Radial Basis Function Methods for Solving Vanilla and Exotic Options:

from Univariate to Multivariate Dimensions

By:

CHU, Hau Man

GUO, Xu

YEH, Ming Ming

May 2012

Contents

Abstract 4

Basic Sections (Chapter 1-4)

1. Introduction for Basic Section 4

   1.1 Theory 4

   1.2 Development 5

2. Methodology for Basic Section 6

   2.1 Meshfree Approximation 6

   2.2 Eigenvalue Representation 10

   2.3 Time Stepping Procedure 11

3. Numerical Computations for Basic Section 13

   3.0 Structure of Program 13

   3.1 Example for European Option 15

   3.2 Example for American Option 22

   3.3 Example for Cash-or-Nothing Option 25

   3.4 Example for Asset-or-Nothing Option 27

4. Conclusion for Basic Section 29

5. Extension I - Using RBF for Multi-Asset Option 30

   5.1 Introduction and Methodology 30

      5.1.1 Option on n Assets 31

      5.1.2 Rainbow Option on Two Assets 33

   5.2 Numerical Computation 35

   5.3 Conclusion 38

6. Extension II - Including Time as Radius of the Radial Kernel 38
6.1 Introduction and Methodology 38
6.2 Numerical Computation 44
6.3 Conclusion 45
7. References 45
8. Appendix 46
8.1 Matlab Codes for Basic Section 46
8.2 Matlab Codes for Chapter 5 52
8.3 Matlab Codes for Chapter 6 54

Abstract

Since the last decades of the 20th century, the radial basis functions (RBF) plays an important role in solving different types of PDEs with many advantages such as mesh-free-property, accuracy, stability and efficiency etc. There are three general areas of study for the RBF: Theory, Computation and Applications. Theory includes the concept of reproducing Hilbert spaces and positive definite kernels in functional analysis, as well as some other concepts in numerical and stochastic analysis. Computation concerns the selection of parameters, as well as construction of stable and efficient algorithms. Applications of the RBF include solving PBEs such as the notable Black Scholes Equation, Cauchy problems, and problems in engineering design etc.

1 Introduction

1.1 Theory

Before we focus on how to solve PDEs by using Radial Basis Function Methods, we will start our discussion on the theoretical basis. A Kernel is defined as a real-valued function of two variables[1]:

$$\phi: \Omega \times \Omega \to \mathbb{R}, where\ \Omega \subseteq \mathbb{R}^d \tag{1.1}$$

while Ω can be rather general, say the Riemannian Manifolds. In this paper, however, Ω is restricted to the subset of a finite dimensional Euclidian space $\mathbb{R}^d$. For some specific chosen points $\mathbf{x}=\{x_1,\ldots,x_N\} \subseteq \Omega$, the associated kernel matrix K of Φ is

$$\mathrm{K_{ij}} = \phi(x_i, x_j), 1 \leq i, j \leq N \tag{1.2}$$

In this paper, we will use the same set of points **x** for both x_i and x_j, If Φ is symmetric, as well as that K is a positive definite matrix (i.e. $x^T Kx$ >0 for any vector x) for any choice of N and $\{x_1,\ldots,x_N\}$, then Φ is called a positive definite kernel.

Let $H(\Phi, \Omega)$ be a real Hilbert space of functions defined on $\Omega \in R^d$ with inner product $\langle \cdot, \cdot \rangle_H$. A function f is called a reproducing kernel[2] of $H(\Phi, \Omega)$ if

[1] Fasshauer et al (2011)

$$\begin{cases} \phi(\cdot, x) \in \mathrm{H}, \forall \mathrm{x} \in \Omega \\ f(x) = \langle f, \phi(\cdot, x) \rangle_{H,} \forall \mathrm{f} \in \mathrm{H} \text{ and } \forall \mathrm{x} \in \Omega \end{cases} \tag{1.3}$$

A Hilbert space of functions which admits a reproducing kernel is called a reproducing kernel Hilbert space (RKHS), and from (1.3) we can recognize why the RKHS have such a name[3].

There are different types of positive definite kernels, while one of the important examples is the Radial Basis Function (RBF). ϕ is called a Radial Basis Function if and only if the value of function is only depending on the radius, i.e.

$$\phi_y(x) = \phi(\|x - y\|), \forall \mathrm{x}, \mathrm{y} \in \Omega \tag{1.4}$$

1.2 Development

As one of the notable RBFs, the Hardy's Multiquadric (MQ) was introduced by Hardy (1971) to approximate 2-dimensional geographical surfaces. The RBF is then perceived to be a new powerful tool to solve the boundary problems of PDEs. Comparing with the traditional methods like Finite element and Finite Difference method, the RBF method do not depend on the mesh, which allows a large flexibility for the method to deal with the PDEs problem. In Franke's (1982) review paper, a number of 29 RBF interpolation schemes was under review and the MQ is viewed as one of the best methods. The RBF method was further enhanced by Hon et al. (1999), and the RBF was applied to solve the well-known Black Scholes equation, while a very good result was obtained in the pricing of the European and American type options.

From Chapter 2 to 4 of this article, our work is mainly based on Hon's (1999) paper, with extension from the European vanilla options to the exotic options. Hon's paper focused mainly on Hardy's MQ, and we will also study the result of Gaussian Kernels and the Thin Plate Spline kernels. We will also review the Parallel Time Stepping Approach, which is an

[2] The concept was first introduced by Aronszajn (1950)

[3] P. 103, Fasshauer (2007)

improvement suggested by Fasshauer et al. (2004). In Chapter 5, we will extend the problem from single asset option to Multi-Asset options. In Chapter 6, we come back to the single asset option, but will try to include time as the radius of the kernel, which is not previously treated as a normal practice due to the different characteristics between time and space.

2 Methodology

2.1 Meshfree approximation

Main Program: ***RBF_OptionPricing.m***

To illustrate how to apply the radial basis functions as for options pricing, we first consider the values of the European options which can be solved by the following Black-Scholes equation:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0 \tag{2.1}$$

where r is the risk-free interest rate, σ is the volatility of stock price S, and $V(S,t)$ is the option value of the underlying asset (stock) at time t and stock price S.

The terminal payoff valuation gives the initial condition:

$$V(S,T) = \begin{cases} max\{X - S, 0\} & for\ a\ put \\ max\{S - X, 0\} & for\ a\ call \end{cases} \tag{2.2}$$

where T is the terminal time and X is the strike price of the option. A simple transformation of $S = e^y$ changes the Equation (2.1) and Condition (2.2) to:

$$\frac{\partial U}{\partial t} + \frac{1}{2}\sigma^2 \frac{\partial^2 U}{\partial y^2} + \left(r - \frac{1}{2}\sigma^2\right)\frac{\partial U}{\partial y} - rU = 0 \tag{2.3}$$

(i.e. with constant coefficient) with initial condition:

$$U(y,T) = \begin{cases} max\{X - e^y, 0\} & for\ a\ put \\ max\{e^y - X, 0\} & for\ a\ call \end{cases} \tag{2.4}$$

Now propose the numerical scheme by using meshfree approximations. Given *N* distinct data points (centres), y_j , we interpolate the unknown function U by the following radial basis functions ϕ_j:

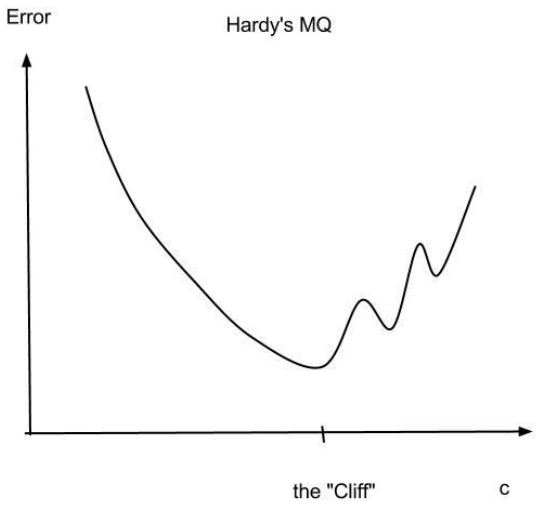
$$U(y,t) \approx \sum_{j=1}^{N} \alpha_j(t)\ \phi\left(|y - y_j|\right) \tag{2.5}$$

where α_j(t)s are unknown coefficients depending on t, $\phi_j(y) = \phi(|y - y_j|)$ denotes the radial basis functions, in which $|y - y_j|$ is the Euclidean distance between two collocation points y_j. In this paper, we will use the 3 most common radial basis functions for our analysis:-

Relevant sub-function: ***choice_of_kernel.m***

$$\phi_j(y) = \begin{cases} \sqrt{(y - y_i)^2 + c^2} & for\ Hardy's\ MQ \\ e^{-c^2(y-y_j)^2} & for\ Gaussian \\ (y - y_j)^c \log(|y - y_j|) & for\ thin\ plate\ spline \end{cases} \tag{2.6}$$

where c is the shape parameters. The notable Hardy's MQ is one of the most popular kernels in RBF methods; the Gaussian is a very classic function in Mathematics named after the Prince of Mathematicians, Carl Friedrich Gauss. For the thin plate spline, the value of c is 1 in the original paper, but we note that by increasing the power we would obtain a better result. For all these 3 cases, the accuracy of the RBFs are highly sensitive by the choices of shape parameter c. For the Hardy's MQ, it is even perceived that the error would be reduced by increasing the value of c under reaching the "cliff", which can be briefly explained by the following graph:-



After reaching the “Cliff”, the Hardy’s MQ would behave very bad, whereas the optimal values of c are still under research. In this paper we will use c to be $4d_{min}$ for Hardy’s Multiquadric (MQ), $e^{-6}d_{min}$ for Guassian, and 2 for thin plate spline.

In (2.3), we choose N collocation points,y_j, and yield the following systems of N equations:

$$\frac{\partial U(y_i,t)}{\partial t} + \frac{1}{2}\sigma^2\frac{\partial^2 U(y_i,t)}{\partial y^2} + \left(r - \frac{1}{2}\sigma^2\right)\frac{\partial U(y_i,t)}{\partial y} - rU(y_i,t) = 0 \qquad (2.7)$$

where

$$\frac{\partial U(y_i,t)}{\partial t} = \sum_{j=1}^{N}\frac{d\alpha_j(t)}{dt}\,\phi(y_i,y_j) \qquad (2.8)$$

$$\frac{\partial U(y_i,t)}{\partial y} = \sum_{j=1}^{N}\alpha_j(t)\frac{\partial\phi(y_i,y_j)}{\partial y} \qquad (2.9)$$

$$\frac{\partial^2 U(y_i,t)}{\partial y^2} = \sum_{j=1}^{N}\alpha_j(t)\frac{\partial^2\phi(y_i,y_j)}{\partial y^2} \qquad (2.10)$$

From (2.6), we have:-

for Hardy’s MQ:

$$\frac{\partial \phi(y_i, y_j)}{\partial y} = \frac{(y_i - y_j)}{\sqrt{(y_i - y_j)^2 + c^2}} \tag{2.11}$$

$$\frac{\partial^2 \phi(y_i, y_j)}{\partial y^2} = \frac{1}{\sqrt{(y_i - y_j)^2 + c^2}} - \frac{(y_i - y_j)^2}{((y_i - y_j)^2 + c^2)^{\frac{3}{2}}} \tag{2.12}$$

for Gaussian:

$$\frac{\partial \phi(y_i, y_j)}{\partial y} = -2c^2(y - y_j)\, e^{-c^2(y-y_j)^2} \tag{2.13}$$

$$\frac{\partial^2 \phi(y_i, y_j)}{\partial y^2} = -2c^2 e^{-c^2(y-y_j)^2} + 4c^4(y - y_j)^2\, e^{-c^2(y-y_j)^2} \tag{2.14}$$

for thin plate spline (for c>1):

$$\frac{\partial \phi(y_i, y_j)}{\partial y} = (y - y_j)^{2c-1} + 2c(y - y_j)^{c-1} \log(|y - y_j|) \tag{2.15}$$

$$\frac{\partial^2 \phi(y_i, y_j)}{\partial y^2} = (4c - 1)(y - y_j)^{2c-2} + (4c^2 - 2c)(y - y_j)^{c-2} \log(|y - y_j|) \tag{2.16}$$

In matrix form, (2.7) becomes

$$L\dot{\alpha} + \frac{1}{2}\sigma^2 L_{yy}\,\alpha + \left(r - \frac{1}{2}\sigma^2\right) L_y \alpha - rL\alpha = 0 \tag{2.17}$$

where α denotes the vector of the unknown coefficientsα_j, while $\dot{\alpha}$ is $\frac{d\alpha_j(t)}{dt}$; $L, L_y, \ and L_{yy}$ are the $N \times N$ matrices with entries $\phi_j(y_i), \frac{\partial \phi(y_i, y_j)}{\partial y}, and \frac{\partial^2 \phi(y_i, y_j)}{\partial y^2}$ given above, respectively. The matrix L is invertible due to the collocation points are distinct, and thus equation (2.17) can be rewritten as the linear homogenous ordinary differential equation system with time dependent coefficients.

$$\dot{\alpha} = -L^{-1}\left[\frac{1}{2}\sigma^2 L_{yy}\alpha + \left(r - \frac{1}{2}\sigma^2\right) L_y \alpha - rL\alpha\right] \equiv P\alpha \tag{2.18}$$

where

$$P = rI - \frac{1}{2}\sigma^2 L^{-1} L_{yy} - \left(r - \frac{1}{2}\sigma^2\right) L^{-1} L_y \tag{2.19}$$

