Journal of Computational Physics 228 (2009) 4976–4999

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

Fast radial basis function interpolation with Gaussians by localization and iteration

Claudio E. Torres [a], L.A. Barba [b,*]

[a] Department of Mathematical Sciences, University of Delaware, United States
[b] Department of Mathematics, University of Bristol, United Kingdom

ARTICLE INFO

Article history:
Received 13 November 2008
Received in revised form 23 February 2009
Accepted 4 March 2009
Available online 19 March 2009

Keywords:
Radial basis functions
Sum of Gaussians
Particle methods
Vortex method
Iterative methods
Interpolation

ABSTRACT

Radial basis function (RBF) interpolation is a technique for representing a function starting with data on scattered points. In the vortex particle method for solving the Navier–Stokes equations, the representation of the field of interest (the fluid vorticity) as a sum of Gaussians is like an RBF approximation. In this application, there are two instances when one may need to solve an RBF interpolation problem: upon initialization of the vortex particles, and after spatial adaptation to replace a set of disordered particles by another set with a more regular distribution. Solving large RBF interpolation problems is notoriously difficult with basis functions of global support, due to the need to solve a linear system with a fully populated and badly conditioned matrix. In the vortex particle method, in particular, one uses Gaussians of very small spread, which lead us to formulate a method of solution consisting of localization of the global problem, and improvement of solutions over iterations. It can be thought of as an opposite approach from the use of compact support basis functions to ease the solution procedure. Compact support bases result in sparse matrices, but at the cost of reduced approximation qualities. Instead, we keep the global basis functions, but localize their effect during the solution procedure, then add back the global effect of the bases via the iterations. Numerical experiments show that convergence is always obtained when the localized domains overlap via a buffer layer, and the particles overlap moderately. Algorithmic efficiency is excellent, achieving convergence to almost machine precision very rapidly with well-chosen parameters. We also discuss the complexity of the method and show that it can scale as $\mathcal{O}(N)$ for N particles.

1. Introduction

The technique of radial basis function interpolation first appears in the literature as a method for scattered data interpolation, and interest in this method exploded after the review of Franke [11], who found it to be the most impressive of the many methods he tested. Later, a scheme for the estimation of partial derivatives using RBFs was proposed by Kansa [14], resulting in a new method for solving partial differential equations [15]. A prominent feature of radial basis function methods is that they are truly meshfree: the methods use nodes in the computational domain whose location can be chosen at will. This feature is very important due to the great computational expense which is mesh generation and maintenance in large-scale applications. However, there is a considerable hurdle in the fact that RBF methods result in a linear system with a dense, ill-conditioned matrix, with N unknowns for N nodes or centers. This makes it quite difficult to use RBF methods in very large data sets. Many authors have partially circumvented this problem by choosing to use basis functions of compact

* Corresponding author. Address: Department of Mechanical Engineering, Boston University, Boston, MA 02215, United States.
E-mail address: labarba@bu.edu (L.A. Barba).

0021-9991/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2009.03.007

support, but sacrificing interpolation accuracy in the process. RBFs of compact support result in sparse matrices, but their approximation properties are inferior to global basis functions [5].

The application driving our interest is the use of radial basis functions in computational methods for fluids simulation. However, RBF interpolation has found many applications over the years, for which the work presented in this paper should be relevant. As listed in [8], RBF as a scattered data fitting technique finds application in the mapping of problems in geodesy, geophysics and meteorology (in general, any interpolation problem has the potential of being formulated with RBFs). As a form of non-uniform sampling, it is used in medical imaging including tomographic reconstruction. RBFs are used in mathematical finance, for options pricing, and in computer graphics, for representation of surfaces from laser scan data. Gaussian functions, in particular, are used extensively in statistics, machine learning, computer vision and others. The applications are many, but in this paper we focus on the use of radial basis functions within particle simulation approaches of fluid dynamics.

In computational fluid dynamics, an approach for solving the governing equations that is also meshfree, and has some relationship with radial basis function methods, is the vortex method. In this method, the Navier–Stokes equations for a viscous fluid are converted to vorticity formulation, and the vorticity field is represented by a superposition of basis functions. Let $\omega(x,t)$ be the vorticity field of a velocity field $u(x,t)$, so $\omega(x,t) = \nabla \times u(x,t)$. The spatial approximation of this field is performed over a set of moving nodes located at x_i, as follows:

$$\omega(x,t) \approx \omega_\sigma(x,t) = \sum_i^N \gamma_i \zeta_\sigma(x,x_i) \tag{1}$$

In the point vortex method, the basis function used is a Dirac delta function, whereas in the smooth vortex particle method a number of different bases ζ_σ can be used. Often, the choice is a Gaussian function, normalized so that it integrates to 1, such as:

$$\zeta_\sigma(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(\frac{-|x-y|^2}{2\sigma^2}\right) \tag{2}$$

The motivation for this type of discretization in the vortex method comes from the physics, and not approximation theory. In the case of a two-dimensional fluid with no viscosity, the vorticity equation takes its simplest form:

$$\frac{\partial\omega}{\partial t} + u \cdot \nabla\omega = \frac{\mathrm{D}\omega}{\mathrm{D}t} = 0 \tag{3}$$

which simply expresses that vorticity is a quantity that is preserved following material trajectories in the flow. This justifies discretizing the vorticity field over particle-like elements, which are then allowed to move following the fluid velocity. Such a Lagrangian approach, added to the fact that the vorticity field is often compact (in contrast with the velocity field), makes vortex methods a very powerful tool to simulate flows involving bluff bodies, massive separation, or in general where the vorticity dominates the dynamics.

To incorporate the viscous effects, many schemes have been proposed over the years. One approach that we can use is to apply the Laplacian operator to the discretized vorticity (1):

$$\nabla^2\omega_\sigma = \sum_i^N \gamma_i \nabla^2 \zeta_\sigma(x,x_i) \tag{4}$$

and use this expression in the 2D Navier–Stokes equation for the discretized vorticity:

$$\frac{\mathrm{D}\omega_\sigma}{\mathrm{D}t} = \frac{1}{Re}\nabla^2\omega_\sigma \tag{5}$$

Such a method, introduced in [10], is analogous to the approach used in the radial basis function methods to solve PDEs [15], where each term involving partial derivatives in the equations is obtained by differentiation of the linear superposition of basis functions. A notable difference is that in vortex methods the nodes are moving, whereas in the standard RBF method for PDEs what changes in time is the value of the nodal coefficients. For a discussion of several other methods for incorporating the viscous term in vortex methods, see [2].

One of the first questions that we may ask when interested in using the vortex method to simulate a given flow problem is how to initially discretize a vorticity field, given by an analytical expression or by a data set, in the form (1). Suppose that the vorticity is known on a set of points $\{x_j\}$ at the initial time. The question is how to accurately represent $\omega(x,t=0)$ if $\omega_j = \omega(x_j)$ is given as data. This is precisely a problem of radial basis function interpolation, where we need to find the nodal coefficients γ_i by collocation:

$$\begin{aligned}\omega_\sigma(x_j) &= \omega_j \\ \omega_\sigma(x_j) &= \sum_i^N \gamma_i \zeta_\sigma(x_j,x_i)\end{aligned} \tag{6}$$

The above problem demands solving a linear system $\mathscr{A}\vec{\gamma} = \vec{\omega}$, with a coefficient matrix formed using the basis function on pairs of data points:

$$\mathscr{A}_{ij} = \zeta_\sigma(x_j, x_i) \tag{7}$$

Clearly, the matrix $\mathscr{A}$ will be full, if basis functions of global support are used. Moreover, it is known that this matrix is not diagonally-dominant, which brings about difficulty in solving the system numerically.

In the radial basis function literature, the first work dedicated to developing a custom preconditioning operator for effectively solving such an ill-conditioned system was presented in [7]. The authors developed a preconditioner for the thin-plate spline (TPS) radial basis, defined as: $\phi(r) = r^2 \log(r)$. Recognizing that the basis function is the fundamental solution of the bi-harmonic equation $\nabla^4\phi = 0$, the preconditioner is based on the fact that the bi-harmonic operator applied to ϕ will produce a Dirac delta function multiplied by a factor. Thus, clustering of the eigenvalues is achieved by operating on the matrix with a discretization of the bi-harmonic operator, using a triangulation with vertices at the data points.

Subsequently, a preconditioning method was presented in [4] which is based on changing the basis by means of approximate cardinal functions. The preconditioning strategy there was tested with the TPS basis and multi-quadrics, $\phi(r) = (c^2 + r^2)^{\frac{1}{2}}$, and experiments were performed with up to 10,000 nodes. The method involves solving a small linear system of size $\beta \times \beta$, with $\beta \ll N$, for each node x_j, to find the coefficients of an approximate cardinal function centered at that node. Based on the same idea of approximate cardinal functions, a Krylov subspace algorithm was developed in [9], which is described as analogous to preconditioned conjugate gradient. The method, however, scales as $\mathcal{O}(N^2)$ in both computational complexity and memory requirements. An acceleration of the method to $\mathcal{O}(N \log N)$ was provided in [13], using the Fast Multipole Method [12] to compute the matrix vector products inside the iterations of the iterative solver. The methods just cited were implemented specifically for polyharmonic kernels, $\phi(r) = r^{2n+1}$, and multi-quadrics.

As mentioned before, in vortex methods it is common to use the Gaussian basis function. Moreover, being a Lagrangian method where each smooth particle moves with the local fluid velocity, the spread of the Gaussians is required to be small. In fact, it is commonly accepted that in a vortex method calculation σ corresponds to the smallest scale that can be resolved in the flow. Because we are interested in the RBF interpolation with Gaussians of small spread, one can devise specialized methods that take into account the fact that the basis functions decay rapidly away from their centers. Recently, a custom preconditioner has been proposed for the Gaussian basis which is able to provide excellent algorithmic efficiency [3]. The preconditioner idea is based precisely on this feature of the basis: its fast decay away from each center. We will present the preconditioning strategy for the Gaussian later on in this paper, for completeness, and because it will be used as well in the present work.

The main contribution in this work is the development and numerical demonstration of solution methods for radial basis function interpolation with Gaussians, which are based on two main ideas. The first is that localization by neglecting the far-field influence of the Gaussians should give an approximate solution of the local interpolation problem in the vicinity of a center. Then, aggregation of the many local problems will give an approximation of the global problem. The second ingredient adds an iterative strategy to use the aggregated local solutions as consecutive approximations of the global solution. The methods are characterized by being amenable to parallelization, by demonstrating excellent algorithmic efficiency (number of iterations to converge), and by excellent interpolation accuracy (close to machine precision).

In the next section, we present the radial basis function interpolation problem more formally, and we summarize the specialized preconditioner developed in [3]. In Section 3, we define the methods of solution and discuss the techniques of localization and iteration. Subsequently, numerical experiments are presented that demonstrate the algorithmic efficiency and accuracy of the method developed in this work. In Section 5, we give some implementation details for the algorithm and develop a study of the computational complexity, demonstrating in practice that we can obtain close to $\mathcal{O}(N)$ complexity. We end with a brief discussion, conclusions and comments about future work.

2. Background on radial basis function interpolation with Gaussians

In a radial basis function interpolation problem we are faced with the question of approximating a function, assumed to be continuous, where only a scattered set of values are known of the function. Let the function values $f(x_i)$ be known for a set of points in the domain $X = \{x_1, \dots, x_N\} \subset \Omega \subset \mathbb{R}^d$. Following common notation, as for example used in [17], we write the approximation to f in the form:

$$s_{f,X}(x) = \sum_{j=1}^{N} \alpha_j \Phi(x, x_j) \tag{8}$$

where $\Phi : \Omega \times \Omega \to \mathbb{R}$ is the basis function, with the following property:

$$\Phi(x, y) = \phi(\|x - y\|_2), \quad \text{with } \phi : [0, \infty) \to \mathbb{R} \text{ (radiality)}. \tag{9}$$

Satisfying the collocation conditions on the data points leads to a linear system for the coefficients $\vec{\alpha} = (\alpha_1, \dots, \alpha_N)^T$. Writing $\vec{f}$ for the function values $\vec{f} = (f(x_1), \dots, f(x_N))^T$, we need to solve:

$$\mathbf{\Phi}\vec{\alpha} = \vec{f} \tag{10}$$

where $\mathbf{\Phi}_{ij} = \phi(\|x_i - x_j\|)$. An important set of theoretical results guarantees a solution for this system if the function $\Phi(x, y)$ is strictly conditionally positive definite and the data distinct [16]. The main difficulty lies in the ill-conditioning of the system, especially for large data sets, and the computational cost.

