality". A method believed to be accurate and efficient in high dimensions is Radial Basis Function generated Finite Differences (RBF-FD). It aims to extenuate the curse of dimensionality by exploiting the best properties of both Radial Basis Functions (RBF) methods and FD methods. However, these methods are very sensitive to a shape parameter $\varepsilon$. Therefore, the idea behind this thesis is to try to implement a Least Squares (LS) version of the RBF-FD methods that could potentially lead to more stability in the error and more precision in the results. Ideally, the goal sought is to obtain a stable error behavior with a great accuracy so that the choice of $\varepsilon$ would not be a matter anymore.
In the next section we will introduce the BSM model and equation, then in Chapter 3 we will show the different numerical methods for option pricing leading up to the RBF-FD method, then we will present the LS implementation of the RBF-FD in Chapter 4, and in the last chapter experiments will be run in one and two dimensions models in order to test our hypothesis.

# 2 Option Pricing

Option pricing has been based on the BSM theory for some time and is still widely used.

## 2.1 Black-Scholes-Merton Model

We consider the standard BSM model that consists of a bond $B$ and a stock $S$ that have the following dynamics

$$\begin{aligned} dB(t) &= rB(t)dt, \\ dS(t) &= \mu S(t)dt + \sigma S(t)dW(t), \end{aligned} \tag{1}$$

where $W$ is a Wiener process, $r$ the interest rate, $\mu$ is the drift parameter and $\sigma$ the volatility of the stock $S$. The arbitrage free price of the option issued on this asset is given by

$$u(S(t)) = e^{-r(T-t)}\mathbb{E}_t^Q[g(S(T))], \tag{2}$$

where $\mathbb{E}_t^Q$ is the expected value at time $t$ under the measure $Q$ and $g$ is the payoff function that is the known value of the option at time of maturity $T$. By using the following dynamics of S:

$$dS(t) = rS(t)dt + \sigma S(t)dW(t), \tag{3}$$

we can obtain the risk-neutral price of the option $u$ using a MC method to calculate the expectation value in (2) with the help of the dynamics in (3). MC methods to compute the arbitrage free prices of financial securities are widely used by financial institutions, an example of this method can be found in [2], and in [12], [13] for the pricing of barrier and Asian options respectively. However, as stated earlier, these methods may give accurate results for problems in high dimensions but they are slowly converging and can calculate only one spot price at a time.

## 2.2 Black-Scholes-Merton Equation

Another way of finding the arbitrage free price of an option is done with the help of the Feynman-Kac formula, which links the solution of a PDE to the solution of its corresponding SDE. Therefore, instead of using the SDE, the BSM model can be solved using the Black-Scholes-Merton equation:

$$\begin{aligned} u_t + rsu_s + \tfrac{1}{2}s^2\sigma^2 u_{ss} - ru &= 0, \\ u(s,T) &= g(s), \end{aligned} \tag{4}$$

where $u_x$ corresponds to the derivative of $u$ with respect to the variable $x$, and $u_{xx}$ the second derivative of $u$ with respect to $x$. Equation (4) can be rewritten in the following manner:

$$\begin{aligned} u_t + \mathcal{L}u &= 0, \\ u(s,T) &= g(s), \end{aligned} \tag{5}$$

where $\mathcal{L}$ is the spatial differential operator. Solving this equation backward in time will give the arbitrage free price for all $s$ and not just one value as it is the case with MC methods.
Also available on the market are options with several underlying assets. In that

case the model becomes:

$$\begin{cases} dB(t) & = rB(t)dt, \\ dS_1(t) & = \mu_1 S_1(t)dt + \sigma_1 S_1(t)dW_1(t), \\ dS_2(t) & = \mu_2 S_2(t)dt + \sigma_2 S_2(t)dW_2(t), \\ & \vdots \\ dS_D(t) & = \mu_D S_D(t)dt + \sigma_D S_D(t)dW_D(t), \end{cases} \tag{6}$$

where $D$ represents the number of underlying assets or dimensionality of the problem. The option price then becomes:

$$u(S_1(t), \ldots, S_D(t), t) = e^{-r(T-t)}\mathbb{E}_t^Q[g(S_1(T), \ldots, S_D(T))], \tag{7}$$

and the BSM equation is given by:

$$\begin{cases} u_t + r\sum_i^D s_i u_{s_i} + \frac{1}{2}\sum_{i,j}^D \rho_{ij}\sigma_i\sigma_j s_i s_j u_{s_i s_j} - ru = u_t + \mathcal{L}u = 0, \\ u(s_1, s_2, \ldots, s_D, T) = g(s_1, s_2, \ldots, s_D). \end{cases} \tag{8}$$

where $\rho_{ij}$ is the coefficient of correlation between asset $i$ and $j$.

# 3 Numerical PDE Methods for Option Pricing

In this section we are going to describe some of the numerical PDE methods used in option pricing. We will start with the widely used FD methods. Then we will present the RBF methods. Finally, we will see how RBF-FD methods are built upon the previous two methods.

## 3.1 Finite Differences

The FD method is a very popular way of solving PDEs numerically and a common way to price options, see [4] and [5]. We describe the method to solve Equation (4). We first define the computational domain in space (stock prices) and time with

$$0 \leq s \leq s^*,$$
$$0 \leq t \leq T,$$

where $s^*$ is set to $4K$. The first step to FD methods is to discretize this domain in space into $N$ space steps, which gets us the discretized stock prices vector $\{s : s^{(i)} = i \times \Delta s\}$ for $i = 0, 1, 2, \ldots, N$, and where $\Delta s = s^*/N$. Now that the domain is discretized in space, we need to construct a discretization of the PDE, that is, each space derivative in the PDE is discretized as shown for example in (9) and (10):

$$u_s(s^{(i)}, t) \approx \frac{u(s^{(i+1)}, t) - u(s^{(i-1)}, t)}{2\Delta s}, \tag{9}$$

$$u_{ss}(s^{(i)}, t) \approx \frac{u(s^{(i+1)}, t) - 2u(s^{(i)}, t) + u(s^{(i-1)}, t)}{(\Delta s)^2}. \tag{10}$$

We also need to define the following boundary conditions:

$$u(0,t) = 0, \tag{11}$$

$$u(s^*,t) = s^* - Ke^{-r(T-t)}. \tag{12}$$

After putting equations (9) and (10) in the PDE of equation (4), we re-arrange it in order to create a semi-discretized equation. We therefore get a system of equations of the form

$$\tilde{u}_t = A\tilde{u}, \tag{13}$$

where

$$\tilde{u} = \begin{pmatrix} u^{(1)}(t) \\ u^{(2)}(t) \\ u^{(3)}(t) \\ \vdots \\ u^{(N)}(t) \end{pmatrix},$$

and $A$ is a tri-diagonal sparse matrix. See more about FD methods to price vanilla and exotic options in [5], [9], and [14]. A great number of PDEs used in finance can be numerically solved with the help of FD methods. The fact that the method generates a sparse matrix also makes it quite fast in low dimensions. However, as disclosed earlier the principal issue with FD methods is the "curse of dimensionality" which means that for a problem in high dimensions the number of unknowns in the linear system of equations used to solve every time step grows exponentially with the number of dimensions.

## 3.2 Radial Basis Functions

RBF methods have also been used to price options, see e.g. [6], and [8]. This method is mesh-free which means we do not need to solve the PDE on a hyper-cube but solely on the domain of interest. This can be of interest with option pricing since we often are interested only in prices near the strike price for example. In addition, the method does not get more complicated in higher dimensions. The first step to RBF methods is to discretize the space into $N$ nodes, then approximate the solution of the PDE problem as a linear combination of RBFs, denoted by $\phi(r)$ centered at node points $s^{(k)}$ for $k = 1, 2, \ldots, N$:

$$u(s,t) \approx \sum_{k=1}^{N} \lambda_k(t)\phi(\varepsilon||s - s^{(k)}||), \tag{14}$$

where $\phi(\cdot)$ is a radial basis function, $\varepsilon$ is a shape parameter, and $\lambda$ are the unknown coefficients. Common radial basis function $\phi(\cdot)$ include

| | |
|---|---|
| Gaussian: | $e^{-(\varepsilon r)^2}$, |
| Multiquadric: | $\sqrt{1+(\varepsilon r)^2}$, |
| Inverse multiquadric: | $1/\sqrt{1+(\varepsilon r)^2}$, |
| Inverse quadratic: | $1/(1+(\varepsilon r)^2)$. |

We will use the Gaussian RBF in the experiments. For a better comprehension we can rewrite (14) as

$$\underbrace{\begin{bmatrix} \phi(\varepsilon\|s^{(1)}-s^{(1)}\|) & \phi(\varepsilon\|s^{(1)}-s^{(2)}\|) & \dots & \phi(\varepsilon\|s^{(1)}-s^{(N)}\|) \\ \phi(\varepsilon\|s^{(2)}-s^{(1)}\|) & \phi(\varepsilon\|s^{(2)}-s^{(2)}\|) & \dots & \phi(\varepsilon\|s^{(2)}-s^{(N)}\|) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(\varepsilon\|s^{(N)}-s^{(1)}\|) & \phi(\varepsilon\|s^{(N)}-s^{(2)}\|) & \dots & \phi(\varepsilon\|s^{(N)}-s^{(N)}\|) \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} \lambda_1(t) \\ \lambda_2(t) \\ \vdots \\ \lambda_N(t) \end{bmatrix}}_{\lambda} = \underbrace{\begin{bmatrix} u_1(t) \\ u_2(t) \\ \vdots \\ u_N(t) \end{bmatrix}}_{\tilde{u}},$$

or even as

$$\tilde{u}(t) = A\lambda(t). \tag{15}$$

The smaller $\varepsilon$ is the wider the support of RBF is and the more accurate the results are. Unfortunately, the interpolation matrix $A$ then becomes ill-conditioned which leads to instability in the results. Therefore $\varepsilon$ need to be chosen carefully. In addition to the sensitivity to the choice of $\varepsilon$, this method creates a dense matrix which makes it slow.