2.2 Eigenvalue Representation

Relevant sub-function: ***Eigenvalue_Representation.m***

We can note that for fixed collocation points y_j, Equation (2.18) is a linear system of first order homogeneous ordinary differential equations with constant coefficient. It is a textbook question and it is proven[4] that if all the eigenvectors $\overrightarrow{w_i}$ of the matrix P are linearly independent, the analytical solution of Equation (2.18) would be given by:-

$$\alpha(t) = \sum_{i=1}^{N} k_i e^{\lambda_i t} \overrightarrow{w_i} \tag{2.20}$$

By substituting t= T which is the terminal time, we have

$$\alpha(T) = \sum_{i=1}^{N} k_i e^{\lambda_i T} \overrightarrow{w_i} = W \begin{bmatrix} k_1 e^{\lambda_1 T} \\ \vdots \\ k_N e^{\lambda_N T} \end{bmatrix} and\ La(T) = \vec{U}_0, \text{where } W = [\vec{w}_1 \quad \dots \quad \vec{w}_N] \tag{2.21}$$

Of course $\vec{U}_0 = [U(y_N, T) \dots U(y_N, T)]'$ (i.e. we focus on Put Option) which is the condition at maturity. So we have

$$\begin{bmatrix} k_1 e^{\lambda_1 T} \\ \vdots \\ k_N e^{\lambda_N T} \end{bmatrix} = W^{-1} L^{-1} \vec{U}_0 \tag{2.22}$$

or by using the notation of MATLAB's element-wise division:

$$\vec{k}_{\text{coefficient}} = W^{-1} L^{-1} \vec{U}_0 .* \begin{bmatrix} e^{-\lambda_1 \tau} \\ \vdots \\ e^{-\lambda_N \tau} \end{bmatrix} \tag{2.23}$$

[4] P. 349, Farlow (2006)

Then the Option price is given by below:

$$U(y,0) = \sum_{j=1}^{N} \alpha_j(0)\phi_{y_j}(y) = \left(W\vec{\mathrm{k}}_{\mathrm{coefficient}}\right)' \vec{\phi} \tag{2.24}$$

For the above, the eigenvalues λ_i and eigenvectors $\vec{w}_i$ of matrix P can be computed by standard matrix computation in MATLAB. The inverse matrix W^{-1} in equation (2.22) is computed by using Gaussian elimination with partial pivoting. Now the constant vector $\vec{k}$ in equation (2.23) can be obtained, and therefore we can obtain U.

2.3 Time Stepping Procedure

Relevant sub-function: ***choice_of_int_sch.m***

Although we will also discuss the numerical analysis about the eigenvalue representation (2.24) in our later section, the following Time Stepping Procedure would play the most important role in this paper to solve the homogenous ODE system (2.18), due to its easy extension/ application to Chapter 5 and Chapter 6. For Time Stepping Procedure, first we look at the explicit first order backward difference time integration scheme (BD1)

$$\alpha_n = \alpha_{n-1} - \Delta t P \alpha_{n-1} \tag{2.25}$$

There is also the explicit second order backward difference time integration scheme (RK2)

$$F_1 = -\Delta t P \alpha_{n-1}$$

$$F_2 = -\Delta t P(\alpha_{n-1} + 0.5F_1)$$

$$\alpha_n = \alpha_{n-1} + 0.5(F_1 + F_2) \tag{2.26}$$

Explicit fourth order backward time integration scheme (RK4) would give an even better numerical result:-

$$F_1 = -\Delta t P \alpha_{n-1}$$

$$F_2 = -\Delta tP(\alpha_{n-1} + 0.5F_1)$$

$$F_3 = -\Delta tP(\alpha_{n-1} + 0.5F_2)$$

$$F_4 = -\Delta tP(\alpha_{n-1} + F_3)$$

$$\alpha_n = \alpha_{n-1} + \frac{F_1 + 2F_2 + 2F_3 + F_4}{6} \qquad (2.27)$$

The implicit backward time integration scheme is also a method mentioned in the original paper:-

$$\alpha_n = \alpha_{n-1} - \Delta tP\,[\theta\,\alpha_{n-1} + (1 - \theta)\alpha_n]$$

or in another form, i.e.

$$P_1 = I_n + (1 - \theta)\Delta tP$$

$$P_2 = (I_n - \theta\Delta tP)$$

$$\alpha_n = {P_1}^{-1}P_2\alpha_{n-1} \qquad (choose\ \theta = 0.5) \qquad (2.28)$$

In this paper, we will choose θ to be 0.5.

The Parallel time stepping scheme is an improvement first introduced by Fasshauer by making use of the Padé approximation[5]:-

$$V_1 = (I_n + \Delta tP)\alpha_{n-1}$$

$$V_2 = (I_n + \frac{2}{5}\Delta tP)\alpha_{n-1}$$

$$V_3 = (I_n + \frac{9}{14}\Delta tP)\alpha_{n-1}$$

$$V_4 = (I_n + \frac{1}{2}\Delta tP)\alpha_{n-1}$$

[5] P.5, Fasshauer et al (2004)

$$\alpha_n = \frac{19}{45}V_1 - \frac{2500}{153}V_2 - \frac{2401}{255}V_3 + \frac{79}{3}V_4 \qquad (2.29)$$

3 Numerical Computations

3.0 Structure of Program

In the following numerical examples, we use our program to compute price of both vanilla and exotic options. For our core sections (i.e. Chapter 1 – 4), our program basically consists of 1 main program and 8 sub-functions as below:-

Main Program: **RBF_OptionPricing.m**

Sub-functions:-

1. **choice_of_standard_data.m** - Instead of changing each parameters one by one, we have constructed 4 sets of values for 4 main sets of standard examples (i.e. European Options, American Options, Random Points and Binary Options). By just changing one value (i.e. datachoice), we can change nearly the values of 9 parameters as below:-

datachoice	1	2	3	4	5
Respective to which examples?	European Put	American Put	Random/ Uniform comparison	Binary Options	Free use
X	10	100	10	15	Free variables
r	0.05	0.1	0.05	0.05	
sigma	0.2	0.3	0.2	0.2	
T	0.5	1	0.5	0.25	
n	81	101	61	101	
m	30	100	5	60	
Smin	1	1	1	1	
Smax	30	exp(6)	30	30	
Index (i.e. S(t=0))	2:2:16	80:5:120	2:2:16	5:1:20	

2. **random_number_generator.m** – For choosing whether the points y_i are uniform or random.
3. **initial_condition.m** – For choice of terminal payoff (i.e. Call, put, or binary option).
4. **type_of_option.m** – For choice of option, including American or European.
5. **choice_of_kernel.m** – For choice of kernel.

6. **transformBack.m** – Once the coefficients of the kernels are known, this sub-function is for transforming the data back in view of the original transformation $S=e^{y}$.
7. **choice_of_int_sch.m** – Choice of integration scheme.
8. **eigenvalue_Presentation.m** – As mentioned before, the time stepping procedure is the main role in this article, while we would only have a slight glance on the eigenvalue representation method. If this sub-function is activated, the eigenvalue representation calculation would be taken after the original calculation, for comparison purpose.

The idea of the main program and the sub-functions can be summarized by the following 2 figures:-

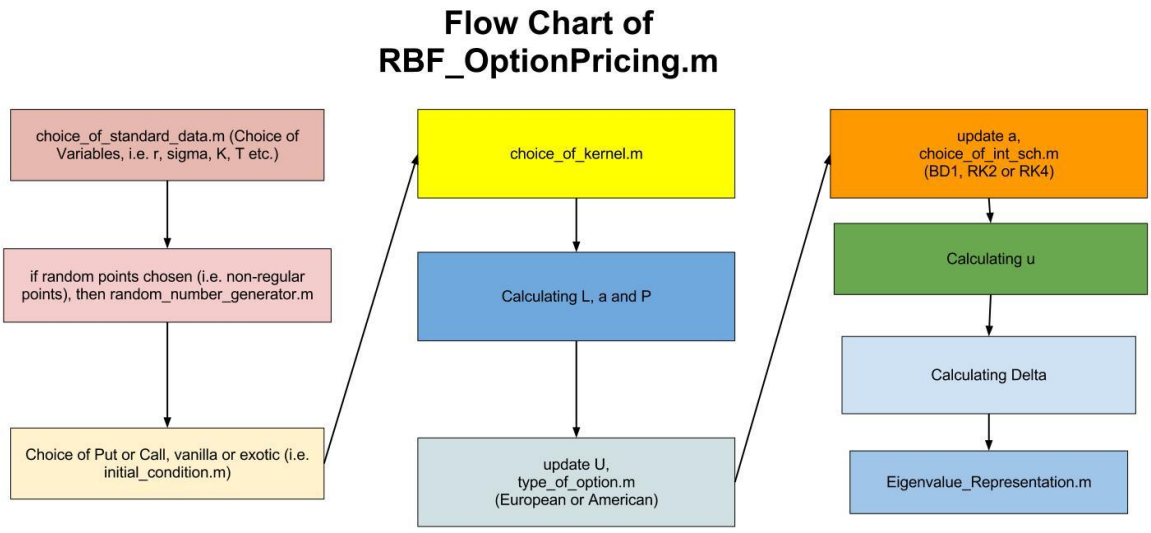


Table of Indexes

Name of choice	datachoice	randomchoice	putorcall
respective function	choice_of_standard_data.m	random_number_generator.m	initial_condition.m type_of_option.m
Choices- Codes	Euro. Put Example - **1** Am. Put Example - **2** Random Example - **3** Binary Options - **4** Free Variables - **5**	Random - **1** Uniform - **0**	Put - **1** Call - **10** Cash-or-Nothing Put - **2** Cash-or-Nothing Call - **20** Asset-or-Nothing Put - **3** Asset-or-Nothing Call - **30**

Name of choice	kernelchoice	optionchoice	intschchoice	eigen_rep_choice
Main function	choice_of_kernel.m transformBack.m	type_of_option.m	choice_of_int_sch.m	eigenvalue_Presentation.m
Choices–Codes	Hardy's MQ - **1** Gaussn. Spl. - **2** Thin Plate Spl. - **3**	European - **10** American - **20**	BD1 - **1** RK2 - **2** RK4 - **4** Implicit - **5** Paral. T.S. - **6**	Original Method only - **1** Both Original & Eigen - **2**

As can be seem from the above flow chart, 7 "choice variables" (i.e. kernelchoice etc.) are contained in the main program **RBF_OptionPricing.m**. With any selection of the combination of the "choice variables", the main program will execute each respective sub-functions corresponding to the choices of index when running the program. Upon the execution, our program will perform the numerical computation and output the result.

3.1 Example for European Option

European options are constrained by the following boundary conditions:

$$\begin{cases} V(0,t) = Xe^{-r(T-t)}, & V(S,t) \to 0 \; as \; S \to \infty \quad for \; a \; put \\ V(0,t) = 0, & V(S,t) \to 0 \; as \; S \to \infty \quad for \; a \; call \end{cases} \tag{3.1}$$

The exact solution of equation (2.1) subject to the initial condition (2.2) and the boundary conditions (3.1) can be obtained by:

$$\begin{cases} V(S,t) = Xe^{-r(T-t)}N(-d_2) - SN(-d_1) & for \; a \; put \\ V(S,t) = SN(d_1) - Xe^{-r(T-t)}N(d_2) & for \; a \; call \end{cases} \tag{3.2}$$

, also known as the Black-Scholes formula, where $N(\cdot)$ is the cumulative normal distribution

$$d_1 = \frac{\log\left(\frac{S}{X}\right) + \left(r + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{\tau - t}} \tag{3.3}$$

and

$$d_2 = \frac{\log\left(\frac{S}{X}\right) + \left(r - \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}} \tag{3.4}$$

To illustrate the algorithm's detail, we consider a European put option with $X = 10, r = 0.05, \sigma = 0.20, T = 0.5$, as given by Wilmott (1996). Let $S \epsilon [S_{min}, S_{max}]$ and thus $y \epsilon [\log S_{min}, \log(S_{max})]$. In our computations, we choose $S_{min} = 1$ and $S_{max} = 30$, sufficiently large to satisfy the infinite-end boundary condition (3.1). Also let $\Delta y = \frac{\log\left(\frac{S_{max}}{S_{min}}\right)}{N-1}$ and $y_j = (j-1)\Delta y$ for $j = 1, 2, \dots, N$. Now the matrixes $L, L_y, and L_{yy}$ can be determined by these fixed values of $y_j's$. Additionally, the inverse matrix L^{-1} and matrix P in equation (2.18) are obtained by using Gaussian Elimination with partial pivoting. The initial condition (2.4) for a put:

$$U(y_i, T) = \max\{X - e^{y_i}, 0\}, i = 1, 2, \dots, N \tag{3.5}$$

gives the initial vector U_0 and the initial vector $\alpha_0 = L^{-1}U_0$ (note that α_0 is in fact $\alpha(T)$, i.e. α at the terminal time) is fixed. The unknown coefficient vector α can be computed directly using the explicit (BD1, RK2 and RK4), implicit, and the parallel time integration schemes as expressed in equations (2.25) - (2.29) orderly.

After obtaining α_M, i.e. the last Alpha in the program, the European put option value $V(S,t) = U(\log(S), t))$ can now be obtained by using the equation (2.5) (Relevant sub-function: **TransformBack.m**). In addition to the RBFs method, this gives a global approximation formula for the solution, unlike the finite difference methods provides only on the grid points. With the obtained values of the coefficients α_j, the option values and all its derivatives can be obtained by using the same matrices $L, L_y, and L_{yy}$.

