cBathy User Manual

Draft 2.0

01/27/21

(please send comments and corrections to Rob Holman,
holman@coas.oregonstate.edu)

0. Preamble:

The cBathy algorithm was published in 2013 and was made freely available to the community. Since then, the algorithm has seen a number of small and large upgrades and now performs much more reliably and accurately. However, the user manual has not been correspondingly upgraded. The goal of this document is to update the original user manual to align it with the operations of the latest version, version 2.0.

1. Introduction:

The biggest limitation to the prediction of nearshore waves dynamics and circulation is typically a lack of recent, accurate bathymetry data. Since collecting bathymetry data using traditional survey methods is logistically difficult and expensive, there has been a long history, dating back to WWII, of attempts to estimate sufficiently accurate bathymetry based on various remote-sensing signatures[1]. Some success has been achieved but algorithms have typically not reached a production level that can be operated and trusted on a daily basis[2].

cBathy is a new algorithm for bathymetry estimation that is based on the well-known dependence of wave celerity, c, on depth. From extensive testing, the current algorithm appears to be quite accurate and robust to noise. This document is intended to complement the journal paper, herein referred to as HPH13, as well as later papers on version 2.0 (not yet submitted), and will not repeat the details of the algorithm or testing. Instead, this document is intended to serve as a user manual to introduce users to the practical aspects of cBathy collection and analysis and to provide a short description of the principles that allow cBathy to succeed even under very noisy conditions. Section 2 describes background about the complications of bathymetry estimation from optical sensors and the principles adopted by cBathy. Section 3 describes the principles for design of the sampling array and analysis design and discusses the three phases of analysis and the corresponding nature of the bathy structure that is the output by cBathy. Section 4 describes the process error that is required for Kalman filtering while section 5 discusses the settings file that is used to configure the analysis for any new site. Section 6 gives a short note on using cBathy with a parallel processing version of

[1] References and more extensive discussion are contained in Holman, et al, 2013, (hereafter HPH13) and will be omitted from the user manual.

[2] BeachWizard is an exception although it is computationally expensive, commonly used with dissipation data only that provide no information outside the surf zone, and requires manual monitoring to deal with the common noise sources in the nearshore and the complications of long-term sampling.

MATLAB. Finally, section 7 contains a lengthy description of debugging support in cBathy. Appendix A includes an m-file used to establish the cBathy pixel collection array for an example field site. While the algorithm has changed significantly in detail, from the user point of view version 2.0 is not substantially different from early versions. The main changes are significant additions in the returned bathy structure and the fact that performance results that used to require Kalman filtering of multiple 17-minute collections can now be achieved from individual 17-minute collections without Kalman filtering. Of course, Kalman filter can and should still be done when possible.

2. Principles and Objectives:

cBathy is based on the linear dispersion relationship,

$$s^2 = gk\tanh(kh) \qquad (1)$$

where σ and k are the radial frequency (equals 2π the wave period, T) and radial wavenumber (equals 2π divided by the wave length, L), respectively, g is the acceleration due to gravity and h is the depth. For the idealized lab conditions of a monochromatic wave field, the wave period and wavelength are easily identified so depth can be found as the only remaining unknown in equation (1).

In real seas, many complications arise. Waves are never monochromatic so the observed ocean surface is really the superposition of many sets of waves with different frequencies, wavelengths and directions (although each frequency-wavelength pair reasonably satisfies equation (1)). Moreover, optical contrast is dominated by short ocean wavelengths, or clutter, not the longer wavelengths that provide a better sensitivity to depth in equation (1) and contain the main energy that drives nearshore circulation. Also, optical data often includes serious contamination in the form of sun glint, rain drops on lenses and opaque fog. Manual intervention can eliminate these bad days, but this introduces an ongoing and unnecessary labor obligation.

As noted in HPH13, the goal of the cBathy algorithm is to extend previous work with an algorithm that is fully two-dimensional but benefits from the high-resolution capabilities of non-Fourier spatial methods. The method must be robust to noise and unanticipated signals like sun glare, passing clouds and rain spots on lenses, and must return confidence estimates that can be used in downstream products such as operational nearshore prediction models.

cBathy contains algorithms that reflect many decades of experimentation and experience. The goal of this user manual is to advise potential users on the implementation practices that will be most consistent with the assumptions and development philosophy of cBathy.

3. The Algorithm

The heart of cBathy is the determination of accurate wavenumbers for a number of candidate frequencies supplied by the user in the settings file[3]. This is carried out at an array of analysis points [x_m, y_m] that are also user supplied. From these results, depths can be estimated using equation (1). The algorithm works in three phases and the results are stored in a bathy MATLAB structure that also has three major components (one for each phase). Details of the three algorithm phases are contained in HPH13 and won't be described. Instead, the design of the sampling array and the details of the bathy structure will be described in sections 3.1 and 3.2, respectively.

Adopting cBathy for a new field site is fairly straightforward and involves three elements: 1) establishing and implementing an appropriate data collection (a fairly simple pixel stack collection in Argus), 2) modifying the settings file (params) to have values appropriate to the new site, and 3) adding a new case to findProcessError to describe the expected natural variability at the site. In addition, there may be bookkeeping issues to be resolved, for example file naming and file storage conventions. For Argus, these are well established but may take some design and trial and error for other data protocols and organizations.

It should be realized that cBathy assumes that the x-axis points offshore and the y-axis increases to the left as you face offshore. For other conventions, you should rotate into a local system like this and run cBathy in this new system. You can rotate back to local for subsequent products.