### 3.3 Radial Basis Functions generated Finite Differences

The idea behind RBF-FD method is to try to combine the advantages of both previous methods, that is the sparsity structure of FD methods mixed with the ease of use in high dimension and mesh-free structure of the RBF methods. These methods were first introduced in [15] and have been successful in some other areas, see e.g. [16]. Let us say that we have a scattered set of nodes with a total of $N$ points. For every point $s_i$ we define a neighborhood of $n-1$ points, and we refer to $n$ as the stencil size. The differential operator is then approximated at every point by

$$\mathcal{L}u(s_i) \approx \sum_{k=1}^{n} w_k^{(i)} u_k^{(i)}, \qquad i = 1, \dots, N, \tag{16}$$

which in matrix form gives

$$\underbrace{\begin{bmatrix} \phi(\varepsilon\|s_1^{(i)}-s_1^{(i)}\|) & \dots & \phi(\varepsilon\|s_1^{(i)}-s_n^{(i)}\|) \\ \phi(\varepsilon\|s_2^{(i)}-s_1^{(i)}\|) & \dots & \phi(\varepsilon\|s_2^{(i)}-s_n^{(i)}\|) \\ \vdots & \ddots & \vdots \\ \phi(\varepsilon\|s_n^{(i)}-s_1^{(i)}\|) & \dots & \phi(\varepsilon\|s_n^{(i)}-s_n^{(i)}\|) \end{bmatrix}}_{A^{(i)}} \underbrace{\begin{bmatrix} w_1^{(i)} \\ w_2^{(i)} \\ \vdots \\ w_n^{(i)} \end{bmatrix}}_{w^{(i)}} = \underbrace{\begin{bmatrix} [\mathcal{L}\phi(\varepsilon\|s-s_1^{(i)}\|)]_{s=s_i} \\ [\mathcal{L}\phi(\varepsilon\|s-s_2^{(i)}\|)]_{s=s_i} \\ \vdots \\ [\mathcal{L}\phi(\varepsilon\|s-s_n^{(i)}\|)]_{s=s_i} \end{bmatrix}}_{l^{(i)}},$$

the weights $w$ are then inserted line by line to make a sparse matrix $W$ approximating the differential operator $\mathcal{L}$ such that we get the discretization of the BSM equation in space:

$$\tilde{u}_t = W\tilde{u}. \tag{17}$$

The issue with RBF-FD is that it is sensitive to the shape parameter $\varepsilon$, when this parameter gets small the stability of the errors gets affected.

### 3.4 Temporal Discretization

Before going into the next PDE method for option pricing, we need to introduce the temporal discretization. In the case of FD and RBF-FD, we derived the space discretization of Equation (4) and we ended up with the semi-discretized equations (13) and (17). For, the time integration of these equations we will use the Backward Differentiation Formula 1 and 2 (BDF-1, BDF-2). An example of this can be seen in [10].

## 4 Least Squares RBF-FD

One of the issues encountered with RBF-FD methods is that the accuracy of the results is really sensitive to the choice of the shape parameter $\varepsilon$. Indeed, as it will be shown later by the experiments, depending on the choice of this parameter, errors can be huge. That is why the hypothesis behind this thesis is to check if implementing a Least Squares (LS) version of the weights calculations could lead to better stability in the errors and also more accuracy. Ideally, the goal sought is to obtain a stable error behavior with a high accuracy so that the choice of $\varepsilon$ would not matter anymore. Let us say that we have a grid with a total of $N$ points. For every point $s_i$ we define a neighborhood of $n-1$ points, we refer to $n$ as the stencil size. However, this time we add what a certain number of points to the initial stencil in order to compute the RBF-FD weights. From now on we will refer to these points as "LS points". It is important to note that these points are used only to get a better computation of the weights and will therefore not be used as evaluation points. That is why these points are taken on a separate grid that corresponds to points inserted in between each point of the original grid (see Section 5.1.1 and 5.2.1). Matrices $A$ and $l$ will then have $n + LS$ lines but still give out a weight vector of $n$ lines. Therefore, the RBF-FD weights is the only part of the method that is directly impacted by the LS implementation. The differential operator in (16) is then approximated at every point by the following matrix calculation:

$$\underbrace{\begin{bmatrix} \phi(\varepsilon\|s_1^{(i)} - s_1^{(i)}\|) & \dots & \phi(\varepsilon\|s_1^{(i)} - s_n^{(i)}\|) \\ \phi(\varepsilon\|s_2^{(i)} - s_1^{(i)}\|) & \dots & \phi(\varepsilon\|s_2^{(i)} - s_n^{(i)}\|) \\ \vdots & \ddots & \vdots \\ \phi(\varepsilon\|s_n^{(i)} - s_1^{(i)}\|) & \dots & \phi(\varepsilon\|s_n^{(i)} - s_n^{(i)}\|) \\ \vdots & \ddots & \vdots \\ \phi(\varepsilon\|s_{n+LS}^{(i)} - s_1^{(i)}\|) & \dots & \phi(\varepsilon\|s_{n+LS}^{(i)} - s_n^{(i)}\|) \end{bmatrix}}_{\tilde{A}^{(i)}} \underbrace{\begin{bmatrix} w_1^{(i)} \\ w_2^{(i)} \\ \vdots \\ w_n^{(i)} \end{bmatrix}}_{w^{(i)}} = \underbrace{\begin{bmatrix} [L\phi(\varepsilon\|s - s_1^{(i)}\|)]_{s=s_i} \\ [L\phi(\varepsilon\|s - s_2^{(i)}\|)]_{s=s_i} \\ \vdots \\ [L\phi(\varepsilon\|s - s_n^{(i)}\|)]_{s=s_i} \\ \vdots \\ [L\phi(\varepsilon\|s - s_{n+LS}^{(i)}\|)]_{s=s_i} \end{bmatrix}}_{\tilde{l}^{(i)}},$$

we then repeat the same process as for standard RBF-FD, that is to say, the weights $w$ are then inserted in a sparse matrix $W$ such that we get the discretization of the BSM equation in space (17). Once the discretization in space has been done, we integrate in time using BDF-1 and BDF-2.
In the next section, experiments will be performed in order to check if the LS computation of the weights leads to more stable and/or more accurate results. Tests will also try to show if the number of added points to perform the LS method has an importance or not.

# 5 Experiments

## 5.1 One Dimensional Experiments

The results of experiments and tests will now be demonstrated in order to check if making a Least Square implementation of the RBF-FD methods could result in more stability in the errors. In order to do so, we try to price, as a support for the tests, a European call option with exercise date $T = 1$, strike price $K = 1$, interest rate $r = 0.03$, and a volatility $\sigma = 0.15$. The payoff function is given by

$$u(s, T) = \max\{s - K, 0\}.$$

The boundary conditions are set up as in Equations (11) and (12) where $s^*$ is set at $4K$. The radial basis function chosen for the experiments is the following Gaussian function

$$\phi(r) = e^{-(\varepsilon r)^2}.$$

Finally, errors were measured on a subdomain of better interest i.e an interval close to the strike price K:

$$s \in \left[\frac{1}{3}K, \frac{5}{3}K\right].$$

The Matlab function used to compute these tests takes as inputs the grid size $N$ i.e the number of points through which the RBFs will be calculated (space discretization), the size of the stencil $n$, the number of time-steps $M$ used for BDF-1 and BDF-2, and of course the number of LS points used in the LS case.

### 5.1.1 One Dimensional Grid

In one dimension (1D) the grid is a simple line of $N$ points equally spaced over the interval [0,4]. For example, Figure 1 shows the grid with $N = 35$, represented by the blue points.. The tests ran in 1D are all done on three different sizes of grid, more precisely for $N = \{100, 1000, 10000\}$. Every point on this grid will be taken as an evaluation point, therefore the greater $N$ means the longer the computational time is as well. With standard RBF-FD, we then build the nearest neighbors stencil, i.e. we select the $n-1$ closest points to the evaluation point and we observe it as a stencil of size $n$, as shown in Figure 1 by the red points. Experiments in 1D have been run over three stencil sizes $n = \{3, 5, 7\}$. Note that in 1D the first doable stencil size is $n = 3$. Finally, the LS points are added to the grid, and more precisely in between each two points on the grid, in order to create the new stencil. It works the same way as for the stencil points, that is once the evaluation point set, we take the $LS$ closest points to it and add them to the original stencil and we therefore have a stencil of size $n + LS$. This is best represented by the black points in Figure 1. Note that these points are equally spaced from each other. It is very important to understand that the LS grid is only used to create the extended stencil, therefore none of these points will be taken as evaluation points. For the clarity of what will come next we introduce a new notation, from now on when we will see $LS = X_Y$ it will mean that we have added $X$ points to the original stencil, and that these $X$ points have been taken on a grid divided by $Y$ equally distanced points. This is better illustrated in Figure 1 where we have for example the case $n = 7$ (original stencil) and $LS = 30_5$ which means we have added 30 points (black)

to the original stencil ones (red) and that these LS points have been taken on a grid where the distance between two points on the original grid has been divided by 5.

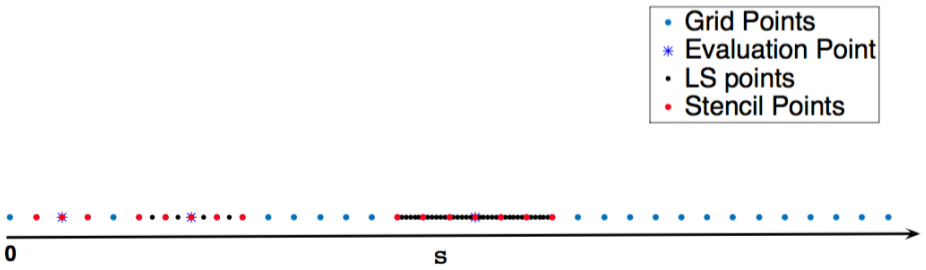



Figure 1: 1D grid: $N = 35$, $n = \{3, 5, 7\}$, $LS = \{0, 4_1, 30_5\}$.

