Updates to cBathy, version 2.0

Rob Holman

07/24/19

The cBathy algorithm was published in 2013 [*Holman et al.*, 2013] as a robust method to estimate nearshore bathymetry based on the phase speeds of ocean surface waves observed using remote sensing. Since that time, cBathy has gone through a number of revisions but typically without extensive testing and documentation. The goal of this paper is to document these revisions and test the most recent, cumulative version of the algorithm against a new test bed data set consisting of 39 surveys spanning from 2015 to 2019 from the Field Research Facility at Duck, NC. Thus, we will describe the algorithm changes then introduce both the new test bed data set and a protocol for quantifying and comparing the performance of this and previous versions.

The original form of the algorithm is now called version 1.0 and has been followed by versions 1.1, 1.2, and now a composite update which we choose to call version 2.0. Below we will start by summarizing the elements of the original 1.0 algorithm plus the various significant updates. We then introduce the test bed data set, which can be used to test both cBathy and any other algorithm in nearshore remote sensing. Finally, we will describe a testing protocol that can be used to diagnostically document performance changes with any algorithm updates.

**cBathy Versions:**

*Version 1.0*

Version 1.0 is fully described in Holman et al [2013], but is summarized herein. For more detail, the reader is referred to the original manuscript. The input data for cBathy consists of time series of image intensity, I(xp, yp, t), at a set of discrete pixel locations, xp, yp that span the domain of interest and adequately sample the typical ocean wave scales while not oversampling them (across-shore and alongshore spacing is commonly 5 and 10 m, respectively, see blue dots in Figure 1). The analysis is carried out at a map array of model locations, xm, ym, (example red dot in Figure 1) and at each map location is based on the observed phase ramps in a tile of observations within some user-specified length scales, Lx and Ly, of the model location (green dots in Figure 1).

Bathymetry estimation is based on the linear dispersion relationship

(1)

where  is the radial frequency (2 divided by the period, T), k the radial wave number (2 divided by the wave- length, L), g the acceleration due to gravity, h the depth and currents and finite amplitude effects have been neglected. This can be inverted to solve for depth as a function of frequency and wavenumber

(2)

Thus, the goal is to estimate the dominant wave frequencies and wavenumbers at each model location and merge that information into a spatially smooth estimated bathymetry using the dispersion relationship. The algorithm is divided into three phases: 1. estimation of frequency-wave number pairs, 2. Estimation of bathymetry from suites of those estimates, and 3. Kalman smoothing separate hourly estimates to create a running average product that is robust to noise and error. The updates since 2013 have all addressed improvements to phases I and II, so algorithm review below will omit phase III.

The algorithm must adapt to incident waves with periods from almost 20 s to about 4 s (shorter waves are insensitive to depth in other than very shallow water so are not helpful) without knowing a priori what conditions will be. Because the analysis is based in frequency space, the first step is to Fourier transform each pixel time series and normalize the Fourier magnitudes, allowing focus on Fourier phase only. Cross-spectra are computed between all pixels in a tile for a suite of candidate frequencies that are usually spaced to allow about 40 degrees of freedom. The (usually four) dominant frequencies are those with the largest total coherence in the resulting cross-spectral matrix and a wavenumber is then estimated for each frequency.

Wavenumber is found from maps of Fourier phase for each tile (Figure 2). As a first step, the cross-spectral matrix is decomposed into complex eigenvectors, v(xp, yp), and only the one with the largest eigenvalue is retained. This eigenvector is modelled as a single dominant plane wave with form

(2)

where  is the wave angle and  is a scalar phase angle. In version 1.0, the three parameters k,  and were found using a weighted nonlinear least-squares fit between the observed and modeled eigenvector phase maps. Details of the weighting are included in the original paper.

The goal of phase II of the algorithm is to use a suite -k phase I estimates to estimate depth at a single point. Each model point can yield up to four frequencies and phase II incorporates estimates from adjacent xm, ym locations from within the tile, weighted by distance from the current estimation point. In version 1.0, the weighting was taken to depend on the normalized eigenvalue and skill of the model fit, both from phase I, as well as the distance from current estimate location in Phase II. The final Phase II result is the single depth that is the nonlinear best fit to predicted depth using the input suite of frequency-wavenumber pairs and the dispersion relationship.

*Version 1.1 Update*

As waves are measured in increasingly deep water, they become decreasingly sensitive to depth. Thus, small errors in measured wavenumber are associated with large errors in estimated depth. To guard against this excessive sensitivity in version 1.0, values deeper than a user-specified maximum depth were neglected. However, this maximum depth should be wave frequency dependent and biases are introduced in deeper water results by simply truncating values. Version 1.1 corrected this problem by accounting better for the dispersion relation sensitivity in phase II weighting.