For the iterative approximations of each scheme, let $\Delta t = T/M$. The coefficients $\alpha_n, n = 1, 2, \dots, M$, can then be obtained by using the equations (2.25) - (2.29). For each time step t_n, we update α_n such that the zero-end boundary condition $V(0,t) = Xe^{-r(\tau - t)}$ (i.e. Put of (3.1))is satisfied. The following shows the procedure of each time:

Boundary update procedure for European put options

Step 1: Compute $U_n = L\alpha_n$.

Step 2: Update the first element $U_n(1) = Xe^{-rn\Delta t}$.

Step 3: Update α_n by $L^{-1}U_n$.

Example 3.1.1:

To investigate the time integration errors of the numerical solutions from various numerical methods to obtain the numerical approximations for the option values, we apply the explicit first order Backward difference (BD1), second order Rundge-Kutta (RK2), forth order Rundge-Kutta (RK4), Implicit method, and the parallel time integration schemes, as given in equations (2.25) – (2.29) in order. Numerical results of approximation by using above methods are shown in Table 3.1.1.

Noted that the explicit time integration scheme is not so stable, and the total number of time steps (M) must be roughly half of the total number of collocation points (N) for convergences. Here we take $N = 81, and\ M = 30$ in our computation. However, the stability problem no longer exists when using the implicit time integration method. The column of the implicit time integration method shows almost the same result as when $M = 5$, which causes Δt is comparatively much larger than when $M = 30$.

Valuing the European options by using the finite difference method (FDM) and the finite element method (FDM), which require the support of mesh, is straightforward. However, it will become not trivial tasks when mesh generation over two or three dimension domains.

Table 3.1.1: Comparison of accuracy for European put options

$X = 10, r = 0.05, \sigma = 0.20, T = 0.5, S_{min} = 1, S_{max} = 30, N = 81, and\ M = 30,$

Hardy's MQ with c $= 4$dy

Stock S	Exact	RBF method				
		Explicit			Implicit ($\theta = 0.5$)	Parallel Time
		BD1	RK2	RK4		
2	7.7531	7.7530	7.7531	7.7531	7.7531	7.7531
4	5.7531	5.7530	5.7531	5.7531	5.7531	5.7531
6	3.7532	3.7530	3.7531	3.7532	3.7532	3.7532
8	1.7987	1.7966	1.7975	1.7983	1.7983	1.7983
10	0.4420	0.4435	0.4423	0.4412	0.4412	0.4412
12	0.0483	0.0480	0.0481	0.0482	0.0482	0.0482
14	0.0028	0.0024	0.0026	0.0028	0.0028	0.0028
16	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
RMSE		**0.0009**	**0.0005**	**0.0003**	**0.0003**	**0.0003**

* datachoice = 1; randomchoice=0; putorcall = 1; kernelchoice = 10; optionchoice = 10; intschchoice = 1 (BD1), 2 (RK2), 4 (RK4), 5 (Implicit), 6 (Parallel Time); eigen_rep_choice=0;

From the above table, we observe that the result of RK4, Implicit Method and Parallel are nearly the same. If we slightly change the N of “datachoice =1” from 81 to 101, we would obtain the followings:-

Table 3.1.1.a: Comparison of accuracy for European put options when N = 101

Stock S	Exact	RK4	Implicit	Parallel Time
2	7.7531	7.7530	7.7531	7.7531
4	5.7531	5.7576	5.7531	5.7531
6	3.7532	3.7573	3.7532	3.7532
8	1.7987	1.8057	1.7990	1.7990
10	0.4420	0.4391	0.4427	0.4426
12	0.0483	0.0546	0.0485	0.0485

14	0.0028	0.0023	0.0028	0.0028
16	0.0001	0.0009	0.0001	0.0001
RMSE		**0.004091**	**0.000270**	**0.000267**

We can obtain that the implicit method works better than RK4 when N=101, while the Parallel time stepping method is another improvement.

Example 3.1.2: To demonstrate the benefit of the proposed RBF method, a truly meshfree computational algorithm, we use the set of 61 (i.e. the N of this example) random collocation points in the interval (0, log(30)) (i.e. Zero to log(S_{max})) by using MATLAB. To insure that all the generated points are distinct, a minimum distance, 0.025 and 0.015, are imposed in the sub-function **random_number_generator.m**. The comparison between 61 uniform and random collocation points for European put options is given in Table 3.1.2.

Table 3.1.2: Comparison of accuracy for European put options by 61 uniform and random collocation points for European put options

$$X = 10, r = 0.05, \sigma = 0.20, T = 0.5, S_{min} = 1, and\ S_{max} = 30, N = 61, M = 5$$

Stock S	**Exact**	**Implicit Time Scheme ($\theta = 0.5$)**		
		Uniform	**Random (min dist = 0.025)***	**Random (min dist = 0.015)***
2	7.7531	7.7531	7.75316	7.8392
4	5.7531	5.7531	5.75308	5.8045
6	3.7532	3.7532	3.75302	3.6158
8	1.7987	1.7995	1.79866	1.8301
10	0.4420	0.4458	0.446646	0.4648
12	0.0483	0.0489	0.052124	0.0305
14	0.0028	0.0028	0.006054	0.0092

16	0.0001	0.0001	0.001646	0.0142
RMSE		**0.0014**	**0.0025**	**0.0623**

*datachoice = 3; randomchoice=0 (Uniform), 1 (Random); putorcall = 1; kernelchoice = 10; optionchoice = 10; intschchoice = 5; eigen_rep_choice=0;

* In **random_number_generator.m**: minimum_distance=0.025 (min dist = 0.025), 0.015 (min dist = 0.015)

As we can see, using the minimum distance (min dist) equals to 0.025 yields more accurate than using min dist = 0.015. We have also observed that when minimum distance is less than 0.01, some of the points are too close to each other and the relative matrix is sometimes nearly singular, while the result is very unstable.

Example 3.1.3: To investigate the most effective and accurate radial basis functions for the valuation of options, we compute the numerical solutions of the European put option by using three common RBFs in an implicit time integration scheme. RBFs that we use in this example are Hardy's MQ, Gaussian Spline, and Thin Plate Spline given by (2.6).

It can be observed that, from Table 3.1.3, these three RBFs with our choices of $\boldsymbol{c}$ provides relative great results for European put option.

Table 3.1.3: Comparison of accuracy for European put options by three different radial basis functions

$$X = 10, r = 0.05, \sigma = 0.20, T = 0.5, S_{min} = 1, S_{max} = 30, N = 81, and\ M = 30$$

Stock S	**Exact**	**Implicit Time Scheme ($\boldsymbol{\theta = 0.5}$)**		
		Hardy's MQ ($\boldsymbol{c = 4d_{min}}$)	**Gaussian Spline** ($\boldsymbol{c = e^6 d_{min}}$)	**Thin Plate Spline** ($\boldsymbol{c = 2}$)

2	7.7531	7.7531	7.7531	7.7531
4	5.7531	5.7531	5.7531	5.7531
6	3.7532	3.7532	3.7532	3.7532
8	1.7987	1.7983	1.7983	1.7982
10	0.4420	0.4412	0.4412	0.4413
12	0.0483	0.0482	0.0482	0.0481
14	0.0028	0.0028	0.0028	0.0028
16	0.0001	0.0001	0.0001	0.0001
RMSE		**0.00031**	**0.00031**	**0.00029**

*datachoice = 1; randomchoice=0; putorcall = 1; kernelchoice = 10 (Hardy's MQ), 20 (Gaussian Spline), 30 (Thin Plate Spline); optionchoice = 10; intschchoice = 5; eigen_rep_choice=0;

As we can see, the Thin Plate Spline have a better result than Hardy's MQ and Gaussian in the above situation, while the result of the latter 2 cases are almost the same. In fact, the output of Hardy's MQ and the Gaussian would have a differences starting from the 6^{th} digit.

Example 3.1.4: When Running our main program **RBF_OptionPricing.m**, if we choose eigen_rep_choice=1 in our main program, the sub-function **Eigenvalue_Representation.m** will be executed, and thus both the values obtained by the Time Stepping Method (i.e. RK2 or RK4 etc.) and the Eigenvalue Method would be calculated for comparison. We take the European Put Option for this example and numerical results are shown in Table 3.1.4.

Table 3.1.4: Comparison of Accuracy for the Eigenvalue Representation Method

$$X = 10, r = 0.05, \sigma = 0.20, \tau = 0.5, S_{min} = 1, S_{max} = 30, N = 81, M = 30)$$

Stock S	Exact	Hardy's MQ		Gaussian	
		RK4	Eig. Rep.	RK4	Eig. Rep.
2	7.7531	7.753100	7.753100	7.753140	7.753070
4	5.5731	5.753100	5.753100	5.753080	5.753080

6	3.3732	3.753180	3.753180	3.753160	3.753160
8	1.7987	1.798330	1.798330	1.798330	1.798330
10	0.442	0.441197	0.441196	0.441198	0.441198
12	0.0483	0.048168	0.048168	0.048168	0.048168
14	0.0028	0.002767	0.002766	0.002766	0.002766
16	0.0001	0.000104	0.000103	0.000103	0.000103
RMSE		0.148655	0.148655	0.148645	0.148645

*datachoice=1; randomchoice=0; putorcall=1; kernelchoice=10 (Hardy's MQ), 20 (Gaussian Spline); optionchoice=10; intschchoice=4; eigen_rep_choice=1;

As we can see, the result obtained by the above Eigenvalue Representation Method is nearly the same as the RK4. The difference occurs only starting from the 6^{th}/ 7^{th} digit.

3.2 Example for American Option

The valuation of the American put options can be treated as a free boundary value problem, and there is no an analytical formula until recently. American options allow for early exercises at any time $t \in [0, T]$ with optimal exercise. Due to the unknown value for the optimal exercise point, it is difficult for most numerical methods to get an accurate solution for the valuation of American options. Early optimal exercise for American options requires the following boundary conditions:

$$V(S,t) = \max\{V(S,T), V(S,t)\} \tag{3.6}$$

where the terminal payoff $V(S,T)$ is given by the initial condition (2.2) for both put and call options. The valuation of American put options can be easily modified by the Boundary update procedure for the European put options to:

Boundary update procedure for American put options:

Step 1: Compute $U_n = L\alpha_n$.

Step 2: For $i = 1, 2, \ldots, N$, update the i^{th} element: $U_n(i) = \max\{X - e^{y^i}, U_n(i)\}$.

Step 3: Update α_n simply by $L^{-1}U_n$.

To demonstrate the accuracy of this RBF method for solving American put options, we give the following example.

Example 3.2.1: Consider an American put option with $X = 100, r = 0.10, \sigma = 0.30, T = 1\,(year)$, as given by Wu and Kwok (1997). Let $y \epsilon [0, 6]$ so that $S \epsilon [1, e^6]$. In our computations, we take $N = 101, and\ M = 100$ and apply the RBF methods with various analytical, explicit, implicit, and parallel time integration schemes to compute the valuation of American put options, shown in Table 3.2.1. Also noting that the only difference in the valuation of European and American put options is the Boundary update procedure, and thus it makes the valuation of American put options relatively easy. Observed that Table 3.2.1 provides appropriate approximation for valuing American put options by using RBF methods with all analytical and numerical time integration schemes.

With the free boundary condition (3.6) for American put option, the eigenvalue representation method benefits less than explicit and implicit time integration schemes. From the results, implicit time integration is preferred due to its stable and accurate performance in the valuation of both European and American options.

Table 3.2.1: Comparison of Accuracy for American put options

$X = 100, r = 0.10, \sigma = 0.30, T = 1, S_{min} = 1, S_{max} = e^6, N = 101, and\ M = 100$

		RBF method				
Stock	**Binomial**	**Explicit**			**Implicit**	**Parallel**
S	**(n=1000)**	**BD1**	**RK2**	**RK4**	**($\theta = 0.5$)**	**Time**

80	20.2689	20.2894	20.2694	20.251	20.2535	20.2514
85	16.3467	16.3521	16.3343	16.3175	16.3180	16.3175
90	13.1228	13.1283	13.1112	13.0953	13.0961	13.0955
95	10.4847	10.4914	10.4756	10.4606	10.4611	10.4606
100	8.3348	8.3482	8.3342	8.3208	8.321	8.3208
105	6.6071	6.6152	6.6019	6.5891	6.5895	6.5892
110	5.2091	5.2205	5.2092	5.1984	5.1985	5.1983
115	4.0976	4.1054	4.0954	4.0859	4.0861	4.0859
120	3.2059	3.2174	3.209	3.201	3.2012	3.2010
RMSE		**0.0110**	**0.0068**	**0.0192**	**0.0186**	**0.0152**

*datachoice = 2; randomchoice=0; putorcall = 1; kernelchoice = 10; optionchoice = 20; intschchoice = 1 (BD1), 2 (RK2), 4 (RK4), 5 (Implicit), 6 (Parall Time); eigen_rep_choice=0;

Example 3.2.2:

As a result that the radial basis functions are infinitely differentiable, the derivatives of the option value are always available from the derivatives of the RBFs. Table 3.2.2 shows the comparison of various RBF methods Delta values $\frac{\partial V(S,0)}{\partial S}$ of an American option:-

Table 3.2.2: Comparison of Accuracy for Delta Values

$$X = 100, r = 0.10, \sigma = 0.30, T = 1, S_{min} = 1, S_{max} = e^6, N = 101, and\ M = 100$$

		RBF method				
Stock S	**Binomial (n=1000)**	**Explicit**			**Implicit** $(\boldsymbol{\theta = 0.5})$	**Parallel Time**
		BD1	**RK2**	**RK4**		
80	-0.8631	-0.8634	-0.8655	-0.8675	-0.8675	-0.8675
85	-0.7109	-0.7121	-0.7109	-0.7098	-0.7101	-0.7098
90	-0.5829	-0.5823	-0.5826	-0.5829	-0.5827	-0.5829

95	-0.4755	-0.4753	-0.4746	-0.4741	-0.4743	-0.4741
100	-0.3856	-0.3849	-0.3848	-0.3848	-0.3847	-0.3847
105	-0.3108	-0.3107	-0.3103	-0.3100	-0.3101	-0.3101
110	-0.2491	-0.2491	-0.2488	-0.2484	-0.2484	-0.2484
115	-0.1986	-0.1987	-0.1985	-0.1982	-0.1982	-0.1982
120	-0.1575	-0.1579	-0.1575	-0.1571	-0.1572	-0.1572
RMSE		**0.0005**	**0.0009**	**0.0017**	**0.0016**	**0.0017**

*datachoice = 2; randomchoice=0; putorcall = 1; kernelchoice = 10; optionchoice = 20; intschchoice = 1 (BD1), 2 (RK2), 4 (RK4), 5 (Implicit), 6 (Parallel Time); eigen_rep_choice=0;

It can be observed that, from Table 3.2.2, the delta values for this American option from the lowest order backward time integration scheme give the closet value to those obtained from the Binomial method.