3.1 The Sampling Array

The input data for the main cBathy analysis (m-file analyzeBathyCollect) is an array of time series (an N by M array called 'data') collected at a grid of user selected 3D locations (an M by 3 array, xyz) at a regular set of sampling times (an N by 1 vector, epoch). In addition, analysis parameters, input filename and the start time of the analysis are passed in using the variable structure 'bathy' (see analyzeSingleBathyRun for an example of usage). These data can be the result of pixel data collections, radar data collections or collections from any other sensor available to the user, assuming sufficient density of coverage is available (see the following). This user guide will assume the input to be optical data collected by Argus and the spatial array will be referred to as a suite of pixel locations[4]. The user is responsible for establishing the data collections at reasonable resolution. Figure 1 shows an example pixel array for Duck, NC. Appendix A lists an m-file used to create a cBathy pixel sampling array for an example Argus site.

[3] The settings m-file creates a variable structure called 'params' that contains all the user-controllable options for any analysis. The terms settings and params will be used interchangeably in this text.

[4] To control data volumes, Argus pixel time series collections sample a small subset of all for the pixels in each image - see HPH13

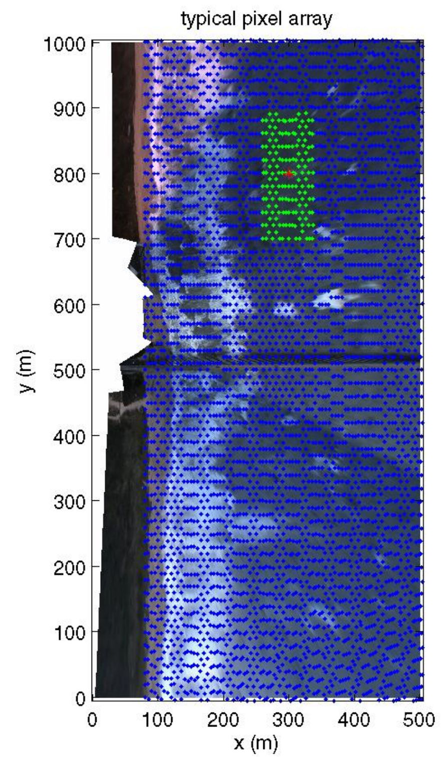

Figure 1. Standard pixel array for Duck, NC. Time series are collected at every dot (half omitted for clarity). Analysis at any example location (e.g. red asterisk) is based on data from surrounding tile of pixels (green dots) of size +/-Lx and Ly.

Temporal sampling at each pixel should be adequate to resolve the expected dominant ocean waves, consistent with how you would sample ocean waves with a single in situ sensor. For Argus, we use 2 Hz sampling and record lengths of 2048 points (a power of 2 to allow Fast Fourier Transforms) or approximately 17 minutes. Data runs are typically collected hourly but can be reduced to half-hourly for field experiments or places where conditions (for example, tide or morphology) change rapidly.

Spatial sampling should be adequate to resolve the anticipated dominant wavelengths. For Atlantic Coast US beaches, we have used cross-shore and alongshore spacing of 5 and 10 m, respectively. For an 8 s wave in 1 m depth, the wavelength will be ~25 m, so this pixel spacing provides 5 points per wavelength in the cross-shore (and alongshore length scales are longer, so can be well resolved with a larger sample spacing). More dense spacing can be used, but mostly slows down the processing time without a significant increase in accuracy.

Samples can feasibly be collected out to very large range (out to the horizon!). But at large range, the pixel resolution becomes worse than the requested array spacing so pixels smear together and waves are no longer resolved. It is recommended that sampling not be established with less than four pixels per expected wavelength.

3.2 The bathy structure (updated from Version 1.0)

The bathy variable that is returned by the cBathy algorithms is a compact structure that contains all of the relevant analysis results in one place. The overall structure contains six initial fields that document the analysis details, followed by three sub-structures, one for each phase in the analysis. An example of the format of the bathy structure is shown below for a specific size of analysis array, xm-ym:

```
bathy =

            epoch: '1285174800'
            sName: [1x64 char]
           params: [1x1 struct]
              ver: [1x1 double]
           matVer: [string]
             tide: [1x1 struct]
               xm: [1x43 double]
               ym: [1x41 double]
            timex: [41x43 double]
           bright: [41x43 double]
             dark: [41x43 double]
       fDependent: [1x1 struct]
        fCombined: [1x1 struct]
   runningAverage: [1x1 struct]
          cpuTime: [1x1 double]
```

Epoch is simply the start time of the data run in standard Unix epoch time (seconds since midnight, January 1, 1970) but could be any time stamp you wish to use. sName is simply the name of the stack file that was analyzed. In the Argus case, we list the filename of the first camera encountered (usually stack data are collected across multiple cameras). Params is a sub-structure that contains all the input settings from the Settings file (described in Section 5). This is stored both as a record of the parameters used and as a method to pass input variables into the analysis routines within analyzeBathyCollect. ver is the chosen version of cBathy while matVer is the version of matlab that was used.

Tide is a small structure that contains the tide level estimate used to correct the local cBathy depth estimates (measured relative to the sea surface) to a local vertical reference frame, for example NGVD88 or AHD. If bathy.tide.zt == nan, its default value, then no tide level correction has been made, so this field is important for documenting whether this correction has or has not been made (similarly, a change of tidal reference level requires adding the old, documented level, subtracting the new level and updating the bathy.tide fields to reflect the new level).

xm and ym are the vectors of analysis points for which bathy is estimated and are determined by the settings file, discussed in Section 5. Timex, bright and dark are coarse time-exposure, brightest and darkest images that have been calculated directly from the stack data (i.e. at all xm, ym locations). These can be used to, for example, determine where wave breaking was occurring.

