SCAT Mobility Grant Report

MQ-RBF Meshless Method for solving CFD problems using an Object-Oriented Approach

by
Luis M. de la Cruz
National Autonomous University of Mexico (UNAM)

Supervisor: Dr. David Emerson,
Computational Engineering Group, STFC-CSED,
Daresbury Laboratory, UK

June 9, 2008

Abstract

Radial Basis Functions (RBF) are well-known as powerful tools for multivariate interpolation from scattered data. Just very recently, RBFs have gained enormous popularity in mesh-free methods for Partial Differential Equations (PDEs). First known applications of RBFs in Computational Fluid Dynamics (CFD) are those from Edward Kansa in 1990, and nowadays this technique has been succesfully used in several practical problems. Various authors have been proved that the Multiquadric (MQ) kernel enjoy exponential convergence. On the other hand, the primary disadvantage of the direct MQ-RBF scheme is that it is global, hence the coefficient matrices obtained from this discretization scheme become full and progressively more ill-conditioned as the rank increases. The ill-conditioning problem has been addressed in various works and there exist several ways to improve the condition number of the coefficient matrices. The techniques include the construction of special preconditioners and the use of Domain Decomposition methods (DD). In this work, the unsymmetric collocation RBF mesh-less method is used to solve typical CFD benchmarks and the principal characteristics of MQ-RBF are studied. To this end, a framework using the Object-Oriented Paradigm (OOP) in combination with the C++ language was constructed. Currently this tool contains classes for generation of points in simple geometries, Thin-Plate Splines (TPS) and MQ RBF kernels, GMRES and Gauss elimination solvers, ACBF preconditioner, KDTree algorithm and classes for solving problems with the alternating Schwartz domain decomposition method in multiprocessor architectures via the use of the MPI library. Solution of Poisson equation, advection-diffusion in 1D and 2D, and incompressible viscous flows problems are presented.

Contents

1 Introduction

CFD is the analysis of systems involving fluid flow, heat transfer and asssociated phenomena such as chemical reactions by means of computer-based simulation. CFD codes are structured around the numerical algorithms that can tackle fluid flow problems. A typical CFD modelling involves three principal steps: (1) Theoretical Model, where physical laws are applied to describe the studied phenomenon in terms of a set of mathematical equations, (2) Discrete Model, where the equations are written in terms of simple algebraic operations and (3) Solution of the resultant linear systems.

In the second step of this process it is required to decide the manner in which the governing equations will be discretized in order to obtain approximated solutions. In the past four decades numerical simulation in CFD has been dominated by either finite difference methods (FDM), finite element methods (FEM), and finite volume methods (FVM), which require a mesh to support the localized approximations. Typically, the simulations are done in complicated two and three-dimensional geometries, in such a way that the construction of a mesh in these domains is a non-trivial problem, and more than the 70% of overall computation is spent by mesh generators.

In the last decade, the so-called meshless methods for PDEs has been intensely studied because its ability of dealing with PDEs in complex domains without a mesh at all. The popular meshless methods include moving least square method [1, 2], generalized finite element method [3] and radial basis function (RBF) [4, 5, 6].

In 1982 Franke [7] compared 29 interpolation methods with analytic two-dimensional test functions. Accordingly, one of the most powerful methods is the continuously differentiable multiquadric MQ-RBF, discovered by Hardy [8, 9]. The MQ-RBF for the interpolation problem has been shown by various authors to possess some very powerful properties. Madych and Nelson [10] proved that interpolation with the MQ-RBF is exponentially convergent.

In 1990, Kansa [4, 5] modified Hardys MQ method [9] to solve partial differential equations. Since then, solving PDEs using RBFs has been used for different sort of applications. Fedoseyev *et al.* [11] demonstrated that the solutions of elliptic PDEs converge exponentially requiring orders of magnitude less points and operations than FDM, FEM and FVM. The main advantages of the MQ-RBF scheme over the traditional methods are that enjoys superior convergence rates, requires less points and is easy to implement in more than one dimensions. On the other side, the principal disadvantage of applying MQ-RBFs to PDE systems is that the resulting coefficient matrix can become quite ill-conditioned as N, the rank of the matrix, increases.

The cost of increasing the accuracy via RBF is usually the ill-conditioning of the associated linear systems that need to be solved: **better conditioning is associated with poorer accuracy and worse conditioning is associated with improved accuracy**. Different approaches have already been proposed to overcome the difficulties, see for example [12, 13, 14, 15, 16, 17, 18, 19]. In these works several ideas are proposed:

- Domain Decomposition Methods (DDMs).
- Use of Krylov solvers (CG, GMRES, etc.) in conjunction with simple and specialized preconditioners.
- Use of a variable shape parameter as a function of the local radius of curvature.
- Use of truncated MQ basis function.
- Optimization of knots distribution.
- Multilevel approximation schemes developed by Fasshauer and Jerome [20], to keep the band-width constant, but refine spatial regions to the desired degree of accuracy.

Particularly, in [14] and [19] two specialized preconditioners for improving the condition number of the matrices are studied. In these two works a preconditioner is constructed using Approximated Cardinal Basis Functions (ACBF). Brown *et al.* [21] compares both preconditioners and found that the LS-ACBF from [19]

is relatively easy to setup and performs better for very bad conditioned matrices. Also it works better for time-dependent problems. The comparison were made in terms of GMRES iterations.

Tipically, when MQ-RBF is applied to solve PDEs using the method from [5], we found that the error is largest near the boundary, by one or two orders of magnitude than those in the domain far from the boundary. A method that improves the error near the boundary is proposed in [11], resulting in a better global accuracy. The main idea is to add an aditional set of points adjacent to the boundary (inside or outside) and, an additional set of collocations equations obtained via collocation of the PDE on the boundary. Adding nodes near the boundary may give rise to troublesome issues and the solution depends on the distribution of this additional set of points. Similarly, when Neumann boundary conditions are imposed the accuracy is poorer compared with pure Dirichlet boundary conditions. In [22] is observed that one solution to this problem is either by refining the mesh size (h-scheme), or by increasing the shape parameter (c-scheme). Both schemes significantly increase the ill-conditioning of the matrices that causes instabilities in the solution. To mitigate the ill-conditioning, an improved truncated singular value decomposition method can be used to solve the systems.

RBF mesh-less methods are recent techniques to deal with PDEs, and the typical tools for testing these new methods is by implementing the algorithms inside high-level frameworks like Mathlab. However, this kind of frameworks are not intended for High-Performance Computing (HPC). The main objetive in this work is to construct a framework using the C++ language in order to provide the tools for an easy developing and testing of the RBF techniques, allowing to run the codes on HPC plataforms. The C++ language was selected because it is possible to construct good and efficient object-oriented code. In general OO designs allows better encapsulation and separation of concerns, thus providing a high degree of modularity and flexibility and greatly increase the potential for reuse of single systems components [23]. The fact that OOP provides these tools is not enough to obtain good quality software, it is also necessary to use a software development process that guides the analysis, design and implemention of the different parts of the code. A simplified software process specially adapted for scientific computing simulations, and based on the Unified Process of Development [24] is used here. To test and calibrate the tools, a set of *unit tests* were defined, and these are PDEs equations coming from CFD benchmarks. Fortran and C libraries exist that provide good performance, so the idea here is not to redo those libraries. Instead, C++ wrappers are used to link with existent high-performance libraries.

2 Radial Basis Functions Meshless Method

The motivation for RBF was originated from applications in geodesy, geophysics, mapping, or meteorology. Later, applications were found in many areas such as in the numerical solution of PDEs, artificial intelligence, learning theory, neural networks, signal processing, sampling theory, statistics, finance, and optimization. It should be pointed out that meshfree local regression methods have been used independently in statistics for more than 100 years, see for example [25] and references therein. Recently, RBFs have been applied in solving PDEs. The theory includes Galerkin methods [26], collocation methods [27, 28], and multilevel schemes [29].

In this section the RBFs methodology for interpolation of scattered data and for solving PDEs is introduced briefly.

2.1 RBF definition

A radial function is defined as:

$$\Phi : \mathbb{R}^d \to \mathbb{R} : (\vec{x}) \to \phi(||\vec{x}||) \tag{1}$$

for some univariate function $\phi : [0, \infty) \to \mathbb{R}$, where $\vec{x} \in \mathbb{R}^d$, and $|| \cdot ||$ is the usual Euclidean norm of $\vec{x} = (x_1, \ldots, x_d)$ and is defined as:

$$||\vec{x}|| = \sqrt{\sum_{i=1}^{d} x_i^2} \tag{2}$$

The equation (1) says that the function value of $\Phi(x)$, only depends on the norm of $\vec{x}$, therefore we have a radial function. If $||\vec{x_1}|| = ||\vec{x_2}||$ then $\Phi(\vec{x_1}) = \Phi(\vec{x_2})$, for $\vec{x_1}, \vec{x_2} \in \mathbb{R}^d$. A useful feature of radial functions is the fact that the interpolation problem becomes insensitive to the dimension d of the space. Instead of having to deal with a multivariate function Φ (whose complexity will increase with the space dimension) it is possible to work with the same univariate function ϕ for all choices of d.

Suppose now, we have a set of points fixed (sometimes called centers) $\{\vec{x}_j\}_{j=1}^N = \{\vec{x}_1, \ldots, \vec{x}_N\} \subset \mathbb{R}^d$ and consider the following linear combination of the function Φ centered at the points $\vec{x}_j$:

$$\tilde{u}(\vec{x}) = \sum_{j=1}^{N} \lambda_j \Phi(\vec{x} - \vec{x}_j) = \sum_{j=1}^{N} \lambda_j \phi(||\vec{x} - \vec{x}_j||) = \sum_{j=1}^{N} \lambda_j \phi(r) \tag{3}$$

where $r = ||\vec{x} - \vec{x}_j||$ is the Euclidean distance between the points $\vec{x}$ and $\vec{x}_j$ and $\{\lambda_j\}_{j=1}^N$ represent a set of unknown coefficients. The function ϕ is the so-called Radial Basis Function, and $\tilde{u}$ is the interpolant that approximates the function u in a given domain.

There are several possible choices for the RBF kernel ϕ, the table 1 lists the most widely used RBF. The parameter c that appears in several of the RBF kernels is known as the shape parameter. Note that the radial basis functions ϕ can be either globally supported or compactly supported.

In the case of the CS-RBF, ρ represents the radius of support whose magnitudes affects the accuracy of the approximation. The notation + is defined as:

$$\left(1 - \frac{r}{\rho}\right)_+^4 = \begin{cases} \left(1 - \frac{r}{\rho}\right)^4 & \text{if } 0 \leq \frac{r}{\rho} \leq 1, \\ 0 & \text{if } \frac{r}{\rho} < 1. \end{cases} \tag{4}$$

More details about RBF can be found in [6, 30, 31].

RBF	Definition
Multiquadric (MQ)	$\phi(r,c) = (r^2 + c^2)^{\beta/2}, \beta = 1, 3, 5, \ldots, 2N+1, \ldots$
Inverse Multiquadric (IMQ)	$\phi(r,c) = (r^2 + c^2)^{-\beta/2}, \beta = 1, 3, 5, \ldots, 2N+1, \ldots$
Gaussian (GA)	$\phi(r,c) = \exp\{-c^2 r^2\}$
Thin-Plate Splines (TPS)	$\phi(r) = r^\beta \log(r), \beta = 2, 4, 6, \ldots, 2N, \ldots$
Smooth Splines (SS)	$\phi(r) = r^\beta, \beta = 1, 3, 5, \ldots, 2N+1, \ldots$
Compactly Supported RBFs (CS-RBF)	$\left(1 - \frac{r}{\rho}\right)_+^4 \left(1 + 4\frac{r}{\rho}\right), d \leq 3$

Table 1: Global Radial Basis Functions.

2.2 Interpolation with RBF

Radial Basis Functions can be used to intepolate scattered data to any point inside the domain of interest. Suppose we know the values of the function $u : \mathbb{R}^d \to \mathbb{R}$ at a set of N fixed points $X = \{\vec{x}_j\}_{j=1}^N \subset \mathbb{R}^d$. Then a RBF basically defines a spatial mapping which maps any location $\vec{x} \in \mathbb{R}^d$ to a value $\tilde{u}(\vec{x})$ represented by:

$$\tilde{u}(\vec{x}) = \sum_{j=1}^{N} \lambda_j \phi(||\vec{x} - \vec{x}_j||). \tag{5}$$

In order to find the coefficients λ_j, we write the above equation for all fixed points where the value of $u(\vec{x}_i)$ is known, that is:

$$u(\vec{x}_i) = u_i = \sum_{j=1}^{N} \lambda_j \phi_j(r_i), \quad \text{for } i = 1, \ldots, N, \tag{6}$$

where $\phi_j(r_i) = \phi(||\vec{x}_i - \vec{x}_j||)$ and $r_i = ||\vec{x}_i - \vec{x}_j||$ is the distance between $\vec{x}_i$ and $\vec{x}_j$. This equation can be written in matrix form as follows:

$$\begin{pmatrix} \phi_1(r_1) & \phi_2(r_1) & \ldots & \phi_N(r_1) \\ \phi_1(r_2) & \phi_2(r_2) & \ldots & \phi_N(r_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(r_N) & \phi_2(r_N) & \ldots & \phi_N(r_N) \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_N \end{pmatrix} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{pmatrix} \tag{7}$$

Observe that the above system is completely full and the terms $\phi_j(r_i)$ for $i = j$ could be equal to zero depending on the RBF chosen.

2.3 Solving PDEs with RBFs

The first attempts to solve PDEs with RBF techniques where taken by Kansa [4, 5] in 1990. Since then, several studies have been conducted to investigate the applicability of this method to numerically approximate the solution of PDEs coming from several fields of study. In constrast to standard numerical methods like Finite Differences, Finite Volume or Finite Element, in RBF techniques the differential operators of a PDE are never discretized, instead each operator is applied to the basis function directly.

Consider the following general form of a boundary value problem:

$$\begin{aligned} \mathcal{L}u(\vec{x}) &= f(\vec{x}) \text{ in } \Omega \in \mathbb{R}^d \\ \mathcal{B}u(\vec{x}) &= h(\vec{x}) \text{ on } \partial\Omega, \end{aligned} \tag{8}$$

where $\vec{x} \in \Omega$, $\mathcal{L}$ and $\mathcal{B}$ are arbitrary differential operators in the domain Ω and on the boundary $\partial\Omega$, respectively.

Assume there exist N total collocation points $X = \{\vec{x}_j\}_{j=1}^{N} \in \bar{\Omega}$, where $\bar{\Omega} = \Omega \cup \partial\Omega$ is known as the closure of the domain. Define N_I as the number of interior points in Ω and N_B as the number of points on the boundary $\partial\Omega$, in such a way that $N = N_I + N_B$. By substituting equation (6) into (8) the boundary value problem can be written as follows:

$$\begin{aligned} \sum_{j=1}^{N} \lambda_j \mathcal{L}\phi_j(\vec{x}_i) = f(\vec{x}_i) = f_i, \quad \text{for} \quad i = 1, \ldots, N_I \\ \sum_{j=1}^{N} \lambda_j \mathcal{B}\phi_j(\vec{x}_i) = h(\vec{x}_i) = h_i, \quad \text{for} \quad i = N_I + 1, \ldots, N. \end{aligned} \tag{9}$$

To find the λ_j coefficients it is necessary to solve the next $N \times N$ linear algebraic system:

$$\left(\begin{array}{ccc|ccc} \mathcal{L}\phi_1(r_1) & \ldots & \mathcal{L}\phi_{N_I}(r_1) & \mathcal{L}\phi_{N_I+1}(r_1) & \ldots & \mathcal{L}\phi_N(r_1) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \mathcal{L}\phi_1(r_{N_I}) & \ldots & \mathcal{L}\phi_{N_I}(r_{N_I}) & \mathcal{L}\phi_{N_I+1}(r_{N_I}) & \ldots & \mathcal{L}\phi_N(r_{N_I}) \\ \hline \mathcal{B}\phi_1(r_{N_I+1}) & \ldots & \mathcal{B}\phi_{N_I}(r_{N_I+1}) & \mathcal{B}\phi_{N_I+1}(r_{N_I+1}) & \ldots & \mathcal{B}\phi_N(r_{N_I+1}) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \mathcal{B}\phi_1(r_N) & \ldots & \mathcal{B}\phi_{NI}(r_N) & \mathcal{B}\phi_{N_I+1}(r_N) & \ldots & \mathcal{B}\phi_N(r_N) \end{array}\right) \left(\begin{array}{c} \lambda_1 \\ \vdots \\ \lambda_{N_I} \\ \hline \lambda_{N_I+1} \\ \vdots \\ \lambda_N \end{array}\right) = \left(\begin{array}{c} f_1 \\ \vdots \\ f_{N_I} \\ \hline h_{N_I+1} \\ \vdots \\ h_N \end{array}\right) \tag{10}$$

Defining sub-matrices WL_{11}, WL_{12}, WB_{21} and WB_{22}

$$\begin{array}{ll} WL_{11} & \text{with elements } \mathcal{L}\phi_j(r_i), \quad \text{for } i, j = 1, \ldots, N_I, \\ WL_{12} & \text{with elements } \mathcal{L}\phi_j(r_i), \quad \text{for } i = 1, \ldots, N_I,\ j = N_I + 1, \ldots, N, \\ WB_{21} & \text{with elements } \mathcal{B}\phi_j(r_i), \quad \text{for } i = N_I + 1, \ldots, N,\ j = 1, \ldots, N_I, \\ WB_{22} & \text{with elements } \mathcal{B}\phi_j(r_i), \quad \text{for } i = N_I + 1, \ldots, N,\ j = N_I + 1, \ldots, N, \end{array}$$

and the vectors $\Lambda = [\lambda_1, \ldots, \lambda_N)]^T$, $F = [f_1, \ldots, f_{N_I}]^T$ and $H = [h_{N+1}, \ldots, g_N]^T$, the system (10) can be written as follows:

$$G = \begin{bmatrix} WL_{11} & WL_{12} \\ WB_{21} & WB_{22} \end{bmatrix} \Lambda = \begin{bmatrix} F \\ H \end{bmatrix} \tag{11}$$

where G is known as the Gramm's Matrix.

