cBathy Version Updates and Comparison

14 December 2020

Introduction

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. This paper documents these revisions, describing in detail the differences between each.

- Version 1.0 is the original version found in CIRN's GitHub repository
- Version 1.1 can also be found in CIRN's GitHub repository
- Version 1.2 is the current master branch of CIRN's GitHub repository
- Version 2.0 is a composite update that we propose as the new default.

Below we summarize the elements of the original 1.0 algorithm plus the various significant updates. We then give a more detailed examination of the differences between Version 1.2 and the Version 2.0.

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, $I(x_p, y_p, t)$, at a set of discrete pixel locations, x_p, y_p 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). The analysis is carried out at a map array of model locations, x_m, y_m, and at each map location is based on the observed phase ramps in a tile of observations within some user-specified length scales, L_x and L_y, of the model location.

Bathymetry estimation is based on the linear dispersion relationship

$$\sigma^2 = gk\, tanh(kh) \tag{1}$$

where σ is the radial frequency (2π divided by the period, T), k the radial wave number (2π divided by the wavelength, 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

$$h = \frac{1}{k} tanh^{-1}\left(\frac{\sigma^2}{gk}\right) \qquad (2)$$

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: (I) estimation of frequency-wave number pairs; (II) estimation of bathymetry from suites of those estimates; and (III) 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. As a first step, the cross-spectral matrix is decomposed into complex eigenvectors, $v(x_p, y_p)$, and only the one with the largest eigenvalue is retained. This eigenvector is modelled as a single dominant plane wave with form

$$v = exp\big(i\big[kcos(\alpha)x_p + ksin(\alpha)y_p + \varphi\big]\big) \qquad (2)$$

where α is the wave angle and ϕ is a scalar phase angle. In Version 1.0, the three parameters k, α and ϕ 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 of σ-k Phase I estimates to estimate depth at a single point. Each model point can yield up to (usually) four frequencies and Phase II incorporates estimates from adjacent x_m, y_m 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

As waves are measured in increasingly deep water, they become decreasingly sensitive to depth. Thus, small errors in measured wavenumber are associated with increasingly 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

$$h = \frac{1}{k} tanh^{-1}\left(\frac{\sigma^2}{gk}\right) = \frac{1}{k} tanh^{-1}\left(\frac{k_0}{k}\right) \tag{3}$$

where k_0 is the wavenumber in deep water. We can define $k_0/k = L/L_0 = \Gamma$, where Γ 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

$$\frac{\Delta h}{h} = \mu(\Gamma)\frac{\Delta k}{k} \tag{4}$$

where μ is the sensitivity, which can be found numerically. 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 (Γ approaching 1.0), the sensitivity rises rapidly, approaching 10 when wavelengths are 0.9 of their deep-water value (Figure 1).

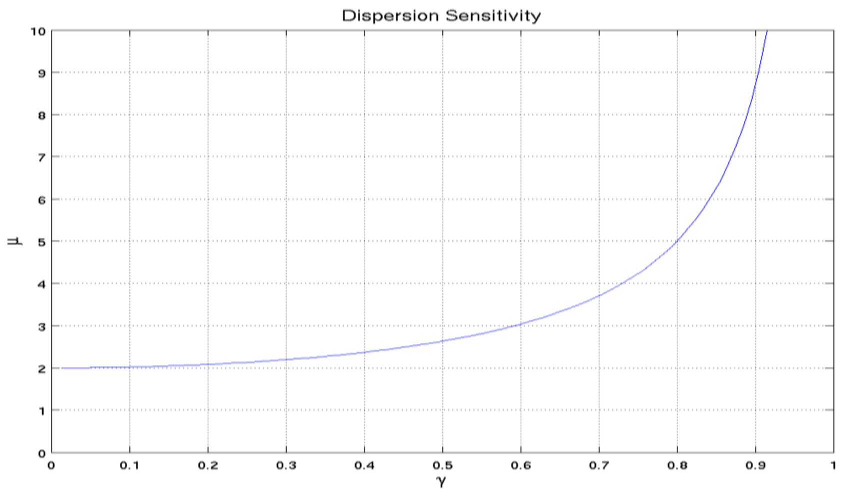

Figure 1. Dispersion sensitivity μ as a function of non-dimensional wavenumber, Γ