The sub-structures fDependent, fCombined and runningAverage contain results from phases 1, 2 and 3 in the analysis and are described below. cpuTime documents the computation time taken, to aid in understanding code efficiency.

3.2.1. The fDependent Sub-structure

The fDependent structure contains output from cBathy phase 1 in which all results are expressed still as a function of frequency. Each field will include maps of results that have the same number of rows as are in ym, the same number of columns that are in xm, and the number of planes (maps) is equal to nKeep, the number of frequencies that the user has asked to save in the settings file. In the case below, the user asked to save the four strongest frequencies so the third array dimension is 4 and results are ordered from strongest to weakest signals. Note that the selection of the strongest frequencies is made on a point by point basis during analysis so that any plane of each variable (i.e. k(:,:,1)) will likely not refer to a single frequency but will instead refer to the frequency of the dominant signal at each analysis point. The field fB contains the actual frequencies selected for each location and plane[5]. Fields for which no satisfactory solution was obtained are left as nan's. This can be relatively common. Any variable can be plotted using (for example)

    imagesc(bathy.ym, bathy.xm, bathy.fDependent.hTemp(:,:,1).

Remember that this will not plot results from a single frequency (important for wave properties like wave angle). See section 7 for support routines.

The fDependent structure has the following fields:

```
bathy.fDependent =

          fB: [41x43x4 double]
           k: [41x43x4 double]
           a: [41x43x4 double]
       hTemp: [41x43x4 double]
        kErr: [41x43x4 double]
        aErr: [41x43x4 double]
    hTempErr: [41x43x4 double]
       skill: [41x43x4 double]
         dof: [41x43x4 double]
        lam1: [41x43x4 double]
     NPixels: [41x43x4 double]
      NCalls: [41x43x4 double]
       kSeed: [41x43x4 double]
       aSeed: [41x43x4 double]
     camUsed: [41x43x4 double]
```

'fB' contains the actual frequencies selected at each location for each plane of analysis (final dimension of the 3D arrays). 'k' contains the magnitude of the

[5] The debugging section below includes a routine that sorts bathy results by frequency

wavenumber (2π divided by the wavelength) and 'a' contains the angle of wave propagation in a "from" sense, in radians (for example, a = -0.5 means that those waves are arriving from 0.5 radians CW from normal (28° from the right of normal, looking offshore, since negative angles correspond to CW from the beach normal). 'hTemp' is the depth computed for this frequency-wavenumber pair using equation (1). These depths are not used in subsequent cBathy depth analysis but are included for diagnostic purposes.

For each variable, an associated error is also computed and saved as maps of 'kErr', 'aErr' and 'hTempErr'. In addition, the skill and number of degrees of freedom of the fit are saved as 'skill' and 'dof'. The skill indicates the percentage of the variance explained by fitting the local phase data to a planar surface (equation (3) in HPH13). Skill is used as a threshold quality control condition (threshold level included in the settings file). Finally, lam1 (the normalized eigenvector described in section 2.1 of HPH13) is recorded and also serves as a quality control variable to distinguish useful signals from noise.

Five new sub-fields have been added in version 2.0 as diagnostics. NPixels shows the number of pixels used in each tile, i.e. for each computation of k-alpha at any xm-ym. In contrast to version 1.0, version 2.0 uses variable size tiles depending on the expected wavelength of the waves being analyzed. NCalls simply counts the number of iterations for the nonlinear fitting at each analysis location and was used to guide efficiency. Version 2.0 includes a much-improved algorithm for finding seed values of k and alpha for the nonlinear search. These values are records in the fields kSeed and aSeed to help understand the performance of this seed algorithm. Finally, camUsed shows which camera was used for each tile. This is mostly relevant at camera seams where more than one camera spans the tile and only the dominant camera is used.

3.2.2 The fCombined Sub-structure

The fCombined sub-structure contains the phase 2 results from the cBathy algorithm. These are maps of the depth estimates at each analysis location that best fit a weighted combination of all of the frequency-wavenumber information available. This yields a single bathymetry map, along with fit error information.

The fCombined sub-structure has the following fields:

```
bathy.fCombined =

         h: [41x43 double]
      hErr: [41x43 double]
         J: [41x43 double]
      fBar: [41x43 double]
```

'h' is the best estimate of depth while hErr is 95% confidence interval on that estimate. J is the Jacobian returned by the nonlinear fitting routine. Finally, fBar is the average frequency used for each tile, dependent on a weighted average of the four fB values from Phase I.

Note that results from phase 1 and 2 will be tide-corrected (i.e. adjusted to the local vertical datum) if the value of bathy.tide.zt is other than 'nan'. In other words, a time series plot of depth estimates over a tidal cycle should show a constant value.