The solution of system (11) give us the coefficients λ_j that are required to approximate u by using (5). In principle, with this technique it is possible to find the value of u in any location inside the domain Ω.

As can be observed, the collocation method with RBFs is very simple and truly mesh-less method. Also it can be applied directly to one, two, three or more dimensions. On the other side, the matrix of the system result fully populated (except for the CS-RBFs), unsymmetrical and in the mostly of the cases ill-conditioned. Several authors had been addressed the ill-conditioning problem, see for example [14, 15, 16, 18, 19]. To solve this kind of linear systems it is possible to use Gauss elimination with partial pivoting or iterative methods like GMRES. However, Gauss elimination require $O(N^3)$ and is very expensive for many collocation points. Iterative algorithms, as in the case of GMRES, require specialized preconditioners in order to reduce the number of iterations in generating approximated solutions in short times.

2.3.1 Polynomial precision

Sometimes the approximation given in the RBF expansion presented in equation (6) is extended by adding a polynomial as follows:

$$u_i = \sum_{j=1}^{N} \lambda_j \phi_j(r_i) + \sum_{k=1}^{M} a_k p_k(\vec{x}_i), \quad \text{for } i = 1, \ldots, N. \tag{12}$$

where the terms $p_k(\vec{x}_i)$ form a basis for the $M = \binom{d+m-1}{m-1}$-dimensional linear space $\prod_{m-1}^{d}$ of polynomials of total degree less than or equal to $m-1$ in d variables. The addition of a polynomial leads us to a system of N linear equations in $N+M$ unknowns, therefore M additional (orthogonality) conditions are required to ensure a unique solution. These conditions are known as the moment constrains and are written as follows:

$$\sum_{j=1}^{N} \lambda_j p_k = 0, \text{ for } k = 1, \ldots, M, \tag{13}$$

While the use of polynomials is somewhat arbitrary, it is expected that the addition of polynomials of total degree at most $m-1$ guarantees polynomial precision, see [6].

Substituting equation (12) in the equation (8), results in

$$\begin{aligned} \sum_{j=1}^{N} \lambda_j \mathcal{L}\phi_j(\vec{x}_i) + \sum_{k=1}^{M} a_k \mathcal{L}p_k(\vec{x}_i) = f(\vec{x}_i) = f_i, \quad \text{for} \quad i = 1, \ldots, N_I \\ \sum_{j=1}^{N} \lambda_j \mathcal{B}\phi_j(\vec{x}_i) + \sum_{k=1}^{M} a_k \mathcal{B}p_k(\vec{x}_i) = h(\vec{x}_i) = h_i, \quad \text{for} \quad i = N_I + 1, \ldots, N. \end{aligned} \tag{14}$$

The system (14) along with the moment constrains (13) can be written in matrix form as follows

$$\begin{bmatrix} WL & PL \\ WB & PB \\ P^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \Lambda \\ A \end{bmatrix} = \begin{bmatrix} F \\ H \\ \vec{0} \end{bmatrix} \tag{15}$$

where $A = [a_1, \ldots, a_M]^T$ is the vector of coefficients for the polynomial, $\mathbf{0}$ is an $M \times M$ matrix consisting of zero elements, $\vec{0}$ is a vector of M zero elements and:

WL	is an $N_I \times N$ matrix with elements $\mathcal{L}\phi_j(r_i)$,	for $i = 1, \ldots, N_I$,	$j = 1, \ldots, N$,
WB	is an $N_B \times N$ matrix with elements $\mathcal{B}\phi_j(r_i)$,	for $i = N_I + 1, \ldots, N$,	$j = 1, \ldots, N$,
PL	is an $N_I \times M$ matrix with elements $\mathcal{L}p_k(\vec{x}_i)$,	for $i = 1, \ldots, N_I$,	$k = 1, \ldots, M$,
PB	is an $N_B \times M$ matrix with elements $\mathcal{B}p_k(\vec{x}_i)$,	for $i = N_I + 1, \ldots, N$,	$k = 1, \ldots, M$.
P^T	is an $M \times N$ matrix with elements $p_k(\vec{x}_i)$,	for $i = 1, \ldots, N$,	$k = 1, \ldots, M$.

2.4 Ill-conditioned linear systems

The matrix systems given in (11) and (15) are generally non-symmetric and full. These systems of equations are known to be ill-conditioned, even for moderate N. This ill-conditioning worsens with N or with a flat RBF, for example with the MQ-RBF with large shape parameter c. Although some very rare combinations of data center arrangements and c can produce a singular matrix, the singularity can be removed by perturbing either the value of c or the data centers.

There is ample evidence that RBFs, especially MQ-RBFs, offers some computational advantages over traditional methods, particularly is a truly mesh-free scheme that possesses very high orders rates of convergence. For small to moderate sized problems, RBFs do outperform traditional methods. The main concern is whether RBFs can be computationally efficient with large scale, complex problems. To circumvent the ill-conditioning problem, for large N it is possible to adapt the procedures used in large scale mesh-based PDE methods by combining preconditioning, domain decomposition, and using parallel computers.

2.4.1 The GMRES iterative method

Solving large systems coming from RBF technique with non-customised methods is computationally expensive. For example, using the usual direct methods to fit an RBF with N centres requires $O(N^2)$ storage and $O(N^3)$ flops. Thus such an approach is not viable for large problems with $N \gg 10,000$. To overcome this situation, the combination of an efficient iterative algorithm and a preconditionator is required.

The GMRES iterative method for non-symmetric systems is a Krylov subspace method and its convergence properties parallel those of the conjugate gradient (CG) method for symmetric positive definite systems. Unlike CG iteration, GMRES iteration requires storage of all the previous search directions, or equivalently storage of an orthonormal basis for k_k, where

$$k_k = \text{span}\{r_0, Ar_0, \ldots, A^{k1}r_0\},$$

and $r_k = b - Ax_k$ is the residual at the kth step of GMRES. In the kth iteration of GMRES x_k is taken as the unique solution of the least squares problem

$$\min_{x \in x_0 + k_k} ||b - Ax||_2$$

The kth iteration of the GMRES algorithm on an $N \times N$ system requires one matrix-vector product and $O(kN)$ additional floating point operations. The method also requires the storage of an orthonormal basis for the Krylov subspace so that conjugate vectors can be formed at each iteration. Hence, if the total number of iterations is K, total storage requirements, excluding any storage needed for the matrix A or computing its action, is $O(KN)$. The corresponding flop count is K matrix-vector products and $O(K^2N)$ other floating point operations. More details of the algortihm can be found in [32].

Due to its features, GMRES is one of the algorithms commonly used to solve the full systems that appears in RBF techniques, and will be used in this work in combination with the preconditioner described in the next section.

2.4.2 Approximated Cardinal Basis Functions Preconditioner

Given a matrix system $G\alpha = b$ as in (11), it is possible to construct a left-hand preconditioner W, and to solve the equivalent system $WG\alpha = Wb$, for the undetermined coefficients α. The application of such a preconditioner to a bad-conditioned matrix, results in clustering the eigenvalues of the matrix and therefore reducing the total number of iterations required for GMRES (or other algorithm) to converge. Using a basis of cardinal functions, see [14], which would be equivalent to a delta-function, $\delta(\vec{x}_i)$ that is one at its center $\vec{x}_i$ and zero everywhere else, would result in $WG = I$, where I represents the identity matrix. For this matrix GMRES would converge in one iteration. However, this is totally impractical because the preconditioner would need to be exactly the matrix inverse. The strategy that is used instead, is to construct the preconditioner using approximate cardinal basis functions (ACBF).

In this work the LS-ACBF preconditioner, proposed by Ling *et al.* [19] is used to improve the condition numbers of the linear systems. The idea in the construction of LS-ACBF preconditioner is to satisfy the above cardinal condition in the least-squares sense. A brief description of this preconditioner is given below.

Let ϕ_I and ϕ_B denote the N_I and N_B RBFs whose center is in Ω and in $\partial\Omega$ respectively. The rows of the matrix G, from equation (11), are discrete function values given by:

$$\{\Psi(\vec{x})\}_{i=1}^{N} = \{\{\mathcal{L}\phi_I(\vec{x}_i - \vec{x})\}_{i=1}^{N_I} \cup \{\mathcal{B}\phi_B(\vec{x}_i - \vec{x})\}_{i=1}^{N_B}\} \tag{16}$$

Each column of G has contributions from N_I entries of $\mathcal{L}\phi_I$ and N_B entries of $\mathcal{B}\phi_B$. For each center $\vec{x}_i$, select a small subset of centers or support of size $m \ll N$ indicated by the index set

$$S_i = [s_i^{(1)}, s_i^{(2)}, \ldots, s_i^{(m)}], i \in S_i, \tag{17}$$

such that we try to enforce the condition

$$\sum_{j \in S_i} w_j \Psi_j(\vec{x}) = \delta(\vec{x}), \text{ for all } \vec{x} \in X, \tag{18}$$

where $\delta_i(\vec{x})$ is one at $\vec{x}_i$ and zero elsewhere.

The index set S_i is a combination of local nearest centers to $\vec{x}_i$ and special points distributed on the domain. S_i will be the m_l local nearest centers to $\vec{x}_i$ that can be found efficiently in $O(NlogN)$ flops, and m_s special points where $m_l + m_s = m$.

In this work the points are ordered using the kD-tree structure. The main idea is to decompose the multidimensional space into hyperrectangles, using splitting planes perpendicular to the coordinate system axes. Every node of a kD-tree, from the root to the leaves, stores a point, as a consequence, each splitting plane must go through one of the points in the kD-tree, see figure 1. More on this kind of techniques can be found in [33].

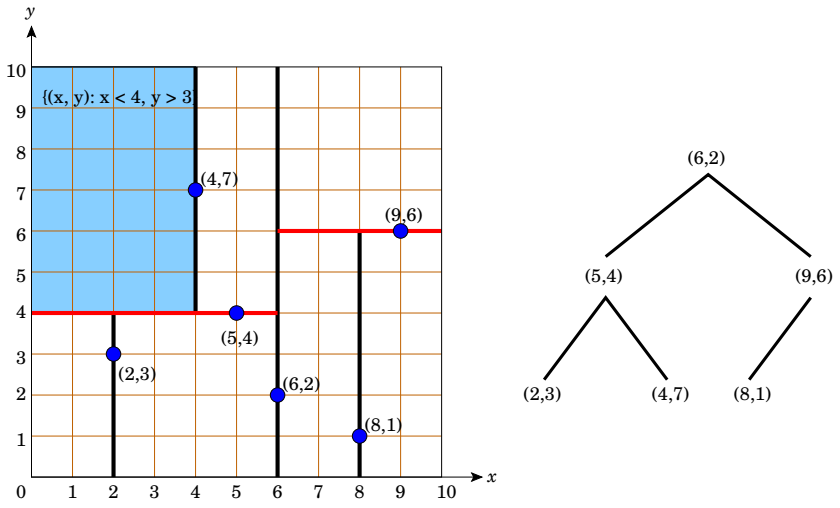


Figure 1: Example of a kDtree structure in 2D.
The first vertical splitting plane, passing by (6,2) divides the domain in two sub-rectangles. Each sub-rectangle contains the half (or so) of the total number of points. Then two horizontal splitting planes (red), passing by (5,4) and (9,6) divide the two sub-rectangles in four, in such a way that the four sub-rectangles contains almost the same amount of points, and so on.

A number of special points to control the shape and to ensure the linear independence of all the ACBFs in the domain Ω, are chosen adequately. When the nearest neighbor subset is unbalanced about $\vec{x}_i$, the least-squares ACBFs centered at different points near the boundary points may become too similar in shape, have considerable overlap, and become considerably less linear independent without the presence of special points. In general, the special points are taken to be located on the boundary of the domain. For an explanation of this behaviour refers to [19].

Equation (18) yields N equations for $m < N$ unknowns. Let B_i be the $m \times N$ matrix formed by selecting m rows of G from the index set S_i. This matrix is written as follows:

$$B_i = \begin{bmatrix} G(s_i^{(1)},1) & G(s_i^{(1)},2) & \dots & G(s_i^{(1)},N) \\ G(s_i^{(2)},1) & G(s_i^{(2)},2) & \dots & G(s_i^{(2)},N) \\ \vdots & \vdots & \vdots & \vdots \\ G(s_i^{(m)},1) & G(s_i^{(m)},2) & \dots & G(s_i^{(m)},N) \end{bmatrix} \in \mathbb{R}^{m\times N}. \tag{19}$$

Then, is possible to rewrite (18) in matrix form as follows:

$$B_i^T \vec{w}_i = \vec{e}_i, \tag{20}$$

where $\vec{w}_i^T = [w_i^{(1)}, w_i^{(2)}, \dots, w_i^{(m)}]$ are the nonzero elements of the ith row of W, and $\vec{e}_i$ is the ith $N\times 1$ standard canonical vector. Each center $\vec{x}_i$ is associated with a row of the preconditioner W that approximates the ith ACBF using m different rows of G. Since B_i has full rank m, $B_iB_i^T$ is a nonsingular $m\times m$ matrix. Thus, $\vec{w}_i$ can be uniquely determined.

The final shape of the preconditioner W will have $\vec{w}_i^T P$ as its rows where P is the $m\times N$ permutation matrix that maps the elements of $\vec{w}_i$ from S_i back to the corresponding global index set of the ith row of W. Mathematically this is expressed in the next formula:

$$W_{ij} = \begin{cases} w_i^{(k)} & \text{if } j = s_i^{(k)} \text{ for } k = 1,\dots,m, \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

The preconditioner W is obtained after solving N least-squares problems. Each row of W will operate on the correct rows of G and the preconditioned matrix WG will have the ACBFs as its rows. The overdetermined linear system (20) can be solved in the least-squares sense: problem has a unique solution $\vec{w}$ that minimizes $||B_i^T\vec{w} - \vec{e}_i||_2$. This problem can be solved via the corresponding local normal equation (L-NE). Multiplying the equation (20) by B_i from the left to gives the normal equations,

$$B_iB_i^T\vec{w}_i = B_i\vec{e}_i. \tag{22}$$

Note that the $m\times m$ normal matrix $B_iB^T \in R^{m\times m}$ on the left of (22) is a submatrix of GG^T, namely the (S_i, S_i) elements of GG^T:

$$B_iB_i^T = \begin{bmatrix} GG^T(s_i^{(1)},s_i^{(1)}) & GG^T(s_i^{(1)},s_i^{(2)}) & \dots & GG^T(s_i^{(1)},s_i^{(m)}) \\ GG^T(s_i^{(2)},s_i^{(1)}) & GG^T(s_i^{(2)},s_i^{(2)}) & \dots & GG^T(s_i^{(2)},s_i^{(m)}) \\ \vdots & \vdots & \vdots & \vdots \\ GG^T(s_i^{(m)},s_i^{(1)}) & GG^T(s_i^{(m)},s_i^{(2)}) & \dots & GG^T(s_i^{(m)},s_i^{(m)}) \end{bmatrix}. \tag{23}$$

Note that only $O(mN)$ elements of GG^T are needed for all N normal matrices, taking this into account a considerable amount of storage and computational savings is obtained. From (23) can be observed that the $m\times m$ elements of GG^T correspond to a particular normal equation, $B_iB_i^T$. The union of all the relevant elements $i = 1, 2, \dots, N$ is depicted in figure 2.

The QR factorization and the SVD methods can also be used to solve (20) directly. More details about the preconditioner and alternate algorithms to solve the least-squares problem can be found in [19, 21].

2.4.3 Domain Decomposition: Additive Schwarz Method

The earliest concept of domain decomposition method (DDM) was introduced as a classical Schwarz alternating algorithm by Schwarz in 1870, which provided a parallel, potentially fast, and robust algorithm for the solution of linear or non-linear systems of equations resulting usually from discretizations of PDEs. With the advent of powerful supercomputers, the DDM has been well developed for FDM, FEM and FVM, see for example [34]. Recently the additive and multiplicative Schwarz iterative techniques has been incorporated into the RBFs

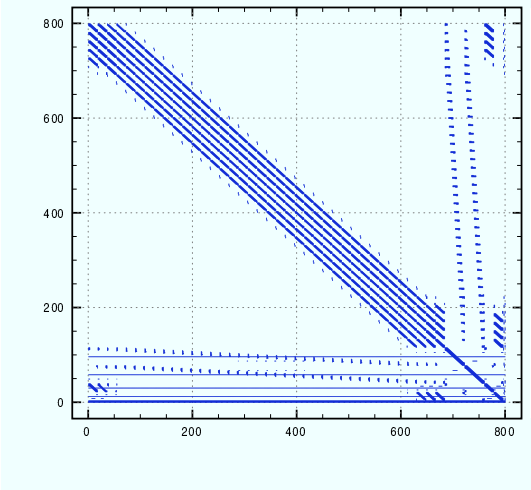


Figure 2: Relevant elements of GG^T necessary for the construction of all $B_i B_i^T$ matrices, for the heat transfer example presented in section 4.1

for solving large scale problems [35, 17, 18]. Numerical results show that this provides an alternative to avoid the ill-conditioning problem by solving many small subdomain problems instead of one global large domain problem.

