Modelling of Extreme Ocean Waves Using High Performance Computing

Stuart Archibald

Department of Civil & Environmental Engineering

Imperial College London

Thesis submitted for the degree of Doctor of Philosophy

Declaration of originality

All the work presented herein is that of the author.

Any work or contributions made by others is referenced appropriately.

Abstract

This thesis describes the development of a fully nonlinear numerical model for the simulation of surface water waves. The model has the ability to compute the evolution of both limiting and overturning waves arising from the focussing of wave components in realistic ocean spectra. To accomplish this task, a multiple-flux implementation of a boundary element method is used to describe the evolution of a free surface in the time domain over an arbitrary bed geometry.

Unfortunately, boundary element methods are inherently computationally expensive and although approximations exist to reduce the complexity of the problem, the effects of their use in physical space is unclear. To overcome some of the computational intensity, the present work employs novel computational approaches to both reduce the run time of the simulations and make accessible predictions of wave fields that were previously unfeasible. The advances in computational aspects are made through the use of parallel algorithms running in a distributed computing environment. Further acceleration is gained by running parts of the algorithm on many-core co-processing devices in the form of the, habitually called, graphics processing unit. Once a reasonably efficient implementation of the boundary element method is achieved, attention is turned to further algorithmic optimisations, particularly in respect of computing the kinematics field underlying the extreme wave events.

The flexibility of the model is demonstrated through the accurate simulation of extreme wave events, this includes near-breaking and overturning wave phenomena. Finally, by harnessing the power of high performance computing technologies, the model is applied to an engineering design problem concerning the wave-induced loading of an offshore jacket structure. The work presented is not merely a study of a single wave event and its interaction with a structure, but rather a whole multitude of wave-structure interaction events that could not have been computed within a realistic time frame were it not for the use of high performance computing. The outcome of this work is the harnessing of distributed and accelerated computing to enable the rapid calculation of numerous fully nonlinear wave loading events to provide a game changing outlook on structural design and

the reliability for offshore structures; such calculations having not previously been possible.

I would like to dedicate this work to my dearest Emily and loving family for putting up with my computers (again!)

Acknowledgements

First, I would like to thank Prof. Chris Swan for having me in his research group, for providing guidance, support and enthusiasm, and for letting me have the freedom of research to achieve my full potential. Thanks also go to Prof. Rod Sobey for his continual support and for always fighting from my computational corner. In addition, I am very grateful for the support provided by the UK Engineering and Physical Sciences Research Council (EPSRC) under grant number F022964/1.

I've been very lucky to have such a lovely bunch of people with which to share research offices. So thanks go to Vanessa, Alice, Eirini, Johannes, Latheef and Francesco. Special thanks are required for: Adam, for being a good friend and kindred spirit in the quest for better research practice. Milena, for so many helpful conversations about BEM (especially the mathematics) and so much amusement regarding Linux, your excitement over `grep` still makes me smile. Jannicke, for tolerating three years of halls and nearly four years of sharing an office with me, always being supportive, for so many laughs and for your friendship. Finally, and most of all, Marios, for helping me with numerical methods, maths and co-authoring the core of EPIC_BEM, and despite having moved on still providing opinions and thoughts on just about anything, but above all for your friendship.

Outside the research group a number of individuals and groups have been very helpful. Thanks go to: Graham Markall for numerous fruitful discussions on accelerator hardware and just about anything else computer related. The entire of ICLUG for keeping the Linux community alive, particularly James Kimber, Simon Fayer and Graham Markall. Alex Lamaison for "man coffee", the many geeky discussions and friendship. Bill and Dave, the lab technicians who always make things entertaining. Becky, for ordering, organising and being helpful and cheerful. Prof. Paul Kelly and the SPO group for many interesting presentations and discussions.

I would like to thank Dr. Tamer Zaki and the Imperial College High Performance Computing Service for loan of a GPU enabled cluster computer and indeed maintaining and being exceedingly reasonable over the use of the HPC machines. Special thanks go to Simon Burbidge and Matt Harvey, sysadmins extraordinaire.

Further thanks go to Matt for the numerous conversations and the odd code snippit regarding CUDA, and to Simon for many helpful and humorous discussions.

Finally, but definitely not least, I would like to thank my family for their support. Mum and Dad, you have always been fantastic to me and I really appreciate it. Thanks to Dad for proof reading this work, it was really helpful, and Mum, you were right, I never was any good at simple problems. Thanks to Claire "subspace" Archibald for proof reading, checking the mathematics and having a million conversations about subspace invariance, you're a brilliant sister. Thanks to Hugh for always being so entertaining, I always feel better about my work after you've appropriately mocked it, you're a brilliant brother!

Above all, I'd like to thank my wife, Emily. You have always been so supportive, patient, loving and caring and I couldn't have done this without you.

Contents

List of Figures

List of Tables

List of Algorithms

Listings

Glossary

This work has many connections to computing and computational matters where acronyms are rife. In addition, there are a number of acronyms associated with boundary element schemes, the numerical model used in the present study. Therefore, for ease of reading, a glossary is provided below to assist those less familiar with the large number of acronyms employed.

2-norm	The 2-norm of variable $\mathbf{x}$, $\|\mathbf{x}\|_2$, is given by $\sqrt{(\mathbf{x}^T\mathbf{x})}$. It is also known as the Euclidean norm or Euclidean length.
ABM	**A**dams, **B**ashforth and **M**oulton, the authors of a predictor-corrector integration scheme as described in Butcher (2003).
ALU	**A**rithmetic **L**ogic **Unit**. An electronic unit that performs both arithmetic and logical operations. They are the building blocks of microprocessors.
API	**A**pplication **P**rogramming **I**nterface. An software interface to allow actions in one piece of software to interact with actions in another.
BIE	**B**oundary **I**ntegral **E**quation. The governing equation on which boundary element solutions are based. See equation (2.2).
BLAS	**B**asic **L**inear **A**lgebra **S**ubprograms. An API standard for the publication of libraries that perform basic linear algebra operations on vectors and matrices.
CPU	**C**entral **P**rocessing **U**nit. The part of a computer that carries out the instructions contained within a program, it is the piece of electronics that allows a computer to function.

CUDA **C**ompute **U**nified **D**evice **A**rchitecture (depreciated acronym). A parallel computing architecture developed by NVIDIA PLC.

FSBCs **F**ree **S**urface **B**oundary **C**ondition**s**. Refers to the dynamic and kinematic free surface boundary conditions, DFBCS and KFSBCS respectively. See §2.3.

GMRES **G**eneralised **M**inimal **Res**idual method. An iterative method for solving a general system of linear equations developed by Saad & Schultz (1986).

GPU **G**raphics **P**rocessing **U**nit. A special microprocessor that offloads graphics processing from the CPU.

HPC **H**igh **P**erformance **C**omputing. The use of specialised computing arrangements for computational work. A prime example of this would be executing programs on a cluster computer.

IP **I**nternet **P**rotocol. Part of the internet protocol suite, a set of communication protocols for networks including the internet. See also TCP.

LRWT **L**inear **R**andom **W**ave **T**heory. A theory for predicting irregular wave forms based on the superposition of sinusoidal forms.

MPI **M**essage **P**assing **I**nterface is a specification for an API that allows processes to communicate by passing messages.

OpenMP **Open** **M**ulti-**P**rocessing is an API that supports multi-platform shared memory multiprocessing programming.

OS **O**perating **S**ystem, a software interface between the computational hardware and the user, responsible for managing hardware resources and software activities.

PC **P**ersonal **C**omputer.

PCIe (bus) **P**eripheral **C**omponent **I**nterconnect **e**xpress bus. An interface for add-in expansion cards and a motherboard level interconnect. The main

	performance difference between PCIe and previous buses is a point-to-point serial link opposed to a shared parallel bus architecture.
SIMD	**S**ingle **I**nstruction, **M**ultiple **D**ata. In computing, the concept of multiple processing elements simultaneously performing the same instruction on multiple data.
SIMT	**S**ingle **I**nstruction **M**ultiple **T**hread. A concept used by NVIDIA PLC. to allow scalar thread processing on a data set.
SM	**S**treaming **M**ultiprocessor. A term coined by NVIDIA PLC. for use in relation to the layout of ALUs in their CUDA enabled devices.
TCP	**T**ransmission **C**ontrol **P**rotocol. Part of the internet protocol suite, a set of communication protocols for networks including the internet. See also IP.

1 Introduction

Computing power has increased manyfold in the past decade, consequently the numerical models run on today's desktops can only have been dreamt of by previous generations of numerical modellers. In the world of offshore oil and gas production, the increase in storm frequency and severity, coupled with the ever increasing challenges that need to be overcome to design and operate in such extreme circumstances, dictate that accurate numerical predictions of wave loading behaviour are of increasing importance. More than a decade ago, it was possible to provide reasonable numerical models of fluid loading on offshore structures. However, these simulations took prohibitively large amounts of computational effort and so were rarely undertaken. With recent increases in computing power it is now not only possible to provide even more accurate wave loading predictions for structures, but with the introduction of so-called "super computers" it is possible to run a large number of these simulations and start to look at wave-induced loading in a very different manner. In addition to simply providing fluid loading information to structural engineers, accurate numerical predictions can be used to provide information about what wave conditions will generate the worst loading on the sub-structure; including variation in both space and time. Furthermore, as numerous wave loading events can be simulated rapidly (and cheaply), the whole concept of the reliability of existing structures can be revisited by statisticians giving better insight into structural safety in the offshore industry. The motivation for the present work is to harness the very considerable power of distributed and accelerated computing to enable the rapid calculation of numerous fully nonlinear wave loading events thereby providing a game-changing outlook on the design and reliability of offshore structures.

1.1 Motivation

The driving force behind the present work comes from two, different, but equally important areas. The first being the advances made in numerical modelling techniques for free surface flows, namely the multiple-flux approach in boundary element methods. This approach was outlined by Hague & Swan (2009) and is directly relevant to simulations involving numerical wave tanks. The second area is that of scientific computing and the readily available large computational processing power made available through super-computing facilities.

The motivation surrounding advances in numerical modelling techniques arises from the achievements of Hague (2006) in the development of a boundary element method (BEM) for the modelling of free surface flows using a multiple-flux approach. This method was shown to work well in two spatial dimensions, and some progress was also made in a three dimensional implementation. However, the three dimensional application suffered two main problems. First, the boundary element method is fundamentally an $O(n^2)$ scheme in computational time (presuming an optimised indirect linear system solver is employed), where n is the number of nodes describing a domain. Second, there was a lack of resolution in the diagonal direction caused by the use of eight node serendipity elements. As a result, execution times for the three dimensional implementation developed by Hague (2006) were excessive. For example, computing a focussed wave event in a fairly standard domain with 12,000 nodes on a ∼3GHz Xeon dual processor workstation took around two weeks. Clearly, this code base gave unacceptable run times and left no possibility for the required increase in resolution or domain size necessary to achieve accurate descriptions of highly nonlinear waves. As a result, reducing the computational run time of such a scheme became a key theme to the present work.

The motivation from the field of scientific computing is best summarised by Gregory F. Pfister who famously wrote in his book on parallel computing, “In Search of Clusters” (Pfister, 1998), that there are three ways to do anything faster: ‘work harder’, ‘work smarter’ and ‘get help’. He then showed how these concepts can be applied to the field of computing. Since Pfister (1998) was first published a lot of the concepts remain relevant, but the computing platforms on which they are implemented have changed. Multi-core processors and multi-processor computers now exist on most people’s desktops, making shared memory programming more readily available. Clusters and massively parallel computers are now very common with more applications making use of scalable computing power. Most recently, accelerator hardware has been developed which will undoubtedly change the face of scientific computing with special relevance to linear algebra.

The idea of ‘working harder’ on a problem can be applied using the newly developed accelerator cards, ‘working smarter’ involves the use of more intelligent algorithms, while ‘getting help’ comes in a variety of forms, but usually involves a distributed computing environment. Within the present study, these concepts are explored on as many levels as possible using a multitude of different programming paradigms, hardware and algorithms.

To summarise, the focus of the present work involves coupling the world of high performance computing with recent advances in the boundary element modelling technique. The desired end result and underlying motivation for the whole project, as mentioned previously, is to have an accurate fully nonlinear numerical modelling technique for the simulation of surface water waves. The model must be able to be run in reasonable time frames such that more information can be gained when it comes to optimising the design (and hence the reliability of) offshore structures.

1.2 Aims

The primary goal of the work is the development and application of a numerical model that is capable of dealing with high resolution, large, computational domains thereby enabling the accurate prediction of extreme wave events and the associated water particle kinematics. Splitting the work into manageable, self contained, sub-tasks gives rise to the following goals.

i) To formulate a BEM to allow the modelling of large, complex or highly nonlinear wave fields with high accuracy.

ii) To validate the model to ensure that it performs correctly.

iii) To optimise the computationally intensive aspects of the BEM such that the solution time for the model becomes tolerable for realistic applications.

iv) To consider the application of the model to an actual engineering problem, identifying the benefits of improved kinematics predictions.

v) To apply the model to investigate extreme ocean waves and, in particular, wave overturning.

1.3 Layout

This thesis is divided into a number of sections and chapters, each containing some introductory and concluding material surrounding the technical material under discussion. In some sections the technical material is in the form of an academic paper to be published in an internationally leading journal. In others, the material is in a traditional thesis format. All the papers are co-authored by Prof. C. Swan and where others are involved their contributions are stated and acknowledged. The chapters are ordered in such a way that they address the above noted goals. However, some goals are accomplished over multiple chapters, requiring a number of techniques to be applied.

Chapter 1 introduces the thesis and describes the content of the work completed. It also briefly reviews the current state of the numerical modelling techniques used to describe extreme ocean waves and the evolving state of computing hardware at the time of writing.

Chapter 2 contains a generic formalisation and derivation of the BEM. The features specific to the BEM employed for this work are discussed along with key choices surrounding options that arise whilst implementing such a scheme.

Chapter 3 discusses methods of reducing the computational run times associated with BEM calculations and describes how BEM problems can be mapped to a distributed computing environment.

Chapter 4 looks at employing a new Krylov subspace based iterative matrix solver on a state-of-the-art parallel processing architecture for application in the matrix solving phase of the BEM solutions.

Chapter 5 investigates the coupling of a BEM running in a distributed computing environment (established in Chapter 3) with the hardware accelerated Krylov subspace based matrix solver of Chapter 4.

Chapter 6 begins with a formal validation of the BEM model developed herein. It returns to some of the theory derived in Chapter 2 and looks at reapplying the theory in a number of different ways to obtain kinematics predictions associated with a given wave simulation. Finally, a real life engineering application of the BEM with respect to loading on the sub-structure of an offshore platform is considered. Comparisons are drawn against best practice outlined in the American Petroleum Institute (APInst) design guidelines.

Chapter 7 turns to the topic of the breaking (or overturning) of focussed wave groups arising from realistic spectra. Results are compared to a number of well known analytical wave theories and some insight into the process of wave breaking in real seas is presented.

Chapter 8 discusses future work that is needed in the field to supplement the work completed in this thesis. This discussion revolves around the mapping of existing algorithms to new hardware and the need for mesh free numerical methods.

1.4 Context

In recent years there has been a considerable amount of research concerning the application of a BEM for simulating free surface waves completed by other members of Professor Swan's research group. Their contributions are readily acknowledged in the following text; their results being important to, but distinct from, the achievements of the present study.

In 2006 Caroline Hague (Hague, 2006) completed her PhD thesis, the emphasis of her research being the development of a two dimensional model for simulating free surface flows using the "multiple-flux method" to mitigate the infamous "corner problem". Both the multiple-flux method and the corner problem are discussed later in this thesis. Hague (2006) also sought to extend the two dimensional model to three dimensions and had some degree of success. The main issues with the three dimensional model were two fold; first, a lack of adaptability of the computational domain generation system meant a regular spaced grid forming a single box being the only option available. Second, the excessively long run times for the code leading to the inevitable restrictions this places on the spatial resolution and hence the accuracy of the wave calculations.

Building on the work of Hague, Christou (2008) extended the two dimensional BEM model to include structural configurations such as barriers (Christou *et al.*, 2009) and breakwaters (Christou *et al.*, 2008), as well as varying bathymetries such as beach slopes. In the final year of Christou's research period, the present author joined Professor Swan's research group. At this stage, it was decided that the three dimensional BEM model of Hague needed considerable further development to fulfil its maximum potential. Christou and the author started from scratch designing and writing an entirely new code base, using the multiple-flux concept of Hague, to develop the present BEM framework. The new framework was designed with full adaptability in terms of the domains and boundary conditions used, and at the same time makes considerable use of distributed computing to accelerate the run time of the model employed. Indeed, if the overall code is considered in its present form, approximately 50% of the code was developed in conjunction with Christou (on a 50:50 basis) and the remainder of the code is entirely due to the present author. In the chapters that follow, contributions by others, (notably Christou) are explicitly noted and gratefully acknowledged.

Following the completion of Christou's work, the author rewrote most of the code base to improve the reliability of the model and added a large number of new features, full details of which are given in the chapters that follow.

1.5 Achievements

Within the present study, the main achievements are as follows:

i) The development of a three dimensional multiple-flux BEM running on a distributed computing framework.

ii) The implementation of a Krylov subspace based linear system solver running a new hardware architecture with a large number of processing cores.

iii) The application of the developed BEM solution to the calculation of the wave loads acting on an offshore jacket (space-frame) structure. This representing a problem of significant practical importance and demonstrating the necessity of accurate (fully nonlinear) descriptions of extreme ocean waves.

iv) New insights into the nature of the breaking of wave groups and their underlying water particle kinematics.

2

Boundary Element Method (BEM); a Generic Formulation

2.1 Introduction

This chapter concerns the mathematical derivation and subsequent numerical implementation of the basic aspects of the BEM. The concept of a "numerical wave tank" is explored and some insight into the choices that are available and the route chosen during different stages of derivation and implementation are discussed. The efficient numerical implementation of a BEM scheme is the subject of later chapters. In the present discussion issues of numerical efficiency are highlighted, but a thorough investigation is delayed until later in the thesis. In effect, this chapter outlines the building blocks on which the final model relies.

2.2 Governing equations

The fluid within a domain, Ω, is assumed to be both incompressible and inviscid, while the fluid flow is considered to be irrotational. These assumptions allow mass continuity to be expressed in terms of Laplace's equation and applied throughout the domain,

$$\nabla^2\phi = 0 \ \in \ \Omega, \tag{2.1}$$

where $\phi(x, y, z, t)$ is the velocity potential. The variables x, y and z define a local Cartesian coordinate system, where (x, y) define the horizontal coordinates, z is measured vertically upwards from the mean water level, and t denotes time. From ϕ, a velocity field $\mathbf{u}$ can be defined as $\mathbf{u} = \nabla\phi = (u, v, w)$.

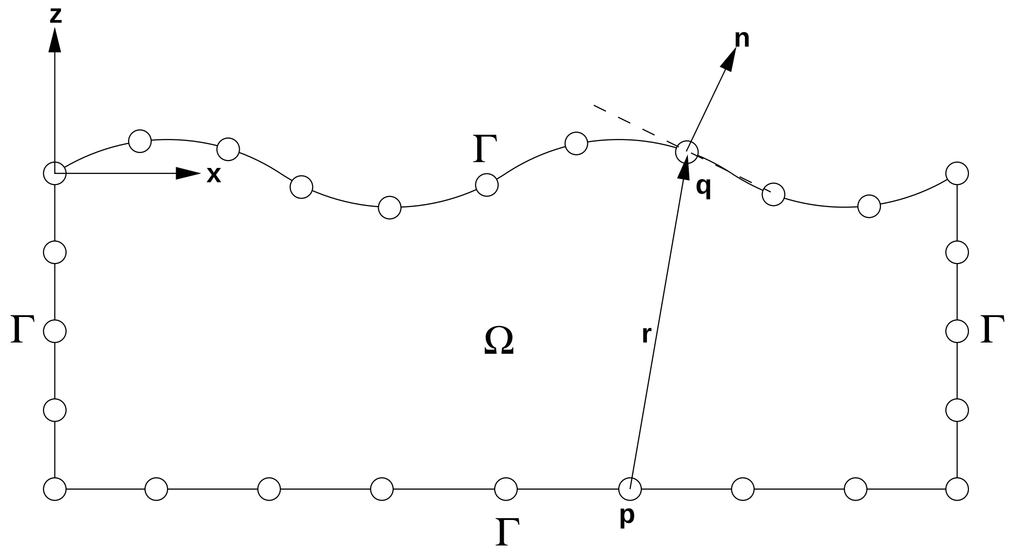


Figure 2.1: 2D cross section in $x - z$ domain, schematic relating to mathematical notation in equation (2.2).

Within a BEM a fundamental solution to equation (2.1) can be described, in the context of two points in real three dimensional space, by the free-space function $G(r) = \frac{1}{4\pi r}$, where $r = |\mathbf{r}|$, the magnitude of the vector between the source and field points. Adopting Green's second identity, the problem can be reduced from one describing volumes to one describing surfaces. The resulting boundary integral equation (BIE) being given by

$$c_p\phi_p + \int_\Gamma \phi_q \frac{\partial G}{\partial n} d\Gamma = \int_\Gamma G \frac{\partial \phi_q}{\partial n} d\Gamma, \tag{2.2}$$

where n is the direction of the outward normal, c_p is a geometric coefficient describing the exterior solid angle of the boundary at the source point, and Γ denotes the boundary of the domain. Figure 2.1 indicates the relationship between the quantities in equation (2.2) for a two dimensional (y = constant) slice through a three dimensional domain. Further discussion of the calculation of c_p follows later as computing this term relies on mathematical relations that have not yet been expressed. As equation (2.2) describes the relation between the field potential, ϕ, and the potential flux, ϕ_n, this relationship can be used to define one of these variables provided the other is known. This corresponds to the fundamental methodology under-pinning any BEM solution.

2.3 Free surface boundary conditions

In order for a numerical model to be able to simulate the evolution of a free surface in the time domain, a number of boundary conditions are required. The boundary conditions are used to describe the behaviour of two fundamental properties of the numerical domain. The first is the position of the points forming the boundary of the domain and the second, the associated potential at these nodal positions. These quantities correspond to the key variables identified in equation (2.2). To ensure that the model has the ability to describe the evolution of the wave field (or the fluid domain), the boundary conditions must be constructed in such a manner that they are amenable to numerical time marching. To achieve this, the boundary conditions must have an associated time dependence. The boundary condition associated with positional change is the so-called “kinematic free surface boundary condition” (KFSBC). This ensures the water surface is a streamline. In contrast, the boundary condition associated with potential change is referred to as the “dynamic free surface boundary condition” (DFSBC). This stipulates that the pressure acting on the water surface is a constant.

In deriving and explaining the boundary conditions in the sub-section that follows, some of the methods from the reference text of Dean & Dalrymple (1984) are used. To begin the derivation some expressions common to the formulation of the boundary conditions are given. These more general expressions lead to some options regarding frames of reference. Following this discussion, a general method for deriving a set of boundary conditions in any frame of reference is provided. With this complete, constraints appropriate to the frames of reference discussed earlier are applied and a complete set of boundary conditions derived.

2.3.1 General expressions

To begin the derivation, it is assumed that there is a scalar velocity potential within the domain

$$\phi = \phi(\mathbf{x}, t), \tag{2.3}$$

where, $\mathbf{x}$ is the position vector (x, y, z). In addition to the velocity potential it is also assumed that there is an associated velocity vector,

$$\mathbf{v}(\mathbf{x}, t), \tag{2.4}$$

which defines the movement of the frame of reference. This velocity vector is entirely separate from the velocity vector describing the fluid velocity, $\mathbf{u}(\mathbf{x}, t)$. However, if desired, $\mathbf{v}$ may have components equal to those in the velocity vector, $\mathbf{u}$, such that the frame of reference moves in some directions at a velocity equal to the velocity of the fluid.

To introduce the required time derivative for these variables, a total derivative based upon the chain rule can be defined as follows:

$$\frac{d}{dt}(\phi(\mathbf{x}, t)) = \frac{\partial \phi}{\partial t} + \nabla\phi \cdot \frac{d\mathbf{x}}{dt}. \tag{2.5}$$

In this case, it is clear that the total derivative is dependant on the velocity vector, $\mathbf{v}(\mathbf{x}, t)$, defined by

$$\frac{d\mathbf{x}}{dt} = \left(\frac{dx}{dt}, \frac{dy}{dt}, \frac{dz}{dt}\right), \tag{2.6}$$

which describes the chosen path $\mathbf{x}(t)$ in space.

2.3.2 Frames of reference

With equation (2.5) dependent on $\mathbf{x}(t)$, it is clear that the movement of this path, or the choice of reference frame, will govern the nature of the derived boundary conditions. If $\mathbf{x}(t)$ is set to zero then the frame of reference is Eulerian which dictates that the positions at which the boundary condition is applicable are fixed in space. In this case the movement of the nodal positions within the domain is forbidden. This condition is not (generally) desirable for the purpose of simulating an evolving nonlinear wave field, although it has obvious benefits in terms of defining the linearised boundary conditions, these being appropriate to the description of infinitesimally small waves. Indeed, Isaacson & Cheung (1990) adopted exactly this approach and showed that with the adoption of a Taylor series expansion about this fixed position (corresponding to still water level) a weakly nonlinear or second-order approximation could be achieved.

An alternative to the Eulerian frame of reference is the so-called Lagrangian frame. This allows movement in all Cartesian directions such that the frame of reference moves with the fluid at a given position. A refinement to this frame of reference is to allow the path to move in directions that are best suited to the physical constraints of the model at a given position. With this in mind, three frames of reference are useful in the current application. These all have some elements of a fully-Lagrangian frame of reference; the reference frame moving in some directions, but being fixed in others.

At this point it is necessary to briefly describe the different situations and associated boundary conditions that may arise when implementing a numerical model of free surface flow. First, if the free surface remains single valued, the waves not being permitted to overturn, the points describing the free surface need only be free to move in the vertical direction. This condition is henceforth referred to as a one-third-Lagrangian, or semi-Lagrangian, frame of reference. However, if the free surface is likely to become multi-valued (waves overturning), the points describing the free surface will need the freedom to move in all Cartesian directions. This condition is henceforth referred to as a fully-Lagrangian frame of reference. It is also possible to have an “intermediate” situation in which the free surface will need to be free to move in the vertical direction and one horizontal direction, yet be fixed in the other. This is hereafter referred to as a two-thirds-Lagrangian frame of reference and might be adopted where, for example, a multi-valued free surface occurs against a reflective wall or barrier.

2.3.3 Generic Lagrangian boundary conditions

The method to derive the one-third, two-thirds and the fully-Lagrangian boundary conditions, corresponding to their respective frames of reference, is the same in each case; the only difference being the restrictions in movement applied to the frame of reference. For a generic KFSBC, the frame of reference is moved with the free surface such that the free surface does not change. This condition is applied through the use of a material derivative operating on an arbitrary function $F(\mathbf{x}, t) = 0$ which describes the free surface,

$$\frac{DF(\mathbf{x}, t)}{Dt} = \frac{\partial F(\mathbf{x}, t)}{\partial t} + \nabla F(\mathbf{x}, t) \cdot \mathbf{u} = 0. \tag{2.7}$$

The material derivative must be equal to zero as the free surface does not change in the frame of reference employed. Expanding and rearranging this expression gives,

$$\frac{\partial F}{\partial t} = -\mathbf{u} \cdot \nabla F. \tag{2.8}$$

Written in this form, the boundary condition expresses the required time derivative in terms of the spatial quantities and is therefore in a form appropriate to time marching. In applying this method to the three (previously mentioned) frames of reference, all that is required is the formulation of a function, $F(\mathbf{x}, t) = 0$, consistent with the given spatial constraints and the consequent evaluation of equation (2.8).

Following the derivation of the KFSBC, it is possible to compute the DFSBC. This seeks to express the time derivative of the velocity potential, ϕ, consistent with the condition that the pressure on the free surface remains constant. With a fully-Lagrangian frame of reference employed, the total derivative of the potential is required and is given by equation (2.5). The first term, $\frac{\partial \phi}{\partial t}$, can be calculated by recalling that the free surface, η, is a streamline and as pressure is constant along a streamline. Bernoulli's equation can therefore be employed to give

$$-\frac{\partial \phi}{\partial t} + \frac{P_\eta}{\rho} + \frac{1}{2}\left[\left(\frac{\partial \phi}{\partial x}\right)^2 + \left(\frac{\partial \phi}{\partial y}\right)^2 + \left(\frac{\partial \phi}{\partial z}\right)^2\right] + g\eta = C, \tag{2.9}$$

where P_η is the pressure on the water surface ($z = \eta$), ρ is the density of the fluid and C is the Bernoulli constant. Defining a gauge pressure such that $P_\eta = 0$, setting $C = 0$ and rearranging gives

$$\frac{\partial \phi}{\partial t} = -g\eta - \frac{1}{2}\left[\left(\frac{\partial \phi}{\partial x}\right)^2 + \left(\frac{\partial \phi}{\partial y}\right)^2 + \left(\frac{\partial \phi}{\partial z}\right)^2\right]. \tag{2.10}$$

This expression can be substituted into equation (2.5) giving

$$\frac{d}{dt}(\phi(\mathbf{x},t)) = -g\eta - \frac{1}{2}\left[\left(\frac{\partial\phi}{\partial x}\right)^2 + \left(\frac{\partial\phi}{\partial y}\right)^2 + \left(\frac{\partial\phi}{\partial z}\right)^2\right] + \nabla\phi\cdot\frac{d\mathbf{x}}{dt}. \tag{2.11}$$

Recalling that $\nabla\phi = \frac{\partial\phi}{\partial\mathbf{x}} = \mathbf{u}$ in a fully-Lagrangian frame of reference, equation (2.11) can be further simplified to give:

$$\frac{d}{dt}(\phi(\mathbf{x},t)) = -g\eta - \frac{1}{2}\mathbf{u}^2 + \mathbf{u}\cdot\frac{d\mathbf{x}}{dt}. \tag{2.12}$$

Expressed in this form, the product $\mathbf{u}\cdot\frac{d\mathbf{x}}{dt}$ is the only term that needs to be evaluated to complete the derivation of the DFSBC. However, since this directly depends upon the frame of reference adopted, equation (2.12) gives the general form of the required boundary conditions.

In calculating the term $\frac{d\mathbf{x}}{dt}$, it is important to note that,

$$\frac{d\mathbf{x}}{dt} = \left(\frac{dx}{dt}, \frac{dy}{dt}, \frac{dz}{dt}\right). \tag{2.13}$$

Given that the position of interest is on the water surface, the movement of the vertical coordinate with respect to time is identical to the movement of the water surface with respect to time, hence, $\frac{dz}{dt} \equiv \frac{\partial\eta}{\partial t}$. With $\frac{\partial\eta}{\partial t}$ calculated using the KFSBC, there is a clear link between the two boundary conditions.

This completes the generic derivation of the boundary conditions, the application of this theory to the three alternative frames of reference follow. In the first case, corresponding to the one-third-Lagrangian case, a full description of the derivation of the boundary conditions is provided. In the following two cases, the similarity in the derivation is such that only limited explanation is required.

2.3.4 One-third-Lagrangian frame of reference

This constraint allows points on the free surface to move in the vertical direction only.

KFSBC

To evaluate the one-third-Lagrangian case, the position of the free surface elevation is a function of the two horizontal directions and time such that

$$z = \eta(x, y, t). \tag{2.14}$$

This can be rearranged into a function of F, with z measured vertically and $z = 0$ corresponding to the still water level,

$$F(x, y, z, t) = z - \eta(x, y, t) = 0. \tag{2.15}$$

Applying equation (2.8) term by term

$$\frac{\partial F}{\partial t} = -\frac{\partial \eta}{\partial t} \tag{2.16}$$

and

$$\nabla F = \left(-\frac{\partial \eta}{\partial x}, -\frac{\partial \eta}{\partial y}, 1\right), \quad \mathbf{u} = (u, v, w), \tag{2.17}$$

$$\nabla F \cdot \mathbf{u} = -u\frac{\partial \eta}{\partial x} - v\frac{\partial \eta}{\partial y} + w, \tag{2.18}$$

hence

$$\frac{\partial \eta}{\partial t} = w - u\frac{\partial \eta}{\partial x} - v\frac{\partial \eta}{\partial y}. \tag{2.19}$$

Equation (2.19) defines the KFSBC appropriate to points that can move in the vertical directions only. This result is identical to that stated by Hague (2006) and Christou *et al.* (2008), although neither provide detailed derivations.

DFSBC

To derive the DFSBC for the one-third-Lagrangian case the path $\mathbf{x}$ on which the surface moves is restricted solely to the vertical direction so

$$\mathbf{x} = (0, 0, z). \tag{2.20}$$

Recalling that $z = \eta(x, y, t)$ from above, it follows that

$$\frac{d\mathbf{x}}{dt} = \left(0, 0, \frac{d\eta}{dt}\right). \tag{2.21}$$

Evaluating the final term in equation (2.12)

$$\mathbf{u} \cdot \frac{d\mathbf{x}}{dt} = (u, v, w) \cdot \left(0, 0, \frac{d\eta}{dt}\right) = w\frac{d\eta}{dt}, \tag{2.22}$$

substituting into equation (2.12) and adopting the $\frac{d\eta}{dt}$ term defined in equation (2.19), gives the required form of the DFSBC appropriate to the one-third-Lagrangian frame of reference,

$$\frac{d}{dt}(\phi(\mathbf{x}, t)) = -g\eta - \frac{1}{2}\mathbf{u}^2 + w\frac{d\eta}{dt}. \tag{2.23}$$

Again, this result is consistent with those stated in Hague (2006) and Christou *et al.* (2008).

2.3.5 Two-thirds-Lagrangian frame of reference

Within this frame of reference, points on the free surface are allowed to move in the vertical direction and one horizontal direction. In the equations that follow, the boundary conditions are derived for the case in which the constrained horizontal direction is aligned with the y coordinate. This condition relates to conditions arising on a boundary in the $x - z$ plane. For completeness, the boundary conditions appropriate to the case in which the surface points are constrained in the x direction are also provided.

KFSBC

To evaluate the two-thirds-Lagrangian case, the position of the free surface elevation is a function of one horizontal direction and time such that,

$$F(y, z, t) = z - \eta(y, t) = 0. \tag{2.24}$$

Applying equation (2.8)

$$\frac{\partial F}{\partial t} = -\frac{\partial \eta}{\partial t}, \tag{2.25}$$

$$\nabla F = \left(0, -\frac{\partial \eta}{\partial y}, 1\right), \quad \mathbf{u} = (u, v, w), \tag{2.26}$$

$$\nabla F \cdot \mathbf{u} = -v\frac{\partial \eta}{\partial y} + w, \tag{2.27}$$

hence

$$\frac{\partial \eta}{\partial t} = w - v\frac{\partial \eta}{\partial y}. \tag{2.28}$$

The corresponding expression for a surface point which is free to move in y and z, but constrained in x, is given by,

$$\frac{\partial \eta}{\partial t} = w - u\frac{\partial \eta}{\partial x}. \tag{2.29}$$

DFSBC

To derive the DFSBC for the two-thirds-Lagrangian case, the path $\mathbf{x}$ on which the surface moves is restricted to the vertical, z, direction and the x direction so,

$$\mathbf{x} = (x, 0, z). \tag{2.30}$$

Recalling that $z = \eta(y, t)$ from above, it follows that

$$\frac{d\mathbf{x}}{dt} = \left(\frac{dx}{dt}, 0, \frac{d\eta}{dt} \right). \tag{2.31}$$

Evaluating the final term in equation (2.12)

$$\mathbf{u} \cdot \frac{d\mathbf{x}}{dt} = (u, v, w) \cdot \left(\frac{dx}{dt}, 0, \frac{d\eta}{dt} \right) = u^2 + w\frac{d\eta}{dt}, \tag{2.32}$$

substituting into equation (2.12) and adopting the $\frac{d\eta}{dt}$ term defined in equation (2.28), gives the required form of the DFSBC appropriate to the two-thirds-Lagrangian frame of reference,

$$\frac{d}{dt}(\phi(\mathbf{x}, t)) = -g\eta - \frac{1}{2}\mathbf{u}^2 + u^2 + w\frac{d\eta}{dt}. \tag{2.33}$$

The corresponding expression for a surface point which is free to move in y and z, but constrained in x, is given by

$$\frac{d}{dt}(\phi(\mathbf{x}, t)) = -g\eta - \frac{1}{2}\mathbf{u}^2 + v^2 + w\frac{d\eta}{dt}. \tag{2.34}$$

2.3.6 Fully-Lagrangian frame of reference

This frame of reference allows points on the free surface to move in all Cartesian directions thereby allowing the simulation of overturning waves.

KFSBC

To evaluate the fully-Lagrangian case, the position of the free surface elevation is a function of time only such that,

$$F(z, t) = z - \eta(t) = 0. \tag{2.35}$$

Applying equation (2.8)

$$\frac{\partial F}{\partial t} = -\frac{\partial \eta}{\partial t}, \tag{2.36}$$

$$\nabla F = (0,0,1)\,, \quad \mathbf{u} = (u,v,w), \tag{2.37}$$

$$\nabla F \cdot \mathbf{u} = w, \tag{2.38}$$

hence,

$$\frac{\partial \eta}{\partial t} = w. \tag{2.39}$$

This expression is identical to that adopted by Grilli *et al.* (2001), Hague (2006) and Christou *et al.* (2008).

DFSBC

To derive the DFSBC for the fully-Lagrangian case, the path $\mathbf{x}$ on which the surface moves is not restricted and so can move in any direction,

$$\mathbf{x} = (x,y,z). \tag{2.40}$$

Recalling that $z = \eta(t)$ above, it follows that,

$$\frac{d\mathbf{x}}{dt} = \left(\frac{dx}{dt}, \frac{dy}{dt}, \frac{d\eta}{dt}\right). \tag{2.41}$$

Evaluating the final term in equation (2.12),

$$\mathbf{u} \cdot \frac{d\mathbf{x}}{dt} = (u,v,w) \cdot \left(\frac{dx}{dt}, \frac{dy}{dt}, \frac{d\eta}{dt}\right) = u^2 + v^2 + w\frac{d\eta}{dt}, \tag{2.42}$$

and substituting into equation (2.12) gives

$$\frac{d}{dt}(\phi(\mathbf{x},t)) = -g\eta - \frac{1}{2}\mathbf{u}^2 + u^2 + v^2 + w\frac{d\eta}{dt}, \tag{2.43}$$

where the $\frac{d\eta}{dt}$ term is given by the fully-Lagrangian KFSBC (equation (2.39)). Evaluating this fully gives,

$$\frac{d}{dt}(\phi(\mathbf{x},t)) = -g\eta - \frac{1}{2}\mathbf{u}^2 + u^2 + v^2 + w^2 = -g\eta + \frac{1}{2}\mathbf{u}^2, \tag{2.44}$$

which is again consistent with the boundary conditions adopted by Grilli *et al.* (2001), Hague (2006) and Christou *et al.* (2008).

2.4 Numerical implementation

This section discusses the numerical implementation of equation (2.2) and the application of the boundary conditions described in §2.3. It also reviews some of the choices that arise when implementing such a method, and explains how the chosen paths give rise to varying outcomes.

2.4.1 Computational domain

The computational domain employed in the present numerical model takes the form of a so-called "numerical wave tank" (NWT). The NWT has clear similarities with its experimental equivalent, not least in the boundary conditions it applies. With reference to Figures 2.2 and 2.3, comparisons between an actual laboratory wave basin (as found in the hydrodynamics laboratory in the Dept. of Civil and Environmental Engineering at Imperial College London) and the NWT used in the present work can be drawn. The input to the laboratory wave basin is produced along one side by flap type wave paddles. The wave basin being relatively wide in comparison to its length to minimise the "shadow" region and associated diffraction effects. In the NWT there is the possibility of having input conditions, Γ_{input}, specified on two sides of the domain. This allows a smaller domain to be used as the shadow region is effectively eliminated. Within the NWT the input conditions simply prescribe a flux along the boundary depending on the wave conditions required.

At the downstream or far end of the wave basin there is a beach structure of a parabolic shape. This is used to dissipate the wave energy in an attempt to minimise the presence of unwanted wave reflections in the working area of the wave basin. In the NWT the "beach" consists of a radiation condition (Sommerfeld, 1949) coupled with a numerical sponge layer, the validity of such an approach affirmed by Hague (2006). The remaining vertical sides of both the laboratory wave basin and the NWT are made up of reflecting walls. In the case of the wave basin this is simply a glass panel, while in the case of the NWT this is a zero flux (no flow) boundary condition. In addition, the two horizontal boundaries in the NWT, the free surface, $\Gamma_{surface}$, and bed, Γ_{bed}, behave identically to those in the laboratory wave basin. One obvious advantage of the NWT over the laboratory wave basin is that changing the bed and tank geometry in the NWT takes a couple of minutes and is limited only by the ability of the programmer designing it. In contrast, changing just the bed geometry within the laboratory wave basin can take many days and the designs are limited to what is achievable within the bounds of the existing basin geometry. Furthermore, it should be noted that it is common for a NWT to model only half the working area of the laboratory wave basin. A plane of symmetry exists in a large number of typical wave scenarios, as a result, employing a reflective wall along the plane of symmetry in a half size NWT allows for greater computational efficiency with no loss of accuracy.

2.4.2 Boundary value problem

The concepts involved in producing a BEM for a NWT are relatively simple. The NWT can be described in terms of mixed boundary conditions whereby the bed and side walls are described in terms of Neumann boundary conditions (prescribed ϕ_n), and at the free surface a Dirichlet boundary condition (prescribed ϕ). For example:

$$\begin{aligned}
\frac{\partial \phi}{\partial n}|_{\Gamma_{bed}} &= 0, && (2.45)\\
\frac{\partial \phi}{\partial n}|_{\Gamma_{side,input,radiation}} &= \mathbf{u} \text{ (specified)}, && (2.46)\\
\phi|_{\Gamma_{surface}} &= \phi \text{ (specified)}. && (2.47)
\end{aligned}$$

Using these results, solving a system based on equation (2.2) allows all the unknowns to be determined. In the present example, this involves solving for $\frac{\partial \phi}{\partial n}$ on the free surface and ϕ on all other boundaries. On obtaining this solution, all the spatial gradients of ϕ and η can be determined, and, the free surface boundary conditions (§2.3) integrated with respect to time to define how both the water surface elevation (η) and the velocity potential on the surface ($\phi|_{z=\eta}$) evolve over a small time step. The calculation procedure can then be repeated to determine the evolution of the wave field over many time steps.

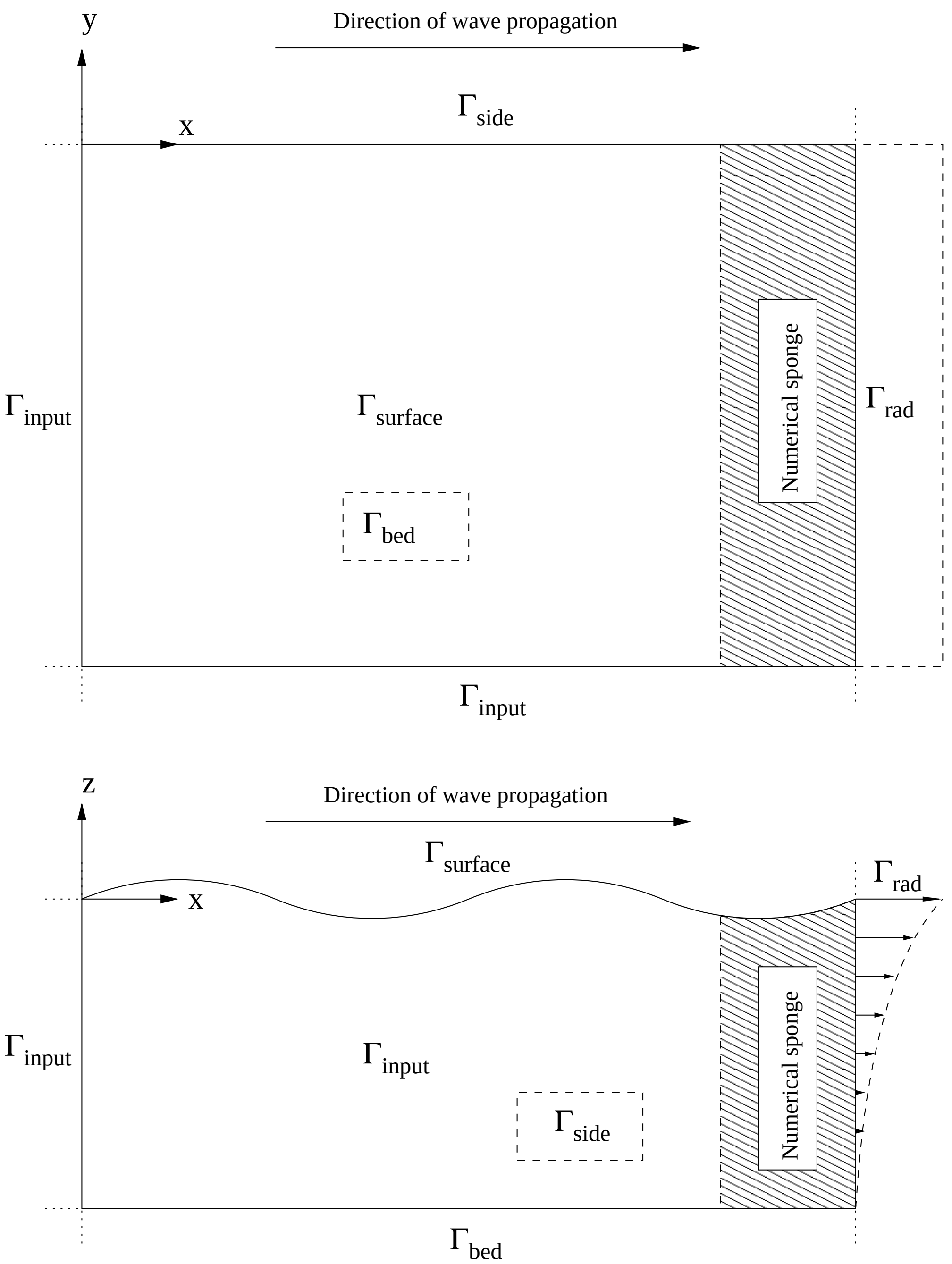


Figure 2.2: Schematic of the numerical wave tank (NWT).

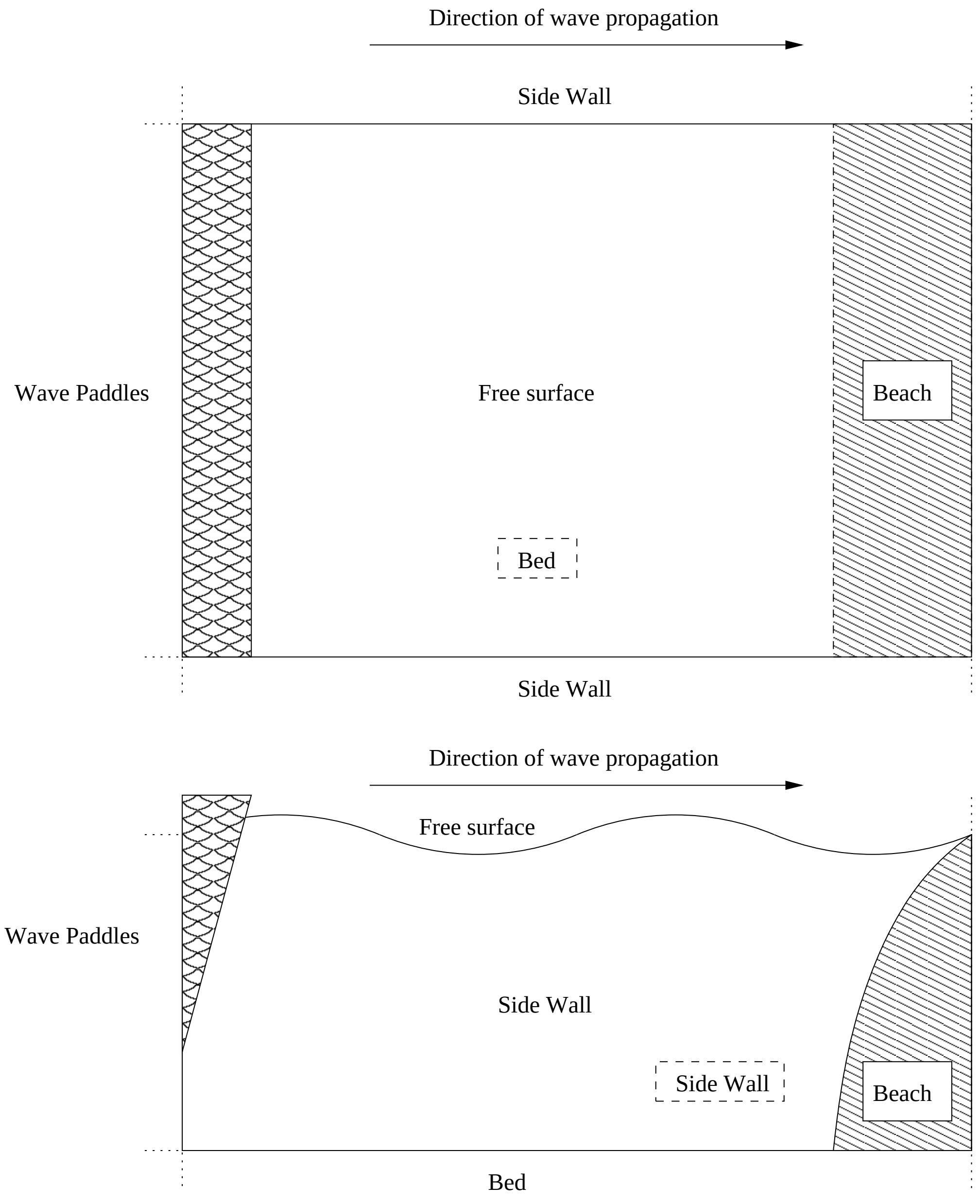


Figure 2.3: Schematic of wave basin found in the fluids laboratory in the Dept. of Civil and Environmental Engineering at Imperial College London.

2.4.3 Application of the free surface boundary conditions

Given the boundary value problem outlined in §2.4.2, the free surface boundary conditions would typically be applied as follows: whilst the model is run using a one-third-Lagrangian frame of reference, all the nodes on the free surface, $\Gamma_{surface}$, would use the boundary conditions given in equations (2.19) and (2.23). If it then became desirable to run the model with the free surface defined in a fully-Lagrangian frame of reference (necessary for the description of wave breaking), then $\Gamma_{surface}$ would use the boundary conditions given in equations (2.39) and (2.44). The only exceptions to this rule being the nodes forming the intersections between the surface and the vertical sides. The nodes at the $\Gamma_{surface}$-Γ_{input} and $\Gamma_{surface}$-Γ_{rad} interfaces would have the one-third-Lagrangian conditions applied (equations (2.19) and (2.23)) such that the nodes at these interfaces can only move in the z direction and therefore do not drift into the domain causing a distortion of the NWT. In contrast, on the $\Gamma_{surface}$-Γ_{side} interface the two-thirds-Lagrangian condition (equations (2.28) and (2.33)) becomes applicable. On this interface the nodes are providing a no flux (reflective) condition in the y direction to simulate a wall, or a line of symmetry, but the nodes need to be free to move in the x and z directions to allow a wave to break along this boundary.

2.4.4 Discretising the BIE

In solving the BIE, equation (2.2), a large number of integrals are required around the boundary of the domain. To facilitate such calculations, the boundary of the domain is discretised with N nodes forming M iso-parametric, bi-quadratic Lagrange elements. Bi-quadratic elements are used based upon the assumption that the discretisation is of sufficiently high resolution that the variation of material quantities (that the elements represent) can be accurately described by a quadratic function over the length scale of the element. The elements are made up of nine nodes, as shown in Figure 2.4, and the nodes are ordered in an anti-clockwise fashion to be consistent with Green's identities. This ensures that the fluid being modelled is explicitly defined as being internal to the domain.

Using iso-parametric bi-quadratic elements to discretise the domain allows the

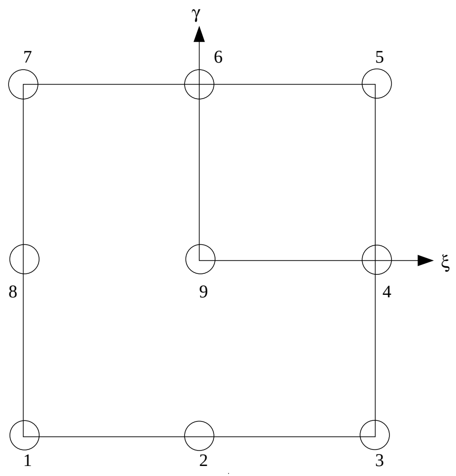


Figure 2.4: Schematic of an individual 9-node element.

use of quadratic shape functions to express any property associated with the element; examples of the latter being position, potential, and potential flux. To aid the implementation of numerical methods, two intrinsic directions are associated with the elements, ξ and γ, as indicated in Figure 2.4. With ξ and γ defined such that $-1 \leq \{\xi, \gamma\} \leq 1$, any variable represented by an element can then be expressed using the generic function,

$$\varrho(\xi,\gamma) = \sum_{k=1}^{9} S_k(\xi,\gamma) f_k \tag{2.48}$$

where f is the function to be represented, k is the node number within an element, and S are the well known quadratic shape functions defined in many texts (see for example Brebbia & Dominguez (1992) and Dominguez (1993)).

By substituting these generic function expressions into equation (2.2), a form closer to that required for computational purposes is obtained:

$$\begin{aligned} c_p\phi_p + \sum_{j=1}^{M}\sum_{k=1}^{9} \phi_k \int_{\xi_j}\int_{\gamma_j} S_k(\xi,\gamma)\frac{\partial G}{\partial n} J(\xi,\gamma) d\xi d\gamma \\ = \sum_{j=1}^{M}\sum_{k=1}^{9} \frac{\partial \phi_k}{\partial n} \int_{\xi_j}\int_{\gamma_j} S_k(\xi,\gamma) G J(\xi,\gamma) d\xi d\gamma, \end{aligned} \tag{2.49}$$

where k is, again, the node number within an element, j indicates the element number, ranging from 1 to M, and $J(\xi,\gamma)$ is used to represent the Jacobian transform required to map the 3D space in which the element resides to the finite

2D space of the intrinsic coordinates,

$$J(\xi,\gamma) = \left|\frac{\partial \mathbf{x}}{\partial \xi} \times \frac{\partial \mathbf{x}}{\partial \gamma}\right|. \tag{2.50}$$

In the interests of vectorising the problem for solution on a computer, the summations over each element in equation (2.49) can be rewritten:

$$\sum_{j=1}^{M}\sum_{k=1}^{9} \phi_k \int_{\xi_j}\int_{\gamma_j} S_k(\xi,\gamma)\frac{\partial G}{\partial n} J(\xi,\gamma) d\xi d\gamma = \sum_{j=1}^{M} \mathbf{h}_j \cdot \phi_j, \tag{2.51}$$

$$\sum_{j=1}^{M}\sum_{k=1}^{9} \frac{\partial \phi_k}{\partial n} \int_{\xi_j}\int_{\gamma_j} S_k(\xi,\gamma) G J(\xi,\gamma) d\xi d\gamma = \sum_{j=1}^{M} \mathbf{g}_j \cdot \frac{\partial \phi}{\partial n}_j, \tag{2.52}$$

where ϕ_j and $\frac{\partial \phi}{\partial n}$ are vectors with 9 members; each member corresponding to the potential and potential flux for a node of element j. In a similar fashion, the vectors with 9 members, $\mathbf{h}_j$ and $\mathbf{g}_j$, correspond to the double integrals in equations (2.51) and (2.52) respectively. The vectors $\mathbf{h}_j$ and $\mathbf{g}_j$ can then be summed and inserted into fully populated arrays (often referred to as "influence matrices"), $\widehat{\mathbf{H}}$ and $\mathbf{G}$ respectively. This leads to a simplification of equation (2.49) lending to,

$$c_p\phi_p + \widehat{\mathbf{H}}\Phi = \mathbf{G}\Phi_n, \tag{2.53}$$

where Φ and Φ_n are used to express vectors of potential and potential flux, each member corresponding to a node within the computational field.

2.4.5 Rigid mode technique

Rather conveniently, the value of c_p defined in equation (2.53) can be computed using a technique called the "rigid mode method" described in Becker (1992). If equation (2.53) is rearranged as

$$c_p\phi_p = \mathbf{G}\Phi_n - \widehat{\mathbf{H}}\Phi, \tag{2.54}$$

the c_p coefficients can be absorbed in the diagonals of $\widehat{\mathbf{H}}$ to give $\mathbf{H}$,

$$\mathbf{G}\Phi_n = \mathbf{H}\Phi. \tag{2.55}$$

The absorption of c_p only affects the diagonals of the $\mathbf{H}$ matrix and it is possible to compute these terms using the following logic.

If a constant potential ϕ_c were applied simultaneously to every node, and the resulting zero change in the potential flux this produces, substituted into the system equation (2.55), the following is obtained:

$$\mathbf{G} \cdot \mathbf{0} = \mathbf{H} \cdot \Phi_c. \tag{2.56}$$

Recalling that the diagonals in the **H** matrix contain unknowns from the introduction of the c_p coefficient, the sum of the diagonals in the **H** matrix can be set equal to the negative sum of the off diagonals,

$$\mathbf{H}_{ii} = -\sum_{i \neq j} \mathbf{H}_{ij}. \tag{2.57}$$

In this way the system can be solved giving a fully populated **H** matrix without needing to explicitly compute c_p. This method is valid for any domain where the net global flux is zero and provides a convenient way of avoiding the computation of c_p.

