HPCSE I

CLASS NOTES

Simulations Using Particles

Petros Koumoutsakos

IMPORTANT NOTES

1. REFERENCES: Much of the material (ideas, definitions, concepts, examples, etc) in these notes is taken *(in some cases verbatim)* for teaching purposes, from several references, and in particular the references listed in the text.

2. CONTENT: The present Notes are a first LaTex draft for the Lectures of *Multiscale Simulations Using Particles*. The notes have been checked but there is no guarantee that they are free of mistakes. Please use with care.

3. DISCLAIMER: The present notes are only informally distributed and are intended ONLY as study aid for ETHZ students registered for the Course High Performance Computing for Science and Engineering (HS 2014).

Contents

ABSTRACT

The simulation of fluid flows using particles is becoming increasingly popular in Computer Graphics (CG). The grid-free character of particles, the flexibility in handling complex flow configurations and the possibility to obtain visually realistic results with a small number of computational elements are some of the main reasons for the success of these methods. In the Computational Fluid Dynamics (CFD) community, the realization that by periodically regularizing the particle locations can lead to highly accurate flow simulations, without detracting from the adaptivity and robustness of the method has led in turn to a renaissance in flow simulations using particles.

In this course we review recent advances in flow simulations using particles with a focus on developing a bridge fostering an interdisciplinary scientific exchange between the CG and the CFD communities. The course will describe advances in particle methods in a comparative, case study driven framework. In this framework we will address for example visual realism of liquid simulations as related to the accuracy of enforcing incompressibility constraints in Smooth Particle Hydrodynamics (SPH) and Vortex Methods (VM). We will discuss the role of advantages and drawbacks for particle simulations when using remeshing, we will present techniques for the effective handling of fluids interacting with solids and free surfaces and in turn the use of Computer Graphics algorithms and hardware to accelerate flow simulations of relevance to the CFD community.

The course will be accessible to researchers in computer graphics with minimal background in Flow physics. We will provide the necessary CFD background of particle simulations dealing with issues such as incompressibility, enforcement of free-surface and wall boundary conditions. The course will provide a concise description of the requirements for accurate simulation of fluid flows and their extension to visually realistic simulations of flows pertinent to computer graphics

1 INTRODUCTION

This introduction is taken in large parts from Koumoutsakos [Koumoutsakos, 2005].

The simulation of the motion of interacting particles is a deceivingly simple, yet powerful and natural method for exploring physical systems as diverse as planetary dark matter and proteins, unsteady separated flows and plasmas.

Particle methods have been advocated for efficient simulations of multiphysics phenomena in complex deforming computational domains in several fields of science ranging from astrophysics to solid mechanics (see [Monaghan, 2005, Liu et al., 1995] and references therein). In computer graphics, particle systems were introduced with the pioneering work of Reeves [Reeves, 1983] and have continued over the years to be the backbone of several impressive animations [Kawaguchi, 1982, Smith, 1984, Sims, 1990, Premoze et al., 2003, Pfister and Gross, 2004, Selle et al., 2005]. In the CFD community, particle methods were the first techniques to ever be used for the numerical simulation of fluids, starting with the pioneering calculations by hand of the evolution of a vortex sheet by Rosenhead [Rosenhead, 1930] and continuing with the works of Chorin [Chorin, 1973] and Leonard [Leonard, 1980]. In Direct Numerical Simulations of compressible and incompressible flows it has been shown that caution must be exercised when using a grid free method [HERNQUIST, 1993] and that regularization of the particle locations is necessary in order for the method to converge [Koumoutsakos, 1997] to the solution of the equations that have been discretized. At the same time in graphics the loss of accuracy of the method in terms of incompressibility [Foster and Metaxas, 1996, Fedkiw et al., 2001, CARLSON et al., 2004] and conservation of geometrical constraints [Elcott et al., 2007] may affect the visual realism of the flow.

Particles can be viewed as objects carrying a physical property of a system, that is being simulated through the solution of Ordinary Differential Equations (ODE) that determine the trajectories and the evolution of the properties carried by the particles. Particle methods amount to the solution of a system of ODEs :

$$\frac{d\mathbf{x}_p}{dt} = \mathbf{u}_p(\mathbf{x}_p, t) = \sum_{q=1}^{N} K\left(\mathbf{x}_p, \mathbf{x}_q; \mathbf{w}_p, \mathbf{w}_q\right) \tag{1.1}$$

$$\frac{d\mathbf{w}_p}{dt} = \sum_{q=1}^{N} F\left(\mathbf{x}_p, \mathbf{x}_q; \mathbf{w}_p, \mathbf{w}_q\right) \tag{1.2}$$

where $\mathbf{x}_p$, $\mathbf{u}_p$ denote the locations and velocities of the N particles, $\mathbf{w}_p$ denote particle properties (such as density, temperature, velocity, vorticity) and K, F represent the dynamics of the simulated physical system. In flow simulations particles are implemented with a Lagrangian formulation of the continuum equations, as in the vorticity formulation of the Navier-Stokes equations, or with systems that are discrete by nature, as in molecular flows at the nanoscale. Continuum flows such as flows in porous media and turbulent flows are inherently multiscale due to the range of scales that govern the underlying physical phenomena. The continuum assumption fails in flow regions containing contact lines and shocks and suitable molecular descriptions become necessary. A consistent and systematic framework is necessary to couple molecular and macroscale descriptions because the macroscale flows determine the external conditions that influence the molecular system, which in turn influences the larger scales by modifying its boundary conditions.

Particle methods such as Vortex Methods (VMs) and Smooth Particle Hydrodynamics (SPH) present an adaptive, efficient, stable and accurate computational method for simulating continuum flow phenomena and for capturing interfaces such as vortex sheets. On the other hand, particle methods encounter difficulties in the accurate treatment of boundary conditions, while their adaptivity is often associated with severe particle distortion that may introduce spurious scales. Ongoing research efforts attempt to address these issues as outlined in the review.

In molecular and mesoscopic simulations particle methods, such as Molecular Dynamics (MD) and Dissipative Particle Dynamics (DPD) are the method of choice because the discrete representation of the underlying physics is inherently linked to interacting particles. Particle methods for continuum and discrete systems present a unifying formulation that can enable systematic and robust multiscale simulations as we outline in this review.

We wish to note that particle formulations of fluid mechanics phenomena can be extended also in the mesoscale and the nanoscale regimes with techniques such as Molecular Dynamics (MD) and Dissipative Particle Dynamics (DPD) inherently linked to the discrete representation of the underlying physics. In fact particle methods enable a unifying formulation that can enable systematic and robust multiscale flow simulations as well as simulations continuum and discrete systems [Koumoutsakos, 2005] Indeed a remarkable feature of particle methods is that their computational structure involves a large number of common abstractions that help in their computational implementation, while at the same time particle methods are distinguished by the fact that they are inherently linked to the physics of the systems that they simulate.

Returning to the realm of continuum flows, particle methods, applied to the solution of convection-dominated problems in the context of Vortex Methods and Smooth Particle Hydrodynamics, enjoy an automatic adaptivity of the computational domain as dictated by the convective map. The field quantities can always be reconstructed by a linear superposition of the individual fields carried by the particles. In smooth particle methods - as opposed to point particle methods - each particle is associated with a smooth core function, or 'blob' enabling the smooth representation of the field quantities and efficient discretizations of the governing equations. The Lagrangian form of particle methods avoids the explicit discretization of the convective term in the governing transport equations and the associated stability constraints. The particle positions are modified according to the local flow map, making the method self-adaptive. This adaptation however comes at the expense of the regularity of the particle distribution as particles move in order to adapt to the gradients of the flow field. Particle regularity can be enforced by remeshing the particle locations on a regular grid as it is discussed in this session.

In this review we focus on updating the reader in methodological advances in Lagrangian particle methods since the first related such review by Leonard in 1985 [Leonard, 1985] with an emphasis towards describing methodologies that enable multiresolution simulations. In the simulation of discrete systems, starting from the review of Koplik and Banavar [Koplik and Banavar, 1995] we focus on hybrid continuum-molecular flow simulations. In this article we do not discuss particle methods for the simulation of kinetic equations for which we refer the reader to Chen and Doolen's (1998) work.

The review is structured as follows: We introduce particle methods for continuous systems by illustrating unifying concepts such as function and derivative particle approximations. We discuss the fundamental problem of particle distortion and remeshing associated with the Lagrangian formulation and we introduce multiresolution particle methods. We briefly outline the key characteristics of molecular simulations and we discuss recent advances in hybrid continuum-molecular

simulations. We conclude by describing efficient tools for large scale simulations using particle methods and we provide an outlook for future developments in particle methods in this new era of multiscale modeling and simulation.
Particle methods can be crudely classified in the following manner. A first class of methods considers discrete systems where particles follow Newton's laws of motion

$$\frac{d\mathbf{x}}{dt} = \mathbf{u}(\mathbf{x}, t) \tag{1.3}$$

$$m\frac{d\mathbf{u}}{dt} = \mathbf{F}(\mathbf{x}, \mathbf{u}, t) \tag{1.4}$$

Methods in this group can actually span a broad range of scales. At one end of the spectrum of lengthscales, we have molecular dynamics (M.D.) where $\mathbf{F}$ models atomic forces. At the other, we have planetary simulations where $\mathbf{F}$ is the law of gravity. Discrete methods have emerged recently at meso-scales. These methods attempt at modeling a continuum at scales where some randomness appears, e.g. Brownian motion.
The other category covers particles which discretize a continuum and laws given as Partial Differential Equations (PDE's). These methods cover most physical lengthscales at the exception of the atomistic ones; their domain of predilection lies in advection-dominated problems such as fluid mechanics.
The present notes aim at giving a short but nevertheless extensive overview of these two families. In a first step, we briefly look at the properties of ordinary differential equations, keeping a particle simulation perspective.

2 PARTICLE APPROXIMATIONS

This chapter covers particle approximations for continuous problems, such as continuum mechanics. For this class of problem, the physics can be described in terms of fields. These fields —in general—display some smoothness and can be differentiated to model diffusive and advective fluxes.
The latter ones are the *raison d'être* of a particle method. The advection equation for a scalar reads

$$\frac{\partial \phi}{\partial t} + \nabla \cdot (\phi \mathbf{u}) = 0 \; ; \tag{2.1}$$

which corresponds to the conservation of ϕ in a material volume

$$\frac{d}{dt} \int_{V_p(t)} \phi \, dV(\mathbf{x}) = 0 \; . \tag{2.2}$$

This is the starting point of continuum particle methods: the computational elements are material volumes which follow the flow map. A more general problem will be stated as

$$\frac{\mathcal{D}\phi}{\mathcal{D}t} = \mathcal{L}(\phi) \tag{2.3}$$

$$\frac{d\mathbf{x}}{dt} = \mathbf{u} \tag{2.4}$$

2.1 Particle approximations of Functions

2.1.1 Point particles

In a first step, we consider a function approximation

$$u(\mathbf{x}) = \sum_p \Gamma_p \delta(\mathbf{x} - \mathbf{x}_p)$$

by a set of point particles with weights $\Gamma_p, p = 1, \ldots, N$ located at positions $\mathbf{x}_p$. Point particles concentrate their function all in one point with a Dirac function.

The Dirac delta function - Definition The Dirac delta is not a proper function. An informal but intuitive description is the following: it is an infinitely narrow Gaussian function

$$\delta(\mathbf{x}) = \lim_{\epsilon \to 0} \eta_\epsilon(\mathbf{x}) = \frac{1}{(2\pi\epsilon^2)^{d/2}} \, \mathrm{e}^{-\frac{|\mathbf{x}|^2}{2\epsilon^2}} \; , \tag{2.5}$$

with its maximum value tending to infinity.
The Dirac can be rigorously defined in terms of a distribution (or generalized function) and we will rather use its defining characteristic

$$\int_{-\infty}^{\infty} f(x) \, \delta(x) \, dx = f(0) \tag{2.6}$$

Properties The following properties follow directly from its definition

$$\int_{-\infty}^{\infty} \delta(x)\, dx = 1 \tag{2.7}$$

$$\int_{-\infty}^{\infty} f(x-a)\, \delta(x)\, dx = f(a) \tag{2.8}$$

As an immediate consequence, the Fourier transform of the Dirac is constant,

$$\hat{f}(k) = \int \mathrm{e}^{-ikx}\ \delta(x)\, dx = 1\ , \tag{2.9}$$

thereby confirming its status of *very unsmooth function*[1]. $\delta(x)$ can be expressed as the limit of several functions, not only the Gaussian of Eq. 2.5,

$$\delta(x) = \lim_{\epsilon \to 0} \frac{1}{\pi} \frac{\epsilon}{x^2 + \epsilon^2} \tag{2.10}$$

$$= \lim_{\epsilon \to 0} \epsilon \mid x \mid^{\epsilon - 1} \tag{2.11}$$

2.1.2 Point Particle Discretization

Let us now use the property of Eq. 2.8 to build the point particle approximation. The integral

$$u(\mathbf{x}) = \int u(\mathbf{y}) \delta(\mathbf{x} - \mathbf{y}) d\mathbf{y} \tag{2.12}$$

is exact and can be decomposed into subvolumes V_k and

$$u(\mathbf{x}) = \sum_k \int_{V_k} u(\mathbf{y}) \delta(\mathbf{x} - \mathbf{y}) d\mathbf{y}\ . \tag{2.13}$$

In $\mathbf{R}^d$, we choose the subvolumes $V_\mathbf{k}$ as aligned on a lattice and switch to a vector index $\mathbf{k} \in \mathbf{Z}^d$ as

$$V_\mathbf{k} = \{\mathbf{x} \in \mathbf{R}^d | (k_i - \frac{1}{2})h < x_i < (k_i + \frac{1}{2})h,\ i = 1, \ldots, d\}.$$

We can approximate the integrals of Eq. 2.13 with a mid-point rule

$$\int_{V_\mathbf{k}} g(\mathbf{x}) d\mathbf{x} \sim w_k g(\mathbf{x}_k)\ . \tag{2.14}$$

where w_k is the measure of V_k (h^d, in this case). This yields

$$u(\mathbf{x}) \simeq u^h(\mathbf{x}) = \sum_k w_k u(\mathbf{x}_k) \delta(\mathbf{x} - \mathbf{x}_k)\ . \tag{2.15}$$

The order of this approximation can be evaluated in the context of Sobolev spaces $W^{m,p}$ [**?**]. We will skip the details as they pertain more to a functional analysis class and just mention the fundamental results.

[1] If $f(x)$ is n times differentiable, its transform $\hat{f}(k)$ decays at infinity as $|k|^{n+2}$

The accuracy of the point particle approximation is actually governed by the accuracy of the mid-point rule

$$E(g) = \sum_k E_k(g) = \sum_k \int_{V_\mathbf{k}} g(\mathbf{x})\, d\mathbf{x} - w_k\, g(\mathbf{x}_k)\ .$$

Lemma 1. *There is a constant $C > 0$ independent of h, such that for every function $g \in W^{m,1}(\mathbf{R}^d)$ and with $m \geq d$*

$$\mid E(g) \leq Ch^m \mid g \mid_{m,1}$$

We will thus have

$$||u(\mathbf{x}) - u^h(\mathbf{x})|| \leq \mathcal{O}(h^m) \tag{2.16}$$

However, this is only valid for a domain where u and all its derivatives vanish at the boundary or decay fast enough at infinity; the mid-point rule breaks down to second order accuracy for bounded domain where u does not vanish.

2.2 Smoothed particle approximations of Functions

A dirac function is not the most convenient object to handle in computations. It is for this reason that regularized or *smoothed* versions of the point particle methods were developed.

2.2.1 Smoothing

Let us now consider the field u and apply a filtering operation

$$u_\epsilon(\mathbf{x}) = \int u(\mathbf{y})\zeta_\epsilon(\mathbf{x} - \mathbf{y})\, d\mathbf{y}$$

where ϵ denotes the width or length scale of the filter.
This smoothing function tends toward the Dirac function as we decrease the filtering width

$$\zeta_\epsilon(\mathbf{x}) \to \delta(\mathbf{x}) \text{ for } \epsilon \to 0$$

and thus

$$u_\epsilon(\mathbf{x}) \to u(\mathbf{x}) \text{ for } \epsilon \to 0\ .$$

Let us study this convergence. First we define

$$\zeta_\epsilon(\mathbf{x}) = \frac{1}{\epsilon^d}\zeta(\frac{\mid \mathbf{x} \mid}{\epsilon})\ .$$

Theorem 2. *If we have the following moment properties for the smoothing function ζ*

$$\int \mathbf{x}^{\boldsymbol{\alpha}}\zeta(\mathbf{x})\, d\mathbf{x} = 0 \textit{ for } 1 \leq |\boldsymbol{\alpha}| \leq r - 1 \tag{2.17}$$

$$\int \mid \mathbf{x} \mid^r \zeta(\mathbf{x})\, d\mathbf{x} \leq \infty \tag{2.18}$$

$$\int \zeta(\mathbf{x})\, d\mathbf{x} = 1 \tag{2.19}$$

we have that $|| u - u_\epsilon ||= \mathcal{O}(\epsilon^r)$

Proof. We need to bound the difference

$$u - u_\epsilon = u(\mathbf{x}) - \int u(\mathbf{y})\zeta_\epsilon(\mathbf{x}-\mathbf{y})dy$$

By the moment condition of Eq. 2.19, we can write

$$\begin{aligned} u - u_\epsilon &= \int u(\mathbf{x})\,\zeta_\epsilon(\mathbf{x}-\mathbf{y})dy - \int u(\mathbf{y})\,\zeta_\epsilon(\mathbf{x}-\mathbf{y})\,d\mathbf{y} \\ &= \int (u(\mathbf{x}) - u(\mathbf{y}))\;\zeta_\epsilon(\mathbf{x}-\mathbf{y})\,d\mathbf{y} \end{aligned}$$

A Taylor series of u yields

$$u(\mathbf{x}) - u(\mathbf{y}) = \sum_{|\boldsymbol{\alpha}|=1}^{r-1} \frac{1}{|\boldsymbol{\alpha}|!}\partial^{|\boldsymbol{\alpha}|}u\,(\mathbf{x}-\mathbf{y})^{\boldsymbol{\alpha}} + \frac{1}{(r-1)!}\int \dots$$

Substituting we obtain that

$$u - u_\epsilon = \int \sum_{|\boldsymbol{\alpha}|=1}^{r-1} \frac{1}{|\boldsymbol{\alpha}|!}\partial^{|\boldsymbol{\alpha}|}u\,(\mathbf{x}-\mathbf{y})^{\boldsymbol{\alpha}}\,\zeta_\epsilon(\mathbf{x}-\mathbf{y})\,d\mathbf{y} + \mathcal{O}(\epsilon^r)\,.$$

The moment properties of Eq. 2.17 then ensure that

$$||u - u_\epsilon|| \leq \mathcal{O}(\epsilon^r)$$

□

Here are some typical mollifiers for particle methods

$$\begin{aligned} &\text{Gaussian:} && \zeta(\rho) = \frac{1}{(2\pi)^{d/2}}\,\mathrm{e}^{-\rho^2/2},\ r=2 \\ &\text{Algebraic:} && \zeta(\rho) = \frac{2(2-\rho^2)}{\pi(1+\rho^2)^4},\ r=3 \\ &\text{High order Gaussian} && \zeta(\rho) = \begin{cases} d=1 & \frac{1}{(2\pi)^{1/2}}(3-\rho^2)\,\mathrm{e}^{-\rho^2/2} \\ d=2 & \frac{1}{(4\pi)}(4-\rho^2)\,\mathrm{e}^{-\rho^2/2} \\ d=3 & \frac{1}{2(2\pi)^{3/2}}(5-\rho^2)\,\mathrm{e}^{-\rho^2/2} \end{cases},\ r=4 \end{aligned}$$

We note that an order $r \geq 3$ can only be achieved with a non-positive ζ.
Exercise: Recover the coefficient a and b in the formula $\zeta(\rho) = (a\rho^2 + b)e^{-\rho^2/2}$ if we want ζ to be fourth order accurate.

2.2.2 Smoothed Particle Discretizations of Functions

We have so far introduced two operations on the field u: discretization (Section 2.1.2) and smoothing (Section 2.2.1). Let us consider now their combination to obtain a *smoothed particle* approximation. More precisely, we apply the filtering operation ζ_ϵ to Eq. 2.14. This replaces the Dirac functions with a smoothing function.

Theorem 3. *The smoothed particle approximation*

$$u_\epsilon^h(\mathbf{x}) = \sum_k w_k u(\mathbf{x}_k)\zeta_\epsilon(\mathbf{x}-\mathbf{x}_k) \tag{2.20}$$

introduces an error

$$||u(\mathbf{x}) - u_\epsilon^h(\mathbf{x})|| \leq C_1\epsilon^r + C_2\left(\frac{h}{\epsilon}\right)^m \epsilon \tag{2.21}$$

where C_1 and C_2 are constants.

Proof. We use a triangular inequality to evaluate the accuracy

$$\begin{aligned} ||u(\mathbf{x}) - u_\epsilon^h(\mathbf{x})|| &= ||u(\mathbf{x}) - u_\epsilon(\mathbf{x}) + u_\epsilon(\mathbf{x}) - u_\epsilon^h(\mathbf{x})|| & (2.22)\\ &\leq ||u(\mathbf{x}) - u_\epsilon(\mathbf{x})|| + ||u_\epsilon(\mathbf{x}) - u_\epsilon^h(\mathbf{x})|| & (2.23)\end{aligned}$$

The first term is the error due to the smoothing operation and can be bounded by $C_1\epsilon^r$. The second one corresponds to the discretization error of this smoothed field and can be shown to be $\mathcal{O}\left(\left(\frac{h}{\epsilon}\right)^m \epsilon\right)$ [**?**, Beale and Majda, 1982, **?**, Cottet and Koumoutsakos, 2000]. □

2.3 Accuracy of smoothed particle approximations - The overlap issue

The bound of Eq. 2.20 hints at a central issue in particle methods and calls for a few remarks

1. In continuum particle methods the field is recovered at every location of the domain when one considers the collective behavior of all computational elements.

2. The finest length scales are characterized by the particle core size ϵ rather than the inter-particle distance h.

3. The essence of particle methods is the communication of information between the particles.

The second term measures the overlap between particles. More explicitly, this term indicates whether there are enough particles to represent the length scales allowed by the filter.
This accuracy break-down has some important practical consequences. Indeed, a particle set that undergoes large distortions will need to regularized to recover convergence.

2.3.1 Regularization of particle strengths

This method is known as Beale's method [Beale and Majda, 1982]; it considers two sets of particles, an old one

$$\tilde{u}(\mathbf{x}_p) = \sum_q \tilde{\Gamma}_q \zeta_\epsilon(\mathbf{x}_p - \tilde{\mathbf{x}}_q)$$

where $\tilde{\mathbf{x}}$ and $\tilde{\Gamma}_q$ denote the old distorted particle locations and strengths, and a new one

$$u(\mathbf{x}_p) = \sum_q \Gamma_q \zeta_\epsilon(\mathbf{x}_p - \mathbf{x}_q)$$

which we have to find the weights w_q of. Beale's method considers that the new particle locations are fixed and seeks to enforce explicitly $\tilde{u}(\mathbf{x}_p) = u(\mathbf{x}_p)$. A linear system has thus to be solved. If we define $\gamma_p = w_p u(\mathbf{x}_p)$ and $[A_{pq}] = V_p \zeta_\epsilon(\mathbf{x}_p - \mathbf{x}_q)$, the system can be written as

$$A_{pq}\Gamma_q = \gamma_p \ .$$

Because of its size, this system will typically be solved with an iterative solver

$$w_p^{(n+1)} = w_p^{(n)} + \gamma_p - \sum_q w_p \Gamma_q^{(n)} \zeta_\epsilon(\mathbf{x}_p - \mathbf{x}_q) \ .$$

This operation has several problems: it is expensive, ill-posed and does not explicitly conserve global diagnostics like $\int u \, d\mathbf{x}$.

2.3.2 Moment conserving interpolations of particles onto cartesian grids

This technique resolves most of the drawbacks of Beale's method by moving away from its collocation approach. We introduce a mesh with mesh nodes being the candidate regularized positions. Let $\tilde{x}_p$ denote the old distorted positions and $x_p = (ih, jh, kh)$ the new regularized positions. Remeshing proceeds by directly interpolating the particle weights

$$\Gamma_p = \sum_q \tilde{\Gamma}_q W(\frac{\mathbf{x}_p - \tilde{\mathbf{x}}_q}{h}) \ . \tag{2.24}$$

W is the interpolation kernel whose properties determine the type and accuracy of the interpolation. The discrepancy between the old and new sets reads

$$E = \sum_p \tilde{\Gamma}_p \delta(x - \tilde{x}_p) - \sum_p \Gamma_p \delta(x - x_p) \ .$$

If we now take the convolution of our particle set with any kernel ϕ (mollifying function, Green's function), we have

$$\begin{aligned} E_\phi &= \sum_p \tilde{\Gamma}_p \phi(\mathbf{x} - \tilde{\mathbf{x}}_p) - \sum_p \Gamma_p \phi(\mathbf{x} - \mathbf{x}_p) \\ &= \sum_p \tilde{\Gamma}_p \phi(\mathbf{x} - \tilde{\mathbf{x}}_p) - \sum_p \sum_q \tilde{\Gamma}_q W\left(\frac{\mathbf{x}_p - \tilde{\mathbf{x}}_q}{h}\right) \phi(\mathbf{x} - \mathbf{x}_p) \\ &= \sum_q \tilde{\Gamma}_q \left[\phi(\mathbf{x} - \tilde{\mathbf{x}}_q) - \sum_p W\left(\frac{\mathbf{x}_p - \tilde{\mathbf{x}}_q}{h}\right) \phi(\mathbf{x} - \mathbf{x}_p) \right] \ . \end{aligned} \tag{2.25}$$

We isolate the factor in the square brackets of Eq. 2.25 and enforce a mass conservation for W

$$\sum_p W(\frac{\mathbf{x} - \mathbf{x}_p}{h}) = 1 \ \forall \mathbf{x} \in \mathbf{R}^d \text{ and } \mathbf{x}_p = \mathbf{k}\, h, \ \mathbf{k} \in \mathbf{Z} \ . \tag{2.26}$$

The error factor then becomes

$$\epsilon_\phi = \sum_p \left(\phi(\mathbf{x} - \tilde{\mathbf{x}}_q) - \phi(\mathbf{x} - \mathbf{x}_p)\right) W\left(\frac{\mathbf{x}_p - \tilde{\mathbf{x}}_q}{h}\right) \ .$$

If we assume that ϕ is differentiable, a Taylor expansion of ϕ about $\mathbf{x} - \mathbf{x}_p$ yields

$$\epsilon_\phi = \sum_p \sum_{\boldsymbol{\alpha}} (\mathbf{x}_p - \tilde{\mathbf{x}}_q)^{\boldsymbol{\alpha}} \frac{\partial^{|\boldsymbol{\alpha}|}\phi}{\partial \mathbf{x}^{\boldsymbol{\alpha}}} W(\mathbf{x}_p - \tilde{\mathbf{x}}_q)$$

From this result we can see that the remeshing accuracy is governed by moment conditions (as for smoothing, Eq. 2.17); the error will be $\mathcal{O}(h^m)$ if and only if condition 2.26 holds and

$$\sum_q (\mathbf{x} - \mathbf{x}_q)^\alpha W(\frac{\mathbf{x} - \mathbf{x}_q}{h}) = 0 \text{ for } 1 \leq |\alpha| \leq m - 1 \ ,$$

or equivalently

$$\sum_q \mathbf{x}_q^\alpha W(\frac{\mathbf{x} - \mathbf{x}_q}{h}) = \mathbf{x}^\alpha \text{ for } 0 \leq |\alpha| \leq m - 1 \ .$$

We shall now derive a few example of interpolation functions.How do we derive these interpolation functions

$m = 1$ this is the Nearest Grid Point interpolation (NGP)

$$W(u) = \begin{cases} 1 & \text{if } u < 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{2.27}$$

$m = 2$ In this case, we need to use two mesh points. Their strengths will satisfy

$$W(\frac{x - x_1}{h}) + W(\frac{x - x_2}{h}) = 1$$
$$(x - x_1)W(\frac{x - x_1}{h}) + (x - x_2)W(\frac{x - x_2}{h}) = 0 \ .$$

The interpolation weights must then be

$$W_1(x) = 1 - \frac{(x - x_1)}{h}$$
$$W_2(x) = 1 + \frac{(x - x_2)}{h}$$

or following the notation used for the NGP

$$W(u) = \begin{cases} 1 - |u| & \text{if } u < 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.28}$$

This is also known as the witch-hat function.

$m = 3$ The conservation of the first three moments on three mesh points leads to the following scheme

$$W(u) = \begin{cases} \frac{(u^2+3u+2)}{2} & \text{if } \frac{-3}{2} \leq u < \frac{1}{2} \\ 1 - u^2 & \text{if } \frac{-1}{2} \leq u < \frac{1}{2} \\ \frac{(u^2-3u+2)}{2} & \text{if } \frac{1}{2} \leq u < \frac{3}{2} \end{cases} \tag{2.29}$$

This scheme presents discontinuities, which can in turn generate oscillations over time. Other schemes with $m = 3$ have been developed to offer smoothness and high order at the cost of a wider stencil, e.g. M_4' with four points [Monaghan, 1985b].

2.4 Particle Approximations for Differential operators

A particle method handles the advective part of a problem in a natural way but in most cases still needs to treat whatever operator lies on the right-hand side of the conservation equation

$$\frac{\mathcal{D}\phi}{\mathcal{D}t} = \mathcal{L}(\phi) \ . \tag{2.30}$$

This could be for example the diffusive flux $D\Delta\phi$ in an advection-diffusion equation.
We now look at the existing methodologies to carry out these differentiations on particles.

2.4.1 Smoothed particle derivatives

We begin with arguably the most straightforward interpretation of derivatives on particles. This technique was mainly developed in the framework of Smoothed Particle Hydrodynamics (SPH) [Gingold and Monaghan, 1977, Monaghan, 1992, Monaghan, 2005]. This method was originally developed for astrophysical compressible flows and needs to compute the divergence of the stress tensor

$$\frac{\mathcal{D}(\rho\mathbf{u})}{\mathcal{D}t} = \nabla \cdot \boldsymbol{\tau} \ . \tag{2.31}$$

If we have a filtered field

$$c_\epsilon(\mathbf{x}) = \int c(\mathbf{y})\zeta_\epsilon(\mathbf{x}-\mathbf{y})\, dy \ ,$$

the following holds for its derivative

$$\begin{aligned} \frac{\partial c_\epsilon(\mathbf{x})}{\partial \mathbf{x}} &= \int c(\mathbf{y})\frac{\partial \zeta_\epsilon}{\partial \mathbf{x}}(\mathbf{x}-\mathbf{y})\, d\mathbf{y} \\ &= \int -c(\mathbf{y})\frac{\partial \zeta_\epsilon}{\partial \mathbf{y}}(\mathbf{x}-\mathbf{y})\, d\mathbf{y} \\ &= -\int_S c(\mathbf{y})\,\zeta(\mathbf{x}-\mathbf{y})\,\mathbf{n}dS + \int \frac{\partial c}{\partial \mathbf{y}}(\mathbf{y})\zeta_\epsilon(\mathbf{x}-\mathbf{y})\, d\mathbf{y} \end{aligned} \tag{2.32}$$

If we consider a field which decays fast enough at the boundaries, we recover the differentiation properties of convolution

$$\begin{aligned} \frac{\partial c_\epsilon}{\partial \mathbf{x}}(\mathbf{x}) &= \frac{\partial c}{\partial \mathbf{x}} * \zeta_\epsilon \\ &= \frac{\partial}{\partial \mathbf{x}}(c * \zeta_\epsilon) \end{aligned}$$

which essentially says that the derivative of a filtered field is equal to the filtered derivative.
An additional useful property concerns radially symmetric kernels

$$\zeta_\epsilon(\mathbf{x}_i - \mathbf{x}_j) = \frac{1}{\epsilon^d}\zeta(\frac{||\mathbf{x}_i - \mathbf{x}_j||}{\epsilon}) \ ;$$

by the chain rule, we have

$$\nabla_i \zeta_\epsilon(\mathbf{x}_i - \mathbf{x}_j) = -\nabla_j \zeta_\epsilon(\mathbf{x}_i - \mathbf{x}_j) \ .$$

This symmetry property plays an important role in the conservation properties of SPH.
Let us now derive a smoothed derivative. We have

$$\int \frac{\partial \zeta_\epsilon}{\partial \mathbf{x}}(\mathbf{x}-\mathbf{y})d\mathbf{y} = \frac{\partial}{\partial \mathbf{x}}\int \zeta_\epsilon(\mathbf{x}-\mathbf{y})d\mathbf{y} = \frac{\partial 1}{\partial x} = 0\ ,$$

which allows to add a null term to Eq. 2.32

$$\begin{aligned}\frac{\partial c_\epsilon}{\partial \mathbf{x}} &= \int c(\mathbf{y})\frac{\partial \zeta_\epsilon}{\partial \mathbf{x}}(\mathbf{x}-\mathbf{y})\,d\mathbf{y} - c(\mathbf{x})\int \frac{\partial \zeta_\epsilon}{\partial \mathbf{x}}(\mathbf{x}-\mathbf{y})d\mathbf{y} \\ &= \int (c(\mathbf{y})-c(\mathbf{x}))\ \frac{\partial \zeta_\epsilon}{\partial \mathbf{x}}(\mathbf{x}-\mathbf{y})\,d\mathbf{y} \end{aligned} \tag{2.33}$$

In discretized form, this gives us

$$\nabla c_\epsilon^h(\mathbf{x}_i) = \sum_j w_j\,(c_j - c_i)\,\nabla\zeta_\epsilon(\mathbf{x}_i-\mathbf{x}_j)\ . \tag{2.34}$$

We now see that the addition of the null term makes this expression vanish for a constant field.
Finally, we note that the loss of overlap is even worse for the field derivative since derivation adds a h^{-1} factor to the discretization error bound, making it $\mathcal{O}(h^{-1}(h/\epsilon)^m)$.

2.4.2 Particle Strength Exchange

An alternative way to compute derivatives requires certain properties for the mollifying kernels. The method of Particle Strength Exchange (PSE) [Degond and Mas-Gallic, 1989] takes a different approach and reverts to the particles initial role, which is quadrature. The PSE indeed builds an approximation of a differential operator as an integral.
On a historical note, the PSE method introduced a deterministic formalism at a time where people were doing random walk in order to model diffusion

$$\frac{\partial c}{\partial t} = \nu\nabla^2 c\ .$$

For the particle discretization $c_\epsilon = \sum_i \Gamma_i\zeta_\epsilon(\mathbf{x}-\mathbf{x}_i(t))$, we have

$$\mathbf{x}_i(t+\Delta t) = \mathbf{x}_i(t) + \mathcal{N}(0,\nu\Delta t)$$

where $\mathcal{N}(\mu,\sigma^2)$ is a normal random variable of mean μ and variance σ^2. While simple and easy to implement, random walk has some drawbacks, the most important being its slow convergence: $\mathcal{O}(N^{-1/2})$.

Theorem 4. *Let η be a kernel satisfying the following moment properties*

$$\begin{aligned}&\int x_i\,x_j\,\eta(\mathbf{x})\,d\mathbf{x} = 2\delta_{ij} \text{ for } i,j = 1,\dots,d \\ &\int \mathbf{x}^{\boldsymbol{\alpha}}\,\eta(\mathbf{x})\,d\mathbf{x} = 0 \text{ if } |\boldsymbol{\alpha}| = 1 \text{ or } 3 \le |\boldsymbol{\alpha}| \le r \\ &\int |\mathbf{x}|^{r+2}\eta(\mathbf{x})\,d\mathbf{x} < \infty\ .\end{aligned}$$

The integral operator

$$\Delta_\epsilon c(\mathbf{x}) = \frac{1}{\epsilon^2}\int [c(\mathbf{y}) - c(\mathbf{x})]\eta_\epsilon(\mathbf{x}-\mathbf{y})d\mathbf{y} \tag{2.35}$$

is an approximation of the Laplace operator with the following error behavior

$$||\Delta_\epsilon c - \Delta c|| \leq C\,\epsilon^r\,||c|| \tag{2.36}$$

Proof. We begin with the Taylor expansion of $c(\mathbf{x})$

$$\begin{aligned} c(\mathbf{y}) =& c(\mathbf{x}) + (\mathbf{y}-\mathbf{x})\nabla c(\mathbf{x}) + \ldots + \frac{1}{(r+1)!}(y-x)^{r+1}\nabla^{r+1}c(x) \\ &+ \mathcal{O}(||\mathbf{y}-\mathbf{x}||^{r+2}\;||\;w\;||_{r+2,\infty} \end{aligned}$$

If we substitute $c(\mathbf{y})$ by its expansion in the right-hand side of Eq. 2.35 and carry out the integration, all the terms corresponding to $|\boldsymbol{\alpha}| = 1, 3, \ldots r$ vanish and the ones with the second moment remain

$$\begin{aligned} \Delta_\epsilon c(\mathbf{x}) =& \frac{1}{2\epsilon^2}\sum_{i=1}^{d}\frac{\partial^2 c}{\partial x_i^2}(\mathbf{x})\int (y_i - x_i)^2\,\eta_\epsilon(\mathbf{x}-\mathbf{y})\,d\mathbf{y} \\ &+ \frac{1}{\epsilon^2}||c||_{r+2,\infty}\mathcal{O}\left(\int |y-x\;|^{r+2}\;\eta_\epsilon(x-y)dy\right) \end{aligned}$$

The error bound then follows directly from the first and last moment conditions. □
A PSE variant for the diffusion problem will then be given by

$$\Gamma_p^{n+1} = \Gamma_p^n + \frac{\nu\delta t}{\epsilon^2}\sum_q (w_p\,\Gamma_q^n - w_q\,\Gamma_p^n)\eta_\epsilon(\mathbf{x}_q^n - \mathbf{x}_p^n)\;.$$

One quickly notices that this scheme is conservative, i.e. $d/dt\sum_p \Gamma_p = 0$. An alternative approach, in line with the smoothed particle ones, is to use the identity that $\nabla_\epsilon^2 = w * \nabla^2\zeta_\epsilon$. This does not result in a conservative approximation though.
We note that the PSE formalism has been extended to any type of derivative and bounded domains(one-sided integrals) [Eldredge et al., 2002].