### 5.1.2 Analysis of Results in One Dimension

The following figures show the results of tests ran in 1D with RBF-FD-LS using three different stencil sizes $n$, i.e $n = \{3, 5, 7\}$, and five different numbers of added LS points, LS $= \{0, 2_1, 10_5, 20_{10}, 100_{50}\}$. Each graph exhibits the maximum error obtained for one hundred values of $\varepsilon \in [10^{-4}, 10^4]$. Furthermore, every graph shows the results for three different sizes of grid $N = \{100, 1000, 10000\}$. Note that the size of $M$ is always kept to 1000. Figures 2 and 3 show the results of tests run on the 1D function for a stencil size of $n = 3$. The first graph of Figure 2 corresponds to the results obtained from the standard RBF-FD, i.e. with no added LS points. It first appears that the greater the size of the grid $N$ is, the smaller the errors get but the more unstable is the method as well. In other words, a high value of $N$ is going to give a better result in the maximum errors but it then makes it very sensitive to the choice of the right $\varepsilon$, and vice-versa. Therefore, what can be said about the non-LS results is that with a grid size of $N = 100$, there is some stability in the errors but with a limited accuracy of $10^{-2}$. In the case of $N = 1000$ and 10000, there is almost no stability in the errors even though some values of $\varepsilon$ give a great accuracy of $10^{-6}$.

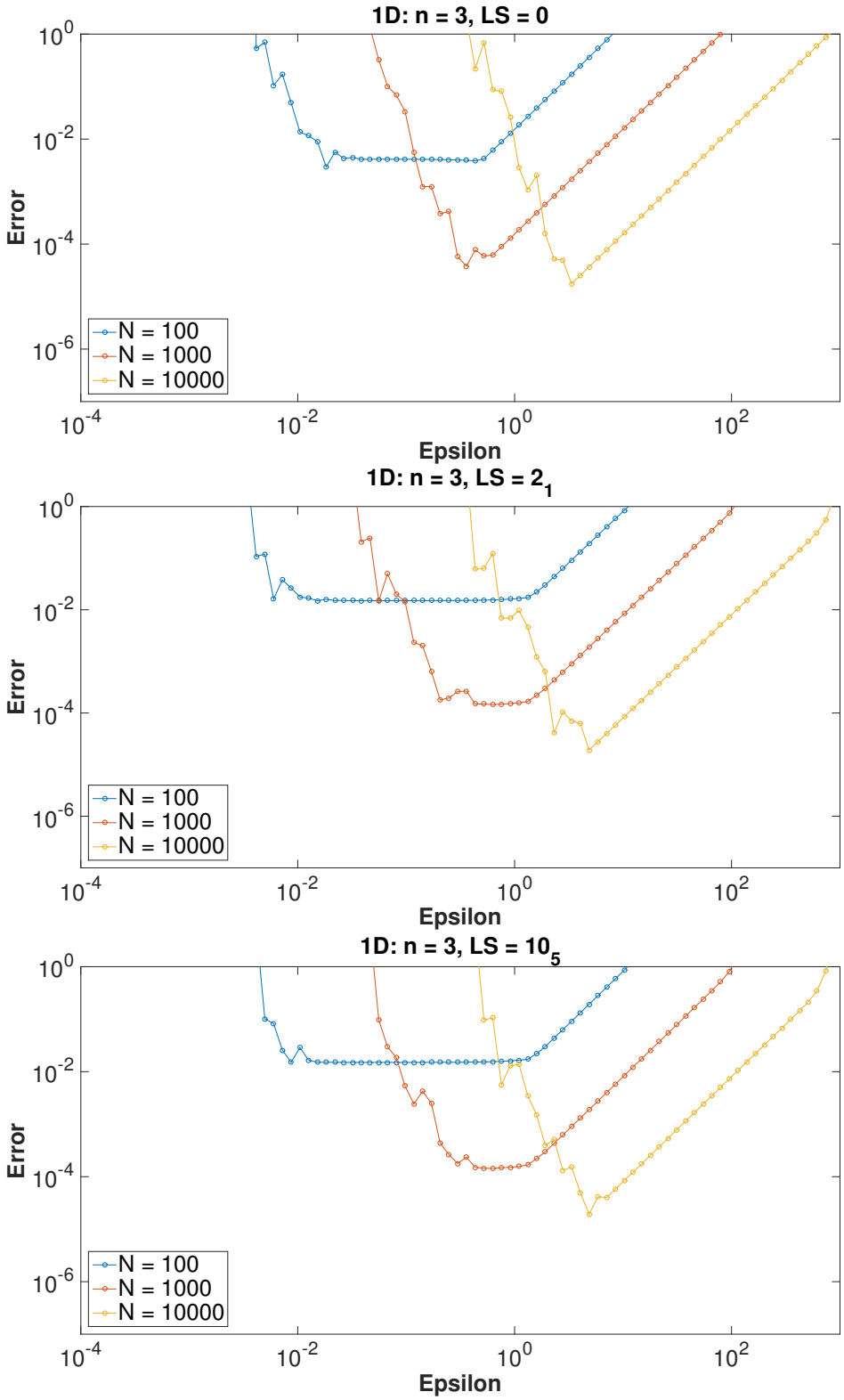



Figure 2: 1D Results: $n = 3$, $LS = \{0, 2_1, 10_5\}$.

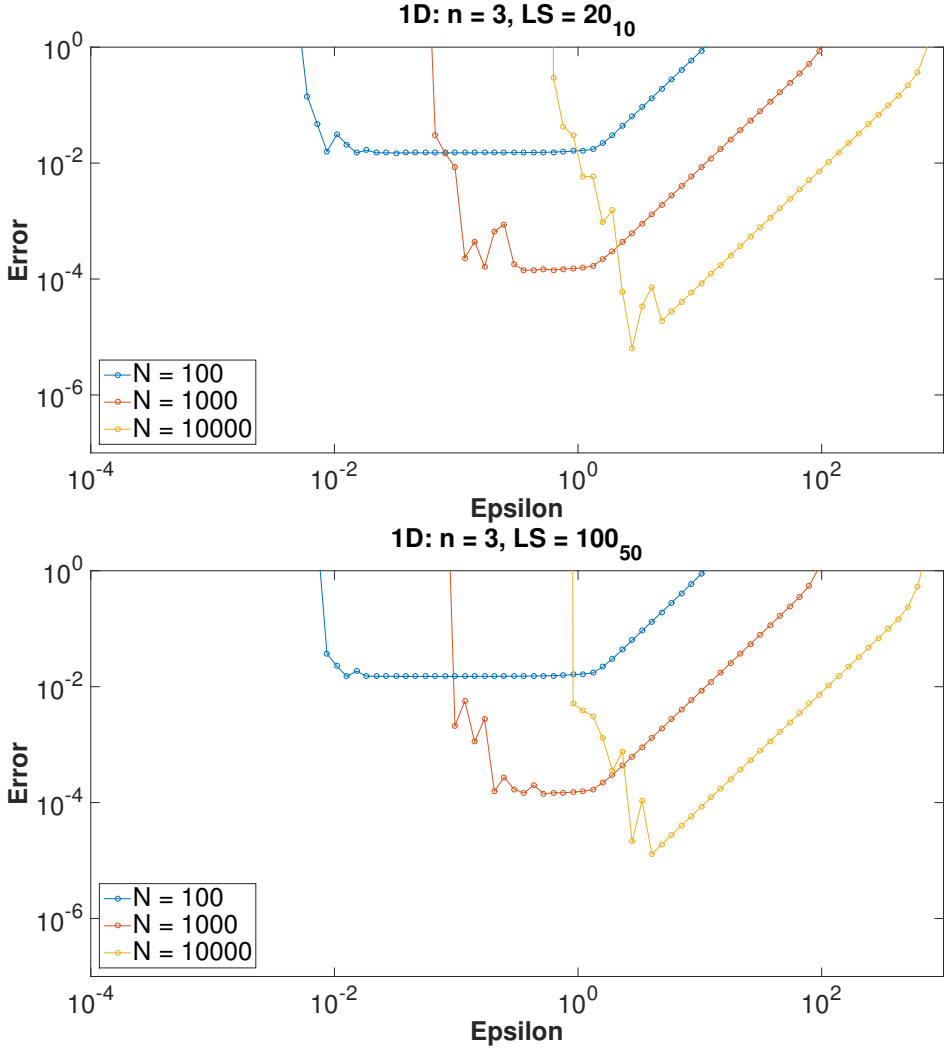



Figure 3: 1D Results: $n = 3$, $LS = \{20_{10}, 100_{50}\}$.

The following graphs of Figures 2 and 3 show the results on the same test ($n = 3$) except this time LS points, i.e $LS = \{2_1, 10_5, 20_{10}, 100_{50}\}$, have been added in the same manner as described in Section 5.1.1. With only 2 added points ($LS = 2_1$), results follow the same scheme but are already pretty different from the non-LS version. Indeed, for $N = 100$, the line of stability observed in the previous case is still there but is actually longer reaching values of $\varepsilon$ smaller than $10^{-2}$ on the left side and greater than $10^0$ on the right side. However, the errors are slightly bigger than without LS. The same conclusions can be basically drawn with $N = 1000$, there is now a small stability line but not as efficient in the errors. As for $N = 10000$, the LS version does not offer any real changes in the results. The hypothesis would be to think that by adding more points the results would get even better. However, when observing the results obtained with more LS points nothing really changes. What has been said for the case with $LS = 2_1$ can be applied for the results with $LS = \{10_5, 20_{10}, 100_{50}\}$. In

summary, with a stencil size of $n = 3$, results are not as good as expected even though there are some small improvements in the stability of the errors, especially for small choices of grid.