As discussed in our introduction, preconditioning strategies have been proposed to alleviate the ill-conditioning and allow the use of iterative solution methods. Available results concentrate in particular on the multi-quadric and thin-plate spline basis functions [7,4]. In our application, where we have Gaussian bases of small spread, one can take advantage of the rapid decay of the basis to propose a specialized preconditioning method. We wish to apply a preconditioner $\mathscr{M}$ to the system (10),

$$\mathscr{M}^{-1}\mathbf{\Phi}\vec{\alpha} = \mathscr{M}^{-1}\vec{f} \tag{11}$$

such that the new system is easier to solve, due to a clustering of the eigenvalues of the coefficient matrix. We want the eigenvalues of $\mathscr{M}^{-1}\mathbf{\Phi}$ to be clustered, which means that a good preconditioner $\mathscr{M}$ is in some way an approximation to $\mathbf{\Phi}$. Due to the rapid decay of the Gaussian, a preconditioner based on a sparse approximation to $\mathbf{\Phi}$ can be built by making zero the matrix elements corresponding to the interaction of two particles which are farther from each other than a chosen threshold. Thus,

$$\mathscr{M}_{ij} = \begin{cases} \zeta_\sigma(x_j - x_i) & \text{if } |x_j - x_i| < R \\ 0 & \text{if } |x_j - x_i| > R \end{cases} \tag{12}$$

In [3], such a preconditioner was found to produce convergence in only a handful of iterations of the GMRES method, and achieve almost machine precision when calculating the interpolation error at the collocation points. The important choice of the parameter R determines the efficacy of the preconditioner, with values above 6σ being effective.

In production codes of the vortex method, it is not uncommon to utilize in the order of 10^6 or even more particles. For such large N, more than just preconditioning is required, especially if one wishes to implement the codes in parallel. The memory requirements of a matrix-based implementation would be prohibitive, and parallelization easily bogged down by inter-processor communication.

In that vein, we develop a method which begins by localizing the global problem, and solving many small systems for local domains. However, this in itself is not enough, as the basis functions are global, so a means of incorporating the long-range effects will be required. In some sense, we take the opposite approach to using compact support basis functions. Basis functions of compact support produce sparse matrices and can be solved easily, but they suffer from low convergence properties and have inferior approximation qualities than global functions. We therefore use global functions, but use an approach to solve the dense system that localizes the effect of the bases, then adds the long-range effects in such a way as to maintain the high accuracy provided by the global interpolation.

3. Methods of solution by localization and iteration

3.1. Simplified one-dimensional description

To illustrate the ideas incorporated in our method, we start by a very simplified description in one dimension. Suppose we have a one-dimensional field $V(x)$ (corresponding to the vorticity in our application of interest) defined on $[a, b]$ where $b > a$ and $a, b, x \in \mathbb{R}$. The goal is obtaining an approximation to $V(x)$ by a linear combination of Gaussian functions centered on a set of points in $[a, b]$. We will call the centers of the Gaussians "particles", even in this one-dimensional description. Let the field $V(x)$ be known at a set of points $\{x_i\}$ in $[a, b]$. Our problem is expressing the field as a sum of basis functions:

$$V(x) = \sum_{i}^{N} \gamma_i \mathbb{G}_\sigma(x, x_i) \tag{13}$$

where the coefficients γ_i are unknown and the basis function centered at x_c is:

$$\mathbb{G}_\sigma(x, x_c) = \mathbb{G}_\sigma(x - x_c) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-|x - x_c|^2}{2\sigma^2}\right) \tag{14}$$

For simplicity in our illustration of the method, let us take $N = 10$ and consider the centers to be equally spaced in the interval $[a, b]$. In other words, the set $\mathscr{P} = \{a = x_1, x_2, \ldots, b = x_{10}\}$ produces an equipartition of $[a, b]$, and the function is known at these locations. The problem of expressing V in the form (13) results in a linear system of equations for the coefficient values, γ_i, which is written in matrix form as

$$\underbrace{\begin{pmatrix} \mathbb{G}_\sigma(x_1 - x_1) & \cdots & \mathbb{G}_\sigma(x_1 - x_{10}) \\ \vdots & \ddots & \vdots \\ \mathbb{G}_\sigma(x_{10} - x_1) & \cdots & \mathbb{G}_\sigma(x_{10} - x_{10}) \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} \gamma_1 \\ \vdots \\ \gamma_{10} \end{pmatrix}}_{\Gamma} = \underbrace{\begin{pmatrix} V(x_1) \\ \vdots \\ V(x_{10}) \end{pmatrix}}_{V} \tag{15}$$

Now, we introduce the idea of a local problem, which considers only a cluster or small sub-set of particles that are close to each other (in the Euclidean sense), and solves the collocation problem locally using the corresponding sub-set of basis functions. For the purposes of our illustration, let us define five local domains with particles $\{x_{2i-1}, x_{2i}\}$ in local domain *i*, for *i* from 1 to 5.

The first and most basic idea of localization would attempt to solve each local problem separately; in the ongoing example, this would translate into solving five 2×2 linear systems. For example, if we consider the problem in local domain 3, the system is:

$$\underbrace{\begin{pmatrix} \mathbb{G}_\sigma(x_5 - x_5) & \mathbb{G}_\sigma(x_5 - x_6) \\ \mathbb{G}_\sigma(x_6 - x_5) & \mathbb{G}_\sigma(x_6 - x_6) \end{pmatrix}}_{A_3} \begin{pmatrix} \gamma_5 \\ \gamma_6 \end{pmatrix} = \underbrace{\begin{pmatrix} V(x_5) \\ V(x_6) \end{pmatrix}}_{V_3} \tag{16}$$

where A_3 denotes the matrix associated to local domain 3. In general A_j will refer to the matrix representing the pair-wise influence of particles in local domain *j*.

Clearly, a straightforward superposition of the solutions to the local problems will be a bad approximation of the global problem, because the global influence of the basis functions will result in additional contributions from each local domain to the function value at each particle center. This will spoil the collocation obtained by the local problem at each particle location.

Hence, the second approach that one may think of would solve each of the local problems, but somehow taking into account the long-range influence of other local domains. Again using our simple 1D illustration, suppose that we are solving local problem 3, but we want to take into consideration all the particles, resulting in the following equation:

$$\begin{pmatrix} \mathbb{G}_\sigma(x_5 - x_1) & \mathbb{G}_\sigma(x_5 - x_2) & \cdots & \mathbb{G}_\sigma(x_5 - x_{10}) \\ \mathbb{G}_\sigma(x_6 - x_1) & \mathbb{G}_\sigma(x_6 - x_2) & \cdots & \mathbb{G}_\sigma(x_6 - x_{10}) \end{pmatrix} \begin{pmatrix} \gamma_1 \\ \vdots \\ \gamma_{10} \end{pmatrix} = V_3 \tag{17}$$

We are left with an underdetermined system of two equations with 10 unknowns — we want to fit only the two particles in local domain 3, but taking into consideration all the particles in the other local domains. At this point, we introduce the strategy of iterations. Suppose that we have an initial guess for the coefficients corresponding to all the particles. Then we proceed to solve each local problem separately, but using the initial guess for the coefficients of the particles *not* in the local domain to include the long-range influence of all particles. Thus, all terms in (17) except those involving γ_5 and γ_6 are moved to the right-hand-side, and we again have a 2×2 system, but with the influence of *all* particles incorporated. Once all the local systems are solved, we have a new set of coefficients γ_i which constitutes the first iteration. The process can now be repeated, and we submit to further investigation whether this strategy will produce convergence to an accurate solution of the global problem (see Section 4).

To continue using our 1D illustration, we introduce notation to refer to the iterations, in addition to the numbering of local domains. Let us continue referring to local domain 3 as an example and consider:

Particles in the local domain: The particles in local domain 3, in this case $\{x_5, x_6\}$, will be identified by $\{x_1^3, x_{L_3}^3\}$ — in a general case this becomes x_i^j where *i* locally enumerates the particles in the local domain *j*, and L_j is the index of the last particle in local domain *j*. We introduce this notation because in general we might have local domains with different numbers of particles. So if local domain 3 were to have more elements, they would be identified as: $\{x_1^3, x_2^3, x_3^3, \ldots, x_{L_3}^3\}$.
Coefficients of particles in a local domain over iterations: The coefficients associated to the particles in local domain 3, $\{\gamma_5, \gamma_6\}$, at iteration 0 will be identified as follows: $\{\gamma_1^{3(0)}, \gamma_{L_3}^{3(0)}\}$ — in a general case this becomes $\gamma_i^{j(t)}$ where *i* and *j* have the same purpose explained above and *t* identifies the iteration number.

Let us denote by $\Gamma^{(0)}$ the vector formed by the initial guess for all the coefficients. The elements of this vector can be written in the 'global notation' or in the 'local notation', as follows:

$$\Gamma^{(0)} = \left(\gamma_1^{(0)}, \gamma_2^{(0)}, \ldots, \gamma_{10}^{(0)}\right)^T \text{ global notation} = \left(\underbrace{\gamma_1^{1(0)}, \gamma_{L_1}^{1(0)}}_{\Gamma_1^{(0)}}, \underbrace{\gamma_1^{2(0)}, \gamma_{L_2}^{2(0)}}_{\Gamma_2^{(0)}}, \ldots, \underbrace{\gamma_1^{5(0)}, \gamma_{L_5}^{5(0)}}_{\Gamma_5^{(0)}} \right)^T \text{ local notation}$$

so in general $\Gamma^{(t)}$ identifies all the coefficients of the particles at iteration *t*, $\Gamma_j^{(t)}$ identifies the coefficients of the particles that belong to local domain *j* at iteration *t*, and ^{T} denotes the transpose operator.

Going back to our example where we are solving for local domain 3, we are left with the following system:

$$\underbrace{\begin{pmatrix} \mathbb{G}_\sigma\left(x_1^3 - x_1^3\right) & \mathbb{G}_\sigma\left(x_1^3 - x_{L_3}^3\right) \\ \mathbb{G}_\sigma\left(x_{L_j}^3 - x_1^3\right) & \mathbb{G}_\sigma\left(x_{L_j}^3 - x_{L_3}^3\right) \end{pmatrix}}_{A_3} \underbrace{\begin{pmatrix} \gamma_1^{3(1)} \\ \gamma_{L_3}^{3(1)} \end{pmatrix}}_{\Gamma_3^{(1)}} = \underbrace{\begin{pmatrix} V(x_5) \\ V(x_6) \end{pmatrix}}_{V_3} - \begin{pmatrix} \mathbb{G}_\sigma(x_5 - x_1) & \cdots & \mathbb{G}_\sigma(x_5 - x_4) \\ \mathbb{G}_\sigma(x_6 - x_1) & \cdots & \mathbb{G}_\sigma(x_6 - x_4) \end{pmatrix} \begin{pmatrix} \gamma_1^{(0)} \\ \vdots \\ \gamma_4^{(0)} \end{pmatrix} \cdots$$

$$- \begin{pmatrix} \mathbb{G}_\sigma(x_5 - x_7) & \cdots & \mathbb{G}_\sigma(x_5 - x_{10}) \\ \mathbb{G}_\sigma(x_6 - x_7) & \cdots & \mathbb{G}_\sigma(x_6 - x_{10}) \end{pmatrix} \begin{pmatrix} \gamma_7^{(0)} \\ \vdots \\ \gamma_{10}^{(0)} \end{pmatrix}$$

Rewriting the right-hand-side, we have:

$$A_3\Gamma_3^{(1)} = V_3 - \underbrace{\begin{pmatrix} \mathbb{G}_\sigma(x_5 - x_1) & \cdots & \mathbb{G}_\sigma(x_5 - x_{10}) \\ \mathbb{G}_\sigma(x_6 - x_1) & \cdots & \mathbb{G}_\sigma(x_6 - x_{10}) \end{pmatrix}}_{\widehat{A}_3} \underbrace{\begin{pmatrix} \gamma_1^{(0)} \\ \gamma_2^{(0)} \\ \vdots \\ \gamma_{10}^{(0)} \end{pmatrix}}_{\Gamma^{(0)}} + A_3\Gamma_3^{(0)}$$

which is written using the matrix notation as

$$A_3\Gamma_3^{(1)} = V_3 - \widehat{A}_3\Gamma^{(0)} + A_3\Gamma_3^{(0)}$$

where in general $\widehat{A}_j$ is the interaction of the particles in local domain j against all the particles. We can interpret the expression $V_j - \widehat{A}_j\Gamma^{(t)}$ as the residual at iteration t on the local domain j.

Therefore, the iterative algorithm in general is expressed by:

$$A_j\Gamma_j^{(t+1)} = V_j - \widehat{A}_j\Gamma^{(t)} + A_j\Gamma_j^{(t)} \tag{18}$$

which now allows us to generalize it to any partition of a domain in any dimension.

The algorithm developed thus far has the following important features:

- For a partition of the global domain containing N particles, where each local domain has $L \ll N$ particles clustered together, we need to solve many small local systems at each iteration, rather than one very large system for the global problem.
- Each local system is solved independently at each iteration, which means that the algorithm is highly parallel.
- We do not need to build the complete coefficient matrix of size $N \times N$ in memory at any time.

The features listed above suggest that we have a potentially excellent method for solving our global interpolation problem, with very large numbers of centers N — if only the iterations would converge to an accurate solution. Unfortunately this is not the case, as will be demonstrated with numerical experiments in the next section.

