Implementation of nonlinear sensitivity measures in PASTET

Final Report

Terry J. Deveau
JASCO Applied Sciences

Prepared By:
JASCO Applied Sciences
32 Troop Ave,
Dartmouth, NS, Canada B3B 1Z1

Contract Project Manager: John Moloney, 902-405-3336
PWGSC Contract Number: W7707-098179/001/HAL
CSA: Sean Pecknold, Defence Scientist, 902-426-3100

Defence R&D Canada – Atlantic

Contract Report
DRDC Atlantic CR 2013-086
December 2013

Abstract

DRDC Atlantic has been developing a research-level acoustic prediction system in support of tactical decision aids and improving operator effectiveness, the System Test Bed (STB). The Portable Acoustic Sensitivity and Transmission Estimation Tool (PASTET) has been, and is being, developed as an adjunct to the STB and extension of existing acoustic propagation modeling capabilities. The first objective of this call-up is to include in PASTET the capability for using an Empirical Orthogonal Function (EOF) analysis of sound speed profiles (SSP) to provide the basis for a nonlinear estimate of the sensitivity of transmission loss (TL) to SSP variance, using a Monte Carlo methodology. This is an extension of the present capability for linear estimation of TL sensitivity to SSP standard deviation.

As a second objective, this call-up also included the capability in PASTET for a nonlinear estimate of the TL sensitivity to bathymetry variability, through a power spectrum analysis of fluctuations and a Monte Carlo methodology. A third objective, also using a Monte Carlo methodology, is to include an option to handle spatial field shifting, which finds the correlation between a reference acoustic field and its perturbed variations.

In addition to the three principal objectives, this call-up is tasked with fixing a number of software defects that have previously been identified in PASTET. This report describes the work that was done during the call-up, the issues that were encountered, and the results that were produced. It includes recommendations for additional work.

Résumé

RDDC Atlantique a mis au point le System Test Bed (STB), un système de prévision acoustique de recherche, dans le but d'appuyer le développement d'aides à la prise de décisions tactiques et l'amélioration de l'efficacité de l'opérateur. Le Portable Acoustic Sensitivity and Transmission Estimation Tool (PASTET) (outil portatif d'estimation de la transmission et de la sensibilité acoustiques) est un logiciel conçu en vue d'appuyer le STB et d'étendre les capacités existantes de modélisation de la propagation acoustique dont le développement se poursuit. Le premier objectif de la commande subséquente consiste en l'intégration d'une analyse de fonctions orthogonales empiriques (FOE) de profils de vitesse du son (PVS) dans PASTET, analyse qui servira de base à l'estimation non linéaire de la sensibilité à l'affaiblissement de transmission (AT) d'une variance de PVS effectuée à l'aide d'une méthode de Monte-Carlo. Il s'agit d'une extension de la capacité actuelle d'estimation linéaire de la sensibilité à l'AT par rapport à l'écart-type de PVS.

Le deuxième se veut l'ajout dans PASTET d'une capacité d'estimation non linéaire de la sensibilité à l'AT de la variabilité bathymétrique effectuée à l'aide d'une analyse du spectre de puissance des fluctuations et d'une méthode de Monte-Carlo. Le troisième objectif, qui repose également sur une méthode de Monte-Carlo, consiste à permettre la prise en charge du changement de champ spatial, qui détermine la corrélation entre un champ acoustique de référence et les variations perturbées de celui-ci.

En plus de ces trois objectifs principaux, la commande subséquente vise la correction d'un certain nombre de défectuosités logicielles relevées dans PASTET. Le présent rapport décrit les travaux effectués dans le cadre de la commande, les problèmes survenus et les résultats produits. Il comprend également des recommandations quant à des travaux supplémentaires

Executive summary

Implementation of nonlinear sensitivity measures in PASTET: Final Report

Terry J. Deveau; DRDC Atlantic CR 2013-086; Defence R&D Canada – Atlantic – December 2013.

Introduction: This work extends the capability of PASTET (Portable Acoustic Sensitivity and Transmission Estimation Tool) to perform a nonlinear analysis of transmission loss (TL) sensitivity to sound speed profile (SSP) variance through an Empirical Orthogonal Function (EOF) analysis and a Monte Carlo methodology. Progress has also been made on further extending PASTET to perform a nonlinear analysis of TL sensitivity to bathymetry variability, through a power spectrum analysis of fluctuations and a Monte Carlo methodology. An additional option to handle spatial field shifting, which finds the correlation between a reference acoustic field and its perturbed variations, was also started. This work additionally addresses some of the software defects that have previously been identified in PASTET.

Results: The PASTET program has been extended to perform an EOF analysis on SSP data and to use it as the basis for a nonlinear estimate of TL sensitivity to SSP variance, using a Monte Carlo methodology. Although work was begun on the nonlinear bathymetry variations sensitivity analysis and the spatial field shifting options, those were not completed. Some corrections to previously identified software errors and deficiencies in the PASTET program were also performed under this call-up.

Significance: Oceanography varies in time and space, resulting in fluctuating transmission loss and therefore sonar performance. In order to obtain good estimates of the variance in sonar performance, a representative sound speed profile and physically meaningful variations of it must be computed. The EOF analysis is a method to computationally obtain a small set of eigenfunction profiles that are demonstrably representative of a large set of SSPs, and which capture a large and measurable fraction of the statistical variance that is present in the large set. These EOF profiles are then used to synthesise a number of randomly generated SSPs for the sensitivity runs. The SSPs generated in this way are intended to represent of the variability present in the larger set, yet remain physically reasonable SSP examples, as opposed other approaches which yield physically unreasonable SSP examples.

Future plans: Providing a robust and portable tool for rapid assessment of sonar performance sensitivity to variations and uncertainty in ocean environmental parameters is the ultimate goal. Preparations have been made in the PASTET software for future work to complete the nonlinear bathymetric sensitivity analysis option, using a power spectrum of bathymetric variations. To a lesser extent, similar preparations have also been made for completing the spatial field shifting option. Additional improvements and enhancements to PASTET's facilities for accessing external data sets, processing the data, performing acoustic modeling, and preparing graphical output products are also seen as near-term objectives for continued PASTET work.

Sommaire

Implementation of nonlinear sensitivity measures in PASTET: Final Report

Terry J. Deveau; DRDC Atlantic CR 2013-086; R & D pour la défense Canada – Atlantique; decembre 2013.

Introduction : La commande subséquente vise à accroître la capacité du Portable Acoustic Sensitivity and Transmission Estimation Tool (PASTET) (outil portatif d'estimation de la transmission et de la sensibilité acoustiques) de manière à permettre l'analyse non linéaire de la sensibilité à l'affaiblissement de transmission (AT) de la variance de profils de vitesse du son (PVS) à l'aide d'une analyse de fonctions orthogonales empiriques (FOE) et d'une méthode de Monte-Carlo. Des progrès ont été réalisés quant à l'ajout de capacités à PASTET en vue d'exécuter une analyse non linéaire de la sensibilité à l'AT de la variabilité bathymétrique à l'aide d'une analyse du spectre de puissance des fluctuations et d'une méthode de Monte-Carlo. Des travaux ont également été entamés en vue de la prise en charge du changement de champ spatial, qui détermine la corrélation entre un champ acoustique de référence et les variations perturbées de celui-ci. Ils visaient de plus à corriger certaines des défectuosités logicielles cernées précédemment dans PASTET.

Résultats : Le programme PASTET a été modifié de manière à ce qu'il puisse exécuter une analyse de FOE de données de PVS et utiliser ladite analyse en tant que fondement à une estimation non linéaire de la sensibilité à AT de la variance de PVS à l'aide d'une méthode de Monte-Carlo. Bien que nous ayons entamé les travaux sur l'analyse de la sensibilité aux variations bathymétriques et sur les options de changement de champ spatial, ceux-ci ne sont pas terminés. Nous avons de plus corrigé certaines erreurs logicielles et lacunes relevés précédemment dans le programme PASTET.

Importance : L'océanographie varie dans le temps et l'espace, ce qui occasionne un AT fluctuant qui influe sur le rendement du sonar. Pour obtenir de bonnes estimations de la variation du rendement du sonar, il faut calculer un PVS représentatif et les variations pertinentes sur le plan physique du profil. L'analyse des FOE permet d'obtenir par des calculs un petit ensemble de profils de fonction propre qui constitue une représentation manifeste d'un ensemble plus gros de PVS et qui saisit une fraction considérable et mesurable de la variance statistique dans ce second ensemble. Les profils de FOE servent ensuite à faire la synthèse d'un certain nombre de PVS générés de façon aléatoire en vue d'analyses de sensibilité. Les PVS ainsi générés visent à représenter la variabilité présente dans le plus gros ensemble, tout en demeurant raisonnables sur le plan physique, contrairement à d'autres méthodes qui fournissent des exemples de PVS irréalistes à ce titre.

Perspectives : L'objectif ultime est de fournir un outil robuste et portatif d'estimation de la sensibilité du rendement des sonars aux variations et à l'incertitude des paramètres environnementaux de l'océan. Nous avons fait des préparations dans le logiciel PASTET en vue de recherches futures, qui permettront d'achever la capacité d'analyse de la sensibilité bathymétrique non linéaire à l'aide du spectre de puissance des variations bathymétriques. Nous avons également fait des préparations similaires, dans une moindre mesure cependant, en vue de

l'achèvement de la prise en charge du changement de champ spatial. D'autres améliorations des mécanismes dont dispose PASTET pour accéder à des ensembles de données externes, traiter des données, réaliser la modélisation acoustique et préparer des produits de sortie graphique sont des objectifs à court terme des travaux en cours du programme PASTET.

This page intentionally left blank.

Table of contents

List of figures

Acknowledgements

Sean Pecknold is the overall scientific authority for this project. He provided the initial C++ / Fortran source code and Eclipse project files for PASTET (Portable Acoustic Sensitivity and Transmission Estimation Tool) as a starting point for the work of the present call-up 015, as they were left after the last internal DRDC software development on PASTET in the spring of 2012. Dr. Pecknold also provided the IDL (Interactive Data Language) code that was used as the template for the development of the C++ code for EOF (Empirical Orthogonal Function) analysis of SSP (Sound Speed Profiles) in PASTET.

This page intentionally left blank.

1 Introduction

DRDC Atlantic has been developing a research-level acoustic prediction system in support of tactical decision aids and improved operator effectiveness: the System Test Bed (STB). The Portable Acoustic Sensitivity and Transmission Estimation Tool (PASTET) has been developed as an adjunct to, and extension of, existing underwater acoustic propagation modeling capabilities. Its primary function is to model the sensitivity of underwater acoustic propagation to the variations in the environmental conditions that most directly impact the propagation.

Previous call-ups have progressed PASTET software development, from an MS-Windows application that was embedded within a much larger proprietary third-party application, to a standalone Linux-based system (which can still run under MS-Windows, if desired, using Cygwin). The data access capability was also expanded to use the new DRDC PDB and the STB-based Environmental Modelling Manager (EMM) data.

Sound speed profiles (SSPs) are a critical input to acoustic propagation modelling. They vary with time and space in the ocean. In order for PASTET to estimate the acoustic transmission loss (TL) sensitivity to these variations, it must run multiple TL calculations with different SSP inputs. Choosing a representative set of SSPs for these runs is the essential challenge. The Empirical Orthogonal Function (EOF) analysis is a method to computationally obtain a small set of eigenfunction profiles that are demonstrably representative of a large set of SSPs, and which capture a large and measurable fraction of the statistical variance that is present in the large set. These EOF profiles are then used to synthesise a number of randomly generated SSPs for the sensitivity calculations. The SSPs generated in this way are intended to represent the variability present in the large set, yet remain physically reasonable SSP examples, as opposed to a mean ± standard deviation approach, which typically yields physically unreasonable SSP examples.

The first objective of this call-up 015 under the STB maintenance standing offer contract[1] is to enhance PASTET with the capability for using an EOF analysis of SSPs to provide the basis for a nonlinear estimate the sensitivity of transmission loss (TL) to SSP variations, using a Monte Carlo methodology. This is an extension of the present capability for linear estimation of TL sensitivity to statistical SSP standard deviation. As a second objective, this call-up is to augment PASTET with the capability for a nonlinear estimate of the TL sensitivity to bathymetry variability, through a power spectrum analysis of fluctuations and a Monte Carlo methodology. A third objective, also using a Monte Carlo methodology, is to include an option to handle spatial field shifting, which finds the correlation between a reference acoustic field and its perturbed variations. In addition to the three principal objectives, this call-up is tasked with fixing a number of software defects that have previously been identified in PASTET.

This report describes the work that was done during this call-up, the issues that were encountered, and the results that were produced; it includes recommendations for additional work.

1.1 Tasking and Scope

Under Call-Up 015 of the STB maintenance standing offer contract, JASCO (the contractor) was assigned the following tasks, to be performed in collaboration with the Scientific Authority.

1) Nonlinear sensitivity measures (algorithm enhancements)

    a. Sound speed profiles

        i. Currently, the point-by-point mean and standard deviation of the sound speed profile set are determined and used to generate perturbed fields for the PASTET calculations, using the "linearized" sensitivity measures described in [Dosso et al (2007) J. Acoust. Soc. Am. 121 (1), 42-45]. This will be generalized to allow use of the "nonlinear" measure, in the following way. Given a set of sound speed profiles, the mean will be extracted, and a set of empirical orthogonal functions of the set of sound speed profiles along with their associated variances will be determined (effectively, the eigenvectors and eigenvalues corresponding to the sound speed profiles). These values will then be used to generate a set of realizations of sound speed profiles for a Monte Carlo sensitivity run. Example IDL code shall be supplied by the Scientific Authority.

    b. Bathymetry

        i. An option for using a nonlinear measure for bathymetric variability will be added in the following way: given the existing returned bathymetry used for the linearized sensitivity measure, the power spectrum of bathymetric fluctuations will be determined in the required area. Utilizing this spectral slope (reduced to one-dimension), a set of random realizations of bathymetry with variability given by this spectrum at the scale of the range step of the Bellhop routine will then be generated. This procedure is described in more detail in DRDC CR2007-103.

    c. Spatial field shifting

        i. The inclusion of an option to handle spatial field shifting will be provided. This procedure, described in more detail in DRDC CR2007-102, involves computing spatial field shifts for the perturbed acoustic fields. Consider a particular range-depth point on the reference acoustic field, which is to be compared to the perturbed field. A two-dimensional window of pre-selected size is defined centred on this point. The correlation is then computed between the reference field (magnitude) over this window and the perturbed field over a series of windows of identical size, which are centred at a grid of points about the point of interest. The window-centre for the perturbed field that produces the highest correlation with the reference field is taken to define the range and depth shift resulting from the environmental perturbation. The shift-corrected sensitivity will be computed, showing that component of sensitivity that are less sensitive to source-receiver geometry changes.