2.4.3 Smoothed Particle Hydrodynamics

In Smoothed Particle Hydrodynamics (SPH), the not in of a smoothing length it is often mixed with the notion of the distance between particles.
The classical SPH approximation is

$$f^{\mathrm{SPH}} = < f(x) > = \int \zeta_h(x-x')f(x')dx' \sim \sum_{i\in\Lambda}\zeta_h(x-x_i)\,f(x_i)\,vol_i \tag{2.37}$$

The SPH kernels are usually required to satisfy the following relationships :

$$\begin{aligned} \int \zeta_h(x-x')dx' &= 1 \\ \zeta_h(x-x')f(x')dx' &\to \delta(x) \text{ with } \; h\to 0 \\ \zeta_h(x-x')dx' &\in C_o^k(R^d) \text{ for } \; k\geq 1 \end{aligned}$$

In general the kernels ζ_h are chosen so that $\zeta_h(x) = \frac{C_d}{h^d}\zeta(x/h)$ and the coefficient C_d is determined so that $\int \zeta_h(x)dx = 1$. Most of the SPH smoothing kernels are either Gaussian or of compact support. Compact support functions are usually chosen because of their computational efficiency while the Gaussian is chosen because of its smoothing properties and its continuous derivatives.
Examples of SPH kernels are (with: $r^2 = ||x||^2 = x_i \cdot x_i$):

$$\text{Gaussian:} \quad \zeta_h(r) = \frac{1}{(\pi h^2)^{d/2}} e^{-\frac{r^2}{h^2}}$$

$$\text{Cubic Spline} \quad \zeta_h(r) = \frac{C}{h^d}\begin{cases} 1 - \frac{3}{2}(r/h)^2 + \frac{3}{4}(r/h)^3 & \text{for: } 0 \le r/h \le 1 \\ \frac{1}{4}(2 - r/h)^3 & \text{for: } 1 \le r/h \le 2 \\ 0 \quad \text{otherwise} \end{cases}$$

$$\text{Quartic Spline:} \quad \zeta_h(r) = \frac{C}{h^d}\begin{cases} 1 - 6(r/h)^2 + 8(r/h)^3 - 3(r/h)^4 & \text{for: } 0 \le r/h \le 1 \\ 0 \quad \text{otherwise} \end{cases}$$

In SPH the kernels are usually selected as symmetric. We denote as $\mathbf{x}_{\alpha\beta} = \overrightarrow{x}_\alpha - \overrightarrow{x}_\beta$ and $\mathbf{x}_{\beta\alpha} = \overrightarrow{x}_\beta - \overrightarrow{x}_\alpha$ with $|x_{\alpha\beta}| = ||\overrightarrow{x}_{\alpha\beta}|| = \sqrt{(x_{\alpha_i} - x_{\beta_i})(x_{\alpha_i} - x_{\beta_i})} = |x_{\beta\alpha}|$. Then kernels that are usually employed in SPH are expressed as :

$$\zeta_h(\mathbf{x}_{\alpha\beta}) = \frac{1}{h^d}\zeta(\frac{|x_{\alpha\beta}|}{h}) = \zeta_{\alpha\beta} = \zeta_{\beta\alpha} \tag{2.38}$$

In case such symmetric kernels are being used then derivative operators on particles can be expressed as follows

$$\begin{aligned} \nabla_\alpha \zeta_{\alpha\beta} &= \nabla\zeta_h(|x - x_\beta|)|_{x=x_\alpha} \\ &= \frac{\partial\zeta_h(|x - x_\beta|)}{\partial(|x - x_\beta|)} \frac{\partial(|x - x_\beta|)}{\partial x_i}\widehat{e}_i \Big| x_i = x_\alpha \\ &= \frac{\mathbf{x}_{\alpha\beta}}{|\mathbf{x}_{\alpha\beta|}} \frac{\partial\zeta_{\alpha\beta}}{\partial x_{\alpha\beta}} \\ &= -\frac{\mathbf{x}_{\beta\alpha}}{|\mathbf{x}_{\beta\alpha|}} \frac{\partial\zeta_{\beta\alpha}}{\partial x_{\beta\alpha}} \\ &= -\nabla_\beta \zeta_{\beta\alpha} \end{aligned}$$

The above symmetry of the smoothing operator and the antisymmetry of its gradients are used in many situations in deriving the equations of SPH and in particular in enforcing conservation of momentum and energy in its formulation.

2.4.4 Hybrid approaches

In smoothed particle methods, or in the PSE, the cost of computing the diffusion at the particle locations scales linearly with the number of particles. The cost per particle is large though; e.g. using a gaussian in 3D, it requires finding about $5^3 = 125$ neighbors and then evaluate 125 times the kernel or its derivative.

On a mesh, the same computation would have involved $\mathcal{O}(10)$ operations within a Finite Difference stencil. A very efficient approach for remeshed particle methods will then consist in the combination of the particles and the mesh. Particles solely handle the advection part of the problem. If there are differential operators other than advection, particles quantities are interpolated onto the mesh, finite differences are used and the derivative is interpolated back to the particles. This scheme results in simple loops over the particles or the mesh nodes, thus avoiding expensive neighbor searches.

2.4.5 Partition of Unity and Reproducing Kernel Particle Methods

One key issue in particle approximations is the consistency of the partition of unity approximation in the presence of boundaries or for disturbed particle positions.
One must satisfy

$$\begin{aligned} \sum_{j=1}^{N} \zeta(x - x_j)\delta x_j &= 1 \\ \sum_{j=1}^{N} (x - x_j)\zeta(x - x_j)\delta x_j &= 0 \end{aligned}$$

It is easy to see that by summing such functions near boundaries (say at $x = 0$) then we will get only $1/2$ of their respective contribution as the other half will be across the boundary. A similar situation emerges when using particle approximations in distorted particle locations. Quantities that are easy to maintain as integrals they break down when they become quadratures.
Partition of unity methods: One solution about maintaining the accurcy of particle approximations is by developing particles such that (Liu MB et al., 2003)

$$\begin{aligned} \zeta(x - x_j, h) &= b_0(x, h) + b_1(x, h)\frac{x - x_j}{h} + b_2(x, h)(\frac{x - x_j}{h})^2 + ... \\ &= \sum_{l=0}^{K} b_l(x - x_j, h)(\frac{x - x_j}{h})^l \end{aligned}$$

3 FIELD EQUATIONS AND THEIR SOLVERS

The simulation of many physical problems involves solving a global problem at every time step. More explicitly, the evaluation of the right-hand side ($\mathcal{L}(\phi)$) or the velocity field ($\mathcal{D}/\mathcal{D}t$) will involve the solution of an elliptic problem, a Poisson equation in most cases, e.g. the pressure in incompressible flows, the electric potential in electrostatics, ...

$$\Delta\psi = f$$

There are essentially two paths to the solution of

$$\mathcal{L}_{\text{ellipt}} u = f \ .$$

In a first one, one first proceed with the discretization of the problem and the operator then carries out the inversion.

$$\begin{array}{lll} \mathcal{L}u = f & \xrightarrow{\text{discretization}} & \mathcal{L}^h u^h = f^h \\ & & \downarrow \text{inversion} \\ & & u^h = \left(\mathcal{L}^h\right)^{-1} f^h \end{array}$$

In a second approach, the operator is inverted then the problem is discretized.

$$\begin{array}{lll} \mathcal{L}u = f & & \\ \downarrow \text{inversion} & & \\ u = \mathcal{L}^{-1} f & \xrightarrow{\text{discretization}} & u^h = \left(\mathcal{L}^{-1}\right)^h f^h \end{array}$$

Both approaches do not yield the same solution, as the inversion and discretization of $\mathcal{L}$ do not commute

$$\left(\mathcal{L}^h\right)^{-1} \neq \left(\mathcal{L}^{-1}\right)^h \ .$$

Furthermore, the first path will involve consistency errors which originate in the discretization of a differential operator.

3.1 Examples

3.1.1 Vortex methods

$$\nabla \times (\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = \frac{-1}{\rho} \nabla p + \nu \nabla^2 \mathbf{u})$$

$$\frac{\partial \omega}{\partial t} + u \cdot \nabla \omega = \nu \nabla^2 \omega$$

$$\begin{aligned} \frac{D\omega}{Dt} &= \nu \nabla^2 \omega \\ \frac{dx}{dt} &= u \end{aligned}$$

The velocity field is obtained from a Poisson equation

$$\omega = \nabla \times \mathbf{u} \text{ and } \nabla \mathbf{u} = 0 \text{ (incompressible flow)}$$

$$\begin{aligned} \nabla \times \omega &= \nabla(\nabla \cdot \mathbf{u}) - \nabla^2 \mathbf{u} \\ \nabla^2 \mathbf{u} &= -\nabla \times \omega \end{aligned}$$

3.1.2 Molecular dynamics

Even though molecular dynamics is a discrete particle method, it can involve field equations when one considers non-local interactions such as electrostatics

$$\frac{d\mathbf{x}}{dt} = \mathbf{u}$$
$$\frac{d\mathbf{u}}{dt} = \frac{\mathbf{F}}{m}$$

where F derives from the electrostatic potential Φ

$$F = \nabla\Phi \text{ and } \nabla^2\Phi = -q \ .$$

3.2 Elliptic equations

A 1D model of elliptic equations are boundary value problems expressed as :

$$\frac{d^2\Phi}{dx^2} = f(x); \tag{3.1}$$

In higher dimensions this may be expressed by the Laplace equation:

$$\nabla^2\Phi = 0, \tag{3.2}$$

the Poisson equation:

$$\nabla^2\Phi = f, \tag{3.3}$$

and the Helmholtz equation:

$$\nabla^2\Phi + k^2\Phi = 0, \tag{3.4}$$

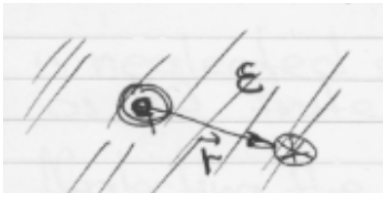
where:

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}. \tag{3.5}$$

Example 1. Equations for a constant electric field.
Given a distribution of electric charges in a dielectric medium, a constant electric field is created. Define as

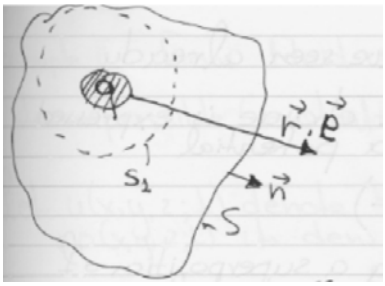
$$\begin{aligned} \rho &= \text{charge density} \\ \vec{e} &= \text{electric field} \\ \varepsilon &= \text{dielectric constant} \end{aligned}$$

Assume a charge q_e is concentrated at a point in a homogeneous medium (*i.e.* $\varepsilon = \text{const.}$)



Coulomb's Law:

$$\vec{e} = \frac{\vec{r}}{\varepsilon r^3} q_e = -\nabla \frac{q_e}{\varepsilon r} \tag{3.6}$$



The **flux** of vector $\varepsilon \vec{E}$ through an arbitrary closed volume with surface S enclosing the charge satisfies the equation

$$\iint\limits_S \varepsilon \vec{e} \cdot \vec{n}\, dS = \iint\limits_{S_1} \frac{\varepsilon \vec{r}}{\varepsilon r^3} q_e \cdot \vec{n}\, dS_1, \tag{3.7}$$

$$= \int_0^{\pi} \int_0^{2\pi} \frac{\varepsilon \vec{r}}{\varepsilon r^3} \cdot \vec{n} q_e\, dS_1, \tag{3.8}$$

$$= \int_0^{\pi} \int_0^{2\pi} \frac{\varepsilon \vec{r}}{\varepsilon r^3} \cdot \frac{\vec{r}}{r} q_e (r \sin(\vartheta)\, d\phi)(r\, d\vartheta), \tag{3.9}$$

$$= \int_0^{\pi} \int_0^{2\pi} q_e \sin(\vartheta)\, d\phi\, d\vartheta = 4\pi q_e, \tag{3.10}$$

$$\tag{3.11}$$

where S_1 is a unit sphere.

Note. The usual form of Coulomb's law is

$$\vec{F} = \frac{1}{\varepsilon} \frac{q_1 q_2}{|r_1 - r_2|^3} (\vec{r_1} - \vec{r_2}), \tag{3.12}$$

therefore the electrostatic field must be understood as a force per unit charge.

In the case of a continuous distribution of charges we refer to a charge density ρ and we may consider as point charge the quantity $\rho\, dV$ where dV is an infinitesimal volume.

From flux conservation we have:

$$4\pi \underbrace{\iiint_V \rho\, dV}_{\substack{this\\ replaces\\ q_e}} = -\iint_S \varepsilon \underbrace{\vec{E}}_{\substack{this\\ replaces\\ \vec{e}}} \cdot \vec{n}\, dS \tag{3.13}$$

Furthermore as we have seen already in Eq. 3.6, the field of a point charge is expressed as a gradient of a potential:

$$\vec{e} = -\nabla(\frac{q_e}{\varepsilon r}) \tag{3.14}$$

$\vec{E}$ can be seen as a superposition of fields described from a potential:

$$\vec{E} = -\nabla\Phi, \tag{3.15}$$

where Φ is the electrostatic potential.
For any closed curve

$$\oint \vec{E}\, d\vec{l} = -\oint \nabla\phi\, dl = 0 \tag{3.16}$$

If we return to Eq. (3.13) we have now:

$$4\pi \iiint \rho\, dV = -\iint \varepsilon\vec{E}\cdot\vec{n}\, dS \overbrace{=}^{\substack{Gauss\\ Ostrogradskiy}} -\iiint \nabla\cdot\left(\varepsilon\vec{E}\right)\, dV \rightarrow \boxed{\nabla\cdot\left(\varepsilon\vec{E}\right) = -4\pi\rho} \tag{3.17}$$

and using (3.16) we get:

$$\oint \vec{E}\, d\vec{l} = \iint_S (\nabla\times\vec{E})\cdot\vec{n}\, dS = 0 \rightarrow \boxed{\nabla\times\vec{E} = 0} \tag{3.18}$$

Then, the curl of an electrostatic field is zero. We can also replace Eq. 3.18 into Eq. 3.17 and obtain:

$$\nabla\cdot(\varepsilon\nabla\Phi) = -4\pi\rho \tag{3.19}$$

if for ε is constant, then:

$$\boxed{\nabla^2\Phi = -\frac{4\pi}{\varepsilon}\cdot\rho} \tag{3.20}$$

3.3 Solving the Laplace Equation analytically

Now we are going to give a general solution for the Laplace equation (Eq. 3.2). Assume that:

$$\Phi(x, y) = X(x)Y(y) \tag{3.21}$$

then, by applying Eq. 3.5, we obtain:

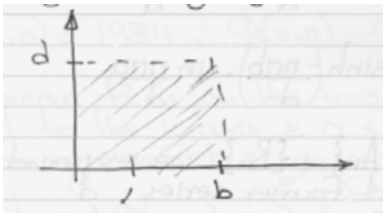
$$\frac{d^2X(x)/dx^2}{X(x)} + \frac{d^2Y(y)/dy^2}{Y(y)} = 0. \tag{3.22}$$

This implies:

$$\underbrace{-\frac{d^2Y(y)}{dy^2}\frac{1}{Y(y)}}_{1} = \underbrace{+\frac{d^2X(x)}{dx^2}\frac{1}{X(x)}}_{2} = \lambda, \tag{3.23}$$

we shall see that this sign selection matches boundary conditions and gives periodic conditions in x. Now, we will consider the solution of the Laplace equation in the rectangle: $0 < x < b, 0 < y < d$, subject to Dirichlet boundary conditions:

$$\Phi(0, y) = f(y), \quad \Phi(b, y) = g(y), \quad \Phi(x, 0) = 0, \quad \Phi(x, d) = 0 \tag{3.24}$$



Hence we obtain

$$\frac{d^2Y(y)}{dy^2} + \lambda Y(y) = 0 \tag{3.25}$$

for $0 < y < d$ and $Y(0) = Y(d) = 0$. This is an eigenvalue problem and the solution for the eigenvalues and eigenfunctions is:

$$\lambda_n = \left(\frac{n\pi}{d}\right)^2, \quad Y_n(y) = \sin\left(\frac{n\pi}{d}y\right) \quad \text{for } n = 1, 2, ... \tag{3.26}$$

The equation for the X-factor is given then as:

$$\frac{d^2X(x)}{dx^2} - (\frac{n\pi}{d})^2X(x) = 0, \quad 0 < x < b, \tag{3.27}$$

this equation has solutions of the form

$$X_n(x) = A_n \sinh(\frac{n\pi}{d})x + B_n \sinh(\frac{n\pi}{d})(x - b), \tag{3.28}$$

then, we substitute Eq. 3.26 and 3.28 into 3.21 we obtain:

$$\Phi(x,y) = \sum_{n=1}^{\infty} (\sin \frac{n\pi}{d} y) \left[A_n \sinh(\frac{n\pi}{d})x + B_n \sinh(\frac{n\pi}{d})(x-b) \right]. \tag{3.29}$$

We now apply the non-homogeneous boundary conditions stated in Eq. 3.24 into Eq. 3.29, resulting in:

$$f(y) = \sum_{n=1}^{\infty} B_n \sinh(\frac{-n\pi b}{d}) \cdot \sin(\frac{n\pi y}{d}) \tag{3.30}$$

$$g(y) = \sum_{n=1}^{\infty} A_n \sinh(\frac{n\pi b}{d}) \cdot \sin(\frac{n\pi y}{d}) \tag{3.31}$$

To evaluate A_n and B_n we expand $f(y)$ and $g(y)$ into generalised Fourier series.

$$f(y) = \sum_{n=1}^{\infty} a_n \sin(\frac{n\pi y}{d}), \quad a_n = \frac{2}{d} \int_0^d f(y) \sin(\frac{n\pi y}{d})\, dy \tag{3.32}$$

$$g(y) = \sum_{n=1}^{\infty} B_n \sin(\frac{n\pi y}{d}), \quad B_n = \frac{2}{d} \int_0^d g(y) \sin(\frac{n\pi y}{d})\, dy \tag{3.33}$$

Hence

$$A_n = -\frac{B_n}{\sinh(\frac{n\pi b}{d})} \tag{3.34}$$

and

$$B_n = \frac{a_n}{\sinh(\frac{n\pi b}{d})} \tag{3.35}$$

3.3.1 Green's functions

In the case of a linear operator, the second path leads to finding the Green's function of the operator

$$\mathcal{L}G(\mathbf{x},\mathbf{x}') = \delta(\mathbf{x}-\mathbf{x}') \; ; \tag{3.36}$$

a general solution is then found by the convolution

$$u(\mathbf{x}) = G * f(\mathbf{x}) \; . \tag{3.37}$$

As an example, we consider the Helmholtz equation

$$\Delta u + k^2 u = f \; .$$

Let us find G. We set $f = \delta(\mathbf{x}-\mathbf{x}')$ and take the Fourier transform

$$(-\mathrm{i}\,\mathbf{k})^2\hat{G}(\mathbf{k}) + a^2\hat{G}(\mathbf{k}) = 1$$
$$\hat{G}(\mathbf{k}) = \frac{1}{a^2-\mathbf{k}^2}$$

The Green's function is then of the form

$$G(\mathbf{x},\mathbf{x}') = \frac{\mathrm{e}^{\pm \mathrm{i}\,a|\mathbf{x}-\mathbf{x}'|}}{4\pi|\mathbf{x}-\mathbf{x}'|} \; .$$

The convolution of Eq. 3.37 makes this approach quite well suited for particle methods as particles are precisely quadrature points. The evaluation at all the particle locations however makes it a $\mathcal{O}(N^2)$ summation. The convolution will thus need to be carried out with fast summation algorithms such as the Fast Multipole Method (FMM) [Barnes and Hut, 1986, Greengard and Rokhlin, 1987].

3.3.2 Discrete solvers

We distinguish mesh relaxation methods, matrix methods and rapid elliptic solvers. We briefly look at the first family.
Consider the elliptic equation

$$\frac{d^2\Phi}{dx^2} = -q$$
$$\frac{\Phi_{i-1}-2\Phi_i+\Phi_{i+1}}{\delta x^2} = q_i \rightarrow$$
$$\bar{\bar{A}}\bar{\Phi} = \bar{q}$$

where $\bar{\bar{A}}$ is an $M \times M$ matrix and $\bar{\Phi}, \bar{q}$ $1 \times M$ vectors. Mesh relaxation methods rely on the splitting of the matrix into two parts $A = B + R$ where B is easily invertible

$$(B+R)\Phi = q$$
$$B\Phi = -R\Phi + q = -(A-B)\Phi + q \; .$$

For mesh relaxation we start with an initial guess Φ^0 and generate a series of iterates according to

$$B\Phi^{n+1} = -(A-B)\Phi^n + q$$

Now as B is easily invertible we can write

$$\Phi^{n+1} = M\Phi^n + B^{-1}q$$

where $M = -B^{-1}A + I$ is the iteration matrix. We define the error of each iteration to be

$$\begin{aligned}\epsilon^n &= \Phi^n - \Phi \\ \epsilon^{n+1} &= M\epsilon^n \\ \epsilon^n &= M^n\epsilon^{(0)}\end{aligned}$$

The behavior of the error is governed by the largest eigenvalue of M since

$$\frac{||\epsilon^n||}{||\epsilon^0||} \leq ||M^n|| \leq ||M||^n \leq \rho^n$$

where $\rho = |\lambda|_{\max}$. The problem of mesh relaxation techniques is that the matrix M has eigenvalues less than or very close to unity.

Jacobi Method $B = diag(A)$ and $\rho = 1 - \frac{1}{2}\frac{\pi^2}{n^2}$.

Gauss-Seidel $B = L(A) =$ lower triangular matrix of A and $\rho = 1 - \frac{\pi^2}{n^2}$

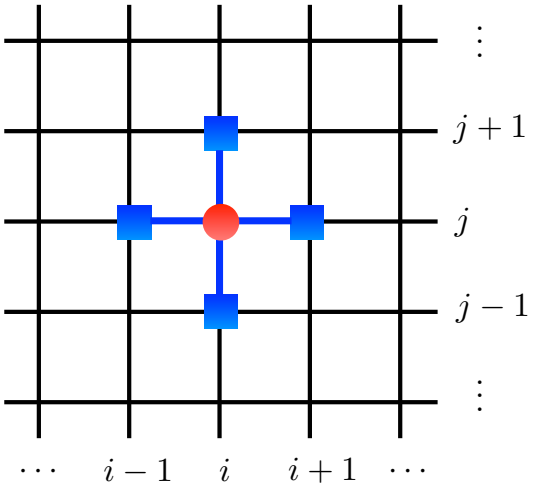
We note that as $n \to \infty$, $\rho \to 1$. Hence a slower and slower convergence. Multigrid methods have been introduced this reason, they work on a coarser mesh (smaller n) to accelerate the decay of low order modes [**?**].

3.4 Solving the Poisson Equation numerically

Here we look at solving the Poisson equation (3.3) numerically, using finite differences on a rectangular grid.

Assume we have a grid with grid spacing $\delta x = \delta y = h$, where we have $M+1$ points in the x-direction and $N+1$ points in the y-direction. The grid node coordinates are labeled by x_i, $i = 0, 1, \ldots, M$, and y_j, $j = 0, 1, \ldots, N$. The boundary points are lie on the grid nodes with indices $i = 0$, $i = M$, $j = 0$ and $j = N$.

Now we use a second order central finite difference scheme to approximate the second order partial differential equation. If we want to compute the second order derivative at the red point i, j, below, we need the values of the function we are differentiating at the blue squares in the figure. This is called the *stencil* of the scheme.



In particular, our approximation to (3.3) becomes

$$\Phi_{i+1,j} - 2\Phi_{i,j} + \Phi_{i-1,j} + \Phi_{i,j+1} - 2\Phi_{i,j} + \Phi_{i,j-1} = h^2 f_{i,j}, \tag{3.38}$$

or

$$\Phi_{i+1,j} - 4\Phi_{i,j} + \Phi_{i-1,j} + \Phi_{i,j+1} + \Phi_{i,j-1} = h^2 f_{i,j}, \tag{3.39}$$

for the interior points $i = 1, 2, \ldots, M-1$ and $j = 1, 2, \ldots, N-1$.

We need to take special care at the boundaries, where we will put the value of the function Φ that is prescribed by the boundary conditions on the right-hand side. For example, for the points $i = 1$ and $j = 2, 3, \ldots, N-2$ we get:

$$\Phi_{2,j} - 4\Phi_{1,j} + \Phi_{1,j+1} + \Phi_{1,j-1} = h^2 f_{1,j} - \Phi_{0,j}, \tag{3.40}$$

where the point with index $(0, j)$ falls on the left boundary and therefore the value of $\Phi_{0,j}$ is prescribed by the boundary conditions.

Now we list all the unknowns in a vector, such that

$$\vec{x} = [\Phi_{1,1}, \Phi_{2,1}, \ldots, \Phi_{M-1,1}, \Phi_{1,2}, \ldots, \Phi_{M-1,N-1}]^T. \tag{3.41}$$

Using this notation, we can rewrite the system of equations given by the finite difference as a matrix-vector equation:

$$A\vec{x} = \vec{f}. \tag{3.42}$$

Writing out the full equation in matrix-vector form for a 6×6 domain ($M = N = 5$), we get

$$
\begin{pmatrix}
-4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -4 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4
\end{pmatrix}
\begin{pmatrix}
\Phi_{1,1} \\ \Phi_{2,1} \\ \Phi_{3,1} \\ \Phi_{4,1} \\
\Phi_{1,2} \\ \Phi_{2,2} \\ \Phi_{3,2} \\ \Phi_{4,2} \\
\Phi_{1,3} \\ \Phi_{2,3} \\ \Phi_{3,3} \\ \Phi_{4,3} \\
\Phi_{1,4} \\ \Phi_{2,4} \\ \Phi_{3,4} \\ \Phi_{4,4}
\end{pmatrix} =
$$

$$
\begin{pmatrix}
h^2 f_{1,1} - \Phi_{0,1} - \Phi_{1,0} \\
h^2 f_{2,1} - \Phi_{2,0} \\
h^2 f_{3,1} - \Phi_{3,0} \\
h^2 f_{4,1} - \Phi_{4,0} - \Phi_{5,1} \\
h^2 f_{1,2} - \Phi_{0,2} \\
h^2 f_{2,2} \\
h^2 f_{3,2} \\
h^2 f_{4,2} - \Phi_{5,2} \\
h^2 f_{1,3} - \Phi_{0,3} \\
h^2 f_{2,3} \\
h^2 f_{3,3} \\
h^2 f_{4,3} - \Phi_{5,3} \\
h^2 f_{1,4} - \Phi_{0,4} - \Phi_{1,5} \\
h^2 f_{2,4} - \Phi_{2,5} \\
h^2 f_{3,4} - \Phi_{3,5} \\
h^2 f_{4,4} - \Phi_{4,5} - \Phi_{5,4}
\end{pmatrix}
$$

We can see from the structure of this matrix that this is a block-tridiagonal matrix, which grows with $(M-1)(N-1)$.

In theory, this matrix-vector equation can be solved for the unknown vector $\vec{x}$ using some direct method. However, for large domains and more than one dimension this becomes prohibitively

expensive. This is why we need to look at iterative solution methods to solve this matrix-vector equation.

3.5 Iterative Solution Methods for Elliptic Equations

The matrices that are being formed here are the result of a numerical approximation. Even when solving the system "exactly" (e.g. via Gauss elimination), we will still have our discretization error. The idea is to solve the matrix equation faster by introducing an error that is less than the error introduced by the discretization. This justifies the use of iterative methods.
Let's look in more detail at the problem above, where we wish to solve $Ax = b$. Let $A = P - Q$, then

$$(P - Q)x = b, \tag{3.43}$$

or

$$Px = Qx + b \tag{3.44}$$

Iterative solutions are now constructed as:

$$Px^{k+1} = Qx^k + b, \quad k = 0, 1, 2, \ldots \tag{3.45}$$

For this iterative scheme to work and to be efficient, we require two properties:

- the iterations must converge
- matrix P must be easily invertible

On the first point, convergence, we can introduce the error at iteration k as

$$\varepsilon^k = x - x^k, \tag{3.46}$$

where x is the exact solution of the system and x^k is the approximate solution after iteration k.
Subtracting now equation (3.45) from equation (3.44) and substituting the above error definition, we get

$$P(x - x^{k+1}) = Q(x - x^k) \tag{3.47}$$

$$P\varepsilon^{k+1} = Q\varepsilon^k. \tag{3.48}$$

$$\varepsilon^{k+1} = (P^{-1}Q)\varepsilon^k. \tag{3.49}$$

Over k iterations, we then get

$$\varepsilon^k = (P^{-1}Q)^k\varepsilon_0, \tag{3.50}$$

where ε_0 is the error of the initial guess.
For convergence we need $\lim_{k\to\infty} \varepsilon^k \to 0$. With λ_i the eigenvalues of $P^{-1}Q$, we therefore require that

$$\rho = |\lambda_i|_{\max} < 1, \tag{3.51}$$

where ρ is called the spectral radius of the matrix.
The matrix $P^{-1}Q$ is thus important in the convergence of the method. For this reason, this matrix is also referred to as the iteration matrix, denoted by R.
What is left is to choose the definitions of P and Q. For this choice, different algorithms exists, some of which will be explained below. In this explanation, we will always assume that we decompose the matrix A into a diagonal part and the negative upper and lower triangular parts: $A = D - L - U$.

3.5.1 Jacobi Iteration

Here

$$P_J = D \tag{3.52}$$

$$Q_J = D - A = L + U \tag{3.53}$$

where $D = \text{diag}(A)$ is the diagonal matrix of A. We thus have for the iteration matrix $R_J = D^{-1}(L+U)$. Without rewriting the full matrices, we note that $D = -4I$, where I is the identity matrix, and Q_J can be produced from A by replacing 1 with -1 and -4 with 0.
For our system of equations $A\Phi = R$, we find the following iteration procedure:

$$\Phi^{k+1} = P_J^{-1} Q_J \Phi^k + P_J^{-1} R \tag{3.54}$$

$$= -\frac{1}{4} Q_J \Phi^k - \frac{1}{4} R. \tag{3.55}$$

For each grid node i, j, this means we have the following iteration step:

$$\Phi_{ij}^{k+1} = \frac{1}{4}\left(\Phi_{i-1,j}^k + \Phi_{i+1,j}^k + \Phi_{i,j-1}^k + \Phi_{i,j+1}^k\right) - \frac{1}{4} R_{ij}. \tag{3.56}$$

Note that we never form the matrix A, and so we don't need to store it.

Convergence To study the convergence of the Jacobi iteration, we have to compute the eigenvalues of $P_J^{-1}Q_J$ so we can test if equation (3.51) holds. As we've seen before, from the definition of P_J and Q_J it follows that

$$P_J^{-1} Q_J = -\frac{1}{4} Q_J. \tag{3.57}$$

We are interested in the eigenvalues of $P_J^{-1}Q_J$. We can show that the eigenvalues are

$$\lambda_{m,n}^J = \frac{1}{2}\left(\cos\left(\frac{m\pi}{M}\right) + \cos\left(\frac{n\pi}{N}\right)\right), \tag{3.58}$$

for $m = 1, \ldots, M-1$ and $n = 1, \ldots, N-1$.
Therefore

$$|\lambda_{m,n}^J| < 1, \tag{3.59}$$

and so we have convergence.
To study the rate of convergence, we recall equation (3.50) and observe that the rate of convergence is determined by the largest values of $|\lambda_{m,n}^J|$. This happens, for instance, when $m = M-1$: then we have $m/M \approx 1$ (especially when M gets very large) and so the maximum eigenvalues get close to 1. This also happens in the case of $m = 1$ and $n = 1$, which is shown in more detail here. We have the following eigenvalue corresponding to $m = n = 1$:

$$\lambda_{1,1}^J = \frac{1}{2}\left(\cos\left(\frac{\pi}{M}\right) + \cos\left(\frac{\pi}{N}\right)\right). \tag{3.60}$$

Now we can do a Taylor expansion for the cosines, and discarding the higher order terms we get:

$$\lambda_{1,1}^J \approx 1 - \frac{1}{4}\left(\frac{\pi^2}{M^2} + \frac{\pi^2}{N^2}\right), \tag{3.61}$$

This means that for large M and N we have $\lambda_{1,1}^J \approx 1$. We conclude that the maximum eigenvalues of the iteration matrix for the Jacobi iteration are $\lambda_{\max}^J \approx 1$. On the basis of this, together with equation (3.50), we conclude that the Jacobi iteration method has a slow convergence.

3.5.2 Gauss-Seidel iteration

Observing equation (3.56), we note that when we arrive at $\Phi_{i,j}^{k+1}$, we have already computed $\Phi_{i-1,j}^{k+1}$ and $\Phi_{i,j-1}^{k+1}$, so we can use those values to be more accurate without extra cost. This observation forms the basis of the Gauss-Seidel iteration method, where we reuse the grid points that we updated already in the same iteration. In particular, we have the following definitions:

$$P_{GS} = D - L \tag{3.62}$$

$$Q_{GS} = U. \tag{3.63}$$

where $A = D - L - U$ and therefore $-L$ and $-U$ are the lower and upper triangular matrices of A, respectively.
For the Gauss-Seidel iterations we get:

$$\Phi_{ij}^{k+1} = \frac{1}{4}\left(\Phi_{i-1,j}^{k+1} + \Phi_{i+1,j}^{k} + \Phi_{i,j-1}^{k+1} + \Phi_{i,j+1}^{k}\right) - \frac{1}{4}R_{ij}. \tag{3.64}$$

What are the convergence characteristics of the Gauss-Seidel method? We can show that the eigenvalues of $P_{GS}^{-1}Q_{GS}$ are

$$\lambda_{m,n}^{GS} = \frac{1}{4}\left(\cos\left(\frac{m\pi}{M}\right) + \cos\left(\frac{n\pi}{N}\right)\right)^2 \tag{3.65}$$

$$= \left(\lambda_{m,n}^{J}\right)^2, \tag{3.66}$$

for $m = 1, \ldots, M-1$ and $n = 1, \ldots, N-1$.
This means the Gauss-Seidel method converges twice as fast as the Jacobi method.

3.5.3 Successive over-relaxation (SOR)

Recall that for Gauss-Seidel, we have

$$(D - L)\Phi^{k+1} = U\Phi^{k} + b, \tag{3.67}$$

where $A = D - L - U$.
Now we introduce d, which is defined as the change of solution between iterations $d = \Phi^{k+1} - \Phi^{k}$.
We can use d to write:

$$\Phi^{k+1} = \Phi^{k} + d. \tag{3.68}$$

To create our new method, we multiply d in the above equation by a scaling factor ω:

$$\Phi^{k+1} = \Phi^{k} + \omega d. \tag{3.69}$$

We can use this scaling factor by setting $\omega > 1$ to accelerate convergence. Now how can we determine ω? This is where Successive Over-Relaxation comes in. First we perform a test-step according to the Gauss-Seidel method to obtain an intermediate solution $\tilde{\Phi}^{k+1}$:

$$D\tilde{\Phi}^{k+1} = L\Phi^{k+1} + U\Phi^{k} + b. \tag{3.70}$$

Now we define the solution at iteration step $k+1$ as a linear combination between the solution at iteration step k, and the Gauss-Seidel approximation $\tilde{\Phi}^{k+1}$:

$$\Phi^{k+1} = \Phi^k + \omega\left(\tilde{\Phi}^{k+1} - \Phi^k\right). \tag{3.71}$$

$$= \left(I - \omega D^{-1}L\right)^{-1}\left((1-\omega)I + \omega D^{-1}U\right)\Phi^k + \left(I - \omega D^{-1}L\right)^{-1}\omega D^{-1}b. \tag{3.72}$$

If A is the matrix arising from central differences of the Poisson equation in a regular domain, then

$$\lambda^{1,2} = \frac{1}{2}\left(\pm\mu\omega \pm \sqrt{\mu^2\omega^2 - 4(\omega-1)}\right),$$

where μ are the Jacobi eigenvalues.
We find that

$$\omega_{m,n}^{\text{opt}} = \frac{2}{1+\sqrt{1-\mu_{\max}^2}}$$

3.6 Multigrid

3.6.1 Residual

We introduce the residual. Starting with an initial guess v^0 for the iteration, we can compute Av to give us a right-hand side f':

$$Av^0 = f'.$$

Since v^0 is an approximation that is usually not equal to the exact solution u, we have $f' \neq f$ and we can introduce the residual:

$$r^0 = f - f' = f - Av^0.$$

In general, after k iterations, the residual r^k is defined as:

$$r^k = f - Av^k.$$

3.6.2 1D setting and weighted Jacobi

The following notes are partially taken from [Briggs et al., 2000].
For the following, we change the setting of our explanations. We consider a 1D Poisson problem:

$$\frac{\mathrm{d}^2u}{\mathrm{d}x^2} = -f \tag{3.73}$$

$$u(0) = u(1) = 0. \tag{3.74}$$

which we can discretize using central second order finite differences to get

$$-u_{i-1} + 2u_i - u_{i+1} = h^2 f_i \qquad \text{for } 1 \leq i \leq N-1 \tag{3.75}$$

$$u_0 = u_N = 0. \tag{3.76}$$

Or, in matrix-vector notation

$$Au = f.$$

In this case matrix A is a tridiagonal matrix with 2 along the diagonal and -1 along the off-diagonal entries.

