8-2020

Improving the Efficiency of Coupled Hydrodynamic Predictions by Implementing a Fetch-based Parametric Wave Model

Samuel Carter Boyd

Improving the Efficiency of Coupled Hydrodynamic Predictions by Implementing a Fetch-based Parametric Wave Model

by

Samuel Carter Boyd

A thesis submitted to
The Department of Ocean Engineering and Marine Sciences of
Florida Institute of Technology
in partial fulfillment of the requirements for the degree of

Master of Science
in
Ocean Engineering

Melbourne, Florida
August, 2020

We the undersigned committee hereby approve the attached thesis, “Improving the Efficiency of Coupled Hydrodynamic Predictions by Implementing a Fetch-based Parametric Wave Model,” by Samuel Carter Boyd.

Robert J. Weaver, Ph.D.
Associate Professor of Ocean Engineering
Major Advisor

Steven M. Lazarus, Ph.D.
Professor of Meteorology

Stephen L. Wood, Ph.D., PE
Professor and Program Chair of Ocean Engineering

Richard B. Aronson, Ph.D.
Professor and Department Head
Ocean Engineering and Marine Sciences

Abstract

Title: Improving Efficiency of Coupled Hydrodynamic Predictions by Implementing a Fetch-based Parametric Wave Model

Author: Samuel Carter Boyd

Advisor: Robert J. Weaver, Ph.D.

Within a restricted estuarine environment, the use of third-generation wave models for predicting wave heights can be computationally expensive, signaling a need for model development that reduces the computational costs of existing coupled hydrodynamic models. This study focuses on the development and testing of a parametric wave solver that incorporates four wave height formulations (SMB, SPM, TMA, and CEM) for predicting wave properties in a restricted estuarine environment. The emphasis is on improved efficiency without affecting accuracy, allowing for ensemble wave-surge forecasting to be performed on desktop computational resources. Evaluation of the performance of the parametric solver is twofold, first both the parametric solver and a third-generation wave model, Simulating Waves Nearshore (SWAN), are compared to *in-situ* ADCP data at a point in the Indian River Lagoon, on Florida's east coast. Then the parametric solver and SWAN solutions are compared across the estuarine domain. The creation of three different synthetic wind fields allows for model comparison, with wind fields permitting testing of the parametric model in order to reproduce (1) fully developed conditions, (2) wind speed variability, and (3) wind direction variability in tropical storm level wind events. For consistency comparison, wave height solutions over the same domain are generated by SWAN and the parametric models. Comparisons made between the parametric model performance and SWAN show a 4-member parametric model is accurate to within 87% globally, with a runtime improvement of over two orders of magnitude compared to SWAN. The parametric model's ensemble average wave height was within 6% of the *in-situ* measured wave heights; SWAN also performed within 6%. Therefore, the parametric wave model proves to be a viable alternative to running an expensive third-generation wave model for predicting waves in an enclosed estuarine system.

Table of Contents

List of Figures

List of Tables

List of Equations

Acknowledgement

I would like to Acknowledge the professors who helped me achieve my dream: Dr. Stephen Wood, Dr. Geoffrey Swain, Dr. Ronald Reichard, Dr. Steven Lazarus, and most of all, Dr. Robert Weaver. Dr. Weaver has been the most influential part of my academic career, from the day I requested he be my advisor to the day I graduated with my Master's degree. He urged me to continue my education, for which I could not be more grateful, and assured me that if I work hard then everything will fall into place. It took me until very recently to understand he was right. Thank you for pushing me to be the best I can be, both academically, and as a person.

Dedication

I would like to dedicate this paper, and my entire academic career, to my Family. To my grandparents Gran, Nana, and Poppy, who supported me emotionally and financially throughout this journey. To my siblings, Conrad, Gabby, and Tanner, who gave me the passion to succeed and the unconditional motivation, love, and support to keep going. And most of all, to my parents, Keith and Sandy, who taught me to treat others with respect, have an open mind, have confidence in my abilities, follow my dreams, pursue my passions, and most importantly, do what I love. Thank you.

1.0 Introduction

This research is motivated by the desire to protect coastal communities from the devastating impacts of severe storms. These storms, such as hurricanes, frequently impact the U.S. east coast and Gulf of Mexico regions and often cause devastation to coastal communities in the form of destroyed infrastructure, disruption to biological communities, and most importantly, loss of lives. Advanced weather and storm surge forecast models help to alleviate the impact of these storms by allowing local decision makers and emergency management teams to assess the areas at highest risk prior to storm impact. In doing so, these organizations are able to evacuate and allocate resources according to the specific locations forecasted by the models. Accurate and efficient numerical storm surge models are therefore necessary tools for coastal communities that allow safety and prosperity to persist in the face of devastating natural events.

Because of the increased rate of computer processing power, the field of numerical modeling has steadily evolved (Voller and Porté-Agel, 2002). This increased processing power is a result of developments in high performance computing (HPC) which are composed of two major components: advanced algorithms capable of accurately simulating complex, real-world problems; and advanced computer hardware and networking with sufficient power, memory, and bandwidth for executing those simulations (Tezduyar *et al.*, 1996). Most individual, university, and industry computers are extremely limited in processing capabilities, resulting in challenges when trying to work with numerical modeling problems where sufficiently large computational resources are required. This computational limitation has led to the exploration of more efficient algorithms to solve problems faster while still using the available hardware.

In the field of Ocean Engineering, numerical models can be classified into four different types according to the relevant physical phenomena (Sánchez-arcilla and Lemos, 1990), Table 1.

Table 1 Surf Zone Phenomena (Sánchez-arcilla and Lemos, 1990)

Surf Zone Phenomena	Spatial Scale (m)	Time Scale
Sediment transport and changes in morphology	100 - 1000	1 day - 1 month
Currents (non-oscillatory flows)	100 - 1000	10 min - 1 hour
Organized oscillatory flows (waves)	1 - 100	10^{-1} sec - 10 min
Random oscillatory flows (turbulence)	10^{-4} - 10^{-1}	10^{-3} sec - 10 sec

The type of model selected for a coastal simulation depends on the scale of simulation, domain size, and problem being addressed. For coastal regions at risk of hurricane impacts, the primary model parameter of interest is storm surge, namely water elevation. Storm surge is an abnormal rise in sea level primarily caused by high wind speeds pushing water towards the coast over a long fetch (distance which wind blows over water) (Yin *et al.*, 2020). The Florida coasts are an example of an area at high risk of hurricane-induced storm surge effects that may be devastating to coastal communities. Forecasting storm surge requires numerical models that account for currents (non-oscillatory flows) and waves (organized oscillatory flows), Table 1. Current-based numerical models are commonly referred to as hydrodynamic models and wave-based numerical models are commonly referred to as wind-wave models; both classes of models were investigated.

A model domain is the area of water or land that will be considered in the model simulation. The model domain for this research is Florida’s Indian River Lagoon (IRL). The IRL is a shallow (mean depth ~0.8 m) and narrow (~3 km wide) estuary extending 251 km between Jupiter Inlet and Ponce Inlet. The IRL is considered one of the most diverse estuaries in North America valued at $3.7B annually (Jiang, 2017). During the ASBPA *Storm Processes and Impacts Workshop (2018)*, decision makers expressed a desire to have a suite of storm surge results for the IRL which they can use for guidance (Weaver, Hartegan and Massey, 2018). In order to fulfill this desire, both components of the storm-surge forecasting system (hydrodynamic circulation and wind-waves) need to be efficient enough to allow for multiple simulation runs in a timely manner.

Within a restricted estuarine environment, the use of wind-wave models for predicting wave height can be computationally expensive, signaling a need for model development that reduces the computational costs of existing models. This study develops and tests a parametric wave solver for predicting wave properties in a restricted estuarine environment. The emphasis is on improved efficiency while maintaining existing accuracy. Such improvement allows for ensemble wave-surge forecasting to be performed on desktop computational resources. The specific models selected for IRL storm surge forecasting were investigated as well as the limitations of the existing hydrodynamic and wind-wave models.

1.1 Hydrodynamic Model

There are a variety of coastal-scale circulation models that are widely used for hydrodynamic predictions. Among these are HYCOM (Bleck, 2002), ADCIRC (Luettich, Westerink and Scheffner, 1992), FVCOM (Chen, Liu and Beardsley, 2003), SELFE (Zhang and Baptista, 2008), and SLOSH (Jelesnianski *et al.*, 1984). The Advanced Circulation, or ADCIRC, model is desirable for modeling circulation in the IRL because it can be implemented on a wide scale of computational domains ranging from deep ocean to estuaries, it allows for two- and three-dimensional calculations (Luettich and Westerink, 1991), it is highly scalable (P. C. Kerr *et al.*, 2013), and uses the finite element method, which allows for computation on highly flexible unstructured grids (Luettich, Westerink and Scheffner, 1992). ADCIRC, is an open-source software package that is used to solve time dependent, free surface circulation, and transport problems (Luettich, Westerink and Scheffner, 1992). ADCIRC solves the generalized wave continuity equation (GWCE) to compute the surface elevation and a modified form of the shallow water equation to compute the current velocity on an unstructured mesh (Luettich, Westerink and Scheffner, 1992). ADCIRC's solution accuracy has been validated by numerous studies including (Bhaskaran *et al.*, 2013; Chen *et al.*, 2013; P. C. Kerr *et al.*, 2013; Garzon and Ferreira, 2016; Akbar, Kanjanda and Musinguzi, 2017); see ADCIRC website for further publications.

1.2 Wind-Wave Model

Similarly, selection of an appropriate wind-wave model can be challenging due to the number of models available and the need to choose the right model for the project needs. Designed for specific applications, the physics driving the models can also differ. Some like the Simulating Waves Nearshore (SWAN) (Booij, Ris and Holthuijsen, 1999; Ris, Holthuijsen and Booij, 1999) and the Steady-State Spectral Wave Model (STWAVE) (Smith, 2001; Massey *et al.*, 2011) are phase averaging spectral wave models; this means that they compute average wave conditions. Others like the Fully Nonlinear Boussinesq Wave Model (FUNWAVE) (Kirby *et al.*, 1998; Bruno, De Serio and Mossa, 2009) and MIKE 21 FW (DHI Software, 2017) are phase resolving wave models; this means that they are able to resolve individual waves. Some are fully 3D two-phase models like OpenFOAM (OpenFOAM, 2014) and some are Lagrangian models that incorporate Smooth Particle Hydrodynamics (SPH) (Dalrymple and Rogers, 2006; Narayanaswamy *et al.*, 2010) solving the conservation equation for each particle.

Because of differing physics and modeling approaches, it becomes critical for the user to have a comprehensive understanding of each of the model's capabilities and limitations. The choice of wind-wave model depends on the size of the model domain, desired resolution of the wave field, and the computational resources available to execute the simulation. Based on the need for large domains and long simulation periods when running large-scale regional forecast models, and the requirement of performing simulation in an efficient manner in order to keep up with forecast cycles, only phase averaging spectral wave models were considered. The most up-to-date, accurate, and widely used phase averaging wave models are third-generation spectral wind-wave models such as WAM (Hasselmann *et al.*, 1988), WAVEWATCH III (Tolman, 1991), TOMAWAC (Benoit, Marcos and Becq, 1997), STWAVE (Smith, 2001), and SWAN (Booij, Ris and Holthuijsen, 1999; Ris, Holthuijsen and Booij, 1999). These models are considered to be “third-generation” because they parameterize all source terms to be proportional to the action density spectrum while imposing approximations, such as the Discrete Interaction Approximation (DIA) (Hasselmann *et al.*, 1985) or the Webb-Resio-Tracy (WRT) approximation (Webb, 1978; Tracy and Resio, 1982; Resio and Perrie, 1991), for the non-linear source terms. Each spectral model has its advantages, intended domain size, and conditions for optimal results.

The Simulating Waves Nearshore, or SWAN, model is desirable for calculating wind-waves in the Indian River Lagoon because the model was designed to compute random, short-crested waves in coastal regions with shallow water and ambient currents (Booij, Ris and Holthuijsen, 1999). SWAN’s accuracy and consistency has been validated by numerous studies including (Gruijthuijsen, 1996; Booij, Ris and Holthuijsen, 1999; Ris, Holthuijsen and Booij, 1999; Wood, Muttray and Oumeraci, 2001; Allard *et al.*, 2004; Padilla-Hernández *et al.*, 2007; Sartini, Mentaschi and Besio, 2015). SWAN solves the evolution of action density N (t, ϕ, θ, σ) over time (t), geographical space (ϕ), and spectral space of direction (θ) and frequency (σ) (Booij, Ris and Holthuijsen, 1999; SWAN, 2014). By integrating action density, wave properties, such as significant wave height and periods, are obtained.

1.3 Coupled Hydrodynamic + Wave Model

The processes described by the hydrodynamic model will affect the processes being described by the wave model and vice versa (Weaver and Slinn, 2005, 2007). The wind induced waves computed by SWAN, may contribute to the water level rise by as much as 35% (Weaver and Slinn, 2005, 2007; Resio and Westerink, 2008; Dietrich *et al.*, 2010). This change in water levels, in addition to the effect of currents computed by ADCIRC, will modify wave propagation and breaking. It is necessary that the two different models integrate with one another into one framework, so that the relevant physical processes and their influence on each other is considered. This technique of model integration is termed model coupling. The ADCIRC+SWAN coupled model has been previously formulated by (Dietrich *et al.*, 2012) and independently validated by (Bhaskaran *et al.*, 2013; P. C. Kerr *et al.*, 2013; Akbar, Kanjanda and Musinguzi, 2017; Chen *et al.*, 2013; Garzon and Ferreira, 2016). The temporal scales that the SWAN and ADCIRC models operate on are given by (Westerink *et al.*, 2018), Figure 1.

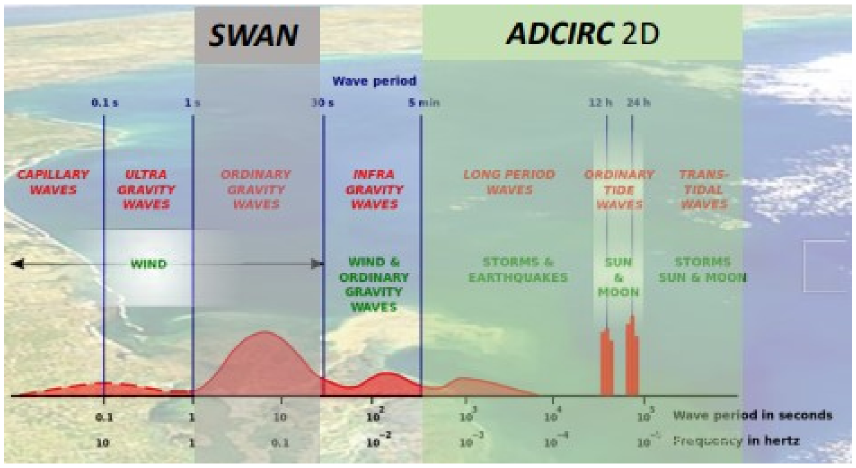


Figure 1 Operational temporal scales for the SWAN and ADCIRC models (Westerink *et al.*, 2018)

For the ADCIRC+SWAN coupled model, ADCIRC passes water levels, current velocities, and roughness lengths to SWAN where water depth, wave propagation, depth-induced breaking, and other wave process are calculated. SWAN then passes wave information, wave action, the ratio of group velocity to phase velocity, and relative frequency to the ADCIRC model where radiation stress gradients are calculated at each vertex (Dietrich *et al.*, 2012). Tightly coupling the models means that both models are able to use the same unstructured mesh and share the same sub-grids in parallel application; this greatly improves scalability of the system (P. C. Kerr *et al.*, 2013).

The coupled modeling approach is more comprehensive than a singular model because it accounts for dynamic process interaction which results in better representation of the wave field during an oceanic simulation. This makes coupled models attractive for coastal storm surge modeling; however, this increased accuracy comes with inherently long runtimes. Each component of the coupled model (hydrodynamic and wave) has its limitations; however, both models share the general limitations imposed by requiring high resolution (e.g. large number of computational nodes), and by the complexity of the numerical schemes developed for discretization of the governing equations. These limitations were explored by developing a wave solver that is both accurate and highly efficient which allows for ensemble storm-surge forecasting in the IRL.

2.0 Background

In this section the limitations of both components of the coupled model (hydrodynamics and waves) are investigated. These limitations include high model resolution, which results in a large number of computational nodes, and expensive numerical scheme discretization. In addition, the methodology behind obtaining a parametric wave height formulation is discussed.

2.1 Model Limitations

2.1.1 Large Number of Computational Nodes

Coastal-scale wind-wave models are expensive because they have model domains with a large number of computational nodes. In a semi-enclosed domain such as an estuary (IRL), bay, or lake, the model domain size is a function of the bathymetric resolution required to propagate the solution accurately. The bathymetric resolution is also a function of the complexity of the area of study. Generally, these semi-enclosed domains are largely heterogeneous, with rapid changes in bathymetry and complex shoreline geometry; the IRL is one such domain. Therefore, to model the IRL and similar domains accurately, the grid resolution may need to be on the order of tens of meters. This high resolution results in gridded domains with hundreds of thousands to millions of computational points.

Furthermore, to accurately model the circulation in an estuarine environment, an additional domain needs to be considered; the domain that connects the semi-enclosed domain to the model boundary, which lies thousands of kilometers away in the open ocean. There are several reasons for including such a large area of the ocean in the estuarine model: first, to accurately resolve complex coastal geometries, bathymetries, and scales of motion a wide range of spatial resolution is required to maintain model stability; second, the open-ocean boundary must be placed far away from the coastal region of study to minimize boundary effects (Westerink *et al.*, 1992); and third, the grid has to provide sufficient resolution for the tidal, wind, and atmospheric pressure forcing's to propagate from the ocean basins to the coastal floodplain. The resulting domain required to model coastal circulation on the U.S. east coast covers the West North Atlantic Ocean, Caribbean Sea, and the Gulf of Mexico between 98°W and 60.7°W and between 9°N and 47°N. An example of this domain is the S08 grid developed by (Westerink *et al.*, 2008), Figure 2.

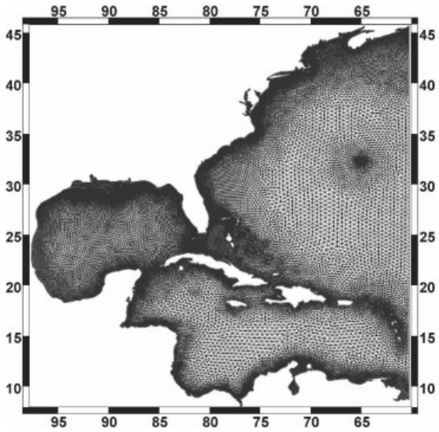


Figure 2 S08 Computational domain specified by (Westerink *et al.*, 2008)

Therefore, to study the hydrodynamics inside of Florida's Indian River Lagoon, much of the Atlantic Ocean and the Gulf of Mexico must be included in the computation; otherwise, the computation may become unstable and unreliable. This necessarily involves model calculations at a large number of points with a high degree of grid flexibility, even though the study is only on a lagoon that is 250 km long. (Taeb and Weaver, 2019).

One solution to the problem of a large number of computational points is to divide the domain into multiple regions; for example, one smaller domain that captures the high resolution of the coastal region's geometry and bathymetry and one larger domain that captures the entirety of the open ocean to the boundary in coarse resolution. An example of the continuous domain and the split domain is shown in Figure 3.

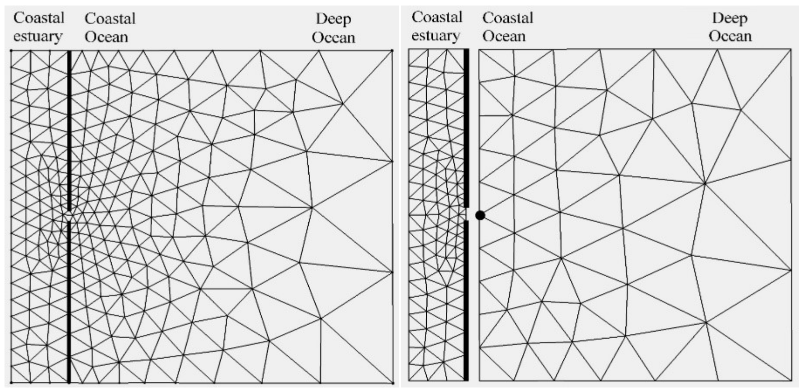


Figure 3 Continuous domain (A) and nested domain (B)

This technique is called one-way nesting and it helps to improve the efficiency of the simulation by reducing the number of computational nodes (Taeb and Weaver, 2019). Nodes are eliminated by allowing the resolution to jump from coarse to fine at the boundary of the two domains without having to smoothly vary as in the single domain approach, Figure 3. This is achievable because computation on the fine grid is forced with the boundary conditions that are generated by the computation on the coarse grid (Harris and Durran, 2010; Ji, Aikman and Lozano, 2010). Since the simulation time step is restricted to the smallest element in the domain, the wall-clock time for model execution is further reduced by allowing a significantly larger time step to be applied to the coarse mesh because of the increased size of the new smallest element. In addition, the one-way nesting technique allows for different physics to be performed on each mesh. For example, in the case of a 3D circulation simulation inside an estuary, it may only be necessary to perform a simpler and quicker 2D barotropic run on the coarse mesh that will produce the necessary boundary conditions to be used for the 3D baroclinic estuarine simulation (Taeb and Weaver, 2019). The combination of a smaller timestep, simpler physics, and less computational nodes, as a result of a nested domain, allows a simulation to run much quicker than on a traditional continuous domain (Blain, Cambazoglu and Kourafalou, 2009; Taeb and Weaver, 2019). The coarsely resolved ocean mesh and the highly resolved lagoon mesh created for the IRL are shown in Figure 4.

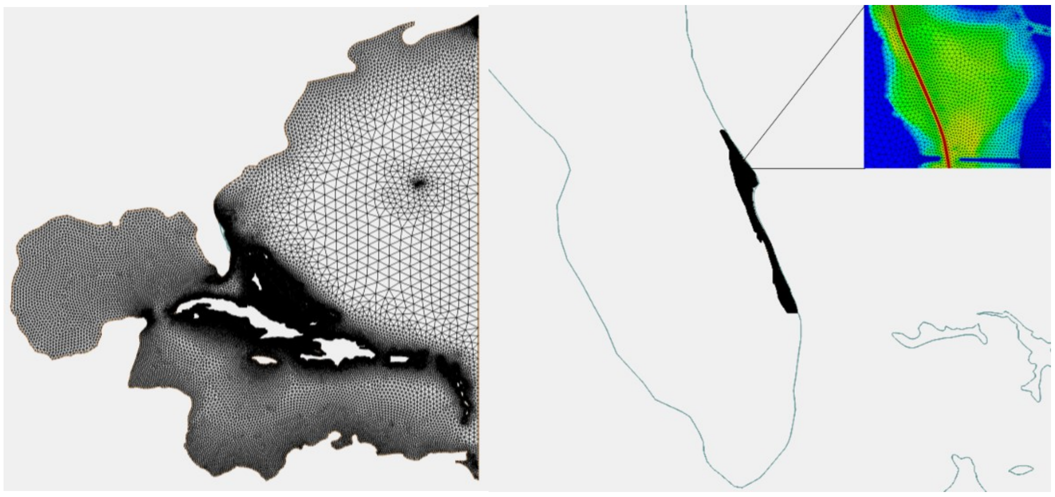


Figure 4 Coarsely resolved ocean mesh with 40,320 nodes (A) and highly resolved IRL mesh with 126,772 nodes (B) used for one-way nesting technique

These two meshes have a combined total of 167,092 nodes, which is about half the number of nodes (314,442) used in the traditional S08 mesh, Figure 2. In addition, the nested-domain technique achieved the same level of resolution as the traditional domain at the area of interest (less than 100 m in the channels). The solution accuracy and physics are also maintained because this approach produces the same surface gravity features at the boundary, which ensures volume and mass conservation between the two domains (Chen *et al.*, 2013), (Chen *et al.*, 2016).

This one-way nesting technique was incorporated into an automated coastal estuarine modeling system termed *Multistage* in a study by (Taeb and Weaver, 2019). *Multistage* takes different wind forcing predictions provided by agencies such as NAM, GEFS, and SREF and uses them to run a coupled circulation and wave model on two unstructured meshes that are one-way nested, Figures 3 and 4. The study implemented the *Multistage* tool in Florida's Indian River Lagoon and found a significant reduction in runtimes by 54% to more than 80% as compared to single-domain approach. This increased efficiency was used to perform repetitive computations, allowing for an ensemble of results. *Multistage* successfully performed 3-5 ensemble simulations using the same number of CPU-hours as a traditional single-domain approach (Taeb and Weaver, 2019). One-way nesting is therefore a method that can help to improve the efficiency of a finite element oceanic numerical simulation by incorporating new algorithms on the same hardware available. In particular, the nested domain approach is able to greatly reduce the number of nodes and simulation time required for a coastal simulation while still achieving the same level of bathymetric resolution, and governing physics.

2.1.2 Numerical Schemes

Another factor that limits coupled simulation efficiency is the complexity of the numerical schemes developed for discretization of the governing equations and the implicit schemes used by some models to propagate solutions forward in time. The governing equations associated with the ADCIRC have the advantage of being explicitly discretizable; this means that they can be directly solved. This is advantageous because expensive iteration schemes are not necessary; however, the resulting simulation timesteps required for model stability are small (on the order of 1 second). The governing equations associated with the SWAN model are handled implicitly; this means that they require expensive iteration schemes for solution convergence and propagation but can advantageously incorporate large time steps (on the order of 10 minutes) while still remaining stable. In either case, small time steps or expensive propagation schemes, resulting from the numerical scheme discretization and solution propagation, ultimately limit the runtime efficiency of either component of the coupled model.

Furthermore, the runtimes for the coupled model are greater than the sum of the runtimes for each model running independently because of the complex interactions between them (Dietrich *et al.*, 2012). For the ADCIRC+SWAN model specifically, the runtimes are highly dependent on the number of iterations SWAN needs for its solution to converge; this is due to the implicit numerical scheme used to propagate SWAN's solution forward in time (SWAN, 2014). The effect of SWAN iterations on the coupled model's total runtime was demonstrated by (Weaver, Hartegan and Massey, 2018) as shown in the table below.

Table 2 Computational wall clock time for coupled ADCIRC+SWAN simulations (Weaver, Hartegan and Massey, 2018)

Run	Iteration	Wallclock time (min)
1	1	46
2	2	70
3	4	120
4	8	170
5	10	180

Limiting SWAN's iterations can be effective in reducing simulation runtime but it comes with a reduction in solution accuracy. This effect is demonstrated by (Weaver, Hartegan and Massey, 2018), Figure 5.

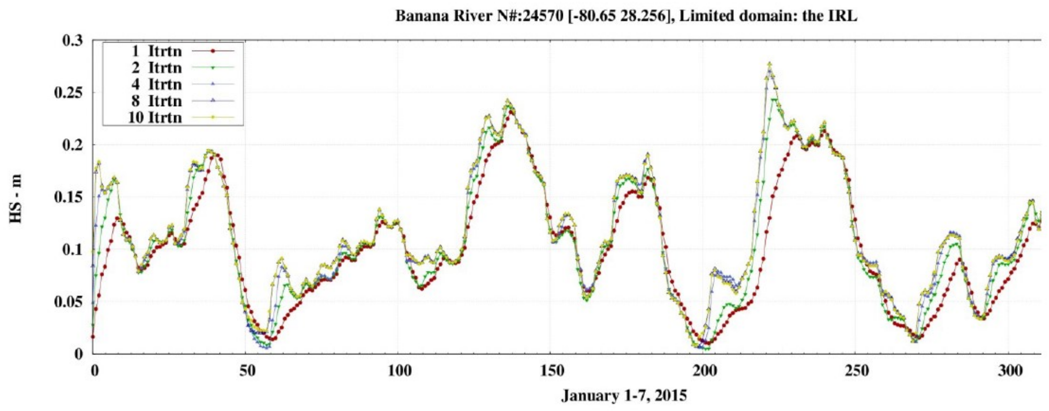


Figure 5 SWAN solution (at node 24570) as a number of iterations. The solution converges as SWAN iterations increase.

To save computational time, *Multistage* currently limits SWAN to 1-iteration within the high resolution IRL domain and in doing so successfully reduces model run-time and allows for 3-5 ensemble runs; however, the solution is not converged and therefore is less accurate than the optimal fully-converged solution. Therefore, the method of improving simulation runtime by reducing SWAN iterations is exhausted.

In addition, the runtimes of the coupled model are highly dependent on the number of processors available for the computation. The coupling between ADCIRC and SWAN (Dietrich *et al.*, 2012) allows for parallel computation; this means that the computational burden can be divided between multiple computer processors which can all function simultaneously. The relationship between runtime and computer processors in a parallel computation is effectively linear, Figure 6.

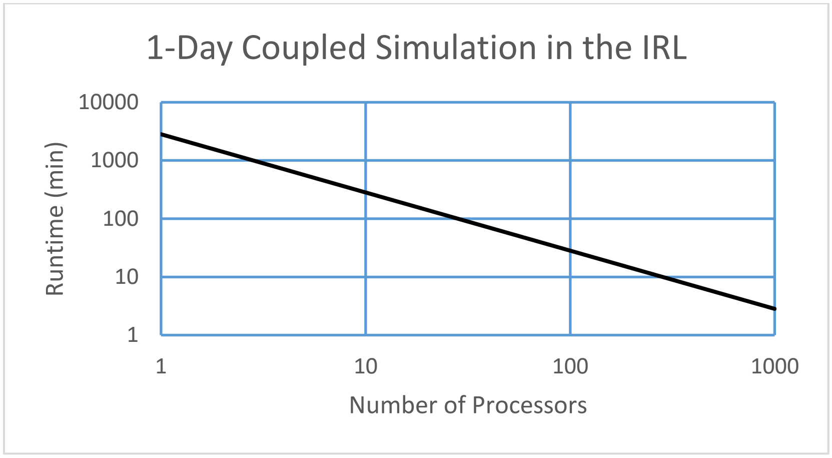


Figure 6 Parallel computational runtime as a function of computer processors. The plot was generated by extrapolating the runtimes given by (Weaver, Hartegan and Massey, 2018) for a 1-day IRL simulation on 45 processors.

Parallelizing the model allows for potentially enormous reduction in simulation runtime; but as mentioned before, the computational resources available for performing these types of simulations are usually limited. In order to improve the efficiency of the wave + circulation coupled model, we looked to implement an alternative solution to SWAN; a solution for wave height that does not need to iterate (i.e., is not time dependent) and can be performed on a single computer processor.