2) Corrections to PASTET software deficiencies

    a. Bug fixes for the following issues to be addressed:

i. Some bad XBT data is being returned by the SSP survey (duplicate depth points with different temperatures); this will be investigated, and fixed if feasible.

ii. If the SSP plot window is open, and the operator changes the SSP averaging radio button setting, the SSP plot should automatically refresh to reflect the new selection. In the current PASTET release it is necessary to close the SSP plot window, and re-open it, to display the newly selected SSP averaging. The automatic refresh behaviour will be added to the program.

iii. The "# samples" GUI field is supposed to be set to "0" when the SSP GUI parameters are manually altered. Currently, it is being set to "1", which is misleading and incorrect. This will be corrected.

iv. If the map window is open with display of survey results locations selected, and the operator performs a new survey operation, the map should automatically refresh to reflect the new results when the survey operation completes. In the current PASTET release it is necessary to close the map window and re-open it, or perform any other mapping operation, such as zoom or pan, to display the new survey results locations. The automatic refresh behaviour will be added to the program

v. In testing the mapping of the survey results, for the default modelling location and both survey areas set to 2-degree squares, if the initial map with the three survey results locations displayed is zoomed-in by four clicks of the mouse thumbwheel, a *segmentation fault* is reported—even if ample time is allowed for the screen to refresh between the thumbwheel clicks. This will be investigated, and fixed if feasible.

vi. In testing the bathymetry survey results, it has been observed that bathymetry profiles are spaced in range at approximately 12 km between points. The underlying bathymetry grid is much denser than this (a few 100 m) so it begs the question as to why the bathymetry grid is being sampled so sparsely. This will be investigated and fixed if feasible, preferably in conjunction with task 1b.

vii. In testing the sediment properties survey results, for certain modelling locations and a large survey area (e.g., a 2-degree square), a number of sediment type results can be reported. However, several of these are duplicates, with the same latitude, longitude, and code value. This will be investigated and fixed if feasible.

viii. There is a bug in the text field GUI object. If the full contexts of the text field are highlighted (turned to white-on-black instead of black-on-white) and a character then typed on the keyboard, PASTET will abort with a *segmentation fault*. This will be investigated and resolved.

ix. Bellhop uses a `temp` directory to store its working files. This directory is created automatically the first time Bellhop is run after a new PASTET installation, if it doesn't already exist. However, the first Bellhop runs still fail for some reason. Subsequent runs are fine. This will be investigated and corrected.

x. A number of “print statements” remain embedded in the PASTET code, left over from various testing efforts. These will be removed or commented-out.

b. <u>Enhancements to be made to the PASTET program:</u>

i. Add the option “none” as one of the menu choices for each of the three survey data sources. When a survey is requested, it would not be performed for data sources selected as “none”; instead, these would preserve their GUI values, and those values would be used in the next run, along with the new survey results.

ii. In the present PASTET release, SSP extension of survey results is performed to a depth dependent on the deepest bathymetry mean+sigma survey return. In the case where the SSP survey is being performed, but the bathymetry is not being surveyed (GUI field parameters being retained), as described above, the SSP extension will be performed to a depth dependent on the GUI field parameters mean+sigma.

iii. The map will be enhanced with the capability to display a pop-up screen with the details of the sediment properties, SSP, or bathymetry when the operator left-clicks the pointer on any such corresponding symbol displayed on the map.

iv. The action of the keyboard “tab” and “shift-tab” keys for focus navigation will be implemented on the GUI screen. Currently these keys have no effect.

v. As a direct result of porting PASTET to C++, numerous compiler warnings are presently generated. These warnings will be rectified. Deprecation based warnings will be rectified first. Leaving these may result in compile failures with new C++ compiler releases. This will also greatly assist in future maintenance and enhancement efforts.

vi. The PASTET source files will be renamed to use the `cpp` suffix. This will enable easy C++ detection for various development environments.

The scope of the work is bounded by the $50K fixed budget for this call-up.[1]

2 Implementation

Following the issue of call-up 015 under the STB Maintenance standing offer [1] a kick-off meeting was held at DRDC-Atlantic on 2012-12-05. Present at this meeting were DRDC scientific staff Dr. Sean Pecknold (Scientific Authority), and JASCO (contractor) staff John Moloney and Terry J. Deveau. The logistics of the work were discussed, including the issues of obtaining the latest version of the existing source code for the PASTET program, and some of the details of the nonlinear sensitivity analysis algorithms.

2.1 Task 1 — Nonlinear Sensitivity Measures

An overview of the implementation that was undertaken on each of the assigned tasks is briefly presented in this section. Separate sections will follow, where warranted, with additional details and discussion of issues encountered.

a. Sound speed profiles

   The Scientific Authority provided the contractor with two programs written in IDL (Interactive Data Language) to serve as templates for implementing the EOF analysis of SSP input in C++. These two programs were very carefully examined and equivalent functions in the C++ programming language were constructed. The C++ implementation of the equivalent functions, in an object-oriented architecture, along with additional elements used in interaction with PASTET, can be seen in Annexes A through D.

   In the course of this development, a number of issues were encountered (see section 2.3.b), however they were all satisfactorily resolved. In addition, the PASTET graphical user interface (GUI) was extended to allow the operator to control the EOF option, new output plots were developed to visually portray major steps in the EOF analysis, which were essential during its development and which will remain a helpful tool for operations and training. Finally, the necessary program changes were made to allow the EOF analysis results to be used as the basis of a Monte Carlo procedure for multiple TL model runs, and the results assimilated in the form of upper and lower bounding envelopes around the TL run with mean inputs. The existing TL plot procedures and comma-separated values (CSV) output file routines were also updated to handle the Monte Carlo results from the EOF-based SSP analysis.

b. Bathymetry

   Due to the object-oriented software engineering approach now being used to advantage in PASTET programming, common functions and data elements are implemented through base classes in C++ and specialized elements and member functions are relegated to derived classes. In this way, source code elements that can be shared between partly-related requirements can be implemented simultaneously. This approach was used to advantage to progress the bathymetry power-spectrum modelling task in parallel with the EOF analysis task described above, since the implementation elements share some commonality, both in the GUI controls and in the internal structures and functions. In addition, work was begun on specific implementation of the power spectrum analysis once the EOF code had been initially completed. However, it soon became apparent that there were issues that needed to be revisited and addressed with the EOF code, and there would not be sufficient project resources to complete the bathymetry power spectrum task, so it was left in an incomplete state, to be resumed in future work.

c. Spatial field shifting

A small amount of work was accomplished on the spatial field shifting implementation, in parallel with some basic commonality of the object oriented programming used in the two tasks discussed above; but this is quite minor, and essentially all of this task has been left to be resumed in future work.

2.2 Task 2 — Corrections to PASTET software deficiencies

Although the primary focus of this call-up is to complete Task 1, and as things evolved, mainly Task 1.a, nevertheless a number of other corrections to PASTET software deficiencies were made in the course of events. This was due, in many cases, to the deficiencies being an impediment to progressing with development and testing of Task 1.a. These situations usually arise because the problem is interfering with some aspect of the implementation, operation, or validation of work on the main task.

In other cases, the opportunity to make the correction is due to the problem and its solution coming to the attention of the developer fortuitously, while working on the main task. In the course of deconstructing the program logic, either for the purpose of appropriately reusing existing elements or investigating irregularities in program operation, it frequently happens that programming errors are discovered. Our approach has been to attempt to correct these deficiencies, when and where they are found, where the effort required to do so is not too great and where doing so does not materially detract from the principle task. The motivation for doing so is that it is more efficient to pursue the solution with the logic deconstruction fresh in mind, rather than taking the time to adequately document the problem, for future pursuit of a solution, which would result in having to later repeat the same work just done.

The most notable of such corrections made during this work are detailed below.

a. Bug fixes for the following known issues were addressed

iii. The "# samples" GUI field is supposed to be set to "0" when the SSP GUI parameters are manually altered. Previously it was being set to "1", which is misleading and incorrect. This was corrected.

vii. In testing the sediment properties survey results, for certain modelling locations and a large survey area (e.g., a 2-degree square), a number of sediment type results can be reported. However, several of these were duplicates, with the same latitude, longitude, and code value. This was investigated and fixed.

viii. There is a bug in the text field GUI object. If the full contexts of the text field are highlighted (turned to white-on-black instead of black-on-white) and a character then typed on the keyboard, PASTET used to abort with a segmentation fault. This was investigated and partially resolved. Most of the time, now, nothing goes wrong, but certain keystrokes can still cause the segmentation fault. This will have to be looked at again in future work.

x. A number of “print statements” were embedded in the PASTET code, left over from various testing efforts. These have been removed. There are still some that are prefixed with “NOTICE:”, though, and these do not appear to originate in the C++ code of PASTET. Perhaps they are in a script file or some other ancillary program. This will have to be investigated further in future work.

b. <u>Enhancements that were made to the PASTET program</u>

vi. The PASTET source files have been renamed to use the cpp suffix. This will enable easy C++ detection for various development environments.

c. <u>Previously unidentified deficiencies now corrected</u>

i. The previous version of PASTET had a conditional MAKEFILE with a switch variable named STB_BUILD to allow either an STB version or a StandAlone version to be built. Recent DRDC work focussed on the STB version and broke the ability to build a StandAlone version. This was fixed as an Eclipse configuration option. It also required some changes to conditional compilation within the C++ source code.

ii. A number of code changes that were made during DRDC refactoring under Unix were causing a segment violation in the stand-alone windows/cygwin version due to stack overflow. These were analysed and corrected.

iii. Debugged code that was causing the stand-alone version to crash after a successful Bellhop run.

iv. Debugged code that caused the environmental data mapping display controls to fail in the stand-alone version. This was due to the functionality having been disabled during DRDC refactoring work.

v. Investigated and resolved problem where changing the location via the lat/long fields on the GUI was not correctly reflected in the map display.

vi. Corrected all standard deviation calculation code loops to use the same approach, one loop instead of two, and divide by (n-1). Also corrected some places where an integer divide was incorrectly taking place, instead of a floating divide.

vii. Eliminated some obsolete variables in the sensitivity job structure.

viii. Added a “uniform” SSP averaging option for better conformity with DRDC-supplied EOF algorithm. This required changes in many parts of the program.

ix. Added some improved labeling and info on profile plots.

x. Resolved some problems with GUI field checking in DRDC refactored code.

xi. Fixed a problem where number of samples was being set to zero even though the value in the text box was not actually changed by editing.

xii. Solved a problem with DRDC refactored code that was exponentially repeating the same status messages.

xiii. Solved some problems pertaining to handling the condition where SSPs and sediment data were not found in a survey.

xiv. Fixed a problem where errors in sediment thickness survey would zero-out valid sediment type survey results, and vice-versa.

xv. Debugged and corrected problems with sampling-mode SSP averaging and min/max selection. The normalization was changed from per point to a depth integration to avoid overweighting very short SSPs with a high deviation from the mean. Also, corrected a sign error in the distance calculation.

xvi. Resequenced the GUI controls and output variables in a more user-friendly manner, using an analogy to the real world, with water above and sea floor below, etc.

xvii. Fixed a problem where sample-based averaging min/max profiles were not being sent to Bellhop (this bug was due to the DRDC refactoring changes).

xviii. Fixed a problem with skipped SSP storage deallocation causing a memory leak.

xix. Fixed a problem in the way that SSP sigma was being set when too few sound speed values are present at a given depth.

xx. Fixed a bug where Bellhop runs would fail due to the SSPs not extending as deep as the bathymetry (this was due to DRDC refactoring changes).

2.3 Implementation Issues

The subsections below provide additional details about implementation issues that were encountered in the course of the work pertaining to call-up 015.

a. Changes pertaining to DRDC refactoring of the PASTET source code

Between the dates of approximately 1 April 2012 to and 6 June 2012 software development on PASTET was performed at DRDC resulting in a "refactored" version of the PASTET source code.[4] This work resulted in a number of significant changes to the source code, including the following:

- Much of the source code was refactored into a higher degree of object oriented programming compliance, with more extensive use of C++ classes.

- The previous version of the source code included two build options: one for a “stand-alone” version and one for a “STB interface” version. This work improved the STB interface version but removed the option for a stand-alone version.

- The previous version of the source code included provisions for building the executable under both the Microsoft Windows/Cygwin and the Linux/Unix development environments. This work removed the provision for the Microsoft Windows/Cygwin executable build.

In order to proceed with the tasks of call-up 015, it was necessary for the contractor to first address some of the issues arising from the previous work on the source code outlined above. In particular, it was necessary to implement provisions for building both a “Stand Alone” version and a “STB Interface” version within the Eclipse development environment; and to make changes to the Eclipse project settings to allow for building the executable both under Microsoft Windows/Cygwin and Linux/Unix. In addition, some of the pre-existing PASTET software functionality was not fully operational anymore, following the refactoring work, and these deficiencies needed to be addressed.

All these issues were resolved by the contrator and work then resumed on the implementation of the EOF modeling SSP data (Task 1.a). This effort was a necessary prerequisite to the planned work, but was not itself foreseen in the plan. Consequently, less progress than was anticipated was made on Tasks 1.b and 1.c during the course of the call-up.

b. Issues related to the IDL template source code for EOF analysis of SSPs

As part of the implementation of Task 1.a, one of the sub-tasks is to implement in the PASTET C++ source code the functionality for the EOF analysis that is exemplified in two IDL-language programs supplied by Sean Pecknold: **`ssp_svd_temp.pro`** and **`eof_analysis.pro`**. In the course of this work, an extremely thorough logic deconstruction of these two programs was necessary, and it turned-up a number of apparent logic errors. Comments are included in the C++ source code to note where the logic implemented by the contractor differed from the IDL original, along with an explanation of how to change it back for testing purposes. The most significant instances where such discrepancies were noted are detailed below.

i. In program **`ssp_svd_temp.pro`**, line 57, the instruction — **`speed_at_zero=spd_mean_vec1[0]`** — should come after the **`endif`** statement, not before, as this leaves the **`speed_at_zero`** variable uninitialized when the **`if`** statement block is not executed, as is usually the case. The logic requires it to have the assigned value.

ii. In program **`ssp_svd_temp.pro`**, lines 93-94, the instructions — **`delta_speed_at_zero=speed_at_zero-spd_vec1[0]`**

`spd_vec1=spd_vec1+delta_speed_at_zero` — are causing a problem. Their presence appears to be detrimental to the correct functioning of the algorithm. Perhaps they originated in programing experimentation at some time and were inadvertently left behind. These statements make it impossible to generate any eigenvectors that embody the real variation in near-surface sound speed that is exemplified by the sample set of sound speed profiles. They do partially provide a somewhat hidden function, though, in that they supply a needed rectification for the extended tail of the short SSPs. Rather than removing these two statements, they can be rehabilitated if the statement discussed in point (i) above is changed to **`speed_at_zero=spd_vec1[0]`** and moved to follow line 76.

iii. The IDL code calculated the number of EOFs to use by counting how many it took to accumulate a hard-coded threshold in the eigenvalues sum in lines 116-117 of program **`ssp_svd_temp.pro`**, but then it ignored that count and used a hard-coded number (4). In the C++ implementation, the hard-coded number is replaced by a class member variable (**`n_eofs`**) that defaults to a fixed number, but can be specified either at object instantiation or afterwards. Also, the threshold idea is retained, with a default likewise adjustable, and which will over-ride **`n_eofs`** if exceeding the threshold requires a higher count.

iv. The IDL code in program **`eof_analysis.pro`** zeroes out the lowest-order eigenvector in line 114. There are some minor details in subsequent code lines that support this measure. It was unclear why this was done in the IDL code, and the C++ implementation does not replicate it. The lowest-order eigenvector is allowed to stand as is.

v. There appears to be an error in line 136 of program **`eof_analysis.pro`** — **`if norm( eofs_small[i,*], /double) gt 0 then begin`** — as the L2 norm of each *column* is tested to be non-zero, but the L2 norm of the corresponding *row* is actually used as a divisor in the next line. It is likely that the array reference in this line should actually be — **`eofs_small[*,i]`**.

vi. The presently implemented EOF-based approach to generating SSP variations is incomplete. The IDL code in program **`ssp_svd_temp.pro`** uses random Gaussian deviates, with $\sigma^2 = 1.0$, to weight the sqrt(eigenvalue) × eigenvector summation, which is then added to the mean SSP. The problem is that the original SSP set does not represent all ranges of eigenvalue weightings, but only a restricted subset. In particular, the randomly positive and negative variations in the Gaussian deviates will cause a strong eigenvector to appear in its mirror-reversed form in some of the synthesized SSPs. For example, Figure 14 shows the SSP variations that result from using a uniform random deviate with the EOFs shown in Figure 13, whereas Figure 1 shows the results in the identical run using Gaussian deviates.

