Boundary intergral formulation for irrotational ocean surface waves

Arthur Schei Nørve

2017

Abstract

This report aim to describe the work put into the derivation, making and testing of a stable numerical code for surface waves in a closed tank of finite size. Although incomplete, the work that has been done will be described and detailed. The foundation for the implementation is the report by Isak Kilen (Tromsø, 2009), and we will here go through and simplify parts of the Kilen's report relevant to the work. Notation have been slightly changed to avoid the sometimes numerous indices showing up everywhere. It is nowhere claimed that this report is anywhere close to satisfactory with regards to progress in the project but, we will nevertheless go through the things that have been done and thought of.

A Problem description

A.1 General

The problem consists of solving for 1D surface waves, due to discretisation we have to limit ourselves to an "ocean" of finite extent. The full derivation is described in detail in [2] and as such we only describe the general approach here and leave the details to the reader. The first thing to notice is the general description of what we are trying to model and how we do it mathematically.

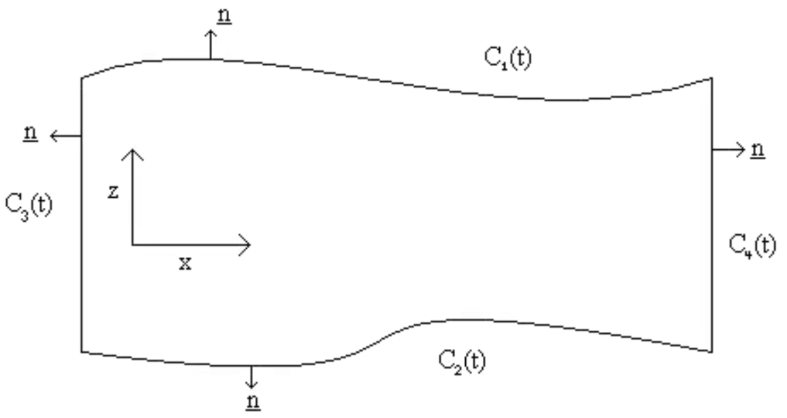

Figure 1: Figure taken from [2] showing the body we seek to model

Part of the BIM solution is using a parametrical description for the edges of the basin, seafloor orography and the liquid surface. This allows for a much more general description than do normal functions, as an example we have multivalued (breaking) solution curves (waves). The parametrical descriptions are denoted by the C^i where the index refers to the relevant side. From the figure we see that 3 and 4 are used for the (flat) edges of the basin, 2 for the bottom and 1 for the surface.

Regarding the parameterisation we need to variables to describe it: one for space and one for time. Kilen uses t for time and s for the spacial parameterisation. As such $C^i(s,t) = (X^i(s,t), Z^i(s,t))$is the complete notation.

The description of the orography and how it develops over time is part of the problem data and as such everything we want to know about it is derivable from first principles. The things we need are of the tangential and normal unit vectors. Part of the solution will be to compute these for the surface as well, but to start this computation we need initial data for the surface. These are set to be flat by Kilen in [2] (p. 39):

$$C^1(s,0) = (s,0) \tag{1}$$
$$\phi^1(s,0) = 0 \tag{2}$$

A.2 Definitions

On page 36 Kilen also lists some useful definitions, we show them here for completeness:

$$n(C^i(s,t)) = (n_x^i(s,t), n_z^i(s,t)) \tag{3}$$
$$p(C^i(s,t)) = (p_x^i(s,t), p_z^i(s,t)) \tag{4}$$
$$\phi^i(s,t) = \phi(C^i(s,t),t) \tag{5}$$
$$\phi_u^i(s,t) = \partial_u \phi(C^i(s,t),t) \tag{6}$$
$$\phi_{\vec{u}}^i(s,t) = \partial_{\vec{u}} \phi(C^i(s,t),t) = \nabla\phi^i(C^i(s,t),t) \cdot \vec{u} \tag{7}$$

A.3 Discretisation

To actually do some numerics on this problem we need to discretise the basin and the variables we are dealing with. As mentioned above we have the two variables t and s, both have to be chopped up into intervals. Since we are dealing with a finite ocean we let s go from -L to L. The discretisation consists of two parts/grids, one main grid and one for the midpoints. These two will be denoted by α_k and s_k respectfully.

$$\alpha_k = -L + k\Delta k \qquad k = 1...N+1 \tag{8}$$
$$s_k = \frac{1}{2}(\alpha_k + \alpha_{k-1}) \qquad k = 2...N+1 \tag{9}$$

The reason for starting at 1 and not 0 is simply to ease the conversion to code. We then discretise the rest of the variables by adding a lower index k on it to indicate the discretisation point. The parametrisations are discretised over the main grid while the normal, tangential vectors and the potential are given over the midpoints:

$$X_k^i(s,t) = X^i(\alpha_k,t) \tag{10}$$
$$Z_k^i(s,t) = Z^i(\alpha_k,t) \tag{11}$$

From these Kilen constructs the discretised normal and tangential vectors (page 42):

$$\vec{p}_k^i(t) = \frac{(X_k^i(t) - X_{k-1}^i(t), Z_k^i(t) - Z_{k-1}^i(t))}{e_k^i t} \tag{12}$$

$$\vec{n}_k^i(t) = \frac{(Z_k^i(t) - Z_{k-1}^i(t), -(X_k^i(t) - X_{k-1}^i(t)))}{e_k^i t} \tag{13}$$

Where $e_k^i t$ is the length of the interval number k in the discretisation of C^i

A.4 Second order scheme with finite differences

For the gradients (x and z derivatives) in the eauations Kilen used an order one method in his Matlab code. In our implementation we have chosen to use a second order central difference in the interior of the interval with a special difference utilised at the endpoints so that the second order can be maintained throughout. By this tone we have:

$$\partial_u v_i = \frac{-3v_i + 4v_{i+1} - v_{i+2}}{2\Delta u} \quad i = 1 \tag{14}$$

$$\partial_u v_i = \frac{v_{i-2} - 4v_{i-1} + 3v_i}{2\Delta u} \quad i = N + 1 \tag{15}$$

$$\partial_u v_i = \frac{v_{i+1} - v_{i-1}}{2\Delta u} \tag{16}$$

These can be derived by manipulating taylor series and truncating so that order 2 is acheived.