Consider the boundary value problem defined in the equation (8) for the domain Ω shown in figure 3. In figure 3 the next definitions are valid: $\Omega = \Omega_1 \cup \Omega_2$; $\Omega, \Omega_1, \Omega_2$ are open; $\partial\Omega$ is the real boundary of Ω; $\bar{\Omega} = \Omega \cup \partial\Omega$ is the closure of the domain; Γ_i is the artificial boundary of Ω_i which lies inside of Ω; $\partial\Omega_i \backslash \Gamma_i$ is the boundary of Ω_i without Γ_i; u_i^n is the approximated solution in $\bar{\Omega}_i$ after n iterations; $u_i^n|_{\Gamma_j}$ is the restriction of u_i^n in Γ_j, for $i \neq j$.

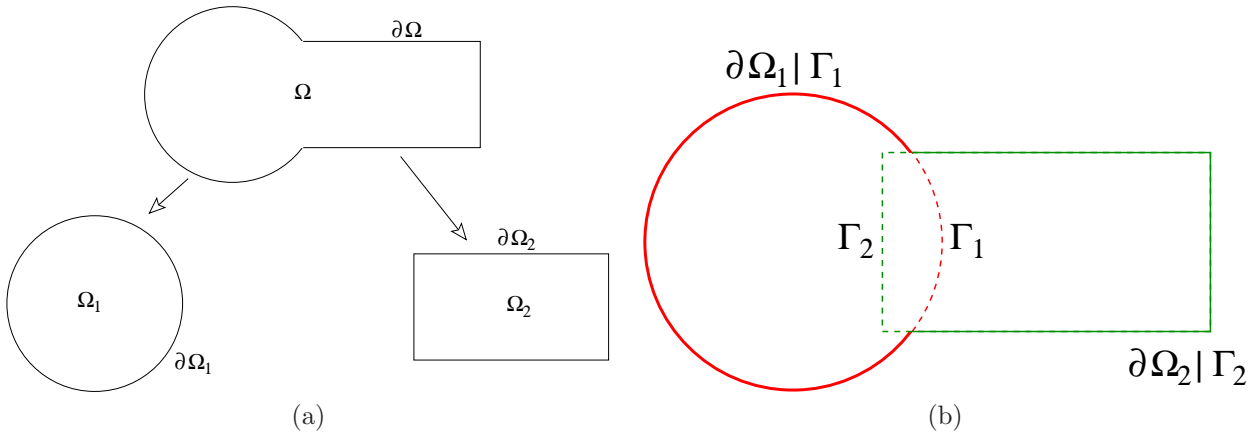


Figure 3: (a) Global domain and (b) its partition in two subdomains.

The classical alternating Schwarz algorithm is given by

$$\begin{aligned} \mathcal{L}u_1^n &= f \text{ in } \Omega_1 \\ \mathcal{B}u_1^n &= g \text{ on } \partial\Omega_1\backslash\Gamma_1 \\ u_1^n &= u_2^{n-1}|_{\Gamma_1} \text{ on } \Gamma_1 \end{aligned}$$

and

$$\begin{aligned} Lu_2^n &= f \text{ in } \Omega_2 \\ u_2^n &= g \text{ on } \partial\Omega_2\backslash\Gamma_2 \\ u_2^n &= u_1^n|_{\Gamma_2} \text{ on } \Gamma_2 \end{aligned}$$

For each subdomain, the same PDE is solved in the interior Ω_i and on the natural boundary condition $\partial\Omega_i\backslash\Gamma_i$. A Dirichlet condition is imposed on the artificial boundary Γ_i, so that numerical solution on subdomain Ω_i matches the newest approximation on Ω_i. All initial guesses, u_i^0, can be initialized with zero vectors.

Suppose we have discretized the boundary value problem using some method (FVM, FEM, FDM or RBF), in such a way that the linear system for the domain Ω_i is written as $A_i u_i = f_i$. The discrete vector asociated with u_i is defined as $u_i^T = (u_{\Omega_i}, u_{\partial\Omega_i\backslash\Gamma_i}, u_{\Gamma_i})$. In the same way, the matrix of the system is $A_i = (A_{\Omega_i}, A_{\partial\Omega_i\backslash\Gamma_i}, A_{\Gamma_i})$. The discrete form of $u_i^n|_{\Gamma_j}$ is written as V_j^n. The basic alternating Schwarz algorithm is:

Alternating Schwarz algorithm

.

$V_1^0 \leftarrow 0$.
For $n = 1, ...$
  Solve for u_1^n:
    $A_1 u_1^n = f_1$ in Ω_1
    $u^n_{\partial\Omega_1\backslash\Gamma_1} = g_1$ on $\partial\Omega_1\backslash\Gamma_1$
    $u^n_{\Gamma_1} = V_1^{n-1}$ on Γ_1
  $V_2^n \leftarrow u_1^n|_{\Gamma_2}$
  Solve for u_2^n:
    $A_2 u_2^n = f_2$ in Ω_2
    $u^n_{\partial\Omega_2\backslash\Gamma_2} = g_2$ on $\partial\Omega_2\backslash\Gamma_2$
    $u^n_{\Gamma_2} = V_2^n$ on Γ_2
  $V_1^n \leftarrow u_2^n|_{\Gamma_1}$
  Check convergency:
    If $||V_1^n - V_1^{n-1}|| \leq \text{tol}_{\Gamma_1}$ y $||V_2^n - V_2^{n-1}|| \leq \text{tol}_{\Gamma_2}$ END.
    If $||u_1^n - u_1^{n-1}|| \leq \text{tol}_{\Omega_1}$ y $||u_2^n - u_2^{n-1}|| \leq \text{tol}_{\Omega_2}$ END.
End For

Note that the previous algorithm is serial. A parallel version for K subdomains will be presented in section 4.7.

3 OOPS: Object-Oriented Programming for Scientific Computing

Scientific computing has changed dramatically within the last decades reaching an unprecedented degree of complexity. The vast advances made in the development of new hardware architectures and modern numerical methods provide us with tools to tackle more and more complex problems. As a result, certain scientific problems can be simulated with a very good accuracy and nowadays, scientific computing has become indispensable in many fields, such as engineering, science, and even in social sciences and medicine. The coding of complex numerical algorithms need to be portable and efficient in order to achieve good performance on the majority of available hardware resources. Consequently, the design and development of software is an essential component within scientific computing, and requires an interdisciplinary cooperation of experts from several areas of study.

Sophisticated algorithms, a wide range of large-scale hardware environments, and an increasing demand for system integration and portability have shown that language level abstraction must be increased without loss of performance. Using numerical methods and reference implementations from popular textbooks is often not sufficient for the development of serious scientific software. In a similar way, the use of high-level programming languages contained in tools like Mathlab, can support the initial development of new numerical methods and the rapid implementation of prototypes. However, these packages are not sufficient as High-Performance Computing (HPC) kernels and neither are they intended for this purpose.

The traditional paradigm for writing scientific software is known as structured where the code is separated into subroutines and/or functions that can be called from a main program. The common languages used are Fortran (77 and 90) and C. While tens of thousands of lines of structured Fortran code may be understandable, as algorithms develop in complexity, and programs stretch to hundreds of thousands of lines, more attention will need to be paid to the software development process. In many cases there is relatively little sharing of scientific code (i.e. software reuse), and scientific codes tend to be extremely specialized. Most codes implement a single method; if a different method is required, usually a new program is written. This can be extreme: often a different program will exist for each system studied.

Solutions to these problems are known. They were developed by computer scientists when the business world suffered its software crisis in the 1980s. Techniques such as object-oriented programming and generic programming are now well established, and have proven themselves. They are even starting to penetrate scientific fields.

The Object-Oriented paradigm exhibit powerful features like abstraction, classes, inheritance and polymorphism, that can ease the development, maintainability, extensibility and usability of complex software. In OOP, a program is broken down into largely independent pieces, interacting via well-defined interfaces. Writing a large OO program is similar to writing lots of small programs, and the implementation of each part can be changed without fear of breaking the whole package. Because OO software tends to be better organized, there are more opportunities for code reuse. Instead of writing a completely new program, a new method can be implemented into a class, then objects of these classes interact with other objects to solve a problem using the new method. Extending an OO program often only requires a few lines of existing code to be changed.

Notwithstanding, mostly of scientific libraries are still implemented in non-object-oriented languages. The main reason of this fact is because the abstractions generally result in a runtime overhead that is sometimes hard to avoid. In HPC, C++ used to have a bad reputation due to a bad runtime performance. However, C++ provides generic programming, via the use of *templates* [36], which allows high performance code to be written, without sacrificing expressiveness. Generic programming allows code to be written for generic types; the types used are determined at compile time, allowing the compiler to perform aggressive optimizations. Other techniques, like *Metaprogramming* [37] and *Expression templates* [38], help to get good performance as well. There are presently several examples where C++, OOP and generic programming have been used in the construction of new frameworks for scientific computing applications. Some real and succesful libraries can be found in [39].

3.1 Software developing process

Scientific software is typically written with little attention paid to design. For small programs, this is often the fastest route to results, but as software grows over time, it is difficult for programmers to maintain such codes and to add new features.

Nowadays exist strategies in order to develop software in an incremental and ordered way. The Unified Software Process [24] is an standard used in designing big software systems in fields like finance and data bases. This process provides a guide for the software development and follows mainly 4 phases: Inception, Elaboration, Construction y Transition, which in turn are divided in milestones. These milestones are developed in iterations, and each iteration consists on: 1) Requirements, 2) Analysis, 3) Design, 4) Implementation, 5) Testing and 6) Documentation. In each iteration a new version of the software is released and this version contains new features, but also must be compatible with previous versions. Figure 4 depicts the Unified Process Model.

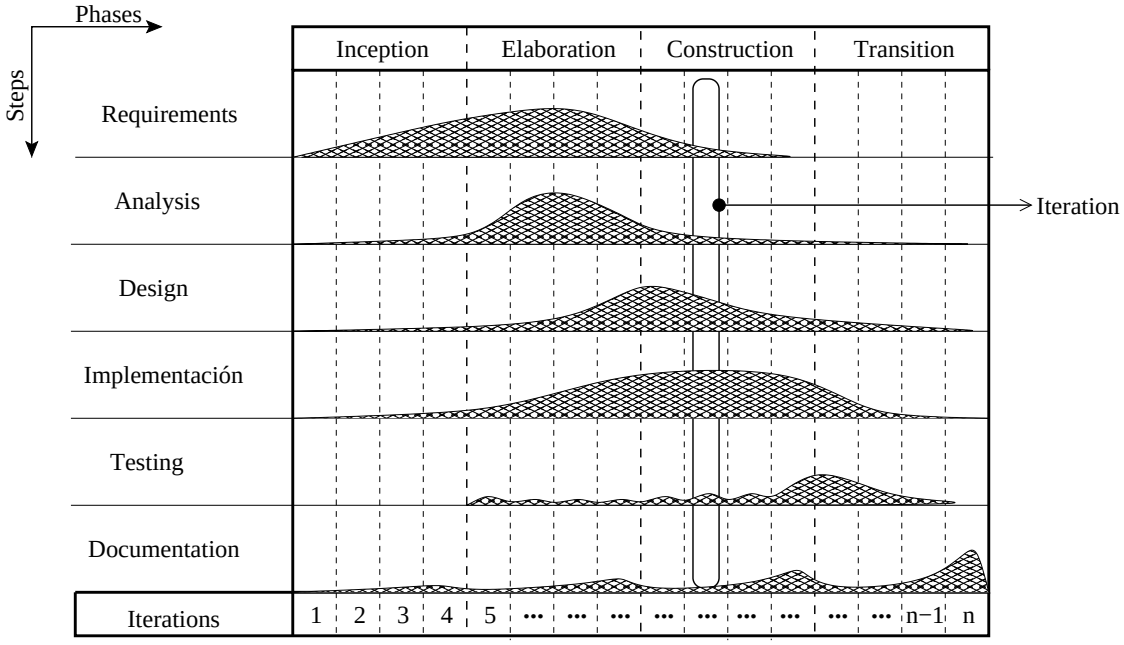


Figure 4: Software developing process (Unified Model).

Some times is not easy to apply the Unified Model to scientific applications. In the figure 5 an adaptation of the Unified Model is presented. The main idea is to include the principal steps of a mathematical modeling as parts of the software developing. This help us to determine the essential abstractions required to solve a given problem. Also, it is important to define a set of problems as a targets to be solved by our system. Beginning with simple problems, the process starts specifying the governing equations that need to be solved, then we followed the steps specified in figure 5 doing several iterations. After several iterations, we get many components that can be used for solving the target problems and likely these components are useful for other sort of problems.

3.2 Template Units for Numerical Applications : RBF method

TUNA::RBF is a set of templated functions, classes and namespaces constructed in this project. This is a framework for solving PDEs coming from CFD problems using RBF mesh-less methods. C++ language was

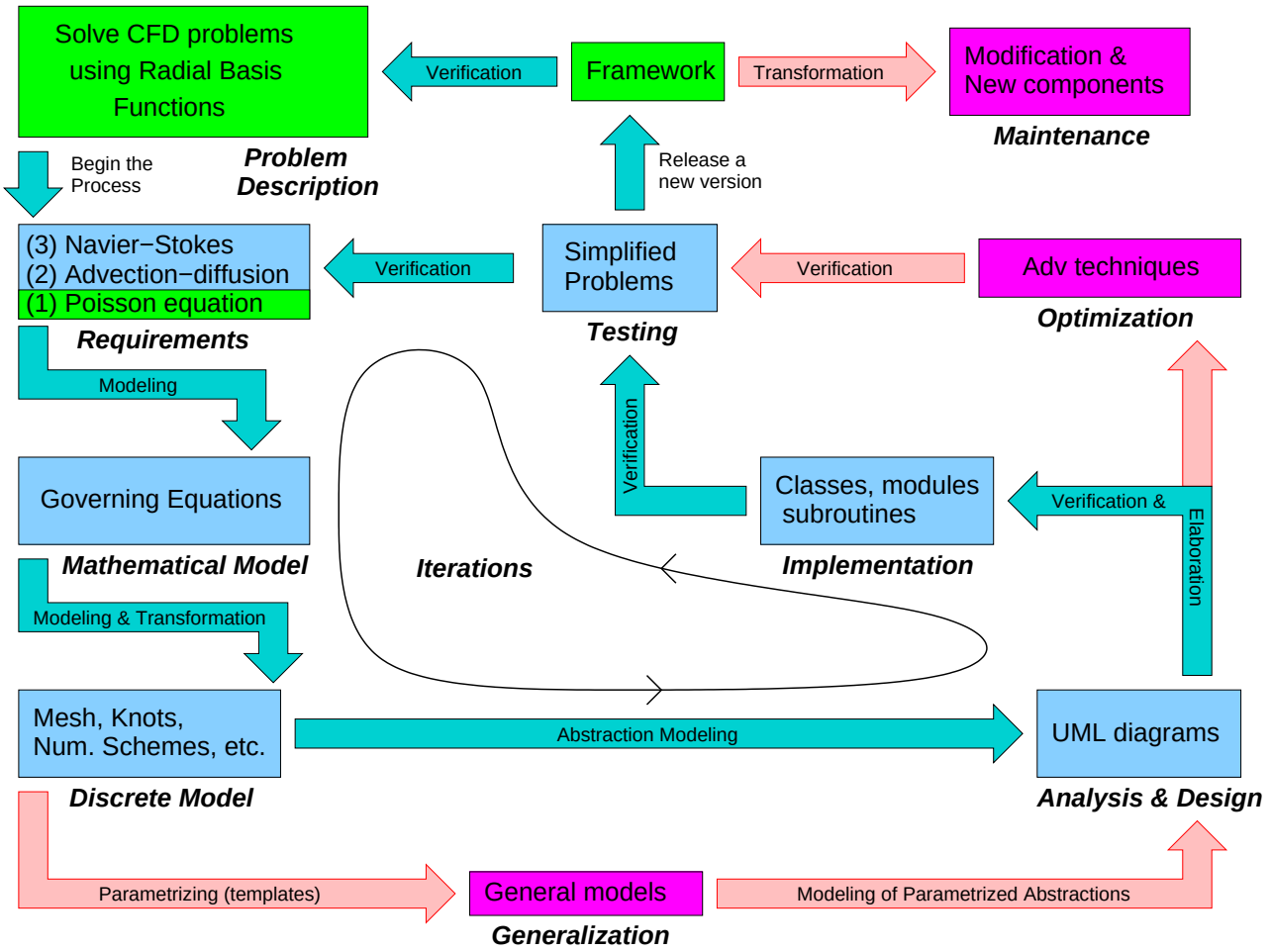


Figure 5: Software developing process for Scientific Computing.

used to implement the components of the frameworks using OOP and generic programming. The software development process showed in figure 5 was applied as well.

TUNA::RBF re-use the next open source libraries for HPC:

- FLENS : Flexible Library for Efficient Numerical Solutions is used for matrix, vector and linear algebra operations. FLENS provide a very expressive and efficient interface that ease the implementation of scientific code. Is based on LAPACK and BLAS. (http://flens.sourceforge.net).
- libKDtree++ for ordering the points and finding the neighborhood of each point in efficient times. (http://libkdtree.alioth.debian.org/).
- MPI-2 for parallel code.