Denoting the solutions from the iterations by $\vec{v}^k$, a basic Jacobi method gives us:

$$v_i^{k+1} = \frac{1}{2}\left(v_{i-1}^k + v_{i+1}^k + h^2 f_i\right), \qquad \text{for } 1 \leq i \leq N-1.$$

A slight but important modification of the Jacobi method is called the weighted Jacobi method. In this case, we update the solution according to:

$$\vec{v}^* = R_J \vec{v}^k + D^{-1}\vec{f} \tag{3.77}$$
$$\vec{v}^{k+1} = (1-\omega)\vec{v}^k + \omega\vec{v}^* \tag{3.78}$$
$$= \left((1-\omega)I + \omega R_J\right)\vec{v}^k + \omega D^{-1}\vec{f} \tag{3.79}$$
$$= R_\omega \vec{v}^k + \omega D^{-1}\vec{f}, \tag{3.80}$$

where

$$R_\omega = (1-\omega)I + \omega D^{-1}(L+U) \tag{3.81}$$
$$= I + \omega(D^{-1}(L+U) - I). \tag{3.82}$$

If we now look at the structure of the matrix $D^{-1}(L+U) - I$, we can see that this matrix is a tridiagonal matrix with -1 along the main diagonal and $\frac{1}{2}$ along the off-diagonal entries. Comparing to our original matrix A, we can observe that the matrix $D^{-1}(L+U) - I$ is equivalent to $-\frac{1}{2}A$. In other words, we can write R_ω as

$$R_\omega = I - \frac{\omega}{2}A. \tag{3.83}$$

This means that, if we are interested in the convergence characteristics of the weighted Jacobi method, we have to look at the eigenvalues of R_ω, which are a function of the eigenvalues of the original matrix A:

$$\lambda(R_\omega) = 1 - \frac{\omega}{2}\lambda(A). \tag{3.84}$$

For the one-dimensional case we are discussing here, we can derive (exercise!) that the eigenvalues and eigenvectors of A are given as

$$\lambda_k(A) = 4\sin^2\left(\frac{k\pi}{2N}\right) \tag{3.85}$$
$$w_{k,j} = \sin\left(\frac{jk\pi}{N}\right). \tag{3.86}$$

where k, $1 \leq k \leq N-1$, is the wavenumber and j, $0 \leq j \leq N$, indicates the component of the eigenvector.

Note: from here on the superscript k denotes the iteration number, the subscript k denotes the mode or wavenumber.

We now can write the eigenvalues and eigenvectors of the matrix R_ω:

$$\lambda_k(R_\omega) = 1 - 2\omega\sin^2\left(\frac{k\pi}{2N}\right) \tag{3.87}$$
$$w_{k,j} = \sin\left(\frac{jk\pi}{N}\right). \tag{3.88}$$

Note that for $0 < \omega \leq 1$, the values of $|\lambda_k(R_\omega)|$ is always smaller than 1, and so the weighted Jacobi method converges. We will look closer at its convergence properties in the next subsection.

3.6.3 Smoothing property

For the next step, we are interested in decomposing the error in the eigenbasis of the matrix R_ω (and also of matrix A, since they share the same eigenvalues), to study how each frequency of the error is affected by the weighted Jacobi iteration. To this end, we expand the initial error in its modes:

$$\varepsilon^0 = \sum_{k=1}^{N-1} c_k \vec{w}_k, \tag{3.89}$$

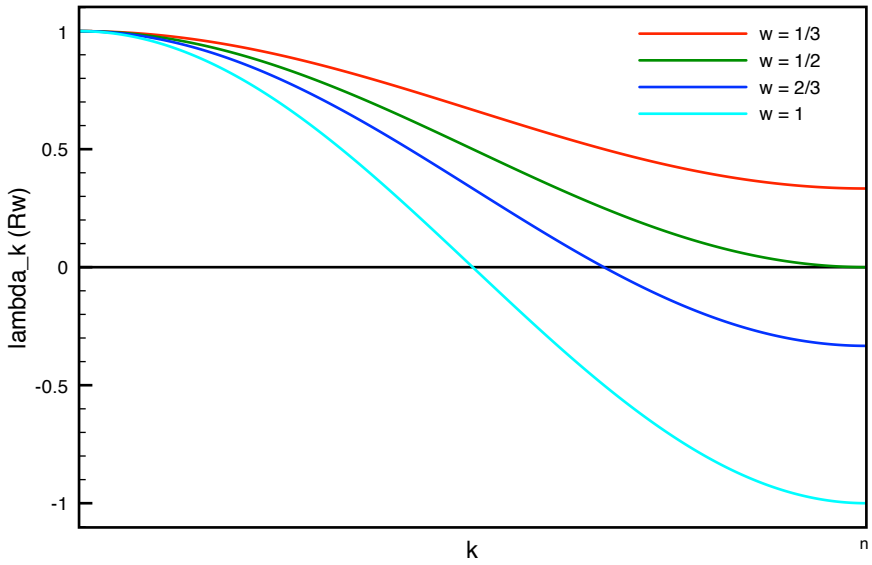
where $c_k \in \mathbb{R}$ is the "amount" of mode k in the initial error, and $\vec{w}_k$ are the eigenvectors of R_ω (the Fourier modes).
We have seen before that after m iterations, we have the following error

$$\varepsilon^{m+1} = (R_\omega)^m \varepsilon^0.$$

Substituting the decomposition of ε^0, we get

$$\varepsilon^{m+1} = (R_\omega)^m \sum_{k=1}^{N-1} c_k \vec{w}_k = \sum_{k=1}^{N-1} c_k (R_\omega)^m \vec{w}_k = \sum_{k=1}^{N-1} c_k \left(\lambda_k(R_\omega)\right)^m \vec{w}_k,$$

where we applied the property of eigenvalues $R_\omega \vec{w}_k = \lambda_k(R_\omega)\vec{w}_k$ repeatedly.
Now we see the importance of the eigenvalues of the iteration matrix: the kth mode of the error is reduced by λ_k at each iteration. If $|\lambda_k|$ is close to zero, then this mode of the error will decrease quickly; if $|\lambda_k|$ on the other hand is close to one, then this mode of the error will decrease very slowly.
In the case of weighted Jacobi, the value of λ_k for different k's depends on the smoothing coefficient ω. Below are the eigenvalues plotted for different values of ω, as a function of k. Note that the actual λ_k is not defined for $k = 0$ and $k = N$, but the plot is extended to these values for clarity.



For the original Jacobi method, $\omega = 1$ and the eigenvalues for both the smooth $k < N/2$ and the oscillatory modes $k \geq N/2$ are damped very badly at the ends of the spectrum. Decreasing ω increases the convergence for the oscillatory modes, but it hardly affects the very smooth modes. This can be seen from an expansion of $\lambda_1(R_\omega)$:

$$\lambda_1 = 1 - 2\omega \sin^2\left(\frac{\pi}{2N}\right) = 1 - 2\omega \sin^2\left(\frac{\pi h}{2}\right) \approx 1 - \frac{\omega\pi^2 h^2}{2}.$$

This shows that for decreasing grid spacing, $h \to 0$, the value of λ_1 approaches 1 for all possible values of ω.
This behavior is called the *smoothing property*: many relaxation schemes have the smoothing property, where oscillatory modes of the error are eliminated effectively, but smooth modes are damped very slowly.

3.6.4 Coarse Grid Correction

The first step towards the multigrid is to realize how to use the smoothing property to our advantage. Observe what happens when we transfer a low-frequency mode onto a coarser grid. For $k = 1, 2, \ldots N/2$, the kth mode is preserved on the coarse grid:

$$w^h_{k,2j} = \sin\left(\frac{2jk\pi}{N}\right) = \sin\left(\frac{jk\pi}{N/2}\right) = w^{2h}_{k,j}$$

(Note that $w^h_{N/2} \to 0$ on the coarse grid). The wavenumber of the smooth mode is preserved, but now the total number of modes that can be represented by the grid has halved and so the mode has moved up the wavespectrum: it is represented on the grid as a high-frequency mode. For example, imagine the $k = 4$ mode on $N = 12$ grid. This mode is the 4th out of a total possible of 11 modes: $k/N = 1/3$ (k/N is frequency). Taking a coarser grid we have the same mode $k = 4$ on a $N = 6$ grid, which means our mode is now the 4th out of a total possible of 5 modes that can be represented on the grid, and now $k/N = 2/3$. The smooth wave (low frequency) on the fine grid becomes an oscillatory wave (high frequency) on the coarse grid.
What happens to the waves on the oscillatory part of the spectrum when transferring to the coarse grid? For $k > N/2$, the mode w^h_k is invisible on the coarse grid, but its effect is observed in a lower-frequency mode. This is called aliasing:

$$\begin{aligned} w^h_{k,2j} &= \sin\left(\frac{2jk\pi}{N}\right) && (3.90)\\ &= -\sin\left(\frac{2\pi j(N-k)}{N}\right) && (3.91)\\ &= -\sin\left(\frac{\pi j(N-k)}{N/2}\right) && (3.92)\\ &= -w^{2h}_{(N-k),j} && (3.93) \end{aligned}$$

Using these properties, we realize that we can use coarse grids to compute an improved initial guess for the fine-grid relaxation. This has two advantages:

- relaxation on the coarse-grid is much cheaper (1/2 as many points in 1D, 1/4 as many points in 2D, 1/8 as many points in 3D)

- relaxation on the coarse-grid has a marginally better convergence rate: $1 - \mathcal{O}(4h^2)$ instead of $1 - \mathcal{O}(h^2)$.

In short then, we proceed as follows. Let v be an approximation to the solution $Au = f$. The residual is $r = f - Av$, and the error $e = u - v$ satisfies $Ae = r$. After relaxing $Au = f$ on the fine grid, the error will be smooth since those components are the least affected by our relaxation. On a coarse grid, however, the error appears more oscillatory, and relaxation will be more effective. Therefore we transfer the error to a coarser grid and relax or solve the residual equation $Ae = r$ with initial guess of $e = 0$. This reduces the high-frequency components of the error on the coarse grid. After this coarse-grid step, we transfer the error back to the fine grid and update our original guess v by $v = v + e$. This can be described in the following algorithm:

1. Relax ν_1 times on $A^h u^h = f^h$ on Ω^h with initial guess v^h
2. Compute the fine-grid residual $r^h = f^h - Av^h$
3. Restrict the residual to the coarse grid Ω^{2h} : $r^{2h} = I_h^{2h} r^h$
4. Solve/Relax on $A^{2h} e^{2h} = r^{2h}$ on Ω^{2h} for the e^{2h} error
5. Interpolate back to the fine grid: $e^h = I_{2h}^h e^{2h}$
6. Correct the fine-grid approximation: $v^h = v^h + e^h$
7. Relax ν_2 times $A^h u^h = f$ on Ω^h with initial guess v^h.

Some comments on this algorithm are in place. First, note that we use the operators I_h^{2h} and I_{2h}^h to transfer solutions from the fine to the coarse grid and vice versa. These steps are called restriction and prolongation, respectively, and we will give more details for them in the next section. We will assume that there is always a factor of 2 between the grid spacing of the fine and coarse grids. Second, in the solve/relax step on the coarse grid we have introduced the matrix A^{2h}, without defining it. In this course we will take matrix A^{2h} as the result of discretizing the original problem on the coarse grid Ω^{2h}. Third, the integers ν_1 and ν_2 are parameters in the scheme that control the number of relaxation sweeps before and after visiting the coarse grid. They are usually fixed at the start of the algorithm.

3.6.5 Prolongation

Transferring information from the coarse and the fine grid is called prolongation, or interpolation. The operator to perform this step is denoted by I_{2h}^h and is defined as

$$v^h = I_{2h}^h v^{2h}$$

We will use here a simple linear interpolation, in 1D:

$$v_{2j}^h = v_j^{2h} \tag{3.94}$$

$$v_{2j+1}^h = \frac{1}{2}(v_j^{2h} + v_{j+1}^{2h}), \tag{3.95}$$

for $0 \leq j \leq \frac{N}{2} - 1$.

In two-dimensions, we use bi-linear interpolation and get the following definitions

$$v^h_{2i,2j} = v^{2h}_{i,j} \tag{3.96}$$

$$v^h_{2i+1,2j} = \frac{1}{2}(v^{2h}_{i,j} + v^{2h}_{i+1,j}) \tag{3.97}$$

$$v^h_{2i,2j+1} = \frac{1}{2}(v^{2h}_{i,j} + v^{2h}_{i,j+1}) \tag{3.98}$$

$$v^h_{2i+1,2j+1} = \frac{1}{4}(v^{2h}_{i,j} + v^{2h}_{i,j+1} + v^{2h}_{i+1,j} + v^{2h}_{i+1,j+1}) \tag{3.99}$$

$$\tag{3.100}$$

for $0 \leq i, j \leq \frac{N}{2} - 1$.
Note that this interpolation works best when u is smooth, and so the nested iteration is most effective when the error is smooth. Note also that the order of the interpolation should in general not be more than the order of the finite-difference discretization, to prevent injection of additional error terms. In our case, the finite-difference discretization is second order whereas the interpolation is first order.

3.6.6 Restriction

To transfer information from the fine to the coarse grid is called restriction. The restriction operator is denoted by I^{2h}_h and is defined by

$$v^{2h} = I^{2h}_h v^h.$$

We use the full-weighted scheme, where we get the solution at the coarse grid points from averaging the fine-grid values. In 1D:

$$v^{2h}_j = \frac{1}{4}\left(v^h_{2j-1} + 2v^h_{2j} + v^h_{2j+1}\right), \tag{3.101}$$

for $1 \leq j \leq \frac{N}{2} - 1$.
In two-dimensions this is defined as follows:

$$v^{2h}_{i,j} = \frac{1}{16}(v^h_{2i-1,2j-1} + v^h_{2i-1,2j+1} + v^h_{2i+1,2j-1} + v^h_{2i+1,2j+1} \tag{3.102}$$

$$+ 2(v^h_{2i,2j-1} + v^h_{2i,2j+1} + v^h_{2i-1,2j} + v^h_{2i+1,2j}) \tag{3.103}$$

$$+ 4v^h_{2i,2j}), \tag{3.104}$$

for $1 \leq i, j \leq \frac{N}{2} - 1$.

3.6.7 V-cycle

In the coarse-grid correction algorithm, we have written the following step: "Solve/Relax on $A^{2h}e^{2h} = r^{2h}$ on Ω^{2h} for the e^{2h} error". However, we haven't discussed yet how to approach this. In fact, one may realize that solving $A^{2h}e^{2h} = r^{2h}$ on Ω^{2h} is a very similar problem to the original problem of solving $A^h u^h = f^h$ on Ω^h, and therefore we can expect similar issues in the convergence of an iterative relaxation method. We can however use the same approach for the problem on Ω^{2h}: we can solve this problem with the aid of an even coarser grid. Applying this reasoning on each grid level gives rise to a recursive algorithm, and eventually one arrives at the V-cycle Multigrid method:

- Relax ν_1 times on $A^h u^h = f^h$ with given initial guess v^h
- Compute $f^{2h} = I_h^{2h} r^h$
  - Relax ν_1 times on $A^{2h} u^{2h} = f^{2h}$ with initial guess $v^{2h} = 0$
  - Compute $f^{4h} = I_{2h}^{4h} r^{2h}$
    - Relax ν_1 times on $A^{4h} u^{4h} = f^{4h}$ with initial guess $v^{4h} = 0$
    - Compute $f^{8h} = I_{4h}^{8h} r^{4h}$

      .

      .

      - Solve $A^{Lh} u^{Lh} = f^{Lh}$

      .

      .

    - Correct $v^{4h} = v^{4h} + I_{8h}^{4h} v^{8h}$
    - Relax ν_2 times on $A^{4h} u^{4h} = f^{4h}$ with initial guess v^{4h}
  - Correct $v^{2h} = v^{2h} + I_{4h}^{2h} v^{4h}$
  - Relax ν_2 times on $A^{2h} u^{2h} = f^{2h}$ with initial guess v^{2h}
- Correct $v^h = v^h + I_{2h}^h v^{2h}$
- Relax ν_2 times on $A^h u^h = f^h$ with initial guess v^h

Some explanation is required on the notation used in this algorithm. In the algorithm, the symbols u, v and f denote generic solution vectors, approximation vectors and right-hand side vectors, respectively. In the second step of the algorithm, where it is written write $f^{2h} = I_h^{2h} r^h$, the notation f^{2h} instead of r^{2h} is use because f^{2h} is just the right-hand side vector for the next step. Similarly, in the next step it is written $A^{2h} u^{2h} = f^{2h}$ instead of $A^{2h} e^{2h} = r^{2h}$, since e^{2h} is just the solution vector for this relaxation step.
In fact, with this notation, a recursive definition of the V-cycle scheme can easily be found. Writing the V-cycle as a function $V^h()$ taking in as arguments the (generic) initial guess v^h and the (generic) right-hand side f^h, we get algorithm 1.

Algorithm 1 V-cycle algorithm

$$v^h \leftarrow V^h(v^h, f^h)$$

1. Relax ν_1 times on $A^h u^h = f^h$ with initial guess v^h
2. If Ω^h is the coarsest grid, go to step 4
   Else:

$$f^{2h} \leftarrow I_h^{2h}(f^h - A^h v^h)$$
$$v^{2h} \leftarrow 0$$
$$v^{2h} \leftarrow V^{2h}(v^{2h}, f^{2h})$$

3. Correct $v^h \leftarrow v^h + I_{2h}^h v^{2h}$
4. Relax ν_2 times on $A^h u^h = f^h$ with initial guess v^h

A V-cycle scheme with ν_1 relaxation sweeps before the correction step and ν_2 relaxation sweeps after the correction step is referred to as a V(ν_1, ν_2)-cycle.
Note that there is still one "solve" step remaining in the algorithm: we have to solve $A^{Lh}u^{Lh} = f^{Lh}$. However, since now we are working on a very coarse grid with grid spacing Lh, this solution step is usually trivial. For example, we can choose our coarsest grid in 2D as a grid with $N = M = 2$, which means we have a 3×3 grid including the boundary points, which means we have only one interior point. Since solving for the single interior point only requires information from the boundary points, we get an exact expression for the solution on this grid as a function of the boundary conditions.

3.6.8 W-cycle

A second scheme that is often used is the W-cycle scheme, which is very similar to the V-cycle scheme but includes an additional recursive call (see algorithm 2).

Algorithm 2 W-cycle algorithm

$$v^h \leftarrow W^h(v^h, f^h)$$

1. Relax ν_1 times on $A^h u^h = f^h$ with initial guess v^h
2. If Ω^h is the coarsest grid, go to step 4
   Else:

$$\begin{aligned} f^{2h} &\leftarrow I_h^{2h}(f^h - A^h v^h) \\ v^{2h} &\leftarrow 0 \\ v^{2h} &\leftarrow W^{2h}(v^{2h}, f^{2h}) \\ v^{2h} &\leftarrow W^{2h}(v^{2h}, f^{2h}) \end{aligned}$$

3. Correct $v^h \leftarrow v^h + I_{2h}^h v^{2h}$
4. Relax ν_2 times on $A^h u^h = f^h$ with initial guess v^h

3.6.9 Full Multigrid (FMG) V-cycle

The Full Multigrid algorithm combines nested iteration with the V-cycle. Nested iteration uses coarse grids to obtain improved initial guesses for the fine-grid problems. For the V-cycle, for instance, we always have to start with an initial guess v^h for the first relaxation on the fine grid Ω^h. The idea of nested iteration proposes to find this initial guess by solving the problem on a coarser grid Ω^{2h}. However, now we need an initial guess for the problem on Ω^{2h}, which, according to the nested iteration idea, can be found by solving a problem on Ω^{4h}. This leads to a scheme that transfers the initial right-hand side f^h to coarser and coarser grids, and the initial problem is then solved on the coarsest grid. This solution is used as an initial guess for the problem on the coarser grid, and so we move up the hierarchy of grids from coarse to fine. The algorithm can be written as follows:

Initialize $f^{2h} \leftarrow I_h^{2h} f^h$, $f^{4h} \leftarrow I_{2h}^{4h} f^{2h}$, ...

- Solve or relax on coarsest grid
  .
  .
  .
- $v^{4h} \leftarrow I_{8h}^{4h} v^{8h}$
- $v^{4h} \leftarrow V^{4h}(v^{4h}, f^{4h})$, ν_0 times
- $v^{2h} \leftarrow I_{4h}^{2h} v^{4h}$
- $v^{2h} \leftarrow V^{2h}(v^{2h}, f^{2h})$, ν_0 times
- $v^h \leftarrow I_{2h}^h v^{2h}$
- $v^h \leftarrow V^h(v^h, f^h)$, ν_0 times

For the Full Multigrid we can also use a recursive formulation that maps well to an actual implementation, see algorithm 3

Algorithm 3 Full Multigrid algorithm

$$v^h \leftarrow FMG^h(f^h)$$

1. If Ω^h is the coarsest grid, set $v^h \leftarrow 0$ and go to step 3
   Else:

$$f^{2h} \leftarrow I_h^{2h}(f^h)$$
$$v^{2h} \leftarrow FMG^{2h}(f^{2h})$$

2. Correct $v^h \leftarrow I_{2h}^h v^{2h}$
3. $v^h \leftarrow V^h(v^h, f^h)$, ν_0 times

As can be seen from this algorithm, each V-cycle is preceded by a coarse grid V-cycle designed to provide the best initial guess possible.
We close this section with a few comments and examples. The choice among methods available for the solution of the field equations will depend on the characteristics of the equations and the boundary conditions.
Equations:

- linearity
- dimensionality
- separability
- variation of coefficients

Boundary conditions:

- region rectangular or arbitrary
- mixed or simple condition
- is it isolated?

3.7 Fast Multipole Method

Note. This section is copied from Chapter 3 of the PhD thesis of Prof. Koumoutsakos [Koumoutsakos, 1993]

Vortex methods , have attracted the interest of many researchers for the study of unsteady incompressible flows at relatively high Reynolds numbers. In such flows the vortical structures, which are the key to describing the whole flow field, tend to be confined in limited regions of the domain and vortex methods inherently adapt to resolve those regions. In this method the vorticity field is described by a set of N particles. The velocity field is computed then at each of the particles, resulting in a set of ODE's that needs to be integrated in time to determine their trajectories. This may alternatively be considered as an N-body problem for the particles . The simplest method for computing the velocities on the particles requires work that is proportional to N^2 as all pairwise interactions need to be computed. This renders the method prohibitive if one wants to use large numbers of particles to resolve all spatial scales in the flow.
The N-body problem appears in a diverse number of scientific fields (astrophysics, plasma physics, etc.) and techniques for the reduction of the computational cost have been addressed in many different perspectives. Solution methods to this problem may be classified in two broad categories: The *hybrid methods* (particle-mesh) and *direct* (particle only) methods. An extensive survey on *hybrid methods* may be found in the book of Hockney and Eastwood [Hockney and Eastwood, 1981]. The cost of the hybrid methods is of $\mathcal{O}(\mathrm{N} + \mathrm{M}\log\mathrm{M})$ (where M is the number of mesh points), but their performance degrades when the clustering of the particles is highly non-uniform and they tend to introduce numerical diffusion. The direct methods compute the approximate velocity field as induced by clusters of particles using a certain number of expansions for each cluster. This number is determined by the accuracy required in the solution of the problem and the

type of the field that is approximated. A hierarchical (tree) data structure is associated with the particles and is implemented in order to determine when those expansions may be used instead of the exact pairwise interactions to preserve the accuracy of the solution. The tree is used to establish interaction lists for the particles.
Originally the *direct* technique was developed for the simulation of gravitational problems by [Barnes and Hut, 1986] and [Appel, 1985] developed a data structure to facilitate such computations. The application of direct schemes in vortex methods has been primarily exploited by [Greengard and Rokhlin, 1987] and [van Dommelen and Rudensteiner, 1989]).They may be referred to as the box-box *(BB)* and particle-box *(PB)* algorithms respectively. Their computational cost theoretically scales as N or $N\log N$, respectively, but their applicability and efficient use of available supercomputer architectures (vector/parallel) is what really determines the cost of these methods. The issue that arises then in the development of these codes is the question of parallel or vector implementation? Although this question is usually answered by the availability of resources it seems worth exploring the possibilities. The nature of the algorithm was originally thought to be such that it would lend itself easier to a parallel implementation [Barnes and Hut, 1986] the main constraint being the recursive descent of the data structure. A naive implementation of this algorithm on a vector computer would require that this recursive scheme is unrolled into a sequence of iterative procedures. Such an implementation however would result in inefficient vectorization and no advantage would be taken of the pipelining capabilities.
The parallel implementation of the methods has been addressed at various degrees of sophistication and for a diverse number of problems (see for example [Pépin and Leonard, 1990], [Katzenelson, 1989] and [Salmon, 1991]). Vectorization of the method has also been successfully addressed in works by [van Dommelen and Rudensteiner, 1989] for vortex methods using the *PB* algorithm. Recently (1990) Hernquist, Makino, and Barnes published a series of papers demonstrating that the tree descent can indeed be vectorized. They applied their strategy to the Barnes - Hut algorithm obtaining orders of magnitude increase in the computational speed of the algorithm. Usually the most time consuming part of these schemes has been the building and descending of the tree in order to establish interaction lists. Once these have been established the velocity field of the particles may easily be determined. The vectorization of the *BB* algorithm for vortex methods is addressed in this work and its efficiency is compared to that of a *PB* scheme.
A data structure is implemented which lends itself to vectorization, in the construction and descent level, combining some of the ideas presented in the works of Barnes, Hernquist and Makino. It is applied to both (*PB* and *BB*) algorithms and comparisons are made as to what efficiencies may be obtained. The timings compare favorably with those reported in the past [van Dommelen and Rudensteiner, 1989] and it is demonstrated that an efficient implementation of the *BB* scheme can outperform the *PB* one for numbers of particles more than a few thousand.

3.7.1 Multipole Methods for the Velocity Evaluation

In the context of vortex methods the vorticity field is discretized by a set of elementary vortices whose individual field is expressed by a cut-off function $\phi(x)$. The superposition of these fields determines the vorticity ω, at any location $\mathbf{x}$, as:

$$\omega(\mathbf{x}) = \sum_{n=1}^{N} \Gamma_n \phi(\mathbf{x} - \mathbf{x}_n) \tag{3.105}$$

If $\phi(\mathbf{x})$ is a δ-function or we are interested in the velocity field (u, v) in a location (x, y) away from the particles, the velocity field is expressed in complex form as:

$$\boxed{V(Z) = \frac{i}{2\pi} \sum_{n=1}^{N} \frac{\Gamma_n}{Z - Z_n}} \tag{3.106}$$

where $Z = x + iy$ and $V = u - iv$. In order to compute the velocity at each one of the particles the above sum implies $\mathcal{O}(N^2)$ operations. To reduce this computational cost the geometrical distribution of the vortices is exploited. The key observation is that the velocity induced by a cluster of particles need not be computed directly from its individual members. Instead the velocity field induced by a group of M particles clustered around a centre Z_M may be approximated by a finite number (P) of multipole expansions. At distances greater than the radius R_M of this cluster this approximation converges geometrically. This is the basis for the *PB* scheme. The *BB* scheme introduces one more step. The expansions of a certain cluster may be translated and computed with the desired accuracy at the center of another cluster. Subsequently those expansions are used to determine the velocity of the particles in the second cluster.
The derivation of those expansions is based on the following two identities of complex numbers:

$$\frac{1}{1-z} = \sum_{n=0}^{\infty} z^n; \qquad \text{for} |z| \leq 1 \tag{$*$}$$

$$\sum_{k=0}^{P} \alpha_k (Z - Z_0)^k = \sum_{l=0}^{P} \left(\sum_{k=l}^{P} \alpha_k \binom{k}{l} (-Z_0)^{k-l} \right) Z^l. \tag{$**$}$$

Adding and subtracting Z_M on the denominator of (3.106) the velocity field induced by the cluster may be expressed as :

$$V(Z) = \frac{i}{2\pi} \frac{1}{Z - Z_M} \sum_{m=1}^{M} \frac{\Gamma_m}{1 + (Z_M - Z_m)/(Z - Z_M)}$$

By expanding now the denominator inside the sum using $(*)$ and keeping (P) terms in the expansions the velocity is determined by (see also proof in section 3.7.6):

$$\boxed{V(Z) = \frac{i}{2\pi} \frac{1}{Z - Z_M} \sum_{k=0}^{P} \frac{\alpha_k}{(Z - Z_M)^k}} \tag{3.107}$$

The complex coefficients α_k express the moments of the discrete vorticity distribution in the cell and are computed by:

$$\boxed{\begin{aligned} \alpha_k &= \sum_{m=1}^{M} \Gamma_m \left(Z_m - Z_M\right)^k \\ k &= 0, \ldots, P. \end{aligned}} \tag{3.108}$$

To make the computations more efficient the coefficients of boxes that belong to coarser levels of the hierarchy are not constructed directly from the particles. Instead they are obtained by a shifting

of the expansions of their descendants. To obtain then the expansions of a parent box from those of its children we use identity (∗∗) to get (see also proof in section 3.7.6):

$$\boxed{\alpha_l^{\text{parent}} = \sum_{k=0}^{l} \binom{l}{k} \alpha_k^{\text{children}} \left(Z_M^{\text{children}} - Z_M^{\text{parent}} \right)^{l-k}} \tag{3.109}$$

We may observe now from (3.107), that at a distance R from the center of the cluster the rate of convergence of the expansions would be proportional to $(R_M/R)^{P+1}$ and, for example, if $R \geq 2R_M$ this rate is proportional to $(1/2)^{P+1}$, implying the geometric convergence of the series. The above expansions are the main tools for the *PB* algorithm. However one may proceed one step further and recast the Laurent series into Taylor series and consider the interactions between groups of particles. These interactions take place in the form of shifting the center of expansions of one cluster (M) to the center of another (G). Those expansions are then used to compute the velocity of the particles in each box. This results in eliminating the $\log \mathrm{N}$ factor from the work count of the scheme.

To obtain these expansions we add and subtract Z_G in the denominator of the expression 3.106 so we have that the velocity induced by the group M at a point Z in the neighborhood of Z_G is equal to:

$$V(Z) = \frac{i}{2\pi} \sum_{k=0}^{P} \frac{\alpha_k}{(Z_G - Z_M)^{k+1} (1+\xi)^{k+1}},$$

where $\xi = (Z - Z_G)/(Z_G - Z_M)$. Expanding this expression for ξ using equation (∗), the velocity field induced by particles in the cluster M, at a point Z in the neighborhood of Z_G is (see also proof in section 3.7.6):

$$\boxed{V(Z) = \frac{i}{2\pi} \sum_{l=1}^{P} \delta_l (Z - Z_G)^{l-1}} \tag{3.110}$$

where the coefficients δ_l are determined by :

$$\boxed{\begin{aligned} &\delta_l = \frac{(-1)^{l+1}}{(Z_G - Z_M)^l} \sum_{k=0}^{P} \binom{l+k-1}{k} \frac{\alpha_k}{(Z_G - Z_M)^k)} \\ &l = 1, \ldots, P \end{aligned}} \tag{3.111}$$

Interactions are computed at the coarsest possible level while satisfying the accuracy criterion. However the velocities of the individual particles are computed from the expansions of the finest boxes in the hierarchy. Hence the expansions of the coarser level boxes have to be transferred down to their descendants as follows :

$$\boxed{\delta_l^{\text{children}} = \sum_{k=l}^{P} \binom{k-1}{k-l} \delta_k^{\text{parent}} \left(Z_M^{\text{children}} - Z_M^{\text{parent}} \right)^{k-l}} \tag{3.112}$$

This shifting uses (∗∗) and, as in equation (3.108), does not introduce any additional errors. One may observe here the different use of the coefficients α_k and δ_l when computing velocities on the particles. The coefficients δ_l of a certain box are used to compute the velocity on the particles of

the same (childless) box whereas the α_k's are used to compute the velocities due to particles that belong to well separated boxes.
Proofs for the above results may be found in a more rigorous and complete form in Greengard and Rohklin (1987). They show that the series converges if the distance between interacting boxes and/or particles is at least twice as large as the radius of the cluster involved. By using the expansions when clusters are separated by larger distances one may reduce the number of expansions that are necessary to obtain the same level of accuracy. This trade-off may be optimized by linking the number of expansions employed to the distance between the interacting pairs dynamically (Salmon, 1991). However this may result in increased cost for the construction of the interaction lists and it would complicate the algorithm and reduce its vectorizability.

3.7.2 The Data Structure

The two-dimensional space is considered to be a square enclosing all computational elements. We apply the operation of continuously subdividing a square into four identical squares until each square has only a certain maximum number of particles in it (see Fig. 1 for an example) or the maximum allowable level of subdivisions has been reached (the latter requirement seems obligatory when one programs in FORTRAN and has to predefine array dimensions). This procedure for a roughly uniform particle distribution results in $\mathcal{O}(\log_4 \mathrm{N})$ levels of squares.
The two fast algorithms under discussion, *PB* and *BB*, exploit the topology of the computational domain each with a different degree of complexity and efficiency. The hierarchy of boxes defines a tree data structure which is common for both algorithms. However its key ingredients and address arrays are implemented in a different way as explained below for the two algorithms. The tree construction proceeds level by level starting at the finest level of the particles and proceeding upwards to coarser box levels.
Due to the simplicity of the geometry of the computational domain the addressing of the elements of the data structure is facilitated significantly. As the construction proceeds pointers are assigned to the boxes so that there is direct addressing of the first and last particle index in them as well as there is direct access to their children and parents. This facilitates the computation of the expansion coefficients of the children from the expansions of the parents for the *BB* algorithm and the expansions of the parents from those of the children for the *PB* algorithm.

Parameters of the Data Structure The data structure is used to determine when the expansions are to be used and when pairwise interactions have to be calculated. Usually this data structure is referred to as the '(family) tree ' of the particles. It helps in communicating to the computer the geometric distribution of the particles in the computational domain. The particles reside at the finest level of the structure. Clusters of particles form the interior nodes of the tree and hierarchical relations are established. The data structure adds to the otherwise minimal memory requirements of the vortex method. This extra memory however is the tradeoff for the speed of the fast algorithms. One may add several features to this data structure trying to relate to the computer architecture as much information as possible for the particle configuration. However these memory requirements have to maintain a certain degree of uniformity for all the levels of the tree. So if one wants to use large numbers of particles (hence several levels of subdivisions) these memory requirements should be kept to a reasonable level.

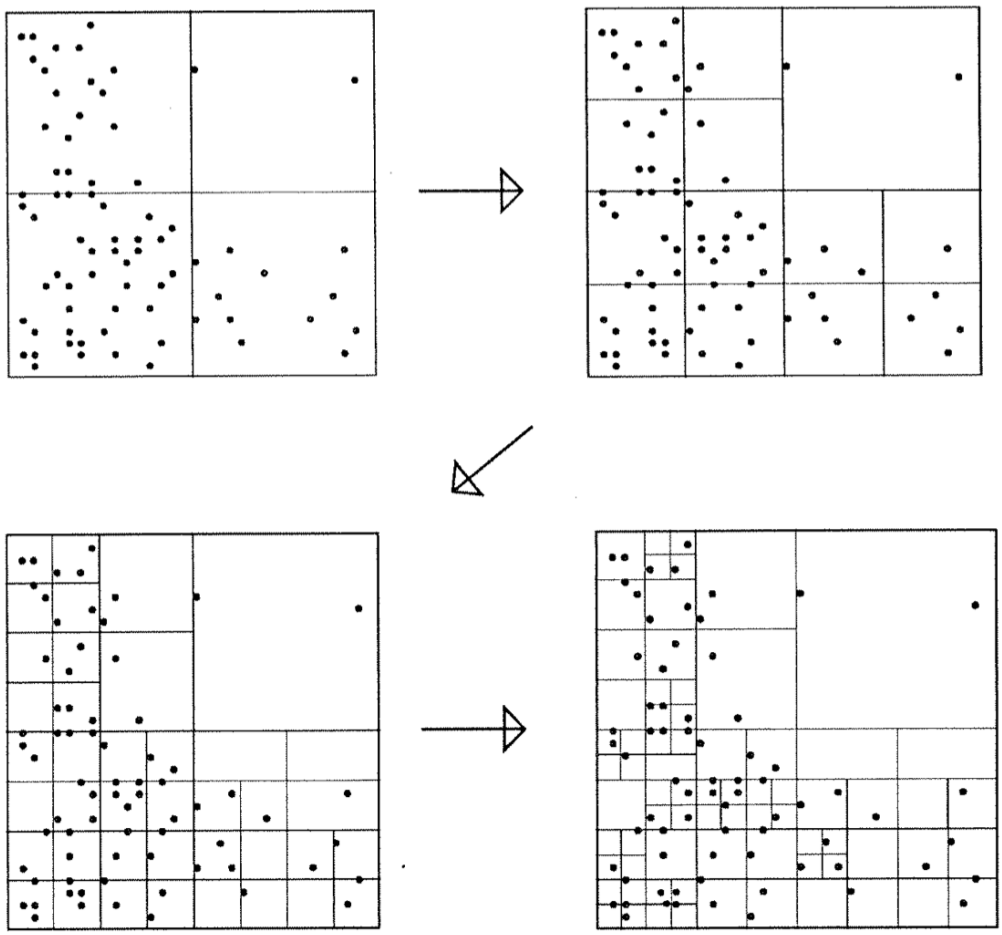


Figure 1: An example showing four stages of subdividing the computational domain. In the final stage each box contains less than three particles.

The formation and descent of the tree add to the cost of the algorithms and a non-refined implementation may result in the degradation of the whole algorithm. A data structure may be Lagrangian or Eulerian. The former adapts automatically to the locations of the particles and can be the same for several steps. The latter has to be reconstructed at every step as the particles change positions in the domain. There are several ways that nearby particles could be clustered together and some of the decisions to be made are :

The center of expansions. This can be either the geometric center of a certain region in space or the center of mass of the distribution of particles or some other location chosen so that the data structure is conveniently addressed and the expansions converge rapidly. Choosing the center of 'mass' implies that for a uniform particle distribution and a certain number of particles per cluster the minimum radius for convergence is expected. On the other hand the geometric center (as in the present work) facilitates tremendously the addressing of the boxes in the data structure.