2.4.6 Numerical integration

To evaluate the double integrals in equations (2.51) and (2.52) standard two dimensional Gaussian quadrature is used on the assumption that the integrals are smoothly varying over the integration field. In the first instance it is possible to express a generic double integral as:

$$\int_{-1}^{1} \int_{-1}^{1} f(\xi, \gamma) d\xi d\gamma = \sum_{l=1}^{N_\xi} \sum_{m=1}^{N_\gamma} f(\xi_l, \gamma_m) w_l w_m \tag{2.58}$$

where, once again, f is the variable to be integrated. The values of N_ξ and N_γ represent the number of integration points in the intrinsic ξ and γ directions respectively, while, w_l and w_m are the weightings for the discrete coordinates ξ_l and γ_m. Tables of w_l and w_m are given in Abramowitz & Stegun (1964), however, the present work uses the a more adaptive approach for dynamically generating the weightings as described in Press *et al.* (1990). From experience, in single precision arithmetic, $(N_\xi = N_\gamma) \geq 6$. However, $N_\xi = N_\gamma = 4$ can be used and, as a consequence, a two fold speed up in computational time is obtained, this being proportional to the square of the number of integration points. Unfortunately, although such a speed up is desirable, the model typically becomes more unstable with time as the cumulative error from the integrals increases.

The double integrals in equations (2.51) and (2.52) both contain $\frac{1}{r^k}$ relations (as part of $G(r) = \frac{1}{4\pi r}$ or $\frac{\partial G(r)}{\partial n} = \frac{-1}{4\pi}\frac{\mathbf{r}\cdot\mathbf{n}}{r^3}$, with $n \geq 1$) such that when $r \to 0$ a singularity occurs. The generic Gaussian quadrature in equation (2.58) requires a smoothly varying field to obtain accurate results and this is not possible should a singularity occur. There are two ways around such a problem. The first is to use a different quadrature scheme that is better equipped to deal with singular behaviour; an example of this being $\tanh - \sinh$ quadrature outlined by Bailey *et al.* (2005). Although, theoretically, this option may produce good results, measurement of the accuracy of the scheme would be difficult and would require extra computational effort, which is undesirable.

The second, and perhaps more standard scheme, is to transform the awkward integrals into integrals over finite fields. There are effectively three possible scenarios in the discretised scheme depending on the distance between the source point p and the field point q.

i) Assume p and q belong to different elements. In this case r is some value greater than or equal to an element length, and therefore equations (2.51) and (2.52) are non-singular. As a result, standard Gaussian quadrature can be employed.

ii) Assume p and q belong to the same element, but correspond to different nodes within the element. In this case the value of r is less than an element length.

    (a) Evaluating equation (2.52), $G(r) \propto \frac{1}{r}$, indicating singular behaviour as $r \to 0$. However, the shape function S_k tends to zero as $r \to 0$ such that the effect of the singularity in $G(r)$ is cancelled out. It therefore follows that standard Gaussian quadrature can again be used to evaluate the integral.

    (b) In evaluating equation (2.51), $\frac{\partial G(r)}{\partial n} \propto \frac{1}{r^2}$ and the shape function S_k tends to zero as $r \to 0$. This effectively leaves a $\frac{1}{r}$ relationship to be evaluated. Some special consideration is required for this term and is computed via coordinate manipulations discussed below.

iii) Assume p and q belong to the same element and correspond to the same nodes within the element. In this case $r = 0$.

   (a) Equation (2.52) becomes singular and therefore directly applying Gauss-Legendre quadrature is inappropriate. In this case some coordinate manipulation, again discussed below, is required.

   (b) The evaluation of equation (2.51) is computed by the rigid mode technique as discussed in §2.4.5.

To avoid the evaluation of the singular integrals noted above some basic coordinate transformations can be employed. The element in question can be divided into a number of triangular sub-elements with one vertex per sub-element at the source node (Dominguez, 1993). Integration can be undertaken over the sub-elements using a new set of intrinsic coordinates associated with the sub-element and the singularity issues are thereby avoided. Preliminary investigations have shown that for single precision arithmetic there is no advantage in using more sub-elements than the minimum required to describe the element. An example of a minimum and a maximum number of sub-elements for a source point being at node 1 of an element is given in Figure 2.5. Some examples of typical sub-element configurations used in the present work are given in Dominguez (1993).

Using sub-elements and their associated intrinsic coordinates ($-1 \leq \{\tilde{\xi}, \tilde{\gamma}\} \leq 1$), the original element's intrinsic coordinates (ξ and γ) can be expressed as,

$$\xi = \sum_{n=1}^{3} \tilde{S}_n(\tilde{\xi}, \tilde{\gamma})\xi_n \tag{2.59}$$

$$\gamma = \sum_{n=1}^{3} \tilde{S}_n(\tilde{\xi}, \tilde{\gamma})\gamma_n, \tag{2.60}$$

where ξ_n and γ_n correspond to the nodes used to form the verticies of the triangular sub-elements and $\tilde{S}_n$ are the shape functions for a triangular quadratic element; details of the latter provided by Dominguez (1993). In a similar manner

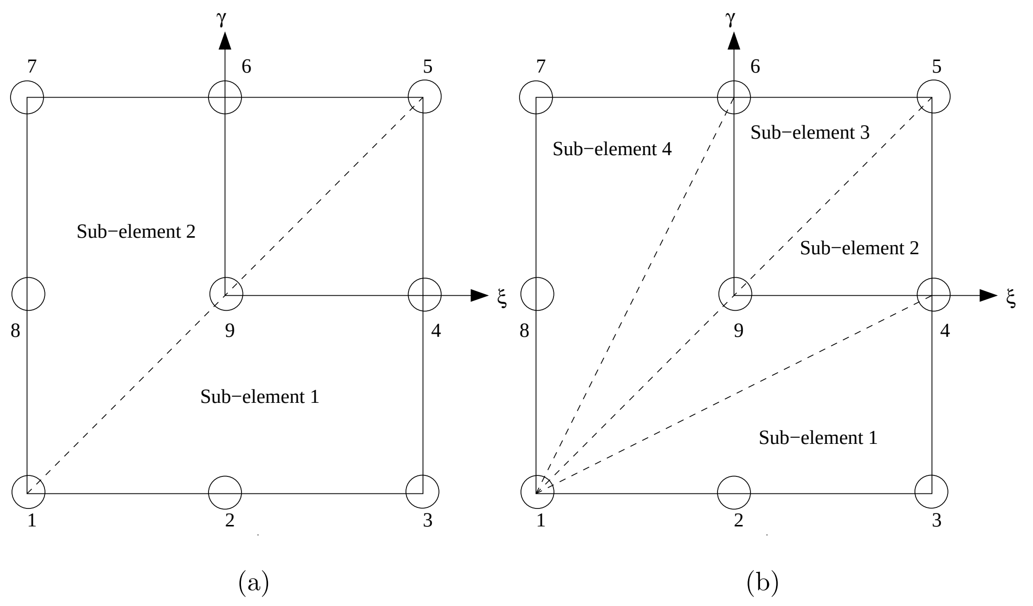


Figure 2.5: Schematic of sub-element configurations adopted to avoid singular integrals. (a) Minimum sub-element configuration for a source point at node 1. (b) Maximum sub-element configuration for a source point at node 1.

to equation (2.49), a Jacobian can be formed from the sub-element coordinate system to make Gaussian quadrature possible over the triangular elements,

$$\tilde{J}(\tilde{\xi},\tilde{\gamma}) = det \begin{vmatrix} \frac{\partial \xi}{\partial \tilde{\xi}} & \frac{\partial \gamma}{\partial \tilde{\xi}} \\ \frac{\partial \xi}{\partial \tilde{\gamma}} & \frac{\partial \gamma}{\partial \tilde{\gamma}} \end{vmatrix} . \tag{2.61}$$

As a consequence of these manipulations, the double integrals in equations (2.51) and (2.52) can now be written in such a way that they evaluate to the correct finite quantities should the occasion arise,

$$\sum_{i=1}^{N_{se}} \int_{-1}^{1} \int_{-1}^{1} S_k(\tilde{\xi},\tilde{\gamma}) \frac{\partial G}{\partial n} \tilde{J}(\tilde{\xi},\tilde{\gamma}) J(\tilde{\xi},\tilde{\gamma}) d\tilde{\xi} d\tilde{\gamma}, \tag{2.62}$$

$$\sum_{i=1}^{N_{se}} \int_{-1}^{1} \int_{-1}^{1} S_k(\tilde{\xi},\tilde{\gamma}) G \tilde{J}(\tilde{\xi},\tilde{\gamma}) J(\tilde{\xi},\tilde{\gamma}) d\tilde{\xi} d\tilde{\gamma}, \tag{2.63}$$

where N_{se} is the number of sub-elements used to divide the element in question. This expression completes the derivation of terms required to implement the BIE (2.2).

2.4.7 The treatment of corners and geometric discontinuities

Dealing with geometric discontinuities, arising in physical space as corners and edges, has long plagued users of boundary element schemes. The BIE (2.2) is only valid for smoothly varying boundaries and clearly corners do not come under this category. Within the literature there are three main approaches for dealing with corners, involving the implementation of *double nodes*, *discontinuous elements* and *multiple-fluxes*.

The *double node* approach allows multiple nodes to exist in the same position, with one node belonging to each element that forms the discontinuity. Each node is then associated with one direction of the outward normal and, as a consequence, one potential flux. When the boundary integral system is formed and solved, compatibility conditions are required to ensure that the potential at all nodes forming the discontinuity is the same such that the model is physically real. Similarly, compatibility conditions for the potential flux can be employed to incorporate additional (known) physical information into the outward normal directions, an example being zero flux at a wall. This method of using multiple nodes to describe discontinuities has been used extensively by Grilli *et al.* (1989) and his co-workers for many years ((Grilli & Subramanya, 1996), (Grilli *et al.*, 2001)). Further discussion of the extended compatibility conditions is provided by Grilli & Svendsen (1990).

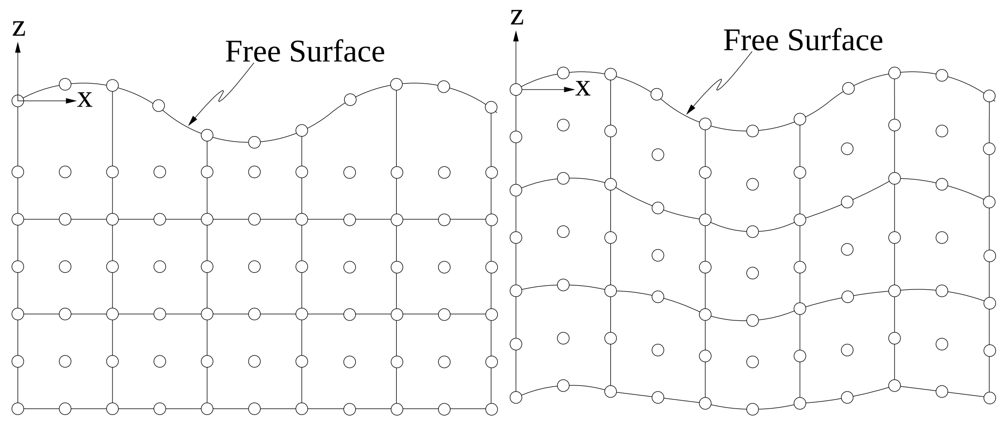
The use of *discontinuous elements* to overcome the problem of discontinuities involves the moving of the nodes of the element that would form a discontinuity away from the intersection and into the actual element (albeit a small amount) such that the discontinuity is not explicitly defined. This method has been used by Hamano *et al.* (2003) and they report that it worked well for their application of nonlinear standing waves in vessels. However, in the case of a NWT, the physics of geometric discontinuities is of substantial interest and their accurate representation essential to the successful modelling of free surface flows. The importance of this matter is reviewed in detail by Hague & Swan (2009).

The *multiple-flux* approach is conceptually different in nodal representation from the other two approaches in that it uses a single node at a discontinuity, this node being shared by the elements that form the boundary of the corner or edge. The fluxes that exist at the discontinuity are associated with the single node, but are referenced with respect to the outward normal of the elements that make up the discontinuity. Hence, in the formulation of the BIE (2.2) the discontinuity can be fully represented in terms of the known information and the system does not require any forcing of compatibility before or after the solution formulation. The multiple-flux method was first employed by Brebbia & Dominguez (1992) in the context of structural applications. This method was then taken and applied successfully to a NWT by Hague & Swan (2009). In the interests of accuracy, the multiple-flux approach is employed in the present work as it is believed that retaining all the information available within the formulation and solution of the BIE (2.2) is critical to the accurate modelling of free surface flows.

2.4.8 Sliding elements for gradients and velocities

To compute the free surface boundary conditions outlined in §2.3, velocities and (if a semi-Lagrangian frame of reference is adopted), free surface gradients, need to be calculated on the surface boundary of the NWT. Additionally, to assist accurate integration of element properties, elements that form the side walls of the NWT need to deform in a uniform pattern; the latter best calculated from the velocity of the fluid represented by the element.

The boundary conditions relevant to the side walls of the domain are usually known or prescribed. For example, Γ_{side} is often prescribed as a "zero flux" condition to simulate the behaviour of a reflective boundary or plane of symmetry. As a result, the nodes below the free surface that have this boundary condition are simply place holders for values of potential used for integration purposes as in equation (2.2). Figures 2.6(a) and 2.6(b) show schematics of part of the boundary making up Γ_{side}. In Figure 2.6(a) only the nodes on the free surface are allowed to move vertically and it can be seen that elements near the surface become very distorted. As a result, they have poor aspect ratios which affects the accuracy of the integration schemes. In contrast, Figure 2.6(b) allows the movement of all nodes on the boundary in proportion to the velocity field at the node concerned. As a result, the nodes below the free surface move to reflect the disturbance of the water surface, the amplitude of the movement reducing with vertical elevation. This ensures that the aspect ratios of the elements are considerably improved.



(a) Boundary conditions applied to give fixed points.

(b) Boundary conditions applied to give moving points.

Figure 2.6: Side wall boundary condition schematic.

With regards to the velocity components on the boundary of the NWT, typically, some sort of numerical differentiation of the potential with respect to space is employed, examples being provided by Grilli & Svendsen (1990), Grilli *et al.* (2001), Hague & Swan (2009) and Christou *et al.* (2008). Similarly, the elevation of the surface boundary can be differentiated with respect to space to gain information about the spatial gradients. Clearly, it would be advantageous in terms of numerical implementation if the same schemes could be used for both types of derivative. Perhaps, the most common methods of numerical differentiation involves fitting polynomials and splines (Press *et al.*, 1990). Grilli *et al.* (2001) use a mixture of cubic splines and polynomials and enforces gradient continuity between elements to get the necessary gradients. Alternatively, Hague & Swan (2009) use a quadratic “sliding element” to give linear derivatives and to maintain a gradient continuity across element boundaries.

In the approach that follows, the numerical model has the facility to use quadratic, cubic and quartic “sliding elements” in their pure polynomial form, with gradient continuity ensured through spanning the interpolating polynomials across the boundaries of the elements of the domain. Preliminary work suggested that quadratic interpolating polynomials were not sufficiently accurate when computing the gradients of the very steepest waves. The cubic interpolating polynomials, although probably adequate for computing gradients of steep waves, have an unfortunate directional bias because four point polynomials are not symmetrical. Figure 2.7 demonstrates the asymmetric nature of a cubic polynomial, the node of interest (highlighted in black) can have its spatial derivatives computed using information including two nodes from upstream and one from downstream (polynomial option 1) or, alternatively, one node from upstream and two from downstream (polynomial option 2). As a result, a directional bias exists which can cause a number of problems and poses questions regarding the accuracy of such a scheme. For these reasons quartic polynomial interpolation is used and appears to be very successful in all but a few very steep wave cases. Further discussion of this is given in §7.3.

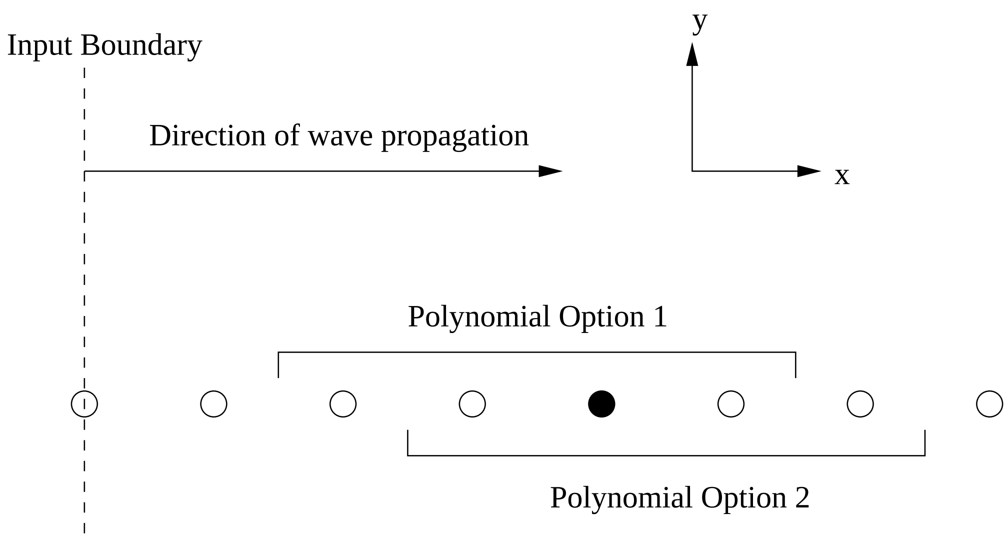


Figure 2.7: A schematic showing bias introduced as a result of a cubic polynomial representation, $\circ$ nodal positions, $\bullet$ node at which a derivative in the x direction is to be calculated.

The derivation of generic n^{th} order shape functions is discussed in §2.4.9. However, at this stage it is possible to discuss their placement and relationships within a "sliding element" and how this relates to physical quantities in a generic fashion.

The fluid velocities in Cartesian space can be summarised as $\mathbf{u} = \frac{\partial \phi}{\partial \mathbf{x}}$; $\mathbf{u}$ corresponding to velocities in the three Cartesian directions denoted by $\mathbf{x}$. Whilst it is possible to calculate these terms entirely from simple geometry, there is the added complication that the nodes used to define the polynomials may not be in line with the Cartesian axes. Fortunately, the polynomials can be orientated to lie in line with the intrinsic coordinate system (ξ, γ), and this can be resolved easily to the Cartesian system. Having mapped the polynomials to the intrinsic coordinate system, the velocities can be found by evaluating the following expressions:

$$\frac{\partial \phi}{\partial x} = \frac{\partial \phi}{\partial s}\cos(\theta_s)\cos(\alpha_s) + \frac{\partial \phi}{\partial m}\cos(\theta_m)\cos(\alpha_m) + \frac{\partial \phi}{\partial n}\cos(\theta_n)\cos(\alpha_n) \quad (2.64)$$

$$\frac{\partial \phi}{\partial y} = \frac{\partial \phi}{\partial s}\cos(\theta_s)\sin(\alpha_s) + \frac{\partial \phi}{\partial m}\cos(\theta_m)\sin(\alpha_m) + \frac{\partial \phi}{\partial n}\cos(\theta_n)\sin(\alpha_n) \quad (2.65)$$

$$\frac{\partial \phi}{\partial z} = \frac{\partial \phi}{\partial s}\sin(\theta_s) + \frac{\partial \phi}{\partial m}\sin(\theta_m) + \frac{\partial \phi}{\partial n}\sin(\theta_n). \quad (2.66)$$

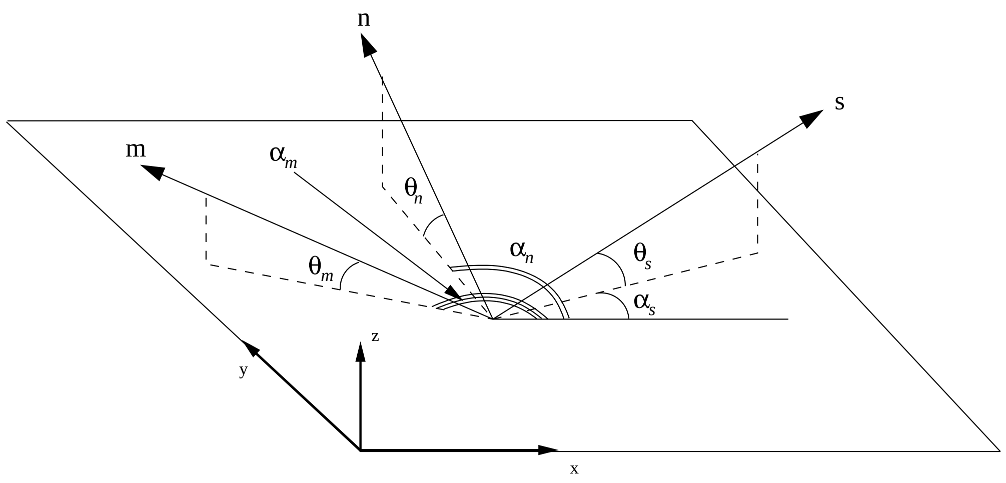


Figure 2.8: Schematic of the angles derived from unit vectors $\{\mathbf{m}, \mathbf{s}, \mathbf{n}\}$. In this example the angles are defined relative to the horizontal plane.

The unknown variables in these equations can be defined as follows. The variables $\theta_{m,s,n}$ are the angles made by the unit vectors $\{\mathbf{m}, \mathbf{s}, \mathbf{n}\}$ to the plane defined by the boundary in which the element resides. Likewise, the variables $\alpha_{m,s,n}$ refer to the associated azimuthal angles made by the vectors $\{\mathbf{m}, \mathbf{s}, \mathbf{n}\}$ with respect to the positive x direction. An example of these definitions, for an element that resides on a horizontal plane, is given in Figure 2.8.

These angles can be computed from the differentiation of the polynomials that form the shape functions within the "sliding element". The velocities in the intrinsic coordinate directions are given by:

$$\frac{\partial \phi}{\partial s} = \frac{\partial \phi}{\partial \xi}\left(\frac{\partial s}{\partial \xi}\right)^{-1} \quad \text{and} \quad \frac{\partial \phi}{\partial m} = \frac{\partial \phi}{\partial \gamma}\left(\frac{\partial m}{\partial \gamma}\right)^{-1} \tag{2.67}$$

and the value of $\frac{\partial \phi}{\partial n}$ is known from the solution to the BIE (2.2). Finally, the expressions $\frac{\partial s}{\partial \xi}$ and $\frac{\partial m}{\partial \gamma}$ are given by,

$$\frac{\partial s}{\partial \xi} = \sqrt{\left(\frac{\partial x}{\partial \xi}\right)^2 + \left(\frac{\partial y}{\partial \xi}\right)^2 + \left(\frac{\partial z}{\partial \xi}\right)^2}, \tag{2.68}$$

$$\frac{\partial m}{\partial \gamma} = \sqrt{\left(\frac{\partial x}{\partial \gamma}\right)^2 + \left(\frac{\partial y}{\partial \gamma}\right)^2 + \left(\frac{\partial z}{\partial \gamma}\right)^2}. \tag{2.69}$$

In a separate stage of the computation, the spatial gradients ($\partial\eta/\partial x$ and $\partial\eta/\partial y$) required for the one-third- and two-thirds-Lagrangian boundary conditions (equations (2.19) and (2.28)) need to be defined. This can be achieved using the same polynomials and "sliding elements" as employed for the velocities derived above. The specific terms required are given by the expressions:

$$\frac{\partial \eta}{\partial x} = \frac{\partial \eta}{\partial \xi}\left(\frac{\partial x}{\partial \xi}\right)^{-1} \quad \text{and} \quad \frac{\partial \eta}{\partial y} = \frac{\partial \eta}{\partial \gamma}\left(\frac{\partial y}{\partial \gamma}\right)^{-1} \tag{2.70}$$

2.4.9 Shape functions and their hierarchical nature

The generic quadratic function defined in equation (2.48) can be described using shape functions, as can the higher order generic functions required for computing derivatives in §2.4.8. For a one dimensional line element lying in the range $-1 \leq \xi \leq 1$, where ξ is an intrinsic coordinate, a simple method can be used to compute shape functions to any order. Once computed, the expression for the line element shape function can be convolved to form shape functions for any dimensioned space on the condition that the space formed from the boundaries of the line elements completely defines the space.

The algorithm for computing generic shape functions is relatively simple and relies on the implementation of two rules. For a one dimensional shape function for node i, S_i, that spans intrinsic coordinates $-1 \leq \xi \leq 1$ and is bound by m evenly distributed nodes, the rules are:

$$S_i \text{ at node } i \text{ must have the value 1,} \tag{2.71}$$

$$S_i \text{ vanishes over any node that is not } i. \tag{2.72}$$

These conditions are sufficient to define the form of the required shape functions. For example, considering a quadratic line element of general form

$$S_i = A\xi^2 + B\xi + C, \tag{2.73}$$

where A, B and C are unknown constants and the element has three nodes with intrinsic coordinates $\xi = \{-1, 0, 1\}$. Applying the above noted rules at each nodal position allows the shape function, S_i, at $i = 1, 2, 3$ to be determined. Details of this simple process being given on Table 2.1 and the results of the generic shape functions for this element given in equations (2.74), (2.75) and (2.76).

	$i=1$	$i=2$	$i=3$	A	B	C
S_1	1	0	0	$\frac{1}{2}$	$-\frac{1}{2}$	0
S_2	0	1	0	-1	0	1
S_3	0	0	1	$\frac{1}{2}$	$\frac{1}{2}$	0

Table 2.1: Quadratic shape functions, S_i, at nodes $i = 1, 2, 3$. Values and coefficients.

$$S_1 = \frac{1}{2}\xi(\xi - 1) \tag{2.74}$$

$$S_2 = 1 - \xi^2 \tag{2.75}$$

$$S_3 = \frac{1}{2}\xi(\xi + 1) \tag{2.76}$$

This simple approach can be adopted for any polynomial representation. Details of the adopted quartic representation are given in Table 2.2 with the coefficients $A - E$ relating to equation (2.77)

$$S_i = A\xi^4 + B\xi^3 + C\xi^2 + D\xi + E. \tag{2.77}$$

	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	A	B	C	D	E
S_1	1	0	0	0	0	2/3	−2/3	−1/6	1/6	0
S_2	0	1	0	0	0	−8/3	4/3	8/3	−4/3	0
S_3	0	0	1	0	0	4	0	−5	0	1
S_4	0	0	0	1	0	−8/3	−4/3	8/3	4/3	0
S_5	0	0	0	0	1	2/3	2/3	−1/6	−1/6	0

Table 2.2: Quartic shape functions, S_i, at nodes $i = 1 \ldots 5$. Values and coefficients.

To extend the shape functions to 2D space, simple convolutions of the expressions obtained for a 1D case at each point can be used. Continuing with the simple quadratic example, the shape function S_1 corresponding to node 1 in the element presented in Figure 2.4, comprises two 1D shape functions,

$$S_1|_\xi = \frac{1}{2}\xi(\xi - 1), \tag{2.78}$$

$$S_1|_\gamma = \frac{1}{2}\gamma(\gamma - 1), \tag{2.79}$$

and the simple convolution of these functions results in the 2D shape function,

$$S_1 = \frac{1}{4}\xi(\xi - 1)\gamma(\gamma - 1). \tag{2.80}$$

This method is rather convenient as it is hierarchical and so can be extended to any order polynomial and yet it is still simple to program (a scripting language can be used to generate the shape functions for use in `Fortran`).

Computing the derivatives of the shape functions with respect to their intrinsic coordinates is equally trivial and can be summarised for the 2D boundary elements used in the present work as follows:

$$\frac{\partial \varrho}{\partial x} = \sum_{i=1}^{N_\xi} \frac{\partial S_i(\xi,\gamma)}{\partial \xi}\varrho_i \tag{2.81}$$

$$\frac{\partial \varrho}{\partial y} = \sum_{i=1}^{N_\gamma} \frac{\partial S_i(\xi,\gamma)}{\partial \gamma}\varrho_i \tag{2.82}$$

$$\tag{2.83}$$

where, as before, ϱ is a generic function, x and y are the Cartesian directions and N_ξ and N_γ are the number of intrinsic coordinates.

2.4.10 Time marching

To calculate the evolution of the free surface, the boundary conditions are time marched with a time step, Δt. The size of the time step is determined according to the (rearranged) Courant condition,

$$\Delta t = \frac{|\Delta \mathbf{r}|\, C_0}{\sqrt{g d_0}}, \tag{2.84}$$

where $\Delta \mathbf{r}$ is the shortest distance between all two node pairings within the NWT, C_0 is the constant Courant number, g is the acceleration due to gravity and d_0 is the initial water depth. Based on a large number of preliminary calculations, $C_0 = 0.4$ was typically found to give the consistently accurate results; a similar value has been adopted by Grilli *et al.* (2001).

When the model ran in a semi-Lagrangian frame of reference, the combined 4^{th}-order-predictor and 5^{th}-order-corrector scheme of Adams, Bashforth and Moulton (ABM), described by Butcher (2003), was adopted. The Adams-Bashforth predictor is given by

$$\varsigma^*_{n+1} = \varsigma_n + \frac{\Delta t}{24}(55 f_n - 59 f_{n-1} + 37 f_{n-2} - 9 f_{n-3}) \tag{2.85}$$

and the Adams-Moulton corrector given by

$$\varsigma_{n+1} = \varsigma_n + \frac{\Delta t}{720}(251 f^*_{n+1} + 646 f_n - 264 f_{n-1} + 106 f_{n-2} - 19 f_{n-3}). \tag{2.86}$$

Within this formulation, Δt is the time step, ς is the variable to be time-marched, and f is its time derivative, or $f = \frac{d\varsigma}{dt}$. With regard to the subscripts, n indicates the current time, t, $n+1$ and $n-1$ indicate $t + \Delta t$ and $t - \Delta t$ respectively, while $n-2$ and $n-3$ denote $t - 2\Delta t$ and $t - 3\Delta t$ respectively. Finally, within equation (2.86), $f^*_{n+1} = f(\varsigma^*_{n+1})$ where ς^*_{n+1} arises from equation (2.85).

In its original form, the ABM scheme requires a fixed time step and information from three previous time steps to predict and correct the next. To provide this information at the beginning of the calculation a classic 4^{th} order Runge-Kutta integration scheme (Press *et al.*, 1990) was adopted using the same fixed time step required for the ABM calculations.

In those calculations in which it was desirable to switch the model from running in a semi-Lagrangian frame of reference to a fully-Lagrangian frame of reference, such that boundary nodes are permitted to move with the fluid's velocity, the necessity to maintain a constant Courant number requires the use of adaptive time stepping. For this reason, in a fully-Lagrangian frame of reference, the model employs an adaptive Runge-Kutta scheme based on keeping a stable Courant number with respect to the minimum nodal spacing between the drifting nodes.

2.5 Algorithmic implementation

Having considered each part of the boundary element scheme together, an algorithm is now required to join the parts to form a numerical model. The algorithm on which the present numerical model is based is described as follows:

i) Data specific to the problem formulation is read into memory. This includes coordinates, boundary conditions, element and node indices.

ii) The first three 'kick start' time steps are performed using a standard Runge-Kutta algorithm. In undertaking this task, each Runge-Kutta step requires four calls to the "intermediate step routine", details of which are described below.

iii) While the simulation time is less than the time at which the simulation switches to run in a fully-Lagrangian frame of reference (the latter necessary to model overturning waves), the solution is time marched using the Adam-Bashforth-Moulton predictor-corrector scheme. This process requires calling the "intermediate step" routine twice per time step.

iv) Once the simulation time is equal to or greater than the time at which the simulation switches to run in a fully-Lagrangian frame of reference, the solution is time marched using the adaptive Runge-Kutta scheme. This involves calling "intermediate step" routine four times per time step.

The "intermediate step" routine comprises the following actions:

a) Create the input fluxes from an appropriate analytical wave theory.

b) Calculate the radiation condition (if requested).

c) Formulate the influence matrix.

d) Solve the influence matrix.

e) Update the boundary conditions (Φ and Φ_n).

f) Compute the fluid velocities.

g) Compute the free surface gradients (only appropriate to calculations undertaken in a semi-Lagrangian frame of reference).

h) Enforce the radiation condition (if requested).

i) Update the free surface boundary conditions, identifying $\frac{d\eta}{dt}$ and $\frac{d\phi}{dt}$ for the time marching outlined in steps ii) to iv) above.

Although simple in construction, developing the algorithm into a workable base took a number of months and aspects of the procedure remain under continual development.

2.5.1 Computational challenges

A number of computational challenges were envisaged in the building of the application, most of these being gleaned from previous experience of other members of the research group and extensive use of the code profiling tool, GNU `gprof()`.

The first challenge in the new implementation of the BEM model (called "EPIC_BEM", the reasons for which become apparent in Chapter 3) was to use a different element type from eight node serendipity elements, thus nine node Lagrangian elements were chosen. This change was implemented to increase the resolution in the diagonal direction and to allow the existence of so called "bubble" modes. These changes were made with the view that they would model wave interactions in an improved manner, capturing more information in the diagonal directions.

Further to this, spatially larger and higher resolution domains were desirable, this requiring more nodes and elements to be used to define the numerical wave tank. As a result of changing the element type and making larger, higher resolution domains standard, both the node and element numbers had increased considerably for a typical domain. Larger domains in the EPIC_BEM model regularly having over twice the number of nodes as used by Hague (2006).

To summarise the requirements of the new EPIC_BEM code base, the following features were set out as targets:

- the ability to run very large domains,
- employing as high a resolution as possible,
- in as short a time frame as possible.

In addition, it was envisaged that the numerical model could be applied to other areas of free surface flow mechanics such as the interaction of waves with ships, breakwaters and platforms to name a few. For this to be possible the code had to have an adaptable, easy to use gridding system to form the numerical wave tank. Furthermore the code had to be well written and modular to allow extensions necessary to deal with specialised areas of free surface flows such as the computation of the movements of floating bodies.

2.6 Conclusion

Although the BEM of Hague & Swan (2009) and Hague (2006) is excellent in that it gives a good deal of insight into accurately predicting free surface flows and addresses the infamous “corner problem”, there are a number of features that the model lacks and indeed some features which are wholly undesirable. For example, the excessive run time and limits to the resolution that this imposes. For these reasons a completely new implementation of the BEM algorithm was produced, with special attention paid to areas in computation efficiency and accuracy that were perhaps overlooked when concentrating more on proof-of-concept work.

3

Parallel Implementation of Matrix Formation

3.1 Introduction

As discussed in §2.5.1 the computational intensity of the BEM is a real cause for concern, especially in respect of simulating high resolution, realistically sized wave fields. This chapter looks at the options available to reduce the run time of the BEM scheme and the use of high performance computing to aid such a program. First, benchmark tests are run on a serial implementation of the BEM code, and a view is taken based on these results as to how to decrease run time and yet retain a high level of accuracy. A method is developed to make use of a distributed computational environment and, based on this, scaling and performance metrics are investigated.

3.2 Parallel implementation of the BEM

Sections §3.3–§3.11 take the form of a technical paper, prepared for submission to an internationally leading journal, addressing the use of distributed computing in a boundary element method for applications in computational wave mechanics. Following completion of the paper, sections §3.12–§3.13 provide some additional information and concluding remarks concerning the parallel implementation of the matrix formulation.

On the use of distributed computing in a boundary element method with applications to free surface waves

S. Archibald

Dept. Civil and Environmental Engineering, Imperial College London, London, SW7 2AZ.

Abstract

This paper discusses the computational requirements of a multiple-flux boundary element method (BEM) applied to the description of free surface water waves that are both fully nonlinear and directionally spread. Computational profiling is undertaken, followed by an investigation into the available methods and technologies necessary to achieve the required reduction in computational effort. A distributed algorithm is developed and shown to be effective for large computational domains, with the added advantage that it can be run on any machine that supports an MPI environment. With the introduction of distributed computing a multiple-flux boundary element model can achieve computational speeds comparable to a BEM code employing a fast multipole method without the uncertainty in the accuracy-computation time relationship associated with the latter.

Key words: *distributed computing, parallel computing, boundary element methods, matrix formation.*

3.3 Introduction

The multiple-flux boundary element method (BEM) first outlined by Brebbia & Dominguez (1992) is ideally suited to the study of mixed boundary value problems involving the specification of both Neumann conditions (prescribed $\frac{\partial \phi}{\partial n}$) and Dirichlet conditions (prescribed ϕ), where ϕ is the velocity potential on the boundary and n the outward pointing normal. With the introduction of multiple-fluxes, difficulties associated with the corner problem are avoided and the method shown to be very successful at modelling surface water waves within what is commonly referred to as a numerical wave tank (Hague & Swan (2009), Christou *et al.* (2008), Christou *et al.* (2009)). However, when this approach is applied in a three-dimensional domain, which is necessary to incorporate the effects of directional spreading, the number of nodal points rapidly increases leading to prohibitively long computational times. To overcome this difficulty, an investigation into methods of increasing the speed of the computation was undertaken and is reported herein. The present paper does not concern the formulation of a 3D BEM, nor does it present wave calculations arising from the model; it purely describes a method to speed up the calculations without having to compromise the accuracy achieved. Whilst this is obviously important from the practical perspective of run-times, it is particularly important in the context of modelling realistic ocean wave fields; the latter being characterised by a broad spread of wave energy across both the frequency and the directional domains. As a consequence, the spatial (or nodal) resolution (Δx, Δy, Δz) necessary to achieve realistic solutions is inevitably high and this, in turn, necessitates a large computational effort.

3.4 Previous methods

There are various methods by which the computational time needed for a BEM can be reduced. These range from algorithm advancements to the use of distributed computing. A selection of commonly applied methods is reviewed as follows.

The method of domain decomposition is described in Trevelyan (1994) and successfully applied by Bai & Eatock Taylor (2007). This approach reduces the computational effort by splitting a large BEM domain into a number of smaller subdomains. As a result, instead of the BEM algorithm being $O(n^2)$, where n is the number of nodal points, it tends to $O(\{\frac{n^2}{\hat{m}}\})$, where $\hat{m}$ is the number of subdomains. Unfortunately, this approach is known to suffer from a number of problems, the most significant of which concerns the convergence of the final solution. Further discussion of the difficulties associated with domain decomposition is given by Strating & De Haas (1997).

A second, perhaps more rigorous, approach to reducing the computational effort of BEMs is the use of a fast multipole method (FMM), full details of which are given in Greengard & Rokhlin (1987). This algorithm reduces the computational effort of forming the influence matrix from $O(n^2)$ to a minimum of $O(n)$. The FMM works by classifying nodes as local or remote to a source node depending on some predetermined kernel length. This allows contributions to the influence matrix from the nodes classed as “remote” to be grouped together thus reducing the computation time normally involved with evaluating every single node. The efficiency of a FMM for a BEM used to model free surface waves has been investigated by Yan *et al.* (2006). Furthermore, the method has been successfully employed to describe surface waves by Xue *et al.* (2001) and Guyenne & Grilli (2006), and for wave-body interactions by Liu *et al.* (2001). However, in considering this method it is clear that the reduced computational effort is achieved by potentially compromising the accuracy of the overall solution; the exact relationship between the accuracy achieved and the computational time taken being uncertain and problem dependent. Chaillat *et al.* (2008) offer some insight into the specific issues relating to the accuracy of the FMM. Whilst it is clear that high accuracy can be achieved using the FMM, it can only be realised at the cost of increased computational complexity. Further discussion regarding error bounds, accuracy and the choice of parameters in the FMM is presented by Darve (2001), Kybic *et al.* (2005) and Gumerov & Duraiswami (2009). However, there are some immediate advantages of using the FMM, perhaps most notably that the method lends itself well to a fast matrix-vector multiplication algorithm due to the hierarchical nature of the algorithm. The benefits of such a scheme are immediately apparent when implemented in indirect numerical linear system solvers, for example GMRES (Saad &

Schultz, 1986).

To avoid difficulties associated with the FMM, efforts have turned to the use of high performance computing (HPC) to assist with the solution of the problem whilst maintaining full accuracy for a known computational effort. As a result, there has been a rise in the use of distributed computing for increasing the speed at which BEM calculations run. This development is timely in two important respects. First, the availability of systems capable of running such code has risen. Second, with changes in design methodologies there is a widely acknowledged need to model the largest most nonlinear waves arising in realistic sea states; the success of which is critically dependent on the ability to solve very large computational domains. In the field of free surface wave mechanics most of the early attempts at parallelisation of the BEM algorithms focussed on improving the speed of the matrix solving phase of the problem, as this was the rate limiting part of the solution e.g. Kreienmeyer & Stein (1995) and Natarajan & Krishnaswamy (1995). However, improvements in matrix solvers and the need for larger computational domains has reversed this trend such that matrix formulation now tends to be the rate limiting factor.

The method chosen to improve the run times within the present study was a novel matrix decomposition method which distributes the computational effort of matrix formation over a number of processes in a distributed computing environment. This allows a large number of higher order elements to be used to more accurately model larger, high resolution domains. As far as the authors are aware this is the first time such a method has been used in a BEM model appropriate to surface water waves.

3.5 Computational profiling

To establish where to focus the parallel programming effort, a serial version of a BEM implementation developed by Hague & Swan (2009) was compiled using GNU gfortran with the $-pg$ switch, to enable code profiling, and as many speed related optimisations as possible. The result of this compilation is an executable which contains profiling symbols such that when it is executed it writes a run-time profile to file. The file can then be parsed by the GNU `gprof` tool to create a call graph and extensive code profile table. The call graph and profile table display information about the number of times a routine is called and how much time is spent computing each routine.

Key information from the profiling of the serial version of the BEM is provided in Table 3.1. Intrinsic system calls have been left out of the table as they are an unavoidable consequence of programming. The numerical scheme comprises two main phases, the first, matrix formulation, and the second, matrix solution. The routine `ghmatpq()`, is used to form the matrix from the integrated elements. The `ghmatpq()` routine calls either `extinpq()`, which integrates elements external to the current node, or `locinpq()`, which integrates elements local to the current node. In addition to the matrix forming routines, `main()`, the entry point to the code, and `gmres()`, the matrix solving routine, also merit mentioning as they take small but significant amounts of run time. With the profiling complete, the integration routines forming the influence matrix were shown to take by far the most time per calculation step. Accordingly, these routines will be the focus of reducing the computational run time.

Routine Name	**% Total Time**	**Number of Calls**
`extinpq()`	85.5	17784
`main()`	1.1	1
`locinpq()`	8.7	1021
`ghmatpq()`	3.1	1
`gmres()`	1.6	1

Table 3.1: Profile summary for serial BEM code.

Within the BEM algorithm, repeated spatial integration is a core activity and, not surprisingly, one that accounts for much of the computational effort. Given the nature of the boundary integral equation, the matrix is built of contributions from every node being integrated with respect to all the nodal points; hence the $O(n^2)$ computational effort. To perform these integrations a Gaussian quadrature scheme is adopted using a $\tilde{k}$ by $\tilde{m}$ grid of integration points in the two intrinsic directions across each element. In terms of the integration, the computational effort required will be $O(\tilde{k} \cdot \tilde{m})$. Therefore, there is the potential for both the requirement of high resolution and ill conditioned integrals to occur in the same problem requiring $O(n^2) \cdot O(\tilde{k} \cdot \tilde{m})$ computational effort.

Finally, it should be noted in Table 3.1 that the general minimised residual matrix solver, `gmres()`, proposed by Saad & Schultz (1986), only takes a small percentage of the total computation time. However, it should be anticipated that this value will rapidly increase with the matrix size and with poor matrix conditioning.

3.6 Developing a fast algorithm

To develop a fast algorithm it is necessary to decide on what level the majority of the computation should take place; at *compute node level* or at *interconnect level.* Memory access times can be used to rapidly decide which scheme would be the most efficient; remembering that a processor can only do work once it has fetched data from memory.

The memory on each computer (compute node) in the distributed environment can be accessed very quickly by the processors local to that specific compute node. The distributed memory can be accessed by all machines but the access is slower as it is achieved via an interconnect. However, this memory has a far greater capacity than on any individual compute node.

Using this information it is logical to optimise the code so that as much computation as possible takes place at compute node level and to only communicate between compute nodes when absolutely necessary. This reduces distributed environment overheads and prevents data bottle-necks at the interconnects.

3.7 The BEM algorithm

It has already been noted that the largest effort in terms of parallelising the algorithm should be concentrated on the integration scheme. A serial version of

Listing 3.1: A serial version of the integration scheme indicating the nested loops.

```
foreach node
  foreach node within each element
    if(node from inner loop is in an element
    to which a node on the outer loop belongs)
      call locinpq()
    else
      call extinpq()
    end
    push results of integration to
    influence matrix
  end
end
```

this scheme is given for reference in Listing 3.1.

It can be seen that there are two loops, one nested in the other, both looping in the order of n times, hence an $O(n^2)$ computational effort (see §3.5). The outer loop can be distributed in an *embarrassingly parallel* fashion due to the complete independence of the results in one part of the integration system to all others; in effect the algorithm displays a high level of exploitable concurrency. Thus, an algorithm could be made to distribute the outer loop among multiple processes (process in the sense of a UNIX process) within a distributed computing environment. It is the *embarrassingly parallel* nature of the integration scheme that lends its name to the present project: Embarrassingly Parallel Integration Code for Boundary Element Methods (EPIC_BEM).

A general boundary element scheme involves the integration of each node with respect to all other nodes to form two influence matrices **G** and **H**; the former based on the free space Green's function $G(r)$, which is the fundamental solution to Laplace's equation, and the latter its normal derivative $\frac{\partial G(r)}{\partial n}$. The system formed by matrix **H** multiplied by potentials $\boldsymbol{\phi}$ and matrix **G** multiplied by the potential fluxes $\boldsymbol{\phi_n}$ essentially defines the boundary integral equation (BIE) and can be manipulated to create the well known system of the form, $Ax = b$. This new system can then solved to give the unknown potentials and potential fluxes that were originally present in the vectors $\boldsymbol{\phi}$ and $\boldsymbol{\phi_n}$ respectively; the detail of how this is achieved in a multiple-flux formulation being provided by Hague & Swan (2009). Referring to Figure 3.1, a theoretically simple one dimensional block cyclic decomposition can be used to distribute the n nodes of the outer loop, in the serial algorithm given above, onto P processes. Each process would then be able to compute the integration for each of its $m = \frac{n}{P}$ nodes forming m rows of the influence matrices.

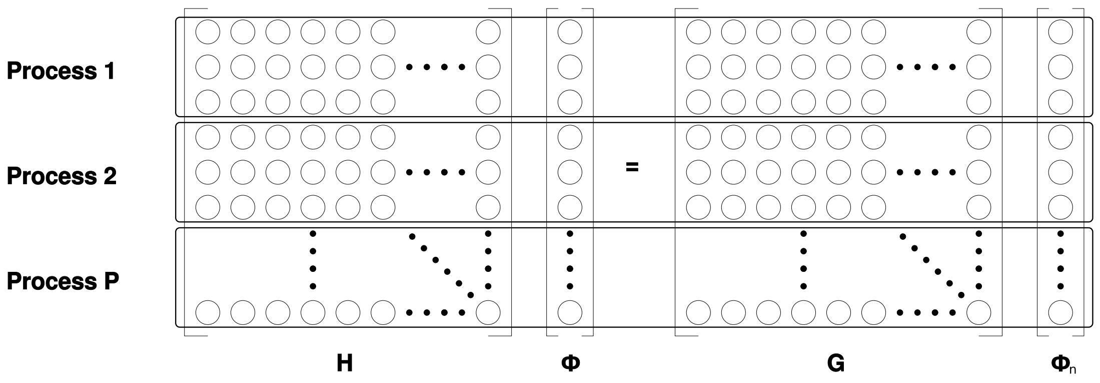


Figure 3.1: A schematic of the G-H matrix decomposition.

Adopting this approach, the computation time, ignoring distributed environment overheads, should decrease in proportion to the number of processes employed so long as each process runs on a separate processor, or, in the case of a processor with multiple cores, each process runs on a single processing core. An example of a *master-slave* algorithm achieving this approach is given in Listing 3.2.

Although this algorithm is simply expressed on paper, programming it using a parallel library such as MPI is nevertheless challenging. Difficulties arise when the n nodes are not divisible by the number of processes available. Similarly, if the slave processes do not have sufficient memory to hold the information to compute the integration, or the result of the integration, then the algorithm requires significant change. Along side these difficulties lie the standard problems with distributed computing such as *dead locking* and *race conditions*, both of which require careful planning to avoid.

Listing 3.2: Master-slave algorithm outline for parallel matrix formation.

```
# A master slave algorithm is shown where by one
# process becomes the ``master'' and distributes
# instructions to the ``slave'' processes.
if(process==master)
  broadcast all information required to complete the
  integration to slave processes. This includes coordinates,
  boundary conditions, and element index information
      foreach slave process
        receive results from slaves for
        integration of m nodes,
        push results of integration to
        influence matrix
      end

else #process is a slave
  receive broadcast of information from master,
  compute which ``m'' nodes should be calculated
  by this slave

  foreach m nodes
    foreach node within each element
      if(node from inner loop is in an element
      to which a node on the outer loop belongs)
        call locinpq()
      else
        call extinpq()
      end
      store integration results locally with
      reference to ``m''
    end
  end
  send results of ``m'' nodes to master
end
```

	Case 1	Case 2	Case 3	Case 4
X-dimension (m)	$0.0 \rightarrow 3.0$	$0.0 \rightarrow 4.0$	$0.0 \rightarrow 6.0$	$0.0 \rightarrow 8.0$
Y-dimension (m)	$-1.0 \rightarrow 0.0$	$-2.0 \rightarrow 0.0$	$-3.0 \rightarrow 0.0$	$-5.0 \rightarrow 0.0$
Z-dimension (m)	$-0.8 \rightarrow 0.0$	$-0.8 \rightarrow 0.0$	$-0.8 \rightarrow 0.0$	$-0.8 \rightarrow 0.0$
Nodes	4962	10242	20162	40322

Table 3.2: Dimensions used to generate the numerical wave tanks.

3.8 The test bed

The boundary element code was applied to a numerical wave tank scheme; the purpose of the computations being to calculate the propagation of regular waves over a horizontal bed. Nine node Lagrange elements were used with a discretisation of 0.05m in all directions. The shape of the tank was cuboid, with a geometry chosen to give approximate problem sizes of 5000, 10000, 20000 and 40000 nodes in a domain; hereafter referred to as Cases 1-4 respectively. Table 3.2 gives the exact geometries used for the domains, each domain having a fixed depth of 0.8m and the two horizontal dimensions altered to give the required number of nodes. The wave simulation involved a regular wave input using a Stokes 5^{th} order analytical solution with a wave number of 4.0369 $(rads \cdot m^{-1})$ and a wave height of $0.02m$. An open tank condition was applied at the downstream end of the numerical domain using a radiation boundary based on Sommerfeld (1949) and the Courant number defining the time step was set to 0.4 for all simulations.

Six integration points were used for each element on the grid such that in the previously used $O(\tilde{k} \cdot \tilde{m})$ notation, $\tilde{k} = \tilde{m} = 6$. For each problem size the solution was time marched in its respective domain for 103 time steps. Given the nature of the information required for the time-marching, the first three time steps were undertaken using a fifth-order Runge-Kutta integration scheme to kick-start the calculation, followed by 100 steps of an Adams-Bashforth-Moulton predictor-corrector scheme; the latter being more efficient given the required accuracy. Further details of the time stepping procedures are given in Hague & Swan (2009).

The high performance computing (HPC) cluster at Imperial College London was used for the speed testing. The cluster comprises a number of compute blades linked by a gigabit ethernet interconnect. A single job on the HPC system can use a maximum of 16 blades, each blade having 4 Xeon processor cores and 16Gb RAM. Thus the largest computation environment encompasses 64 processors and 256Gb RAM. In the interests of fair testing, far more RAM was requested in each job than needed by the tests to ensure that the use of virtual memory did not skew the results. The code was run for each problem size on 1, 4, 8, 16, 32, and 64 processors; in all cases only one MPI process was run on a given processor at a particular instance in time.

3.9 Discussion of results

The results from the proposed scheme were extremely promising. The most notable result being that for large problem sizes (n large) the computation time is limited by the `gmres()` routine that solves the influence matrix to obtain the solution.

Figure 3.2 shows the average computation time for one Adams-Bashforth-Moulton time step when running the code on 1, 4, 8, 16, 32 and 64 processors for Case 3 (20000 nodes). This figure illustrates that the time spent in serial operations and performing the `gmres()` routine is, as expected, independent of the number of processes used as they are undertaken as serial operations.

The speed up of the `ghmat()` routine as more processes are added to the parallel environment is quantified in Table 3.3. This indicates that the `ghmat()` scales well from 4-16 processors for all but the largest case (Case 4). However, the rate of increase of computational speed decreases with the addition of further processes; the benefit of 32 and 64 processes being progressively marginal. The most likely explanation for this lies in the development of bottle-necks on the interconnect. This slow down can also be seen in the case with the largest number of nodes, Case 4. Once again this is believed to be due to bottle-necks within the HPC system architecture.

Table 3.4 concerns the speed up of the computation time for the entire EPIC_BEM scheme. This table indicates that the use of a relatively inexpensive high end workstation with 8 processing cores would compute BEM calculations approximately

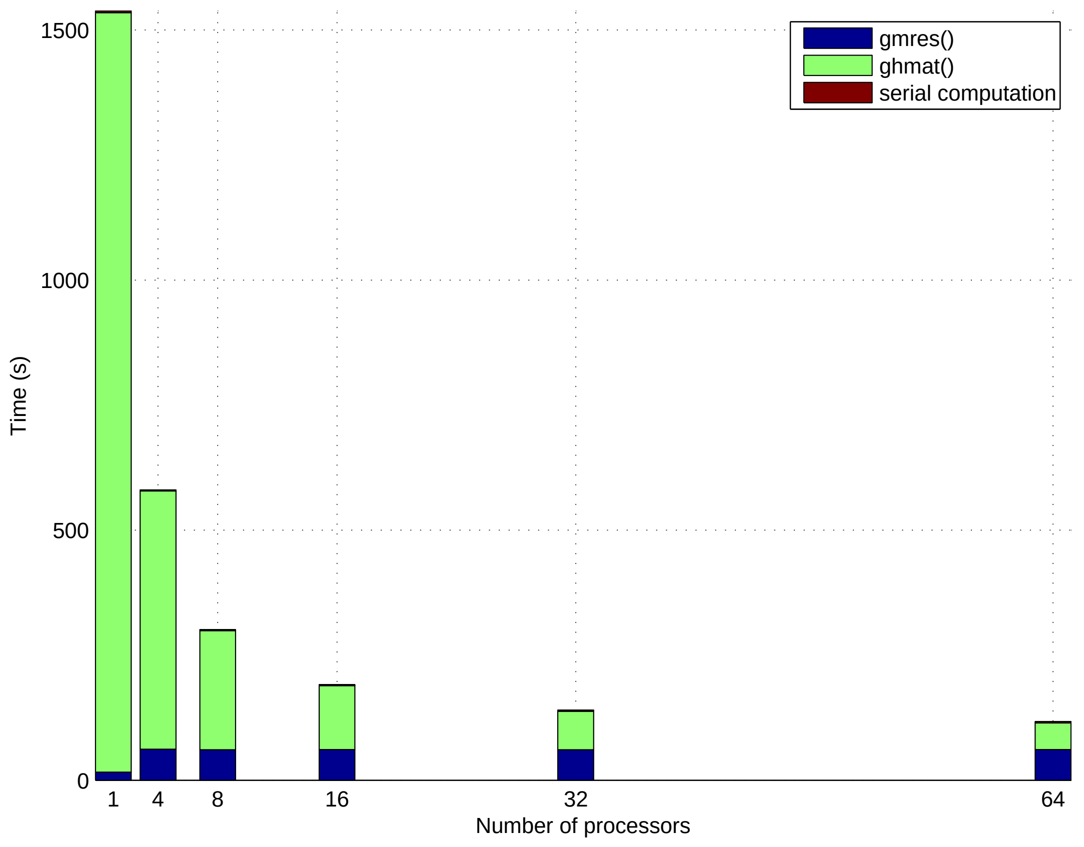


Figure 3.2: Average computation times for one time step for 1, 4, 8, 16, 32 and 64 processors solving a problem of size 20000 nodes. Note: the time occupied undertaking the serial calculation is very small and corresponds to the line at the top of each bar.

Processes	4	8	16	32	64
Case 1	1	2.16926	4.06822	6.6744	9.39565
Case 2	1	2.17013	4.05827	6.69643	9.69873
Case 3	1	2.16884	4.04839	6.70988	9.7109
Case 4	-	1	1.86879	3.10181	4.48771

Table 3.3: Increase in the computational speed of the `ghmat()` routine defined in terms of the average computation time for one time step normalised with respect to that achieved using 4 processors. (Note: in Case 4 the speed is normalised relative to that achieved with 8 processes; the reason for this being explained below.)

five times faster than a serial BEM code. The table also demonstrates that it is possible to reduce the run time of a very large problem by more than an order of magnitude with a suitable algorithm and sufficient processors.

Processes	1	4	8	16	32	64
Case 1	1	2.81477	5.77145	9.77199	14.4517	18.0688
Case 2	1	2.74463	5.44375	8.92724	12.6888	15.7050
Case 3	1	2.64956	5.10663	8.03879	10.9712	13.1195
Case 4	1	-	4.82355	7.40905	9.8004	11.6004

Table 3.4: Increase in computational speed for the BEM scheme as a whole in comparison to a 1 process "serial" version.