Final fCombined results for any run can be nicely displayed using the routine plotBathyCollect

3.2.3 The runningAverage Sub-structure

The runningAverage sub-structure contains results from the Kalman filter stage, phase 3 of the cBathy algorithm. This sub-structure and the Kalman calculations are unchanged in version 2.0. It contains the final results of cBathy, with errors, as well as information used in the Kalman filtering operation. While phases 1 and 2 of cBathy depend only on the current data collection, Kalman filtering depends on both the current and the previous bathy result. Since data collections are often retrieved from an Argus Station out of order, phases 1 and 2 are usually run by a separate code (analyzeBathyCollect) for a set of data collections, then phase 3 Kalman filtering is run later (using runningFilterBathy), after all data collections have been retrieved and analyzed to phase 2, to fill in the runningAverge structure. Filtering is a fast computation and can also be re-done at any time if the user wishes to change the filter in some way, for example by changing the process error. Running the Kalman filter (phase 3) does not change phase 1 and 2 results in any way (fortunately, since these early phases are the more computationally intense). 'nan' values of runningAverage results indicates that Kalman filtering has not yet taken place or that no valid data has yet been encountered at those pixels with nans. This might occur for analysis locations that are usually high on the beach, so see waves only on occasion.

The runningAverage sub-structure has the following fields:

```
bathy.runningAverage =

        h: [41x43 double]
     hErr: [41x43 double]
        P: [41x43 double]
        Q: [41x43 double]
        K: [41x43 double]
    prior: [1x64 char]
```

The fields 'h' and 'hErr' are the final depth estimates of cBathy and their estimated error, so these are the main products of the analysis. The fields 'P', 'Q' and 'K' are components of the Kalman computation, described in section 2.3 of HPH13. K is the critical value and is the Kalman gain that expresses the extent to which the current estimate is believed compared to the previous running average values and depends on the ratio of the error variance on the prior estimate (P) compared to the error variance of the current estimate (hErr^2) plus P (equation (5), HPH13). So when the current estimate is much noisier than the prior, K~0 and the new bathymetry estimate is essentially ignored. But when the current estimate is much better than the prior (for some reason), K~1 and the new result is dominated by the new information. For typical conditions, after the filter has incorporated a number of collections so has stabilized, K will commonly be around 0.1 so the new results only slowly nudge the average slowly.

'Q', the process error, represents to amount that our faith in the prior running average degrades with elapsed time due to natural sediment transport and changes in morphology. The process error is discussed further below.

The final field, 'prior', is simply the file name of the prior estimate used in the Kalman filter and can be used to confirm that files have been filtered in the proper order.

Final running average results for any run can be nicely displayed using the routine plotBathyCollectKalman.

4.0 The Process Error, Q

In the early versions of cBathy, the phase 3 Kalman filter processing was key to obtaining reliable, robust output based on noisy individual collection results that can sometimes be partly or completely unusable. Version 2.0 produces much better estimates from single runs, but Kalman filtering will always improve the output. New Kalman results are averaged into the running average result according to a pixel-by-pixel Kalman gain that expresses the believability of the new result compared to that of the running average (section 2.3 of HPH13). Since the running average bathymetry already reflects an average of many individual results, it is likely that new estimates will only slowly nudge the average, i.e. Kalman gains will typically be small.

If bathymetry were unchanging, then the estimated error variance of the running average bathymetry should monotonically reduce with additional estimates, much as predicted by the central limit theorem. However, bathymetry and nearshore morphology change in time due to sediment transport processes, so that confidence in a prior running average estimate should degrade with time since that estimate. For example, you would believe that a survey from yesterday would be a much better representation of today's bathymetry than a survey from last month or last year. This temporal degradation in confidence in a prior estimate (or temporal increase in error variance) is represented by the process error, Q.

There is little available data from which estimates of the magnitude and environmental dependences of Q can be made. HPH13 discuss one very extensive set of daily bathymetry surveys which were used to develop the following form:

$$Q(x,H_{m0}) = C_Q H_{m0}^2 \exp\left\{-\left[\frac{(x-x_0)}{\sigma_x}\right]^2\right\}\Delta t \tag{2}$$

Bathymetric variability was found to increase linearly with time, Δt, and with the square of the offshore wave height. Bathymetric variability is expected to vary spatially, being largest near the shoreline and sand bar system and lower offshore. This is modeled with a Gaussian in cross-shore distance, centered at a location x_0 with standard deviation σ_x.

The 38-day bathymetric data set used to establish this form showed the largest variability to be in a narrow spatial band around the bar location at that time. However, we know that bars move in the cross-shore over time so have used a user-selected broader set of values for the spatial Gaussian form ($x_0 = 150$, $\sigma_x = 100$ for Duck, NC).

It is recommended that users applying cBathy to new sites use their best judgment to establish the coefficients for their new site in the routine 'findProcessError', choosing spatial parameters that center on locations where the maximum variability is expected and choosing a spread, σ_x, that allows reasonable variability over the region of expected climatological bathymetric variability (a reasonable sand bar envelope region). Criteria for selecting an appropriate value of C_Q for a new site are not well known. A value that is too large will lead to running average bathymetries that respond quickly to changes and presumably to noise. A value that is too small means that bathymetric results will vary more slowly than truth and will lag behind natural changes at the site. Development of an improved understanding of appropriate values is an ongoing research topic at the Coastal Imaging Lab.

Examples of Q for different sites are contained in the routine findProcessError.