Most existing numerical methods have encountered difficulty in valuing American options, and the decision of choosing the unknown optimal exercise boundary $B(t)$ is not trivial.

3.3 Example for Cash-or-Nothing Options

Cash-or-Nothing (CON) is a binary option, which has discontinuous payoffs. For example of the cash-or-nothing put, it pays a fixed amount H if the stock price *S,* (i.e. $S = e^y$), ends up below the strike price *X* (i.e. for Put); otherwise, it pays nothing. In this article, we choose H = 1.

To summarize, the initial (i.e. terminal in another sense) conditions of the CONs are given below:

$$V(S,T) = \begin{cases} 1 & if \ \ (X - e^y) > 0 \\ 0 & if \ \ (X - e^y) < 0 \end{cases} \qquad for\ a\ put$$

$$V(S,T) = \begin{cases} 1 & if \ \ (e^y - X) > 0 \\ 0 & if \ \ (e^y - X) < 0 \end{cases} \qquad for\ a\ call \tag{3.7}$$

Also, its boundary conditions are imposed as following:

$$\begin{cases} V(0,t) = 1e^{-r(T-t)}, & V(S,t) \to 0 \; as \; S \to \infty & for \; a \; put \\ V(0,t) = 0, & V(S,t) \to 1e^{-r(T-t)} \; as \; S \to \infty & for \; a \; call \end{cases} \tag{3.8}$$

The exact valuations of the cash-or-nothing options subject to the conditions (3.7) and (3.8) are given by:

$$\begin{cases} V(S,t) = 1e^{-r(T-t)}N(-d_2) & for \; a \; put \\ V(S,t) = 1e^{-r(T-t)}N(d_2) & for \; a \; call \end{cases} \tag{3.9}$$

where d_2 is given in equation (3.4).

Boundary update procedure for a cash-or-nothing put option:

Step 1: Compute $U_n = L\alpha_n$.

Step 2: Set the boundary conditions, i.e. $V(0,t) = 1e^{-r(T-t)}$ (i.e. for Put)

Step 3: Update α_n by $L^{-1}U_n$.

Example 3.3.1: Same as above for European and American options, we illustrate how the cash-or-nothings are applied by using RBF methods. Here we consider a cash-or-nothing. Applying the RBF methods with various analytical, explicit, implicit, and parallel time integration schemes, we compute the valuation of a cash-or-nothing put option as shown in Table 3.3.1.

Table 3.3.1: Comparison of Accuracy for Cash-or-Nothing put options

$$X = 15, r = 0.05, \sigma = 0.20, T = 0.25, S_{min} = 1, S_{max} = 30, N = 101, and \; M = 60$$

		RBF method		
		Explicit	**Implicit**	**Parallel**

Stock S	Exact	BD1	RK2	RK4	($\theta = 0.5$)	Time
5	0.9876	0.9876	0.9876	0.9876	0.9876	0.9876
6	0.9876	0.9876	0.9876	0.9876	0.9876	0.9876
7	0.9876	0.9876	0.9876	0.9876	0.9876	0.9876
8	0.9876	0.9876	0.9876	0.9876	0.9876	0.9876
9	0.9876	0.9876	0.9876	0.9876	0.9876	0.9876
10	0.9875	0.9876	0.9875	0.9875	0.9875	0.9875
11	0.9864	0.9864	0.9863	0.9862	0.9862	0.9862
12	0.9722	0.9714	0.9712	0.9710	0.9710	0.9710
13	0.9011	0.8950	0.8953	0.8956	0.8956	0.8956
14	0.7216	0.7078	0.7084	0.7090	0.7090	0.7090
15	0.4643	0.4480	0.4480	0.4480	0.4480	0.4480
16	0.2327	0.2204	0.2199	0.2194	0.2194	0.2194
17	0.0912	0.0842	0.0839	0.0836	0.0836	0.0836
18	0.0285	0.0253	0.0253	0.0254	0.0254	0.0254
19	0.0073	0.0060	0.0062	0.0063	0.0063	0.0063
20	0.0016	0.0012	0.0012	0.0013	0.0013	0.0013
RMSE		**0.00664**	**0.00663**	**0.00662**	**0.00662**	**0.00662**

*datachoice = 4; randomchoice=0; putorcall = 2; kernelchoice = 10; optionchoice = 10; intschchoice = 1 (BD1), 2 (RK2), 4 (RK4), 5 (Implicit), 6 (Parallel Time)

We can observe that, from Table 3.3.1, RBF method with all numerical time integration schemes provides appropriate approximation to the Cash-or-Nothing put option.

3.4 Example for Asset-or-Nothing Option

Asset-or-nothing (AON) is another binary option, and it has similar property as cash-or-nothing option. For example of the asset-or-nothing put, it pays the terminal valuation of its

stock price S (i.e. $S = e^y$) if the strike price X ends up below the stock price; otherwise, it pays nothing.

For asset-or-nothing options, its initial condition is given below:

$$V(S,T) = \begin{cases} S & if \ \ (X - e^y) > 0 \\ 0 & if \ \ (X - e^y) < 0 \end{cases} \qquad for\ a\ put$$

$$V(S,T) = \begin{cases} S & if \ \ (e^y - X) > 0 \\ 0 & if \ \ (e^y - X) < 0 \end{cases} \qquad for\ a\ call \tag{3.10}$$

And its boundary conditions are imposed as following:

$$\begin{cases} V(0,t) = Se^{-r(T-t)}, & V(S,t) \to 0\ as\ \ S \to \infty & for\ a\ put \\ V(0,t) = 0, & V(S,t) \to Se^{-r(T-t)}\ as\ \ S \to \infty & for\ a\ call \end{cases} \tag{3.11}$$

The exact valuations of the asset-or-nothing options subject to the conditions (3.10) and (3.11) are given by:

$$\begin{cases} V(S,t) = SN(-d_1) & for\ a\ put \\ V(S,t) = SN(d_1) & for\ a\ call \end{cases} \tag{3.12}$$

where d_1 is given in equation (3.3).

Example 3.4.1: To illustrate how Asset-or-nothing options can be applied by using RBF methods, we use the same data values as in Example 3.3.1 with $X = 15, r = 0.05, \sigma = 0.20, T = 0.25\ (year)$, and let $y\ \epsilon\ [0, \log(30)]$ so that $S\ \epsilon\ [1, 30]$. In this example, we also take $N = 101, and\ M = 100$, and apply the RBF methods with various analytical, explicit, implicit, and parallel time integration schemes to compute the valuation of a cash-or-nothing put option as shown in Table 3.4.1.

Table 3.4.1: Comparison of Accuracy for Asset-or-Nothing put options

$$X = 15, r = 0.05, \sigma = 0.20, T = 0.25, S_{min} = 1, S_{max} = 30, N = 101, and\ M = 60$$

		RBF method

Stock S	Exact	Explicit			Implicit ($\theta = 0.5$)	Parallel Time
		BD1	RK2	RK4		
5	5.0000	5.0000	5.0000	5.0000	5.0000	5.0000
6	6.0000	6.0000	6.0000	6.0000	6.0000	6.0000
7	7.0000	7.0000	7.0000	7.0000	7.0000	7.0000
8	8.0000	8.0000	8.0000	8.0000	8.0000	8.0000
9	9.0000	9.0000	9.0000	9.0000	9.0000	9.0000
10	9.9995	9.9997	9.9996	9.9994	9.9994	9.9994
11	10.9812	10.9824	10.9809	10.9794	10.9794	10.9794
12	11.7616	11.7491	11.7460	11.7429	11.7430	11.7429
13	11.6408	11.5490	11.5527	11.5563	11.5563	11.5563
14	9.7538	9.5437	9.5532	9.5627	9.5626	9.5627
15	6.4581	6.2107	6.2111	6.2114	6.2114	6.2114
16	3.2960	3.1094	3.1016	3.0939	3.0940	3.0939
17	1.3063	1.2015	1.1969	1.1924	1.1924	1.1924
18	0.4112	0.3632	0.3636	0.3641	0.3640	0.3641
19	0.1056	0.0871	0.0889	0.0906	0.0905	0.0906
20	0.0227	0.0168	0.0178	0.0189	0.0189	0.0189
RMSE		**0.1008**	**0.1005**	**0.1004**	**0.1004**	**0.1004**

*datachoice = 4; randomchoice=0; putorcall = 3; kernelchoice = 10; optionchoice = 10; intschchoice = 1 (BD1), 2 (RK2), 4 (RK4), 5 (Implicit), 6 (Parallel Time); eigen_rep_choice=0;

4 Conclusion for Basic Section

Until this section in the article, we have done the options pricing in both vanilla and exotic options by using the Radial Basis Functions method. Our numerical results indicate that the RBF method provide highly efficient and accurate valuations in both European and American vanilla options as well as cash-or-nothing and asset-or-nothing binary options. Unlike the

finite element method which approximates the solution by interpolating low-order piecewise continuous polynomials or the finite difference method which approximates the derivatives of the solution by finite quotients, the proposed RBF method provides the global interpolation formula for not only its solutions but also the derivatives. This technique provides a great improvement for some important indicators like Delta values. In addition, most currently used numerical methods encounter difficulties when valuing American options with free-boundary conditions. However, this no longer exists by using RBF techniques, and the valuation of American options is almost the same with European options except for the Boundary Update Procedure. With all our research and numerical results, we believe that this truly mesh-free RBF method would be an alternative to the commonly used finite element and finite difference methods.

5 Extension I – Using RBF for Multi-Asset Option

5.1 Introduction and Methodology

In our previous basic section, we apply the global radial basis function as a special approximation for the numerical solutions of the options value and its derivatives in the Black-Scholes equation for the single-asset option. However, RBF is in fact not only applicable to a 1 spatial dimensional case. In fact, RBF also behaves very well in high dimensional case, not only in the accuracy but also the efficiency. For the traditional mesh dependent method like the Finite Difference or the Finite Element method, regarding a n-dimensional space the calculation time increases on the basis of M^n. However, it is not the case for RBF, since the value of RBF only depends on radius, and the points can be even obtained from a random basis. A rough analogy is that the mesh dependent methods are trying to measure the n-dimensional room by using a set of 1-dimensional sticks, while RBF is just filling the room by using a set of n-dimensional spheres.

In this chapter, we will use the application of meshfree Radial Basis Function (RBF) method in the 2-asset case to approximate the numerical solutions of the Rainbow option values from the Black-Scholes differential equations.

5.1.1 Option on n Assets

The case of numerical method for the multi-asset is very similar to for a single-asset one.

Let's consider an option for which the payoff depends on n asset prices, $S_1, S_2, \dots, S_n$, with an assumption of a generalization of Geometric Brownian Motion for the underlying assets, where

$$dS_i = (\alpha_i - \delta_i)S_i\, dt + \sigma_i S_i\, dZ_i \quad i = 1, \dots, n \tag{5.1}$$

The pairwise correlation S_i and S_j is ρ_{ij} . Let $V(S_1, S_2, .., S_n, t, T)$ be the value of this claim. Consider a portfolio consisting of the claim, n assets, and bonds W, such that

$$I = V + \sum_{i=1}^{n} N_i S_i + W \tag{5.2}$$

By using the multivariate version of Ito's Lemma, the change in the value of the portfolio becomes:

$$dI = V_t dt + \sum_{i=1}^{n} V_{S_i} dS_i + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} dS_i dS_j\, V_{S_i} dt + \sum_{i=1}^{n} N_i\, dS_i + dW \tag{5.3}$$

Moreover, in order to get delta-hedge V , we set $N_i = -V_{S_i}$. Hold bonds to finance the residual such that $I = 0$. Then we derive the following Black–Scholes-type PDE of n-factor model for V [6]:

$$V_t + \sum_{i=1}^{n} (r - \delta_i) S_i V_{S_i} + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \rho_{ij}\sigma_i\sigma_j S_i S_j\, V_{S_i S_j} = rV \tag{5.4}$$

[6] P. 701, McDonald (2006), or P. 239 of Seydel (2009)

The initial (or terminal) conditions and the boundary conditions of this differential equation vary for different kinds of options. Suppose we use a simple transformation $y_i = log(S_i)$, for the convenience of our following work. Thus, we can change equation (5.4) to (5.5) :

$$\frac{\partial U}{\partial t} + \frac{1}{2}\sum_{i,j=1}^{n} \rho_{i,j}\sigma_i\sigma_j y_i y_j \frac{\partial^2 U}{\partial y_i \partial y_j} + \frac{1}{2}\sum_{i=1}^{n} \sigma_i^2 \frac{\partial^2 U}{\partial y_i^2} + \sum_{i=1}^{n}\left(r - \frac{1}{2}\sigma_i^2\right)\frac{\partial U}{\partial y_i} - rU = 0 \qquad (5.5)$$

, and our constraint conditions have to change correspondingly.