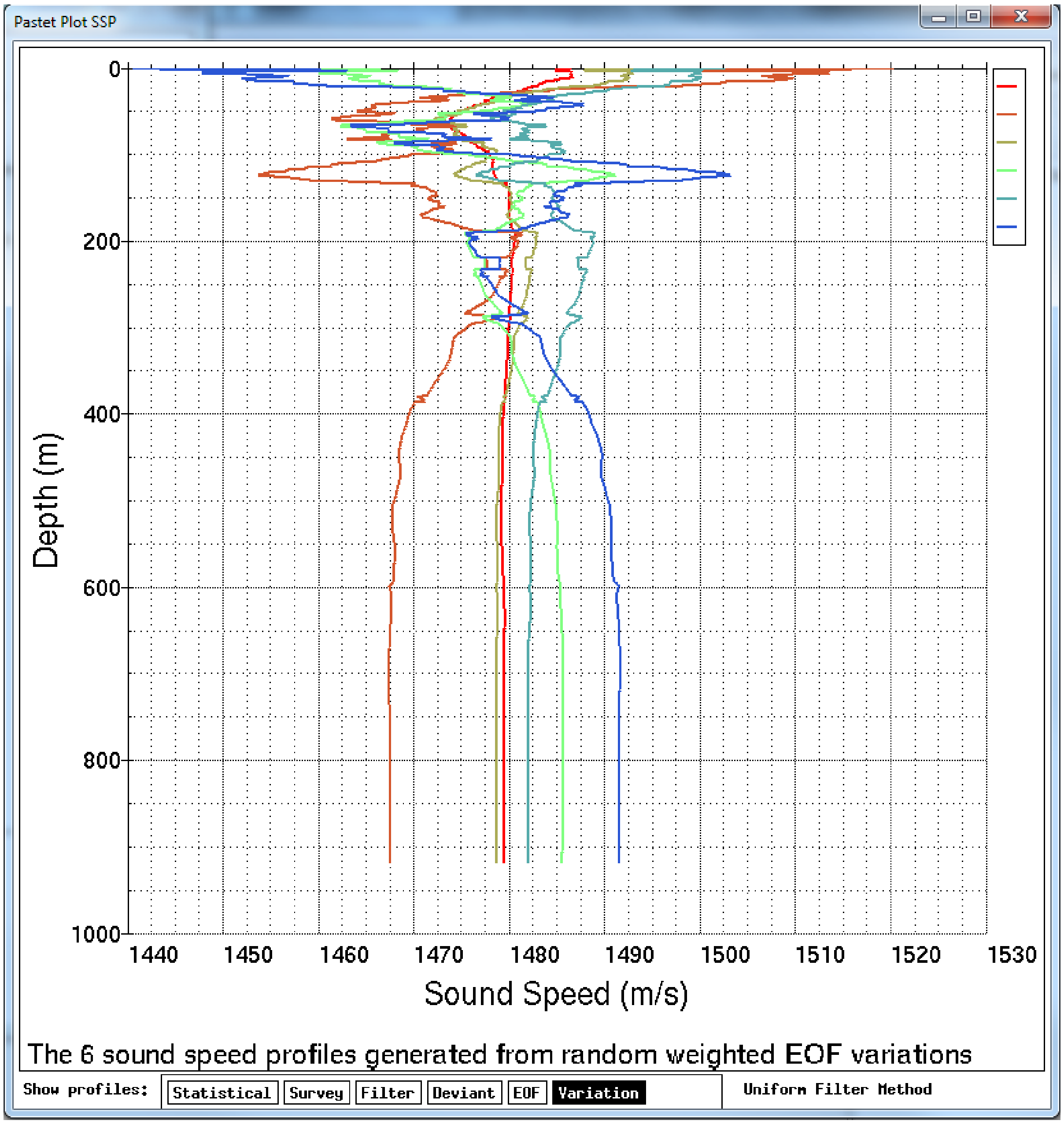


Figure 1. An example of SSP variations generated with Gaussian random deviates applied to the EOFs shown in Figure 12.

In the preferred form of the C++ implementation, the Gaussian deviates are replaced with uniformly positive random deviates. This does not fully solve the problem, however, because the real SSP may in fact use a negative weight on some of these eigenvectors, so the positive weight may be the one that generates the mirror reversed version. But at least both normal and mirror-reversed versions of a strong eigenvector will not simultaneously be represented in the synthesized SSP variations. More work is required to base the EOF weightings on the actual range of eigenvalue weightings that are found in the original SSPs.

c. Problems encountered while implementing an eigenvector solver

The heart and core of the Task 1.a implementation is the calculation of the EOF eigenfunctions. As mentioned above, DRDC supplied the contractor with IDL code to use as a guide for the C++ implementation in this call-up. However the IDL code as supplied is not complete; it depends on an internal IDL function — **`EIGENQL`** — which was not supplied. A copy of the IDL source code for this function was located, though, without too much difficulty. But it depends in turn on two more primitive IDL functions — **`TRIRED`** and **`TRIQL`** — which were harder to locate. Having followed this course, the contractor came to see continuing in this vein would not be the best approach, as source code conversion from IDL to C++ can be quite tricky, especially where intricate matrix math is involved, as is the case here.

Since eigenvalue and eigenvector calculation is a relatively generic mathematical toolbox function, the contractor sought out an existing C/C++ freeware solution. After some initial investigation, it appeared that the ARPACK library would be a good choice, as it comes supplied with the Linux and Cygwin distributions. Also, not included in those distributions, but freely available in C++ source code download, is ARPACK++, which is a C++ wrapper application to the ARPACK library (which is actually a Fortran subroutine library).

In trying to compile ARPACK++ in the development environment, however, a large number of issues were encountered. Some appeared to be related to the gcc compiler conflicts with the ARPACK++ source code conventions, some appeared to be conflicts with the Eclipse code development environment, and some appeared to be a requirement for access to additional libraries, SUPERLU for one.

Considerable time was spent trying to work through all these conflicts in order to obtain a working eigenvector solver under C++, but despite the amount of effort expended, the contractor eventually had to abandon the approach of using ARPACK and ARPACK++ due to it being unworkable in this development environment, with the amount of time that could be devoted to getting it running, without overly compromising other project objectives.

So after reluctantly discarding the effort already invested, and some further searching, the contractor found that the *Numerical Recipes for C++* [5] has some suitable source code that is adequate to the needs of this project, and that is freely available. The main source code files are **`tred2.cpp`** and **`tridag.cpp`**, along with a few other minor ones that they call in turn.

So in the end the problem was solved, but a lot more time was spent on it than was anticipated, due to what initially seemed like the best and most obvious approach to the solution, eventually turning out to be unworkable and expensive; whereas an obscure, seemingly unsophisticated, and apparently anachronistic approach ultimately turned out to be efficient and very suitable.

2.4 Known Software Deficiencies

To avoid having information about the outstanding unresolved software deficiencies scattered in different sections of this report, they are all collected together in this section. This means there is some duplication with information provided elsewhere in this report. These outstanding deficiencies are also renumbered here, so previous numbering that may have applied is not reproduced in this section.

1. The EOF-based generation of SSP variations should be further developed, addressing the aspect of the random weightings of the EOFs being restricted to the range appropriate for the subset of weighting ranges that reproduce the original SSP input set.

2. The bathymetry power spectrum variations model that has been started should be completed.

3. The spatial field shifting option that has been started should be completed.

4. Some bad XBT data is being returned by the SSP survey (duplicate depth points with different temperatures); this should be investigated, and fixed if feasible.

5. It appears that the start/stop date selection option for the SSP survey may not be operating properly. Separately selecting each of the twelve months yielded a different total number of returned SSPs than selecting the whole year. Also, selecting the first 15 days of a given month, then separately selecting the rest of the month, did not always give the same total as for the whole month. This requires further investigation.

6. If the SSP plot window is open, and the operator changes the SSP averaging radio button setting, the SSP plot should automatically refresh to reflect the new selection. In the current PASTET release it is necessary to close the SSP plot window, and re-open it, to display the newly selected SSP averaging. The automatic refresh behaviour should be added to the program.

7. If the map window is open with display of survey results locations selected, and the operator performs a new survey operation, the map should automatically refresh to reflect the new results when the survey operation completes. In the current PASTET release it is necessary to close the map window and re-open it, or perform any other mapping operation, such as zoom or pan, to display the new survey results locations. The automatic refresh behaviour should be added to the program

8. In testing the mapping of the survey results, for the default modelling location and both survey areas set to 2-degree squares, if the initial map with the three survey results locations displayed is zoomed-in by four clicks of the mouse thumbwheel, a segmentation fault is reported—even if ample time is allowed for the screen to refresh between the thumbwheel clicks. This should be investigated, and fixed if feasible.

9. In testing the bathymetry survey results, it has been observed that bathymetry profiles are spaced in range at approximately 12 km between points. The underlying bathymetry grid is much denser than this (a few 100 m) so it begs the question as to why the bathymetry grid is being sampled so sparsely. This should be investigated and fixed if feasible.

10. There is a bug in the text field GUI object. If the full contexts of the text field are highlighted (turned to white-on-black instead of black-on-white) and a character then typed on the keyboard, PASTET will normally ignore the selected highlight, but in some cases (certain keystrokes) it will abort with a segmentation fault. This should be investigated and resolved.

11. Bellhop uses a temp directory to store its working files. This directory is created automatically the first time Bellhop is run after a new PASTET installation, if it doesn't already exist. However, the first Bellhop runs still fail for some reason. Subsequent runs are fine. This should be investigated and corrected.

12. A number of "print statements" remain embedded in the some ancillary program or script that is called by PASTET during a survey operation. These status messages are prefixed with "NOTICE:". They may be left over from previous testing efforts. These should be removed.

13. The option "none" should be added as one of the menu choices for each of the three survey data sources. When a survey is requested, it would not be performed for data sources selected as "none"; instead, these would preserve their GUI values, and those values would be used in the next run, along with the new survey results.

14. In the present PASTET release, SSP extension of survey results is performed to a depth dependent on the deepest bathymetry mean+sigma survey return. In the case where the SSP survey is being performed, but the bathymetry is not being surveyed (GUI field parameters being retained), as described above, the SSP extension should be performed to a depth dependent on the GUI field parameters mean+sigma.

15. The map should be enhanced with the capability to display a pop-up screen with the details of the sediment properties, SSP, or bathymetry when the operator left-clicks the pointer on any such corresponding symbol displayed on the map.

16. The action of the keyboard "tab" and "shift-tab" keys for focus navigation should be implemented on the GUI screen. Currently these keys have no effect.

17. As a direct result of porting PASTET to C++, numerous compiler warnings are presently generated. These warnings will be rectified. Deprecation based warnings should be rectified first. Leaving these may result in compile failures with new C++ compiler releases. This will also greatly assist in future maintenance and enhancement efforts.

18. In testing the SSP survey operations, there were a few rare combinations of parameters that caused the survey to hang, and never return. One example was near Emerald Basin, with 2-degree square area, and the READB/TSD data source. This issue requires investigation.

19. In testing the bathymetry survey results, three of the four available bathymetry data sources appear to return depth values that are consistent with each other, however the fourth one (ETOPO5 from READB) appears to return bathymetry results that are drastically out of line with the other three sources. This requires further investigation.

20. When the operator alters GUI settings that directly impact the applicability of the current run results, such as the coherency option, the frequency, and the source depth, any existing run results that are currently offered for operator selection should be erased, as they would not conform anymore to the indicated GUI settings.

21. The intent of the status message screen was originally so that the operator could see the progression of the survey or the propagation runs in the ongoing status messages. However, as it is presently implemented, the status screen is only updated once, at the termination of the survey or run operation. The status display should be implemented differently so that ongoing status messages are seen while the operation is progressing.

22. The status message screen is a fixed-size screen. Once it is full, further messages cannot be seen at all. There is a “clear” button that the operator can use to erase the message screen. After clicking on it, new messages will now appear again at the top of the fresh screen, but the old messages are lost. The addition of a scroll bar to this message window would allow the operator to scroll back and look at old messages that had scrolled off the top of the screen. New messages would always appear at the bottom of the screen and would not be lost. A scroll bar should be added to the status message screen.

23. It appears that some of the refactoring code changes may have altered the way that environmental parameters, and their statistics, are being written to the Bellhop input files; in such a way that the actual Bellhop runs may not be using the environmental values implied by the GUI settings. This possible issue was noticed near the end of the present call-up and there was no time left to investigate it. This issue should be investigated and fixed, if necessary.