As it turns out, the strategy introduced so far *does* produce an excellent method if we add only one more ingredient: a "buffer" area around each local domain, where particles are considered local and solved for together with the local problem. In our previous 1D example, when considering local domain 3, we would take particles on each side of domain 3 to include in the local solution. The extent of this "buffer" zone needs to be investigated, but let us start by considering simply the same size as the local domain. Thus for local domain 3, the buffer consists of domains 2 and 4. Let us use underlines to denote a local domain with its buffer area, so $\underline{3}$ denotes a larger local domain consisting of sub-domains 2, 3 and 4. Now we can use the algorithm of localization and iteration, as described previously, on the underlined sub-domain with its buffer.

$$\underline{A_j}\underline{\Gamma_j}^{(t+1)} = \underline{V_j} - \underline{\widehat{A}_j}\Gamma^{(t)} + \underline{A_j}\underline{\Gamma_j}^{(t)} \tag{19}$$

The clue for the method to converge, as we will demonstrate experimentally, is that the solution to this larger local domain is not used in its entirety to update $\Gamma^{(t+1)}$ from $\underline{\Gamma_j}^{(t+1)}$. In fact, only the part of the solution vector corresponding to the local domain *without* the buffer area is used, and the part of the solution vector corresponding to the buffer zone is discarded. This is why we call it a "buffer": it provides a zone that cushions the new solution from the local effect of the long-range influence of the previous iteration. So, in terms of the ongoing 1D example where we solve for the buffered domain $\underline{3}$, we can illustrate the procedure with:

$$\underline{\Gamma_3}^{(1)} = \begin{pmatrix} \vdots \\ \hline \Gamma_3^{(1)} \\ \hline \vdots \end{pmatrix} \rightarrow \text{recover } \Gamma_3^{(1)} \text{ from } \underline{\Gamma_3}^{(1)} \rightarrow \text{update } \Gamma^{(1)} \text{ using } \Gamma_3^{(1)}$$

In summary, we solve each local problem including its buffer area, but recover only the coefficients associated with the local domain to update the global solution, discarding the new values at the buffer zone.

3.2. *Two-dimensional application*

In fact, the iterative algorithm presented is applied exactly in the same way for any dimension. We just need to identify what is meant by the local domain and the buffer zone for the two-dimensional case. This is best illustrated by means of a figure. Consider Fig. 1, where we have sketched a computational domain in two dimensions, where the particles reside. The local domains here are represented by the square areas, but we emphasize that the local domains can have any shape: they could be circular, or they could be formed by some clustering algorithm. In the figure, we show how a local domain, indicated by the darker square, is surrounded by a "buffer zone", indicated by the lighter shaded squares, all around. Again, the linear size of the buffer zone is here assumed to be the same as the local domain size, but this is not a requirement. Rather, we submit to further investigation which is the optimal size of the buffer zone, such that it is as small as possible but ensures convergence of the iterative algorithm.

The description of the algorithm given by Eq. (19) applies with no modification in two and three dimensions. In the following section, we present numerical experiments in 1D and 2D.

3.3. *Providing an initial guess*

In the vortex method, it is common that initialization will be performed by laying out particles on a square lattice and estimating the particle weights simply by using the local value of vorticity times a rough estimate of the particle areas (or volumes in 3D). This initialization is quite standard, and is expressed as follows:

$$\gamma_i = \omega(x_i)h^d \tag{20}$$

where d is the dimension. In [3], it has been proved that this initialization amounts to a Gaussian blurring of the original field. It is a simple method, performed with very low computational effort, but the accuracy is quite low. For our purposes, however, it provides for an excellent initial guess to be used in the iterative algorithm.

3.4. *Summary of the algorithm*

To summarize the complete algorithm, as developed in this section, we provide a listing in pseudo-code. See Algorithm 1, below. To be specific, the algorithm refers to our problem of interest, *i.e.* representing the vorticity field of a fluid by a sum of Gaussian particles, instead of a general RBF interpolation problem. We treat the two-dimensional case, where the particle circulations (the solution of the RBF interpolation) are scalars. We also do not give details of the part of the algorithm that generates the local domains, and just list a call to a function `generateLocalDomains`(·); this could be a boxing or a clustering method of choice. It should include a method for determining the adjoining boxes or clusters which constitute the buffer layer for each local domain. We indicate this by the variable `Buffer` which contains in element i the indices of the elements of `LD` that belong to the buffer layer of local domain i. Note that the number of local domains `K` does not depend on the data `Z` itself, but rather on the extreme values of the data in each dimension and the size of the local domains, stipulated as input. We list a call to a function `FGT(Z,G)` to represent the computation of the vorticity values induced by particles located at positions `Z` with circulation weights `G`. This method can be an implementation of a fast summation algorithm such as the Fast Gauss Transform, as discussed in Section 5.2.

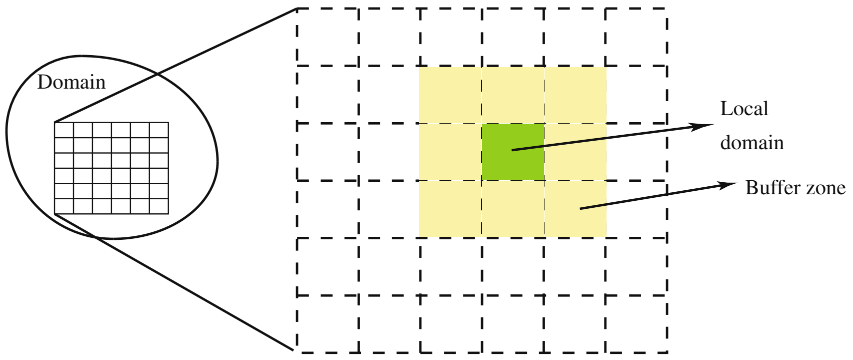


Fig. 1. Illustration of the spatial decomposition used in the method of localization with iteration, using a buffer zone, as explained in the text.

Algorithm 1. RBF solution by localization and iteration

```
Require: V {Exact value of the vorticity at particle locations},
         Z {Location of the Gaussian particles},
         I {The number of iterations to be performed by the algorithm},
         $\sigma, h$ {Size of the Gaussian particles, and their separation parameter},
         sizeLD {The multiple of $\sigma$ defining the linear size of local domains},
         d {Dimension of the data, 2 in our case}
Ensure: G {The weight (circulation) of the particles}
1:  G ← V × $h^{d}$ {Get the initial guess}
2:  [LD, Buffer, K] ← generateLocalDomains($\sigma$, Z, sizeLD, d) {Build the local domains LD with the data Z,
    defining the size of each local domain with sizeLD and d, and get the list Buffer of buffer domains. K is the
    resulting number of local domains.}
3:  for n = 1:I do
4:    Ve ← FGT(Z, G) {Compute the vorticity induced by the particles with circulation G}
5:    fori = 1:K do
6:      LDi_Buffer ← [LD{i}; LD{Buffer{i}}] {Temporarily store the particles in the local domain i and the
    particles in its buffer layer}
7:      [A, M] = buildA(LDiBuffer) {Build the matrix A of interaction among the particles in LDi_Buffer and the
      preconditioner M to be used for solution of the local system}
8:      b = V(LDi_Buffer)-Ve(LCi_Buffer) +A·G(LCi_Buffer) {Construction of RHS based on Eq. (19)}
9:      x = gmres(A, b, M) {Solve the linear system $M^{1}Ax = M^{1}b$ using the gmres iterative method}
10:     iG(LD{i}) = recoverParticlesAtLDi(x) {Recover the circulation values associated to the particles that
    are in local domain i ignoring particles that are in the buffer layer}
11:   end for
12:   G ← iG {Update the circulation values}
13: end for
```

4. Numerical experiments

4.1. Experiments with a 1D test function

We begin with a demonstration of the main ideas presented on a one-dimensional test function. The test function used is plotted in Fig. 2, and it was formed with a number of shifted Gaussians of varying width and height, resulting in a function with support in the domain $[-1,1]$.

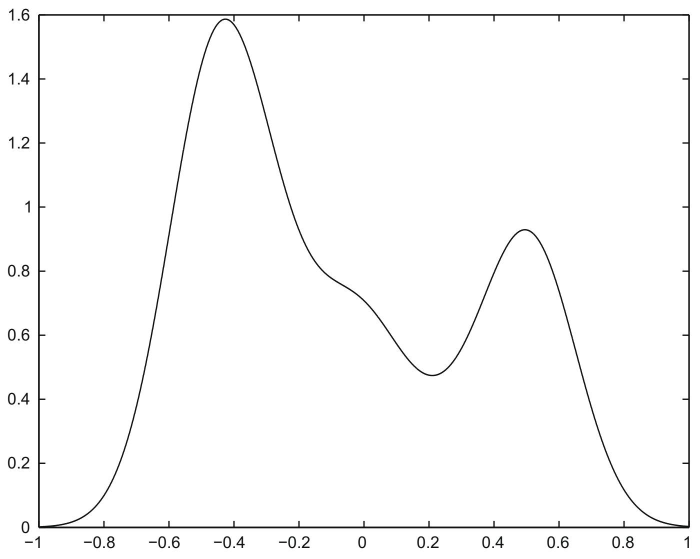


Fig. 2. The test function in one dimension.

First, we demonstrate the simplest localization idea, which solves many small local systems, and superposes the local solutions. This of course does not work, and as seen on Fig. 3 results in γ values which reconstruct the function with larger error than the simple estimation described in Section 3.3. The error in this very small 1D test is $\mathcal{O}(1)$. Note that in Fig. 3 the border between the local domains is shown on the top horizontal axis of the plot by diamond markers. There are 20 local domains, of size 0.1 each, and 51 equally spaced centers for the "particles". The one-dimensional Gaussian bases at each center have a scale of $\sigma = 0.05$ and so the ratio between their spacing and scale is $h/\sigma = 0.04/0.05 = 0.8$

Next, we demonstrate the second approach discussed in Section 3.1, and expressed by the iterative algorithm (18). In this case, we are solving small local problems, but the long-range effect of the other local domains enters on the right-hand-side of the system, as described in Section 3.1. The method starts with an initial guess, which in this case is obtained by (20). Fig. 4 shows the spatial error of interpolation on our 1D test function, for five consecutive iterations of this method. It can be seen that the iterations do not improve on the initial guess in any way. In fact, if one continues iterating, the solution eventually diverges, as shown in Fig. 5.

Finally, we demonstrate the third and last method developed in Section 3.1 — the complete method of localization and iteration with buffer domains. The algorithm is expressed in Eq. (19), and iterates on an initial guess, solving local problems with a buffer layer. It was left pending in Section 3.1 that we investigate the size of the buffer layer that will produce convergence of the method. Consider first using local domains of length 2σ and buffer domains of the same length. The result on our 1D test function is shown in Fig. 6, where we can see that the error is reduced considerably after 25 iterations, with respect to the initial guess. The convergence of the solution is rather slow, however, as shown in Fig. 7. To convince ourselves that the method *does* indeed converge to a good solution, the experiment was continued to 160 iterations. As shown in Fig. 8, the error reaches machine precision eventually.

Increasing the size of the buffer layer around each local domain has the effect of speeding convergence of the iterations. This is illustrated in the next experiment, where the same 1D test function is interpolated using more particles this time, with smaller spread. On the previous examples, the Gaussian bases used had $\sigma = 0.05$, and now we use $\sigma = 0.02$ resulting in 126 particles on the 1D domain. With local domains and buffer domains of length equal to 10σ each, the iterative method improves on the initial guess very rapidly. Fig. 9 shows the spatial error of the interpolation in this case, and Fig. 10 shows the convergence as indicated by the L^2-norm error at each iteration. The method provides close to machine precision after about 10 iterations.

All the experiments presented in this section were realized using MATLAB, and the local systems were solved using the built-in backslash operator, \.

4.2. Experiments with a Lamb–Oseen vorticity distribution in 2D

The Lamb–Oseen vortex is an analytical solution of the 2D Navier–Stokes equations, and it is often used to verify vortex codes. We use this axisymmetric vorticity distribution as the first test case in two dimensions. The vorticity is given by:

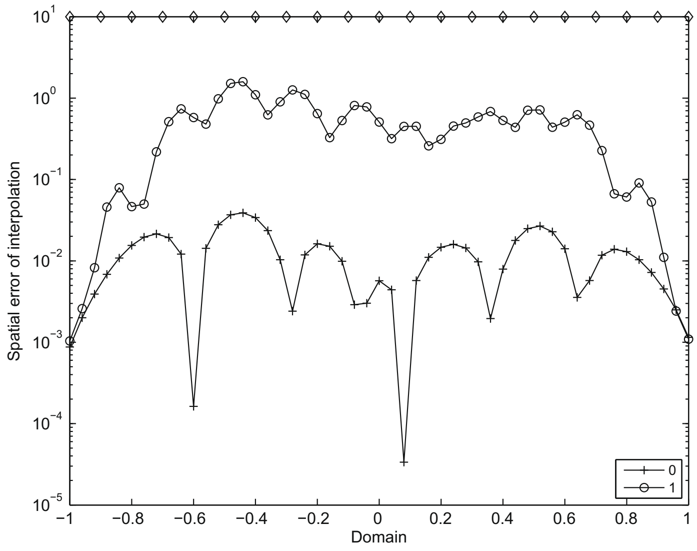


Fig. 3. 1D spatial error of interpolation using the test function of Fig. 2, with 51 Gaussian particles, $\sigma = 0.05$, overlap $h/\sigma = 0.8$. The particle coefficients are found solving 20 local problems, with local domain size $\approx 2\sigma$. The diamond markers on the top horizontal axis indicate the point that separates one block from the next. The error of the initial guess is also shown. Solving local problems does not work, because when added, the long-range effects spoil the local solutions.