The idea of the proposed numerical scheme is to approximate the unknown function *U* using *N* radial basis functions ϕ , namely:

$$U(\boldsymbol{y}, t) := \sum_{j=1}^{N} \alpha_j(t)\, \phi\left(\left\|\boldsymbol{y} - y_j\right\|\right) \qquad (5.6)$$

$$where\ \boldsymbol{y} = \left(\boldsymbol{y^1}, \boldsymbol{y^2}, \dots, \boldsymbol{y}^n\right), \qquad y_j = \left({y_j}^1, {y_j}^2, \dots, {y_j}^n\right)$$

where $\alpha_j(t)$ are unknown coefficients depending on time and RBFs $\phi\left(\left\|\boldsymbol{y} - y_j\right\|\right)$, and $\left\|\boldsymbol{y} - y_j\right\|$ is a Euclidean radius taking into account of n assets. Note that the superscribe k of the y^k are with respect to the dimension but not power. It is also worth to note that N is respective to the number of points we chosen, while n is respective to the number of assets of the model.

We also have to determine the partial derivatives of equation (5.5) in order to get the solutions we need. Since the RBF does not depend on time, the time derivative of *U* is simply given in terms of the time derivatives of the coefficients:

$$\frac{\partial U(y_i, t)}{\partial t} = \sum_{j=1}^{N} \frac{d}{dt}\alpha_j(t)\, \phi\left(y_i, y_j\right) \qquad (5.7)$$

where $y_i = ({y_i}^1, {y_i}^2, \dots, {y_i}^n),\ y_j = \left({y_j}^1, {y_j}^2, \dots, {y_j}^n\right)$.

The first and second partial derivatives of *U* with respect to ***y*** are given, respectively, as below:

$$\frac{\partial U(y_i,t)}{\partial \boldsymbol{y^k}} = \sum_{j=1}^{N} \alpha_j(t) \frac{\partial \phi(y_i,y_j)}{\partial \boldsymbol{y^k}}, \qquad k = 1,\dots,n \tag{5.8}$$

$$\frac{\partial^2 U(y_i,t)}{\partial \boldsymbol{y^k} \partial \boldsymbol{y^p}} = \sum_{j=1}^{N} \alpha_j(t) \frac{\partial^2 \phi(y_i,y_j)}{\partial \boldsymbol{y^k} \partial \boldsymbol{y^p}}, \qquad k = 1,\dots,n;\ p = 1,\dots,n \tag{5.9}$$

To substitute above partial derivatives into the general equation (5.5) we finally get the solutions of the unknown function *U* and the value of options directly.

5.1.2 Rainbow Option on Two Assets

As an example, we consider an exotic European-style option, a rainbow call option, start from time 0 to the maturity T, with payoff equal to:

$$V(S_1,S_2,T) = max\{\, S_1(\mathrm{T}), S_2(\mathrm{T}), K \,\} \tag{5.10}$$

We adopt Hardy's MQ for our RBFs by choosing the shape parameter *c* to be $4d_{ave}$, where d_{ave} is the average distance from each collocation point y_j to its nearest neighbor, i.e. $c = 4\,log\left(\frac{S_{max}}{S_{min}}\right)/(n-1)$. Note that we choose the same S_{max} and S_{min} for both S_1 and S_2 for the sake of simplicity. For this 2-factor case, the transferred PDE (5.5) with initial condition (5.10) becomes:

$$\frac{\partial U}{\partial t} + \rho_{12}\sigma_1\sigma_2 \frac{\partial^2 U}{\partial y_1 \partial y_2} + \frac{1}{2}\sum_{i=1}^{2} \sigma_i^2 \frac{\partial^2 U}{\partial y_i^2} + \sum_{i=1}^{2} \left(r - \frac{1}{2}\sigma_i^2\right) \frac{\partial U}{\partial y_i} - rU = 0 \tag{5.11}$$

By choosing Hardy's MQ as our kernel, we can deduce the partial derivatives of the transferred PDE straightforward as the following:

Relevant sub-function: ***Two_Dim_Kernel.m***

$$\phi_j\left(\boldsymbol{y} = (\boldsymbol{y^1}, \boldsymbol{y^2})\right) = \sqrt{\left(\boldsymbol{y^1} - y_j^1\right)^2 + \left(\boldsymbol{y^2} - y_j^2\right)^2 + c^2}$$

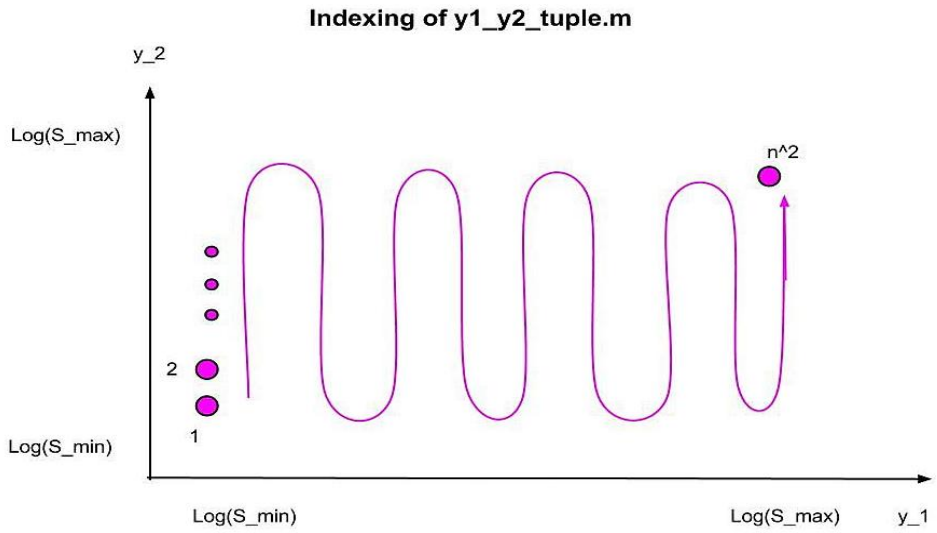
$$\frac{\partial \phi(\boldsymbol{y})}{\partial \boldsymbol{y^i}} = \frac{\boldsymbol{y^i} - y_j^i}{\phi_j(\boldsymbol{y})} \qquad i = 1, 2$$

$$\frac{\partial^2 \phi(\boldsymbol{y})}{\partial (\boldsymbol{y^i})^2} = \frac{1}{\phi_j(\boldsymbol{y})} - \frac{\left(\boldsymbol{y^i} - y_j^i\right)^2}{\phi_j(\boldsymbol{y})^3} \qquad i = 1, 2$$

$$\frac{\partial^2 \phi(\boldsymbol{y})}{\partial \boldsymbol{y^1} \partial \boldsymbol{y^2}} = -\frac{\left(\boldsymbol{y^1} - y_j^1\right)\left(\boldsymbol{y^2} - y_j^2\right)}{\phi_j(\boldsymbol{y})^3} \tag{5.12}$$

However, since j is also 2 dimensional, the overall (5.12) would become a 4-dimensional tensor. If we want to write (5.12) in matrix form, we need to use some special indexing technique to reduce the 2-dimensional plane into a 1 dimensional curve, and we do it as below:-

Relevant sub-function: ***y1_y2_tuple.m***



Therefore the dimension of j is reduced from 2 to 1, and we can now denote the matrix with entries $\phi_j\left(\boldsymbol{y_i} = (\boldsymbol{y_i^1}, \boldsymbol{y_i^2})\right)$ as L. Thus we can now write (5.11) into a matrix form as below:-

$$\dot{\alpha} = P\alpha \tag{5.13}$$

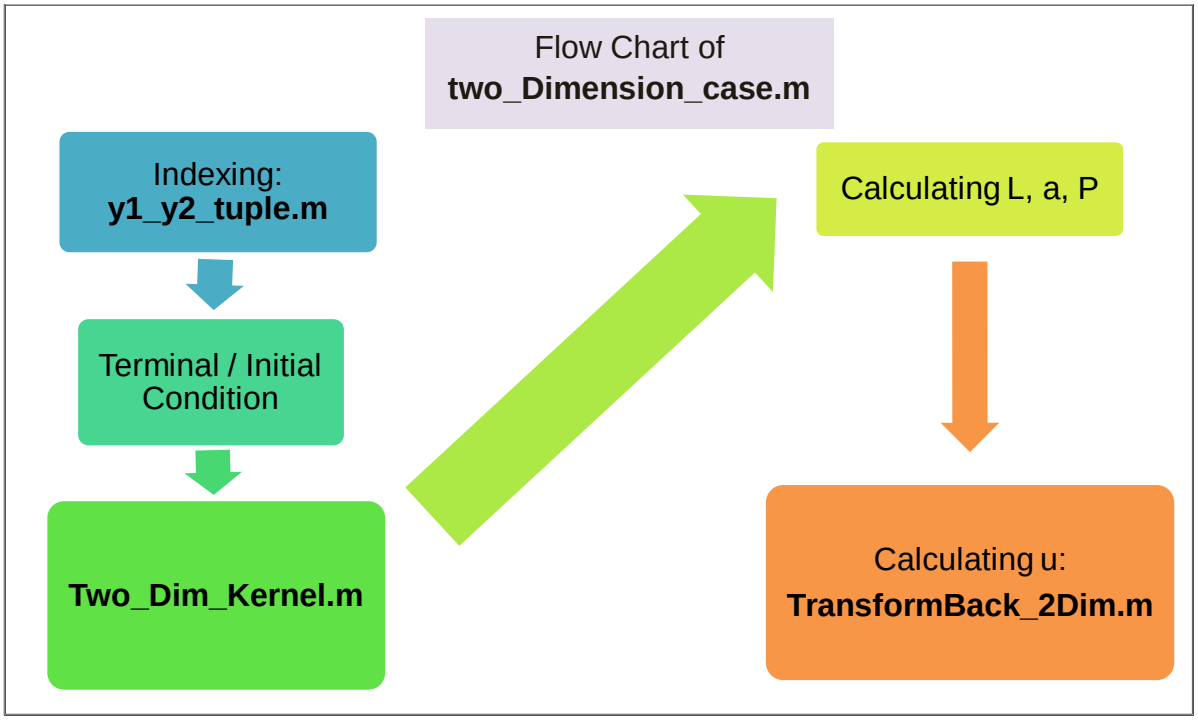
where

$$P = L^{-1}\left[-\rho\sigma_1\sigma_2 L_{12} - \frac{1}{2}\sigma_1^2 L_{11} - \left(r - \frac{1}{2}\sigma_1^2\right)L_1 - \sigma_2^2 L_{22} - \left(r - \frac{1}{2}\sigma_2^2\right)L_2 + rL\right] \tag{5.14}$$

Similar to the 1-dimensional case, the Lij are respective to the derivatives $\partial_i\partial_j L$ of the kernel. By using (5.14), we can use the techniques of 1-dimensional BD1 given by (2.25) to solve problems of the 2-dimensional cases.

5.2 Numerical Computation

The main program for chapter 5 is **Two_Dimensional_Case.m**, with 3 sub-functions, namely **y1_y2_tuple.m**, **Two_Dim_Kernel.m** and **TransformBack_2Dim.m**. The flow chart of the program is as below:-

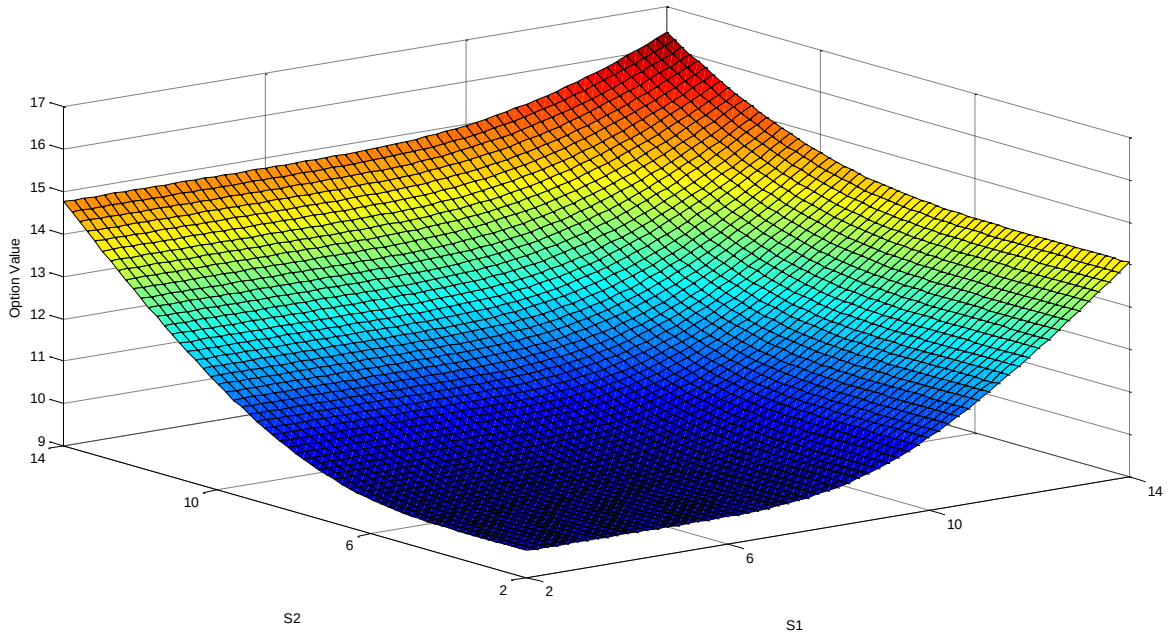


Consider two assets S_1 and S_2, both ranging from [0, 30] with volatilities 0.25 and 0.3 respectively, the correlation coefficient ρ equals to 0.3. Assume S_1 and S_2 have the same values of N and M, i.e. N=21 and M=30. The risk-free interest rate is 5%, T is 9 months, and the strike prices for both S_1 and S_2 are 10. We are now going to determine the value of European rainbow call option on S_1 and S_2.