Figures 4 and 5 show results of tests run in 1D for a stencil size of $n = 5$. In the first graph of Figure 4 the test has been run without added point. The same analysis can be drawn from the case with $n = 3$, that is, the greater the size of the grid $N$ gets the smaller the errors are but the smaller the stability in the errors gets as well, except that in this case stability is smaller but so are the errors. Therefore, what can be said about the non-LS results is that with a grid size of $N = 100$, there is some stability in the errors but with an improved accuracy of less than $10^{-3}$. In the case of $N = 1000$ and 10 000, there is almost no stability in the errors even though some values of $\varepsilon$ give very nice results in the errors. Overall, results are inexistent when $\varepsilon$ become smaller than $10^0$. The next graphs in Figures 4 to 5 show the results on the same test ($n = 5$) but with added points, i.e $LS = \{4_1, 20_5, 40_{10}, 200_{50}\}$, that have been added in the same manner as described in Section 5.1.1. Already with $LS = 4_1$, the results are considerably different from the non-LS version. Indeed, for $N = 100$, there is a clear zone of stability between $10^{-3}$ and $10^1$, with an accuracy of less than $10^{-2}$ in the $10^0$ range of $\varepsilon$. However, the same issue encountered with $n = 3$ happens i.e the accuracy is slightly worse than without LS. Big changes also happen with $N = 1000$, there are now two zones of stability and especially with $\varepsilon$ smaller than $10^1$ where a stability line is observed with an accuracy of less than $10^{-4}$. Same problem than with $N = 100$ though, the best accuracy without LS was down to $10^{-5}$ even if it was only one point reaching this level. Concerning $N = 10000$, even though there are many more points appearing in the results, there is no real line of stability. However, the best point goes down to an accuracy of $10^{-6}$. Pretty much the same thing can be said about the results with more added LS points. Indeed, there is some kind of stability for $N = 100$ and $N = 1000$ and not really for $N = 10000$. To sum it up, with a stencil size of $n = 5$, results are clearly better with the addition of the LS points even though the results are still not reaching the goal researched of an almost perfect stability in the errors and a high accuracy.

Finally, results observed in Figures 6 and 7 are issued from tests run on the 1D function for a stencil size of $n = 7$. In the first graph of Figure 6 the test has been run for the standard RBF-FD, i.e. without added points. More or less the same analysis can be drawn from the case with n = 5. Results seem even less stable with no stability line even for $N = 100$, and results are inexistent when $\varepsilon$ become smaller than $10^0$. The following graphs of Figures 6 and 7 show the results on the same test ($n = 7$) but with LS points, i.e LS $= \{6_1, 30_5, 60_{10}, 300_{50}\}$, that have been added in the same manner as described in Section 5.1.1. Again, results can be compared with the ones obtained with $n = 5$. Indeed, with only 1 added LS point, the results are already considerably different from the non-LS version. For $N = 100$, there are two clear zones of stability in $[10^-3, 10^-1]$ and $[10^0, 10^1]$, with an accuracy of less than $10^{-2}$ in the second zone. However, the same issue encountered with $n = 3$ and 5 happens i.e the accuracy of the errors is not as good as without LS. Concerning $N = 1000$ and $N = 10000$, results show little difference from the ones obtained with $n = 5$. That is to say, results are much better than without LS points in the sense that there are much more

points in the error and $\varepsilon$ ranges of interest, especially with $N = 1000$. However, results are still not the ones wished for. Once again, there is no clear difference in the outputs that depends on the number of added LS points. In conclusion, concerning the 1D version, LS implementation increases the stability of the results in most cases. However, it is not enough to solve the stability problem in its whole. New zones of stability appear but the accuracy is not as good as without LS and these zones are still too short to be useful in practice.

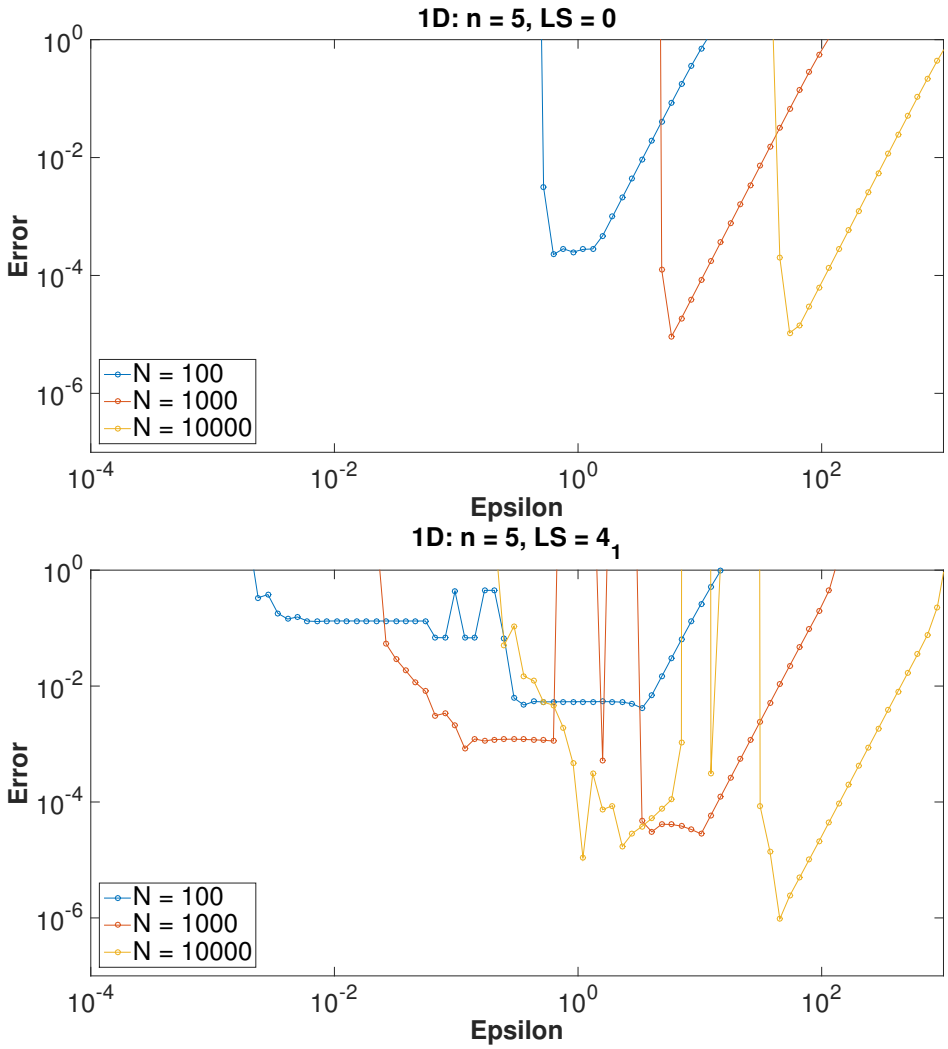



Figure 4: 1D Results: $n = 5$, $LS = \{0, 4_1\}$.

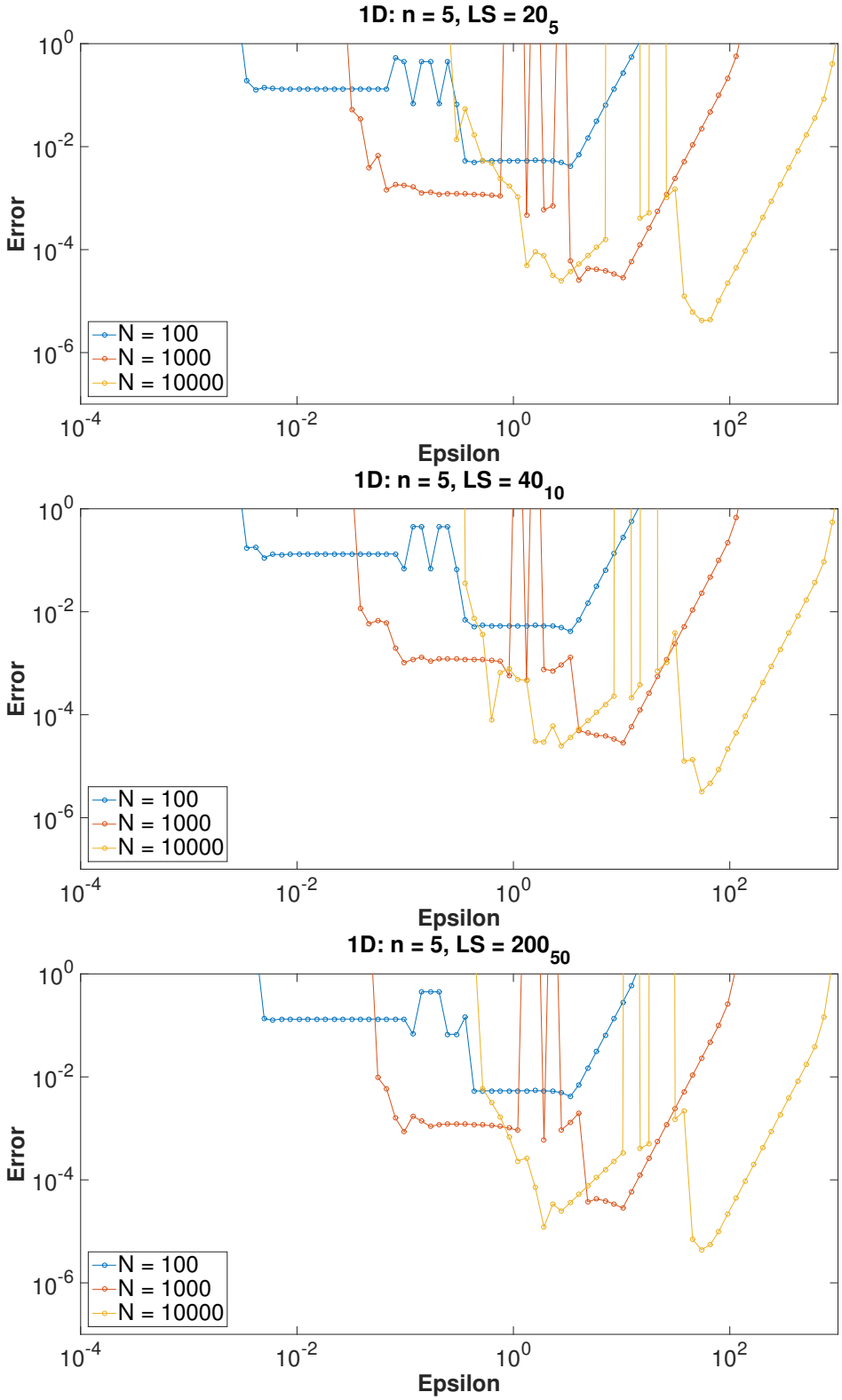



Figure 5: 1D Results: $n = 5$, $LS = \{20_5, 40_{10}, 200_{50}\}$.