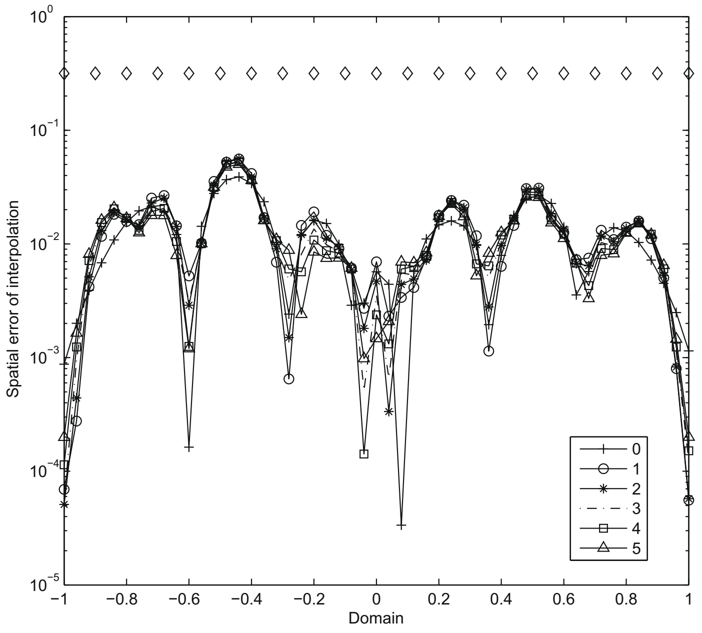


Fig. 4. 1D spatial error of interpolation using the method of localization with iterations, as described in the text. Here, 20 local domains (whose edges are indicated by the diamond markers on the top axis) were used, with 51 particles, $\sigma = 0.05, h/\sigma = 0.8$. This method does not produce convergence.

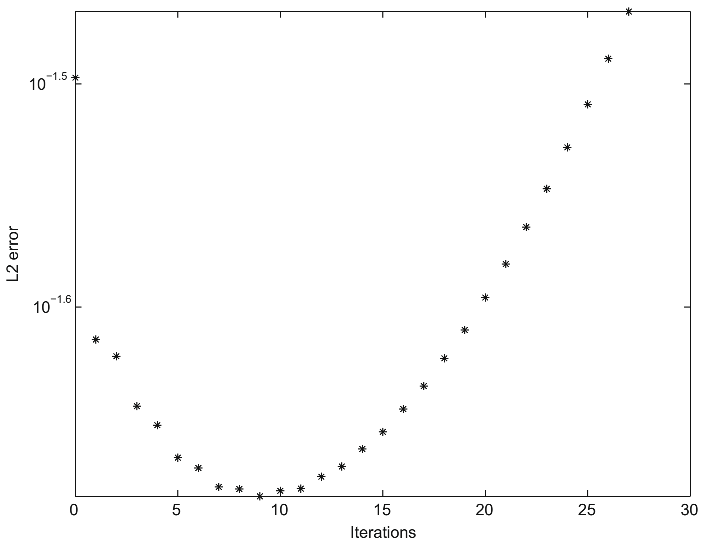


Fig. 5. L^2-norm error of interpolation of the 1D test function using the method of localization with iterations. The spatial error of the first 5 iterations is shown in Fig. 4. It can be seen that after about 10 iterations, the solution starts diverging.

$$\omega(r,t) = \frac{\Gamma_0}{4\pi\nu t}\exp\left(-\frac{r^2}{4\nu t}\right) \tag{21}$$

Here, the parameter Γ_0 corresponds to the total circulation of the vortex and ν is the viscosity of the fluid. The solution corresponds to a spreading vorticity distribution, subject to diffusion effects only. Our test case is a Lamb–Oseen vortex centered at the origin, in a domain of size $[-1,1]^2$. The parameters are: $\Gamma_0 = 1,\ \nu = 0.1$ and $t = 1$. With these parameters, the vorticity distribution is plotted in Fig. 11(a). The experiment consists of laying Gaussian particles of spread $\sigma = 0.02$, with an overlap $h/\sigma = 0.8$ on the two-dimensional domain. The resulting number of particles is $N = \left(\frac{2}{h}\right)^2 = 15,625$, which would be prohibitive for a global solution of the interpolation problem — especially in terms of memory requirements, if building the coefficient matrix. We use the method of localization with iterations and buffer layer, with a local domain length in each direction of 10σ. With this, we estimate that each local domain has ≈156 particles. The buffer domains

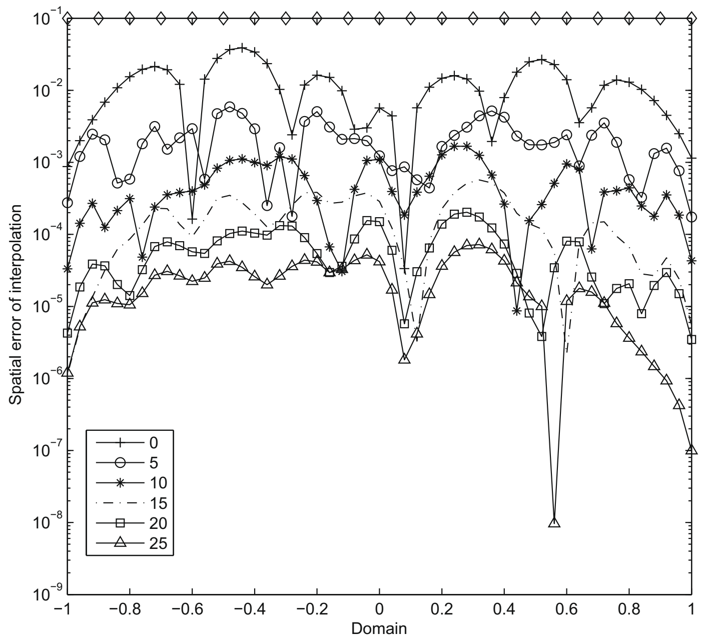


Fig. 6. Spatial error of interpolation of the 1D test function, using the method of localization with iterations and buffer layer. As in the previous examples, the approximation uses 51 Gaussian particles with $\sigma = 0.05$, $h/\sigma = 0.8$; the length of the local domains and buffer domains is $\approx 2\sigma$.

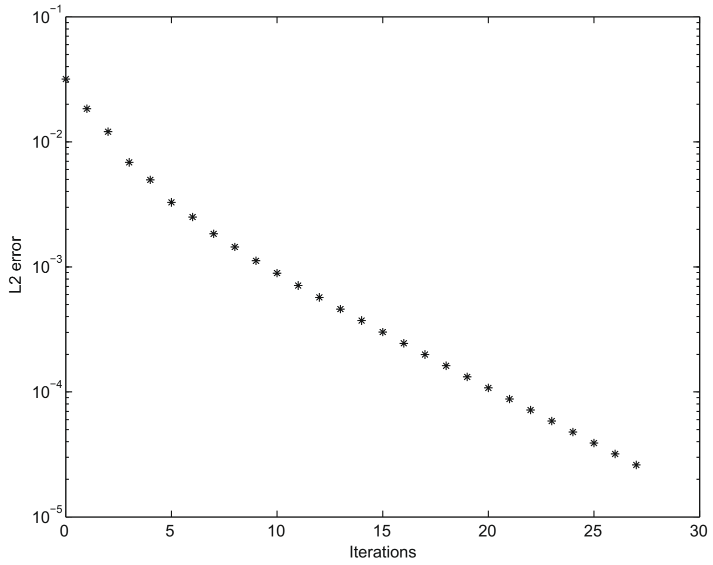


Fig. 7. L^2-norm error of interpolation of the 1D test function using the method of localization with iterations and buffer layer. Parameters as in the caption of Fig. 6.

are of the same width in each direction as the dimensions of the local domain. Therefore, on average each local problem involves the solution of a linear system of size 1404×1404. Fig. 11(b) shows the solution obtained after 9 full iterations of the method, when convergence is achieved. Fig. 12 shows the convergence of the method, in terms of the L^2-norm error over iterations.

In Figs. 13 and 14, the logarithm of the point-wise error of vorticity (normalized by the maximum vorticity) is plotted in the 2D domain for consecutive iterations. The initial guess, using Eq. (20), results in an error of $\mathcal{O}(10^{-2})$. The iterative method improves on the initial solution very quickly, with an error of $\mathcal{O}(10^{-12})$ after 8 iterations.

In this experiment, each local domain (with its buffer particles) is solved using a preconditioned GMRES iterative method. The sparse preconditioner used is that described in Section 2, using as sparsity criterion a tolerance level for the matrix

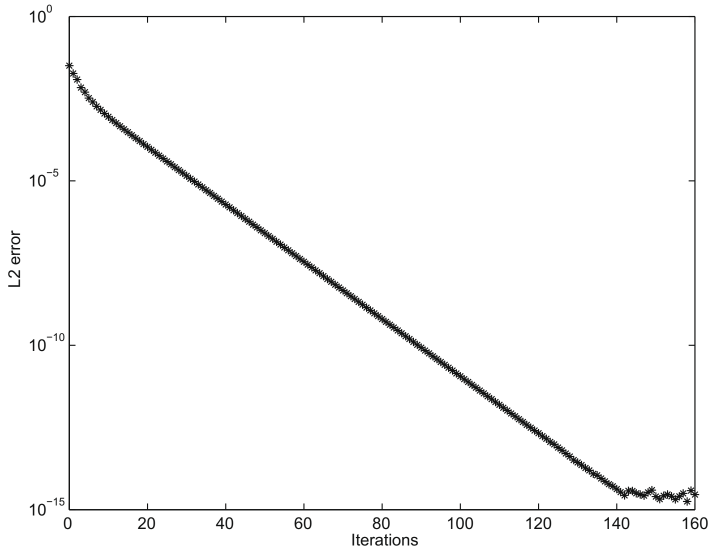


Fig. 8. L^2-norm error of interpolation of the 1D test function using the method of localization with iterations and buffer layer. Parameters as in the caption of Fig. 6.

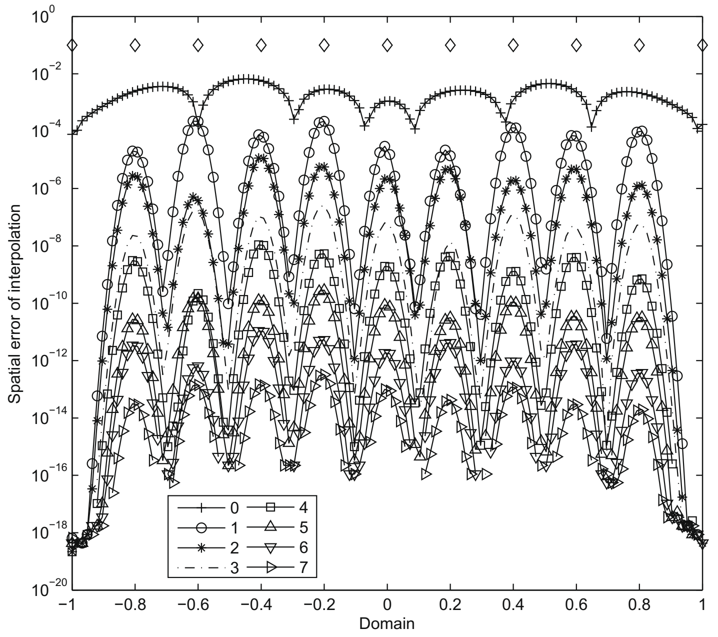


Fig. 9. Spatial error of interpolation of the 1D test function, using the method of localization with iterations and buffer layer. Interpolation with 126 particles, $\sigma = 0.02,\ h/\sigma = 0.8$ and buffer size 10σ.

values of 10^{-6}. The GMRES was forced to exit at 10 iterations, as these local solutions will be improved on with the global iterations.

It was mentioned in passing before that the method of localization does not have any constraints in terms of the geometry. Certainly, the local domains do not have to be rectangular. To demonstrate the method in the context of different geometry features, we now present an experiment using the same Lamb–Oseen initial condition, but with a clustering algorithm to obtain the local domains. The clustering algorithm used is the k-means method [6]. It works by first choosing k cluster centers randomly among the source data, and subsequently assigning each other data point to the cluster center that is closest to it. Iteratively, the centers are updated by taking the average of all the data points in its cluster, until the algorithm converges.

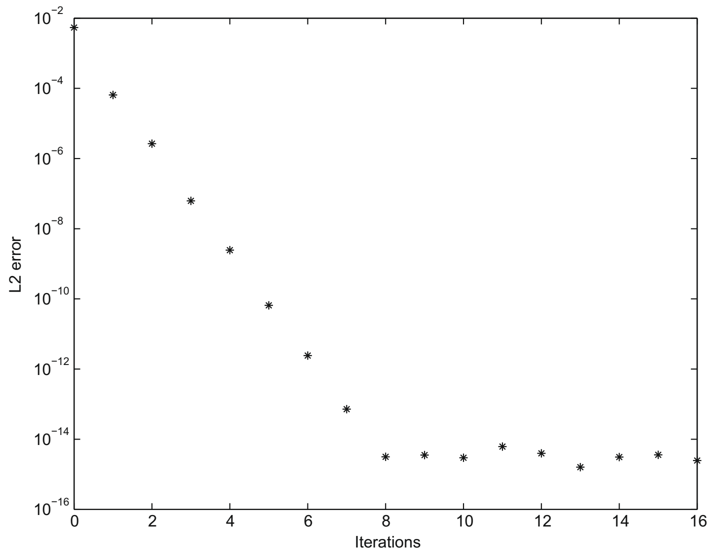


Fig. 10. L^2-norm error of interpolation of the 1D test function, using the method of localization with iterations and buffer layer. Interpolation with 126 particles, $\sigma = 0.02$, overlap = 0.8 and buffer size 10σ.