The cluster size. Accuracy requirements impose that the cluster size should be as small as possible while efficiency considerations dictate that clusters should contain enough particles so that the use of the expansions is beneficial. There are different approaches to that as others require the finest

clusters to contain fewer than L_{min} (e.g. for the Barnes-Hut algorithm $L_{min} = 2$) particles where others impose the finest level of subdivision beyond which no further subdivisions take place. The former procedure seems to offer more accurate expansions whereas the latter economizes in memory and compactness of the data structure especially when more than one particle resides at the finer boxes and large numbers of terms are to be kept in the expansions. In the present algorithms we follow a hybrid strategy as we keep at least $\mathrm{L_{min}}$ particles per box until we reach a predetermined finest level of boxes. The number $\mathrm{L_{min}}$ may be chosen by the user depending on the particle population and configuration so as to achieve an optimal computational cost.
A related issue is how one subdivides a cluster or box with more than L_{min} particles. Practical implementations of the method resort basically to two techniques. One may split the parent cluster into two children resulting in a so called binary tree. The direction of this dissection depends on the distribution of the particles, attempting to optimally adapt the data structure to the locations of the particles in the computational domain. In practice this has the benefit of requiring fewer terms in the expansions to obtain a certain accuracy (for a relatively uniform particle distribution). Alternatively, as in the present scheme, one may simply divide the box into four boxes. This would require a larger number of expansions to be calculated and stored to obtain the required accuracy, as the radius for convergence is not minimized. However these 'extra' terms have a minimal effect to the overall cost of the scheme as they appear in fully vectorized parts of the algorithm. Moreover the regularity of such a procedure facilitates the logic of the algorithm and the construction and addressing of the data structure.
Addressing the clusters. A key factor in the computer implementation of the method, addressing should be such that it does not inhibit the vectorization or the parallel implementation of the method. Traversing and building the hierarchical tree is highly dependent on this procedure. A simpler technique would be to store information (such as geometric location, size, family ties etc.) for the tree nodes in the memory. Such a procedure may severely limit the number of particles that can be computed [Pépin and Leonard, 1990] if excess information is stored. Besides it would degrade the performance for machines (such as the CRAY-2) where memory access is computationally intensive. However both restrictions do not seem to consist major drawbacks in the present implementation on the CRAY YMP, provided a reasonable number of subdivisions (less than 10) is allowed.
Another key issue is the addressing of the particles. As particles are usually associated with a certain box it is efficient to sort the particle locations in the memory so that *particles that belong to the same box occupy adjacent locations in the memory devoted to the particle arrays*. Such memory allocation enhances the vectorization tremendously as very often we loop over particles of the same box (e.g., to construct the expansions at the finest level, or to compute interactions) and the loops have an optimum stride of one.
As one can deduce, the possible combinations of the above features may result in a number of different implementations. Depending on the implementation a different degree of vectorization may then be achieved.

3.7.3 Description of the Algorithms

In both algorithms, described herein, we may distinguish three stages:

- Building the data structure (tree)
- Establishing the interaction lists (by non-recursively descending the tree)

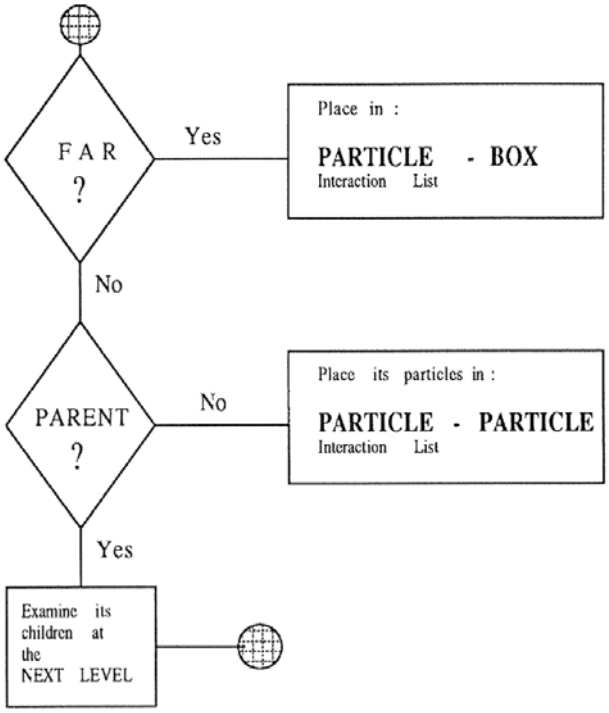


Figure 2: Flow Chart for the tree traversal of the *PB* algorithm for *all particles of a childless box*

- Velocity evaluations for all particles in the domain.

The building of the data structure is common for both algorithms but they differ in the tree descent and the velocity evaluation. Basic requirements for an efficiently vectorized code are the simplicity of the algorithm, the existence of long vectorizable loops (i.e., simple loops, odd stride e.t.c) and the reduction of memory referencing. Care has been exercised at all stages so that the maximum degree of vectorization is achieved and efficient calculations result. In the present implementation the building of the data structure consumes about $5-7\%$ of the time whereas the descent consumes another $5-10\%$ so that the largest amount is spent in computing the velocities.

The Particle-Box Algorithm The hierarchical structure of the algorithm has a logical complexity that implies a recursive procedure. The algorithm may be easily described by the following code:

```
subroutine interact (n,C)
IF (particle n is well separated from cell C) THEN
   velocity = sum of expansions of cell C on n
ELSE
   CALL interact (n,children of C)
END IF
```

Note the recursiveness of this subroutine in the ELSE block. A straightforward approach to the

non-recursive programming of this subroutine, in an attempt to vectorize it, would be to unroll it. Such a procedure would introduce a depth-first search of the tree. However this is not very efficient because for most applications the depth of the tree is not long enough to enable optimized vector operations. As reported by Makino (1990), Barnes introduced interaction lists associated with each particle and then vectorized by looping over the particles. Every particle had its own interaction list but increased memory referencing and the additional complexity of the algorithm resulted in degradation of the performance. In our algorithm we employ the following alternate procedure:

Step 1 : Building the data structure (tree)

Step 1a : For each of the squares at each level that are not further subdivided we compute the p-terms of the multipole expansions. These expansions are used to describe the influence of the particles at locations that are well separated from their cluster. The *cost* of this step is $\mathcal{O}(Np)$.

Step 1b : The expansions of all parent boxes are constructed by shifting the expansion coefficients of their children. The tree is traversed upwards in this stage. Rather than constructing the expansions of all the members of a family (that is traverse each branch until the root is reached) we *construct the expansions of all parent boxes at each level simultaneously*. This enables long loops over the parent boxes at each level. Care is taken so that the procedure is fully vectorized by taking advantage of the regularity of the data structure and the addressing of the boxes in the memory. Moreover the regularity of the data structure allows us to precompute many coefficients that are necessary for the expansions. The cost of this step is $\mathcal{O}(Np^2)$, as each shifting requires p^2 operations and there are at most $(4N-1)/3$ boxes in the computational domain.

Step 2 : Establishing of interaction lists In the present algorithm a **breadth-first** search is performed at each level to establish the interaction lists of each particle (cell). This search is facilitated by the regularity of the data structure and the identification arrays of the cells in the tree. At each level interaction lists are established for the particles (cells) by looping across the cells of a certain level.

Algorithmically this operation is represented as follows:

```
subroutine setlist (n)
For l=1,levelmax
   CALL FARCLOSE(Z(n), ds(l), IZ(l), kxm, Lstxm, kfr, Lstfr, kcls, Lstcls)
   CALL CLOSECHECK(kcls, Lstcls, kpp, Lstpp, kxm, Lstxm)
   CALL GATHER(kfr, Lstfr, kpp, Lstpp, ZP, GP, ZB, EB)
END For
return
```

For each particle **n** the hierarchy of cells is traversed breadth first. The check for faraway or close boxes is performed within the subroutine FARCLOSE that is fully vectorized. Initially the addresses of `kxm` cells that belong to the coarsest level are stored in array `Lstxm`. By checking the locations of the boxes at the grid of level `l` and the respective location of the particle on that grid the cells are identified as either faraway (stored in `Lstfr`) or nearby (stored in `Lstcls`). Boxes that are far interact with the particle and are placed in a P-B interaction list `Lstpp` along with their expansions `EB` and locations `ZB` in subroutine GATHER. Boxes that are close however are further examined in the fully vectorized subroutine CLOSECHECK. Those that are childless have their particles stored in an array `Lstpp` so that they interact directly. Those that are parents have their children stored in array `Lstxm` so that they are fed back in FARCLOSE at the finer level.

Note that at the finest level all boxes are considered childless and subroutine CLOSECHECK need not be called. This way the interaction lists for the particles are formed successively and a fully vectorized force calculation calculates the velocities of the particles at the following stage.
Note now that this depth first search for interaction lists is further facilitated by the following observation. Every particle belongs to a childless box. It is easy then to observe that *all particles in the same box share the same interaction list comprised of members of the tree that belong to coarser levels*. This way the tree is traversed upwards for all particles in a childless box together and downwards separately for each particle. It is evident that this procedure is more efficient for uniformly clustered configurations of particles as there would be more particles that belong to childless boxes at the finest level.
The *cost* of this step scales as $\mathcal{O}((N/L_{\rm min})\log N)$ as there are $\log N$ levels of boxes and the tree is traversed $(N/L_{\rm min})$ times.
Step 3 : Computation of the interactions.
Once the interaction lists have been established the velocities of the particles are computed by looping over the elements of the lists. For particles that have the same boxes in their interaction list this is performed simultaneously so that memory referencing is minimized. Moreover by systematically traversing the tree the particle-particle interactions are made symmetric so that the cost of this computation is halved. Care has been exercised to compute the velocities with the minimum possible number of operations (Section 3.7.4).
The *cost* of this step is $\mathcal{O}(Np)$.

The Box-Box Algorithm. This scheme is similar to the *PB* scheme except that here every node of the tree assumes the role of a particle. In other words interactions are not limited to particle-particle and particle-box but interactions between boxes are considered as well. Those interactions are in the form of shifting the expansion coefficients of one box into another and the interaction lists are established with respect to the locations of every node of the tree.
The scheme distinguishes five categories of interacting elements of the tree with respect to a cell denoted by **c**.

- **List 1 :** All childless boxes neighboring **c**.
- **List 2 :** Children of colleagues of b's parents that are well separated from **c**. All such boxes belong to the same level with **c**.
- **List 3 :** Descendants (not only children) of **c's** colleagues (boxes of the same size as **c**), whose parents are adjacent to **c** but are not adjacent to **c** themselves. All such boxes belong to finer levels.
- **List 4 :** All boxes such that box **c** belongs to *their* List 3. All such boxes are childless and belong to coarser levels.
- **List 5 :** All boxes well separated from **c's** parents. Boxes in this category do not interact directly with the cell **c**.

If the cell **c** is childless it may have interacting pairs that belong to all four lists. However if it is a parent it is associated with boxes that belong to lists 2 and 4 as described above. These observations are directly applied in our algorithm and we may distinguish again the following 4 steps.

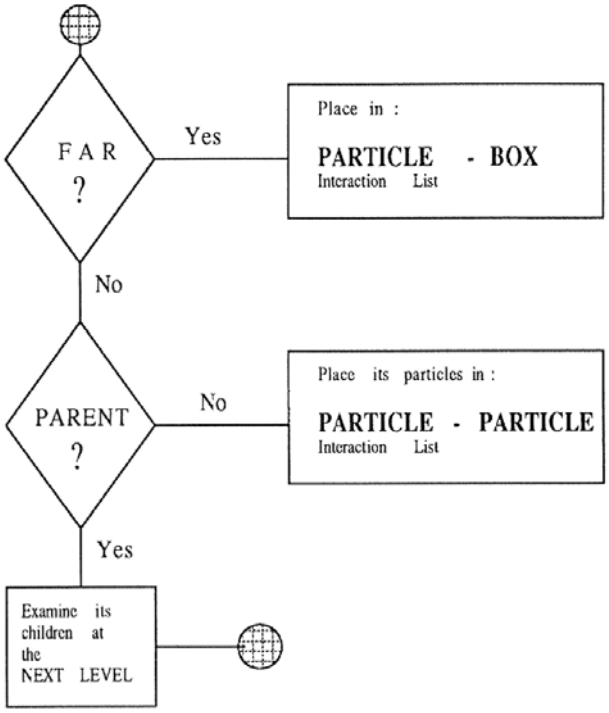


Figure 3: Flow Chart for the Tree Traversal of the *BB* scheme for a *cell* **c**

Step 1 : Building the data structure.
This procedure is the same as for the *PB* scheme. This fact enables us actually to compare directly the two algorithms and asses their efficiency.
Step 2 : Construction of Interaction Lists.
To establish the interaction lists we proceed again level by level starting at the coarsest level. For each level we distinguish childless and parent boxes. In establishing lists 1 and 3 we need only loop over childless boxes whereas to establish lists 2 and 4 we loop over all cells that are active in a certain level.
Step 2a : Here we establish lists 1 and 3. We start at the level of the parents of box **c** and we proceed level by level examining again breadth first, until we reach the finest level of the structure (the particles). The elements of lists 1 are basically the particles and account for the particle-particle interactions. Care is exercised so that this computation is symmetric and we need to traverse the tree downwards only. The elements of List 3 are the boxes and are accountable for the particle-box interactions in this scheme.
More specifically an algorithmic description of our algorithm is given by :
The outer loop here is over all childless boxes. The colleagues of their parents are identified (either by being retrieved directly from an array or by using FARCLOSE) and subsequently their children are placed in `Lstxm`. Subsequently a call to FARCLOSE distinguishes between faraway and nearby boxes at different levels. Boxes placed in `Lst1` are to interact directly whereas those placed in `Lst3` are to interact as particle-box.
Step 2b : Here we establish interaction lists 2 and 4 for all boxes in the hierarchy. We start at

```
subroutine Lists1&3 (lpr,kchlds)
For k=1,kchlds
  Find the colleagues of k's parents (Lclg)
  Place the children of (Lclg) boxes in (Lstxm)
  For l=lpr,lmax
    CALL FARCLOSE(ZC, ds(l), Kxm, Lstxm, k3, Lst3, kcls, Lstcls)
    CALL CLOSECHECK(kcls, Lstcls, k1, Lst1, kxm, Lstxm)
  END For
END For
return
```

the coarsest possible level and proceed downward until reaching the level of box **c** to establish the interaction lists. To do so for a certain box we start by examining boxes that are not well separated from its parents (otherwise they would have been dealt with at the coarser level). Subsequently the children of those boxes are examined to establish interaction lists. Algorithmically this operation is represented by:

```
subroutine Lists2&4 (lpr,kbox)
For k=1,kbox
  For l=l0,lpr
    Boxes close to parents (?)(use FARCLOSE)
    Close to parents - Childless(?)(use CLOSECHECK)
    Close to parents and Childless - Close to box (?)(Use FARCLOSE)
  END For
END For
return
```

Step 3 : Computations of the interactions
In this scheme we consider three kinds of interactions: the box-box, particle-box and particle-particle interactions. The latter two categories were discussed in the previous section. For the box-box interactions once the respective interaction lists have been established (with members of lists 2 and 4) we need to transfer those expansions down to the ones of the children and add them to the existing ones. This procedure is vectorized by looping over the number of boxes at each level. The use of pointers to access the children of each box enhances this vectorization. Note that an arbitrarily high number of expansions can be calculated efficiently by unrolling the loop over the number of expansions into the previously mentioned loop.

3.7.4 Practical Formulas for Velocity Calculations

Once the interaction lists have been established the velocities on the particle locations need to be calculated using formulas (3.106)–(3.112). Because this is the most time consuming part of the algorithm an effort is made to reduce as much as possible the computational cost. A possible increase in the computational speed may be obtained by some ASSEMBLY programming for the velocity evaluations. Following are some of the formulas employed to reduce this computational cost.

- *Particle-Particle Interactions :* In order to achieve higher convergence rates in the vortex methods more complicated functions than the delta functions are usually necessary for the description of the vorticity field. This will reduce the efficiency of the algorithm but to minimize the extra cost one could make use of look up tables. In these look up tables in order to get the accuracy required we should not only store the values of the function but its derivatives as well. Using the MATH77 scientific libraries of the CRAY would help reduce this computational cost.

- *Particle-Box Interactions :* Formula (3.107) may be calculated using recursive relations. By computing separately real and imaginary parts for the velocities we have that for a particle located at $z = x + iy$ the contribution from a box located at $X_M = X_M + iY_M$ having expansions $\alpha_k = \lambda_k + i\mu_k$ may be computed with the following algorithm.

$$r_0 = \frac{x - X_M}{(x - X_M)^2 + (y - Y_M)^2} \tag{3.113}$$

$$f_0 = \frac{y - Y_M}{(x - X_M)^2 + (y - Y_M)^2} \tag{3.114}$$

$$r_k = r_{k-1} \cdot r_0 - f_{k-1} \cdot f_0 \tag{3.115}$$

$$f_k = r_{k-1} \cdot f_0 + f_{k-1} \cdot r_0 \tag{3.116}$$

and the velocities (u,v) would be given as :

$$u = -\sum_{k=0}^{k=P} \lambda_k \cdot r_k - \mu_k \cdot f_k \tag{3.117}$$

$$v = \sum_{k=0}^{k=P} \lambda_k \cdot f_k + \mu_k \cdot r_k \tag{3.118}$$

- *Parent Cell Expansions :* The expansions of parent cells are computed from the expansions of their children. This procedure does not introduce any errors and it helps in economizing computer time compared to a direct calculation using the particle locations. The amount of necessary calculations is reduced by exploring the regularity of the location of the children boxes with respect to the parents. Thus in formula (3.109) we have

$$Z = Z_M^{\text{children}} - Z_M^{\text{parent}} = \frac{d}{2} \begin{cases} -1 + i & \text{for Box 1;} \\ -1 - i & \text{for Box 2;} \\ 1 - i & \text{for Box 3;} \\ 1 + i & \text{for Box 4.} \end{cases} \tag{3.119}$$

where d is the size of the child cell. Hence we may calculate explicitly the powers $\binom{l}{k} Z^{l-k}$ for all values of l,k thus speeding up the computations inside the loop for the boxes.

- *Cell-Cell Interactions:* A procedure similar to the one for the particle-cell interactions is followed here. If $\delta_l = \eta_l + i\theta_l$ are the coefficients of a box located at $Z_G = X_G + iY_G$

which are contributed by a box located at $Z_M = X_M + iY_M$ then these coefficients are computed with the following scheme :

$$r_0 = \frac{X_G - X_M}{(X_G - X_M)^2 + (Y_G - Y_M)^2} \tag{3.120}$$

$$f_0 = \frac{Y_G - Y_M}{(X_G - X_M)^2 + (Y_G - Y_M)^2} \tag{3.121}$$

$$r_k = r_{k-1} \cdot r_0 - f_{k-1} \cdot f_0 \tag{3.122}$$

$$f_k = r_{k-1} \cdot f_0 + f_{k-1} \cdot r_0 \tag{3.123}$$

and finally:

$$R_l = -\sum_{k=0}^{k=P} \lambda_k \cdot r_k - \mu_k \cdot f_k \tag{3.124}$$

$$F_l = \sum_{k=0}^{k=P} \lambda_k \cdot f_k + \mu_k \cdot r_k \tag{3.125}$$

$$\eta_l = r_l \cdot R_l - f_l \cdot F_l \tag{3.126}$$

$$f_k = r_l \cdot F_l + f_l \cdot R_l \tag{3.127}$$

- *Children Expansions :* Cell-Cell interactions are performed at the coarsest possible level. Then the expansions of the parent boxes are transferred down to the expansions of their children and are added to the existing ones until the finest level at any branch of the tree is reached. Those expansion coefficients are computed using formula (3.112) in a similar way as for the expansions of parent cells discussed above.

- *Cell - Particle Interactions :* Once the expansions of all childless boxes have been obtained they are used to compute the velocities on the particles. Formula (3.110) is used and a procedure similar to the one discussed above is followed. Vectorization is achieved by looping over all particles in a certain childless box. This may be inefficient if only a small number of particles is contained in this box.

3.7.5 Performance of the Algorithms

Comparisons were made for the two fast algorithms and the $\mathcal{O}(N^2)$ in terms of efficiency and computational speed. We require that the fast algorithms produce the same results as the N^2 scheme with minimum accuracy of six significant figures in the velocity field of a random uniform distribution of particles in a square. We allow up to eight levels in the hierarchy of theboxes and use ten terms in the multipole expansions at all levels.
In figure 4 we show the efficiency in the vectorization achieved for the fast algorithms on the CRAY/XMP-18. The *PB* algorithm is much more efficient than its *BB* competitor in regards with vectorization. This is a result that was expected because of the simplicity of the PP scheme versus the algorithmic complexity of the BB scheme.
This is not the whole story, however, since in applications we are mainly interested in the overall speed of the algorithm and not necessarily in its Mflps efficiency. Figure 5 shows the CPU time

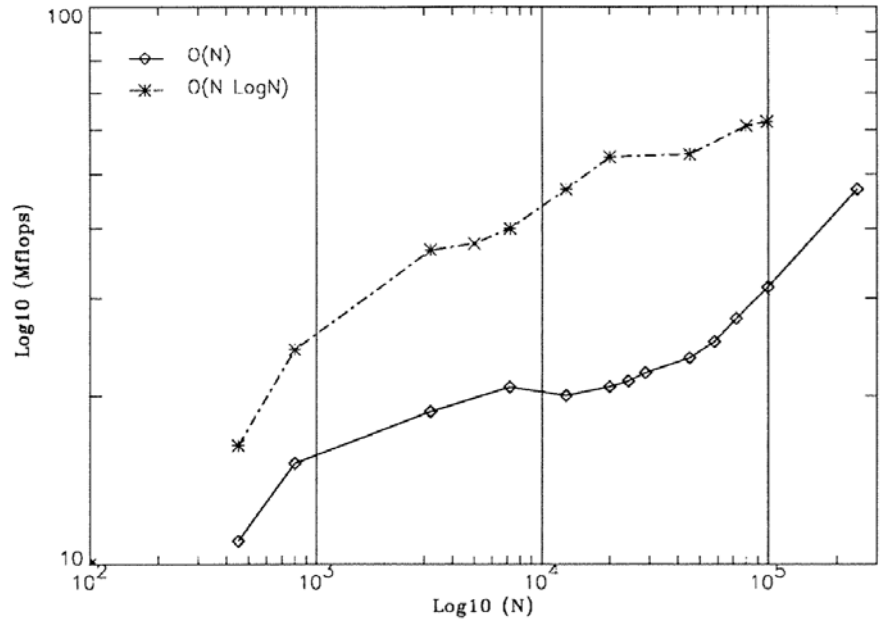


Figure 4: Performance in Mflops of the *PB* and *BB* algorithms on the CRAY/XMP-18

required on a single processor of an XMP-18 in each of the three methods for an evaluation of all N velocities for N elements with a minimum accuracy of 10^{-6}. Note that fast methods become advantageous for computations involving only $N \approx 500$ particles with the *BB* method breaking even with the *PB* method for $N \approx 4400$. Note that the CPU time required on a YMP processor is about 2/3 of the time required on an XMP. A timestep requiring one evaluation of all velocities for a million particles requires about one minute on single processor of a CRAY YMP while the N^2 algorithm would require roughly 24 hours.

3.7.6 Derivation of some relevant equations

Particle to Multipole We know

$$V(Z) = \frac{i}{2\pi} \sum_{n=1}^{N} \frac{\Gamma_n}{Z - Z_n}.$$

We want to show that we can write this also as

$$V(Z) = \frac{i}{2\pi} \frac{1}{Z - Z_M} \sum_{k=0}^{\infty} \frac{\alpha_k}{(Z - Z_M)^k},$$

where

$$\alpha_k = \sum_{n=1}^{N} \Gamma_n (Z_n - Z_M)^k,$$

as long as $\left(\frac{\max_n |Z_n - Z_M|}{|Z - Z_M|} \right) < 1.$

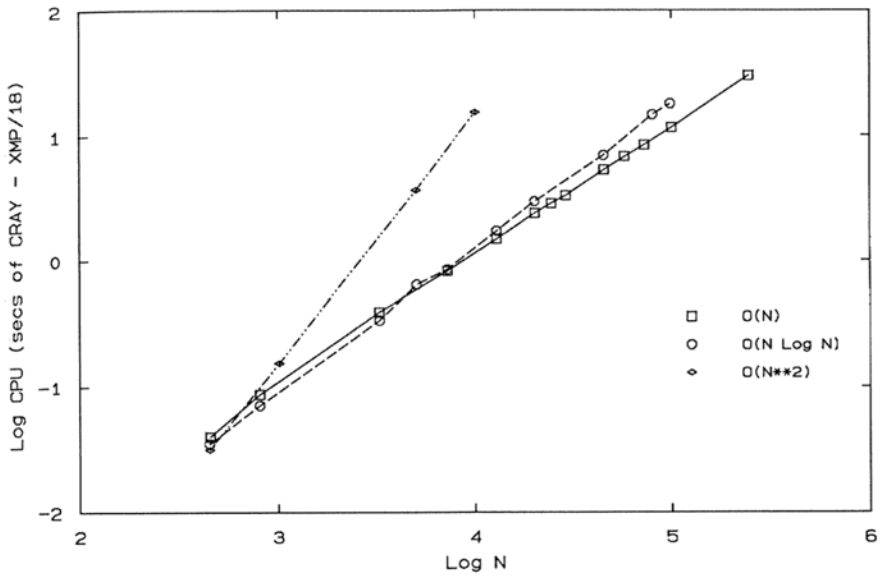


Figure 5: Computational Cost of the *PB*, *BB* and direct summation algorithms

Proof We can rewrite the first equation

$$\frac{i}{2\pi}\sum_{n=1}^{N}\frac{\Gamma_n}{Z-Z_n} = \frac{i}{2\pi}\sum_{n=1}^{N}\frac{\Gamma_n}{Z-Z_M+Z_M-Z_n} \tag{3.128}$$

$$= \frac{i}{2\pi}\frac{1}{Z-Z_M}\sum_{n=1}^{N}\frac{\Gamma_n}{\left(1-\dfrac{Z_n-Z_M}{Z-Z_M}\right)}. \tag{3.129}$$

Now we use the expansion, valid for $|x|<1$,

$$\frac{1}{1-x} = \sum_{k=0}^{\infty} x^k,$$

to get

$$\frac{i}{2\pi}\frac{1}{Z-Z_M}\sum_{n=1}^{N}\frac{\Gamma_n}{\left(1-\dfrac{Z_n-Z_M}{Z-Z_M}\right)} = \frac{i}{2\pi}\frac{1}{Z-Z_M}\sum_{n=1}^{N}\Gamma_n\sum_{k=0}^{\infty}\left(\frac{Z_n-Z_M}{Z-Z_M}\right)^k \tag{3.130}$$

$$= \frac{i}{2\pi}\frac{1}{Z-Z_M}\sum_{k=0}^{\infty}\frac{\sum_{n=1}^{N}\Gamma_n(Z_n-Z_M)^k}{(Z-Z_M)^k} \tag{3.131}$$

$$= \frac{i}{2\pi}\frac{1}{Z-Z_M}\sum_{k=0}^{\infty}\frac{\alpha_k}{(Z-Z_M)^k}, \tag{3.132}$$

where

$$\alpha_k = \sum_{n=1}^{N} \Gamma_n (Z_n - Z_M)^k.$$

Due to the constraints on using the expansion, the above holds only for $\left(\frac{\max_n |Z_n - Z_M|}{|Z - Z_M|} \right) < 1$.

Multipole translations - Child to Parent We have

$$V(Z) = \frac{i}{2\pi} \frac{1}{Z - Z_M} \sum_{k=0}^{\infty} \frac{\alpha_k}{(Z - Z_M)^k},$$

where

$$\alpha_k = \sum_{n=1}^{N} \Gamma_n (Z_n - Z_M)^k$$

We want to show that we can translate this expression to a different center of expansion Z_M^P, and get

$$\frac{i}{2\pi} \frac{1}{(Z - Z_M)} \sum_{k=0}^{\infty} \frac{\alpha_k}{(Z - Z_M)^k} = \frac{i}{2\pi} \frac{1}{(Z - Z_M^P)} \sum_{l=0}^{\infty} \frac{\alpha_l^P}{(Z - Z_M^P)^l}, \tag{3.133}$$

where

$$\alpha_l^P = \sum_{k=0}^{l} \alpha_k \binom{l}{k} (Z_M - Z_M^P)^{l-k} \tag{3.134}$$

Proof We leave away the prefactor $\frac{i}{2\pi}$ for the proof. First we write

$$\frac{1}{(Z - Z_M)} \sum_{k=0}^{\infty} \frac{\alpha_k}{(Z - Z_M)^k} = \sum_{k=0}^{\infty} \frac{\alpha_k}{(Z - Z_M)^{k+1}} \tag{3.135}$$

$$= \sum_{k=0}^{\infty} \frac{\alpha_k}{(Z - Z_M + Z_M^P - Z_M^P)^{k+1}} \tag{3.136}$$

$$= \sum_{k=0}^{\infty} \frac{\alpha_k}{(\tilde{Z} - Z_0)^{k+1}}, \tag{3.137}$$

where $\tilde{Z} = (Z - Z_M^P)$ and $Z_0 = (Z_M - Z_M^P)$.
Now we use the identity (see 'shortcourse FMM', page 21, bottom - the binomial theorem)

$$(Z - Z_0)^{-k} = \sum_{l=k}^{\infty} \binom{l-1}{k-1} \frac{Z_0^{l-k}}{Z^l},$$

and find

$$\sum_{k=0}^{\infty} \frac{\alpha_k}{(\tilde{Z}-Z_0)^{k+1}} = \sum_{k=0}^{\infty} \alpha_k \sum_{l'=k+1}^{\infty} \binom{l'-1}{k} \frac{Z_0^{l'-k-1}}{\tilde{Z}^{l'}} \tag{3.138}$$

$$= \sum_{k=0}^{\infty} \alpha_k \sum_{l=k}^{\infty} \binom{l}{k} \frac{Z_0^{l-k}}{\tilde{Z}^{l+1}} \tag{3.139}$$

$$= \frac{1}{\tilde{Z}} \sum_{k=0}^{\infty} \alpha_k \sum_{l=k}^{\infty} \binom{l}{k} \frac{Z_0^{l-k}}{\tilde{Z}^{l}} \tag{3.140}$$

$$\tag{3.141}$$

where we replaced the dummy index l' with $l = l' - 1$.
Replacing the original definitions of $\tilde{Z}$ and Z_0 gives then

$$\frac{1}{\tilde{Z}} \sum_{k=0}^{\infty} \alpha_k \sum_{l=k}^{\infty} \binom{l}{k} \frac{Z_0^{l-k}}{\tilde{Z}^{l}} = \frac{1}{(Z-Z_M^P)} \sum_{k=0}^{\infty} \sum_{l=k}^{\infty} \alpha_k \binom{l}{k} \frac{(Z_M-Z_M^P)^{l-k}}{(Z-Z_M^P)^{l}} \tag{3.142}$$

Using the fact that the binomial coefficient $\binom{n}{k}$ is zero whenever $k > n$, we can first extend the bottom limit of the second summation to start from $l = 0$ instead of $l = k$. Then we can swap the summations of k and l (since we are working under conditions such that the series is convergent), and use again the property of the binomial coefficient to truncate the (inner) sum of k at l:

$$\frac{1}{(Z-Z_M)} \sum_{k=0}^{\infty} \frac{\alpha_k}{(Z-Z_M)^k} = \frac{1}{(Z-Z_M^P)} \sum_{k=0}^{\infty} \sum_{l=k}^{\infty} \alpha_k \binom{l}{k} \frac{(Z_M-Z_M^P)^{l-k}}{(Z-Z_M^P)^{l}} \tag{3.143}$$

$$= \frac{1}{(Z-Z_M^P)} \sum_{k=0}^{\infty} \sum_{l=0}^{\infty} \alpha_k \binom{l}{k} \frac{(Z_M-Z_M^P)^{l-k}}{(Z-Z_M^P)^{l}} \tag{3.144}$$

$$= \frac{1}{(Z-Z_M^P)} \sum_{l=0}^{\infty} \sum_{k=0}^{\infty} \alpha_k \binom{l}{k} \frac{(Z_M-Z_M^P)^{l-k}}{(Z-Z_M^P)^{l}} \tag{3.145}$$

$$= \frac{1}{(Z-Z_M^P)} \sum_{l=0}^{\infty} \sum_{k=0}^{l} \alpha_k \binom{l}{k} \frac{(Z_M-Z_M^P)^{l-k}}{(Z-Z_M^P)^{l}} \tag{3.146}$$

Now we can identify the parent coefficients by comparing terms, to find that

$$\frac{1}{(Z-Z_M)} \sum_{k=0}^{\infty} \frac{\alpha_k}{(Z-Z_M)^k} = \frac{1}{(Z-Z_M^P)} \sum_{l=0}^{\infty} \frac{\alpha_l^P}{(Z-Z_M^P)^l}, \tag{3.147}$$

where

$$\alpha_l^P = \sum_{k=0}^{l} \alpha_k \binom{l}{k} (Z_M-Z_M^P)^{l-k} \tag{3.148}$$

Multipole shifting We have

$$V(Z) = \frac{i}{2\pi}\frac{1}{Z - Z_M}\sum_{k=0}^{\infty}\frac{\alpha_k}{(Z - Z_M)^k},$$

where

$$\alpha_k = \sum_{n=1}^{N}\Gamma_n(Z_n - Z_M)^k$$

We want to show that, once we have computed the influence of sources with center Z_M at a target location Z_G, we can compute the velocities at a point Z close to Z_G as follows:

$$V(Z) = \frac{i}{2\pi}\frac{1}{(Z - Z_M)}\sum_{k=0}^{\infty}\frac{\alpha_k}{(Z - Z_M)^k} = \frac{i}{2\pi}\sum_{l=1}^{\infty}\delta_l(Z - Z_G)^{l-1}, \tag{3.149}$$

where

$$\delta_l = \frac{(-1)^{l+1}}{(Z_G - Z_M)^l}\sum_{k=0}^{P}\binom{l+k-1}{k}\frac{\alpha_k}{(Z_G - Z_M)^k}, \tag{3.150}$$

as long as $\left|\dfrac{Z - Z_G}{Z_G - Z_M}\right| < 1$.

Proof We leave away the prefactor $\frac{i}{2\pi}$ for the proof. First we write

$$\sum_{k=0}^{\infty}\frac{\alpha_k}{(Z - Z_M)^{k+1}} = \sum_{k=0}^{\infty}\frac{\alpha_k}{(Z - Z_G + Z_G - Z_M)^{k+1}} \tag{3.151}$$

$$= \sum_{k=0}^{\infty}\frac{\alpha_k}{(Z_G - Z_M)^{k+1}\left(\dfrac{Z - Z_G}{Z_G - Z_M} + 1\right)^{k+1}} \tag{3.152}$$

$$= \sum_{k=0}^{\infty}\frac{\alpha_k}{(Z_G - Z_M)^{k+1}(1+\xi)^{k+1}}, \tag{3.153}$$

where $\xi = \dfrac{Z - Z_G}{Z_G - Z_M}$. We assume that $|\xi| < 1$.
Now we use the expansion

$$\frac{1}{(1+x)^k} = \sum_{l=0}^{\infty}(-1)^l\binom{l+k-1}{l}x^l \tag{3.154}$$

valid for $|x| < 1$. We substitute this expansion on the term with ξ, and get

$$\sum_{k=0}^{\infty}\frac{\alpha_k}{(Z_G - Z_M)^{k+1}(1+\xi)^{k+1}} = \sum_{k=0}^{\infty}\frac{\alpha_k}{(Z_G - Z_M)^{k+1}}\sum_{l=0}^{\infty}(-1)^l\binom{l+k}{l}\xi^l \tag{3.155}$$

$$= \sum_{k=0}^{\infty}\sum_{l=0}^{\infty}\frac{\alpha_k}{(Z_G - Z_M)^{k+1}}(-1)^l\binom{l+k}{l}\xi^l. \tag{3.156}$$

We use the facts that for a convergent sum we can switch the order of the summations, and

$$\binom{l+k}{l} = \binom{l+k}{k},$$

to get

$$\sum_{k=0}^{\infty}\sum_{l=0}^{\infty} \frac{\alpha_k}{(Z_G - Z_M)^{k+1}}(-1)^l \binom{l+k}{l} \xi^l = \sum_{l=0}^{\infty}\sum_{k=0}^{\infty} \frac{\alpha_k}{(Z_G - Z_M)^{k+1}}(-1)^l \binom{l+k}{k} \xi^l. \quad (3.157)$$

Substituting back the definition of ξ we obtain

$$\sum_{l=0}^{\infty}\sum_{k=0}^{\infty} \frac{\alpha_k}{(Z_G - Z_M)^{k+1}}(-1)^l \binom{l+k}{k} \xi^l = \sum_{l=0}^{\infty}\sum_{k=0}^{\infty} \frac{\alpha_k}{(Z_G - Z_M)^{k+1}}(-1)^l \binom{l+k}{k} \frac{(Z - Z_G)^l}{(Z_G - Z_M)^l} \quad (3.158)$$

$$= \sum_{l=0}^{\infty} (Z - Z_G)^l \left(\frac{(-1)^l}{(Z_G - Z_M)^l} \sum_{k=0}^{\infty} \binom{l+k}{k} \frac{\alpha_k}{(Z_G - Z_M)^{k+1}} \right). \quad (3.159)$$

Truncating the sums at order P and introducing the definition of δ_l then gives

$$\sum_{l=0}^{\infty} (Z - Z_G)^l \left(\frac{(-1)^l}{(Z_G - Z_M)^l} \sum_{k=0}^{\infty} \binom{l+k}{k} \frac{\alpha_k}{(Z_G - Z_M)^{k+1}} \right) = \sum_{l=0}^{P} \delta_l (Z - Z_G)^l,$$

where

$$\delta_l = \frac{(-1)^l}{(Z_G - Z_M)^l} \sum_{k=0}^{P} \binom{l+k}{k} \frac{\alpha_k}{(Z_G - Z_M)^{k+1}}.$$

Due to the used expansion, this is valid under the condition $|\xi| < 1$, or $\left| \dfrac{Z - Z_G}{Z_G - Z_M} \right| < 1$.

Finally, to bring the answer to the form of the lecture notes, we replace the summation index l with a new index $l' = l - 1$:

$$\sum_{l=0}^{P} \delta_l (Z - Z_G)^l = \sum_{l'=1}^{P} \delta_{l'-1} (Z - Z_G)^{l'-1} = \sum_{l'=1}^{P} \tilde{\delta}_{l'} (Z - Z_G)^{l'-1},$$

where

$$\tilde{\delta}_l = \delta_{l-1} = \frac{(-1)^{l-1}}{(Z_G - Z_M)^{l-1}} \sum_{k=0}^{P} \binom{l+k-1}{k} \frac{\alpha_k}{(Z_G - Z_M)^{k+1}} \quad (3.160)$$

$$= \frac{(-1)^{l+1}}{(Z_G - Z_M)^{l}} \sum_{k=0}^{P} \binom{l+k-1}{k} \frac{\alpha_k}{(Z_G - Z_M)^{k}} \quad (3.161)$$

$$(3.162)$$

4 REMESHED PARTICLE METHODS

Particle methods are often defined as grid-free methods making them an attractive alternative to mesh based methods for flows past complex and deforming boundaries. However the adaptivity provided by the Lagrangian description can introduce errors and particle methods have to be conjoined with a grid to provide consistent, efficient and accurate simulations. The grid does not detract from the adaptive character of the method and serves as a tool to restore regularity in the particle locations via *Remeshing* while it simultaneously enables systematic *Multiresolution* particle simulations [Bergdorf and Koumoutsakos, 2006], allows *Fast velocity evaluations* [Harlow, 1964] and facilitates *Hybrid Particle-Mesh* methods capable of handling different numerical methods and different equations in various parts of the domain [Cottet, 1990].