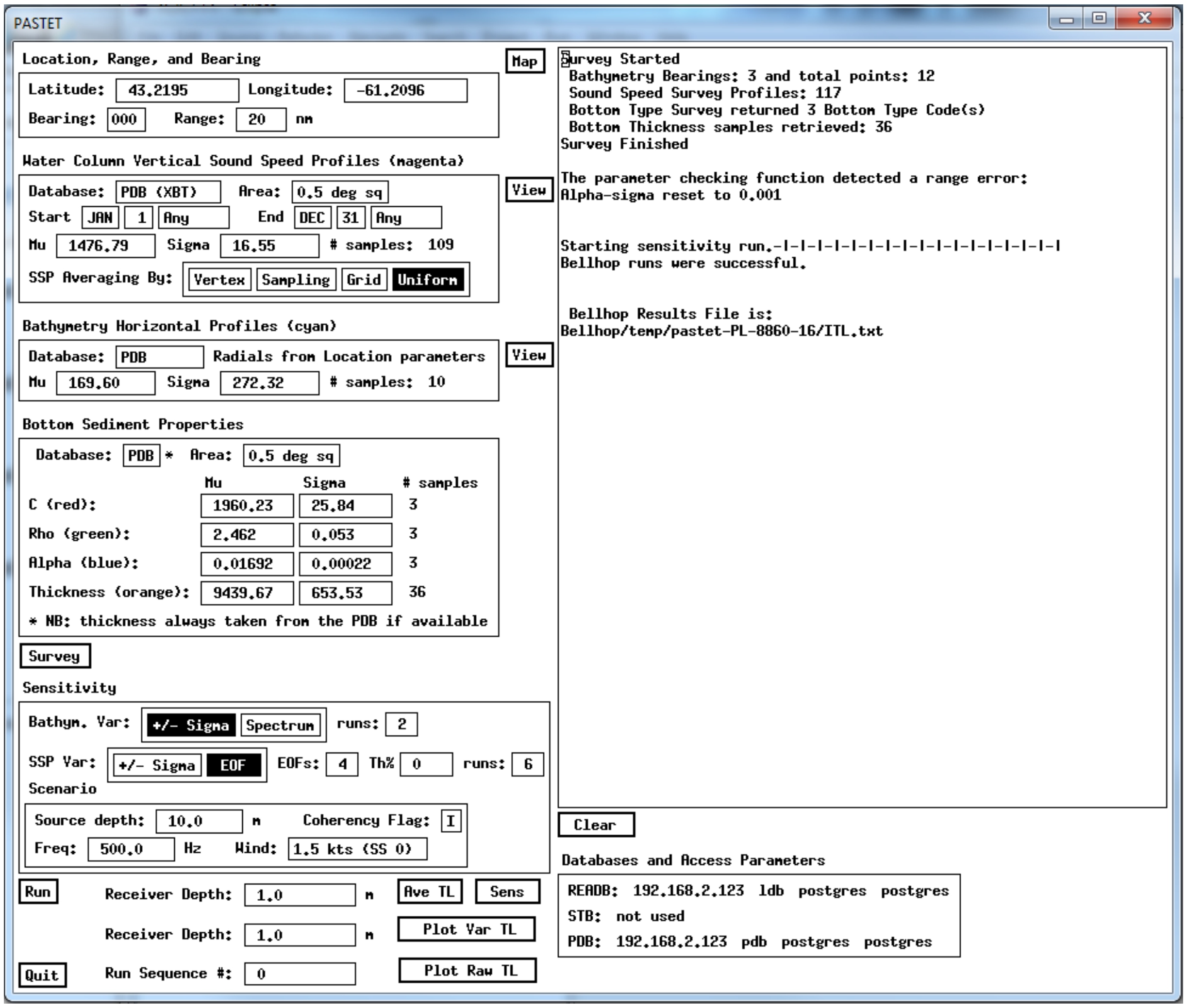
24. The legends on some of the present plots are not satisfactory. This applies in particular to the new EOF-related profile plots. Due to the time required to complete the EOF computational implementation, there wasn’t enough time to properly address some of the plot legend issues. The plot legends should be corrected if necessary, or removed if not appropriate.

25. Internal structures for storing copies of various sets of SSPs currently replicate both depth and sound speed vectors, in many cases, where the depth vectors are exact copies. These structures should be improved to avoid wasting memory on multiple copies of identical depth vectors.

3 Results

3.1 Revised PASTET GUI

The revised control page for the PASTET GUI is shown in Figure 2. This view depicts the state of the GUI after the `Survey` and `Run` operations have completed. The four `Plot` buttons at the bottom centre of the page, along with their modifier menus on their left, are only present after the `Run` operation is complete.



Location, Range, and Bearing
Latitude: 43.2195 Longitude: -61.2096
Bearing: 000 Range: 20 nm

Water Column Vertical Sound Speed Profiles (magenta)
Database: PDB (XBT) Area: 0.5 deg sq
Start JAN 1 Any End DEC 31 Any
Mu 1476.79 Sigma 16.55 # samples: 109
SSP Averaging By: Vertex Sampling Grid [✓ Uniform]

Bathymetry Horizontal Profiles (cyan)
Database: PDB Radials from Location parameters
Mu 169.60 Sigma 272.32 # samples: 10

Bottom Sediment Properties
Database: PDB * Area: 0.5 deg sq

	Mu	Sigma	# samples
C (red):	1960.23	25.84	3
Rho (green):	2.462	0.053	3
Alpha (blue):	0.01692	0.00022	3
Thickness (orange):	9439.67	653.53	36

* NB: thickness always taken from the PDB if available

Sensitivity
Bathym. Var: [✓ +/- Sigma] Spectrum runs: 2
SSP Var: +/- Sigma [✓ EOF] EOFs: 4 Th% 0 runs: 6
Scenario
Source depth: 10.0 m Coherency Flag: I
Freq: 500.0 Hz Wind: 1.5 kts (SS 0)

Receiver Depth: 1.0 m
Receiver Depth: 1.0 m
Run Sequence #: 0

```
Survey Started
 Bathymetry Bearings: 3 and total points: 12
 Sound Speed Survey Profiles: 117
 Bottom Type Survey returned 3 Bottom Type Code(s)
 Bottom Thickness samples retrieved: 36
Survey Finished

The parameter checking function detected a range error:
Alpha-sigma reset to 0.001


Starting sensitivity run.-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
Bellhop runs were successful.


 Bellhop Results File is:
Bellhop/temp/pastet-PL-8860-16/ITL.txt
```

Databases and Access Parameters
READB: 192.168.2.123 ldb postgres postgres
STB: not used
PDB: 192.168.2.123 pdb postgres postgres

Figure 2. New control page for the PASTET GUI.

The normal operational sequence is for the operator to first specify a latitude-longitude point of origin and LOB in the `Location` box (upper left). The latitude-longitude can either be hand-entered in the corresponding text fields, or graphically selected using the `Map` button (described below). The

Bearing and Range are free-format text fields that can be entered and edited using the pointer and the keyboard.

Next, the operator may wish to specify the Sound Speed box Database and Area controls in preparation for the Survey. These determine which database the Survey will query for the sound speed data, and what size geographic square centred on the origin will be considered in the search. The default values will often be correct, and if a selection is made, it remains in effect until the program is shut down (or until another selection is made). The same comments apply to the Database menu controls in the Bathymetry and Sediment boxes. The Sound Speed box also has a start/end date control. These controls allow the operator considerable flexibility in filtering the SSPs that will be included in the averaging calculations.

The Bathymetry box doesn't have an Area control. That's because the bathymetry search area is controlled by the LOB bearing and range. Three (or more) bathymetry profiles are taken, one along the specified LOB, and additional ones along radials on either side of it (the radial spacing is database dependent, and more radials are used for longer ranges). The Sediment box does have an Area control.

Once the filter controls described above are set to the operator's satisfaction, the next step is to press the Survey button. PASTET then submits a set of queries to the selected databases, calculates the means (Mu) and standard deviations (Sigma) for each of the parameters, and updates the GUI text fields with the results.

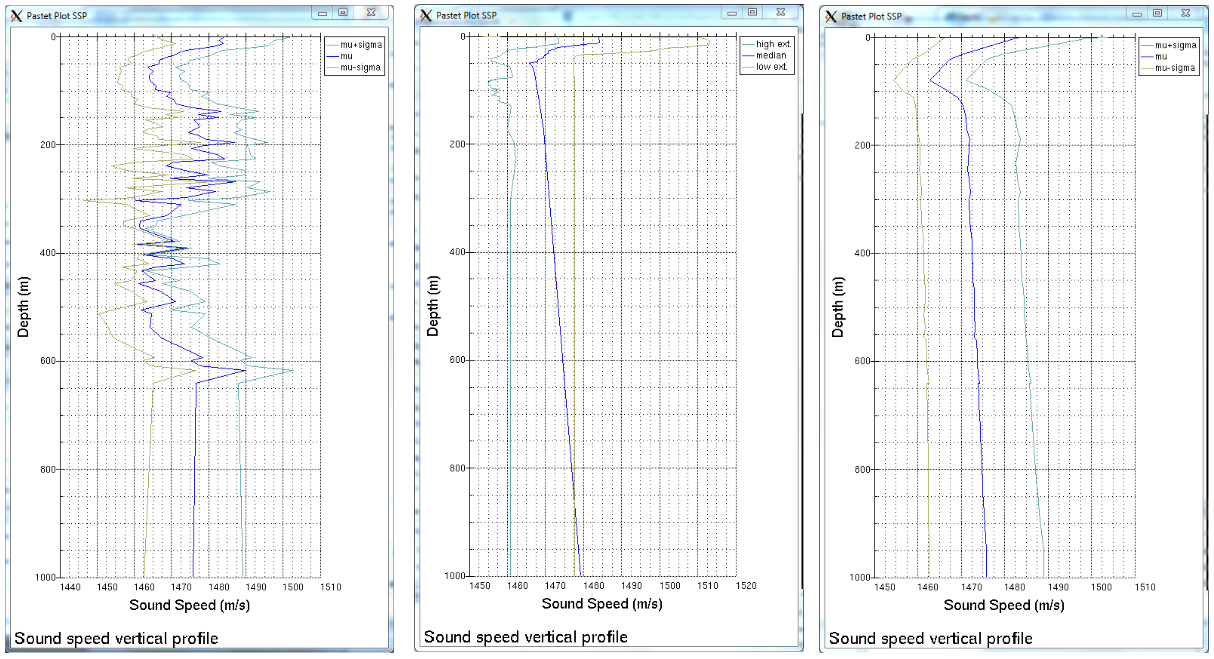


Figure 3. Examples of the average SSP plots that are displayed using the View *button, for three of the SSP averaging options:* Vertex, Sample, *and* Grid, *respectively.*

In the SSP case, an averaging method selection is also available: (a) Vertex-based SSP averaging, (b) a Sampling approach where the median profile is selected along with the two most extreme examples higher and lower than it in sound speed, (c) a `Grid`-based averaging method where all the SSPs are interpolated on to the same regular, but adjusted, depth spacing before averaging, and (d) a `Uniform`-based averaging method that is new with this release of the PASTET software, which is similar to the `Grid` approach, but which uses a strict 1 m spacing.

The `View` button can be clicked to pop-up a window showing the result of the SSP averaging. Figure 3 shows an example of three different averaging methods on the same survey results. The averaging method selection can be changed before or after the `Survey` operation; it doesn't affect the operation of the `Survey`, only how the `Survey` results are subsequently processed.

While the survey is in progress, informational messages are shown in the log box in the upper right half of the GUI window. The `Clear` button can be used to erase the contents of the log box if desired. In addition to the log box message "`survey complete`", the operator is alerted that the survey is finished by the `Survey` button changing colour from black to white.

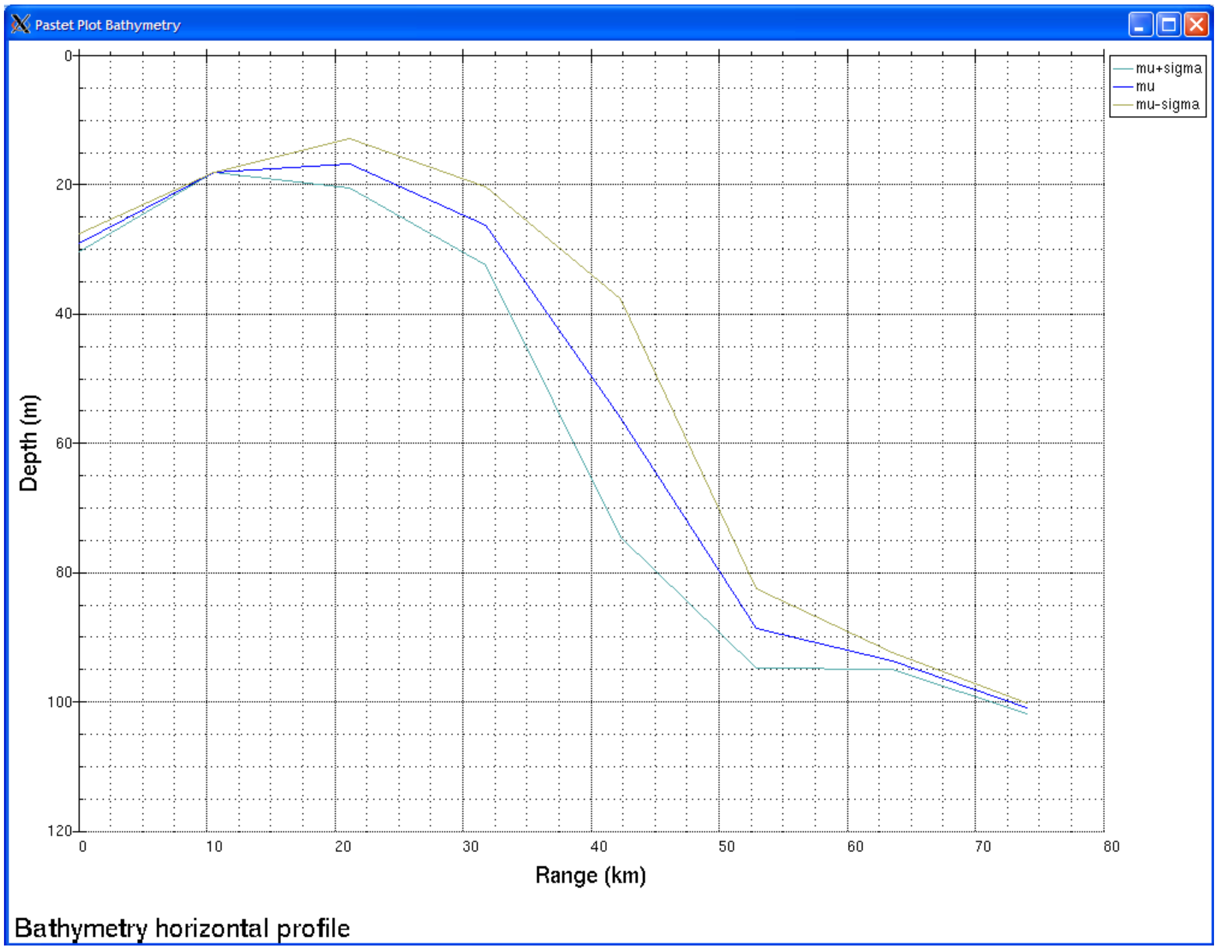


Figure 4. Example of `View` *plot of an internal PASTET average bathymetry profile, with plus sigma and minus sigma variations.*

Although the PASTET GUI only shows one overall `Mu` and `Sigma` for each of the bathymetry and sound speed profiles, for simplicity of display, it actually has an individual `Mu` and `Sigma` for each point along the profile, internally. These internal `Mu` and `Sigma` profiles can be shown by clicking on the corresponding `View` button. Examples of a PASTET `View` of the internal `Mu` and `Sigma` profiles for sound speed are shown in Figure 3, and bathymetry in Figure 4.