Next section shows results obtained with the application of TUNA::RBF. The components of the software, examples of use and a reference manual can be obtained from http://www.dci.dgsca.unam.mx/lmcs/soft.html.

4 CFD examples

The examples presented in this section are intended to test and calibrate the framework that has been developed during this proyect. These examples have been instensely studied and used to test new numerical methods, and there exist a plenty number of works to compare with. The idea in this work, is to solve the PDEs of these problems using the asymmetric collocation RBF mesh-less method from Kansa [5] and to study the behavior of numerical solutions under different conditions.

For all the examples that follow the Multiquadric RBF kernel for $\beta = 1$ and its derivatives are used to calculate the numerical solution. In the mostly of the cases the shape parameter is set equal to $c = 1/\sqrt{N}$ as is recommended in [19].

4.1 Poisson equation in 2D

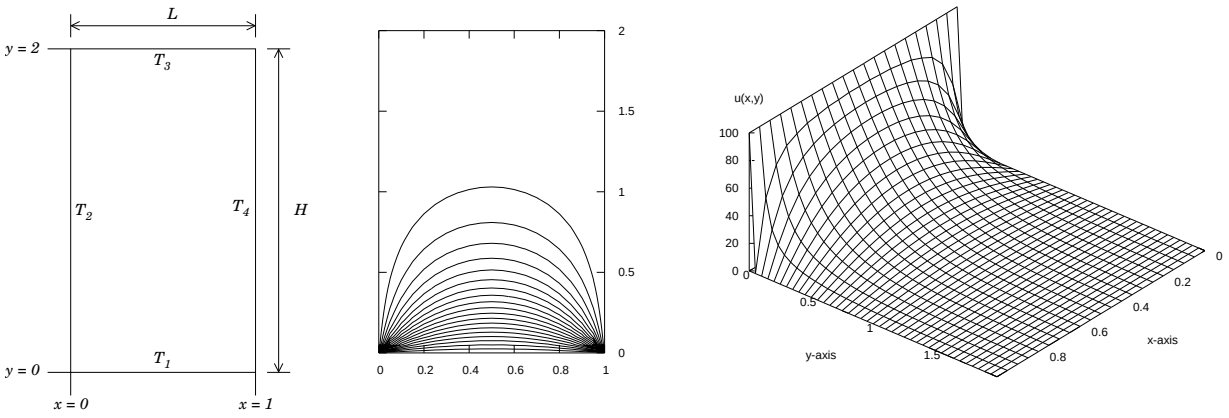
Suppose that it is required to obtain the steady-state temperature distribution on a two-dimensional rectangular plate as shown in figure 6(a). The governing equation and the imposed boundary conditions for this example are:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial x^2} = 0 \tag{24}$$

$$\begin{array}{cccc} T = T_1 = 100 & \text{for } y = 0, & T = T_2 = 0 & \text{for } x = 0, \\ T = T_3 = 0 & \text{for } y = H, & T = T_4 = 0 & \text{for } x = L \end{array}$$

and the analytical solution is, see [40]:

$$T = T_1 \left[2 \sum_{n=1}^{\infty} \frac{1 - (-1)^n}{n\pi} \frac{\sinh\left(\frac{n\pi(H-y)}{L}\right)}{\sinh\left(\frac{n\pi H}{L}\right)} \sin\left(\frac{n\pi x}{L}\right) \right] \tag{25}$$



(a) (b) (c)

Figure 6: (a) Rectangular plate subject to constant temperature distribution at the boundaries. (b) Contours and (c) surface of the exact solution, equation (25).

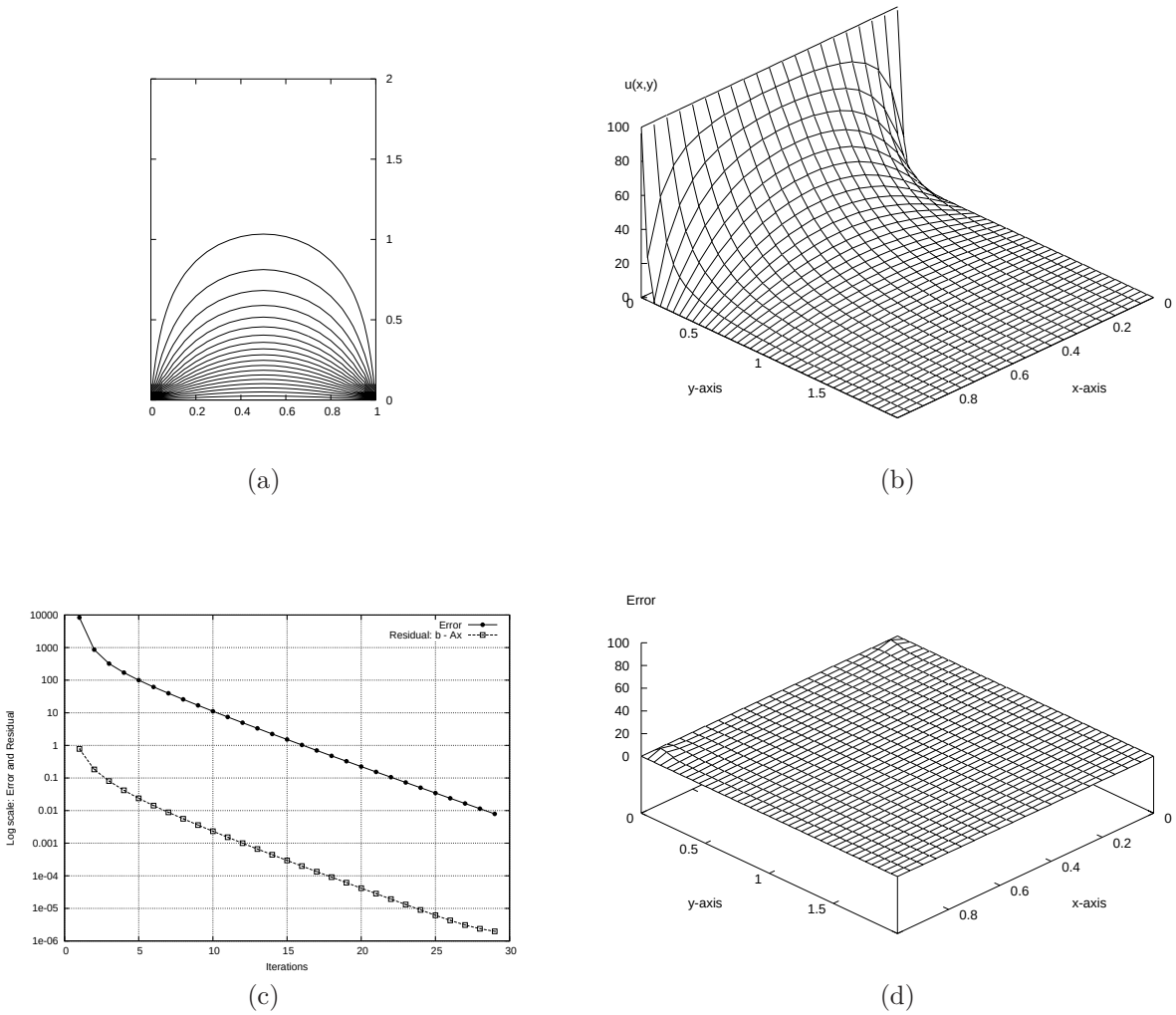
The problem above described was first solved using the Finite Volume (FV) method, see [41, 42]. To this end, the TUNA::FV library [43] was used. This is an Object-Oriented and Generic library, written in C++ using *templates* to obtain good performance. Whenever possible, this tool will be used to obtain numerical solution of the examples presented in this report for comparing purposes.

The same values for the parameters as in [40] were used here, that is: $Nx = 21$, $Ny = 41$, $ERRORMAX = 0.01$ and the error is defined as:

$$ERROR = \sum_{i,j} |T_{i,j}^{k+1} - T_{i,j}^{k}|, \text{ for } i = 2, \ldots Nx - 1, j = 2, \ldots, Ny - 2 \tag{26}$$

where $T_{i,j}^{k}$ represents the value of the temperature in the point (x_i, y_j) at iteration k.

The linear systems coming from the FV method are solved in TUNA::FV using the Thomas algorithm for tridiagonal systems, extended to 5 diagonals for problems in 2D, and 7 diagonals for 3D, see Malalasekera[42]. The main idea is to sweep line by line in along the axis, solving tridiagonal systems for each line and updating the variables on each point on the line swept. In this example two sweep by iteration are done, one in x direction and one in y direction. The number of iterations to get the prescribed error were 29; the RMS error and the maximum error were 0.2607, and 3.0470 respectively. The figure 7 shows the result of this calculation. The table 5 compares these results with those obtained using RBFs.



(a) (b)

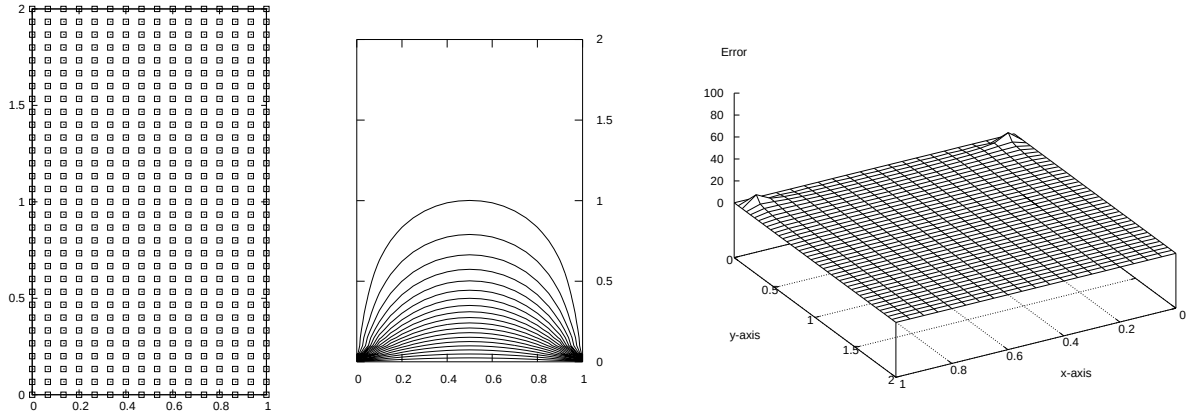
(c) (d)

Figure 7: Solution of the equation (24) usin FV method. (a) Contours and (b) surface. (c) Error and residual ($r = |b - Ax|$), and (d) error distribution ($|T_{exact} - T_{num}|$).

4.1.1 MQ-RBF: Uniform distribution

In this section the MQ-RBF solution of the previously described problem is obtained. A set of collocation points are distributed as follows: N_I random or uniform points in the interior of the domain $\Omega = (0, L) \times (0, H)$ and N_B equispaced points on the boundary $\partial\Omega = \partial\{[0, L] \times [0, H]\}$. Following the methodology of section 2.3, without adding any polynomial, the equation (6) is substituted into the equation (24) to obtain a linear system similar to that represented in equation (10). In this case $\mathcal{L} = \nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ and $\mathcal{B} = I$. For this example $f = 0$ and the boundary conditions requires g be equal to zero on the boundary except on the points that lie on the bottom wall, where the value of g is equal to 100.

The first test was with a uniform distribution of collocation points. The total is $N = N_x \times N_y = 16 \times 31 = 496$, where N_x is the number of points in the x-axis, N_y is the number of points in the y-axis, see figure 8(a).



(a) (b) (c)

Figure 8: (a) Uniform points distribution. (b) Contours and (c) error distribution.

The solution depicted in figure 8(b) shows good agreement with the analytical solution, however the numerical error, figure 8(c) is greater than for the case of FV. In this case the RMS error was 1.0905 and the maximum error was 9.7053. The problem was solved using Gauss elimination, GMRES and GMRES preconditioned. A comparison of these algorithms is shown in table 2.

Algorithm	Precond.	Iter	GT / T
Gauss	-	-	1.00
GMRES	-	240	1.69
GMRES	Jacobi	108	4.40
GMRES	ACBF(10)	25	16.5

Table 2: Performance for the different algorithms.
GT / T represent the rate of speedup of the algorithm with respect to the Gauss elimination.

As can be seen from the table 2, the GMRES performs well compared against Gauss elimination. Also, it is convenient to use a preconditioner to reduce the condition number of the original matrix. In this case the simple Jacobi preconditioner make very good job reducing the number of iterations to less than a half of that required by the no preconditioned GMRES.

The ABCF preconditioner is from Kansa [19] and is based upon constructing the least-squares Approximate Cardinal Basis Functions that targets ill-conditioned problems, see section 2.4.2. The consuming time of this algorithm is proporcional to m. In table 2, $m = 10$ is used and the performance was measured only for the

GMRES iteration without taking in to account the time for constructing the preconditioner. In the table 3 the time to calculate the preconditioner in function of m is reported.

m	GMRES Iter.	GMRES Time	ACBF Precond. KDT	FN	LS	Total
8	28	0.05	0.01	0.17	0.42	0.65
10	25	0.04	0.01	0.21	0.39	0.65
20	17	0.03	0.01	0.24	0.70	0.98
30	15	0.03	0.01	0.26	1.14	1.44
50	13	0.03	0.01	0.39	2.10	2.53
100	10	0.03	0.01	0.46	5.77	6.27

Table 3: CPU time to construct the ACBF preconditioner. All the times are in seconds.
KDT : KD-Tree construction; FN : Finding the neighbors to all the collocation points; LS : Least-square problem solution.

From table 3 it is obvious that the consuming time for the ACBF preconditioner is not economic for large m. The total time presented in the last column would be the total time for solving the linear system including the construction of the preconditioner. In this case the best value of m in terms of calculation time and number of GMRES iterations is 10.

Even though the consuming time for constructing the ACBF preconditioner is expensive for this example, in the case of time-dependent problems it could be very valuable to use this preconditioner, because for fixed collocation points the preconditioner just needs to be calculated once at the beginning, and for every time step the number of GMRES iterations will be reduced drastically compared with the Jacobi preconditioner. Besides, here the L-NE technique was used to solve the N- least-square problems required in the construction of the preconditioner, but it is possible to use other optimized algorithms that can reduce the consuming times, see [19] for more details.

4.1.2 MQ-RBF: Random distribution

One of the advantages of RBFs is the possibility of using a random distribution of collocation points to approximate the solution of a PDE. The examples of this section are intended to illustrate the dependence of the solution with respect to the point distribution.

In figure 9 results for two different distributions are shown. Table 4 shows the comparison between the two random distribution. In both examples $m = 10$ and $c = 0.04$.

Dist.	GMRES Iter.	GMRES Time (secs)	ACBF Precond. (secs) KDT	FN	LS	Total	RMS	Max. Error
1	48	0.06	0.01	0.05	0.51	0.63	2.79798	18.9085
2	32	0.04	0.01	0.21	0.33	0.59	0.39212	2.78903

Table 4: Results for random distribution.
1 is the distribution shown in figure 9(a) and 2 is that shown in figure 9(d).

As can be seen from figure 9 and table 4 the error growth for a badly distribution of points. In order to get a better precision it is necessary that the points cover uniformly all the domain and there are not empty zones or cluster of points. If the points have a good distribution, especially near the boundaries, or where the strong gradients are located, the precision can be better than that of a regular distribution or even than that from the Finite Volume Method, see table 5 for a global comparison.

At this stage, it is important to notice the need of controlling random distribution in order to avoid bad results. The distribution of figure 9(d) was controlled allowing a point be randomly located in a limited region

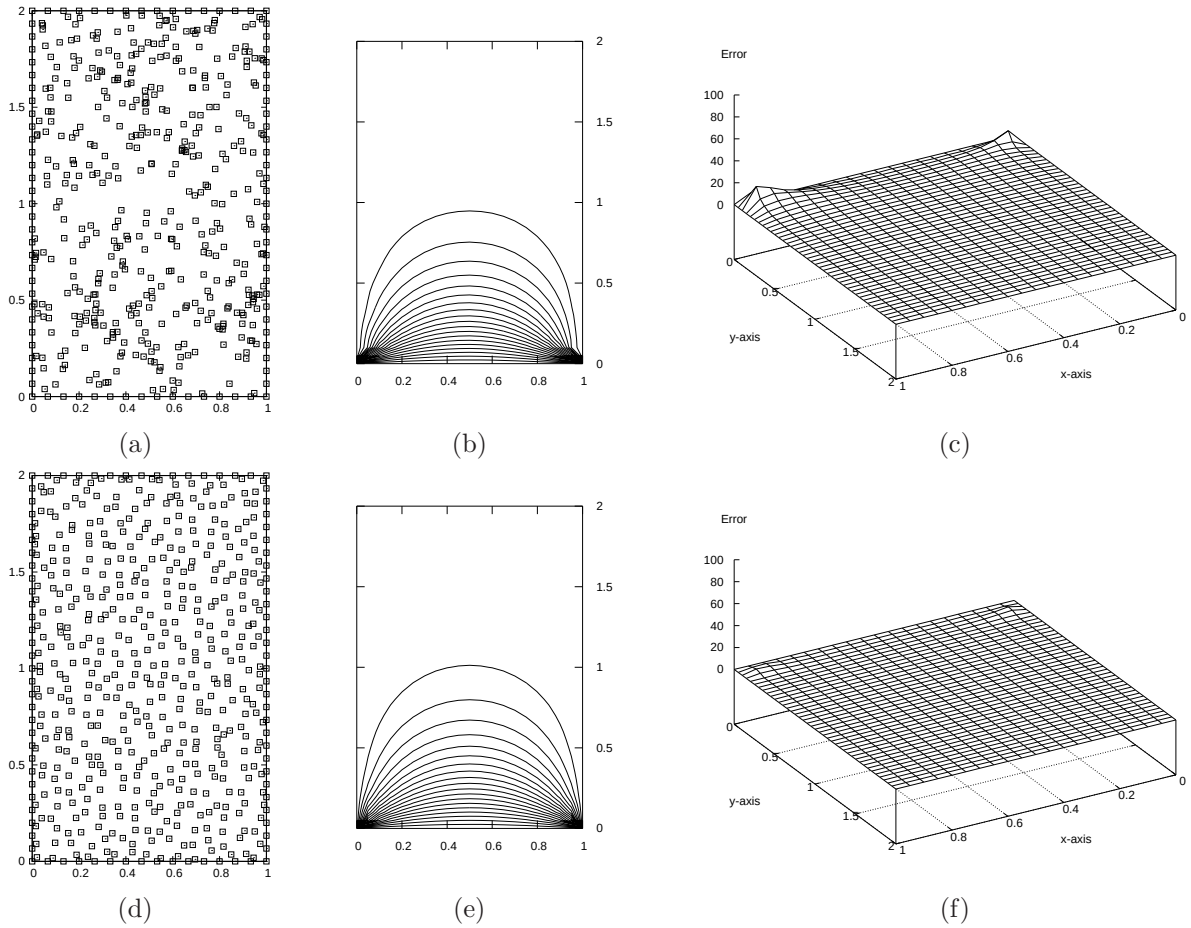


Figure 9: Random distribution of collocation points for the heat transfer example.
Distribution 1 : (a), (b) and (c), results for a badly distribution. Distribution 2 : (d), (e) and (f), better global random distribution.