In considering the data presented in Table 3.3 and Table 3.4, two problems arise relating to the implementation of Case 4 involving the largest computational domain (n=40000 nodes). The first problem arises on both Table 3.3 and Table 3.4 and concerns the distributed implementation of the code on 4 processors on the same blade. In this case, there is simply insufficient memory to cope with the amount of data shuffling required to deal with a fully allocated $n \times n$ matrix storage area for the master process and three $n \times m$ matrices as required by the slave processes. For this reason the implementation of Case 4 on 4 processors does not feature in either of these tables and is missing from the subsequent figures.

The second problem concerns the implementation of Case 4 using 1 serial process. In this case the speed of the computation was such that it did not complete the 103 calculation steps within the three day run-time limit imposed on the Imperial College London HPC system. To overcome this difficulty the Case 4 ratios presented on Table 3.4 were based upon an average time step calculated over the 38 steps that were completed within the three day limit.

In most parallel implementations of an algorithm there are significant computational overheads associated with invoking a parallel environment. This means that the scaling from 1 to 4 processes will not lead to a four times speed up in the parallel section of the code. However, in the present case, these overheads require approximately the same computational effort for four processes as for x processes, where $x > 2$. It therefore follows that in the case of the `ghmat()` algorithm, once the decision to use a parallel environment has been made, adding more processes to the computation environment simply speeds up the step time almost in proportion to the number of processes added. This result demonstrates that the slowest part of a boundary element computation, the influence matrix formulation, scales very well in a parallel environment. This is demonstrated in Figure 3.3 in which the average time step for four or more processors is compared with the perfectly proportional scaling.

In §3.4 it was noted that the fast multipole method (FMM) is commonly adopted to speed up BEM calculations. Figure 3.4 contrasts the overall scaling of the EPIC_BEM scheme with the theoretical speed of an FMM algorithm. Within this figure both axes have been non-dimensionalised: the x-axis representing the problem size, expressing the number of nodes as a ratio of those in Case 1, and the y-axis the average time step expressed as a ratio of the value achieved in a "serial" implementation of case 1. In respect of the FMM solution two indicative lines are drawn, $O(n)$ based on the findings of Greengard (1988) and $O(n \cdot log(n))$ based on the those of Barnes & Hut (1986). Although these lines are not absolute they give an idea of the scaling properties of two of the most popular FMM algorithms (assuming that for Case 1 the FMM algorithms take the same average step time as the serial code for Case 1). The $O(n^2)$ line is also included on Figure 3.4 allowing comparisons with a standard "serial" BEM algorithm.

Comparisons between these theoretical limits and the EPIC_BEM results indicate that the use of 16 processors, with one process per processor, gives an equivalent run-time to that of even the fastest serial FMM codes. Should faster run-times be required, simply adding more processors to the environment significantly reduces the run time. For example, with 64 processors the EPIC_BEM code is more than 40% faster than the FMM limit assuming they have the same run time characteristics for Case 1 in serial. Most importantly, this enhanced computational speed can be achieved without loss of accuracy.

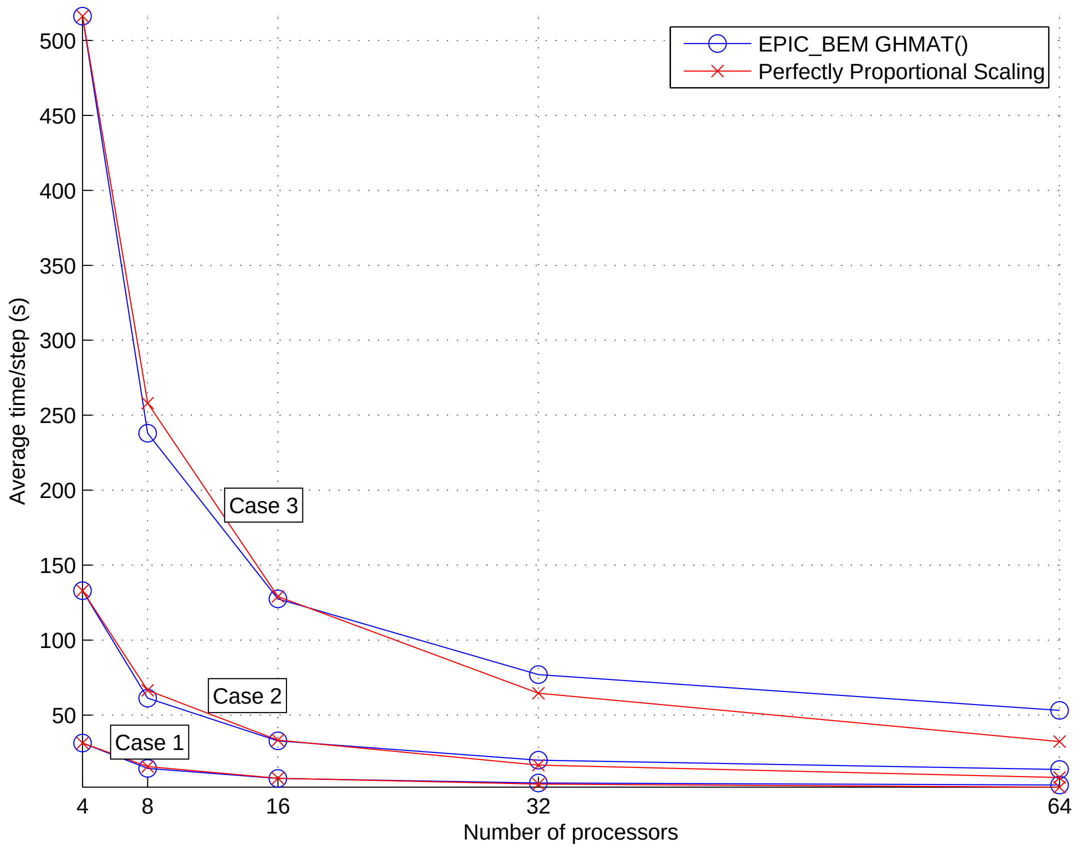


Figure 3.3: Demonstrating the scalability of the core integration algorithm.

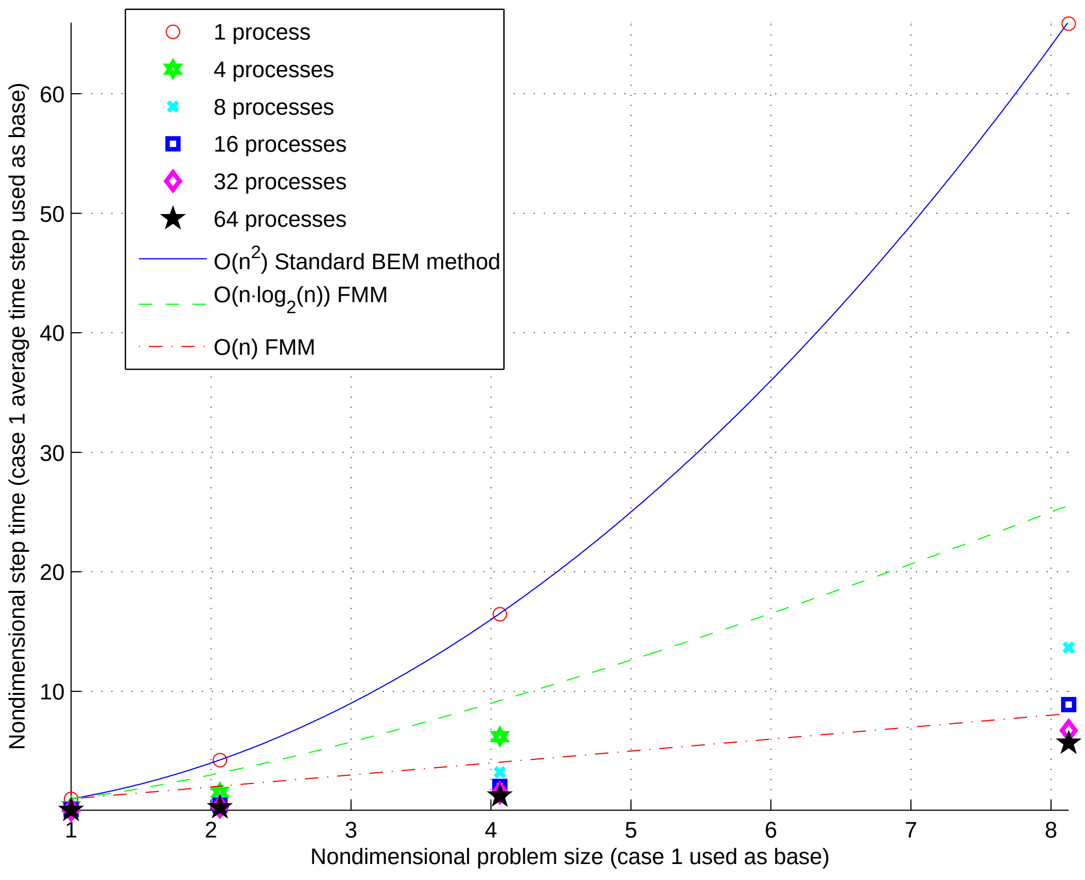


Figure 3.4: Non-dimensional plot of problem size against step size using 1, 4, 8, 16, 32 and 64 processors with comparisons to the FMM.

3.10 Further considerations

In considering the results presented above, it has already been noted that the scaling from 1 to 4 processes is poor due to the inevitable parallel overheads. However, there is another factor due to the HPC environment on which the code ran that exaggerates this behaviour and leads to an underestimate of the benefit of the distributed algorithm. When the code was compiled for running on the HPC cluster, the `-parallel` switch on the Intel compiler was used to request that as many OpenMP type instructions as possible were built into the final executable. The cluster blades each have 4 processing cores and any idle time on a core will, where possible, be taken up processing OpenMP type instructions from the other processes' workloads. This is very important when seeking to run the code on a single processor using a "serial" algorithm. In this case there are three additional processors on the blade that are part of the job allocation. This means that although the serial code is running in "serial", there are 3 idle processors on the blade performing any OpenMP type instructions in the code and running the operating system load. This would not be the case in the "parallel" implementation as none of the processors on a blade would be idle and therefore able to help with the workload of others. Unfortunately there are no available single processing core blades that are identical (in all other respects) to the blades on the cluster. As a result, the serial code undoubtedly has a faster run time than it would in a true single processor environment. Consequently the data presented on Table 3.4 and Figure 3.4 are conservative in terms of the speed increases achieved.

With the code designed to run primarily on homogeneous distributed memory cluster computers, the master-slave model inevitably leads to data bottle-necks. Additionally, as the slave process undertake near-identical computations, it is likely they will finish their tasks at approximately the same time. As a result they will all wish to communicate with the master process simultaneously; the latter taking place over the interconnect. In undertaking some initial calculations, it was found that a 1Gb backbone ethernet interconnect was detrimentally flooded with communication; the problem becoming exacerbated with increases in the size of the computational domain (large n). Nevertheless, the effects of data transfer bottle-necks can only be seen in the larger domain sizes, especially when using large numbers of processes. As the domain sizes increase, continued problems are anticipated.

Another problem that will inevitably arise in the near future is that boundary element codes will reach and exceed physical memory limits. The influence matrix scales as $O(n^2)$ such that 16Gb of RAM (16×2^{30}) can hold $\sim 2.14 \times 10^9$ numbers (assuming 4 bytes per float). Given the nature of the problem this also approximately represents the square of the maximum number of nodes, n, defining a problem. To avoid such limits it is possible to adopt distributed storage of the matrix as in Cunha *et al.* (2002). However, this would cause the parallel solution of the problem to become more *fine grained* and therefore communication intensive at the interconnect level. This, in turn, would lead to a significant increase in the complexity and run-time of the problem.

3.11 Concluding remarks

This paper demonstrates that a simple matrix decomposition method is suitable for the parallelisation of a multiple-flux BEM. The method is shown to scale well across large numbers of processes with the added advantage that it can be run on any machine that supports an MPI environment. Using this approach it is possible to achieve computation speeds comparable to that of a BEM code employing a fast multipole method. Most importantly, these speeds can be achieved without (potentially) having to compromise the accuracy of the solution and without introducing uncertainties in the accuracy-computation time relationship.

Acknowledgements

The authors gratefully acknowledge the financial support provided by the UK Engineering and Physical Sciences Research Council (EPSRC) under grant number F022964/1, and the High Performance Computing centre at Imperial College London.

3.12 HPC systems

To assist the development of the EPIC_BEM code, the author designed and built a small (∼12 computational nodes) cluster computer in the fluids laboratory in the Dept. Civil and Environmental Engineering at Imperial College London. The cluster computer was made of compute nodes that were not identical in processor power or memory, hence the cluster computer had a heterogeneous architecture. Having a private cluster computer was beneficial in the development of the algorithms and ideas found in §3.3–§3.11 as there was no queue for resources, the software layer on which parallel algorithms ran could be tuned and modified at will and any code written for a heterogeneous machine required more thought with respect to aspects such as load balancing and network latency. The latency of the network being rather an important issue as it forms part of the serial workload as seen in Figure 3.2 and therefore affects the maximum speed increase obtainable.

3.13 Conclusions

It is apparent from §3.3–§3.11 that implementing the influence matrix construction algorithm `ghmatpq()` to make use of a distributed computing environment is conceptually sound and works well in terms of performance gain. The run time of the construction phase is reduced sufficiently such that realisticly sized, high resolution computational domains can now be computed within a reasonable time frame.

There is, however, at least one further optimisation that could be made to the algorithm that would help address the issue of a network bottle-neck in the gathering phase of the algorithm when the parts of the influence matrices are pooled on the master node. In a homogeneous distributed environment it is likely, given that the decomposition of the work is one dimensional, that the compute nodes will all have approximately the same run time for the work performed. Therefore the gathering of the distributed matrix by the master node will occur almost simultaneously across all the nodes, hence the network connection of the master node becomes a bottle-neck for data transfer. There are two ways around such a problem. The first is to use a lower latency network interconnect such as Infiniband (InfiniBand Trade Association, 2000). The second, is to use a "guided" (as in a guided schedule in OpenMP (Chapman *et al.*, 2007)) decomposition of the workload. Imperial College London's HPC service has an Infiniband interconnect on some parts of the system and so the first option has been tested. As expected, additional performance is gained from using a higher performance connection. The second option has not been tested as it requires a lot of computational book keeping and overhead. In addition, there are other areas of greater concern with respect to computational run time such that the small lag caused by network bottle-necks is not considered the most pressing issue. These additional areas of concern are addressed in the next chapter.

4

Matrix Solving on GPUs

4.1 Introduction to matrix solving

It has been shown in Chapter 3 that the distribution of the influence matrix formation over multiple processes drastically reduces the run time of the code. Indeed, the wall time of the formation stage is reduced so much that the time taken to solve the matrix system can, in the worst cases, take up to approximately 65% of the total time per step. This result is due to the fact that the influence matrix condition generally worsens with time and therefore the results differ from those presented in Chapter 3; the latter considering the first 100 steps of a calculation when the matrix is better conditioned.

In the present EPIC_BEM formulation, the matrix solving algorithm only makes use of one compute node (a shared memory programming limitation). This means that during the solution time the other nodes in the distributed computing environment are idling. Consequently, the distributed computing environment is only being used approximately one third of the time, and this is clearly a poor use of computing power.

In the interests of addressing the problem of idle compute nodes during the solution phase of the algorithm, described in §2.5, there are a number of options:

a) Use the idling processing cores on the compute node designated to matrix solving to assist with the computation.

b) Use the idling processing cores on the distributed compute nodes to assist with the computation.

c) Accelerate the matrix solving phase in some way such that the solution time becomes insignificant. As a result, the idle time will become negligible.

In both Chapters 2 and 3 it was noted that the matrix solver used to obtain a solution to the system formed by the influence matrices, $\mathbf{H} \cdot \Phi = \mathbf{G} \cdot \Phi_n$, is an implementation of the generalised minimal residual method (GMRES) first proposed by Saad & Schultz (1986). The original implementation of the GMRES code used for this work was written by Christou *et al.* (2008). However, the implementation in current use is a version optimised for multi-core processors, via the addition of OpenMP instructions, by the author. The OpenMP instructions speed up the solver by making use of additional processing cores within a compute node to increase the performance of BLAS (Basic Linear Algebra Subprograms, (Lawson *et al.*, 1979)) type operations and to distribute code loops. The performance gain using OpenMP instructions is, however, limited as there are numerous medium/fine grained operations (e.g. Jacobi rotations) within the GMRES algorithm, a finite bandwidth on the bus transporting data to the processors and significant overheads associated with a OpenMP environment.

With respect to points a)–c) above, the purpose being to enable a better use of computing power during the matrix solving phase, the OpenMP enabled GMRES algorithm addresses the first idea of using the idling processing cores within the compute node on which the matrix is solved. Whilst there is some performance gain using this method over a serial algorithm running on a single processing core, the gain is not particularly large; full details of the comparison are provided in §4.7.

The second idea of using the idling distributed environment to help solve the matrix system may, in principle, seem worth investigating as there is potentially a lot of processing power available. However, the distributed compute nodes are harnessed together via a switched network and communicate via a TCP/IP layer. This means that all the latencies and lag involved in communicating data would slow any proposed algorithm down. This problem is commented on in the ScaLA-

PACK manual (Blackford *et al.* (1997)) and is generally tackled in their code via extensive tuning. In addition, Vardon *et al.* (2009) considers using a hybrid (distributed with MPI and threaded with OpenMP) parallel conjugate gradient algorithm for solving sparse matrices resulting from finite element calculations. The conclusion drawn from this work is that an approximately six times speed up can be gained for very large problems. However, this requires a very expensive Infiniband interconnect fabric to link the processing nodes and most of the gain in speed was achieved within the addition of the first 7 processing cores.

Following a careful investigation by the author, it was revealed that the size of the system to be solved in the present boundary element formulations is awkward in the sense that it is at the upper limit of what a single compute node can solve in a reasonable time. Compounding the awkwardness, the system size is generally too small to make distributed solving worth while, especially if no Infiniband fabric is available. The second idea, b), was therefore disregarded.

Recently, a new architecture called Compute Unified Device Architecture (now depreciated and simply referred to as "CUDA") was brought into the realm of commonly accessible scientific computing by engineers at NVIDIA Ltd. The CUDA architecture takes the physical form of a specialised graphics processing unit (GPU) accessed via a standard PCIe bus. The architecture uses a method akin to vector processing called single instruction multiple thread (SIMT) and can be described as "many-core" with each core being able to run thousands of threads simultaneously.

There has been a lot of interest in the many-core computing scene with important contributions from NVIDIA's CUDA (NVIDIA, 2009), ATI's CTM (Advanced Micro Devices Inc, 2006) now depreciated in favour of OpenCL (Khronos OpenCL Working Group, 2009), Sony/Toshiba/IBM Cell BE architecture (Hofstee, 2005), the CSX SIMD array architecture from ClearSpeed (ClearSpeed Technology, 2001-2009), and the yet to be released Larrabee architecture from Intel (Seiler *et al.*, 2008). NVIDIA appear to have clinched the early market by releasing a relatively easy to use application programming interface (API) that allows `C/C++` code to be written with some extra instructions to permit the execution of code on a specialised GPU running with many thousands of threads. With several research groups starting to publish promising results (NVIDIA, 2009), it was decided that it was worth investing the time and money in some GPU hardware; the purpose of the work being to substantially reduce the wall time of the matrix solution phase corresponding to point c) above.

4.2 Using CUDA architecture

To be able to write efficient software to run on any hardware, some knowledge of the hardware and its general programming/operation is required. An introduction to the CUDA architecture and programming for NVIDIA's CUDA enabled GPUs is provided in the next sub-section.

4.2.1 CUDA hardware

The CUDA hardware comes in the form of a, habitually called, graphics processing unit and is attached to a standard personal computer either via the PCIe bus directly or via a PCIe data cable attaching an external GPU to the PCIe bus. The CUDA enabled GPUs range from video cards that can be bought "off-the-shelf" relatively cheaply in most PC stores, through to rack mounted multiple GPU enabled units. The latter being, perhaps, of greater interest to the scientific community.

As most generic graphics hardware is used simply to offload the task of graphics processing from the CPU, acting as a co-processor, the build process and electrical design of these GPUs does not require them to be able to deal with general purpose computing. The memory in these GPUs is not burned in and checked to ensure the full memory range behaves as expected, simply because the occasional off-colour pixel at a refresh rate of 60Hz is not going to be noticed. However, for use in general purpose computing the full range of memory available must be working correctly as the occasional floating point number not being manipulated or stored correctly could easily lead to the wrong result in a numerical simulation. As a result, NVIDIA released the professional 'Tesla' range of hardware that is guaranteed to have a fully working memory unit (via soak tests among others).

The work described in the section that follows was performed on these GPUs.

In order to optimise a program it is essential to have some understanding of the hardware on which the optimisation will take place. With this in mind the architecture of the CUDA GPUs is briefly explored with reference to the schematic of the architecture in Figure 4.1.

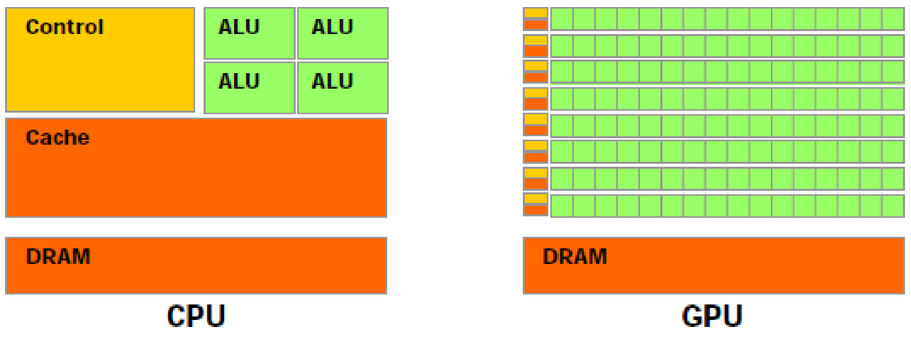


Figure 4.1: Schematic of the CUDA GPU architecture. Image taken from NVIDIA (2008*b*).

The first obvious difference between a CPU and GPU architecture is that the GPU assigns many more of its transistors (the electronic components that do the work) to arithmetic logic units (ALUs) for data processing. This fact leads to operations that are data parallel, and have a high arithmetic intensity, performing far better on a GPU as there are simply more transistors to deal with the computation. Each ALU can be dedicated to processing one data element, so fine grained parallel operations are well suited to this architecture. In direct relation to this, an ALU will deal with very few pieces of data at a time, and all the ALUs will be running the same set of instructions. Therefore, there is no need to dedicate a lot of processing power to flow control, this making the GPUs potentially even faster for a given transistor density. The other most notable difference between CPU and GPU hardware is the lack of a general cache on the GPU. This is because if the amount of data parallel computation is sufficiently large, the memory access latencies can be hidden by other data computations taking place whilst the data is being accessed.

The processing power of the CUDA GPU is arranged in the form of a number of "streaming multiprocessors" (SM) each of which contain (currently) 8 ALUs, some fast local memory in the form of "shared" memory, a texture cache, a constant cache and an instruction unit. In addition, the ALUs within the SMs all have a large register file to try and help reduce memory requests. The CUDA GPU also has a device memory (DRAM) available for general purpose storage. This memory is accessible to all SMs and so can be used for sharing information between them since no direct message passing is available between SMs. For more information see NVIDIA (2009).

4.2.2 CUDA programming model

The CUDA programming model revolves around the concept of exposing the data parallel processing nature of the GPU to the programmer. This is done via the concept of parallel processing threads; a thread being a particular computation path often resulting from a fork in program execution. To make the most use of the ALUs on the GPU hardware, many threads should be in existence ("in flight") simultaneously such that all the ALUs are busy all the time. As a result, any memory access latency is hidden by the threads queued for execution being run whilst other threads wait for data. The threads are organised into blocks of 32, called warps, and each warp must execute the same function on all its assigned ALUs simultaneously (instruction lock stepping). The thread scheduler requires four clock cycles to switch single instruction multiple data (SIMD) groups and only 192 threads are required to be in flight to cover pipeline latency, hence zero latency is easily achieved if sufficient arithmetic intensity is present and the data required for computation is available.

To aid the use of the CUDA enabled GPUs for general purpose computing, NVIDIA have extended the `C` language by adding some CUDA specific syntax and instructions. The most important of these extensions is the concept of "kernels" which are specialised `C` functions that when called execute "N" times in parallel on "N" different CUDA threads. This is in opposition to the calling of a standard function from a CPU which would only be executed once on one thread. Further to this, the programmer has the freedom to arrange the threads that are being executed into arbitrarily sized execution blocks within a grid of up to three dimensions. Clearly it is advantageous to arrange the thread blocks such that they align with the hardware warp sizes, hence thread blocks are commonly multiples of 32 in size.

The CUDA architecture and programming model are fascinating and there are many other features available in CUDA enabled hardware, especially memory related, but they are beyond the scope of the present work. Those features that have not yet been discussed, but which are used in the following work, will be briefly explained immediately prior to the description of their use.

4.3 CUDA Induced Dimensional Reduction Solver

As the BEM involves the solution of a linear system involving a dense matrix, and, in the field of accelerated computing, dense linear algebra is known to map well to multi-core architectures, using multi-core architectures to solve such problems may be advantageous. This work is to be considered as proof-of-concept work and is designed to test ideas that can be implemented in the future.

4.3.1 Appropriate solvers

In its original form the GMRES algorithm (Saad & Schultz, 1986), although having a number of excellent features, suffers from the fact that its memory footprint increases with the number of iterations performed during the solution. During each iteration a new basis in the Krylov subspace needs to be generated which is of the same size as the leading dimension of the system. At each iteration, each of these vectors needs to be stored and so, for a large number of iterations, the memory requirement can be substantial. The GMRES memory footprint can be forced to be of fixed size by restarting the solver with the basis of the last iteration as the (re)starting basis. However, these restarts are expensive in terms of iterations performed because useful, previously discovered, information is disregarded. More importantly, should the solving algorithm be restarted there is no guarantee that the new basis will actually allow the iterative system to converge on a solution without considerable additional work being undertaken (Joubert, 1994).

Recently, Sonneveld & van Gijzen (2008) have developed a new Krylov based gradient solver, called IDR(s), based on the concept of *induced dimensional reduction* (Wesseling & Sonneveld, 1980). This solver has the advantage of the user being able to specify the size of the solution space in which the residual shall be found and therefore has a fixed memory footprint. On first review it appears that the fine/medium grained looping that makes GMRES hard to speed up is less prevalent in this alternative method. In addition, the algorithm appears more simple to implement numerically than GMRES. Furthermore, although the properties of the solver have not been investigated to the same extent as those of GMRES, the initial results presented by Sonneveld & van Gijzen (2008) appear promising. On the basis of algorithmic simplicity and the apparent level of parallelism achievable in this algorithm, the IDR(s) method was chosen as the preferred algorithm to develop for the CUDA architecture.

4.3.2 IDR(s) theory

A complete and detailed explanation of the IDR(s) theorem, algorithm and proofs can be found in Sonneveld & van Gijzen (2008). In the context of the present study, a brief overview of the IDR(s) concept is given below. More detailed information relating the how the method works and further supporting commentary can be found in Appendix A and Appendix B respectively.

Basic concepts

To solve a linear system of equations of the form,

$$Ax = b, \tag{4.1}$$

an extensive repertoire of solvers is available. In the context of the boundary element method considered herein, this linear system is obtained by manipulating the $\mathbf{H} \cdot \Phi = \mathbf{G} \cdot \Phi_n$ BIE as described in Hague & Swan (2009). When solving this system of equations, the most efficient methods, in terms of the computational effort required, are perhaps the iterative solvers based on solutions found in the

Krylov subspace. The Krylov subspace is defined as

$$\mathcal{K}^n(A, r_0) = span(r_0, Ar_0, A^2r_0, ..., A^nr_0) \tag{4.2}$$

with n being the iteration number and an initial residual r_0 being defined as

$$r_0 = b - Ax_0, \tag{4.3}$$

where x_0 is the initial guess to the solution. Within equation (4.2), the *span* of a set Q, $span(Q)$, is the set of all linear combinations of elements of Q.

Until recently, the search for efficient Krylov based solvers has revolved around generalisations of the conjugate gradient method. These are categorised based on their recurrences; short recurrences meaning that only a small number of vectors are required to compute a solution. With the removal of the requirement of short recurrences, methods such as GMRES are available. However, this method suffers from a large memory footprint, as discussed earlier. If the requirement of short recurrences is kept, a large family of modified conjugate gradient algorithms with various efficiencies and features is available. For an extensive review of these iterative methods, including those based on conjugate gradients, the reader is directed to Saad & Vorst (2000).

The IDR method was originally developed by Wesseling & Sonneveld (1980), but was largely ignored in favour of the conjugate gradient based solvers. The IDR method has short recurrences and is guaranteed to compute an exact solution in at most $2N$ steps, N being the system size. The basic concept of the IDR method is that it generates residuals that are forced to be in subspaces of $\mathcal{G}_j$ of decreasing dimension. These subspaces are nested and are related as given in equation (4.4). An indication as to how this relation arises is given in Appendix B.1.

The IDR theorem, outlined below, follows directly the approach described by Sonneveld & van Gijzen (2008), and begins as follows: Let $\mathbf{A}$ be any matrix in $\mathbb{C}^{N\times N}$, let v_0 be any non-zero vector in $\mathbb{C}^N$, and let $\mathcal{G}_0$ be the full Krylov space $\mathcal{K}^N(\mathbf{A}, v_0)$. Let $\mathcal{S}$ denote any (proper) subspace of $\mathbb{C}^N$ such that $\mathcal{S}$ and $\mathcal{G}_0$ do not share a nontrivial invariant subspace of $\mathbf{A}$, and define the sequence $\mathcal{G}_j, j = 1, 2, ...,$ as

$$\mathcal{G}_j = (I - w_j\mathbf{A})(\mathcal{S} \cap \mathcal{G}_{j-1}). \tag{4.4}$$

Within this equation, I is the identity matrix, ω_j are weightings and $\mathcal{S}$ is a fixed proper subspace of $\mathbb{C}^N$. This relationship can be used to provide a recursion between successive iterated residuals, and therefore successive iterated solutions. In the paragraphs that follow an overview of the IDR(s) scheme is provided. This is sufficient to provide a basic understanding of the method adopted and to understand its place within the current numerical scheme. However, for those readers who require additional information, on both the background and the implementation of the IDR(s) method, this is provided in Appendices A and B.

On the basis of equation (4.4) an iterative procedure can be formed, the essential components of which are as follows:

a) The procedure is appropriate to the solution of a set of linear simultaneous equations in the for $Ax = b$.

b) It involves the calculation of a system of successive residuals of the form $r_n = b - Ax_n$.

c) It seeks to provide a recursion relation for the residuals, (r_n), which will in turn provide a recursion for the desired solution x.

d) The residuals must lie in a subspace $\mathcal{G}_j$ of reducing dimension and must intersect with a chosen subspace $\mathcal{S}$ (which is of dimension s by N, where s is user defined, see Appendix A). The fact that the intersection is also of reducing size leads to the convergence of the solution. A schematic indicating the subspace nesting appropriate to the IDR theorem is given in Figure 4.2.

e) Within the chosen scheme the residual at the $n + 1^{th}$ iteration, r_{n+1}, can be formed from the previous iteration (r_n) coupled with information concerning the change in the residual (Δr_{n+1}) from prior iterations.

f) By limiting the information used to the most recent iterations, the depth of the recursion can be maintained in successive iterations thereby limiting the memory (or storage) required.

g) When initiating the calculation an iteration similar to the modified Richardson iteration (see Appendix B.1) is employed to provide sufficient information to allow the computation of the residual differences, the latter being key to the scheme.

h) In calculating the next iteration of the residual, a weighting function (ω_j) is introduced to optimise the reduction of the residual. This is achieved by minimising the 2-norm of the residual.

i) On completion of an iteration, calculation of r_{n+1} and hence the updating of x_{n+1} is undertaken.

j) Assuming the residual remains larger than some target threshold, the change in the residual (Δr_n) is calculated and the procedure repeated. The calculation of Δr_n is based upon information from the most recent changes in residual, the extent of this information being based on the chosen depth of recursion.

Further information concerning the IDR(s) method, including the prototype algorithm with full explanation and supporting material, is provided in Appendices A and B.

Optimised IDR(s) algorithm

In their original paper, Sonneveld & van Gijzen (2008) gave a perfectly acceptable algorithm using the IDR(s) method (Algorithm 1, in Appendix A). However, they note that this algorithm is a pure translation of the mathematics. They also state that there is considerable freedom available with regard to the choice of matrix P (which describes the subspace S), the way in which the weightings (w_{j+1}) are selected, and how the intermediate residuals are computed. In a follow up paper van Gijzen & Sonneveld (2008) exploited this flexibility and altered the manner in which the iteration vectors were formed, using bi-orthogonalisation properties between P and ΔR; ΔR containing successive residual differences. This later method is slightly more stable, more accurate and, most importantly, has a lower vector operation count than the directly translated algorithm. The actual mathematics used in the later method is beyond the scope of the present work and so is not presented. Nevertheless, it is a fascinating paper emphasising the

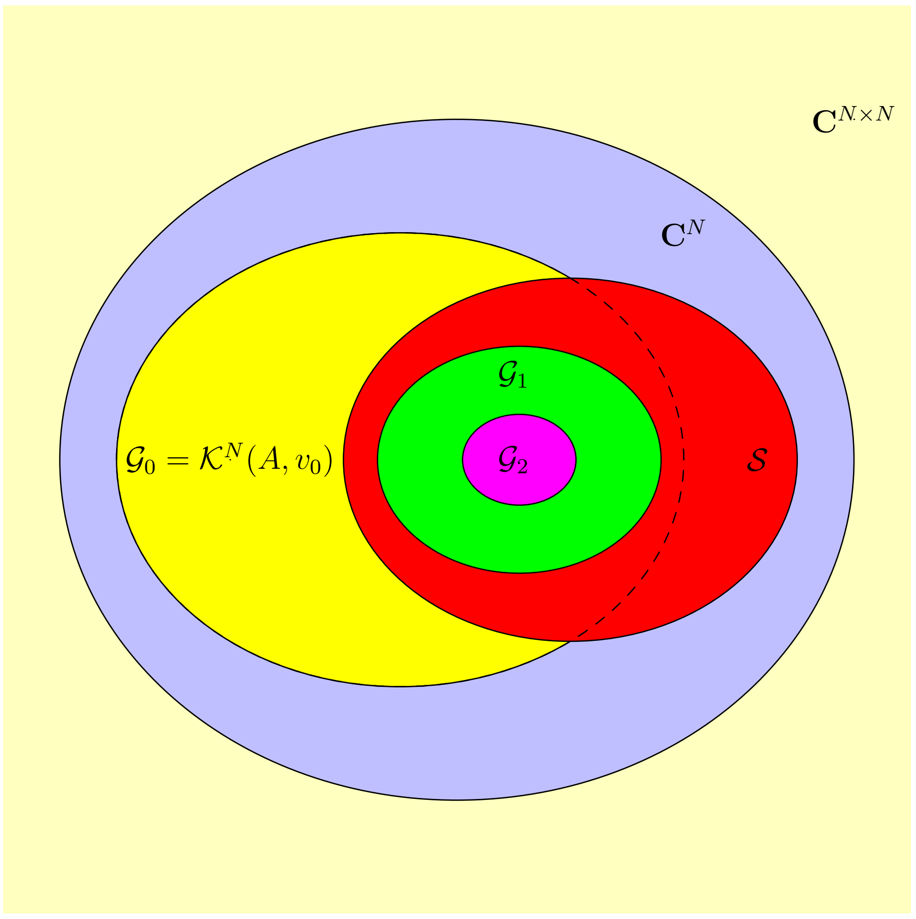

$\mathbf{C}^{N \times N}$
$\mathbf{C}^{N}$
$\mathcal{G}_1$
$\mathcal{G}_0 = \mathcal{K}^N(A, v_0)$
$\mathcal{G}_2$
$\mathcal{S}$

Figure 4.2: Schematic of the subspace nesting key to the IDR theorem.

freedom available in the IDR(s) framework. For reasons of numerical stability and efficiency, it is this later algorithm that is employed in the present work.

4.3.3 Preconditioners

To improve the rate of convergence of gradient based solvers it is common to apply a (matrix) preconditioner, M, to the system matrix in an attempt to improve the amenability of the system to computation and, equally, to reduce the spectrum of the system matrix. In applying a preconditioner, the new system can be classed in one of three ways depending on the preconditioner used. These classes are described as being *left* (equation (4.5)), *right* (equation (4.6)) and *left-right* preconditioned (equation (4.7))

$$M^{-1}Ax = M^{-1}b, \tag{4.5}$$

$$AM^{-1}y = b \Longrightarrow x = M^{-1}y, \tag{4.6}$$

$$M_L^{-1}AM_R^{-1}y = M_L^{-1}b \Longrightarrow x = M_R^{-1}y, \tag{4.7}$$

where M_L and M_R denote the left and right preconditioners respectively. Obviously a penalty is paid, in terms of the computational effort, for the use of a preconditioner. This arises because the preconditioner needs to be computed, in some cases stored, and then applied. The optimal preconditioner is $M^{-1} = A^{-1}$, but that would require calculating the inverse of A which is of no advantage. To avoid issues relating to the computational effort, but still achieving a reasonable level of preconditioning, a simple *Jacobi* preconditioner was used in the present work. This can be defined as,

$$diag(A).^1 \tag{4.8}$$

In practice the inverse of M is never computed explicitly. Instead, a system $Mv = c$ is formed and solved for unknown v; the requirement of solving such a system being another reason why *Jacobi* preconditioning is particularly favourable as its inverse is trivial.

[1]$diag(X)$, is a square matrix of the same dimensions as X with the leading diagonal being identical to that of the leading diagonal of X and all other entries being zero.

4.4 Using CUDA

As CUDA is a relatively new technology there is not a great deal of literature available and a general lack of the commonly found "programming manuals". Consequently, strategies for converting and optimising algorithms rely on experience, some guidelines by NVIDIA (NVIDIA, 2008*b*), and the comparatively small amount of published material. The basic methodology adopted by the author in converting the outline of the IDR(s) algorithm provided by van Gijzen & Sonneveld (2008) into code capable of running on CUDA hardware was as follows:

a) Write and optimise the algorithm in `Fortran`.

b) Convert the `Fortran` algorithm into optimised `C/C++`.

c) From the `C/C++` algorithm try and classify operations such that similar operations can be grouped together (a similar strategy to BLAS (Lawson *et al.*, 1979), performed purely to reduce the amount of code required).

d) Address issues arising from calling `C/C++` from `Fortran`.

e) Prototype naïve kernels for the classified operations wrapped in 'helper' functions that copy data explicitly to and from the GPU either side of each CUDA kernel.

f) Move working prototypes into the main `C/C++` code isolating them with compiler preprocessor directives.

g) Once all the kernels have been written, allocate memory on the GPU to share all the necessary data used in the algorithm.

h) Remove the 'helper' functions and just call the kernels.

i) Rewrite the naïve kernels so that they are efficient, use shared memory and have non-divergent warps. Try and make memory access coalesce and make extensive use of accumulators in the GPU registers.

j) Create streams for copying data onto the GPU while the CPU/other GPU streams do something useful.

k) Page lock host memory if possible to enable fast copies/bounce buffering.

l) Map large, read only, arrays to texture types for cached access.

m) Deal with cases when the system will not fit in the memory of a single GPU.

The following sections address each of these issues in turn:

4.4.1 Write and optimise the algorithm in `Fortran`

This task was not particularly challenging, the presence of a translation of the algorithm into `Matlab` code by van Gijzen & Sonneveld (2008) made this task a lot simpler. `Matlab` and `Fortran` both use one based indexing and are column major languages, so translating the algorithm from one language to another did not prove difficult. The author made a number of small changes to the original algorithm, not least because many of the functions available in `Matlab` do not exist in `Fortran`. In particular, a modified (for stability) Gram–Schmidt algorithm (Golub & Van Loan, 1996) was used to compute the orthonormal matrix of random numbers for ‘P’. In addition, a Jacobi preconditioning was added to the algorithm to help speed up convergence as it is cheap to apply both in terms of memory and of time.

4.4.2 Convert the `Fortran` algorithm into `C/C++`

Converting `Fortran` to `C/C++` was again a relatively simple task once two essential points were noted. The first being the underscoring notation used to describe symbols to functions in object files differs between `C/C++` and `Fortran` compilers. The second being the respective row and column major data storage systems of `C/C++` and `Fortran`, special care was require to ensure that the correct data was accessed and that the access was performed in the most efficient manner. Clearly, `C++` name mangling also needed to be addressed by ensuring all functions had the `extern "C"` qualifier.

4.4.3 Prototype naïve kernels

Prototyping the naïve kernels involved writing test beds for common functions similar to the thinking behind BLAS (Lawson *et al.*, 1979). In all prototypes a linear grid and block system was used for simplicity of kernel calls and thread management. Simple programs were written as a harness for the prototype kernels which dealt with input and output (IO) and the set up to the helper functions. Once a kernel appeared to be performing correctly the device emulation switch on the NVIDIA compiler was used to compile the code to run purely on CPU with an effective warp size of one. This was done to make sure that no race conditions, hardware induced thread masking issues, or similar bugs existed in the code.

4.4.4 Move working prototypes into main `C/C++` code

On passing testing, the prototype kernels with their helper functions were added to the main algorithm. The calls to CUDA functions were isolated from the main algorithm with the use of compiler preprocessor directives thus the GPU based parts of the algorithm could be switched on and off for debugging purposes.

4.4.5 Allocate memory on the GPU

Once the algorithm worked using purely GPU kernels wrapped in 'helper' functions, the task of swapping host (CPU) side memory allocations with their GPU counterparts involved using the CUDA equivalent to `C`'s `malloc()` to allocate memory on the GPU for each host variable. The host side variables could then be copied to the GPU ready for use. It was important to test the calls to CUDA's `malloc()` on the GPU to ensure that they were successful as fitting large systems onto the GPU memory was not always possible. This problem is further discussed later in the chapter.

4.4.6 Remove 'helper' functions and just call the kernels.

Switching from host only memory with the 'helper' functions copying data to and from the GPU to using pure GPU memory was probably the hardest part of the process to debug. To achieve this operation in the most efficient manner, a logical dependency table of all variables and functions was produced. Then, with extreme care, host side variables and functions were swapped in logical order for their GPU counterparts ensuring data dependencies were resolved. Once this was completed and the whole algorithm ran on the GPU, the CUDA profiling tools were used to observe the nature of the running code. Unsurprisingly, the initial copying of data from host memory to GPU memory took a significant amount of the solution time (5%). However, the BLAS level 2 type operations involving the system matrix multiplied by a vector dominated the other 95%.

4.4.7 Rewrite naïve kernels optimising efficiency

Rewriting kernels to perform more efficiently revolved around the use of shared memory access on the GPU. A 400-600 clock cycle penalty is paid (NVIDIA, 2008*b*) to access data in the GPU global memory. As a result, it is important to minimise read/writes to this memory region. It is also necessary, for optimum performance, to coalesce data access on the GPU, with all threads in a given warp accessing contiguous GPU global memory regions. This requirement places a further, complicated but important, constraint on the code optimisation.

The most computationally intensive kernel required was a matrix-vector multiplication kernel for multiplying the system matrix, A, by an intermediate vector. Although there are several implementations of this kernel in the literature, none of them are suitable (or optimal) for use within the CUDA implementation of IDR(s). For example Fujimoto (2008) uses texture references (see §4.4.10) which add complexity to an already complex code; while the NVIDIA cuBLAS (NVIDIA, 2008*a*) dense matrix-vector kernel appears to have less than optimal performance that fluctuates with N with a period of 16; this being the half-warp size (Fujimoto, 2008). Indeed, by the time this thesis is completed it is expected that the cuBLAS library will use the method of Fujimoto (2008). Furthermore, Volkov & Demmel (2008) show that it is possible to considerably out perform the cuBLAS implementation for the BLAS-3 matrix-matrix multiplication operation, perhaps indicating that the vendor based cuBLAS is not optimal.

Subspace size s	Solution time using author's kernel (s)	Solution time using `cublasSgemv()` (s)	Speed increase (%) compared to cuBLAS
2	1.606	1.748	8.848
4	1.581	2.032	28.527
8	1.657	1.570	-5.254
16	1.836	1.734	-5.551
32	2.052	2.101	2.398

Table 4.1: Comparisons between `cublasSgemv()` and the author's implementation of a matrix-vector multiplication kernel. Note: s is defined in §4.3.2 and relates to the number of columns in matrix P.

In the interests of personal development, and to understand more about optimising kernels, the author decided to implement a new version of the matrix-vector multiplication kernel. At the outset the author used a method similar to that developed by Fujimoto (2008). However, following discussion with Matt Harvey (Harvey *et al.* (2007), Harvey *et al.* (2009)) and his donation of some proof of concept code, a faster kernel was developed making extensive use of accumulators that are assumed to reside in the ALU registers. The final kernel runs at about 30Gb/s, which is near half the bandwidth of the Tesla GPU on which it was developed (according to NVIDIA's bandwidth test application, maximum achievable bandwidth was 74.1Gb/s), and runs without the use of texture data types. To draw a comparison to the equivalent cuBLAS kernel (`cublasSgemv()`), the run times for solving a problem of dimension $n \sim 16000$ using the IDR(s) method with subspace sizes 2, 4, 8, 16 and 32 are given in Table 4.1. From these results it is apparent that the author's kernel executes at approximately the same speed as cuBLAS without the performance fluctuations leading to poor performance at some subspace sizes.

Once the matrix-vector multiplication kernel and others were optimised (to a reasonable extent), attention was diverted to other slow parts of the code with particular attention being paid to the copying of data between host side memory and GPU memory.

4.4.8 Using streams and interleaving

To overcome the problem of waiting for the A, x_0 and b vectors to be copied to the GPU via `cudaMemcpy()` (this being a thread blocking function), some more advanced features of the CUDA API can be used. If the A matrix is sufficiently small that it can be stored in page locked memory (a more readily accessible form that is explained in the next section), then CUDA streams can be employed to allow asynchronous copying of memory to the device. As a result, a second stream is free to deal with other computations whilst the copying takes place. The use of page locked memory is quite dangerous in terms of causing operating system (OS) lockups and the general slow down of a computer. If large amounts of memory are page locked, the amount of free memory available to the OS for internal functions and applications is limited. If this becomes too small, the system will grind to a halt as the OS has to page everything to disk. With this in mind, should the matrix A be too large to store in page locked memory, a technique similar to double buffering and perhaps closest to bounce buffering can be employed (Gorman, 2004).

4.4.9 Bounce buffering and page locked memory

Access to host memory is handled by the OS which has the freedom to page out sections of memory to disk, should it be required. Consequently, if a request for data is made and the data is located in paged out memory (i.e. buffered on to a disk) the OS has to move the data from the disk back in to main memory, which is very slow. It is possible to allocate memory as “page locked” which in simple terms means that the OS is forbidden to page the memory to disk; it has to sit in memory. Clearly, there is a performance gain whilst copying from page locked memory: no interaction with the OS is required as it is known *a priori* that all the required data is stored in memory. Accordingly, a `memcpy()` can proceed blindly and unhindered.

With knowledge of streams, asynchronous copying and page locked memory, a technique similar to bounce buffering can be used for moving data from the host to the GPU. In this technique two streams work together, copying data and executing computational kernels on the GPU. To start the method, a region of page locked memory is created as a buffer in host memory and two regions of memory are allocated on the GPU. The GPU memory regions can be conveniently accessed via a double pointer of a type based on the type of data to be stored. Each double pointer entry is then set to contain a pointer to an allocated region of GPU memory.

With the memory allocated, stream one asynchronously copies some data into the buffer and then onto the GPU memory (this is a “bounce” through the page locked memory buffer) and stream two idles. Stream one then executes the kernel on the data it “bounced” onto the GPU and stream two bounces its data onto the vacant half of the GPU memory. Once stream one finishes, stream two can execute the kernel on its data and stream one then makes use of the page locked memory buffer to bounce its data onto its half of the GPU memory.

In comparison to a standard sequential ‘copy followed by compute’ mechanism, where the GPU has to wait for the copy to complete prior to computing, a bounce buffered method using interleaved concurrent streams leads to a shorter execution time. The faster execution comes simply from hiding the copying of data and its associated latencies by overlapping them with a computation stage. This technique provides a theoretically faster mechanism for performing operations on sets of data that are too large to fit in GPU memory, a problem that will be discussed later.

4.4.10 Map large arrays to textures for cached access

As the GPUs are fundamentally designed for graphics processing there exists the opportunity to use native graphics based storage types. Operations such as binding arrays to textures and then performing data loads via texture references, making use of spatial reordering and locality in memory, can lead to faster execution times for kernels that display spatial locality in their memory access patterns. The data accessed via a texture reference is locally cached, which may also be beneficial for kernels that make repeated use of certain parts of an array. However, these features lead to an extra layer of complexity in the code base and there is a time penalty associated with the GPU reordering the data in GPU global memory.

Furthermore, the textures are read only. After careful consideration, the author decided not to use texture based functions. The reason for this was two fold; first, it was not readily apparent that the kernels for the IDR(s) algorithm would necessarily benefit from data caching since they are BLAS level 1 and 2 or similar. Second, the extra layer of complexity would have hindered development speed in an already complicated environment.

4.4.11 Deal with the memory limit of a single GPU.

The maximum memory available on any current NVIDIA CUDA GPU is 4Gb. Speculatively, this is due to the limitations of a 32 bit addressing system, and will undoubtedly be increased in future revisions of the architecture. The implication of this limitation on the CUDA IDR(s) solver is that for a large system size, the whole system may not fit in the memory of a single GPU. This gives rise to a number of options:

a) Switch to a solely CPU based algorithm for large problems,

b) use a bounce buffering technique to move data on and off the GPU as required, or,

c) if multiple cards are available, distribute the computation across multiple cards.

Addressing each of these options in turn. First switching to a solely CPU based algorithm for large problems defeats the objective of having a GPU accelerated solver. This option was not considered. Second, using bounce buffering to move data on and off the GPU, although faster than waiting on blocking copies, still produces a very slow execution time as data copying has to go through the PCIe BUS. A full 16 lane PCIe 2.0 BUS can support 8Gb/s of data transfer at peak rate, although this is unlikely to be achieved. In contrast, a NVIDIA Tesla GPU has a theoretical maximum data transfer rate of about 102.4Gb/s (practical maximum achieved was 71.4Gb/s). As a result, the GPU will be starved of data, permanently having to wait for data to be bounced to it. Trials using this method suggested that although the execution time on the GPU would give results significantly faster than using a CPU, the latency from data copying slows the algorithm down to running at CPU speed or worse. As a result, this option was abandoned as being too slow.

Third, there are a number of ways in which to distribute data over multiple GPUs. All of these methods arise because, in the high level API, once a CPU thread makes a call to the CUDA libraries, an immutable *context* is set that binds that thread to a given GPU. Hence, should multiple GPUs be required to solve a problem, either the low level API must be used or, the high level API calls need to be threaded. The author attempted, but failed, to use the UNIX `pthreads` library to continue using the high level API due to the code becoming very complicated. As a result, the conclusion was drawn that simply rewriting the code to use the low level API, or indeed source to source translating the API calls in the code, would be a better option. However, the framework needed to distribute the A matrix over multiple GPUs, and call the necessary kernels, had been implemented. Therefore, the multi-GPU method can be tested using a single GPU by storing multiple sections of the A matrix in GPU memory and then performing the necessary kernel calls sequentially and using the shared memory programming model of sharing results via the host's memory. Although this may not provide the optimal result in terms of speed, as there is no concurrency between kernel invocations as would be the case in a CPU threaded multiple GPU environment, this method does provide some idea of the execution time wasted in having to explicitly share partial results from different GPUs in host memory.

In simulating a multiple GPU environment, as suggested above, distributing the data across multiple GPUs takes lot of book keeping (similar to that in MPI) to keep track of allocated memory regions. The matrix-vector multiplication kernel was the only kernel that required the use of the A matrix and so it was only this kernel that needed to be rewritten to make use of multiple GPUs. For the simulation of a multiple GPU environment the A matrix was split into two. For example, in Listing 4.4.1, allocating A as floating point type memory of size, `bytes`, in a number of blocks, `blocks`, and storing their addresses in `d_data`.

When the A matrix matrix-vector multiplication was required, the vector was copied to the host's memory. The host then copied this vector to the GPU re-

Listing 4.4.1 Referencing memory regions in blocks.

```
float ** d_data;
d_data=(float **)malloc(blocks*sizeof(float *));
for(i=0;i<blocks;i++) //loop through devices
{
   // were cudaSetDevice(i); called here, it would fail on the
   // second loop iteration as the first call would have set
   // an immutable context binding this thread to device 0.
   cudaMalloc((void **) &d_data[i], bytes);
}
```

peatedly, once per block (as would be required in a multiple GPU environment). The matrix-vector multiplication operation was then executed simply by calling the kernel, in turn, on each block with respect to its part of the matrix and vector operation. The result of each invocation was then copied to memory on the host and once all invocations were complete, the result was then copied from the host back to the GPU's memory. Execution of the CUDA IDR(s) algorithm then proceeded as normal.

There is some slowing of the execution time when using the multiple GPU based algorithm, in comparison to when the A matrix is not split into blocks, due to copying vectors to and from the host and executing the kernels in turn. However, this method is considerably faster than using bounce buffering. Indeed, the results (see §4.7) suggest this method scales well and should definitely be reimplemented to use multiple GPUs.

4.5 Problems using CUDA for GPU accelerated matrix solvers

Although CUDA and GPUs provide a solution for accelerating the speed of matrix solving, the methods and hardware do not come without problems; the most important issues being discussed below.

4.5.1 Data precision

The boundary element method for which the CUDA IDR(s) solver was developed works sufficiently well in practice with single precision results from the matrix solving process. If, however, single precision results were not sufficient for a given application, the CUDA enabled GPUs do have a number of double precision ALUs available. However, this number is small in comparison to the number of single precision ALUs. In undertaking the calculations, the algorithm should map to the GPU using double precision arithmetic in exactly the same way as single precision. However, due to the reduced number of double precision ALUs available on the GPU the computation will proceed more slowly. In the future it is believed that NVIDIA will produce GPUs with larger numbers of double precision ALUs; the current revision of the CUDA architecture being very much a first revision.

4.5.2 Internal precision

Whilst performing computations on a CPU, the CPU uses extended precision internally thereby preserving results that might have otherwise under-flowed or cancelled. Unfortunately, due to the more simple processing cores on GPUs, this feature does not exist in the SM ALUs. If an algorithm is particularly susceptible to under-flows or cancellations then the use of double precision is perhaps advisable. Fortunately, the CUDA IDR(s) algorithm does not seem to be particularly prone to this problem unless a very high level of accuracy is required. As a result, single precision was adopted throughout and the absence of extended internal precision was not considered an issue.

4.5.3 Memory

By far the greatest concern with respect to dense matrix solving on GPUs is the GPU memory requirement. Although methods have been discussed to deal with distributing the A matrix across multiple GPUs, the problem of keeping sufficient GPU memory available for uses other than storing the A matrix puts some overheads on the algorithm. Thankfully, the IDR(s) method has a fixed memory footprint for a given problem size. As a result, it is complicated, but possible, to work out exactly how much memory is needed to solve a particular problem. This point is fundamentally important when working in limited memory environments, giving iterative methods with, fixed depth, short recurrences an inherent advantage over those with long recurrences such as GMRES (Saad & Schultz (1986)).

4.6 Test bed

To demonstrate the improvements in speed achievable using GPUs, it was necessary to look at a number of problem sizes and a number of algorithms. The machine on which the code was run is a high end 'gaming rig' with the following specification:

- CPU: Intel(R) Xeon(R) CPU E5405 Quad Core @ 2.00GHz
- Host Memory: 4x4GiB DDR2-667 ECC FBDIMM
- Intel D5400 XS Motherboard 'Skull Trail'
- NVIDIA Quadro FX3700 512MB PCI Express GPU
- NVIDIA Tesla C1060 GPU
- Operating system: Fedora 10 (Linux) with 2.6.27.25-170.2.72.x86_64 kernel
- CUDA SDK 2.1, 64 bit driver version 185.12

This machine was specifically chosen for development work and is far from what would be used in a production level HPC cluster environment. However, it does allow some important insights into the speed up and scaling properties of CPU and GPU based codes.

To investigate the algorithms and their various implementations, five methods were tested on eight typical matrix orders. Details of the methods tested and their properties can be found in Table 4.2. For method 5, the NETLIB release of LAPACK's (Anderson *et al.*, 1999) single precision real linear equation solver, `SGESV()`, was used such that comparisons to the industry standard algorithms can be made furthering the completeness of the study. All other methods were as described previously.

Method Number	Algorithm	Language	Version	Executed on
1	GMRES	`Fortran`	Serial	1 Core of CPU
2	GMRES	`Fortran`	OpenMP	All Cores (4) of CPU
3	IDR(s)	`Fortran`	Serial	1 Core of CPU
4	IDR(s)	`C/C++`	CUDA	CPU/GPU
5	Direct	`Fortran`	LAPACK	1 Core of CPU

Table 4.2: Matrix solving algorithms and their properties.