5.0 The Settings File

The settings file contains all the site-specific analysis parameter in one structure, 'params'. For Argus, settings files are m-files that are stored in the directory cBathy/SETTINGS/ and are named after the Argus station name. For example, for the site argus02b (our name for Duck), the m-file is called argus02b and has the following content:

{Contents of file argus02b.m}
##

```
%%% Site-specific Inputs
params.stationStr = 'argus02b';
params.dxm = 10;                    % analysis domain spacing in x
params.dym = 25;                    % analysis domain spacing in y
params.xyMinMax = [80 500 0 1000];   % min, max of x, then y
                                   % default to [] for cBathy to choose
params.tideFunction = 'cBathyTide';  % tide level function for eval
```

```
%%%%%%%   Power user settings from here down   %%%%%%%
params.MINDEPTH = 0.25;            % for initialization and final QC
params.QTOL = 0.5;                 % reject skill below this in csm
params.minLam = 10;             % min normalized eigenvalue to proceed
params.Lx = 2*params.dxm;          % tomographic domain smoothing
params.Ly = 2*params.dym;          %
params.kappa0 = 2;                 % increase in smoothing at outer xm
params.DECIMATE = 1;            % decimate pixels to reduce work load.
params.maxNPix = 80;       % max num pixels per tile (decimate excess)
params.minValsForBathyEst = 4;     % need this many pixels to solve
params.shortLengthNFreqs = 4;      % need this many for coherence
                                   % versus magnitude shorting sorting

% f-domain etc.
params.fB = [1/18: 1/50: 1/4];           % frequencies for analysis
(~40 dof)
params.nKeep = 4;                  % number of frequencies to keep

% debugging options
params.debug.production = 1;
params.debug.DOPLOTSTACKANDPHASEMAPS = 1;  % top level debug of phase
params.debug.DOSHOWPROGRESS = 1;           % show progress of tiles
params.debug.DOPLOTPHASETILE = 1;   % observed and EOF results per pt
params.debug.TRANSECTX = 200;        % for plotStacksAndPhaseMaps
params.debug.TRANSECTY = 900;        % for plotStacksAndPhaseMaps

% default offshore wave angle.  For search seeds.
params.offshoreRadCCWFromx = 0;
```

params.nlinfit = 1; % flag, 0 = use LMFnlsq.m to do non-linear fitting
###

The main items to change for a new site are the first five entries. stationStr is simply the Argus name of this site, which is used to key the proper tide retrieval and also to select the correct form for the process error in findProcessError (and possibly some minor labeling issues). dxm and dym are the desired cross-shore and alongshore spacing for analysis points (discussed in section 3.1) while xyMinMax specifies the spatial extent of the analysis grid. If no values are entered for xyMinMax, the program will default to values that just span the supplied pixel array. However, this can create problems if the pixel array changes size at any point in the collection lifetime because the returned bathymetry arrays will change size and the Kalman filter will crash. So it is better to hard code analysis domain limits in xyMinMax. Finally, tideFunction is a user provided function that will be called to find the tide elevation correction. This may require some user fiddling for non-standard Argus sites.

The next section of code is described as being for power users only (down to the f-domain section) and can be changed, but only if you understand what you are doing. MINDEPTH is the limit set for the nonlinear depth search in phase 2 (i.e. values outside this limit are rejected and nan’s returned). MINDEPTH is invoked because shallow water celerity is not valid in the swash, so this should not likely be changed.

QTOL and lam1 are threshold quality control parameters. Results will be rejected if the skill of the fit in phase 1 is less than QTOL or if the normalized variance explained by the first EOF is less than lam1 (see HPH13). These values can be reduced if data sets will always be noisy for some reason and answers are still desired. But this is not recommended.

Lx and Ly are smoothing length scales for the spatial extent of the Hanning smoothing function in several weighting schemes in the analysis. It should be determined as the reasonably desired spatial resolution of the underlying bathymetry. Allowing these values to scale as twice the pixel spacing (as illustrated) forces consistency in the analysis and also ensures that a reasonable number of pixels are included in fitting phases to a planar surface. Since the natural scales of morphology typically become larger offshore, it makes sense to increase this smoothing scale from the shoreward to the seaward boundary. Kappa0 is a multiplier that increases Lx and Ly linearly from the shoreward to the seaward boundary (so smoothing increases linearly from Lx and Ly at the shoreward boundary to kappa0 times these values at the seaward boundary). For a site like a tidal delta for which shorter scale features may be widespread, kappa0 should be set to 1.0 (no increase). Note that the effects of currents in this type of environment are not accounted for yet in cBathy so errors will occur (this is an ongoing research problem). The increase in Lx due to kappa0 > 1 means that the number of pixels in each analysis tile will also increase. Since execution time depends on the square of the number of pixels, this can slow processing substantially. maxNPix sets a maximum number of pixels per tile, with the residual removed by decimation. Note that the example value of 80 points yields plenty of data for fitting the phase data to a planar surface (80 points to solve for two slope values). DECIMATE allows brute force decimation of excess pixels and was used in development to speed processing. This should not be changed unless you need speed for debugging.

fB is the list of frequencies for phase 1 analysis. The range of frequencies (min and max) should match the likely wave frequencies that will be encountered at your site. For example, Pacific Ocean beaches may see periods as long as 18 s and down to 3 or 4 seconds whereas a semi-enclosed sea may see no waves longer than 10 or 12 seconds. Note that short waves are sensitive only to shallow depths, so are not as helpful as longer period waves. The frequency resolution (in this case, 1/50) should be selected to yield a cross-spectrum with adequate stability. Functionally, this means that each frequency band (1/50 Hz wide in this case) should include approximately 20 Fourier frequencies (40 degrees of freedom), so the frequency resolution should be chosen as around 20 divided by the length of the time series in seconds (1024 s for typical Argus collections). Higher resolutions can be used (fewer Fourier frequencies than 20) but choices should be made with a good understanding of the signal processing consequences. nKeep is the number of frequencies that will be analyzed, prioritized by the overall coherence over the analysis tile for each frequency. Increasing this number will slow computation and could provide better stability in cases with broad spectral energy (waves with a wide range of frequencies typically present), but in most cases examining extra frequencies will simply be analyzing noise with little gain.