This sensitivity provides a convenient weighting measure. If we had denoted our previous Phase II weighting value as W for any σ-k pair, we can divide this weight by the sensitivity as a simple method of preferentially weighting σ-k pairs that are in shallower water and de-emphasizing those that 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 updates dealt mostly with a problem of poor bathymetry estimates at the seams between cameras. Argus image data that is a common input to cBathy analysis is often collected from multiple cameras and merged into a map of data coverage. 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 errors in the estimated 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 has 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, α and ϕ, at each map point to solving for only k and α, and c) introduction of a much better algorithm to find seed values for k and α before the nonlinear search for each tile. The cumulative consequence of these modifications was a major restructuring of the code. Each component will be described in turn.

i) Adaptive Tile Size

In all earlier versions, cross-shore and alongshore tile sizes were set by the user using the parameters L_x and L_y. 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 but 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 k_L times the expected wavelength, where k_L is taken to be approximately 1.0. However, 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 (below) that can provide good estimates of k under all wave conditions. Thus, given an initial tile based on generic user input, Version 2.0 feeds all of the available pixels to the routine to find seed values for k and α, then truncates the original tile size to k_L 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.

ii) Number of Parameters

In all earlier versions, the model for wave phase, Equation 2, had three parameters, each of which was found using a standard Levenberg-Marquardt solver. But only two of the parameters, k and α, are expected to be well behaved in a nonlinear gradient-descent search.

The third variable, ϕ, 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 (lag) 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 N^2 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 ϕ for each search iteration that will allow a sensible search for k and α. The solution was to force the measured and modeled phase to be the same at the tile center (the pixel closest to x_m, y_m). This was done by finding $d\phi$, the difference in phase at the middle pixel, and multiply all modeled complex values of the eigenvector, v, by $e^{id\phi}$. The nonlinear search was then reduced to two dimensions.

iii) Seed Algorithm

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 α. The search is complicated by the fact that Fourier phase has 2π 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). The angle that corresponded to the maximum variance was selected as the seed wave angle, α. 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 (x_m, y_m), to be used in finding ϕ_0.

Additional Updates

In 2020, several further updates were made which improved reliability and the fraction of successful returns (non-nan values). These updates were largely related to changes in the routine **prepareTiles** associated with finding the k and alpha seed values. First, the previous version of **prepareTiles** called **findKAlphaSeed** twice, once with the full set of pixels and second with the reduced set after tile size reduction and decimation to maxNPix pixels. Because the second call was based on many fewer pixels it was found to be less stable and was eliminated. Second, in order to carry out the radon transform in **findKAlphaSeed**, the pixel phase data had to be interpolated onto a regular grid. The interpolation returned nan values for extrapolated values on the edges of the grid that were artificially filled with a mean phase angle with no apparent impact on the wave angle seed estimate. However, the phase data were then re-gridded to align with this wave angle to allow k estimation from the median of the wave-oriented phase gradient. In that case the filled in values all contributed values of zero phase gradient which could in fact dominate the median calculation. This was found to be common along camera seams where there were fewer useable pixels, thus more extrapolation.

Several other features were changed. The eigenfunction call was changed to eigs(C,1) that specifically only finds the first eigenvector. Several structures were also initialized to speed up processing. Finally, a number of new fields were added to the bathy structure including:

- Course time exposure, brightest and darkest images found directly from the pixel time series
- Wavenumber and wave angle seed values were added to the fDependent structure to ease debugging, if needed.
- A map of the weighted frequency contributions to every phase II depth.
- The MATLAB version is recorded.