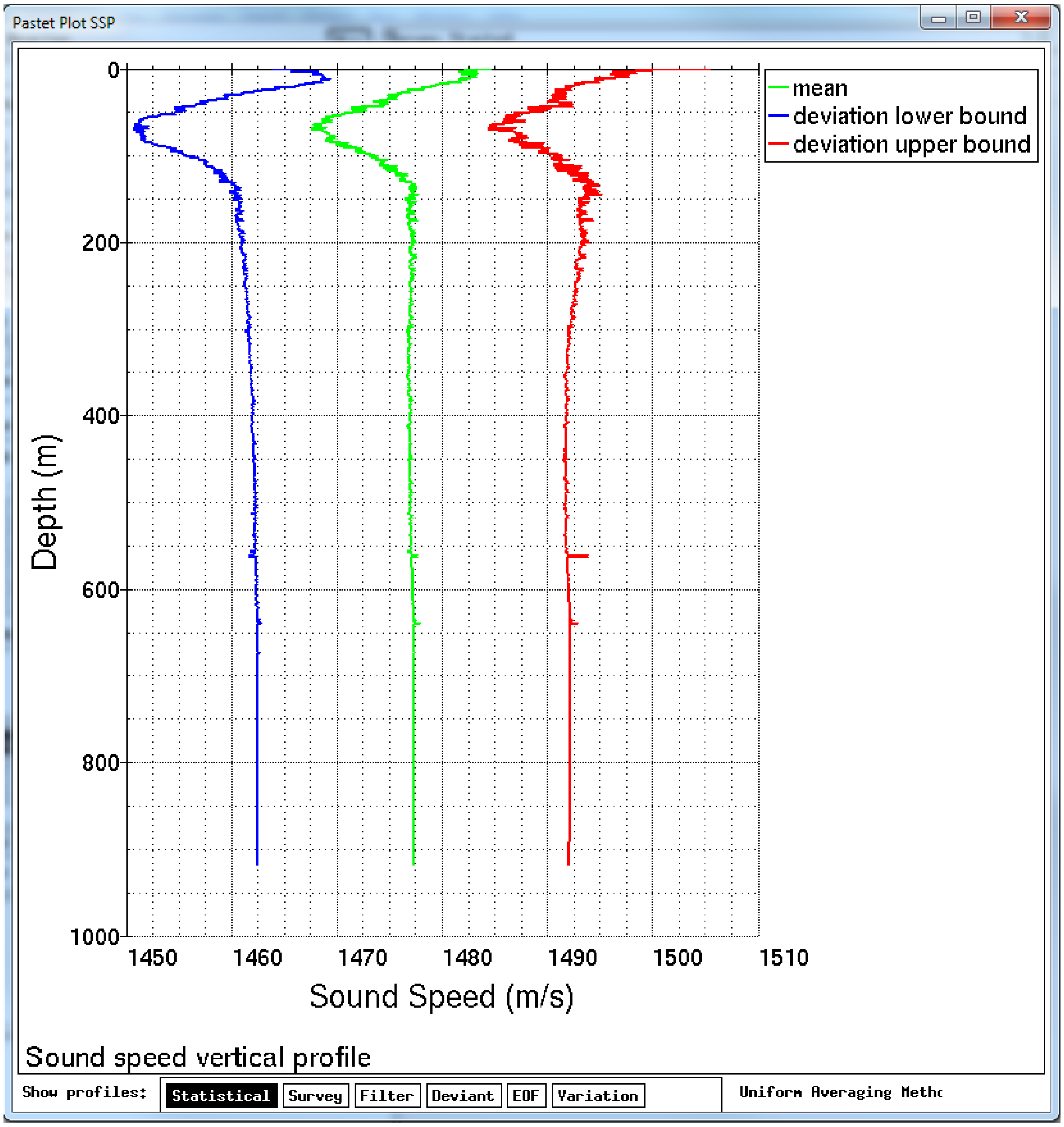
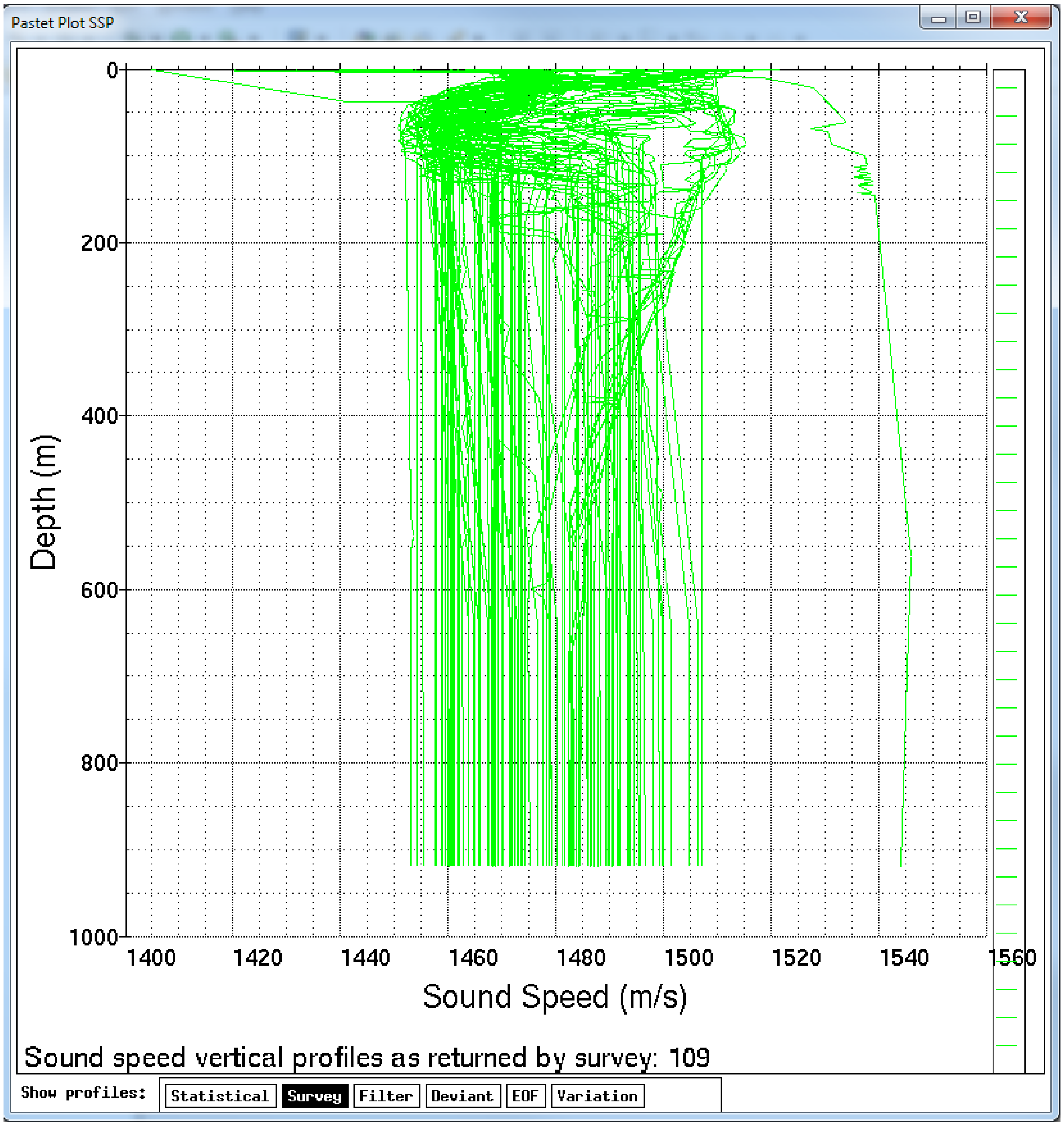


Figure 5. Example of "Statistical" sound speed profile plot.

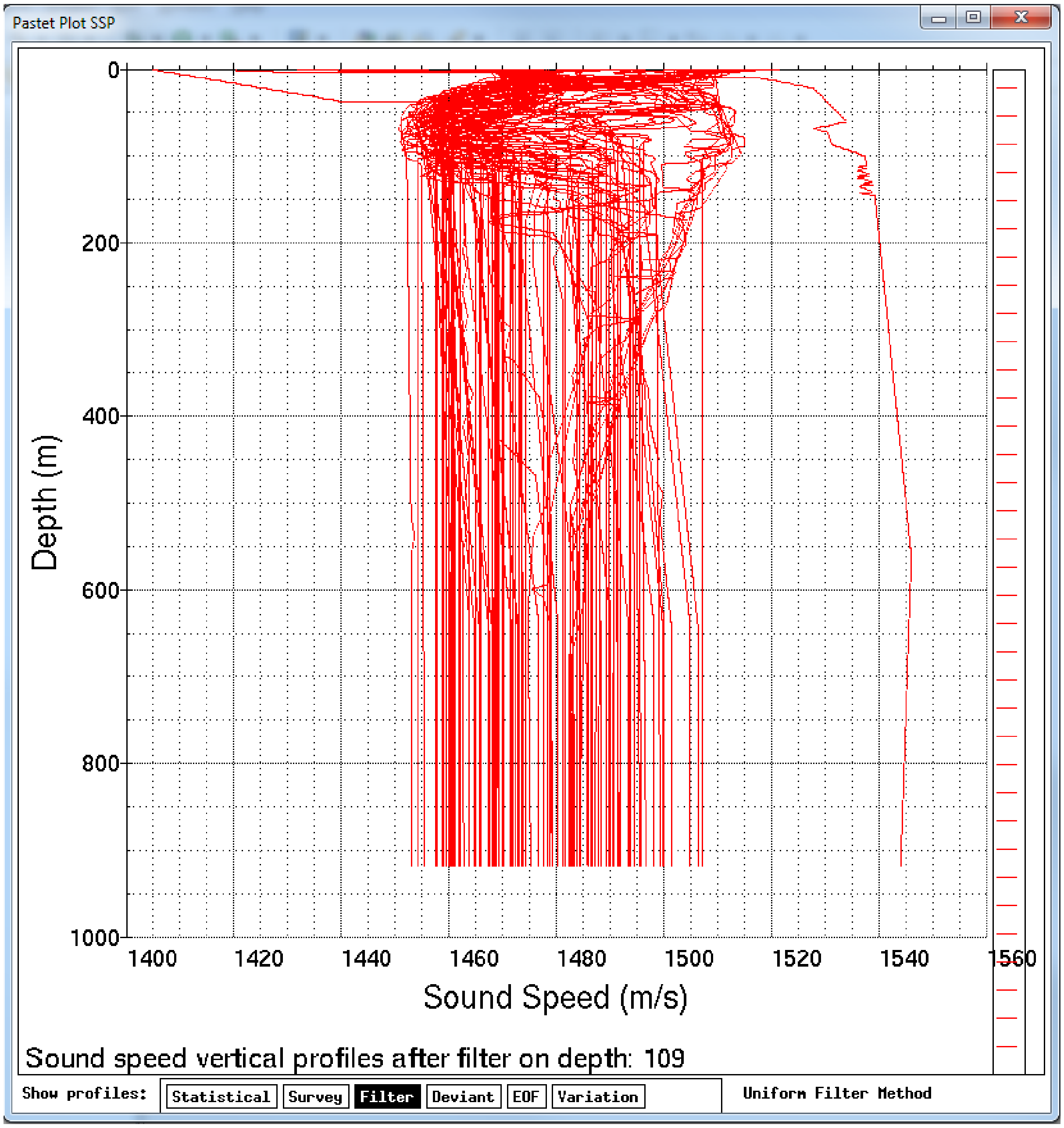
With the work of the present call up, several new options are available on the SSP "View" plot window. The "Statistical" option is the mean and standard deviation (mu and sigma) plot described above, which was the only SSP plot available in the previous version of PASTET. An example of it, showing the new plot selection option buttons, can be seen in Figure 5. Now there is also the "Survey" option, which plots all of the SSP profiles that were returned from the latest `Survey` operation. Naturally, there will be a lot of overlap, and individual SSPs will be difficult to trace, but the purpose of the plot is just to give the operator a general sense of the range of variability in the survey results. An example of a plot of the "Survey" option is shown in Figure 6.



Sound speed vertical profiles as returned by survey: 109

Figure 6. Example of "Survey" plot of the set of sound speed profiles returned by a survey.

The “Filter” option on the SSP plot window is similar to the “Survey” option, however, these profiles have been filtered by the selected SSP averaging option. In most cases the SSPs will have been re-interpolated on to a common depth grid. An example of a plot of “Filter” SSP data is shown in Figure 7. There are additional SSP plot options, but these will be discussed later when the new EOF analysis facility is presented.



Sound speed vertical profiles after filter on depth: 109

Figure 7. Example of “Filter” plot of sound speed profiles.