The debugging options in params will be discussed in the section on debugging.

The final parameter, offshore RadCCWFromx is the offshore wave angle, measured in radians CCW from shore normal, that serves as the seed for the nonlinear search in phase 1. Typically this value will be 0.0 on the assumption that waves will, on average, come from offshore. For a site for which waves predominantly come from a particular direction, changing his value may improve the speed of the search. Note that this value is in RADIANS. Changing this value is not recommended except for very special cases. {This value does not appear to be used in V2.0}.

Note that there have been some changes since version 1.0. MAXDEPTH is no longer needed since the algorithm automatically de-weights deeper values. minValsFromBathyEst is the minimum number of pixels allowable for a successful solution. shortLengthNFreqs specifies the minimum number of frequencies per frequency band for which you can still use the coherence sorting methods for prioritizing frequencies. If you have less than this number (because you have short record lengths) then you should use the special development version, 3.0, of cBathy. The nlinfit allows you to run the analysis without the matlab toolbox that includes nlinfit, substituting instead an open source version LMFnlsq.m which is a bit slower. Note also that despite the phase I calculations using automated tile sizing, you still need to specify Lx, Ly and kappa0 fields since these are used in the phase II smoothing.

6.0. Parallel code

cBathy is compute intensive and can take significant processing time. The code has been written to be compatible with a parallel processing version of MATLAB, if this is available. This requires declaring a pool of parallel processors in MATLAB (use help matlabpool). Using 6 processors on a fairly standard linux machine yields approximately 100 second run times at OSU. Without parallel processing, runs take more than 200 seconds.

7.0. Debugging

cBathy contains a suite of algorithms and will run even when presented with random data (although nan's will be returned). So when the program returns results that don't seem sensible, it is important to have a suite of good debugging tools and to understand the nature or and evidence for possible failure modes. Debugging for phase 1 computations is controlled by a series of debug flags in the settings file. The first flag, 'production', is a master switch. If set to 1 (true), no further debugging will be done. If set to 0, the remaining five flags and parameters determine the debugging actions.

7.1. Phase 1 Debugging

The main reason for bad results is bad input data, i.e. no waves are present. The most direct way of testing the quality of the data is first to look at the time stacks to ensure that wave signals are present, and second to look at example phase maps to ensure that progressive waves are being detected. A best first step in examining your data is to view the stack as a movie using the routines

```
[x,y,map,wt] = findInterpMap(xyz,pa,map,kn,doNan);
```

Then

```
out = useInterpMap2( in, map, weight );
```

This will result in the stack data being played as a movie so that you can ensure that you see visible waves progressing toward the beach (not, for example, a fog bank).

More detailed checks are made using the debug flag DOPLOTSTACKANDPHASEMAPS. Since this aspect of debugging is done at run time, the easiest way to examine a data collection for Argus is to use analyzeSingleBathyRun(stackName) where you chose some representative stack. Again, for non-Argus systems, this file may need editing to make it consistent with local conventions. By default, the analysis will proceed from shoreward to seaward, so will commonly start with analysis locations on the beach or in the swash zone that are not very informative. You can change to a more typical location (say just outside the surf zone) by specifying a new minx, maxx, dx, miny, maxy, dy to make a new analysis sub-array at locations that are much more representative. For example, at Duck, I might use [200 250 25 700 750 25] to examine a set of analysis tiles that are slightly offshore and away from the FRF pier.

In debug mode, cBathy starts by producing two figures that allow you to examine the entire pixel data collection. MATLAB Figure 10 shows example cross-shore and alongshore time stacks to test whether waves are visible in the original stack data. The alongshore location for the cross-shore transect is determined by the params file field debug.TRANSECTY while the cross-shore location of the alongshore transect is determined by debug.TRANSECTX. The user should set these values to be representative of where they wish to examine the waves. The figure has a 2 by 2 array of plots with the upper two showing the location of the transect pixels from within the pixel array (just to confirm the locations) and the lower two subplots showing the time stacks for the cross-shore (left) and alongshore (right) transects. The main diagnostic figure is the bottom left plot showing the cross-shore timestack. Figure 2 shows an example stack from Duck, clearly showing the hoped-for oblique trajectories of waves as they progress landward (toward the left) in time (toward the bottom). I have manually zoomed in on this figure to focus on a short time span so that the waves are more visible. You should ensure that you also see shoreward wave progression.

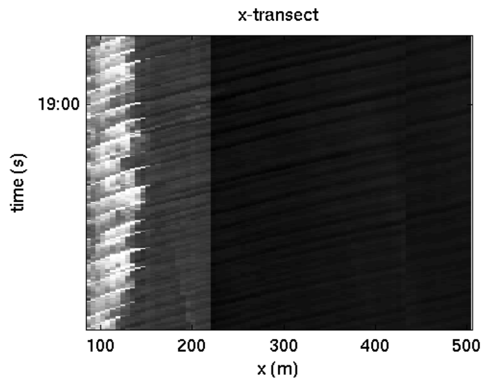

Figure 2. Representative cross-shore time stack from cBathy plot figure 10, subplot(2,2,3). Oblique trajectories from top right to lower left correspond to waves propagating from offshore to landward (left) as time passes (lower in the figure). The figure has been manually zoomed (in the time axis) to allow propagation to be clearly seen (you should also do this to ensure that you actually see wave propagation). The right-hand portion of the figure is darker because those pixels are from a different camera than on the right and had a different gain. This is normalized in the analysis so is not an issue.