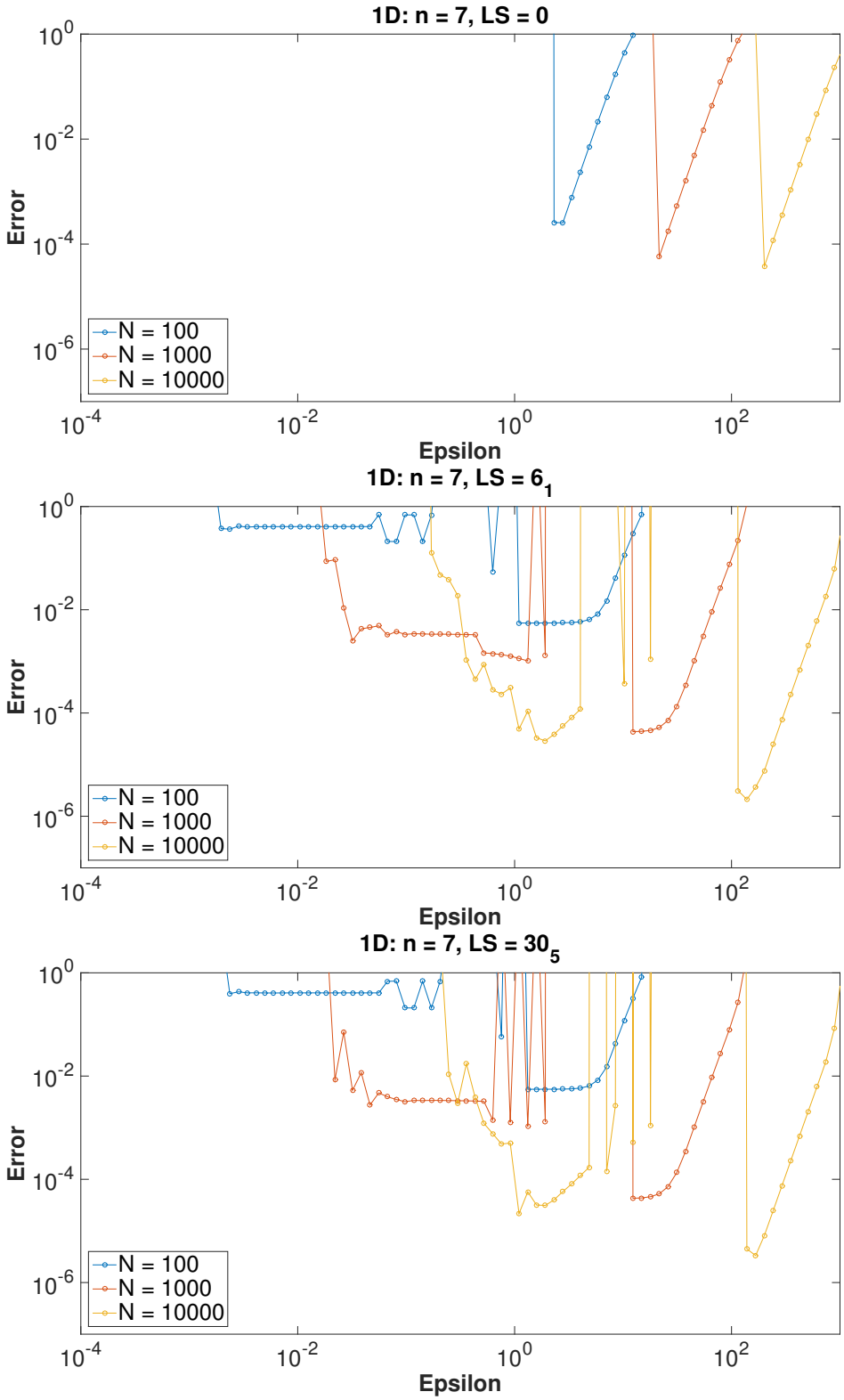



Figure 6: 1D Results: $n = 7$, $LS = \{0, 6_1, 30_5\}$.

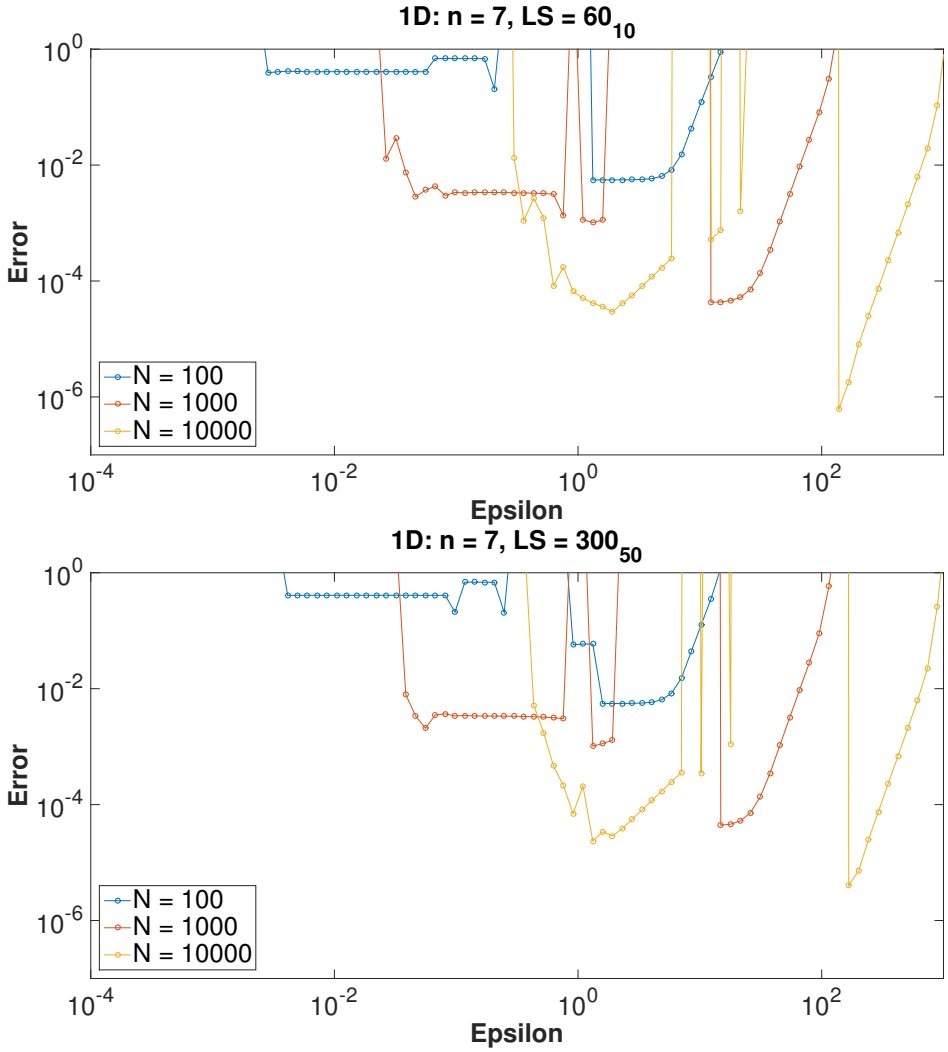



Figure 7: 1D Results: $n = 7$, $LS = \{60_{10}, 300_{50}\}$.

## 5.2 Two Dimensional Experiments

The results of experiments and tests in two dimensions (2D) will now be demonstrated in order to check if RBF-FD-LS could give more stability in the errors, and especially when the values of $\varepsilon$ get smaller than $10^0$.
In order to do so, we try to price, as a support for the tests, a European basket option with exercise date $T = 1$, strike price $K = 1$, interest rate $r = 0.03$, volatilities $\sigma_1 = \sigma_2 = 0.15$, and $\rho = 0.5$.
The payoff function is given by

$$u(s_1, s_2, T) = \max\left\{\frac{s_1(T) + s_2(T)}{2} - K, 0\right\}$$

The following boundary conditions are also set up

$$\begin{aligned} u(0,0,t) &= 0, \\ u(s_1^*, s_2^*, t) &= \tfrac{1}{2}(s_1^*(t) + s_2^*(t)) - e^{-r(T-t)}K. \end{aligned}$$

where $\{s_1^*, s_2^*\}$ represents the boundary condition set at $\{s_1^*, s_2^*\} = 8K$.
The radial basis function chosen for the experiments is the following Gaussian function

$$\phi(r) = e^{-(\varepsilon r)^2}.$$

Finally, errors were measured on a subdomain of better interest i.e an interval close to the strike price K:

$$(s_1, s_2) \in \left[\frac{1}{3}K, \frac{5}{3}K\right] \times \left[\frac{1}{3}K, \frac{5}{3}K\right].$$

The Matlab function used to compute these tests takes as inputs the grid size $N$ i.e the number of points through which the RBFs will be calculated (space discretization), the size of the stencil $n$, the number of time-steps $M$ used for BDF-1 and BDF-2, and of course the number of LS points used in the LS case.

### 5.2.1 Two Dimensional Grid

The grid in 2D takes the aspect of a triangle with $N_{total} = N(N+1)/2$ points, $N$ being the number of points for which each axis is equally divided by, that is each axis contains $N$ points. Note that in 1D we had $N_{total} = N$. Figure 8 and 9 show a 2D grid with $N = 25$, therefore 25 points equally spaced on each axis for a total of 325 grid points, represented by the blue dots on the figures. Most of the tests that have been run were made on grids of size $N = \{60, 80, 100\}$, which means grids with $N_{total} = \{1830, 3240, 5050\}$ points respectively. On these grids are then added the $n = \{9, 25\}$ stencil points corresponding to the red dots on the figures, for which the central point is the evaluation point, denoted by the blue star dot. Note that from now on we will refer to Figure 8 when we deal with a stencil of size $n = 9$ and Figure 9 when $n = 25$. As for 1D, the code goes through every single grid points as an evaluation point and takes the $n-1$ nearest neighbors (stencil) to compute the results. It becomes therefore clear that computational time can become large with great values of $N$. Next come the LS grid points represented by the black dots. For $n = 9$ we can distinguish five cases i.e. $LS = \{0, 4_1, 16_1, 36_1, 8_{10}\}$, we reintroduce the notation $X_Y$ meaning that we take the $X$ nearest LS points to the the evaluation point in a LS grid where the distances between each initial grid points has been divided equally by $Y$ . Again, this is necessary since we will later see that both the number of LS points and their distance to the evaluation point played a role in the results. Therefore in the case with $LS = 8_{10}$ the distance between each initial grid point has been divided by 10 to create the "LS grid" and then the 8 closest neighbors points to the evaluation point are taken to create the new stencil. In the case $n = 25$, we have four cases i.e. $LS = \{0, 48_1, 228_3, 8_{25}\}$