4.1 (the need of) Remeshing for Particle Distortion

Particle methods, when applied to the Lagrangian formulation of convection-diffusion equations enjoy an automatic adaptivity of the computational elements as dictated by the flow map. This adaptation comes at the expense of the regularity of the particle distribution because particles adapt to the gradients of the flow field. The numerical analysis of vortex methods shows that the truncation error of the method is amplified exponentially in time, at a rate given by the first order derivatives of the flow that are precisely related to the amount of flow strain. In practice, particle distortion can result in the creation and evolution of spurious vortical structures due to the inaccurate resolution of areas of high shear and to inaccurate approximations of the related derivative operators.
To remedy this situation, *location processing* techniques reinitialize the distorted particle field onto a regularized set of particles and simultaneously accurately transport the particle quantities. The accuracy of remeshing has been thoroughly investigated in [Koumoutsakos, 1997]. The Remeshing is shown to introduce numerical dissipation that is far below the dissipation introduced by time and spatial discretizations. One way to regularize the particles is setting the new particle positions to be on the grid node positions and recomputing the transported quantities with a particle-mesh operation.

In order to demonstrate the need of the remeshing step, we consider the vorticity equation without the viscosity term ($\nu = 0$).
In this case the vorticity evolves according to the Euler equation $\frac{D\omega}{Dt} = 0$. As initial condition we set a radial function:

$$\omega_0(\mathbf{x}) = \omega_{\max} \cdot \max\left(0, 1 - \|\mathbf{x}\|/R\right), \tag{4.1}$$

where W is the maximum vorticity and R controls the support of ω_0. Since the vorticity is radially symmetric and there is no diffusion, the system is in a steady state: the exact solution in time is just the initial condition ($\omega(t) = \omega_0$). We can therefore use this problem as validation test and study how is important to remesh the particles during the simulation. Figure 6 shows the crucial difference between performing and not performing the remeshing step. In this case we used $W = 100$, $R = 0.5$, and a time step $\delta t = 5 \cdot 10^{-3}$. When no remeshing step is performed, the solver generates growing spurious structures which lead to a break in the radial symmetry of the vorticity field. This break causes an highly increasing inaccuracy of the computed solution.

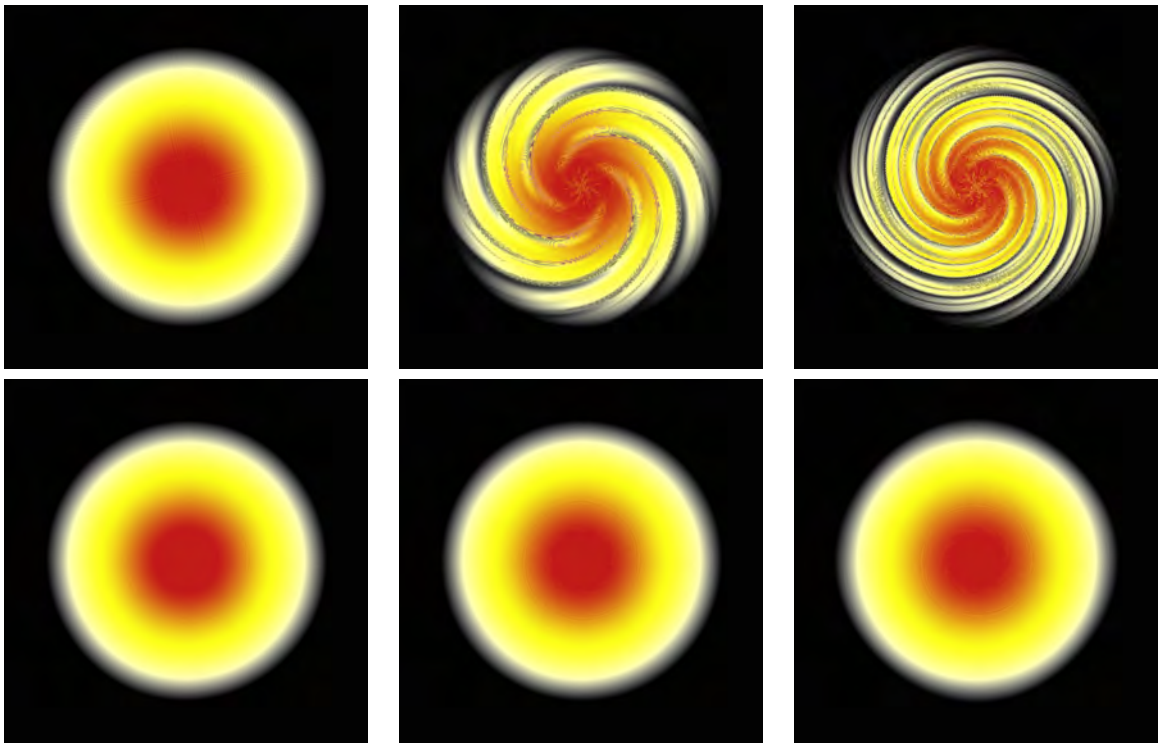


Figure 6: **Why do we need to remesh ?** Inviscid evolution of a 2D axisymmetric vorticity field (an exact solution of the Euler equations) at time $t = 0.01$, $t = 0.10$ and $t = 0.15$; using a second order time integrator for the Euler equation. A grid-free particle simulation produces spurious artifacts that becomes progressively stronger (top). Remeshing the particles enables the method to maintain the axisymmetric profile and to provide an accurate solution of the Euler equations (bottom).

The problem of extracting information on regular grid from a set of scattered points has a long history in the fields of interpolation [Schoenberg, 1946]. To facilitate the analysis we restrict our attention to a one dimensional equispaced regular grid with unit mesh-size onto which we interpolate quantities (q_n) from scattered particle locations(x_n) :

$$Q(x) = \sum_n q_n \, W(x - x_n). \tag{4.2}$$

The properties of the interpolation formulas can be analysed through their behavior in the Fourier space [Schoenberg, 1946]. The *characteristic function* $g(k)$ of the interpolating function W(x) is defined as :

$$g(k) \;=\; \int_{-\infty}^{+\infty} W(x)\, e^{-ikx}\, dx.$$

When W decays fast at infinity, g is a smooth function and the interpolation formula Eq.4.2 is of degree m if the following two conditions hold simultaneously: (i) $g(k) - 1$ has a zero of order m at $k = 0$ and (ii) $g(k)$ has zeros of order m at all $k = 2\pi n, \quad (n \neq 0)$. These requirements

translated back in the physical space are nothing but the moment properties of the interpolant

$$\int W(y)\,dy = 1\ ;\ \int y^{\alpha} W(y)\,dy = 0,\ \text{if } 1 \leq |\alpha| \leq m-1\,.$$

This is reminiscent of the conditions for accurate function particle approximations using moment conserving kernels. In fact the interpolation accuracy [Hockney and Eastwood, 1988] can be described by splitting the interpolation error into a convolution and sampling error reminiscent of the smoothing/quadrature error for function approximations. Hence, good interpolation schemes are those that are band-limited in the physical space and are simultaneously close approximations of the ideal low pass filter in the transformed space. Monaghan [Monaghan, 1985a] presents a systematic way of increasing the accuracy of interpolating functions, such as B-splines, while maintaining their smoothness properties using extrapolation. He constructs interpolation formulas such that if $m = 3$ or $m = 4$ the interpolation will be exact for quadratic functions, and the interpolation will be third or fourth order accurate. One widely used formula involves the so-called M_4' function

$$M_4'(x) = \begin{cases} 0 & \text{if } |x| > 2 \\ \frac{1}{2}(2-|x|)^2(1-|x|) & \text{if } 1 \leq |x| \leq 2 \\ 1 - \frac{5x^2}{2} + \frac{3|x|^3}{2} & \text{if } |x| \leq 1 \end{cases} \tag{4.3}$$

Interpolations in higher dimensions can be achieved by tensorial products of these formulas. However, these tensorial products require particle remeshing on a regular grid. For non grid-conforming boundaries, remeshing introduces particles onto areas that are outside the flow domain and violates the flow boundary conditions. Remedies such as one-sided interpolation have been proposed and a working solution can be obtained ([Cottet and Poncet, 2004] and [Ploumhans et al., 2002]) by eliminating particles outside the domain and adjusting accordingly the modification of particle strengths by re-enforcing the boundary conditions in a fractional step algorithm. Alternatively, *Weight Processing schemes* attempt to explicitly [Beale, 1986] or implicitly [Strain, 1997] modify the particle weights in order to maintain the accuracy of the calculation but they result in rather costly computations.

Finally one lingering question is : when to remesh ? Practice indicates that remeshing at every time step does not detract from the accuracy of the method and at the same time enables the use of the grid to develop differential operators. In [Hieber and Koumoutsakos, 2005] a criterion was developed that is well suited to particle methods and their caharcteristics as a partition if unity technique. In order to determine the rate at which particle remeshing is necessary it is possible to introduce a measure of distortion. This measure relies on the fact that the weighted sum $H(t)$ over all particles must be equal to unity in a regularised particle map

$$H(t) = \sum_j v_j(t)\zeta_\epsilon(\mathbf{x}_p(t) - \mathbf{x}_j(t)) \tag{4.4}$$

$$H(0) = H_0 = 1. \tag{4.5}$$

The average change of $H(t)$ over all particles is a measurement of distortion

$$\Delta H = \frac{1}{N_p}\sum_j \frac{|H_j(t) - H_{0,j}|}{H_{0,j}}, \tag{4.6}$$

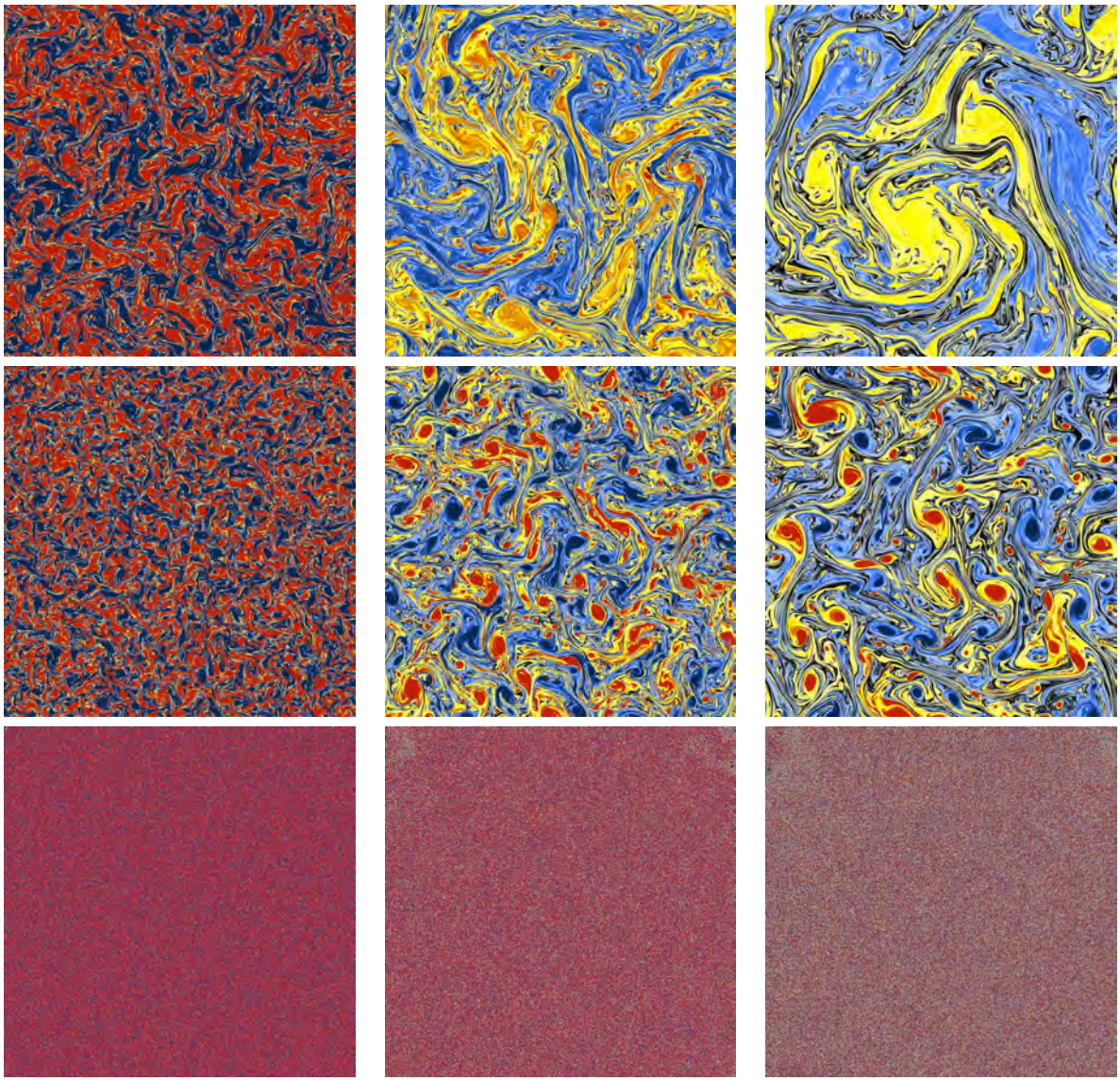


Figure 7: **Why do we need to remesh ?** Evolution of an initially random vorticity distribution, using a first order time integrator (top), a 4th order time integration (middle), a 4th order time integrator without remeshing during the simulation (bottom), at time $t = 0.01$, $t = 0.1$ and $t = 0.2$. Note that without remeshing the random field remains random, while first order time integration schemes introduce high viscosity to the flow.

where N_p is the number of particles. When considering particles undergoing a solid body motion or rotation there is no particle distortion and as such $\Delta H = 0$ Remeshing can be invoked each time the function ΔH exceeds a small prespecified threshold.

4.2 Communication between particles and meshes

The combined use of particles and meshes has been dictated in the past by the need to accelerate the velocity evaluations in vortex particle methods as in the PIC methodology. As we discuss here, in addition to facilitating the computation of the velocity field the mesh is needed for the regularisation of the particle locations.
The mesh is used in the present framework to regularize the particle locations, with the grid nodes becoming particles that are convected in a Lagrangian manner in the following time step. Going between particles and mesh requires the definition of two interpolating operators. The **Particle-**

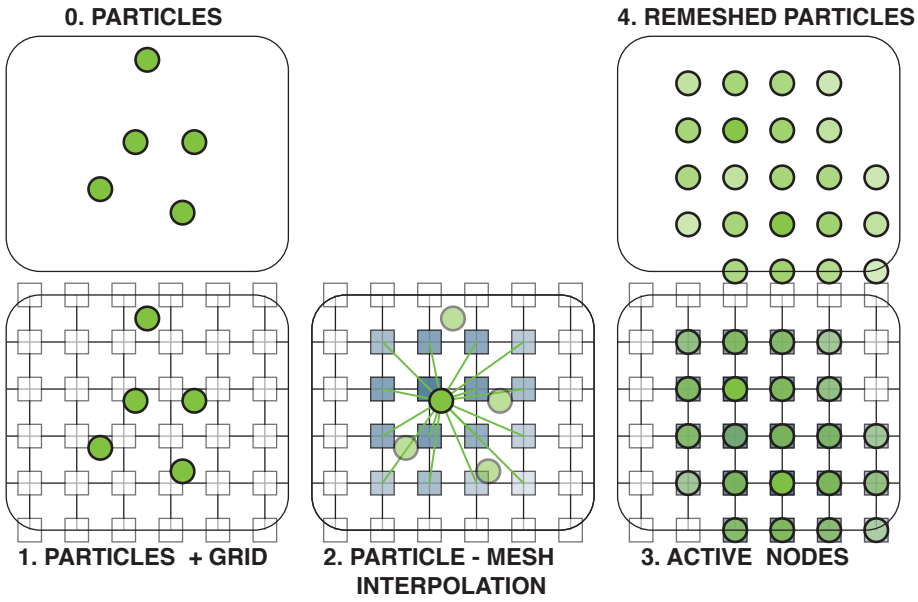


Figure 8: Remeshing of Particles on a Regular Grid. The particles are superimposed on the grid and their values are interpolated onto the grid nodes. After eliminating grid nodes with value below a threshold the grid nodes become particles ready to be convected by the flow field

Mesh interpolation ($M \leftarrow P$) is denoted as $\mathbf{I}_P^M$. On a given a set of particles $\{(\omega_p, \mathbf{x}_p)\}$, the $\mathbf{I}_P^M$ maps the particle vorticity onto grid nodes with grid spacing h as

$$(\mathbf{I}_P^M\{(\omega_p, \mathbf{x}_p)\}, \{\mathbf{x}_m^{mesh}\}) \rightarrow \omega_m^{mesh} = \sum_p \omega_p \cdot W(\frac{1}{h}(\mathbf{x}_m^{mesh} - \mathbf{x}_p)). \tag{4.7}$$

The **Mesh-Particle** interpolation ($P \leftarrow M$) is denoted as $\mathbf{I}_M^P$. Given the vorticity on the mesh one can recover the vorticity of each particle by defining the mesh-particle operation $\mathbf{I}_M^P$:

$$(\mathbf{I}_P^M\{(\omega_m, \mathbf{x}_m^{mesh})\}, \{\mathbf{x}_p)\} \rightarrow \omega_p = \sum_m \omega_i^{mesh} \cdot W(\frac{1}{h}(\mathbf{x}_p - \mathbf{x}_m^{mesh})). \tag{4.8}$$

The interpolation kernel can be expressed as tensorial product $W(x,y) = W(x)W(y)$. For example the M_4' kernel conserves the total value, the linear and angular impulse between quantities on particles and the mesh and it is expressed as :

$$W(x)\, M_4'(x) = \begin{cases} 0 & \text{if } |x| > 2 \\ \frac{1}{2}(2-|x|)^2(1-|x|) & \text{if } 1 \le |x| \le 2 \\ 1 - \frac{5}{2}x^2 + \frac{3}{2}|x|^3 & \text{if } 1 \ge |x| \end{cases} \tag{4.9}$$

Note that using M_4' as interpolation kernel is equivalent to the Catmull-Rom spline interpolation used in computer graphics.
To formally express the remeshing operation, we can suppose that we have a particle set $S = \{(\omega_p, \mathbf{x}_p)\}$ and $W(.)$. Then, the result of a remeshing operation is the following particle set:

$$\text{Remeshing}(\ \{(\omega_p, \mathbf{x}_p)\}\ ) = (\mathbf{I}_P^M\{(\omega_p, \mathbf{x}_p)\}, \{\mathbf{x}_m^{mesh}\}). \tag{4.10}$$

4.2.1 Efficient implementation of Particle-Mesh interpolation

The efficiency gain of hybrid particle methods over non-hybrid particle methods hinges on the efficient implementation of Particle-Mesh and Mesh-Particle interpolations. As we have seen before, the $\mathrm{P} \to \mathrm{M}$ interpolation has the form

$$q_i = \sum_p Q_p\, M(i\,h - x_p)\,. \tag{4.11}$$

The most straightforward way to implement **??** is a verbatim translation of the expression into code, *i.e.* to sum over all particles and evaluate the kernel function M. Clearly, this is very inefficient due to the locality of the kernel M. A more efficient approach would be to use cell lists an loop over cell lists located around the target grid point i. Still this involves calculating distances of particles to i, and we perform many extra operations.
The key to an efficient implementation of **??** lies in realizing that the operation is usually performed for a whole grid, *i.e.* a set of grid points i, and then to flip the loops as illustrated by Algorithm 4.

4.3 Evaluation of differential operators

Our hybrid particle mesh methods have a one-to-one relation between particles and the mesh. Thus we can make use of the efficiency of the evaluation of differential operators on a regular grid by interpolating particle quantities onto the grid ($\mathrm{P} \to \mathrm{M}$), evaluating the operators ($\mathrm{M} \to \mathrm{M}$) and interpolating the result back onto the particles ($\mathrm{M} \to \mathrm{P}$):

Algorithm 5 Mesh to Particle to Mesh Operators

(1) $\mathrm{P} \to \mathrm{M}$ $q_i = \sum_p Q_p\, M(i\,h - x_p)$
(2) $\mathrm{M} \to \mathrm{M}$ $r_i = (\Delta^{h,\mathrm{FD}}\, q)_i$
(3) $\mathrm{M} \to \mathrm{P}$ $(\Delta q)(x_p) = \sum_j M(x_p - j\,h)\, r_j$

Algorithm 4 $\mathrm{P} \to \mathrm{M}$ interpolation; in this 3D example we assume that the computational domain starts at the origin, the indices start at zero, and that we are using a kernel with support 4. The index i is a symbolic abbreviation for i, j, k, *e.g.* $i1[1] \equiv j1$.

For $p \in \mathcal{P}$
$\hat{x} = x_p\, h^{-1}$
$i1 = \mathrm{INT}(\hat{x})$; $i0 = i1 - 1$; $i2 = i1 + 1$; $i3 = i1 + 2$
$x1 = \hat{x} - \mathrm{REAL}(i1)$; $x0 = x1 - 1.0$; $x2 = x1 + 1.0$; $x3 = x1 + 2.0$
$a0 = M(x0)$; $a1 = M(x1)$; $a2 = M(x2)$; $a3 = M(x3)$
In these kernel evaluation statements we can exploit the fact that the kernel M is usually a piecewise polynomial, and that we a-priori know which interval of the kernel $x0$, $x1$, *etc.* fall into; this saves us from using any conditionals.
$q[i0[0], i0[1], i0[2]] = q[i0[0], i0[1], i0[2]] + Q_p\, a0[0]\, a0[1]\, a0[2]$
$q[i0[0], i0[1], i1[2]] = q[i0[0], i0[1], i1[2]] + Q_p\, a0[0]\, a0[1]\, a1[2]$ ...
$q[i3[0], i3[1], i3[2]] = q[i3[0], i3[1], i3[2]] + Q_p\, a3[0]\, a3[1]\, a3[2]$
END For
return

Together this yields

$$(\Delta q)(x_p) = \sum_i M(x_p - i\,h)(\Delta^{h,\mathrm{FD}})\Big(\sum_{p'} Q_{p'}(i\,h - x_{p'})\Big)\,, \tag{4.12}$$

or

$$\Delta^h q = \underline{M}^{\mathrm{T}}\, \underline{D}\, \underline{M}\, Q\,. \tag{4.13}$$

Next to the efficiency gain by bypassing particle-particle interactions, this approach plays an enabling role in adopting immersed interface techniques (see Chapter 3), and multiresolution (see Chapter 5).

Velocity evaluation The velocity evaluation involves solving the Poisson equation 6.23 and computing the curl of the resulting stream function Ψ. In the following we will be dealing with unbounded or periodic flows only. In these cases we use FFT-based Poisson solvers. As we do not require the stream function as such, but only its curl, we can perform the curl also in Fourier space. Thus the velocity evaluation takes the form: (i) Transform vorticity into Fourier space, (ii) evaluate velocity as $\hat{u}(k) = k \times \hat{\omega}\,|k|^{-2}$, (iii) transform velocity into physical space. In the case of unbounded domains we also use FFTs and apply the technique introduced by Hockney and Eastwood in [Hockney and Eastwood, 1988].

Stretching and dissipation The stretching and the dissipation are computed in physical space on the grid as

$$\begin{aligned}
&\nabla^h \cdot (u_x\,\omega) + \mathrm{Re}^{-1}\,\Delta^h\,\omega_x\\
&\nabla^h \cdot (u_y\,\omega) + \mathrm{Re}^{-1}\,\Delta^h\,\omega_y\\
&\nabla^h \cdot (u_z\,\omega) + \mathrm{Re}^{-1}\,\Delta^h\,\omega_z\,,
\end{aligned}$$

where ∇^h is the fourth-order finite difference approximation of ∇, and Δ^h is the fourth-order finite difference approximation of the the Laplacian. As the velocities are spectrally accurate

using fourth-order differences here is beneficial if the flow is well resolved, and leads to more accurate results.

Figure 9: Simulations of the thin double shear layer. Red denotes high positive vorticity, and blue denotes high negative vorticity. From left to right: simulation using 256x256, 512x512 and 2048x2048 particles. Note the development of a spurious vortex for the two lower resolutions

4.3.1 The advantages of structure

On a structured grid physical neighborhood and logical neighborhood (in memory, access) usually coincide. Compare the following evaluation of Δq in 2D at a given point (on a particle, on a grid point, respectively):

Algorithm 6 Laplacian on the grid

$\Delta^{h=1} q \approx (q_{i+1,j} + q_{i-1,j} + q_{i,j+1} + q_{i,j-1} - 4\, q_{i,j})$

Algorithm 7 Laplacian on particles

For $p' = 1, N_{\text{neighbors}}(p)$
  $l = \text{NeighborIndex}(p')$
  $(\Delta^{\varepsilon,h} q)_{p'} += \eta(x_p - x_l)\,(q_l - q_p)$
END For

Evidently, the minimal size, direct access evaluation on the grid is more efficient and in general leads to less cache misses than the particle evaluation. Additionally, the grid-based evaluation is much easier to program as it does not require auxiliary data structures such as verlet lists, or cell lists.
The particle solver has been validated on a number of benchmark tests for accuracy and efficiency. The thin double shear layer is a challenging benchmark for incompressible flow solvers. Brown

and Minion [Minion and Brown, 1997] have demonstrated that in under-resolved simulations spurious vortices infiltrate the numerical solution in discretizations by various computational methods. We have computed the double shear layer problem as presented in [Minion and Brown, 1997], studying the effect of solving the vorticity at low-resolutions often associated with the creation of secondary spurious vortices. The domain is again the unit square with periodic boundary conditions with the initial condition for the velocity field $\mathbf{u} = (u, v)$ in the following non-dimensional form:

$$\begin{cases} u(x,y) = \tanh(\rho \cdot \min(y - 0.25, 0.75 - y)) \\ v(x,y) = \delta \cdot sin(2\pi(x + 0.25)) \end{cases}$$

In the present simulations we consider the thin shear layer obtained by setting $\delta = 0.05$, $\rho = 80$ and a viscosity $\nu = 10^{-4}$.
All simulation were performed using the 4th order Runge-Kutta with a timestep $dt = 0.02$ for $t_{\text{end}} = 1.0$ The numerical results are depicted in 9 in the form of vorticity for three different resolutions. It is evident that a spurious vortex is present in the simulation result with the coarsest resolution. The spurious vortices are eliminated using a hybrid particle method with 512x512 grid/particles. Note however the solution shows some minor undulations instead of the expected straight line [Minion and Brown, 1997], near to the center of the domain.
This numerical artifact disappears when using 2048x2048 computational elements.
Next we consider the case of viscous vorticity decay from an initially uniform random distribution with an average of zero vorticity and a maximum value of 400. The considered physical domain was the unit square with periodic boundary conditions, and the viscosity was set to $\nu = 10^{-7}$. Figure 7 shows the evolution of the flow obtained with a first order time integrator and a timestep $dt = 0.001$ and with a 4th order Runge-Kutta time integrator. The utilization of a first order time integration scheme introduces a large amount of numerical viscosity producing large, weak vortex cores. On the other hand, the 4th order Runge-Kutta scheme succeeds in restraining the effects of numerical viscosity producing smaller vortices of higher intensity. Furthermore we note in particular the role of remeshing : In the absence of remeshing the random vorticity field remains random, as the low viscosity is overwhelmed by the chaotic motion of the particles and no structure emerges as one would expect from a viscous flow field.

4.4 A REMESHED particle method

We finally present the algorithm that is being used in our function evaluations
We note here two important facts: this algorithm is fast because every differential operation is performed in an Eulerian frame, without evaluating any kernel. Also the convection is performed in a fast way: we are dealing with particles, so we do not have a classical CFL condition on $\mathbf{u}$, we can take large time steps because they are bounded according to $\delta t \sim 1/||\nabla \mathbf{u}||_2$.
At the same time the convection is solved accurately because is solved in a Lagrangian fashion, i.e. following the characteristics of the solution. This method has been implemented efficiently in parallel computer architectures leading to unprecedented simulations using billions of particles (Figure 16) the simulation of aircraft wakes [**?**]

Algorithm 8 Procedure of a hybrid vortex method using Williamson's Runge-Kutta scheme

Set up, initial conditions, *etc.* , $t = 0$
Particle quantities will be stored in arrays, *e.g.* vorticity:
$\omega \in \mathcal{R}^{\ni \times \mathcal{N}}$. For the ODE solver we need two temporary variables: $u0$, and $d\omega 0$
while $t \leq T$
  For $l = 1$ to 3 stages of the ODE Solver
    Interpolate ω onto the grid ($\omega \rightarrow \omega_{ijk}$)
    Compute velocity u_{ijk} from ω_{ijk}
    $d\omega_{ijk} \leftarrow$ Compute stretching and dissipation from u_{ijk} and ω_{ijk}
    $d\omega \leftarrow$ Interpolate $d\omega_{ijk}$ onto the particles
    $u0 \leftarrow u + \alpha_l \, u0$;
    $d\omega 0 \leftarrow d\omega + \alpha_l \, d\omega 0$
    $\alpha = (0, -\frac{5}{9}, \frac{153}{128})$
    $x \leftarrow x + \delta t \, \beta_l \, u0$;
    $\omega \leftarrow \omega + \delta t \, \beta_l \, d\omega 0$
    $\beta = (\frac{1}{3}, \frac{15}{16}, \frac{8}{15})$
  END For
END **while**

5 TIME INTEGRATORS FOR PARTICLES

5.1 Introduction

In a vast majority of particle methods, the evolution of the particles affects their location $\mathbf{x}$ and their properties

$$\frac{d\mathbf{x}}{dt} = \mathbf{u} \tag{5.1}$$

$$\frac{dq}{dt} = f(\mathbf{x}, q, \mathbf{u}, \ldots) \tag{5.2}$$

An example is Newton's laws of motion where the property is momentum

$$\frac{d\mathbf{x}}{dt} = \mathbf{u} \tag{5.3}$$

$$m\frac{d\mathbf{u}}{dt} = \mathbf{F} \tag{5.4}$$

where m is the particle mass and $\mathbf{F}$ is the force on the particles. For example a particle with charge q moving in electric field $\mathbf{E}$ and magnetic field $\mathbf{B}$ is the sum of the electrical force $q\mathbf{E}$ and the Lorentz force $q\mathbf{u} \times \mathbf{B}$.
This chapter considers the discretization of the time derivatives on the left-hand side. In particular, we discuss the properties of integrations schemes and how they affect our choice in the case of particle methods.

5.2 Properties of time integration schemes

A first characteristic is the implicit or explicit character of a scheme. A general linear multi-step scheme for Eqs. 5.1 and 5.2 can be written as

$$\sum_{i=0}^{k} \alpha_{k-i} \begin{pmatrix} \mathbf{x} \\ q \end{pmatrix}^{n+k-i} = \delta t \sum_{i=0}^{k} \beta_{k-i} \begin{pmatrix} \mathbf{u} \\ f \end{pmatrix}^{n+k-i} . \tag{5.5}$$

From this expression, we can isolate two families of schemes

Explicit Schemes $\beta_k = 0$, $(\mathbf{x}, q)^{n+k}$ can be directly solved in terms of the previously computed quantities.

Implicit Schemes $\beta_k \neq 0$, we need to solve for $(\mathbf{x}, q)^{n+k}$. Such schemes display very good stability (see Section **??**) but are expensive.

In addition, the choice of an integration scheme will involve the following criteria,

1. consistency
2. accuracy
3. stability
4. efficiency

We now discuss these requirements, while keeping in mind the form of the particle evolution equations.

5.3 Consistency

After time discretization, the resulting set of equations should approximate approximate the original differential equation, as we take $\delta t \to 0$. Consider the ODE $\frac{dx}{dt} = u$. This verification can be carried out through a Taylor series expansion of each term in the scheme. Let us take the approximation $\frac{x^{n+1}-x^n}{\delta t} = u^n$ as an example. We develop the term $x^{n+1} = x(t+\delta t)$ about t

$$\begin{aligned} x^{n+1} - x^n &= x(t^n + \delta t) - x(t^n) \\ &= [x(t^n) + \frac{dx}{dt}\,|_{t=t^n}\,\delta t + \frac{d^2x}{dt^2}\,|_{t=t_n}\,\frac{\delta t^2}{2} + \dots] - x(t^n) \end{aligned}$$

We the obtain a modified equation

$$\left.\frac{dx}{dt}\right|_{t=t^n} + \left.\frac{d^2x}{dt^2}\right|_{t=t_n} \frac{\delta t}{2} + \dots = u^n \tag{5.6}$$

which tends toward the analytical one as $\delta t \to 0$. Our scheme is therefore consistent.
The idea of consistency can be taken further. One can develop integration schemes which conserve some of the structure and the features of the exact equations. One such important feature is time reversibility, a feature of Newton's laws of motion, Eq. 5.3. The integration of particles forward in time, then reversing velocity and time should place the particles back at their original positions.

Example 5.1. *We consider a first scheme*

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \mathbf{u}^n * \delta t \tag{5.7}$$

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \frac{\mathbf{F}(\mathbf{x}^n)}{m}\,. \tag{5.8}$$

A step back can be written as $\mathbf{x}^{n'} = \mathbf{x}^{n+1} - \mathbf{u}^{n+1}\delta t$*, which in terms of* $\mathbf{x}^n$ *yields* $\mathbf{x}^{n'} = \mathbf{x}^n + (\mathbf{u}^n - \mathbf{u}^{n+1})\delta t$*. Because, in general,* $\mathbf{u}^n \neq \mathbf{u}^{n+1}$*, this is not reversible.*

The error term $\mathbf{u}^{n+1} - \mathbf{u}^n$ suggests that time reversible difference approximations are rather obtained by defining time-centered differences. By considering velocity values at $t^{n+1/2}$, we can center the whole scheme[2], which gives us the Leapfrog scheme

$$\frac{\mathbf{x}^{n+1} - \mathbf{x}^n}{\delta t} = \mathbf{u}^{n+1/2} \tag{5.9}$$

$$\frac{\mathbf{u}^{n+1/2} - \mathbf{u}^{n-1/2}}{\delta t} = \mathbf{F}(\mathbf{x}^n) \tag{5.10}$$

When one can not employ the *time-staggered* arrangement of Eqs. 5.9 and 5.10, the time symmetry requirement often leads to an implicit scheme.

5.4 Accuracy

The accuracy of time integration is influenced by two factors: the trunctation error and the round-off error.is the scheme itself, the other is the discrete arithmetic, inherent to computations.

[2] this only holds for a situation where the forces $\mathbf{F}$ only depend on the positions $\mathbf{x}$; an example is damping, a case where the force depends on the velocities and incidentally, energy is not conserved.

5.4.1 Truncation error

This first error originates in the scheme itself: it is the difference between the exact time integration from t to δt and its approximation

$$T^n = \frac{\mathbf{x}^{n+1}_{\text{exact}} - \mathbf{x}^{n+1}_{\text{numerical}}}{\delta t} \ . \tag{5.11}$$

It is evaluated by substitution of $\mathbf{x}^{n+1}_{\text{exact}}$ by a Taylor series centered at $(t^n, \mathbf{x}^n)$

$$\mathbf{x}^{n+1}_{\text{exact}} = \mathbf{x}^n + \delta t \left.\frac{d\mathbf{x}}{dt}\right|_{t=t^n} + \frac{\delta t^2}{2} \left.\frac{d^2\mathbf{x}}{dt^2}\right|_{t=t^n} + \mathcal{O}(\delta t^3) \ .$$

and replacing $\mathbf{x}^{n+1}_{\text{numerical}}$ with the expression given by the scheme.

Example 5.2. *Let us look at at the accuracy of a few schemes.*

Explicit Euler *The scheme is $\mathbf{x}^{n+1} = \mathbf{x}^n + \delta t\, \mathbf{u}^n$. The truncation error is given by*

$$T^n = \frac{\mathbf{x}^n + \delta t\, \mathbf{u}^n + \delta t^2/2\, d^2\mathbf{x}/dt^2 + \cdots - \mathbf{x}^n - \delta t\, \mathbf{u}^n}{\delta t} \simeq \frac{\delta t}{2} \frac{d^2\mathbf{u}}{dt^2}$$

which makes it a first order term.

Implicit Euler $\mathbf{x}^{n+1} = \mathbf{x}^n + \delta t\, \mathbf{u}^{n+1}$. *This scheme is also first order accurate.*

Trapezoidal rule $\mathbf{x}^{n+1} = \mathbf{x}^n + \frac{\delta t}{2}(\mathbf{u}^n + \mathbf{u}^{n+1})$. *This scheme is second order accurate. The averaging cancels the first order errors of the implicit and explicit Euler.*

Adams-Bashforth 2 $\mathbf{x}^{n+1} = \mathbf{x}^n + \frac{\delta t}{2}(3\mathbf{u}^n - \mathbf{u}^{n-1})$. *This scheme is second order accurate, $T^n = \frac{5}{12}\frac{d^3x}{dt^3}\delta t^2$, at the cost of the storage of a past value $\mathbf{u}^{n-1}$.*

Adams-Bashforth 3 $\mathbf{x}^{n+1} = \mathbf{x}^n + \frac{\delta t}{12}[23\mathbf{u}^n - 16\mathbf{u}^{n-1} + 5\mathbf{u}^{n-2}] + O(\frac{3}{8}\mathbf{x}^{(4)}\delta t^3)$

Leapfrog *The approximation of Eqs. 5.9 and 5.10 is second order accurate. Indeed, one can substitute the expressions of $\mathbf{u}^{n-1/2}$ and $\mathbf{u}^{n+1/2}$ from Eq. 5.9 in Eq. 5.10,*

$$\frac{\mathbf{x}^{n+1} - 2\mathbf{x}^n - \mathbf{x}^{n-1}}{\delta t^2} = \frac{\mathbf{F}(\mathbf{x}^n)}{m} \ . \tag{5.12}$$

We can consider the modified equation for the exact solution $\mathbf{x}^n_{exact}$. We first develop the solution around t^n, for $t^n + \delta t$ and $t^n - \delta t$

$$\mathbf{x}^{n+1}_{exact} = \mathbf{x}^n + \delta t \left.\frac{d\mathbf{x}}{dt}\right|_{t^n} + \frac{\delta t^2}{2} \left.\frac{d^2\mathbf{x}}{dt^2}\right|_{t^n} + \frac{\delta t^3}{6} \left.\frac{d^3\mathbf{x}}{dt^3}\right|_{t^n} + \mathcal{O}(\delta t^4)$$
$$\mathbf{x}^{n-1}_{exact} = \mathbf{x}^n - \delta t \left.\frac{d\mathbf{x}}{dt}\right|_{t^n} + \frac{\delta t^2}{2} \left.\frac{d^2\mathbf{x}}{dt^2}\right|_{t^n} - \frac{\delta t^3}{6} \left.\frac{d^3\mathbf{x}}{dt^3}\right|_{t^n} + \mathcal{O}(\delta t^4)$$

If we take the sum of the two expressions, we find the modified equation

$$\frac{\mathbf{x}^{n+1}_{exact} - 2\mathbf{x}^n_{exact} - \mathbf{x}^{n-1}_{exact}}{\delta t^2} = \left.\frac{d^2\mathbf{x}}{dt^2}\right|_{t^n} + \mathcal{O}(\delta t^2) = \frac{\mathbf{F}(\mathbf{x}^n)}{m} \ . \tag{5.13}$$

5.4.2 Round-off error

The round-off error arises from the discrete arithmetics and the finite precision of a floating point value on a computer.