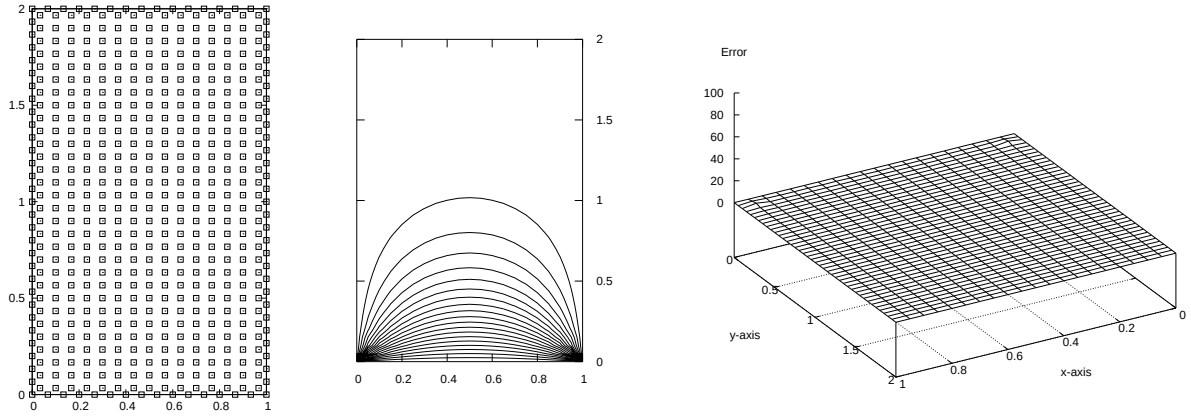
of the domain, but not on the whole domain. In this case, every point is allowed to move inside cells defined by a regular mesh.

4.1.3 MQ-RBF: Points near the boundary

As was noted in the previous section, the distribution of points for a constant shape parameter is very important, especially near the strong gradients. In the problem studied here, the strongest gradient is located near the bottom boundary.

The distribution shown in figure 10 contains a set of points near to the boundary. This distribution produce very good results. Similar point distribution was used in [22] resulting in an improved accurary of the numerical results. In this case the RMS error was 0.250542 and the Maximun error was 1.74965. This result is better than that obtained with FVM. The table 5 shows a global comparison.

This is a very simple example, however the discontinuity on the boundary conditions makes difficult to get a good numerical solution. In the case of FV, the solution is obtained very fast in comparison with RBF, but more points are needed to get good pecision. In particular, is not possible to get better results with a finer mesh, especially near the bottom boundary. On the other hand, the cost of getting a good solution with RBF is



(a) (b) (c)

Figure 10: Best distribution collocation points for $c = 0.04$.

Case	Points	Iter	RMS	Max	T / FVT
FV	861	29	0.2607	3.0470	1.0
RBF 1	496	25	1.0950	9.7053	4.0
RBF 2	496	48	2.7979	18.9085	6.0
RBF 3	540	32	0.3921	2.7890	4.0
RBF 4	540	25	0.2505	1.74965	3.0

Table 5: Global comparison for the heat transfer problem.

FVT / T is the ratio between the Finite Volume consuming time and the consuming time of the RBF method for different distributions. FV: Finite Volume solution; RBF1: uniform distribution, figure 8(a); RBF2: badly random distribution, figure 9(a); RBF3: good random distribution, figure 9(d); RBF4: Best regular distribution, figure 10(a).

expensive compared with FV. However the best solution is only 3 times slower than FV. Besides the precision is better and the implementation of RBF methods is simple: the same functions can be used for 2D and 3D problems. The powerful of RBF methodology should be clear when dealing with time-dependent problems and in complicated geometries.

4.1.4 Shape parameter

The MQ-RBF incorporate a user-defined shape parameter c. The value of this scalar determines the region of influence of the MQ-RBF kernel. Many numerical studies using MQ-RBF have shown that this kernel gives better performance than others, and it has been observed that the accuracy of the numerical solutions depends heavily on the value of c. An optimal value of this parameter is not easy to find and is still an open problem in the literature. The influence of the shape parameter on the accuracy of the solution follows a U-shape curve. The figure 12 depicts the behaviour of the error in the case of the problem studied in the previous sections. As can be seen, a short value of c produce bad accuracy, a minimum of the error is reached when $c \approx 1/\sqrt{N}$, and after this value the accuracy become worse as c is incremented. The value $c = 1/\sqrt{N}$ has been used in several studies with good results, see for example [19].

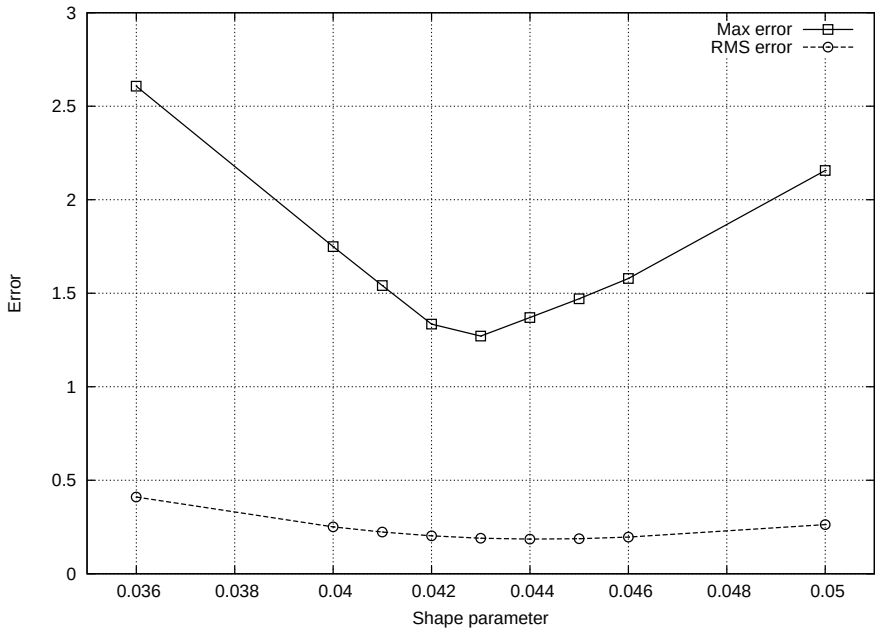


Figure 11: U-behaviour of shape parameter.

4.2 Laplace equation in a Semi-circle

One advantage of mesh-less methods is the possibility of solving problems in complex geometries. As a test, the solution of Laplace equation in the semi-circle shown in figure 12(a) is obtained using MQ-RBF. In terms of code developing, it is only needed to implement a function or a class to distribute the points on the irregular domain and to reuse the classes developed before. The analytical solution to this problem is:

$$T(r,\theta) = \sin(\theta)/r \tag{27}$$

where θ and r represent the angle and the radius in polar coordinates. The boundary conditions for this problem are, see figure 12(a).

$$\begin{array}{lllll} T = \sin(\theta)/R_{0A} & \text{for segment AC;} & T = \sin(\theta)/R_{0B} & \text{for segment BD;} \\ T = 0 & \text{for segment AB;} & T = 1/r & \text{for segment CD,} \end{array}$$

where R_{0A} and R_{0B} are the radii from point 0 to A and from point 0 to B respectively. The analytical solution is depicted in figure 12(b).

Because this problem is very simple, and there is not high gradients present, it is possible to get a very good precision in the numerical solution using a random distribution of points. In this case the solution was obtained using $N_I = 126$ and $N_B = 46$, that is $N = 172$ points, see figure 12(c). The GMRES-ACBF with $m = 30$ was used to solve the system, and a shape parameter $c = 1/\sqrt{N}$. The RMS error with respect the analytical solution was of 0.1 % and the maximum was of 0.27 %. Figure 12(d) shows the error distribution on the semicircle domain.

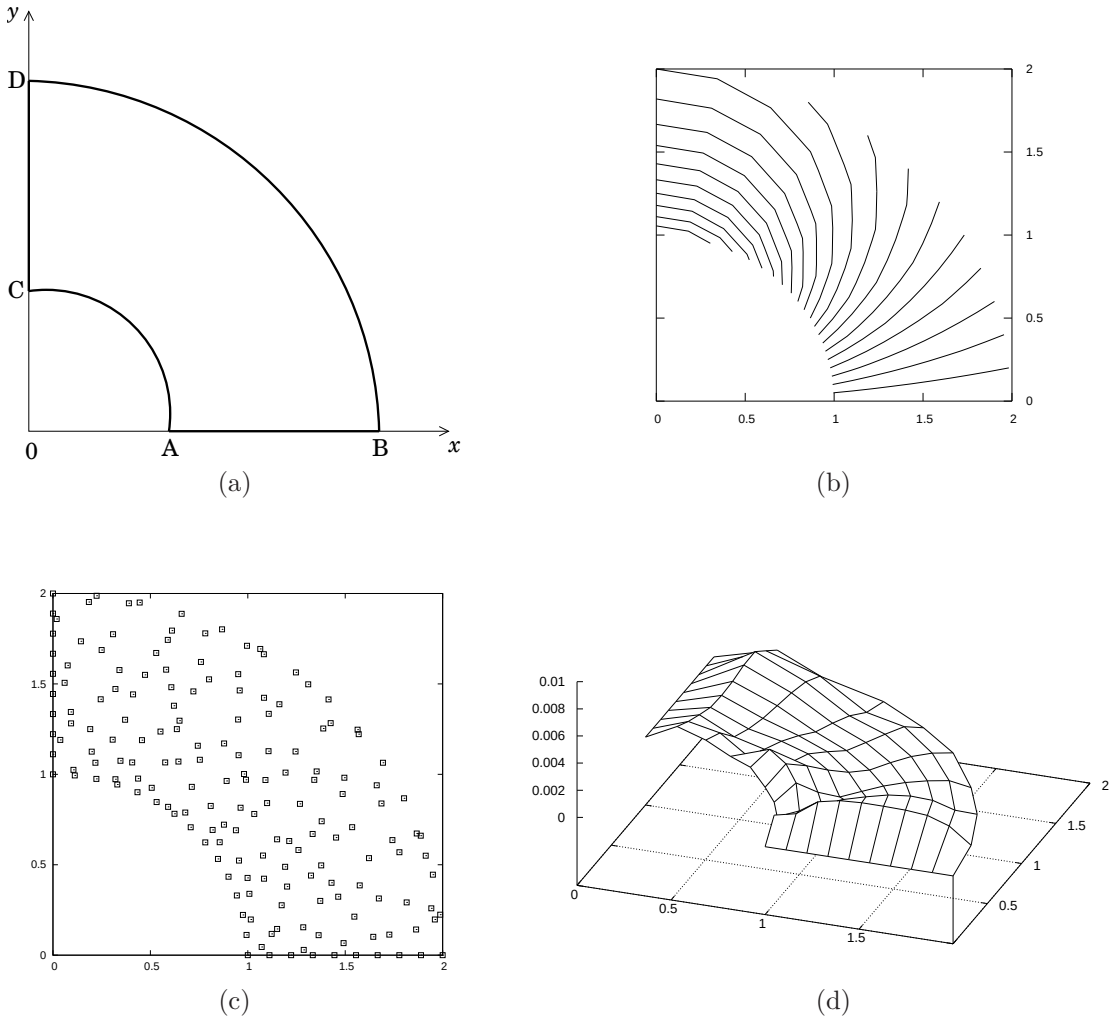


Figure 12: (a) Domain of study, (b) analytical solution, (c) Point and (d) error distribution.

4.3 Linear time-dependent advection-diffusion in 1D

The time-dependent advection-diffusion equation in one dimension is written as follows:

$$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} = \Gamma\frac{\partial^2 f}{\partial x^2} \tag{28}$$

where x and t are the space and time coordinates respectively, Γ is the diffusion coefficient, u is a constant input velocity. The initial and boundary conditions for this example are

$$f(x,0) = 0, \text{ for } 0 \leq x \leq \infty, \tag{29}$$
$$f(0,t) = 1, \text{ for } t > 0, \tag{30}$$
$$f(L,t) = 0, \text{ for } t > 0, L \to \infty. \tag{31}$$

The exact solution of this moving front problem for $\Gamma > 0$ is

$$f(x,t) = 0.5\left[\text{erfc}\left(\frac{x-ut}{2\sqrt{\Gamma t}}\right) + \exp\left(\frac{ux}{D}\right)\text{erfc}\left(\frac{x+ut}{2\sqrt{\Gamma t}}\right)\right] \tag{32}$$

where erfc is the complementary error function.

To construct the solution in terms of RBF, the MQ kernel and its derivatives are assumed to be fixed and the coefficients vary in time, that is $\lambda_j = \lambda_j(t)$. Let $f^k \approx f(x, k\Delta t)$. A backward Euler difference scheme is used to approximate the time derivative in equation (28):

$$\frac{\partial f(x,t)}{\partial t} \approx \frac{f^{k+1} - f^k}{\Delta t} \tag{33}$$

then, using (33) in (28) results in the next sequence:

$$f^{k+1} + \Delta t\left[u\frac{\partial f^{k+1}}{\partial x} - \Gamma\frac{\partial^2 f^{k+1}}{\partial x^2}\right] = f^k$$

Now, expanding the left-hand side of this equation using MQ-RBF the next linear system is obtained

$$\sum_{j=1}^{N}\lambda_j^{k+1}\phi_j(r_i) + \Delta t\left[u\sum_{j=1}^{N}\lambda_j^{k+1}\frac{\partial\phi_j(r_i)}{\partial x} - \Gamma\sum_{j=1}^{N}\lambda_j^{k+1}\frac{\partial^2\phi_j(r_i)}{\partial x^2}\right] = f^k, \text{ for } 1 \leq i \leq N_I, k = 0,1,\ldots \tag{34}$$

where λ_j^{k+1} represents the value of the coefficient λ_j at $t = (k+1)\Delta t$. Note that the solution at $t = k\Delta t$ is used as the initial guess for the matrix system defined at $t = (k+1)\Delta t$. For the first step, the right-hand vector is given by the initial conditions (29). The system (34) is completed with the expansion of boundary conditions, equations (30) and (31) in terms of MQ-RBF, yielding an $N \times N$ linear system.

The parameters used in this example were $u = 1.0$, $\Delta t = 0.002$, $t_{final} = 1.0$, which implies 500 time steps, and $D = 0.001$. The numerical solution was calculated using Finite Volume and MQ-RBF. In the case of FV the Upwind, Central and Quick schemes were applied to approximate de advective terms, and a Central scheme for the diffusive term. The table 6 shows the results for 50, 100, 200, 250 and 350 points. In figure 13(a) the solution obtained for 50 points using FV is depicted. Figure 13(b) shows the best result using FV, that is FV-Central with 350 points. For MQ-RBF approach, the system was solved using GMRES with the ACBF preconditioner and the shape parameter was defined as $c = 1/\sqrt{N}$. Observe from table 6 that the error obtained with MQ-RBF is small even with 50 points and there is not big improvement adding more points. The figures 13(c) and (d) show the results obtained using 50 and 350 points. There is not a perceptible difference between these two figures.

Points	RMS error			
	FV–Upwind	FV–Quick	FV–Central	MQ–RBF
50	0.1143	0.0480	0.0712	0.0228
100	0.0877	0.0346	0.0383	0.0193
200	0.0658	0.0398	0.0263	0.0188
250	0.0598	0.0451	0.0220	0.0188
350	0.0517	0.0572	0.0193	0.0188

Table 6: RMS error for FV and RBF.