Assume we have the same index selection, 2, 6, 10, and 14 for both $S_1(0)$ and $S_2(0)$, by

using RBF method to solve this problem, we will finally obtain a (4×4) sized matrix for the value of rainbow option. If we choose a more frequent index (e.g. 2:0.2:14) and obtain a large matrix (i.e. (61×61)) for the final output, we would obtain an even smoother solution and hereby use **Figure 5.1** to explain the situation:-

Figure 5.1



As mentioned above, we choose Hardy's MQ as our kernel and use Backward Difference (BD1) time integration for our approximation. Finally, we get the solutions given in **Table 5.1**.

Table 5.1 Using RBF for 2-D European Rainbow Call Option

Payoff = max($S_1(T)$, $S_2(T)$, X)

X = 10.0; r = 0.05; σ_1 = 0.25; σ_2=0.3; T = 0.75; N=21; M=30; ρ=0.3; S_{min}=1.0; S_{max}=30;

Hardy's MQ; c=4*dy; Backward Difference time integration

*By using the program **Two_Dimensional_Case.m***				
S1\S2	**2**	**6**	**10**	**14**
2	9.63265	9.64169	10.704	14.0511

6	9.79216	9.79898	10.7841	14.062
10	11.488	11.4901	12.0555	14.5718
14	14.7639	14.7613	14.9742	16.3782
Exact Solution				
S1\S2	**2**	**6**	**10**	**14**
2	9.631944177	9.658455109	10.84101535	14.10075877
6	9.640269812	9.666091021	10.84319919	14.10096564
10	10.67476923	10.68369496	11.43892956	14.24816302
14	14.04281618	14.04351962	14.22123638	15.58823779

In order to verify our answer, we had also given out the exact values in **Table 5.1**, while the exact pricing formula[7] of European rainbow call option is given by the followings:-.

$$RainCall(S_1, S_2, K, \sigma_1, \sigma_2, \rho, T)$$

$$= S_1\left\{N(d_{12}) - N_2\left[-d_1(S_1), d_{12}, \frac{\rho\sigma_2 - \sigma_1}{\hat{\sigma}}\right]\right\} + S_2\left\{N(d_{21}) - N_2\left[-d_1(S_2), d_{21}, \frac{\rho\sigma_1 - \sigma_2}{\hat{\sigma}}\right]\right\}$$

$$+ Ke^{-rT}N_2[-d_2(S_1), -d_2(S_2), \rho] \qquad (5.13)$$

where $d_1(S_1) = \frac{\log\left(S_1/K\right) + (r + 0.5{\sigma_1}^2)T}{\sigma_1\sqrt{T}}$, $d_1(S_2) = \frac{\log\left(S_2/K\right) + (r + 0.5{\sigma_2}^2)T}{\sigma_1\sqrt{T}}$

$$d_2(S_1) = d_1(S_1) - \sigma_1\sqrt{T}\,, \qquad d_2(S_2) = d_1(S_2) - \sigma_2\sqrt{T}$$

$$d_{12} = \frac{\log\left(S_1/S_2\right) + 0.5\hat{\sigma}^2T}{\hat{\sigma}\sqrt{T}}, \quad d_{21} = \frac{\log\left(S_2/S_1\right) + 0.5\hat{\sigma}^2T}{\hat{\sigma}\sqrt{T}}, \quad \hat{\sigma} = \sqrt{\sigma_1^2 + \sigma_2^2 - 2\rho\sigma_2},$$

$$\mathrm{N}_2(\mathrm{a}, \mathrm{b}; \rho) = \mathrm{Prob}(\mathrm{z}_1 < a, \mathrm{z}_2 < b; \rho), where\ \mathrm{z}_1 \text{ and } \mathrm{z}_2 \text{ are normal random variable}$$

As can be seen from **Table 5.1**, the results obtained by RDF method are very close to the exact value as a whole.

[7] See Robert L. McDonald, *"Derivatives Markets-2ed "*, P. 734-735

5.3 Conclusion

In conclusion, the example presented in this chapter suggests that meshfree approximation methods should be considered as a possible way to solve multi-asset option pricing problems.

The advantage of this extension seems to be obvious. It is possible to obtain the value of the option for any combination of stock prices simply by evaluating the expansion (5.6). For other mesh depending method such as finite difference method, we may need to include an extra interpolation step, which in the multi-asset case is a challenge in itself.

In addition, RBF method is able to produce highly accurate approximations to spatial derivatives, which is more efficient and cost-effective in practice.

6 Extension II – Including Time as Radius of the Radial Kernel

6.1 Introduction and Methodology

In our previous section, only the Spatial Measure y is included in the Radius calculation of the "Pure Spacial Kernels", while the Alphas are treated as time dependent functions, i.e. α(t). In this section, we will try to use another approach to look into the problem. We will try to incorporate time as the radius of the kernel, forming a family of kernels that we called as "Time-Space Mixture Kernels", and see if it would give some insight for us in the RBF analysis.

Come back to our transformed Black-Scholes Equation after the application of $S=e^y$:-

$$\frac{\partial U}{\partial t} + \frac{1}{2}\sigma^2\frac{\partial^2 U}{\partial y^2} + \left(r - \frac{1}{2}\sigma^2\right)\frac{\partial U}{\partial y} - rU = 0 \qquad (6.1)$$

, which is the start of our investigation in this section. In previous sections, we use T as the terminal time of the option. We now introduce another Greek letter τ to represent the remaining time of the option, which is defined as $\tau = T - t$. We prefer to use τ instead of t in the model, with the reason to be explained later. Therefore the Black-Scholes equation (6.1) becomes:-

$$-\frac{\partial U}{\partial \tau} + \frac{1}{2}\sigma^2 \frac{\partial^2 U}{\partial y^2} + \left(r - \frac{1}{2}\sigma^2\right)\frac{\partial U}{\partial y} - rU = 0 \tag{6.2}$$

For the sake of simplicity, we will only focus on the European put option, whose payoff is as below:-

$$U(y, t = T(or\ \tau = 0)) = max\{X - e^y, 0\} \quad for\ European\ put \tag{6.3}$$

Of course, we can change our investigation easily from put option to call option, by just changing the boundary conditions and initial conditions with slight modification in the Matlab program. However, since we would like to focus on the properties of "Time-Space Mixture Kernel" in this section and we had already investigated the application of RBF on call options (i.e. and American options as well) in Chapter 2 and 3, we are not going to make further discussion on neither Call nor American options here.

Now it comes to the choice of points. In contrast to treating each spacial y_j as a point in our previous sections, we now treat each space-time tuple (y_j, τ_j) as our point. Therefore we interpolate the unknown function U by the following radial basis functions ϕ:-

$$U(y, \tau) \approx \sum_{j=1}^{N} \alpha_j \, \phi_{y_j, \tau_j}(y, \tau) \tag{6.4}$$

There are 2 points to note. First, as we now put the time into the kernel, Alpha is no longer time dependent. Instead, $\alpha_j(t)$ is now replaced by the time independent constant α_j. Second, we hereby explain the reason why we use τ in the kernel instead of using t. Since t is just an index mainly determined by the reference starting time that we chosen, τ is the remaining

time to the termination which is more meaningful in some sense of the "Radius of Time". As such, it is believed that τ will give a better behaviour than t for our study.

Since it is also believed that time and the stock price would have different behavior/ scale in fact, we suggest not to directly use the Euclidean distance for the calculation of radius in the kernel. Instead, we will add another parameter λ into the kernel, so as to balance the scale difference between the time and space dimensions. Therefore the kernel we used in this section, i.e. the Hardy's MQ , would become the followings:-

$$\phi_{y_j,\tau_j}(y,\tau) = \sqrt{(y - y_j)^2 + \lambda(\tau - \tau_j)^2 + c^2} \qquad for\ Hardy's\ MQ \tag{6.5}$$

where c is the shape parameters. In this paper, we will keep using c to be $4d_{min}$, same as the "Pure Spacial Kernel" case. As explained above, λ is with respect to the scale difference between time and space, and we now take

$$\lambda = Scale \times \left(\frac{\log(S_{\max})}{T}\right)^2 \tag{6.6}$$

with choosing Scale to be 150 here.

Now, the Black-Scholes equation (6.2) becomes:-

$$-\frac{\partial U(y_i,\tau_i)}{\partial \tau} + \frac{1}{2}\sigma^2\frac{\partial^2 U(y_i,\tau_i)}{\partial y^2} + \left(r - \frac{1}{2}\sigma^2\right)\frac{\partial U(y_i,\tau_i)}{\partial y} - rU(y_i,\tau_i) = 0 \tag{6.7}$$

where

$$\frac{\partial U(y_i,\tau_i)}{\partial \tau} = \sum_{j=1}^{N} \alpha_j \frac{\partial \phi_{y_j,\tau_j}(y_i,\tau_i)}{\partial \tau} \tag{6.8}$$

$$\frac{\partial U(y_i,\tau_i)}{\partial y} = \sum_{j=1}^{N} \alpha_j \frac{\partial \phi_{y_j,\tau_j}(y_i,\tau_i)}{\partial y} \tag{6.9}$$

$$\frac{\partial^2 U(y_i,\tau_i)}{\partial y^2} = \sum_{j=1}^{N} \alpha_j \frac{\partial^2 \phi_{y_j,\tau_j}(y_i,\tau_i)}{\partial y^2} \tag{6.10}$$

for Hardy's MQ:

Relevant sub-function: ***Space_Time_Kernel.m***

$$\frac{\partial \phi_{y_j,\tau_j}(y_i,\tau_i)}{\partial y} = \frac{(y_i - y_j)}{\sqrt{\lambda(\tau - \tau_j)^2 + (y_i - y_j)^2 + c^2}} \tag{6.11}$$

$$\frac{\partial^2 \phi_{y_j,\tau_j}(y_i,\tau_i)}{\partial y^2} = \frac{1}{\sqrt{\lambda(\tau - \tau_j)^2 + (y_i - y_j)^2 + c^2}} - \frac{(y_i - y_j)^2}{(\lambda(\tau - \tau_j)^2 + (y_i - y_j)^2 + c^2)^{\frac{3}{2}}} \tag{6.12}$$

$$\frac{\partial \phi_{y_j,\tau_j}(y_i,\tau_i)}{\partial \tau} = \frac{\lambda(\tau - \tau_j)}{\sqrt{\lambda(\tau - \tau_j)^2 + (y_i - y_j)^2 + c^2}} \tag{6.13}$$

In matrix form, the Black Scholes Equation (6.7) becomes

$$\left[-L_\tau + \frac{1}{2}\sigma^2 L_{yy} + \left(r - \frac{1}{2}\sigma^2\right) L_y - rL\right] \alpha = 0 \tag{6.14}$$

where α denotes the vector of the unknown coefficients α_j, and L, L_y, L_{yy}, L_τ are the matrices with entries $\phi_{y_j,\tau_j}(y_i,\tau_i)$, $\frac{\partial \phi_{y_j,\tau_j}(y_i,\tau_i)}{\partial y}$, $\frac{\partial^2 \phi_{y_j,\tau_j}(y_i,\tau_i)}{\partial y^2}$, $\frac{\partial \phi_{y_j,\tau_j}(y_i,\tau_i)}{\partial \tau}$ given above, respectively. The matrix L is invertible due to the collocation points are distinct, and thus equation (6.14) can be rewritten as the equation below:-

$$P\alpha = 0 \tag{6.15}$$

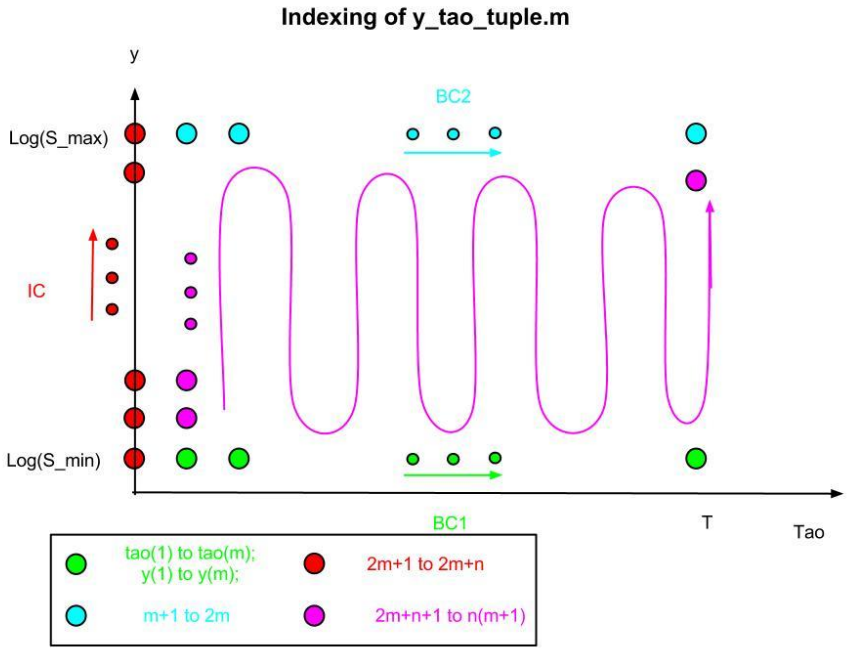
where P is the operator:-

$$P = -L_\tau + \frac{1}{2}\sigma^2 L_{yy} + \left(r - \frac{1}{2}\sigma^2\right) L_y - rL \tag{6.16}$$