### 5.2.2 Analysis of Results in Two Dimensions

Figures 11 through 16 show the results of tests ran in 2D with RBF-FD-LS using different stencil sizes $n$ and number of $LS$ points in two dimensions. Each

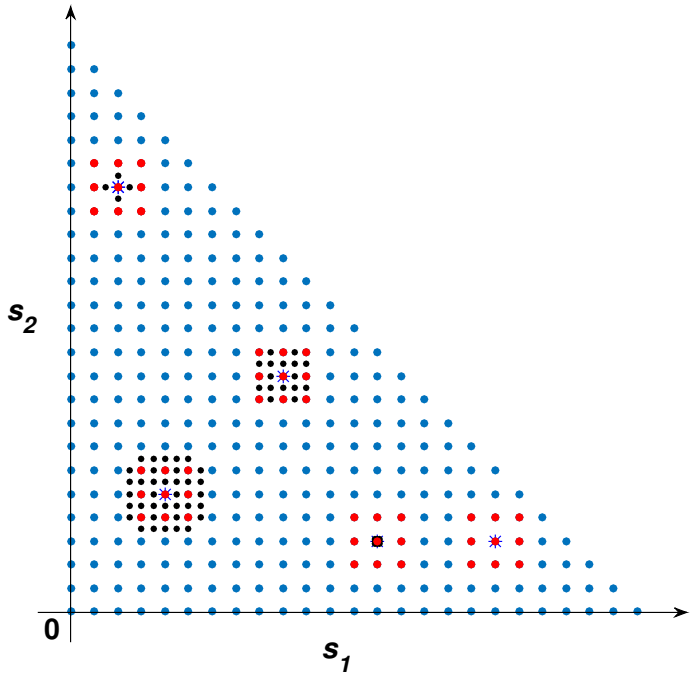



Figure 8: 2D grid: $N = 25$, $n = 9$, $LS = \{0, 4_1, 16_1, 36_1, 8_{10}\}$.

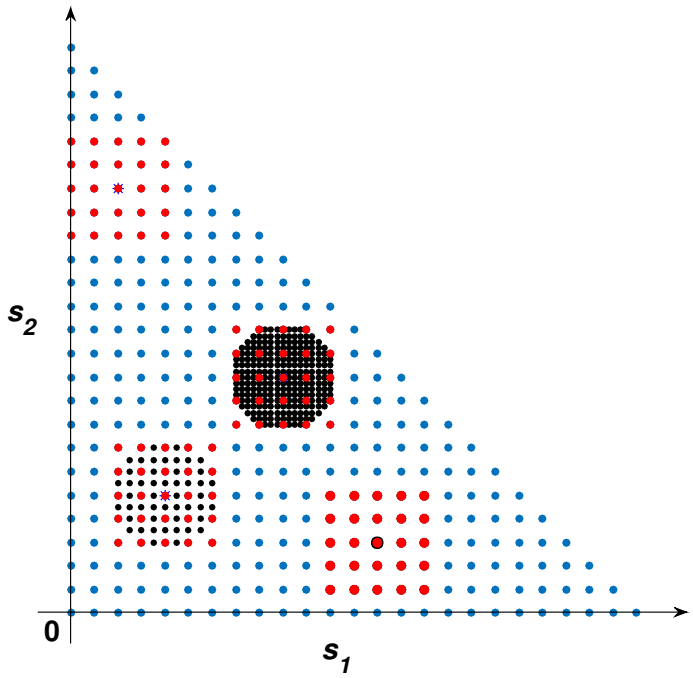



Figure 9: 2D grid: $N = 25$, $n = 25$, $LS = \{0, 48_1, 228_3, 8_{25}\}$.

graph brings out the maximum error obtained for one hundred values of $\varepsilon \in [10^{-4}, 10^{4}]$. Furthermore, every graph shows the results for three different sizes of grid $N = \{60, 80, 100\}$. Note that the size of $M$ is always kept to 1000. Finally, every graph will be divided in four zones for a simpler analysis, those four zones are set up as shown in Figure 10. The first graph in Figure 11 shows the results for the standard RBF-FD, i.e. without extra points, with a stencil of size $n = 9$. There are four zones of interest. Zone 1 and zone 4 show the results for which the maximum error is quite stable and is not affected by the value of $\varepsilon$ or the size of the grid $N$ (especially zone 4), however, the error is too high to consider the $\varepsilon$ optimal. Zone 3 shows a region where both the error size and the stability in $\varepsilon$ are interesting, however, the goal researched is to have a longer zone 3 before getting to the instability of zone 2. Indeed, the second zone is really unstable and sometimes gives errors greater than $10^{200}$. The second and third graph of Figure 11 show the results for a stencil of size $n = 9$, and $LS = \{4_1, 16_1\}$ respectively as shown in Figure 8. There are several changes in the results that can be observed. First of all, zone 1 is slightly longer but more important it has moved down from $10^{0}$ to $10^{-1}$ which is quite an improvement. The largest change occurs in zone 2, where the use of LS points seems to have stabilized a lot the errors, especially for $N = 60$ and 80. Indeed, the lines do not have any jump above $10^{0}$. However, zone 3 appears to be a little bit shorter than without LS points, which is the opposite of the goal. Finally, zone 4 does not show any major changes. There are no big differences between the results with $LS = 4_1$ and $LS = 16_1$, however when seeing the results for $LS = 36_1$ (first graph Figure 12), it appears that zone 3 gets a lot smaller. This could be the results of taking LS points that are outside the initial stencil, as shown in the grid in Figure 8.

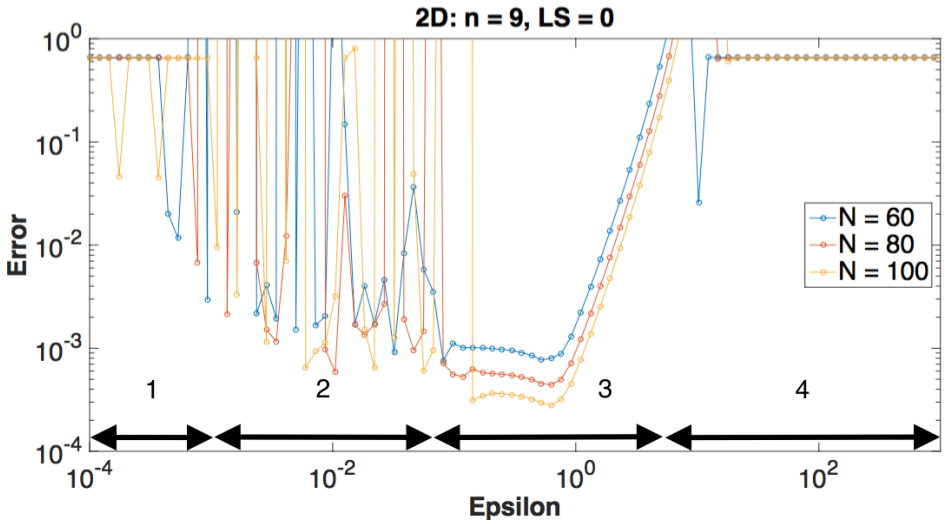



Figure 10: Zones of analysis for 2D results

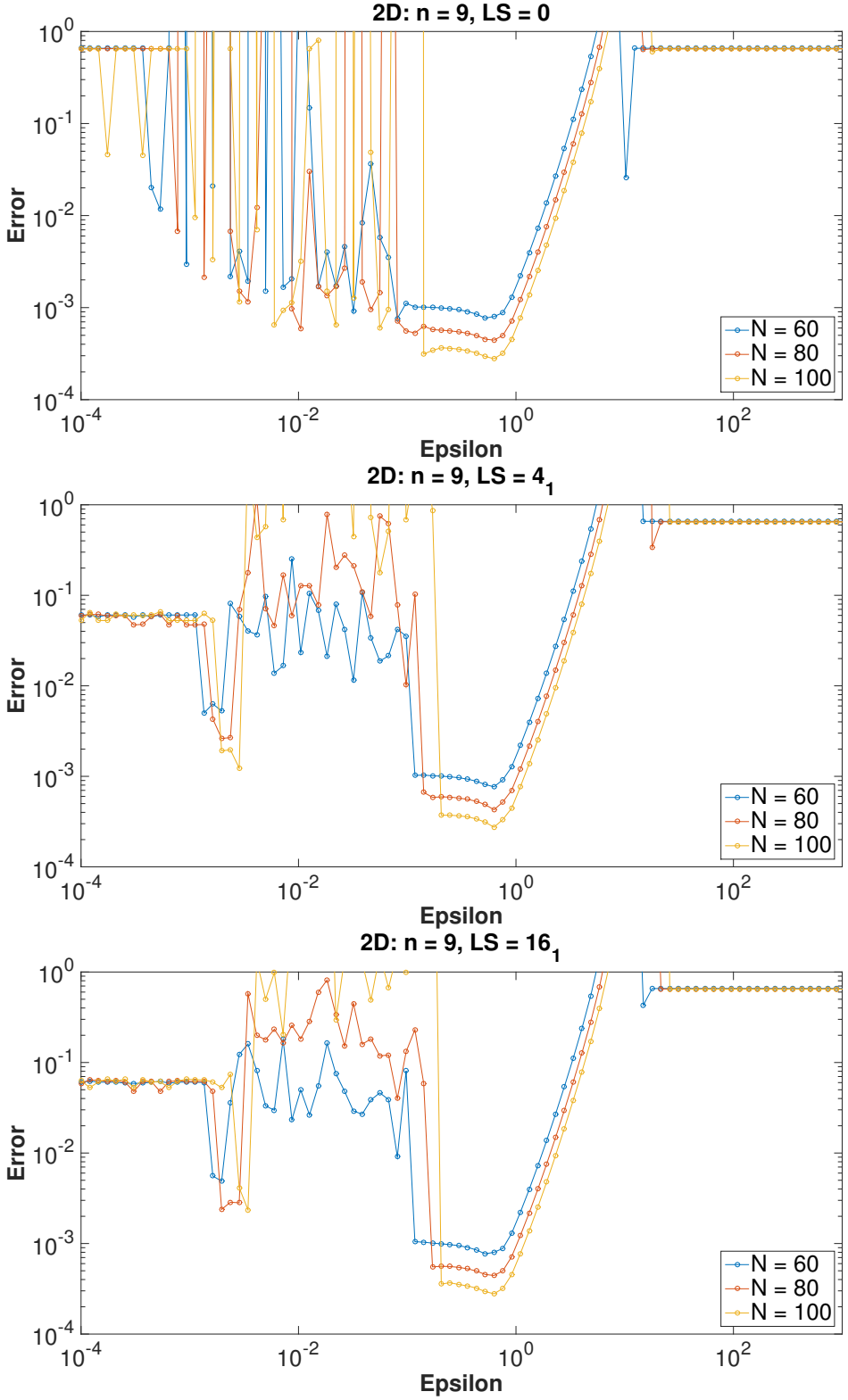



Figure 11: 2D Results: $n = 9$, $LS = \{0, 4_1, 16_1\}$.