This is achieved through the use of a simple parametric wave solver that is coupled to ADCIRC within the *Multistage* system with an application of ensemble storm surge forecasting. In the process of creating the simplified wave solver, different parametric formulations for calculating wave height and physical processes was investigated. In order to be successful, the solution should not be degraded by using parametric wave model vs the SWAN model. Both the parametric solver and the SWAN model solutions were evaluated against *in-situ* ADCP data at a single site in the test domain. Additionally, the parametric wave solver's accuracy and run time was compared against the fully converged nonstationary SWAN solution across the entire IRL domain. Since third-generation wave models are found to perform similarly in accuracy and computational time (Padilla-Hernandez, Perrie and Toulany, 2004; P. C. Kerr *et al.*, 2013), the results should hold for any third-generation wave model.

2.2 Parametric Wave Height Solution

The problem of mathematically representing an ocean state can be approached in several ways. The two most common methods for addressing this problem are the significant wave method, and the wave spectra method. Both methods have certain advantages and certain disadvantages and vary fundamentally in their approach to the problem. It is important to understand that all formulations developed for representing ocean waves are empirically derived and may also differ in their underlying assumptions; thus, they may differ in their accuracy depending on how closely the application region is to the data set from which the equations were derived. A study of the different forecasting methods found that each method works better for the particular region from which the principle data were obtained (Roll, 1957). The significant wave method and the wave spectra method are investigated in the following sections.

2.2.1 Significant Wave Method

The Significant wave method is a deterministic approach to modeling the properties of surface waves. It is very simple because it describes only a single representative wave with a representative height and period for the entire system being modeled (Tolman, 2010). The basic tenet of this prediction method is that interrelationships among dimensionless wave parameters are governed by universal laws (U.S. Army Corps of Engineers, 2002).

The first rigorous method for wave forecasting was developed by Sverdrup and Munk (Sverdrup and Munk, 1947) who combined classical wave theory with available data to obtain semiempirical wave forecasting relationships. As shown by (Johnson, 1950), the generating parameters for ocean waves may be related by use of the PI-theorem (Buckingham, 1914) and one-dimensional analysis. The following parameters are:

$$\frac{gH}{U^2} = f_1\left(\frac{gH}{U^2}, \frac{gt}{U}\right) \qquad [1]$$

$$\frac{C_0}{U} = \frac{gT}{2\pi U} = f_2\left(\frac{gF}{U^2}, \frac{gt}{U}\right) \qquad [2]$$

where,

$H = H_{\frac{1}{3}}$ = significant wave height [m],

$T = T_{\frac{1}{3}}$ = significant wave period [s],

g = acceleration of gravity [m/s^2],

C_0 = deep water wave speed [m/s],

U = wind speed [m/s],

t = wind duration [s],

f = fetch length [m], defined as the uninterrupted horizontal length that wind blows over the water in a constant direction.

Equations [1] and [2] are developed under the assumptions of constant wind speed and direction. The relationships between the above variables can be written:

$$\frac{gH}{U^2} = F_1 \qquad [3]$$

$$\frac{gT}{2\pi U} = F_2 \qquad [4]$$

where F_1 and F_2 are functions of wind speed, fetch length, and wind duration. A form for solutions to these equations was initially given by (Wilson, 1955) and modified by (C.L. Bretschneider, 1952) who proposed revised coefficients from additional empirical data. The result is the following explicit solutions.

$$\frac{gH_s}{U^2} = 0.283\, tanh\left[0.0125\left(\frac{gF}{U^2}\right)^{0.42}\right] \qquad [5]$$

$$\frac{gT_s}{2\pi U} = 1.2\, tanh\left[0.077\left(\frac{gF}{U^2}\right)^{0.25}\right] \qquad [6]$$

These solutions give estimates for wave height and wave period development in terms of only three variables: gravity, wind speed, and fetch distance. The limitation of these solutions however is that they assume a deep-water condition and as such are not suitable for estimating wave characteristics in intermediate water. Further revisions to these equations were made to include the approximation of wave development in shallow and intermediate water depths. This was achieved by successive approximations in which wave energy is added due to wind stress and subtracted due to bottom friction and percolation. The deep-water solutions given in (Hasselmann *et al.*, 1976), similar to in Equations [5] and [6], were used to determine the energy added due to wind stress. Wave energy loss due to bottom friction and percolation is determined from the relationships developed by (Charles L. Bretschneider, 2011). Resultant wave heights and periods are obtained by combining the above relationships by numerical methods (CERC, 1984):

$$H_s(SPM) = 0.283 \tanh\left[0.530\ \left(\frac{gd}{\overline{U_A}^2}\right)^{0.75}\right] \tanh\left\{\frac{0.00565\left(\frac{gF}{\overline{U_A}^2}\right)^{0.50}}{\tanh\left[0.530\left(\frac{gd}{\overline{U_A}^2}\right)^{0.75}\right]}\right\}\frac{\overline{U_A}^2}{g} \qquad [7]$$

$$T(SPM) = 7.54 \tanh\left[0.833\ \left(\frac{gd}{\overline{U_A}^2}\right)^{0.375}\right] \tanh\left\{\frac{0.0379\left(\frac{gF}{\overline{U_A}^2}\right)^{0.333}}{\tanh\left[0.833\left(\frac{gd}{\overline{U_A}^2}\right)^{0.375}\right]}\right\}\frac{\overline{U_A}^2}{g} \qquad [8]$$

where,

$U_A = 0.731\ {U_s}^{1.23}$

and U_s is the wind speed at the water surface [m/s].

These solutions give estimates for wave height and wave period development in terms of four variables: gravity, wind speed, and fetch distance, and water depth. Equations [7] and [8] are referred to as the Sverdrup-Munk-Bretschneider (SMB) parametric formulations after the initial work performed by (Sverdrup and Munk, 1947) modified by (C. L. Bretschneider, 1952; C. L. Bretschneider, 2011), and later revised by (Mitsuyasu, 1970; Hasselmann *et al.*, 1973). They will be referred to as the SPM equations because of they are published in the Shore Protection Manual (CERC, 1984).

A study by (Malhotra and Fonseca, 2007) used these formulations with a slight variation; the wind is explicitly cast in terms of speed at the water surface, as opposed to the adjusted wind speed in the initial formulation. The coefficients and exponents in the equations are modified accordingly as seen in the following Equation.

$$H_s(SMB) = 0.283 \tanh\left[0.530\ \left(\frac{gd}{\bar{U}^2}\right)^{0.75}\right] \tanh\left\{\frac{0.0125\left(\frac{gF}{\bar{U}^2}\right)^{0.42}}{\tanh\left[0.530\left(\frac{gd}{\bar{U}^2}\right)^{0.75}\right]}\right\}\frac{\bar{U}^2}{g} \qquad [9]$$

The study by (Malhotra and Fonseca, 2007) did not use a modified form for the SPM wave period as wave period was not considered in the study. As such, the wave period used for the SMB formulation will be the same as the SPM wave period, Equation [8].

Another form of Equations [3] and [4] is given by (U.S. Army Corps of Engineers, 2002) under different assumptions about wave growth. The first assumption is fundamentally different from the SMB formulation as it neglects the effect of depth on the growth of waves in shallow water. This conclusion was reached based on work by (Bouws *et al.*, 1985; Janssen, 1989, 1991) which found that fetch-limited wave growth in shallow water appears to follow grown laws that are quite close to deep water waves for the same wind speeds. This assumption implies that any bottom-induced physical transformation effects on waves (such as friction, percolation, and shoaling) should be similarly neglected. The second assumption is that a local wave field propagates at a group velocity approximately equal to 0.85 times the group velocity of the spectral peak; this factor accounts for both frequency distribution of energy in a JONSWAP spectrum and angular spreading. Lastly, that deep water wave growth formulae should be used for all depths, with the constraint that no wave period can grow past a limiting value as shown by (Bouws *et al.*, 1985). The resulting Equations, [10] and [11], presented in the Coastal Engineering Manual (U.S. Army Corps of Engineers, 2002) (CEM) are:

$$H_s(CEM) = 0.0413\left(\frac{gF}{{U_*}^2}\right)^{0.5}\frac{{U_*}^2}{g} \qquad [10]$$

$$T(CEM) = 0.651\left(\frac{gF}{{U_*}^2}\right)^{0.33}\frac{U_*}{g} \qquad [11]$$

In conclusion, the significant wave concept is a method that forecasts the principle parameters, i.e., the significant wave height and significant wave period, based on fundamental assumptions about the growth of wind-waves. Based on different assumptions, various expressions for Equations [3] and [4] can be derived with additional empirical data, Equations [7]-[11].

2.2.2 Wave Spectra Method

The wave spectra method was introduced on the basis that waves at sea represent a stochastic process and that a single representative wave height, as described by the significant wave method, does not adequately describe the wave field (Tolman, 2010). The wave spectra method is fundamentally opposite to the significant wave method; that is, the significant wave method predicts the unit form of the theoretical spectrum from which the wave spectrum and the normal form of the directional spectrum can be derived, while the wave spectra method predicts the direction spectrum from which the one-dimensional spectrum and the significant wave height are then determined (Bretschneider and Tamaye, 1977). This method requires more computational resources but may be more accurate, especially in the distribution of wave energy with frequency (CERC, 1984). This method is traditionally approached by assuming that ocean waves are a weakly steady-state ergodic random process, where the wave profiles are distributed according to the normal probability distribution with zero mean and a variance representing the sea state severity. The statistical properties can be evaluated by analysis of the time history of a single wave record. By using the auto-correlation function, a wave record may be represented by the time average of wave energy (Ochi, 1998). This average may also be expressed in terms of the wave frequency, in radians per second, by applying the Parseval Theorem and a Fourier transform. The average wave frequency may then be written in terms of the spectral density function which represents the average energy of random waves with respect to time. By assuming the spectral density function is Rayleigh distributed (Longuet-Higgins, 1952), one can obtain meaningful wave characteristics by taking moments of a wave spectrum, Equation [12]. Wave characteristics such as significant wave height, Equation [13] and average wave period, Equation [14], may therefore be obtained.

$$m_i = \int_0^\infty f^i * S(f)\, df \qquad [12]$$

where,
m_i = the *i-th* moment, and
$S(f)$ = the wave energy spectrum in terms of frequency (*f*).

$$H_s \approx 4\sqrt{m_0} \qquad [13]$$

$$T_{mean} = 4\sqrt{\frac{m_0}{m_2}} \qquad [14]$$

The first proposed spectral formulation was developed by (Phillips, 1958) to represent the upper bound of the wind-generated deep-water gravity waves:

$$E_{Phillips}(f) = \alpha g^2 f^{-5}(2\pi)^{-4} \qquad [15]$$

where,
f = frequency,
g = gravity,
$\alpha \approx 8 * 10^{-3}$.

This formulation represents the steady-state equilibrium range of wind-generated waves in deep water. This spectral formulation was advanced by (Pierson and Moskowitz, 1964) who added an additional term which more accurately represents the low frequency forward face of the spectrum:

$$E_{PM}(f) = E_m(f)e^{-\frac{5}{4}\left(\frac{f}{f_m}\right)^{-4}} \qquad [16]$$

where, f_m is the frequency of the spectral peak which was empirically estimated from 10-meter high wind speeds (U) as:

$f_m = \frac{0.82g}{2\pi U}$ (Pierson and Moskowitz, 1963).

This formulation was extended to include partially developed waves by (Hasselmann *et al.*, 1973) with the addition of another factor:

$$E_{JONSWAP}(f) = E_m(f)e^{-\frac{5}{4}\left(\frac{f}{f_m}\right)^{-4}}\gamma^{exp\left[\frac{\left(-\frac{f}{f_m-1}\right)}{2\sigma^2}\right]^2} \quad [17]$$

where,

$\alpha = 0.076\left(\frac{gX}{U^2}\right)^{-0.22}$,

$f_m = T(TMA) = 3.5\left(\frac{g}{U}\right)\left(\frac{gX}{U^2}\right)^{-0.33}$, [18]

$\gamma = 7.0\left(\frac{gX}{U^2}\right)^{-0.143}$ (Mitsuyasu, 1982),

U = windspeed,

X = fetch distance,

σ = 0.07 for $f_m \geq f$ *and* 0.09 *for* $f_m < f$.

This spectrum more accurately represents narrower spectra which are typical of growing wind seas in deep water. This formulation was further extended to include the effect of shallower water depths on the shape of the wave spectrum (Hughes, 1984) with the introduction of yet another factor:

$$E_{TMA}(f,h) = E_m(f)e^{-\frac{5}{4}\left(\frac{f}{f_m}\right)^{-4}}\gamma^{exp\left[\frac{\left(-\frac{f}{f_m-1}\right)}{2\sigma^2}\right]^2}\left[\frac{k^{-3}(\omega,h)\frac{dk(\omega,h)}{d\omega}}{k^{-3}(\omega,\infty)\frac{dk(\omega,h)}{d\omega}}\right] \quad [19]$$

where, ω = 2πf. This spectrum was named the TMA spectrum after the data sets used for validation (Texel, MARSEN, and ARSLOE). By applying linear wave theory, making several assumptions about the primary frequency components, and integrating the spectrum (Hughes, 1984) a parametric expression for total energy in the spectrum may be written in the form:

$$E_{TMA2} = \frac{\alpha g h}{4(2\pi)^2(0.9f_m)^2} \qquad [20]$$

By taking the relationship between total energy and significant wave height developed by (Longuet-Higgins, 1952) the energy-based significant wave height may now be approximated as:

$$H_{m0} = H_s(TMA) = 4(E_{TMA2})^{0.5} \qquad [21]$$

The parametric form of the TMA spectrum for wave period will be taken simply as the modal frequency described by (Hasselmann *et al.*, 1973) in Equation [18]. Now parametric estimates for significant wave height and wave period in terms of only 3 variables (gravity, fetch, and wind speed) is available.

Several different spectral density formulations, similar to those mentioned above, have been developed such as the Neumann Spectrum (Neumann and Pierson, 1957), the Two-parameter spectrum (C. L. Bretschneider, 2011), the Six-parameter spectrum (Ochi and Hubble, 1977), the Toba Spectrum (Toba, 1972, 1973), and the SMB spectrum (Bretschneider, 1959). All of these spectra are attempts to mathematically model the stochastic process of ocean waves which vary in their assumptions and therefore relative accuracy in representing a sea-state. Since simplified parametric forms of these spectra were not readily available, the parametric form of the TMA spectrum, Equation [21], was the only spectral-based equation used in the parametric wave solver.

In conclusion, the spectral method is a more comprehensive analysis of the evolution of waves while the significant wave method is greatly simplified. It is interesting to note that simple parametric formulations for wave height and period can be derived by the spectral method (Equations [18] and [21]) as well as the significant wave method (Equations [7]-[11]) which all have forms that resemble the initial theoretical relationships (Equations [1]-[4]). Using these parametric formulations allows for greatly simplified and much more efficient wave height computation. This efficiency is used by the parametric wave solver by incorporating Equations [7]-[11], [18], and [21].

2.3 Wave Solvers

Different wave solving codes have been developed that solve for the spectral moments, Equation [12], by incorporating discretization schemes for the partial differential terms. The third-generation wave solvers account for complicated physical phenomena such as non-linear wave-wave interactions, breaking, shoaling, diffraction, friction, turbulence, etc. As mentioned in Section 2, SWAN is one of these types of wave solvers and it represents the wave field as a phase-averaged spectrum (Booij, Ris and Holthuijsen, 1999). SWAN allows the wave action density to evolve in time, geographic space, and spectral space according to the user-specified spectrum (JONSWAP is default). It then integrates the wave action density curve at every point in the domain at every time step in the simulation according to user-specified frequency and angular resolution.

According to the SWAN input parameters specified in the *Multistage* tool, Table 8, computation on the IRL domain involves 126,000 nodes with 36 directional bins each and a 45-frequency resolution for every directional bin. With a timestep of 10 minutes, SWAN has to solve approximately $2.94*10^9$ unknowns for every day of model simulation. This is also assuming only a single iteration SWAN solution, which as previously stated is not the most accurate solution.

In contrast, to solve any of the parametric formulations for wave height (SMB Equation [7], WEMO Equation [9], CEM Equation [10], and TMA Equation [21]), a wave solver needs only to solve the parametric wave height equation for every node in the domain, assuming the values of fetch, depth, and gravity are constant throughout the simulation. This means that a code that solves all four parametric equations with the same timestep as SWAN (10 minutes) will need to solve for approximately $7.23*10^6$ unknowns per day of model simulation. Theoretically, the parametric code will be about 400 times faster ($2.94*10^9$ / $7.23*10^6$) than SWAN at computing wave heights throughout the same domain with the same simulation timestep with all other processes neglected. The magnitude of runtime improvement will be investigated in section 4.

3.0 Methodology

The first step in creating a parametric wave solver is to consider the driving mechanism behind the wave heights: wind. Waves are a result of the wind blowing across the water's surface and transferring energy down into the water column. In deep water, wave heights are a function of wind speed, wind duration, and fetch distance. The parametric equations initially developed by (Hasselmann *et al.*, 1976) were simplified in the Shore Protection Manual (CERC, 1984) to a form that relates the variables governing the development of a sea state, Equation [22].

$$t = 0.893 \left(\frac{F^2}{U_A}\right)^{1/3} \qquad [22]$$

where,
t is time [hr],
F is the fetch length [km], and
U_A is the adjusted wind speed [m/s] defined in Equation [8].

In deep water, wave development will continue to occur as long as the wind forcing is sustained. Ultimately a sea will reach a state of equilibrium in which the wind, turbulence, and waves all balance; in this state waves can no longer grow and are said to be fully developed. A fully developed sea-state is very uncommon in the ocean, as it requires combinations of either long wind durations or long fetch distances in deep water (Hwang, 2006; Fontaine, 2013). Therefore, the three limiting cases for wave development are depth, fetch, and wind duration. When considering a smaller and shallower domain such as a lake, estuary, or lagoon, these three limiting cases must all be considered in the wave growth formulation.

In the IRL, the surrounding land limits the wind to a maximum fetch distance of about 15 km and an average fetch distance of about 1 km. With a limited fetch, the duration time for wind required to create a fully developed condition is greatly reduced. Equation [9] was used to create the family of wave growth curves (Figure 7) which show the combinations of wind speed (U), and fetch length (F) which result in a duration-limited sea state in the IRL when water depth is neglected.

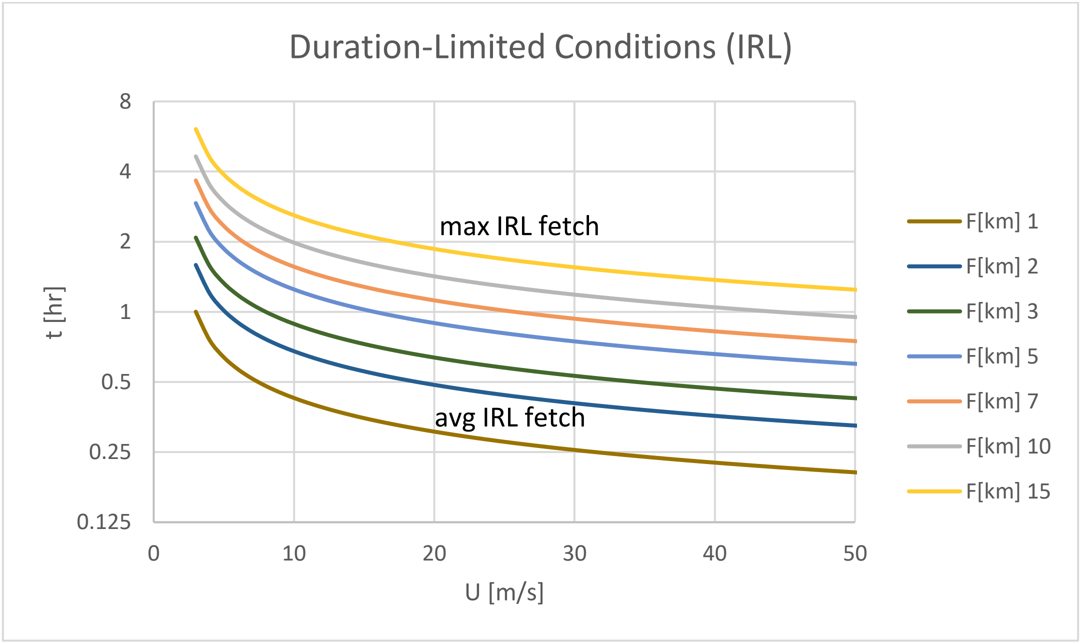


Figure 7 Combinations of wind speed and fetch length which result in duration-limited conditions for Equation [9] according to the deep-water growth relationship described by Equation [22]

As the fetch becomes smaller or wind speed becomes larger, the time required to reach a fully developed sea state is reduced, Figure 7. In an area such as the IRL, where the maximum fetch length is about 15 km, the wave generation is usually always duration-limited; this occurs when the wind duration exceeds the line represented by the combination of wind speed and fetch length at 15 km (yellow line). With an average fetch length of 1 km (brown line), this condition is usually met when wind greater than 5 m/s blows for a duration of 30 minutes or more. Similarly, Equation [22] may be rearranged to solve for the fetch distance that represents a fully developed sea state, i.e., the fetch-limiting case. The combinations of wind speed and wind duration that represent fully developed conditions are shown by the family of curves in Figure 8.

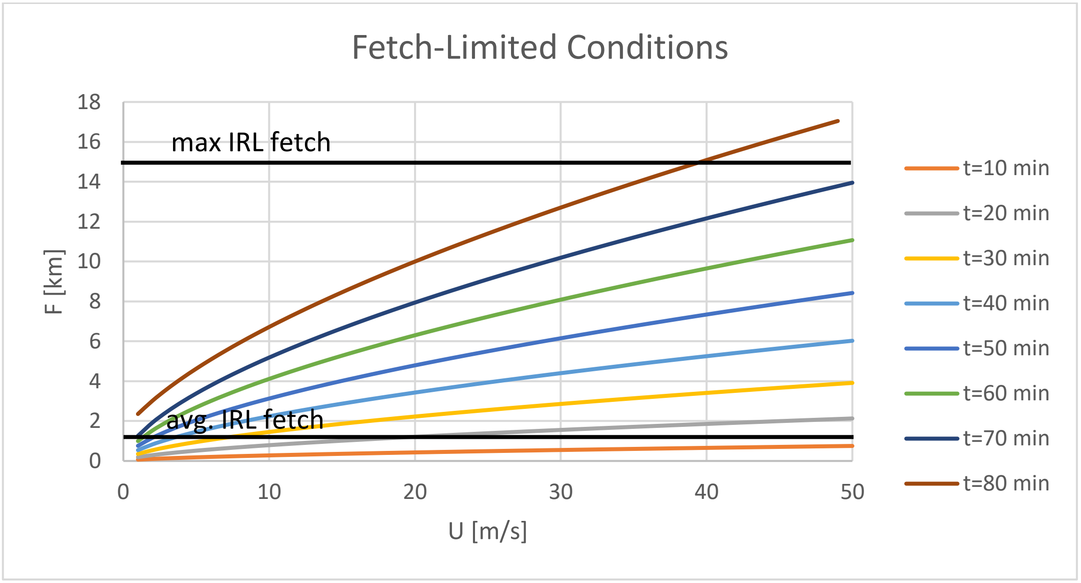


Figure 8 Combinations of wind speed and wind duration which result in fetch-limited conditions

From Figure 8, it can be seen that as the wind duration or wind speed become smaller, the fetch length required to reach a fully developed sea state is reduced as well. In an area such as the IRL where the maximum fetch length is about 15 km, the wave generation is usually always fetch-limited; this occurs when the fetch length exceeds the line represented by the combination of wind speed and wind duration.

With an average fetch length of 1km, this condition is usually met when wind greater than 5 m/s blows for a duration of 30 minutes or more; this also occurs almost constantly and corroborates the approximate wind duration (30+ min) and speed (5+ m/s) which result in limiting conditions as predicted by the duration-limited condition.

Commonly, wave growth is a combination of both the fetch-limited and duration-limited cases. A parametric model necessarily needs to assume that one of these cases are met because they represent asymptotic approximations to the general problem of wave growth. These limiting cases are exploited to create the parametric formulations in Equations [7], [9], [10], and [21]. Since wave growth with wind duration is not as well understood as wave growth with fetch length (CERC, 1984; Hwang and Wang, 2004), one can simplify the problem by assuming an infinite wind duration and only consider the limitation imposed by fetch length. The fetch limitation assumption is a conservative estimate that assumes the sea state to be fully developed. By making this assumption, the asymptotic approximations of wave growth for the fetch-limiting case can be exploited and any of the parametric formulations may be used for explicitly predicting wave heights. An example of wave development curves for the SMB model is shown in Figure 9.

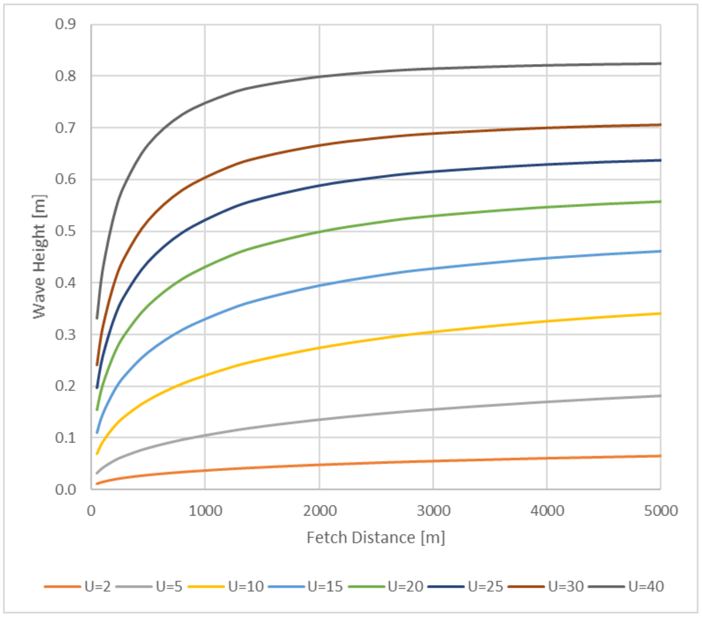


Figure 9 SMB wave height development curves generated from Equation [9] at average IRL water depth (1.8m)

Wave propagation and generation are also highly dependent on the depth of water in which waves propagate (Malhotra and Fonseca, 2007). According to linear (Airy) wave theory, waves are classified into categories based on water depth; these categories are defined in Table 3 according to the magnitude of d/L, where d is the water depth and L is the wavelength.

Table 3 Wave classification

Classification	d/L
Deep	>1/2
Intermediate	1/25 to 1/2
Shallow	< 1/25

The parametric SMB, SPM, and TMA formulations were derived to account for wave height variation with depth; this is why they include a depth term. The CEM formulation, however, does not include a depth term and does not consider any classification of waves beyond deep water from which it was derived.

By assuming a wave model operates in an enclosed domain and predicts waves that are only limited by fetch distance and water depth, the parametric formulations (SPM, SMB, CEM, and TMA) given by Equations [7], [9], [10], and [21] respectively, may be used. These formulations are functions of the following variables: gravity, water depth, fetch length, and wind speed. The methodology behind obtaining these variables will be discussed in the following section.

3.1 Model Input

The parametric wave code is written in C++ language in a way such that it pre-processes as much of the computation as possible. The parametric variables needed for wave height computation are water depth, wind speed, and fetch length. By assuming the values of water depth and fetch length are constant across a given domain, the solver may pre-compute these values and be forced by the only remaining variable, wind speed. The solver computes fetch lengths for every node radiating outward in all directions. The solver also computes water depth at each node and bottom slope in every direction. This information computed by the pre-processing part of the solver is written to files that are read in by the operational part of the solver. The operational code uses these domain-specific pre-computed values, in addition to the wind forcing provided, to compute wave heights according to the parametric formulations from Equations [7], [9], [10], and [21].

There are several additions to this process. First, the straight-line fetch rays computed by the code are cosine weighted in order to account for the effect of variations in wind transferring energy into the water column. Second, the depths are inverse distance weighted along straight-line fetch rays to account for depth variations upwind of the computational node. Third, the physical processes of friction, shoaling, and breaking are computed and used to modify the initial predicted wave heights.

In addition, the parametric solver is highly customizable. It requires user input of the averaging parameters, bathymetric and radial resolution, and friction coefficients applied to the physical processes. Because the parametric solver is flexible in these parameters, it may be used in different enclosed domains and may be optimally configured with the availability of additional data from future studies. A schematic of the code processes is shown in Figure 10 and is explained in following sections.

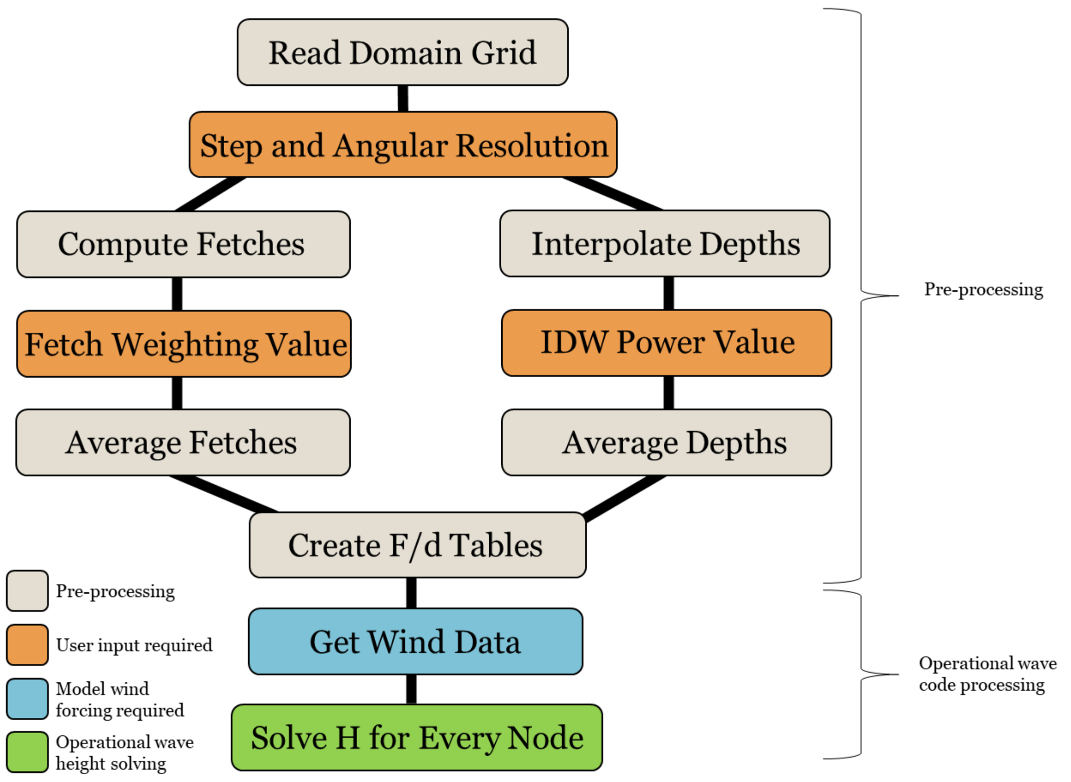


Figure 10 Parametric wave solver schematic

3.1.1 Fetch Length

The parametric wave solver works by starting at each wet node; it incrementally steps forward some distance in space, specified by the user, checking depths at each step until land is reached. Once the step hits dry land, the code computes the total distance traveled based on how many steps were taken. The code also linear interpolates for the final step in order to best estimate exactly where land meets water with an accuracy equal to half of the user-specified step distance, i.e., if the step size is 50m, the estimated fetch distance will have an error of no more than 25m; this is a high degree of precision when considering the average fetch length of 1000m. The step size partially determines how long the pre-processing of the code will take, i.e., the smaller resolution specified, the more steps required and the longer the run-time will be. However, care must be taken so that the step size is small enough to resolve the geometry and bathymetry of the grid being used; for the IRL a 50m step size was deemed adequate for resolving the domain's bathymetry and shoreline geometry.