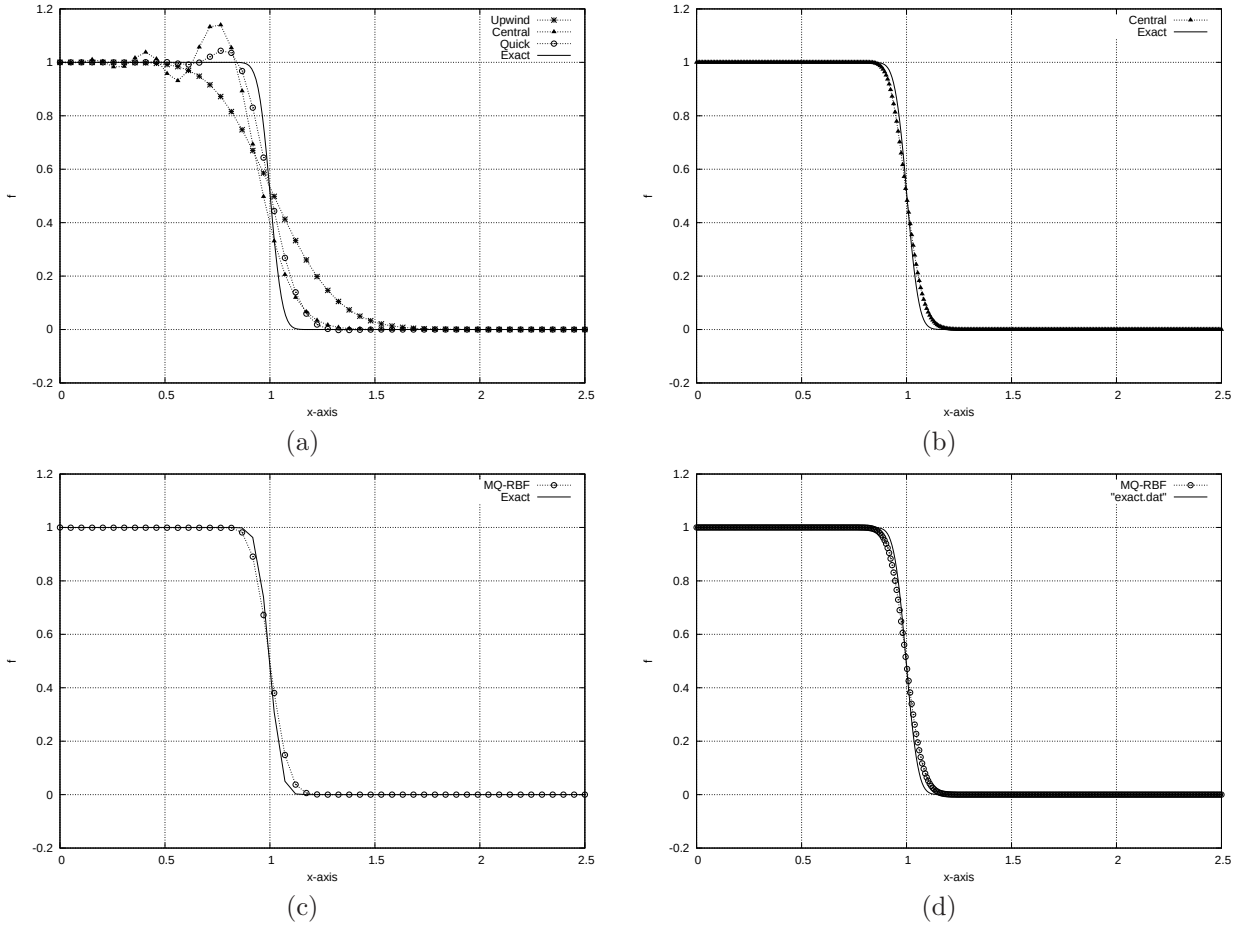


Figure 13: Solution of the advection-diffusion in 1D. (a) Upwind, Quick and Central (50 points); (b) Central (350 points); MQ-RBF for (a) 50 and (d) 350 points respectively.

From table 6 it is clear that the RMS error for FV-Central with 350 points and MQ-RBF with 100 points is the same until 4 digits. In these cases the total CPU times were of 5.56 and 5.76 seconds respectively. The ACBF preconditioner was constructed with $m = 25$ and no special points The average number of GMRES iterations was of 18. However, when two special points are added (at the two ends of the domain), the average number of iterations is down to 8 and the CPU time is reduced to 3.82 seconds.

4.4 Forced Convection in 2D

In this example the time-dependent convection-diffusion equation in two-dimesnions is solved. The equation is written as follows:

$$\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y} = \Gamma\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) \tag{35}$$

where (u, v) is a prescribed velocity field that fulfills the continuity equation and is given by the next formula:

$$\begin{aligned} u(x,y) &= -A\cos(\pi y)\sin(\pi\lambda x/l_x)\text{and} \\ v(x,y) &= \frac{A\lambda}{l_x}\sin(\pi y)\cos(\pi\lambda x/l_x). \end{aligned}$$

The initial condition is $T = 0$ and the boundary conditions are as shown in the next figure:

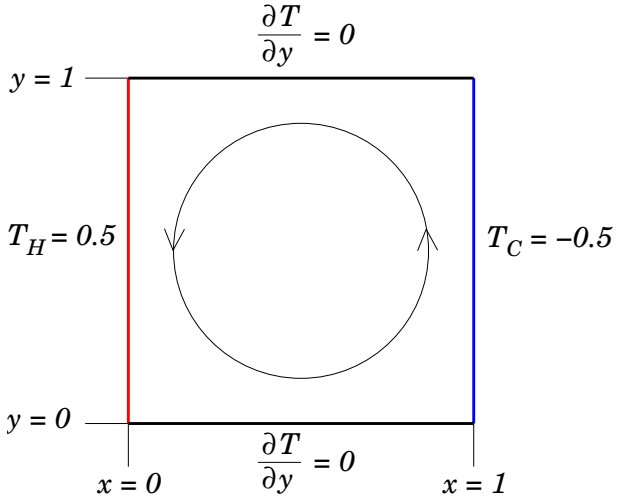


Figure 14: Geometry and boundary conditions

The boundary conditions of this problem includes a couple of Neumann conditions in the top and bottom walls. In terms of RBFs these conditions are written as follows

$$\begin{aligned} \frac{\partial T}{\partial y}\Big|_{y=0} &\approx \sum_{j=1}^{N}\lambda_j\frac{\partial\phi_j(r_i)}{\partial y}\Big|_{y=0} = 0, \text{ for } N_I+1 \le i \le N \\ \frac{\partial T}{\partial y}\Big|_{y=1} &\approx \sum_{j=1}^{N}\lambda_j\frac{\partial\phi_j(r_i)}{\partial y}\Big|_{y=1} = 0, \text{ for } N_I+1 \le i \le N \\ T\big|_{x=0} &\approx \sum_{j=1}^{N}\lambda_j\phi_j(r_i)\big|_{x=0} = 0.5, \text{ for } N_I+1 \le i \le N \\ T\big|_{x=1} &\approx \sum_{j=1}^{N}\lambda_j\phi_j(r_i)\big|_{x=1} = -0.5, \text{ for } N_I+1 \le i \le N \end{aligned} \tag{36}$$

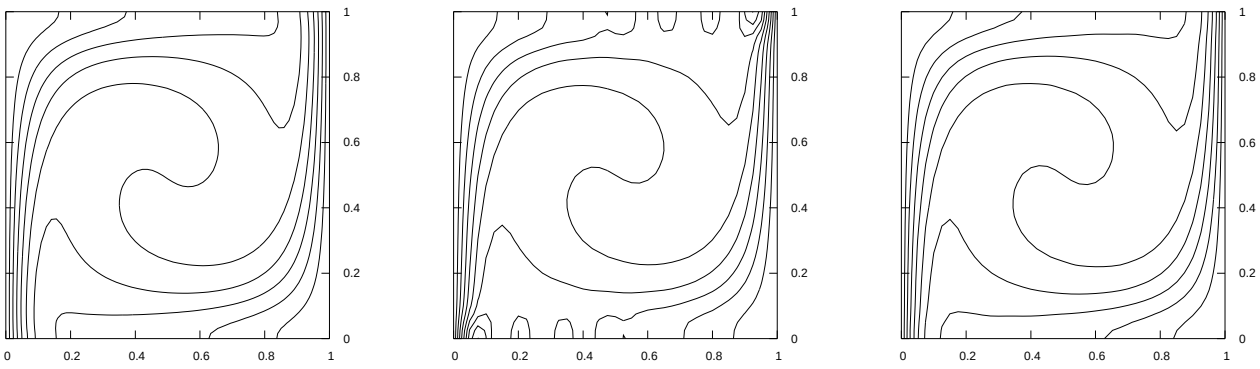
The equations (36) are incorporated in the global linear system as was described in section 2.3. The same formulation for the temporal derivatives as in section 4.3 was used here. The final linear system is written as follows

$$\begin{bmatrix} WL_{11} & WL_{12} \\ WB_{21} & WB_{22} \end{bmatrix}^k \Lambda = \vec{T}^k \tag{37}$$

where the indices k and $k+1$ represents values at $t = k\Delta t$ and $t = (k+1)\Delta t$. The sub-matrices WL_{11}, WL_{12}, WB_{21} and WB_{22} are as defined in section 2.3 and the operators $\mathcal{L}$ and $\mathcal{B}$ are:

$$\begin{aligned} \mathcal{L} &= I + \Delta t \left[u\frac{\partial}{\partial x} + v\frac{\partial}{\partial y} - \Gamma\left(\frac{\partial^2}{\partial^2 x} + \frac{\partial^2}{\partial^2 y}\right)\right] \\ \mathcal{B} &= I, \text{ for } x = 0, 1 \\ \mathcal{B} &= \frac{\partial}{\partial y}, \text{ for } y = 0, 1 \end{aligned} \tag{38}$$

The results presented in figure 15 show the solution of equation (35) using FV-QUICK scheme in a mesh of $65 \times 65 = 4225$, and the solution obtained using MQ-RBF with $c = 1/\sqrt{N}$ and $c = 0.25$ for $16 \times 16 = 256$ uniformly distributed set of points. In this example $\Delta t = 1e-04$, $A = 100$, and $m = 50$ for the ACBF preconditioner.



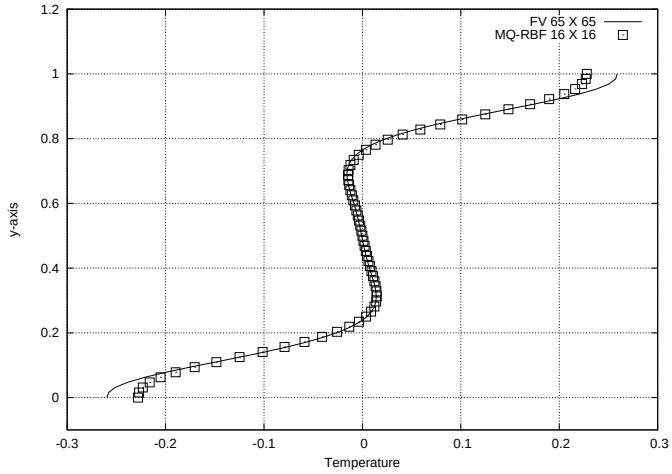
(a) (b) (c)

Figure 15: Solution of the convection-diffusion in 2D.

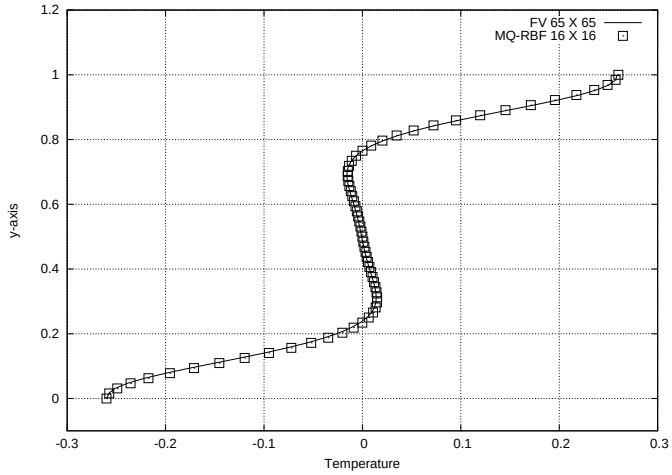
(a) FV-QUICK in a mesh of 65×65; (b) MQ-RBF for 16×16 points and $c = 1/\sqrt{N} = 0.0625$. (c) MQ-RBF for 16×16 points and $c = 0.25$.

It can be observed in figure 15(b) that the solution using MQ-RBF presents small oscillations near the boundaries were the Neumann boundary conditions are imposed. This is a well known problem in RBF techniques: the approximation of the derivatives is less accurate than the function itself. In this case, the approximation of the normal derivatives from Neumann conditions by the MQ-RBF kernel is not good. Some authors have proposed solutions to this problems, see for example [22]. In figure 15(c) a better approximation is shown. That solution was obtained increasing the shape parameter to $c = 0.25$. However the computational effort was also increased, see the comparison in table 7 Figures 16 (a) and (b) present the temperature profiles

across the center of the square along y-axis. In this figure it is possible to observe the departing of the MQ-RBF solution with respect to the FV solution, near the Neumann boundary condition for a small shape parameter.



(a)



(b)

Figure 16: Temperature profiles for the line $x = 0.5$.

(a) MQ-RBF for 16×16 points and $c = 1/\sqrt{N} = 0.0625$. (b) MQ-RBF for 16×16 points and $c = 0.25$.

Method	N	c	GMRES avrg	Iter	Error	CPU Time
FVM	65×65	–	–	6111	1e-06	15.55
MQ-RBF	16×16	0.0625	10	1633	1e-06	14.89
MQ-RBF	16×16	0.25	78	1545	1e-06	98.00

Table 7: Comparison of the results for FV and MQ-RBF methods. Times are in seconds.

4.5 Lid-driven cavity

The objetive in this section is to solve the well known lid-driven cavity using MQ-RBF. This is one of the most studied fluid problem in CFD, because the simplicity of the cavity geometry allows us to apply almost any numerical method on this flow problem. Despite its simple geometry, the driven cavity flow retains a rich fluid flow physics manifested by multiple counter rotating recirculating regions on the corners of the cavity depending on the Reynolds number. The geometry and boundary conditions are shown in the next figure.

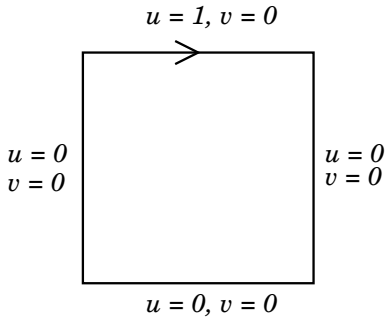


Figure 17: Lid-Driven Cavity geometry.

In this examples the streamfunction-vorticity formulation is used. The equations are written as follows:

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial x^2} = -w \tag{39}$$

$$u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} = \frac{1}{Re}\left(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial x^2}\right) \tag{40}$$

where the realtionship between velocity, streamfunction and vorticity are defined as:

$$u = \frac{\partial \psi}{\partial y} \quad v = -\frac{\partial \psi}{\partial x} \tag{41}$$

$$w = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \tag{42}$$

In this formulation the boundary conditions for the streamfunction are:

$$\psi = 0, \frac{\partial \psi}{\partial x} = 0 \quad \text{on} \quad x = 0, 1 \tag{43}$$

$$\psi = 0, \frac{\partial \psi}{\partial y} = 0 \quad \text{on} \quad y = 0 \tag{44}$$

$$\psi = 0, \frac{\partial \psi}{\partial y} = 1 \quad \text{on} \quad y = 1 \tag{45}$$

For the case of the vorticity there are several manners to impose the boundary conditions, see [44]. The calculation of second derivatives of the streamfunction is required to impose these boundary conditions. In order to avoid to deal with RBF derivatives for the boundary conditions, a finite difference nine point stencil is used to calculate those derivatives. The formula for the upper-left corner is written as follows:

$$\frac{1}{3h^2}\begin{bmatrix} \bullet & \bullet & \bullet \\ \bullet & -2 & \frac{1}{2} \\ \bullet & \frac{1}{2} & 1 \end{bmatrix}\psi^h + \frac{1}{9}\begin{bmatrix} \bullet & \bullet & \bullet \\ \bullet & 1 & \frac{1}{2} \\ \bullet & \frac{1}{2} & \frac{1}{4} \end{bmatrix} w^h = -\frac{1}{2h} \tag{46}$$

and for the top wall the formula is:

$$\frac{1}{3h^2}\begin{bmatrix} \bullet & \bullet & \bullet \\ \frac{1}{2} & -4 & \frac{1}{2} \\ 1 & 1 & 1 \end{bmatrix}\psi^h + \frac{1}{9}\begin{bmatrix} \bullet & \bullet & \bullet \\ \frac{1}{2} & 2 & \frac{1}{2} \\ \frac{1}{4} & 1 & \frac{1}{4} \end{bmatrix} w^h = -\frac{1}{h} \tag{47}$$

In the above formula a •means a point on the boundary. For other corners and walls the formulae are similar.