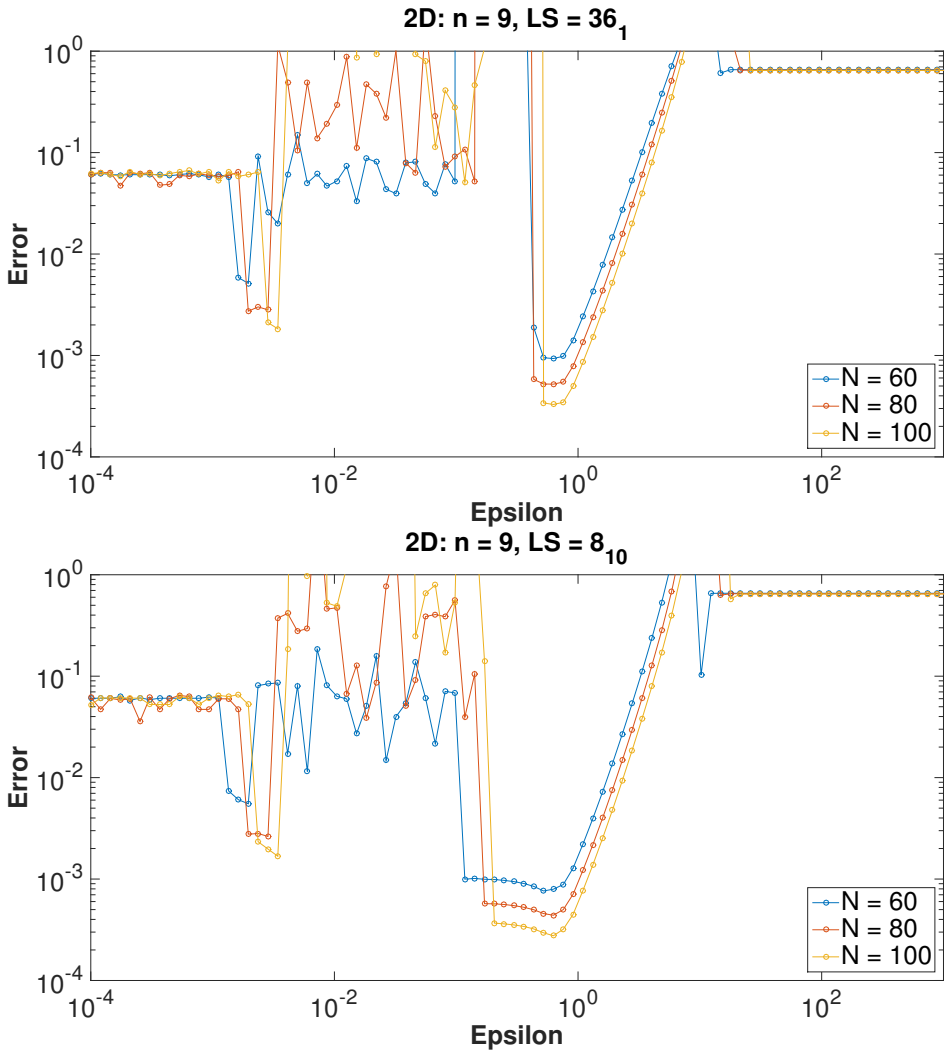



Figure 12: 2D Results: $n = 9$, $LS = \{36_1, 8_{10}\}$.

The second graph of Figure 12 displays results with a smaller number of LS points (the 8 points that encircle the central point) but closer to the evaluation point. Indeed, the LS grid now contains 10 points in between each two grid points. It appears that the results are close to the ones obtained with $LS = \{4_1, 16_1\}$ points, therefore zone 3 is back to a correct size but still smaller than in the standard RBF-FD case. The small difference with the other two figures is that zone 1 is lightly longer. Finally, a test is ran with a smaller grid $N$ equals to 15, or 120 grid points, and respectively $LS = \{0, 16_1, 8_{10}\}$. It is interesting to see that with LS points the errors are stabilized, that is to say without any jump outside the graph limits, contrary to the resulting line of standard RBF-FD that shows instability when $\varepsilon$ get smaller than $10^{-2}$. Therefore, it seems that the LS version of the code stabilizes very well the results for small grid size. It also looks like the number of LS points does not make a big difference, at least in this case.

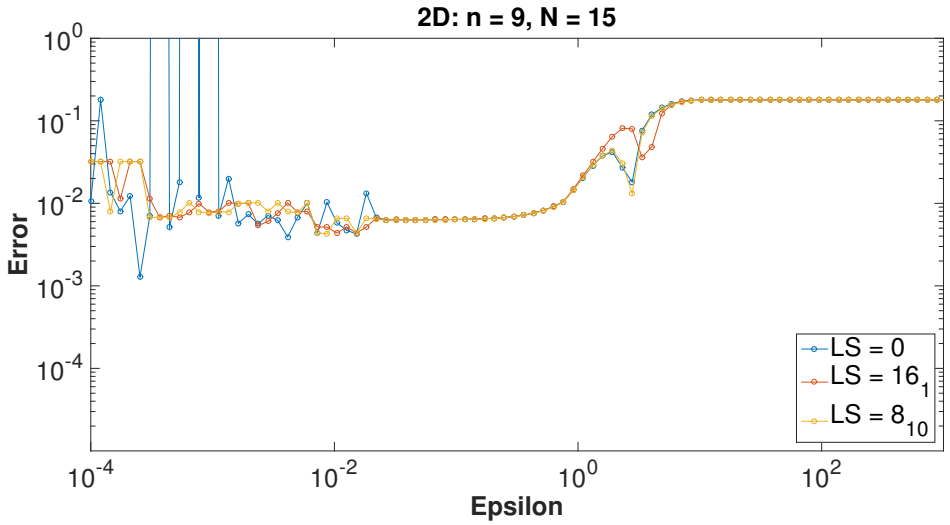



Figure 13: 2D Results for $N = 15$, $n = 9$, and LS $= \{0, 16_1, 8_{10}\}$.

We will now observe the results obtained from tests ran on the 2D function for a stencil of size $n = 25$. The first graph of Figure 14 the test has been run with standard RBF-FD, that is without extra points, as shown by the grid in Figure 9. More or less the same analysis can be drawn as from the case with $n = 9$. Errors are very unstable for $\varepsilon$ smaller than $10^0$. Zone 3 gets an interesting accuracy of $10^{-4}$ but with almost no stability. Overall, results are very bad for $\varepsilon$ smaller than $10^0$. The other graphs in Figures 14 and 15 show the results on the same test ($n = 25$) but with LS points, i.e $LS = \{48_1, 228_3, 8_{25}\}$ that have been added in the same manner as described in section 5.2.1. Again, results can be compared with the ones obtained with $n = 9$. Indeed, with only 1 added LS point, the results are already considerably different from the non-LS version. Zone 1 and 2 are now much more stable and without any jumps outside the graph's limits with an accuracy of less than $10^{-1}$. However, the same issue encountered with $n = 9$ happens i.e the accuracy of the errors is not as good as without LS. Note that the second and third graph of Figure 14 the extended stencil contains a big number of points taken from the LS grid and in Figure 15 there are only the points encircling the evaluation point. This is because, it has been noticed through different experiments that the number of extra points does not change the results much but the closest they are to the evaluation point the better it seems to get. Indeed, in the results with $LS = 8_{25}$, we can see that for $N = 60$ there are more points in zone 3, it is not a huge difference but it still looks more stable with only one jump outside the graph's limits. Finally, a test is ran with a smaller grid $N$ equals to 15, or 120 grid points, and respectively $LS = \{0, 48_1, 8_{25}\}$. Results from this experiment can be seen in Figure 16. It is interesting to see that with LS points the errors are stabilized, that is to say without any jump outside the graph limits, contrary to the resulting line without extra points that shows instability when $\varepsilon$ get smaller than $10^{-2}$. Therefore, it seems that the LS version of the code stabilizes very well the results for small sizes of grid.