The code steps incrementally along straight-line fetch rays checking for elevation at each step, but elevation information is only available at nodes. The code will land inside a triangular element made up of 3 nodes at every incremental step along a fetch ray. An example of the step location landing inside of a triangular element can be seen in Figure 11.

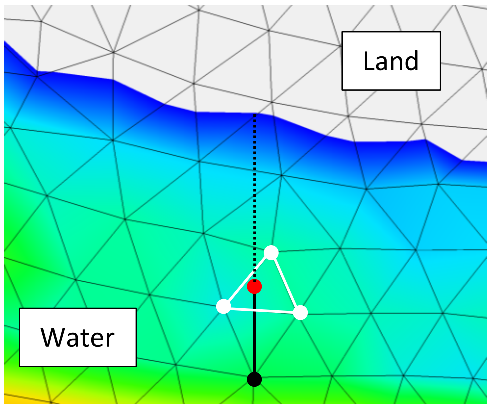


Figure 11 Small portion of the IRL domain with starting point (black dot), first step (solid black line) along North oriented straight-line fetch ray (dotted black line), with interrogation point (red dot), and containing element (white triangle) with elevation information at nodes (white dots)

The code must search through many elements to find which element the stepping point is contained in; this is done by using a cross product manipulation of node coordinates (latitude and longitude). If the cross product between the arbitrary step location and all three surrounding nodes is positive, then the step location lies inside of the element composed of the 3 surrounding nodes. Once the containing element is found, the elevation information at each of the 3 surrounding nodes is analyzed through Barycentric interpolation, Equation [23], to find the approximate elevation at the step point.

$$h_x = \frac{W_1 h_1 + W_2 h_2 + W_3 h_3}{W_1 + W_2 + W_3} \qquad [23]$$

where h_x is the unknown elevation, h_1, h_2, h_3 are the elevations of the surrounding nodes, and W_1, W_2, W_3 are the associated weights used when averaging, defined as:

$$W_1 = \frac{(Y_2 - Y_3)(h_x - X_3) + (X_3 - X_2)(h_y - Y_3)}{(Y_2 - Y_3)(X_1 - X_3) + (X_3 - X_2)(Y_1 - Y_3)},$$

$$W_2 = \frac{(Y_3 - Y_1)(h_x - X_3) + (X_1 - X_3)(h_y - Y_3)}{(Y_2 - Y_3)(X_1 - X_3) + (X_3 - X_2)(Y_1 - Y_3)},$$

$$W_3 = 1 - W_1 - W_2,$$

where X's correspond to the node's x-coordinate [deg. longitude] and Y's correspond to the node's y-coordinate [deg. latitude]. If this elevation is positive than the location is still wet and another step is taken and the process is repeated; if the elevation is negative than the location is dry and the fetch length is calculated up to this point with the haversine formula. The haversine, or great circle distance formula, is used to account for the effect of the earth's curvature on distance; it is more accurate than assuming a linear distance model and is given by:

$$F = 2\,R\,arcsine\sqrt{sin^2\left(\frac{\varphi_1 - \varphi_2}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\,sin^2\left(\frac{\emptyset_1 - \emptyset_2}{2}\right)} \qquad [24]$$

where F is the fetch length [m] or distance between the starting point and land, φ's are latitude [deg], ϕ's are longitude [deg], and R is the radius of the earth calculated by:

$$R = \sqrt{\frac{({r_1}^2 * \cos(\varphi))^2 + ({r_2}^2 * \sin(\varphi))^2}{(r_1 * \cos(\varphi))^2 + (r_2 * \sin(\varphi))^2}}$$

where r_1 is the Earth's radius at sea level (6378.137 km) and r_2 is the radius at the poles (6356.752 km). Here the φ value is the average wet latitude in the computational domain which is computed by the pre-processing code.

Once a fetch ray's distance has been calculated, the code will go back to the wet starting node and restart the process again for the fetch ray at the next angle. The proceeding fetch rays will again be straight lines but will be at an angle offset by the first fetch ray by the angular resolution specified by the user, Figure 12.

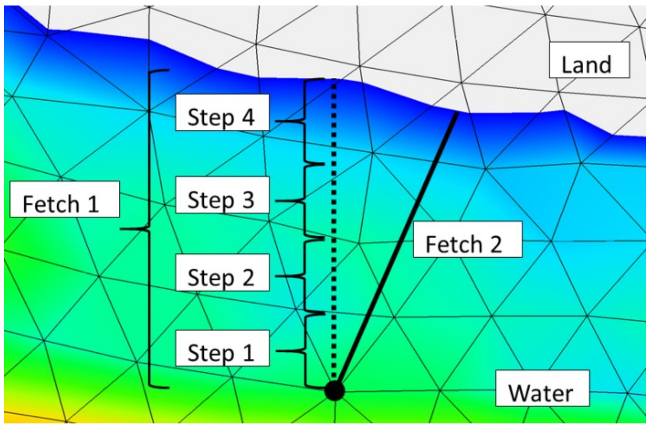


Figure 12 Second fetch ray (solid black line) offset by angular resolution (20° here) from first fetch ray (dotted black line)

Again, a lower angular resolution for the proceeding fetch rays will result in faster pre-processing times, but the solution becomes more unrealistic the coarser the angular resolution is. The effect of the angular resolution can be seen in the wave height contour plot, Figure 13, which was generated using Equation [9] for a section of the IRL. All parameters are held constant except for the angular resolution specified in the pre-processing fetch distance code.

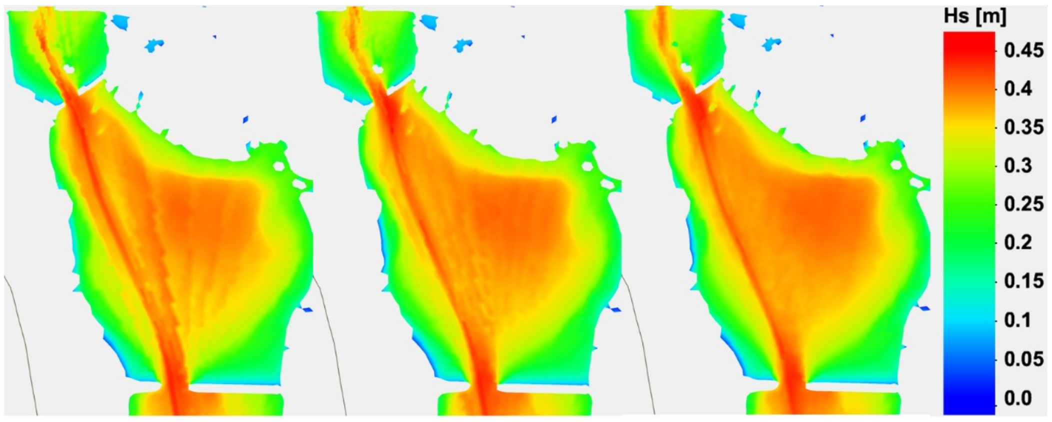


Figure 13 Wave height generated with fetch rays with angular resolution of (A) 10°, (B) 5°, and (C) 2°

As seen in Figure 13, the straight-line fetch distance scheme used to generate the fetches becomes a visible part of the wave height contour plot at lower angular resolutions. Care must be taken that an adequate angular resolution be provided such that the shoreline geometry and wind angle precision are adequate for the domain being investigated. Therefore, it is recommended that the angular resolution be as high as possible, especially for an irregular domain such as the IRL; in this study a 2-degree angular resolution was deemed adequate i.e., the solution is smoothed without adding extraneous fetch lengths to the computation. This means that the code will compute $360/\theta_{res}$ different fetch rays for every wet node in the computational domain; in the IRL this means 360/2° or 180 fetch rays per wet node.

3.1.2 Fetch Width

When considering the length of a fetch ray, it is important to also consider fetch width if the simulation occurs in an area with irregular shoreline geometry (Smith, 1991). In these domains the simple fetch length in a given compass direction, Equation [24], may give unrealistic results since the width of fetch can place a substantial restriction on fetch length. One strategy in accounting for restricted fetch width is to modify the simple fetch length by considering fetch lengths in off-wind directions. This is achieved by applying weighting factors to the fetch lengths and averaging these weighted lengths over large arcs on either side of the wind direction (Smith, 1991). A method initially proposed by (Saville, 1954) used a cosine weighted average modification to the simple fetch rays shown in Equation [25].

$$Eff\ F_i = \frac{\sum_{j=-n}^{n} F_j \cos(\theta_j)}{\sum_{j=-n}^{n} \cos(\theta_j)} \qquad [25]$$

where,

$Eff\ F_i$ = effective fetch distance for the i^{th} direction fetch ray,

F_j = simple (straight-line) fetch ray for j^{th} direction at $\theta_j * n$ degrees from *i* center,

θ_j = angle between i^{th} center and j^{th} fetch ray, and

n = number of rays considered on either side of the *i* starting ray.

This method was implemented by (Malhotra and Fonseca, 2007) who used this effective fetch formulation under the assumption that wind moving over water surface transfers energy to the water in the direction of the wind and in all directions within 45° on either side of the wind direction (Saville, 1954; Smith, 1991). The effective fetches were used with Equation [25] to calculate wave heights with moderate success (Malhotra and Fonseca, 2007).

It has been argued by (Resio and Vincent. C. Linwood, 1977) that wave conditions in fetch-limited areas are relatively insensitive to the width of a fetch; consequently, the CEM formulation recommends that the effective fetch method (Equation [25]) not be used. However, based on the methodology used to compute fetch distances at specific angular increments, the use of the effective fetch averaging method is also important in ensuring a smooth solution. This effect can be seen in Figure 14 which was generated by using different angular distance averages for the fetch values in Equation [9] while keeping all other parameters constant.

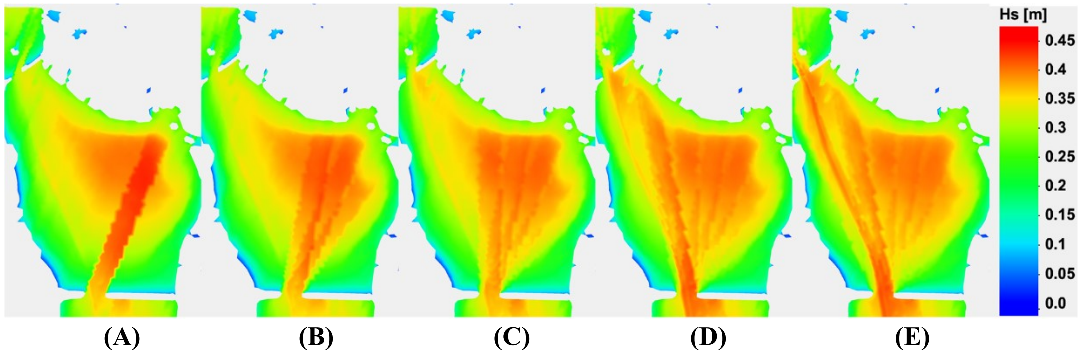
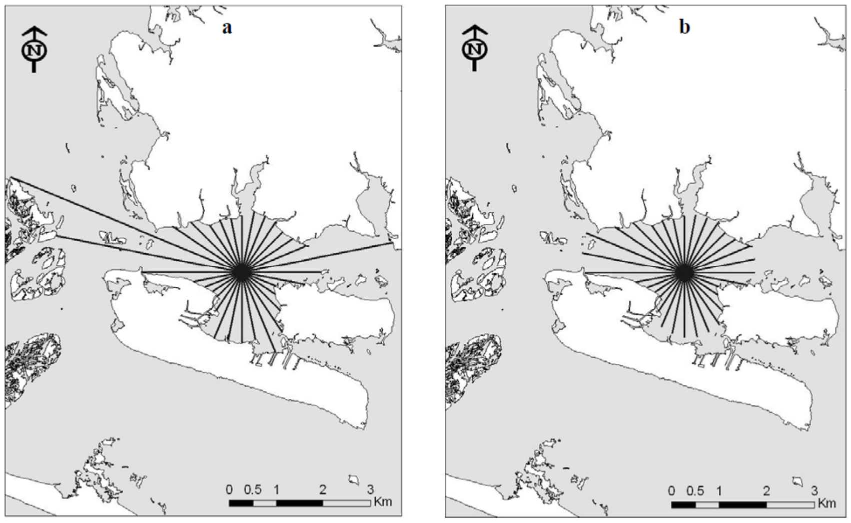


Figure 14 Wave height generated with cosine-weighted fetch rays with averaging distance of (A) 0°, (B) 10°, (C) 20°, (D) 30°, and (E) 40° on either side of the straight-line fetch ray

As seen in Figure 14, the use of a 40° cosine average yields a much smoother solution which is also consistent with (Saville, 1954; Smith, 1991). Therefore, the method of effective fetch rays is used in the parametric wave solver. The straight-line fetches calculated by Equation [24] are modified, according to Equation [25], before being used in the wave height calculation, Equations [7], [9], [10], and [21]. In addition, the contours in Figure 14 were created with straight-line fetch rays with 10° angular spacing. This coarse resolution was chosen to demonstrate the effect of solution smoothness with the cosine weighted effective fetch method; however, the contours in Figure 14(E) will be much smoother, since a finer resolution (2°) is used in the code executable, Figure 13(C).

This effective fetch formulation is currently employed in the parametric wave solver with a user-specified angular resolution considered on either side of the straight-line fetch, although the default setting is to include the effective fetch method with an area of 40° considered on either side of the straight-line fetch rays. This averaging method effectively trims down the fetch lengths and gives a more conservative wave height estimate for the longest fetch rays. This effect can be seen in Figure 15 (Malhotra and Fonseca, 2007) where raw fetch rays and effective fetch rays are both shown.



(A) (B)

Figure 15 Raw fetch rays (A) and effective fetch rays (B) (Malhotra and Fonseca, 2007)

3.1.3 Water Depth

The parametric formulations (Equations [7], [9], [10], and [21]) all have a fundamental limitation for depth; they take one depth value at the point of calculation but do not consider any of the depths upwind of this point. This is problematic because there may be dramatic changes in water depth along a single straight-line fetch ray, which will affect the resulting wave height at the node. To account for the effect of depth variations upwind of a node, an inverse distance weighting averaging scheme (Shepart, 1968), Equation [26], was applied to the depths along each straight-line fetch ray.

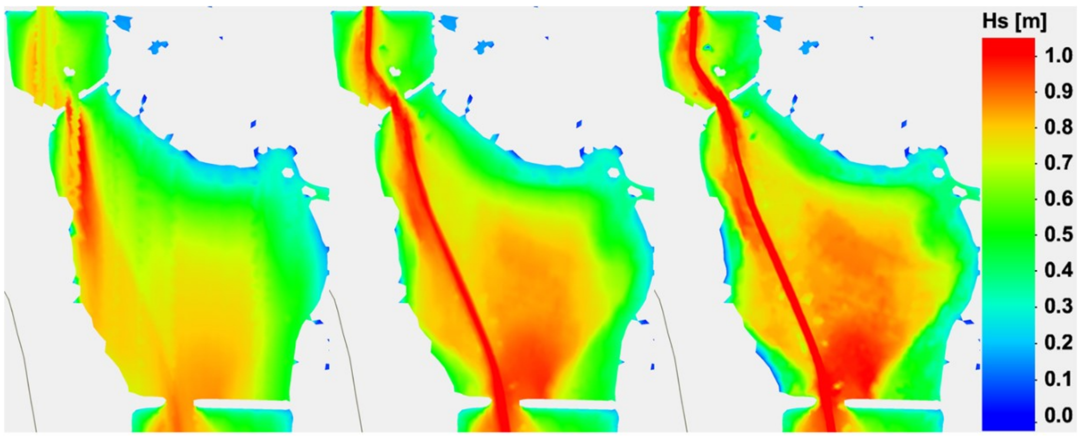
$$d(x) = \frac{\sum_{i=1}^{N} w_i(x) d_i}{\sum_{i=1}^{N} w_i(x)} \qquad [26]$$

where,

$$w_i(x) = \frac{1}{\Delta(x, x_i)^p} \qquad [27]$$

The equation describes a general form for finding an interpolated value (depth or d) at a given point (x), based on samples d_i for *i=1, 2,…,N.* Here $d(x)$ is the inverse distance weighted average water depth along a fetch ray at a distance x upwind from the calculation point and d_i is the water depth at step i upwind from the calculation point, and p is the power parameter for the weighting function.

The power parameter (p) applied to the distance between calculation points $\Delta(x, x_i)$; must be a positive real number. It affects the weight given to the depth at increasing distances away from the node. The effect of p-value variation is seen when comparing wave height contours, Figure 16.



(A) (B) (C)

Figure 16 Wave height generated with inverse distance weighting average for depth at p-values of (A) 0.10, (B) 1.0, and (C) 10

The effect of the inverse distance weighting p-value is that it weights upwind depths more heavily at low values, Figure 16(A), and approaches the raw depth at the node at higher values, Figure 16(C). Choosing the optimal power parameter for an application is not trivial. Many studies have been performed on p-value optimization (Lu and Wong, 2008; Li and Gao, 2014; Mei, Xu and Xu, 2016). It has also been suggested that the optimal p-value is a function of wind speed (Malhotra and Fonseca, 2007). However, performing an optimized p-value analysis is domain dependent and requires validation against field data which is not available in the IRL domain. As such, several constant values were tested, and the results compared to the SAWN solution. In this initial analysis, it was determined that a p-value of 5.0, Equation 27, gave parametric model results that most closely resembled SWAN's solution across the domain. Therefore, a p-value of 5.0 will be applied to the water depths along each fetch ray across the domain for all simulations in this study.

3.1.4 Model Input Summary

The final result of the processes described in this section is a table of cosine weighted fetch distances and a table of inverse distance weighted depths along each fetch ray for every node in the domain. This is the output of the pre-processing part of the parametric wave code.

The code first computes straight line fetch rays through the step by step process described in Section 4.2 by finding the depth at each step through Barycentric Interpolation (Equation [23] and Figure 11) and computes straight-line fetch distance once land is reached (Equation [24]); this process is repeated for the next fetch ray offset by the user-specified angular resolution (Figure 12). Once all of the fetch rays are computed for the computational wet node, the processes are repeated for every wet node in the domain. The straight-line fetch rays are then cosine averaged (Equation [25] and Figure 14). The depth values computed from the interpolation (Equation [23] and Figure 11), that occur during the fetch distance process, are stored for every node and straight-line fetch ray. These depth values are then inverse distance weighted (Equation [26]) according to the p-value provided (Figure 16); 5.0 is used in this study. These modified depth and fetch tables are output in two separate files which will be used with a simulation's wind forcing to solve the parametric wave height and period equations (Equations [7], [9], [10], and [21]) in the operational part of the code.

3.2 Physical Processes

The significant wave height predicted by Equations [7], [9], [10], and [21] only considers the wave height as a result of wind energy input to the water column;

they do not explicitly consider any other physical processes or mechanisms of energy dissipation/transformation. As such, the significant wave height predicted by the parametric equations may be inaccurate in areas where other physical processes are significantly contributing to total wave energy. Many complex processes can affect wave height; however, the three processes, which have simple parametric solutions and are considered most prominent, are shoaling, breaking, and friction. In the context of the parametric computations, these effects will occur for waves which propagate along the straight-line fetch rays at a constant compass heading determined by the wind direction at each timestep in the simulation. The significant wave heights calculated by Equations [7], [9], [10], and [21] will be modified according to:

$$H_s = \begin{cases} H_s \pm H_{shoal} - H_{friction} \text{, when no breaking occurs } \left(\frac{\mathrm{h}}{\mathrm{L}} > 0.05\right) \\ H_{break} \text{, when breaking occurs } \left(\frac{h}{L} < 0.05\right) \end{cases} \quad [28]$$

3.2.1 Wave Shoaling

Wave shoaling is the effect by which waves change in height due to changes in water depth. Shoaling is caused by the fact that the group velocity, which is also the wave-energy transport velocity, changes with water depth. Under stationary conditions, a decrease in transport speed must be compensated by an increase in energy density in order to maintain a constant energy flux (Longuet-Higgins and Stewart, 1964). The formulation for wave shoaling coefficient, or the ratio of wave height (H_s) to equivalent deep water wave height (H_0), given by (May, 2006) is:

$$K_s = \frac{H_s}{H_0} = \left(\frac{2*cosh^2(kd)}{\sinh(2kd)+2kd}\right)^{0.5} \quad [29]$$

where,
k = wave number (2π/L),
d = water depth.

The shoaling coefficient varies according to depth as shown in Figure 17.

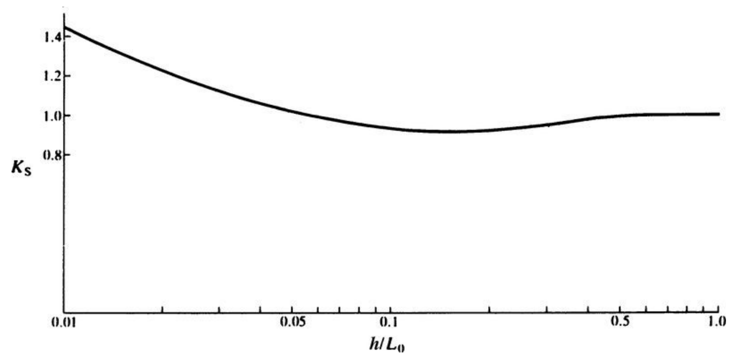


Figure 17 Shoaling coefficient as a function of water depth and deep-water wavelength (May, 2006)

The shoaling coefficient approaches 1 in deep water, drops slightly in intermediate water, and then exceeds 1 as the wave enters shallow water. Therefore, the net effect may be either a decrease or an increase in wave height depending on the relative water depth at the point of calculation.

3.3.2 Wave Breaking

Waves break when they become unstable; this may occur in deep water or in shallow water depending on the mechanism that causes waves to exceed linear stability. In deep water, the wave breaking limit is generally set at d/L = 1/7, but this is not appropriate for shallow water (Dean and Dalrymple, 1984). The Wave breaking limit in shallow water depends not only on the relative depth ratio but also on the beach slope (Booij, 1994). The expression for wave breaking in shallow water is given by (Wood, Muttray and Oumeraci, 2001; Goda, 2010) as:

$$H_{break} = L_0 * A\left[1 - exp\left(-1.5\frac{\pi d_b}{L_0}\left(1 + 15tan^{4/3}\alpha\right)\right)\right] \qquad [30]$$

where,

H_{break} = wave height at breaking,
L_0 = wave length predicted by each model,
$A = 0.17$ (Goda coefficient) (Wood, Muttray and Oumeraci, 2001; Goda, 2010),

d_b = water depth at breaking, and
$\tan(\alpha)$ = bottom slope.

The bottom slope is calculated as the difference in water depth one step upwind of the node. If the breaking criteria is satisfied, the pre-computed value of bottom slope in the direction upwind of the node is used. In addition, although the formulation specifies wavelength for deep water, much of the domain is in intermediate and shallow water; as such, the simple wavelength as calculated from the dispersion equation (Equation [31]) in linear wave theory is used.

$$\sqrt{gk \tanh(kh)} = \sqrt{\frac{2\pi g}{L} \tanh\left(\frac{2\pi h}{L}\right)} \qquad [31]$$

where,

k = wave number (2π/L),
h = water depth,
L = wave length.

3.2.3 Bottom Friction

There are various effects which contribute to energy dissipation in surface waves, such as bottom friction, viscous boundary flow, and percolation. Bottom friction however, is the most prominent (Carniello *et al.*, 2005); as such, bottom friction was the only dissipation mechanism considered in the parametric model. The expression for the friction decay factor given by (Putnam and Johson, 1949; Charles L. Bretschneider, 2011) is given in Equation [32].

$$H_{friction} = H_s/K_f = H_s\left[1 + \frac{64\pi^3}{3g^2}\frac{fH_sF}{T^4}\frac{{K_s}^2}{sinh^3\left(\frac{2\pi d}{L}\right)}\right] \qquad [32]$$

where,

H_s = wave height at calculation point [m],

f = friction factor defined by (Charles L. Bretschneider, 2011) as: $f = \frac{g}{\left(\frac{1.486}{n} * d^{1/6}\right)^2}$,

d = water depth from Equation 26 [m],

n = Manning's n value,

F = Fetch length [m],

T = wave period [s],

L = wave length [m], and

K_s = shoaling coefficient = H_{shoal}/H_s.

Manning's n values are available in most fluid mechanics books ranging from 0.01 to 0.05 for smooth to rocky/weedy channels. Due to the lack of field measurements performed in the IRL domain, an estimate for Manning's n is unavailable. To address this, two different values of Manning's n (0.01 and 0.02) will be tested and applied to Equation [32]. The friction decay factor approaches 1 in deep water and becomes exceptionally large in shallow water. In all cases, it modifies the significant wave height by H_s/K_f and therefore acts to reduce wave heights, except in deep water where its effect is nullified.

3.2.4 Section Summary

The parametric wave solver takes a wind forcing every simulation timestep. It calculates the wave height (SMB, SPM, TMA, and CEM formulations) based on the fetch distance and water depth which correspond to the wind direction at each timestep; it also linear interpolates for wind directions which fall between the pre-computed fetch distances (2° increments). The code then modifies the wave heights, calculated by Equations [7], [9], [10], and [21], according the physical processes: friction, shoaling, and breaking. The resulting wave height, after all processes are accounted for, is described by Equation [28]. To investigate the relative effect that each process may have on wave height, we look to compare the contribution of each process as a function of depth, which is a variable that all processes have in common.

An example of each physical process's effect on total wave height is shown in Figure 18.

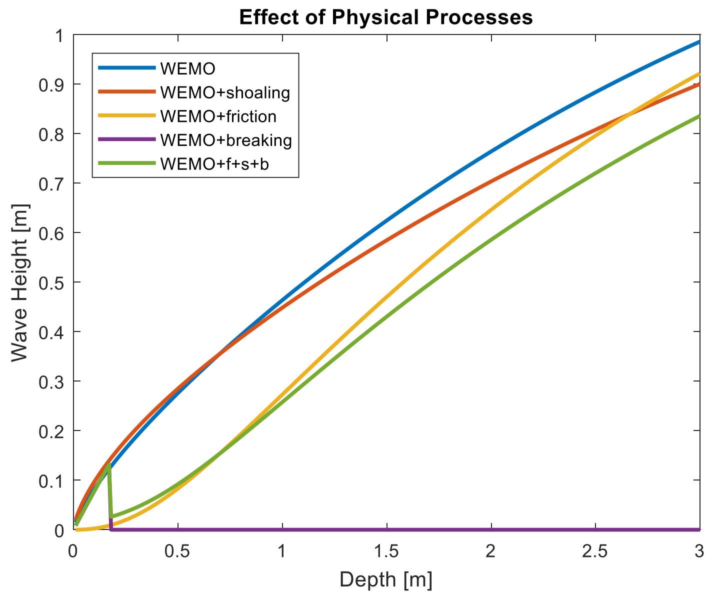


Figure 18 Effect of physical processes on wave height according to Equation [9] with wind speed of 30m/s and fetch distance of 5000m

Each parametric formulation's optimal combination of physical processes will be investigated in the following section.

3.3 Parametric Model Evaluation

Evaluations of the parametric system are based on a single point comparison to *in-situ* measured data and in comparison to the converged SWAN results over the entire domain.

3.3.1 Model Validation against ADCP

Both the parametric wave solver and SWAN are compared to field data collected by an acoustic Doppler current profiler, ADCP, deployed in the IRL for a 3-month period from January to March 2020. The ADCP, a SONTEK Argonaut-XR, was calibrated according to the specifications in (SONTEK, 2007). The instrument records pressure bursts every 3 hours with a sampling frequency, $f_s = 2$ Hz, Figure 19.

Figure 19 SONTEK Argonaut-XR pressure bursts

The ADCP uses pressure bursts to detect sea surface elevations. Significant wave height is computed from the surface elevation data by the SONWave-PRO software. Wave height can be reliably extracted from a pressure signal as long as the instrument depth and sampling frequency are sufficient for the wave field being measured ('SonWave-PRO : Directional Wave Data Collection', 2001). With a sampling frequency, $f_s = 2$ Hz, waves with frequencies lower than the Nyquist frequency of $0.5\ f_s$ are able to be resolved. The ADCP is therefore able to resolve waves with periods greater than 1 second.

The wave record obtained by the instrument needs to consist of waves long enough to meet this criterion to avoid aliasing. In order to ensure this criterion is met, a wind event was chosen in which relatively high wind speeds act along a fetch direction that allow for waves with periods greater than 1 sec to be produced at the instrument location. The wind event was selected during the period of data collection in which the wind speed and direction were relatively constant for a 10-hour period.

The wind data was obtained from the nearest NOAA station at Port Canaveral's Trident Pier, Figure 20.

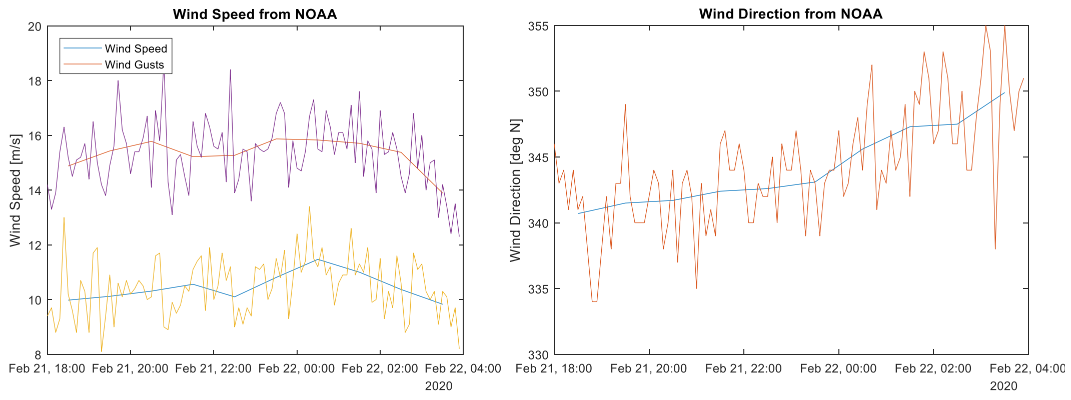


Figure 20 Wind speed and direction from the NOAA trident pier station from (2/21/20 at 18:00) to (2/22/20 at 4:00)

The average wind speed (10.5 m/s) and direction (344°) from NOAA for the 10-hour period was used to force the parametric and SWAN wave models. The model output will be compared to the average wave height measured by the ADCP (40.2 cm) over the same time interval (10 hours). An example of the wave energy distribution for the first pressure burst as well as the sensor location is shown in Figure 21.