Choice of Points and Indexing

Relevant sub-function: ***y_tao_tuple.m***

After the incorporation of Time as the radius, the dimension of the kernel increased from 1 (i.e. Space only) to 2 (i.e. both Space and Time). As in the 1 dimensional case choosing y uniformly among the domain, we now choose the points (y, τ) uniformly among the domain. However, we are not going to index the points in a 2 dimensional way (y_j, τ_k). Instead, we would still index the points in a 1 dimensional way (y_j, τ_j) with pictorially explanation as below, which would be more convenience for us in the future use:-



As it can be seen from the above figure, the n(m+1) points of { (y_1, τ_1),…, $(y_{n(m+1)}, \tau_{n(m+1)})$ } is indexed from the following order:-

i. (y_1, τ_1) up to (y_m, τ_m): The bottom line, which is referring to the lower boundary condition BC1 (i.e. $V(0, \tau) = Xe^{-r\tau}$, for a put).

ii. (y_{m+1}, τ_{m+1}) up to (y_{2m}, τ_{2m}): Upper boundary condition BC2 (i.e. V(S,τ)→0, as S→∞ for a put).

iii. (y_{2m+1}, τ_{2m+1}) up to (y_{2m+n}, τ_{2m+n}): Left line, which refers to the Initial Condition IC (i.e. Payoff of the option).

iv. $(y_{2m+n+1}, \tau_{2m+n+1})$ up to $(y_{n(m+1)}, \tau_{n(m+1)})$: At last, the points of the middle will be indexed in a snake-shaped manner.

After the Indexing, the points will then placed in the matrix form as below:-

$$\begin{bmatrix} \phi_1(y_1,\tau_1) & \cdots & \phi_{n(m+1)}(y_1,\tau_1) \\ \vdots & \ddots & \vdots \\ \phi_1(y_{2m+n},\tau_{2m+n}) & \cdots & \phi_{n(m+1)}(y_{2m+n},\tau_{2m+n}) \\ P_1(y_{2m+n+1},\tau_{2m+n+1}) & \cdots & P_{n(m+1)}(y_{2m+n+1},\tau_{2m+n+1}) \\ \vdots & \ddots & \vdots \\ P_1(y_{n(m+1)},\tau_{n(m+1)}) & \cdots & P_{n(m+1)}(y_{n(m+1)},\tau_{n(m+1)}) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_{2m+n} \\ \alpha_{2m+n+1} \\ \vdots \\ \alpha_{n(m+1)} \end{bmatrix} = \begin{bmatrix} BC1 \\ BC2 \\ IC \\ \vec{0} \end{bmatrix} \quad (6.17)$$

In short, it becomes

$$\begin{bmatrix} L_{boundary} \\ P \end{bmatrix} \alpha = \begin{bmatrix} BC1 \\ BC2 \\ IC \\ \vec{0} \end{bmatrix} \quad (6.18)$$

Where $L_{boundary}$ is a (2m+n) × n(m+1) matrix which is with respect to the boundary points only, and P is a [n(m+1)-(2m+n)] × n(m+1) matrix which is with respect to the middle points only. Please note that the lower part of (6.18) is in fact respective to (6.15). The combination of these 2 matrix achieves a n(m+1) × n(m+1) matrix. As we can see, it is the reason why we prefer a one dimensional index of (y_i, τ_i), instead of a two dimensional index of (y_i, τ_k), since we prefer the 2-dimensional matrix notation instead of the 4-dimensional tensor notation for the sake of simplicity.

By using the above matrix equation, it is easy to solve for α, which is a n(m+1) × 1 vector and the coefficients of the kernels. Since the α is no longer time dependent, there is no difference in the choices of BD1, RK2 etc. We can easily obtain α by just taking the inverse of the square matrix $\begin{bmatrix} L_{boundary} \\ P \end{bmatrix}$ in (6.18).

Finally, we can obtain the value of the option price by using α and ϕ.

6.2 Numerical Computation

The main program for this section is **Time_as_radius.m**, with 2 sub-functions of **y_tao_tuple.m** and **Space_Time_Kernel.m**.

To indicate the numerical accuracy of the "Time-Space Mixture Kernels" method as a approximation for the option values, we use it to compute the European Put Option values with different pair of (n, m) tuples.

By trying different pair of (n, m) tuples, we note that small value of n with big value of m (relatively comparing to the use of (n, m) in the Pure Kernel Section) would give a relatively better result:

Table 6.1: Comparison of Accuracy for the Mixture Kernels

For European Put Option,

$X = 10, r = 0.05, \sigma = 0.20, \tau = 0.5, S_{min} = 1, S_{max} = 30, scale = 150$, c=4dy

Stock S	**Exact**	**n, m of Mixture Kernels**						**Pure Kernels***	
		21, 100	21, 90	21, 160	31, 100	31, 90	31, 80	RK2	RK4
2	7.7531	7.7402	7.8397	7.8219	7.9034	7.9195	7.9439	7.7530	7.7530
4	5.5731	5.7539	5.7976	5.7804	5.8226	5.8321	5.841	5.7530	5.7531
6	3.3732	3.7748	3.8074	3.7777	3.8261	3.8416	3.8612	3.7532	3.7532
8	1.7987	1.8239	1.8405	1.8289	1.8334	1.8315	1.8274	1.8094	1.8095
10	0.442	0.448	0.4521	0.4745	0.4257	0.4286	0.4365	0.4786	0.4784
12	0.0483	0.0482	0.048	0.0612	0.0407	0.0345	0.0281	0.0562	0.0561
14	0.0028	0.0306	0.0268	0.017	0.0234	0.0209	0.0141	0.0033	0.0033
16	0.0001	0.0315	0.0291	0.0121	0.0241	0.0286	0.0328	0.0001	0.0001
RMSE	**0.1568**	**0.1767**	**0.1635**	**0.1912**	**0.1990**	**0.2088**	**0.1493**	**0.1493**	**0.1568**

*(n, m)=(21,100)

As we can see from the above, when (n ,m) = (21, 100), the approximated value would give errors smaller than 10%. However, when Stock S = 16 which is far out of the money, almost all of the pair does not give a good approximation value.

Comparing to the Pure Spacial Kernels, the Pure Kernels is relatively less accurate on the at-the-money option, but the Mixture Kernels work well on the at-the-moneys. In the contrary, Pure Kernels work better on both in-the-money and out-of-the-money option, while the Mixture kernels works relatively worse on the out-of-the-money option.

6.3 Conclusion

Since the Mixture Kernels also include “time” as the radius of the kernel, we conjecture that it is the reason of a better approximation on the “at-the-money option”, as the incorporation of time is “diluting” the variance of error made from the Spacial Dimension. However, for the far-out-of-the-money option, the mixture kernels do not work very well.

7 References

[1]. Aronszajn N. (1950), “Theory of Reproducing Kernels”, *Trans. Amer. Math. Sox.* 68, pp. 337-404

[2]. Farlow, S. J. (2006), “An introduction to Differential Equations and Their Applications”, *Dover Publications*

[3]. Fasshauer, G. E. & Khaliq, A. O. M. & Voss, D. A. (2003) “Using Meshfree Approximations for Multi-asset American Option Problems”, *Submitted*

[4]. Fasshauer G. E. et al (2004), “A Parallel Time Stepping Approach Using Meshfree Approximations for Pricing Options with Non-Smooth Payoffs”,

[5]. Fasshauer G. E. (2007), “Meshfree Approximation Methods with Matlab”, *World Scientific Publishing*

[6]. Fasshauer G. E. et al (2011), “Positive Denite Kernels, Past, Present and Future”

[7]. Franke, R. (1982), “Scattered Data Interpolation”, *Test of Some Methods*

[8]. Hardy R.L. (1971), “Multiquadric Equations of Topography and Other Irregular Surfaces”,

[9]. Hon Y. C. & Mao, X. Z. *(*1999), “A Radial Basis Function Method for Solving Options Pricing Models”, *Financial Engineering 8*

[10]. McDonald, R. L. (2006), “Derivatives Ma*rkets-2ed”, Addison Wesley*

[11]. Seydel R. U. (2009), “Tools for Computational Finance-4ed”, *Springer-Verlag Berlin Heidelberg*

[12]. Wilmott P. (1996), “The Mathematics of Financial Derivatives”, *Cambridge University Press*

[13]. Wu, L. and Kwok Y. K. (1997), “A Front-Fixing Finite Difference Method for the Valuation of American Options”, *Journal of Financial Engineering - 6, 1997, P. 83*

8 Appendix

8.1 Matlab Codes for Basic Section

a) Main code: *RBF_OptionPricing.m*

```
% It is the main program for basic section

clear
datachoice=1;
randomchoice=0;
putorcall=1;
kernelchoice=10;
optionchoice=10;
intschchoice=1;
eigen_rep_choice=0;
tic
[X r sigma T n m Smin Smax index]=choice_of_standard_data(datachoice);

indexlength=size(index,2);
dt=T/m;
k=1:n;
dy=log(Smax/Smin)/(n-1);
small_ds=dy/100; % for calculating delta

if randomchoice==0
    y=log(Smin)+(k-1)*dy;
elseif randomchoice==1
    y=log(Smin)+random_number_generator(Smax,n);
    % we still define dy as constant, for the use of Hardy's MQ
end

IC=initial_condition(X,y,putorcall);
```

```
[L DL D2L,c] = choice_of_kernel(kernelchoice,n,y,dy);

inv_L=inv(L);
a=inv_L*IC;
P=r*eye(n,n)-0.5*sigma^2*inv_L*D2L-(r-0.5*sigma^2)*inv_L*DL;

for i = 1:m %time
    a=type_of_option(L,inv_L,a,X,i,r,dt,y,n,optionchoice,putorcall);
    a=choice_of_int_sch(a,P,dt,n,intschchoice);
end

% Backward transform to obtain the European option prices (Compare with
Table 2 on Page 11)
[u uplus] = TransformBack(index,y,c,a,small_ds,indexlength,kernelchoice);

delta=(uplus-u)/small_ds;
[['Stock  '; 'Option '; 'Delta  '] num2str([index; u;delta])]
toc

if eigen_rep_choice==1;
    alpha_by_eigen =
Eigenvalue_Representation(P,L,inv_L,IC,T,index,y,c,small_ds,indexlength,ker
nelchoice);
elseif eigen_rep_choice==0;
end
```

b) Function: *choice_of_standard_data.m*

```
function [X r sigma T n m Smin Smax
index]=choice_of_standard_data(datachoice);

% don't change the value of choice 1, choice 2, choice 3 and choice 4
unless necessary.
% for free variables, use choice 5 only.

if datachoice==1 % for example of European put; n can be 21, 61, 101,
141
    X = 10.0; r = 0.05; sigma = 0.2; T = 0.5; n=81; m=30;
    Smin=1.0; Smax=30;
    index=2:2:16;

elseif datachoice==2 % for example of American put;

    X = 100.0; r = 0.1; sigma = 0.3; T = 1; n=101; m=100;
    Smin=1.0; Smax=exp(6);
    index=80:5:120;

elseif datachoice==3 % for example of random;

    X = 10.0; r = 0.05; sigma = 0.2; T = 0.5; n=61; m=5;
    Smin=1.0; Smax=30;
    index=2:2:16;

elseif datachoice==4 % for example of Binary Options;
```

```
    X = 15.0; r = 0.05; sigma = 0.2; T = 0.25; n=101; m=60;
    Smin=1.0; Smax=30;
    index=5:1:20;

elseif datachoice==5 % for free choices;

    X = 10.0; r = 0.05; sigma = 0.2; T = 0.5; n=61; m=30;
    Smin=1.0; Smax=30;
    index=2:2:16;

end

end
```

c) Function: *random_number_generator.m*

```
function  y=random_number_generator(Smax,n)

loop=1;
minimum_distance=0.025;
y=[0 log(Smax)*sort(rand(1,n-1))];

while loop==1
    flag=0;
    loop=0;
    for i=2:n
        if y(i)-y(i-1)<minimum_distance
            % to ensure that the generated points are distinct, so that
no singularity.
            loop=1;
            flag=[flag i];
        end
    end

    [x size_of_flag_plus_1]=size(flag);
    flag=flag(2:size_of_flag_plus_1);

    y(flag)=log(Smax)*rand(1,size_of_flag_plus_1-1);
    y=sort(y);

end

end
```