Example 5.3. *We compute $x^{n+1} = x^n + u^n * \delta t$ and use 6 significant digit arithmetic. We assume to have $\mathcal{O}(1)$ quantities for the positions and velocities*

$$x_n = x.xxxxxx \mid \dots$$
$$u_n = y.yyyyyy \mid \dots$$

Along with a decreasing time-step value, a increasingly severe round-off can happen

$$\delta t = 10^{-3} \text{ and } \delta t * u_n = 0.00yyyy \mid yy$$
$$\delta t = 10^{-5} \text{ and } \delta t * u_n = 0.0000yy \mid yyyy$$

where the symbol $\mid$ represents the digits lost after addition to x^n.

5.4.3 The Verlet Algorithm

An alternative to leapfrog is the Verlet algorithm

$$\begin{aligned} \mathbf{x}(t_n + \delta t) &= \mathbf{x}(t_n) + \delta t \frac{d\mathbf{x}}{dt} \mid_{t=t^n} + \frac{\Delta t^2}{2} \frac{d^2\mathbf{x}}{dt^2} \mid_{t=t^n} + \frac{d^3\mathbf{x}}{dt^3} \frac{\Delta t^3}{6} + \dots \\ &= \mathbf{x}(t^n) + \mathbf{u}(t^n)\Delta t + \frac{\mathbf{F}(t^n, x^n)}{2m} \Delta t^2 + \frac{\Delta t^3}{6} \frac{d^3x}{dt^3} + \dots \end{aligned}$$

Similarly expanding $x(t^n - \delta t)$ around $x(t^n)$ we obtain that:

$$x(t^n - \delta t) = x(t^n) - u(t^n)\delta t + \frac{f(t^n, x^n)}{2m} - \frac{\delta t^3}{6} \frac{d^3x}{dt^3} + \dots$$

Summing the above equation we obtain that:

$$x(t^n + \delta t) + x(t^n - \delta t) = 2x(t^n) + \frac{f(t^n, x^n)}{m} \delta t^2 + O(\delta t^4)$$

or

$$x(t^n + \delta t) = 2x(t^n) - x(t^n - \delta t) + \frac{f(x^n, t^n)}{m} \delta t^2 + O(\delta t^4)$$

The Verlet algorithm does not need velocities which can be computed as

$$u(t^n) = \frac{x(t^n + \delta t) - x(t^n - \delta t)}{2\delta t} + O(\delta t^2) \, .$$

Variants of this algorithm will be presented in Section 5.7.

5.5 Stability

Stability is concerned with the propagation of errors and the development of errors over time. It arises due to the non-physical solution of the discretized equations.

5.5.1 Linear stability and stability conditions

In general though, a stability analysis cannot be developed for any right-hand side. It is only possible for certain cases of right-hand sides (linear, quadratic) but the first order behavior of the propagation of the error is known to determine the stability behavior. This corresponds to the homogeneous part of the equation

$$\frac{dx}{dt} = f(x,t) \simeq f(x_0,t_0) + \alpha(x-x_0) + \beta(t-t_0)$$

which is the model problem

$$\frac{dx}{dt} = \alpha x \ , \text{ with } x(t=0) = x_0 \ . \tag{5.14}$$

The exact solution is

$$x(t) = x(0)e^{\alpha t} \ .$$

If we solve this equation with explicit Euler, we find that the numerical solution will be

$$x^n = (1+\alpha\,\delta t)^n x^0$$

where we clearly identify an *amplification factor* $(1+\alpha\,\delta t)$. We note that the exact exponential can be developed as

$$e^{\alpha\,\delta t} = 1 + \alpha\,\delta t + \frac{1}{2}(\alpha\,\delta t)^2 + \ldots + \frac{1}{n!}(\alpha\,\delta t)^n + \ldots \tag{5.15}$$

and we see that Euler can reproduce the first two terms —another way of showing that Euler explicit is first order accurate.
In general, α can be complex $\alpha = a + b$ i; the solution is then

$$x(t) = e^{\alpha t} = e^{at}(\cos(bx) + \mathrm{i}\sin(bx)) \ .$$

In the case of growth ($\mathrm{Re}(\alpha) = a > 0$), both the exact and discrete solutions will grow exponentially; this becomes an issue of accuracy. For $\mathrm{Re}(\alpha) = a < 0$, the exact solution decays but the approximate may not decay at all, as in the case $\alpha\,\delta t < -2$. More precisely, the condition

$$|1 + \alpha\,\delta t| < 1 \tag{5.16}$$

defines a region of the complex $\alpha\,\delta t$-plane where the numerical solution decays. Eq. 5.16 is the *stability condition* of explicit Euler and the corresponding complex plane region (see Fig. 10) is also called the *stability region* of the time integration scheme. If we consider the implicit Euler scheme, we have

$$x^{n+1} = x^n + \delta t\,\alpha\,x^{n+1} \Rightarrow x^{n+1} = \frac{1}{1-\alpha\,\delta t}x^n$$

and the amplification factor can be developed as

$$\frac{1}{1-\alpha\,\delta t} \simeq 1 + \alpha\,\delta t + (\alpha\,\delta t)^2 + \ldots \ ;$$

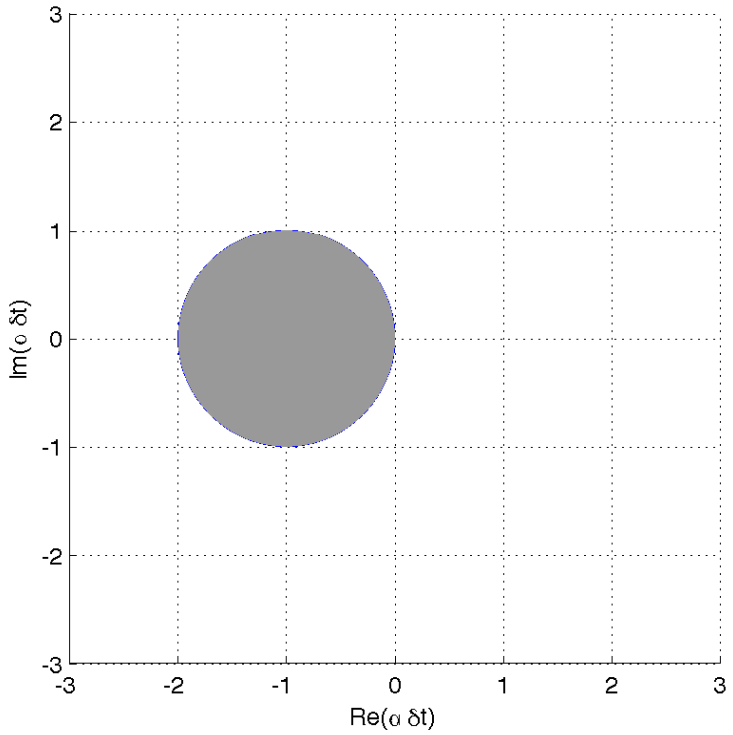


Figure 10: Stability region of Euler explicit

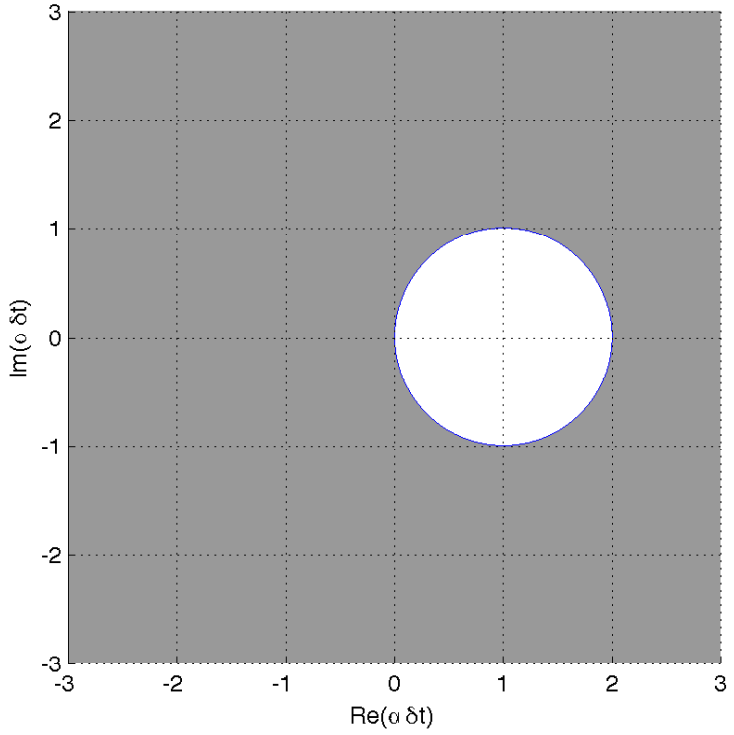


Figure 11: Stability region of Euler implicit

the second order term does not match the one of Eq. 5.15, which shows that Euler implicit is also first order accurate. The stability is this time quite different. We have

$$\frac{1}{1-\alpha\,\delta t} < 1 \text{ for } \mathrm{Re}(\alpha) < 0\ ,$$

making the scheme *unconditionally stable*. The stability region is plotted in Fig. 11. The analysis above can be transposed to the case of a vector variable $\mathbf{x}$

$$\frac{d\mathbf{x}}{dt} = A\mathbf{x} \tag{5.17}$$

The amplification factor becomes an *amplification matrix*. This matrix can be diagonalized

$$\frac{d\chi}{dt} = \Lambda\chi$$

where $\Lambda = E^{-1}AE$, $\chi = E^{-1}\mathbf{x}$ and E is the matrix of the eigenvectors of A. One then sees that the procedure above can be applied to every eigenvelue of A, which will have to fall within the stability region of the integration scheme.

5.5.2 The Verlet algorithm

We now consider the stability of integration schemes for discrete particles. We start with the Verlet algorithm. only in terms of positions. First we look at the behavior of the solution both with and without errors due to discrete arithmetics, $\mathbf{x}^n$ and $\mathbf{X}^n$ respectively. Let us consider the initial conditions $x^0 = X^0, x^1 = X^1$and the first couple of time steps.
In the absence of any error (roundoff) we will get:

$$\text{Step 1: } X^2 - 2X^1 + X^0 = \frac{F(X^1)}{m}\delta t^2$$
$$\text{Step 2: } X^3 - 2X^2 + X^1 = \frac{F(X^2)}{m}\delta t^2\ .$$

Due to inexact arithmetic however we obtain

$$\text{Step 1: } x^2 - 2X^1 + X^0 = \frac{F(X^1)}{m}\delta t^2$$
$$\text{Step 2: } x^3 - 2x^2 + X^1 = \frac{F(X^2)}{m}\delta t^2\ .$$

If we consider the error $\epsilon^n = x^n - X^n$, we get

$$(X^3 + \epsilon^3) - 2(X^2 + \epsilon^2) + X^1 = \frac{F(X^2 + \epsilon^2)}{m}\delta t^2$$

After subtracting the exact equation, we obtain

$$\epsilon^3 - 2\epsilon^2 = \frac{F(X^2 + \epsilon^2)}{m}\delta t^2 - \frac{F(X^2)}{m}\delta t^2$$

Expanding the right-hand side yields

$$\epsilon^3 - 2\epsilon^2 = \epsilon^2(\frac{\delta f}{\delta x} \mid_{x=X^2} \frac{\delta t^2}{m})$$

and similarly, for the next step

$$\epsilon^4 - 2\epsilon^3 + \epsilon^2 = \epsilon^3 \frac{\delta f}{\delta x} \mid_{x=X^3} \frac{\delta t^2}{m}$$

By induction, the error behavior has to be governed by

$$\epsilon^{n+1} - 2\epsilon^n + \epsilon^{n-1} = \epsilon^n \frac{\delta f}{\delta x} \mid_{x=X^n} \frac{\delta t^2}{m} .$$

Before continuing, we remind ourselves of the main stability question: does ϵ^n grow more rapidly than the solution? In the case of a non-growing —oscillatory—solution, does it grow at all? In such a case, we have

$$\frac{d^2X}{dt^2} = -\Omega^2 X \text{ and } X \sim e^{i\Omega t} .$$

Ω corresponds to the stiffness of a mass/spring problem. In our case, we will find a maximum stiffness for the maximum negative value of $\frac{\delta f}{\delta x}$

$$\epsilon^{n+1} - 2\epsilon^n + \epsilon^{n-1} = -\left|-\frac{\delta f}{\delta x}\right|_{\max} \frac{\delta t^2}{m} \epsilon^n$$

If we assume a solution of the form[3] $\epsilon^{(n)} = \lambda^n$, we find

$$\lambda^2 - 2\lambda + 1 = -(\Omega \delta t)^2 \lambda \tag{5.18}$$

where we set $\Omega^2 = \left|-\frac{\delta f}{\delta x}\right|_{\max} / m$. The solutions are

$$\lambda_\pm = 1 - \frac{(\Omega\,\delta t)^2}{2} \pm \left(\frac{(\Omega\,\delta t)^2}{2}\right)\left(1 - \frac{4}{(\Omega\,\delta t)^2}\right)^{1/2} .$$

For a time integration scheme to be stable, we need the complex roots $\lambda_\pm$ to fall within or on the unit circle, i.e. $|\lambda_\pm| \leq 1$.

5.5.3 The leapfrog algorithm

Remember the leapfrog scheme

$$x^{n+1} - x^n = u^{n+1/2}\delta t$$
$$u^{n+1/2} - u^{n-1/2} = \frac{F^n}{m}\delta t$$

[3] We do a temporary change of notation: $^{(n)}$ denotes a quantity at step n, while ^{n} is a exponent

Let $x = X + \epsilon_x$ where X is the exact solution and ϵ_x, the error. Let $u = U + \epsilon_u$ where U is the exact solution and ϵ_u, the error. Assuming ϵ is small so that F can be expanded in terms of a Taylor series

$$\epsilon_x^{n+1} - \epsilon_x^n = \epsilon_u^{n+1/2}\delta t$$
$$\epsilon_u^{n+1/2} - \epsilon_u^{n-1/2} = \frac{\delta F^n}{\delta x}\frac{\delta t}{m}\epsilon_x^n = -\Omega^2\delta t\epsilon_x^n$$

Now we define the error vector

$$\epsilon^n = \begin{bmatrix} \epsilon_x^n \\ \epsilon_u^{n-1/2} \end{bmatrix}$$

and we can write the previous equations in matrix form as:

$$\epsilon^{n+1} - \epsilon^n = \begin{bmatrix} 0 & \delta t \\ 0 & 0 \end{bmatrix}\epsilon^{n+1} + \begin{bmatrix} 0 & 0 \\ -\Omega^2\delta t & 0 \end{bmatrix}\epsilon^n$$

Exercise: See how this works. Collecting terms we write:

$$\begin{bmatrix} 1 & -\delta t \\ 0 & 1 \end{bmatrix}\epsilon^{n+1} = \begin{bmatrix} 1 & 0 \\ -\Omega^2\delta t & 1 \end{bmatrix}\epsilon^n \Rightarrow$$
$$\epsilon^{n+1} = \begin{bmatrix} 1-\Omega^2\delta t^2 & \delta t \\ -\Omega^2\delta t & 1 \end{bmatrix}\epsilon^n \Rightarrow$$
$$\epsilon^{n+1} = G\epsilon^n$$

where

$$G = \begin{bmatrix} 1-\Omega^2\delta t & \delta t \\ -\Omega^2\delta t & 1 \end{bmatrix}.$$

Note that this is an example of amplification matrix, discussed at the end of Section 5.5.1. If E is the matrix of eigenvectors and Λ the diagonal matrix of eigenvalues then we obtain:

$$E^{-1}\epsilon^{n+1} = (E^{-1}GE)(E^{-1})\epsilon^n \Rightarrow \epsilon'^{n+1} = \Lambda\epsilon'^n$$

A time integration scheme is stable if the eigenvalues of its amplification matrix lie on or within the unit circle, i.e. $\mid \lambda \mid_{max} \leq 1$. Quite interestingly, the characteristic polynomial of A is identical to Eq. 5.18.

5.5.4 Remarks

From these few examples, we can already draw some conclusions and remarks

- implicit schemes offer excellent stability but one step can be expensive due to the solution of a complex system
- a subset of implicit schemes offer unconditional stability [**?**, **?**], implicit Euler is an example
- explicit schemes have a more stringent conditional stability but one step is relatively cheap
- we have to keep in mind that the goal is to minimize the time to solution; an explicit time-step might be cheap, but its relatively poor stability can also restrict δt to prohibitively small values.

5.6 Efficiency

Two aspects of efficiency are at play.

5.6.1 Algorithm optimization

With the advent of new super-computing paradigms, and in particular the one of massively parallel machines with little per-processor memory (e.g. less than $1GB$ on IBM Blue Gene), one has to be even more careful in the choice of an integration scheme.
Its *storage* requirements will determine the maximum problem size and for a given size, its stability will affect the *computation time* to reach a solution time.
How does one choose a scheme then? The stability will restrict the time step and from the accuracy point of view, high order schemes can take larger steps...
High-order schemes still have stability limitations however and are expensive both in terms of storage and function evaluations. As a rule of thumb, we will use low order time steps with appropriate time-steps, e.g. Leapfrog|Verlet that have as an extra benefit time symmetry.
A broadly used family of schemes are the low storage Runge-Kutta schemes. We consider the classic 4-th order Runge-Kutta scheme [**?**, **?**]

$$
\begin{aligned}
k_1 &= \delta t\, f(x^n) \\
k_2 &= \delta t\, f(x^n + 1/2 k_1) \\
k_3 &= \delta t\, f(x^n + 1/2 k_2) \\
k_4 &= \delta t\, f(x^n + k_3)
\end{aligned}
$$

$$
x^{n+1} = x^n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + O(\delta t^4)
$$

If N is the dimension of x, the nominal storage requirement is $4N$. This is achieved through the sequential construction of the k_i's. More explicitly, we have

- N for x^n and x^{n+1}
- N for the argument of $f()$ during the computations of the k_i's
- N for the evaluation of f
- N for the storage and accumulation of $k_1 + 2k_2 + 2k_3 + k_4$.

For N_P particles in three dimensions, we have $N = 6N_P$, as

$$
x = (\mathbf{x}_p, \mathbf{u}_p) = \left((x_p^i)_{i=1,2,3}, (u_p^i)_{i=1,2,3}\right) \ .
$$

The Runge-Kutta family of schemes offers some degrees of freedom and it is possible to derive low-storage schemes. The Williamson schemes [Williamson, 1980] are second,third and fourth-order Runge-Kutta schemes that require only $2N$ storage.

Example 5.4. *Third order Williamson scheme*

$$
\begin{aligned}
z^{(0)} &\leftarrow x^n \\
\textit{Substep 1: } q^{(1)} &= \delta t\, f(z^{(0)}) \\
z^{(1)} &= z^{(0)} + \frac{1}{3} q^{(1)} \\
\textit{Substep 2: } q^{(2)} &= \delta t\, f(z^{(1)}) - \frac{5}{9} q^{(1)} \\
z^{(2)} &= z^{(1)} + \frac{15}{16} \delta t\, q^{(2)} \\
\textit{Substep 3: } q^{(3)} &= \delta t f(z^{(2)}) - \frac{153}{128} q^{(2)} \\
z^{(3)} &= z^{(2)} + \frac{8}{15} q^{(3)} \rightarrow x^{n+1}
\end{aligned}
$$

The $z^{(*)}$ represent the state of our integrated variable in memory: the exponent indicates the substep. And similarly, $q^{(*)}$ refers to the temporary storage required by this scheme. Additional storage is needed for the result of the evaluation of $f(z^{(*)})$. This means that only $2N$ additional storage units are needed.

5.6.2 Right-hand side computation

A second origin for efficiency lies in the optimization of the right-hand side computation.
Let us consider the case of Newton's laws where the forces have a short range, i.e. $\mathbf{f}_{ij} = 0$ if $r_{ij} > r_c$. The basic method for evaluating $\mathbf{F}^{\text{sr}}$ is to go through all particles $j = 1, \dots N_p$ and check whether the separation $r_{ij} = |\mathbf{x}_i - \mathbf{x}_j|$ is less than r_c and if so compute $\mathbf{f}_{ij}^{\text{sr}}$ and add it to $\mathbf{F}^{\text{sr}}$. This scales as $O(N_p^2)$ operations.
There are a few algorithms and data structures which make this procedure faster. We here give a couple of them.

5.6.3 Chaining Mesh

We fit a regular lattice of $M_1 \times M_2 \times M_3$ cells, with $M_i = \text{ceil}(L_i/r_c)$ over our set of particles. A cell side is taken as $h = r_c$.
The particles j that interact with a particle i will have to be either in the same cell as i or the 26 neighboring cells (8 in $2D$). If the particle coordinates are sorted into lists for each chaining cell, the computation of the force $\mathbf{F}_i^{\text{sr}}$ on particle i involves $27\, N_c$ tests where $N_c = \frac{N_P}{M_1\, M_2\, M_3} =$ is the average number of particles per cell. One can further use the symmetry of the interactions and compute the $\mathbf{f}_{ij}$ only once and add it to $\mathbf{F}_i$ and $\mathbf{F}_j$. This brings the number of tests to $13.5 \frac{N_P^2}{M_1\, M_2\, M_3}$.

5.6.4 Linked Lists

For certain computers (serial) it is computationally more efficient to sort the coordinate addresses rather than the coordinates themselves.
Let $HOC(q)$ be the head of chain table entry for chaining cell q and $LL(i)$ the link coordinate for particle i.

1. Set $HOC(q) = 0$ for all q.
2. For all particles i
   a) $q = int(\frac{x_1}{HC_1}, \frac{x_2}{HC_2}, \frac{x_3}{HC_3})$
   b) add particle i to head of list for cell q

$$LL(i) = HOC(q)$$
$$HOC(q) = i$$

Sorting 2|3 operations per particle

Once the HOC and LL tables are filled a zero entry in $HOC(q)$ implies no particles. A non-zero entry gives the address of the coordinates of the first particle in the list. The link coordinate of the particle either gives the address of the coordinate of the next particle in the list or a zero particle to indicate the end of the list.

5.7 Considerations for Molecular Dynamics

Newton's equations of motion are time reversible and so should our schemes. In most algorithms however, reversing momenta and time direction would lead to different phase spaces. We note that even reversible,a scheme would be hindered by finite arithmetic.

5.7.1 Liouville operator formalism and symplectic operators

A class of many-body integrators can derived in an elegant formalism [Tuckerman et al., 1992]. These integrators, called symplectic integrators, display very important properties, such as energy conservation, time reversibility and phase space area conservation.
Consider a quantity f which is a function of all the coordinates and momenta of the N particles in a classical many body problem

$$f(\mathbf{x}_p(t)|_{p=1\ldots N}\,,\,\mathbf{u}_p(t)|_{p=1\ldots N})\;. \tag{5.19}$$

The time derivative of f is

$$\frac{df}{dt} = \dot{\mathbf{x}}\frac{\partial f}{\partial \mathbf{x}} + \dot{\mathbf{u}}\frac{\partial f}{\partial \mathbf{u}} = \mathrm{i}\,Lf \tag{5.20}$$

where we introduce the Liouville operator

$$\mathrm{i}\,L = (\mathrm{i}\,L_x + \mathrm{i}\,L_u)f = \dot{\mathbf{x}}\frac{\partial}{\partial \mathbf{x}} + \dot{\mathbf{u}}\frac{\partial}{\partial \mathbf{u}}\;. \tag{5.21}$$

The formalism of Eq. 5.20 allows to integrate f in time

$$f(\mathbf{x},\mathbf{u}) = e^{\mathrm{i}\,Lt} f(\mathbf{x}(0),\mathbf{u}(0)) = e^{(\mathrm{i}\,L_x + \mathrm{i}\,L_u)t} f(\mathbf{x}(0),\mathbf{u}(0))\;.$$

We note that

$$e^{(\mathrm{i}\,L_x+\mathrm{i}\,L_u)t} \neq e^{\mathrm{i}\,L_x t}\cdot e^{\mathrm{i}\,L_u t}$$

as the operators $\mathrm{i}\,L_x$ and $\mathrm{i}\,L_u$ do not commute.

Let us consider the special case of $\mathrm{i}\,L = \dot{\mathbf{x}}(0)\frac{\partial}{\partial \mathbf{x}}$,

$$f(\mathbf{x},\mathbf{u}) = e^{\mathrm{i}\,Lt} f(\mathbf{x}(0),\mathbf{u}(0)) = e^{\mathrm{i}\,L_x t} f(\mathbf{x}(0),\mathbf{u}(0)) \;;$$

a Taylor series about $t = 0$ yields

$$\begin{aligned} f(\mathbf{x}(t),\mathbf{u}(t)) &= f(\mathbf{x}(0),\mathbf{u}(0)) + \mathrm{i}\,L_x t f(\mathbf{x}(0),\mathbf{u}(0)) + \frac{(\mathrm{i}\,L_x t)^2}{2} f(\mathbf{x}(0),\mathbf{u}(0)) + \cdots \\ &= \sum_{n=0}^{\infty} \frac{(\dot{\mathbf{x}}(0)t)^n}{n!} \frac{\partial^n}{\partial \mathbf{x}^n} f(\mathbf{x}(0),\mathbf{u}(0)) \end{aligned}$$

and thus the effect of a shift in position

$$f(\mathbf{x}(t),\mathbf{u}(t)) = e^{\mathrm{i}\,L_x t} f(\mathbf{x}(0),\mathbf{u}(0)) = f(\mathbf{x} + \dot{\mathbf{x}}(0)t, \mathbf{u}(0)) \;.$$

Similarly, we find for a velocity shift

$$f(\mathbf{x}(t),\mathbf{u}(t)) = e^{\mathrm{i}\,L_u t} f(\mathbf{x}(0),\mathbf{u}(0)) = f(\mathbf{x}(0), \mathbf{u}(0) + \dot{\mathbf{u}}(0)t) \;.$$

Let us now combine the effects of these two partial operators. We remember the Trotter identity

$$e^{\alpha+\beta} = \lim_{P\to\infty} (e^{\alpha/2P}\, e^{\beta/P}\, e^{\alpha/2P})^P$$

and in particular, for a large finite P,

$$e^{\alpha+\beta} = (e^{\alpha/2P}\, e^{\beta/P}\, e^{\alpha/2P})^P\; e^{\mathcal{O}(1/P^2)}$$

We apply it to the partial operators by setting $\Delta t = t/P$, $\alpha/P = \mathrm{i}\,L_u t/P = \Delta t\,\dot{\mathbf{u}}(0)\frac{\partial}{\partial \mathbf{u}}$ and $\beta/P = \mathrm{i}\,L_x t/P = \delta t\,\dot{\mathbf{x}}(0)\frac{\partial}{\partial \mathbf{x}}$ with

$$e^{(\mathrm{iL}_u + \mathrm{iL}_x)t} = (e^{\frac{\mathrm{iL}_u t}{2P}}\, e^{\frac{\mathrm{iL}_x t}{p}}\, e^{\frac{\mathrm{iL}_u t}{2P}})^P\; e^{\mathcal{O}(1/P^2)} \;.$$

Now we apply this decomposition, starting with the rightmost one,

$$f_1 = e^{\frac{\mathrm{iL}_u \Delta t}{2}} f(\mathbf{x}(0),\mathbf{u}(0)) = f(\mathbf{x}(0), \mathbf{u}(0) + \frac{\Delta t}{2}\dot{\mathbf{u}}(0))$$

and then, successively,

$$f_2 = e^{\frac{\mathrm{iL}_x \Delta t}{p}}\, f_1 = f(\mathbf{x}(0) + \Delta t\,\dot{\mathbf{x}}(\Delta t/2), \mathbf{u}(0) + \frac{\Delta t}{2}\dot{\mathbf{u}}(0)))$$

$$f_3 = e^{\frac{\mathrm{iL}_u \Delta t}{2}}\, f_2 = f(\mathbf{x}(0) + \Delta t\,\dot{\mathbf{x}}(\Delta t/2), \mathbf{u}(0) + \frac{\Delta t}{2}\dot{\mathbf{u}}(0) + \frac{\Delta t}{2}\dot{\mathbf{u}}(\Delta t)) \;.$$

This sequence can be translated into a time integration scheme by considering the case of a vector $f = (\mathbf{x},\mathbf{u})$

$$\begin{aligned} \mathbf{x}(\Delta t) &= \mathbf{x}(0) + \Delta t\,\mathbf{u}(0) + \frac{\Delta t^2}{2}\frac{\mathbf{F}(\mathbf{x}(0))}{m} \\ \mathbf{u}(\Delta t) &= \mathbf{u}(0) + \frac{\Delta t}{2m}\left(\mathbf{F}(\mathbf{x}(0)) + \mathbf{F}(\mathbf{x}(\Delta t))\right) \end{aligned}$$

which is the velocity-Verlet algorithm [**?**]. It is easy to show that the trajectories obtained with this algorithm are equivalent to the trajectories of the original Verlet algorithm. It is fourth order accurate for positions, and second order for velocities.
The present formalism allows the design of a whole family of schemes. We could have considered a reversed order for the decomposition $\mathrm{iL} = \mathrm{iL}_x + \mathrm{iL}_u$. This produces the position-Verlet algorithm

$$\begin{aligned} \mathbf{u}(\Delta t) &= \mathbf{u}(0) + \Delta t\, \mathbf{F}\left(\mathbf{x}(0) + \frac{\Delta t}{2m}\mathbf{u}(0)\right) \\ \mathbf{x}(\Delta t) &= \mathbf{x}(0) + \frac{\Delta t}{2}\left(\mathbf{u}(0) + \mathbf{u}(\Delta t)\right) \ . \end{aligned}$$

A fourth order variant (both for positions and velocities) is the velocity-corrected Verlet algorithm

$$\begin{aligned} x^{n+1} &= x^n + u^n \delta t + \frac{f(x^n, t^n)}{2m}\delta t^2 + O(\delta t^4) \\ u^n &= \frac{u^{n+1/2} + u^{n-1/2}}{2} + \frac{\delta t}{12}[\frac{f(t^{n-1}, x^{n-1})}{m} - \frac{f(t^{n+1}, x^{n+1})}{m}] + O(\delta t^4) \end{aligned}$$

Note that we can compute the velocities only after the next time step.

5.7.2 Phase-space volume preservation

For a Hamiltonian system, we have that

$$d\mathbf{x}(t)d\mathbf{u}(t) = d\mathbf{x}(0)d\mathbf{u}(0)$$

or that the volume is conserved. Verlet algorithms can be shown to have this conservation property.

5.7.3 Lyapunov Instability

Let $\mathbf{r}(t)$ be the position of one particle i at time t. $\mathbf{r}(t) = f(\mathbf{x}(0), \mathbf{u}(0); t)$ where here $\mathbf{x}(0)$, $\mathbf{u}(0)$ imply the positions and velocities of all the particles. If we perturb the position of this particle at $t = 0$ by ϵ, we will obtain a different trajectory for the particle $\mathbf{r}'_i(t)$. The difference between the two trajectories $\delta(t) = |\mathbf{r}'_i(t) - \mathbf{r}_i(t)|$ will grow as

$$\delta(t) = \epsilon(0)e^{\lambda t}$$

where λ is the Lyapunov exponent. So, for a given acceptable error $\delta_{\max}$ and a maximum time, the acceptable perturbation decreases exponentially

$$\epsilon \sim \delta_{\max}\, e^{-\lambda t_{\max}} \ .$$

It is therefore impossible to find an algorithm that predicts the trajectories of all the particles for long and short times. The trajectory of the phase space is sensitive to initial conditions. We should expect that any integration error —no matter how small—will be exponentially increased. This is the Lyapunov instability.
In molecular dynamics however, we are interested in statistical predictions, in the average behavior of the system but by no means every little detail of the system. This philosophy is quite different from the prediction of satellite trajectories, where local error is critical.

Example 5.5. *Take* 10^3 *particles interaction in* $3D$ *via LJ potentials. Take 2 of these particles and change their velocities by* 10^{-10} *(in reduced units). Plot* $\sum_{i=1}^{N} \mid r_i(t) - r_i'(t) \mid^2$ *as a function of time.*

6 PARTICLE METHODS AND FLOW SIMULATIONS

The flows we describe in this class can be effectively cast in the following form:

$$\frac{\partial u}{\partial t} + \text{div}\,(\mathbf{U}u) = F(u, \nabla u, \cdots) \tag{6.1}$$

where u is a scalar flow property (e.g density) or a vector (e.g. momentum) advected by the velocity vector field $\mathbf{U}$. Equation (6.1) is an advection equation in conservation form in the sense that it if $F = 0$ and no flow comes form the boundaries of the computational box, the scalar property is conserved as :

$$\frac{d}{dt}\int u\,dx = 0.$$

We note that the right hand side F can take various forms involving derivatives of u and depends on the physics of the flow systems that is being simulated. An example for F is the diffusion term ($F(u, \nabla u, \cdots) = \nabla^2 u$). The velocity vector field ($\mathbf{U}$) can itself be a function of u, which leads to *nonlinear* transport equations. A number of examples of different $F, \mathbf{U}, \mathbf{u}$ are given in the following sections.

For simplicity, we first consider the case $F \equiv 0$. The conservative form of the model can be translated in a Lagrangian framework by sampling the mass of u on individual points, or **point particles** whose locations can be defined with the help of Diract δ-functions. Hence when u is initialized on a set of point particles it maintains this descriptions, with particle locations obtained by following the trajectories of the flow:

$$u(x,t) = \sum \alpha_p \delta(x - x_p(t)) \tag{6.2}$$

where

$$\frac{dx_p}{dt} = \mathbf{U}(x_p, t). \tag{6.3}$$

In practice this system of differential equation is solved by a time-discretization method (sometimes called in this context a *"particle pusher"*).

6.1 Smooth Particles for Simulations of Continuum Systems

The point particle approximations have useful computational features as they provide an *exact* representation of convection effects, a feature that has been extensively used in the computer animations of fluids (for example in : [Foster and Metaxas, 1996, CARLSON et al., 2004]). At the same time the point particle approximations need to be enhanced in order to recover continuous fields (see [Cottet and Koumoutsakos, 2000, Pfister and Gross, 2004] and references there in). Among the different approaches of recovering continuous fields from point samples, for reasons that will become evident below, we consider the approach of regularising their support, replacing δ by a smooth *cut-off* function which has the same mass (unity) and a small support:

$$\delta(x) \simeq \zeta_\epsilon(x) = \epsilon^{-d}\zeta(\frac{x}{\epsilon}) \tag{6.4}$$

where d is the dimension of the computational space and $\epsilon << 1$ is the range of the cut-off. In most cases, on uses a function with radial symmetry A typical and often used example is the Gaussian

$$\zeta(x) = \pi^{-d/2}\exp\left(-|x|^2\right)$$

Other functions that "resemble" more to the dirac mass, in the sense that they have the same values (zero) for higher moments, can be constructed in a systematic way, a topic that is beyond the scope of these class notes. We refer to [Cottet and Koumoutsakos, 2000] for further discussions on this issue.
The particle representation formula (6.2) then becomes a *blob* representation

$$u(x,t) \simeq u_\epsilon(x,t) \ = \ \sum \alpha_p \zeta_\epsilon(x - x_p(t)). \tag{6.5}$$

Most importantly, regularization can be used to compute local (e.g. algebraic functions) or non-local (in particular derivatives of any order) quantities based on u. We will show later a number of examples of how this principle is used in practice.
Using smooth particles to solve 6.1 in the general case ($F \neq 0$), one further needs to increment the particle strength by the amount that is dictated from the right hand side F. For that purpose, local values of F at particle locations multiplied by local volumes around particles are required. The local values of F can always be obtained from regularization formulas (6.5). The volumes v of the particles are updated using the transport equation

$$\frac{\partial v}{\partial t} + \text{div } (\mathbf{U}v) = -v \,\text{div } \mathbf{U} \tag{6.6}$$

The particle representation of the solution is therefore given by (6.2), (6.3) complemented by the differential equations

$$\frac{dv_p}{dt} = -\text{div } \mathbf{U}(x_p,t)\, v_p \ , \ \frac{d\alpha_p}{dt} = v_p F_p. \tag{6.7}$$

In (6.2), particle masses represent local integrals of the desired quantity around a particle. Typically, if particles are initialized on a regular lattice with grid size Δx, one will set $x_p^0 = (p_1 \Delta x, \cdots, p_n \Delta x)$ and $\alpha_p = (\Delta x)^d \, u(x_p, t = 0)$. One may also write the weight of the particles as the product of the particle strength and particle volume that are updated separately :

$$\alpha_p = v_p \, u_p. \tag{6.8}$$

6.2 Examples: SPH and Vortex Methods

Two of the most widely used particle methods for flow simulations are Smoothed Particle Hydrodynamics (SPH) and Vortex Methods (VM). We outline here the key elements of these methods with an emphasis on their underlying principles. Extensive reviews of these methods can be found in [Monaghan, 2005, Koumoutsakos, 2005].