As we can observe, the equations (39) and (40) are strongly coupled by the convective terms. In order to deal with this non-linear coupling the next pseudo-temporal formulation is used:

$$\frac{\partial \psi}{\partial \tau} - \left\{ \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} + w \right\} = 0 \tag{48}$$

$$\frac{\partial w}{\partial \tau} - \left\{ \frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} - Re\left(u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} \right) \right\} = 0 \tag{49}$$

Substituing the RBF formulation for w and ψ, as was seen in section 2.3, and applying a backward Euler scheme for the pseudo-temporal derivatives the above equations are converted in the next two linear systems:

$$\sum_{j=1}^{N} [\lambda_\psi]_j^k \left[\phi_{ij}^k - \Delta\tau \left\{ \frac{\partial^2 \phi_{ij}^k}{\partial x^2} + \frac{\partial^2 \phi_{ij}^k}{\partial y^2} \right\} \right] = \phi_{ij}^{k-1} + w^k \Delta\tau \tag{50}$$

$$\sum_{j=1}^{N} [\lambda_w]_j^k \left[\phi_{ij}^k - \Delta\tau \left\{ \frac{\partial^2 \phi_{ij}^k}{\partial x^2} + \frac{\partial^2 \phi_{ij}^k}{\partial y^2} - Re\left(u\frac{\partial \phi_{ij}^k}{\partial x} + v\frac{\partial \phi_{ij}^k}{\partial y} \right) \right\} \right] = \phi_{ij}^{k-1} \tag{51}$$

where $[\lambda_w]_j^k$ and $[\lambda_\psi]_j^k$ are the unknown cofficients for the MQ-RBF formulation of the vorticity and the streamfunction respectively; the superscript k means the value of a variable at the k-th iteration; $\phi_{ij} = \phi_j(r_i)$; $i, j = 1, \ldots N$; $\Delta\tau$ is the pseudo-time step.

The velocity components are calculated via the first derivatives of the streamfunction. These derivatives can be calculated using the RBF kernel as follows:

$$u_i = \frac{\partial \psi_i}{\partial y} = \sum_{j=1}^{N} [\lambda_\psi]_j \frac{\partial \phi_{ij}}{\partial y} \tag{52}$$

$$v_i = -\frac{\partial \psi_i}{\partial x} = -\sum_{j=1}^{N} [\lambda_\psi]_j \frac{\partial \phi_{ij}}{\partial x} \tag{53}$$

The algorithm used to solve the coupled equations of this problem is as follows:

RBf algorithm to solve the Lid-driven cavity problem

```
00    Initialize and guess variables w, ψ, u, v
01    Calculate the streamfunction linear system
02    Calculate the ACBF preconditioner for streamfunction matrix
03    WHILE ( error > tolerance OR iter < Max. iterations)
04        Calculate the vorticity linear system
05        Calculate the Jacobi preconditioner for vorticity matrix
06        Update Boundary Conditions for the vorticity
07        Solve for the vorticity using GMRES
08        Update the source term of streamfunction
09        Solve for the streamfunction using GMRES
10        Evaluate the velocity
11        Check for convergence
12    END WHILE
```

The solution of the lid-driven cavity was carried on using $51 \times 51 = 2601$ points uniformly distributed on the domain, the shape parameter was $c = 1/\sqrt{N} = 0.0196$, and $m = 50$ for the ACBF preconditioner. The results obtained with the implementation of the algorithm above presented are resumed in the table 8 In that table the total number of pseudo-time iterations, average GMRES iterations for solving the streamfunction and vorticity, and the u_{min}, v_{min}, v_{max} velocity values are compared for different Reynolds number. It can be observed that the ACBF preconditioner used for the streamfunction makes very good work in reducing the ill-conditioning of the matrix, in such a way that the average GMRES iterations is maintained constant for all Reynolds numbers. As the Reynolds number is increased the ill-conditioning of the system growths and more iterations are needed to achieve the prescribed tolerance. Finally the values of the u_{min}, v_{min}, v_{max} are in good agreement with the values obtained by Ghia *et al.* [45], where the same problem was solved for a mesh of $129 \times 129 = 16641$ that is roughly 6 times the number of points used in our implementation.

Re	Pseudo-temp iter	Vorticity iter	Stream iter	MQ-RBF			Ghia *et al.* [45]		
				u_{min}	v_{min}	v_{min}	u_{min}	v_{min}	v_{max}
0	1329	244	17	-0.213524	-0.197132	0.197132	–	–	–
100	1477	265	17	-0.226325	-0.273417	0.195655	-0.21090	-0.24533	0.17527
400	2126	331	17	-0.343091	-0.45963	0.317437	-0.32726	-0.44993	0.30203
1000	3351	476	16	-0.385597	-0.515246	0.378291	-0.38289	-0.51550	0.37095

Table 8: Comparison of MQ-RBF results against Ghia *et al.* [45].

Figure 18 presents the distribution of the streamfunction and vorticity for different Reynolds numbers. Observe that for $Re = 1000$ a little oscillations near the top-right corner appear. In fact, was not possible to solve the problem for $Re > 1000$ with the same number of points. One manner to get ride of these oscillations is increasing the shape parameter on the boundary or to use more points.

Finally, velocity profiles where obtained for the cases $Re = 100, 400$ and 1000. Figures 19(a), (c) and (e) depicts the u-velocity across the center of the cavity along the line $x = 0.5$. Figures 19 (b), (d) and (f) depicts the v-velocity across the center of the cavity along the line $y = 0.5$. In these cases it is possible to compare our results with those obtained by Ghia *et al.* [45] and Erturk [46]. In the three cases our results are in good agreement with the values obtained by Ghia and Erturk.

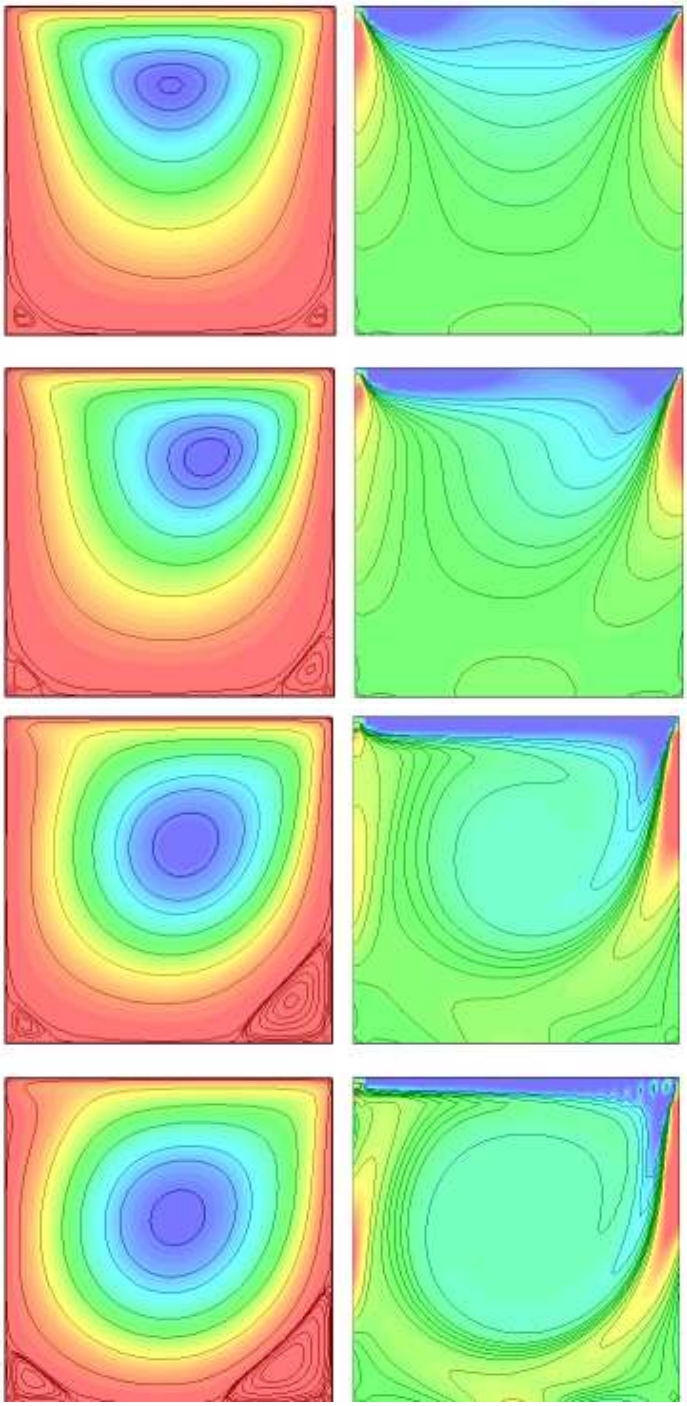
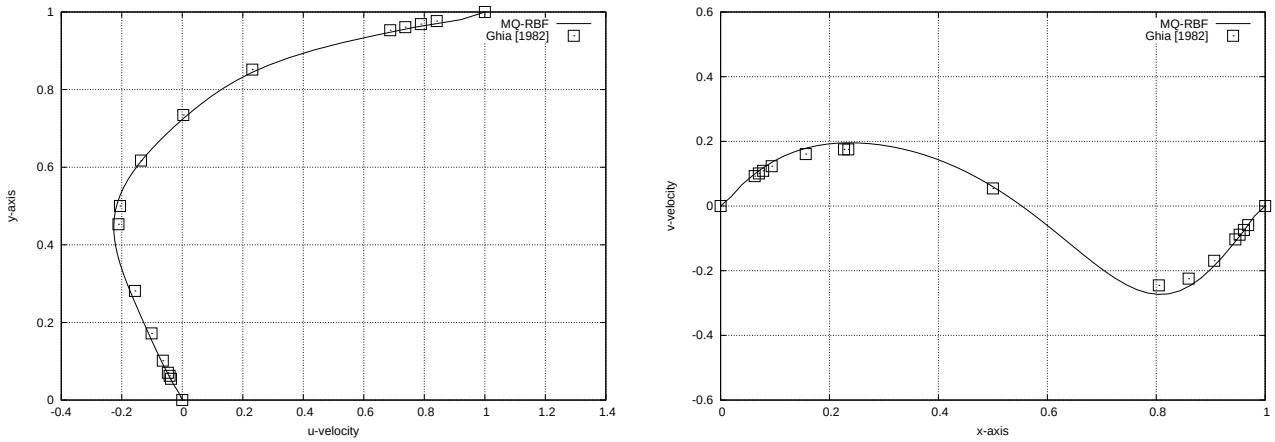
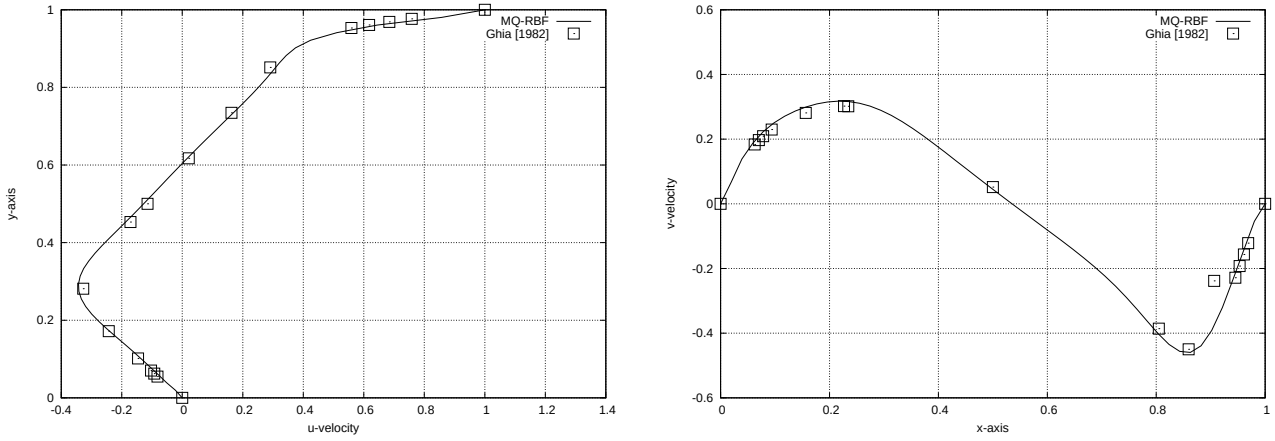


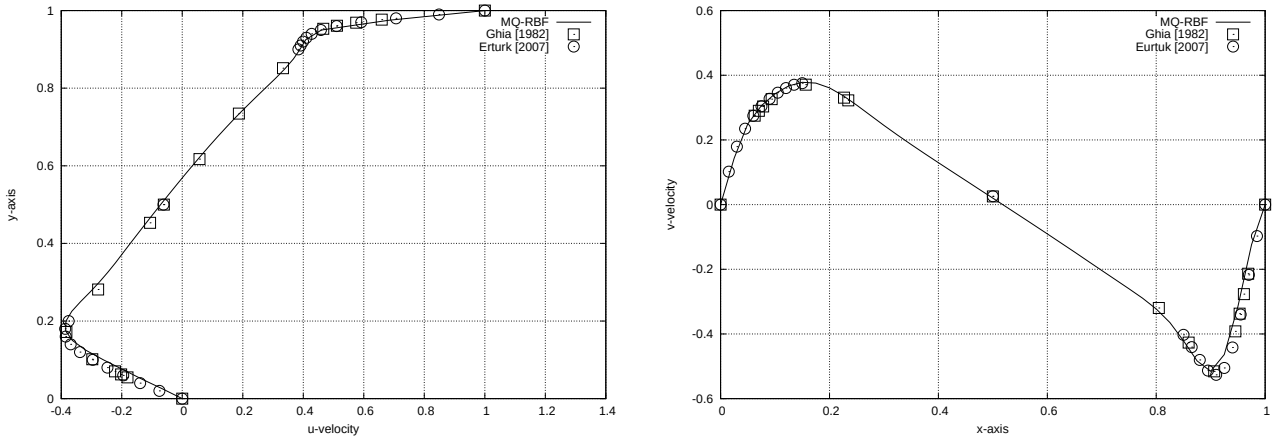
Figure 18: Streamfunction and Vorticity for 51×51.
$Re = 0$, first row; $Re = 100$ second row; $Re = 400$, third row; $Re = 1000$ fourth row;



(a) (b)



(c) (d)



(e) (f)

Figure 19: MQ-RBF profiles compared against Ghia and Erturk results.

4.6 Backward-facing step

Fluid flows in channels with flow separation and reattachment of the boundary layers are encountered in many flow problems. Typical examples are the flows in heat exchangers and ducts. Among this type of flow problems, a backward-facing step can be regarded as having the simplest geometry while retaining rich flow physics manifested by flow separation, flow reattachment and multiple recirculating bubbles in the channel depending on the Reynolds number and the geometrical parameters such as the step height and the channel length.

In this example the backward-facing step is solved, the geometry is shown in the next figure:

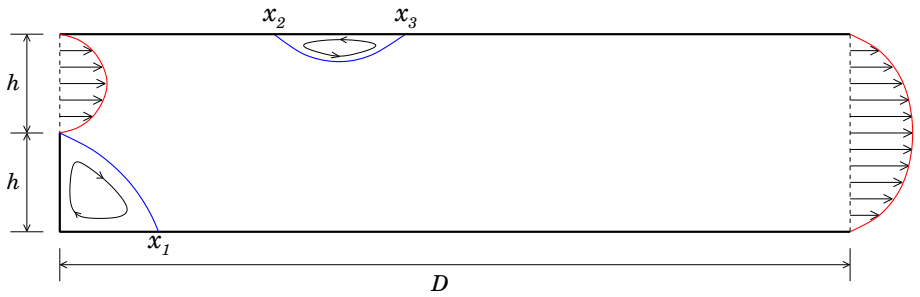


Figure 20: Backward-Facing Step geometry.

In this study, the inlet boundary is in the upper-half of the left wall. The height of the channel is $2h$ and the length is D. The Reynolds number is defined as $Re = 2h\bar{U}/\nu$, where $\bar{U}$ is the mean velocity at inlet and ν is the kinematic viscosity. The mathematical formulation and the RBF methodology are the same as in the case of the Lid-Driven Cavity.

At the inlet boundary a fully developed Plane Poiseuille flow is imposed, such that the inlet velocity profile is parabolic. At the exit boundary we used a parabolic velocity profile as well. The velocity and the streamfunction are written as follows:

Velocity	Streamfunction	Wall
$u(y) = 24y(\frac{1}{2} - y)$	$\psi(y) = 2y^2(3 - 4y))$	Inlet
$u(y) = \frac{3}{4}(1 - 4y^2)$	$\psi(y) = \frac{1}{4}(3y - 4y^3 + 1)$	Outflow

At top wall the value $\psi = 0.5$ is imposed. The vorticity value at the walls are calculated using Jensen's formula, see [44]

The results for different Reynolds numbers are presented in table 9. These results were obtained for $2h = 1$, $D = 30$, $60 \times 15 = 900$ uniformly distributed points, $m = 20$ for the ACBF preconditioner, and $c = 1/\sqrt{N}$. Using the MQ-RBF method was possible to calculate the reattachment length x_1 with good accuracy in comparison with the work of Erturk [47], where a similar study was done for an ample range of Reynolds numbers. For $Re > 400$ the systems becomes unstable and there is not convergence, more points or a bigger shape pararmeter is required. Figure 21 shows the distribution of steamfunction and vorticity for the Reynolds number studied here.