Equation (2) can be rewritten

(3)

where k0 is the wavenumber in deep water. We can define k0/k = L/L0 = , where  is a non-dimensional wavelength (or wavenumber), going from small in shallow water to a maximum value of 1.0. We can define a sensitivity to wavenumber error by the equation

(4)

where  is the sensitivity, which can be found numerically. Figure 3 shows sensitivity of wavenumber inversion as a function of non-dimensional wavelength. In shallow water the value is 2.0, so that fractional bathymetry errors are twice the magnitude of fractional wavenumber errors. This is the best case. Closer to deep water ( approaching 1.0), the sensitivity rises rapidly, approaching 10 when wavelengths are 0.9 of their deep-water value.

This sensitivity provides a convenient weighting measure. If we denote our previous Phase II weighting value as W for any -k pair, we can divide this weight by the sensitivity as a simple method of preferentially weighting -k pairs that are in shallower water and de-emphasizing those approach the deep-water limit. Thus, the main change in the version 1.1 algorithm is this modified weighting in Phase II. A side benefit is that the user no longer needs to specify a maximum depth beyond which to no longer believe an estimate.

*Version 1.2*

Version 1.2 dealt mostly with a problem of poor bathymetry estimates at the seams between cameras. Argus image data that is a most common input to cBathy analysis is often collected from multiple cameras and merged into a map of data coverage, for example as shown in Figure 1. When these data were analyzed as a single array, estimated bathymetries along camera seams were often anomalous.

Two causes were identified. The most obvious issue is problems with camera geometry used to map from image to world coordinates [*Holman and Stanley*, 2007]. The sampling pixel array is usually designed as a regular world grid, mapped to pixels for each camera using originally accurate camera geometries. If camera geometries shift slightly, the world spacing of those pixels on either side of a camera seam will begin to differ from the original spacing. Thus, waves will appear to move too quickly across a shortened sample gap or too slowly for a stretched gap, leading to errors in depth.

The second plausible cause of cross-seam anomalies can come from loss of synchronization of the cameras. Argus cameras are usually synchronized by either a trigger or by computer bus synchronization. However, there can be rare frame slips that allow time shifts between cameras that would be interpreted as wave speed anomalies for tiles that span camera seams.

Initial attempts were made to mitigate both of these problems. However, it soon became clear that the simple solution was to never mix pixels from multiple cameras in any tile. Thus, the pixels from the majority tile are used and the others simply neglected. This results in fewer degrees of freedom, but otherwise had no significant negative impact.

Version 1.2 also included a new version for finding the seed wavenumber and wave angle but this change has been overtaken by a much better seed routine in Version 2.0.

*Version 2.0*

The upgrade to version 2.0 involves three significant changes, so is considered a major revision and is given a new leading version number. In order of importance, the changes are a) to change from tile sizes that are fixed by the user to those that change depending on expected wavelengths, b) to change from solving for three variables, k,  and 0, at each map point to solving for only k and , and c) introduction of a much better algorithm to find seed values for k and  before the nonlinear search for each tile. The cumulative consequence of these modification was a major restructuring of the code. Each component will be described in turn.

*Version 2.0a.* In all earlier versions, cross-shore and alongshore tile sizes were set by the user using the parameters Lx and Ly. Suggestions for best values were ad hoc with a belief that the search would work best if the tile was typically about a wavelength long but an implicit faith that even mis-matched tile sizes would solve well somewhere within the nonlinear fitting routine. The same tile size was used for 4 s and 16 s waves despite at least a four-fold difference in wavelengths. This approach was still successful since tiles that were poorly designed simply failed to converge and returned nan’s rather than a poor depth.

Issues with this approach became especially clear for short period incident wave, for example 4-5 s waves. These could have 4-5 wavelengths within the tile, so convergence of the nonlinear search usually behaved very poorly since there was not a simple global cost function minimum unless you started with a very accurate seed. The problem was made worse by a feature of the code that decimates the original number of pixels in a tile down to a user-defined maximum, maxNPix, to help reduce processing time. Commonly a standard tile was reduced from 250 collected pixels down to 80 that would be analyzed, enough to map a single typical wavelength way too few to map out five short wavelengths in a tile, aliasing the true signal.

Version 2.0 fixes these problems by defining the phase I tile sizes to be kL times the expected wavelength, where kL is taken to be approximately 1.0. But the expected wavelength depends of the frequency and depth, neither of which is known a priori. Thus, there is strong motivation to develop a seed-finding algorithm (section 2.0c) that can provide good estimates of k under all wave conditions. Thus, given an initial tile based on generic user input, version 2 feeds all of the available pixels to the routine to find the seed k and , then truncates the original tile size to kL times this wavelength, a size that varies considerably. The truncated tile is then decimated, if necessary, to maxNPix to speed up processing. Because all tiles are roughly one wavelength, it is likely that fewer pixels are needed for search convergence, again speeding up processing.

*Version 2.0 b.* In all earlier versions, the model for wave phase, equation 2, had three parameters, each of which was found using a standard Levenburg Marquardt solver. But only two of the parameters, k and , are expected to be well behaved in a nonlinear gradient-descent search. The third variable, 0, is a scalar offset between the measured and modeled phase maps and can jump around a great deal, in ways that are inconsistent with a search for a global cost function minimum. Thus, this variable is not well estimated and likely just confuses the search.