The matrix orders, N, investigated in the present study are based on typical sizes for real applications of the BEM and the matrices are populated with genuine BEM data. To put these sizes into perspective, the following guidance relating matrix order and problem type is relevant.

a) A matrix order of $N \sim 4000$ lies at the top end of a 2D BEM model e.g. Christou *et al.* (2008).

b) An order of $N \sim 12000$ is typical of the size used by Peric (2010) for work on wave-structure and wave-ship interactions.

c) An order of $N \sim 16000$ is chosen as a typical size for modelling the working area of the wave basin at Imperial College London at $7.5cm$ resolution. This would be adequate for modelling non-breaking waves (Chapter 6).

d) An order of $N \sim 24000$ corresponds to a typical size for a higher resolution BEM solution for the previously mentioned wave basin. This would, for example, be required for the modelling of breaking waves (Chapter 7). Simulation would typically require a 7 day execution time on a 64 processor cluster computer.

e) An order of $N \sim 30000$ was chosen as being close to the largest size that would fit in the 4Gb memory of a single Tesla GPU, with memory space left for the GPU to execute the kernels.

Matrix order (N)	Condition number
2026	312.0251
4026	448.5835
5986	550.2700
8066	640.5623
12154	787.4979
16146	911.6653
20066	1024.5050
23938	1125.7493
26226	1182.6130
30098	1272.2650

Table 4.3: Matrix order and condition numbers.

For each matrix size the condition numbers were computed using LAPACK's `SGESDD()` routine to obtain a singular value decomposition of the matrix. Then, the ratio of the largest and smallest elements of the diagonal matrix (this contains the matrix's singular values and is referred to as `SIGMA` in the routine's header) was taken to give the condition number. The exact matrix orders and condition numbers are given in Table 4.3.

For each matrix order the code was run ten times; the time taken to complete a run based on calls to the OpenMP run time library timing routines. In each case the run times from each execution of the code were averaged and intercomparisons provided. The test for having achieved the desired accuracy was based on the computation of the 2-norm of the residual from the solution in relation to the 2-norm of the right hand side vector b,

$$accuracy = \frac{||r||_2}{||b||_2}. \tag{4.9}$$

Bettering or achieving a similar accuracy to that of the serial GMRES algorithm provided the definition of a successful solution. When the accuracy of the solution from the algorithm being tested (Table 4.2) reached the accuracy of the solution from the serial GMRES algorithm for a given problem size, the solution was considered good enough and the timers were stopped. In addition, to improve the general performance of all the solvers, except in the case of LAPACK, a *Jacobi* preconditioner was applied to the system matrix as described in §4.3.3.

As the IDR(s) algorithm has the ability to change the size of the shadow space, s, in which the solution is sought, a number of sizes for s were tried for each matrix order. The value of s was started at two and doubled on successive runs until it reached 32. However, the variation in the run times with s was not considered a problem as writing a simple code that varies s to get an optimal run time could easily be added to any scheme that was to use the IDR(s) algorithm.

4.7 Discussion of results

Figure 4.3 contrasts the results arising from testing the various solving methods and confirms that the IDR(s) algorithm running on the GPU is very effective. In making these comparisons the run times for the LAPACK routine are predictably larger due to the complexity of a direct solver which scales as $O(n^3)$. As a result, data relating to this routine are not included. Considering the data presented in Figure 4.3, it appears that the execution time for the IDR(s) algorithm scales more poorly than the GMRES algorithm for the CPU implemented cases. Furthermore, in these cases, it also appears that the size of subspace s used has a considerable effect on the rate of convergence. Interestingly, this appears not to be the case for the GPU enabled IDR(s) algorithm; the execution time relating to the various subspace sizes effectively lying on top of each other on this scale plot.

Although Figure 4.3 is useful in terms of a general overview, the more important results are found by comparing the CPU IDR(s) algorithm against the GPU IDR(s) algorithm, and comparing the GPU IDR(s) algorithm against GMRES running on multi-core CPUs. Figure 4.4 addresses the first of these tasks, identifying the speed gain from using the GPU in comparison to the CPU for the IDR(s) algorithm. The GPU IDR(s) scheme indicates a favourable speed up, and most importantly, this speed up is linear, or better, with respect to matrix size. This behaviour is expected as the larger matrices will perform better simply because the memory access latencies can be hidden, or have proportionally less effect, due to the excessive amount of computations on-going in such cases. This implies that

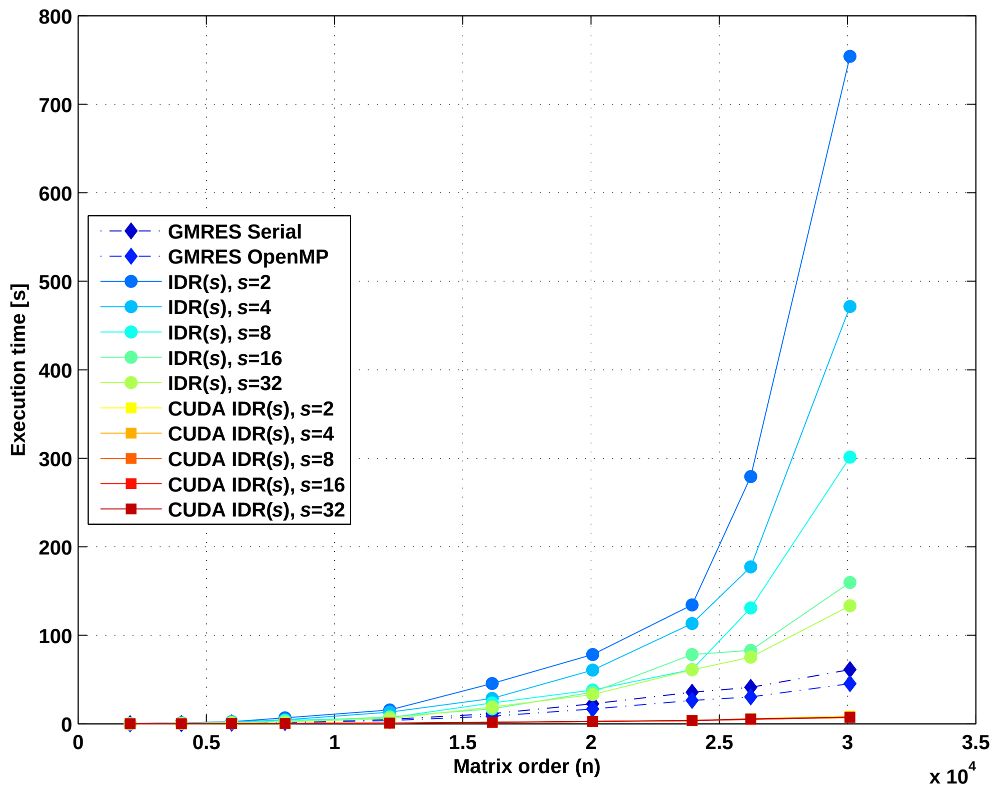


Figure 4.3: Raw execution timings for different algorithms solving various matrix sizes. The results for CUDA IDR(s) are not missing, they just lie on top of each other at this scale.

larger matrices will be solved fastest, relatively, and that as CUDA architecture becomes more efficient the benefits will become more apparent.

Some additional results corresponding to a matrix size of $N \approx 23000$ are also included on Figure 4.4; the relevant data points denoted by asterisks. These results relate to the proof-of-concept trial where the A matrix was distributed over multiple memory regions on the GPU (details of the process outlined in §4.4.11) with data shared via the host's memory when required. The behaviour of the solver appears to be largely unaffected by such a scheme. This is very promising for the future whereby whole numerical models could be run on multiple GPUs on a single machine; the GPUs communicating via a shared memory model through the RAM on the host.

The primary purpose of the work undertaken in this chapter was to see if the bottle-neck associated with the matrix solving phase of the boundary element scheme could be reduced by increasing the speed at which the solving phase takes place. The multi-core version of GMRES was, in practice, the fastest available solver prior to this investigation and so this must be the basis for any comparisons. Accordingly, the speed up of the execution time obtained by CUDA IDR(s) over the multi-core CPU based GMRES is indicated on Figure 4.5.

For completeness, Figure 4.6 shows the level of accuracy of the solutions obtained by the five different methods. The CPU GMRES and multi-core CPU GMRES display the same accuracy. This is to be expected, as the non-deterministic behaviour of OpenMP reduction clauses are unlikely to have any effect as underflows and catastrophic cancellations do not regularly occur in the GMRES algorithm. The LAPACK routine is clearly most accurate. Again, this is to be expected since it uses a direct method. However, this optimal accuracy comes at a very considerable computational cost. Indeed, were this approach to be adopted, calculation of the most interesting wave forms (outlined in Chapters 6 and 7) would simply become impossible. Furthermore, the CPU and GPU versions of IDR(s) display different accuracies of solution; the CPU version perhaps tending to be slightly more accurate for the smaller matrix sizes. However, the general result is that the level of accuracy obtained from IDR(s) was, as required, better than that obtained from GMRES. The one exception to this rule occurs for the

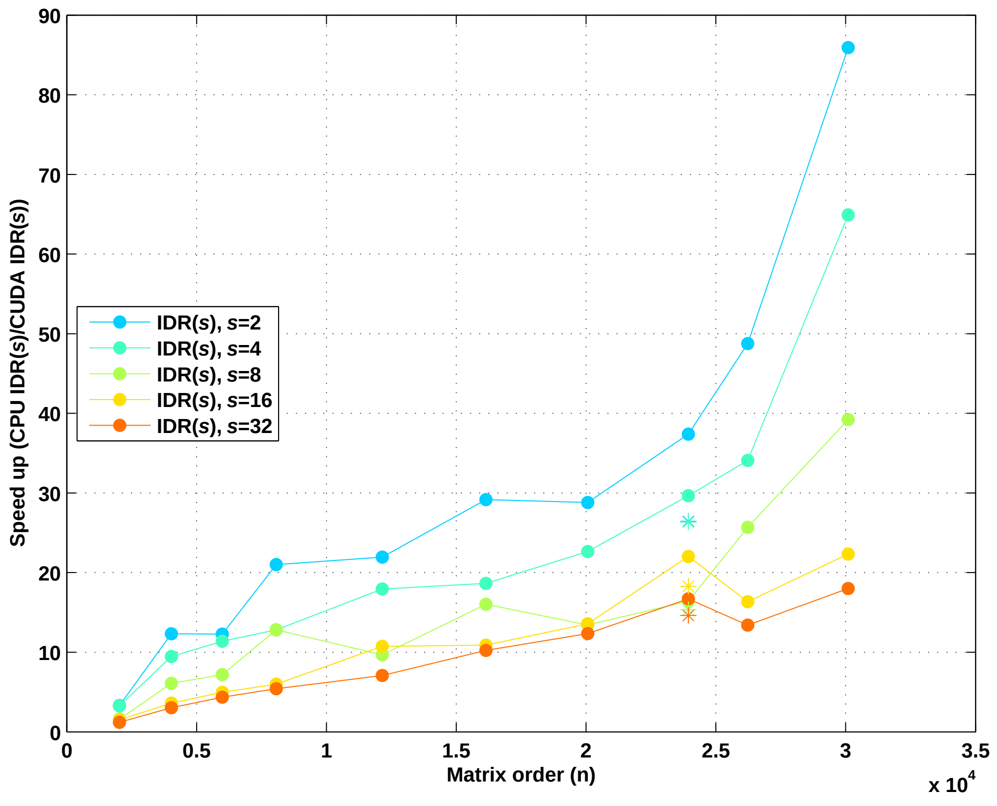


Figure 4.4: Comparing IDR(s) implemented on a CPU with IDR(s) on a GPU, both solving various matrix sizes. The additional markers ($*$) at a matrix size of $N \approx 23000$ relate to using multiple memory regions and distributing computations involving the A matrix as outlined in §4.4.11. The colour of the additional markers indicate the size of the subspaces (s) employed used in the computation and are consistent with the colours defined in the legend.

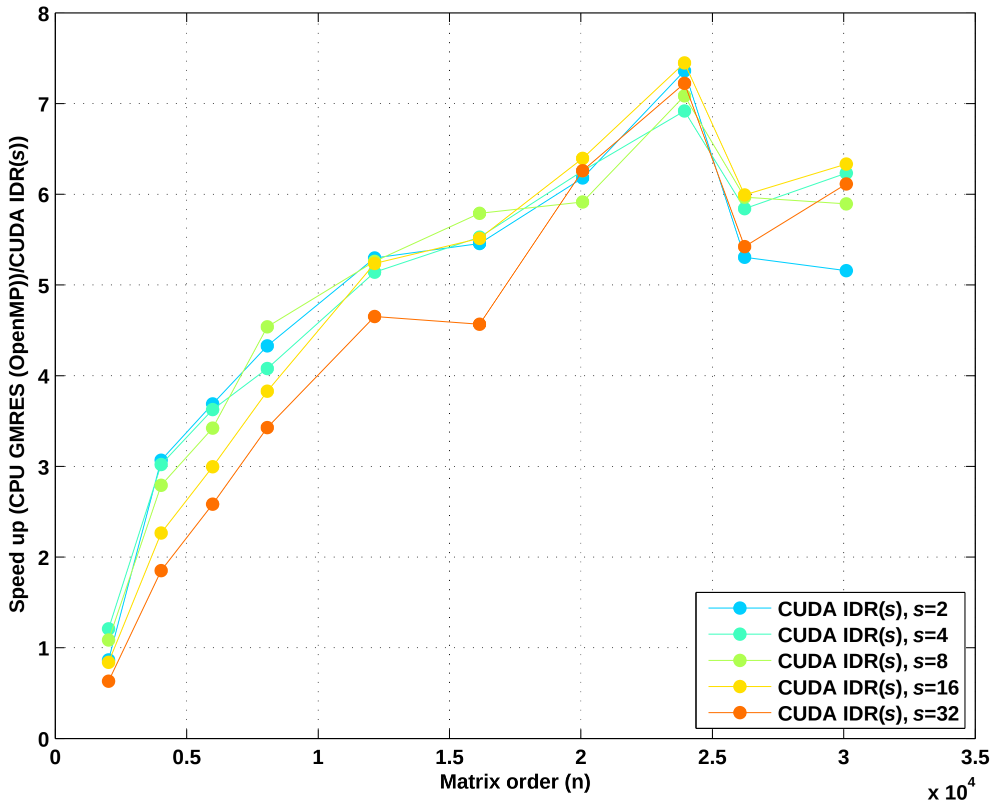


Figure 4.5: Comparing the solution times for GMRES on a CPU with IDR(s) on a GPU, both solving various matrix sizes.

matrix system of order ∼30000 when solved using CPU based IDR(s) where the accuracy seems to be less good for subspaces of size 16 and 32. The reason for this is as yet unknown. Nevertheless, on the basis of these results, there is no reason why the CUDA enabled IDR(s) code should not be used in a production environment as a replacement for GMRES in boundary element modelling; consistent accuracy and improved speed being readily achievable.

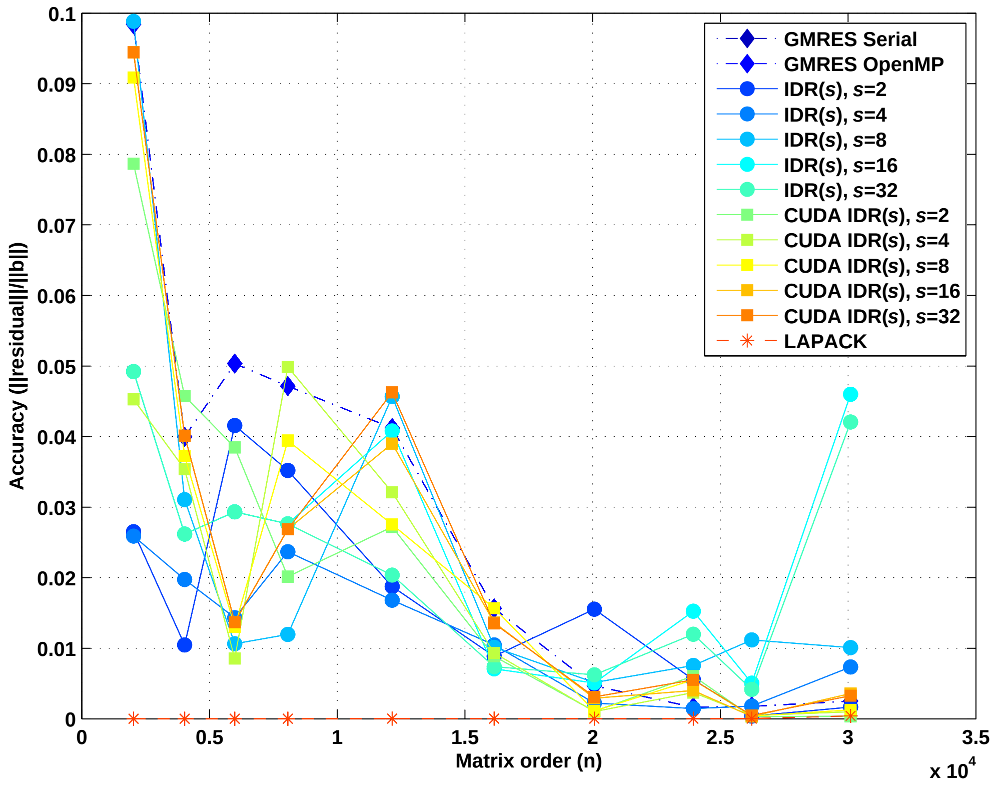


Figure 4.6: Comparing the accuracy of the solution obtained by CPU GMRES, CPU GMRES with OpenMP, CPU IDR(s), GPU IDR(s) and LAPACK.

4.8 Outstanding issues.

There are two main issues that remain to be resolved with the CUDA IDR(s) solver, both being due to external factors. The first is associated with the limited memory available on GPUs. This has been discussed earlier along with the problems it causes in terms of deciding how much memory can be allocated on a GPU without disrupting its internal function. Hopefully, this problem will be resolved in the future with the release of a set of tools to probe internal memory usage, or, perhaps more simply, the availability of cards incorporating large amounts of memory.

The second issue relates to an interesting problem at compiler level. To further speed up the multiple GPU matrix-vector multiplication process (discussed in §4.4.7) the inherent parallelism of the tasks occurring individually on each GPU need to be exploited. In particular, the copying of the vector from the host memory to the GPUs and the subsequent kernel execution could be carried out simultaneously. The results from each GPU can also be copied back to the host simultaneously as there is no overlap between different parts of the result vector. This whole process lends itself well to shared memory programming techniques using threads; for example, those available in an OpenMP environment or calling the UNIX `pthread` library. The author tried for some time to get this method to work. However, it appears that some of the thread contexts assigned when invoking OpenMP are lost between parallel regions, making it rather hard to implement within the current code base. For the future, this issue clearly needs to be addressed either by rewriting some of the code base or perhaps the use of the low level API which seems to preserve the contexts. There are also some further issues involved with mixing OpenMP thread information between different compilers. The author is confident that this can be addressed, but the details lie outside the scope of the present project description.

4.9 Concluding remarks

It is evident that GPU technology is very powerful; provided it is programmed well and applied to an appropriate problem, the technology easily out performs a standard CPU. The CUDA hardware and extensions to the `C` programming language seem suitable for use in a scientific environment and allow access to the compute power of a GPU co-processor with little difficulty. Nevertheless, the present study has clearly shown that some issues still remain due to the infancy of the technology.

The IDR(s) algorithm does not perhaps perform, in terms of raw speed, as well as GMRES for the types of linear systems investigated. Even so, it represents a promising method and is attractive in view of its adjustable and predictable memory footprint. It is therefore concluded that there is no reason why the GPU IDR(s) algorithm could not be used in combination with the distributed computing methods outlined in Chapter 3 to create a combined BEM run time acceleration scheme using both technologies. This concept is the topic of the next chapter.

5

Efficient BEM Algorithm

5.1 Introduction

This chapter evaluates the possibility of combining the concepts in Chapters 3 and 4 to give a boundary element scheme that uses distributed computing for the influence matrix formation phase and is then locally accelerated on GPU(s) for the matrix solving phase. The majority of the background to this combined code has been explained in Chapters 3 and 4. In Chapter 3, the MPI libraries were shown to provide an effective method with which to distribute computational workload in a distributed computing environment. Likewise in Chapter 4, GPU hardware was shown to provide an excellent platform on which to accelerate the computation. With the introduction of NVIDIA Tesla C1070 hardware to a production level cluster computer, as found in the HPC centre at Imperial College London, it is possible to integrate both MPI and CUDA technologies in the same application. Before investigating the performance characteristics of this combined code, some additional discussion is required concerning the compiling of code with multiple compilers and multiple languages. Once this is complete, the basis for comparison will be outlined and the performance of the model reviewed.

5.2 Code with multi-level parallelism

The code for the EPIC_BEM project was written in a reasonably modular fashion, at least to the extent that `Fortran` allows. This approach was adopted for ease of development. In addition, the core routines of the code were written with the premise that they could be swapped and built *ad hoc*. The source code used in a particular build is configured using GNU Autotools configuration scripts to which a number of arguments can be appended on the command line using the `--enable` and `--with` handles. These arguments, applied via the use of compiler pre-processor directives, determine the functions that are included in the final executable. As a result, the resulting executable does not have a monolithic function set and is, perhaps, more optimised in the sense that it is tailored to both the executing machine's architecture and the requirements of the numerical model defined by the user.

The code for the CUDA based matrix solver, outlined in Chapter 4 was written in such a way that it could be used as a drop in replacement to any matrix solving routine written in `Fortran` or `C/C++`. Consequently, it was relatively easy to swap the OpenMP enabled GMRES routine used by default in the EPIC_BEM code for the CUDA IDR(s) solver, on the condition that a CUDA enabled GPU is present in the host machine (the machine executing MPI Rank 0).

With the addition of the CUDA enabled matrix solver, the EPIC_BEM code uses MPI to distribute the work as outlined in Chapter 3 and then the resulting matrix is solved using the CUDA IDR(s) solver as described in Chapter 4. Hence, the benefits of a fast, distributed, influence matrix formation phase are combined with the benefits of accelerated matrix solving phase using the CUDA hardware. The code combining the two technologies displays what is, perhaps, a more novel parallel behaviour; work being executed in parallel on different CPUs using MPI and then, potentially, on different threads of a GPU using CUDA. Studying this behaviour and identifying its potential benefit is the focus of this chapter.

5.3 Compiling code comprising multiple languages

As described in Chapter 4 it is possible to link `Fortran`, `C` and `C++` code in the same application. Adding MPI into the equation simply adds another level of complexity to the problem. The MPI `Fortran` compiler, `mpif77`, is often a wrapper for the `Fortran` compiler standard to a system. Put more simply, the underlying compiler has a number of includes and libraries against which to link, and the wrapper just makes the problem of dealing with the link order easier for the user.

Taking this into account, and bearing in mind that only routines needing to use MPI (and the final executable) need to be built with `mpif77`, it is possible to classify the build process of the mixed MPI-CUDA code into routines that are solely `Fortran`, `Fortran` with MPI calls, `C`, `C++` and CUDA. Once this has been achieved, it is simply a question of creating the object files for each routine based on the contents of the source and linking them. In undertaking this task, the main difficulty lies in mixing `Fortran` code with `C/C++` code.

5.4 Mixing `Fortran` with `C/C++`

As discussed in §4.4, `Fortran` is a column major language. As a result, sequential elements of an array stored in memory are accessed by incrementing the left most index of an array descriptor. To compute a matrix-vector product in the fastest possible manner the caching behaviour of the processing unit needs to be exploited. In very simple terms, when a memory request is made, a whole strip of data (that is known to contain the piece of information requested) is pulled from the main memory on the computer and dropped into the processor's cache. Therefore, if the data is accessed sequentially, there is a high likelihood that the next piece of data will also be found in cache, thereby speeding up the calculation taking place.

Based on this understanding, it is advantageous to form and store the influence matrix in a transposed form (relative to what is seen mathematically) such that the rows of the matrix are aligned with the column major storage format used in `Fortran`. The advantage of this approach becomes apparent when a matrix-vector product is required in the solving phase of the scheme. In this case the operation will benefit from the caching behaviour of the processor. Furthermore, the net result of aligning the matrix ordering with the storage ordering is doubly beneficial when it comes to passing the array pointer to `C/C++`. This arises from the array being effectively transposed by the column major storage order of `Fortran` and then transposed again by storing sequential elements to be aligned with that order. Hence, the net effect is that the array is oriented in the correct way for fast access from either language simply because sequential elements in the array are next to each other in the memory.

5.5 The test bed

The compute cluster on which the behaviour of the MPI-CUDA algorithm was tested is a part of the CX1 cluster in the HPC centre at Imperial College London. Each node of this part of the cluster contains two Intel Xeon processors each with four processing cores, 16Gb RAM and a 10GigE ethernet interconnect along with the usual parts found in a compute node. In addition to the compute nodes there are GPU nodes. These are specialised units made up of four NVIDIA Tesla C1060 GPUs connected in pairs to the compute nodes so that each compute node has access to two GPUs. The per compute node environment is summarised below:

- CPU: 2× Intel® Xeon® CPU E5462 Quad Core @ 2.80GHz
- Host Memory: 4× 4GiB DDR2-800 FBDIMM
- Supermicro motherboard-X7DWT
- 2× NVIDIA Tesla C1060
- Operating system: RHEL 5 2.6.18-164.6.1.el5 kernel
- CUDA SDK 2.3, 64 bit driver version 190.18

To investigate the properties of the MPI-CUDA algorithm, a number of test cases were chosen. To be consistent with Chapter 3 the same problems of sizes 4962, 10242 and 20162 were used as defined in Table 3.2. The problem size of 40322 would not fit within the 4Gb of memory available on a Tesla C1060 GPU and so an alternative case with 30578 nodes assembled from a domain with a 0.05m discretisation in all directions occupying the dimensions $0.0\text{m} \leq x \leq 5.9\text{m}$, $-5\text{m} \leq y \leq 0.0\text{m}$ and $-0.8\text{m} \leq z \leq 0.0\text{m}$, is used in its place.

The smallest available number of processing cores available for a job was eight, as opposed to the four used in Chapter 3. This is simply because processor technology advances very quickly, perhaps in accordance to Moore's law (Moore, 1965). As a result, the transistor density on processors could double twice within the three years of research commonly required to write a doctoral thesis. The number of processors on which the code ran are therefore all multiples of eight; specifically 8, 16, 32 and 64. For each number of processors the code was run twice, both for 103 steps. In the first run GMRES was used to solve the system matrix, whilst in the second run CUDA IDR(s) provided the matrix solution.

In accordance with the comparisons outlined in Chapter 3, the first 103 steps of a simulation involving a Stokes 5^{th}-order wave entering the domain were timed, the first 3 steps were ignored, and the rest used to calculate an average computation time for a single time step. To maintain compatibility with the earlier (Chapter 3) comparisons, several additional metrics were collected. These include the time taken to form the system matrix, the time taken to solve the system matrix and the time taken carrying out any serial processing needed.

The subspace chosen for the CUDA IDR(s) code was of size four. This is because of its apparently stable behaviour with respect to both its speed increase over the GMRES algorithm and the solution accuracy; full details of which are provided in §4.7. Additionally, van Gijzen & Sonneveld (2008) suggests that a recursion level of four is a good default value in terms of accuracy and the work required to achieve this accuracy.

5.6 Discussion of results

The results of this investigation are rather promising both in terms of the increased speed associated with the use of the CUDA IDR(s) solver and in terms of some very interesting information about the solving characteristics of the GMRES and IDR(s) algorithms. Figure 5.1 concerns the average execution time per time step as described previously. The benefits of using the CUDA IDR(s) solver are immediately apparent when comparing the execution time across an identically sized distributed computing environments. The results show, once again, that using more processors produces shorter average step times. However, in the context of the present study it is important to note that there is a constant decrease in average step time for a given problem size when employing the CUDA IDR(s) solver.

These benefits can also be expressed as a speed up in the computations by dividing the time taken using only CPUs, by the time taken using CPUs with GPUs. This data is given as Figure 5.2 and shows that for the larger distributed computing environments, the benefits of using CUDA IDR(s) are (relatively) more beneficial. This is not particularly surprising considering that the formation stage is going to be quicker if more processors are used, but the solving time remains approximately constant. For large sized problems, using 64 CPUs and one GPU, a speed up of approximately 2.1 times is achieved. Whilst this figure may not sound particularly impressive as a performance gain, the better use of computational resources this permits is of considerable benefit.

The improvement in the use of the computational resources is best demonstrated by considering the effort associated with the various stages of a computational step; specifically the general serial code, the parallel matrix formation and the matrix solution. Details of this breakdown are provided in Figure 5.3; the data arising from a 64 processor environment and relating to the problem sizes discussed in §5.5. From this figure, it is readily apparent that the GPU solver reduces the execution time of the matrix solving phase to the extent that the distributed processing environment is almost constantly busy. It therefore appears that using GPUs to accelerate the solving phase not only speeds up the overall step time but, in addition, results in a far better use of computational resources. This is entirely consistent with the aim set out in §4.1.

In addition to the obvious benefits of using an accelerated computing platform for the matrix solving stage, some secondary (beneficial) effects are evident when comparisons are made with the commonly adopted GMRES algorithm. Figure 5.4 displays the execution time for the GMRES and the CUDA IDR(s) solvers

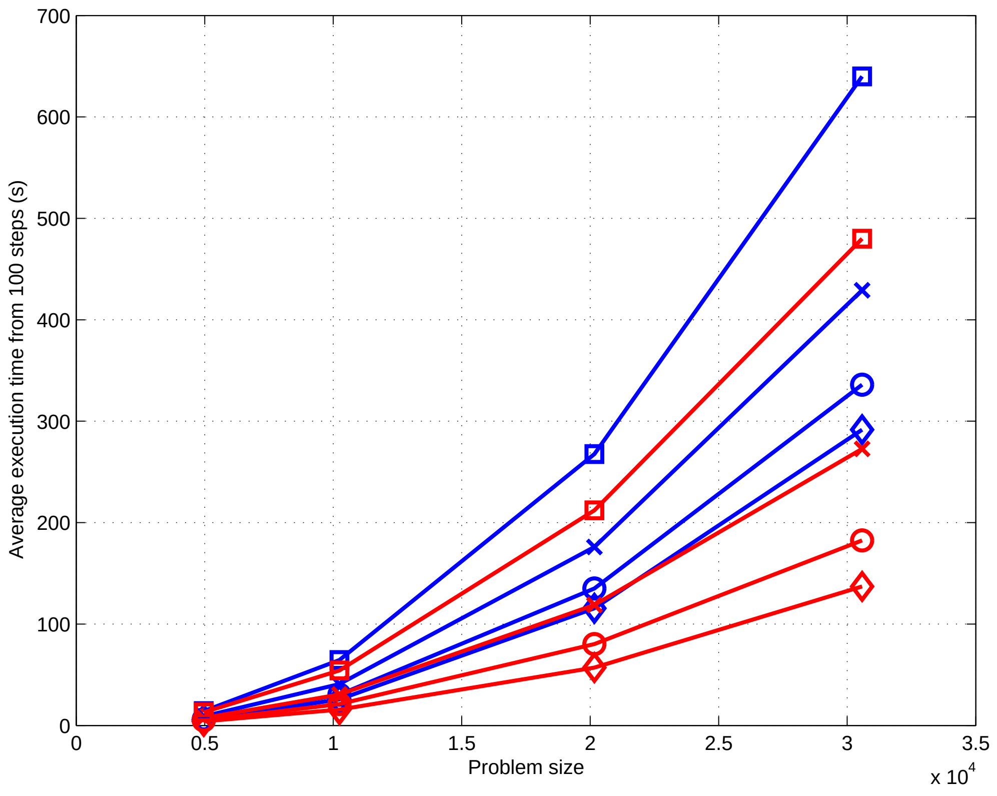


Figure 5.1: Execution times for pure CPU and CPU with GPU implementation, both applied to the solution of a range of matrix sizes. CPU only (━━) and CPU with GPU acceleration (━━). Symbols used: 64 CPUs ($\diamond$), 32 CPUs ($\circ$), 16 CPUs ($\times$) and 8 CPUs ($\square$).

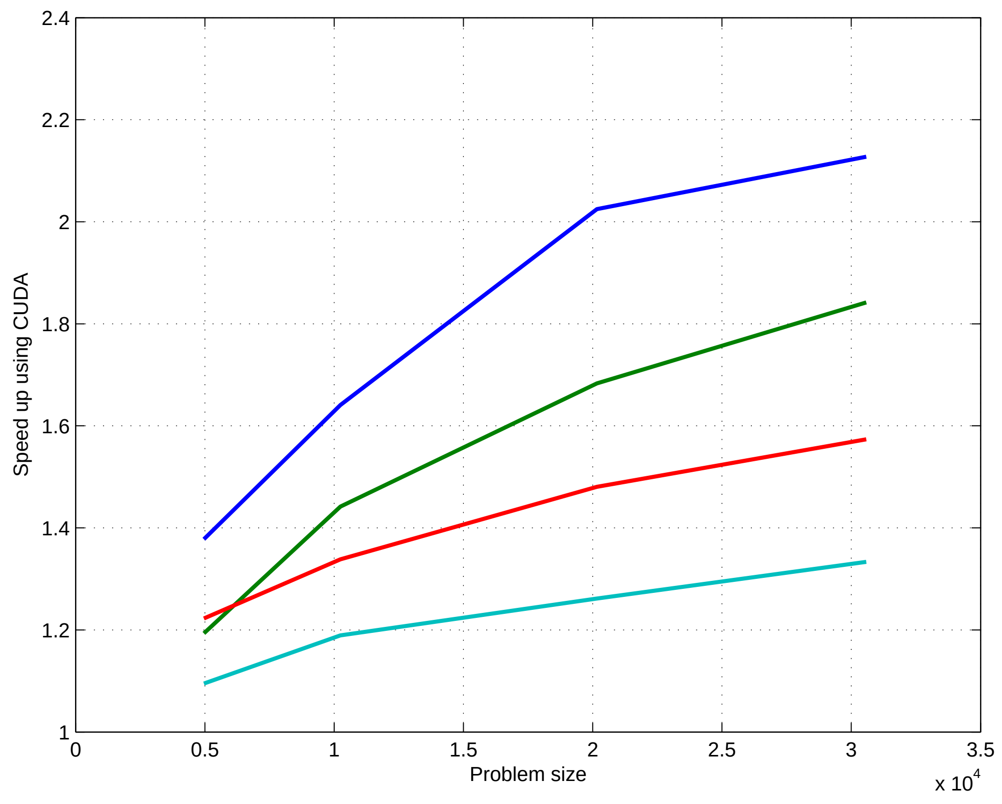
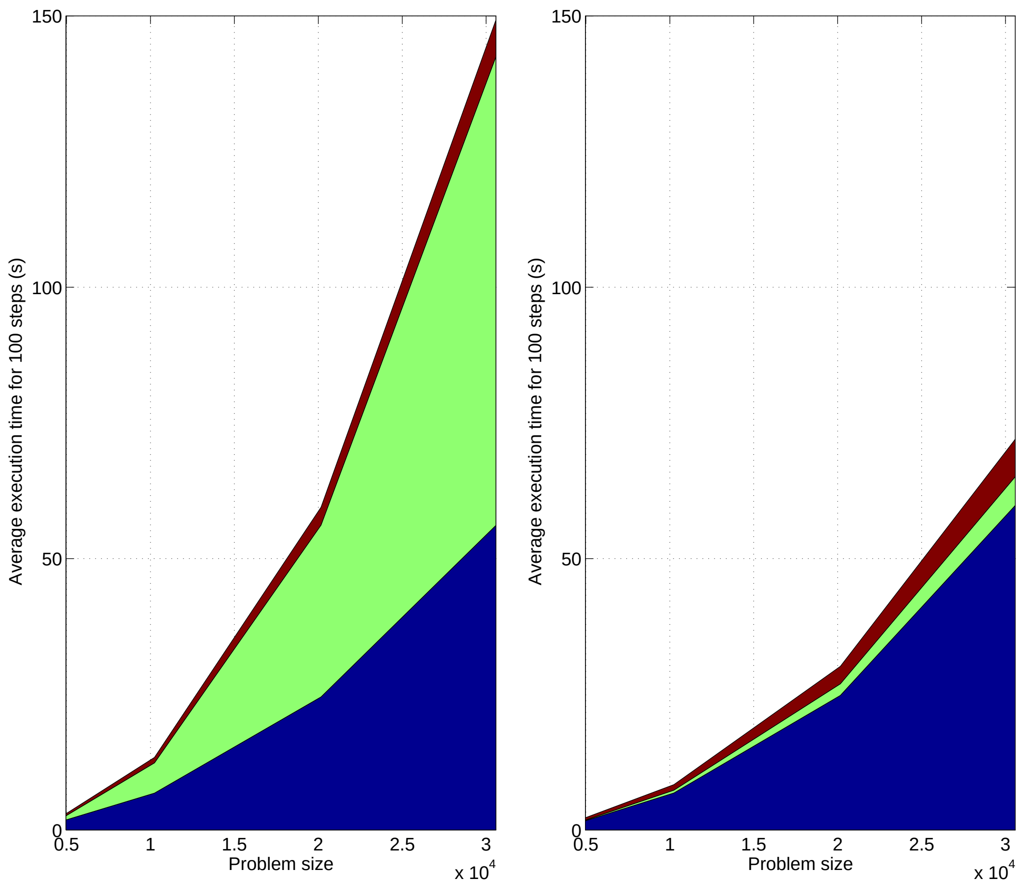
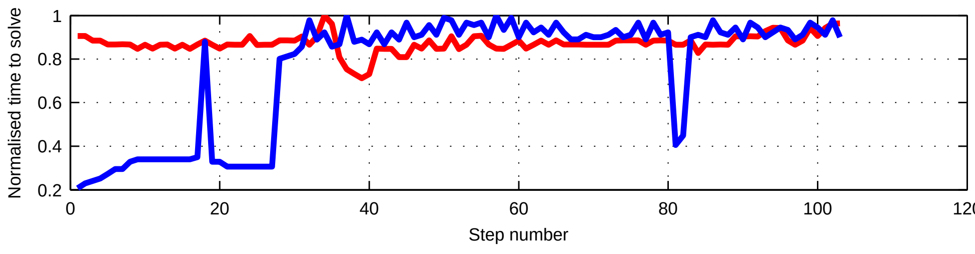


Figure 5.2: Speed up in the computation time achieved using CPUs with the GPU implementation compared to using CPUs alone. 8 CPUs (🟦), 16 CPUs (🟥), 32 CPUs (🟩) and 64 CPUs (🟦).

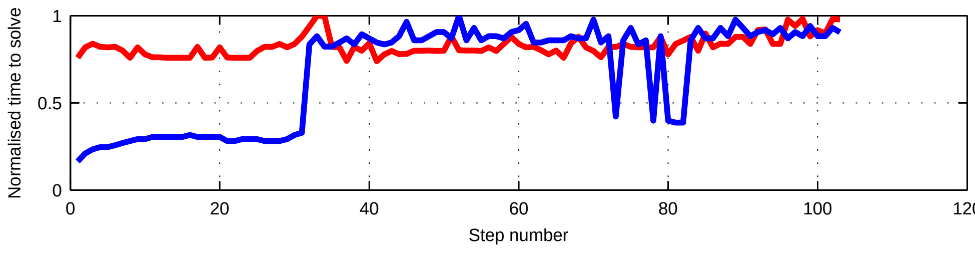


(a) CPU only (b) CPU with GPU

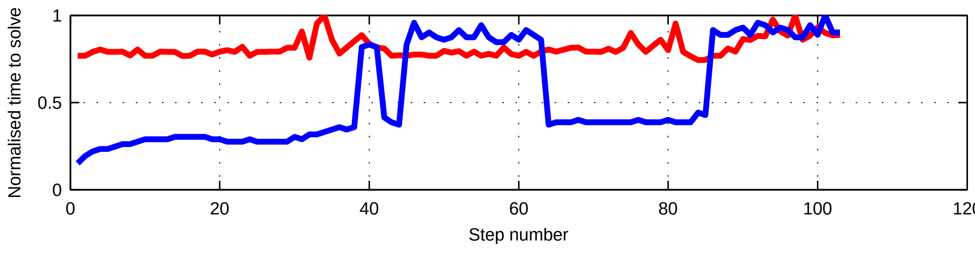
Figure 5.3: Average time spent running each stage of the BEM calculations with 64 processors applied to a range of problem sizes, (a) CPU only, (b) CPU with GPU acceleration. Matrix formation (🟦), matrix solution (🟩) and serial processing (🟥).



(a) 30578 nodes



(b) 20162 nodes



(c) 10242 nodes

Figure 5.4: Normalised execution times for GMRES and IDR(s), s = 4, with respect to the BEM time step number for problem sizes as indicated below each sub-figure. GMRES (🟦) and IDR(s) (🟥).

for each of the 103 time steps, normalised against the maximum execution time for each solver respectively. There is evidently a large fluctuation in the execution time of the GMRES solver. The reasons for this are presently unknown. However, it is possible that there is some complicated interaction pattern between the BEM scheme that forms the matrices and the GMRES solver which results in a greater range of possible matrix conditions and therefore a large fluctuation in solving times. This behaviour is presently under investigation, but regardless of this fact, it is clear from these figures that the IDR(s) solver behaves far more predictably in terms of execution time when compared to the GMRES algorithm. This characteristic is particularly useful when it comes to predicting run times.

5.7 Conclusion

To conclude this chapter it is clear that the BEM benefits greatly from the use of GPUs in the matrix solution phase. The use of CUDA IDR(s) for matrix solving allows a considerable gain in performance to be achieved and results in a more effective use of distributed processing power. Indeed, the entire processing environment is almost constantly busy as opposed to the 50% work-50% idle scheme that often occurs when just using CPUs. Furthermore, some interesting characteristics surrounding the solution time behaviours of GMRES and IDR(s) solvers have been noted. This suggest that it may be beneficial to use a scheme with short recurrence relations should a more constant solving time be desired.

The combining of CPU and GPU technologies has led to some further thoughts on the future of numerical modelling. Harnessing the massively parallel architecture found in GPUs is a challenge, but if done correctly the benefits can be considerable; evidence of this having been provided in the past two chapters. As the cluster computer in the HPC centre at Imperial College London has a number of GPU enabled nodes available, the question of the use of distributed GPU enabled codes can be raised. Some discussion of this subject is found in §8.3.2 and will almost certainly be fully implemented in the EPIC_BEM model in the future.

This chapter concludes the work on improving the performance of the multiple-flux BEM through the use of specialised hardware and software. This, however, is not the end of the search for faster simulation times. Finally, it should be mentioned at this point that although employing the algorithms and hardware presented within this chapter lead to considerably improved simulation run times, the hardware on which this was performed (specifically the C1070 GPU units) was loaned to the author for the purpose of benchmarking this code. Therefore it should be noted that all the results presented in the following chapters use solely CPUs and implement the algorithms and methods described in Chapters 2 and 3.

6

Kinematics Calculations, Code Validation and Practical Application; Non-breaking waves.

6.1 Introduction

The purpose of this chapter is twofold. First, a formal validation of the EPIC_BEM model is presented. Second, the model is employed in a practical application, and in so doing, the necessity of such a model is demonstrated. To begin the chapter, the methodology and calculation procedure relating to the prediction of the water particle kinematics is described, this being a key part of both the validation and practical application. Following this, a formal validation of the model is undertaken in which both regular and irregular waves are considered. This addresses predictions of the water surface elevations and the underlying water particle kinematics, both being compared to well known analytical theories and existing numerical models. Having established the capabilities of the EPIC_BEM model, the discussion turns to the properties of real seas. The focus of the discussion becomes the numerical simulation of a wave with an annual probability of exceedance of 10^{-4}, this being commonly used as a design condition for the appraisal of offshore structures. The chapter finishes by presenting results obtained from employing the EPIC_BEM in a practical situation relating to fluid loading on a jacket structure. A brief review of fluid loading is given and the results from the EPIC_BEM model are reconciled with existing design practice. The purpose is to demonstrate that the adoption of a substantially improved wave model has significant benefits in terms of practical design calculations.

6.2 Calculating the internal water particle kinematics

As described previously, the BEM solution is undertaken on the discretised boundary enclosing the fluid volume. As a result, only information such as the potential, the potential flux and the fluid velocities relating to points (or nodes) on this boundary is known throughout the duration of the numerical calculation. Indeed, this information is all that is required for the simulation to proceed. However, if required, information at positions internal to the boundary can be computed to facilitate a fuller description of the behaviour of the fluid within the domain. In the present context, it is precisely this computation that is required to provide a full description of the velocity field within the domain, or some specified sub-domain. In turn, as will be described later, the velocity field is also required for the computation of fluid loads acting on a body present in the flow. Indeed, the computation of velocities at points internal to the domain is a necessary part of using the EPIC_BEM model in any practical application.

6.2.1 Method for computing internal kinematics

Within any BEM solution of a numerical wave tank (NWT) the fluid velocities are known or calculated on the boundaries of the computational domain as these are required to compute the free surface boundary conditions that drive the fluid flow. To compute the water particle kinematics internal to the domain (henceforth referred to as the "internal kinematics"), the boundary integral equation (2.2) can be employed with a modified fundamental solution to reflect the fact that the velocity vector is given by the derivative of the potential with respect to space. It its original form, the boundary integral equation (BIE) contains a fundamental solution based on potentials only. Once the BIE has been modified, it is simply a question of integrating the new equations in a similar manner to that set out in §2.4.6.

Clearly, any position internal to the domain cannot fall directly on a boundary node. It therefore follows that no explicitly singular integral exists. However, the accuracy of the velocity components obtained for points computed near the boundary nodes may be poor because the Gauss-Legendre numerical quadrature scheme struggles to produce accurate results as the integral function tends towards a singularity. This was previously discussed in §2.4.6. A partial solution to this problem is discussed later in this section and is performed as part of an adaptive integration scheme.

In deriving expressions for the computation of the velocity vector at a given point internal to the domain, the BIE must first be differentiated with respect to the three Cartesian directions. Taking the original BIE, equation (2.2), and rearranging gives

$$c_p\phi_p = \int_\Gamma \left[G\frac{\partial \phi_q}{\partial n} - \phi_q \frac{\partial G}{\partial n} \right] d\Gamma, \tag{6.1}$$

where the fundamental solution $G(r) = \frac{1}{4\pi r}$ and $r = |\mathbf{r}|$; all parameters having previously been defined on Figure 2.1. Differentiating the fundamental solution with respect to the Cartesian directions, in vector form, gives,

$$\nabla\phi_p = \int_\Gamma \left[Q\frac{\partial \phi_q}{\partial n} - \phi_q \frac{\partial Q}{\partial n} \right] d\Gamma, \tag{6.2}$$

$$Q(r) = \frac{\partial G(r)}{\partial \mathbf{x}} = \frac{1}{4\pi r^3}\mathbf{r}, \tag{6.3}$$

$$\frac{\partial Q}{\partial n}(r) = \frac{1}{4\pi r^3}\left[\mathbf{n} - 3(\mathbf{r}\cdot\mathbf{n})\frac{\mathbf{r}}{r^2} \right], \tag{6.4}$$

where, again, the symbols have the same meaning as those outlined in §2.4.6. As internal points lie within the boundaries of the domain, $c_p = 1$ and has been removed from the equation.

In terms of positioning any internal kinematics calculations within the algorithm set out in §2.5, there is really only one suitable location. Equation (6.2) requires prior knowledge of both the potential and the potential flux at all points defining the boundary. Consequently, the only logical place to compute internal kinematics is following the formation and solution of the influence matrix within the "intermediate step" routine (see §2.5). To minimise the computational effort involved, the internal kinematics routines are only executed on the last call to the "intermediate step" routine within the time marching scheme. As a result, the values computed are in synchronisation with the coordinate, velocity and boundary condition vectors that are written to output files prior to undertaking the next time step.

6.2.2 Mitigating the so-called "boundary layer" problem

As mentioned previously, an important issue arises regarding the accuracy of the internal kinematics if the position at which the kinematics are required is close to a boundary. In the field of BEM solutions, this problem is commonly referred to as the "boundary layer" problem. In this context the term "boundary layer" refers to the proximity to the computational boundary and has nothing to do with a viscous shear layer commonly described in other areas of fluid mechanics. More explicitly, the "boundary layer" can be defined as any position internal to the domain for which

$$r \lesssim \frac{L}{2\sqrt{2}}, \tag{6.5}$$

where L is a length scale representative of length the of the element nearest to the point of interest (Christou, 2008).

Unfortunately, the problem of computing velocities at points within this so-called boundary layer cannot generally be avoided. The explanation for this is two-fold. First, the most interesting physics generally occurs close to the free surface boundary, not least because the boundary conditions applied on this surface are the primary source of the nonlinearity which makes the problem so difficult to solve. Second, the velocities arising close to this boundary have considerable importance in terms of practical engineering, especially in terms of the applied fluid loads.

Regrettably, there has been very little published work on the subject of computing integrals within this boundary layer. One exception is Sobey (2006) who makes the observation that the BIEs are poorly calculated due to the rapidly varying geometry that occurs as an internal point approaches a boundary. The solution proposed in Sobey (2006) is to form and solve a system of ordinary differential equations (ODEs). These provide a better description of the geometric variation and, as a consequence, reduce the steepness of the integral required from the BIE. Alternatively, Christou (2008) suggests that for a two dimensional BEM, applied to similar problems considered herein, increasing the number of evaluation points within the Gauss-Legendre quadrature scheme has a similar effect to that described in Sobey (2006) without the need to compute and solve systems of ODEs. Christou (2008) proposes an empirical relationship between the number of quadrature points required based upon the proximity of the internal point to the nearest element and its associated length scale. However, this scheme is computationally inefficient and somewhat cumbersome due to the arbitrary nature of the empirically derived relations. To overcome these difficulties a new scheme is proposed in the following section. This proves reliable with respect to computing accurate internal kinematics, particularly close to the domain boundaries, whilst at the same time minimising the required computational effort.

6.2.3 A self scheduling adaptive integration scheme for computing internal kinematics

The key factors in developing a numerical scheme for computing internal kinematics within the EPIC_BEM model were twofold. First, the scheme had to be robust in the sense that it had to always work and, without exception, produce an accurate result. Second, as the computational effort associated with computing the kinematics at one point is similar to that of computing one row of the influence matrices, a serial implementation of any scheme was completely out of the question; parallel computing was essential.

The core operation in the calculation of the internal kinematics is the evaluation of equation (6.1). Numerically, this involves exactly the same behaviour as that outlined in §2.4.6. Simply put, every node in every element is visited in turn and the integral, equation (6.1), is evaluated to form two row vectors similar to those found in the rows of the **H** and **G** influence matrices. These vectors are then multiplied by the known potential and potential fluxes and the velocity vector for that point is obtained.

To assist the robustness of the method, during the integration loop around all the nodes and all the elements, a function is called to assess the proximity of the internal point to the element under evaluation. Should the point fall within a length scale associated with the maximum length across the element currently queued for integration, a switch is triggered to demand adaptive integration to be used for this element. If the point does not fall within this length scale a standard, typically ten point, Gauss-Legendre quadrature is used (details of which are given in §2.4.6.)

If the adaptive integration switch is triggered for an element, the 2-norm of both integrals in equation (6.1) with respect to each direction is computed. The number of integration points is then doubled and the 2-norms recomputed. An arbitrary convergence of 1×10^{-5} between subsequent integrations for all Cartesian directions and both integrals in equation (6.1) is required for the accuracy of the integral to be deemed satisfactory. Should the number of integration points required for convergence exceed 5000, after Christou (2008), the point is simply discarded because the computed velocities are deemed too unreliable to be considered useful.

In making the method efficient in time, the processors within the distributed computing environment can be used. Due to the adaptive work required in some cases for the integrals to converge, the evaluation of each point may take a variable amount of time. As a result, the one dimensional decomposition outlined in Chapter 3 is unsuitable. Instead, a simple self scheduling environment is employed where each computational process is given an internal point at which kinematics are required. When it has finished computing kinematics for that point it will send back the result to a controlling process and request another, the process continuing until all the required points are evaluated. The water particle kinematics calculated using this method are validated later in §6.3.2.

6.3 Model validation

Within this section the numerical accuracy of the EPIC_BEM model is considered. A number of comparisons are made between existing analytical wave theories, other available numerical models, and the EPIC_BEM model. In all cases the success of the present model is clearly demonstrated indicating that the code is well founded.

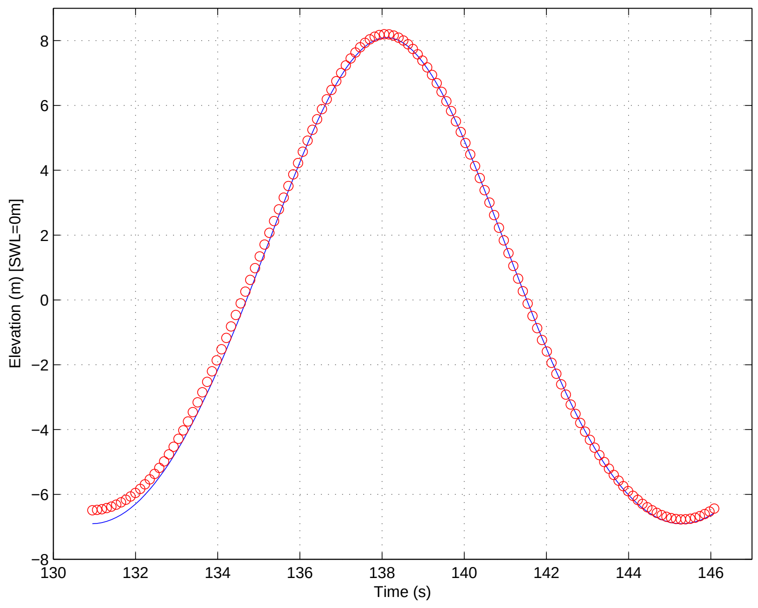
6.3.1 Prediction of the water surface elevation

Regular wave forms

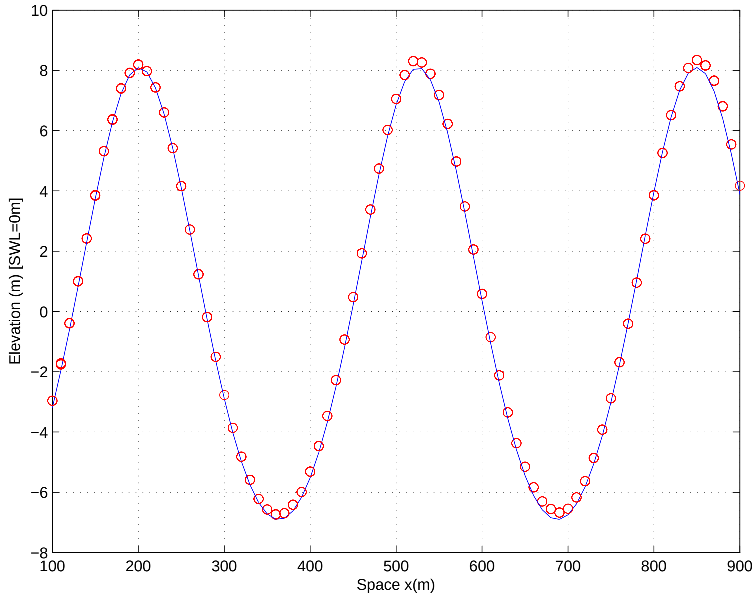
To begin, comparisons are made with the classical 5^{th}-order steady wave solution of Fenton (1985) which is itself extended from the 2^{nd} order solution of Stokes (1847). The NWT adopted had the following dimensions, $0m \leq x \leq 1000m$, $-80m \leq y \leq 0m$ and $-126.4m \leq z \leq 0m$, with a discretisation of $\Delta x = \Delta y = 10m$ used in the horizontal directions, (x, y), and $\Delta z = 12.64m$ in the vertical direction, (z). The boundary Γ_{input} at $x = 0m$ was prescribed in terms of a time dependent Φ_n, based upon the solution of Fenton (1985). The two boundaries running in the x direction (at $y = -80m$ and $y = 0m$) were prescribed in terms of a no-flux condition such that they acted as reflective walls. The radiation boundary, Γ_{rad}, employed a Sommerfeld radiation condition (Sommerfeld, 1949) and $170m$ of numerical sponge condition was adopted (§2.4.1).

A regular wave with a period of $T = 14.36s$ and a wave height of $H = 15m$ was introduced into the NWT, with an initial ramping up period of $3s$. This period is necessary to mitigate numerical shock waves and avoid the excitation of a longitudinal resonance or seiching within the NWT. Figure 6.1(a) considers a single wave period within the developed wave field and contrasts the results of the EPIC_BEM solution with the analytical Stokes solution (Fenton, 1985). Despite the steepness of the wave form under consideration ($Hk/2 = 0.2967$, where H is the wave height and k the wave number) the agreement is very good. The small departures undoubtedly being due to the onset of Benjamin-Feir type instabilities (Benjamin & Feir, 1967), that are real, incorporated within the EPIC_BEM solution, but omitted in the analytical model.

Figure 6.1(b) provides further comparisons of this wave case, providing a description of the water surface elevation in the spatial domain. Once again, the comparison between the EPIC_BEM solution and the analytical solution is very good.



(a) Time history of the water surface elevation, $\eta(t)$, for a steady wave form, comparisons between the EPIC_BEM solution (○) and the analytical model of Fenton (1985) (—).



(b) Spatial history of the water surface elevation, $\eta(x)$, for a steady wave form, comparisons between the EPIC_BEM solution (○) and the analytical model of Fenton (1985) (—).

Figure 6.1: Comparisons in time and space of the water surface elevation for a steady wave form.