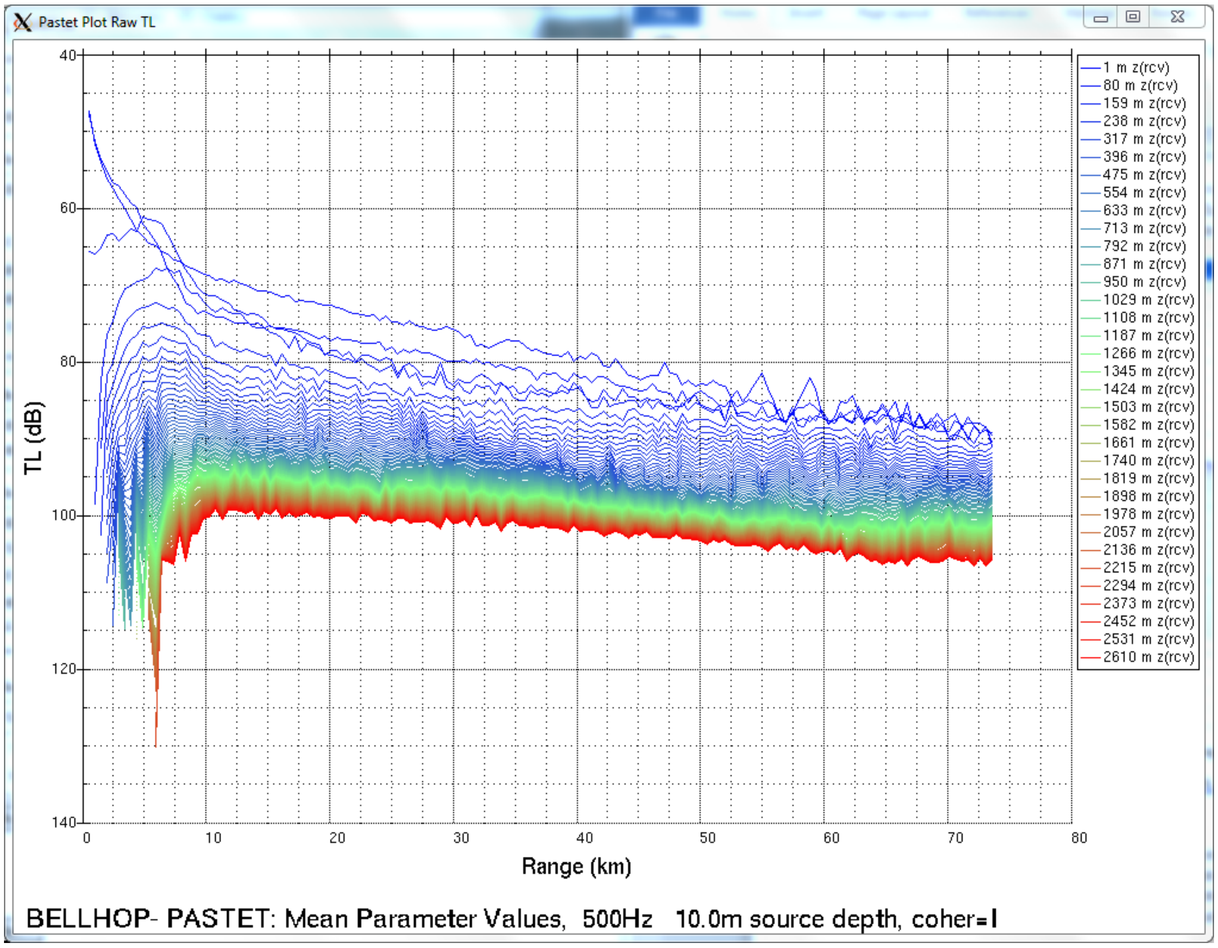
Following the completion of the `Survey` operation, the operator would typically click on the `Run` button to begin the TL and sensitivity calculations. The operator does have the ability, however, to manually adjust any of the parameters prior to starting the `Run`. In fact, the operator can use entirely hand-entered data, if desired, and start the `Run` without having done a `Survey` at all. However, there is no way to hand-enter complex bathymetry profiles and sound speed profiles. If the operator edits these GUI fields, the nominal hand-entered profile will be used instead of the survey-derived profile (if any). In the case of bathymetry, the hand entered profile consists of the same depth (`Mu`) at every range point (plus and minus the same `Sigma`), and the hand-entered SSP consists of the same isospeed (`Mu`) at every depth (plus and minus the same `Sigma`). If the operator does edit these fields, and the hand-entered data is thereby selected, the operator is alerted that this has happened by the corresponding "`# samples`" figure being reset to 0. If it was done by mistake the operator can correct it by re-doing the `Survey`.

Prior to selecting `Run`, the operator can also specify the source depth, frequency, wind speed, and coherency option in the `Scenario` box. Each of these settings will affect the TL and sensitivity results, but PASTET is not presently set up to calculate sensitivities for these parameters.

Once `Run` is clicked, PASTET begins to sequentially call the Bellhop model to calculate TL versus range and receiver depth for the mean parameter case, and individually plus and minus sigma for each of six parameters (thirteen runs in total). Additional progress messages are written to the log box during this time. In addition to the log box message "`Completed sensitivity run`", the operator is alerted that the run is finished by the `Run` button changing colour from black to white and the appearance of four `Plot` buttons and related controls to the right of the `Run` button (see Figure 2).

3.2 TL Output Products

In addition to the bathymetry and SSP profile `View` output products described above (see Figure 3 and Figure 4), there are three types of TL plots available. The `Average TL` and `Variable TL` plots each include a receiver depth selection menu. A file selection menu is provided for the `Raw TL` plot—the "`Run Sequence #`" menu allows the operator to specify which of the 13 TL files from the latest run to be selected for plotting. A concise description of the particular file selected is printed along the bottom edge of the plot itself (see Figure 8), if the selection by sequence number is a bit cryptic. The sequence number selection control has the advantage of ensuring an exact correspondence with the file name on disk.

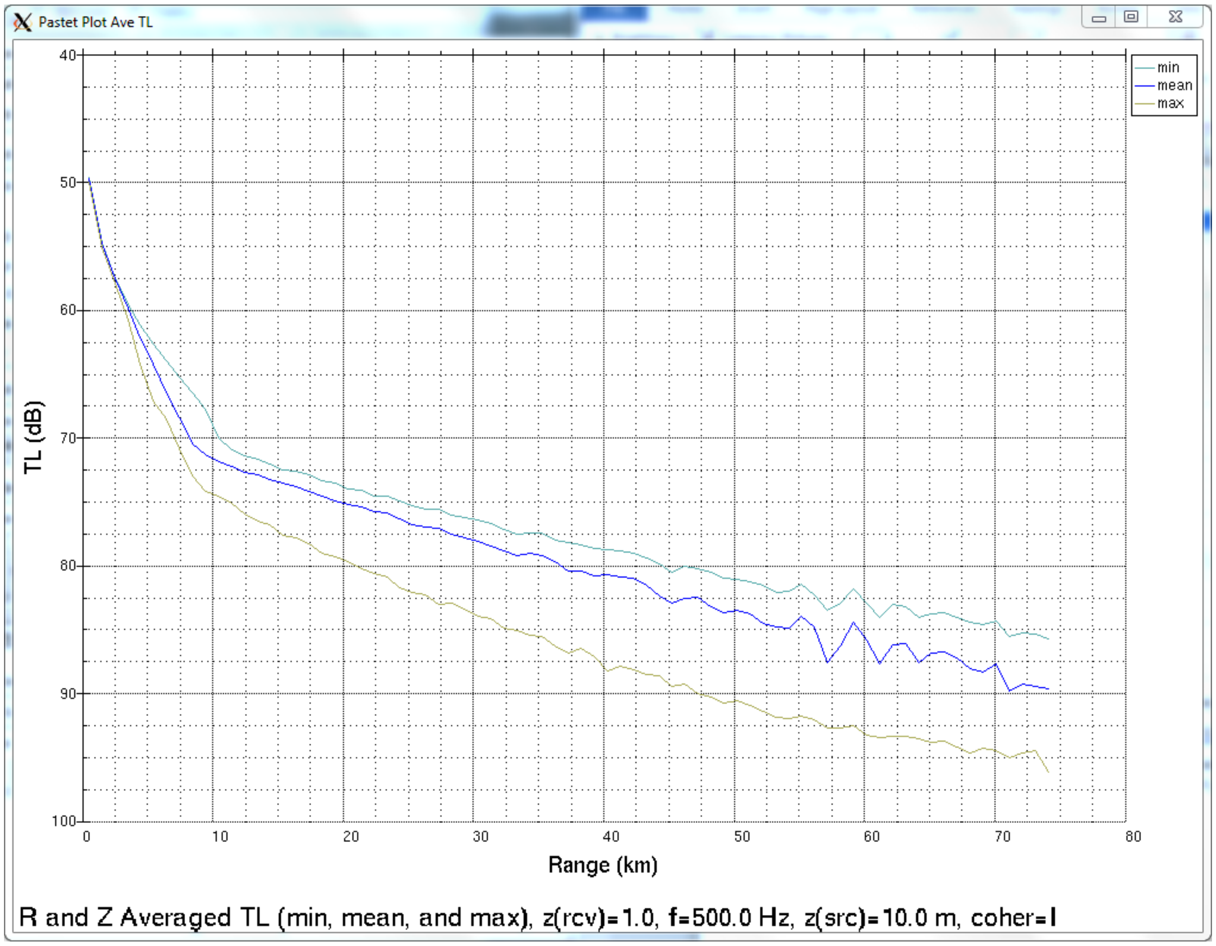


BELLHOP- PASTET: Mean Parameter Values, 500Hz 10.0m source depth, coher=I

Figure 8. Example of `Raw TL` *plot.*

The `Raw TL` files have 100 receiver depths; these are decimated to 34 curves for the purpose of the plot. This is partly because doing so visually improves the look of the plot, but also because the plotting utility currently in use has a maximum of 34 curves per plot.

An example of the plot displayed by the "`Ave TL`" button is shown in Figure 9. In addition to portraying an overview of the minimum and maximum TL variations above and below the mean TL, for all input perturbations considered together, it also has been "averaged" (i.e., smoothed) in range and receiver depth.



R and Z Averaged TL (min, mean, and max), z(rcv)=1.0, f=500.0 Hz, z(src)=10.0 m, coher=I

Figure 9. Example of `Ave TL` *plot, showing the mean TL curve and the maximum and minimum envelope curves due to the plus and minus sigma perturbations of all the input variables.*

The third type of TL plot available in PASTET is the "`Plot Var TL`" output product, an example of which is shown in Figure 10. Here the full-resolution TL is shown (not averaged or decimated in range or receiver depth) and 13 separate curves show the mean and the +/- sigma variations of each parameter. Note that this particular example was run with the coherency option set to "`C`" (coherent pressure summation), as opposed to the previous figures that showed the results of the same case, except for the selection of coherency option "I" (incoherent pressure summation). When the `Var TL` plot is shown for the incoherent case, it looks quite similar to Figure 9.

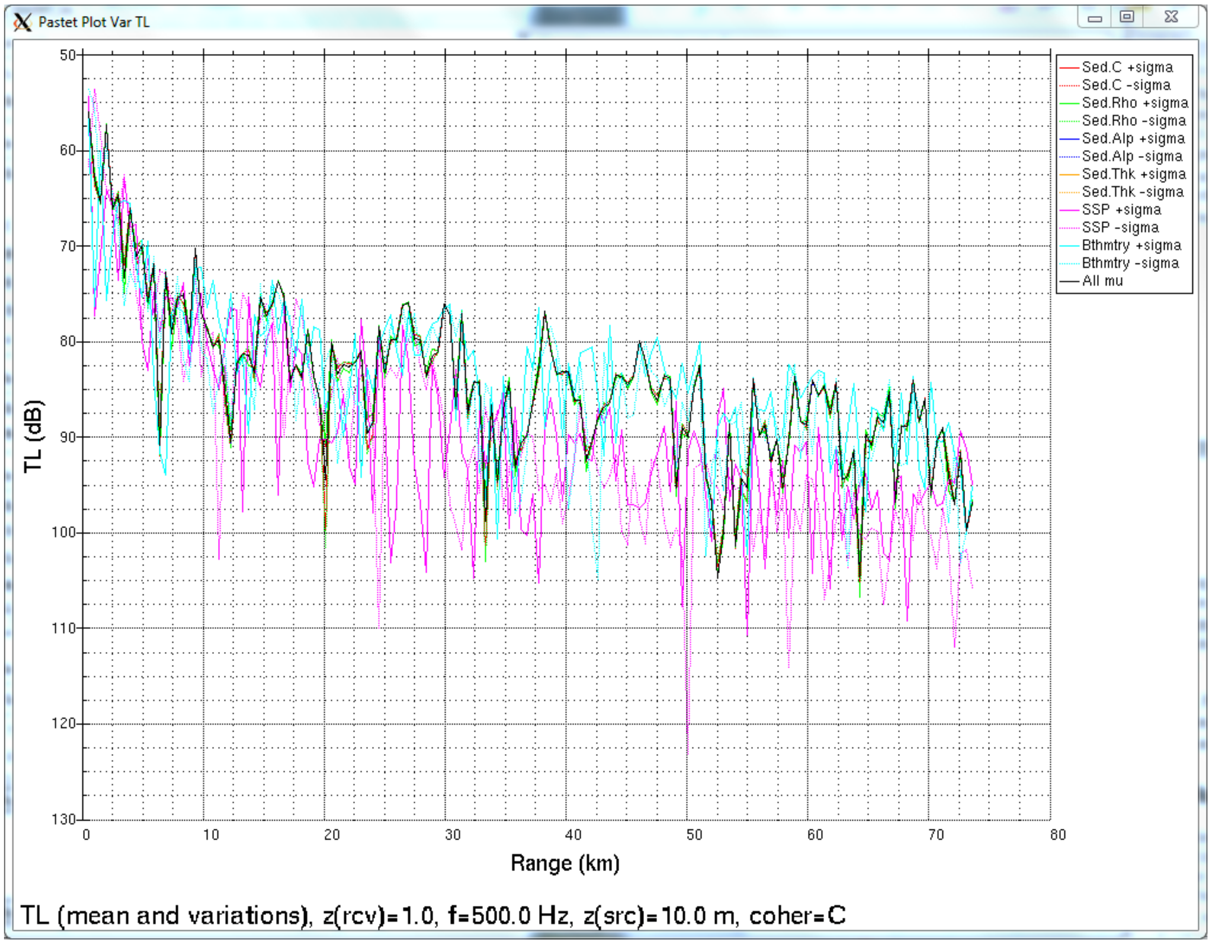


Figure 10. Example of `Var TL` *output product, run using coherent pressure summation, instead of incoherent as in the previous figures.*

The fourth type of TL-related graphical output product is the sensitivity line plot, which consists of a set of coloured lines, in dB versus, range for the six statistical variables. This plot shares the same receiver depth selection menu as the Average TL plot (and the two buttons are side by side on the main GUI window). An example of a sensitivity plot is shown in Figure 11. Although shown in dB in the plot, the sensitivity is calculated internally in linear units, and is sometimes zero; since a value of zero cannot be portrayed in a dB plot (i.e., it would be negative infinity in dB) gaps will appear in the line plot where the value is calculated as zero—this can be seen, for example, in the sediment density sensitivity (green line) in Figure 11.

Each of these plots can be opened by clicking on the corresponding “Plot” button, of course. The button turns black while the plot is open. The plot can be closed by clicking the button again (it is a toggle), or by clicking the “X” in the upper right corner of the window. If another `Run` is started, all the open plot windows will be automatically closed, and even the plotting controls will be removed until the new run is complete.

In addition to the graphical output products, PASTET writes a full copy of all input and output variables and values for every run to a comma-separated-values (CSV) disk file. The previous files are automatically renamed to serve as an archive. The file name is `PASTET_runs.csv`.