cBathy Figure 11 shows phase maps over the entire collection array for each of the candidate frequencies, fB, selected by the user in the settings file. User manual Figure 3 shows several example frequencies (2 out of 11 possible fB values, in this case). The main point of this figure is also to determine if the pixel stack contains wave-like motions. This is confirmed by seeing full color banding in this figure showing that cBathy is detecting wave phases changing spatially through the full range from $-\pi$ to π (a smooth-ish transition from blue through yellow to red then a jump back to blue) with spatial scales that seem comparable to ocean wave scales. Bad data will yield colors that are not the full range or will yield patterns without wave-scale features. Note that these phase maps are created for a suite of individual (single) frequencies bands in the Fourier transform, so are typically quite noisy. Thus, the patterns may look less organized than you might hope for ocean waves, particularly for frequencies with little ocean energy on that day. Similarly, some high frequencies may show up with long apparent wavelengths if they correspond to harmonics of a stronger low-frequency wave swell. The main point is to ensure that wave-like motions are detected.

Continuing after the prompt by hitting carriage return, cBathy will step through the analysis locations supplied by the user showing map results for each tile and frequency as well as numerical results. An example tile map is shown in Figure 4 below and has a 2 by 2 array of subplots (note that multiple plots will be made and may be lying on top of each other so might need rearranging on your screen). The left two subplots show the observed and modeled phase surface for the first EOF for a particular frequency while the right plots show the observed and modeled amplitude maps. The observed phase map should show a phase ramp from right to left (jumping at each transition from π to $-\pi$), illustrating shoreward propagation of the wave. The magnitude and direction of wavenumber are found from the slopes of this phase surface.

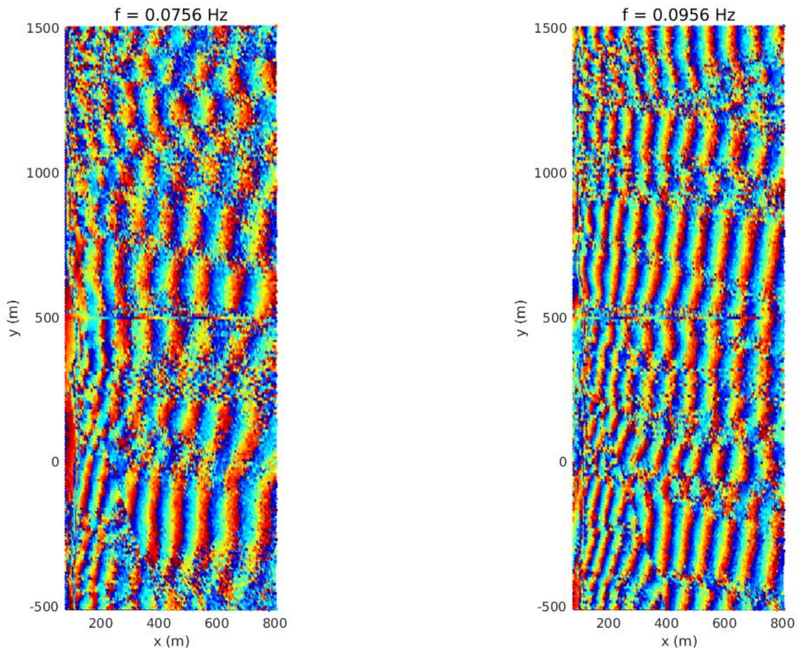

Figure 3. Phase maps for the entire collection domain for 2 of 11 possible frequencies from an example analysis. Colors show wave phase varying reasonably from –π to π with scales typical of ocean waves, confirming that the input stack data appears to have useful signals. Note that the higher frequency waves in the right panel have shorter wavelengths than the lower frequency waves on the left panel, consistent with the dispersion relationship.

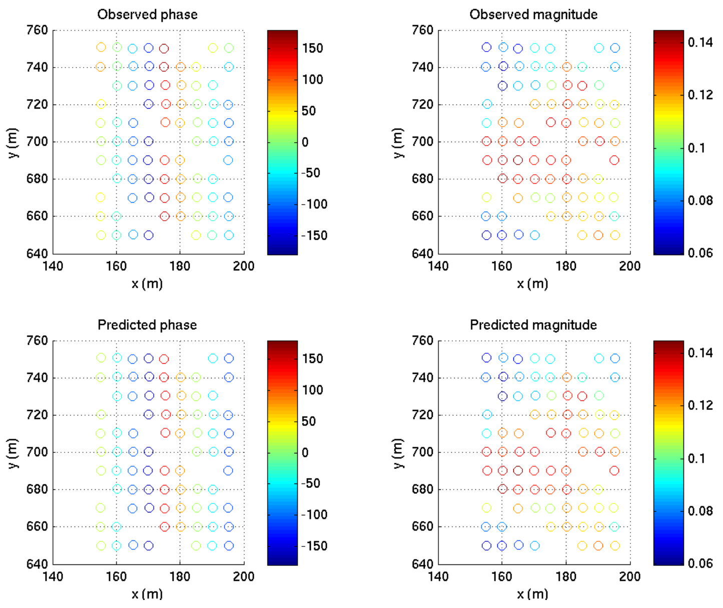

Figure 4. Phase maps of the dominant EOF within an example analysis tile with EOF phase on the left and magnitude of the right. The observed values (upper panel)

should match the modeled values (lower panel). This example is MATLAB window 4, so corresponded to the fourth-most important analysis frequency for this tile.