Irregular wave forms

Due to the simplicity of their formulation, regular waves are a useful tool for validation purposes. However, they do not challenge the model in terms of producing a representation of a wave arising in a realistic sea state. As discussed previously, a real sea state is directional, unsteady and nonlinear in nature. Unfortunately, there are no analytical models capable of accurately modelling all these properties, hence the development of models such as the present one. It therefore follows that to provide a set of analytical results against which to validate the EPIC_BEM model, some of the properties of a real sea state must be simplified. The easiest way of doing this is through the use of a linear random wave theory, where the surface profile takes the form of the superposition of wave components, each assumed to be sinusoidal in form. Such an approach includes the unsteadiness and directionality, but negelects the nonlinearity. As a result, it is appropriate to the description of smaller (linear) waves. In adopting this approach, it is common practice to define a frequency spectrum and a directional spread, and to give each component of the superposition a different amplitude, frequency, phase and direction of propagation. These components are then summed according to

$$\eta(x,y) = \sum_{i=1}^{n} a_i \cos(k_{xi}x + k_{y_i}y + \omega_i t + \psi_i), \tag{6.6}$$

where η is the surface elevation, a is the wave amplitude, (k_x, k_y) is the wave number resolved into the x and y directions respectively, ω is the wave frequency, t is the time, ψ is the phase, and the subscript i refers to the i^{th} wave component which is summed from $i = 1$ to n; n being the total number of components. Associated with this surface profile is a velocity potential given by

$$\phi(x,y,z) = \sum_{i=1}^{n} a_i\omega_i \frac{\cosh(k_i(z+d))}{\sinh(k_i d)} \cos(k_{xi}x + k_{y_i}y + \omega_i t + \psi_i), \tag{6.7}$$

where the symbols represent the same quantities as those noted above with the addition of $\phi(x,y,z)$ being the velocity potential and d the local water depth. Assuming all the wave components are freely propagating, the wave number k and the angular frequency ω are related by linear dispersion equation,

$$\omega^2 = gk\tanh(kd), \tag{6.8}$$

where g defines the acceleration due to gravity.

The irregular wave form chosen for validation is based on the focusing of a JONSWAP spectrum to create a *NewWave* event (Tromans *et al.*, 1991). This latter solution described the most probable shape of a large linear wave based upon the superposition (or focussing) of the underlying wave crests; effectively setting $\psi_i = 0$ at $x = y = t = 0$ in equation (6.6). In the present example, the spectrum comprises 128 components with a peak period of $T_p = 16.64s$, a linear amplitude sum of $A = \sum_{i=1}^{n} a_i = 4m$ and a peak enhancement factor of $\gamma = 2.5$; the latter value being representative of a realistic sea state. The spectral content was directionally spread over 76 directions, according to a Mitsuyasu (1975) spreading parameter of $s = 7$. Again, this is a realistic value (Jonathan & Taylor, 1997) and corresponds closely to a wrapped normal distribution with a standard deviation of $\sigma_\theta = 30°$. This directional spread was applied uniformly to all frequency components with θ lying in the range $-\frac{\pi}{3} \leq \theta \leq \frac{\pi}{3} rad$, $\theta = 0$ indicating the mean direction of wave propagation.

In undertaking these calculations the NWT used had the following dimensions, $-720m \leq x \leq 600m$, $-720m \leq y \leq 0m$ and $-126.4m \leq z \leq 0m$, with a discretisation of $\Delta x = \Delta y = 15m$ and $\Delta z = 10.53m$. The boundary Γ_{input} at $x = -720m$ and the boundary running in the x direction at $y = -720m$ were prescribed a value of Φ_n derived from equation (6.7). The boundary running in the x direction at $y = 0$ was prescribed a no-flux condition such that it acted as a reflective wall. The radiation boundary Γ_{rad} employed a Sommerfeld radiation condition and $25m$ of numerical sponge condition was used, the combination of these conditions providing effective absorption/transmission at the downstream boundary (§2.4.1). In this form the NWT is symmetrical about the plane $y = 0m$. This is consistent with the evolving wave field, and only simulating half the physical domain leads to a significant computational saving.

Figure 6.2 concerns a time-history of the water surface elevation taken at $x = 0m$ for the focussed wave event described above. The agreement between the EPIC_BEM calculations and the analytical solution (based on equation (6.6)) is clearly very good. Indeed, the only discrepancy is small and arises in the leading wave trough (at $t \approx -8s$). It is believed that this is due to the highest frequency wave components having not reached the focus position at $t \approx -8s$. However, in the following wave trough (at $t \approx +8s$) it is clear that they have reached the focus position ($x = y = 0m$) and no such discrepancy arises.

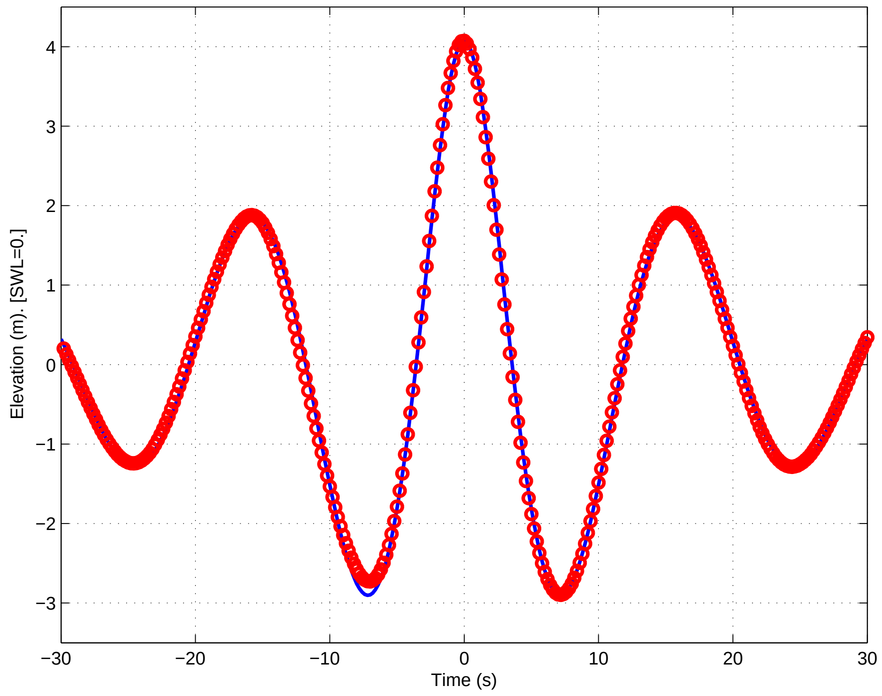
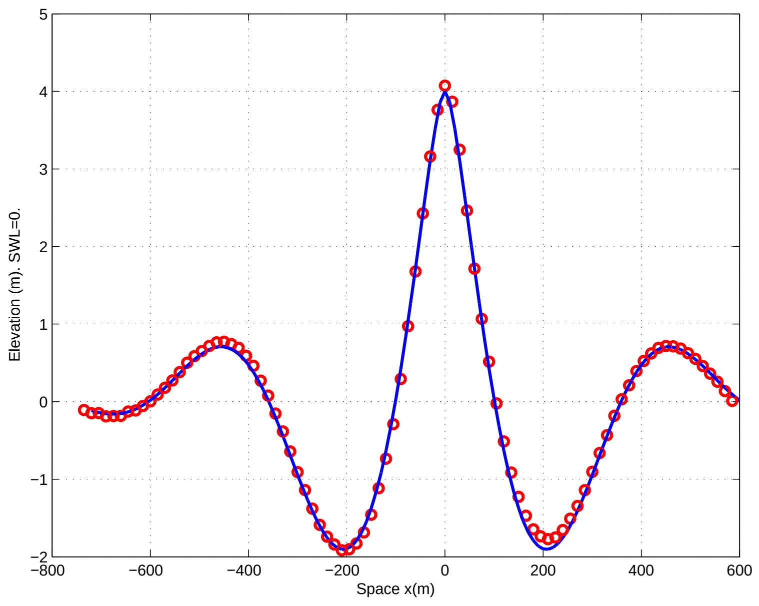


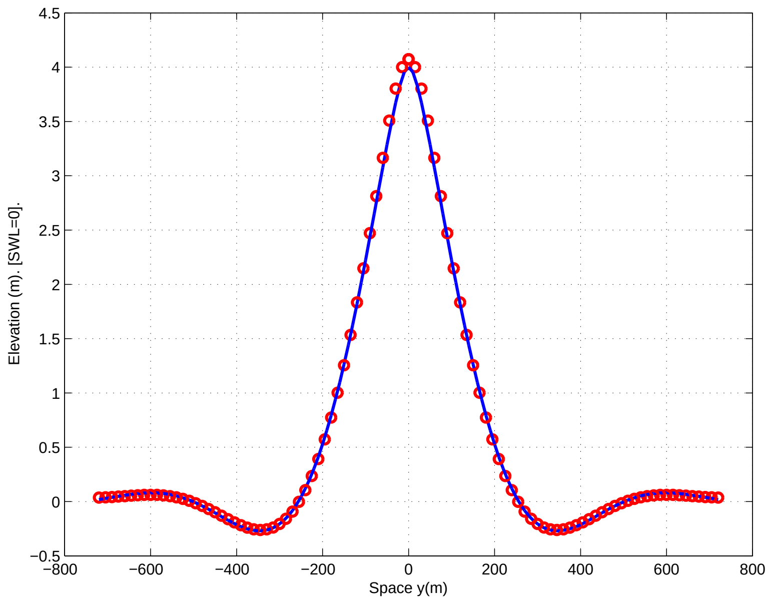
Figure 6.2: Time-history of the water surface elevation, $\eta(t)$, for a linear focused *NewWave* event, comparisons between the EPIC_BEM solution (○) and an analytical solution based on linear random wave theory (——).

Figures 6.3(a) and 6.3(b) concern the same focused *NewWave* event, providing spatial profiles of the water surface elevation in the x and y directions respectively. Once again, the agreement between the EPIC_BEM solution and the analytical solution is very good. The only small discrepancy arises in the trough immediately down stream of the focus event in x, the explanation for this being the same as that described above in relation to Figure 6.2. The data presented in Figure 6.3(b) directly relates to the directional spread, $\sigma_\theta = 30°$ or $s = 7$, and confirms that this has, indeed, been correctly introduced.

Setting aside the accuracy of the temporal and spatial comparisons provided in Figures 6.2–6.3(b), it is possible to perform further checks to ensure that the focused wave has the desired spectral content. Figure 6.4 contrasts the data aris-



(a) Spatial history of the wave surface elevation in the mean wave direction (x) at the focus time, $\eta(x)$ at $t = 0s$, comparisons between the EPIC_BEM solution ($\circ$) and the results of linear random wave theory (——).



(b) Spatial history of the wave surface elevation in the transverse wave direction at the focus time, $\eta(y)$ at $t = 0s$, comparisons between the EPIC_BEM solution ($\circ$) and the results of linear random wave theory (——).

Figure 6.3: Spatial histories of the water surface elevation for an irregular wave form.

ing from a discrete Fourier transform of the water surface elevation defined in Figure 6.2, with the required spectral shape (JONSWAP, *NewWave*). It is clear from this comparison that the generated wave spectrum is very close to the target specified. In addition, a discrete Fourier transform also defines the phasing of the components present in the signal. If the generated wave event is truly focused, all of the wave components present within the spectrum with a non-negligible amplitude will have a phase angle equal to zero ($\psi_i = 0$) at the focus location. Figure 6.5 shows the phasing of the generated wave components. By superimposing a normalised amplitude spectrum, it is clear which wave components have a non-negligible amplitude. Evidently, the data presented Figure 6.5 indicates that all such components effectively have a phase angle equal to zero. This confirms that the wave event generated within the NWT is correctly focused. In combination, the data presented on Figures 6.3(b), 6.4 and 6.5 confirm that the desired *NewWave* event has indeed been generated.

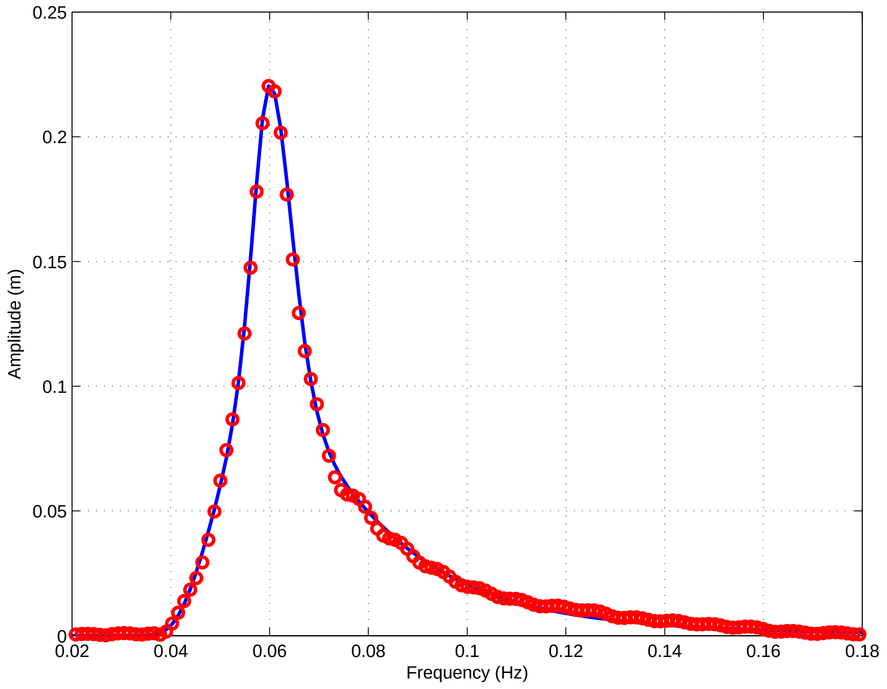


Figure 6.4: Comparison between the target *NewWave* spectrum (——) and the results of a discrete Fourier transform of the time history of the water surface profile (○), $\eta(t)$, given in Figure 6.2.

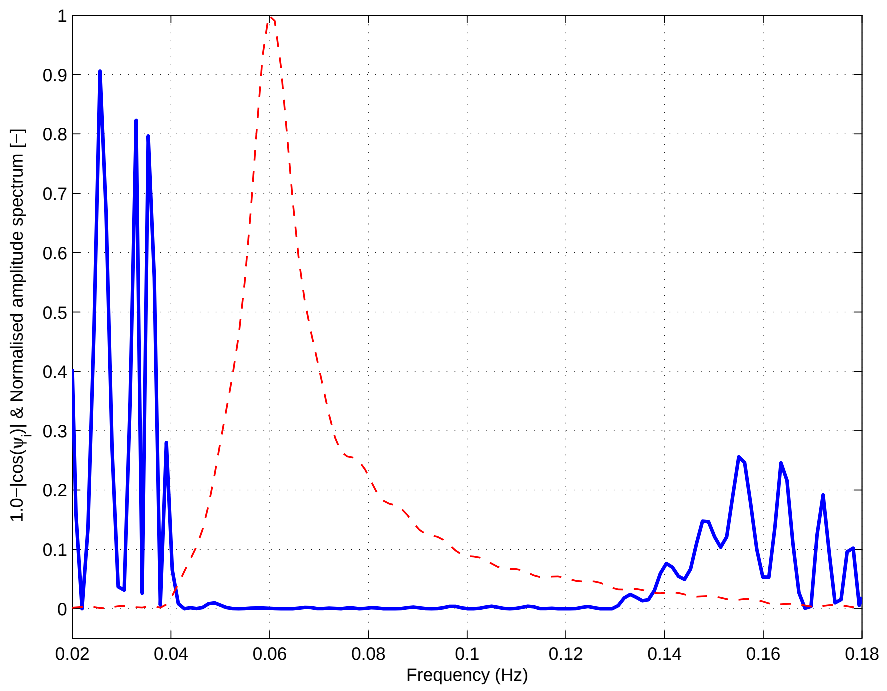


Figure 6.5: Phasing of the wave components present at the focus location. Phasing (——) and normalised amplitude spectrum (− − −).

6.3.2 Predictions of the water particle kinematics

Having demonstrated that the EPIC_BEM model is capable of accurately modelling both regular and irregular wave profiles, the fluid velocities arising at the water surface must be accurately predicted since it is these velocities which are used to formulate the boundary conditions on which the model is based. However, further validations of the predicted kinematics are necessary to ensure the accuracy of the velocity profiles with respect to depth (beneath the water surface) and the phasing of fluid velocities with respect to the surface profiles. Once again consideration is given to both regular (or steady) waves and irregular (or unsteady) waves.

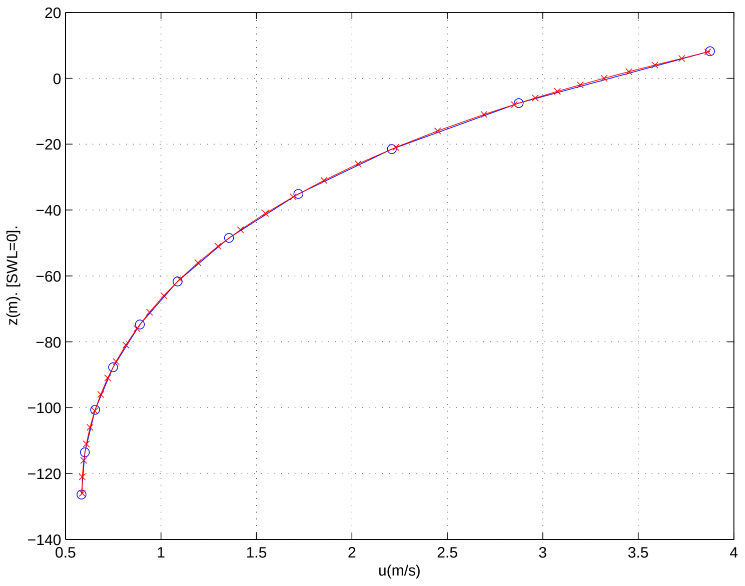
Regular waves

The details of the regular wave considered in this section are the same as those described in §6.3.1. In order to prove that the method for computing the "internal kinematics" is correct, its consistency with respect to water particle kinematics calculated using the "sliding elements" method, applied on a side wall or vertical boundary (as described in §2.4.8) is presented in Figure 6.6(a). Clearly, the difference between the solutions is negligible, confirming that the "internal kinematics" are consistent with the boundary kinematics. However, this does not offer proof as to the accuracy of the kinematics in comparison to known analytical theories. This issue is considered in Figure 6.6(b) which compares the 5^{th}-order regular wave solution of Fenton (1985), the Fourier series based model of Sobey *et al.* (1987) and the internal kinematics of the EPIC_BEM model. Again, the comparisons are extremely good; the small discrepancy, most noticeable at the bed, being associated with the development of a Lagrangian drift and its corresponding return flow, that is correctly predicated by the EPIC_BEM solution, but is not present in the analytical and Fourier theories.

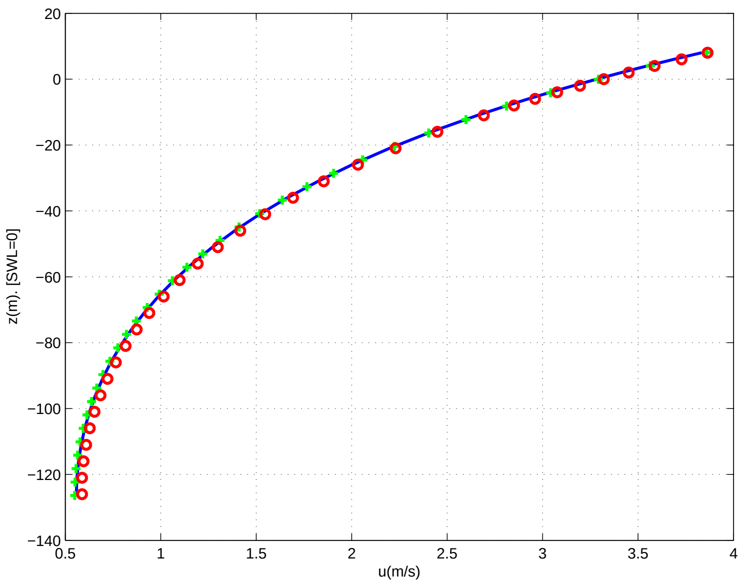
Irregular waves

The irregular wave form used in this validation is identical to that presented in §6.3.1. As with the regular wave forms, internal and boundary calculated kinematics are again compared. However, in the case of a directionally spread focused wave form, the directionality (or short-crestedness) will cause the profile shape to change with the y position for a fixed x location. Figure 6.7 compares the boundary kinematics calculated at $x = y = 0m$ (corresponding to the focus position) with the internal kinematics calculated at $x = 0m, y = -1m$. The agreement between these velocity profiles, $u(z)$ again confirms the success of the internal kinematics calculations, the small discrepancy between the calculated values being due to the difference in the y coordinate.

Figure 6.8 concerns the accurate modelling of the directional nature of the focused wave event. Within this figure comparisons are made between velocity predictions based upon a uni-directional focused wave, arising in the same underlying frequency spectrum, and the EPIC_BEM model for the directionally spread focused wave event. To correct for high frequency contamination occurring in the analytical solution the "stretching" algorithm of Wheeler (1970) is applied. In addition, a directional spread of $s = 7$ (Mitsuyasu, 1975) equates to a theoretically calculated velocity reduction factor (VRF) of $\Upsilon = 0.87$; the latter applied to ac-



(a) Prediction of the horizontal fluid velocities beneath the crest of a regular wave form. Comparisons between the internal EPIC_BEM kinematics calculations (—×—) and the EPIC_BEM calculations undertaken on a vertical side wall (—○—).



(b) Prediction of the horizontal fluid velocities beneath the crest of a regular wave form. Comparisons between the internal EPIC_BEM kinematics calculations (○) and two established wave solutions, a 5^{th}-order Stokes solution (——) and a high order Fourier solution (+).

Figure 6.6: Predictions of the horizontal fluid velocities beneath the crest of a regular wave form.

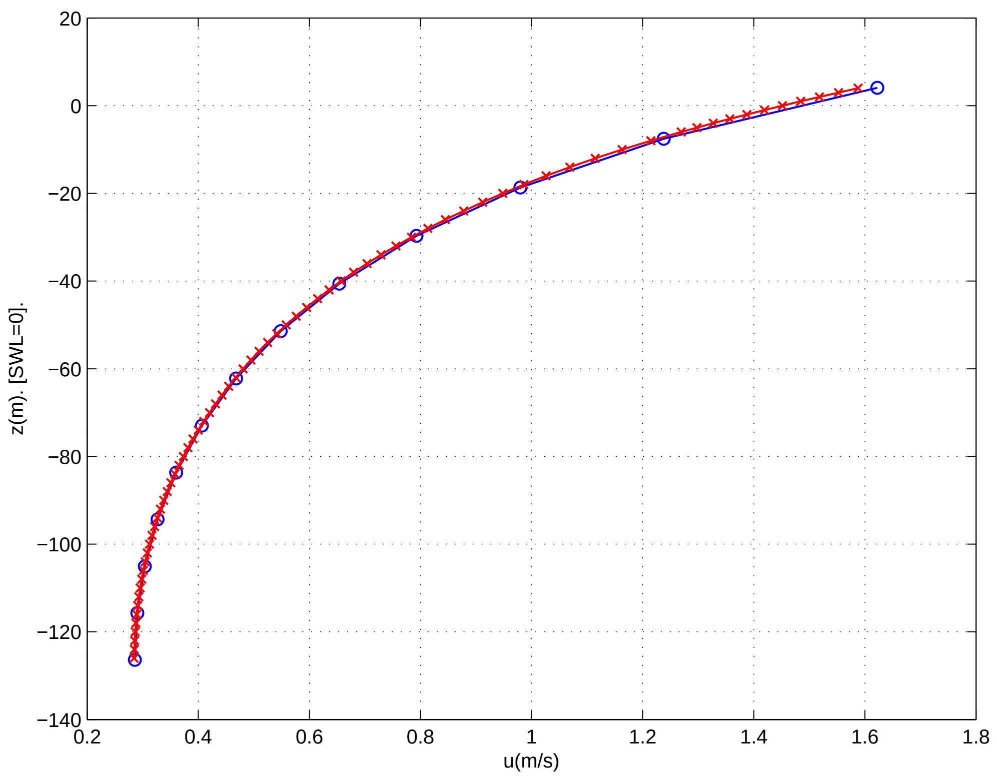


Figure 6.7: Prediction of the horizontal fluid velocities beneath the crest of a focused wave event, comparisons between the internal EPIC_BEM kinematics calculations (—×—) and the EPIC_BEM calculations undertaken on a vertical side wall (—○—).

count for the lack of directionality in a uni-directional wave. Having applied these two corrections (Wheeler stretching and a velocity reduction factor of $\Upsilon = 0.87$), Figure 6.8 demonstrates good agreement between the EPIC_BEM model and the corrected uni-directional velocity profile. This agreement confirms that the input to the EPIC_BEM model correctly described the target directional distribution ($s = 7$), the calculated kinematics being entirely consistent with the water surface elevation, $\eta(y)$, presented in Figure 6.3(b)

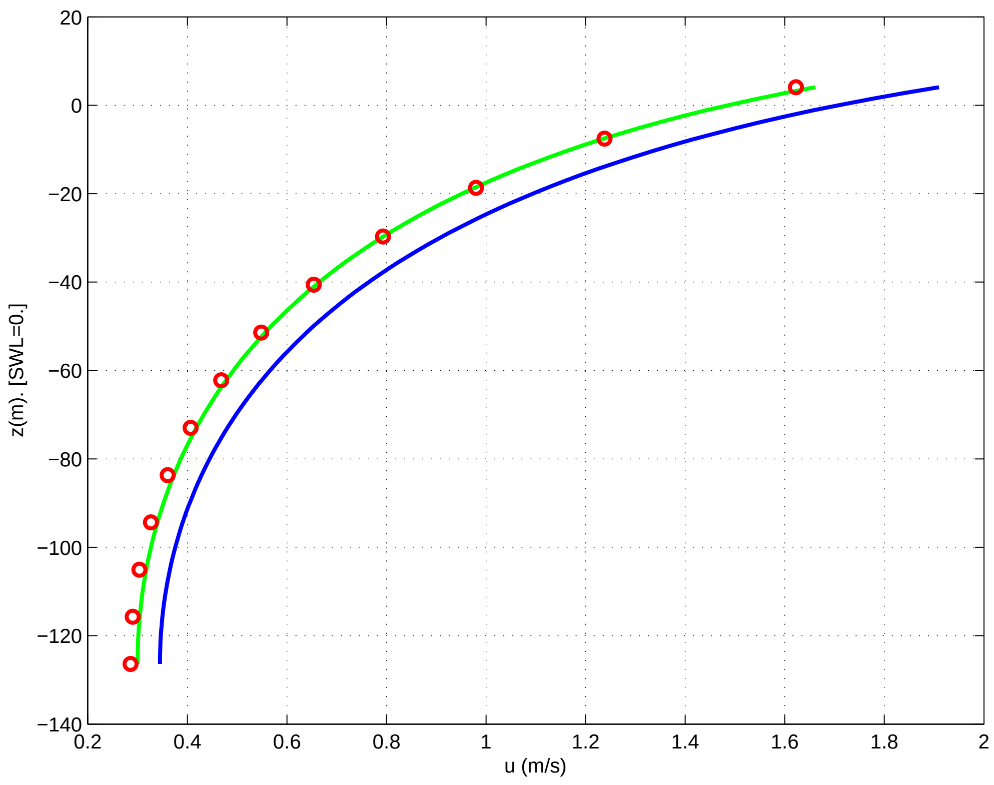


Figure 6.8: Comparisons between the horizontal velocity profile, $u(z)$, predicted using the EPIC_BEM model (○) and a linear, uni-directional solution (——). The linear solution is corrected using Wheeler stretching and a velocity reduction factor of $\Upsilon = 0.87$, the latter corresponding to $s = 7$ (——).

Finally, Figure 6.9 depicts the relative phasing of the fluid velocities (u, v, w) calculated using the EPIC_BEM solution. The data provides time-histories of the three velocity components $(u(t), v(t), w(t))$ at a single position, ($x = y = 0m$ and $z = -3m$), and superimposes these results on the time-history of the water surface elevation, $\eta(t)$. Comparisons between these results show the velocity records are exactly as expected, $u(t)$ is in phase with $\eta(t)$, $w(t)$ is phase shifted by 90°, and $v = 0m/s$ due to the symmetry of the wave event.

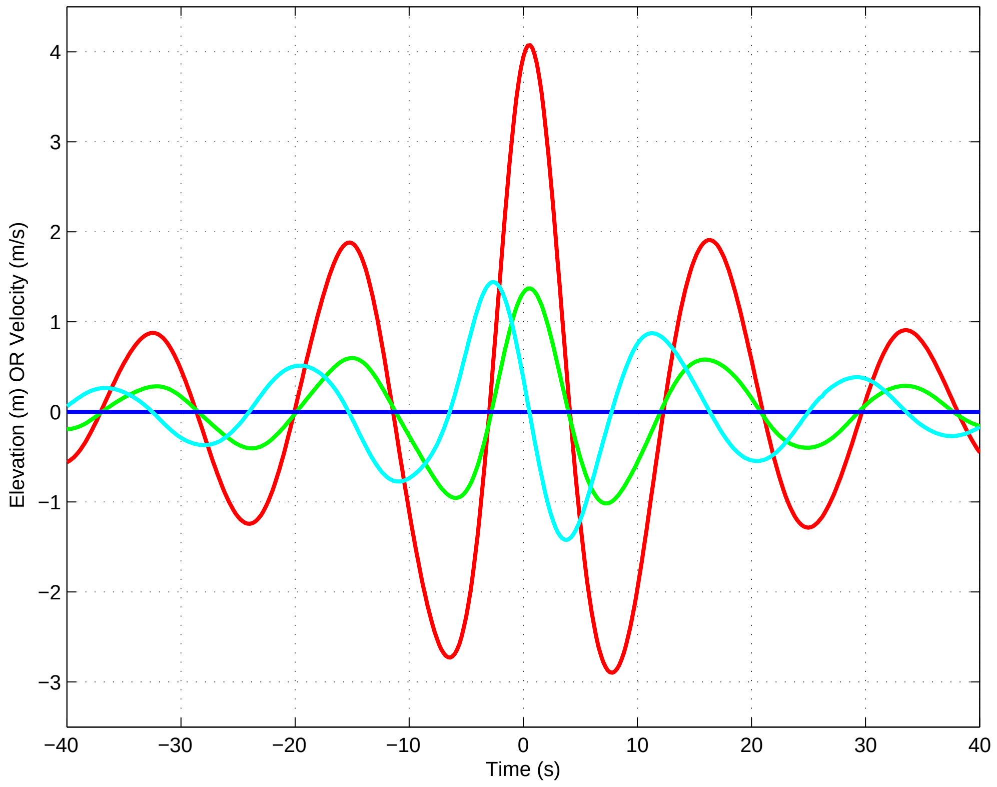


Figure 6.9: Time-histories of $\eta(t)$ (🟥), $u(t)$ (🟩), $v(t)$ (🟦) and $w(t)$ (🟦) at $z = -3m$ beneath a focused wave crest.

6.4 Calculations of a 10^{-4} design wave case

Having validated the proposed wave model, attention is turned to using the model in a practical application. This section primarily discusses the properties of real seas and goes on to outline an extreme design wave event, typical of those used in industry in the appraisal of offshore structures. The section continues with a description of the model configurations that will be used throughout the remainder of this chapter, and provides some results relating to the convergence of the chosen configurations. Finally, results concerning a 10^{-4} (annual probability of exceedance) design wave case are presented and contrasted to a comparable Stokes 5th-order regular wave solution. The Stokes 5th-order regular wave solution being commonly adopted by industry in both the design and re-assessment of offshore structures, both fixed and floating.

6.4.1 Properties of real seas

To emphasise the potential importance of the calculations undertaken using the EPIC_BEM code, it is important to contrast the characteristics of a steady wave solution, such as a Stokes 5^{th}-order model, with those arising from the EPIC_BEM model. The EPIC_BEM model providing an accurate and realistic representation of the design wave event.

Unsteady or transient behaviour

The unsteadiness of a wave field relates to its evolution in space and time and arises from the fact that any sea state is made up from a number of wave components with different frequencies, amplitudes, directions and phasing. The most important characteristics are defined by the frequency and directional spectra. Real waves are therefore made of wave components travelling at different velocities and with different relative phasing, the combined effect leading to a free surface that evolves rapidly in both space and time. In the context of fully nonlinear wave modelling, the EPIC_BEM model can easily simulate the underlying frequency spectrum and can hence model the unsteady behaviour of waves. In contrast, a Stokes 5^{th}-order wave solution is steady in that the surface profile propagates without changing shape. Having failed to incorporate the appropriate evolution, a steady wave solution is also limited in terms of the local wave steepness. The combination of steady behaviour and limited steepness leads to the predictions of unrealistic wave profiles when compared to extreme waves arising in real seas. It therefore follows that the associated predictions of the water particle kinematics will be equally unrealistic, particularly when close to the water surface.

Directionality or short-crestedness

As mentioned above, a real sea consists of a large number of wave components, all potentially travelling in different directions. The directional properties are characterised by a directional spectrum, or approximated by a directional spread, based upon an assumed Gaussian or cosine-squared distribution. The directionality of a sea is important in that it will create a surface profile that varies in all spatial directions. As a consequence, the water particle kinematics will also vary in all spatial directions. Unfortunately, a steady wave solution such as a Stokes 5^{th}-order solution is, by definition, unidirectional and cannot therefore model the directional nature of a real sea. As a result, simplified corrections need to be applied such as the velocity reduction factors applied in §6.3.2. In contrast, the EPIC_BEM model correctly simulates the directional nature of a sea state, both in terms of the surface profile and the associated water particle kinematics.

Nonlinear effects

The nonlinear effects arising within a real sea state are both numerous and complicated. First, nonlinear interactions between individual wave components can lead to significant changes in the local wave spectrum. The effects arising at second and third order are most significant, leading to an enhanced crest-trough asymmetry, and in deep water, a transfer of wave energy to the higher frequency wave components. These effects will increase the magnitude of water particle velocities arising close to the water surface. A Stokes 5^{th}-order wave solution cannot hope to model these effects, not least because the wave solution is based on a single frequency component and its associated bound harmonics. In contrast, the EPIC_BEM model correctly incorporates the correct underlying frequency spectrum and applies the fully nonlinear boundary conditions. As a result, the energy transfers noted above will be correctly modelled.

In addition to the local changes in the wave spectrum, the nonlinear wave-wave interactions can also affect the local directionality of a large, isolated wave event. As a result, the largest waves tend to be more long crested than would be expected based on the underlying sea state. This also has important implications for the fluid velocities and hence the applied fluid loads, and is correctly incorporated within the EPIC_BEM code. Unfortunately, a Stokes 5^{th}-order solution is uni-directional and neglects the underlying spectral shape. Subtle changes in the directionality, irrespective of how important they are for the kinematics predictions, are completely omitted.

Finally, it is most important to note that the effects of nonlinearity increase with the steepness of a local wave form. Since design wave conditions will necessarily involve the description of very large, steep waves, the practical importance of the nonlinearity should not be underestimated.

6.4.2 Model configuration for the 10^{-4} design conditions

Model configuration

The NWT tank used for the simulation had the same dimensions as those outlined in §6.3.1, the only difference being the discretisation employed. In the x and y directions the nodes were spaced at $\Delta x = \Delta y = 10m$, whilst in the z direction, $\Delta z = 9.72m$. To confirm the applicability of this discretisation, the focused wave described in Table 6.1 (see below) was simulated with a linear input amplitude sum of $A = 16m$. For this case a discretisation of $\Delta x = \Delta y = 20m$ and $\Delta x = \Delta y = 15m$ was compared to $\Delta x = \Delta y = 10m$. Details of this comparison are provided in Figure 6.10 in which the time histories of the water surface elevation, $\eta(t)$ at the focus position, $x_f = 60m$, are superimposed. The differences between the data arising from the three different discretisations are small and convergence is clearly seen to occur; evidence of the latter provided by the close-up of the wave crest which is provided in the small insert. This data confirms the suitability of a $\Delta x = \Delta y = 10m$ discretisation for the modelling of the 10^{-4} design wave case described in Table 6.1.

10^{-4} *design conditions*

Table 6.1 lists a set of parameters that describe a 10^{-4} wave event; the actual data relating to a hindcast of a tropical cyclone in the Southern hemisphere. The EPIC_BEM model was set up with these parameters distributed in the same manner as described in §6.3.1 and simulated in the NWT described in above. As the evolution of extreme waves is known to be highly nonlinear, a number of iterations were required to match the crest elevation specified in Table 6.1. The results of these preliminary calculations confirmed that a linear amplitude sum of

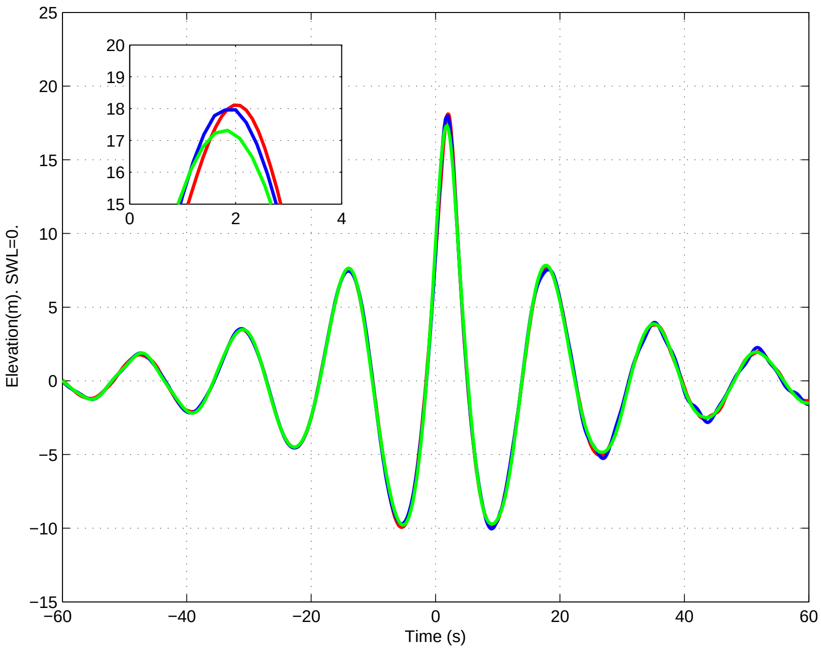


Figure 6.10: A comparison of the time history of the water surface, $\eta(t)$, at focus position for a linear input amplitude sum of $A = 16m$ simulated with $\Delta x = \Delta y = 10m$ (🟥), $15m$ (🟦) and $20m$ (🟩) resolutions. Note: the small insert depicts a close up of the focused crest region.

Property	**Value**	**Comment**
Spectral shape	JONSWAP, NewWave	NewWave theory following Tromans *et al.* (1991)
Peak period (T_p)	$16.64s$	-
Peak enhancement factor, γ	2.3	-
Directional spread	$s = 7$ or $\sigma_\theta = 30°$	Mitsuyasu (1975) spreading or equivalent wrapped normal distribution
Water depth (d)	$126.4m$	Includes storm surge
Wave crest elevation (η_{max})	$21.98m$	Maximum single crest elevation
Wave height (H)	$35.24m$	Corresponding wave height
Courant number (C_0)	0.4	Modelling parameter chosen to maintain numerical stability (see §2.4.10)

Table 6.1: Parameters relating to a typical 10^{-4} design wave event.

$A = 18.9m$ (equation (6.6)) gave a nonlinear wave crest elevation of $\eta_{max} = 22.19m$ and it was decided that this provided a suitable match to the specified 10^{-4} crest elevation of $21.98m$.

6.4.3 Discussion of results

To emphasise the importance of using the present nonlinear kinematics calculations, it is necessary to compare the nonlinear calculations to those derived from an "equivalent" 5^{th}-order Stokes solution. Figures 6.11(a) and 6.11(b) contribute to this comparison by respectively providing a time history, $\eta(t)$, and a spatial history, $\eta(x)$, of the water surface elevation for both the fully nonlinear EPIC_BEM and the Stokes 5^{th}-order calculations. Within these comparisons, the Stokes 5^{th}-order solution used was based on the 10^{-4} design conditions: $H = 35.24m$ and $T = 15.87s$, where the local wave period was based upon $T = 0.95T_p$. It is obvious from these comparisons that the fully nonlinear calculations describe a wave form that is very different from that predicted by the Stokes wave solution. In particular:

a) The wave crest elevation, η_{max} is higher.

b) The adjacent wave troughs (both preceding and following the largest wave crest) are less deep.

c) The overall wave height is slightly smaller.

d) The crest-trough asymmetry is significantly larger; almost two-thirds of the wave height (H) lying above the still water level (SWL).

e) The wave profile is steeper, with the steepest section of the wave profile occurring at significant elevation above SWL.

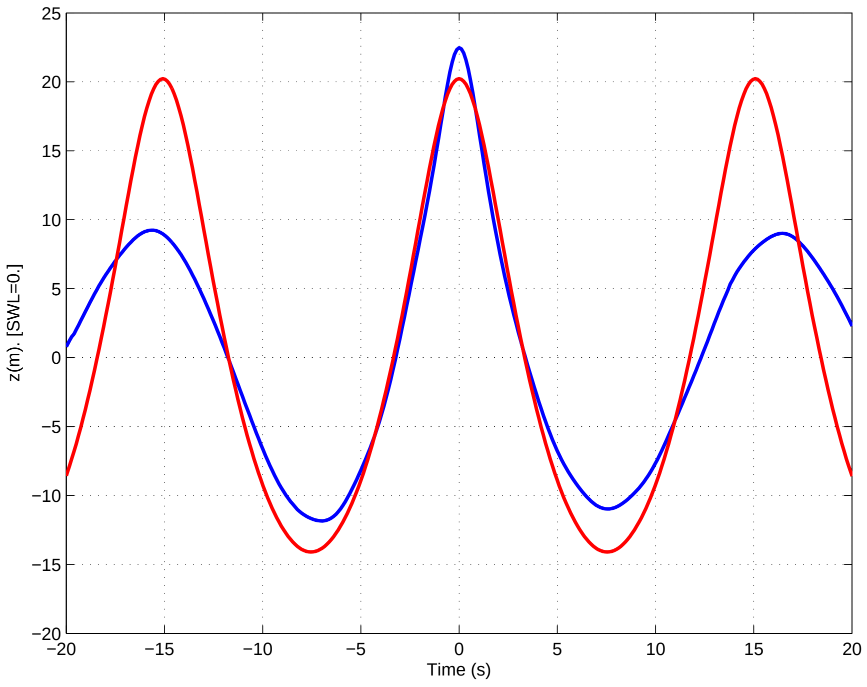
f) The wave crest is narrower in both space and time.

Additionally, Figure 6.12 provides a spatial description of the water surface profile in the y direction, $\eta(y)$, at the instant of wave focussing. No comparison with the "equivalent" Stokes solutions are available in the transverse direction since the Stokes solution is uni-directional and therefore infinitely long crested. In considering these results it is important to note that the nonlinear half wave period (from leading wave trough to maximum wave crest) is only $6.96s$. During this time interval the water surface elevation changes from a minimum of $-11.86m$ to a maximum of $22.19m$ ($H = 34.05$). This corresponds to an extremely steep wave profile, far in excess of that which could be sensibly represented by a Stokes 5^{th}-order wave solution.

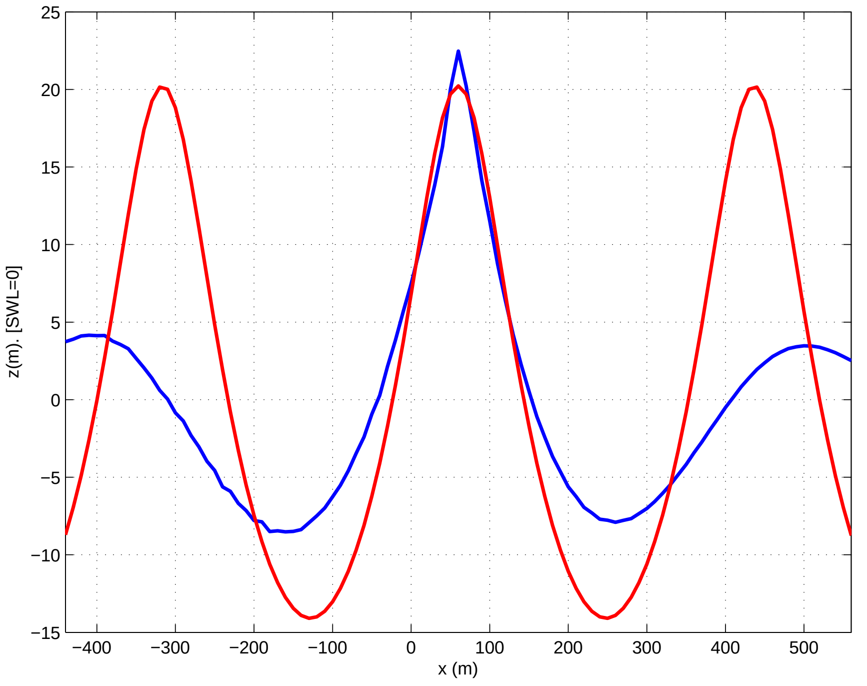
Furthermore, Figure 6.13(a) concerns the vertical variation of the maximum horizontal velocity arising beneath the crest of the focused wave event. Once again, comparisons are made between the predictions of the EPIC_BEM solution and a 5^{th}-order Stokes solution. Fitting the predicted velocity time-histories with a cubic spline interpolation allows the time derivative of the water particle kinematics and hence the unsteady component of the horizontal accelerations to be determined. Figure 6.13(b) depicts the vertical distribution of the maximum horizontal accelerations ($\frac{\partial u}{\partial t}$) occurring beneath the steepest part of the focused wave profile; comparisons between the present EPIC_BEM predictions and the results of a Stokes 5^{th}-order solution again being provided. The data provided on Figures 6.13(a) and 6.13(b) confirm that the fully nonlinear predictions show marked departures from a 5^{th}-order Stokes solution, the practical importance of these results being emphasised by the calculation of the applied fluid loads (see below). Taken as a whole, Figures 6.11-6.13 demonstrate the capabilities of the EPIC_BEM solution when applied to the description of an extreme (10^{-4}) wave event.

6.5 Practical application: load predictions

It is evident in the previous section that, in the case of a 10^{-4} design wave event, the fully nonlinear water particle kinematics predictions of the EPIC_BEM solution are substantially different to the Stokes 5^{th}-order solution commonly employed in industry. Wave-induced fluid loading is entirely a function of the water particle kinematics acting on a body. In particular, drag based loading is proportional to the square of the water particle velocity. As a result, even relatively small differ-



(a) $\eta(t)$



(b) $\eta(x)$

Figure 6.11: A 10^{-4} design wave case, comparisons between the EPIC_BEM solution (——) and an "equivalent" Stokes 5^{th}-order wave (——), (a) time history of the water surface elevation $\eta(t)$ and (b) spatial history of the water surface elevation $\eta(x)$.

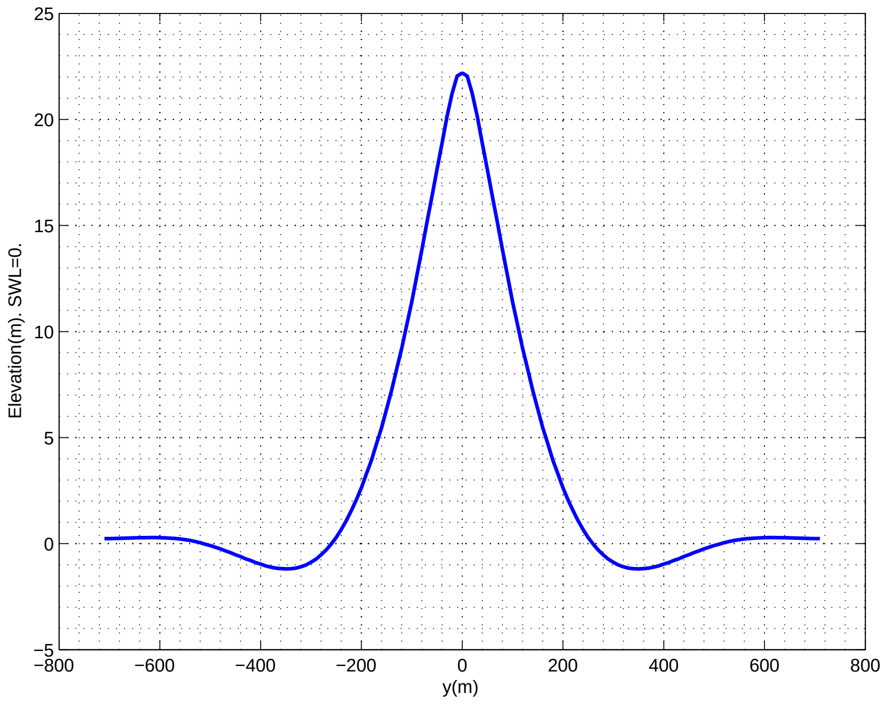
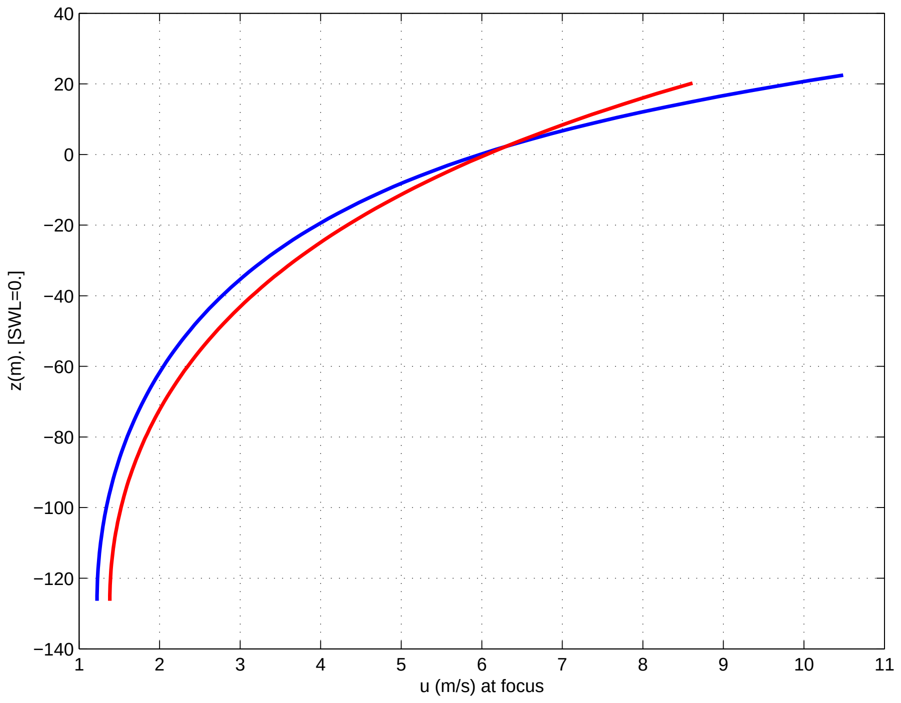
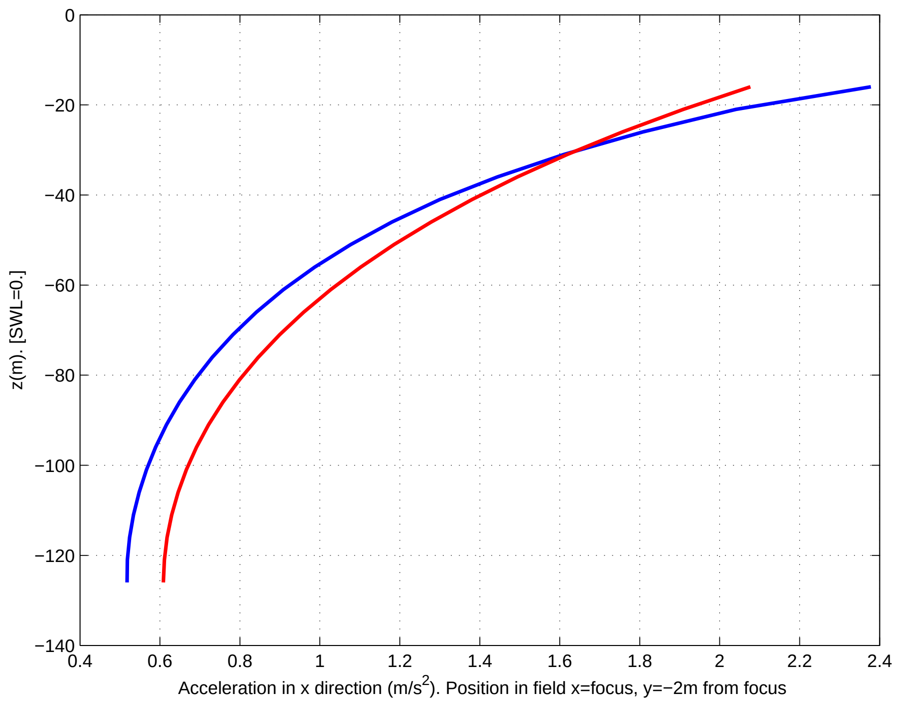


Figure 6.12: A spatial history of the water surface elevation in the transverse direction, $\eta(y)$, at the focus time. A 10^{-4} design wave case based upon $\eta_{max} = 21.98m$, $T_p = 16.64s$.



(a)



(b)

Figure 6.13: Kinematics predictions appropriate to a 10^{-4} design wave case, comparisons between the EPIC_BEM solution (——) and an "equivalent" Stokes 5th-order wave (——), (a) $u(z)$ and (b) $\frac{\partial u}{\partial t}(z)$.

ences in the predicted water particle kinematics based upon the EPIC_BEM and Stokes 5^{th}-order wave solutions may lead to significant differences in the predicated wave-induced fluid loading. This will be considered in the following section.

6.5.1 Fundamentals of fluid loading

Calculating the wave induced loads acting on an offshore structure is both complicated and central to the design process. The loads are related to the wave-induced fluid velocities, the wave-induced fluid accelerations and any wave-structure interactions; the latter leading to a local modification of the incident wave field due to the occurrence of wave diffraction. To reduce this complexity, it is common to adopt simplifications based on the flow regime in which the body is present. An empirical relation, derived primarily from experimentation and commonly referred to as the Keulegan-Carpenter number (KC), is used for this purpose. This, non-dimensional quantity, is defined by

$$KC = \frac{UT}{D}, \tag{6.9}$$

where U is the amplitude of the horizontal wave-induced orbital velocity, T is the wave period and D is a length scale based on the diameter of the structural member under consideration. In flow regimes with a low KC number, typically $KC < 5$, the fluid loading will be potential. If $D/\lambda < 0.2$, where λ is the incident wave length, these potential loads will be dominated by inertia forces. In contrast, if $D/\lambda > 0.2$, wave diffraction also becomes important and the nature of the wave-structure interaction needs to be taken into account. Alternatively, if $KC > 20$, the fluid loading is dependent on viscous forcing and characterised as being drag dominated. In many practical structures, including all jacket (or space frame) structures, the size of the individual members is such that there is no significant disturbance of the incident wave field (diffraction effects being negligible). In such cases the loads applied to the structure are referred to as slender body loads and the structure said to lie within the so-called *drag-inertia* regime. Within this regime it is common to apply the “Morison’s equation” (Morison *et al.*, 1950). This relates the force acting on a structural member to the sum of the drag and inertial forces

$$F = \underbrace{C_d \frac{1}{2} \rho D u |u|}_{drag\ term} + \underbrace{C_m \rho \frac{\pi}{4} D^2 \frac{\partial u}{\partial t}}_{inertia\ term}, \tag{6.10}$$

where F is the force per unit length acting on a cylindrical member of diameter D in a fluid of density ρ. C_d and C_m are empirically derived loading coefficients relating to drag and inertia respectively, whilst u is the incident wave-induced velocity and $\frac{\partial u}{\partial t}$ the corresponding unsteady acceleration.

Equation (6.10) frequently forms the basis for computing global loading; global loading in the sense of a total load induced by fluid flow incident to the sub-structure. For each member in a sub-structure the equation is applied taking into account the local fluid velocities, the local accelerations and the dimensions of the member. Once the loading on all the individual members has been calculated the sum of these forces is computed to give a total base shear. Similarly, the forces on individual members can be multiplied by the appropriate moment arm (representing the distance from the sea bed, or foundations), and the sum of these moments can be used to define a total overturning moment. Taken together, the base shear and overturning moment provide appropriate measures of the externally applied global structural loading. In the calculations which follow, the sub-structural loading calculations were undertaken by WS Atkins using the WAJAC fluid loading software (Det Norske Veritas, 2010) based upon wave kinematics data provided by the author.

6.5.2 Loading recipes

In most industrial applications, there are governing bodies and international standards which produce guidelines for commonly undertaken tasks. In terms of the design of offshore platforms, the American Petroleum Institute (APInst) guidelines (American Petroleum Institute, 2000) is a commonly referenced document. In the "Design Loads" section of the guidelines appropriate to fixed structures it suggests that: *"The wave loads on a platform are dynamic in nature. For most design water depths presently encountered, these loads may be adequately represented by their static equivalents"*. The document continues with a description of the design procedure. It suggests that the wave loading induced by water particle kinematics should be based on an appropriate two-dimensional wave theory computed using predetermined parameters such as a wave height, associated period (to include a Doppler shift due to any co-existing current) and a storm water depth. It then states that, *"in many cases a Stokes V* [sic] *order wave theory will produce acceptable accuracy"* and goes on to explain the computation of total loads via Morison's equation (as given above), with some modifications to account for marine growth and other practical issues.