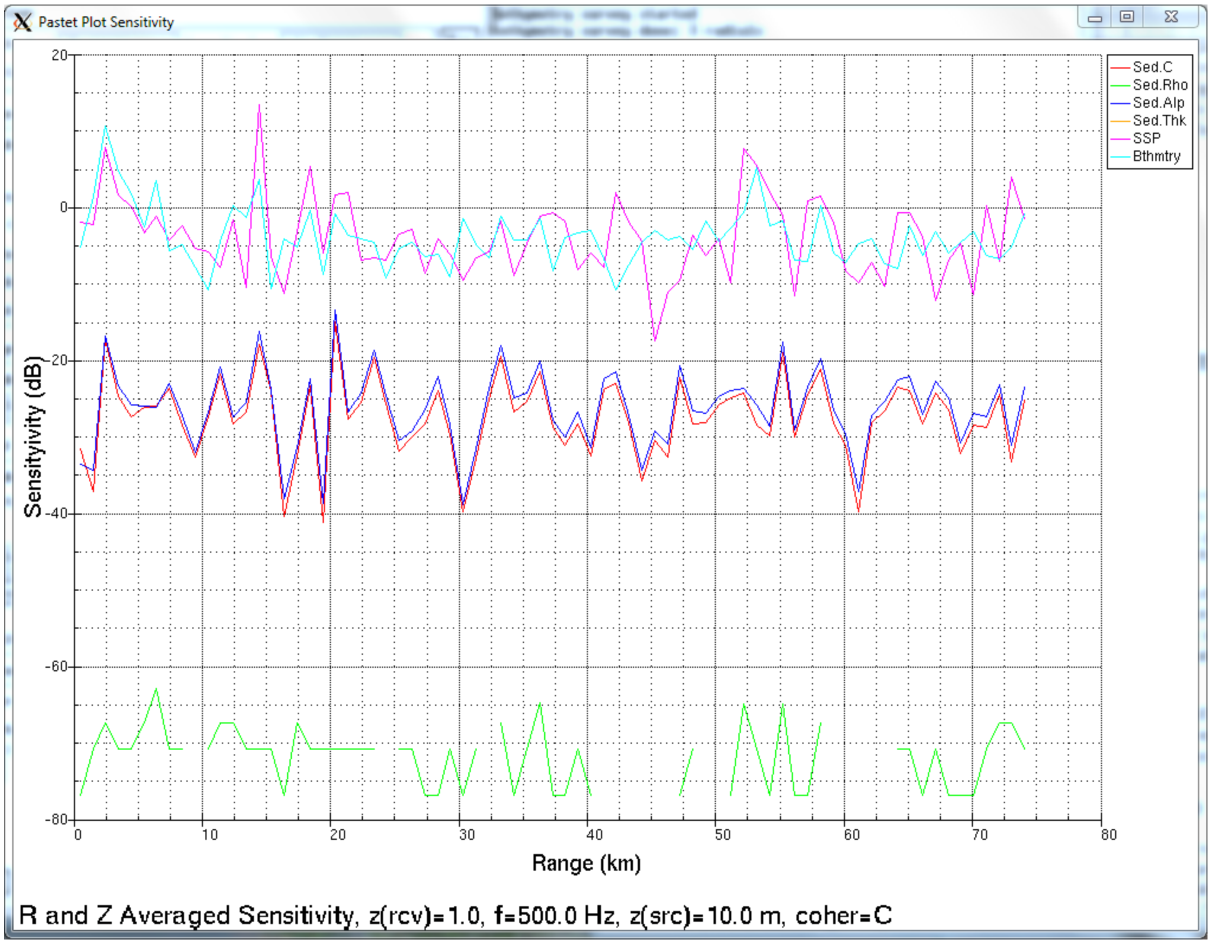


Figure 11. Example of sensitivity line plot, in dB versus range, for each of the six statistical variables.

3.3 EOF Analysis

In addition to representing the SSP variations in terms of a mean profile and plus/minus standard deviations, the option is available to use an EOF analysis to model the range of SSP variation inherent in the survey results, and its impact of the TL sensitivity calculations. Unlike the mu ± sigma option, the EOF option cannot be driven by GUI input alone, but requires a survey return of at least 5 SSPs. Consequently, the EOF button will not be selectable until after a successful survey is returned.

The requirement for a minimum of specifically 5 SSPs is contained in the header file for the Monte Carlo methodology base class, **`Monte_Carlo_methods.h`**, and defined by the static constant named **`MIN_SAMPLES_MONTE_CARLO`**. However, it is also an argument to the **`SSP_EOF`** class constructor and the default can be overridden when that object is instantiated.

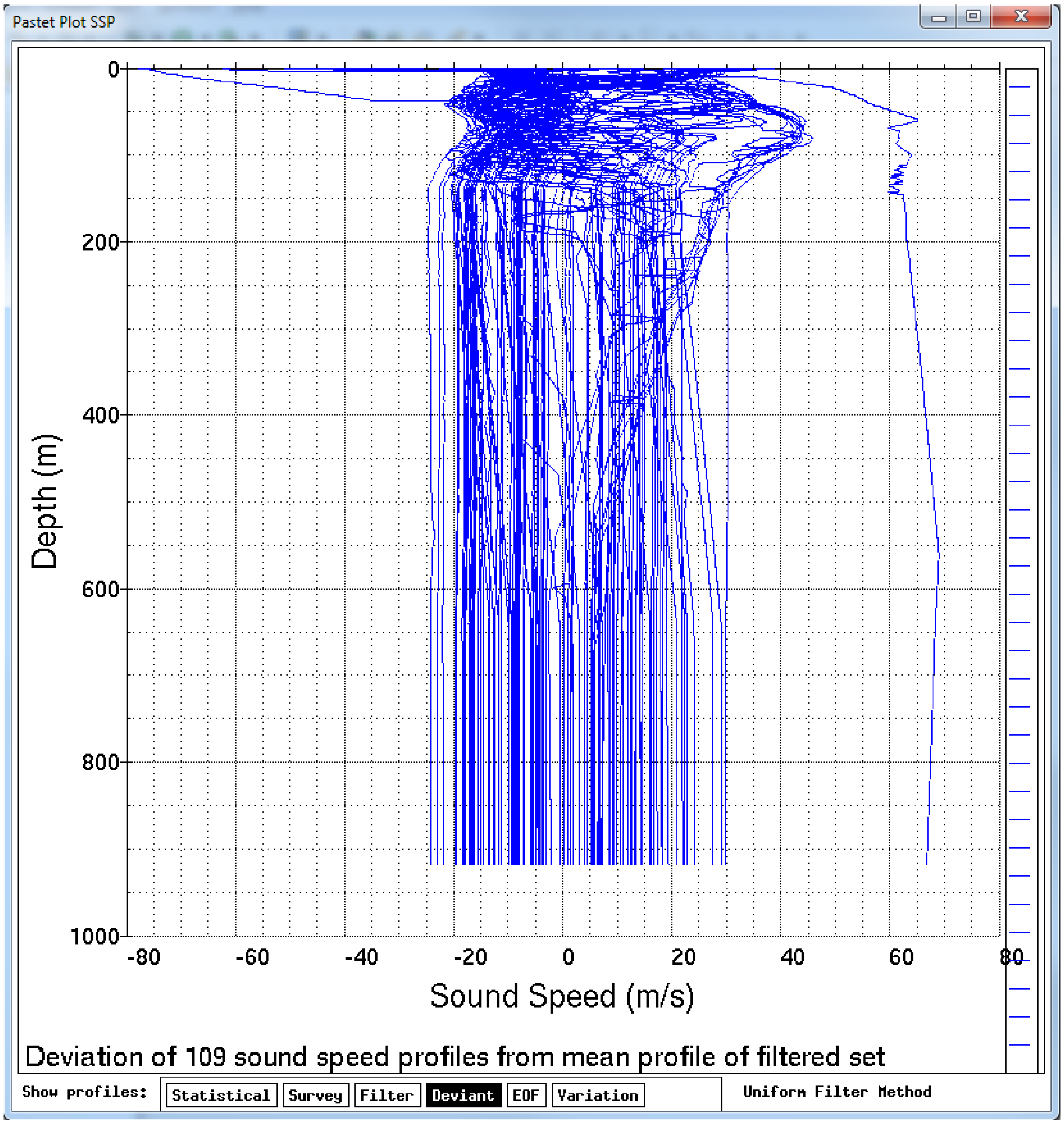


Figure 12. Example of the "Deviant" sound speed profile plot option.

If the EOF button is then selected, it will reveal three additional controls that can be used to fine-tune the EOF analysis, or they can be left to their default values. The "EOFs:" field allows the operator to specify a minimum number of EOFs that will be used in generating the synthetic SSPs. The "Th%" field allows the specification of a minimum threshold for eigenvalue "energy" or variance; making this percentage non-zero may mean that more EOFs will be used than the number specified in the "EOFs" field, if additional eigenvalues are required to reach the threshold value. The "runs:" field indicates the number of propagation loss runs that will be used to assess the SSP variability impact on TL sensitivity. A greater number of runs will result in a longer wait time for the run results to be returned.

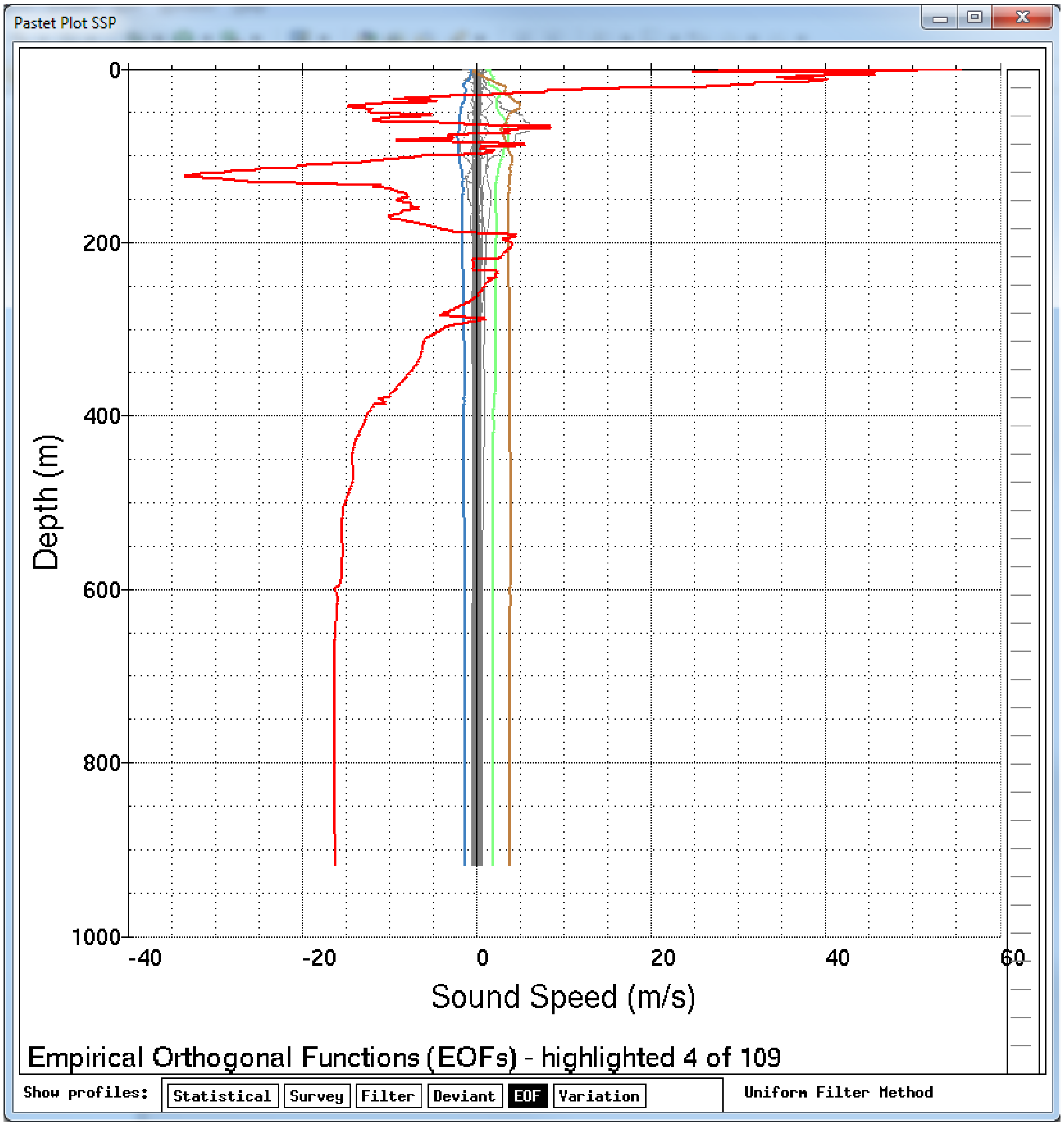


Figure 13. Example of the “EOF” sound speed profile plot option.

When the EOF option is selected, it works in concert with the SSP averaging method selected in the “Sound Speed” survey specification box. Best results for EOF runs are to be expected when the “Uniform” SSP averaging method is selected, but this will also require the longest run times. Other SSP averaging methods can be tried to see what EOF results they also produce.

There are additional SSP plot options that are available on the SSP “View” window when the EOF analysis option is activated. These can be used to inspect the EOF analysis details, which allow the operator to evaluate whether the EOF analysis is going to be appropriate or successful with this particular set of survey results. The “Deviant” plot option is illustrated in Figure 12. It is similar to the “Filter” plot option previously discussed in an earlier section, but instead of showing the actual SSP, it shows each SSP after the mean profile has been subtracted from it; effectively showing how the input SSPs vary about the mean profile.