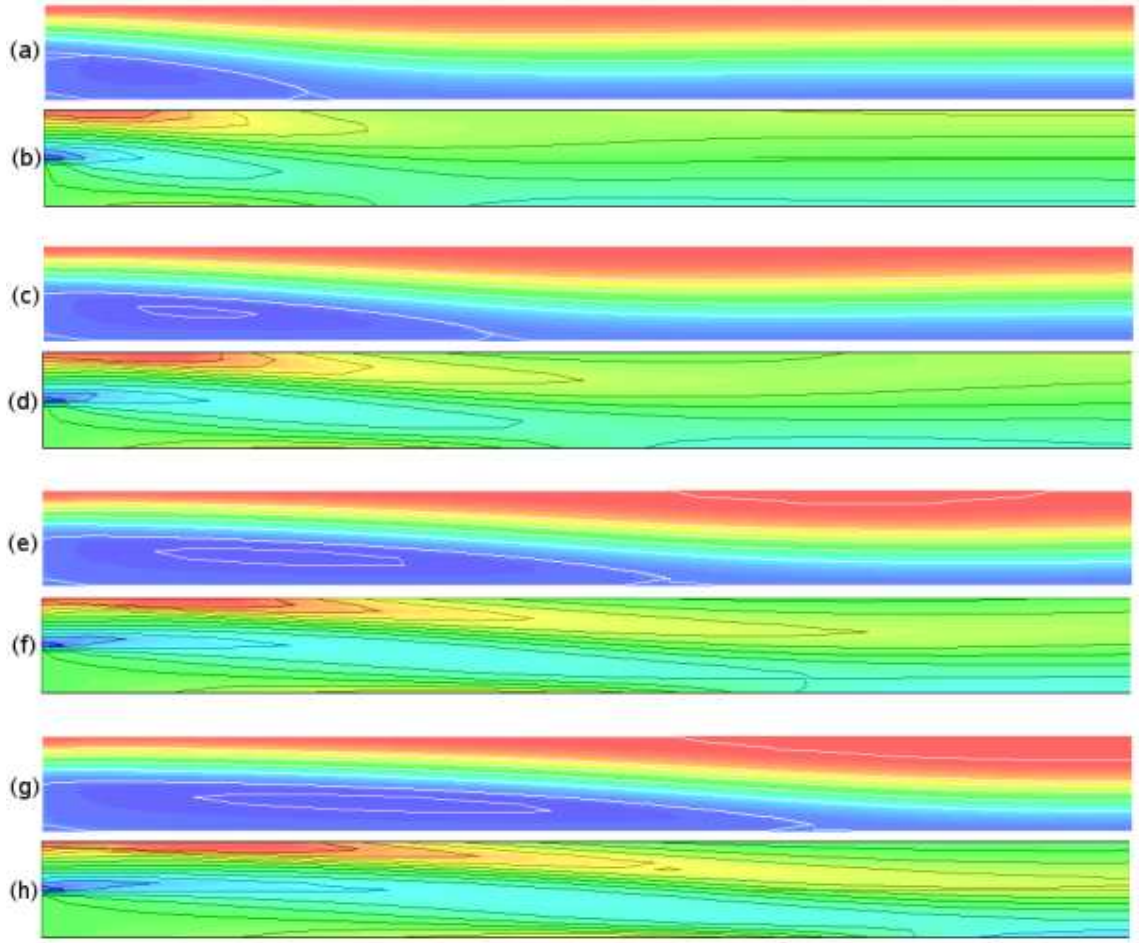


Figure 21: Streamfunction and Vorticiy for the Backward-Facing Step.
(a), (b) $Re = 100$, (c), (d) $Re = 200$, (e), (f) $Re = 300$, (g), (h) $Re = 400$.

Re	Pseudo-temp iter	Vorticity iter	Stream iter	x_1 MQ-RBF	x_1 Erturk [47]
100	536	121	97	2.930	2.922
200	794	143	103	4.975	4.982
300	1306	166	102	6.625	6.751
400	2065	189	101	8.120	8.237

Table 9: Comparison of the results for FV and MQ-RBF methods.

4.7 Domain Decomposition

As a final example, the problem described in section 4.1 was solved using the alternating Schwarz algorithm. The main idea is to partitionate the domain into several subdomains where the same PDE is solved, and the boundary conditions at the interfaces of the subdomains are obtained from the neighbors. The numeration showed in figure 22 (a) correspond to that given by the Cartesian communicator from MPI-2. Four partitions were used in this example: 2, 4, 8 and 16 subdomains, figure 22 (b)

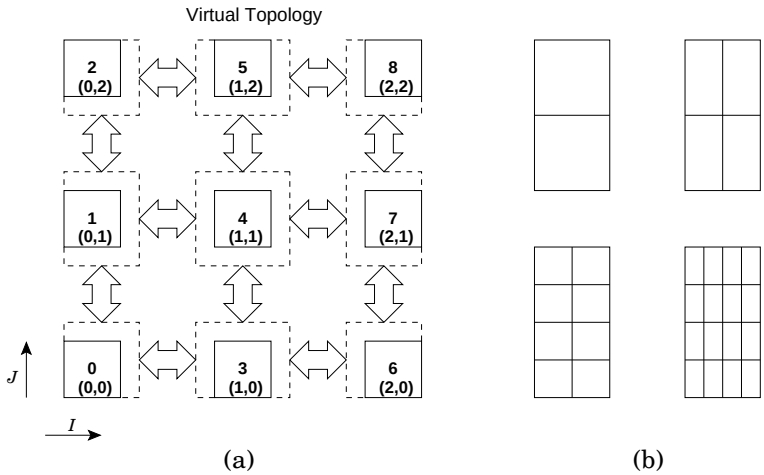


Figure 22: (a) Cartesian communicator in MPI-2, (b) four partitions used in this example.

The algorithm presented in section 2.4.3 is easily extended to a parallel version as follows:

Parallel alternating Schwarz algorithm

.

Make a partition of the domain
Define the problem in each subdomain
$W_1^0 \leftarrow 0$, ..., $W_K^0 \leftarrow 0$
Parallel For $k = 1, ..., K$
  For $n = 1, ...$
    Solve for u_k^n:
      $A_k u_k^n = f_k$ en Ω_k
      $u^n_{\partial\Omega_k \backslash \Gamma_k} = g_k$ on $\partial\Omega_k \backslash \Gamma_k$
      $u^n_{\Gamma_k} = W_k^{n-1}$ on Γ_k
    Check convergency :
      If $||W_k^n - W_k^{n-1}|| \leq \text{tol}_{\Gamma_k}$ END
      If $||u_k^n - u_k^{n-1}|| \leq \text{tol}_{\Omega_k}$ END
  End For
  Wait for all subdomains
  Send $u_k^n|_{\Gamma_k - 1}$ to the neighbours subdomains
  $W_k^0 \leftarrow u^n_{nb}|_{\Gamma_k}$
End Parallel For

Table 10 shows the result obtained for the 4 partitions depicted in figure 22 (b). The shape parameter was calculated as usual $c = 1/\sqrt{N}$, for $N = 1152$, in such a way that the value $c = 0.0294$ was used in all the cases.

Also, and overlapping of 10% was used. The DDM iter represent the total number of cycles given by the Parallel For of the above algorithm. As can be observed, the number of GMRES iterations is reduced as the number of subdomains is increased, which means that the resulting linear system is better conditioned due to the less number of points used in each subdomain. As the number of subdomains is increased, more communications are required, so the number of DDM iterations is also increased. In the same way, the numerical error growths with more subdomains.

Subdomains	Points in each subdom.	GMRES avrg ACBF(20)	DDM Iter	Error	
				RMS	Max
$1 \times 1 = 1$	$24 \times 48 = 1152$	46	1	0.4436	4.7768
$1 \times 2 = 2$	$24 \times 24 = 576$	28	9	0.4528	5.7764
$2 \times 2 = 4$	$12 \times 24 = 288$	22	15	1.1891	10.3094
$2 \times 4 = 8$	$12 \times 12 = 144$	12	18	1.0480	10.4310
$4 \times 4 = 16$	$6 \times 12 = 72$	8	19	3.5420	14.1025

Table 10: Results for the Poisson equation using the parallel Schwarz algorithm.

5 Concluding remarks

During this project several CFD examples were solved using the asymmetric collocation RBF methodology. The Multiquadric kernel was used in all examples obtaining very good accuracy for less number of points in comparison with the FV method. However, the computational effort is in general similar or greater than in the case of FV. Besides, special attention must be paid to the point distribution and to the shape parameter that appear in the MQ-RBF kernel. The shape parameter has a U-shape behaviour with respect to the numerical error: for smaller values of c the error is big, and it decrease as c is increased until reach a minimum. This minimum is commonly obtained when $c = 1/\sqrt{N}$ and this value provides better conditioned systems reducing the computational effort. For $c > 1/\sqrt{N}$ the accuracy is improved but the linear systems become more ill-conditioned and sometimes the matrix is singular. In the case of problems with Neumann boundary conditions, the shape parameter must be adjusted in order to get good results. This adjusting generally consists in increasing c until get the accuracy required.

For Lid-driven cavity and Backward-facing step examples, it was possible to solve the Navier-Stokes equations in the streamfunction–vorticity for moderate Reynolds numbers. For higher values of the Reynolds number it is required either to increase the number of points or to adjust the shape parameter to a value $c > 1/\sqrt{N}$. Both techniques increase the ill-conditioning of the linear system and better algorithms and cheap preconditioners are required. The ACBF preconditioner in general do very good work in reducing the number of condition of the matrices, however, cheaper algorithms need to be implemented in order to construct this preconditioner in reasonable times.

Another way for reducing the condition number of the matrices is by using a Domain Decomposition technique. In this work a small example using the alternating Schwarz algorithm (additive) was implemented to solve the Poisson equation. It was noted that the ill-conditioning is efectively reduced with the number of subdomains, however the numerical error growths. Better parallel algorithms in order to reduce the numerical error and the ill-conditioning need to be developed, particularly the use of the multiplicative version of the Shwarz method could help to reduce the computional work. Also it is important to test these algorithms into a multiprocessor hardware and to measure the speedup for a high number of processors.

The Object-Oriented framework for solving PDEs using the RBF methodology developed during this project, is intended to be one of the first in using this technique, that also can be executed in parallel hardware. This is an open source code and can be obtained from http://ww.dci.dgsca.unam.mx/lmcs/soft.hmtl.

References

[1] T Belytschko, Y Y Lu, L Gu, and P Krysl. Element-free galerkin methods. *International Journal of Numerical Methods in Engineering*.

[2] T Belytschko, Y Krongauze, D Organ, M Fleming, and P Krysl. Meshless methods: an overview and recent developments. *Computational Methods Appl. M.*

[3] I Babuska, U Banerjee, and J E Osborn. Survey of meshless and generalized finite element methods: a unified approach. *Acta Numerica*.

[4] Edward J. Kansa. Multiquadrics–a scattered data approximation scheme with applications to computational fluid-dynamics–i. surface approximations and partial derivatives estimates. *Computers Math. Applic.*, 19(8/9):127–145, 1990.

[5] Edward J. Kansa. Multiquadrics–a scattered data approximation scheme with applications to computational fluid-dynamics–ii. solutions to parabolic, hyperbolic and elliptic partial differential equations. *Computers Math. Applic.*, 19(8/9):147–161, 1990.

[6] Martin D. Buhmann and M. D. Buhmann. *Radial Basis Functions*. Cambridge University Press, New York, NY, USA, 2003.

[7] R Franke. Scattered data interpolation: Tests of some methods. *Math. Comp.*, 38(157):292–300, 1982.

[8] R L Hardy. Multiquadric equations of topography and other irregular surfaces. *J. Geophys. Res.*, 176:1905–1915, 1971.

[9] R L Hardy. Theory and applications of the multiquadric-biharmonic method. 20 years of discovery 19681988. *Computers Math. Applic.*, 10(8/9):163–208, 1990.

[10] W R Madych and S A Nelson. Nelson, multivariate interpolation and conditionally positive definite functions. *Approx. Theory Appl.*

[11] A.I Fedoseyev, M.J Friedman, and E J Kansa. Improved multiquadric method for elliptic partial differential equations via pde collocation on the boundary. *An International Journal Computers & Mathematics with Applications*, 43:439–455, Feb 2002.

[12] B.J.C Baxter. Preconditioned conjugate gradients, radial basis functions, and toeplitz matrices. *An International Journal Computers & Mathematics with Applications*, 43:305–318, Feb 2002.

[13] R K Beatson and G N Newsam. Fast evaluation of radial basis functions: Moment-based methods. *SIAM Journal of Scientific Computing*, 19(5):1428–1449, Apr 1998.

[14] R K Beatson, J B Cherrie, and C T Mouat. Fast fitting of radial basis functions: Methods based on preconditioned gmres iteration. *Advances in Computational Mathematics*, 11:253–270, Oct 1999.

[15] R K Beatson, W A Light, and S Billings. Fast solution of the radial basis function interpolation equations: Domain decomposition methods. *SIAM Journal of Scientific Computing*, 22(5):1717–1740, Dec 2000.

[16] E J Kansa and Y C Hon. Circumventing the ill-conditioning problem with multiquadric radial basis functions: Applications to elliptic partial differential equations. *An International Journal Computers & Mathematics with Applications*, 39:123–137, Jan 2000.

[17] X Zhou, Y C Hon, and Jichun Li. Overlapping domain decomposition method by radial basis functions. *Applied Numerical Mathematics*, 44:241–255, Dec 2003.

[18] Leevan Ling and E J Kansa. Preconditioning for radial basis functions with domain decomposition methods. *Mathematical and Computer Modelling*, 40:1413–1427, Jan 2004.

[19] Leevan Ling and E J Kansa. A least-squares preconditioner for radial basis functions collocation methods. *Advances in Computational Mathematics*, 23:31–54, Mar 2005.

[20] G. Fasshauer and J. Jerome. Multistep approximations algorithms: Improved convergence rates through postconditioning with smooting kernels. *Adv. Comp. Math.*, 10:1–27, 1999.

[21] Damian Brown, Leevan Ling, E J Kansa, and Jermy Levesley. On approximate cardinal preconditioning methods for solving pdes with radial basis functions. *Engineering Analysis with Boundary Elements*, 29:343–353, Apr 2005.

[22] Nicolas Ali Libre, Arezoo Emdadi, E J Kansa, Mohammad Rahimian, and Mohammad Shekarchi. A stabilized rbf collocation scheme for neumann type boundary value problems. *Computer Modelling in Engineering & Sciences*, 24(1):61–80, Feb 2008.

[23] Craig Larman. *Applying UML and patterns: an introduction to object-oriented analysis and design*. 1998.

[24] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. 1999.

[25] W Haerdle and M G Schimek, editors. *Smoothing by local regression: principles and methods*, Statistical Theory and Comutational Aspects of Smoothing. Springer, 1996.

[26] H Wendland. Meshless galerkin methods using radial basis functions. *Math. Comp.*, 68:1521–1531, 1999.

[27] C Franke and R Schaback. Convergence orders of meshless collocation methods using radial basis functions. *Advances in Comp. Math.*, 8:381–399, 1998.

[28] C Franke and R Schaback. Solving partial differential equations by collocation using radial basis functions. *Appl. Math. Comp.*, 93:73–82, 1998.

[29] G E Fasshauer. Solving differential equations with radial basis functions: multilevel methods and smoothing. *Advances in Comp. Math.*, 11:139–159, 1999.

[30] L.R. Hardy. Theory and applications of the multiquadric-biharmonic method: 20 years of discovery. *Computers Math. Applic.*, 19:163–208, 1990.

[31] M.J.D. Powell. The theory of radial basis function approximation in 1990. In *Advances in Numerical Analysis, Vol II*. Oxford Science Publications, Oxford, 1992.

[32] Y Saad and M H Schultz. Gmres: a generalized minimal residual algorithm for solving nonsymmetric linear equations. *SIAM J. Sci. Statist. Comput.*

[33] F P Preparata and M I Shamos. *Computational Geometry: An introduction*. Springer-Verlag, 1985.

[34] B. F. Smith, P. E. Bjorstad, and W. D. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996.

[35] A S M Wong, Y C Hon, T S Li, S L Chung, and E J Kansa. Multizone decomposition for simulation of time-dependent problems using the multiquadric scheme. *An International Journal Computers & Mathematics with Applications*, 37:23–43, Feb 1999.

[36] D. Vandevoorde and N. M. JosuttisOB. *C++ Templates*. 2003.

[37] Todd Veldhuizen. Using C++ template metaprograms. *C++ Report*, 7(4):36–43, May 1995. Reprinted in C++ Gems, ed. Stanley Lippman.

[38] Todd L. Veldhuizen. Expression templates. *C++ Report*, 7(5):26–31, June 1995. Reprinted in C++ Gems, ed. Stanley Lippman.

[39] The object-oriented numerics page. http://www.oonumerics.org.

[40] K. A. Hoffman. *Computational Fluid Dynamics for Engineers*. The University of Texas at Austin, 1989.

[41] S. V. Patankar. *Numerical Heat Transfer and Fluid Flow*. 1980.

[42] H.K. Versteeg and W. Malalasekera. *An introduction to computational fluid dynamics: The finite volume method*. Longman, 1995.

[43] Template units for numerical application: The finite volume method. http://www.dci.dgsca.unam.mx/lmcs/soft.html.

[44] C. Fletcher. *Computational Techniques for Fluid Dynamics 1: Specific Techniques for Different Flow Categories*. Springer–Verlag, second edition edition, 1991.

[45] U Ghia, K N Ghia, and C T Shin. High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method. *Journal of Computational Physics*, 48:387–411, Dec 1982.

[46] Ercan Erturk, T C Corke, and C Gokcol. Numerical solutions of 2-d steady incompressible driven cavity. *International Journal for Numerical Methods in Fluids*, 48:747–774, Mar 2005.

[47] Ercan Erturk. Numerical solutions of 2-d steady incompressible flow over a backward-facing step, part i: High reynodls number solutions. *Computers & Fluids*, 37:633–655, 2008.