In the calculation of the applied loads on a structure, it is logical to assume that the largest and steepest waves will provide the critical conditions in terms of the wave elevation and the water particle velocities. Unfortunately, the APInst's suggested use of a Stokes 5^{th}-order wave (or similar) gives rise to a situation where the simulated wave does not adequately reflect the underlying physics governing the evolution of either the largest or the steepest ocean waves. In particular, it is well known that extreme ocean waves are transient or unsteady, directionally spread and highly nonlinear. Unfortunately, a Stokes wave solution is regular, or steady, uni-directional and only partially incorporates the correct nonlinearity. The practical consequences of this can readily be seen by employing a Stokes 5^{th}-order solution with a large wave height, H, and noting that the elevation achieved by the wave crest, η_{max}, is considerably lower than that expected. In addition, the crest-trough asymmetry is poorly represented, as is the maximum local wave steepness. Furthermore, with a poor representation of an extreme wave profile, coupled with inadequacies in the underlying physics, it is hardly surprising that large errors in the representation of the associated water particle kinematics also arise. When such solutions are used in the calculation of the drag loads, a 10% error in the predicted kinematics will lead to a 20% error in the predicted loads; a result clearly derived from equation (6.10). Furthermore, errors in a predicted wave form, over length scales comparable to that of the overall structure, may lead to additional inaccuracies in the applied global loading.

With respect to employing the EPIC_BEM model, it is clear that a more accurate representation of a specified design wave event and its associated water particle kinematics will inevitably lead to an improved prediction of the applied loads. Indeed, it is interesting to note that those closely involved with the initial formulation of the APInst guidelines (almost 20 years ago) appear to have anticipated this development. For example, in a review of the APInst guidelines, Heideman & Weaver (1992) noted in relation to the large uncertainty in the force predictions for individual waves that:

"to reduce this uncertainty significantly one would probably have to abandon the deterministic approach (i.e. Stokes 5^{th}-order)*, and more accurately model the surface and kinematics of individual waves in a three dimensional non-linear, random wave sense".*

It is exactly this reduction in uncertainty that the present study seeks to address.

6.5.3 Loading calculations

The structure used in the following calculations is not a conceptual design or test structure formulated for the purpose of this investigation, rather, it is a real life structure that is currently used for oil and gas production. The structure comprises a space frame construction and has a densely packed riser assembly located between the centroid and the North side of the platform (a schematic is provided in Figure 6.14). The structure resides in $126.4m$ of water and has a deck elevation of $19.975m$ above mean water level. The plan area of the jacket (substructure) is approximately $83.1m$ (North-South) $\times$ $67.7m$ (East-West). However, to allow for as much flexibility as possible in the placement of the wave crest, water particle kinematics calculations were carried out over an area of $240m$ by $120m$ in plan (this gives a working area of $240m$ by $240m$ because of the reflective symmetry in the model) with a discretisation of $\Delta x = \Delta y = 10m$. An exception to this discretisation was required for the points nearest the reflective boundary ($y = 0$), as they would lie directly on the boundary wall. As discussed previously, computing internal kinematics at points on or near boundaries is not desirable in the interests of accuracy, therefore points that would lie on the boundary wall were moved to $y = \pm 2m$.

In the z direction, internal kinematics were computed at $5m$ intervals from the bed to the elevation of the deepest wave trough. Above this level and up to

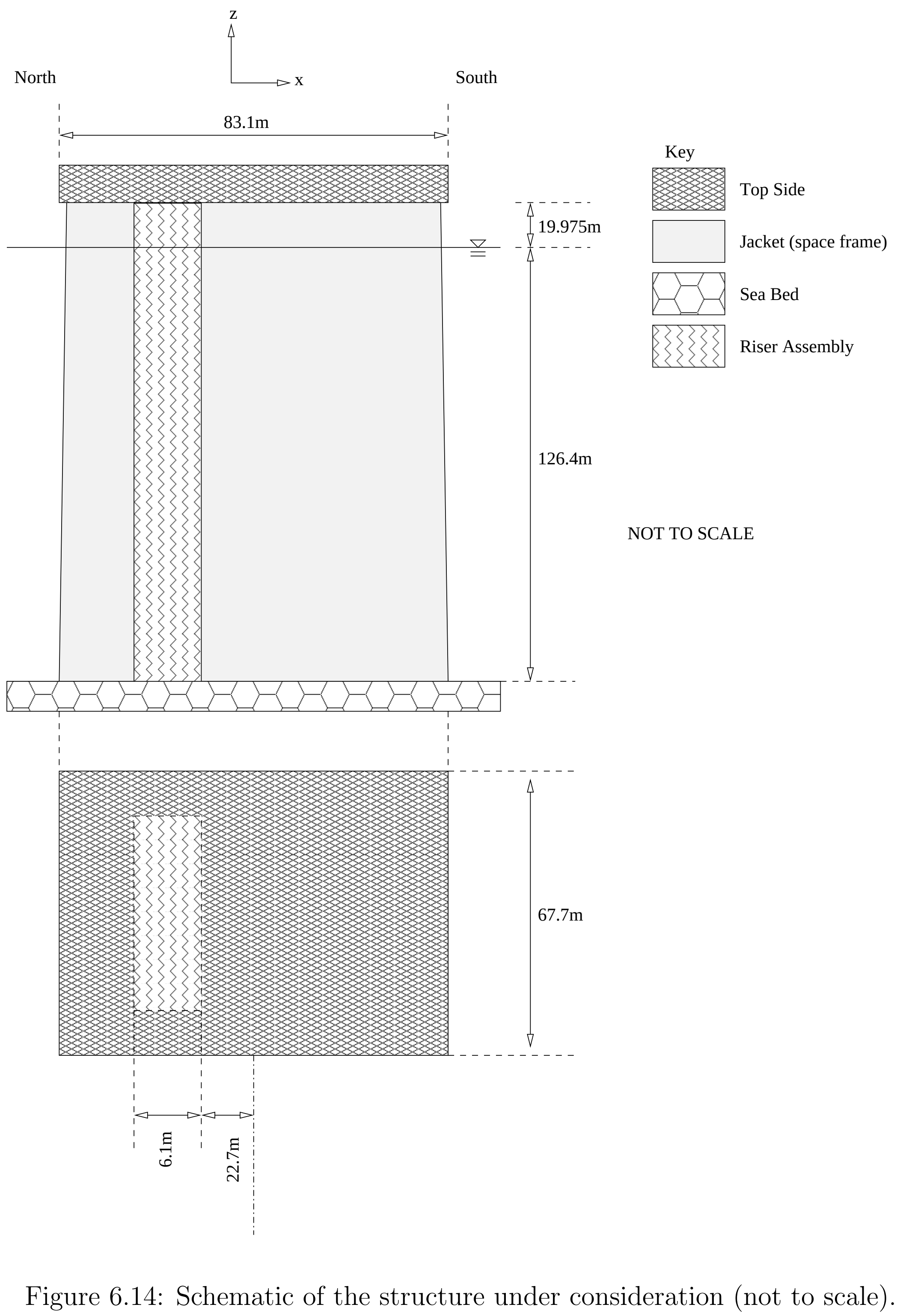


Figure 6.14: Schematic of the structure under consideration (not to scale).

the elevation of the highest wave crest, η_{max}, a vertical spacing of $\Delta z = 2m$ was adopted. This change in resolution was necessary due to the rapid variation of the fluid velocities in the higher part of the water column. The kinematics data was produced with a frequency of approximately 2Hz for a period of $40s$ either side of the focus event. In retrospect, it was decided that only data corresponding to the period of time between the crest prior to, and after, the focus event were required as it was unlikely that smaller waves would produce greater loads. Once the internal kinematics data were computed on the required grid, additional kinematics data from the free surface and the bed were added to form an overall data set. This full data set was then manipulated further in order to provide the data in a form that was compatible with the structural loading program (WAJAC).

The first step in computing the sub-structure load was to determine the location in both space and time of the maximum applied load. In the case of the Stokes wave solution this was a simple procedure as the wave is steady. In this case both time and space can be reduced to a simple phase angle. The loading on the sub-structure was calculated for every 1° of wave phase. For the Stokes wave with a 10^{-4} design wave period of $T = 15.87s$ in a water depth of $126.4m$ and with a wave height of $H = 35.24m$ (Table 6.1), the corresponding wave length was $\lambda = 408.3m$. As a result, calculations were undertaken at an effective spacing of $\Delta x = 1.13m$ or $\Delta t = 0.0441s$.

Finding the maximum loads using the nonlinear kinematics calculations from the BEM was considerably harder, not least because the nonlinear wave is unsteady and therefore varies both in time and space. To reduce the search space it was initially assumed that the kinematics under the highest crest elevation would produce the largest loads. The kinematics data relating to this particular event was first identified and the crest was positioned throughout the sub-structure in a similar manner to the Stokes wave. These calculations were first undertaken with a spacing of $\Delta x = 5m$. Once complete, and an approximate location of the maximum obtained, a second refined set of calculations were undertaken with $\Delta x = 1m$. The combination of these results confirm that for both the Stokes 5^{th}-order wave and the BEM nonlinear kinematics, both the base shear and the overturning moment have maximum values at $x_s = -20m$. This corresponds to the focussed wave event, or η_{max}, being located $20m$ North of the centroid of the structure. This result is exactly as expected given the location and layout of the riser assembly.

Having found the position at which the maximum load occurs in space, time histories of the nonlinear kinematics were applied at locations in the immediate vicinity of the $x_s = -20m$ maximum. The purpose of this latter exercise was to ensure that there was not some slightly lower crest elevation, positioned at a slightly different spatial location, that could produce a larger total sub-structural load, perhaps due to inertial loading effects on the riser assembly. In considering both the base shear and the overturning moment, the maximum load was indeed found to occur at $x_s = -20m$. Taken together, these results confirm that this positioning relates to the maximum load in both space and time.

Loading Component	Base Shear, F, (KN)	Overturning moment, M, (KNm)
Calculations based on Stokes kinematics:	103490	10485000
Calculations based on fully non-linear BEM kinematics:	84731	8999037
Ratio of predicted loads, BEM/Stokes:	0.819	0.858
Percentage reduction:	**18.1%**	**14.2%**

Table 6.2: Comparisons of the sub-structure loads arising in the 10^{-4} design wave conditions from calculations undertaken using the fully nonlinear BEM and the Stokes 5^{th}-order regular wave predictions.

Table 6.2 summarises the results arising from the calculations noted above. Comparisons between the loads based upon the Stokes 5^{th}-order wave model and the EPIC_BEM solution comfirm that the use of nonlinear kinematics dramatically reduces the sub-structural loads both in terms of the base shear and the overturning moment.

6.5.4 Loading trends

Given that the loads computed using nonlinear kinematics are considerably lower than those calculated using "equivalent" Stokes kinematics, some additional work was required to put this difference in context, especially with regards to the APInst guidelines.

The strength of the APInst guidelines arises from the fact that they build upon the best available field observations in which both the water surface elevations and the corresponding sub-structural loads were recorded. Indeed, a large number of the field observations used to construct the guidelines came from an extensive field monitoring program at Shell's Tern platform. As a result, it is difficult to criticise the procedures recommended in the APInst guidelines as they have provided a design "recipe" that has effectively been calibrated using field observations. However, such arguments only apply if the flow regime to which the structure is subjected lies within the bounds of the calibration.

In the case of the current calculations, the steepness of the 10^{-4} design wave event under consideration is far in excess of the observations used to calibrate the APInst guidelines. As a result, it is likely that the procedure specified in the guidelines is not wholly appropriate. Indeed, it would be very surprising if accurate estimates of the applied loads could be obtained using inaccurate (or unrealistic) descriptions of the water particle kinematics (Figures 6.11 – 6.13). It therefore follows that to achieve a more accurate representation of the sub-structural loads a set of fully nonlinear kinematics that better represent the reality of the situation should be adopted. However, in making such an assertion it is important to stress that if the steepness of the incident waves reduces, the predicted loads based upon the fully nonlinear kinematics must converge to the APInst guidelines. Afterall, these guidelines are based upon the best available field observations.

To address this point, a number of additional nonlinear kinematics calculations were undertaken. The additional calculations were again based on the spectral properties of the 10^{-4} sea state, as outlined in Table 6.1, but involve the simulation of individual or focussed wave events with reduced crest elevations. By keeping the spectral peak period constant and reducing the crest elevation, a series of waves with varying steepness were produced. The details of these additional wave cases are given in Table 6.3.

Case	**Linear amplitude sum:** $A = \Sigma a_n\ (m)$	**Crest elevation:** $\eta_{\mathbf{max}}(m)$	**Wave height:** $\mathbf{H_{dc}}(m)$	**Wave period:** $\mathbf{T_{dc}}(s)$
(a)	4.0	4.07	6.80	15.51
(b)	8.0	8.45	13.67	15.53
(c)	12.0	13.04	20.81	15.4
(d)	14.0	15.49	24.55	15.53
(e)	16.0	17.95	27.66	15.42
(f)	17.0	19.48	29.78	15.2
(g)	17.5	20.17	31.00	15.25
(h)	18.9	22.19	34.05	15.13

Table 6.3: Down-crossing properties of the additional wave events calculated using the fully nonlinear BEM solution. All waves were generated using the parameters outlined in Table 6.1.

Although eight wave cases were calculated, only the largest six cases $((c)-(h))$ were used in the following analysis. The reason for not using cases (a) and (b) is simply that with a reduction in wave height there is a corresponding reduction in wave induced fluid velocities. This, in turn, leads to a relative increase in the inertial loading. Since such results are not consistent with the 10^{-4} wave case, they were ignored. Having generated each of these wave events using the EPIC_BEM model the resulting water surface elevations were analysed using a down-crossing analysis to determine the wave height, H_{dc}, and the local wave period, T_{dc}, relating to the focal event, the subscript $_{dc}$ referring to a down crossing value. This method was adopted to be consistent with the analysis of field data used in the verification of the APInst guidelines Heideman (2010). Once the fully nonlinear kinematics and the "equivalent" Stokes 5^{th}-order wave solutions based upon the down-crossing analysis were obtained, the data was again passed to WS Atkins for use in the WAJAC structural loading model.

It is important to note that, when compared to the original 10^{-4} wave case, two modifications were made in relation to cases $(c) - (h)$.

- The numerical model was run using a slightly coarser grid with $\Delta x = \Delta y = 15m$ spacing. This change in resolution is also reflected in the internal kinematics sub-domain.
- It was assumed that the largest load was generated by the focused wave event (or η_{max}) and that the position at which this kinematics data was placed matched the position of the maximum load found from the "equivalent" Stokes wave.

The reason for both of these modifications was to reduce the computational workload. The discretisation used for the 10^{-4} wave event led to a run time of approximately ten days on 96 processors. Running the eight wave cases outlined above (Table 6.3) at the same resolution would take in excess of two months and this was considered too long to wait for speculative work. The second modification was a result of the substantial work undertaken in the 10^{-4} wave case to find that the position of both the nonlinear and Stokes 5^{th}-order wave was identical in terms of producing maximum loads. Based on the earlier work, and noting that the wave cases (a)-(g) are significantly less steep than the original 10^{-4} wave case, it was concluded that these minor modifications had no significant effect on the magnitude of the maximum loads.

6.5.5 Load comparisons

Table 6.4 defines the sub-structural loads arising from the kinematics calculations outlined in Table 6.3. The columns of most interest are three and four which define the ratio of the predicted forces (F_{BEM}/F_{Stokes}), and overturning moments (M_{BEM}/M_{Stokes}). As the ratio of the predicted forces is generally less than unity, the percentage reduction in the applied local load can be expressed as $(1 - ratio) \times 100\%$. These data are given in columns five and six and presented graphically in Figures 6.15 and 6.16, the former relating to base shear and the latter to overturning moment. In addition, these figures also contain a result from a related study undertaken on a structure in the Ekofisk field (Swan (2007), Johansen & Nestegård (2008)). The significance of this additional result is explained later. In considering the values presented in Table 6.4, and plotted in Figures 6.15

Case	**Wave steepness:** $H_{dc}k/2$	**Base shear:** $\frac{F_{BEM}}{F_{Stokes}}$	**Overturning moment:** $\frac{M_{BEM}}{M_{Stokes}}$	**Base Shear % reduction:** $1-\frac{F_{BEM}}{F_{Stokes}}\times 100$	**Overturning moment % reduction:** $1-\frac{M_{BEM}}{M_{Stokes}}\times 100$
(c)	0.181	0.975	1.016	2.5	-1.5
(d)	0.210	0.942	0.997	5.8	0.3
(e)	0.240	0.961	0.983	3.9	1.7
(f)	0.265	0.940	0.972	6.0	2.8
(g)	0.274	0.934	0.973	6.6	2.7
(h)	0.310	0.871	0.886	12.9	11.4

Table 6.4: Ratios of the predicted sub-structure loads, $\frac{F_{BEM}}{F_{Stokes}}$ and $\frac{M_{BEM}}{M_{Stokes}}$ for six wave cases with increasing wave steepness.

and 6.16, it is important to note that a number of hanging platforms (large box like elements) located high in the sub-structure have been removed from the calculations undertaken in the WAJAC software. The reason for removing these elements is that their presence prevented meaningful inter-comparisons between the BEM and Stokes 5^{th}-order loads. The larger crest-trough asymmetry present in the BEM waves meant that these waves would often just clip these platforms and cause unrealistic, and incomparable, increases in the applied loads. In the case of the 10^{-4} wave event described in §6.5.3 the platforms were present, hence the reduction in load in comparison to a Stokes "equivalent" wave was greater in the results presented previously (Table 6.2).

Prior to drawing conclusions from Figures 6.15 and 6.16, it is informative to

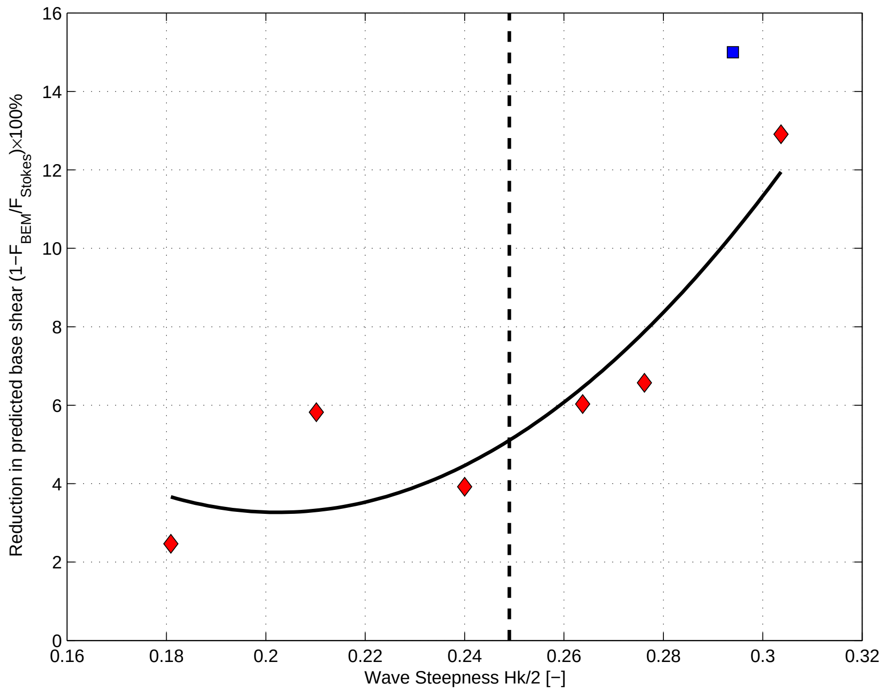


Figure 6.15: Reduction in predicted base shear as a function of wave steepness. Calculated data (♦), results from Ekofisk calculations (■), trendline (quadratic least squares) (——) and limit of wave steepness used for calibration of APInst guidelines (– – –).

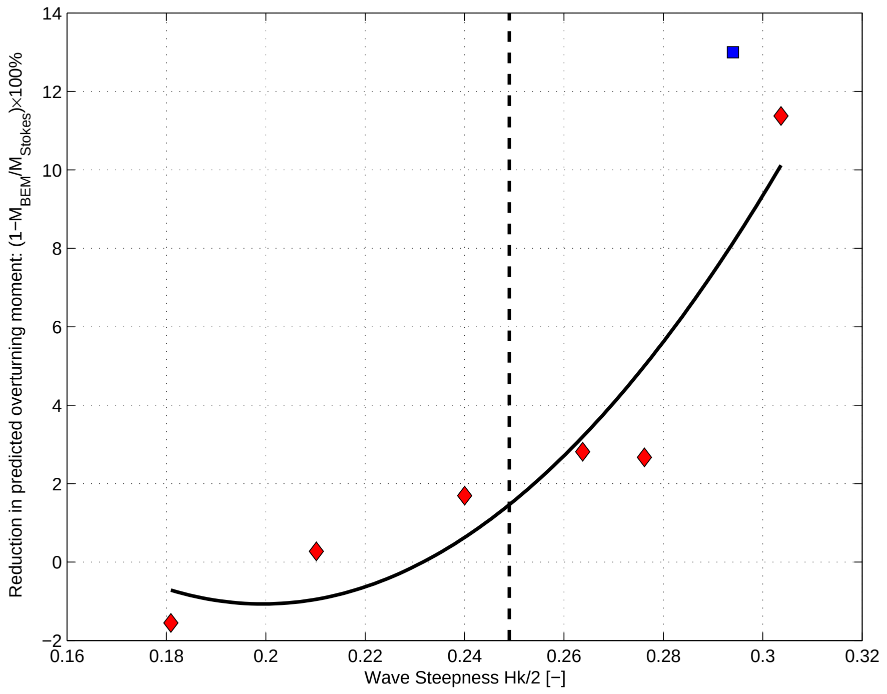


Figure 6.16: Reduction in predicted overturning moment as a function of wave steepness. Calculated data (♦), results from Ekofisk calculations (■), trendline (quadratic least squares) (——) and limit of wave steepness used for calibration of APInst guidelines (− − −).

consider some further information surrounding the calibration of the APInst guidelines and a similar calculation performed on a different platform. First, Heideman & Weaver (1992) note that the quality assurance procedure performed on many of the waves records taken at Shell's Tern platform meant that many of the largest waves recorded were discarded as spray adversely affected the recording equipment. As a result, the wave force data used in the formulation of the APInst guidelines related to wave heights in the range $8.6m \leq H_{dc} \leq 21.5m$. Unfortunately, Heideman & Weaver (1992) give little information about wave periods found in the Tern study. However, this information can be gleaned from a summary provided by Atkins *et al.* (1993). With regards to a number of key storms occurring in 1992, Atkins *et al.* (1993) give $T_p \approx 15s$. Adopting the belief that the local down-crossing period is approximately $0.9T_p$, leads to $T_{dc} = 13.5s$. Evidence to support this value is provided by Atkins *et al.* (1993) who present a record of the largest wave measured at Tern with $T_{dc} = 13.4s$. This specific wave however had a wave height of $H_{dc} = 25.1m$ and can therefore be assumed to have been removed by the quality assurance procedures outlined by Heideman & Weaver (1992). Further evidence of the wave periods recorded at Tern are given by Jonathan & Taylor (1995). After removal of any (obviously) spurious wave components, they provide data describing the ten largest waves recorded at Tern in 1993. These wave records include wave heights lying within the range $21 - 22m$, with local down-crossing periods of $T_{dc} = 13.0 - 14.5s$. A brief summary of the wave conditions recorded at Tern is given in Table 6.5.

Second, a study performed by Swan (2007) and Johansen & Nestegård (2008) relating to the Ekofisk field specified a one in ten thousand year wave event to have $H_{max} = 32.96m$ and a corresponding $T_p = 15.9s$ (again defined by $0.9T_p$), the resulting wave steepness being $H_{dc}k/2 = 0.294$. The importance of this study will be made clear shortly.

Returning to the results presented in Figures 6.15 and 6.16 the data presented in red defines the percentage load reduction as a function of wave steepness for the wave cases indicated in Table 6.4 along with the results from the 10^{-4} design wave event. The solid black line is used to indicate the trend present in the data and the dashed black line indicates the limit of the wave steepness encountered

Source	**Wave height:** $H_{dc}\vert_{max}$ (m)	**Wave period:** T_{dc} (s)	**Wave steepness:** $H_{dc}k/2$
Heideman & Weaver (1992)	21.5	13.5*	0.238
Atkins *et al.* (1993)	25.1	13.4	0.282
Jonathan & Taylor (1995)	19.7	13.0	0.235

Table 6.5: Summary of wave observations recorded at the Tern platform in the Northern North Sea. (*Note: this value of T_{dc} was based upon $0.9T_p$, where $T_p = 15s$ was adopted as the best available estimate.)

in the recording program at Tern. The conclusions that can be drawn from these figures are as follows.

a) At the lower values of wave steepness ($H_{dc}k/2 = 0.2$) the agreement between the loads calculated using the fully nonlinear BEM model and the “equivalent” Stokes 5^{th}-order solution is good. In terms of the base shear, the fully nonlinear calculations suggest that the Stokes 5^{th}-order solution may over predict the loads by 2.5%, while in the case of the overturning moment it under predicts by 1.5%. In both cases it is clear that the results arising from the use of the fully nonlinear kinematics are very similar to those arising from a Stokes 5^{th}-order solution, within this range of steepness.

b) When a fully nonlinear kinematics field is employed, an increase in the wave steepness leads to a reduction in the applied loading when compared to that predicted by an “equivalent” Stokes 5^{th}-order wave model.

c) At the limit of wave steepness used in the validation and calibration of the APInst guidelines (denoted by the vertical dashed line) the nonlinear calculations suggest that the base shear is reduced by approximately 5% and the overturning moment by 2%.

d) Although a 5% reduction in load, found at the limit of the wave steepness observed at the Tern platform, may not be practically significant, in the context of the present study, the predicted trends are entirely consistent with analysis of field observations. Indeed, this is entirely congruent with expectations. The largest loads recorded at the Tern platform are slightly lower that those predicted by the APInst guidelines. This confirming that the present approach is fully consistent with the best available field observations.

e) At $H_{dc}k/2 = 0.294$ a blue marker can be seen. This point relates to calculations undertaken for ConocoPhillips and BP for a similar jacket structure in the Ekofisk field (Johansen & Nestegård, 2008). This point has not been used in the calculation of trend lines and therefore provides an independent result that is consistent with the trends seen in the present study. The small departure in the load reduction relating to this data point is undoubtedly due to changes in the effective water depth: at the Ekofisk field $k_p d = 1.18$, while at the present case $k_p d = 1.91$ (where k_p is the wave number associated with the wave component at the peak of the frequency spectrum). The agreement of the present data with the Ekofisk study is even more important because the Ekofisk data was produced by the entirely separate (non-breaking) numerical model of Bateman *et al.* (2003). In effect, this result provides a completely independent verification of the present calculations.

6.6 Conclusions

This chapter began with a formal validation of the EPIC_BEM model, the comparisons provided confirming that the model performs correctly when simulating a number of well established wave conditions. Specifically, numerical predictions of the free surface, in both time and space, for both regular and irregular waves showed excellent agreement with available analytical theories. Furthermore, the shape of the underlying spectrum, and the phasing of the wave components, in the focused irregular wave case closely matched their respective analytical counterparts. Building on the success of the free surface descriptions, predictions of the water particle kinematics in both regular and irregular wave cases were shown to be equally successful when compared to analytical results. It can therefore be concluded that the EPIC_BEM model produces accurate numerical predictions of a number of established wave conditions.

On the basis of these results the model was applied to more testing wave conditions. Specifically the EPIC_BEM model was applied to the description of a design wave event with an annual probability of exceedance of 10^{-4}. Comparisons between the fully nonlinear kinematic predictions and the results of an "equivalent" Stokes 5^{th}-order wave calculations, the latter commonly used in design calculations, show marked differences both in terms of the water surface elevation and the underlying kinematics. To emphasise the importance of these differences the kinematics data were input into an established fluid loading model and the loads on an offshort jacket structure calculated. When compared to established design procedures (based on a Stokes 5^{th}-order wave solution) the reduction in the global loading when using the fully nonlinear kinematics predictions was found to be significant. Indeed, an 18.1% reduction in the total base shear and a 14.2% reduction in the total overturning moment were identified.

Given the magnitude of the presented load reductions, it was clearly necessary to explain them in the context of the existing APInst guidelines. To this effect, additional calculations were undertaken involving waves with the same spectral properties but reduced wave height and therefore reduced steepness. Incorporating these results with those from the 10^{-4} design wave it was possible to establish trend lines indicating the reduction in the applied sub-structural loads in relation to predictions based on "equivalent" Stokes 5^{th}-order kinematics. It was found that the steeper the incident waves, the greater the reduction in both the base shear and the overturning moment. These trend lines provide a clear explanation for the reduction in load prediction with wave steepness and, most importantly, allow the present results to be reconciled with established design guidelines; the latter having been calibrated using the best available field data.

The next chapter investigates another type of extreme wave, a deep water breaking wave. Recent research undertaken in the Crest JIP (Joint Industry Project) has confirmed that such waves should be taken into account within the design process. The EPIC_BEM solution provides a realistic tool with which to do exactly this.

7

Overturning Wave Groups

7.1 Introduction

This chapter begins by reviewing progress to date in the numerical modelling of overturning ocean waves in realistic sea states. Following this review, the calculation method for kinematics information on the domain boundary is revisited. It appears that the accurate provision of this information is key to simulating, and therefore understanding the nature of, breaking waves. This leads to some further discussion of the EPIC_BEM model relating specifically to the investigation of breaking waves. Particular attention is paid to the role of the so-called switching time, when the free surface boundary conditions are changed from one-third-Lagrangian form to a fully-Lagrangian form; the latter necessary to model overturning waves. The final sections of the chapter provide some description of wave breaking in a realistic ocean spectrum. Insights into the water particle kinematics associated with these waves are provided and comparisons to analytical and fully nonlinear non-breaking waves given.

7.2 Models of wave breaking

In the investigation of three dimensional wave breaking, the boundary element method has been the preferred numerical scheme for quite some time. Due to the non-dissipative nature of its formulation, the large physical space and evolution times required to simulate the growth of extreme waves in realistic sea states can be modelled without the loss of energy that typically occurs in other numerical formulations. In this regard, an important contrast can be drawn between models based on a BEM formulation and those based on the direct solution of the Navier Stokes equations (Yan & Ma, 2010).

To date, the subject of wave breaking has been investigated by several research groups. Historically, publications on the topic have considered two dimensional wave breaking; the wave breaking event triggered by a variety of scenarios. Within this early literature there appear to be two trends. The first, that two dimensional wave modelling was universally chosen due to lack of computing power and modelling techniques. The second, that most studies of wave breaking concerned the simulation of regular steady waves or solitary waves. In addition, the earliest models tended to use periodic physical boundaries, while later models adopt NWTs, often with bathymetries to assist the formation of breaking waves.

The first ever breaking wave simulation was undertaken by Longuet-Higgins & Cokelet (1976) in a two dimensional, periodic, conformally mapped domain. This initial simulation was the stimulus for a large volume of research that worked towards removing the original constraints of periodicity, an assumed infinite water depth and, the implementation within conformally mapped space. In striving towards these goals, notable contributions have been made by Ortiz & Douglass (1993), Grilli *et al.* (1997), and Drimer & Agnon (2006). With regards to two dimensional wave breaking arising in realistic frequency spectra, Christou *et al.* (2007) provide an excellent example of the use of a JONSWAP spectrum with realistic parameters to generate a two dimensional overturning wave and its associated water particle kinematics.

Although the simulation of two dimensional wave breaking is of interest, the use of these models for practical applications is limited for obvious reasons. However, Guyenne & Grilli (2006) noted that there are some similarities between the breaking of two and three dimensional solitary waves. Indeed, on the basis of these results it appears that a two dimensional model may be of use for an initial investigation of a breaking wave field as the computational effort required is considerably less.

In the field of the numerical modelling of three dimensional wave breaking, a number of attempts have been made. Again, these tend to be based upon solitary waves, regular waves or the directional focussing of a small number of identical regular or steady wave components. The groups of Grilli, Dias, Guyenne, Fochesato, along with their associated researchers, have provided a number of important insights into the field of three dimensional wave breaking. Grilli *et al.* (2001) first demonstrates solitary wave breaking on a uniform (plane) slope, giving a two dimensional wave profile, and contrasts these results with solitary wave breaking on a ridged slope. This work was improved by Fochesato *et al.* (2005), in terms of the accuracy of the boundary conditions applied. This led to a similar (related) publication by Guyenne & Grilli (2006). More recently, the computational efficiency of their code was then improved by the implementation of the fast multipole algorithm, details provided by in Fochesato & Dias (2006).

In terms of simulating wave breaking in realistic, open ocean, conditions , Grilli and his co-workers have made two attempts with varying success. In both cases the waves were formed by the directional focussing of wave energy. This approach is referred to in their paper as "directional energy focussing" and entirely neglects the underlying frequency spectrum. For example, Brandini & Grilli (2001) attempted to create a realistic breaking wave using eight regular waves, of identical frequency and amplitude, with directions calculated such that they converge in phase to form a steeper wave with a surface profile that varies in the two horizontal directions. In the context of laboratory wave generation, these wave events are commonly referred to as "bulleyes" due to the nature of their surface elevation contour plots. Unfortunately, this scenario failed to produce a breaking wave despite the steepness of the wave conditions present in the NWT. The authors suggest that the failure was due to waves starting to spill near the input boundary. This caused the rapid convergence of nodes which, in turn, caused "quasi-singularities" due to node proximity. In terms of the mathematics described earlier in this thesis, their computation broke down because $r \to 0$ as the nodes became close and therefore "quasi-singular"; details of the effect being given in §2.4.6.

In a later paper (Fochesato *et al.*, 2007) a second attempt was made at "directional energy focussing" to create a realistic breaking wave. Fochesato *et al.* (2007) first argued that this was a suitable mechanism for the creation of realistic breaking waves by claiming that *"the large focussed waves show very similar features near their crest and, hence, somewhat locally lose the memory of the physical phenomenon that has caused energy focussing"*. Such arguments seek to ignore the fact that the real sea states are broad-banded in both frequency and direction with frequency dispersion or frequency focussing playing a significant role in the evolution of the largest waves. Indeed this process lies at the heart of the commonly adopted NewWave model (Tromans *et al.*, 1991) which describes the most probable shape of a large linear wave and which has been extensively validated on the basis of field observations (Jonathan & Taylor, 1997). However, setting aside arguments concerning the validity and practical relevance of the method of creating these waves, Fochesato *et al.* (2007) used thirty identical regular wave components in an identical manner to Brandini & Grilli (2001) and achieved a breaking wave event. In addition, some insight into the kinematic field lying beneath breaking waves of this nature was provided.

Outside the work undertaken by Grilli and his co-workers, it appears that only one other group have produced 3D breaking waves using the BEM. Xue *et al.* (2001) produced 3D wave breaking using a 5^{th}-order Stokes regular wave solution as the input, and then applied a 3D pressure distribution across the free surface to modulate the initially long crested waves. Their model appears successful in the sense that it produced breaking waves and allowed considerable insight into the associated kinematics and acceleration fields. Unfortunately the stability of the computations was poor and so a smoothing filter was applied to the free surface at regular intervals. The loss of information and accuracy of the results arising from such a method is of concern. Furthermore, as was the case in the earlier 2D attempts at modelling wave breaking, the practical relevance of using Stokes waves as the initial or starting wave form needs to be carefully considered.

In the interests of completeness, some success in the modelling of 3D wave breaking has also been reported using other numerical methods. In particular, Yan & Ma (2010) use a finite element based method to simulate a NWT in a similar fashion to that presented in the current work. A number of good results, in comparison to other numerical models, are obtained for both 2D and 3D wave breaking. The work is limited to solitary and regular waves, with the onset of wave breaking instigated by the use of artificial reefs or slopes. As a result of both the method used to produce wave breaking and the input wave characteristics, the resulting wave forms are not comparable or consistent with the occurrence of wave breaking in the open ocean.

To summarise this sub-section, as far as the author is aware there are no examples of numerically modelled three dimensional breaking waves arising from the focussing of wave components in a realistic ocean spectrum; the latter being broad banded in both frequency and direction. It is precisely this key step to a better understanding of real wave breaking that is presented in the following work.

7.3 A re-assessment of boundary kinematics

It is imperative that the water particle kinematics (u, v, w) computed on the domain boundaries, particularly $\Gamma_{surface}$, are accurate as it is these values that dominate the boundary conditions which drive the model. In §2.4.8 is was shown that velocities and spatial gradients associated with nodal positions around the domain can be easily computed by applying polynomial fits to potential and potential fluxes at known locations in a local grid, or "sliding element", around the node of interest. For most circumstances this method is very fast and sufficiently accurate. However, in modelling the extreme behaviour associated with breaking waves, a number of potentially important problems arise.

The first problem associated with the use of polynomials occurs when a derivative is required at the interface between two boundaries, the boundary interface. In this case, the end point of a polynomial fit is used. This is a well known recipe for inaccuracy (Press *et al.*, 1990). Fortunately, at the edges of the NWT where this situation arises, additional physical information associated with the fluid flow can be used to supplement the definition of the polynomial derivatives. For example, at a reflecting wall, it is known that the spatial and potential gradients perpendicular to the wall are zero. Hence, the exact value determined by the physics of the problem can be used in the boundary conditions in place of the interpolated value. This principle can be extended to all boundary interfaces within the NWT.

When considering the occurrence of wave breaking, two further problems arise concerning the calculation of kinematics information. First, to allow a wave to break, the model must run applying fully-Lagrangian boundary conditions to nodes on the water surface. As a result of these boundary conditions, the surface nodes can drift with the fluid. This permits the free surface to become multi-valued such that a wave can break. The physical processes involved in wave breaking often cause the nodes that form the water surface to drift in all Cartesian directions. As a result the elements forming the domain become distorted. Furthermore, as a consequence of the elements on the boundaries becoming distorted, the sliding elements (§2.4.8) associated with the computation of the velocities and gradients will also become distorted.

In §2.4.8 it was assumed that the $\{\mathbf{m}, \mathbf{s}, \mathbf{n}\}$ vectors, used to compute the spatial gradients, were orthogonal. This holds true on the water surface ($\Gamma_{surface}$) in the semi-Lagrangian frame of reference. However, in a fully-Lagrangian frame of reference once the nodes start to drift, the orthogonality of the unit vectors $\{\mathbf{m}, \mathbf{s}\}$, used in the computation of velocities and gradients, ceases to hold. To the author's knowledge this problem has only been discussed once in the published literature, by Fochesato *et al.* (2005). In considering this effect they acknowledge the problem in relation to their earlier publication (Grilli *et al.*, 2001) and make a correction to their coordinate system to account for the change. To emphasis the importance of this effect Fochesato *et al.* (2005) show some examples of solitary waves breaking over a reef and compare results arising from their original formulation and their new (corrected) formulation. A marked difference was seen in the wave evolution following the application of the corrected boundary conditions, thereby highlighting the importance of this effect. In light of this information, if the EPIC_BEM model is run using fully-Lagrangian boundary conditions, the kinematics information on the boundaries is computed with the necessary corrections to deal with the lack of orthogonality in the reference coordinate system as suggested by Fochesato *et al.* (2005).

The second problem associated with wave breaking occurs when a wave begins to overturn. A point often forms at the tip of the overturning wave crest and this is generally modelled by a single node. Clearly, fitting a polynomial around such large a geometric irregularity can cause considerable problems with accuracy, especially with regard to the continuity of velocities around the plunging part of the wave. To the author's knowledge there are no solutions to this problem within the present literature. However, to overcome this difficulty, some hints at a potential solution are presented in §8.3.1.

7.4 Computational domain and model set-up

The set-up used in the EPIC_BEM model for the simulation of breaking waves is similar to that described in Chapter 6. Within this discussion two NWTs were used in the simulation of the desired wave events: NWT-HR and NWT-LR, HR and LR refer to high resolution and low resolution respectively. Both the NWTs simulate an area that is approximately the same size as the wave basin in the fluids research laboratory in the Dept. of Civil and Environmental Enginering at Imperial College London. The exact dimensions of the domains and the discretisations employed are given in Table 7.1. In both cases it should be noted that, to take advantage of the symmetry of the wave field (about $y = 0m$), half size domains are used, as described in §6.3.1.

	NWT-HR	**NWT-LR**
X dimension (m)	$[-410 : 5 : 300]$	$[-410 : 10.25 : 307.5]$
Y dimension (m)	$[-410 : 5 : 0]$	$[-410 : 10.25 : 0]$
Z dimension (m)	$[-110 : 5 : 0]$	$[-110 : 11 : 0]$

Table 7.1: Parameters relating to the NWTs used in the simulation of a breaking wave event. Notation used, [*start coordinate:discretisation step size:end coordinate*]

The only significant difference between the two NWTs is the discretisation used in forming the boundary elements; NWT-HR being formed at considerably higher resolution than NWT-LR. The reason for choosing two NWTs of different resolution lies in the difficulty of finding a set of initial conditions that produce a breaking wave. Essentially, a parametric search was required to find a set of initial conditions that create a wave that breaks by plunging or overturning. In order to proceed through the search space as efficiently (or quickly) as possible, a low resolution domain is used so that the computation time is reduced. Once a likely set of conditions are found they are then simulated in the higher resolution NWT. Following a considerable number of trials, the conditions given in Table 7.2 were found to generate a realistic overturning wave in deep water.

Property	**Value**	**Comment**
Spectral form	JONSWAP *NewWave*	*NewWave* formulated following Tromans *et al.* (1991)
Peak period (T_p)	$10.0s$	-
Spectral peak enhancement factor (γ)	1.0	-
Directional spread (s)	7	Mitsuyasu spread (Mitsuyasu, 1975)
Water depth (d)	$110m$	-
Input amplitude (A)	$19.0m$	Based on equation (6.6) as outlined in §6.3.1
Courant number (C_0)	0.4	-

Table 7.2: Metocean parameters relating to the modelling of a breaking wave event.

As described in §2.5 the EPIC_BEM model can be switched to use a fully-Lagrangian frame of reference for computing both the boundary conditions and the movement of the nodes which form the boundary or domain. Switching to a fully-Lagrangian representation of the boundary conditions is necessary to allow the formation of a multi-valued surface required for the occurrence of wave breaking. At present there is no method employed for the automatic switching of boundary conditions and so numerous switching times are tested. Each attempted switching time results in slightly different movement of the nodes on the free surface. As a consequence, some switching times cause nodes to "crash" prematurely into each other, whilst others cause large elongations of the surface elements and the rapid escalation of numerical errors. In both cases the numerical scheme comes to an abrupt halt. In contrast, other switching times give rise to the desired overturning motion. At first sight this appears to be a nondeterministic approach. However, the movement of the free surface up to the point at which the model breaks down is consistent regardless of switching time employed. It just so happens that in some cases the model breaks down more quickly than in others. Proof of the deterministic nature of this approach is given in §7.5.

Evidently, trying a large number of switching times with the model running from its start position is a waste of computing time. Therefore, to reduce the computational effort associated with exploring different switching times, the EPIC_BEM code writes to disk all the information required to restart the code using Adams-Bashforth-Moulton time stepping (see §2.4.10). This effectively creates a continuous set of check points for the simulation that allow it to be restarted from a previous state. This is a particularly useful feature of the model as it means that once the wave field has been simulated to a position near breaking, the model can be restarted at multiple instances, to trial multiple switching times. This avoids the unnecessary recomputing of known information.

With the information regarding switching times and restarting the model from arbitrary check points in mind, the following parameters defining activities within the time domain were found to produce a breaking wave. First, the model was run from $-230s$ to approximately $-43s$ in a semi-Lagrangian frame of reference. At the outset of these calculations, a $30s$ window was used to "ramp up" the potential flux along the input boundary (thus avoiding the instigation of numerical shock waves). The model was then restarted from a check point near $-43s$ and a number of switching times were used. Initial tests undertaken within the NWT-LR domain suggested that switching times in the range $-21s \leq t_s \leq -16s$ were likely candidates for producing overturning waves. Accordingly, switching times within this range dictated the search for the development of breaking waves in the higher resolution (NWT-HR) domain. A comprehensive set of results and observations relating to the occurrence wave breaking within the NWT-HR domain is provided in §7.5. Prior to presenting the detailed numerical results, a brief discussion of a new method used to compute the second-order wave model of Sharma & Dean (1981) is provided. This method, developed by the author, is used for comparisons with the breaking wave computations presented in §7.5. However, given the practical importance of second-order calculations in design applications, the proposed model has much wider applications than those simply applied herein.

7.4.1 Second-order random wave theory; an efficient calculation procedure

In the prediction of ocean wave profiles and their underlying water particle kinematics, linear random wave theory (equations (6.6) and (6.7)) is often used as a first approximation for incorporating the directional characteristics of the frequency spectrum describing the sea state. Unfortunately, weakly nonlinear waves cannot be predicted accurately by such theories, but are not sufficiently steep to warrant the use of a fully nonlinear wave model. In such cases the second-order wave theory proposed by Sharma & Dean (1981) is often considered appropriate. Indeed, this solution is often used in design calculations (or, at least, it should be), because it allows the water surface elevations and the water particle kinematics to be defined in a way which is consistent with commonly predicted crest statistics (Forristall, 2000). However, one drawback of the method proposed by Sharma & Dean (1981) is that the algorithm is $O(n^2)$, where n is the number of components in the wave field distributed over both frequency and direction. As a result, the method can be computationally intensive and, consequently, is not used as often as it ought to be. This section briefly discusses a modification to the algorithm that reduces the computational workload by at least an order of magnitude, for the majority of calculations, without any significant loss of accuracy.

The second-order model of Sharma & Dean (1981) is based upon the summation of the two-wave interactions first identified by Longuet-Higgins & Stewart

(1960) and Longuet-Higgins (1963); the former dealing with uni-directional waves and the latter incorporating the effects of directional spreading. In formulating these interactions, it is shown that if two linear wave components, both freely propagating and hence satisfying the dispersion equation, co-exist they will interact to produce both frequency-sum and frequency-difference terms; the former being at high frequency and the latter at low frequency. In addition, both individual linear wave components will have second-order Stokes', or self-interaction, terms associated with them. All of these second-order terms (the sum-terms, the difference-terms and the Stokes' terms) correspond to bound waves and, as such, do not satisfy the dispersion equation. In the case of the two-wave interactions, the phase velocity depends upon the two interacting free waves and the nature of the term involved (sum or difference); while the Stokes' terms simply travel at the speed of their associated free wave.

In applying these solutions to a random or irregular wave field, involving a large number of frequency components, Sharma & Dean (1981) simply summed up the sum and difference terms arising from every possible pair of wave interactions and added the self interaction or Stokes terms. This gives rise to n^2 second-order components, where n is again the total number of wave components distributed over both frequency and direction.

In undertaking a random wave simulation, the repeat period of the wave record is inversely proportional to the spacing between adjacent frequency components (Δf). If, as is usually the case, a long repeat period is required, this implies a small (Δf) and hence large n for realistic ocean spectra. However, if n becomes too large, the n^2 computations may require prohibitively long computational runs. Problems of this type become particularly apparent when calculations need to be undertaken at a large number of spatial locations, as would be the case in, for example, a pipeline design. The present work avoids this difficulty by the introduction of a technique referred to as *spectral priming.*

Background

Sharma & Dean (1981) state that in a random sea the velocity potential, ϕ, and the sea surface elevation, η, can be represented as follows:

$$\phi(x,y,z,t) = \phi^{(1)}(x,y,z,t) + \phi^{(2)}(x,y,z,t) + \ldots, \tag{7.1}$$

$$\eta(x,y,t) = \eta^{(1)}(x,y,t) + \eta^{(2)}(x,y,t) + \ldots, \tag{7.2}$$

where the superscript defines the order of the terms involved, $^{(1)}$ indicating terms of $O(ak)$ and $^{(2)}$ terms of $O(a^2k^2)$. At a first order of wave steepness the solution is as outlined in §6.3.1 and can be rewritten as

$$\phi^{(1)} = \sum_{i=1}^{n} b_i \frac{\cosh k_i(d+z)}{\cosh k_i d} \cdot \sin(\mathbf{k}_i \cdot \mathbf{x} - \omega_i t + \psi_i) \tag{7.3}$$

and

$$\eta^{(1)} = \frac{1}{g}\sum_{i=1}^{n} b_i\omega_i \cdot \cos(\mathbf{k}_i \cdot \mathbf{x} - \omega_i t + \psi_i) = \sum_{i=1}^{\infty} a_i \cos(\Psi_i) \tag{7.4}$$

where

$$a_i = \frac{b_i\omega_i}{g}, \tag{7.5}$$

$$\omega_i^2 = gk_i \tanh(k_i d), \tag{7.6}$$

$$\Psi_i = \mathbf{k}_i \cdot \mathbf{x} - \omega_i t + \psi_i, \tag{7.7}$$

and

$$\mathbf{k}_i = (k_x, k_y) = (\mathbf{k}_i \cos\theta_i, \mathbf{k}_i \sin\theta_i). \tag{7.8}$$

Within this solution, $\mathbf{x}$ is the position in the horizontal plane, (x,y). The expression $\mathbf{k}_i$ is the wave number vector associated with wave component i; k_x denoting the wave number in the x direction, k_y the wave number in the y direction and $k_i = \sqrt{{k_x}^2 + {k_y}^2}$. The variables a_i, ω_i, θ_i and ψ_i are respectively the wave amplitude, angular frequency, direction of travel and an arbitrary phase angle associated with wave component i. The variable n indicates the number of wave components under consideration. The symbol g defines the acceleration due to gravity, t expresses time and d represents the local water depth. The variable, Ψ, is used as an argument expressing the overall phasing, taking into account the spatial location $(\mathbf{x})$, the time (t) and the arbitrary phase angle (ψ).

At a second-order of wave steepness,

$$\begin{aligned}\phi^{(2)} = {} & \tfrac{1}{4}\sum_{i=1}^{\infty}\sum_{j=1}^{\infty} b_i b_j \frac{\cosh\left(k_{ij}^{-}(d+z)\right)}{\cosh(k_{ij}^{-}d)} \cdot \frac{D_{ij}^{-}}{(\omega_i - \omega_j)} \sin(\Psi_i + \Psi_j) + \\ & \tfrac{1}{4}\sum_{i=1}^{\infty}\sum_{j=1}^{\infty} b_i b_j \frac{\cosh(k_{ij}^{+}(d+z))}{\cosh(k_{ij}^{+}d)} \cdot \frac{D_{ij}^{+}}{(\omega_i + \omega_j)} \sin(\Psi_i + \Psi_j)\end{aligned} \tag{7.9}$$

and

$$\begin{aligned}\eta^{(2)} = {} & \frac{1}{4}\sum_{i=1}^{\infty}\sum_{j=1}^{\infty} a_i a_j \\ & \left\{ \left[\frac{D_{ij}^{-} - (\mathbf{k}_i \cdot \mathbf{k}_j + R_i R_j)}{\sqrt{R_i R_j}} + (R_i + R_j) \right] \cdot \cos(\Psi_i - \Psi_j) + \right. \\ & \left. \left[\frac{D_{ij}^{+} - (\mathbf{k}_i \cdot \mathbf{k}_j - R_i R_j)}{\sqrt{R_i R_j}} + (R_i + R_j) \right] \cdot \cos(\Psi_i + \Psi_j) \right\},\end{aligned} \tag{7.10}$$

$$k_{ij}^{-} = |\mathbf{k}_i - \mathbf{k}_j|, \tag{7.11}$$

$$k_{ij}^{+} = |\mathbf{k}_i + \mathbf{k}_j|, \tag{7.12}$$

and

$$\begin{aligned}D_{ij}^{-} = {} & \Big(\Big\{ (\sqrt{R_i} - \sqrt{R_j}) \left[\sqrt{R_j}(k_i^2 - R_i^2) - \sqrt{R_i}(k_j^2 - R_j^2) \right] \Big\} \\ & \div \left[(\sqrt{R_i} - \sqrt{R_j})^2 - k_{ij}^{-} \tanh(k_{ij}^{-}d) \right] \Big) \\ + {} & \Big(\left[2(\sqrt{R_i} - \sqrt{R_j})^2 (\mathbf{k}_i \cdot \mathbf{k}_j + R_i R_j) \right] \\ & \div \left[(\sqrt{R_i} - \sqrt{R_j})^2 - k_{ij}^{-} \tanh(k_{ij}^{-}d) \right] \Big),\end{aligned} \tag{7.13}$$

$$\begin{aligned}D_{ij}^{+} = {} & \Big(\Big\{ \left(\sqrt{R_i} + \sqrt{R_j} \right) \left[\sqrt{R_i}(k_j^2 - R_j^2) + \sqrt{R_j}(k_i^2 - R_i^2) \right] \Big\} \\ & \div \left[(\sqrt{R_i} + \sqrt{R_j})^2 - k_{ij}^{+} \tanh(k_{ij}^{+}d) \right] \Big) \\ + {} & \Big(\left[2(\sqrt{R_i} + \sqrt{R_j})^2 (\mathbf{k}_i \cdot \mathbf{k}_j - R_i R_j) \right] \\ & \div \left[(\sqrt{R_i} - \sqrt{R_j})^2 - k_{ij}^{+} \tanh(k_{ij}^{+}d) \right] \Big),\end{aligned} \tag{7.14}$$

and

$$R_i = k_i \tanh(k_i d). \tag{7.15}$$

With the aim of reducing the computational effort required to calculate the second-order contributions, several points need to be considered. First, the interaction of component i with j is identical to the interaction of component j with i. This means that if the second-order contributions between the $\mathbf{Z}_{ij}$ and $\mathbf{Z}_{ji}$ components were placed in the i^{th} and j^{th} positions of an n by n matrix, $\mathbf{Z}$, the matrix would be symmetrical. This important fact reduces the number of interactions to be calculated from n^2 to $n(n-1)/2$. Additionally, the self interaction, or Stokes', terms lie on the diagonal of the matrix. These terms exist only once and so can be computed at the same time as the first order components.

Although the introduction of this symmetry is important in the numerical evaluation of a second-order model, the method remains a fundamentally $O(n^2)$ process. As a result, for any practically relevant problem, involving a large number of wave components, the solution will be computationally intensive. In terms of reducing this intensity, whilst maintaining a high degree of accuracy, a new technique is introduced as outlined in the following section.

Spectral priming

In applying a second-order model to the description of the design wave conditions, outlined in Chapter 6, it became clear that many of the two-wave couplings involving pairs of freely propagating wave components within a sea state yield negligible contributions to the second-order description of both the water surface elevation and the underlying water particle velocities. As a result, as far as the calculation of the second-order terms is concerned, the underlying spectrum defining the sea state can be manipulated, to include only those waves that are involved in significant two-wave interactions. This approach is subsequently referred to as *spectral priming*; its objective simply being to reduce the number of free wave components involved in the calculation of the second-order terms, thereby significantly reducing the overall run time.

Spectral priming is performed by first creating a focussed wave group by giving all components of the sea state a zero phase angle ($\psi_i = 0$). The second-order description of this wave is then computed using the self interaction terms and all possible couplings of two freely propagating wave components, as stated in

Sharma & Dean (1981). At this stage no manipulation of the underlying wave spectrum has occurred and the calculations are only undertaken for one instant in time and at one spatial location, corresponding to the occurrence of the largest (focussed) wave crest. This point was chosen because, with the phasing of all the wave components set to zero, the magnitude of all the second-order contributions is maximised. The second-order calculation is then performed a number of times; the starting point for these calculations being the inclusion of only those components that when coupled together give the largest contributions to the second-order part of the calculation. The numerical code then loops using increasingly more components until a predetermined percentage (usually 90%) of the magnitude of the complete second-order part of the calculation is reached. If the second-order contribution accounts for 10% of the total solution (linear plus second-order), the inclusion of 90% of the second-order total gives 99% of the total result. The pairs of wave components necessary to achieve this threshold are then stored for later use in their respective modified elevation and velocity routines.

Within the program structure there is a spectral priming routine for the calculation of both the water surface elevation and the water particle kinematics, the separate calculation routines being based on the results of the specific spectral priming. Once the algorithm complexity is reduced, as previously discussed, there are simply two vectors containing pairs (primed pairs) of wave components that interact to produce significant second-order contributions. Accordingly, the Sharma & Dean (1981) algorithm can be modified to use only the primed pairs and can be easily exploited using parallel computing. The modified algorithm can simply be cast as an embarrassingly parallel reduction operation which lends itself well to both OpenMP and hybrid OpenMP-MPI programming models.

In the following sections all the results relating to the second-order wave model have been calculated using the *spectral priming* method with at least 90% of the second-order contribution to the calculation included. Generating second-order water surface and velocity profiles without this method, using exactly the same set of wave components as employed by the EPIC_BEM model, would have taken prohibitively large amounts of computation time and resources and therefore would not have been easily possible.

From a computational stance, it should be noted that the results presented in the following sections were produced using the OpenMP enabled version of spectral priming and the modified Sharma & Dean (1981) algorithm, whilst the hybrid OpenMP-MPI version is available within the EPIC_BEM code base to produce second-order input conditions for the Γ_{input} boundaries if desired.

It should also be noted that the spectral priming method was originally developed with the view of reducing the computational run time of the EPIC_BEM model. As the BEM scheme is primarily $O(n^2)$ in computational complexity, reducing n is key in reducing run times. To reduce the magnitude of n whilst keeping resolution constant, the computational domain must be reduced in size such that the number of nodes is reduced. As, in general, the water depth and vertical resolution is fixed, the only way of reducing the number of nodes in the NWT is by reducing dimensions of the free surface area that is simulated. In shrinking the free surface area, the distance from the input boundaries to the focal position for focussed linear random wave events is shorter. As a result, the wave field at the input boundaries is less dispersed, and therefore considerably steeper. Consequently, prescribing ϕ_n derived from linear random wave theory on the input boundary becomes invalid as the waves are too nonlinear. To overcome this issue, a second-order wave theory such as Sharma & Dean (1981) can be employed to describe ϕ_n on the input boundaries. However, as explained above, such a method is computationally intensive and therefore a new technique was required to overcome this challenge, hence the spectral priming method was developed.