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-α seeds and for establishing the required size of the tile. Thus, the full tile of size L_x, L_y is initially passed into the main analysis routine, **csmInvertKAlpha**. This routine then calls **prepareTiles** to find the dominant frequencies and, for each frequency, find the dominant eigenvector and the k-α seeds. Then, **prepreTiles** reduces 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 σ-k results.

Several new fields have been added to the bathy output structure. These include time exposure, brightest and darkest maps at the desired x_m, y_m points, maps of the k and α 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 changes were made to speed up processing. The redundant calls to the MATLAB version were removed. The computation of the empirical operator function EOF was replaced with the built-in MATLAB function eigs which is set to only find the dominant eigenvalue and eigenvector. Originally, this was set to find the six largest eigenvalues, but was changed, as will be discussed in a section below. A weighted frequency was added as a map for Phase II (a new field fBar). This is given by sum(w.*f)/sum(w) and utilizes MATLABs 'leverage' function which is based on the Jacobian.

Differences between Versions 1.2 and 2.0

This section outlines specific significant differences between the software found in Version 1.2, that is the master branch on the CIRN GitHub repository, and Version 2.0, provided by Rob Holman in October 2020 and proposed as the new default.

Figure 2 shows the logic flow for version 1.2 with major routines color-coded to indicate each of the three analysis phases. Phases I and II are called by *analyzeBathyCollect* while phase III is run by *KalmanFilterBathy*. This figure can be compared to Figure 3 which shows the updated logic flow for version 2.0.

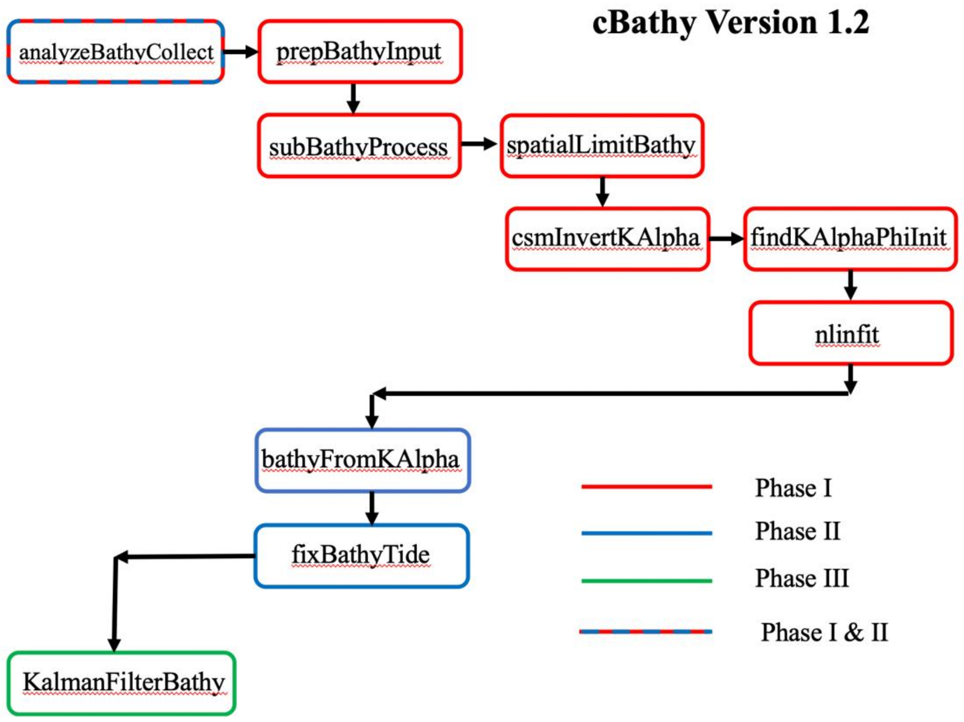

Figure 2. The logic flow for cBathy version 1.2. Phases of the cBathy algorithm are indicated by color, with the legend in the bottom right. "nlinfit" indicates the call to the nonlinear fitting routine for k and a which is logically the same as the optional Levenberg Marquardt solver, LMfnlsq.