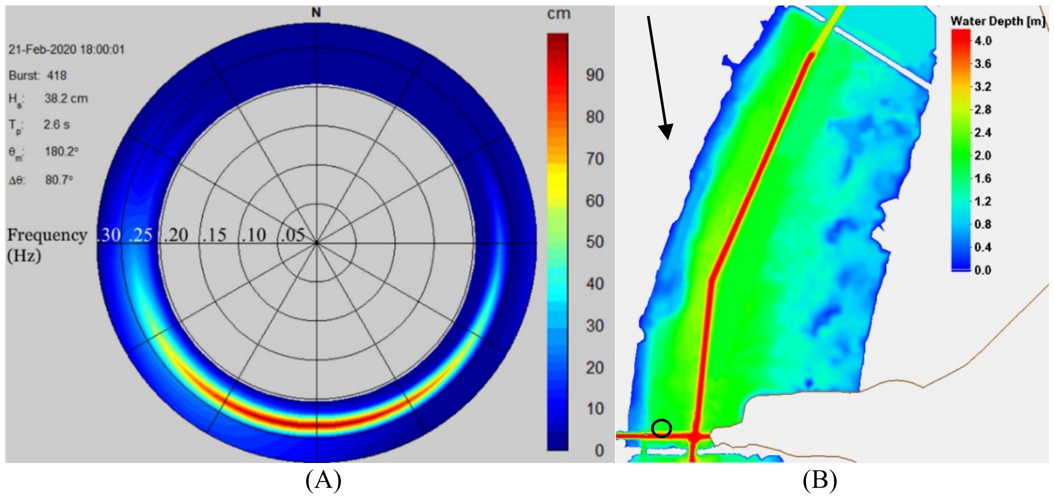


Figure 21 Directional wave energy distribution for burst 418 at 2/21/2020 18:00 (A) and sensor location (B). The Black arrow indicates the approximate wind direction (344°) and the black circle indicates the approximate sensor location (28° 24.5627' N, 80° 39.5741').

Figure 21(A) shows the measured direction of wave propagation which is due south; this corresponds to a wind direction of due north. Due to the westward location of the node, some shoaling is expected which may account for the slight deviation in expected wave direction indicated by the black arrow in Figure 21(B). The wave height solutions for both SWAN and the parametric model at the same location are shown in Figure 22 and Table 4.

3.3.2 Model Validation against SWAN

Currently, coupled wave circulation models rely on a third-generation wave model to produce the wave prediction. As long as the parametric wave solver produces a result comparable to a third-generation wave model in our domain, the parametric model may be used in place of the third-generation wave model to provide the information necessary for hydrodynamic coupling.

The significant wave heights calculated from Equations [7], [9], [10], and [21] are compared to the significant wave heights calculated by a fully converged non-stationary version of SWAN under the same wind forcing (SWAN model input in Appendix). Several different wind fields were created to test the performance of the parametric wave solver.

The creation of the first wind field tests the ability of the models to reproduce the depth and fetch limited fully developed conditions that occur with a wind of constant speed blowing in a constant direction, for a long duration. A 30 m/s wind speed simulates tropical storm-strength conditions. The IRL's longest fetch lengths and corresponding largest wave heights are associated with northerly winds, 0°; therefore, modeled winds are from this direction.

The second wind field was created to test the model response to varying wind speeds by having a wind blow from a constant direction (0° North) but vary in magnitude. The wind speed starts at 0 m/s, ramps up to 30 m/s, and then back down to 0 m/s according to the sine function.

The third wind field created tests the directional capability of the model by having a constant wind blowing at 30 m/s that starts from the north (0°) and rotates 10° clockwise every hour of the simulation until completing a full 360-degree rotation.

An investigation of the effect of incorporating the physical process of shoaling, breaking and friction, according to Equations [29]-[32], takes place for each of the parametric formulations (Equations [7], [9], [10], and [21]) and each of the three different wind fields.

The statistical metrics used to quantify the accuracy of the significant wave height results are as follows: mean absolute error (MAE), normalized mean absolute error (NMAE), and forecast time reduction (FTR). These metrics are defined in Equations [33], [34], and [35] respectively.

$$MAE = \frac{\sum_{i=1}^{n} \left| H_{s\,(Swan)i} - H_{s(Parametric)i} \right|}{n} \quad [33]$$

$$NMAE = 100\% * \frac{MAE}{H_{s(SWAN)}} \quad [34]$$

$$FTR = 100 * \frac{t_{SWAN} - t_{Parametric}}{t_{Parametric}} \quad [35]$$

where,

i = node number,
n = total number of nodes (126772 in the IRL domain),
H_s = significant wave height predicted by each model (SWAN and Parametric), and
t = total time of simulation.

Lastly, the model simulation wall-clock runtimes for both the parametric models and SWAN are compared for run time efficiency. The parametric models are all computed simultaneously in the same executable; as such, the runtime for the parametric model is reported as the total time to compute results for all 4 parametric equations.

4.0 Results

The performance of the four different parametric wave height formulations is investigated to find the model that performs best with respect to SWAN and *in-situ* ADCP data. In addition, the ensemble average between all four models is investigated to determine the validity of using this as the cumulative parametric solution.

4.1 Models vs. ADCP

The average wave heights computed from the ADCP data over the 10-hour time period is 40.2 cm (‘SonWave-PRO : Directional Wave Data Collection’, 2001). The wave height solutions for both SWAN and the parametric models at the same location are shown in Figure 22 and Table 4.

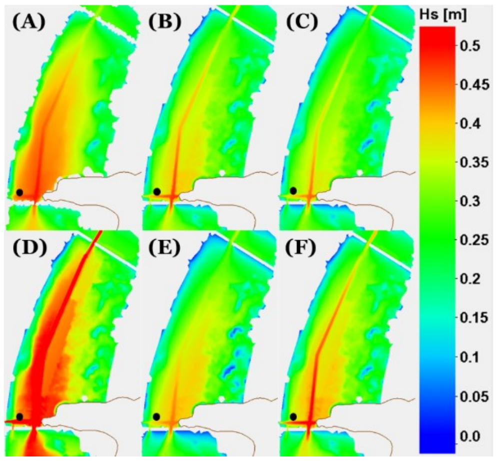


Figure 22 Wave height contours for SWAN (A) and parametric models SMB (B), SPM (C), TMA (D), CEM (E), and Ensemble (F) during ADCP comparison at the nearest node (black dot)

Table 4 Model comparison with respect to the average SONTEK sensor wave height. Model wave heights were computed with wind speed (10.5 m/s) at (344°) for the nearest node (35891) with a depth of 2.0 m. The (+/-) sign indicates an over/under prediction of the parametric model with respect to the SONTEK wave height.

	Measured	**3rd Gen.**	Parametric Models				Ensemble
	SONTEK	SWAN	SMB	SPM	TMA	CEM	Average
Hs [cm]	40.2	42.4	36.0	33.2	45.3	37.3	38.0
Difference [cm]		+2.2	-4.2	-7.0	+5.1	-2.9	-2.2
Difference [%]		+5.5	-10.4	-17.4	+12.7	-7.2	-5.5

4.2 Parametric vs. SWAN

First, the effect of incorporating the physical process of shoaling, breaking, and friction, according to Equations [29], [30], and [32], was investigated for each of the parametric formulations (Equations [7], [9], [10], and [21]) for each of the three different wind fields. Each of the parametric models' responses to the physical processes, presented as global averaged wave height, Figure 23, indicate the sensitivity of the formulations to friction, shoaling and breaking.

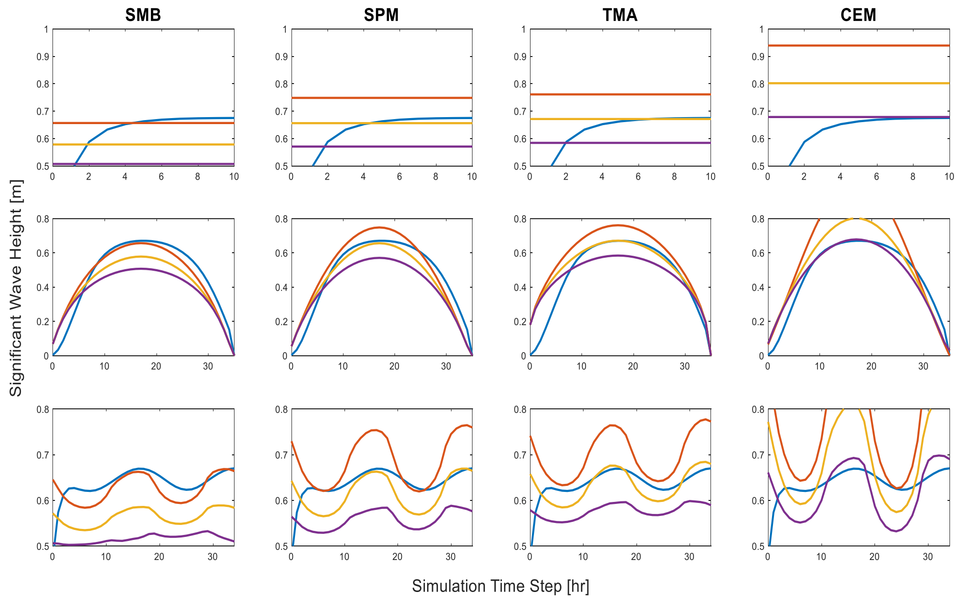


Figure 23 Globally averaged parametric results for wind simulations 1-3, where the top row corresponds to the constant wind simulation, the middle row corresponds to the varying magnitude wind simulation, and the bottom row corresponds to the varying direction wind simulation. The colors correspond to (blue) unstructured SWAN, (orange) parametric model with no physical processes, (yellow) parametric model with physical processes and Manning's n=0.01, and (purple) parametric model with physical processes and Manning's n=0.02.

Each respective model has an optimal configuration when comparing the performance against SWAN. For example, the SMB model (Equation [9]) performs best with no physical processes included, the SPM and TMA models (Equations [7] and [21]) perform best with physical processes included with a Manning's *n* value of 0.01 and the CEM model (Equation [10]) performs best with physical processes included with a manning's *n* value of 0.02. These realizations hold true for each of the three different simulations. The globally averaged results for each model under optimal physical forcing is shown in Figure 24.

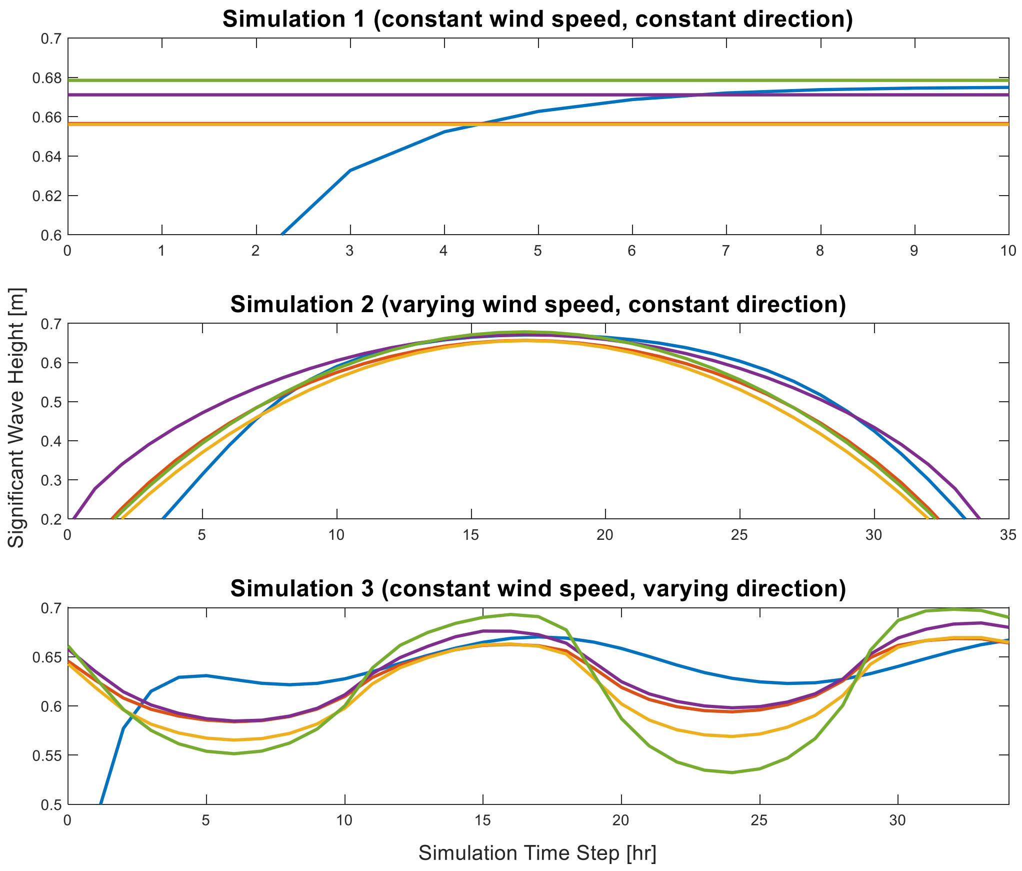


Figure 24 Globally averaged optimal parametric model results for simulations 1-3, where colors correspond to (blue) unstructured SWAN, (orange) SMB, (yellow) SPM, (purple) TMA, and (green) CEM (note, Y-axis is modified to show model deviations)

In addition, the model performance may be visualized throughout the domain. The various models' deviation with respect to SWAN is analyzed by percent deviations (over/under estimations) for the stationary wind simulation at every computational node, Figure 25.

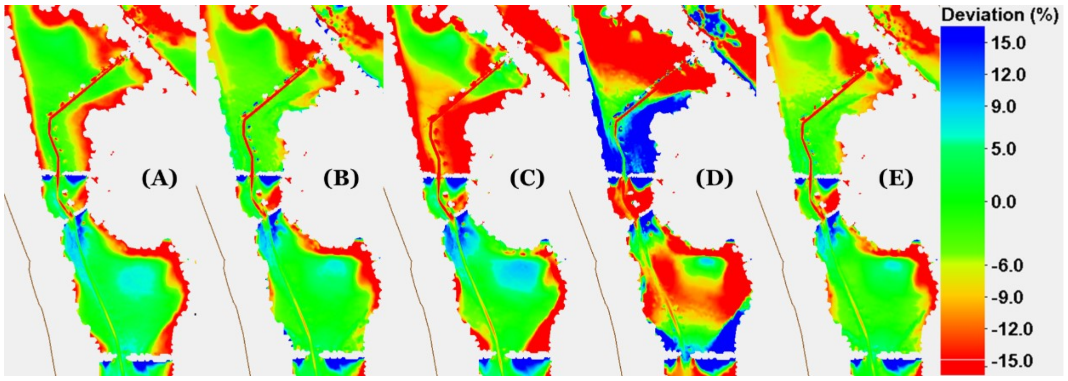


Figure 25 Model deviation contours for SMB (A), SPM (B), TMA (C), CEM (D), and ensemble average (E) where (-) deviation (red) indicates parametric model over-estimation, (+) deviation (blue) indicates parametric model under-estimation, and (+/- 5%) deviation (green) indicates model agreement with respect to SWAN for the stationary wind simulation (U=30m/s, θ=0° North).

The models vary in their agreement with SWAN. Qualitatively, the SMB solution appears to match SWAN the best, with the majority of solution agreement in the middle of the lagoon. In general, the models appear to be over-estimating along the edges of the lagoon where water depths are the smallest, and under-estimating directly behind the causeway where fetches are small. To investigate the areas where model deviation is highest, contour plots of the relative effect of friction and shoaling are analyzed, Figure 26.

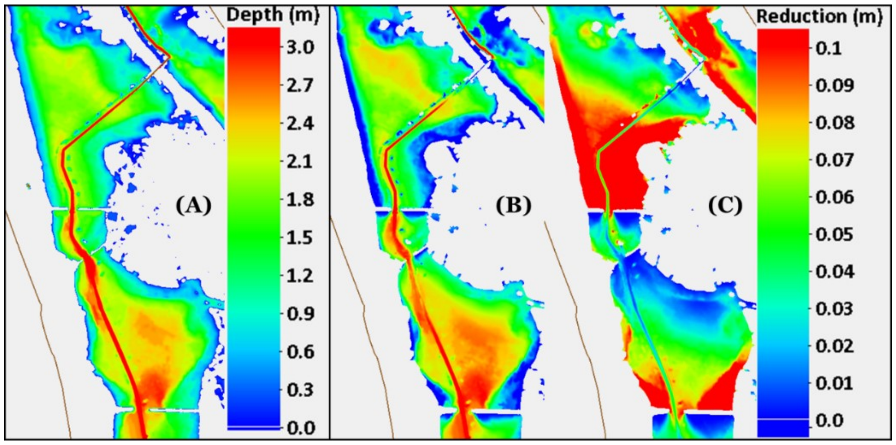


Figure 26 IRL water depth (A) and relative effects of shoaling (B) and friction (C)

Quantitatively, the wave height deviation between the parametric models and SWAN is computed with the metrics defined in Equations [33]-[35], respectively. Wave height deviation for simulations 1-3 are presented in Table 5.

Table 5 Parametric model performance

	MAE [cm]					N.MAE [%]				
Simulation	SMB	SPM	TMA	CEM	Ensemble Average	SMB	SPM	TMA	CEM	Ensemble Average
1	5.9	6.9	6.8	11.8	7.0	8.8	10.3	10.1	17.5	10.3
2	6.2	7.1	6.3	9.3	6.2	12.8	15.7	11.8	17.7	11.9
3	8.8	10.4	9.5	16.2	10.4	13.7	16.3	14.9	25.3	16.4
Average	7.0	8.1	7.5	12.4	7.9	11.8	14.1	12.3	20.2	12.9

In addition, the parametric models were compared against SWAN for run time efficiency. The parametric models are all computed simultaneously in the same executable; as such, the runtime for the parametric model is the total time to compute results for all 4 parametric models. The total wall clock time for each of the simulations is shown in Table 6.

Table 6 SWAN and parametric model run time performance. Simulation wall clock runtime (in minutes) is for a 1.5 day simulation.

		SWAN [min]		Parametric [min]		
Simulation	Run Length [hr]	Runtime	Time/day of sim.	Runtime	Time/day of sim.	FTR [%]
1	36	317.02	211.35	0.77	0.52	99.76
2	36	315.07	210.05	0.89	0.59	99.72
3	36	308.14	205.43	0.78	0.52	99.75
Average	36	313.41	208.94	0.82	0.54	99.74

5.0 Discussion

Each parametric formulation (Equations [7], [9], [10], and [21]) has an optimal setup with respect to the physical processes (Equations [29], [30], and [32] and Figure 23); each model's optimal set up is shown in the Appendix. Applying the optimal configurations found in this study, the parametric formulations yield comparable results with respect to SWAN for all wind conditions, Figure 24. It is unclear that the addition of these physical processes causes any consistent deviations between the parametric model and SWAN, Figure 25; more specifically, the areas where friction and shoaling caused the largest reduction in wave heights (red areas in Figure 26) do not correlate to the areas of wave height underestimation (blue areas in Figure 25). In addition, although wave breaking is accounted for, its effect is negligible in the IRL domain; for example, all models predicts less than 0.5% of nodes will experience wave breaking. This is largely due to the relatively deep water in which relatively small waves are predicted; however, in a larger and shallower model domain, the effect of wave breaking is expected to increase. A more comprehensive study of the effects that friction and shoaling have on the parametric wave height solution is required to diagnose the areas of highest deviation, Figure 25. More research is also required to determine the optimal physical process configurations (e.g. drag coefficient and Manning's *n*).

Additionally, for strong wind events, the parametric models' results provide a good representation of actual measured wave heights during a wind event as realized by the ADCP comparison. The computed wave heights from the pressure sensor were made by imposing a low-frequency cutoff of 0.2 Hz; this is because wave periods in this domain are limited by the depth and fetch. The most accurate parametric results with respect to the ADCP were obtained from the CEM formulation, which predicted wave height to within 2.9 cm, or 7.2% (Table 4) at the sensor location. The parametric models both underestimated (SMB, SPM, and CEM) and overestimated (TMA) the expected wave height. The measured wave height is within the four-member ensemble spread. If the ensemble average result is considered, the parametric solver predicted a wave height to within 2.2 cm, or 5.5%. The ensemble average produces a more accurate solution than any individual parametric model for this wind forcing and is equally as accurate as the SWAN solution at this location. In the future, the system can be optimized by evaluating and selecting the most accurate parametric models to include in an ensemble.

In addition, the parametric models performed well with respect to SWAN though differences do exist. For example, SWAN is run in nonstationary mode and is time dependent, resulting in a lag in solution wave height with respect to time. This lag is apparent in all 3 simulations: the ramping up required for the stationary wind simulation, the phase lag in the magnitude varying wind simulation, and the dampened signal in the directionally varying wind simulation, Figures 23 and 24.

According to Table 5, the models collectively performed best when compared to SWAN for the stationary case (10.3% ensemble NMAE), second for the magnitude varying wind (11.9% avg. ensemble NMAE) and third for the directional varying wind (16.4% avg. ensemble NMAE). It is not surprising that the parametric models perform best for conditions where wind speed and direction are fairly constant for an extended period of time; however, it was convincing that the models still perform considerably well with large changes in wind speed and direction as shown in simulations 2 and 3.

The SMB model performed the best (11.8% avg. NMAE), followed by the TMA model (12.3% avg. NMAE), then the SPM model (14.1% avg. NMAE) and finally the CEM model (20.2% avg. NMAE). The SMB model results for the stationary simulation (8.8% avg. NMAE) obtained the best performance compared to SWAN. In general, projections of the performance of the parametric models indicate global average accuracy within approximately 12.9% (total ensemble NMAE) with respect to SWAN. Although the ensemble average can produce enhanced performance, Table 4, it is too early to conclude that the ensemble approach is the optimal result of the parametric solver.

The statistical metrics (Equations [33]-[35]) in Table 5 were computed by excluding any statistical outliers; wave height values were not compared for time steps where the MAE values were outside of the simulation's inter quartile range (within 25% and 75% of the median). The reason for this is that the nonstationary SWAN solution has to ramp up from an initial state of zero until the completion of several computation time steps. Prior to model spin-up, SWAN solutions are unreliable; therefore, they are discarded.

According to Table 6, the parametric models are over two orders of magnitude faster than the nonstationary converged version of SWAN running in serial. On average, the parametric solver calculating each of the four parametric wave equations takes 48.8 seconds (0.8133 min.) for 1.5 days of simulation and SWAN takes 314.41 min. The parametric model is, on average, about 38000% faster than SWAN; i.e., one may perform 380 parametric simulations in the same amount of time it takes to perform one SWAN simulation; this is in agreement with the estimated 400 times improvement made by the back of the envelope estimation in Section 2.3. This corresponds to a model simulation time reduction of 99.74%. For coupled estuarine modeling applications (e.g. ADCRC+SWAN), the wave model and the circulation model generally share computational time equally. Implementing a parametric wave solver cuts the simulation runtime by at least a factor of two.

The parametric model does require a significant amount of pre-processing time, currently about 3.7 days to precompute the fetch and depth tables for a single domain. This pre-processing time is a function of grid resolution, number of nodes, and angular fetch resolution. Pre-processing needs to be performed only once for any particular estuarine domain. For direct comparison, SWAN was run in serial as opposed to parallel. SWAN may be run in parallel with a potentially large reduction in runtime corresponding to the number of processors available. However, to achieve a comparable runtime to the parametric model, SWAN would need to run in parallel on about 380 processors. This number of processors is not an issue for large research and government HPC resources, but for local efforts, these processor numbers are a limiting factor.

Lastly, it has been argued by (Liu, Schwab and Jensen, 2001) among many others, that the current, state-of-the-art wind-wave models are fundamentally limited. They have concluded the following:

> Therefore, upon laborious and conscientious deliberations, we would like to suggest that the present concept of the wind wave spectrum, which has been a central concept in wind wave studies for over five decades, has reached the limit of its usefulness as the basis for modeling wind wave processes. The application of wave spectrum analysis is an approach that was basically a recourse for convenience and expediency rather than for intrinsic and deterministic dynamical reasons.

Therefore, although the parametric model is relatively simple and omits many of the details employed by third-generation wind wave models, the results may be equally as valid. The exercise of comparing the parametric models to a third-generation wave model (SWAN) was done in the absence of comprehensive field measurements. As we continue to advance our understanding in the area of wave modeling, we will develop new formulations for wave height. If these formulations are able to be parameterized in terms of fetch, water depth, and wind speed, then the parametric solver can incorporate them and continue to be a useful tool in the coastal engineering community.

6.0 Conclusion

A parametric solver was created which incorporates four different wave height formulations (SMB, SPM, TMA, CEM) and can be used to create an ensemble average of parametric results. With validation from an ADCP deployed in the domain, the parametric models produced an average accuracy within 6% for wave height. This model also accurately simulates tropical storm wind events including variability in wind speed and direction with an average global accuracy of within 12.9% compared to SWAN. The parametric model runs much faster than SWAN, approximately 380 times faster. This reduction in wave model run times will increase the efficiency of an ensemble coupled model system such as the Multi-Stage (Taeb and Weaver, 2019), as well as any real time coupled prediction model. The parametric wave model is therefore a viable alternative to running an expensive third-generation wave model for predicting waves in an enclosed estuarine system.

7.0 Future Work

This study was a proof a concept for the applicability of a general parametric wave model. Many improvements can be made which will allow the solver to be faster, more accurate, and more user-friendly as well as operational in a coupled model system.

The first step towards increased efficiency is code optimization and parallelization which will provide for drastically reduced processing times.

Second, a more comprehensive study on optimal configuration of the fetch and depth averaging schemes (Equations [25] and [26]), the inverse distance weighting power parameter (Equation [27]), the additional physical processes (Equations [29], [30], and [32]), and Manning's *n* values, is required to increase the accuracy of the parametric solver. These topics of investigation warrant the need for more field data which is critical for further validation. In addition, the solver needs to be tested in different domains to see if the results are consistent with the IRL results presented here.

Third, as stated, the parametric model needs to be coupled with a circulation model in order for it to be used in storm-surge forecasting. The already available *Multistage* system (Taeb and Weaver, 2019) is the intended application for model coupling. The parametric solver needs to read in updated water levels throughout a simulation, calculate radiation stresses induced by the waves, and pass these radiation stresses back to the circulation model. Work on this topic is currently underway, and a functional parametric wave + circulation coupled model running inside of the nested *Multistage* domains will soon be available. This will allow for real-time ensemble storm-surge forecasting for the IRL and any other semi-enclosed environment. Once this is complete, the coupled model will need to be validated against the SWAN+ADCIRC coupled model, and any field data available, to compare run-times and simulation results.

8.0 References

Akbar, M. K., Kanjanda, S. and Musinguzi, A. (2017) 'Effect of Bottom Friction , Wind Drag Coefficient , and Meteorological Forcing in Hindcast of Hurricane Rita Storm Surge Using SWAN + ADCIRC Model', (2005). doi: 10.3390/jmse5030038.

Allard, R. *et al.* (2004) *Validation Test Report for the Simulating Waves Nearshore Model (SWAN): Cycle III , Version 40 . 11*, *Naval Research Laboratory*.

Benoit, M., Marcos, F. and Becq, F. (1997) 'TOMAWAC: A prediction model for offshore and nearshore storm waves', in *Proceedings, Congress of the International Association of Hydraulic Research, IAHR*.

Bhaskaran, P. K. *et al.* (2013) 'Performance and validation of a coupled parallel ADCIRC-SWAN model for THANE cyclone in the Bay of Bengal', *Environmental Fluid Mechanics*. doi: 10.1007/s10652-013-9284-5.

Blain, C. A., Cambazoglu, M. K. and Kourafalou, V. H. (2009) 'Modeling the Dardanelles Strait outflow plume using a coupled model system', in *MTS/IEEE Biloxi - Marine Technology for Our Future: Global and Local Challenges, OCEANS 2009*.

Bleck, R. (2002) 'An oceanic general circulation model framed in hybrid isopycnic-Cartesian coordinates', *Ocean Modelling*. doi: 10.1016/S1463-5003(01)00012-9.

Booij, N. (1994) 'Basic wave mechanics: for coastal and ocean engineers', *Dynamics of Atmospheres and Oceans*. doi: 10.1016/0377-0265(94)90016-7.

Booij, N., Ris, R. C. and Holthuijsen, L. H. (1999) 'A third-generation wave model for coastal regions 1. Model description and validation', *Journal of Geophysical Research: Oceans*, 104(C4), pp. 7649–7666. doi: 10.1029/98JC02622.

Bouws, E. *et al.* (1985) 'Similarity of the wind wave spectrum in finite depth water 1. Spectral form.', *Journal of Geophysical Research*, 90(C1), pp. 975–986. doi: 10.1029/JC090iC01p00975.

Bretschneider, C.L. (1952) 'Revised Wave Forecasting Relationships', in *Proceedings of Second Conference on Coastal Engineering*, pp. 1–5.

Bretschneider, C. L. (1952) 'The generation and decay of wind waves in deep water', *Eos, Transactions American Geophysical Union*. doi: 10.1029/TR033i003p00381.

Bretschneider, C. L. (1959) 'Wave variability and wave spectra for wind-generated gravity waves', *Beach Erosion Board, Tech. Memo. US Army Corps of Engineers,*.

Bretschneider, Charles L. (2011) 'MODIFICATION OF WAVE SPECTRA ON THE CONTINENTAL SHELF AND IN THE SURF ZONE', *Coastal Engineering Proceedings*. doi: 10.9753/icce.v8.2.

Bretschneider, C. L. (2011) 'Revisions in wave forecasting: deep and shallow water', *Coastal Engineering Proceedings*, 1(6), p. 3. doi: 10.9753/icce.v6.3.

Bretschneider, C. L. and Tamaye, E. E. (1977) 'Hurricane wind and wave forecasting techniques.', *Fifteenth Coastal Engng. Conf. (HAWAII UNIV.,U.S.A.:JUL.1117, 1976)*, 1, New Yo, pp. 202–237.

Bruno, D., De Serio, F. and Mossa, M. (2009) 'The FUNWAVE model application and its validation using laboratory data', *Coastal Engineering*. doi: 10.1016/j.coastaleng.2009.02.001.

Buckingham, E. (1914) 'On physically similar systems; Illustrations of the use of dimensional equations', *Physical Review*. doi: 10.1103/PhysRev.4.345.

Carniello, L. *et al.* (2005) ‘A combined wind wave-tidal model for the Venice lagoon, Italy’, *Journal of Geophysical Research: Earth Surface*. doi: 10.1029/2004JF000232.