In a very early version of this algorithm known as Beach Wizard (Lite), the solution was sought not in x-y space phase maps as here, but in cross-spectral space. This had the advantage that a phase offset was not required (phase differences just increased with lag from zero) but it meant that if you had N pixels in your tile you were searching in an N2 space (each pixel compared to every other pixel). In addition, visualization of measured and modeled results was not as clear. Thus, we wish to retain the simplicity of working in x-y space maps but find a method for estimating 0 for each search iteration that will allow a sensible search for k and . The solution was to force the measured and modeled phase to be the same at the tile center (the pixel closest to xm, ym). This was done by finding d, the difference in phase at the middle pixel, and multiply all modeled complex values of the eigenvector, v, by eid. The nonlinear search was then reduced to two dimensions.

*Version 2.0 c*. Both the success of adaptive tile sizes and of the nonlinear search depend on the accuracy of the initial seed search values for k and . The search is complicated by the fact that Fourier phase has 2 jumps whose slight mis-positioning adds to cost functions out of proportion to the error. It rapidly became clear that the full resolution of the tile was needed, i.e. decimation down to maxNPix could only occur after sub-sampling to the adaptive tile size.

The solution was found by making use of the radon transform. First the observed phases from the eigenvector were interpolated onto a regular grid, a somewhat noisy process due to the phase jumps. Next the radon transform was found to estimate the phase variance when projected along a suite of candidate directions (-45 to +45° in 2° increments; Figure 4). The angle that corresponded to the maximum variance was selected as the seed wave angle, . Finally, the phase map was interpolated onto a new grid that is oriented in the direction of wave propagation and the median of the phase gradient was used as the seed value of k. This search also returned the location of the pixel that lay closest to the center of the tile (xm, ym), to be used in finding 0.

*Algorithm Organization Changes:*

The changes in version 2.0 have forced a major restructuring of the algorithm. Instead of reducing the number of pixels prior to any analysis steps, it was clear that full pixel resolution was required for finding the k- seeds and for establishing the required size of the tile. Thus, the full tile of size Lx, Ly is initially passed into the main analysis routine, csmInvertKAlpha, where the routine prepareTiles is called to a) find the dominant frequencies and, for each frequency, find the dominant eigenvector and the k- seeds, then reduce the tile to an adaptive size and to a maximum number of pixels. These outputs are then passed back to csmInvertKAlpha to do the nonlinear search for best values and their errors then to build the results structures and find the depths from the -k results.

Several new fields have been added to the bathy output structure. These include a time exposure map at the desired xm, ym points, maps of the k and  seeds that will allow testing of how well the seed algorithm works and why it might fail. Maps are also included of the number of pixels used in every tile and the number of calls that were made to the forward model predictCSM in the nonlinear search, a key to algorithm speed. Finally, the elapsed CPU time for each analysis is saved.

Further recent changes – I found out that phase II was taking 100 s for processing a standard data set (x = [75: 12.5: 500], y = [0: 25: 1000], 2048 points at 2 Hz, maxNPix = 80). The profiler showed that 2/3 of that time was spent asking whether the user had the stats toolbox using the command ver. This was done twice per tile. Doing this call once per collection only reduced the processing time to 33 s for phase II. I also found that the bulk of the phase I time was NOT spent on nlinfit but now is spent in prepareTiles.m. I was unclear what was slow but suspected the EOF calculation. I ended up finding a faster EOF, called eigs in matlab, that finds only the first six EOFs (default). This ran more than twice as fast (0.25 s versus 0.58 s for the full set of eigenvectors). I tried asking for just the dominant EOF but that ran slower, likely due to error checking for a non-standard case. Using this, a full standard run takes 119 s on my machine. I also tried lower resolution runs with dx, dy = [25, 50], so ¼ of the normal estimation points. This ran in 41 s. I tried halving maxNPix to 40 but this made no significant difference in run time or output. The 90% statistics of absolute depth difference for 40 versus 80 pixels was only 0.21 m. BTW, be aware that using the default Lx = 3 dxm, Ly = 3 dym, can yield very large initial tiles (and phase II tiles) that include many cameras near the tower – it is possible to have the dominant camera not even include xm, ym. A fix for this bug was implemented.

Finally, I have added a weighted frequency as a map for phase II (a new field fBar). This is given by sum(w.\*f)/sum(w) where I first tried using w as the weights used in the nonlinear fit. Then Greg convinced me to use matlab’s ‘leverage’ function which is based on the Jacobian. Both gave close to the same result for some test cases so I went with leverage.

Holman, R. A., N. G. Plant, and K. T. Holland (2013), cBathy: A robust algorithm for estimating nearshore bathymetry, *Journal of Geophysical Research*, *118*, 1-15, doi:10.1002/jgrc.20199.

Holman, R. A., and J. Stanley (2007), The history and technical capabilities of Argus, *Coastal Engineering*, *54*, 477-491.