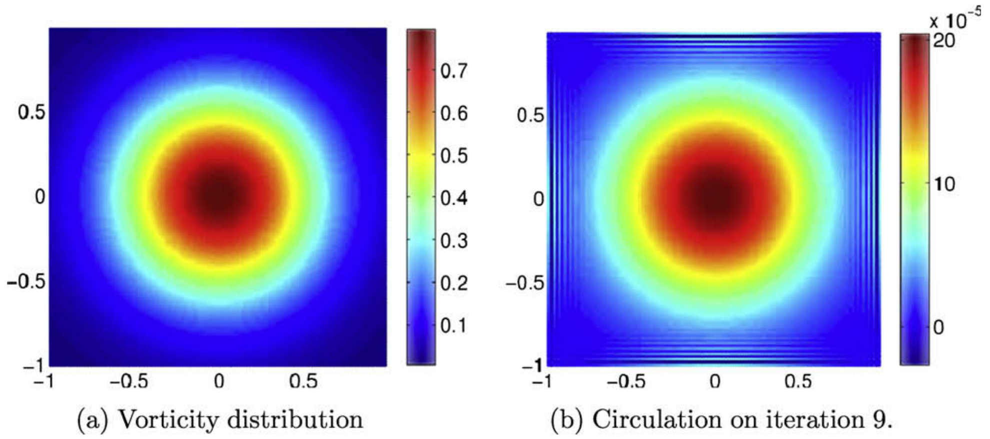


Fig. 11. The two-dimensional test problem: a Lamb–Oseen vortex, with parameters as given in the text. (For interpretation to colours in this figure, the reader is referred to the web version of this paper.)

The experiment is set up so that the number of clusters coincides with the number of square local domains that were used in the previous experiment, shown in Figs. 13 and 14. The neighbor clusters that will constitute the buffer layer of each local domain are chosen as those whose center is a distance of 10σ or less from the center of the local domain cluster. All particle parameters were the same as for the experiment with the square local domains of size 10σ, shown above. The spatial distribution of the errors of interpolation obtained with the k-means clusters over a sample of iterations is shown in Figs. 15 and 16. It can be appreciated that the method still converges with the irregular clusters, but the convergence in this experiment is slightly slower than with square local domains. Nevertheless, we still reach close to machine precision with excellent algorithmic efficiency, see Fig. 17 .

4.3. Experiments with a physically relevant 2D vortex flow

The results presented in the previous section are very impressive, but the Lamb–Oseen vorticity distribution is quite benign in the sense that it is very smooth, simple and has no interesting features. We will now use as test case a vorticity distribution which has been obtained by a vortex method calculation of a flow with physically relevant features. The vortex code used to evolve this flow is the same as was used in the study of vortex tripoles in [1]. The actual description of this fluid situation is not really relevant to this paper, but its features are common in vortex flows, including concentrations of vorticity and filaments. The vorticity/circulation field is shown in Fig. 18; we will call this case "dipoles" just to give it a name.

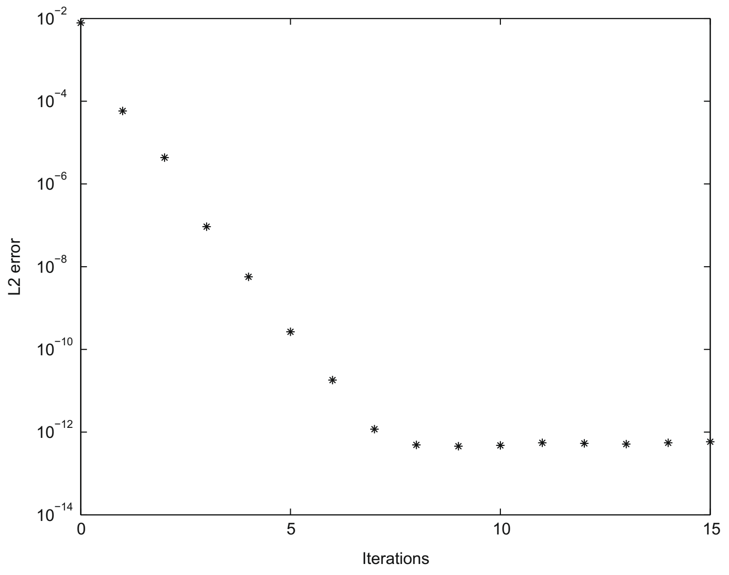


Fig. 12. L^2-norm error over iterations when interpolating the vorticity of the Lamb–Oseen test problem, with 15,625 particles, $\sigma = 0.02$, $h/\sigma = 0.8$, local domain and buffer length is 10σ to each side, and the domain is $[-1,1]^2$. Each local system is solved using a preconditioned GMRES iterative method.

For this test case, we have used N = 71,289 Gaussian particles of spread $\sigma = 0.11278$ and overlap $h/\sigma = 0.8$. The domain is $[-12,12]^2$, and the number of particles in a local domain is $n = \left(\frac{10}{0.8}\right)^2 \approx 156$ on average. There are a total of 441 (21×21) local domains. With a buffer layer which is formed by buffer domains of the same size as the local domain, of length 10σ in each direction, the local systems being solved have $9 \times 156 = 1404$ unknowns. These systems are not that small, and we solve them using the GMRES iterative method, with the preconditioner presented in (12). Instead of using a distance threshold for the sparse preconditioner, we used a threshold in the matrix entry itself, making it zero if it was smaller than 10^{-6}.

Fig. 19 shows that the method again converges very rapidly to high accuracy. Figs. 20 and 21 present the spatial distribution of the error of interpolation for this case, for iterations from 0 to 9. We note that, in all cases, the maximum of the errors occur on and around the boundaries of the local domains. To illustrate this more clearly, Fig. 22 shows a close-up over a region covering only 9 local domains, around the origin of the coordinate system. This close-up corresponds to iteration 9, with the full domain error field shown in Fig. 21(d).

To conclude this subsection, we include a plot showing the *difference* in circulation between the initial guess and the final iteration for this experiment, see Fig. 23. We see that the iterative method has worked harder in the area around the filaments of vorticity, producing a tightening of the field which was generated by the initial guess. As discussed in Section 3.3, the initial guess used is equivalent to a Gaussian blurring of the original field. Therefore, a more accurate solution of the RBF interpolation problem counteracts this blurring by increasing the definition near features of the field, in this case the filaments of vorticity.

4.4. Experimental study of the convergence with respect to parameters

There are several parameters which affect not only the convergence of our method but the accuracy which can be achieved by the RBF representation. For interpolation accuracy, one crucial parameter is the overlap of the smooth basis functions, *i.e.* for Gaussian bases, the ratio h/σ. Theoretical studies of the convergence of the vortex method in a time marching calculation have relied on the assumption that $h/\sigma < 1$. At the same time, numerical experiments have demonstrated that the quality of the approximation using Gaussian bases converges super-exponentially with particle overlap [2]. As the particles overlap more, there is an increase in the quality of the approximation. But there is also an increase in the ill-conditioning of the RBF interpolation problem. We anticipate that this parameter will influence the rate of convergence of our iterative method.

The second parameter of importance for the convergence of our method is the size of the local domain, which has to be at least a few σ's wide in each direction. Therefore, with the local domain size measured in multiples of σ and the value of the overlap, h/σ, we performed a number of experiments combining different values of these parameters (the length of the buffer layer was chosen equal to the local domain size, for simplicity). For each experiment, we measure the *slope* of the L^2 error of consecutive iterations, and we have plotted this measure in a color map; see Fig. 24. Table 1 shows the actual values of the slope for the different calculations. As could be expected, the fastest convergence is observed for the widest local domains (of length 12σ in each direction) and the largest overlap ratio (representing *less* particle overlap, and therefore better

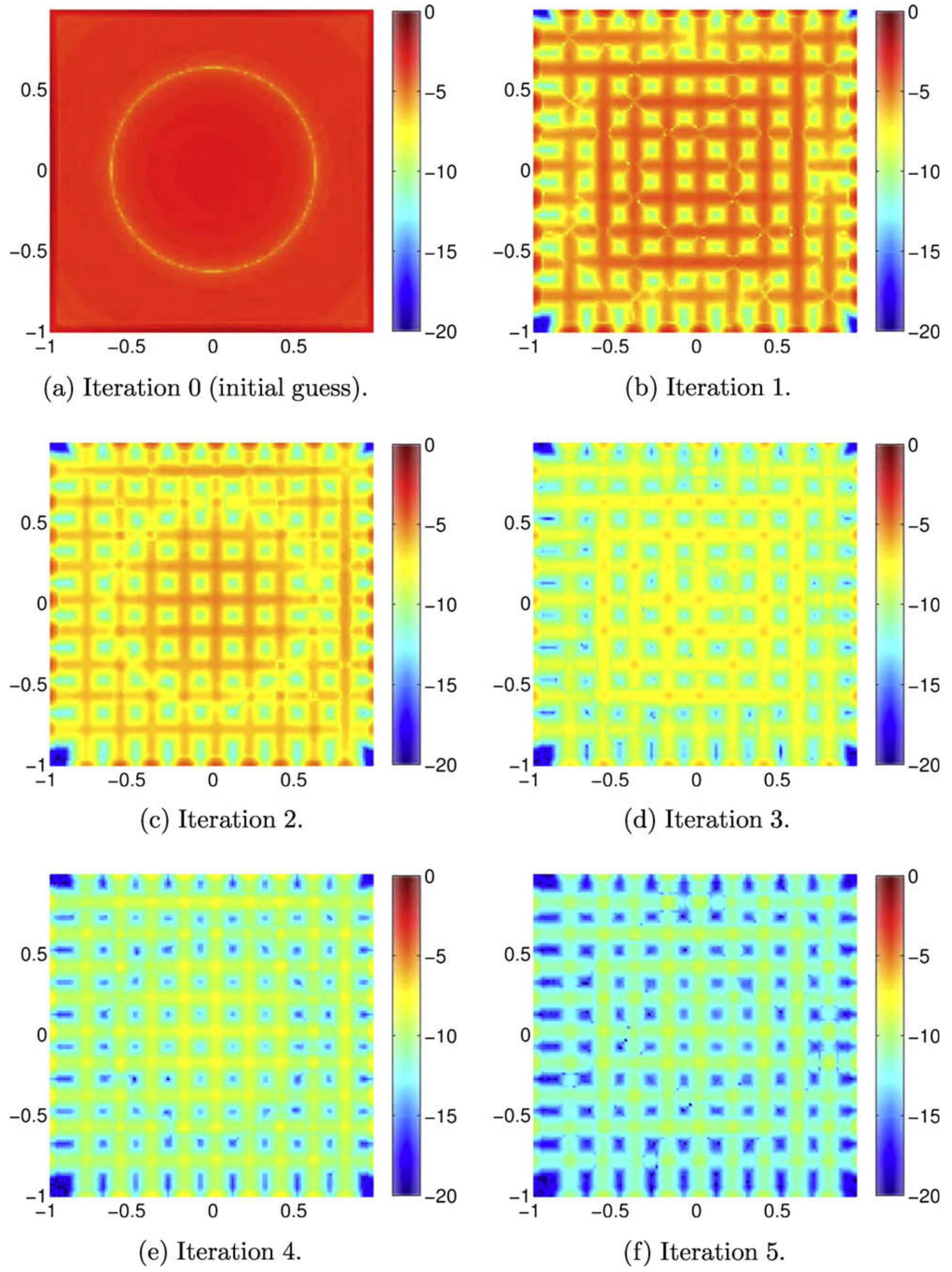


Fig. 13. Spatial error of the interpolation over consecutive iterations of the method of solution using localization and iteration. The color map shows the logarithm of the absolute value error in the vorticity field (normalized by maximum vorticity). Parameters are indicated in the caption of Fig. 12. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

conditioning of the linear systems). The worse rate of convergence is obtained with local domains of length 6σ in each direction and $h/\sigma = 0.7$. Even in this case, convergence is achieved, with a rate given by a slope of -0.345 in the logarithm of the L^2 error.