CERC (1984) ‘Shore Protection Manual’, in *Shore Protection Manual*. Dept. of the Army, U.S. Army Corps of Engineers, Washington., pp. 1–208.

Chen, C. *et al.* (2013) ‘Extratropical storm inundation testbed: Intermodel comparisons in Scituate, Massachusetts’, *Journal of Geophysical Research: Oceans*. doi: 10.1002/jgrc.20397.

Chen, C. *et al.* (2016) ‘Circulation in the Arctic Ocean: Results from a high-resolution coupled ice-sea nested Global-FVCOM and Arctic-FVCOM system’, *Progress in Oceanography*. doi: 10.1016/j.pocean.2015.12.002.

Chen, C., Liu, H. and Beardsley, R. C. (2003) ‘An unstructured grid, finite-volume, three-dimensional, primitive equations ocean model: Application to coastal ocean and estuaries’, *Journal of Atmospheric and Oceanic Technology*. doi: 10.1175/1520-0426(2003)020<0159:AUGFVT>2.0.CO;2.

Dalrymple, R. A. and Rogers, B. D. (2006) ‘Numerical modeling of water waves with the SPH method’, *Coastal Engineering*. doi: 10.1016/j.coastaleng.2005.10.004.

Dean, R. G. and Dalrymple, R. A. (1984) ‘Water wave mechanics for engineers and scientists.’ Prentice-Hall Inc.

DHI Software (2017) ‘Mike 21 Boussinesq Wave Module Scientific Documentation’, *DHI Water & Environment, Hørsholm, Denmark*. Available at: http://www.cnki.net/KCMS/detail/detail.aspx?FileName=FSJS201403033&DbName=CJFQ2014.

Dietrich, J. C. *et al.* (2010) ‘A High-Resolution coupled riverine flow, tide, wind, wind wave, and storm surge model for southern louisiana and mississippi. Part II: Synoptic description and analysis of hurricanes katrina and rita’, *Monthly Weather Review*. doi: 10.1175/2009MWR2907.1.

Dietrich, J. C. *et al.* (2012) ‘Performance of the unstructured-mesh, SWAN+ ADCIRC model in computing hurricane waves and surge’, *Journal of Scientific Computing*. doi: 10.1007/s10915-011-9555-6.

Fontaine, E. (2013) ‘A theoretical explanation of the fetch-and duration-limited laws’, *Journal of Physical Oceanography*, 43(2), pp. 233–247. doi: 10.1175/JPO-D-11-0190.1.

Garzon, J. L. and Ferreira, C. M. (2016) ‘Storm surge modeling in large estuaries: Sensitivity analyses to parameters and physical processes in the Chesapeake Bay’, *Journal of Marine Science and Engineering*. doi: 10.3390/jmse4030045.

Goda, Y. (2010) ‘Random seas and design of maritime structures’, in *Advanced Series on Ocean Engineering*, pp. 1–732.

Gruijthuijsen, M. F. Van (1996) *Validation of the wave prediction model SWAN using field data from Lake George , Australia*. Delft University of Technology.

Harris, L. M. and Durran, D. R. (2010) ‘An Idealized Comparison of One-Way and Two-Way Grid Nesting’, *Monthly Weather Review*. doi: 10.1175/2010MWR3080.1.

Hasselmann, K. *et al.* (1973) ‘Measurements of wind-wave growth and swell decay during the joint North Sea wave project (JONSWAP).’

Hasselmann, K. *et al.* (1976) ‘A Parametric Wave Prediction Model’, *Journal of Physical Oceanography*, 6(2), pp. 200–228. doi: 10.1175/1520-0485(1976)006<0200:apwpm>2.0.co;2.

Hasselmann, K. *et al.* (1988) 'The WAM model - a third generation ocean wave prediction model.', *J. Phys. Oceanography*.

Hasselmann, S. *et al.* (1985) 'Computations and parameterizations of the nonlinear energy transfer in a gravity-wave spectrum. Part II: parameterizations of the nonlinear energy transfer for application in wave models.', *J. PHYS. OCEANOGR.* doi: 10.1175/1520-0485(1985)015<1378:capotn>2.0.co;2.

Hughes, S. A. (1984) *TMA shallow-water spectrum: description and application*, *Technical Report CERC (US Army Engineer Waterways Experiment Station Coastal Engineering Research*.

Hwang, P. A. (2006) 'Duration- and fetch-limited growth functions of wind-generated waves parameterized with three different scaling wind velocities', *Journal of Geophysical Research: Oceans*. Blackwell Publishing Ltd, 111(2). doi: 10.1029/2005JC003180.

Hwang, P. A. and Wang, D. W. (2004) 'Field measurements duration-limited growth of wind-generated ocean surface waves at young stage of development', *Journal of Physical Oceanography*. doi: 10.1175/1520-0485(2004)034<2316:FMODGO>2.0.CO;2.

Janssen, P. A. E. M. (1989) 'Wave-Induced Stress and the Drag of Air Flow over Sea Waves', *Journal of Physical Oceanography*. doi: 10.1175/1520-0485(1989)019<0745:wisatd>2.0.co;2.

Janssen, P. A. E. M. (1991) 'Quasi-linear theory of wind-wave generation applied to wave forecasting', *J. Physical Oceanography*. doi: 10.1175/1520-0485(1991)021<1631:QLTOWW>2.0.CO;2.

Jelesnianski, C. P. *et al.* (1984) 'SLOSH - A hurricane storm surge forecast model', in *Oceans Conference Record (IEEE)*. doi: 10.1109/oceans.1984.1152341.

Ji, M., Aikman, F. and Lozano, C. (2010) ‘Toward improved operational surge and inundation forecasts and coastal warnings’, *Natural Hazards*. doi: 10.1007/s11069-009-9414-z.

Jiang, M. (2017) *Modeling ecosystem dynamics in the Indian River Lagoon and assessing the potential impacts of climate change*. Harbor Branch Oceanographic Institute.

Johnson, J. W. (1950) ‘Relationships between wind and waves, Abbotts Lagoon, California’, *Eos, Transactions American Geophysical Union*. doi: 10.1029/TR031i003p00386.

Kerr, P. C. *et al.* (2013) ‘U.S. IOOS coastal and ocean modeling testbed: Inter-model evaluation of tides, waves, and hurricane surge in the Gulf of Mexico’, *Journal of Geophysical Research: Oceans*. doi: 10.1002/jgrc.20376.

Kerr, P C *et al.* (2013) ‘U . S . IOOS coastal and ocean modeling testbed : Inter-model evaluation of tides , waves , and hurricane surge in the Gulf of Mexico’, 118, pp. 5129–5172. doi: 10.1002/jgrc.20376.

Kirby, J. T. *et al.* (1998) ‘Funwave 1.0, fully nonlinear boussinesq wave model documentation and user’s manual’, *Center for Applied …*.

Li, Z. and Gao, J. (2014) ‘Intelligent optimization on power values for inverse distance weighting’, in *Proceedings - 2013 International Conference on Information Science and Cloud Computing Companion, ISCC-C 2013*. doi: 10.1109/ISCC-C.2013.81.

Liu, P. C., Schwab, D. J. and Jensen, R. E. (2001) ‘Has wind-wave modeling reached its limit?’, *Ocean Engineering*, 29(1), pp. 81–98. doi: 10.1016/S0029-8018(00)00074-3.

Longuet-Higgins, M. S. (1952) ‘On the Statistical Distribution of the Heights of Sea Waves’, *Journal of Marine Research*.

Longuet-Higgins, M. S. and Stewart, R. w. (1964) ‘Radiation stresses in water waves; a physical discussion, with applications’, *Deep-Sea Research and Oceanographic Abstracts*. doi: 10.1016/0011-7471(64)90001-4.

Lu, G. Y. and Wong, D. W. (2008) ‘An adaptive inverse-distance weighting spatial interpolation technique’, *Computers and Geosciences*. doi: 10.1016/j.cageo.2007.07.010.

Luettich, R. A. and Westerink, J. J. (1991) ‘A solution for the vertical variation of stress, rather than velocity, in a three-dimensional circulation model’, *International Journal for Numerical Methods in Fluids*. doi: 10.1002/fld.1650121002.

Luettich, R. A., Westerink, J. J. and Scheffner, N. W. (1992) *ADCIRC: An advanced three-dimensional circilational model for shelves, coasts, and estuaries*. Washington, DC 20314-1000.

Malhotra, A. and Fonseca, M. S. (2007) *WEMo (Wave Exposure Model): Formulation, Procedures and Validation. NOAA Technocal Memorandum NOS NCCOS #65. 28p.*

Massey, T. C. *et al.* (2011) *STWAVE : Steady-State Spectral Wave Model User ’ s Manual for STWAVE , Version 6.0*, *Erdc/ChL Sr-11-1*.

May, J. P. (2006) ‘Shoaling coefficient’, in *Beaches and Coastal Geology*. Schwartz. doi: 10.1007/0-387-30843-1_413.

Mei, G., Xu, N. and Xu, L. (2016) ‘Improving GPU-accelerated adaptive IDW interpolation algorithm using fast kNN search’, *SpringerPlus*. doi: 10.1186/s40064-016-3035-2.

Mitsuyasu, H. (1970) ‘On the Growth of the Spectrum of Wind-Generated Waves’, *Coastal Engineering in Japan*. doi: 10.1080/05785634.1970.11924105.

Mitsuyasu, H. (1982) ‘Direction spectra of ocean waves in generation area’, in *ASCE.*, pp. 87–102.

Narayanaswamy, M. *et al.* (2010) ‘SPHysics-FUNWAVE hybrid model for coastal wave propagation’, *Journal of Hydraulic Research*. doi: 10.1080/00221686.2010.9641249.

Neumann, G. and Pierson, W. J. (1957) ‘A detailed comparison of theoretical wave spectra and wave forecasting methods’, *Deutsche Hydrographische Zeitschrift*. doi: 10.1007/BF02020059.

Ochi, M. K. (1998) *Ocean waves: the stochastic approach*. Ocean Tech, *Cambridge University Press*. Ocean Tech. Press Syndicate of the University of Cambridge.

Ochi, M. K. and Hubble, E. N. (1977) ‘Six-parameter wave spectra.’, *Fifteenth Coastal Engng. Conf. (HAWAII UNIV. U.S.A.: JUL.11-17,1976)*.

OpenFOAM (2014) ‘Open∇FOAM - The Open Source CFD Toolbox - User Guide’, *OpenFOAM Foundation*. doi: 10.1023/A.

Padilla-Hernández, R. *et al.* (2007) ‘Modeling of two northwest Atlantic storms with third-generation wave models’, *Weather and Forecasting*. doi: 10.1175/2007WAF2005104.1.

Padilla-Hernandez, R., Perrie, W. and Toulany, B. (2004) ‘An intercomparison of modern operational wave models’, *Workshop on Wave*.

Phillips, O. M. (1958) ‘The equilibrium range in the spectrum of wind-generated waves’, *Journal of Fluid Mechanics*. doi: 10.1017/S0022112058000550.

Pierson, W. J. and Moskowitz, L. (1963) *A proposed spectral form for fully developed wind seas based on the similarity theory*. New York, NY.

Pierson, W. J. and Moskowitz, L. (1964) ‘A proposed spectral form for fully developed wind seas based on the similarity theory of S. A. Kitaigorodskii’, *Journal of Geophysical Research*. doi: 10.1029/jz069i024p05181.

Putnam, J. A. and Johson, J. W. (1949) ‘The dissipation of wave energy by bottom friction’, *Eos, Transactions American Geophysical Union*. doi: 10.1029/TR030i001p00067.

Resio, D. and Perrie, W. (1991) ‘A numerical study of nonlinear energy fluxes due to wave-wave interactions Part 1. Methodology and basic results’, *Journal of Fluid Mechanics*. doi: 10.1017/S002211209100157X.

Resio, D. T. and Vincent. C. Linwood (1977) ‘Estimation of winds over the Great Lakes’, *J Waterw Port Coastal Ocean Div Proc ASCE*.

Resio, D. T. and Westerink, J. J. (2008) ‘Modeling the physics of storm surges’, *Physics Today*. doi: 10.1063/1.2982120.

Ris, R. C., Holthuijsen, L. H. and Booij, N. (1999) ‘A third-generation wave model for coastal regions: 2. Verification’, *Journal of Geophysical Research: Oceans*. American Geophysical Union (AGU), 104(C4), pp. 7667–7681. doi: 10.1029/1998jc900123.

Roll, H. U. (1957) ‘Forecasting Surface Gravity Waves: Conference on Long Waves and Storm Surges’, in. Wormley, England: National Institute of Oceanography.

Sánchez-arcilla, A. and Lemos, C. M. (1990) *Surf-zone Hydrodynamics*, *Centro Internacional de Métodos Numéricos de Ingeniería*. Pineridge Press.

Sartini, L., Mentaschi, L. and Besio, G. (2015) ‘Evaluating third generation wave spectral models performances in coastal areas. An application to Eastern Liguria’, in *MTS/IEEE OCEANS 2015 - Genova: Discovering Sustainable Ocean Energy for a New World*. doi: 10.1109/OCEANS-Genova.2015.7271395.

Saville, T. (1954) *The effect of fetch width on wave generation*, *Beach Erosion Board*.

Shepart, D. (1968) ‘Two- dimensional interpolation function for irregularly- spaced data’, in *Proc 23rd Nat Conf*.

Smith, J. M. (1991) *Wind-Wave Generation on Restricted Fetches*. Vicksburg, Mississippi. doi: 10.1107/s0365110x55001916.

Smith, M. (2001) ‘Modeling Nearshore Wave Transformation with STWAVE’, *Development*.

SONTEK (2007) ‘Argonaut SL/SW/XR User’s Manual’. San Diego, CA: YSI inc.

‘SonWave-PRO : Directional Wave Data Collection’ (2001). San Diego, CA: SONTEK/Xylem Inc., pp. 1–12.

Sverdrup, H. U. and Munk, W. H. (1947) ‘Wind, Sea, and Swell. Theory of Relations For FOrecasting’, *Office*.

SWAN (2014) ‘Swan User Manual’, *SWAN Cycle III version 41.01*.

Taeb, P. and Weaver, R. J. (2019) ‘An operational coastal forecasting tool for performing ensemble modeling’, *Estuarine, Coastal and Shelf Science*. Academic Press, 217, pp. 237–249. doi: 10.1016/j.ecss.2018.09.020.

Tezduyar, T. *et al.* (1996) ‘Flow simulation and high performance computing’, *Computational Mechanics*, 18(6), pp. 397–412. doi: 10.1007/BF00350249.

Toba, Y. (1972) ‘Local balance in the air-sea boundary processes - I. on the growth process of wind waves’, *Journal of the Oceanographical Society of Japan*. doi: 10.1007/BF02109772.

Toba, Y. (1973) ‘Local balance in the air-sea boundary processes - III. On the spectrum of wind waves’, *Journal of the Oceanographical Society of Japan*. doi: 10.1007/BF02108528.

Tolman, H. L. (1991) ‘A Third-Generation Model for Wind Waves on Slowly Varying, Unsteady, and Inhomogeneous Depths and Currents’, *Journal of Physical Oceanography*. doi: 10.1175/1520-0485(1991)021<0782:atgmfw>2.0.co;2.

Tolman, H. L. (2010) ‘Practical Wind Wave Modeling’, in. doi: 10.1142/9789814304245_0006.

Tracy, B. A. and Resio, D. T. (1982) ‘Theory and calculation of the nonlinear energy transfer between sea waves in deep water.’

U.S. Army Corps of Engineers, D. of the A. (2002) *Coastal Engineering Manual*, *Coastal Engineering Manual*. doi: 10.1093/intimm/dxs026.

Voller, V. R. and Porté-Agel, F. (2002) ‘Moore’s law and numerical modeling’, *Journal of Computational Physics*. doi: 10.1006/jcph.2002.7083.

Weaver, R. J., Hartegan, D. and Massey, C. (2018) *USCRP Technology Challenge : Development and coupling of a parametric wave model to improve efficiency of high- resolution storm surge modeling allowing an ensemble approach using ADCIRC USACE BAA CHL-15 WEAVER USACE BAA CHL-15 USCRP Technology Challenge*.

Weaver, R. J. and Slinn, D. N. (2005) ‘Effect of wave forcing on storm surge’, in *Proceedings of the Coastal Engineering Conference*. doi: 10.1142/9789812701916-0122.

Weaver, R. J. and Slinn, D. N. (2007) ‘Real-time and probabilistic forecasting system for waves and surge in tropical cyclones’, in *Proceedings of the Coastal Engineering Conference*. doi: 10.1142/9789812709554_0114.

Webb, D. J. (1978) ‘Non-linear transfers between sea waves’, *Deep-Sea Research*. doi: 10.1016/0146-6291(78)90593-3.

Westerink, J. *et al.* (2018) ‘The Evolution of Process and Scale Coupling in Coastal Ocean Hydrodynamic Modeling The hydrodynamics of the coastal ocean and floodplain Understanding coastal sustainability and risk means understanding’, in.

Westerink, J. J. *et al.* (1992) ‘Tide and storm surge predictions using finite element model’, *Journal of Hydraulic Engineering*, 118(10), pp. 1373–1390. doi: 10.1061/(ASCE)0733-9429(1992)118:10(1373).

Westerink, J. J. *et al.* (2008) ‘A basin- to channel-scale unstructured grid hurricane storm surge model applied to southern Louisiana’, *Monthly Weather Review*, 136(3), pp. 833–864. doi: 10.1175/2007MWR1946.1.

Wilson, B. W. (1955) *Graphical approach to the forecasting of waves in moving fetches*.

Wood, D. J., Muttray, M. and Oumeraci, H. (2001) ‘The SWAN model used to study wave evolution in a flume’, *Ocean Engineering*. doi: 10.1016/S0029-8018(00)00033-0.

Yin, J. *et al.* (2020) ‘Response of Storm-Related Extreme Sea Level along the U.S. Atlantic Coast to Combined Weather and Climate Forcing’, *Journal of Climate*. doi: 10.1175/jcli-d-19-0551.1.

Zhang, Y. and Baptista, A. M. (2008) ‘SELFE: A semi-implicit Eulerian-Lagrangian finite-element model for cross-scale ocean circulation’, *Ocean Modelling*. doi: 10.1016/j.ocemod.2007.11.005.

9.0 Appendix A

Table 7 Parametric model set up

	SMB	SPM	TMA	CEM
Breaking	NO	YES	YES	YES
Shoaling	NO	YES	YES	YES
Friction	NO	YES	YES	YES
Manning's n	none	0.01	0.01	0.02
Eff. Fetch θ	40°	40°	40°	40°
IDW p- value	5.0	5.0	5.0	5.0

Table 8 SWAN model set up

Mode	Nonstationary	Breaking	default
	Two dimensional	Friction	JONSWAP (default)
Coordinates	Spherical (CCM)		constant, CF=0.38
Cgrid	Unstructured (circle)	QUAD	default
	MDC=36	TRIAD	default
	FLOW=0.031384	Refraction	Off
	MSC=45	Propagation	BSBT
Readgrid	Unstructured	Numerics	STOPC
	ADCIRC		DABS (0.005)
Inpgrid	Wind		DREL (0.01)
	Unstructured		CURVAT (0.005)
Readinp	Wind		NPNTS (95)
Generation	3		MXITNS (50)
Wave growth	KOMEN, AGROW	TEST	1,0
WCAP	KOMEN (default)	COMPUTE	NONSTAT 600 sec

9.1 Appendix B

The following two codes “pre_proc.cpp” and “param_wave.cpp” are the pre-processing and operational parts of the parametric wave solver that were used to obtain the results for this research. There are several other supplemental codes, including the user instructions, that were also used in this research which are hosted at: https://github.com/sboyd2014/PARAM_wave_solver. The user instructions, pre-processing code, and operational parametric wave code are included below.

User instructions: Read me file

User Instructions for Parametric Wave Solver

By: Samuel C. Boyd, and Robert J. Weaver

Date: 6/15/2020

Institution: Florida Institute of Technology

Department: Ocean Engineering and Marine Sciences

Reference: ECSS paper and thesis titled 'Improving Efficiency of Coupled Hydrodynamic Predictions by Implementing a Fetch-based Parametric Wave Model'

****

The workflow of the codes is as follows

files inside parenthesis are input/output to the codes with executables that are not in parenthesis. Each of the .cpp files shown in the workflow are explained in detail below.

```
                              PARAMETRIC CODE                                                                 SWAN
____________________________________________________________________________________________________________________

                                                                                                           (fort.14)
                                                                                                               |
                             (fort.14)                              wind_gen.cpp                      (lat_lon_swan.txt)
                                 |                                        |                                    |
                           pre_proc.cpp                               (fort.22)---------------------------> (INPUT)
                                 |                                        |                                    |
                                 |                                        |                                    |
avg_modify.cpp <------- (Distance_data.txt)                               |                                    |
        |          \----- (Depth_data.txt)                                |                                    |
        |                        +                                        |                                swan.exe
        |-------------------> (Slope_upwind.txt) ----------\              |                                    |
        |-------------------> (IDW_depths.txt) -------------\             |                                    |
        |-------------------> (Eff_fetch.txt) ------------------->  param_wave.cpp                             |
                                                                          |                                    |
                                                                   (SMB_Hs.63)------->  DS_v7.cpp    <---(swan_Hs.63)
                                                                   (SPM_Hs.63)-----/       |
                                                                   (TMA_Hs.63)---/         |
                                                                   (CEM_Hs.63)--/      (results.txt)
                                                                   (ENS_Hs.63)--/      (results.xls)
```

****

The pre-processing part of the solver is titled "pre_proc.cpp"

Function:

This code calculates fetch distances and depth values for all nodes in a given domain. The user inputs a distance value for the stepping process (should be on the same order as the smallest element in the domain). The code then calculates fetch distances along straight-line fetch rays at angular headings with spacing defined by the user. Once all fetches are calculated, the cosine weighted effective fetch is computed according to the user's angular averaging distance considered. Inverse distance weighted depths are also calculated along straight-line fetch rays according to the IDW p-value specified by the user (5.0 was used for the IRL domain). Bottom slope is computed from a wet node to 1 step upwind of that node at each angular heading. The final output is 3 files (IDW_depths_2_deg.txt, Eff_fetch_2_deg.txt, Slope_upwind_2_deg.txt) that are later read in by the operational part of the solver

Steps:

1) Code requires a grid file in the same directory as the code. The grid file needs to be in ADCIRC fort.14 format and named 'fort.14' OR the infile.open("fort.14"); line needs to be renamed according to the grid's file name. The following files on lines 74-80:

"Distance_data_2_deg.txt"
"Depth_data_2_deg.txt"
"Slope_upwind_2_deg.txt"
"p_values_2_deg.txt"
"IDW_depths_2_deg.txt"
"Eff_fetch_2_deg.txt"
"Raw_depth_2_deg.txt"

define the names of the files that the code will generate upon execution. The file contents, formats, and variables are defined in the Appendix section (bottom of this file).

2) Code requires an increase in stack size to allow for local array declarations. The command is:

```
$ ulimit -s 1000000
```

The exact stack size required may depend on the domain and is also limited by the machine. Try the largest ulimit the machine will allow and see if a segmentation fault (core dump) still occurs. This size works for the IRL domain which has 126772 nodes. Future code improvement would be to reformat the variable arrays so that they are dynamically allocated, and a large stack size is not required.

3) Code needs to be complied in the terminal with

```
$ g++ pre_proc.cpp -o pre_proc
```

before execution.

4) It is executed with command

```
$ ./pre_proc
```

and node numbers will be outputted on the terminal as the stepping process is running. This code takes 3.7 days to run for the IRL domain with 126772 nodes, 2 degree angular resolution, 40 degree cosine weighted average value, 50m bathymetric resolution step size, and constant IDW p-value of 5.0.

****

The operational part of the parametric solver is titled "param_wave.cpp"

Function:

This code computes wave heights based on 4 different parametric formulations (SMB, SPM, TMA, and CEM) in addition to the ensemble average. The wave heights are calculated from the fetch and depth data created by the pre-processing part of the code and are then modified according to the physical processes of friction, shoaling, and breaking. The results are files containing the final wave heights in ADCIRC fort.63 format.

Steps:

1) This code reads in the pre-processed data for fetch, depth, and bottom slope created by the pre-processing part of the solver titled "pre_proc.cpp". These files are titled:

"Eff_fetch_2_deg.txt"	line: 189 and 246
"IDW_depths_2_deg.txt"	line: 276
"Slope_upwind_2_deg.txt"	line: 368

and must be placed in the same directory as the operational "param_wave.cpp" code. In addition, these files must be referenced by their names properly on the lines on which they are called (shown above).

2) A wind file is required to force the solver and it must be in ADCIRC fort.22 (NWS=5) format and must be placed in the same directory as "param_wave.cpp"

There is a code provided which can create 3 different wind fields. The wind field options are:

1) constant wind speed constant direction (input #0)
2) variable wind speed constant direction (input #2)
3) variable wind speed variable direction (input #1)

The wind field type as well as simulation length will be entered in the terminal window upon wind code execution. The code is called "wind_gen2.cpp" and needs to be compiled and executed as mentioned above. SWAN is in Meteorological Convention while ADCIRC and PARAM are in Oceanographic convention. The wind code will format the fort.22 file accordingly. Currently all wind fields are with wind from the North (for theta wind #1 the wind starts from the north and rotates clockwise according to the angular resolution specified).

The wind field is called at lines 324 and 440 and the file name needs to match this name. Code linearly interpolates wave heights if the wind direction provided falls in between the angular resolution specified.

The results for each of the 4 parametric formulations, as well as the ensemble average between all 4 models are written to the files:

"WEMO_Hcum_const_wind.63"	line 407
"TMA_Hcum_const_wind.63"	line 413
"CEM_Hcum_const_wind.63"	line 419
"SPM_Hcum_const_wind.63"	line 425
"ENS_Hcum_const_wind.63"	line 431

in ADCIRC fort.63 file format. The results may be visualized in SMS.

For reproducibility of the results the wind files that were generated for the results of the study are also provided. These are labeled "PARAM_const_U_30.22" "PARAM_sin_U_30.22" and "PARAM_theta_U_30.22" and similarly "SWAN_const_U_30.22" "SWAN_sin_U_30.22" and "SWAN_theta_U_30.22"

****

In the case that the user wants to change the inverse distance weighting function's p-value or effective fetch cosine angular value without re-running the pre-processing part of the code, "avg_modify.cpp" may be executed and parameters may be changed.

Steps:

1) fort.14 grid file must be in the same directory as the code executable in order to read total number of nodes.

2) The files generated by the pre-processing code: "Distance_data_2_deg.txt" and "Depth_data_2_deg.txt" need to be in the same directory also.

3) The new files generated upon completion are called "IDW_depth_MOD.txt" "Eff_fetch_MOD.txt" and "Slope_upwind_MOD.txt". These modified names need to be adjusted within the "param_wave.cpp" before it can read these files OR rename the files to their original names and move them to the "param_wave.cpp" directory before execution.

****

For comparing the wave height results of param_wave.cpp to the wave heights generated by SWAN for the same wind forcing, the code 'DS_v7.cpp' is provided. This code needs to be compiled and executed as mentioned above.

Steps:

1) This code requires the SWAN wave height file (generated by the table output requested in the SWAN INPUT file) in the same directory as the executable. The SWAN INPUT file that was used to generate the results is also provided for reference. This file will be converted to fort.63 file format upon code execution. The raw SWAN file needs to be accurately named on line 60 of DS_v7.cpp. The fort.63 file that will be created is named on line 63. This file is called again on line 122, so ensure that the names match within the code.

2) The code also requires the parametric wave height solution (fort.63 file generated by param_wave.cpp) in the same directory as the executable this is called on line 119.

Note: only 1 wave height solution (SMB,SPM,TMA,CEM, or ENS) may be compared to SWAN per code execution. The parametric files need to be renamed within the code to the appropriate name of the model solution of interest to be compared to SWAN (rename file on line 119 for comparison).

3) The results of the solution comparison can be seen for every timestep, as well as the total simulation statistics, in the results.txt file named on line 124. These results are also formatted in an XLS, or .csv file format in the file named on line 126.

****

Appendix

Variable Definitions:

N_tot:	total number of nodes in the computational domain (fort.14 file)
Ang_res:	user-specified angular resolution of the stepping process (in degrees)
Num_angs:	total number of angular increments (360/Ang_res) or number of fetch rays per node
num_steps:	number of steps from wet node to d_num_steps (varies per node and per direction)
step_res:	user-specified step distance [m]
d_num_steps:	depth value along straight-line fetch ray right before land is reached
n1,n2,n3... :	node numbers
F1,F2,F3... :	fetch values [m]
d1,d2,d3... :	depth values [m] along fetch rays
dir_1,dir_2,dir_3:	direction number (1 to Ang_res) per node
d_IDW1,d_IDW2,d_IDW3:	inverse distance weighted depth according to p-value function

File Formats:

"Distance_data_2_deg.txt"

```
Fetch Distances at spatial res: Step_res[m] and angular res: Ang_res[deg]
Grid used to generate file: 'fort.14'
n1  F1 F2 F3 ... F(Num_Angs)
n2
n3
.
.
.
N_tot  F1 F2 F3 ... F(Num_Angs)
```

notes: if a node is dry then the fetches will not be written. This file represents the raw fetch rays generated by the stepping process that will be read in by pre_proc.cpp at line (1462) which will then be cosine averaged according to the users Eff_res value.

"Depth_data_2_deg.txt"

```
Water Depths along fetch rays from 'fort.14'
N_tot Ang_res
n1 dir_1 num_steps
d1
d2
d3
...
d_num_steps
n1 dir_2 num_steps
d1
d2
d3
...
d_num_steps

.
.
.

n2 dir_1 num_steps
d1
d2
d3
...
d_num_steps
n2 dir_2 num_steps
d1
d2
d3
.
.
.
d_num_steps
```

(repeat process for all nodes)

notes: this file contains the interpolated depth values along a straight-line fetch ray that are calculated at every step until land is reached. d_num_steps represents the last depth recorded before land is reached. This file will be read in by the code at line (1529) where the values will then be inverse-distance averaged according to the p-value specified by the user.

"Slope_upwind_2_deg.txt"

```
n1  m1 m2 m3 ... m(Num_Angs)
n2
n3
.
.
.
N_tot  m1 m2 m3 ... m(Num_Angs)
```

notes: This file represents the bottom slope between the depth at the wet node and the depth 1 step upwind (step according to step_res) at every angular heading (Num_Angs)

"Eff_fetch_2_deg.txt"

```
(N_tot)  (Ang_res)
n1  F1 F2 F3 ... F(360/Angres)
n2  F1 F2 F3 ... F(360/Angres)
.
.
.
N_tot  F1 F2 F3 ... F(360/Angres)
```

notes: this file contains the cosine weighted fetch values that are averaged from the "Depth_data_2_deg.txt" file based on the user-specified p-value.