The spectral priming algorithm was implemented such that the computational effort was distributed across multiple processors using an MPI-OpenMP framework, this reducing the run time even further. Unfortunately, in practice, even with the advancements noted above, it remains unclear whether the computational effort of calculating the second-order input for a smaller computational domain will consistently produce lower run times than the use of a larger domain and the computationally cheap linear random wave theory input. This is largely due to the rather unpredictable run times of the GMRES solver as the influence matrix condition worsens with time. However, ignoring the direct applicability of the spectral priming method to the EPIC_BEM code, the overall scheme is extremely successful and has many applications particularly with regards to long time domain simulations of random seas. In the section that follows all the second-order calculations were undertaken using the spectral priming algorithm.

7.5 Discussion of results

The results presented herein are believed to be the first of their kind. The key result is the evolution of a three dimensional breaking wave, arising in a sea state that is broad banded in both frequency and direction. In other words, the sea state is represented by both a realistic frequency spectrum (JONSWAP *NewWave*) and a realistic directional spread ($s = 7$). First, proof of the convergence of the wave profiles associated with different switching times is offered. Next, the evolution of the surface profile of the breaking wave event is analysed in both space and time with comparisons made to both linear and second-order wave theories. These results show that in the vicinity of the breaking event, the departures from the established theoretical models are quite remarkable. Attention is then turned to the kinematics field underlying the breaking wave with comparisons being made with both a focussed linear wave of identical crest elevation and a second-order wave theory. The section concludes with some suggestions relating to further work needed in the area of breaking waves.

7.5.1 Breaking wave profiles

In undertaking the high resolution numerical calculations it was found that a switching time of $t_{sw} = -17s$ resulted in the most complete description of the evolution of the breaking wave. To prove the convergent nature of the calculation based upon different switching times, the surface profiles arising along the centre line ($\eta(x)$ on $y = 0m$) immediately prior to the point at which the computation broke down are presented in Figure 7.1. Data relating to several different switching times (t_{sw}) are presented and, in each case, the closest match in time (adaptive time stepping means exact matches are impossible) from the $t_{sw} - 17s$ switching time case is also plotted. These comparisons confirm that regardless of the switching time employed, the evolution of the free surface is essentially identical. This proves that the switching time has no influence over the evolution of surface profile, it simply affects the end point of the numerical calculations; the latter being critically dependent on the local node spacing which is in turn dependent on the switching time t_{sw}. In the results that follow, all of the data relate to a switching time of $t_{sw} = -17s$, such that data allowing the furthest possible evolution of the overturning wave is employed.

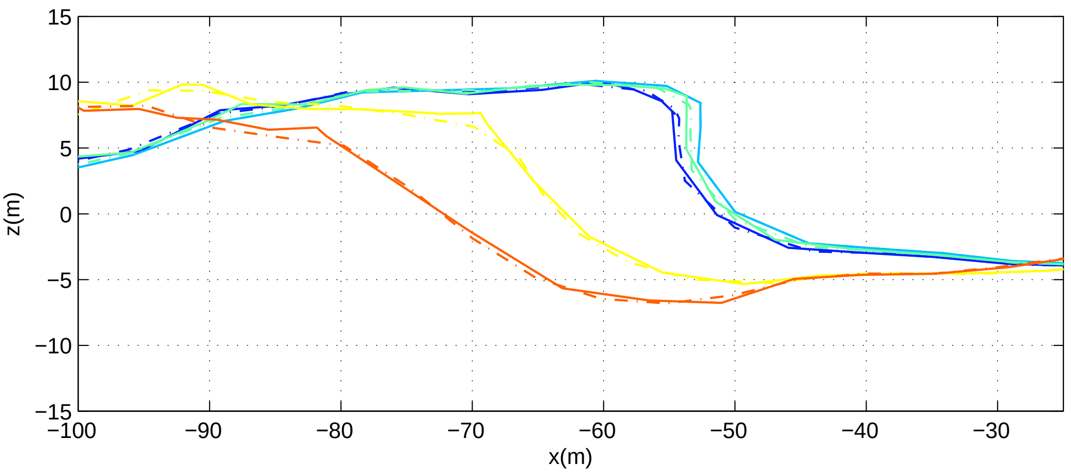
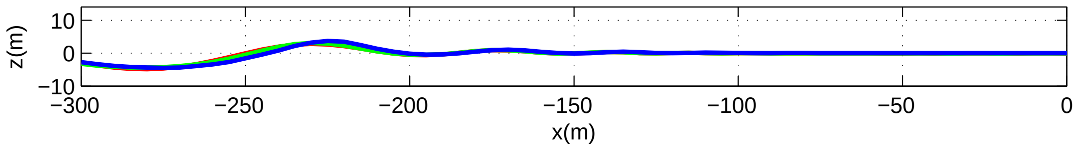
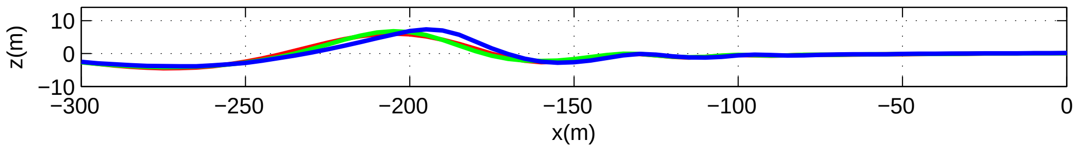


Figure 7.1: Spatial description of the water surface elevation, $\eta(x)$, on $y = 0$; comparisons between different switching times and data corresponding to the equivalent times (indicated by line style ($\cdot - \cdot$) of the same colour) taken from the best case switching time of $t_{sw} = -17s$. Legend: $-17s$ (——), $-18s$ (——), $-19s$ (——), $-20s$ (——), $-21s$ (——).

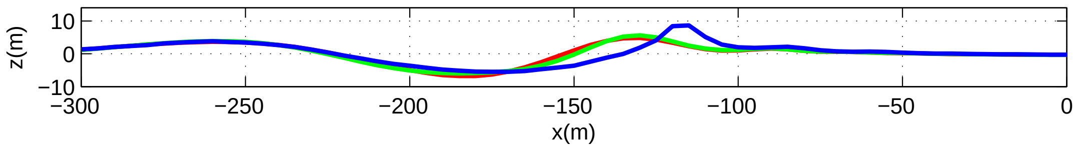
Having established that the use of different switching times produces convergent results, further analysis can be undertaken. Figure 7.2 provides a series of plots in time that show the evolution of the surface profile taken along the centre line ($y = 0m$) of the domain. In addition to the nonlinear profile, calculations based upon a linear solution (see equation (6.6)) and the second-order solution of Sharma & Dean (1981) (see §7.4.1) are superimposed. It is clear from these comparisons that the fully nonlinear sea state evolves in a very different manner from that predicted by the analytical solutions. In less steep wave situations arising at earlier times ($t = -42.2s$ and $t = -32.2s$ in Figures 7.2(a) and 7.2(b)) a reasonable level of agreement is observed. However, at times beyond this point the evolution of the fully nonlinear free surface is markedly different, most noticeably so in Figure 7.2(h) in which the wave overturns. It is apparent from the figures showing the time evolution of the breaking wave that the fully nonlinear wave has a higher phase velocity than the waves predicted by analytical theories. This behaviour is entirely consistent with both the experimental and the numerical observations reported by Johannessen & Swan (2001, 2003).



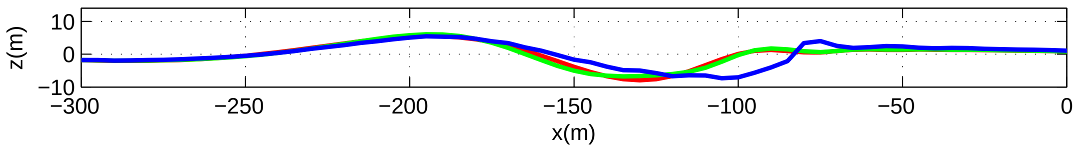
(a) $t = -42.2s$



(b) $t = -32.2s$



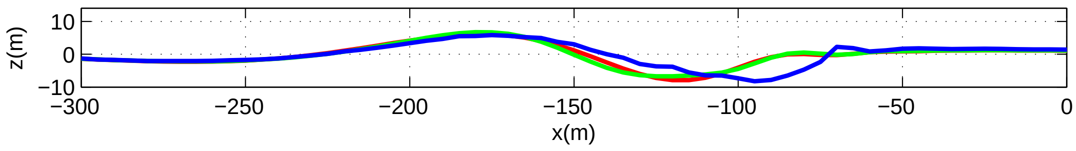
(c) $t = -26.2s$



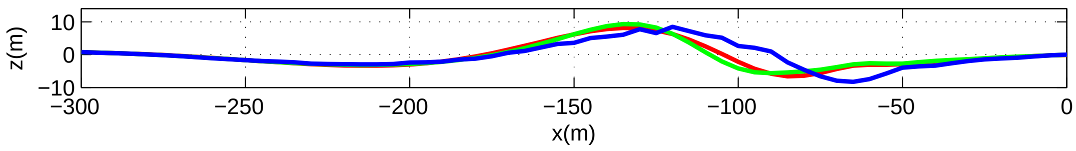
(d) $t = -22.2s$

Figure 7.2: Evolution of the spatial surface profile, $\eta(x)$ on $y = 0m$, at varying times (t); comparisons between linear (——), second-order (——) and fully nonlinear (——) computations.

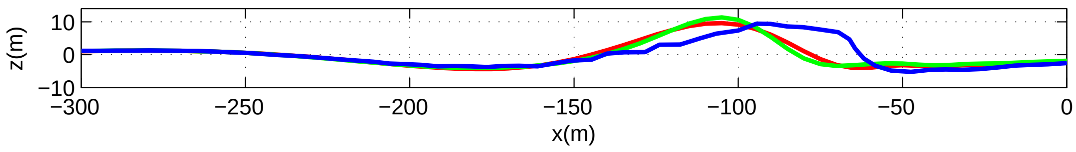
Figure 7.3 concerns the transverse variation of the spatial profile of the breaking wave, $\eta(t)$ at varying y, and presents comparisons with the linear and second-order analytical solutions displaying similar trends to those identified in the time domain.



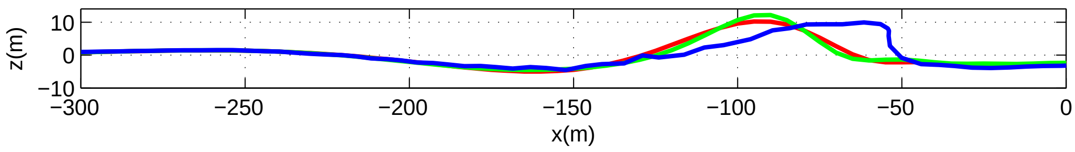
(e) $t = -21.2s$



(f) $t = -18.2s$



(g) $t = -16.2s$

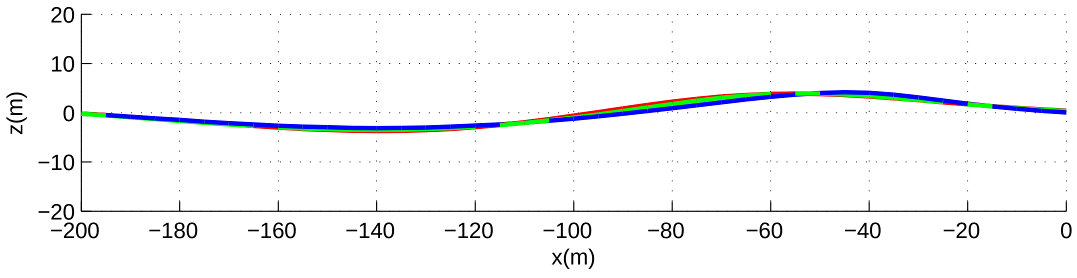


(h) $t = -15.2s$

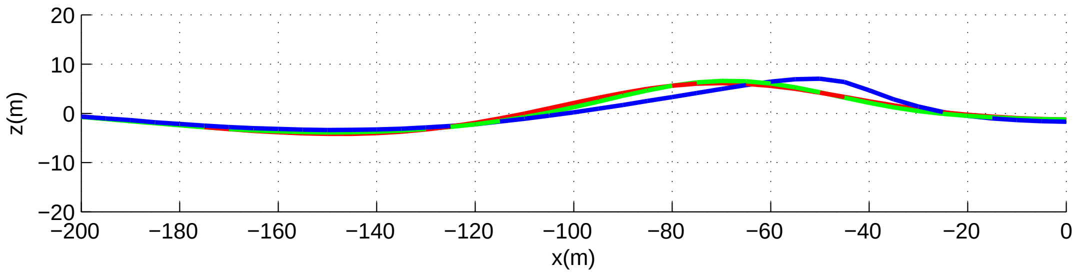
Figure 7.2: (continued). Evolution of the spatial surface profile, $\eta(x)$ on $y = 0m$, at varying times (t); comparisons between linear (🟥), second-order (🟩) and fully nonlinear (🟦) computations.

Within this figure, the individual subplots display cross sections in the x direction taken at the y location specified under each plot. All the plots relate to the furthest point of wave evolution prior to the break down of the computation. Comparisons with the linear and second-order analytical solutions show reasonable agreement for the less steep wave profiles occurring some distance from the centreline; $y = -90m$ on Figure 7.3(a). However, as the cross sections are taken at positions progressively closer to the centre line of the domain (Figures 7.3(b) – 7.3(h)) the departure of the fully nonlinear predictions from the analytical theories becomes more rapid. The final cross section at $y = 0m$ is identical to that in Figure 7.2(h) as the spatial and time profiles coincide at this point. To reiterate the comments made previously, the difference between analytical and fully nonlinear profiles is most apparent at this time and position; the differences having obvious practical implications.

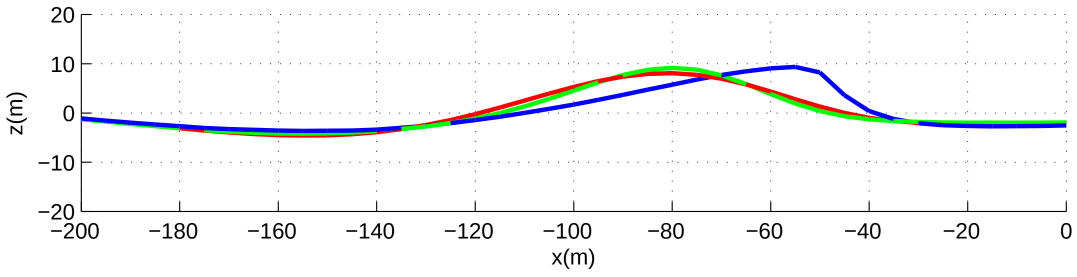
Figures 7.2 and 7.3 evidently provide a two dimensional interpretation of the evolving wave field. In contrast, Figures 7.4 and 7.5 provide a number of three-dimensional plots indicating the evolution of the wave field at varying times (Figure 7.4) and a close up of the predicted wave form (Figure 7.5) as it undergoes wave overturning. Taken together, these figures allow some insight into the way in which an extreme ocean wave evolves and breaks. The first observation is simply that the wave predictions are consistent with the notion of a large breaking wave appearing, apparently, from nowhere. This is commonly reported by mariners, several examples being reported by Lawton (2001) and Kharif & Pelinovsky (2003). In respect of the present calculations, it is clear that outside the immediate vicinity of the breaking wave the wave motion is substantially smaller. In fact it is smaller than that predicted by linear theory because the large wave has evolved due to nonlinear energy gains from the surrounding sea. With respect to the overall shape of the breaking wave, it evidently has a very flat top in both horizontal dimensions (x, y). Again, this is likely to be caused by energy exchanges in the local wave spectrum forcing more energy into wave components aligned with the mean wave direction (Gibson & Swan, 2007). As a result, the wave becomes more long crested than would be envisaged based upon the underlying directional spread.



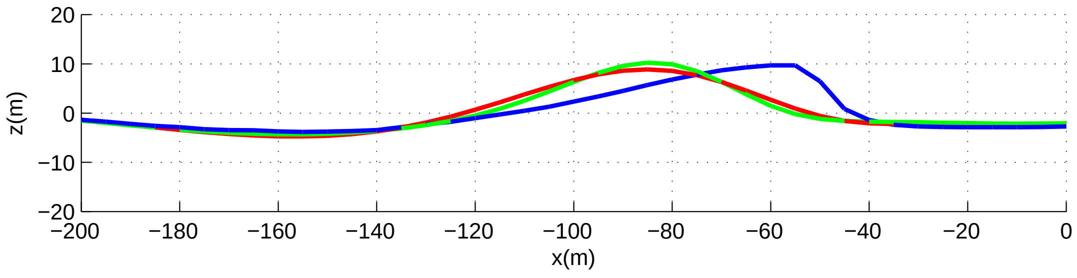
(a) $y = -90m$



(b) $y = -70m$

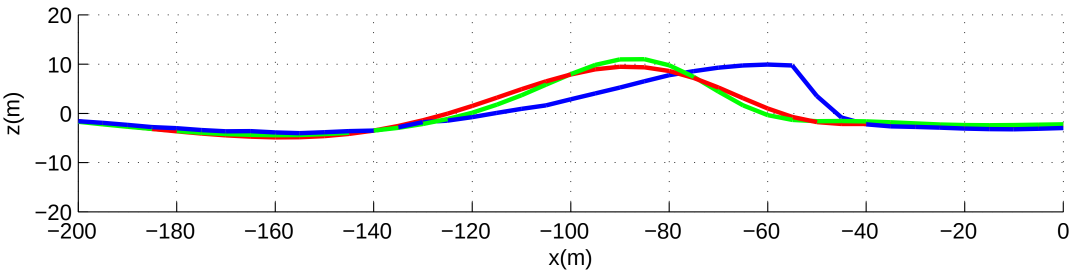


(c) $y = -50m$

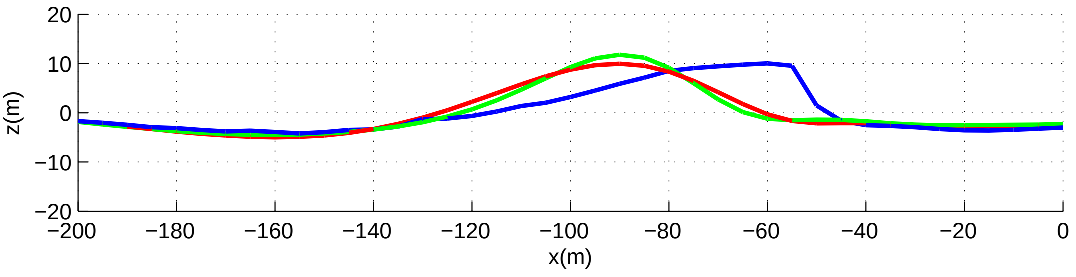


(d) $y = -40m$

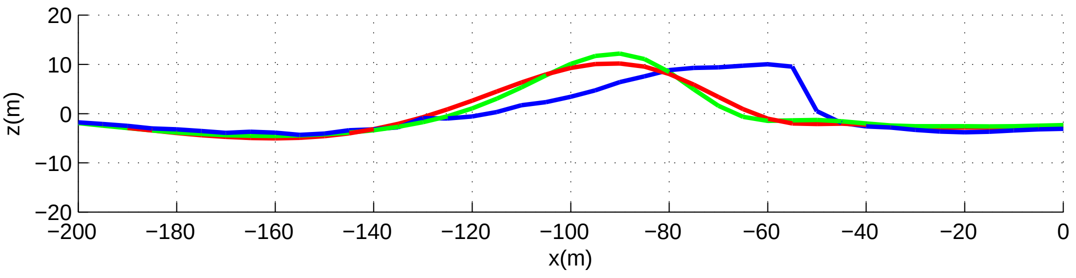
Figure 7.3: A spatial description of the water surface elevation at the instant of wave breaking, $\eta(x)$ at $t = -15.2s$. Comparisons between linear (🟥), second-order (🟩) and fully nonlinear (🟦) computations at varying cross sections.



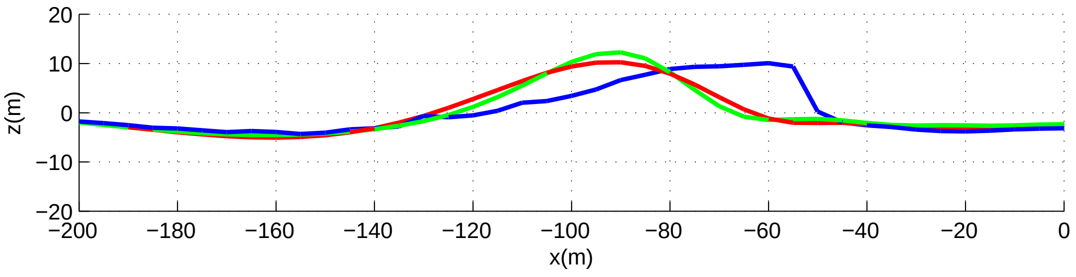
(e) $y = -30m$



(f) $y = -20m$



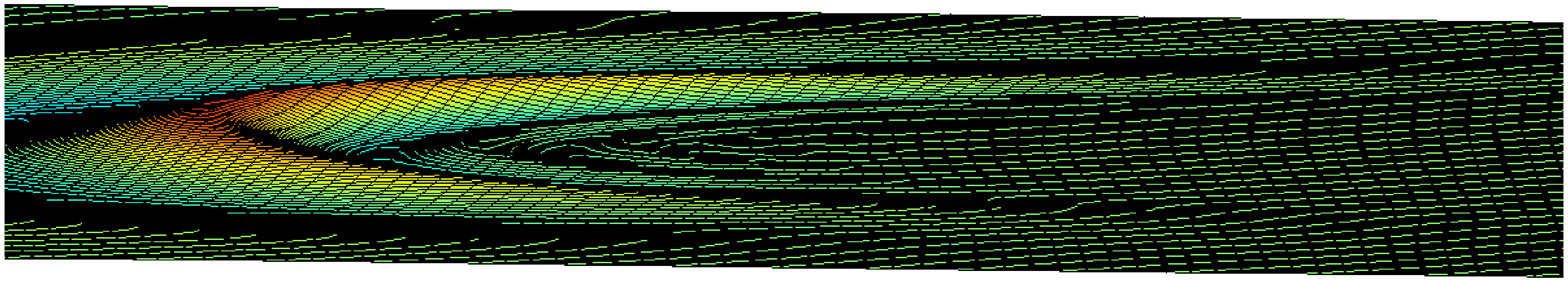
(g) $y = -10m$



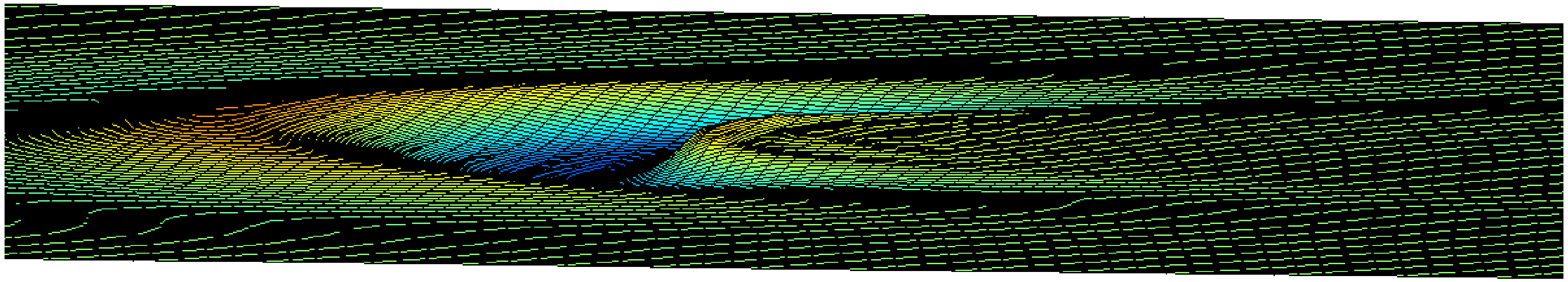
(h) $y = -5m$

Figure 7.3: (continued). A spatial description of the water surface elevation at the instant of wave breaking, $\eta(x)$ at $t = -15.2s$. Comparisons between linear (—), second-order (—) and fully nonlinear (—) computations at varying cross sections.

In addition, the wave face is exceedingly steep with water being thrown forward as part of the breaking jet. This motion will further exacerbate the flat topped nature of the breaking wave and may even account for a small depression (corresponding to a locally reduced crest elevation) forming along the centreline of the wave form. In terms of the shape of the breaking wave face, it is fairly broad with a relatively shallow trough. Furthermore, higher surface elevations (presenting "shoulders" to the wave) can be seen to form at either side of the jet formation where the wave is insufficiently steep to break. As a consequence of this formation, the face that this wave would present to a shipping vessel or offshore structure would be approximately $15m$ high by $60m$ wide. This appears to be entirely consistent with the frequently used description of a "wall of water". Although this may not seem particularly formidable in comparison to the waves described in Chapter 6, the kinematic results presented in the next section clearly suggest otherwise.

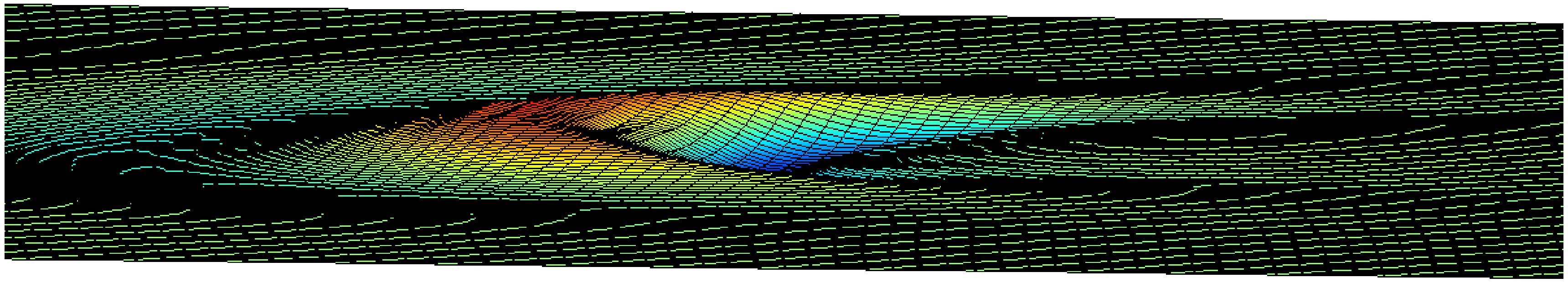


(a) $t = -42.2s$

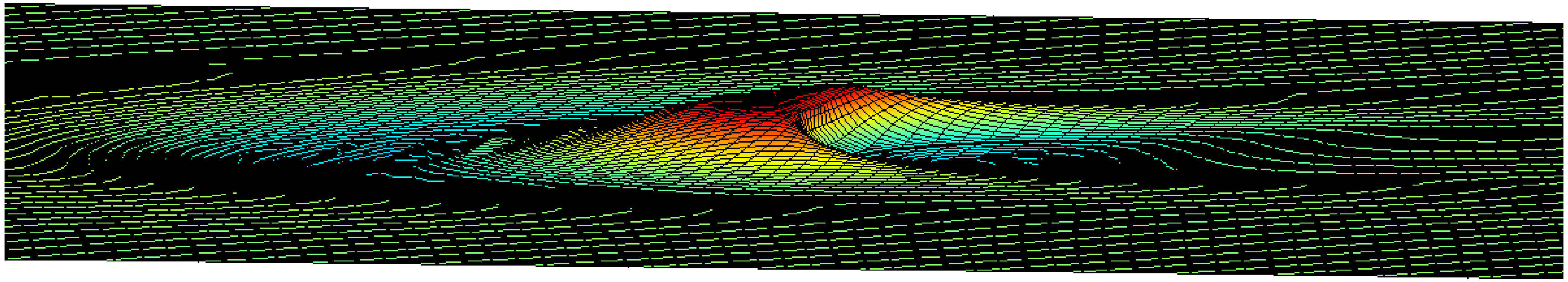


(b) $t = -32.2s$

Figure 7.4: Evolution of the water surface, $\eta(x, y)$, at varying times (t).



(c) $t = -18.2s$



(d) $t = -15.2s$

Figure 7.4: (continued). Evolution of the water surface, $\eta(x, y)$, at varying times (t).

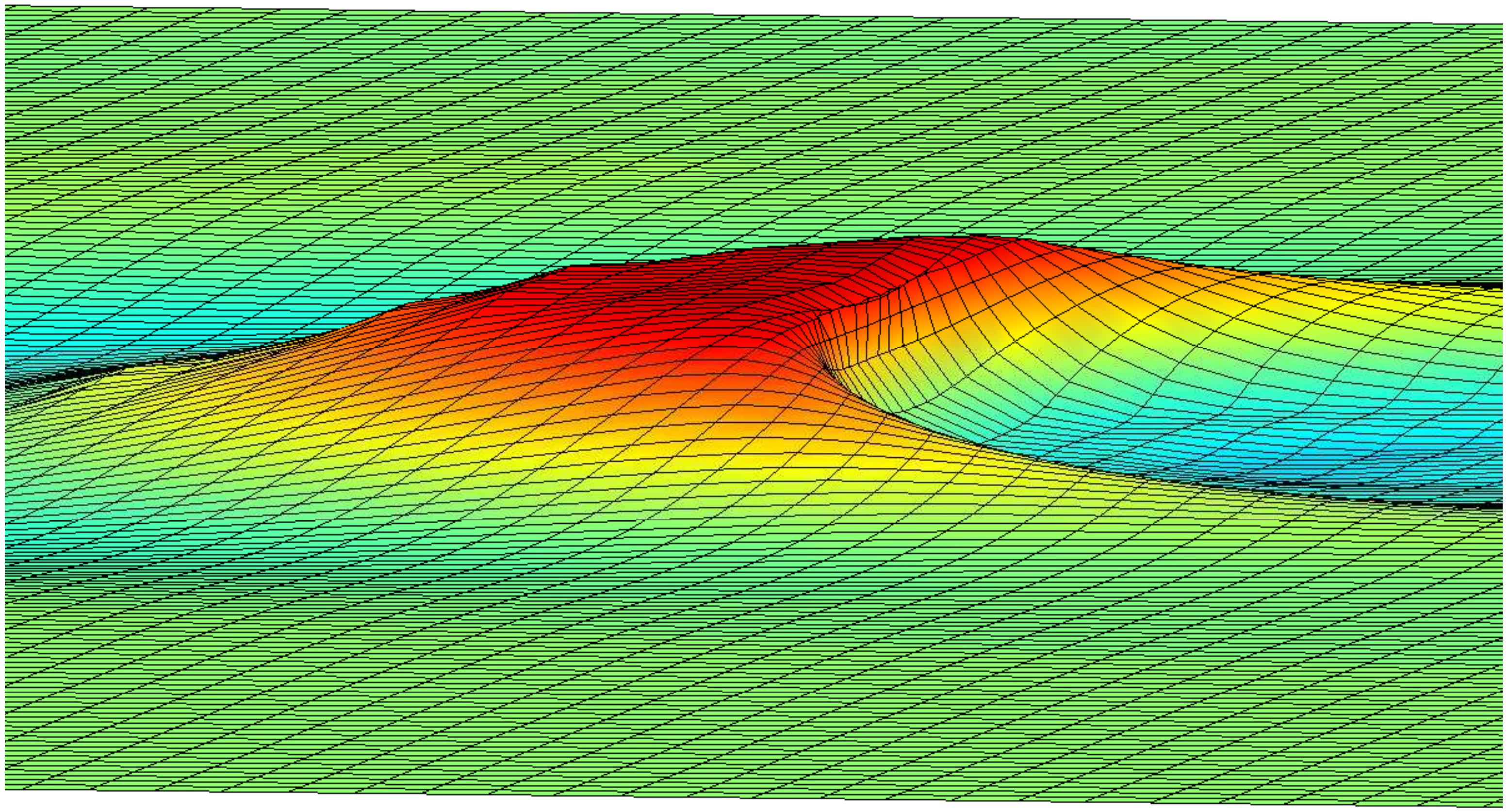
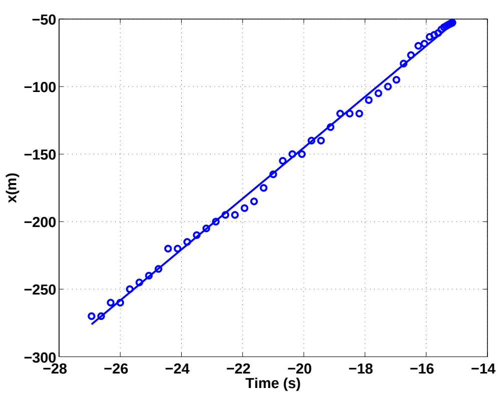


Figure 7.5: Enhanced view of Figure 7.4(d), a 3D image of the overturning wave taken at time, $t = -15.2$, rendered using OpenGL patches.

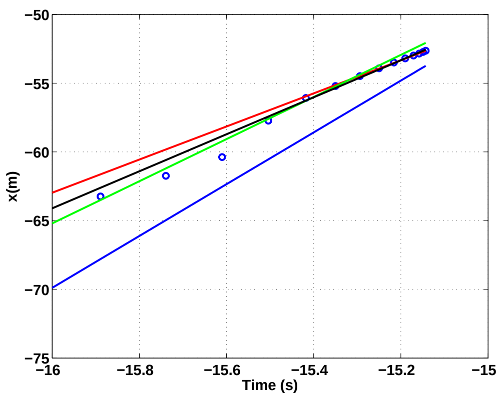
7.5.2 Kinematics predictions beneath the breaking wave

In considering the water particle kinematics underlying the breaking wave, the analysis begins by quantifying the horizontal velocity in the x direction with respect to the phase velocity (c) of the breaking wave. To obtain the phase velocity of a wave it is common to track the position of the wave crest as it evolves in time, thereby giving the necessary space-time relationship from which c can be calculated. For a non-breaking wave case, this task is relatively straight forward as the crest is well defined and can be found by searching for the maximum local elevation of the free surface at a given instant in time. Unfortunately, the 3D breaking wave produced in this study has a very flat top (as noted previously). This feature makes it very difficult to decide the exact position of the wave crest for a given time; a simple search for the maximum local elevation is not suitable. To overcome this difficulty, the evolution of the breaking wave crest in the present study was estimated by visual means, the relevent data is presented on Figure 7.6(a).

To compute the phase velocity of the breaking wave, the gradient of the estimated crest positions with respect to time can be taken. To this end, a linear least squares fit was applied to all the data (Figure 7.6(a), —) and the gradient of this line gives a phase velocity of $c = 18.85 m/s$. However, recent work suggests that the wave crest should slow down as it approaches the breaking point, the physical explanation for this is as follows. As the wave becomes very steep, the energy distribution in the local spectrum changes with lower frequency wave components transferring energy to higher frequency wave components. This has been observed by a number of researchers, most notably Johannessen & Swan (2003) and Gibson & Swan (2007). The explanation of this shift is the rapid growth of third-order resonant (or near resonant) wave components. By simply considering the dispersion relationship (equation (6.8)) it can be seen that higher frequency wave components propagate more slowly. As a result, this energy transfer in the vicinity of a large wave event should result in a gradual reduction in phase velocity as the wave steepens. Taking this information into account, a number of alternative (perhaps better) estimates of the phase velocity of the breaking wave can be



(a)



(b)

Figure 7.6: Position of the breaking wave crest as it evolves in time (o) with linear least squares best fit based upon: all data points (——), data arising in the 0.2s between the wave becoming vertical and the end of the computational run (——), data arising in the 0.2s before the wave face becomes vertical (——) and data from the 0.2s either side of the wave face becoming vertical (——). Note: Sub-figure (a) displays all data and corresponding least squares fit. Sub-figure (b) is a close up of (a) with the addition of lines representing the alternative data fitting criteria.

obtained by using only data relating to when the wave becomes particularly steep (Figure 7.6(b)). The first approach considers only data arising from the point at which the wave face becomes vertical ($t \simeq -15.3s$) to the point of computational breakdown (approximately $0.2s$ later) results in a phase velocity of $c = 12.02m/s$ and is denoted by (——) on Figure 7.6(b). Second, data relating to the time interval of approximately $0.2s$ immediately prior to the wave face becoming vertical gives a phase velocity of $c = 15.32m/s$. Finally, data arising from the $0.2s$ before and after the wave face becomes vertical results in a phase velocity of $13.47m/s$. Clearly, phase velocities calculated using information relating to the final stages of computation are slower than those calculated if all the data are used. Whilst this is consistent with the findings of Gibson & Swan (2007), as described previously, it is also apparent that the notion of a phase velocity is somewhat arbitrary and considerably harder to compute for waves without distinct (maximum) crest elevation. In light of this difficulty, the phase velocity arising from crest position data both before and after the wave face becomes vertical ($c = 13.47m/s$) will be used for normalisation purposes.

For wave breaking to occur, it is usually anticipated that the x component of the horizontal water particle velocity (or the component in line with the mean wave direction) exceeds the phase velocity. This condition is met when the wave face becomes vertical at $-15.315s$; where the x component of the maximum horizontal fluid velocity is $14.32m/s$. At this point, the ratio of the water particle velocity to the phase velocity is defined by $c_{rat} = 1.063$. This result is interesting, not least because it is in stark contrast to the equivalent 2D calculations presented by Christou (2008) which gives c_{rat} values in the range of $1.21 - 1.29$. The larger ratios observed by Christou are likely to arise because the model he employed was two dimensional and therefore incapable of incorporating directional effects. Indeed, when observing the real ocean, most waves that break do so through collapsed plunging jets or local spilling at or near the wave crest (New *et al.*, 1985). This is consistent with a small c_{rat} and reinforces the realism of the presented 3D breaking wave.

With regards to the overall magnitude of the velocities present in the breaking wave, the evolution of the velocity field on the centre line ($y = 0$) can be seen in

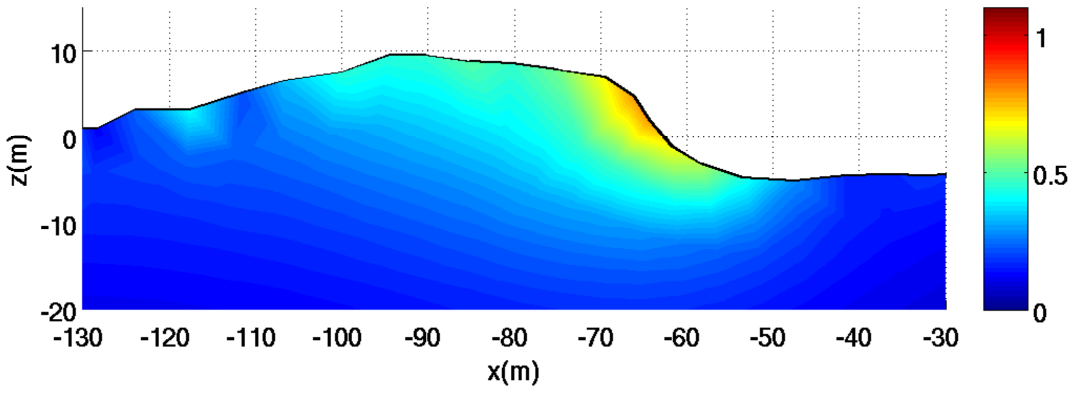
Figure 7.7. The colour scheme in these images is set such that it indicates the magnitude of the velocity field normalised against the phase velocity of the wave,

$$\frac{\sqrt{u^2 + v^2 + w^2}}{c}, \tag{7.16}$$

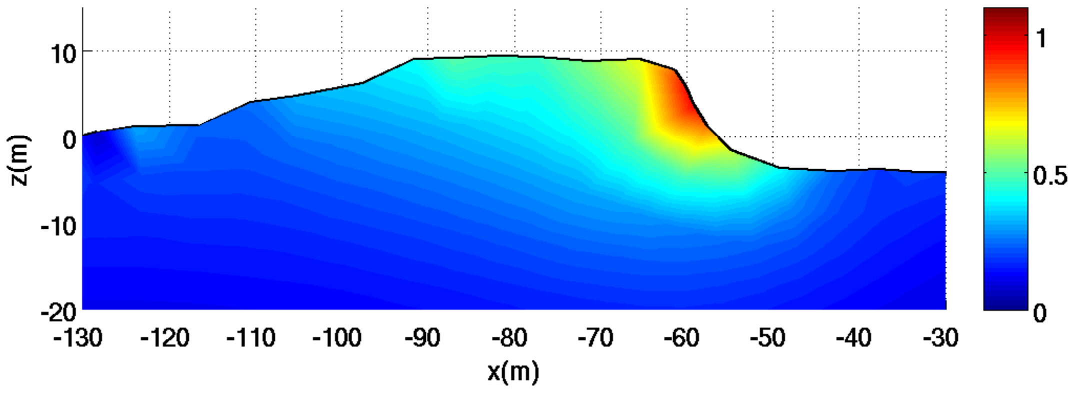
where (u, v, w) are the three components of the wave induced fluid velocity. This normalisation makes it considerably easier to visualise that portion of the wave that matches, or exceeds, the phase velocity of the wave, this being a key feature in the following analysis.

From the sequence of images in Figure 7.7, a number of interesting conclusions can be drawn. To begin with, the time elapsed between the first (Figure 7.7(a)) and last image (Figure 7.7(c)) is approximately one second. This indicates a very rapid evolution of both the free surface and the associated fluid velocities; the latter implying very large fluid accelerations. What is perhaps more remarkable is that in Figure 7.7(a) the magnitude of the fluid velocity barely reaches $0.75c$ with the majority of the fluid that makes up the wave crest having a velocity magnitude of approximately $0.5c$. In contrast, a mere half a second later (Figure 7.7(b)), a significant volume of the wave face has a velocity magnitude near that of the phase velocity, but the bulk of the wave crest still remains with a velocity magnitude of near $0.5c$. Finally, the velocity magnitudes near the point of computational breakdown are displayed in Figure 7.7(c). At this point, the wave has started to overturn and, as expected, the magnitude of the velocity at the breaking wave face exceeds the phase velocity of the wave. It is also apparent that all the water on the wave face and above the still water line has a velocity magnitude at or above that of the phase velocity. However, it should also be noted that the magnitude of the velocity below still water level, and outside the immediate influence of the breaking wave face, is very small and remains unchanged over the time interval under consideration.

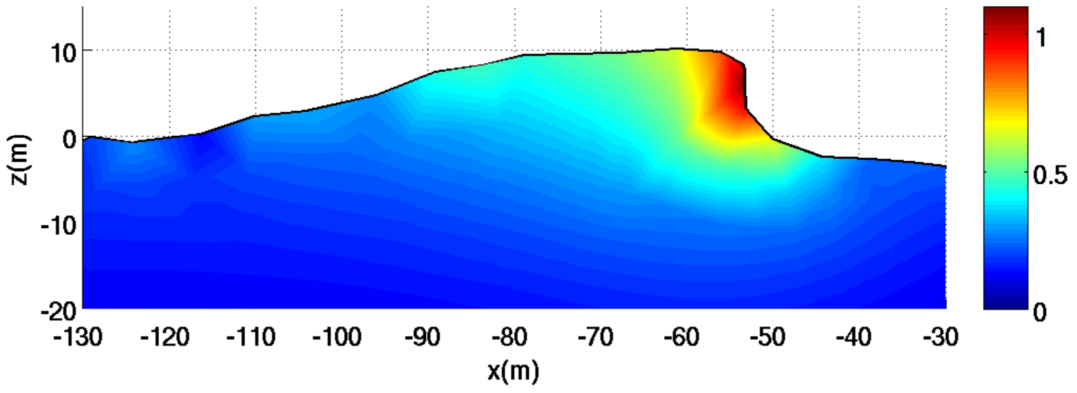
To enable further comparisons with the breaking wave case, velocity predictions from a number of additional wave models have been employed. For each model, the parameters in Table 7.2 were used to describe the distribution of the wave components in both the frequency and directional domain. Each wave was then generated as a focussed wave event with a crest elevation specified to match the



(a) $t = -16.2s$



(b) $t = -15.7s$



(c) $t = -15.2s$

Figure 7.7: Spatial plots of the water particle kinematics along the centre line ($y = 0m$) of a directionally spread breaking wave. The colour bars indicate the magnitude of the water particle kinematics, $\sqrt{u^2 + v^2 + w^2}$, normalised with respect to the phase velocity of the wave ($c = 13.47m/s$). The time at which each snap shot was taken is indicated below each sub-plot.

maximum found in the case of the breaking wave ($\eta_{max} = 10.11m$). In total, three alternative wave models were chosen for comparison purposes. First, an analytical linear model derived from linear random wave theory (equation (6.7)) with empirical stretching applied following Wheeler (1970). Second, an analytical second-order model after Sharma & Dean (1981) and implemented as outlined in §7.4.1. Finally, the fully nonlinear EPIC_BEM solution as outlined herein was also applied (as described above) to create an 'equivalent' non-breaking wave case.

Figure 7.8 concerns each of the three wave models, superimposing the predicted spatial variation of the water surface along the centre line of the domain ($\eta(x)$ on $y = 0m$). In addition, the profile corresponding to the breaking wave case has also been superimposed. Individual wave profiles have, where necessary, been shifted in space such that the maximum elevation for each model aligns with the point ($x = 0m$); the shift in coordinates simply employed to facilitate comparisons between the different predictions of the water surface. At the position $x = 0m$, the depth variation in the x-component of the wave induced velocities has been calculated for each of the four solutions and the data presented on Figure 7.9. In the case of the breaking wave, the velocities were actually computed on the section ($x = 0m, y = -2m$) with relation to Figure 7.8. This small shift in the location has no bearing on the magnitude of the predicted velocities, but allows the data to be calculated using the internal kinematics scheme presented in §6.2.1. This, in turn, allows the resolution of the velocity profile to be improved relative to that which is available on the boundary of the domain ($y = 0m$).

In considering these comparisons, it is clear that the linear, second-order and fully nonlinear (non-breaking) velocity profiles shown in Figure 7.9 are very similar. In particular, they all have approximately the same profile shape. The only obvious difference is present in the second-order velocity profile in the range ($5m \leq z \leq 10.11m$) where the solution has clearly broken down. This effect arises due to inconsistencies in the way in which the second-order solution predicts the velocities arising from the wave components in the tail of the spectrum. This effect has been noted by others (notably Jensen (2004)), only arises in steep wave conditions, and cannot be eliminated by a simple truncation of the frequency spectrum. Indeed, in the present calculations the frequency spectrum has been truncated at three

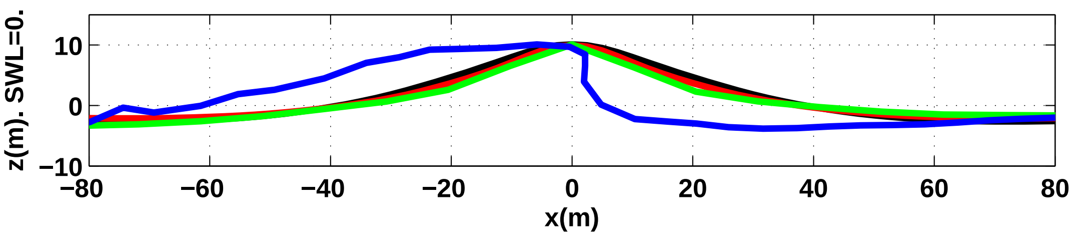


Figure 7.8: Spatial representation of the water surface elevation ($\eta(x)$) along the centre line of the domain ($y = 0m$). Linear theory (——), second-order theory after Sharma & Dean (1981) (——), fully nonlinear BEM profile applied to a non-breaking and matched to the predicted crest elevation (——), fully nonlinear BEM from the breaking wave simulation (——). Note: all profiles are shifted in space such that their maximum elevation is located at $x = 0m$.

times the spectral peak ($a(\omega) = 0 \ \forall \ \omega \geq 3\omega_p$, where ω_p defines the spectral peak).

In making comparisons with the breaking wave profile, it is clear that there is reasonable correspondence with the other profiles from the sea bed to approximately $10m$ below still water level. At this point there is a rapid departure from the other three profiles; the breaking wave case achieving a maximum horizontal velocity of $12.85m/s = 0.95c$. This maximum is just over twice the value of the maximum velocities predicted by the other three wave models despite the fact that all the wave models have been applied to identical crest elevations. The velocity profiles presented on Figure 7.9 clearly indicate that as far as the water particle kinematics are concerned, the occurrence of wave breaking is fundamentally a near-surface issue. This being evident from the kinematics at some depth beneath the still water level being largely unchanged. However, very significant changes can occur above still water level, and particularly high in the wave crest. Indeed the increase in the maximum fluid velocities can be very substantial and will have important implications in relation to structures and vessels for both local (impact) loading and global loads, particularly the total overturning moment.

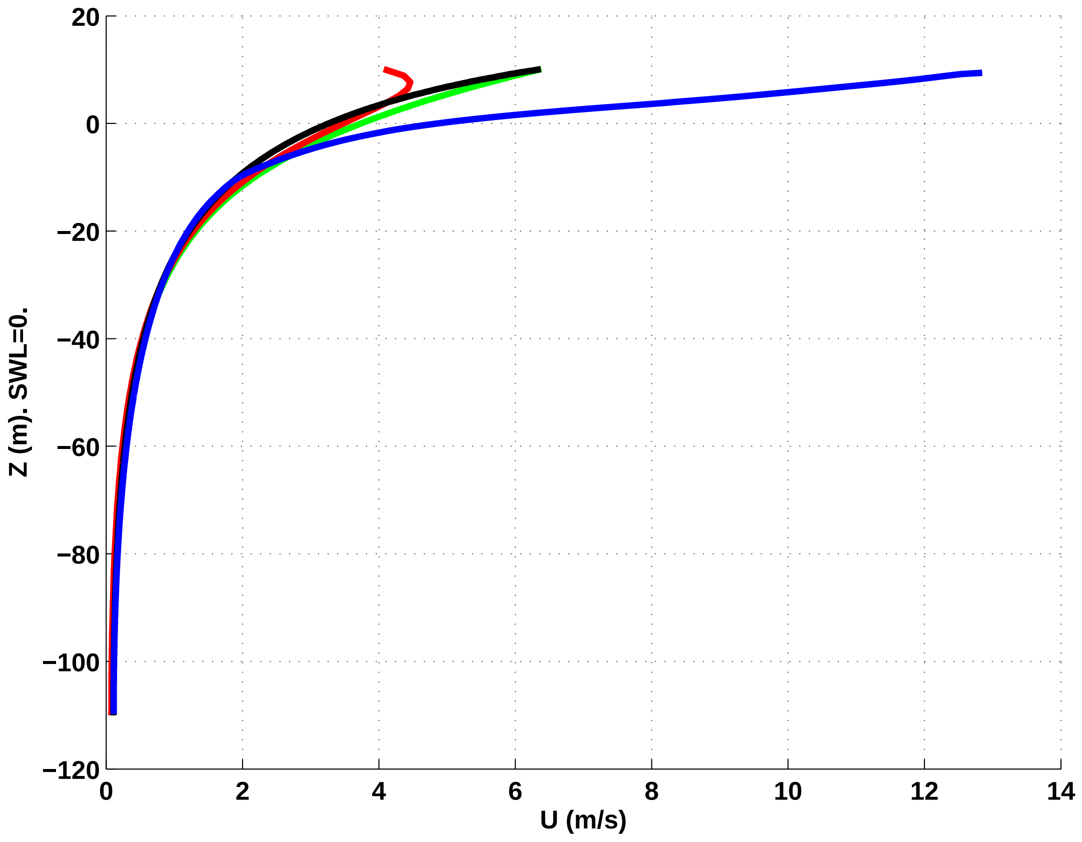


Figure 7.9: Depth variation in the x-component of the horizontal velocity at $x = 0m$. Linear theory with Wheeler (1970) stretching (——), second-order theory after Sharma & Dean (1981) (——), fully nonlinear BEM profile applied to a non-breaking and matched to the predicted crest elevation (——), fully nonlinear BEM from the breaking wave simulation (——).

7.6 Conclusions

This chapter has involved the application of the EPIC_BEM solution to the description of a breaking wave event in a realistic sea state. Although wave breaking is an attractive topic from an academic point of view, evidence of this being provided by the number of publications in the area, the provision of results relating to waves arising from realistic ocean conditions is rare. The present study has produced an example of such a result in three dimensional space and this is believed to be the first of its kind. In addition to the calculation of a breaking wave event, the evolution of both the wave profile and the water particle kinematics has been considered. Perhaps the most striking result that can be taken from the present work is the very rapid evolution of the wave event, both in terms of the wave profile and water particle velocities. Furthermore, it has been shown that the breaking wave event is highly localised in both space and time. Indeed, it has been shown that in both space and time, departures in the free surface profile from the well known wave theories is relatively small outside the immediate vicinity of the breaking wave event. Similarly, the velocity profile, $u(z)$, associated with the breaking wave event only differs from that computed using standard models in the very upper most parts of the water column. However, this difference can be substantial in the breaking wave crest where velocities are twice that predicted by other means. Clearly, the engineering implications of such rapid wave evolution and large crest velocities are considerable. This is notable particularly in relation to the loads associated with wave slamming where the magnitude of the load is proportional to the square of the incident fluid velocity.

8

Concluding Remarks

The primary goal of the present study was the development and application of a numerical model that is capable of dealing with relatively large computational domains, with a high spatial resolution, to enable the accurate prediction of extreme wave events and their associated water particle kinematics. The primary goal was split into a series of tasks, set out below:

- Formulate a BEM to allow the modelling of large, complex wave fields with high accuracy.
- Validate the model to ensure that it performs correctly.
- Deal with the computationally intensive aspects of the BEM such that the solution time for the model becomes tolerable for practical applications.
- Apply the model to a practical engineering problem requiring state-of-the-art wave predictions coupled with high spatial and temporal resolution.
- Use the model to investigate extreme ocean waves and wave breaking.

8.1 Principal achievements

The above tasks have been completed in the following manner:

a) A full discussion of a BEM scheme applied to a NWT has been provided; specific attention being paid to details of the mathematical model, the boundary conditions, fundamental numerical methods, core algorithms and the implementation choices available.

b) The development of a modular code base including:

   (a) A source code organisation and revision scheme.

   (b) The use of the GNU Autotools tool chain to detect, set-up and build the EPIC_BEM code.

   (c) Support for multiple programming languages and environments within the tool chain.

   (d) Modular `Matlab`/GNU `Octave` code for the assembly of the NWT with the scope to easily implement additional features within the NWT.

   (e) A patch and path builder, hooking into the OpenGL libraries, to allow 3D visualisations of the NWT.

c) The application of a multiple-flux BEM to a 3D NWT.

d) Providing a parallel implementation of a block decomposition method for assembling the BEM influence matrices in a distributed computing environment with MPI support.

e) An investigation into, and the application of, accelerator hardware in the form of NVIDIA CUDA enabled GPUs for solving linear systems via the IDR(s) method. This included much discussion on the nature of the hardware and techniques for overcoming common problems and was followed by the description of a work flow to map serial CPU based code to code for the massively parallel GPU architecture.

f) Some insight into the application of multiple levels of parallel computing behaviour. This was achieved through the coupling of a distributed influence matrix assembly scheme with an accelerated linear system solving scheme.

g) The validation of the EPIC_BEM 3D NWT free surface modelling in relation to both regular waves and irregular waves.

h) The design and implementation of a self scheduling, distributed, method of computing kinematics internal to a NWT. This included some discussion of methods to mitigate the "boundary layer" problem and highlighted methods of increasing algorithmic efficiency; the overall scheme being fully validated with respect to both regular and irregular waves.

i) The developed wave model has been applied to the description of a 10^{-4} design wave condition and the kinematics prediction used for a re-appraisal of the applied sub-structure loads on a typical jacket structure. Substantial loads reductions are identified. In addition, comments are made regarding the validity of the current APInst loading recipe and suggestions on its future improvement.

j) A review of the current state of the numerical simulation of wave breaking, particularly in respect of modelling realistic sea states. This was followed by some discussion on the importance of the non-orthogonal nature of the mapping to the reference coordinate system resulting from element distortion.

k) The presentation of the first three dimensional, fully nonlinear, directionally-spread, breaking wave event based upon a realistic ocean wave spectrum. Included in these results were the kinematics fields associated with these waves, with comparisons to their non-breaking equivalent focussed waves.

In summary, the targets set out at the beginning of the dissertation have been achieved. As a result, there have been successes in a number of areas and the final numerical model is comparable to the state-of-the-art. However, in the endeavour of research there is always scope for improvement and some of these areas are discussed shortly. First, some comments on the relevance of the work to engineering practice must be made.

8.2 Engineering significance

The present work is of significance in a number of fields. Clearly there is considerable benefit in the use of a numerical model, like EPIC_BEM, in the calculation of the loading on an offshore jacket structure; the accurate computation of the water particle velocities being key to the study. As a consequence of the present investigation, there also arises the question of whether overturning waves with reduced wave heights could produce larger local and perhaps also global loads. Some insight into this is given in §7.5.2. This highlights the necessity for the accurate modelling of realistic 3D wave breaking, with direct applications in offshore engineering design.

In addition to the offshore engineering application, some advances have been made in the field of software engineering. The most important of these lies in the coupling of a distributed and accelerated code for respectively forming and solving the BIE (equation 2.2). This code runs on a distributed computing framework with centralised GPU acceleration and is an indication of one of the possible future directions to be taken by software engineering developed in the context of numerical modelling.

8.3 Further work

In keeping with the theme of the thesis, there are two main areas for discussion with regards to the scope of further work. The first concerns further work required in the field of wave modelling, while the second looks at the use of computing techniques and hardware.

8.3.1 Wave modelling

There are a number of improvements possible to enhance the accurate modelling of free surface waves within EPIC_BEM. A few suggestions for further research are given as follows.