5. Complexity study and implementation details

We have given ample illustration of the capability of this method for solving large RBF interpolation problems with Gaussian bases of small spread (as required in the vortex particle method). But the reader may wonder if the method is computationally efficient. The calculations shown in the previous section, one with more than 70 thousand particles, were carried out using 1 core of a high-end desktop computer (3GHz Intel Xeon Mac Pro). Hence, ‘production level’ calculations

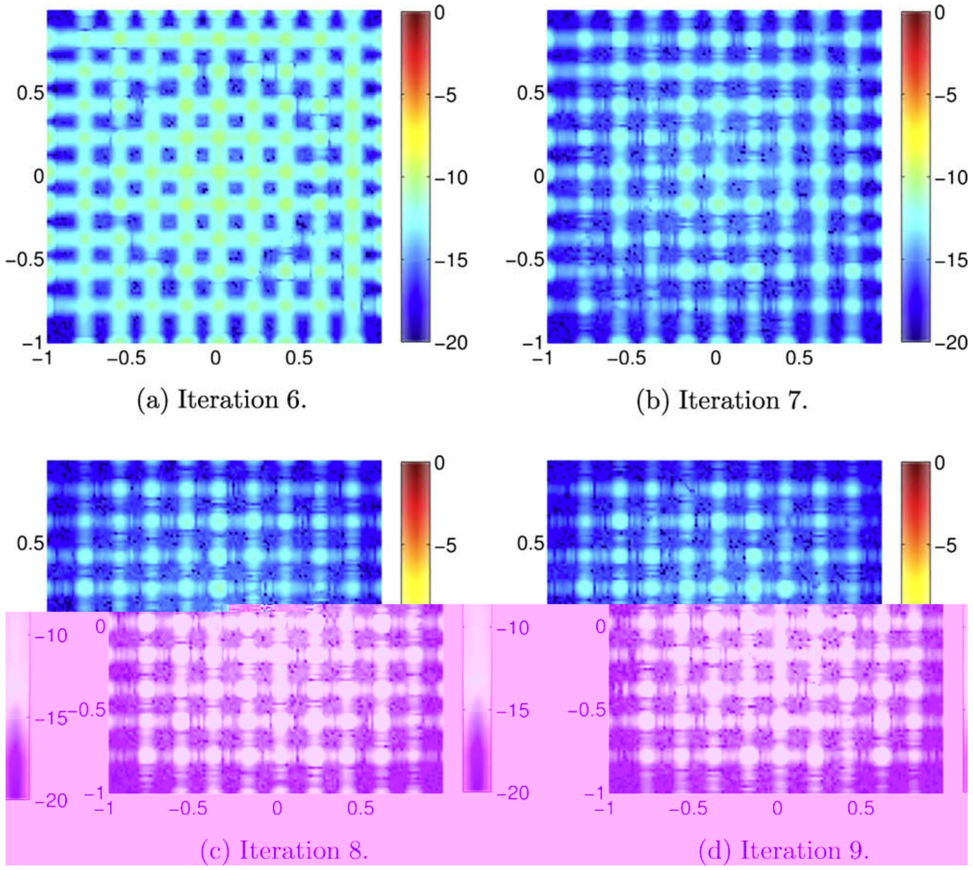


Fig. 14. Continued from Fig. 13.

are possible using modest computational resources. So far, we have only developed prototype codes, using MATLAB; an implementation in a compiled language, perhaps in parallel, would easily provide the capability to solve for millions of unknowns. Recall that the algorithm is highly parallel, as each local system can be solved independently at each iteration. But the important question remains of how the algorithm scales with the number of particles, N.

In this section, we analyze the computational complexity of the algorithm, and present numerical demonstration of the observed scaling with problem size. To start, we can identify three algorithmic components:

I Generation of the local domains and assignment of particles.
II Evaluation of the radial basis function summation at all centers.
III Solution of the radial basis function interpolation on the local domains.

For these algorithmic components, first we discuss the complexity that can be obtained in theory, and subsequently we give some experimental demonstrations.

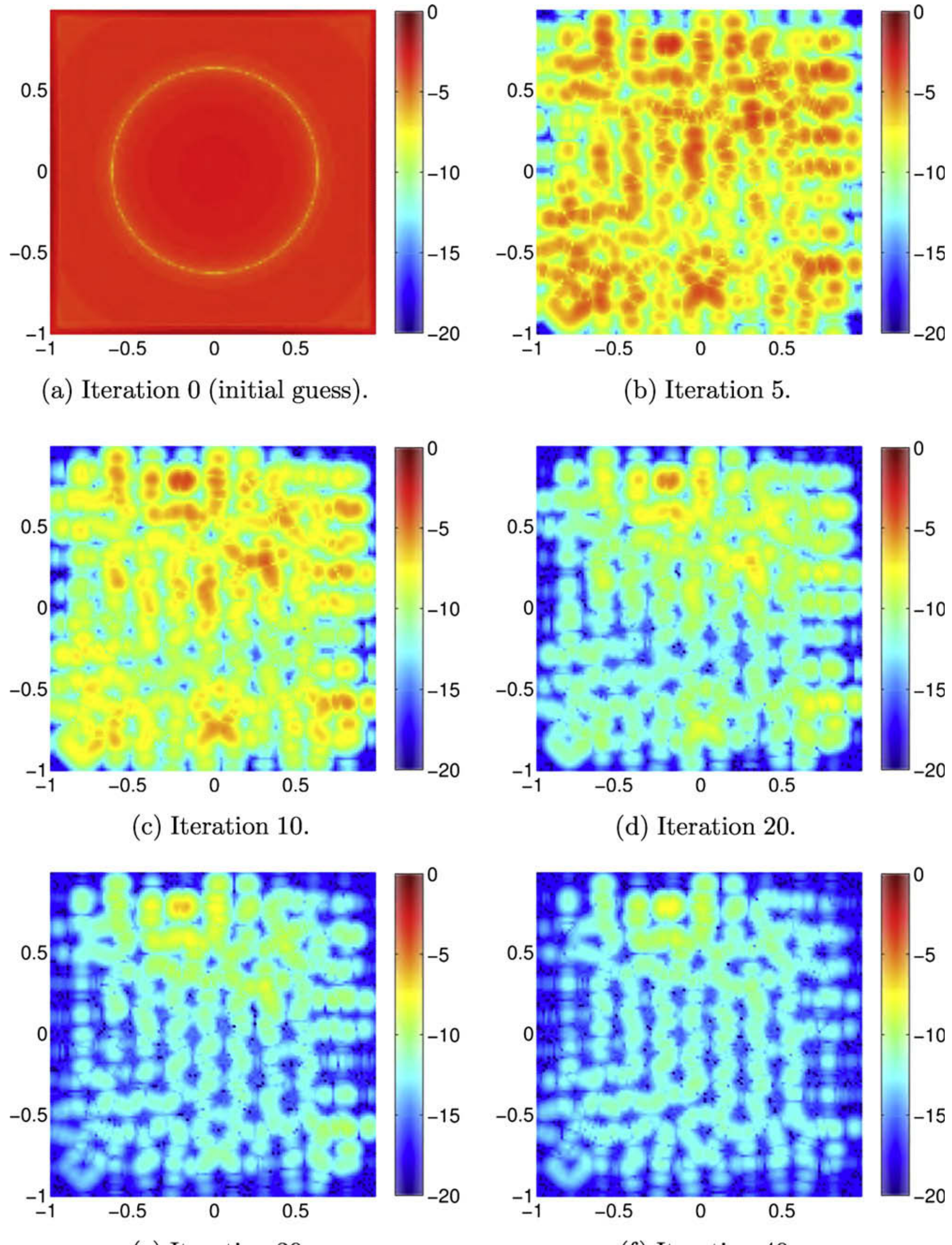
Note that the algorithm iterates over components *II* and *III*, but we have shown previously that the convergence is fast. The number of iterations to converge will depend on the initial guess, but the rate of convergence depends only on the size of the local domains and the particle overlap, h/σ, and does not depend on N.

Let us define the variables that we will use in the complexity analysis:

N	number of particles in the global domain
n	number of particles per local domain
K	number of local domains
d	dimension of the domain

5.1. Generation of the local domains

Component *I* of the algorithm corresponds to the generation of the local domains from an unordered set of locations for the data. The problem is analogous to reordering the set of indices that identify the locations of the data. It is possible to



(a) Iteration 0 (initial guess).

(b) Iteration 5.

(c) Iteration 10.

(d) Iteration 20.

(e) Iteration 30.

(f) Iteration 40.

Fig. 15. Spatial error of the interpolation over a sample of iterations of the method of solution using localization and iteration with a buffer zone; this time the local domains are generated using the *k*-means clustering algorithm. The color map shows the logarithm of the absolute value error in the vorticity field. Particle parameters indicated in the caption of Fig. 12. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

implement a method that assigns particles to local domains in the shape of cells with $\mathcal{O}(N)$ operations, using a space-filling curve or *z*-order and hashing functions (geometric hashing). Certainly, the *k*-means clustering algorithm is $\mathcal{O}(N)$. We have implemented a more simple method for space partitioning into cells, which is described below.

For simplicity, we restrict the discussion here to a square domain in two dimensions. Consider an iterative sub-division of the domain, applied first in one linear dimension, and subsequently in the second linear dimension (in three dimensions, it would be applied a third time, clearly without affecting the numerical complexity). Using the first linear dimension (say, the horizontal), the domain is divided in two sections; all particle locations are compared with the limit between the sections and assigned to one or the other in $\mathcal{O}(N)$. The two sections are divided again in two parts each; all particles in Section 1 will be compared with its dividing limit and assigned to one subsection or the other in $\frac{N}{2}$ operations, and the same is true of

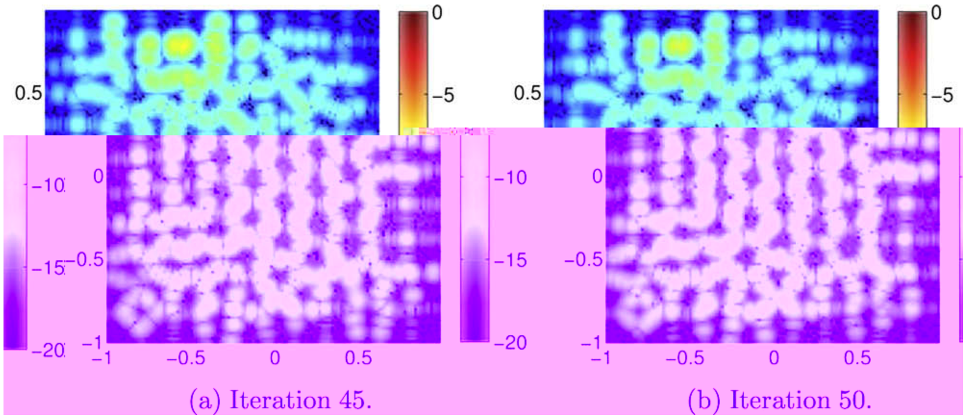


Fig. 16. Continued from Fig. 15.

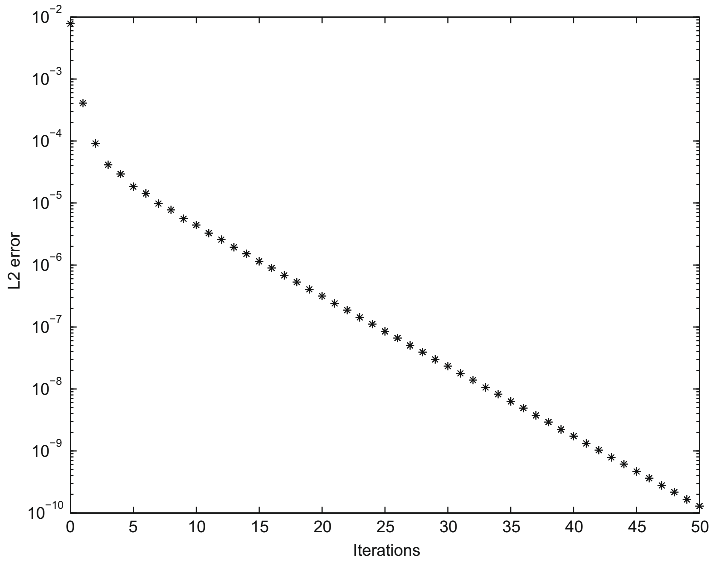


Fig. 17. L^2-norm error over iterations when interpolating the vorticity of the Lamb–Oseen test problem with k-means clustering. The spatial distribution of the errors is shown in Figs. 15 and 16. Convergence is fast, but slightly slower than with square local domains.

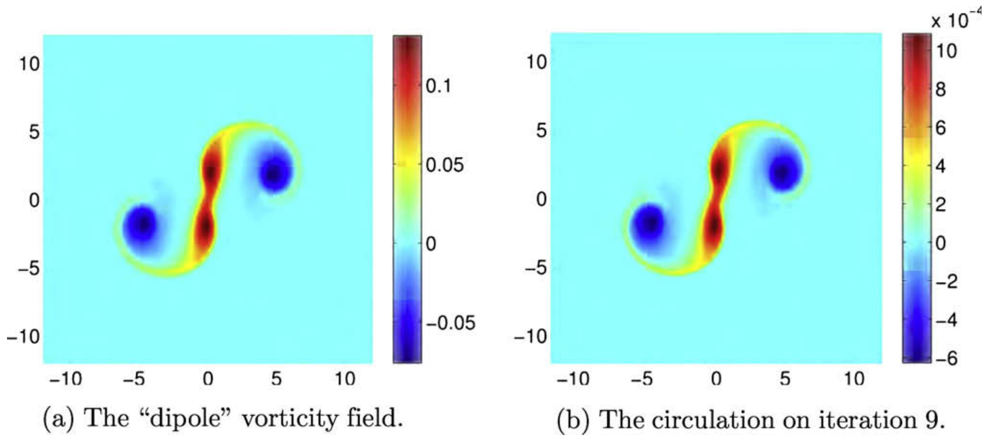


Fig. 18. Vorticity field used for the final experiments, and circulation obtained by the iterative method.