"IDW_depths_2_deg.txt"

```
(N_tot)  (Ang_res)
n1  d_IDW1 d_IDW2 d_IDW3 ... F(360/Angres)
n2  d_IDW1 d_IDW2 d_IDW3 ... F(360/Angres)
.
.
.
N_tot  d_IDW1 d_IDW1 d_IDW1 ... F(360/Angres)
```

notes: this file contains the cosine weighted fetch values that are averaged from the "Depth_data_2_deg.txt" file based on the user-specified p-value.

C++ code: pre_proc.cpp

```
//This program pre-processes the fetch and depth data needed for the parametric wave code.
//It reads in a grid file (fort.14) and calculates fetch rays (cosine averaged), depth values (I.D.W. averaged),
//and bottom slopes for all wet nodes in the domain at angular increments (2 deg recommended)
//
//By: Samuel Boyd
//Date: 04/13/2020

#include <cmath>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>
#include <stdlib.h>
#include <cstdlib>
#include <math.h>
#include <vector>
#include <sstream>

using namespace std;

//variable definitions

char content[10];
double a;
double distlon,distlat;
double lat0,lon0,dlat1,dlat2,dlat3,dlat4,dlon1,dlon2,dlon3,dlon4;
int wet_tot,dry_tot,x,i,c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,ii,step1,step2,step3,step4,node1,node_tot,elm_tot;
double res;   //resolution (meters) for spatial step
int Angres;   //res (deg) for angular step
bool check;
double p,p_temp,Eff_res;                    //IDW power value and cosine weighted angular averaging value
double vx[4] = {};
double vy[4] = {};
double dx;                      //deg lat
double dy;                      //deg lon
double xloc[4],yloc[4] = {};
double D,delx,dely,theta;
double crossprod1,crossprod2,crossprod3;
int contELM1,contELM2,contELM3,contELM4;
double w1,w2,w3;
vector<double> dephold1,dephold2,dephold3,dephold4;
double meanwetlat,meanwetlon,wetlonsum,wetlatsum = 0;
double pi = 4*atan(1);
double Rearth;
double Req=6378137;
double Rpo=6356752;
double dhav1,dhav2,dhav3,dhav4;
int emin1,emax1,emin2,emax2,emin3,emax3,emin4,emax4=0;
int line,x1,x2,c,Ftop,Ftop2,Favg,k,k1,k2,num_depths,direction,dir_tot=0;
string s;
char temp[10];
double g=9.7918;
double Eff_top,Eff_bot,Eff_fetch,depth_temp,depth_tot,depth_cur,depth_prev,Eff_depth,denominator,Up;

int main(){

        float clock_t,time_req;
        time_req=clock();                                   //initializing the clock for timing tasks
//****************************************************************************************************
//These are the names of the final output files created for the grid's depth and fetch information
//****************************************************************************************************
        ofstream outfile6("Distance_data_2_deg.txt");
        ofstream outfile7("Depth_data_2_deg.txt");
        ofstream outfile8("Slope_upwind_2_deg.txt");
        ofstream outfile9("p_values_2_deg.txt");
        ofstream outfile2("IDW_depths_2_deg.txt");
```

```
		ofstream outfile1("Eff_fetch_2_deg.txt");
		ofstream outfile5("Raw_depth_2_deg.txt");

		ifstream infile;
		infile.open("fort.14");					//name of grid file
									//reading element and node total from fort.14
			while(infile >> content){
				x=x+1;
				if(x==2){
					elm_tot = atoi(content);
				}
				if(x==3){
					node_tot = atoi(content);
					break;
				}
			}
		x=0;

		infile.close();

		cout<<endl;
		cout<<"Element Total: "<<elm_tot<<"	Node Total: "<<node_tot<<endl;
		cout<<endl;
									//defining arrays once dimensions have been read
	double elv[node_tot+1];			//all nodes elevation
	double lon[node_tot+1];			//all nodes longitude
	double lat[node_tot+1];			//all nodes node latitude
	double node[node_tot+1];		//all nodes node number
	double wetnode[node_tot+1];		//wet node number
	double wetelv[node_tot+1];		//wet node elevation
	double wetlon[node_tot+1];		//wet node longitude
	double wetlat[node_tot+1];		//wet node latitude
	int ELM[elm_tot];			//triangular elements
	int NE[elm_tot][3];			//neighboring nodes in each element

		infile.open("fort.14");
									//reading lat,lon,elv,node,ELM,and NE data from fort.14
			while(infile >> content){
				x=x+1;
				if(x>=4 && x<= 4 + (node_tot*4)){	//4 is from node number(1),lat(2),lon(3),elv(4)
					i = i+1;
					a = atof(content);
					if(((i+0)%4) == 0){
						c1=c1+1;
						elv[c1] = a;
					}
					if(((i+1)%4) == 0){
						c2=c2+1;
						lat[c2] = a;
					}
					if(((i+2)%4) == 0){
						c3=c3+1;
						lon[c3] = a;
					}
					if(((i+3)%4) == 0){
						c4=c4+1;
						node[c4] = a;
					}
				}
	//Element information is being read once code reaches end of regular nodal info
				if(x >= (4 + (node_tot*4)) && x < (4 + (node_tot*4))+(elm_tot*5)){
					ii = ii+1;
					a = atoi(content);
					if(((ii+4)%5) == 0){
						c7 = c7+1;
						ELM[c7] = a;
					}
					if(((ii+2)%5) == 0){
						c8 = c8+1;
```

```
                    NE[c8][1] = a;
                }
                if(((ii+1)%5) == 0){
                    c9 = c9+1;
                    NE[c9][2] = a;
                }
                if(((ii+0)%5) == 0){
                    c10 = c10+1;
                    NE[c10][3] = a;
                }
            }
        }
    infile.close();
//calculating average wet lat and lon in domain
        for(int j=1; j<=node_tot; j++){
            if(elv[j] > 0){
                c5=c5+1;
                wetlatsum = wetlatsum+lat[j];
                wetlonsum = wetlonsum+lon[j];
            }
            if(elv[j] < 0){
                c6=c6+1;
            }
        }       //close node_tot loop
        wet_tot = c5;
        dry_tot = c6;
        meanwetlat = wetlatsum/wet_tot;
        meanwetlon = wetlonsum/wet_tot;
        cout<<"Mean longitude (wet): "<<meanwetlon<<"       mean latitude wet(wet): "<<meanwetlat<<endl;
        cout<<endl;
//interpolating the radius of earth between equator and poles wrt avg wet lattitude of the domain
    Rearth = sqrt((pow((Req*Req*cos(meanwetlat*pi/180)),2.0)+pow((Rpo*Rpo*sin(meanwetlat*pi/180)),2.0))/
         (pow((Req*cos(meanwetlat*pi/180)),2.0)+pow((Rpo*sin(meanwetlat*pi/180)),2.0)));
    cout<<"Radius of Earth at this location: "<<Rearth/1000<<" km"<<endl;
//haversine forumla distance(m) for 1 deg lat and 1 deg lon based on avg wetlat and wetlon of domain
    distlon = 2*Rearth*asin(sqrt(cos((meanwetlat-.5)*pi/180)*cos((meanwetlat+.5)*pi/180)*
                                        sin(.5*pi/180)*sin(.5*pi/180)));
    distlat = 2*Rearth*asin(sqrt(sin(.5*pi/180)*sin(.5*pi/180)));
    cout<<"1 deg lattitude = "<<distlat<<" m"<<endl;
    cout<<"1 deg longitude = "<<distlon<<" m"<<endl;
    cout<<endl;
    double d12;
    double d13;
    double d23;
    double dmin;
    double dhold=100000;
//calculating minimum distance (dhold) between wet nodes for spatial step reccomendation
    for(int e=1; e<=elm_tot; e++){
        if(elv[NE[e][1]]>0 && elv[NE[e][2]]>0 && elv[NE[e][3]]>0){
        d12=distlon*sqrt(pow(lon[NE[e][1]]-lon[NE[e][2]],2)+(pow(lat[NE[e][1]]-lat[NE[e][2]],2)));
        d13=distlon*sqrt(pow(lon[NE[e][1]]-lon[NE[e][3]],2)+(pow(lat[NE[e][1]]-lat[NE[e][3]],2)));
        d23=distlon*sqrt(pow(lon[NE[e][2]]-lon[NE[e][3]],2)+(pow(lat[NE[e][2]]-lat[NE[e][3]],2)));
            //cout<<"d12: "<<d12<<" d13: "<<d13<<" d23: "<<d23<<endl;
            if(d12<d13 && d12<d23){
                dmin=d12;
            }
            if(d13<d12 && d13<d23){
                dmin=d13;
            }
            if(d23<d12 && d23<d13){
                dmin=d23;
            }
            //cout<<"dmin: "<<dmin<<endl;
            if(dmin<dhold){
                dhold=dmin;
    //cout<<"e: "<<e<<" NE[1]: "<<NE[e][1]<<" NE[2]: "<<NE[e][2]<<" NE[3]: "<<NE[e][3]<<" dmin: "<<dhold<<endl;
            }
        }
        dmin=0;
```

```
            }
double steprec=floor(3*dhold);          //recomended spatial step resolution
//User input for step and angular resolution
            cout<<"Enter spatial step resolution (recommended = "<<steprec<<"m): ";
            cin>>res;
            dx = res/distlon;               //50m step in lat/lon degrees specified by the step resolution
            dy = res/distlat;
            //cout<<"dx("<<res<<"m): "<<dx<<" deg lon"<<endl;
            //cout<<"dy("<<res<<"m): "<<dy<<" deg lat"<<endl;
            cout<<"Enter angular resolution (2,5,10,18,or 30 deg): ";
            cin>>Angres;
            check=false;
            if(Angres==2 || Angres==5 || Angres==10 || Angres==18 || Angres==30){
                        check=true;
                        cout<<endl;
            }
            while(check==false){
                        cout<<"Inter valid angular resolution (2,5,10,18, or 30 deg): ";
                        cin>>Angres;
                        if(Angres==2 || Angres==5 || Angres==10 || Angres==18 || Angres==30){
                                    check=true;
                                    cout<<endl;
                        }
            }
//initializing solution array once angular resolution is known
            double d[node_tot+1][360/Angres];
            double Ares = (360/Angres)/4;                       //Num of angles per quadrent
            int Ares2 = Ares;                                   //cast to integer for indexing array
            double fetch[node_tot+1][360/Angres];
            double F[node_tot+1][360/Angres];
            double depth[node_tot+1][360/Angres];
            double slopehold[node_tot][360/Angres];
            double startelv;
            for (int j=1; j<=node_tot; j++){                    //initializing slope array with zeros
                        for(int m=1; m<=360/Angres; m++){
                                    slopehold[j][m]=0;
                        }
            }
double Pelv1[1+(360/(4*Angres))],Pelv2[1+(360/(4*Angres))],Pelv3[1+(360/(4*Angres))],Pelv4[1+(360/(4*Angres))];
double XP1[1+(360/(4*Angres))],XP2[1+(360/(4*Angres))],XP3[1+(360/(4*Angres))],XP4[1+(360/(4*Angres))];
double YP1[1+(360/(4*Angres))],YP2[1+(360/(4*Angres))],YP3[1+(360/(4*Angres))],YP4[1+(360/(4*Angres))];
double Phold1[1+(360/(4*Angres))],Phold2[1+(360/(4*Angres))],Phold3[1+(360/(4*Angres))],Phold4[1+(360/(4*Angres))];
double dpast1[1+(360/(4*Angres))],dpast2[1+(360/(4*Angres))],dpast3[1+(360/(4*Angres))],dpast4[1+(360/(4*Angres))];
            cout<<"Enter a positive power parameter for depth weights"<<endl;
            cout<<"(100 = no depth weighting, 0 = variable weighting): ";
            cin>>p_temp;
            check=false;
            if(p_temp>0){
                        check=true;
                        cout<<endl;
            }
            if(p_temp==0){                          //variable weighting 0 number
                        check=true;
                        cout<<endl;
            }
            while(check==false){
                        cout<<"Inter valid power parameter (greater than or equal to zero): ";
                        cin>>p_temp;
                        if(p_temp>0){
                                    check=true;
                                    cout<<endl;
                        }
            }
            cout<<"Enter an angular resolution for effective fetch weighting (deg): ";
            cin>>Eff_res;
            cout<<"Effective angular averaging will consider "<<floor(Eff_res/Angres)<<" rays on either side"<<endl;
            cout<<endl;
            cout<<"Enter Avg. wind speed expected in domain (m/s)"<<endl;
            cout<<"(this will be used for IDW p-value if chosen): ";
```

```
        cin>>Up;
        //creating raw depth output (just the depth at each node)
        for(int i=1; i<=node_tot; i++){
                outfile5<<i<<"  "<<elv[i]<<endl;
        }
        outfile5.close();
//***************************************Begin Fetch and Depth Code****************************************
        cout<<"**********Calculating Fetch and Depth**********"<<endl;
        outfile6<<"Fetch Distances at spatial res: "<<res<<"[m] and angular res: "<<Angres<<"[deg]"<<endl;
        outfile6<<"Grid used to generate file: 'fort.14'"<<endl;
        outfile7<<"Water Depths along fetch rays from 'fort.14'"<<endl;
        outfile7<<node_tot<<" "<<360/Angres<<endl;
        //outfile8<<"Upwind bottom slope from node used for breaking criteria"<<endl;
        //outfile8<<"Grid used to generate file: 'fort.14'"<<endl;
        for(int j=1; j<=node_tot; j++){                 //for all nodes
                cout<<"node: "<<j<<endl;
                outfile6<<j<<"  ";
                outfile8<<j<<"  ";
                lon0 = lon[j];
                lat0 = lat[j];
                if(elv[j] < 0){                         //if node is dry, write 0 slopes for output file
                        for(int k=1; k<=360/Angres; k++){
                                outfile8<<"0 ";
                        }
                }
                if(elv[j] > 0){                         //if node is wet
                        startelv=elv[j];                //depth at the start node
//*************************************Q1*********************************************************************
                        for(int m=0; m<Ares; m++){   //Angular resolution including 0 excluding Ares [0-90) deg
                                outfile7<<j<<" ";
                                outfile7<<m+1<<" ";
                                dephold1.clear();       //erases elements from dephold vector
                                step1 = 0;
                                Pelv1[m] = elv[j];
                                Phold1[m] = elv[j];
                                dpast1[m]=0;
                                                        //min and max from elm search optimization
analysis
                                emin1=2*j-floor(.0163*j+1500);
                                emax1=2*j+1500;
                                while(Pelv1[m] > 0){    //while (XP,YP) is wet
                                        dephold1.push_back(Pelv1[m]);   //stores every Pelv value
                                        step1=step1+1;                  //increment XP and YP 1 step further
                                        XP1[m] = lon0 + (dx*cos((pi/180)*(90-(Angres*m))))*step1;
                                        YP1[m] = lat0 + (dy*sin((pi/180)*(90-(Angres*m))))*step1;
//if first step already taken, compare the 1st neighboring node of the previous containing element
//so that the restricted element search is wrt the containing element instead of the start node j
//Pelv1 at step1==2 corresponds to the first step's interpolated depth. Calculate slope from start to here
//BUT slope values are stored in slopehold array in flipped indexes. Q1<-->Q3 and Q2<-->Q4 because we want
//the index to correspond to upwind slope for determining wave breaking
                                        if(step1==2){
                                                slopehold[j][m+1]=(startelv-Pelv1[m])/res;
//if slope is less than 1/1000 set to 0, this inludes negative slopes (from shallow to deeper water)
                                                if(slopehold[j][m+1] < 1/1000){
                                                        slopehold[j][m+1]=0;
                                                }
                                                outfile8<<slopehold[j][m+1]<<" ";
                                        }
                                        if(step1>1){
                                                emin1=2*NE[contELM1][1]-floor(.0163*NE[contELM1][1]+1500);
                                                emax1=2*NE[contELM1][1]+1500;
                                        }
                                        if(emin1<1){
                                                emin1=1;
                                        }
                                        if(emax1>=elm_tot){
                                                emax1=elm_tot;
                                        }
                                        contELM1 = false;       //resent containing element
```

```
//****Element Search*****
//restricted element search according to emin and emax values
                                                            for(int e=emin1; e<=emax1; e++){
                                                                    for(int i=1; i<=3; i++){       //vectors of neighboring nodes
                                                                            xloc[i] = lon[NE[e][i]];
                                                                            yloc[i] = lat[NE[e][i]];
                                                                            delx = xloc[i] - XP1[m];
                                                                            dely = yloc[i] - YP1[m];
                                                                            D = pow((pow(delx,2.0)+pow(dely,2.0)),0.5);
                                                                            theta = atan2(dely,delx);

                                                                            vx[i] = D*cos(theta);
                                                                            vy[i] = D*sin(theta);
                                                                    }
                                                                                    //cross products of neighboring node
vectors
                                                                    crossprod1 = vx[1]*vy[2] - vy[1]*vx[2];
                                                                    crossprod2 = vx[2]*vy[3] - vy[2]*vx[3];
                                                                    crossprod3 = vx[3]*vy[1] - vy[3]*vx[1];
                                                                                    //if cross products are all >= 0,
containing ELM
                                                                    if(crossprod1>=0 && crossprod2>=0 && crossprod3>=0){
                                                                            contELM1 = ELM[e];
//cout<<"R.SearchFound.   contELM1: "<<contELM1<<"   emin:  "<<emin1<<"   emax: "<<emax1<<endl;
//****Barycentric elevation interpolation****
                                                    w1 =((yloc[2]-yloc[3])*(XP1[m]-xloc[3])+(xloc[3]-xloc[2])*(YP1[m]-yloc[3]))/
                                                      ((yloc[2]-yloc[3])*(xloc[1]-xloc[3])+(xloc[3]-xloc[2])*(yloc[1]-yloc[3]));
                                                    w2 =((yloc[3]-yloc[1])*(XP1[m]-xloc[3])+(xloc[1]-xloc[3])*(YP1[m]-yloc[3]))/
                                                      ((yloc[2]-yloc[3])*(xloc[1]-xloc[3])+(xloc[3]-xloc[2])*(yloc[1]-yloc[3]));
                                                    w3 = 1-w1-w2;
                                                    Pelv1[m] = (w1*elv[NE[e][1]]+w2*elv[NE[e][2]]+w3*elv[NE[e][3]])/(w1+w2+w3);
                                                                            if(Pelv1[m] > 0){
                                                                                    Phold1[m] = Pelv1[m];     //The last wet
point
                                                                            }
                                                                            break;          //out of for elements loop
                                                                    }               //close if(crossprod > 0)
                                                            }               //close restricted element search for loop
                                                    if(contELM1 == false){
//cout<<"Long Search Q1************** steps: "<<step1<<endl;
//if containing element is not found, check all elements to see if restricted seach missed it
                                                            for(int e=1; e<elm_tot; e++){
                                                                    for(int i=1; i<=3; i++){       //vectors of neighboring nodes
                                                                            xloc[i] = lon[NE[e][i]];
                                                                            yloc[i] = lat[NE[e][i]];
                                                                            delx = xloc[i] - XP1[m];
                                                                            dely = yloc[i] - YP1[m];
                                                                            D = pow((pow(delx,2.0)+pow(dely,2.0)),0.5);
                                                                            theta = atan2(dely,delx);

                                                                            vx[i] = D*cos(theta);
                                                                            vy[i] = D*sin(theta);
                                                                    }
                                                                                    //cross products of neighboring node
vectors
                                                                    crossprod1 = vx[1]*vy[2] - vy[1]*vx[2];
                                                                    crossprod2 = vx[2]*vy[3] - vy[2]*vx[3];
                                                                    crossprod3 = vx[3]*vy[1] - vy[3]*vx[1];
                                                                                    //if cross products are all >= 0,
containing ELM
                                                                    if(crossprod1>=0 && crossprod2>=0 && crossprod3>=0){
                                                                            contELM1 = ELM[e];
//****Barycentric elevation interpolation****
                                                    w1 =((yloc[2]-yloc[3])*(XP1[m]-xloc[3])+(xloc[3]-xloc[2])*(YP1[m]-yloc[3]))/
                                                      ((yloc[2]-yloc[3])*(xloc[1]-xloc[3])+(xloc[3]-xloc[2])*(yloc[1]-yloc[3]));
                                                    w2 =((yloc[3]-yloc[1])*(XP1[m]-xloc[3])+(xloc[1]-xloc[3])*(YP1[m]-yloc[3]))/
                                                      ((yloc[2]-yloc[3])*(xloc[1]-xloc[3])+(xloc[3]-xloc[2])*(yloc[1]-yloc[3]));
                                                    w3 = 1-w1-w2;
                                                    Pelv1[m] = (w1*elv[NE[e][1]]+w2*elv[NE[e][2]]+w3*elv[NE[e][3]])/(w1+w2+w3);
```

```
                                                                        if(Pelv1[m] > 0){

                                                                                Phold1[m] = Pelv1[m];     //The last wet
point
                                                                        }
                                                                        break;          //out of for elements loop
                                                                }               //close if(crossprod > 0)
                                                        }               //close for all elements loop
//cout<<"!!!!contELM1: "<<contELM1<<"   emin:  "<<emin1<<"   emax: "<<emax1<<endl;
                                                }               //close if contELM1 == false
//if containing element is still false, then the element is out of bounds
                                                        if(contELM1 == false){
                                                                Pelv1[m]=0;
                                                        }
                                                }               //close while Pelv (XP,YP) > 0 loop
                                                outfile7<<dephold1.size()<<endl;
                                                for(int h=0; h<dephold1.size(); h++){
                                                        outfile7<<dephold1[h]<<endl;
                                                }
                                                        //estimating distance past the last wet point

                                                dpast1[m] = Phold1[m]/((Phold1[m] + abs(Pelv1[m]))/res);
                                                dlon1 = abs(XP1[m] - lon0);             //distance from start to land in lat/lon
                                                dlat1 = abs(YP1[m] - lat0);
        //if NOT the first step and does NOT go out of bounds, this is the usual condition
        //distance from start to land-res = dist from start to step before land + dist past that last point
                                                dhav1 = 2*Rearth*asin(sqrt(sin(((dlat1)/2)*pi/180)*
                                                                        sin(((dlat1)/2)*pi/180)+
                                                                (cos(lat0*pi/180)*cos(YP1[m]*pi/180)*
                                                                        sin(((dlon1)/2)*pi/180)*
                                                                        sin(((dlon1)/2)*pi/180))))-res+dpast1[m];
        //if first step hits land, dhav = dpast
                                                if(step1 == 1){
                                                        dhav1=dpast1[m];
        //if first step is out of bounds, set dhav = 0
                                                        if(contELM1 == false){
                                                                dhav1=0;
                                                        }
                                                }
        //if NOT first step and out of bounds,calculate dhav = d(out of bounds)-res/2, this is a round estimate
                                                if(step1!=1 && contELM1 ==false){
                                                        dhav1 = 2*Rearth*asin(sqrt(sin(((dlat1)/2)*pi/180)*
                                                                                sin(((dlat1)/2)*pi/180)+
                                                                        (cos(lat0*pi/180)*cos(YP1[m]*pi/180)*
                                                                                sin(((dlon1)/2)*pi/180)*
                                                                                sin(((dlon1)/2)*pi/180))))-(res/2);
                                                }
                                                outfile6<<dhav1<<" ";
                                        }               //close for m
//*************************************Q4*******************************************************************
                                        for(int m=0; m<Ares; m++){                      //including 0 excluding Ares [90-180) deg
                                                outfile7<<j<<" ";
                                                outfile7<<m+Angres<<" ";
                                                dephold4.clear();
                                                step4 = 0;
                                                Pelv4[m] = elv[j];
                                                Phold4[m] = elv[j];
                                                dpast4[m]=0;
                                                emin4=2*j-floor(.0163*j+1500);
                                                emax4=2*j+1500;
                                                while(Pelv4[m] > 0){                            //while (XP,YP) is wet
                                                        dephold4.push_back(Pelv4[m]);           //stores every Pelv value
                                                        step4=step4+1;
                                                        XP4[m] = lon0 + (dx*cos((pi/180)*(Angres*m)))*step4;
                                                        YP4[m] = lat0 - (dy*sin((pi/180)*(Angres*m)))*step4;
//if first step already taken, compare the 1st neighboring node of the previous containing element
//so that the restricted element search is wrt the containing element instead of the start node j
//Pelv4 at step4==2 corresponds to the first step's interpolated depth. Calculate slope from start to here
//BUT slope values are stored in slopehold array in flipped indexes. Q1<-->Q3 and Q2<-->Q4 because we want
```

```
//the index to correspond to upwind slope for determining wave breaking
                                        if(step4==2){
                                                slopehold[j][m+Ares2+1]=(startelv-Pelv4[m])/res;
//if slope is less than 1/1000 set to 0, this inludes negative slopes (from shallow to deeper water)
                                                if(slopehold[j][m+Ares2+1] < 1/1000){
                                                        slopehold[j][m+Ares2+1]=0;
                                                }
                                                outfile8<<slopehold[j][m+Ares2+1]<<" ";
                                        }
                                        if(step4>1){
                                                emin4=2*NE[contELM4][1]-floor(.0163*NE[contELM4][1]+1500);
                                                emax4=2*NE[contELM4][1]+1500;
                                        }
                                        if(emin4<1){
                                                emin4=1;
                                        }
                                        if(emax4>=elm_tot){
                                                emax4=elm_tot;
                                        }
                                        contELM4 = false;       //resent containing element
//****Element Search****
                                        for(int e=emin4; e<=emax4; e++){
                                                for(int i=1; i<=3; i++){
                                                        xloc[i] = lon[NE[e][i]];
                                                        yloc[i] = lat[NE[e][i]];
                                                        delx = xloc[i] - XP4[m];
                                                        dely = yloc[i] - YP4[m];
                                                        D = pow((pow(delx,2.0)+pow(dely,2.0)),0.5);
                                                        theta = atan2(dely,delx);

                                                        vx[i] = D*cos(theta);
                                                        vy[i] = D*sin(theta);
                                                }
                                                crossprod1 = vx[1]*vy[2] - vy[1]*vx[2];
                                                crossprod2 = vx[2]*vy[3] - vy[2]*vx[3];
                                                crossprod3 = vx[3]*vy[1] - vy[3]*vx[1];
                                                if(crossprod1>=0 && crossprod2>=0 && crossprod3>=0){
                                                        contELM4 = ELM[e];
//****Barycentric elevation interpolation****
                                w1 =((yloc[2]-yloc[3])*(XP4[m]-xloc[3])+(xloc[3]-xloc[2])*(YP4[m]-yloc[3]))/
                                   ((yloc[2]-yloc[3])*(xloc[1]-xloc[3])+(xloc[3]-xloc[2])*(yloc[1]-yloc[3]));
                                w2 =((yloc[3]-yloc[1])*(XP4[m]-xloc[3])+(xloc[1]-xloc[3])*(YP4[m]-yloc[3]))/
                                   ((yloc[2]-yloc[3])*(xloc[1]-xloc[3])+(xloc[3]-xloc[2])*(yloc[1]-yloc[3]));
                                w3 = 1-w1-w2;
                                Pelv4[m] = (w1*elv[NE[e][1]]+w2*elv[NE[e][2]]+w3*elv[NE[e][3]])/(w1+w2+w3);
                                                        if(Pelv4[m] > 0){
                                                                Phold4[m] = Pelv4[m];   //The last wet
point
                                                        }
                                                break;          //out of for elements loop
                                                }               //close if(crossprod > 0)
                                        }               //close restricted elemt search for loop
//if no containing element, search all nodes
                                if(contELM4 == false){
//cout<<"Long Search Q4************** steps: "<<step4<<endl;
                                        for(int e=1; e<elm_tot; e++){
                                                for(int i=1; i<=3; i++){

                                                        xloc[i] = lon[NE[e][i]];
                                                        yloc[i] = lat[NE[e][i]];
                                                        delx = xloc[i] - XP4[m];
                                                        dely = yloc[i] - YP4[m];
                                                        D = pow((pow(delx,2.0)+pow(dely,2.0)),0.5);
                                                        theta = atan2(dely,delx);

                                                        vx[i] = D*cos(theta);
                                                        vy[i] = D*sin(theta);
                                                }
                                                crossprod1 = vx[1]*vy[2] - vy[1]*vx[2];
```

```
                                                            crossprod2 = vx[2]*vy[3] - vy[2]*vx[3];
                                                            crossprod3 = vx[3]*vy[1] - vy[3]*vx[1];
                                                            if(crossprod1>=0 && crossprod2>=0 && crossprod3>=0){
                                                                    contELM4 = ELM[e];
//****Barycentric elevation interpolation****
                                            w1 =((yloc[2]-yloc[3])*(XP4[m]-xloc[3])+(xloc[3]-xloc[2])*(YP4[m]-yloc[3]))/
                                               ((yloc[2]-yloc[3])*(xloc[1]-xloc[3])+(xloc[3]-xloc[2])*(yloc[1]-yloc[3]));
                                            w2 =((yloc[3]-yloc[1])*(XP4[m]-xloc[3])+(xloc[1]-xloc[3])*(YP4[m]-yloc[3]))/
                                               ((yloc[2]-yloc[3])*(xloc[1]-xloc[3])+(xloc[3]-xloc[2])*(yloc[1]-yloc[3]));
                                            w3 = 1-w1-w2;
                                            Pelv4[m] = (w1*elv[NE[e][1]]+w2*elv[NE[e][2]]+w3*elv[NE[e][3]])/(w1+w2+w3);
                                                                    if(Pelv4[m] > 0){
                                                                            Phold4[m] = Pelv4[m];     //The last wet
point
                                                                    }
                                                            break;          //out of for elements loop
                                                            }               //close if(crossprod > 0)
                                                    }               //close restricted elemt search for loop
//cout<<"!!!!contELM4: "<<contELM4<<"   emin:  "<<emin4<<"   emax: "<<emax4<<endl;
                                            }               //close if contELM4 == false
//if contELM4 is still false, then out of bounds
                                                    if(contELM4 == false){
                                                            Pelv4[m]=0;
                                                    }
                                            }               //close while Pelv (XP,YP) > 0 loop
                                            outfile7<<dephold4.size()<<endl;