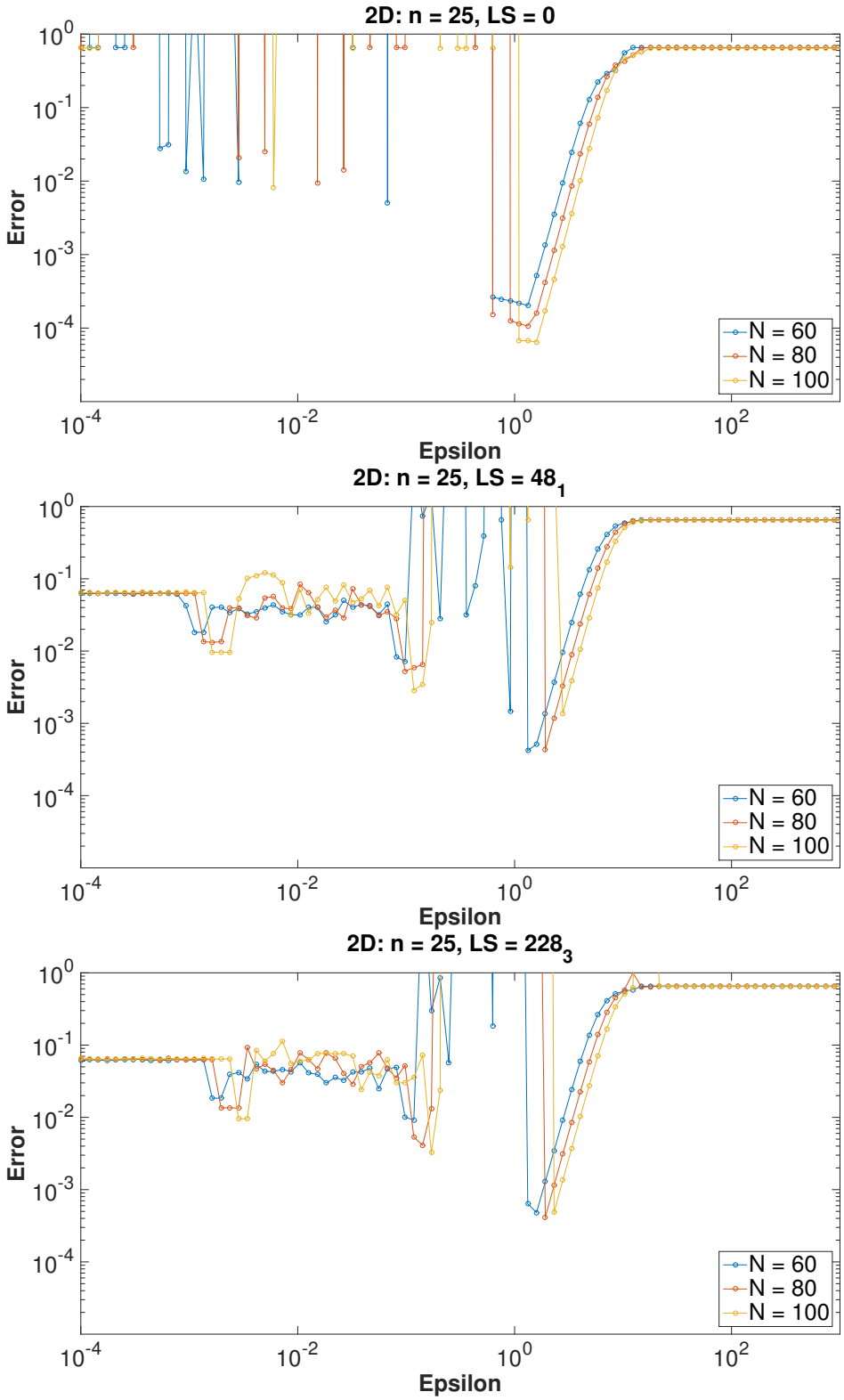



Figure 14: 2D Results: $n = 25$, $LS = \{0, 48_1, 228_3\}$.

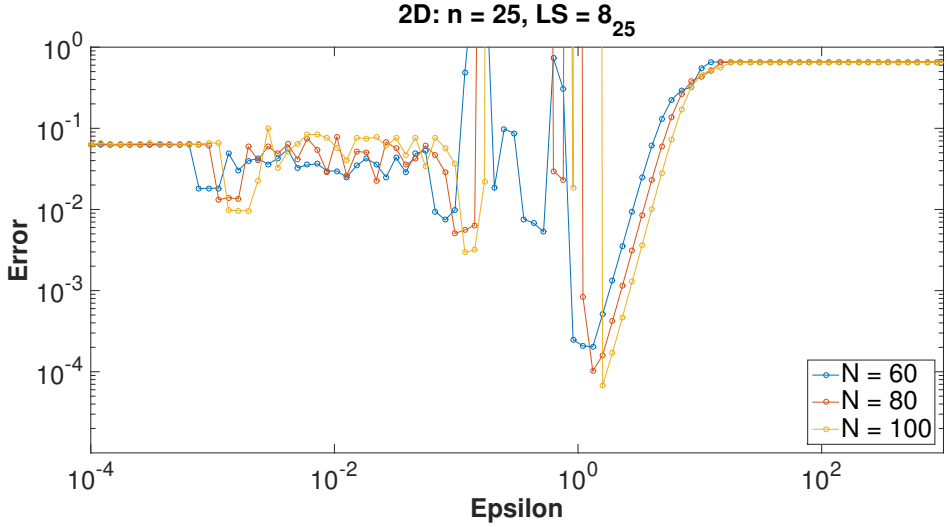



Figure 15: 2D Results: $n = 25$, $LS = \{8_{25}\}$.

Concerning the LS-RBF-FD results, it seems that $LS = \{8_{25}\}$ performs slightly better with no jump outside the graph of Figure 16. This could mean that the closest the LS points are to the evaluation point the more stable the results get. In addition to the increased stability, the results with LS also give some points with a better accuracy of $10^{-3}$.

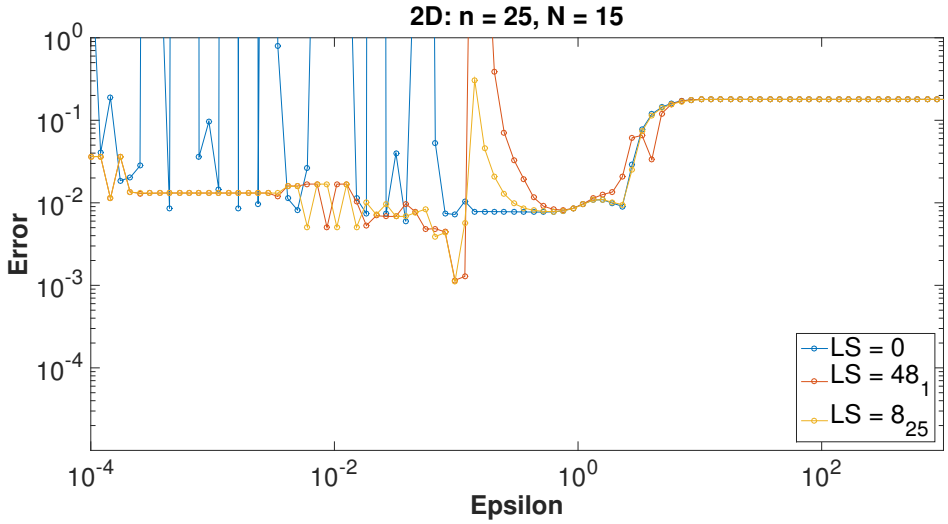



Figure 16: 2D Results for $N = 15$, $n = 25$, and LS $= \{0, 48_1, 8_{25}\}$.

In conclusion, concerning the 2D version, the RBF-FD-LS implementation improves the stability of the results in most cases especially in zone 1 and 2 where stability or even results where very unstable in the standard case. Unfortunately, it is not enough to solve the stability problem in its whole. New zones of stability appear but the accuracy is not as good as what we would like it to be.

However, in some cases both stability and accuracy are improved which means that LS implementation can be a source of progress. The next step would be to be able to compute results with greater sizes of LS grid as it looks like stability and accuracy can be improved but it then becomes extremely time consuming.

# 6 Conclusion

In this thesis we have presented a new implementation of the RBF-FD methods, RBF-FD-LS. The goal of the implementation was to check if this update could bring more stability to the error behavior and/or see if the results get more accurate. It is clear now that the LS implementation of the standard RBF-FD method is bringing some kind of improvement in the stability of the method. Indeed, in most cases stability has been increased clearly and especially when $\varepsilon$ becomes smaller than $10^0$. When $N$ is reasonably small, results are even better with an error behavior almost stable. However, the error behavior does not reach the goal wished for - instability does not disappear totally. This is especially true for large values of $N$. Further experiments such as making the LS grid even smaller could help improving the results, but this method then becomes quite slow. Perhaps the use of different grid geometry (e.g. non-equidistant grids) coupled with the LS implementation could also give better results.

# References


[1] Merton, R.C. *Theory of Rational Option Pricing*. The Bell Journal of Economics and Management Science, 4:141-183, 1973.

[2] Boyle, P.P. Broadie, M. and Glasserman, P. *Monte Carlo Methods for Security Pricing*. Applied Numerical Mathematics, 53:1 - 18, 2005.

[3] Black, F. and Scholes, M. *The Pricing of Options and Corporate Liabilities*. Journal of Political Economy, 81:637-654, 1973.

[4] Meyer, G.H. *The numerical valuation of options with underlying jumps*. Acta Mathematica Universitatis Comenianae, 67:69-82, 1998. Comenius University Press.

[5] Tavella, D. and Randall, C. *Pricing Financial Instruments: The Finite Difference Method*. Wiley Series in Financial Engineering, 2000, Wiley.

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

[7] Björk, T. *Arbitrage Theory in Continuous Time* (3.ed.). Oxford University Press, 2009.

[8] Hon, Y.-C. and Mao, X.-Z. *A Radial Basis Function Method for Solving Options Pricing Models*. Journal of Financial Engineering, 8:31-49, 1999.

[9] Hirsa, A. *Computational Methods in Finance*. Chapman and Hall/CRC Financial Mathematics Series, 2012.

[10] Almendral, A. and Oosterlee, C.W. *Numerical Valuation of Options with Jumps in the Underlying*. Applied Numerical Mathematics, 53:1 - 18, 2005.

[11] Hull, J.C. *Options, Futures, and other Derivatives* (8.ed.). Prentice Hall, 2012.

[12] Kemna, A.G.Z. and Vorst, A.C.F. *A Pricing Method for Options Based On Average Asset Values*. Journal of Banking and Finance, 14:113 - 129, 1990.

[13] Moon, K.-S. *Efficient Monte Carlo Algorithm for Pricing Barrier Options*. Communications of the Korean Mathematical Society, 23(2):285 - 294, 2008.

[14] Seydel, R. *Tools for Computational Finance* (4.ed.). Springer, 2009.

[15] Tolstykh, A.I. *On using RBF-based differencing formulas for unstructured and mixed structured-unstructured grid calculations*. In Proc. of the 16th IMACS World Congress, 228:4606 - 4624, 2000.

[16] Tolstykh, A.I. *On using RBF-based differencing formulas for unstructured and mixed structured-unstructured grid calculations*. In Proc. of the 16th IMACS World Congress, 228:4606 - 4624, 2000.

# A Appendix

## A.1 List of Figures