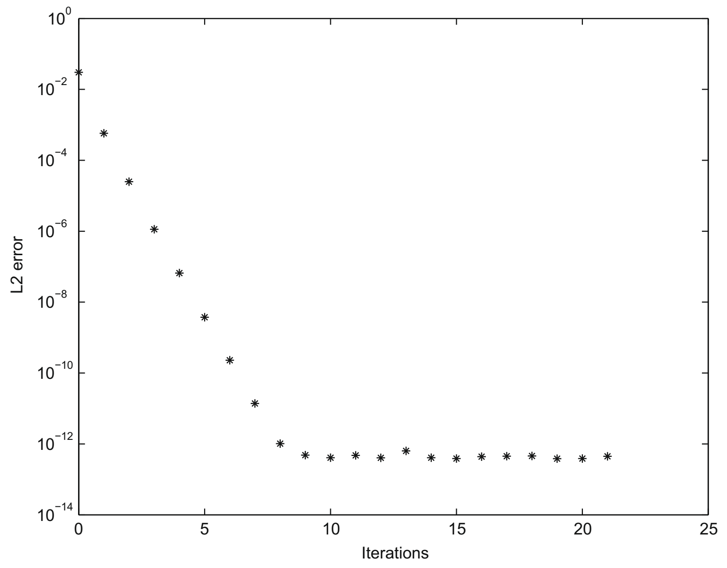


Fig. 19. L^2-norm error over iterations when interpolating the vorticity of the dipoles, with 71,289 particles, $\sigma = 0.11278$, overlap = 0.8, buffer length is 10σ to each side, and the domain is $[-12, 12]^2$.

Section 2. Successively, each subsection is divided again in two. Assignment of the particle locations to each sub-division is done in $\mathcal{O}(N)$, for each level of sub-division, until level l. When the process is finished, there will be approximately $N/2^l$ particles in each domain section, assuming they are regularly distributed. When the sub-division of the domain is performed in the other linear dimension, the local domains are finally obtained, with on average n particles contained in each of them.

Consider that the N particles are regularly distributed in the square domain, on a lattice of separation h in each direction. This means that there are $\sqrt{N}$ particles per side of the domain. After the sub-divisions in the first linear dimension, there will be rectangular sections which have $\sqrt{N} \times \sqrt{n}$ particles each; see Fig. 25. Together with the discussion in the previous paragraph, this results in $N/2^l = \sqrt{Nn}$, which can be solved for l giving $l = \frac{1}{2}\log_2(N/n)$. Finally, the work needed to complete the process is N comparisons l times, which is $\sim \frac{N}{2}\log_2(N/n)$. This will be repeated for each linear dimension. With n much smaller than N and in fact approximately constant, given that the size of local domains is defined as a multiple of the particle size, the asymptotic behavior for this part of the algorithm is $\mathcal{O}(N \log N)$. We repeat that this is a first implementation, useful for our purposes but rather naive. Methods to produce a similar result in $\mathcal{O}(N)$ are known, but we have not implemented one. We will see that this part of the algorithm does not dominate the total computational time, so a simple method is sufficient.

5.2. Evaluation of the RBF field at all particle locations

A direct summation of the influence of all particles at one evaluation point requires N operations, and thus the direct evaluation at all points is in principle an $\mathcal{O}(N^2)$ calculation. However, fast summation methods for RBF evaluations are well-known. If the basis function has a long-range effect (which would be the case with multi-quadric bases, and with a Coulomb potential, for example), then the fast multipole method (FMM) can be used to evaluate the field in $\mathcal{O}(N)$ operations [12]. For Gaussian bases, which decay so fast that long-range effects can be neglected entirely, a specialization of the FMM has been developed, the Fast Gauss Transform (FGT), which again accelerates the evaluation to $\mathcal{O}(N)$ [18]. We would not go into the details here; let us just say that with an FGT implementation, the evaluation of the summation of Gaussians is $\mathcal{O}(N)$.

5.3. Solution of the radial basis function interpolation on the local domains

The algorithm described here requires the solution of many small linear systems, corresponding to the local domains with their buffer layer, instead of the global problem. The local systems are of size $n3^d$, assuming that the buffer layers will have the same linear dimensions as the local domain, for simplicity. The number of particles per local domain, n, is approximately constant and not a function of N. This is because the size of local domains is chosen a priori as a function of σ, the spread of the Gaussians, and the particle density is given by the overlap parameter, h/σ. For example, if we choose the length of local domains to be 10σ, and the overlap $h/\sigma = 0.8$, we always obtain $n \approx 156$ (in 2D).

The work required to solve the local systems is, therefore, $(n3^d)^2$ per iteration, using a GMRES iterative method. The number of iterations in these GMRES solves is not important, as the solutions are approximate and outer iterations will ensure the final convergence. We have to solve K such linear systems; therefore the total work is $\sim K(n3^d)^2$. The number of local domains can be approximated by $K = \frac{N}{n}$, the total number of particles divided by the number of particles in local domains. This results in an $\mathcal{O}(N)$ estimate for the final part of the algorithm.

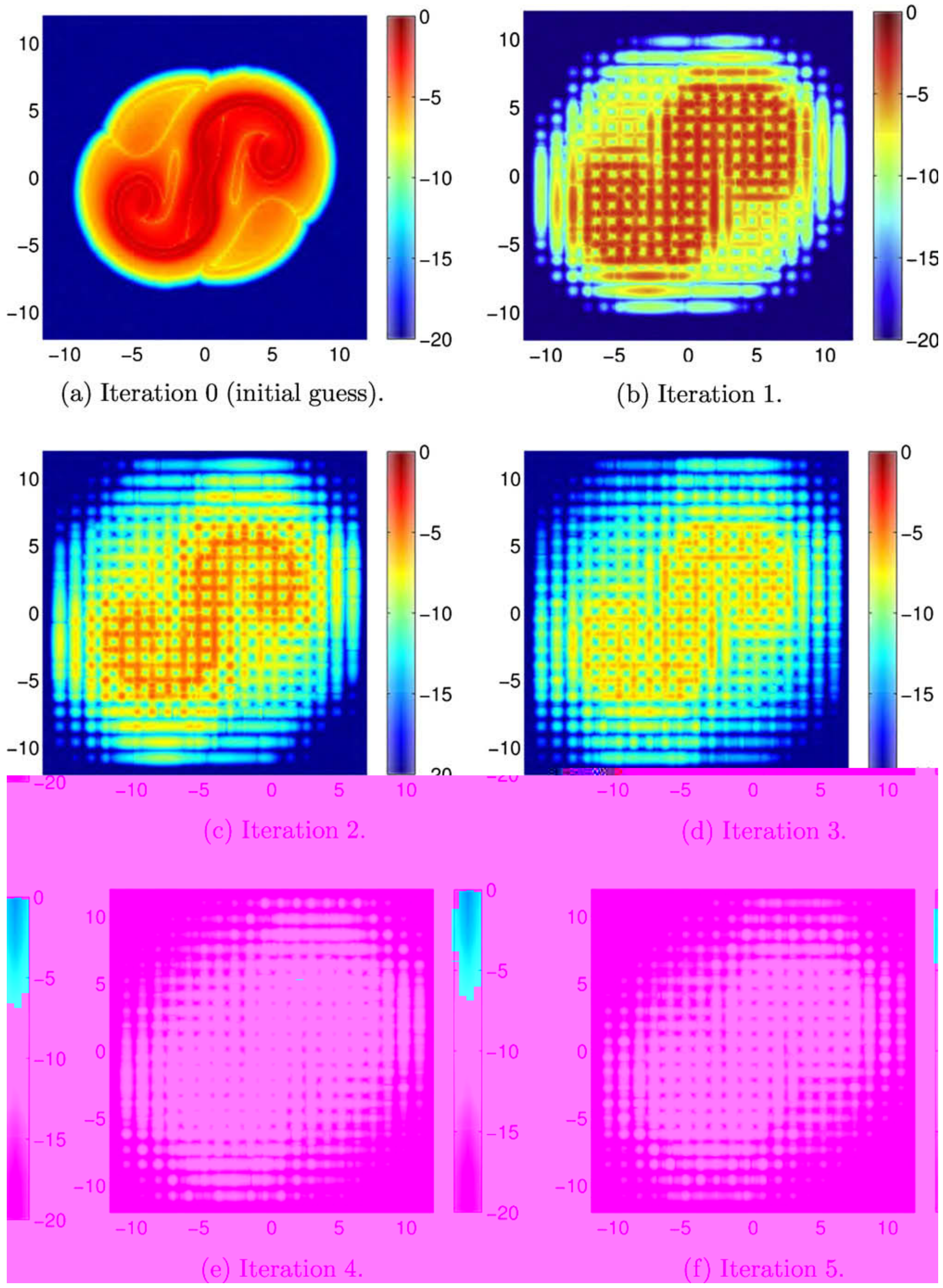


Fig. 20. Spatial error of the interpolation over consecutive iterations of the method of solution using localization and iteration with buffer layer. The color map shows the logarithm of the absolute value error in the vorticity field, normalized by maximum vorticity. Parameters indicated in the caption of Fig. 19. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

5.4. The final complexity

In summary, the three algorithmic components give the following complexity:

I Generation of the local domains — this process can be done in $\mathcal{O}(N)$ but our naive implementation is $\mathcal{O}(N \log N)$.

II Evaluation of the radial basis function summation at all centers — using a known fast summation method, such as FGT, this can be done in $\mathcal{O}(N)$.

III Solution of the radial basis function interpolation on the local domains — this is estimated at $\mathcal{O}(N)$.

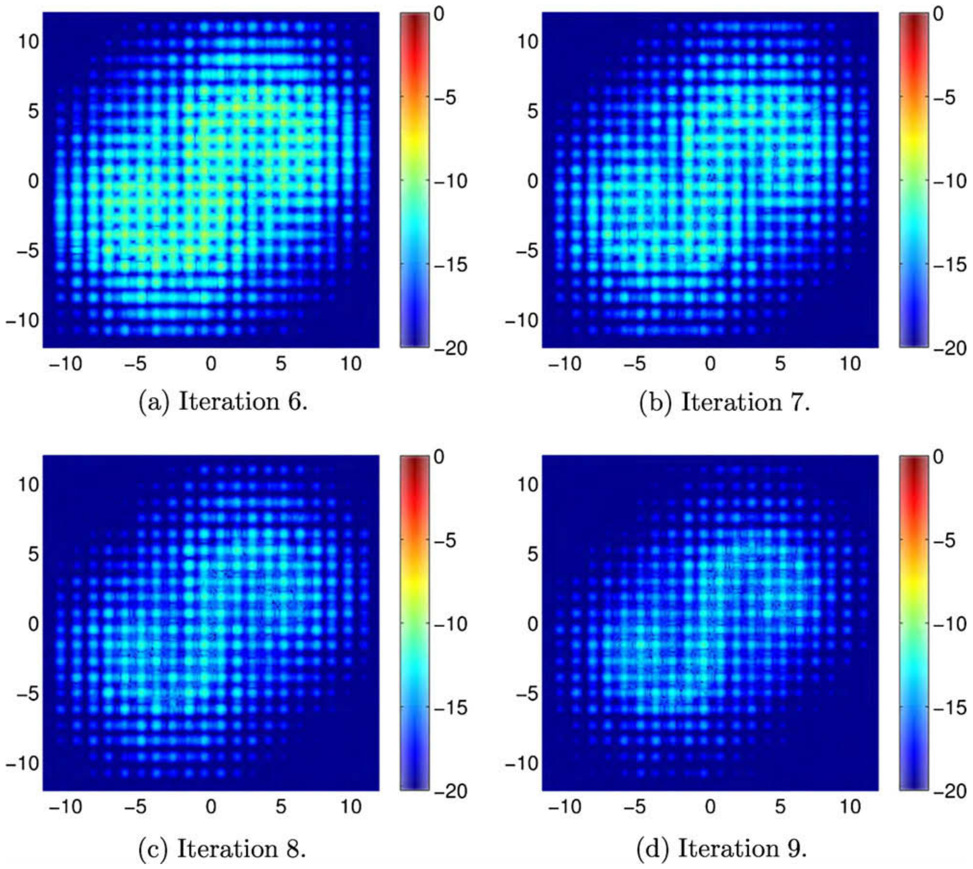


Fig. 21. Continued from Fig. 20.

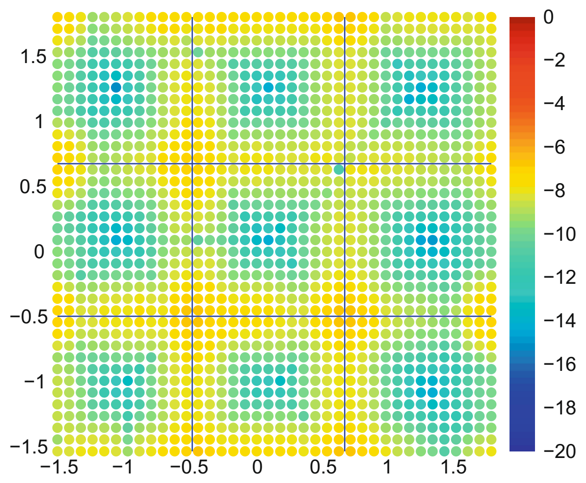


Fig. 22. Zoom-in to an area near the center of Fig. 21(d), showing that the maximum errors occur at the edges and corners of the physical blocks.