Computing boundary kinematics

In Chapter 7 it was seen that a point often forms at the tip of the overturning wave crest and this is generally modelled by a single node. Clearly, fitting a polynomial around such a large geometric irregularity can cause considerable problems with accuracy, especially with regard to the continuity of velocities around the plunging part of the wave. In considering this issue, a new formulation to completely avoid the use of polynomial approximations for calculating velocities on the fluid boundary would be highly desirable.

Based on a similar theory to that proposed in §6.2.1 for computing internal kinematics, it should be possible to compute the velocity vector at a given node on the boundary by employing the BIE. In effect, equation (6.1) could be used in the exact same form as for the internal kinematics calculations. However, there are two additional problems. The first is associated with the occurrence of singularities because $|\mathbf{r}| \to 0$. However, it is anticipated that these can be mitigated using the methods described previously. Second, the c_p term (the external solid angle) is not zero as the node of interest is on the boundary. As a result, this term has to be explicitly evaluated. Nevertheless, since both Φ and Φ_n are known *a priori* from the solution of the BIE (2.2), the following simple manipulation of the BIE leads to an expression for the velocity vector at point p. To begin, the exterior solid angle c_p can be computed from

$$c_p = \int_\Gamma \left[G \frac{\partial \phi_q}{\partial n} - \phi_q \frac{\partial G}{\partial n} \right] d\Gamma / \phi_p, \tag{8.1}$$

where the symbols have the same meaning as given in §2.2. Furthermore, taking the spatially differentiated BIE given in equation (6.2) and rearranging gives

$$c_p \nabla \phi_p = \int_\Gamma \left[Q \frac{\partial \phi_q}{\partial n} - \phi_q \frac{\partial Q}{\partial n} \right] d\Gamma, \tag{8.2}$$

where the symbols have the same meaning as given in §6.2.1. Combining these equations and eliminating c_p gives

$$\mathbf{u}_p = \frac{\phi_p \int_\Gamma \left[Q \frac{\partial \phi_q}{\partial n} - \phi_q \frac{\partial Q}{\partial n} \right] d\Gamma}{\int_\Gamma \left[G \frac{\partial \phi_q}{\partial n} - \phi_q \frac{\partial G}{\partial n} \right] d\Gamma}, \tag{8.3}$$

which defines the velocity vector at point p. In terms of a method for implementing the mathematics, a decomposition similar to that outlined in Chapter 3 would be sensible, given the obviously heavy computational workload required.

Automatic switching

In the present implementation of the EPIC_BEM code, user input defines t_{sw}, the time at which the boundary condition framework alters to allow the fully-Lagrangian movement of selected nodes. Although this method is good in the sense that it quickly develops the users' instinct for when a wave will break, the experimental nature of such an approach has to be questioned. A more intelligent approach would involve the projection of the movement of the surface nodes in a fully-Lagrangian frame of reference onto the computed movement of the nodes whilst remaining in a semi-Lagrangian frame of reference. If a good match is achieved then the semi-Lagrangian frame of reference is suitable for the wave conditions present. If the match is poor, then the model should automatically switch to using the fully-Lagrangian frame of reference. Such an approach would remove the somewhat arbitrary nature of the present methods which are largely based upon some wave steepness.

Absorbing input boundary

The input for the EPIC_BEM model is currently expressed in the form of the normal derivative of the velocity potential calculated using a user selected analytical wave theory; the derivative being applied to the nodes on the input boundaries at each time step. It would be of obvious benefit if the input boundaries had the ability to absorb incoming waves in the same manner as the radiating boundary. This would allow longer simulations of domains containing reflecting bodies such as ships (Peric, 2010). Since the potential derivative is known on the input boundaries and the gradient of the potential can be easily computed in the adjacent volume of fluid, implementing absorbing input boundaries does not represent a difficult task.

Post breaking wave behaviour

In Chapter 7, wave groups were simulated up to the point at which a plunging jet is formed. Unfortunately, computations break down at this point due to the proximity of nodes creating quasi-singular behaviour. In addition, the BEM domain can only be deformed elastically, it cannot be broken. Therefore, even if it were possible to simulate the plunging jet to the point of re-entry on the water surface, this would be the absolute limit of calculations possible using the BEM.

An alternative class of numerical methods revolve around the use of discrete particles to simulate a fluid flow; smoothed particle hydrodynamics being one example. Within such methods the fluid is modelled as a set of discrete units such that the fluid can separate and reform. Obviously, the separation and reformation of a fluid body is an important feature of wave breaking as the plunging jet forms and re-enters the water surface. It therefore follows that this behaviour, along with other impacts such as those on structures, could be modelled using particle based methods. However, an unfortunate disadvantage of particle based methods is that the free surface is described by a large number of discrete particles and so the exact position of the free surface is unlikely to be defined as accurately as it is in the BEM. This is exacerbated by the fact that the free surface boundary conditions are not explicitly applied within particle based methods.

Taking into account the above, there is an obvious advantage in coupling a BEM with a discrete particle method. In this way, an accurate description of the free surface up to the point of wave breaking can be simulated using the BEM. This can then be followed by a particle method to simulate wave plunging, the re-entry of the jet, and the post breaking behaviour. A coupled scheme of this form is currently under investigation within the Fluid Mechanics Section in the Department of Civil and Environmental Engineering at Imperial College London.

8.3.2 Computing techniques and hardware

A number of improvements to the EPIC_BEM code base could be made both in relation to the use of new hardware and to the introduction of improved numerical methods.

Massively parallel element integration

In Chapter 4 it was explained that computing architectures now exist that allow massively parallel fine grained operations. With this in mind, it is not difficult to see a large number of ways in which the influence matrix assembly could be designed to run on a massively parallel architecture. What is most important is not the implementation of this concept, but the optimisation of the work distribution within the implementation to achieve the most efficient use of the processing power. This idea is currently being explored in the field of finite element matrix assembly in the Department of Computing at Imperial College London (Markall, 2010). It is hoped that some of the results of this work can be shared and be equally implemented within the EPIC_BEM code.

Kinematics field

In §6.2.1 calculations of the kinematics field internal to a BEM domain were outlined. Fortunately, in the case presented, the free surface remained single valued and so deciding whether each point in the kinematics grid lay within the fluid was just a question of testing if the point was below the free surface. With regard to overturning waves, the free surface is not single valued and so deciding if a point is within the fluid becomes much more complicated. It is recommended that a technique based on knowledge of the Jordan curve theorem is applied (Hales, 2007). Alternatively, the more complicated *alpha shape* (Edelsbrunner & Mücke, 1992) of the domain could be computed to assist with this important task.

Preconditioners

In Chapter 4 a *Jacobi* preconditioner was used to accelerate the convergence rate of the system solution. Many other preconditioners exist, some of which can be tuned to the nature of the system being solved; for example, successive over-relaxation (Young, 1950). In the present context, it is recommended that an eigenvalue analysis of the system matrix is undertaken to assist in finding a more optimal preconditioner; the work of Golub & Van Loan (1996) being particularly useful for this task.

8.4 Final thoughts

Within the lifetime of the present work, computer hardware, and consequently programming techniques, have evolved rapidly. Indeed, at the conception of the present work dual core processors were just becoming available. Currently, it is common for server hardware to have two processing sockets with each processor having four or six cores with hyper-threading. Thus giving an order of magnitude of more computing power. However, this extra computing power can only be harnessed through the use of threaded programs. Furthermore, with the availability of massively parallel devices (as seen in Chapter 4) acceleration of fine grained algorithms is possible in a manner not previously available. Taking this information into account, the future of numerical modelling clearly lies in the exploitation of modern hardware through the use of parallel programming. The techniques used in the present work merely represent the tip of the parallel iceberg.

References

Abramowitz, M. & Stegun, I. A. 1964 *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, ninth dover printing, tenth gpo printing edn. New York: Dover.

Advanced Micro Devices Inc 2006 *ATI CTM Guide Technical Reference Manual*. Advanced Micro Devices Inc.

American Petroleum Institute 2000 Recommended Practices for Planning, Designing and Constructing Fixed Offshore Platforms - Working Stress Design. Report.

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A. & Sorensen, D. 1999 *LAPACK Users' Guide*, 3rd edn. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Atkins, N., Lyons, R. & Rainey, R. 1993 Summary of findings of wave load measurements on the Tern platform. Report prepared by WS Atkins for the UK Health and Safety Executive., Data presented on Table 4.1, pp15.

Bai, W. & Eatock Taylor, R. 2007 Numerical simulation of fully nonlinear regular and focused wave diffraction around a vertical cylinder using domain decomposition. *Applied Ocean Research* **29, February-April, Issues 1 - 2**, 55 – 71.

Bailey, D. H., Jeyabalan, K. & Li, X. S. 2005 A Comparison of Three High-Precision Quadrature Schemes. *Experimental Mathematics* **14,3**.

Barnes, J. & Hut, P. 1986 A hierarchical O(N log N) force-calculation algorithm. *Nature* **324**, 446 – 449.

Bateman, W., Swan, C. & Taylor, P. 2003 On the calculation of the water particle kinematics arising in a directionally spread wave field. *Journal of Computational Physics* **186**, 70 – 92.

Becker, A. A. 1992 *The Boundary Element Method in Engineering: A Complete Course*. London: McGraw-Hill Book Company.

Benjamin, T. B. & Feir, J. E. 1967 The disintegration of wave trains on deep water Part 1. Theory. *Journal of Fluid Mechanics* **27** (03), 417 – 430.

Blackford, L. S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D. & Whaley, R. C. 1997 *ScaLAPACK Users' Guide*. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Brandini, C. & Grilli, S. 2001 Modeling of Freak Wave Generation in a 3D-NWT. In *The Proceedings of The Eleventh (2001) International Offshore and Polar Engineering Conference Volume III*.

Brebbia, C. A. & Dominguez, J. 1992 *Boundary Elements: An Introductory Course*, 2nd edn. London: Computational Mechanics Publications (McGraw-Hill).

Butcher, J. C. 2003 *Numerical methods for ordinary differential equations*, 2nd edn. John Wiley and Sons.

Chaillat, S., Bonnet, M. & Jean-Franois Semblat 2008 A multi-level fast multipole BEM for 3-D elastodynamics in the frequency domain. *Computer Methods in Applied Mechanics and Engineering* **197** (49-50), 4233 – 4249.

Chapman, B., Gabriele, J. & van der Pas, R. 2007 *Using OpenMP: portable shared memory parallel programming Scientific and engineering computation Scientific Computation Series*. MIT Press.

Christou, M. 2008 Fully Nonlinear Computations of Waves and Wave-structure Interaction. PhD thesis, Imperial College London.

Christou, M., Hague, C. & Swan, C. 2009 The reflection of nonlinear irregular surface water waves. *Engineering Analysis with Boundary Elements* **33** (5), 644 – 653.

Christou, M., Swan, C. & Gudmestad, O. 2008 The interaction of surface water waves with submerged breakwaters. *Coastal Engineering* **55** (12), 945 – 958.

Christou, M., Swan, C. & Gudmestad, O. T. 2007 The Description of Breaking Waves and the Underlying Water Particle Kinematics. In *Proceedings of the 26th International Conference on Offshore Mechanics and Arctic Engineering: Volume 5*.

ClearSpeed Technology 2001-2009 ClearSpeed Technology: The CSX architecture. http://www.clearspeed.com. Online.

Cunha, M. T. F., Telles, J. C. F. & Coutinho, A. L. G. A. 2002 Parallel Boundary Elements Using Lapack and ScaLapack. In *Proceeding of the 14th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'02)*, p. 0051.

Darve, E. 2001 The Fast Multipole Method I: Error Analysis and Asymptotic Complexity. *SIAM Journal on Numerical Analysis* **38** (1), 98 – 128.

Dean, R. G. & Dalrymple, R. A. 1984 *Water Wave Mechanics for Engineers and Scientist*, *Advanced Series on Ocean Engineering*, vol. 2. World Scientific Publishing Co. Pte. Ltd.

Det Norske Veritas 2010 SESAM loading suite. On line. http://www.dnv.com/services/software/products/sesam/, visited 08/06/2010.

Dominguez, J. 1993 *Boundary Elements in Dynamics*. Computational Mechanics Publications, Elsevier Applied Science, Southampton Boston.

Drimer, N. & Agnon, Y. 2006 An improved low-order boundary element method for breaking surface waves. *Wave Motion* **43** (3), 241 – 258.

Edelsbrunner, H. & Mücke, E. P. 1992 Three-dimensional alpha shapes. In *VVS '92: Proceedings of the 1992 workshop on Volume visualization*, pp. 75 – 82. New York, NY, USA: ACM.

Fenton, J. D. 1985 A fifth-order Stokes theory for steady waves. *Journal of Waterway, Port, Coastal and Ocean Engineering* **111**, 216 – 234.

Fochesato, C. & Dias, F. 2006 A fast method for nonlinear three-dimensional free-surface waves. *Proceedings A of The Royal Society* **462** (2073), 2715 – 2735.

Fochesato, C., Grilli, S. & Dias, F. 2007 Numerical modeling of extreme rogue waves generated by directional energy focusing. *Wave Motion* **44** (5), 395 – 416.

Fochesato, C., Grilli, S. T. & Guyenne, P. 2005 Note on non-orthogonality of local curvilinear co-ordinates in a three-dimensional boundary element method. *International Journal for Numerical Methods in Fluids* **48** (3), 305 – 324.

Forristall, G. Z. 2000 Wave Crest Distributions: Observations and Second-Order Theory. *Journal of Physical Oceanography* **30** (8), 1931 – 1943.

Fujimoto, N. 2008 Dense matrix-vector multiplication on the cuda architecture. *Parallel Processing Letters* **18**, 511 – 530.

Gibson, R. & Swan, C. 2007 The evolution of large ocean waves: the role of local and rapid spectral changes. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science* **463** (2077), 21 – 48.

Golub, G. H. & Van Loan, C. 1996 *Matrix computations*, 3rd edn.

Gorman, M. 2004 *Understanding the Linux Virtual Memory Manager*. Prentice Hall. Part of the Bruce Perens' Open Source Series series.

Greengard, L. 1988 *The rapid evaluation of potential fields in particle systems*. MIT Press.

Greengard, L. & Rokhlin, V. 1987 A Fast Algorithm for Particle Simulations. *Journal of Computational Physics* **73**, 325 – 348.

Grilli, S. & Svendsen, I. 1990 Corner problems and global accuracy in the boundary element solution of nonlinear wave flows. *Engineering Analysis with Boundary Elements* **7** (4), 178 – 195.

Grilli, S. T., Guyenne, P. & Dias, F. 2001 A fully non-linear model for three-dimensional overturning waves over an arbitrary bottom. *International Journal for Numerical Methods in Fluids* **35** (7), 829 – 867.

Grilli, S. T., Skourup, J. & Svendsen, I. A. 1989 An efficient boundary element method for nonlinear water waves. *Engineering Analysis with Boundary Elements* **6** (2), 97 – 107.

Grilli, S. T. & Subramanya, R. 1996 Numerical modeling of wave breaking induced by fixed or moving boundaries. *Computational Mechanics* **17** (6), 374 – 391.

Grilli, S. T., Svendsen, I. A. & Subramanya, R. 1997 Breaking Criterion and Characteristics for Solitary Waves on Slopes. *Journal of Waterway, Port, Coastal, and Ocean Engineering* **123** (3), 102 – 112.

Gumerov, N. A. & Duraiswami, R. 2009 A broadband fast multipole accelerated boundary element method for the three dimensional Helmholtz equation. *The Journal of the Acoustical Society of America* **125** (1), 191 – 205.

Guyenne, P. & Grilli, S. T. 2006 Numerical study of three-dimensional overturning waves in shallow water. *Journal of Fluid Mecahnics* **547**, 361 – 388.

Hague, C. & Swan, C. 2009 A multiple flux boundary element method applied to the description of surface water waves. *Journal of Computational Physics* **228** (14), 5111 – 5128.

Hague, C. H. 2006 Fully Nonlinear Computations of Directional Waves, Including Wave Breaking. PhD thesis, Imperial College London.

Hales, T. C. 2007 The Jordan curve theorem, formally and informally. *Amer. Math. Monthly* **114** (10), 882 – 894.

Hamano, K., Murashige, S. & Hayami, K. 2003 Boundary element simulation of large amplitude standing waves in vessels. *Engineering Analysis with Boundary Elements* **27** (6), 565 – 574.

Harvey, M., Giupponi, G., Villa-Freixa, J. & De Fabritiis, G. 2007 *Distributed and Grid Computing - Science Made Transparent for Everyone. Principles, Applications and Supporting Communities*, chap. PS3GRID.NET: Building a distributed supercomputer using the Playstation 3.

Harvey, M. J., Giupponi, G. & Fabritiis, G. D. 2009 ACEMD: Accelerating Biomolecular Dynamics in the Microsecond Time Scale. *Journal of Chemical Theory and Computation* **5(6)**, 1632 – 1639.

Heideman, J. & Weaver, T. 1992 Static wave force procedure for platform design. In *Proceedings of the International Conference: Civil Engineering in the Oceans (V).*. ASCE.

Heideman, J. C. 2010 Personal communication between P. S. Tromans and J. C. Heideman. Email correspondence.

Hofstee, H. P. 2005 Power Efficient Processor Architecture and The Cell Processor. *High-Performance Computer Architecture, International Symposium on* **0**, 258 – 262.

InfiniBand Trade Association 2000 Infiniband Architecture Specification Volume 1 Release 1.0. Available from the InfiniBand Trade Association, http://www.infinibandta.org.

Isaacson, M. & Cheung, K.-F. 1990 Time-Domain Solution for Second-Order Wave Diffraction. *Journal of Waterway, Port, Coastal, and Ocean Engineering* **116** (2), 191 – 210.

Jensen, J. J. 2004 Conditional short-crested second waves in shallow water and with superimposed current. In *Proc. OMAE 2004*. Vancouver, BC, Canada.

Johannessen, T. & Swan, C. 2001 A laboratory study of the focusing of transient and directionally spread surface water waves. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* **457** (2008), 971 – 1006.

Johannessen, T. B. & Swan, C. 2003 On the nonlinear dynamics of wave groups produced by the focusing of surface-water waves. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* **459** (2032), 1021 – 1052.

Johansen, A. & Nestegård, A. 2008 ConocoPhillips Greater Ekofisk area jacket load comparison, stokes 5th order regular wave and fully non-linear random wave kinematics. *Tech. Rep.* Report No. 2008-1264. Revision No. 01. Det Norske Veritas.

Jonathan, P. & Taylor, P. 1995 Vertical asymmetry of ocean waves: Measurements of waves at Tern. External Shell Report, number RKER.95.066.

Jonathan, P. & Taylor, P. H. 1997 On Irregular, Nonlinear Waves in a Spread Sea. *Journal of Offshore Mechanics and Arctic Engineering* **119** (1), 37 – 41.

Joubert, W. 1994 On the convergence behavior of the restarted GMRES algorithm for solving nonsymmetric linear systems. *Numerical Linear Algebra with Applications* **1**, 427 – 447.

Kharif, C. & Pelinovsky, E. 2003 Physical mechanisms of the rogue wave phenomenon. *European Journal of Mechanics - B/Fluids* **22**, 603 – 634.

Khronos OpenCL Working Group 2009 The OpenCL Specification. Online: http://www.khronos.org/registry/cl/specs/opencl-1.0.43.pdf.

Kreienmeyer, M. & Stein, E. 1995 Parallel implementation of the boundary element method for linear elastic problems on a MIMD parallel computer. *Computational Mechanics* **15**, 342 – 349.

Kybic, J., Clerc, M., Faugeras, O., Keriven, R. & Papadopoulo, T. 2005 Fast multipole acceleration of the MEG/EEG boundary element method. *Physics in Medicine and Biology* **50** (19), 4695 – 4710.

Lawson, C. L., Hanson, R. J., Kincaid, D. R. & Krogh, F. T. 1979 Basic Linear Algebra Subprograms for Fortran Usage. *ACM Trans. Math. Softw.* **5** (3), 308 – 323.

Lawton, G. 2001 Monsters of the deep (the perfect wave). New Scientist, Issue 2297, pp28 - 32.

Liu, Y., Xue, M. & Yue, D. K. P. 2001 Computations of fully nonlinear three-dimensional wave-wave and wave-body interactions. Part 2. Nonlinear wave and forces on a body. *Journal of Fluid Mechanics* **438**, 41 – 66.

Longuet-Higgins, M. S. 1963 The effect of non-linearities on statistical distributions in the theory of sea waves. *Journal of Fluid Mechanics* **17** (03), 459 – 480.

Longuet-Higgins, M. S. & Cokelet, E. D. 1976 The Deformation of Steep Surface Waves on Water. I. A Numerical Method of Computation **350** (1660), 1 – 26.

Longuet-Higgins, M. S. & Stewart, R. W. 1960 Changes in the form of short gravity waves. *Journal of Fluid Mechanics* **8**, 565 – 583.

Markall, G. 2010 Making Faster FEM Solvers, Faster. MPhil Transfer Report. Internal Publication of Imperial College London.

Mitsuyasu, H. 1975 Observations of directional spectrum of ocean waves using a cloverleaf buoy. *Journal of Physical Oceanography* **16**, 750 – 760.

Moore, G. E. 1965 Cramming more components onto integrated circuits. Electronics Magazine.

Morison, J. R., O'Brien, M. P., Johnson, J. W. & Schaaf, S. A. 1950 The forces exerted by surface waves on piles. *Journal of Petroleum Technology, AIME* **189**, 149 – 157.

Natarajan, R. & Krishnaswamy, D. 1995 A Case Study in Parallel Scientific Computing: The Boundary Element Method on a Distributed-Memory Multicomputer. In *Supercomputing, 1995. Proceedings of the IEEE/ACM SC95 Conference*, pp. 33 – 33.

New, A. L., McIver, P. & Peregrine, D. H. 1985 Computations of Overturning Waves. *Journal of Fluid Mechanics* **150**, 233 – 251.

NVIDIA 2008*a* *CUDA CUBLAS Library*, 2nd edn. NVIDIA PLC.

NVIDIA 2008*b* *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*, 2nd edn.

NVIDIA 2009 http://www.nvidia.com/object/cuda_home.html. On line.

Ortiz, J. & Douglass, S. 1993 *Boundary Elements XV Vol 1 Fluid Flow and Computational Aspects*, chap. Boundary element solution of water particle velocities of waves breaking on mild slopes., pp. 221 – 232. Worcester Polytechnic Institute, Worcester, MA.: Worcester Polytechnic Institute.

Peric, M. 2010 Nonlinear wave interactions of multiple bodies in close proximity. (Unpublished PhD thesis).

Pfister, G. F. 1998 *In search of clusters*. Upper Saddle River, NJ : Prentice Hall PTR.

Press, W., Teukolsky, S., Vetterling, W. & Flannery, B. 1990 *Numerical Recipes in C*. Cambridge, UK: Cambridge University Press.

Saad, Y. & Schultz, M. H. 1986 GMRES: A generalized minimum residual algorithm for solving nonsymmetric linear systems. *SIAM Journal of Statistical and Scientific Computing* **7** (3), 856 – 869.

Saad, Y. & Vorst, H. A. V. D. 2000 Iterative Solution of Linear Systems in the 20th Century. *Journal of Computational and Applied Mathematics* **123**, 1 – 33.

SEILER, L., CARMEAN, D., SPRANGLE, E., FORSYTH, T., ABRASH, M., DUBEY, P., JUNKINS, S., LAKE, A., SUGERMAN, J., CAVIN, R., ESPASA, R., GROCHOWSKI, E., JUAN, T. & HANRAHAN, P. 2008 Larrabee: a many-core x86 architecture for visual computing. *ACM Trans. Graph.* **27** (3), 1 – 15.

SHARMA, J. N. & DEAN, R. G. 1981 Second-Order Directional Seas and Associated Wave Forces. *Society of Petroleum Engineering Journal* **4**, 129 – 140.

SOBEY, R. J. 2006 Wave kinematic fields from the boundary integral method. *International Journal for Numerical Methods in Fluids* **51**, 773 – 790.

SOBEY, R. J., GOODWIN, P., THIEKE, R. & WESTBERG, R. 1987 Application of Stokes, cnoidal and Fourier wave theories. *Journal of Waterway, Port, Coastal and Ocean Engineering , ASCE* **113**, 565 – 580.

SOMMERFELD, A. 1949 *Partial Differential Equations in Physics*. New York: Academic Press.

SONNEVELD, P. & VAN GIJZEN, M. B. 2008 IDR(s): A Family of Simple and Fast Algorithms for Solving Large Nonsymmetric Systems of Linear Equations. *SIAM Journal on Scientific Computing* **31** (2), 1035 – 1062.

STOKES, G. G. 1847 On the theory of oscillatory waves. *Transcripts of the Cambridge Philosophical Society* **8**, 441 – 455.

STRATING, P. & DE HAAS, P. C. A. 1997 *High-Performance Computing and Networking*. London, UK: Springer-Verlag.

SWAN, C. 2007 Calculation of the Nonlinear Water Particle Kinematics at the Ekofisk Field - Stage 1. *Tech. Rep.*. Imperial College London.

TREFETHEN, L. N. & BAU, D. 1996 *Numerical linear algebra*. SIAM.

TREVELYAN, J. 1994 *Boundary Elements for Engineers: Theory and Application*. Computational Mechanics Publications.

TROMANS, P. S., ANATURK, A. & HAGEMEIJER, P. 1991 A new model for the kinematics of large ocean wavesapplication as a design wave. In *Proc. 1st Offshore and Polar Engineering Conf. (ISOPE)*, , vol. 3, pp. 64 – 71. Edinburgh.

VAN GIJZEN, M. B. & SONNEVELD, P. 2008 An elegant idr(s) variant that efficiently exploits bi-orthogonality properties. *Tech. Rep.*. Reports of the Department of Applied Mathematical Analysis, Delft University of Technology.

VARDON, P., BANICESCU, I., CLEALL, P., THOMAS, H. & PHILP, R. 2009 Coupled thermo-hydro-mechanical modelling: A new parallel approach. *Parallel and Distributed Processing Symposium, International* **0**, 1 – 9.

VOLKOV, V. & DEMMEL, J. W. 2008 Benchmarking GPUs to tune dense linear algebra. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pp. 1 – 11. Piscataway, NJ, USA: IEEE Press.

VAN DER VORST, H. A. 1992 Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing* **13** (2), 631 – 644.

WESSELING, P. & SONNEVELD, P. 1980 *Approximation Methods for Navier-Stokes Problems*, , vol. Volume 771/1980. Springer Berlin / Heidelberg.

WHEELER, J. D. 1970 Method for calculating forces produced by irregular waves. *Journal of Petroleum Technology* **249**, 359 – 367.

XUE, M., XU, H., LIU, Y. & YUE, D. 2001 Computations of fully nonlinear three-dimensional wave-wave and wave-body interactions. Part 1. Dynamics of steep three-dimensional waves. *Journal of Fluid Mechanics* **438**, 11 – 39.

YAN, H., LIU, Y. & YUE, D. K. P. 2006 An efficient computational method for nonlinear three-dimensional wave-wave and wave-body interactions. *Journal of Hydrodynamics, Series. B* **, Volume 18, Issue 3, Supplement 1, Proceedings of the Conference of Global Chinese Scholars on Hydrodynamics** (Issue 3, Supplement 1, Proceedings of the Conference of Global Chinese Scholars on Hydrodynamics), 84 – 88.

Yan, S. & Ma, Q. W. 2010 QALE-FEM for modelling 3D overturning waves. *International Journal for Numerical Methods in Fluids* **63** (6), 743 – 768.

Young, D. M. 1950 Iterative methods for solving partial difference equations of elliptical type. PhD thesis, Harvard University.

Appendix A

IDR(s) theory

This appendix outlines the IDR theorem of Sonneveld & van Gijzen (2008). For completeness and easy of reading, some parts of §4.3.2 are repeated in the sections that follow. Additional information on some of the concepts involved in the IDR(s) theory can also be found in Appendix B.

A.1 IDR theorem

The basic concept of the IDR(s) method is that it generates residuals that are forced to be in subspaces of $\mathcal{G}_j$ of decreasing dimension. These subspaces are nested and are related as given in equation (A.1). An indication as to how this relation arises is given in Appendix B.1. The development of the IDR theorem, presented in the following sections, follows the arguments originally outlined by Sonneveld & van Gijzen (2008). In the present case the methodology is applied to the solution of an equation set defined by $Ax = b$.

Let $\mathbf{A}$ be any matrix in $\mathbb{C}^{N\times N}$, let v_0 be any non-zero vector in $\mathbb{C}^N$, and let $\mathcal{G}_0$ be the full Krylov space $\mathcal{K}^N(\mathbf{A}, v_0)$. Let $\mathcal{S}$ denote any (proper) subspace of $\mathbb{C}^N$ such that $\mathcal{S}$ and $\mathcal{G}_0$ do not share a nontrivial invariant subspace of $\mathbf{A}$, and define the sequence $\mathcal{G}_j, j = 1, 2, ...,$ as

$$\mathcal{G}_j = (I - w_j\mathbf{A})(\mathcal{S} \cap \mathcal{G}_{j-1}). \tag{A.1}$$

Within this solution, I is the identity matrix, ω_j are weightings and $\mathcal{S}$ is a fixed proper subspace of $\mathbb{C}^N$.

A Krylov based solver produces iterations x_n for which the residuals $r_n = b - Ax_n$ are in the Krylov spaces $\mathcal{K}^n(A, r_0)$; with x_0 being an initial estimate of the solution. Given a recursion for the residuals r_n, it must be possible to produce a corresponding recursion for x_n. As a result, the residuals, r_n, can be written as $\Phi_n(A)r_0$, where Φ_n is an n^{th} degree polynomial: $\Phi_n \in \mathbb{P}^n \setminus \mathbb{P}^{n-1}$ ($\setminus$ is the set difference operator and $\mathbb{P}$ polynomial space). Assuming this has to be possible for the residuals up to the n^{th} step; it must then be possible to calculate x_{n+1} from the equation

$$A\Delta x_n = -\Delta r_n = [\Phi_n(A) - \Phi_{n+1}(A)]r_0, \tag{A.2}$$

without actually solving an equation with the matrix A. In formulating this solution, the forward difference operator $\Delta u_k = u_{k+1} - u_k$ is used. Additional information surrounding the nature of this polynomial can be found in Appendix B.2.

Adopting this approach, a general Krylov-type solver can be described by recursions of the following form:

$$r_{n+1} = r_n - \alpha A v_n - \sum_{l=1}^{\hat{l}} \gamma_l \Delta r_{n-l}, \tag{A.3}$$

$$x_{n+1} = x_n + \alpha v_n - \sum_{l=1}^{\hat{l}} \gamma_l \Delta r_{n-l}, \tag{A.4}$$

where v_n is any computable vector in $\mathcal{K}^n(A, r_0) \setminus \mathcal{K}^{n-1}(A, r_0)$, the integer $\hat{l}$ is the depth of recursion, α is a weighting parameter and γ_l are unknown coefficients. Additional information relating to the form of the generic Krylov recursions is given in Appendix B.1. If $\hat{l} = n$ then the method has a so-called *long recurrence* implying that the amount of work and memory grow with n. On the other hand, if $\hat{l}$ is fixed and small (compared to N), then the method has a so-called *short recurrence*, which is what makes it attractive with respect to computational effort and the limited availability of memory on a GPU.

The IDR theorem can be applied by generating residuals, r_n, that are forced to be in subspaces $\mathcal{G}_j$, where j is non-decreasing with increasing n. Under the assumptions of equation (A.1) the system will be solved after, at most, N dimensional reduction steps.

The residual r_{n+1} lies in $\mathcal{G}_{j+1}$ if

$$r_{n+1} = (I - w_{j+1}A)v_n \quad \text{with} \quad v_n \in \mathcal{G}_j \cap \mathcal{S}. \tag{A.5}$$

If v_n is chosen as

$$v_n = r_n - \sum_{l=1}^{\hat{l}} \gamma_l \Delta r_{n-l}, \tag{A.6}$$

the expression for r_{n+1} becomes

$$r_{n+1} = r_n - \omega_{j+1}Av_n - \sum_{l=1}^{\hat{l}} \gamma_l \Delta r_{n-l}, \tag{A.7}$$

which has been seen before as the general Krylov-solver recursion in equation (A.3).

Without loss of generality, it can be assumed that the space, $\mathcal{S}$, is the left nullspace of some $N \times s$ matrix P:

$$P = (p_1, p_2, ..., p_s), \quad \mathcal{S} = \mathcal{N}(P^H), \tag{A.8}$$

where the superscript H indicates the Hermitian transpose (transpose with conjugate negation). Since v_n is also in $\mathcal{S} = \mathcal{N}(P^H)$, it additionally satisfies

$$P^H v_n = 0. \tag{A.9}$$

Combining equations (A.6) and (A.9) yields an $s \times \hat{l}$ linear system for the $\hat{l}$ coefficients γ. Under normal circumstances this system is uniquely solvable if $\hat{l} = s$. Consequently, computing the first vector in $\mathcal{G}_{j+1}$ requires $s+1$ vectors in $\mathcal{G}_j$, and r_n can be expected to be in $\mathcal{G}_{j+1}$ if and only if $n \geq (j+1)(s+1)$.

Defining the matrices

$$\Delta\mathbf{R}_n = (\Delta r_{n-1}, \Delta r_{n-2}, \cdots, \Delta r_{n-s}), \tag{A.10}$$

$$\Delta\mathbf{X}_n = (\Delta x_{n-1}, \Delta x_{n-2}, \cdots, \Delta x_{n-s}), \tag{A.11}$$

the computation of $r_{n+1} \in \mathcal{G}_{j+1}$ can be implemented by the following algorithm,

$$\text{Calculate} : c \in \mathbb{C}^s \text{ from } (P^H \Delta\mathbf{R}_n)c = P^H r_n, \tag{A.12}$$

$$v = r_n - \Delta\mathbf{R}_n c, \tag{A.13}$$

$$r_{n+1} = v - \omega_{j+1}Av. \tag{A.14}$$

Since $\mathcal{G}_{j+1} \subset \mathcal{G}_j$, repeating these calculations will produce new residuals r_{n+2}, r_{n+3}, $\ldots$ in $\mathcal{G}_{j+1}$. Once $s+1$ residuals in $\mathcal{G}_{j+1}$ have been computed, the next residual can be expected to be in $\mathcal{G}_{j+2}$. In the calculation of the first residual in $\mathcal{G}_{j+1}$, ω_{j+1} can be chosen freely but the same value must be used in the calculations for subsequent residuals in $\mathcal{G}_{j+1}$. A suitable choice for ω_{j+1} is the value that minimises the 2-norm of r_{n+1}; a similar approach having been adopted in the Bi-CGSTAB algorithm (van der Vorst, 1992).

Following some detailed experimentation conerning the optimal values for matrix P consistent with maximising the rate of convergence, Sonneveld & van Gijzen (2008) chose the matrix P to be an orthogonal matrix of random vectors as this seemed to work well in a wide range of cases.

A.2 Explanation of the IDR(s) method

Having outlined the IDR theorem, some further explanation is required regarding the algorithmic implementation of each part of the theorem. To begin, a number of concepts are discussed that are key to understanding the implementation of the IDR(s) method given in section A.3.

Application of s minimum norm steps

To begin the IDR(s) method, the columns of the matrices $\Delta\mathbf{R}_n$ and $\Delta\mathbf{X}_n$ must be populated, respectively, with the differences found between subsequent iterations of the residual and solution vectors ($\Delta\mathbf{r}_n$ and $\Delta\mathbf{x}_n$). To assemble these vectors, s minimum norm steps are used to build sufficient vectors in $\mathcal{G}_0$ such that if the IDR theorem is applied, the next residual vector will be in $\mathcal{G}_1$; the process of subspace nesting having been initialised. Clearly, vectors generated in the s minimum norm steps must also be members of the Krylov subspace, $\mathcal{K}^n(A, r_0)$.

Minimising the residual norm

The value of ω_{j+1} is chosen such that the 2-norm of the residual r_{n+1} is minimised and is given by

$$\omega_{j+1} = \frac{\mathbf{v}^H \mathbf{r}_n}{\mathbf{v}^H \mathbf{v}}, \tag{A.15}$$

where $\mathbf{v} = \mathbf{A}\mathbf{r}_n$. The derivation of this weighting is rather elusive. However, by noting that it takes a form similar to that of the Rayleigh quotient (Trefethen & Bau, 1996), a similar path to its derivation can be pursued. First, by expressing the 2-norm of the residual in terms of the previous iteration's residual, the system matrix and the unknown value ω_{j+1},

$$||\mathbf{r}_{n+1}||_2 = ||\mathbf{r}_n - \omega_{j+1}\mathbf{A}\mathbf{r}_n||_2, \tag{A.16}$$

it can be seen that this is simply a $[n \times 1]$ least squares problem of the form,

$$\mathbf{r}_n \approx \omega_{j+1}\mathbf{A}\mathbf{r}_n. \tag{A.17}$$

Using the substitution $\mathbf{t} = \mathbf{A}\mathbf{r}_n$ the true nature of the system can be seen,

$$\underset{[N\times 1]}{\mathbf{r}_n} \approx \underset{[1\times 1][N\times 1]}{\omega_{j+1}\mathbf{t}} \tag{A.18}$$

where the content of the subscripted brackets indicates the dimension of the variable directly above. Clearly equation (A.18) is over determined and so a least squares approximation is required to find an appropriate solution for ω_{j+1}. To obtain this solution, the normal equations can be applied with ω_{j+1} as the free variable,

$$\mathbf{t}^H \mathbf{r}_n = \omega_{j+1}\mathbf{t}^H \mathbf{t}, \tag{A.19}$$

which can be simply rearranged to yield ω_{j+1},

$$\omega_{j+1} = \frac{\mathbf{t}^H \mathbf{r}_n}{\mathbf{t}^H \mathbf{t}}. \tag{A.20}$$

Computing v_n

From the IDR theorem, $v_n \in \mathcal{G}_j \cap S$, where S is a subspace that is effectively free to be chosen for performance or convenience subject to the condition that S only shares trivial invariant subspaces[1] of A (such as $\{\mathbf{0}\}$). As v_n was previously defined as being composed of known values relating to r_n, all multiplied by unknowns (γ_l), an intelligent choice of subspace S would be one that allows the solution of the system in which the unknowns reside such that the next iterates can be computed. This is accomplished by assuming that S is the left nullspace of some $N \times s$ matrix P,

$$P = (p_1, p_2, \cdots, p_s), \quad S = \mathcal{N}(P^H). \tag{A.21}$$

By definition, the left nullspace of matrix P consists of all vectors m such that $m^H P = 0^H$, or equally, $P^H m = 0$. Since v_n is in S and $S = \mathcal{N}(P^H)$ it is clear that,

$$P^H v_n = 0. \tag{A.22}$$

As equation (A.22) gives a relation between v_n and some fixed value (in this case the zero vector), it is now possible to build a method to compute the unknowns (γ_l) present in v_n.

If equation (A.6) is substituted into equation (A.22) and rearranged the following is found,

$$\underset{[s\times N][N]}{P^H r_n} = \underset{[s\times N][N\times\hat{l}]}{P^H \sum_{l=1}^{\hat{l}} \gamma_l \Delta r_{n-l}}, \tag{A.23}$$

where the square brackets beneath each component again indicate its size. From this system it is evident that the dimension $\hat{l}$ must be s to create a uniquely solvable system, hence the choice of s sets the level of recursion. From this assertion it can be deduced that, to enter subspace $\mathcal{G}_{j+1}$ from $\mathcal{G}_j$, $s+1$ residual vectors are needed. In other words, s residual vectors are required to build the $[s \times s]$ system in the current subspace so that, upon application of its solution, the next residual vector computed will be in the next subspace.

[1] A subspace M is invariant under the operator A if $Ax \in M$ for every $x \in M$.

A.3 The IDR(s) algorithm

Algorithm (1) is taken from Sonneveld & van Gijzen (2008) and is presented with the view of aiding an explanation of the method. The IDR(s) method has been derived as a direct method as noted in §4.3.2. It can be shown that in infinite precision arithmetic the solution will be found in $2N$ iterations (Sonneveld & van Gijzen, 2008). Unfortunately, computers can only employ finite precision arithmetic and therefore the method must be programmed as an iterative scheme such that the solution to the system is considered satisfactory only if it meets some specified criteria based upon the properties of the residual, r_n.

With regards to Algorithm (1), the **Require** statement, as expected, dictates the space in which each variable must exist. All the variables have been defined previously with the exception of TOL and $MAXIT$. In relation to the previously explained issue around finite precision arithmetic, the variable TOL holds a value that is used to determine if the accuracy of the solution is acceptable (TOL is usually set to the limit of near arithmetic precision for a given data type, typically 1×10^{-6} for a 32 bit float). The variable $MAXIT$ is simply the maximum number of iterations deemed acceptable in the search for a solution. In infinite precision arithmetic this value would be $2N$. However, in finite precision arithmetic $4N$ is sufficient in practice. The following **Ensure** statement explicitly expresses that the 2-norm of the residual must be less than the defined tolerance, TOL, for the algorithm to be successful. This feature is regularly incorporated in Krylov subspace based methods for solving linear systems. The excellent reference text by Golub & Van Loan (1996) provides further discussion on this and several related issues.

As explained in Appendix A.2, to initialise the scheme enough residual vectors must be formed in $\mathcal{G}_0$ to allow the next residual to reside in the $\mathcal{G}_1$ subspace. This operation is carried out in lines `4-9`. Evidently, this initialisation is very similar to the "Modified Richardson Iteration" (see Appendix B, equation (B.6)) with ω chosen to minimise the 2-norm of the $(n+1)^{th}$ residual. After each of the s initialisation steps, the arrays $\Delta\mathbf{R}_{n+1}$ and $\Delta\mathbf{X}_{n+1}$ can have the respective residual and solution differences ($\Delta\mathbf{r}_n$ and $\Delta\mathbf{x}_n$) inserted into their n^{th} columns (assuming

Algorithm 1 The IDR(s) algorithm. Taken verbatim from Sonneveld & van Gijzen (2008).

```
Require: $\mathbf{A} \in \mathbb{C}^{N\times N}$; $\mathbf{x}_0, \mathbf{b} \in \mathbb{C}^N$;$\mathbf{P} \in \mathbb{C}^{N\times s}$;$TOL \in (0,1)$; $MAXIT > 0$
Ensure: $x_n$ such that $||\mathbf{b} - \mathbf{A}x_n|| < TOL$
 1: $\{$Initialisation$\}$
 2: Calculate $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$
 3: $\{$Apply $s$ minimum norm steps, to build enough vectors in $\mathcal{G}_0\}$
 4: for $n = 0$ to $s - 1$ do
 5:   $\mathbf{v} = \mathbf{A}r_n; \omega = (v^H r_n)/(v^H v)$;
 6:   $\Delta\mathbf{x}_n = \omega\mathbf{r}_n; \Delta\mathbf{r}_n = -\omega\mathbf{v}$
 7:   $\mathbf{r}_{n+1} = \mathbf{r}_n + \Delta\mathbf{r}_n; \mathbf{x}_{n+1} = \mathbf{x}_n + \Delta\mathbf{x}_n$;
 8: end for
 9: $\Delta\mathbf{R}_{n+1} = (\Delta\mathbf{r}_n \ldots \Delta\mathbf{r}_0); \Delta\mathbf{X}_{n+1} = (\Delta\mathbf{x}_n \ldots \Delta\mathbf{x}_0)$;
10:
11: $\{$Building $\mathcal{G}_j$ spaces for $j = 1, 2, 3, \ldots\}$
12: $n = s$
13: $\{$Loop over $\mathcal{G}_j$ spaces$\}$
14: while $||\mathbf{r}_n|| > TOL$ and $n < MAXIT$ do
15:   $\{$Loop inside $\mathcal{G}_j$ space$\}$
16:   for $k = 0$ to $s$ do
17:     Solve $\mathbf{c}$ from $\mathbf{P}^H\Delta\mathbf{R}_n\mathbf{c} = \mathbf{P}^H\mathbf{r}_n$
18:     $\mathbf{v} = \mathbf{r}_n - \Delta\mathbf{R}_n\mathbf{c}$;
19:     if $k = 0$ then
20:       $\{$Entering $\mathcal{G}_{j+1}\}$
21:       $\mathbf{t} = \mathbf{A}\mathbf{v}$;
22:       $\omega = (\mathbf{t}^H\mathbf{v})/(\mathbf{t}^H\mathbf{t})$;
23:       $\Delta\mathbf{r}_n = -\Delta\mathbf{R}_n\mathbf{c} - \omega\mathbf{t}$;
24:       $\Delta\mathbf{x}_n = -\Delta\mathbf{X}_n\mathbf{c} + \omega\mathbf{v}$;
25:     else
26:       $\{$Subsequent vectors in $\mathcal{G}_{j+1}\}$
27:       $\Delta\mathbf{x}_n = -\Delta\mathbf{X}_n\mathbf{c} + \omega\mathbf{v}$;
28:       $\Delta\mathbf{r}_n = -\mathbf{A}\Delta\mathbf{x}_n$;
29:     end if
30:     $\mathbf{r}_{n+1} = \mathbf{r}_n + \Delta\mathbf{r}_n$
31:     $\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta\mathbf{x}_n$
32:     $n = n + 1$
33:     $\Delta\mathbf{R}_n = (\Delta\mathbf{r}_{n-1} \ldots \Delta\mathbf{r}_{n-s})$;
34:     $\Delta\mathbf{X}_n = (\Delta\mathbf{x}_{n-1} \ldots \Delta\mathbf{x}_{n-s})$;
35:   end for
36: end while
```

zero-based indexing).

Once the method is initialised using s iterations, the iteration index variable, n, is set to s as the residual and solution indexes are zero-based (line `12`). Next, a **while**-loop is formed in which all further computation takes place (lines `14-36`). The conditions for exiting the while loop are instances when the 2-norm of the residual is less than TOL or the number of iterations, n, equals or exceeds the maximum number of iterations $MAXIT$. A **for** loop is then created that loops inside the $\mathcal{G}_j$ subspaces (lines `16-35`). The purpose of this loop is to generate the residual and solution differences needed to update the residual and the solution and also to allow the creation of new information in $\Delta\mathbf{R}_n$ and $\Delta\mathbf{X}_n$ which are critical for advancing the solution to the next subspace.

The loop iterates using k as its induction variable from 0 to s, obviously this is one iteration greater than the level of recurrence. The reason for this extra iteration is that s residuals are required in the current subspace to allow the computation of a new residual which will be in the next subspace (see the paragraph proceeding equation (A.9)). It is therefore clear that a branch in execution is needed to create a new vector in the $\mathcal{G}_{j+1}$ subspace, this occurs when $k = 0$ (lines `19-25`). Furthermore, the proceeding **else** condition (lines `25-29`) catches all other indices and is used to assemble s residual and solution differences in the $\mathcal{G}_{j+1}$ subspace to allow the solution to advance to the next subspace when k loops back to zero.

Having outlined the logic behind implementing the IDR(s) method, the manner in which the mathematics is implemented can be discussed. Each time the **for** loop executes a new residual difference and a new solution difference are computed, these are inserted into ΔR_n and ΔX_n respectively such that these arrays only contain the s most recently generated differences. As this update takes place in every loop, the small $[s\times s]$ system (see equation A.23 and proceeding text) changes with every loop. Therefore, the first action of the loop is to compute the vector $\mathbf{c}$ which contains the solution for the unknowns γ_l (equation (A.12)). Following this, the vector $\mathbf{v}$, corresponding to v_n in equation (A.13), can be computed. Both this and the previously mentioned computations can be seen on lines (`17-18`). In practice, the small $s \times s$ system is solved using the classic LU decomposition by

Press *et al.* (1990), which was modified by the author to use zero based indexing that is common to the `C` language.

Once v and c are computed, all the information needed to compute $\mathbf{r}_{n+1}$ (equation (A.14)), and consequently $\mathbf{x}_{n+1}$, is available with the exception of the weighting parameter ω_{j+1}. The variable ω_{j+1} is effectively used to accelerate the convergence of the iterations and is chosen to minimise the 2-norm of the ${r_{n+1}}^{th}$ residual as explained previously in Appendix A.2. The value of ω_{j+1} is only updated when $k = 0$ such that it relates solely to the first residual vector computed in a given subspace (line `22`). Clearly the minimisation of a 2-norm is, computationally, relatively cheap, as it is just inner products of length N vectors, such that the ${r_{n+1}}^{th}$ residual could be minimised for each vector in a given subspace. However, in practice this approach leads to severe cancellations, as noted by van der Vorst (1992), and experimentation by the author confirms this to be the case.

To update the residual and solution vectors the corresponding residual and solution difference vectors are used (lines `30-31`). As noted previously, there is considerable freedom available in the choice of formation of the s residuals required to compute a residual in the $\mathcal{G}_{j+1}$ subspace. It would be advantageous if the method of forming the new residuals was computationally cheap and indeed this is possible through the use of relations between r_n and x_n. In the $k = 0$ execution branch, as the $\mathcal{G}_{j+1}$ subspace is entered, the new value for ω_{j+1} is computed and then the residual and solution differences are computed explicitly using information from the previous subspace (lines `23-24`). For all other values of k the residual and solution difference vectors are computed in the $\mathcal{G}_{j+1}$ subspace. Therefore, all that is required is that the residuals are members of the $\mathcal{G}_{j+1}$ subspace and a cheap method of computing this is through a relationship between the residual difference and the solution difference (lines `27-28`).

Finally, lines `32-34` simply perform housekeeping tasks by incrementing the subscript value (which effectively causes different columns of arrays to be addressed), and inserts the residual and solution difference vectors to the correct positions in their respective storage matrices.

Appendix B

Further information relating to IDR(s)

This appendix contains additional information and explanation regarding the IDR(s) method presented in Chapter 4.

B.1 An indication of how the general Krylov recursion is formed

For solving the system $\mathbf{Ax} = \mathbf{b}$, a number of simple methods can be derived from a technique based on matrix splitting. If the system is scaled such that the diagonal components of matrix $\mathbf{A}$ are set equal to unity ($a_{ii} = 1$), then $\mathbf{A}$ can be decomposed as,

$$\mathbf{A} = -\mathbf{L} + \mathbf{I} - \mathbf{U}, \tag{B.1}$$

where $\mathbf{L}$ and $\mathbf{U}$ are respectively the strictly lower and upper triangular parts of $\mathbf{A}$ and $\mathbf{I}$ is the identity matrix. Substituting this identity into $\mathbf{Ax} = \mathbf{b}$ and rearranging,

$$\mathbf{x} = (\mathbf{L} + \mathbf{U})\mathbf{x} + \mathbf{b}. \tag{B.2}$$

If $\mathbf{x}$ is known then this equation holds true. However, in the usual case $\mathbf{x}$ is unknown and so equation (B.2) can have subscripts added for the purposes of creating an iterative scheme in the variable $\mathbf{x}$,

$$\mathbf{x}_{n+1} = (\mathbf{L} + \mathbf{U})\mathbf{x}_n + \mathbf{b}, \tag{B.3}$$

where subscripts indicate the iteration number and $n \in \mathbb{Z}^+$, where $\mathbb{Z}^+$ defines the set of positive integers. If the residual, $\mathbf{r}_n$, of the iterate $\mathbf{x}_n$ is given by,

$$\mathbf{r}_n = \mathbf{b} - \mathbf{A}\mathbf{x}_n, \tag{B.4}$$

then equation (B.3) can be expressed as,

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \mathbf{r}_n, \tag{B.5}$$

by recalling that $(\mathbf{I} - \mathbf{A}) = (\mathbf{L} + \mathbf{U})$.

The convergence of such an iterative scheme can often be improved by multiplying the residual by a weight, ω. In the present case such a modification leads to the well know method, the “modified Richardson iteration”,

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \omega\mathbf{r}_n. \tag{B.6}$$

A similar expression can be obtained for the residual by using the $(n+1)^{th}$ term of equation (B.4),

$$\mathbf{r}_{n+1} = \mathbf{b} - \mathbf{A}\mathbf{x}_{n+1}. \tag{B.7}$$

In the interests of forming expressions similar to the general IDR relation (equation (4.4)), substitution of $\mathbf{x}_{n+1}$ from equation (B.6) into the above equation yields,

$$\mathbf{r}_{n+1} = \mathbf{b} - \mathbf{A}(\mathbf{x}_n + \omega\mathbf{r}_n), \tag{B.8}$$

and further substituting the n^{th} residual, $\mathbf{r}_n$, from equation (B.4) yields,

$$\mathbf{r}_{n+1} = \mathbf{r}_n - \omega\mathbf{A}\mathbf{r}_n. \tag{B.9}$$

This expression can be rearranged as,

$$\mathbf{r}_{n+1} = (\mathbf{I} - \omega\mathbf{A})\mathbf{r}_n, \tag{B.10}$$

which has a very similar form to that of equation (4.4). Equally, if equation (B.6) is combined with equation (B.4) and rearranged,

$$\mathbf{x}_{n+1} = (\mathbf{I} - \omega\mathbf{A})\mathbf{x}_n + \omega\mathbf{b}, \tag{B.11}$$

the relation between solution iterates can also be seen to take a very similar form to that of equation (4.4).

B.2 The nature of the polynomial Φ_n

It was noted in §4.3.2 and Appendix A.1 that given a recursion for the residuals r_n it must be possible to produce a corresponding recursion for x_n. This is clearly a result of the relation $r_n = b - Ax_n$. As a result the residuals, r_n, can be written as $\Phi_n(A)r_0$, where Φ_n is an n^{th} degree polynomial: $\Phi_n \in \mathbb{P}^n \setminus \mathbb{P}^{n-1}$ ($\setminus$ is the set difference operator and $\mathbb{P}$ indicates the set of polynomials). This can easily be shown to be true through an example using iterations of equation (B.10),

$$\begin{aligned}
\mathbf{r}_1 &= (\mathbf{I} - \omega\mathbf{A})\mathbf{r}_0 && \text{(B.12)}\\
\mathbf{r}_2 &= (\mathbf{I} - \omega\mathbf{A})\mathbf{r}_1 = (\mathbf{I} - \omega\mathbf{A})(\mathbf{I} - \omega\mathbf{A})\mathbf{r}_0 && \text{(B.13)}\\
&\vdots \\
\mathbf{r}_n &= (\mathbf{I} - \omega\mathbf{A})^n\mathbf{r}_0. && \text{(B.14)}
\end{aligned}$$

Assuming this has to be possible for the residuals up to the n^{th} step, it must then be possible to calculate x_{n+1} from the equation

$$A\Delta x_n = -\Delta r_n = [\Phi_n(A) - \Phi_{n+1}(A)]r_0, \tag{B.15}$$

without actually solving an equation with the matrix A. Here the forward difference operator $\Delta u_k = u_{k+1} - u_k$ is used. A simple expansion of terms including the forward difference operator show that this is correct. It also highlights the reliance of the IDR(s) method on the computation of successive residual and solution differences.

Expressing the residual in this way is always possible if the polynomial difference $\Phi_{n+1}(\tau) - \Phi_n(\tau)$ is divisible by τ, i.e. $\Phi_{n+1}(\tau) = \Phi_n(\tau) + \tau\Psi_n(\tau)$, with $\Psi_n \in \mathbb{P}^n \setminus \mathbb{P}^{n-1}$. Additionally, with $\Phi_0 \equiv 1$, this implies that $\Phi_n(0) = 1 \; \forall n \geq 0$, which is clearly necessary to generate the first member of the Krylov subspace $\mathcal{K}^n(A, r_0)$.

B.3 Explaining general Krylov recursions

The general Kylov recursions are given by

$$x_{n+1} = \underbrace{x_n + \alpha v_n}_{\text{Richardson iteration}} \underbrace{-\sum_{l=1}^{\hat{l}} \gamma_l \Delta x_{n-l},}_{\text{Correction}} \tag{B.16}$$

$$r_{n+1} = \underbrace{r_n - \alpha A v_n}_{\text{Richardson iteration}} \underbrace{-\sum_{l=1}^{\hat{l}} \gamma_l \Delta r_{n-l},}_{\text{Correction}} \tag{B.17}$$

where v_n is some vector related to r_n, α is some weighting and the summation terms are composed of differences between successive iterations that are scaled by some parameters γ_l. If these equations are modified such that they are limited to a single term recurrence (i.e. $r_{n+1} = f(r_n)$, where f is some generic function) and v_n is chosen to be simply r_n, then these equations reduce to the “modified Richardson iteration” as described previously in Appendix B.1 and indicated by the underbraces on the above equations. However, in the interest of creating a more efficient scheme, it would make sense to use information from more than just the present iterate when assembling the next iterate. It is this notion of combining information from a number of previous steps that leads to the idea of recurrence levels. The recurrence level simply refers to the number of successively previous iterations that contribute information to the computation of the next iteration (indicated by “correction” in the underbrace in the equations). The IDR(s) method is particularly attractive because the level of recurrence is fixed by the dimension of subspace S. Other methods, for example GMRES, have so called “long recurrences” meaning that information from every previous iteration is required to compute the next iteration. Long recurrence methods obviously have storage requirements that increase with the number of iterations performed and so are not particularly well suited for computing systems in which storage is limited.

On the assumption that a more optimal iteration can be performed if information from more than one recurrence level is used, the variable v_n is chosen as,

$$v_n = r_n - \sum_{l=1}^{\hat{l}} \gamma_l \Delta r_{n-l}, \tag{B.18}$$

where Δr_{n-l} is a matrix made from columns created from the differences between successive residuals and γ is a weighting function specific to each residual difference (each column). If $\hat{l}$ is chosen to be value s, the recurrence level becomes fixed as s and this is the base on which the IDR(s) method is formed.