6.2.1 Compressible Flows and SPH

The method of SPH was introduced for the study of gas dynamics as they pertain to astronomical systems [Lucy, 1977, Gingold and Monaghan, 1977]. In these notes we introduce for simplicity the numerical formulation of Smoothed Particle Hydrodynamics (SPH), using the Euler equations for gas dynamics in one space dimension. The equations of gas dynamics for the density ρ and the velocity u can be cast in the following form

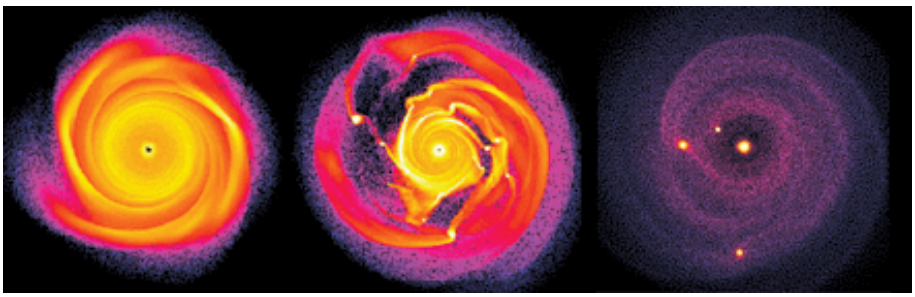


Figure 12: SPH Simulations of protoplanetary disk formation [Kerr, 2002].

$$\frac{\partial \rho}{\partial t} + u \frac{\partial \rho}{\partial x} = -\rho \frac{\partial u}{\partial x} \tag{6.9}$$

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \frac{\partial \tau}{\partial x} \tag{6.10}$$

where τ denotes the fluid stress. This system needs a closure to determine the fluid stress, which in turn requires an energy equation and a constitutive law for the gas under consideration. In that case, particles weights are written using (6.8). To obtain local values (ρ_p, u_p) of density and velocities, the right hand side of (6.9),(6.10) are computed from regularized evaluations of the velocity. Unfortunately, there is a discrepancy between the notation used in the SPH literature and the rest of the particle literature (or the other way around !). In SPH related works the cut-off function are denoted by W, the cut-off range is h instead of ϵ, and $\zeta_\epsilon(x)$ becomes $W(x,h)$. With these notations the divergence of the velocity is given by

$$\frac{\partial u}{\partial x}(x_p) = \sum_q v_q (u_q - u_p) \frac{\partial W}{\partial x}(x_p - x_q, h)$$

The particle representation for u and ρ is therefore given by

$$\rho(x,t) = \sum_p v_p \rho_p \delta(x - x_p) \tag{6.11}$$

$$u(x,t) = \sum_p v_p \delta(x - x_p), \tag{6.12}$$

where the weights of the particle are obtained by solving the differential equations :

$$\dot{v}_p = v_p \sum_q v_q (u_q - u_p) \frac{\partial W}{\partial x}(x_p - x_q, h) \tag{6.13}$$

$$\dot{\rho}_p = -\rho_p \sum_q v_q (u_q - u_p) \frac{\partial W}{\partial x}(x_p - x_q, h) \tag{6.14}$$

$$\dot{u}_p = \sum_q v_q (\tau_q - \tau_p) \frac{\partial W}{\partial x}(x_p - x_q, h). \tag{6.15}$$

This system has to be closed by an additional energy equation.

Note that, in the expression giving the divergence, we have subtracted from the expected expression $\sum_q v_q u_q \frac{\partial W}{\partial x}(x_p - x_q, h)$ the term $\sum_q v_q u_p \frac{\partial W}{\partial x}(x_p - x_q, h)$. In the limit of an infinite number of particles, this term vanishes since it tends to the integral of the the function $\frac{\partial W}{\partial x}$. Its contribution is to maintain for a finite number of particles the conservativity of the method. This issue of conservation (of mass, energy ..) is indeed a central issue in SPH methods and has been the subject of many works. The approach to derive schemes that have these properties is often very closely related to deriving particles dynamics that mimic at the discrete level the underlying physics. For these reasons, SPH simulations are very appealing and often give qualitatively satisfying results even with a rather small degrees of freedom, and this explains their popularity in the graphics community and the animation industry (see Fig.13 and [Losasso et al., 2008])

However, as one may wish to increase the reliability, and not only the visual plausibility of the simulations, the use of SPH for flow simulations raise some serious concerns [Chaniotis et al., 2003]. We will revisit this key issue in Section 3.

Figure 13: Two-way coupled SPH and particle level set fluid simulation [Losasso et al., 2008].

SPH methods have been originally designed for compressible flows with an emphasis on gas dynamics of astrophysical systems (see Fig.12). In the case of incompressible flows, the need to define ad-hoc constitutive law, they have to resolve unphysical waves along with the related numerical stability constraints and the necessary artificial viscosity. In recent years a number of efforts [Hu et al., 2007, Ellero et al., 2007] have been presented in order to introduce incompressibility into SPH formulations with variable degrees of success. Note that in incompressible formulations a Poisson equation is necessary in order to ensure the incompressibility of the flow and efforts to bypass this relatively expensive computation have to maintain the right balance of accuracy and computational cost. Alternatively, vortex methods based on the vorticity form of the incompressible Navier-Stokes equations can be considered as more suitable numerical tools for these flows.

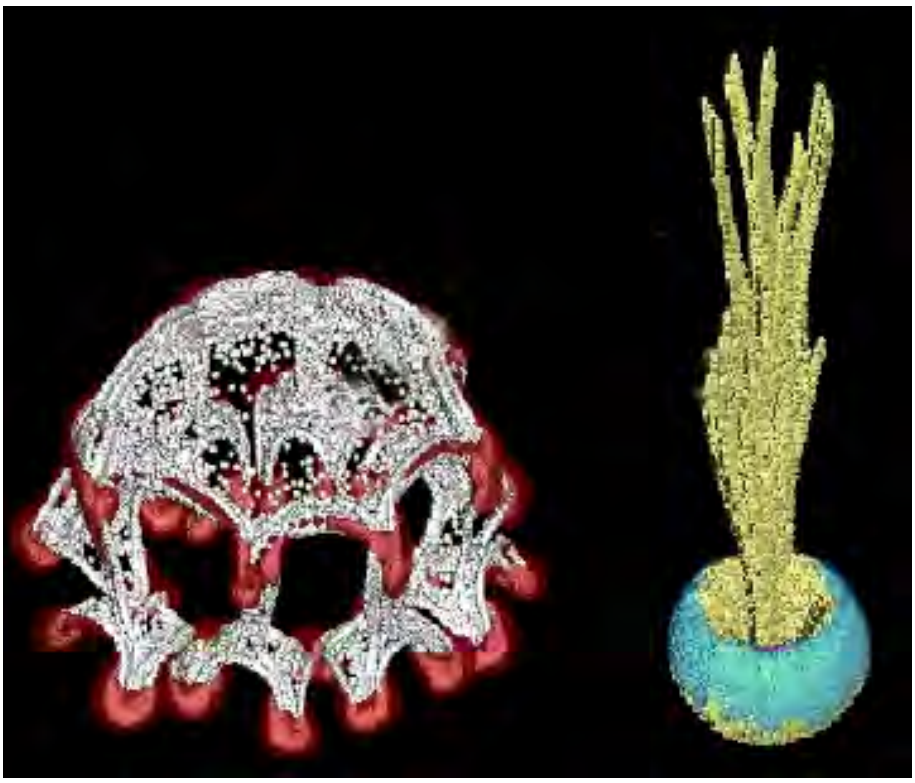


Figure 14: Vortex Method simulations of solid particle ladden flows with two way coupling [Walther and Koumoutsakos, 2001].

6.2.2 Incompressible Flows and Vortex Methods

In Vortex Methods particles discretize the velocity-vorticity formulation of the Navier-Stokes equations which takes the following form

$$\frac{\partial \omega}{\partial t} + (u \cdot \nabla)\,\omega \;=\; (\omega \cdot \nabla)\,u + \nu \Delta \omega \tag{6.16}$$

The vorticity field ω is related to the velocity field u by $\omega = \nabla \times u$. This equation together with the incompressiblity constraint

$$\text{div}\; u = 0 \tag{6.17}$$

and suitable boundary conditions allows to express the velocity in terms of the vorticity (we will come back later to this important point). Equation (6.16) thus appears as a nonlinear advection-diffusion equation for the vorticity. The diffusion term is just one particular instance of the right hand side F in (6.1). One particular and appealing feature of this form of the flow equations is that the divergence free constraint is not directly involved in the transport equation. Also, the equation points straight to the very nature of vorticity dynamics in incompressible flows: transport and dissipation (in 2 and 3D) , change of orientation and strengthening (in 3D only, since in 2D the term $(\omega \cdot \nabla)\,u$ vanishes). A vortex particle method consists of sampling vorticity on points

that follow the flow field:

$$\omega(x,t) = \sum_p \omega_p \delta(x - x_p) \tag{6.18}$$

with

$$\frac{dx_p}{dt} = u(x_p,t) \tag{6.19}$$

$$\frac{d\omega_p}{dt} = [\nabla u(x_p,t)]\,\omega_p + \nu\,\Delta\omega(x_p). \tag{6.20}$$

In the above equation $[\nabla u(x_p,t)]$ is the tensor made of all derivatives of the velocity. The way to compute this term as well as the diffusion term remains to be specified. Vortex Methods are distinct from SPH, in that they enforce explicitly the incompressibility of the flow while in addition they resolve gradients of the flow field rather than primitive variables. Furthermore they use computational elements only where the vorticity field is non-zero which at times can be only a small fraction of the domain, thus providing increased efficiency (Fig.14). At the same time they require the solution of a Poisson equation to recover the velocity field from the vorticity, while their implementation in the presence of boundaries requires the reformulation of the velocity boundary conditions. A monograph by the authors [Cottet and Koumoutsakos, 2000] discusses several of the methodological developments of Vortex Methods.
Here we wish to mention that another particle possible approach to 3D incompressible flows is possible using filaments instead of point particles. A filament method will consist of tracking markers on lagrangian curves that sample the vorticity field and reconstructing the corresponding curves at each time step, in order to recompute velocities and so on. Filaments are curves that carry one scalar quantity called the circulation. A filament can be viewed as a vortex tube, that is a space volume with vorticity is parallel to the walls, shrunk on its centerline. The circulation is the vorticity flux across the sections of the tube. Filaments are both very physically and numerically appealing for several reasons. Their Lagrangian character and the fact that their circulation remains constant when the flow is inviscid, translates Kelvin and Helmoltz theorems which are the two major facts in incompressible flows. From a numerical point of view, they allow to give with few degrees of freedom a rather detailed description of albeit complex dynamics. They have been investigated in some of the first ever 3D simulations in CFD [Leonard, 1980] and they have been successfully used in several graphics works(see for instance [Angelidis and Neyret, 2005]). However, several points detract them from being considered as a general tool to model and simulate incompressible flows. Except in specific (nonetheless interesting) situations, like rings and jets, they are not so easy to initialize for a given flow. Moreover, following a filament eventually requires at some point ad hoc decisions to avoid tiny loops or to decide reconnections between nearby filaments (something which is called filament surgery). Finally there is no clear cut way to simulate diffusion with filaments. For all these reasons we will only consider point particles in the rest of this class.

6.3 Grid-Free and Hybrid Particle Methods

The distinction between **grid-free** particle and hybrid **particle-grid** methods emerges when dealing with flow related equations besides the advection. These additional equations may be necessary in order to determine the right hand F in (6.1) or when the advection velocity is not given as a function of the advected quantity (as in the Biot-Savart law for the velocity-vorticity formulation).

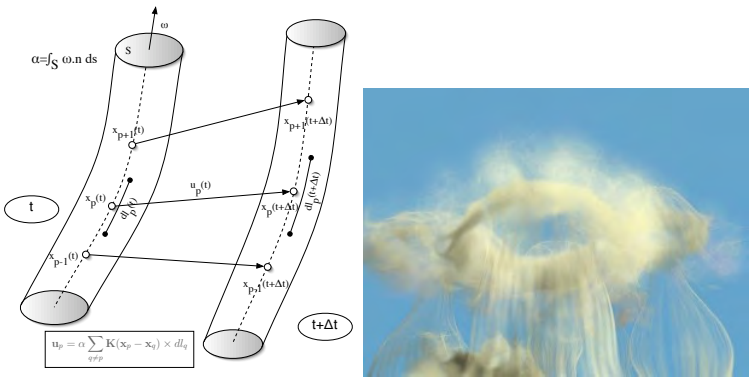


Figure 15: Top picture: vortex tube, filament and circulation. Evolution from time t to time $t+\Delta t$. the markers along the filament allow to reconstruct the curve and compute velocities for the next time step. The velocity formula is obtained from the Bio-Savart law (6.21).Bottom picture: example of image synthesis implementing filaments from [Angelidis and Neyret, 2005].

When we discuss *Grid Free particle methods* , we imply methods that rely solely on the particles to compute these quantities. By *Hybrid: Particle-Grid methods*, we imply methods which also use and underlying fixed grid.
Hybrid methods involve combinations of mesh based schemes and particle methods in an effort to combine computational advantages of each method. The first such method involves the Particle in Cell algorithm pioneered by Harlow [Harlow, 1964] in which a particle description replaces the non-linear advection terms and mesh based methods can be used to take advantage of the efficiency of Eulerian schemes to deal with elliptic or hyperbolic problems. In the following we give two examples of these methods one for each of the two classes of methods we have introduced.

6.3.1 SPH and Particle Mesh Hydrodynamics

First for SPH methods, we have underlined in (6.13), (6.14) a method which is grid-free. Instead of evaluating the term $\partial u/\partial x$ with the kernel W one might have chosen to use an underlying fixed grid and to increment particle density through the following successive steps

- *assign* velocity values u_i on the grid from the known particle quantities u_p
- evaluate by finite-differences on the grid derivatives of u on the grid
- interpolate back these quantities on the particles to obtain particle quantities div_p
- finally solve $d\rho_p/dt = -\rho_p\, div_p$

The same approach can be used to determine the stresses in the right hand side of the momentum equation. In case the stresses result form an energy carried by particle, there is an additional quantity carried by the particles, and both particles velocity and energy have to be assigned on the grid to compute the stresse which are next interpolated on the particles. The use of a mesh in the context of SPH helps accelerate the calculations and as we will see later it helps maintain the accuracy

of the method. This combination of grids and particles, that we baptized *PMH : Particle-Mesh Hydrodynamics* [**?**]has been shown to be highly effective in a number of flow systems that have been challenging for traditional SPH. The two phases of assignment and interpolation between grid and particles are crucial to ensure that the process is both accurate and does not introduce spurious oscillations. A lot of effort has been devoted in CFD to this issue. We will come back later when we discuss remeshing which somehow is currently the most effective way to approach this problem. Grid-free SPH have a symmetric issue for the choice of the kernel W and renormalization techniques to ensure conservation properties. Both methods crucially need to care about the number of particles per grid-size (for PIC method) or inside the range of the kernel W (for grid-free methods). It is important to realize at that point that in particle methods particles have a numerical meaning not as individual points but only through their collective contribution. This is a definite difference with finite-diffrence, finite element or finite volume methods.

6.3.2 Vortex Methods : Grid Free and Hybrid

For the second example, we consider vortex methods of the the inviscid ($\nu = 0$) Navier-Stokes equation (6.16). In that case, we only need to determine the velocity values necessary to push particles and to update particle vorticity values. The grid-free way to do it relies on the so-called Biot-Savart law.
The Biot-Savart law is an integral expression of the velocity in terms of the vorticity. Consider first the case of a non-bounded flow. A divergence-free velocity u with vorticity ω and vanishing at infinity is given by

$$u(x,t) = \int \mathbf{K}(x-y) \times \omega(y)\, dy \tag{6.21}$$

where the kernel $\mathbf{K}$ is given by the formula $\mathbf{K}(x) = \frac{1}{4\pi}\frac{x}{|x|^3}$. If the velocity has a given non zero value at infinity, this contribution has just to be added in the right hand side of (6.21).
The case of a flow with solid boundaries is more involved. In that case the boundary condition to be imposed on the velocity is in general a condition on the normal component of the velocity (a condition on the other component becomes necessary and physically relevant only for viscous flows). For the classical case of no-flow through a boundary Σ enclosing a fluid domain Ω, the theory of integral equations leads to the addition of a potential to the formula (6.21) :

$$u(x,t) = \int_{\Omega} \mathbf{K}(x-y) \times \omega(y)\, dy + \int_{\Sigma} \mathbf{K}(x-y) \times q(y)\, dy \tag{6.22}$$

where q is a potential to be determined through an integral equation such that the resulting velocity satisfies:

$$u(x,t) \cdot n(x) = 0 \text{for } x \text{on } \Sigma$$

The enforcement of the kinematic boundary conditions result in boundary integral equations that can be solved using boundary element methods [Hess, 1973] an approach that is widely used in engineering.
Let us now turn to the hybrid particle-grid counterpart of this method. As for the case of gas dynamics, one first needs to assign the quantity advected by particles - the vorticity values in this case - to grid nodes. Once it is done, one can reformulate the problem of finding the velocity in terms of the vorticity as a Poisson equation. Indeed, since div $u = 0$, one may write $u = \nabla \times \psi$

where $\boldsymbol{\psi}$ is a divergence-free stream function. Then one gets $\omega = \nabla \times u = -\Delta\boldsymbol{\psi}$. We are thus left with the following Poisson equation:

$$-\Delta\boldsymbol{\psi} = \omega. \tag{6.23}$$

This problem can be solved by off-the-shelf grid based Poisson solvers. To handle in a simple fashion boundary conditions of no-through flow type, it is in general advisable to use an additional scalar potential ϕ and look for u under the following form:

$$u = \nabla \times \boldsymbol{\psi} + \nabla\phi. \tag{6.24}$$

The stream vector $\boldsymbol{\psi}$ has to to be divergence-free and satisfy (6.23). The scalar potential ϕ does not contribute to vorticity. To give a divergence-free contribution it must satisfy

$$-\Delta\phi = 0 \tag{6.25}$$

in the computational domain. Its boundary condition is then adjusted to give no-though flow at the boundary Σ: from (6.24) it has to satisfy

$$\frac{\partial\phi}{\partial n} = -(\nabla \times \boldsymbol{\psi}) \cdot n.$$

This is classical Neumann-type boundary condition that complements (6.25). The advantage of this formulation is that it facilitates the calculation of the stream function. Without this potential, the stream function would have to satisfy boundary conditions coupling its components in order to be consistent with the divergence free condition.
To illustrate the method we present below a sketch of an hybrid particle-grid method using Williamson's low-storage third order Runge-Kutta scheme number 7 to integrate the equations of the particles. The scheme limits the numerical dissipation introduced into the flow, and it is memory efficient, requiring only one N additional storage per variable. The overall procedure is illustrated by Algorithm 9.

Algorithm 9 A Particle-in-Cell method using Williamson's Runge-Kutta scheme no.7.

Set up, initial conditions, *etc.* , $t = 0$ Particle quantities stored in arrays, *e.g.* vorticity: $\omega \in \mathcal{R}^{\ni \times \mathcal{N}}$. For the ODE solver we need two temporary variables: $u0$, and $d\omega 0$ While$t \leq T$ For $l = 1$ to 3 stages of the ODE Solver Interpolate ω onto the grid ($\omega \rightarrow \omega_{ijk}$) Compute velocity u_{ijk} from ω_{ijk} $u0 \leftarrow$ Interpolate u_{ijk} onto the particles $u0 \leftarrow u + \alpha_l\, u0$; $d\omega 0 \leftarrow d\omega + \alpha_l\, d\omega 0$ $\alpha = (0, -\frac{5}{9}, \frac{153}{128})$ $x \leftarrow x + \delta t\, \beta_l\, u0$; $\omega \leftarrow \omega + \delta t\, \beta_l\, d\omega 0$

We note that this hybrid formulation has enabled simulations using an unprecedented number of **10 billion particles** [**?**] of computational elements for the simulation of aircraft vortex wakes (see Fig:16)

6.3.3 Grid-Free vs. Hybrid - The winner is....

Let us now pause to compare the respective merits of the grid-free and particle-grid approaches. Clearly the grid-free approach is appealing in that it fully maintains the lagrangian nature of the

method. If short range interactions of particles are involved in the right hand side F one may devise particle interactions on physical basis. Particle methods can then be seen both as numerical methods and as discrete physical models. For incompressible flows the Biot-Savart law is required to compute non-local interactions. One is thus led to a N-body problem. If the vorticity is sampled on N particles, the simple minded calculation of the right hand side of (6.21) requires $O(N^2)$ operations, something which is not affordable for N beyond a few hundreds. To overcome this problem, a lot of effort has been devoted, following the pioneering work of Greengard and Rokhlin [**?**], to reduce this cost to something approaching $O(N)$. To summarize, the idea is to divide the particle distribution in clusters of nearby particles. The exact interaction of particles in one cluster with particles of another well separated cluster is replaced by an algebraic expansion using the moments of clusters of particles around their center. The number of terms only depends on the desired accuracy and never goes beyond a single digit number. Only interaction between particles in the same cluster are computed by direct summation. For maximal efficiency, the clustering of particles is done using a tree algorithm which creates boxes at different level of refinements containing always about the same number of particles. These fast summation formulas are now routinely used in CFD particle-based grid-free codes.

Unfortunately, practice shows that the construction of the tree, the evaluation of expansion coefficients and of the direct interaction of nearest particles, remain expansive, in particular in 3D. As a matter of fact the turnover point where the fast summation formulas become cheaper than the direct summation formulas is always beyond a few hundreds, which means that the constant in front of N or $N \log N$ in the evaluation of the complexity of the method is quite high. The cost of fast summation formulas is definitely much higher than that of FFT-based grid Poisson solvers. This is the reason why hybrid particle-grid methods can be seen as attractive alternative to grid-free methods. In an hybrid method, one first need to overlay a fixed grid to the particle distribution. This immediately rises the question of artificially closing a computational domain which is some cases should not. In grid-based method this is classical issue but it is somehow disturbing to introduce it in particle methods which in principle could avoid it. Also, in many instances vorticity is localized in a narrow part of the space, and the grid is certainly going to waste a number of points. All these observations point to the fact that hybrid grid-particle methods go in some sense against the very nature and advantages of particle methods, and thus should only be used if they come with a significantly lower computational cost. Clearly this will heavily depend on the ratio number of particles/ number of grid points, and thus on how much the vorticity support is concentrated and how far the artificial boundaries of the grid should be pushed if we deal with an external flow (a wake or a jet for instance). It also depends on the efficiency of the grid-based Poisson solver.

In many cases of practical interest, the result of this comparison, taking into account all these factors, is in favor of hybrid particle-grid methods. Figure 17 shows the computational times on a single processor machine for various 3D methods found in the literature. The computational time shown in this experiment corresponds to one evaluation of the velocities on all particles. The grid-free method has been used in an unbounded doamin, that is without the additional complexity of identifying the boundary potential, either with direct or fast summation formulas derived in [Krasny, 1986]. The hybrid particle-grid method has been used either in a square box, together with a FFT method, or in a cylindrical box, with a cyclic method to solve the linear system obtained from the discretization of the Poisson equations on the grid. In the first case the particles fill the whole box (like in an homogeneous turbulence experiment), while in the other case they fill only about 25% of the box. This last configuration is typical of what we would get for the simulation of

the wake behind a circular cylinder. Not surprisingly, the hybrid method gives the largest speed up (about a factor 100 compared to grid-free methods with fast summations) in the first configuration. But even in the second configuration, where one would think that many grid points are wasted, the speed up is already significative (about a factor 10).
Recently we have developed a Parallel Particle-Mesh (PPM) software library [Sbalzarini et al., 2006] that facilitates large-scale calculations of transport and related problems using particles. The library implements grid-particle methods. The library provides the mechanisms necessary to achieve good parallel efficiency and load balancing in these situations where both meshes and particles operate as computational elements. The PPM library scales to systems with up to 16,000 processors, with an efficiency of 80% and allows simulations using billions of computational elements [**?**]

6.3.4 Further Hybridization

The above two concepts of Grid-Free and Hybrid can be further combined and extended giving rise to a variety of numerical methods. In this context *Lagrangian-Eulerian* domain decomposition methods use high order grid methods and vortex methods in different parts of the domain [Cottet, 1990, Ould-Salihi et al., 2000] and can even be combined with different formulations of the governing equations. A finite difference scheme (along with a velocity-pressure formulation) can be implemented near solid boundaries, and vortex methods (in a velocity-vorticity formulation) can be implemented in the wake to provide the flow solver with accurate far-field conditions. In this approach Eulerian methods handle the wall boundary conditions and can be complemented with immersed boundary methods [Mittal and Iaccarino, 2005] to handle complex geometries. A rigorous framework for particle based immersed boundary methods has been developed based on a unified formulations of the equations for flow-structure interaction [Cottet, 2002]. Simulations involving this formulation are a subject of ongoing investigations.

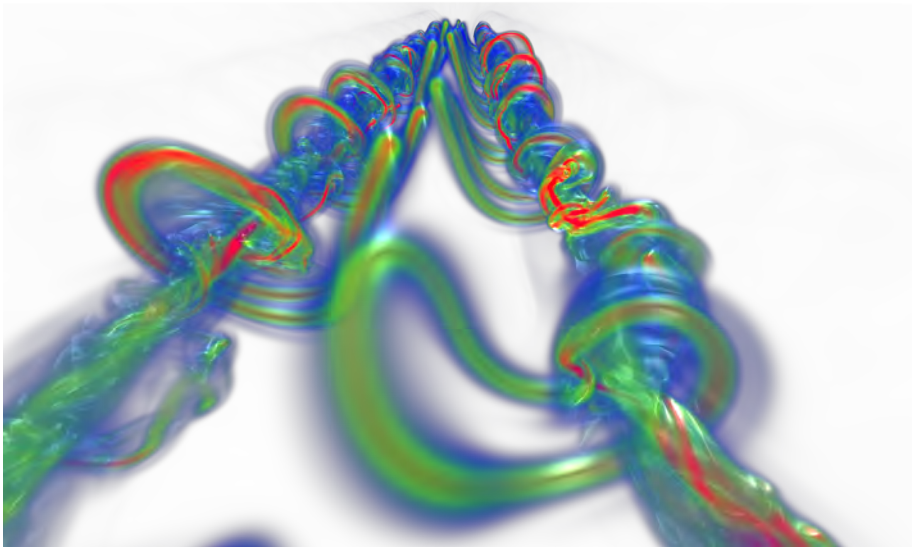


Figure 16: Billion Particle Simulations of Aircraft Wakes using Remeshed Vortex Methods [**?**]

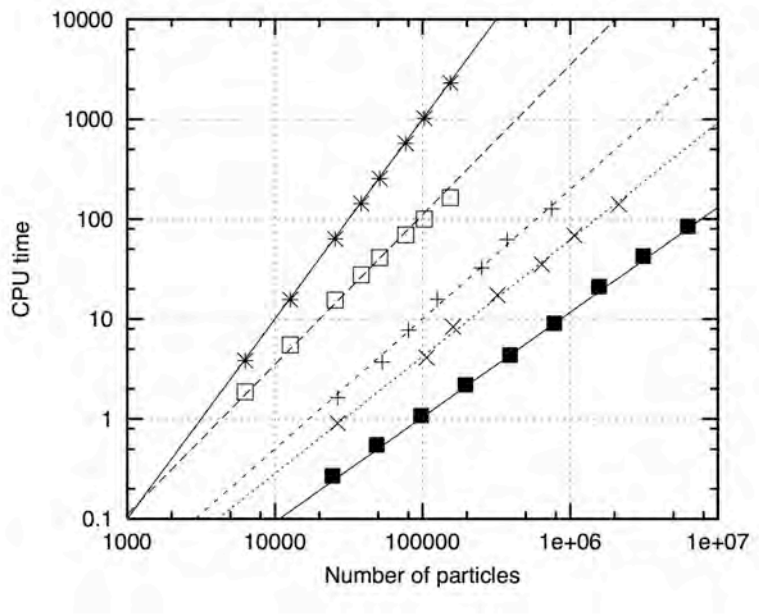


Figure 17: Implementation of a tree code/ Illustration of the clustering of particles in boxes obtained by successive subdivisions of the computational box.

7 Stochastics

These notes are partially based on 'A minimum of stochastics for scientists', by Noel Corngold

7.1 Motivation

A simple of a stochastic (*stochastic* = random, pertaining to conjecture) process is the trajectory of a tram. The timetable tells us when the tram will be at the current stop, and when at the next stop, etc. However, there is some inherent uncertainty in these times, since trams can be late or early dependent on the traffic conditions or external circumstances. To describe to someone, in our language we use terms such as 'the tram will be here around 10:30', which account for this uncertainty, and basically set up a probability distribution.
Consider a chemical reaction, for instance inside a biological cell, or inside the atmosphere. Let's say we have two species A and B, and they interact such that $A + B \to 2B$. At $t = 0$ we have 100 molecules of A and B each. After the first reaction, we have 99 molecules of A and 101 molecules of B, and so on. If we want to describe this process, we have two approaches. The first approach is to look at the concentration of molecules of A and B, which involves an averaging over all molecules, and solve a differential equation for the time rate of change of the concentration of each molecule. This approach corresponds to a coarse scale representation of the system. The second approach is to consider each reaction individually, and tracking the number of molecules of A and B over time. This approach is a fine-scale representation of the system, but we need a different approach to obtain the solution, for instance by solving the question: how often do a molecule of A and B collide? The second approach involves some stochasticity and is the one we will be focusing on.
The difference between the stochastic approach and the continuum approach can be illustrated in more detail with the very simple decay reaction:

$$A \xrightarrow{k1} \emptyset \tag{7.1}$$

Given N_A molecules of A, this reaction can be described by the ODE:

$$\frac{\mathrm{d}\langle N_A \rangle}{\mathrm{d}t} = -k_1 \langle N_A \rangle, \tag{7.2}$$

where $\langle N_A \rangle(t)$ denotes the average number of molecules (concentration) of A at time t. Describing the reaction with this ODE is the continuum approach. On a more detailed level, however, we should model the individual reactions. Given that the ith reaction happens at time t_i, we can describe the evolution as follows:

$$N_A \to N_{A-1} e^{t_1} \tag{7.3}$$
$$\to N_{A-2} e^{t_2} \tag{7.4}$$
$$\tag{7.5}$$

Describing the reaction with these processes is the basis for the stochastic approach. Of course, for large enough number of molecules, the two approaches should converge; for very few molecules the different approaches might well give different results.

7.2 Introductory definitions and concepts

First let us introduce the *frequentist approach* (Von Mises): The probability of some event is the limit of the relative frequency of the event in a series of observations as the series becomes infinity long. (*Kollektiv*).
There are however two problems associated with this approach. Firstly, we wish to define probabilities with a finite sequence of data, and secondly, we wish to deal with probabilities for time-dependent events. This means we have to look at probabilities and ensembles, which will be the topic of the rest of this subsection.

Ensemble and ensemble average An *ensemble* is defined as a collection of a large number n of equally prepared systems/models. A widely used ensemble is the *Microcanonical Ensemble*: a set of particles N with the same total energy E and in the same volume V (also called NVE-ensemble).
In a discrete setting, we assume a generic ensemble that contains a finite, discrete number of models n. Let Q be a quantity of interest, and let $Q(i)$ be the value of Q in model i, where $i = 0, \ldots, n$. For instance, you can think of Q as the outcome of a dice roll (and so it can only take integer values between 1 and 6), and of $Q(i)$ as the value of the ith dice roll. In this setting, we can define the *ensemble average* over n experiments as follows:

$$\langle Q \rangle = \sum_{i=1}^{n} \frac{Q(i)}{n}. \tag{7.6}$$

In a continuous setting, a different definition of the ensemble average exists. Let's assume the models are characterized by some continuous parameter, that we name $\omega \in \Omega$, where Ω is the state-space (containing all possible states of ω)[4]. Furthermore, let $\eta(\omega)$ be a weight function characterizing the ensemble. In the simplest case when all models are weighted equally we have just $\eta(\omega) = \omega$. Then our alternative definition for the ensemble average, based on this weight function, is as follows:

$$\langle Q \rangle = \frac{\int_\Omega Q(\omega) \mathrm{d}\eta(\omega)}{\int_\Omega \mathrm{d}\eta(\omega)} \tag{7.7}$$

Probabilities as ensemble averages Let A be a property which the ensemble members, have it or not[5]. Then we introduce $\chi(\omega)$ as:

$$\chi_A(\omega) = \begin{cases} 1 & \text{if } \omega \text{ member has } A \\ 0 & \text{otherwise.} \end{cases} \tag{7.8}$$

The probability of A in the ensemble is then

$$p(A) = \langle \chi_A(\omega) \rangle = \frac{\int_\Omega \chi_A(\omega) \mathrm{d}\eta(\omega)}{\int_\Omega \mathrm{d}\eta(\omega)}. \tag{7.9}$$

[4] Think for instance of a hypothetical situation where we do at every instant in time a dice roll, so that we can associate with every point in time an outcome for the dice roll. In this case, the value ω would be the time, $Q(\omega)$ would be the outcome of the dice roll at this time, and Ω would be the time interval over which we perform this experiment.
[5] In our continuous dice roll example, A could be for example the property of $Q(\omega) \leq 3$

So the probability is equal to the relative weight in the ensemble of those members that have property A. The equivalent definition in the discrete case is:

$$p(A) = \sum_{i=1}^{n} \frac{\chi_A(i)}{n} \tag{7.10}$$

Stochastic variables A stochastic variable is an object which is defined by a space of states (space of events, phase space) and by a probability density over this set. We will write capital letters for stochastic variables, and lower-case letter for their possible values. For example, we can have a stochastic variable X (such as the outcome of a dice roll) and its N possible outcomes $x_1, \ldots, x_N$ (for the dice, $N = 6$ and $x_i = i$). With each outcome we associate a probability $p(x_j) \geq 0$.
When the outcomes may be regarded as varying continuously over some interval, then the probability that the outcome lies in the interval $(x, x + \mathrm{d}x)$ is written as $p(x)\mathrm{d}x$. Here we introduced the *probability density* $p(x)$. Another way of writing this is:

$$p(x)\mathrm{d}x = \text{Prob}(X \in [x, x + \mathrm{d}x]). \tag{7.11}$$

For the probability of the outcome of X lying in the interval $[a, b]$, we integrate the probability density:

$$\int_a^b p(x)\mathrm{d}x = \text{Prob}(X \in [a, b]), \tag{7.12}$$

and so

$$\int_{-\infty}^{\infty} p(x)\mathrm{d}x = 1 \tag{7.13}$$

With stochastic variables defined like this, we can introduce a stochastic process: $Y = F(X, t)$, as a relationship between two stochastic variables.

Time dependency Our stochastic variable X can be a time-dependent variable: $X(t)$. This means the probability distribution also depends on time: $p(x, t)$. Consider for instance a process for which we carried out measurements at times $t_1, t_2, \ldots, t_N$. If we inquire about the probability of the variable being x_1 at t_1, x_2 at t_2, $\ldots$, and x_N at t_N, we need to construct a combined probability that all these events happen, which is formally written as:

$$p_N(x_1, t_1; x_2, t_2; \ldots; x_N, t_N).$$

The subscript N in p_N indicates that our probability depends on N points in state space. For example, consider a process as a train schedule with N stations. This is a N-point sampling of the train's trajectory (every station is a point), which can be represented by a function $x = F(x_0, t)$, where the trajectory $F(x_0, t)$ is selected by the initial value x_0, with probability density $P(x_0)$.
More formally, let there be a stochastic variable X assuming values x with probability distribution $P(x)$ (we use uppercase P to distinguish it from the combined probability distribution $p(x, t)$). Let us form $F(X, t) \equiv X(t)$, a function of X and t such that $F(X, 0) \equiv X \equiv X(0)$. The function assume values $F(x, t)$. Then we can write:

$$p_1(x, t) = \int \mathrm{d}x_0 \; P(x_0)\delta(x - F(x_0, t)) \tag{7.14}$$

This equation states that out of all possible initial conditions x_0 (each associated with its probability $P(x_0)$ we only take into account those that result in the process being at state x at time t: if $F(x_0, t) \neq x$ the integral evaluates to zero. The probability of arriving at x at time t is therefore only dependent on x_0 (with probability $P(x_0)$ and the corresponding trajectory function $F(x_0, t)$. For a process with N times, we can write the distribution for $X(t)$ as:

$$p_N(x_1, t_1; x_2, t_2; \ldots; x_N, t_N) = \int \mathrm{d}x_0 \; P(x_0)\delta(x_1 - F(x_0, t_1))\delta(x_2 - F(x_0, t_2)) \ldots \delta(x_N - F(x_0, t_N)). \tag{7.15}$$

We can use these statements to introduce two more definitions:

$$\langle X(t_2)X(t_1)\rangle_{P_N} = \int \mathrm{d}x_1 \ldots \int \mathrm{d}x_N p_N(x_1, t_1; x_2, t_2; \ldots; x_N, t_N) x_1 x_2 \tag{7.16}$$

$$= \int \mathrm{d}x_1 \int \mathrm{d}x_2 p_2(x_1, t_1; x_2, t_2) x_1 x_2 \tag{7.17}$$

$$= \int \mathrm{d}x_0 \; P(x_0) F(x_0, t_1) F(x_0, t_2). \tag{7.18}$$

$$p_2(x_1, t_1; x_2, t_2) = \int \mathrm{d}x_3 \int \mathrm{d}x_4 \ldots \int \mathrm{d}x_N \; p_N(\ldots). \tag{7.19}$$

Conditional probabilities A conditional probability deals with questions such as: given that x_i happened at time t_i, what is the probability that x_j happens at time t_j? We introduce the notation $p(x_a, t_a; x_b, t_b | x_c, t_c; x_d, t_d)$, is the probability of the process being in state x_a at time t_a *and* in state x_b at time t_b *given* that the process was in state x_c at time t_c *and* in state x_d at time t_d. Here, for simplicity, we restrict ourselves only to three events, and we can use the above notation to write:

$$p_3(x_1, t_1; x_2, t_2; x_3, t_3) = p(x_2, t_2; x_3, t_3 | x_1, t_1) p_1(x_1, t_1) \tag{7.20}$$

$$= p(x_3, t_3 | x_1, t_1; x_2, t_2) p_2(x_1, t_1; x_2, t_2). \tag{7.21}$$

Markov process A Markov process is a stochastic process that depends *only* on the previous state. It can be seen as a process without memory: the probability depends only on the previous state, and nothing before that. Formally, we make the following approximation by considering the process a Markov process:

$$p(x_N, t_N | x_1, t_1; x_2, t_2; \ldots; x_{N-1}, t_{N-1}) = p(x_N, t_N | x_{N-1}, t_{N-1}), \tag{7.22}$$

which asserts that after a long journey, the trajectory having passed through $1, 2, 3, \ldots$, the probability of x_N at t_N depends *only* upon the most recent pair (measurement) (x_{N-1}, t_{N-1}). Such as assumption implies a considerable loss of memory. Nevertheless, almost all treatments of stochastic processes in the sciences appear to be based upon this approximation, sometimes justified, sometimes questionable.