The important aspect of tile phase maps like Figure 4 are that the observed and modeled phase maps look alike. This means that the MATLAB fit to a progressive plane wave has been successful.

This debug option also lists the frequency dependent results for each tile, for example as shown in Figure 5, a case for which the dominant 4 frequencies were saved (nKeep = 4 in the settings file).

```
frequency 1 of 4, normalized lambda 44.4
  frequency 2 of 4, normalized lambda 39.3
  frequency 3 of 4, normalized lambda 35.0
  frequency 4 of 4, normalized lambda 31.3

fDependent =

          fB: [0.0956 0.1156 0.0756 0.1756]
           k: [0.1132 0.1362 0.0918 0.2145]
           a: [0.0057 0.0435 0.0405 -0.0015]
         dof: [16.3925 18.9482 16.8483 17.5866]
       skill: [0.9522 0.9350 0.9020 0.8782]
        lam1: [44.3637 39.2987 34.9696 31.2874]
        kErr: [0.0549 0.0545 0.0549 0.0523]
        aErr: [0.1827 0.1597 0.2297 0.0960]
       hTemp: [2.9704 3.0573 2.7807 3.0715]
    hTempErr: [2.9153 2.5250 3.3081 1.6877]
```

Figure 5. Text listing of frequency-dependent results for an example tile, explained in the text.

The first four lines list the normalized eigenvalue for each of the four dominant frequencies. These must exceed lam1 from the setting file to be retained with large values like these indicating that the first EOF has a lot more energy than expected by chance. The following lines show all of the f-Dependent results from the bathy structure. It is often useful to look at the frequencies since these should be comparable to those from a wave gage; the wave angle, a, since these should make sense; the skill since this expresses the quality of the fit; and the individual estimates of depth, hTemp, which should be reasonable and (ideally) consistent.

7.2. Phases 1 and 2 Debugging

The MATLAB routine 'examineSingleBathyResult' is a simple way to look at the main results from phases 1 and 2 AFTER computation of the bathy structure. This routine is a bit simpler to use since you need only supply the bathy structure as input. It is useful to determine if results look sensible, but it doesn't have the base level diagnostic capabilities like time stacks and phase maps of the above debugging routine.

Unlike the previous routine, examineSingleBathyResult is sorted by frequencies from the lowest possible value of fB to the highest. For each frequency, the standard products of wavenumber, wave angle, hTemp and phase 2 depth are mapped along with their errors. Because nKeep is usually much smaller than the list of possible frequency values listed in the settings file, there will usually be maps or regions of maps that are filled with nans, indicating that no values were returned for that frequency for that location. However, there are real advantages in seeing some variables like wavenumber and wave angle sorted by frequency rather than by importance in the analysis. For interpretation, the last subplot (lower right corner) shows maps of the order of importance of the plotted frequency (so a value of 2 for a map location means the displayed frequency was the second-most important at that location). Cycling through the frequencies is a good way to get a feel for which frequencies are contributing to the final depth estimate, h, for any data run.

Note that the top center (untitled) plot is the wave angle, a.

The routine plotBathyCollect provides a nicely formatted display of the phase 2 bathymetry and error.

7.3. Phase 3 Kalman Debugging.

Kalman filtering results can be examined for Argus using the routine showHourlyKalmanResults and specifying the station and the beginning and end times to examine. This routine calls the m-file plotBathyKalmanStep to display each ongoing result in a series of sequential bathy estimates. For non-Argus systems, you should call this function from within a wrapper m-file that is consistent with your system.

For each bathy file within the sequence, this routine plots the prior running average bathymetry, the current estimate (from fCombined in this bathy record), the updated running average, as well as errors on the prior and current estimates. The last panel is the Kalman gain. Cycling through a series of results lets you examine how the Kalman filter ingests good and bad data and how the final averaged results evolve. It is instructive to watch the Kalman gain since these numbers should normally be small once the filter has settled down (ingested a few days worth of hourly data).

Appendix A: Example Pixel Collection m-file

The following is an example m-file used to create a cBathy array for Agate Beach, Oregon, in 2011. It works only for Argus systems but can be used as an illustration or template for other systems. The cBathy-specific part of the code is the five lines follow the comment '% create a large matrix coverage for cBathy' and simply creates a matrix of pixel instruments over the sampling area with x and y spacing of 10 and 20 m, respectively.

```
% pixel array design for Agate Beach
```

```
%
%  Designed 10/05/11
%  Holman

clear all;
DBConnect;
global DBIsOkForAutoGeoms
DBIsOkForAutoGeoms ='noautook'

PIXForget;
PIXSetStation('argus00');

% create a large matrix coverage for cBathy
x1 = 150; x2 = 2500; dx = 10.0;
y1 = -1300; y2 = 2000; dy = 20;
tide=0;
iid1=PIXCreateInstrument('mBW','matrix', PIXFixedZ+PIXDeBayerStack);
PIXAddMatrix(iid1,x1, y1, x2, y2, tide, dx, dy);

% Now schedule the collection.
epoch=matlab2Epoch(now);
cams = [0 1 3];
idBWPack = PIXCreatePackage(demoTest, [iid1]);
r=PIXBuildCollect(idBWPack,epoch,tide,cams);
PIXScheduleCollectIII(cams,2048,2,r);
```