The final complexity would seem to be dominated by component I of the algorithm at $\mathcal{O}(N \log N)$, in our implementation. We observe in practice, however, that this part of the algorithm does not dominate the total computational time. At least for the problem sizes we have experimented with, the generation of the local domains is less time consuming than the solution of the linear system iteratively.

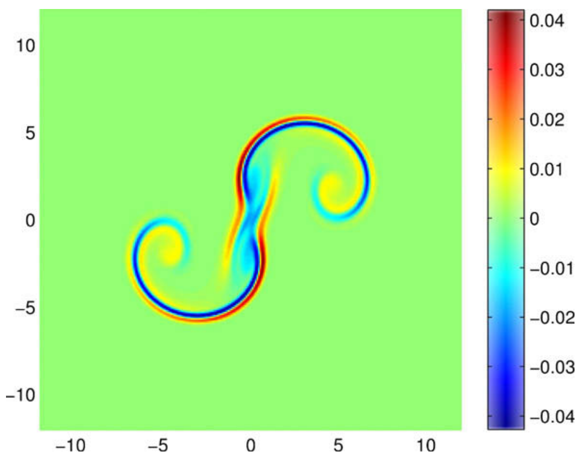


Fig. 23. The difference between the first guess of the circulation or iteration 0 and the last value of the circulation or iteration 9.

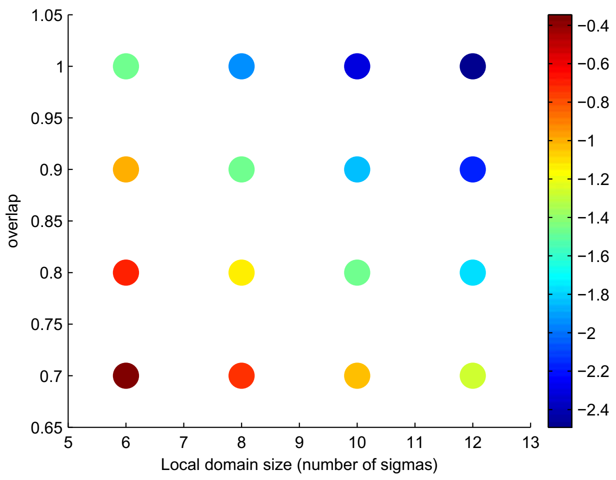


Fig. 24. Parameter study of convergence rate. Each marker represents one calculation; the color maps the value of the slope of the L^2-norm error on a log plot, for the parameters indicated on the axes: size of the local domain in terms of multiples of σ and overlap ratio, h/σ. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 1
Slope of the logarithmic convergence rate, in terms of the L^2-norm error, for combinations of parameters: on the first column is indicated the overlap ratios, h/σ, and on the header row is the width of the local domains in multiples of σ.

	6	8	10	12
1.0	−1.4567	−1.9416	−2.3243	−2.4927
0.9	−1.0003	−1.4581	−1.8280	−2.1811
0.8	−0.6900	−1.1170	−1.4700	−1.7596
0.7	−0.3450	−0.7305	−1.0392	−1.2702

5.5. Complexity observed in numerical experiments

Fig. 26 shows the results of performing various numerical experiments with increasing problem size, *N*. The computational time required for components *I* and *III* of the algorithm is shown. The time for component *II* is not included because we have not at this time implemented a fast summation method; as we have said, known algorithms exist for performing this part in $\mathcal{O}(N)$. As can be seen in the plot, the expected $\mathcal{O}(N)$ complexity is achieved in practice for component *III* of the algorithm, the solution of the linear system iteratively. Component *I* of the algorithm, the generation of the local domains, exhibits a slope of 1.0732. This is close to $\mathcal{O}(N\log N)$ in this range of *N*, but note that the time required for this part of the

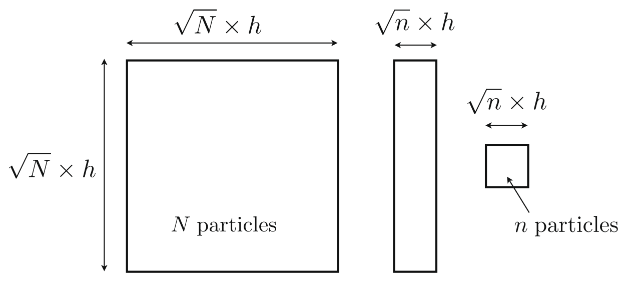


Fig. 25. Sketch showing the whole square domain holding *N* particles, a sub-divided domain in only one linear dimension, and the final local domains holding *n* particles each.

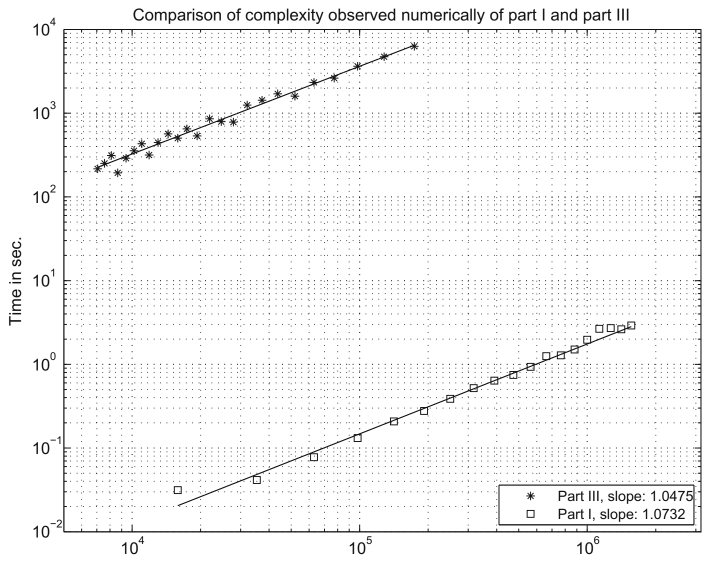


Fig. 26. Observed computational complexity in numerical experiments using local domains of size $12\sigma \times 12\sigma$ in 2D. The slopes obtained from regression show that the expected $\mathcal{O}(N)$ is achieved in practice for component *III* of the algorithm, the solution of the linear system iteratively. Component *I* of the algorithm, the generation of the local domains, exhibits a slope of 1.0732, which is close to $\mathcal{O}(N \log N)$ for this range of *N*, but is not dominant, being several orders of magnitude less time consuming than component *III*.

algorithm is several orders of magnitude smaller than the solution of the RBF interpolation problem on the local domains. Moreover, the generation of local domains needs to be completed only once, while the complete algorithm includes iterations on components *II* and *III*. A more careful study would count floating point operations, rather than report time, but we plan to perform such a study later, when the method has been implemented in a compiled language and in parallel. This is future work.

6. Conclusions

In the vortex method for the solution of the Navier–Stokes equations, the vorticity field of a fluid is represented by a superposition of smooth particles. This amounts to a radial basis function (RBF) interpolation problem, which needs to be solved normally in two stages of a vortex method calculation: at initialization, and after spatial adaptation of the particles using a meshfree method. Mesh-based spatial adaptation schemes exist that do not require RBF interpolation; they are formulated using tensor products and interpolate the circulation (strength) of particles rather than the vorticity field. Although these methods have been used with great success, they do require a regular mesh on the domain and they can introduce some diffusive interpolation errors. RBF interpolation offers a high order of convergence and the potential of high accuracy. On the other hand, it requires the solution of a large, ill-conditioned linear system. In this work, we have presented a method to solve this system when one uses a basis function that decays rapidly away from its center (like the Gaussian). The method consists of the construction of many local domains, where a small system can be solved, but introducing the global influence of the rest of the domain through iterations. The algorithm is highly parallel, because each local system is solved independently within each iteration. Moreover, there is never a need to construct the global coefficient matrix in memory. These features allow one to solve large problems using moderate computational resources.

We have developed the method first using a one-dimensional example, where the importance of considering the global influence in the local problems has been demonstrated. Then we have shown the effect of the iterative approximation to the solution on two-dimensional tests, using both rectangular local domains, as well as irregular cluster-type domains. One test problem presented corresponds to a field of physical significance, where a vortical flow has developed multi-polar structures and filaments. In all cases, the method demonstrates excellent algorithmic efficiency.

Finally, a study of the complexity of the different algorithmic components of the method has shown that it is not expensive computationally. The solution of the many local systems, with a buffer layer, and repeatedly within iterations, sounds like a lot of computational work. In fact, the work scales linearly with the number of unknowns. The evaluation of the sum of basis functions can also be done in $\mathcal{O}(N)$ operations, using for example an implementation of the Fast Gauss Transform. In our implementation of the algorithm, the construction of the local domains is done in $\mathcal{O}(N \log N)$ operations, but this can be improved. There are known methods to produce the geometric division of space in $\mathcal{O}(N)$. We have not implemented one, but we observe that the time required to generate the local domains is nevertheless much smaller than that required by other parts of the algorithm.

In future work, we will proceed with implementing the method in a compiled language, and demonstrating its highly parallel features. A parallel implementation should be able to handle millions of particles easily with modest computational resources, and we aim to demonstrate this capacity.

Note added in proof

A code in Python has been produced that reproduces the Matlab code used in this work. We are making the Python code available to the community, and welcome correspondence from interested users. The code is found at http://code.google.com/p/pyrbf/. We thank Dr Rio Yokota for translating the Matlab code to Python.

Acknowledgments

The authors thank Felipe A. Cruz and L. F. Rossi for helpful discussions. C.E.T. acknowledges support from the SCAT project via EuropeAid contract II-0537-FC-FA, www.scat-alfa.eu, for an extended research visit during which this work was initiated. L.A.B. acknowledges support from EPSRC under grant contract EP/E033083/1.

References

[1] L.A. Barba, A. Leonard, Emergence and evolution of tripole vortices from net-circulation initial conditions, Phys. Fluids 19 (1) (2007) 017101.
[2] L.A. Barba, A. Leonard, C.B. Allen, Advances in viscous vortex methods – meshless spatial adaption based on radial basis function interpolation, Int. J. Numer. Methods Fluids 47 (5) (2005) 387–421.
[3] L.A. Barba, L.F. Rossi, Global field interpolation for particle methods (submitted for publication), <http://people.bu.edu/labarba/pubs/BarbaRossi2008.pdf>.
[4] R.K. Beatson, J.B. Cherrie, C.T. Mouat, Fast fitting of radial basis functions: methods based on preconditioned GMRES iteration, Adv. Comput. Math. 11 (2–3) (1999) 253–270.
[5] M.D. Buhmann, Radial Basis Functions. Theory and Implementations, Cambridge University Press, 2003.
[6] R. Duda, P. Hart, D. Stork, *k*-Means Clustering, Pattern Classification, John Wiley & Sons Inc., 2001. pp. 526–527 (Chapter 10.4.3).
[7] N. Dyn, D. Levin, S. Rippa, Numerical procedures for surface fitting of scattered data by radial functions, SIAM J. Sci. Stat. Comput. 7 (1986) 639–659.
[8] Gregory E. Fasshauer, Meshfree Approximation Methods with MATLAB, World Scientific, 2007.
[9] A.C. Faul, G. Goodsell, M.J.D. Powell, A Krylov subspace algorithm for multiquadric interpolation in many dimensions, IMA J. Numer. Anal. 25 (2005) 1–24.
[10] D. Fishelov, A new vortex scheme for viscous flows, J. Comput. Phys. 86 (1990) 211–224.
[11] R. Franke, Scattered data interpolation: tests of some methods, Math. Comput. 38 (157) (1982) 181–200.
[12] L. Greengard, V. Rokhlin, A fast algorithm for particle simulations, J. Comput. Phys. 73 (2) (1987) 325–348.
[13] N.A. Gumerov, R. Duraiswami, Fast radial basis function interpolation via preconditioned Krylov iteration, SIAM J. Sci. Comput. 29 (5) (2007) 1876–1899.
[14] E.J. Kansa, Multiquadrics — a scattered data approximation scheme with applications to computational fluid-dynamics, I. Surface approximations and partial derivative estimates, Comput. Math. Appl. 19 (8/9) (1990) 127–145.
[15] E.J. Kansa, Multiquadrics — A scattered data approximation scheme with applications to computational fluid-dynamics, II. Solutions to parabolic, hyperbolic and elliptic partial differential equations, Comput. Math. Appl. 19 (8/9) (1990) 147–161.
[16] C.A. Micchelli, Interpolation of scattered data: distance matrices and conditionally positive definite functions, Constr. Approx. 2 (1986) 11–22.
[17] R. Schaback, H. Wendland, Characterization and construction of radial basis functions, in: N. Dyn, D. Leviatan, D. Levin, A. Pinkus (Eds.), Multivariate Approximation and Applications, Cambridge University Press, 2001, pp. 1–24.
[18] C. Yang, R. Duraiswami, N.A. Gumerov, Improved Fast Gauss Transform, Dept. of Computer Science, University of Maryland, CS-TR-4495, 2003.