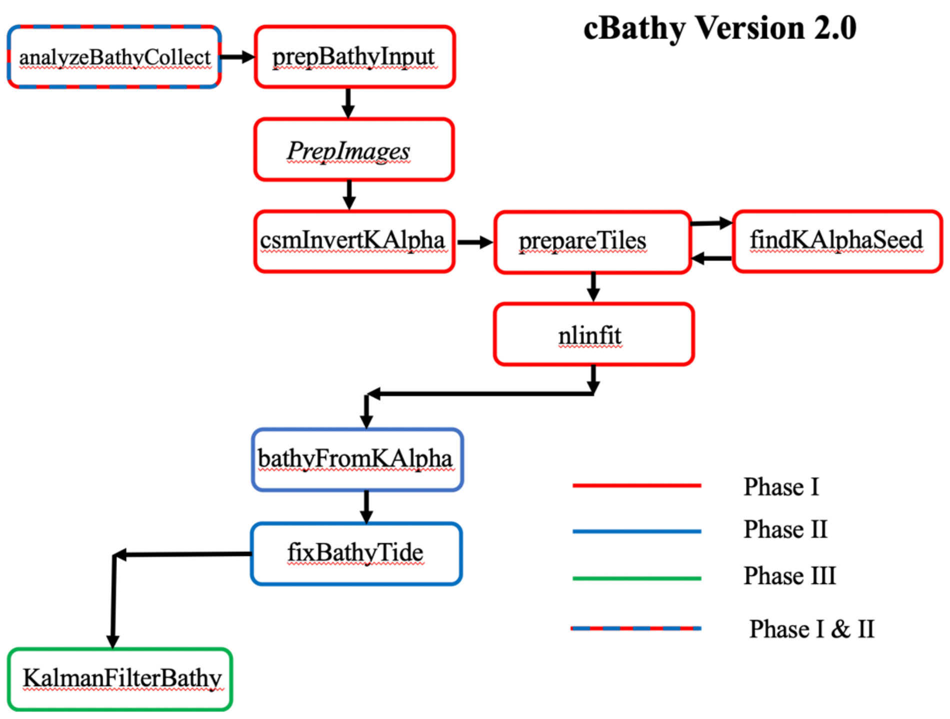

Figure 3. The logic flow for cBathy version 2.0. Phases of the cBathy algorithm are indicated by color, with the legend in the bottom right. "*PrepImages*" is not a subroutine but is a section of new code in analyzeBathyCollect that finds the course resolution time exposure, brightest and darkest images. The algorithm changes and re-ordering are reflected in the routines csmInvertKAlpha, prepareTiles and findKAlphaSeed.

Output Bathy Structure Changes:

1) Bathy.matVer has been added to document the MATLAB version used,
2) Bathy.Timex, .darkest and .brightest have been added as coarse images computed directly from the stack data
3) Bathy.cpuTime has been added to record the required processing time.
4) Bathy.fDependent has several new fields
   a) NPixels – the number of input pixels in each time and for each frequency
   b) NCalls – the number of calls to predictCSM in the nonlinear solution
   c) kSeed – the wavenumber seed for each tile and frequency
   d) aSeed – the wave angle seed used for each tile and frequency
   e) camUsed – the camera used for each tile (one per tile)
5) bathy.fCombined – the field fBar has been added as the weighted average of the nKeep frequencies that contributed to each h estimate

Some new files have been added to Version 2.0 that were not in Versions 1.2. These functions are described here:

1) **findKAlphaSeed** – this function replaces **findKAlphaPhiInit** in Version 2.0
   a) Code is called from **prepareTiles** instead of directly from **csvInvertKAlpha**
   b) This code finds **phi** directly instead of including it in the least-squares fit
   c) Least-squares fit finds **k** and **alpha**.
   d) Uses radon transform in the computation, which requires the Image Processing Toolbox in MATLAB.
2) **prepareTiles**
   a) Compute all the needed seeding estimates for **csmInvertKAlpha**
      i) Dominant frequencies in prioritized order
      ii) Initial seeds for **k** and **alpha**
      iii) Complex eigenvectors
      iv) **xy** locations
      v) Camera used
      vi) Normalized eigenvalues of first EOF (empirical orthogonal functions)
      vii) Index of the **xy** location closest to the tile center.
3) **tstat3**
   a) Compute one of three **t**-statistics
   b) Provided for those without access to the Statistics toolbox in MATLAB.