d) Function: *initial_condition.m*

```
function IC=initial_condition(X,y,initialcondchoice)

if initialcondchoice == 1     % American (or European) put
    IC = max(X-exp(y'), 0);
```

```
elseif initialcondchoice == 10      % American (or European) call
    IC = max(exp(y')-X, 0);

    % Cash-or-Nothing
elseif initialcondchoice == 2      % cash-or-nothing put
    IC = 1*(X-exp(y')>0);

elseif initialcondchoice == 20     % cash-or-nothing call
    IC = 1*(exp(y')-X>0);

    % Asset-or-Nothing
elseif initialcondchoice == 3      % asset-or-nothing put
    IC=exp(y').*(X-exp(y')>0);

elseif initialcondchoice == 30      % asset-or-nothing call
    IC=exp(y').*(exp(y')-X>0);

end
```

e) Function: *choice_of_kernel.m*

```
function [L DL D2L, c] = choice_of_kernel(kernelchoice,n,y,dy)
deltaY=meshgrid(y,y)'-meshgrid(y,y);

if kernelchoice == 10 % suppose kernelchoice = 10 refers to Hardy's MQ;
    c=4*dy;
    L=sqrt(deltaY.^2+c^2);
    DL=deltaY./L;
    D2L=1./L-deltaY.^2./L.^3;

elseif kernelchoice == 20 % suppose kernelchoice = 20 refers to
Gaussian spline;
    c = exp(6)*dy;
    L=exp(-(c*deltaY).^2);
    DL=-2*(c^2)*deltaY.*L;
    D2L=-2*(c^2)*deltaY.*DL-2*(c^2)*L;

elseif kernelchoice == 30 % which refers to thin plate spline
    c=2; % Power of the kernel
    L=deltaY.^(2*c).*log(abs(deltaY));
    DL=deltaY.^(2*c-1)+2*c*L./deltaY;
    D2L=(4*c-1)*deltaY.^(2*c-2)+(4*c^2-2*c)*L./deltaY.^2;

    % to place the ¡§NaN¡¨ diagonal by the limit of the function
    L(1:length(L)+1:numel(L)) = 0;
    DL(1:length(DL)+1:numel(DL)) = 0;
    D2L(1:length(D2L)+1:numel(D2L)) = 0;

end
end
```

f) Function: *type_of_option.m*

```
function
a=type_of_option(L,inv_L,a,X,i,r,dt,y,n,optionchoice,putorcall)

U=L*a;

if optionchoice == 10 & putorcall==1 % European put option;
    U(1)=X*exp(-r*i*dt);

elseif optionchoice == 10 & putorcall==10 % European call options
    U(1)=0;
    U(n)=exp(y(n));      % boundary condition for call

elseif optionchoice == 10 & putorcall==2 % European Cash or Nothing put
option;
    U(1)=1*exp(-r*i*dt);

elseif optionchoice == 10 & putorcall==20 % European Cash or Nothing
call options
    U(1)=0;
    U(n)=1*exp(-r*i*dt);

elseif optionchoice == 10 & putorcall==3 % European Asset or Nothing
put option;
    U(1)=0;

elseif optionchoice == 10 & putorcall==30 % European Asset or Nothing
call options
    U(n)=exp(y(n));

elseif optionchoice==20 & putorcall==1 % American put option;
    for j=1:n
        U(j)=max(X-exp(y(j)),U(j));
    end

elseif optionchoice==20 & putorcall==10 % American call option;
    for j=1:n
        U(j)=max(exp(y(j))-X,U(j));    % but it should in fact same as
the European case since we don't consider dividend
    end

elseif optionchoice==20 & putorcall==2 % American Cash or Nothing put
option;
    for j=1:n
        U(j)=max(X-exp(y(j))>0,U(j));
    end

elseif optionchoice==20 & putorcall==20 % American Cash or Nothing call
option;
    for j=1:n
        U(j)=max(exp(y(j))-X>0,U(j));
    end

elseif optionchoice==20 & putorcall==3 % American Asset or Nothing put
option;
    for j=1:n
        U(j)=max(exp(y(j))*(X-exp(y(j))>0),U(j));
    end
```

```
elseif optionchoice==20 & putorcall==30 % American Asset or Nothing
call option;
    for j=1:n
        U(j)=max(exp(y(j))*(exp(y(j))-X>0),U(j));
    end

end

a=inv_L*U;
end
```

g) Function: *choice_of_int_sch.m*

```
function a=choice_of_int_sch(a,P,dt,n,intschchoice)
if intschchoice == 1                %choice of BD1
    a = a-dt*P*a;
elseif intschchoice == 2                %choice of RK2
    F1 = -dt*P*a;
    F2 = -dt*P*(a+.5*F1);
    a = a+.5 *(F1+F2);
elseif intschchoice == 4                %choice of RK4
    F1 = -dt*P*a;
    F2 = -dt*P*(a+.5*F1);
    F3 = -dt*P*(a+.5*F2);
    F4 = -dt*P*(a+F3);
    a = a+1/6*(F1+2*F2+2*F3+F4);

elseif intschchoice == 5                %choice of Implicit backward
time integration
    theta = 0.5;
    matrix1=eye(n)+dt*(1-theta)*P;
    matrix2=eye(n)-dt*theta*P;
    a=matrix1\matrix2*a;

elseif intschchoice == 6                %choice of Parallel Time Step

    V1=inv((eye(n)+1*dt*P))*a;
    V2=inv((eye(n)+2/5*dt*P))*a;
    V3=inv((eye(n)+9/14*dt*P))*a;
    V4=inv((eye(n)+1/2*dt*P))*a;
    a=19/45*V1-2500/153*V2-2401/255*V3+79/3*V4;   %Wn of R(z) of Page 5
of Fasshauer 2004

end
```

h) Function: *TransformBack.m*

```
function [u uplus] =
TransformBack(index,y,c,a,small_ds,indexlength,kernelchoice)

for i = 1:indexlength
```

```
    if kernelchoice==10
        u(i)=sqrt((log(index(i))-y).^2 + c^2)*a;
        uplus(i)=sqrt((log(index(i)+small_ds)-y).^2 + c^2)*a;

    elseif kernelchoice==20
        u(i)=exp(-c^2*(log(index(i))-y).^2)*a;
        uplus(i)=exp(-c^2*(log(index(i)+small_ds)-y).^2)*a;

    elseif kernelchoice==30
        u(i)=((log(index(i))-y).^(2*c).*log(abs(log(index(i))-y)))*a;
        uplus(i)=((log(index(i)+small_ds)-
y).^(2*c).*log(abs(log(index(i)+small_ds)-y)))*a;
    end
end
```

i) Function: *Eigenvalue_Representation.m*

```
function alpha_by_eigen =
Eigenvalue_Representation(P,L,inv_L,IC,T,index,y,c,small_ds,indexlength,ker
nelchoice)
tic
[W,lamda] = eig(P);
k_coefficient=(W\inv_L*IC).*exp(-diag(lamda)*T);
alpha_by_eigen=W*k_coefficient;
[u uplus] =
TransformBack(index,y,c,real(alpha_by_eigen),small_ds,indexlength,kernelcho
ice);

delta=(uplus-u)/small_ds;

fprintf('\n') %line break in the command view
display('Above is the result of the Original Result')

[['Stock  '; 'Option '; 'Delta  '] num2str([index; u;delta])]
toc

fprintf('\n') %line break in the command view
display('Above is the result obtained by the Eigen Value Representation
Method')
```

8.2 Matlab Codes for Chapter 5

a) Main code: *two_Dimension_case.m*

```
% It is the main program for Extension 1
clear
tic
X = 10.0; r = 0.05; sigma1 = 0.25; sigma2=0.3; T = 0.75; n=21; m=30
rho=0.3; % correlation
Smin=1.0; Smax=30; %for both S1 and S2 (i.e. assumed similar scale)
index=2:4:14; % also for both S1 and S2
indexlength=size(index,2);
```

```
dt=T/m;
dy=log(Smax/Smin)/(n-1);
[y1 y2] = y1_y2_tuple(n,Smin,Smax,dy);
totallength=length(y1);    % i.e. totallength = n^2

% Terminal/ Initial Condition, see P. 734, McDonald (2006)
IC=max(X,max(exp(y1),exp(y2)));

[P inv_L c]= Two_Dim_Kernel(y1,y2,dy,rho,sigma1,sigma2,r);

% BD1
a=inv_L*IC;
for i = 1:m
a=a-dt*P*a;
end

u = TransformBack_2Dim(index,y1,y2,c,a,indexlength);
result=[0 index; index' u];
result=num2str(result);
result(1,1:5)='S1\S2';
result

toc
```

b) Function: *y1_y2_tuple.m*

```
function [y1 y2] = y1_y2_tuple(n,Smin,Smax,dy)

A=[log(Smin):dy:log(Smax)]'*ones(1,n);
y2=reshape(A,n^2,1);
y1=reshape(A',n^2,1);

end
```

c) Function: *Two_Dim_Kernel.m*

```
function [P inv_L c]= Two_Dim_Kernel(y1,y2,dy,rho,sigma1,sigma2,r);


deltaY1=meshgrid(y1,y1)'-meshgrid(y1,y1);
deltaY2=meshgrid(y2,y2)'-meshgrid(y2,y2);


%Hardy's MQ
c=4*dy;
L=sqrt(deltaY1.^2+deltaY2.^2+c^2);
D1L=deltaY1./L;
D2L=deltaY2./L;
D11L=1./L-deltaY1.^2./L.^3;
D22L=1./L-deltaY2.^2./L.^3;
D12L=-deltaY1.*deltaY2./L.^3;

inv_L=inv(L);
```

```
% from P. 701, McDonald (2006) (i.e. Multi-variate Black Scholes Equation)
P=L\(-rho*sigma1*sigma2*D12L-0.5*sigma1^2*D11L-(r-0.5*sigma1^2)*D1L-...
    sigma2^2*D22L-(r-0.5*sigma2^2)*D2L+r*L);

end
```

d) Function: *TransformBack_2Dim.m*

```
function u = TransformBack_2Dim(index,y1,y2,c,a,indexlength)

matrix1=meshgrid(index,index);
matrix2=meshgrid(index,index)';

for i = 1:indexlength
    for j=1:indexlength

        u(i,j)=sqrt((log(matrix1(i,j))-y1).^2+(log(matrix2(i,j))-y2).^2 +
c^2)'*a;

    end
end

end
```

8.3 Matlab Codes for Chapter 6

a) Main code: *Time_as_radius.m*

```
clear
tic
X = 10.0; r = 0.05; sigma = 0.2; T = 0.5; n=21; m=100;
Smin=1.0; Smax=30;
scale=150;
index=2:2:16;
indexlength=size(index,2);
dt=T/m;
k=1:n;
dy=log(Smax/Smin)/(n-1);
[y tao] = y_tao_tuple(n,m,Smin,Smax,dy,T,dt);
totallength=length(y);   % i.e. totallength = (m+1)*n

%Initialize BC1 and BC2 as column matrix
BC1=[0; 0];
BC2=[0; 0];
IC=[0; 0];

% Spatial and Time boundary condition
for i=1:m   % time
    BC1(i)=X*exp(-r*i*dt); % for European Put Option; i for tao = (m-i)
for t
    BC2(i)=0;
```

```
    IC=max(X-exp(y(2*m+1:2*m+n)),0);
end

lamda=scale*(log(Smax)/T)^2;
c=4*dy;

for i = 1:2*m+n
    for j = 1:totallength
        L_boundary(i,j)=sqrt((y(i)-y(j))^2+lamda*(tao(i)-
tao(j))^2+c^2);
    end
end

[L DL D2L DtaoL] = Space_Time_Kernel(c,n,m,totallength,y,tao,lamda);

P=-DtaoL+0.5*sigma^2*D2L+(r-0.5*sigma^2)*DL-r*L; %Operator

a=[L_boundary; P]\[BC1; BC2; IC;zeros(m*(n-2),1)];

for i = 1:indexlength
    u(i)=sqrt((log(index(i))-y).^2 + lamda*(T-tao).^2+c^2)'*a;
end
[index; u]
toc
```

b) Function: *y_tao_tuple.m*

```
function [y tao] = y_tao_tuple(n,m,Smin,Smax,dy,T,dt)

y=log(Smin)*ones(m,1); % Spatial lower boundary
tao=[dt:dt:T]';

y=[y;log(Smax)*ones(m,1)]; % Spatial upper boundary
tao=[tao;tao];

y=[y; [log(Smin):dy:log(Smax)]']; % Time Initial Boundary
tao=[tao;zeros(n,1)];

% points in the middle
A=[log(Smin)+dy:dy:log(Smax)-dy]'*ones(1,m);
y=[y; reshape(A,(n-2)*m,1)];
clear A
A=[dt:dt:T]'*ones(1,n-2);
tao=[tao; reshape(A',(n-2)*m,1)];

end
```

c) Function: *Space_Time_Kernel.m*

```
function [L DL D2L DtaoL] =
Space_Time_Kernel(c,n,m,totallength,y,tao,lamda)

deltaY=meshgrid(y(2*m+n+1:totallength),y)'-
```

```
meshgrid(y,y(2*m+n+1:totallength));
deltaTao=meshgrid(tao(2*m+n+1:totallength),tao)'-
meshgrid(tao,tao(2*m+n+1:totallength));
L=sqrt(deltaY.^2+lamda*deltaTao.^2+c^2);
DL=deltaY./L;
D2L=1./L-deltaY.^2./L.^3;
DtaoL=lamda*deltaTao./L;

end
```