                                            for(int h=0; h<dephold4.size(); h++){
                                                    outfile7<<dephold4[h]<<endl;
                                            }
                                            dpast4[m] = Phold4[m]/((Phold4[m] + abs(Pelv4[m]))/res);
                                            dlon4 = abs(XP4[m] - lon0);
                                            dlat4 = abs(YP4[m] - lat0);
                                            dhav4 = 2*Rearth*asin(sqrt(sin(((dlat4)/2)*pi/180)*
                                                                    sin(((dlat4)/2)*pi/180)+
                                                             (cos(lat0*pi/180)*cos(YP4[m]*pi/180)*
                                                                    sin(((dlon4)/2)*pi/180)*
                                                                    sin(((dlon4)/2)*pi/180))))-res+dpast4[m];
                                            if(step4 == 1){
                                                    dhav4=dpast4[m];
                                                    if(contELM4 == false){
                                                            dhav4=0;
                                                    }
                                            }
                                            if(step4!=1 && contELM4 ==false){
                                                    dhav4 = 2*Rearth*asin(sqrt(sin(((dlat4)/2)*pi/180)*
                                                                            sin(((dlat4)/2)*pi/180)+
                                                                     (cos(lat0*pi/180)*cos(YP4[m]*pi/180)*
                                                                            sin(((dlon4)/2)*pi/180)*
                                                                            sin(((dlon4)/2)*pi/180))))-(res/2);
                                            }
                                            outfile6<<dhav4<<" ";
                                    }       //close for m
//*************************************Q3*******************************************************************
                                    for(int m=0; m<Ares; m++){                      //including 0 excluding Ares (0-80 deg)
                                            outfile7<<j<<" ";
                                            outfile7<<m+Angres+Angres-1<<" ";
                                            dephold3.clear();
                                            step3 = 0;
                                            Pelv3[m] = elv[j];
                                            Phold3[m] = elv[j];
                                            dpast3[m]=0;
                                            emin3=2*j-floor(.0163*j+1500);
                                            emax3=2*j+1500;
                                            while(Pelv3[m] > 0){                            //while (XP,YP) is wet
                                                    dephold3.push_back(Pelv3[m]);           //stores every Pelv value
                                                    step3=step3+1;
                                                    XP3[m] = lon0 - (dx*cos((pi/180)*(90-(Angres*m))))*step3;
                                                    YP3[m] = lat0 - (dy*sin((pi/180)*(90-(Angres*m))))*step3;
//if first step already taken, compare the 1st neighboring node of the previous containing element
```

```
//so that the restricted element search is wrt the containing element instead of the start node j
//Pelv3 at step3==2 corresponds to the first step's interpolated depth. Calculate slope from start to here
//BUT slope values are stored in slopehold array in flipped indexes. Q1<-->Q3 and Q2<-->Q4 because we want
//the index to correspond to upwind slope for determining wave breaking
						if(step3==2){
							slopehold[j][m+Ares2+Ares2+1]=(startelv-Pelv3[m])/res;
//if slope is less than 1/1000 set to 0, this inludes negative slopes (from shallow to deeper water)
							if(slopehold[j][m+Ares2+Ares2+1] < 1/1000){
								slopehold[j][m+Ares2+Ares2+1]=0;
							}
							outfile8<<slopehold[j][m+Ares2+Ares2+1]<<" ";
						}
						if(step3>1){
							emin3=2*NE[contELM3][1]-floor(.0163*NE[contELM3][1]+1500);
							emax3=2*NE[contELM3][1]+1500;
						}
						if(emin3<1){
							emin3=1;
						}
						if(emax3>=elm_tot){
							emax3=elm_tot;
						}
						contELM3 = false;		//resent containing element
//****Element Search****
						for(int e=emin3; e<=emax3; e++){
							for(int i=1; i<=3; i++){
								xloc[i] = lon[NE[e][i]];
								yloc[i] = lat[NE[e][i]];
								delx = xloc[i] - XP3[m];
								dely = yloc[i] - YP3[m];
								D = pow((pow(delx,2.0)+pow(dely,2.0)),0.5);
								theta = atan2(dely,delx);

								vx[i] = D*cos(theta);
								vy[i] = D*sin(theta);
							}
							crossprod1 = vx[1]*vy[2] - vy[1]*vx[2];
							crossprod2 = vx[2]*vy[3] - vy[2]*vx[3];
							crossprod3 = vx[3]*vy[1] - vy[3]*vx[1];
							if(crossprod1>=0 && crossprod2>=0 && crossprod3>=0){
								contELM3 = ELM[e];
//****Barycentric elevation interpolation****
					w1 =((yloc[2]-yloc[3])*(XP3[m]-xloc[3])+(xloc[3]-xloc[2])*(YP3[m]-yloc[3]))/
					   ((yloc[2]-yloc[3])*(xloc[1]-xloc[3])+(xloc[3]-xloc[2])*(yloc[1]-yloc[3]));
					w2 =((yloc[3]-yloc[1])*(XP3[m]-xloc[3])+(xloc[1]-xloc[3])*(YP3[m]-yloc[3]))/
					   ((yloc[2]-yloc[3])*(xloc[1]-xloc[3])+(xloc[3]-xloc[2])*(yloc[1]-yloc[3]));
					w3 = 1-w1-w2;
					Pelv3[m] = (w1*elv[NE[e][1]]+w2*elv[NE[e][2]]+w3*elv[NE[e][3]])/(w1+w2+w3);
								if(Pelv3[m] > 0){
									Phold3[m] = Pelv3[m];	//The last wet
point
								}
							break;		//out of for elements loop
							}		//close if(crossprod > 0)
						}		//close restricted element search for loop
					if(contELM3 == false){
//cout<<"Long Search Q3************** steps: "<<step3<<endl;
						for(int e=1; e<elm_tot; e++){
							for(int i=1; i<=3; i++){
								xloc[i] = lon[NE[e][i]];
								yloc[i] = lat[NE[e][i]];
								delx = xloc[i] - XP3[m];
								dely = yloc[i] - YP3[m];
								D = pow((pow(delx,2.0)+pow(dely,2.0)),0.5);
								theta = atan2(dely,delx);

								vx[i] = D*cos(theta);
								vy[i] = D*sin(theta);
							}
```

```
                                                            crossprod1 = vx[1]*vy[2] - vy[1]*vx[2];
                                                            crossprod2 = vx[2]*vy[3] - vy[2]*vx[3];
                                                            crossprod3 = vx[3]*vy[1] - vy[3]*vx[1];
                                                            if(crossprod1>=0 && crossprod2>=0 && crossprod3>=0){
                                                                    contELM3 = ELM[e];
//****Barycentric elevation interpolation****
                                            w1 =((yloc[2]-yloc[3])*(XP3[m]-xloc[3])+(xloc[3]-xloc[2])*(YP3[m]-yloc[3]))/
                                               ((yloc[2]-yloc[3])*(xloc[1]-xloc[3])+(xloc[3]-xloc[2])*(yloc[1]-yloc[3]));
                                            w2 =((yloc[3]-yloc[1])*(XP3[m]-xloc[3])+(xloc[1]-xloc[3])*(YP3[m]-yloc[3]))/
                                               ((yloc[2]-yloc[3])*(xloc[1]-xloc[3])+(xloc[3]-xloc[2])*(yloc[1]-yloc[3]));
                                            w3 = 1-w1-w2;
                                            Pelv3[m] = (w1*elv[NE[e][1]]+w2*elv[NE[e][2]]+w3*elv[NE[e][3]])/(w1+w2+w3);
                                                                    if(Pelv3[m] > 0){
                                                                            Phold3[m] = Pelv3[m];     //The last wet
point
                                                                    }
                                                            break;          //out of for elements loop
                                                            }               //close if(crossprod > 0)
                                                    }               //close restricted element search for loop
//cout<<"!!!!contELM3: "<<contELM3<<"   emin:  "<<emin3<<"   emax: "<<emax3<<endl;
                                            }               //close if contELM3 == false;
//if still no containing element, out of bounds
                                                    if(contELM3 == false){
                                                            Pelv3[m]=0;
                                                            //break;                                  //break to...
                                                    }
                                            }               //close while Pelv (XP,YP) > 0 loop
                                            outfile7<<dephold3.size()<<endl;
                                            for(int h=0; h<dephold3.size(); h++){
                                                    outfile7<<dephold3[h]<<endl;
                                            }
                                            dpast3[m] = Phold3[m]/((Phold3[m] + abs(Pelv3[m]))/res);
                                            dlon3 = abs(XP3[m] - lon0);
                                            dlat3 = abs(YP3[m] - lat0);
                                            dhav3 = 2*Rearth*asin(sqrt(sin(((dlat3)/2)*pi/180)*
                                                                    sin(((dlat3)/2)*pi/180)+
                                                             (cos(lat0*pi/180)*cos(YP3[m]*pi/180)*
                                                                    sin(((dlon3)/2)*pi/180)*
                                                                    sin(((dlon3)/2)*pi/180))))-res+dpast3[m];

                                            if(step3 == 1){
                                                    //dold3[m] = dpast3[m];
                                                    dhav3=dpast3[m];
                                                    if(contELM3 == false){
                                                            //dold3[m] = 0;
                                                            dhav3=0;
                                                    }
                                            }
                                            if(step3!=1 && contELM3 ==false){
                                                    dhav3 = 2*Rearth*asin(sqrt(sin(((dlat3)/2)*pi/180)*
                                                                            sin(((dlat3)/2)*pi/180)+
                                                                     (cos(lat0*pi/180)*cos(YP3[m]*pi/180)*
                                                                            sin(((dlon3)/2)*pi/180)*
                                                                            sin(((dlon3)/2)*pi/180))))-(res/2);
                                            }
                                            outfile6<<dhav3<<" ";
                                    }               //close for m
//************************************Q2*******************************************************************
                                    for(int m=0; m<Ares; m++){                          //including 0 excluding Ares (0-80 deg)
                                            outfile7<<j<<" ";
                                            outfile7<<m+Angres+Angres+Angres-2<<" ";
                                            dephold2.clear();
                                            step2 = 0;
                                            Pelv2[m] = elv[j];
                                            Phold2[m] = elv[j];
                                            dpast2[m]=0;
                                            emin2=2*j-floor(.0163*j+1500);
                                            emax2=2*j+1500;
                                            while(Pelv2[m] > 0){                                //while (XP,YP) is wet
```

```
                                        dephold2.push_back(Pelv2[m]);           //stores every Pelv value
                                        step2=step2+1;
                                        XP2[m] = lon0 - (dx*cos((pi/180)*(Angres*m)))*step2;
                                        YP2[m] = lat0 + (dy*sin((pi/180)*(Angres*m)))*step2;
//if first step already taken, compare the 1st neighboring node of the previous containing element
//so that the restricted element search is wrt the containing element instead of the start node j
//Pelv1 at step1==2 corresponds to the first step's interpolated depth. Calculate slope from start to here
//BUT slope values are stored in slopehold array in flipped indexes. Q1<-->Q3 and Q2<-->Q4 because we want
//the index to correspond to upwind slope for determining wave breaking
                                        if(step2==2){
                                                slopehold[j][m+Ares2+Ares2+Ares2+1]=(startelv-Pelv2[m])/res;
//if slope is less than 1/1000 set to 0, this inludes negative slopes (from shallow to deeper water)
                                                if(slopehold[j][m+Ares2+Ares2+Ares2+1] < 1/1000){
                                                        slopehold[j][m+Ares2+Ares2+Ares2+1]=0;
                                                }
                                                outfile8<<slopehold[j][m+Ares2+Ares2+Ares2+1]<<" ";
                                        }
                                        if(step2>1){
                                                emin2=2*NE[contELM2][1]-floor(.0163*NE[contELM2][1]+1500);
                                                emax2=2*NE[contELM2][1]+1500;
                                        }
                                        if(emin2<1){
                                                emin2=1;
                                        }
                                        if(emax2>=elm_tot){
                                                emax2=elm_tot;
                                        }
                                        contELM2 = false;           //resent containing element
//****Element Search****
                                        for(int e=emin2; e<=emax2; e++){
                                                for(int i=1; i<=3; i++){
                                                        xloc[i] = lon[NE[e][i]];
                                                        yloc[i] = lat[NE[e][i]];
                                                        delx = xloc[i] - XP2[m];
                                                        dely = yloc[i] - YP2[m];
                                                        D = pow((pow(delx,2.0)+pow(dely,2.0)),0.5);
                                                        theta = atan2(dely,delx);

                                                        vx[i] = D*cos(theta);
                                                        vy[i] = D*sin(theta);
                                                }
                                                crossprod1 = vx[1]*vy[2] - vy[1]*vx[2];
                                                crossprod2 = vx[2]*vy[3] - vy[2]*vx[3];
                                                crossprod3 = vx[3]*vy[1] - vy[3]*vx[1];
                                                if(crossprod1>=0 && crossprod2>=0 && crossprod3>=0){
                                                        contELM2 = ELM[e];
//****Barycentric elevation interpolation****
                                w1 =((yloc[2]-yloc[3])*(XP2[m]-xloc[3])+(xloc[3]-xloc[2])*(YP2[m]-yloc[3]))/
                                   ((yloc[2]-yloc[3])*(xloc[1]-xloc[3])+(xloc[3]-xloc[2])*(yloc[1]-yloc[3]));
                                w2 =((yloc[3]-yloc[1])*(XP2[m]-xloc[3])+(xloc[1]-xloc[3])*(YP2[m]-yloc[3]))/
                                   ((yloc[2]-yloc[3])*(xloc[1]-xloc[3])+(xloc[3]-xloc[2])*(yloc[1]-yloc[3]));
                                w3 = 1-w1-w2;
                                Pelv2[m] = (w1*elv[NE[e][1]]+w2*elv[NE[e][2]]+w3*elv[NE[e][3]])/(w1+w2+w3);
                                                        if(Pelv2[m] > 0){
                                                                Phold2[m] = Pelv2[m];     //The last wet
point
                                                        }
                                                break;          //out of for elements loop
                                                }               //close if(crossprod > 0)
                                        }               //close restricted element search for loop
                                if(contELM2 == false){
//cout<<"Long Search Q2************** steps: "<<step2<<endl;
                                        for(int e=1; e<elm_tot; e++){
                                                for(int i=1; i<=3; i++){
                                                        xloc[i] = lon[NE[e][i]];
                                                        yloc[i] = lat[NE[e][i]];
                                                        delx = xloc[i] - XP2[m];
                                                        dely = yloc[i] - YP2[m];
                                                        D = pow((pow(delx,2.0)+pow(dely,2.0)),0.5);
```

```
                            theta = atan2(dely,delx);

                            vx[i] = D*cos(theta);
                            vy[i] = D*sin(theta);
                        }
                        crossprod1 = vx[1]*vy[2] - vy[1]*vx[2];
                        crossprod2 = vx[2]*vy[3] - vy[2]*vx[3];
                        crossprod3 = vx[3]*vy[1] - vy[3]*vx[1];
                        if(crossprod1>=0 && crossprod2>=0 && crossprod3>=0){
                            contELM2 = ELM[e];
//****Barycentric elevation interpolation****
                w1 =((yloc[2]-yloc[3])*(XP2[m]-xloc[3])+(xloc[3]-xloc[2])*(YP2[m]-yloc[3]))/
                   ((yloc[2]-yloc[3])*(xloc[1]-xloc[3])+(xloc[3]-xloc[2])*(yloc[1]-yloc[3]));
                w2 =((yloc[3]-yloc[1])*(XP2[m]-xloc[3])+(xloc[1]-xloc[3])*(YP2[m]-yloc[3]))/
                   ((yloc[2]-yloc[3])*(xloc[1]-xloc[3])+(xloc[3]-xloc[2])*(yloc[1]-yloc[3]));
                w3 = 1-w1-w2;
                Pelv2[m] = (w1*elv[NE[e][1]]+w2*elv[NE[e][2]]+w3*elv[NE[e][3]])/(w1+w2+w3);
                            if(Pelv2[m] > 0){
                                Phold2[m] = Pelv2[m];     //The last wet
point
                            }
                        break;          //out of for elements loop
                        }               //close if(crossprod > 0)
                    }                   //close all element search for loop
//cout<<"!!!!contELM2: "<<contELM2<<"   emin:  "<<emin2<<"   emax: "<<emax2<<endl;
                }                   //close if contELM2 == false
//if containing element still false, out of bounds
                    if(contELM2 == false){
                        Pelv2[m]=0;
                    }
                }                   //close while Pelv (XP,YP) > 0 loop
                outfile7<<dephold2.size()<<endl;
                for(int h=0; h<dephold2.size(); h++){
                    outfile7<<dephold2[h]<<endl;
                }
                dpast2[m] = Phold2[m]/((Phold2[m] + abs(Pelv2[m]))/res);
                dlon2 = abs(XP2[m] - lon0);
                dlat2 = abs(YP2[m] - lat0);
                dhav2 = 2*Rearth*asin(sqrt(sin(((dlat2)/2)*pi/180)*
                                            sin(((dlat2)/2)*pi/180)+
                                    (cos(lat0*pi/180)*cos(YP2[m]*pi/180)*
                                            sin(((dlon2)/2)*pi/180)*
                                            sin(((dlon2)/2)*pi/180))))-res+dpast2[m];
                if(step2 == 1){
                    dhav2=dpast2[m];
                    if(contELM2 == false){
                        dhav2=0;
                    }
                }
                if(step2!=1 && contELM2 ==false){
                    dhav2 = 2*Rearth*asin(sqrt(sin(((dlat2)/2)*pi/180)*
                                                sin(((dlat2)/2)*pi/180)+
                                        (cos(lat0*pi/180)*cos(YP2[m]*pi/180)*
                                                sin(((dlon2)/2)*pi/180)*
                                                sin(((dlon2)/2)*pi/180))))-(res/2);
                }
                outfile6<<dhav2<<" ";
            }                   //close for m
        }                   //close if wet
        outfile6<<endl;
        outfile8<<endl;
    }                           //close for all nodes loop
outfile6.close();
outfile7.close();
outfile8.close();
    time_req=clock()- time_req;
    cout<<"***Time taken: "<<(float)time_req/CLOCKS_PER_SEC<<" seconds***"<<endl;
//**************Begin Averaging Schemes***************************************************************************
    ifstream infile1;
```

```
        infile1.open("Distance_data_2_deg.txt");                    //reading in fetch data
        float time_req2;
        time_req2=clock();                          //initializing the clock for timing tasks
        c=0;
        if(infile1.is_open()){
                cout<<"Reading Fetch Data"<<endl;
                while(getline(infile1,s)){
                        line=line+1;
                        if(line>3 && line<node_tot+3){          //ignoring header lines
                                if(s.length()>7){               //if the length of line > 7 characters
                                        stringstream ss;
                                        ss<<s;                          //read the line as string (s)
                                        ss>>content;                    //read number by number from s as
(content)
                                        x=0;                            //indexing variable reset to 0
                                        while(!ss.eof()){               //while values in the line (string ss)
                                                ss>>temp;       //read each value
                                                x=x+1;                  //indexing variable
                                                if(x==1){       //if first number, store value as node number
                                                        node1=atoi(content);
                                                        //cout<<"NODE: "<<node<<endl;
                                                }
                                                if(x<=360/Angres){//if any of the next 36 numbers, store as fetch
                                                        fetch[node1][x]=atof(temp);
                                                }
                                        }               //close while !eof for ss
                                }               //close if s.length>7
                        }               //close if header >3 and <126773
                }               //close while getline for "Distance_data4.txt"
        }               //close if "Distance_data4.txt" is open
// AVG fetch
        for(int j=1; j<=node_tot; j++){
                Ftop=0;
                for(int i=1; i<=360/Angres; i++){
                        Ftop=Ftop+fetch[j][i];
                }
                Ftop=Ftop/(360/Angres);
                Ftop2=Ftop2+Ftop;
                c=c+1;
        }
        Favg=Ftop2/c;
        cout<<"Average Fetch in domain: "<<Favg<<" m"<<endl;
        time_req2=clock()- time_req2;
        cout<<"***Time taken: "<<(float)time_req2/CLOCKS_PER_SEC<<" seconds***"<<endl;
//****************Reading in Depth Data and computing IDW depth
        line=0;
        ifstream infile2;
        infile2.open("Depth_data_5_deg.txt");                       //reading in depth data
        outfile9<<"p-values determined from variable calculation with U=30"<<endl;
        float time_req3;
        time_req3=clock();
        c=0;
        if(infile2.is_open()){
                cout<<"Reading Depth Data and Computing Averages with p: "<<p<<endl;
                while(getline(infile2,s)){
                        line=line+1;
                        if(line==2){    //second line is where node total and direction total are stored
                                stringstream ss;
                                ss<<s;
                                ss>>content;
                                node_tot=atof(content);
                                //cout<<"Node_tot: "<<node_tot<<endl;
                                ss>>temp;
                                dir_tot=atof(temp);
                                //cout<<"Dir_tot: "<<dir_tot<<endl;
                        }
//x2 indexes each line of Depth_data from 0 to num_depths and resets after every direction
//x1 indexes the number of values per line after the first value
//format is:    node   dir.index   #steps
```

```
//                depth at step1
//                depth at step2
//                       ...
//                depth at stepN (node before land)
//      repeat for next dir. index, then once all directions are performed, repeat for next node
                        if(line>2){ //for all other lines
                                x2=x2+1;        //indexing variable 2 progress
                                x1=0;                   //indexing variable 1 reset when there is a new line
                                stringstream ss;
                                ss<<s;
                                ss>>content;
                                while(!ss.eof()){
                                        x2=0;           //indexing varriable 2 reset when there is a new
value
                                        ss>>temp;       //read in each value of ss
                                        x1=x1+1;        //progress indexing variable 1
                                        check=false;    //reset boolean check value to false
                                        if(x1==1){      //first value is direction (1:4*Ares)
                                                direction=atof(temp);
                                                check=true;  //if the direction value is read, check is reset
                                                    //to true because you know a new node has been reached
                                        }
                                        if(check=true && depth_tot!=0){                 //if a new node has been
reached
                                                direction=direction-1;                  //reset direction to the
previous
                                                if(direction==0){                       //if==0 reset to the last
value
                                                        direction=360/Angres;
                                                }
                                        //depth averaging scheme
                                                d[node1][direction]=depth_tot/denominator;
                                                depth_tot=0;            //resetting depth_tot to 0 once value is
stored
                                                denominator=0;
                                        }
                                        if(x1==2){      //second value is number of depths(needed for averaging)
                                                num_depths=atof(temp);
                                        }
                                }               //close while !ss.eof
                                if(x2==0){              //reading in node number
                                        node1=atof(content);
                                        outfile9<<endl;
                                        outfile9<<node1<<"  ";
                                        //cout<<"N: "<<node<<endl;
                                }
                                if(x2==1){              //first depth value special case
                                        depth_temp = atof(content);
//******************************************************************************************
//variable p-value, CAN be implemented at this step. Presently 1 constant p-value is taken for all fetches
//*****************************************************************************************
                                        p=p_temp;
                                        if(p_temp==0){
                                //p=log((10924*exp(-0.064*Up))/depth_temp) / log(fetch[node1][direction]);
                                        }
                                        if(p < 0){      //occurs when depth is set to -99999
                                                if(p_temp==0){
                                //p=log((10924*exp(-0.064*Up))/depth_temp) / log(fetch[node1][direction]);
                                                }
                                                if(p_temp!=0){
                                                        p=p_temp;
                                                }
                                        //cout<<"p<0 at ["<<node<<"]["<<direction<<"]"<<endl;
                                        }
                                        outfile9<<p<<" ";
                                        depth_tot = (2*depth_temp)/(pow(res,p));
                                        denominator = 2/(pow(res,p));
                                        //cout<<"depth["<<x2<<"]: "<<depth_temp<<endl;
                                }
```

```
						if(x2>2){						//skip 2nd line, built into special case, read 3rd line+
in the depths and sum together for average
							depth_temp = atof(content);
							depth_tot = depth_tot+(depth_temp/(pow(((x2-1)*res),p)));
							denominator = denominator+(1/(pow(((x2-1)*res),p)));
							//cout<<"depth["<<x2<<"]: "<<depth_temp<<endl;
						}
					}		//close if line > 2
				}		//close while getline
	}		//close if infile2 is_open
outfile9.close();
// AVG depth
	Ftop2=0;
	c=0;
	for(int j=1; j<=node_tot; j++){
		Ftop=0;
		for(int i=1; i<=360/Angres; i++){
			Ftop=Ftop+d[j][i];
		}
		Ftop=Ftop/(360/Angres);
		Ftop2=Ftop2+Ftop;
		c=c+1;
	}
	Favg=Ftop2/c;
	cout<<"Average Depth in domain: "<<Favg<<" m"<<endl;
	time_req3=clock()- time_req3;
	cout<<"***Time taken: "<<(float)time_req3/CLOCKS_PER_SEC<<" seconds***"<<endl;
//storing depth avgs in an output file
	for(int i=1; i<=node_tot; i++){
		outfile2<<i<<"  ";
		for(int j=1; j<=360/Angres; j++){
			outfile2<<d[i][j]<<" ";
		}
		outfile2<<endl;
	}
	outfile2.close();
// AVG bottom slope
	Ftop2=0;
	c=0;
	for(int j=1; j<=node_tot; j++){
		Ftop=0;
		for(int i=1; i<=360/Angres; i++){
			Ftop=Ftop+slopehold[j][i];
		}
		Ftop=Ftop/(360/Angres);
		Ftop2=Ftop2+Ftop;
		c=c+1;
	}
	Favg=Ftop2/c;
	cout<<"Average slope in domain: "<<Favg<<endl;
//******Effective fetch calculation*******
	cout<<"Calculating Effective Fetch with res: "<<Eff_res<<" deg"<<endl;
	outfile1<<node_tot<<"  "<<Angres<<endl;
	float time_req5;
	time_req5=clock();
	for(int j=1; j<=node_tot; j++){					//for all nodes
		//cout<<"Node(j): "<<j<<endl;
		outfile1<<j<<"  ";
		for(int i=1; i<=360/Angres; i++){	//for all angles
			k2=0;						//reset all counting and temp variables
			k1=0;
			Eff_top=0;
			Eff_bot=0;
			//cout<<"Angle(i): "<<i<<endl;
			if(Eff_res==0){					//case for no effective resolution (use raw fetches)
				F[j][i]=fetch[j][i];
				outfile1<<F[j][i]<<" ";
			}
			else{						//case for any effective resolution
```

```
                                                  F[j][i]=0;
                                        for(int n=-floor(Eff_res/Angres); n<=floor(Eff_res/Angres); n++){
//rounding down to the nearest integer division for direction given the Effective fetch resolution specified
                                                  //cout<<"k0: "<<i+n<<endl;
                                                  check=false;                                        //resetting special case check value
          //case for start nodes (1:4) when <0 is encountered
                                                  if(i+n < 1 && i+n+(360/Angres)>k1){    //direction within Eff_res deg left or right of
                                                                                  //the current node. for values that go below 1
                                                            k1=i+n+(360/Angres);      //reset index to (33:36)
                                                            //cout<<"k1: "<<k1<<endl;
                                                            //cosine weighted average numerator and denominator, as summations
                                                            Eff_top = fetch[j][k1]*cos(n*Angres*pi/180) + Eff_top;
                                                            Eff_bot = cos(n*Angres*pi/180) + Eff_bot;
                                                            check=true;               //checks for this special case
                                                  }
          //case for start nodes (33:36) when >36 is encountered
                                                  if(i+n > 360/Angres && i+n-(360/Angres)>k2){ //for values that go above max direction
                                                            k2=i+n-(360/Angres);      //reset index to (1:4)
                                                            //cout<<"k2: "<<k2<<endl;
                                                            Eff_top = fetch[j][k2]*cos(n*Angres*pi/180) + Eff_top;
                                                            Eff_bot = cos(n*Angres*pi/180) + Eff_bot;
                                                            check=true;               //checks for this special case
                                                  }
          //case for all other nodes(normal +/- 45 deg on each side of start node
                                                  if(check==false){                        //if neither special case, than regular case
                                                            k=i+n;
                                                            //cout<<"k : "<<k<<endl;
                                                            Eff_top = fetch[j][k]*cos(n*Angres*pi/180) + Eff_top;
                                                            Eff_bot = cos(n*Angres*pi/180) + Eff_bot;
                                                  }
                                        }         //close for -4<n<4 loop
                                        }         //close else statement
                                        if(Eff_res!=0){
                                                  Eff_fetch = Eff_top/Eff_bot;             //devide sum top/sum bot for Effective fetch
                                                  F[j][i]=Eff_fetch;
                                                  outfile1<<F[j][i]<<" ";
                                        }
                              //cout<<"fetch: "<<fetch[j][i]<<"  Eff_fetch: "<<Eff_fetch<<endl;
                              }         //close for 1<i<36 loop
                              outfile1<<endl;
          }         //close for all nodes loop
          outfile1.close();
          time_req5=clock()- time_req5;
          cout<<"***Time taken: "<<(float)time_req5/CLOCKS_PER_SEC<<" seconds***"<<endl;
return 0;
}
```