A.5 Solving the integral relation

Kilen develops an ingegral relation based on the Laplace equation and then discretises the Green's functions, as is normal with these boundary integral relations one end up with a system of equaitons that has to be solved. I this case the system takes the form seen in [2] 7.3 (page 43).

B The complete solution

Here we list for convenience the complete solution process as outlined by Kilen on page 45. For the particular code written we have implemented a solution algorithm based on the Euler ODE scheme. For completeness we show it here:

$$y' = f(t, y) \tag{17}$$

$$t_n = t_0 + n\Delta t \quad y_n = y(t_n) \tag{18}$$

$$y_{n+1} \approx y_n + \Delta t f(t_n, y_n) \tag{19}$$

For our case the y variable would be a large vector, as can be seen in the following equation, showing the system we acutally have to solve:

$$\partial_t \begin{bmatrix} X_k^1(t) \\ Z_k^1(t) \\ \phi_k^1(t) \end{bmatrix} = \begin{bmatrix} \partial_x \phi^1(\alpha_k, t) \\ \partial_z \phi^1(\alpha_k, t) \\ \frac{1}{2}((\partial_{xk}\phi^1(t))^2 + (\partial_{zk}\phi^1(t))^2 - gZ(s_k, t)) \end{bmatrix} \tag{20}$$

Sending this through the Euler scheme from above we obtain the following:

$$\begin{bmatrix} X_k^1(t_{n+1}) \\ Z_k^1(t_{n+1}) \\ \phi_k^1(t_{n+1}) \end{bmatrix} = \begin{bmatrix} X_k^1(t_n) \\ Z_k^1(t_n) \\ \phi_k^1(t_n) \end{bmatrix} + \begin{bmatrix} \partial_x \phi^1(\alpha_k, t_n) \\ \partial_z \phi^1(\alpha_k, t_n) \\ \frac{1}{2}((\partial_{xk}\phi^1(t_n))^2 + (\partial_{zk}\phi^1(t_n))^2) - gZ(s_k, t_n)) \end{bmatrix} \tag{21}$$

Now between the initial data and actually time stepping this equation there is a lot of things that have to be done. As mentioned Kilen lists these in a sandwhich recipe like fashion in his article. We relist it here for completeness using vector notation instead of the cumbersome components:

1. $\nabla\phi^2 = \partial_t C2(s, t)$
2. $\partial_{\vec{n}}\phi_k^2 = \partial_x \phi_k^2(t)\partial_x n_k^2(t) + \partial_z \phi_k^2(t)\partial_z n_k^2(t)$
3. $\partial_{\vec{n}}\phi_k^1$ is found by solving the linear system referenced in section A.4.
4. $\partial_{\vec{p}}\phi_k^1 = \partial_s \phi^1$
5. $\partial_x \phi_k^1 = \partial_n \phi_k^1(t)\partial_x n_k^1(t) + \partial_p \phi_k^1(t)\partial_x p_k^1(t)$
6. $\partial_z \phi_k^1 = \partial_n \phi_k^1(t)\partial_z n_k^1(t) + \partial_p \phi_k^1(t)\partial_z p_k^1(t)$

C Code

When it comes to the code, it's written in Julia; a super fast, semi-functional programming language made for scientific computing. To keep track of all the numbers and especially all the indices in our solution we have implemented a special data type called TwoVector. This datatype stores the discretisation as two component matrices (X and Z) with both matrices listing each time step as a new row. This eases implementation and makes keeping track of all our data rather easy. Beacause of Julia's functional overloading we can implement the binary operators (+,-,*,/) when operating on two TwoVectors or on a scalar and a TwoVector. This proves to be extremely powerfull as we can move a lot of the tedious code into a general description of operators.

The code currently consists of five files:

main.jl The file from which everything is launched and commanded

boundary.jl Implementation details and euler loop

TwoVector.jl TwoVector data type, overloaded operators and convecience functions

matrix_elements.jl Functions for fetching the matrix elements

functions.jl More convenience functions

The code written so far is publically available in the following GitHub repo: UiT-MAT3810-Surface-Waves

D Results

Due to lack of more time, we only got to the point where the code was written and started to spit out results. The problem that remains is to stabilise it and optimise it. As such there is no visible results to point to except the code itself and my understanding of the topic.

E Conclusion and Moving on

So far the thing that have been done is basically taking a look at the report by Kilen, and simplifying the solution process itself and rewriting it. The code has also been written and run but as mentioned in the result section, no notable results have been obtained so far. What is needed now is to actually make the code behave properly and spit out something that looks reasonable. At that point intense testing is in order to verify that what comes out actually reperesents what we aimed to model in the first place.

References

[1] Per Kristen Jakobsen. The theory of fluid dynamics: A short ingdtoduction.

[2] Isak Kilen. Simulation of ideal irrotational liquid using the boundary integral method. 2009.