The following files have also changed significantly in Version 2.0 from Versions 1.2:

1) **analyzeBathyCollect**
   a) Variables are initialized outside of loops to save memory
   b) Calls to **subBathyProcess** is no longer needed
   c) Creates time exposure, timex, bright, and dark using **useInterpMap**
2) **bathyFromKAlpha**
   a) added a flag to choose an alternative to nlinfit
   b) An optional Levenberg-Marquardt least squares function **LMFnlsq** was removed for this version but will be merged into future updates.
3) **csmInvertKAlpha**
   a) Significant method changes to this function
   b) Reordered the loop through frequencies
   c) Uses **prepareTiles** to find the seeds for **k** and **alpha**
   d) Removed **phi** from the least-squares fitting
4) **predictCSM**
   a) added Global variables as in **csmInvertKAlpha**
   b) Changed computational use of **phi**, which is no longer an input
5) **plotPhaseTile**
   a) image display option changes (scatter plots now ‘filled’, axes now ‘equal’)

   b) Colormap changed to ‘hot’
6) **bathyCI**
   a) Added option to use **tstat3** instead of t-statistic computation using MATLAB Statistics toolbox

The following changes were made in multiple places

1. The function calls to **nanmean, nanmedian,** and **nansum** were replaced with **mean(*,’omitnan’) median(*,’omitnan’),** and **sum(*,’omitnan’)** where used
   a. This capability was added to MATLAB Version R2016B to eliminate a dependency on the Statistical Toolbox and is now recommended by MathWorks

Some files were placed in subfolder called **Extra_Files** in the repository**.** Among these are two sample parameter files:

- **argus02a**
- **argus02b**

Summary Tables of Changes in Versions 1.2 and 2.0

The following tables summarize the major difference between functions that can be found in versions of the cBathy source code.

Function csmInvertKAlpha		
	Version 1.2	**Version 2.0**
Inputs	**kappa** and **params** are separate inputs	Fourier coefficients: **G** **XY** instead of **XYZ** Variable **params** replaced by **bathy** Input **bathy** does not include **kappa**
Finding the least-squares seed	Found using **findKAlphaPhiInit**	Found using **prepareTiles,** outside of frequency loop. The function **prepareTiles** uses **findKAlphaSeed**. **phi** is no longer part of the seed.
Loops that compute **$C=G^{*}G$**	Inside the frequency loop	Separate loop over all frequencies, not just kept frequencies
Least-Squares	Finds k, alpha, and phi	Finds only k and alpha
Radon Transform used	No	Yes
Data quality check	No	A check is added to check the input quality of the nlinfit call and set values to nan if they fail this criterion

Function analyzeBathyCollect		
	Version 1.2	**Version 2.0**
Function call to find **fDep**	**subBathyProcess**	**csmInvertKAlpha**
Initializing of variables	Not done	Initialized **fDep** and **camUsed**
Variable **kappa**	Input into **subBathyProcess**	Now directly computed and used within **bathyFromKAlpha**
Epoch Time	Incompatible units corrected	Not checked for compatibility
XYZ and **time**	Found inside main loop	Found outside main loop
Run time tracking	Not done	Added tic/toc to compute run time of processing

Function bathyCI		
	Version 1.2	**Version 2.0**
tstat3	Not included	Used if MATLAB Statistics toolbox is not available

Function predictCSM		
	Version 1.2	**Version 2.0**
Use of **phi**	**phi** input	**phi** computed implicitly

Bibliography

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.