C++ code: param_wave.cpp

```
//Code for reading in the fetch and depth data created by pre-proccessing parametric code.
//Here wave heights for each of the 4 formulations (WEMO,SPM,TMA,and CEM) are calculated based on
//the wind forcing provided (ADCIRC .22 file) in addition to the physical processes of friction,breaking,and
//shoaling as computed from each formulation's period.
//The final results are 5 global elevation files (ADCIRC .63 files), one for each parametric formulation, and
//one for the ensemble average.
//
//By: Samuel Boyd
//Date: 04/13/2020

#include <cmath>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>
#include <stdlib.h>
#include <cstdlib>
#include <math.h>
#include <vector>
#include <sstream>
#include <algorithm>
#include <ctime>
#include <stdio.h>
#include <string.h>
#define pi 3.14159265

using namespace std;

//**********************************************************************************
//Here the manning's n values which correspond to optimal model performance from
//previous tests are used. WEMO does not have any additional physical formulations
double mannings_n_SPM=0.01;
double mannings_n_TMA=0.01;
double mannings_n_CEM=0.02;
//**********************************************************************************
char content[10],temp[10];
int x,i,n,di_low,di_high,node_tot,Angres,dir,dir_tot,c,node;
double g=9.7918;		//gravity at meanlat (28.1091 deg)
int NDSETSE,NP,DTDP,NSPOOLGE,IRTYPE,TIME,IT;
bool check;
string s;
double Hw,Hwabove,Hwbelow,u,di_temp,di_input,y3,PRN, HWEMO,HSPM,HCEM,HTMA,WSPM,TSPM,TCEM,TTMA,Tw;
double HwaboveWEMO,HwbelowWEMO,HwaboveSPM,HwbelowSPM,HwaboveTMA,HwbelowTMA,HwaboveCEM,HwbelowCEM;
double TaboveSPM,TbelowSPM,TaboveTMA,TbelowTMA,TaboveCEM,TbelowCEM;
double WEMO_shoal,WEMO_fric,WEMO_cum,H_break;
double kSPM,LSPM,ksSPM,kfSPM,SPM_shoal,SPM_fric,SPM_cum;
double kCEM,LCEM,ksCEM,kfCEM,CEM_shoal,CEM_fric,CEM_cum;
double kTMA,LTMA,ksTMA,kfTMA,TMA_shoal,TMA_fric,TMA_cum;
double ENS_cum,H_break_SPM;
double sigma;
double tol;
double eps;
double knew,kold,k0;
double k;
double L;
double ks,kf,Tm,fm;
int line;
int Ftop,Ftop2,Favg,v=0;
double Dtop,Davg,dconst=0;
//*****************TMA formulation (EQ 19 from TMA paper)
double WaveTMA(double g, double f, double d, double u){
		double alpha;
		alpha = 0.076*pow( ((g*f)/(u*u)),-0.22);
		double fm;
		fm = 3.5*(g/u)*pow( (g*f/(u*u)),-0.33);
		double ET;
		ET = alpha*g*d/(16*pi*pi*0.9*0.9)*1/(fm*fm);
		double WTMA;
		WTMA = 4*pow(ET,0.5);
return WTMA;
}
double PeriodTMA(double g, double f, double d, double u){
		double fm = 3.5*(g/u)*pow(((g*f)/(u*u)),-0.33);
		TTMA = 1/fm;
return TTMA;
}
//***************CEM formulation (Eq II-2-36 in CEM)
double WaveCEM(double g, double f, double u){
		double Cd;
		Cd = 0.001*(0.009042*u + (-4.44*pow(10,-8)*f*f + 3.56*pow(10,-4)*f + 1.10949));		//Colvin
		//Cd = 0.001*(1.1+0.035*u);
		double Ustsq;
		Ustsq = Cd*u*u;
		double WCEM;
		WCEM = (Ustsq/g)*0.0413*pow((g*f/Ustsq),0.5);
return WCEM;
}
double PeriodCEM(double g, double f, double u){
		double Cd;
		Cd = 0.001*(0.009042*u + (-4.44*pow(10,-8)*f*f + 3.56*pow(10,-4)*f + 1.10949));		//Colvin
		//Cd = 0.001*(1.1+0.035*u);
		double Ustsq;
		Ustsq = u*u*Cd;
		double TCEM;
		TCEM = (pow(Ustsq,0.5)/g)*0.651*pow((g*f/Ustsq),0.3333333);
		//if(node==34283){
		//cout<<" Cd: "<<Cd<<" Ustsq: "<<Ustsq<<" TCEM: "<<TCEM<<endl;
		//}
return TCEM;
}
//**************SMB formulation in WEMO paper
double WaveWEMO(double g, double f, double d, double u){			//from WEMO
		Hw = 0.283*tanh(0.530*pow((g*d/(u*u)),0.75))*tanh((0.0125*pow((g*f/(u*u)),0.42))/
			tanh(0.530*pow((g*d/(u*u)),0.75)))*(u*u)/g;
return Hw;
}
//*************SMB from SPM H and T formulations Eqs (3-39) and (3-40)
double PeriodSPM(double g, double f, double d, double u){
		double Ua = 0.713*pow(u,1.23);
		Tw =  7.54*tanh(0.833*pow((g*d/(Ua*Ua)),0.375))*tanh((0.0379*pow((g*f/(Ua*Ua)),(0.333333)))/
			tanh(0.833*pow((g*d/(Ua*Ua)),0.375)))*(Ua/g);
return Tw;
}
double WaveSPM(double g, double f, double d, double u){					//from SPM
		double Ua = 0.713*pow(u,1.23);
		WSPM = 0.283*tanh(0.530*pow((g*d/(Ua*Ua)),0.75))*tanh((0.00565*pow((g*f/(Ua*Ua)),0.50))/
			 tanh(0.530*pow((g*d/(Ua*Ua)),0.75)))*(Ua*Ua)/g;
return WSPM;
}
//**************Linear Interpolation Function
double Interp(double x1, double x2, double x3, double y1, double y2){
		y3 = ((x2-x3)*y1 + (x3-x1)*y2) / (x2-x1);
return y3;
}
int main(){
		float clock_t,time_req;
		time_req=clock();					//initializing the clock for timing tasks
		ifstream infile1;
		infile1.open("Eff_fetch_2_deg.txt");			//reading in fetch data
		if(infile1.is_open()){
			cout<<endl;
			cout<<"Reading in fetch data"<<endl;
		}
		else{
			cout<<endl;
```

```
				cout<<"**************ERROR Fetch input file not read"<<endl;
				cout<<endl;
		}
//******These numbers need to be read in from EFF_fetch, make the format as such
		x=0;
		while(infile1 >> content){			//reading headers
				x=x+1;							//content counter;
				if(x==1){
						node_tot=atoi(content);
				}
				if(x==2){
						Angres=atoi(content);
						dir_tot=360/Angres;
						break;
				}
		}
		cout<<"Node total: "<<node_tot<<endl;
		cout<<"Angular resolution: "<<Angres<<endl;
		cout<<"Number of angles: "<<dir_tot<<endl;
		double F[node_tot+1][dir_tot];
		double d[node_tot+1][dir_tot];
		double WVX[node_tot+1];
		double WVY[node_tot+1];
		double WDIR[node_tot+1];
		double WVEL[node_tot+1];
		double slope[node_tot][dir_tot];
		infile1.close();
		infile1.open("Eff_fetch_2_deg.txt");
		x=-3;
		n=0;
		while(infile1 >> content){
				x=x+1;
				if(x%(dir_tot+1)==0){
						n=n+1;						//node counter increment
						i=-1;						//direction counter reset at new node
				}
						i=i+1;						//direction counter increment
						F[n][i]=atof(content);			//fetch values
				//if(n<15){
				//		cout<<"F["<<n<<"]["<<i<<"]: "<<F[n][i]<<endl;
				//}
		}
		infile1.close();
		ifstream infile2;
		infile2.open("IDW_depths_2_deg.txt");			//reading in depth data
		if(infile2.is_open()){
				cout<<endl;
				cout<<"Reading in depth data"<<endl;
		}
		else{
				cout<<endl;
				cout<<"**************ERROR IDW depth input file not read"<<endl;
				cout<<endl;
		}
		n=0;								//reset counters
		x=-1;
		i=0;
		while(infile2 >> content){
				x=x+1;							//content counter;
				if(x%(dir_tot+1)==0){
						n=n+1;						//node counter increment
						i=-1;						//direction counter reset at new node
								//dir index (i) = 0 corresponds to node number
				}
						i=i+1;						//direction counter increment
						d[n][i]=atof(content);			//depth values
				if(n==126765 || n==126769 || n==126770 || n==126771 || n==126772){	//for nan values
						d[n][i]=0;
						//cout<<n<<endl;
		}
	}
	infile2.close();
	ifstream infile3;					//reading wind field data
	infile3.open("PARAM_const_U_30.22");
	if(infile3.is_open()){
		cout<<endl;
		cout<<"Reading in wind data (fort.22)"<<endl;
	}
	else{
		cout<<endl;
		cout<<"**************ERROR wind input file not read"<<endl;
		cout<<endl;
	}
	NDSETSE=1;
	x=0;
	while(infile3 >> content){		//counting NDSETSE number of time steps
		x=x+1;
		if(x<=node_tot*4){
		}
		else{
			x=1;
			NDSETSE=NDSETSE+1;
		}
	}
	infile3.close();
	cout<<"NDSETSE: "<<NDSETSE<<endl;
//avg bottom slope
	ifstream infile4;
	infile4.open("Slope_upwind_2_deg.txt");			//reading in bottom slope data
	x=-1;
	n=0;
	while(infile4 >> content){
		x=x+1;
		if(x%(dir_tot+1)==0){
			n=n+1;					//node counter increment
			i=-1;					//direction counter reset at new node
		}
			i=i+1;					//direction counter increment
			slope[n][i]=atof(content);		//fetch values
		//if(n<15){
			//cout<<"slope["<<n<<"]["<<i<<"]: "<<slope[n][i]<<endl;
		//}
	}
	infile4.close();
	NP=node_tot;		//# of nodes
	DTDP=1;			//ADCIRC time step in seconds
	NSPOOLGE=3600;		//output is written to fort.63 every NSPOOLGE time steps
	IRTYPE=1;	//the record type (1 for elevation files, 2 for velocity files)
	TIME=1;			//model time (in seconds) (TIME = STATIM*86400 + IT*DTDP)
	IT=1;			//model time step number since the beginning of the model run
	ofstream outfile14("WEMO_Hcum_const_wind.63");
	outfile14<<"Wave Height SMB TEST:3  U="<<u<<"(m/s)  Dir="<<di_temp<<"(deg N)"<<" from Mesh.grd"<<endl;
	outfile14<<NDSETSE<<"     "<<NP<<"     "<<DTDP*(NSPOOLGE)<<"     "<<NSPOOLGE<<"     "<<IRTYPE<<endl;
	outfile14<<NSPOOLGE<<"  "<<NSPOOLGE<<endl; //should be TIME<<"  "<<IT<<endl; if they are not equal
	ofstream outfile15("TMA_Hcum_const_wind.63");
	outfile15<<"Wave Height SMB TEST:3  U="<<u<<"(m/s)  Dir="<<di_temp<<"(deg N)"<<" from Mesh.grd"<<endl;
	outfile15<<NDSETSE<<"     "<<NP<<"     "<<DTDP*(NSPOOLGE)<<"     "<<NSPOOLGE<<"     "<<IRTYPE<<endl;
	outfile15<<NSPOOLGE<<"  "<<NSPOOLGE<<endl; //should be TIME<<"  "<<IT<<endl; if they are not equal
	ofstream outfile16("CEM_Hcum_const_wind.63");
	outfile16<<"Wave Height SMB TEST:3  U="<<u<<"(m/s)  Dir="<<di_temp<<"(deg N)"<<" from Mesh.grd"<<endl;
	outfile16<<NDSETSE<<"     "<<NP<<"     "<<DTDP*(NSPOOLGE)<<"     "<<NSPOOLGE<<"     "<<IRTYPE<<endl;
	outfile16<<NSPOOLGE<<"  "<<NSPOOLGE<<endl; //should be TIME<<"  "<<IT<<endl; if they are not equal
	ofstream outfile17("SPM_Hcum_const_wind.63");
	outfile17<<"Wave Height SMB TEST:3  U="<<u<<"(m/s)  Dir="<<di_temp<<"(deg N)"<<" from Mesh.grd"<<endl;
	outfile17<<NDSETSE<<"     "<<NP<<"     "<<DTDP*(NSPOOLGE)<<"     "<<NSPOOLGE<<"     "<<IRTYPE<<endl;
	outfile17<<NSPOOLGE<<"  "<<NSPOOLGE<<endl; //should be TIME<<"  "<<IT<<endl; if they are not equal
	ofstream outfile18("ENS_Hcum_const_wind.63");
	outfile18<<"Wave Height SMB TEST:3  U="<<u<<"(m/s)  Dir="<<di_temp<<"(deg N)"<<" from Mesh.grd"<<endl;
```

```
outfile18<<NDSETSE<<"     "<<NP<<"     "<<DTDP*(NSPOOLGE)<<"     "<<NSPOOLGE<<"     "<<IRTYPE<<endl;
outfile18<<NSPOOLGE<<"  "<<NSPOOLGE<<endl; //should be TIME<<"  "<<IT<<endl; if they are not equal
cout<<"Time Step: "<<IT<<" at "<<NSPOOLGE<<" seconds"<<endl;
infile3.open("PARAM_const_U_30.22");                                    //opening wind field data again
//ofstream outfile3("fort.22_out");
x=0;
c=0;
node=0;
//int Q=0;
//while(infile3 >> content){
//if(infile3.is_open()){
x=0;
c=0;
node=0;
line=1;
IT=1;
int Q=0;
int y=0;
HWEMO=-99999;
HSPM= -99999;
HCEM= -99999;
HTMA= -99999;
while(getline(infile3,s)){
        //outfile3<<s<<endl;
        node=node+1;
        if(node<=126772){
                stringstream ss1;
                ss1<<s;                                 //read the line as string (s)
                ss1>>content;
                y=0;
                //if(node<15 && IT==1){
                        //cout<<s<<endl;
                //}
                //cout<<s<<endl;
                while(!ss1.eof()){      //while values in the line (string ss)
                        y=y+1;
                        ss1>>temp;  //read each value
                        if(y==1){
                                WVX[node]=atof(temp);
                        }
                        if(y==2){
                                WVY[node]=atof(temp);
                        }
                        if(y==3){
                                PRN=atof(temp);
                                WVEL[node] = sqrt(WVX[node]*WVX[node] + WVY[node]*WVY[node]);
                                if(WVX[node] <= 0 && WVY[node] < 0){                    //Q1 [0,90)
                                        WDIR[node] = atan(WVX[node]/WVY[node]) * 180/pi;
                                        //WDIR[node] = 180+atan(WVX[node]/WVY[node]) * 180/pi;
                                }
                                if(WVX[node] < 0 && WVY[node] >= 0){                    //Q4 [90,180)
                                        WDIR[node] = 90+atan(abs(WVY[node]/WVX[node]))*180/pi;
                                        //WDIR[node] = 270+atan(abs(WVY[node]/WVX[node]))*180/pi;
                                }
                                if(WVX[node] >= 0 && WVY[node] > 0){                    //Q3 [180,270)
                                        WDIR[node] = 180+atan(WVX[node]/WVY[node]) * 180/pi;
                                        //WDIR[node] = atan(WVX[node]/WVY[node]) * 180/pi;
                                }
                                if(WVX[node] > 0 && WVY[node] <= 0){                    //Q2 [270,360)
                                        WDIR[node] = 270+atan(abs(WVY[node]/WVX[node]))*180/pi;
                                        //WDIR[node] = 90+atan(abs(WVY[node]/WVX[node]))*180/pi;
                                }
                                u = WVEL[node];
                                di_input=WDIR[node]/Angres;//di_input is interpolation direction index
                                di_low=floor(di_input);    //rounds down to the direction index below
                                di_high=ceil(di_input);    //rounds up to the direction index above
                                //casting the double to int for the evenly divisible case
                                dir = int(di_input);
        //if converged (evenly divisible case) no interpolation
```

```
if(node==1){
//cout<<"BEFORE, WDIR: "<<WDIR[node]<<"  di_input: "<<di_input<<"  di_low: "<<di_low<<"  di_high: "<<di_high<<"  dir: "<<dir<<" IT:
"<<IT<<endl;
//cout<<"dir: "<<dir<<endl;
}
                                        //if( abs(((WDIR[node]+Angres)/Angres)-(2*IT)) < 0.01 || di_low==di_high){
                                        if(abs(di_input-di_low) < 0.01 || abs(di_input-di_high) < 0.01){
                                                        if(node==1){
                                                cout<<"NO interpolation at time "<<IT;
                                                        }
                                                        if(abs(di_input-di_low) < 0.01){          //round down
                                                                WDIR[node]=floor(WDIR[node]);
                                                        }
                                                        if(abs(di_input-di_high) < 0.01){         //round up
                                                                WDIR[node]=ceil(WDIR[node]);
                                                        }
                                                        di_input=WDIR[node]/Angres;
                                                        //+1 is because 0 deg N corresponds to index 1
                                                        dir = int(di_input)+1;
if(node==1){
cout<<"          WVEL: "<<WVEL[node]<<"  WDIR: "<<WDIR[node]<<endl;
//cout<<endl;
//cout<<"F["<<node<<"]["<<dir<<"]: "<<F[node][dir]<<"  d["<<node<<"]["<<dir<<"]: "<<d[node][dir]<<endl;
//cout<<endl;
//cout<<"AFTER, WDIR: "<<WDIR[node]<<"  di_input: "<<di_input<<"  di_low: "<<di_low<<"  di_high: "<<di_high<<"  dir: "<<dir<<endl;
}
                                                        HWEMO = WaveWEMO (g,F[node][dir],d[node][dir],u);
                                                        HSPM  = WaveSPM  (g,F[node][dir],d[node][dir],u);
                                                        HCEM  = WaveCEM  (g,F[node][dir],u);
                                                        HTMA  = WaveTMA  (g,F[node][dir],d[node][dir],u);
                                                        TSPM  = PeriodSPM(g,F[node][dir],d[node][dir],u);
                                                        TCEM  = PeriodCEM(g,F[node][dir],u);
                                                        TTMA  = PeriodTMA(g,F[node][dir],d[node][dir],u);
                                                        if(F[node][dir]==0 || d[node][dir]==0 || HWEMO < 0.0001){
                                                                HWEMO=-99999;
                                                                HSPM =-99999;
                                                                HTMA =-99999;
                                                                HCEM =-99999;
                                                        }
                                                }               //close if converged (evenly divisible no interpolation)
//*****case for an angle in between (not evenly divisible by Angres) using linear interpolation
                                                else{
//case for setting maxindex(360)back to 0 deg
                                                        if(WDIR[node]>360-Angres){
                                                                di_high=1;
                                                        }
                                                        if(node==1){
                        cout<<"     Interpolation between "<<di_low*Angres<<" and "<<di_high*Angres<<" degrees"<<endl;
//cout<<"F[node][di_high+1]: "<<F[node][di_high+1]<<" F[node][di_low+1]: "<<F[node][di_low+1]<<endl;
//cout<<"d[node][di_high+1]: "<<d[node][di_high+1]<<" d[node][di_low+1]: "<<d[node][di_low+1]<<endl;
//cout<<"WDIR: "<<WDIR[node]<<"  di_input: "<<di_input<<"  di_low: "<<di_low<<"  di_high: "<<di_high<<"  dir: "<<dir<<endl;
                                                        }
                                                HwaboveWEMO = WaveWEMO(g,F[node][di_high+1],d[node][di_high+1],u);
                                                HwbelowWEMO = WaveWEMO(g,F[node][di_low+1],d[node][di_low+1],u);
                                                HwaboveSPM  = WaveSPM(g,F[node][di_high+1],d[node][di_high+1],u);
                                                HwbelowSPM  = WaveSPM(g,F[node][di_low+1],d[node][di_low+1],u);
                                                HwaboveTMA  = WaveTMA(g,F[node][di_high+1],d[node][di_high+1],u);
                                                HwbelowTMA  = WaveTMA(g,F[node][di_low+1],d[node][di_low+1],u);
                                                HwaboveCEM  = WaveCEM(g,F[node][di_high+1],u);
                                                HwbelowCEM  = WaveCEM(g,F[node][di_low+1],u);
                                                TaboveSPM = PeriodSPM (g,F[node][di_high+1],d[node][di_high+1],u);
                                                TbelowSPM = PeriodSPM (g,F[node][di_low+1],d[node][di_low+1],u);
                                                TaboveTMA = PeriodTMA (g,F[node][di_high+1],d[node][di_high+1],u);
                                                TbelowTMA = PeriodTMA (g,F[node][di_low+1],d[node][di_low+1],u);
                                                TaboveCEM = PeriodCEM (g,F[node][di_high+1],u);
                                                TbelowCEM = PeriodCEM (g,F[node][di_low+1],u);
//resetting index to 36 to account for di_high
                                                        if(di_temp>360-Angres){
                                                                di_high=360/Angres;
                                                        }
                                                        HWEMO =
Interp(di_low,di_high,di_input,HwbelowWEMO,HwaboveWEMO);
                                                        HSPM  =
Interp(di_low,di_high,di_input,HwbelowSPM,HwaboveSPM);
                                                        HTMA  =
Interp(di_low,di_high,di_input,HwbelowTMA,HwaboveTMA);
                                                        HCEM  =
Interp(di_low,di_high,di_input,HwbelowCEM,HwaboveCEM);

                                                        TSPM  = Interp(di_low,di_high,di_input,TbelowSPM,TaboveSPM);
                                                        TTMA  =
Interp(di_low,di_high,di_input,TbelowTMA,TaboveTMA);

                                                        TCEM  = Interp(di_low,di_high,di_input,TbelowSPM,TaboveCEM);
                                                        if(F[node][dir]==0 || d[node][dir]==0 || HWEMO < 0.0001){
                                                                HWEMO=-99999;
                                                                HSPM =-99999;
                                                                HTMA =-99999;
                                                                HCEM =-99999;
                                                        }
                                                        //outfile3<<node<<"  "<<HWEMO<<endl;
                                                }   //close interpolation else statement
                                                check=false;
                                                if(F[node][dir]==0 || d[node][dir]==0 || HWEMO < 0.0001){
                                                        HWEMO=-99999;
                                                        HSPM= -99999;
                                                        HCEM= -99999;
                                                        HTMA= -99999;
                                                        WEMO_cum=-99999;
                                                        TMA_cum =-99999;
                                                        SPM_cum =-99999;
                                                        CEM_cum =-99999;
                                                        ENS_cum =-99999;
                                                        TSPM=0;
                                                        TCEM=0;
                                                        TTMA=0;
                                                        check=true;
                                                }
//*******Dispersin Equation for k and L
                                                if(check==false){//wet node with non-zero fetch and depth values
                                                        //SPM
                                                        Tm=TSPM;
                                                        fm=1/Tm;
                                                        sigma=2*pi*fm;
                                                        k0=sigma*sigma/g;
                                                        kold=k0;
                                                        eps=10;
                                                        int b=0;
                                                        while(eps>0.0001){
                                                                knew=sigma*sigma/(g*tanh(kold*d[node][dir]));
                                                                eps=abs(knew-kold);
                                                                kold=knew;
                                                                b=b+1;
                                                                if(b>1000000){          //statement for no
convergence

                                                break;
                                                                }
                                                        }
                                                        kSPM=knew;
                                                        LSPM=2*pi/kSPM;
                                                        //CEM
                                                        Tm=TCEM;
                                                        fm=1/Tm;
                                                        sigma=2*pi*fm;
                                                        k0=sigma*sigma/g;
                                                        kold=k0;
                                                        eps=10;
                                                        b=0;
                                                        while(eps>0.0001){
                                                                knew=sigma*sigma/(g*tanh(kold*d[node][dir]));
```

```
                                                                                    eps=abs(knew-kold);
                                                                                    kold=knew;
                                                                                    b=b+1;
                                                                                    if(b>1000000){
                                                                                                break;
                                                                                    }
                                                                        }
                                                                        kCEM=knew;
                                                                        LCEM=2*pi/kCEM;
                                                                        //TMA
                                                                        Tm=TTMA;
                                                                        fm=1/Tm;
                                                                        sigma=2*pi*fm;
                                                                        k0=sigma*sigma/g;
                                                                        kold=k0;
                                                                        eps=10;
                                                                        b=0;
                                                                        while(eps>0.0001){
                                                                                    knew=sigma*sigma/(g*tanh(kold*d[node][dir]));
                                                                                    eps=abs(knew-kold);
                                                                                    kold=knew;
                                                                                    b=b+1;
                                                                                    if(b>1000000){
                                                                                                break;
                                                                                    }
                                                                        }
                                                                        kTMA=knew;
                                                                        LTMA=2*pi/kTMA;
double Ch_SPM=(1.486/mannings_n_SPM)*pow(d[node][dir],1/6);
double Ch_TMA=(1.486/mannings_n_TMA)*pow(d[node][dir],1/6);
double Ch_CEM=(1.486/mannings_n_CEM)*pow(d[node][dir],1/6);
double ffactor_SPM=g/(Ch_SPM*Ch_SPM);
double ffactor_TMA=g/(Ch_TMA*Ch_TMA);
double ffactor_CEM=g/(Ch_CEM*Ch_CEM);
//shoaling coefficient ks (varies from 1 in deep water, dips to 0.9 in intermediate and grows larger than 1 as it approaches shallow water.
Multiply H by this)
ksCEM = pow(((2*pow(cosh(kCEM*d[node][dir]),2)) / (sinh(2*kCEM*d[node][dir]) + 2*kCEM*d[node][dir]) ),0.5);
ksSPM = pow(((2*pow(cosh(kSPM*d[node][dir]),2)) / (sinh(2*kSPM*d[node][dir]) + 2*kSPM*d[node][dir]) ),0.5);
ksTMA = pow(((2*pow(cosh(kTMA*d[node][dir]),2)) / (sinh(2*kTMA*d[node][dir]) + 2*kTMA*d[node][dir]) ),0.5);
//friction factor kf (varies from 1 in deep water to larger than 1 in shallow. Divide H by this)
kfCEM = 1 + (64*pi*pi*pi)/(3*g*g) * (ffactor_CEM*HCEM*F[node][dir]/pow(TCEM,4)) *
            (ksCEM*ksCEM/pow(sinh(2*pi*d[node][dir]/LCEM),3));
kfSPM = 1 + (64*pi*pi*pi)/(3*g*g) * (ffactor_SPM*HSPM*F[node][dir]/pow(TSPM,4)) *
            (ksSPM*ksSPM/pow(sinh(2*pi*d[node][dir]/LSPM),3));
kfTMA = 1 + (64*pi*pi*pi)/(3*g*g) * (ffactor_TMA*HTMA*F[node][dir]/pow(TTMA,4)) *
            (ksTMA*ksTMA/pow(sinh(2*pi*d[node][dir]/LTMA),3));
//double slope=1/30;                    //impliment formula for calculating beach slope
double A=0.17;
double H_break=0;
//SPM / WEMO
//WEMO_shoal = HWEMO*ksSPM;
//WEMO_fric  = HWEMO/kfSPM;
//WEMO_cum   = HWEMO - (HWEMO-WEMO_shoal) - (HWEMO-WEMO_fric);
WEMO_cum   = HWEMO;                              //wind only
SPM_shoal  = HSPM*ksSPM;
SPM_fric   = HSPM/kfSPM;
SPM_cum    = HSPM - (HSPM-SPM_shoal) - (HSPM-SPM_fric);
//if(ksSPM>1 && IT==1){
//cout<<"SPM positive shoal effect at node: "<<node<<" ksSPM: "<<ksSPM<<endl;
//}
            //if(IT==1){
            //cout<<"d["<<node<<"]["<<dir<<"]: "<<d[node][dir]<<" LSPM: "<<LSPM<<" d/L: "<<d[node][dir]/LSPM<<endl;
            //}
if(d[node][dir]/LSPM < 0.05){
            H_break_SPM  = LSPM*A*(1-exp (-1.5* (pi*d[node][dir]/LSPM)*(1+15*pow(slope[node][dir],4/3)) ) );
            //if(IT==1){
            //cout<<"SPM BREAK!!! at node:"<<node<<" SPM_cum: "<<SPM_cum<<" SPM_break: "<<H_break_SPM<<endl;
            //}
            WEMO_cum = H_break_SPM;
```

```
			SPM_cum  = H_break_SPM;
}
else{
			H_break_SPM = -99999;
}
//CEM
CEM_shoal = HCEM*ksCEM;
CEM_fric  = HCEM/kfCEM;
CEM_cum   = HCEM - (HCEM-CEM_shoal) - (HCEM-CEM_fric);
if(d[node][dir]/LCEM < 0.05){
			H_break = LCEM*A*(1-exp (-1.5* (pi*d[node][dir]/LCEM)*(1+15*pow(slope[node][dir],4/3)) ) );
			CEM_cum = H_break;
			//cout<<"BREAK!!! node:"<<node<<endl;
}
//TMA
TMA_shoal = HTMA*ksTMA;
TMA_fric  = HTMA/kfTMA;
TMA_cum   = HTMA - (HTMA-TMA_shoal) - (HTMA-TMA_fric);
if(d[node][dir]/LTMA < 0.05){
			H_break = LTMA*A*(1-exp (-1.5* (pi*d[node][dir]/LTMA)*(1+15*pow(slope[node][dir],4/3)) ) );
			TMA_cum = H_break;
			//cout<<"BREAK!!! node:"<<node<<endl;
}
//statement to fix infinity values on some ks denominator values, in this case reset cumulative wave heights
//back to the wind wave height only value
if(isinf(sinh(2*kCEM*d[node][dir]) + 2*kCEM*d[node][dir])  ==1){		//denominator in ks, causing infinity
CEM_cum=HCEM;										//reset to wind wave
height
}
if(isinf(sinh(2*kSPM*d[node][dir]) + 2*kSPM*d[node][dir])  ==1){		//denominator in ks, causing infinity
SPM_cum=HSPM;
WEMO_cum=HWEMO;										//reset to wind wave
height
}
if(isinf(sinh(2*kTMA*d[node][dir]) + 2*kTMA*d[node][dir])  ==1){		//denominator in ks, causing infinity
TMA_cum=HTMA;										//reset to wind wave
height
}
ENS_cum = (WEMO_cum + SPM_cum + TMA_cum + CEM_cum)/4;				//ensemble avg.
								}		//close if check=false
								else{		//fetch and depth are zero. Hshoal=-99999
								ks=1;
								kf=1;
								}
								outfile14<<node<<"  "<<WEMO_cum   <<endl;
								outfile15<<node<<"  "<<TMA_cum   <<endl;
								outfile16<<node<<"  "<<CEM_cum   <<endl;
								outfile17<<node<<"  "<<SPM_cum   <<endl;
								outfile18<<node<<"  "<<ENS_cum   <<endl;
if(node==35884){
//cout<<"10 dir: "<<dir<<endl;
}
if(node==35884){
//cout<<fixed<<setprecision(3)<<endl;
//cout<<"node: "<<node<<" HWEMO: "<<WEMO_cum<<" HSPM: "<<SPM_cum<<" HTMA: "<<TMA_cum<<" HCEM: "<<CEM_cum<< "
ENS_cum: "<<ENS_cum<<endl;
//cout<<"dir: "<<dir<<" F: "<<F[node][dir]<<" d: "<<d[node][dir]<<" HSPM: "<<HSPM<<" SPM_cum: "<<SPM_cum<<endl;//" SPM_shoal:
"<<SPM_shoal<<" SPM_fric: "<<SPM_fric<<endl;
//cout<<"node: "<<node<<" TWEMO: "<<TSPM<<" TSPM: "<<TSPM<<" TTMA: "<<TTMA<<" TCEM: "<<TCEM<<endl;
}
//statement to write wave height solutions of 0 to -99999 for ADCIRC and SMS
								HWEMO=0;
								HCEM =0;
								HTMA =0;
								HSPM =0;
								TSPM =0;
								TCEM =0;
								TTMA =0;
								WEMO_cum=-99999;
```

```
                        TMA_cum =-99999;
                        SPM_cum =-99999;
                        CEM_cum =-99999;
                        ENS_cum =-99999;
                }       //close if y==3
            }       //close while!eof
            //cout<<node<<"  "<<WVX[node]<<"  "<<WVY[node]<<"  "<<WDIR[node]<<"  "<<IT<<endl;
        }       //close if node <=126772
        if(node==126772){       //new timestep, reset node counter
            if(IT==NDSETSE){        //to break out of loop before writing next time step
                break;
            }
            IT=IT+1;                //model timestep number
            TIME=NSPOOLGE*IT;       //model time (in seconds)
            node=0;
            outfile14<<TIME<<"  "<<TIME<<endl;
            outfile15<<TIME<<"  "<<TIME<<endl;
            outfile16<<TIME<<"  "<<TIME<<endl;
            outfile17<<TIME<<"  "<<TIME<<endl;
            outfile18<<TIME<<"  "<<TIME<<endl;
            cout<<"Time Step: "<<IT<<" at "<<TIME<<" seconds"<<endl;
            cout<<"Number of SPM breaking nodes: "<<v<<endl;
            v=0;
        }
    }       //close while getline infile3
    //outfile3.close();
    outfile14.close();
    outfile15.close();
    outfile16.close();
    outfile17.close();
    outfile18.close();
    time_req=clock()- time_req;
    cout<<"***Time taken: "<<(float)time_req/CLOCKS_PER_SEC<<" seconds***"<<endl;
return 0;
}
```