7.3 Chemical kinetics

We are working on a setting of chemical kinetics, where we have a well-stirred reaction volume V and N different species $S_1, S_2, \ldots, S_N$ in numbers $X_1, X_2, \ldots, X_N$. These species collide randomly and reactions take place through M channels $R_1, R_2, \ldots, R_M$. The time of the experiment is T.
Let's look at a single of these reactions:

$$R_1 : X_1 + X_2 \xrightarrow{k1} 2X_1 \tag{7.23}$$

We can express this reaction in two forms. First, in a macroscopic/continuous formulation we can write the evolution equation for the concentrations of species S_1 and S_2:

$$\frac{\mathrm{d}x_1}{\mathrm{d}t} = k_1 x_1 x_2 \tag{7.24}$$

$$\frac{\mathrm{d}x_2}{\mathrm{d}t} = -\frac{\mathrm{d}x_1}{\mathrm{d}t} = -k_1 x_1 x_2 \tag{7.25}$$

These are two coupled ODEs that describe the evolution of the concentrations as a limit process.
Second, we can consider this system in a discrete/stochastic setting. In this case, we look at the individual molecules and these numbers are actually stochastic variables with a probability distribution attached. We can simulate the evolution of individual instances of the system and take averages to numerically estimate the probability distributions. In these simulations, every time a reaction happens, we change the number of molecules in the system accordingly. A plot of the number of molecules of a given species versus time would be a 'stair-case' plot: every time a reaction happens, the number jumps. In contrast, the ODE solution always gives us a smooth curve for the concentrations of the number of molecules.

7.4 Jump processes and Master equation

Jump processes are processes with discrete changes of state. They arise in systems involving numbers of particles, or objects. In an appropriate limit macroscopic deterministic laws of motion arise, about which the random nature of the process generates fluctuations.
Let's look at a simple system, where we provide $\lambda(x)$ and we have a probability that the system is in state x and it jumps to the next state. The probability that the system jumps is $\lambda(x)\mathrm{d}t$. This means that the probability that the system does not jump is $1 - \lambda(x)\mathrm{d}t$ (a consequence of the jump process: either you jump or you don't). Furthermore, the probability that the system goes from x' to x during a time interval $\mathrm{d}t$ is $Q(x', x)\lambda(x)\mathrm{d}t$.
We can use this information to get the expectation for jump processes. This expectation is the solution of the Master equation, which describes how the probability of finding yourself in a certain state changes in time. A general formulation of the Master equation in continuous form is:

$$\frac{\partial P}{\partial t}(x, t) = \int \mathrm{d}x' [w(x, x')P(x', t) - w(x', x)P(x, t)],$$

where $w(x, x') = \lambda(x')Q(x, x')$ is the transition probability from x' to x. In the current context of discrete processes, the Master equation takes the form:

$$\frac{\partial P}{\partial t}(n, t) = \sum_{n'} [w(n, n')P(n', t) - w(n', n)P(n, t)].$$

The Master Equation is a conservation law for the probability to find the system in some state. The first term describes GAIN from other states. The second term describes LOSS to other states.

One-step jump processes A simple class of stochastic jump processes are the one-step jump processes. In the one step jump process, the range of states consists of all integers n and the matrix of the transition probabilities per unit time, allows only jumps between adjacent sites. In other words, the transition probability can be written as

$$w(n', n) = \begin{cases} \neq 0 & \text{for } n' = n \pm 1 \\ = 0 & \text{otherwise} \end{cases} \tag{7.26}$$

The Master equation then simplifies to

$$\frac{\mathrm{d}P}{\mathrm{d}t}(n,t) = w(n,n{+}1)P(n{+}1,t){+}w(n,n{-}1)P(n{-}1,t){-}w(n{+}1,n)P(n,t){-}w(n{-}1,n)P(n,t). \tag{7.27}$$

This equation describes all possible changes from the state n. The first two terms on the right hand side are gains from other states: respectively, the probability to change state from $n+1$ to n and the probability to change state from $n-1$ to n. The last two terms on the right hand side are losses to other states: respectively, the probability to change state from n to $n+1$ and the probability to change state from n to $n-1$.
We now introduce two new variables to simplify the notation. We define:

$$w(n, n+1) = r(n+1) \tag{7.28}$$
$$w(n, n-1) = g(n-1). \tag{7.29}$$

In other words, $r(n)$ is the transition probability at which we go from state n to state $n-1$ (death process), and $g(n)$ is the transition probability at which we go from state n to state $n+1$ (birth process). We also define the total transition rate $\lambda(n) = r(n) + g(n)$. With these new variables, the Master equation becomes:

$$\frac{\mathrm{d}P}{\mathrm{d}t}(n,t) = r(n+1)P(n+1,t) + g(n-1)P(n-1,t) - (r(n)+g(n))P(n,t). \tag{7.30}$$

Several examples of the simple one-step jump processes can be imagined. For instance, the process of radioactive decay can be describe by a death process: $g(n) = 0$ and $r(n) = \gamma n$, where γ is a rate of decay. We then get the Master equation:

$$\frac{\partial P}{\partial t}(n,t) = \gamma(n+1)P(n+1,t) - \gamma n P(n,t).$$

The direct solution of the Master equation may not be efficient to compute in high dimensions. Fortunately, the Master equation can be solved numerically by simulating the stochastic process. In this case, the process is represented by particles, and each particle jumps from one state to the next with given transition rates. This generates a single realization of the stochastic process. When a sufficiently great number of realizations has been generated, the quantities of interest can be evaluated as ensemble averages.

7.5 Stochastic Simulation Algorithm (SSA)

In order to simulate a stochastic system, we need to simulate each individual reaction (think of colliding molecules: each time two molecules collide, a reaction occurs). We therefore need to answer two questions: what is the time to the next reaction (or jump), and which reaction occurs?

Time to next jump We know that the probability that a jump occurs within infinitesimal δt is $\lambda(n)\delta t$ ($\lambda(n)$ is known). The probability that the next jump occurs after $N+1$ timesteps δt is therefore:

$$q = (1 - \lambda(n)\delta t)^N \lambda(n)\delta t$$

In other words, this is the probability that nothing happens for N timesteps: $(1-\lambda(n)\delta t)^N$, multiplied by the probability that something happens at the next timestep: $\lambda(n)\delta t$.
Now let $t = (N+1)\delta t$ and substitute for δt. Letting $\delta t \to 0$ and $N \to \infty$. we obtain the waiting time distribution:

$$q = \lambda \exp(-\lambda t)\delta t = f(t)\delta t.$$

Therefore, we can get the time to the next jump by sampling from this exponential distribution.
To sample the exponential distribution, we use the inversion method. This means first we sample a number ξ from a uniform distribution:

$$\xi \in U[0,1].$$

Inversing the exponential distribution then gives us an explicit expression based on ξ for the time until the next reaction τ:

$$\tau = -\frac{1}{\lambda(n)} \log \xi, \quad \xi \in U[0,1].$$

Which reaction occurs In the one-step process, we have only two possibilities for every reaction in state n: either the particle reaches state $n-1$ with probability $y_1 = r(n)/\lambda(n)$, or the particle reaches state $n+1$ with probability $y_2 = 1 - y_1 = g(n)/\lambda(n)$. We sample from these probabilities to find out which reaction occurs.
The SSA algorithm for this one-step process is then:

Step 1: Draw a uniformly distributed random number $\xi_1 \in U[0,1)$. Compute $\tau = -\frac{1}{\lambda(n)} \log(\xi_1)$.

Step 2: Draw a $\xi_2 \in U[0,1)$ and if

$$\xi_2 < y_2 = \begin{cases} \text{true:} & \nu = +1 \\ \text{false:} & \nu = -1 \end{cases} \tag{7.31}$$

Step 3: Advance the process: $t \to t + \tau$, $n \to n + \nu$.

Note that a number of realizations is necessary in order to extract statistics for the quantities of interest of this process.
The two main characteristics of SSA, answering the questions 'when does the next reaction occur' and 'what is the next reaction', can then be summarized by:

- For M reactions, the time until any reaction is $\tau \sim \mathcal{E}(1/a_0)$, where a_0 is the total propensity $a_0 = \sum_{j=1}^{M} a_j$.
- The reaction index comes out of a point-wise distribution $p(j = l) = a_l/a_0$.

7.6 Accelerated Stochastic Simulations

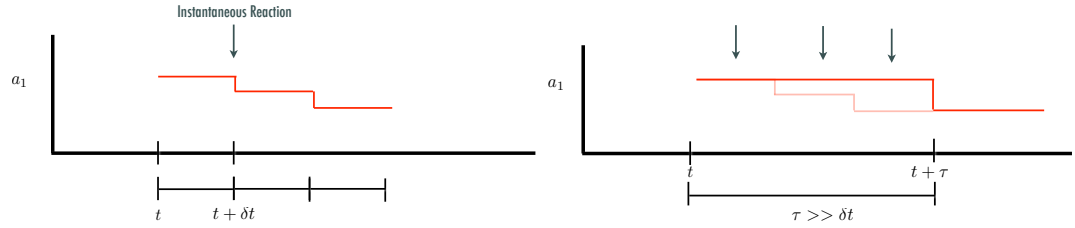
Since SSA relies on simulating every single reaction event, it is exact, but at the same time it can be very slow for system with large numbers of molecules. Several methods have been proposed to speed up SSA simulations. Two of these accelerated algorithms are discussed here: τ-leaping and R-leaping.

τ-leaping The main assumption behind τ-leaping is that the reaction propensities a_i remain essentially constant over τ, in spite of several firings. This assumption allows to process several reaction events over a single timestep. Over this timestep τ, the number of reaction firings K_j is governed by a Poisson distribution:

$$K_j \sim \mathcal{P}(a_j \tau) \tag{7.32}$$

$$\mathbf{X}(t+\tau) = \mathbf{X}(t) + \sum_{j=1}^{M} K_j \nu_j \tag{7.33}$$

The cost of this method is M Poisson samplings per timestep. The picture below shows the evolution for a decay reaction $X \to \emptyset$. On the left is pictured how an SSA simulation would evolve X: one reaction at the time. On the right is the evolution according to a τ-leaping simulation: we advance the system over a time τ, and then fire many reactions at the same time.



The parameter τ can be used to control accuracy and speed-up in τ-leaping simulations.
The τ-leaping algorithm has been successful in speeding up stochastic simulations, but it has some inherent problems attached as well. The most obvious of these problems is that for systems with small numbers of molecules, τ-leaping can generate negative populations by firing many reactions at the same time. Solutions to this problem exist (binomial τ-leaping, modified τ-leaping) that rely on bounds on the propensity changes in order to control number of reactions being fired in case of large changes in propensities.

R-leaping In R-leaping, the thinking behind τ-leaping is inversed: leaps are made in a prescribed number of reaction firings L across all reaction channels. The time-increment corresponding to L reaction firings is given by a Γ-distribution and can be computed: $\tau_L \sim \Gamma(L, 1/a_0)$. After this time interval τ_L we will have K_m firings of reaction channel R_m, such that $\sum_{m=1}^{M} K_m = L$. One step in the R-leaping algorithm, given L, can be described as follows:

- Compute the time corresponding to L reaction firings: $\tau_L \sim \Gamma(L, 1/a_0)$
- Sample the index j of the reaction to be fired: $P(j = l) = a_l/a_0$ for $l = 1, \ldots, L$
- Compute the number of reactions for channel m: $K_m = \sum_{l=1}^{L} \delta_{l,m}$

- Update the species and the time: $\mathbf{X}(t+\tau_L) = \mathbf{X}(t) + \sum_{j=1}^{M} K_j \nu_{\mathbf{j}}$.

In contrast to τ-leaping, the reaction channels to be fired K_j are not independent.
Implementing the R-leaping algorithm requires to sample the M values of K_j. In the first iteration of the algorithm (the R_0 algorithm) a pointwise sampling of L independent reaction indices is employed $P(j = l) = a_l/a_0$. This is a simple approach, but it scales with L, and therefore the computational workload in R-leaping will be close to the workload in SSA. However, we can employ a theorem to speed-up the R-leaping algorithm. According to this theory, the distribution of K_1 (the number of firings of reaction 1) is a binomial distribution: $K_1 \sim \mathcal{B}(L, a_1/a_0)$, and for every $m \in 2, \ldots, M$ the conditional distribution of K_m given the events $(K_1, \ldots, K_{m-1}) = (k_1, \ldots, k_M)$ is

$$K_m \sim \mathcal{B}\left(L - \sum_{i=1}^{m-1} k_i, \frac{a_m}{a_0 - \sum_{i=1}^{m-1} a_i}\right). \tag{7.34}$$

Furthermore, this result is invariant under any permutation of the indices. The proof of this theorem can be found in the slides on the course website.
We can use this theorem to create the R_1 algorithm: instead of sampling L independent reaction indices, we now sample M correlated binomial variables. A final speed-up can be gained by realizing that once $\sum_{i=1}^{m-1} = L$, the sampling is done since we have reached the number of reactions we wanted to fire. We can minimize the average m (the number of samplings we need) by permuting the indices, such that a'_j is decreasing: $a_M > a_3 > a_1 \ldots$. In other words, we process first the reactions with the biggest propensities. This will minimize the number of samplings m, since the reactions with the largest propensities will on average fire the most.

8 MATHEMATICAL TOOLS

8.1 Fourier Transforms - A reminder

Fourier Transform (FT):

$$\hat{f}(k) = \int_{-\infty}^{\infty} f(x)e^{-ikx}dx$$

Inverse Fourier Transform (IFT):

$$f(k) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{f}(k)e^{+ikx}dx$$

Consider the differential equation of the form

$$-\frac{d^2u}{dx^2} + au = h$$

in $-\infty < x < \infty$.

Taking its Fourier Transform we have that:

$$-\frac{\hat{d}^2u}{dx^2} + a\hat{u} = \hat{h} \sim -(ik)^2\hat{u}(k) + a\hat{u}(k) = \hat{h}(k)$$

$$\rightarrow \hat{u}(k) = \frac{\hat{h}(k)}{a^2 + k^2}$$

Examples

Example 1:

$$f(x) = \delta(x) \rightarrow \hat{f}(k) = \int_{-\infty}^{\infty} \delta(x)e^{-ikx}dx = 1$$

Example 2:

$$f(x) = \begin{cases} -1, & \text{if } -L \leq x \leq L \\ 0, & \text{otherwise} \end{cases}$$

$$\rightarrow \hat{f}(k) = \int_{-L}^{L} f(x)e^{-ikx}dx = \int_{-L}^{L} 1e^{-ikx}dx = \frac{-1}{ik}e^{-ikx} \mid_{-L}^{+L} = \frac{-2sin(kL)}{k}$$

Example 3 (decaying "impulse"):

$$f(x) = e^{-ax}, x >= 0$$

$$\rightarrow \hat{f}(k) = \int_{0}^{\infty} \delta(x)e^{-ax}e^{-ikx}dx = \frac{e^{-(a+ik)}}{-a-ik} \mid_{0}^{\infty} = \frac{1}{a+ik}$$

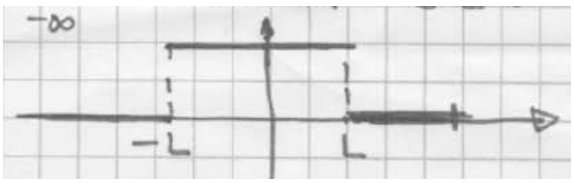


Figure 18: Function of example 2

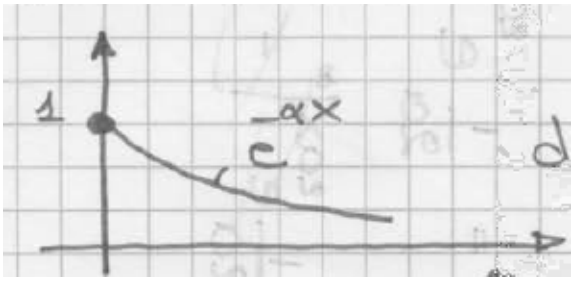


Figure 19: Function of example 3

Example 4:

$$f(x) = e^{-a|x|}$$

$$\rightarrow \hat{f}(k) = \frac{1}{a+ik} + \frac{1}{a-ik} = \frac{2a}{a^2+k^2}$$

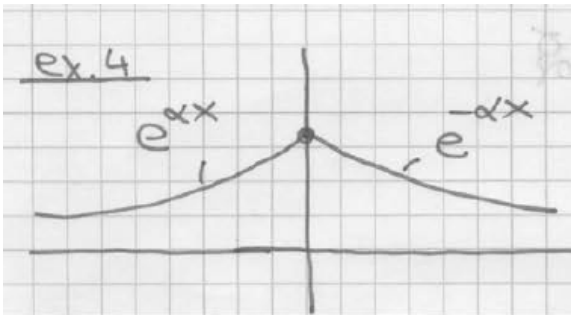


Figure 20: Function of example 4

Example 5:

$$f(x) = \begin{cases} -e^{ax}, & \text{if } x \leq 0 \\ e^{ax}, & \text{if } x \geq 0 \end{cases}$$

$$\rightarrow \hat{f}(k) = \frac{1}{a+ik} - \frac{1}{a-ik} = \frac{-2ik}{a^2+k^2}$$

Plancherel's Formula

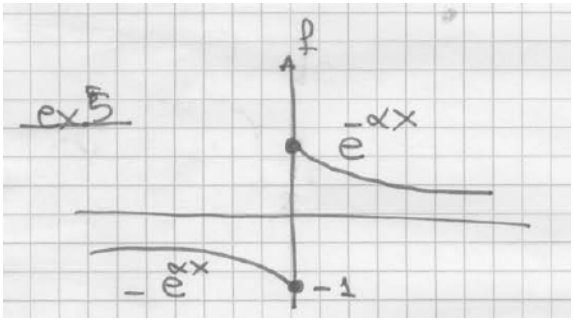


Figure 21: Function of example 5

$$2\pi \int_{-\infty}^{\infty} |f(x)|^2 dx = \int_{-\infty}^{\infty} |\hat{f}(k)|^2 dk$$

where,

$\int_{-\infty}^{\infty} |f(x)|^2 dx$: norm of f in L^2
$\int_{-\infty}^{\infty} |\hat{f}(k)|^2 dk$: "norm" of $\hat{f}$ in $\hat{L}^2$

More generally:

$$2\pi(f, g) = (\hat{f}, \hat{g})$$

the inner product before and after the transform is the same

8.2 Green's functions and the Solution of Boundary Value Problems

Consider a 2nd order ODE:

$$-\frac{d^2u}{dx^2} + a^2u = h(x)$$

This equation can be solved with the help of Fourier Transforms as:

$$-(-k^2\,\hat{u}(k)) \;+\; a^2\hat{u}(k) \;=\; \hat{h}(k)$$

so that its solution in the Fourier space is readily obtained as:

$$\hat{u}(k) = \frac{\hat{h}(k)}{k^2 + a^2}$$

The solution in the Fourier space contains a component ($\hat{h}(k)$) that depends on the right hand side as well as a component ($\frac{1}{k^2+a^2}$) that reflects the structure of the ODE.
We can develop a solution for any right hand side by first obtaining the solution of the ODE when the right hand side is a δ-function. In this case we can write:

$$h(x) = \delta(x) \rightarrow \hat{h}(k) = 1$$

So for this special right hand side the solution of the ODE is: and

$$\hat{u}(k) = \frac{1}{k^2 + a^2}$$

This is the Fourier Transform of the decaying pulse after dividing by $2a$.
So we write for this special case the solution to be

$$G(x) = \frac{1}{2a} e^{-a|x|}$$

.
Now, having derived G we can solve the ODE for ANY right hand side h as we can write

$$\hat{u}(k) = \hat{G}(k)\hat{h}(k)$$

The inverse Fourier Transform of a product is given as a convolution so that we write:

$$u(x) = \int_{-\infty}^{\infty} G(x-y)h(y)dy = G * h$$

where the $*$ is used to denote convolution.
The use of Green's functions for the solution of ODEs can be extended to Partial Differential Equations (PDEs). So in order to solve Poisson's equation

$$-\nabla^2 \Psi(r) = \rho(r) \tag{8.1}$$

we first consider its Green's function solution.

$$\nabla^2 G(r|r') = \delta(r - r')$$

along with suitable Boundary Conditions (BC's) for this problem.
We write for

$$\Psi(r) = \int dV' G(r|r')\rho(r') = f(r)$$

Substituting into Eq. 8.1 we get

$$-\nabla^2 f(r) = \int dV'[-\nabla^2 G(r|r')]\rho(r') = \int dV' \delta(r - r')\rho(r') = \rho(r)$$

So this satisfies the equations. We first consider unbounded domains. We distinguish 2D and 3D domains and denote respectively $G^2(r|r')$ and $G^3(r|r')$.

Define $R = r - r'$.
For a source point at the origin $r' = 0$ then $R = r$ is the same as the position vector in the field.
We require that the solution is going to zero at infinity.
Solution is

$$G_0^{(3)}(r|r') = \frac{1}{4\pi R}. \tag{8.2}$$

This yields that

$$\Psi(r) = \int \frac{dV'\rho(r')}{4\pi(r - r')} \tag{8.3}$$

which asserts that the potential at r is the linear combination of the potentials $\frac{dV'\rho(r')}{4\pi(r-r')}$ due to all the source elements $dV'\rho(r')$ at r'.
Let $r' = 0$ and $r = R$.

Theorem 1.

$$-\nabla^2 G_0^{(3)}(r|0) = \delta(r) \tag{8.4}$$

$$G_0^{(3)}(r \to \infty) = 0 \tag{8.5}$$

$$\implies G_0^{(3)}(r|0) = \frac{1}{4\pi r} \tag{8.6}$$

Proof. Adopt spherical polar coordinates and look for a spherically symmetric solution.
Acting on functions of r alone ∇^2 reduces to:

$$\nabla_r^2 = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 \frac{\partial}{\partial r}\right). \tag{8.7}$$

So for $r \neq 0$

$$\nabla^2 G_0^{(3)} = \nabla_r^2 G_0 = 0. \tag{8.8}$$

The solution $\frac{1}{r}$ satisfies this because:

$$-\nabla_r^2 \frac{1}{r} = -\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 \frac{\partial}{\partial r}\frac{1}{r}\right) \tag{8.9}$$

$$= -\frac{1}{r^2}\frac{\partial}{\partial r}\left[r^2\left(-\frac{1}{r^2}\right)\right] \tag{8.10}$$

$$= -\frac{1}{r^2}\frac{\partial}{\partial r}(-1) = 0 \tag{8.11}$$

To verify that Theorem 1 satisfies also the Theorem 1 at $r = 0$ we integrate both sides of Equation 8.4 with respect to volume over a sphere with an arbitrarily small radius ε.
We have

$$\int_W dV\delta(r) = 1 \tag{8.12}$$

On the left:

$$-\int_W dV\nabla^2 G_0 = -\int dV\nabla\cdot(\nabla G_0) \tag{8.13}$$

$$= -\int_{r=\varepsilon} dS\cdot\nabla G_0 \tag{8.14}$$

$$= -\int dS\cdot\frac{-\hat{r}}{4\pi\varepsilon^2} \tag{8.15}$$

$$= \int\frac{\varepsilon^2 d\Omega}{4\pi\varepsilon^2} = 1 \tag{8.16}$$

If $F(r)$ is any function behaving like $\frac{C}{r}$ where $r\to 0$ then $-\nabla^2 F$ contains a component $4\pi C\delta(r)$. For instance, at points $r\neq 0$ the function

$$F(r) = \frac{e^{\mu r}}{r} \tag{8.17}$$

satisfies the homogeneous Helmholtz equation:

$$-(\nabla^2+\mu)F = 0, \tag{8.18}$$

but if V includes $r=0$, then the true equation eyed by F is

$$-(\nabla^2+\mu)F = 4\pi\delta(r) \tag{8.19}$$

In 2D

$$G_0^{(2)}(r|r') = \frac{1}{2\pi}log\left(\frac{a}{R}\right) \tag{8.20}$$

where a are arbitrary constants.

In 1D

$$-\frac{\partial^2 G^{(1)}}{\partial x^2}(x|x') = \delta(x-x') \tag{8.21}$$

$$G_0^{(1)}(x|x') = -\frac{1}{2}|x-x'| + A(x')x + B(x') \tag{8.22}$$

Boundary conditions $\rightarrow$ make G_0 (symmetric) as in 2D, 3D, thus enforce

$$A(x') = C, \tag{8.23}$$

$$B(x') = Cx', \tag{8.24}$$

where C is a constant independent of x and x'.

$$G_0^{(1)}(x|x') = -\frac{1}{2}|x-x'| + C(x+x') \tag{8.25}$$

Boundary conditions 2: we make $G(x\to\infty|x') = G(x\to-\infty|x') \implies C=0$ and

$$G_0^{(1)}(x|x') = -\frac{1}{2}|x-x'| \tag{8.26}$$

□

We return now to the vorticity field

$$\omega = \nabla \times \mathbf{u}. \tag{8.27}$$

We specify the velocity field in terms of a stream function

$$\mathbf{u} = \nabla \times \psi \tag{8.28}$$

such that

$$u = \frac{\partial \psi}{\partial y} \tag{8.29}$$

$$v = \frac{\partial \psi}{\partial x} \tag{8.30}$$

where $\psi = (0, 0, \psi)$ and $\mathbf{u} = (u, v, 0)$.
Note that using the stream function gives:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = \frac{\partial^2 \psi}{\partial x \partial y} - \frac{\partial^2 \psi}{\partial x \partial y} = 0. \tag{8.31}$$

So it satisfies trivially the continuity equation.
Alternatively we write

$$\omega = \nabla \times \mathbf{u} = \nabla \times \nabla \times \psi = -\nabla^2 \psi + \underbrace{\nabla \cdot \overbrace{(\nabla \times \psi)}^{\mathbf{u}}}_{=0} \tag{8.32}$$

from which we obtain

$$\omega = -\nabla^2 \psi. \tag{8.33}$$

References

[Angelidis and Neyret, 2005] Angelidis, A. and Neyret, F. (2005). Simulation of smoke based on vortex filament primitives. In *ACM-SIGGRAPH/EG Symposium on Computer Animation (SCA)*.

[Appel, 1985] Appel, A. (1985). An efficient program for many-body simulation. *SIAM Journal on Scientific and Statistical Computing*, 6(85).

[Barnes and Hut, 1986] Barnes, J. and Hut, P. (1986). A hierarchical $\mathcal{O}$ N $\log$ N force-calculation algorithm. *Nature*, 324:446–449.

[Beale, 1986] Beale, J. T. (1986). A convergent 3-D vortex method with grid-free stretching. *Math. Comput.*, 46:401–424.

[Beale and Majda, 1982] Beale, J. T. and Majda, A. (1982). Vortex methods. II: Higher order accuracy in two and three dimensions. *Math. Comput.*, 39(159):29–52.

[Bergdorf and Koumoutsakos, 2006] Bergdorf, M. and Koumoutsakos, P. (2006). A Lagrangian particle-wavelet method. *Multiscale Model. Simul.*, 5(3):980–995.

[Briggs et al., 2000] Briggs, W. L., Emden Henson, V., and McCormick, S. F. (2000). *A Multigrid tutorial*. SIAM, 2nd edition.

[CARLSON et al., 2004] CARLSON, M., MUCHA, P., and TURK, G. (2004). Rigid fluid: Animating the interplay between rigid bodies and fluid.

[Chaniotis et al., 2003] Chaniotis, A. K., Frouzakis, C. E., Lee, J. C., Tomboulides, A. G., Poulikakos, D., and Boulouchos, K. (2003). Remeshed smoothed particle hydrodynamics for the simulation of laminar chemically reactive flows. *J. Comput. Phys.*, 191(1):1–17.

[Chorin, 1973] Chorin, A. J. (1973). Numerical study of slightly viscous flow. *J. Fluid Mech.*, 57(4):785–796.

[Cottet, 2002] Cottet, G. (2002). A particle model for fluid-structure interaction. *C.R. Acad. Sci. Paris*, Ser. I(335):833–838.

[Cottet and Poncet, 2004] Cottet, G. and Poncet, P. (2004). Advances in direct numerical simulations of 3d wall-bounded flows by vortex-in-cell methods. *Journal of Computational Physics*, 193(1):136–158.

[Cottet, 1990] Cottet, G. H. (1990). A particle-grid superposition method for the Navier-Stokes equations. *J. Comput. Phys.*, 89:301–318.

[Cottet and Koumoutsakos, 2000] Cottet, G.-H. and Koumoutsakos, P. (2000). *Vortex Methods – Theory and Practice*. Cambridge University Press, New York.

[Degond and Mas-Gallic, 1989] Degond, P. and Mas-Gallic, S. (1989). The weighted particle method for convection-diffusion equations. part 2: The anisotropic case. *Mathematics of Computation*, 53(188):509–525.

[Elcott et al., 2007] Elcott, S., Tong, Y. Y., Kanso, E., Schroder, P., and Desbrun, M. (2007). Stable, circulation-preserving, simplicial fluids. *ACM TRANSACTIONS ON GRAPHICS*, 26(1):4.

[Eldredge et al., 2002] Eldredge, J. D., Leonard, A., and Colonius, T. (2002). A general deterministic treatment of derivatives in particle methods. *J. Comput. Phys.*, 180(2):686–709.

[Ellero et al., 2007] Ellero, M., Serrano, M., and Espanol, P. (2007). Incompressible smoothed particle hydrodynamics. *J. Comput. Phys.*, 226(2):1731–1752.

[Fedkiw et al., 2001] Fedkiw, R., Stam, J., and Jensen, H. W. (2001). Visual simulation of smoke. In Fiume, E., editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 15–22. ACM Press / ACM SIGGRAPH.

[Foster and Metaxas, 1996] Foster, N. and Metaxas, D. (1996). Realistic animation of liquids. *Graphical models and image processing: GMIP*, 58(5):471–483.

[Gingold and Monaghan, 1977] Gingold, R. A. and Monaghan, J. J. (1977). Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Month Notices Roy. Astron. Soc.*, 181:375–389.

[Greengard and Rokhlin, 1987] Greengard, L. and Rokhlin, V. (1987). A fast algorithm for particle simulations. *Journal of Computational Physics*, 73(325–348).

[Harlow, 1964] Harlow, F. H. (1964). Particle-in-cell computing method for fluid dynamics. *Methods Comput. Phys.*, 3:319–343.

[HERNQUIST, 1993] HERNQUIST, L. (1993). Some cautionary remarks about smoothed particle hydrodynamics. *ASTROPhys. JOURNAL*, 404(2):717–722.

[Hess, 1973] Hess, J. L. (1973). Higher order numerical solution of the integral equation for the two-dimensional Neumann problem. *Comp. Meth. Appl. Mech. & Engng.*, 2:1–15.

[Hieber and Koumoutsakos, 2005] Hieber, S. E. and Koumoutsakos, P. (2005). A Lagrangian particle level set method. *J. Comput. Phys.*, 210:342–367.

[Hockney and Eastwood, 1981] Hockney, R. and Eastwood, J. (1981). *Computer simulation using particles*. McGraw-Hill Inc.

[Hockney and Eastwood, 1988] Hockney, R. W. and Eastwood, J. W. (1988). *Computer Simulation Using Particles*. Institute of Physics Publishing, Bristol, PA, USA, 2 edition.

[Hu et al., 2007] Hu, W., Tong, B., and Liu, H. (2007). A numerical study on mechanism of s-starts of northern pike (esox lucius). *Journal of Hydrodynamics, Ser. B*, 19(2):135 – 142.

[Katzenelson, 1989] Katzenelson, J. (1989). Computation structure of the n-body problem. *SIAM Journal on Scientific and Statistical Computing*, 11:787–815.

[Kawaguchi, 1982] Kawaguchi, Y. (1982). A morphological study of the form of nature. *SIGGRAPH Comput. Graph.*, 16(3):223–232.

[Kerr, 2002] Kerr, R. A. (2002). Planetary origins: A quickie birth for jupiters and saturns. *Science*, 298(5599):1698b–1699.

[Koplik and Banavar, 1995] Koplik, J. and Banavar, J. R. (1995). Corner flow in the sliding plate problem. *Phys. Fluids*, 7(12):3118–3125.

[Koumoutsakos, 1993] Koumoutsakos, P. (1993). *Direct numerical simulations of unsteady separated flows using vortex methods*. PhD thesis, Caltech.

[Koumoutsakos, 1997] Koumoutsakos, P. (1997). Inviscid axisymmetrization of an elliptical vortex ring. *J. Comput. Phys.*, 138:821–857.

[Koumoutsakos, 2005] Koumoutsakos, P. (2005). Multiscale flow simulations using particles. *Annu. Rev. Fluid Mech.*, 37:457–487.

[Krasny, 1986] Krasny, R. (1986). A study of singularity formation in a vortex sheet by the point vortex approximation. *JFM*, 167:65–93.

[Leonard, 1980] Leonard, A. (1980). Review. vortex methods for flow simulation. *J. Comput. Phys.*, 37:289–335.

[Leonard, 1985] Leonard, A. (1985). Computing three-dimensional incompressible flows with vortex elements. *Annu. Rev. Fluid Mech.*, 17:523–559.

[Liu et al., 1995] Liu, W., Jun, S., and Zhang, S. (1995). Reproducing kernel particle methods. *Int. J. for Numer. Methods In Engng.*, 20(8–9):1081–1106.

[Losasso et al., 2008] Losasso, F., Talton, J. O., Kwatra, N., and Fedkiw, R. (2008). Two-way coupled sph and particle level set fluid simulation. *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS*, 14(4):797–804.

[Lucy, 1977] Lucy, L. B. (1977). A numerical approach to the testing of the fission hypothesis. *Astron. J.*, 82:1013–1024.

[Minion and Brown, 1997] Minion, M. L. and Brown, D. L. (1997). Performance of underresolved two-dimensional incompressible flow simulations, II. *J. Comput. Phys.*, 138:734–765.

[Mittal and Iaccarino, 2005] Mittal, R. and Iaccarino, G. (2005). Immersed boundary methods for viscous flow. *Annu. Rev. Fluid Mech.*, 37:to appear.

[Monaghan, 1985a] Monaghan, J. J. (1985a). Extrapolating B splines for interpolation. *J. Comput. Phys.*, 60(2):253–262.

[Monaghan, 1985b] Monaghan, J. J. (1985b). Particle methods for hydrodynamics. *Comput. Phys. Rep.*, 3:71–123.

[Monaghan, 1992] Monaghan, J. J. (1992). Smoothed particle hydrodynamics. *Annu. Rev. Astron. Astrophys.*, 30:543–574.

[Monaghan, 2005] Monaghan, J. J. (2005). Smoothed particle hydrodynamics. *J. Comput. Phys.*, 68(8):1703–1759.

[Ould-Salihi et al., 2000] Ould-Salihi, M. L., Cottet, G.-H., and El Hamraoui, M. (2000). Blending finite-difference and vortex methods for incompressible flow computations. *SIAM J. Sci. Comput.*, 22(5):1655–1674.

[Pépin and Leonard, 1990] Pépin, F. and Leonard, A. (1990). Concurrent implementation of a fast vortex method. *5th Distributed Memory Computing Conference*, I:453–462.

[Pfister and Gross, 2004] Pfister, H. and Gross, M. (2004). Point-based computer graphics. *IEEE COMPUTER GRAPHICS AND APPLICATIONS*, 24(4):22–23.

[Ploumhans et al., 2002] Ploumhans, P., Winckelmans, G. S., Salmon, J. K., Leonard, A., and Warren, M. S. (2002). Vortex methods for direct numerical simulation of three-dimensional bluff body flows: Applications to the sphere at $Re = 300$, 500 and 1000. *J. Comput. Phys.*, 178:427–463.

[Premoze et al., 2003] Premoze, S., Tasdizen, T., Bigler, J., Lefohn, A., and Whitaker, R. T. (2003). Particle-based simulation of fluids. *COMPUTER GRAPHICS FORUM*, 22(3):401–410.

[Reeves, 1983] Reeves, W. T. (1983). Particle systems — a technique for modeling a class of fuzzy objects. *ACM T. Graphic*, 2(2):91–108.

[Rosenhead, 1930] Rosenhead, L. (1930). The spread of vorticity in the wake behind a cylinder. *Proc. R. Soc. Lond. A*, 127(A):590–612.

[Salmon, 1991] Salmon, J. (1991). *Parallel Hierarchical N-Body Methods*. PhD thesis, Caltech.

[Sbalzarini et al., 2006] Sbalzarini, I. F., Walther, J. H., Polasek, B., Chatelain, P., Bergdorf, M., Hieber, S. E., Kotsalis, E. M., and Koumoutsakos, P. (2006). A software framework for portable parallelization of particle-mesh simulations. *Lect. Notes Comput. Sc.*, 4128:730–739.

[Schoenberg, 1946] Schoenberg, I. J. (1946). Contribution to the problem of approximation of equidistant data by analytic functions. *Quart. Appl. Math.*, 4:45–99, 112–141.

[Selle et al., 2005] Selle, A., Rasmussen, N., and Fedkiw, R. (2005). A vortex particle method for smoke, water and explosions. *ACM Trans. Graph.*, 24(3):910–914.

[Sims, 1990] Sims, K. (1990). Particle animation and rendering using data parallel computation. *Computer Graphics (Siggraph '90 proceedings)*, pages 405–413.

[Smith, 1984] Smith, A. R. (1984). Plants, fractals, and formal languages. *SIGGRAPH Comput. Graph.*, 18(3):1–10.

[Strain, 1997] Strain, J. (1997). Fast adaptive 2D vortex methods. *J. Comput. Phys.*, 132:108–122.

[Tuckerman et al., 1992] Tuckerman, M., Berne, B. J., and Martyna, G. J. (1992). Reversible multiple time scale molecular dynamics. *J. Chem. Phys.*, 97(3):1990–2001.

[van Dommelen and Rudensteiner, 1989] van Dommelen, L. and Rudensteiner, E. (1989). Fast, adaptive summation of point forces in the two-dimensional Poisson equation. *Journal of Computational Physics*, 83:126–148.

[Walther and Koumoutsakos, 2001] Walther, J. H. and Koumoutsakos, P. (2001). Three-dimensional particle methods for particle laden flows with two-way coupling. *J. Comput. Phys.*, 167:39–71.

[Williamson, 1980] Williamson, J. H. (1980). Low-storage Runge-Kutta schemes. *J. Comput. Phys.*, 35:48–56.