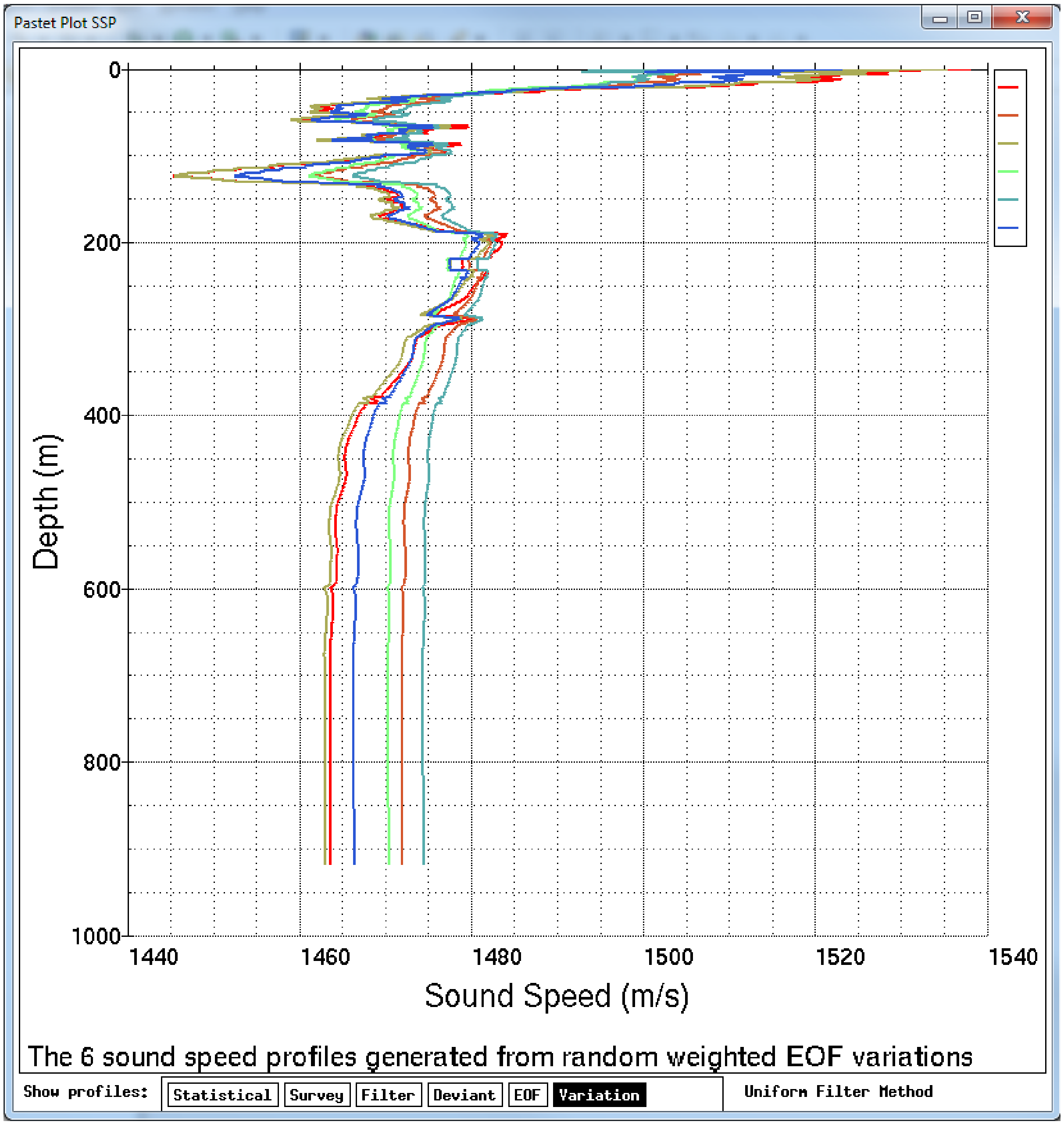


Figure 14. Example of SSP variations plot generated by the EOF analysis feature.

The “EOF” plot option, as shown for example in Figure 13, displays a form of the computed EOFs—actually the normalized eigenfunctions scaled by the square root of its eigenvalue. The specified number “EOFs:” of profiles are coloured in a sequence of bright or pastel colours, then the remaining (unhighlighted and unselected) EOFs are portrayed in the background in a neutral grey colour.

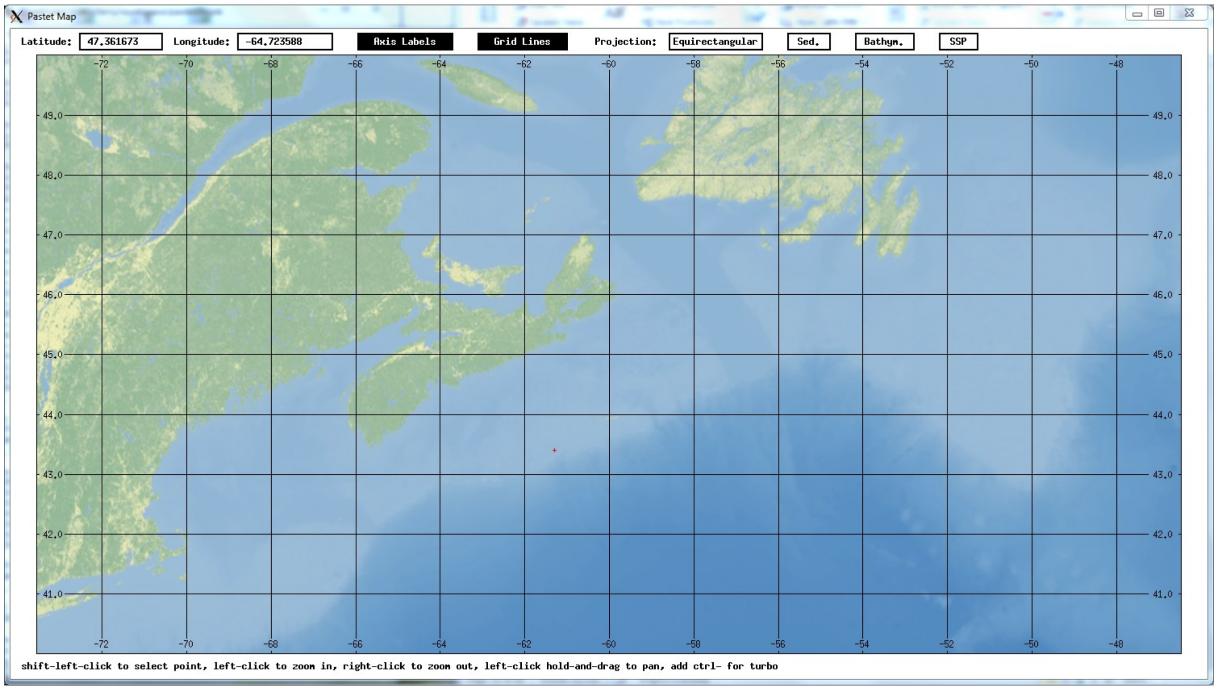
The selected EOFs are used to randomly generate a set of “runs:” SSP variations. These can be seen in the plot that is shown using the “Variation” option. See Figure 14 for example.

3.4 Bathymetry Spectrum Analysis

Although the GUI presently includes a couple of controls relating to the analysis of bathymetric variations through a power spectrum model of fluctuations, they don’t have any effect at this time, as the implementation of this feature is not yet complete.

3.5 Interactive Map

The interactive map includes a map projection menu and options for axis labelling and grid lines. An example of the map is displayed in Figure 15. The map is opened when the operator clicks on the `Map` button, which then changes colour to black to indicate that the map is open; the map can be closed either by clicking on the `Map` button again (it is a toggle), or by clicking on the “X” in the top right corner of the map window.


Latitude: 47.361673 Longitude: -64.723588 Projection: Equirectangular

shift-left-click to select point, left-click to zoom in, right-click to zoom out, left-click hold-and-drag to pan, add ctrl- for turbo

Figure 15. Example of PASTET interactive map.

The main function of the map is to allow graphical selection of the PASTET modelling location (i.e., origin point). This can be done by placing the pointer at the desired place on the map, and holding down the SHIFT key on the keyboard, then clicking the left pointer button once. When the SHIFT key is held down, the pointer changes from an arrow to a cross-hair, to allow more accurate point selection. Once the origin location point has been selected, a small red cross is placed on the map to mark the spot. The selected location point can be moved to a different place using the same procedure. The latitude-longitude of the pointer “hot spot” is continuously updated in the two text fields at the upper left of the map whenever the pointer is on the map.

The map has several other interactive functions. The axis labelling and the grid lines can be independently toggled on or off. The map projection can be selected; currently there are two map projections available: equirectangular and Mercator. Additional map projections could be added in a future upgrade of PASTET, if required.

The operator can zoom in closer to a point on the map by clicking the left pointer button; likewise, zoom-out is performed by clicking the right pointer button. The equivalent operations can also be accomplished by upward and downward rotation of the pointer device thumbwheel, if it has one. The map can also be panned in any direction by clicking the left pointer button on the map and dragging it in the opposite direction (this has the effect of grabbing and dragging the map sheet as if it was a sheet of paper).

Both the zoom and pan functions described above can have their effects multiplied by a factor of two if the CTRL button is held down while the operation is performed.

There are three options in the upper right of the map border, to select the display of the locations of the survey results data for sediment properties, bathymetry, and SSP, respectively. In the case of sediment properties, there are two distinct map symbols drawn: an orange diamond for sediment thickness locations and a green square for sediment type locations. The bathymetry locations are shown by a grey “x” symbol and the SSP locations are represented by a magenta “+” symbol. Figure 16 shows an example of a map where all the survey results locations are shown together, however the sediment properties, bathymetry, and SSP locations symbols can be switched on and off independently.

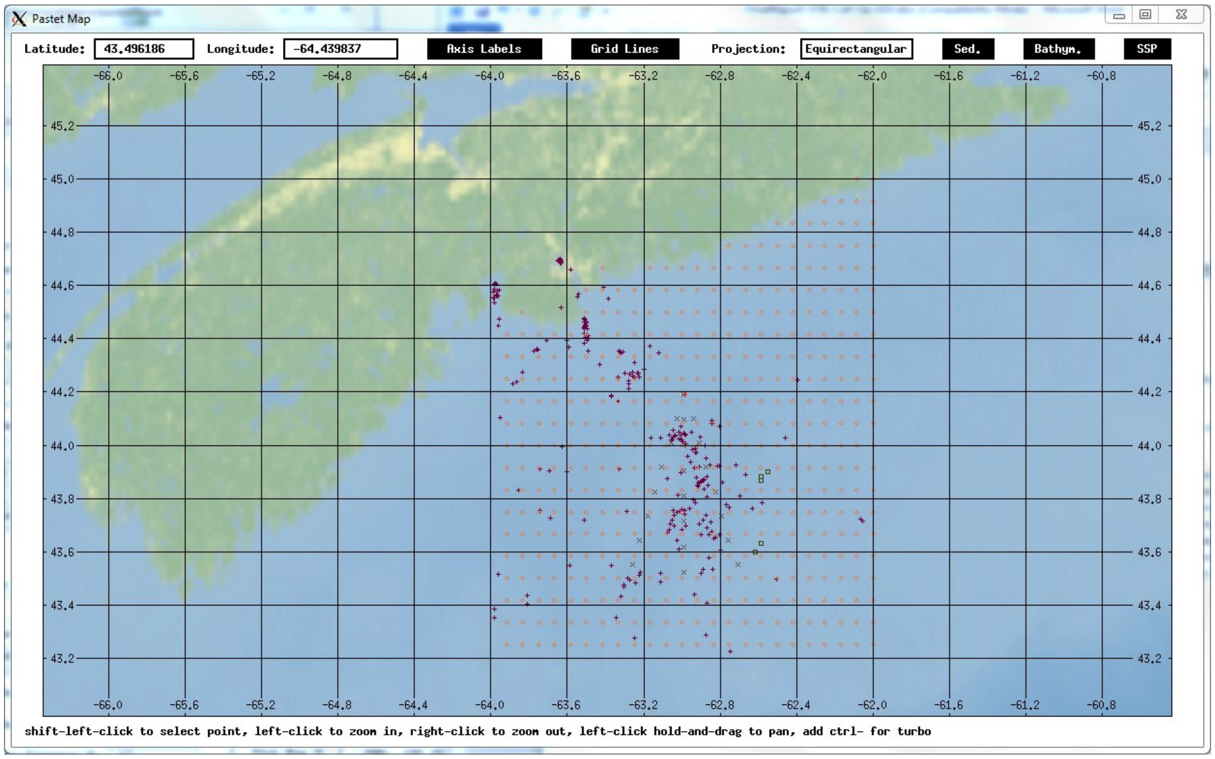

Pastet Map
Latitude: 43.496186
Longitude: -64.439837
Axis Labels
Grid Lines
Projection: Equirectangular
Sed.
Bathym.
SSP
shift-left-click to select point, left-click to zoom in, right-click to zoom out, left-click hold-and-drag to pan, add ctrl- for turbo

Figure 16. Example of a PASTET map display with all the optional survey results locations symbols turned on.

4 PASTET Installation and Development Notes

PASTET is now a C++ application integrated into the Eclipse Software Development Environment. [2] The following tips provide advice to get the most from the Eclipse environment with PASTET:

a. Ensure that the CDT plug-in is installed in Eclipse;

b. Import the project files into Eclipse;

c. Select the build configuration, either “STB” or “StandAlone”;

d. Build the project.

In order to run the PASTET application, an X-Windows terminal server must be running. An example shell script that launches the application is stored in the file named “p” in the top level of the project directory. Consequently, it can be launched from an X-Term window by switching to that directory and entering the keyboard command “. p” (without quotes).

5 Conclusions

The present call-up has augmented PASTET with the capability to perform a nonlinear estimate of the transmission loss (TL) sensitivity to sound speed profile (SSP) variance through an Empirical Orthogonal Function (EOF) analysis using a Monte Carlo methodology. Progress has also been made on further extending PASTET to perform a nonlinear estimate of the TL sensitivity to bathymetry variability, through a power spectrum analysis of fluctuations and a Monte Carlo methodology. Work has also begun on an additional option to handle spatial field shifting, which finds the correlation between a reference acoustic field and its perturbed variations.

The present call-up has also investigated and corrected some previously identified software errors and deficiencies in the PASTET program.

Some of requested tasks in this call-up could not be accomplished, due to the scope of work being bounded by a $50K fixed budget. The tasks that were not completed are the nonlinear sensitivity analysis of bathymetry variations and the field shifting enhancements, along with some of the previously identified software deficiencies.

Recommendations for additional work are included in the next section of this report.

6 Recommendations

1. The EOF-based generation of SSP variations should be further developed, addressing the aspect of the random weightings of the EOFs being restricted to the range appropriate for the subset of weighting ranges that reproduce the original SSP input set.

2. The bathymetry power spectrum variations model that has been started should be completed.

3. The spatial field shifting option that has been started should be completed.

4. Finish the implementation of two more output products: the full-field plot of sensitivity versus range and receiver depth, and the full-field plot of transmission loss versus range and receiver depth.

5. Investigate and resolve the known unresolved software deficiencies that are detailed in Section 2.4 above.

6. Implementation of additional graphical output products, including advanced display of the sensitivity results and other graphical output products that will enhance the effectiveness and utility of the PASTET application. This may include graphical display of some of the results from the environmental survey aspect of the application as well as from the propagation and sonar performance analysis aspects. One example would be a probability of detection display.

7. Enhancements to the operational and analysis capability of the PASTET application, such as support for multiple radials, uncertainty ellipsoids, etc.

8. PASTET currently makes limited use internally of an X11-Instrinsics facility called “work procs”. When used to full effect, this will allow the X11 GUI elements to refresh and update even while a lengthy `Survey` or `Run` is in progress. The internal use of “Work Procs” in PASTET should be extended to cover those situations.

9. The databases available to PASTET for the extraction of bathymetric profiles should be extended to include commonly available stand-alone bathymetry databases, such as SRTM30plus. PASTET does not currently access those databases for the purpose of bathymetric profiles.

10. PASTET should be enhanced to use a dynamic configuration file, so that it can start up with variable values and settings that it was using when it previously shut down.

11. PASTET should be enhanced so that when the plot windows and map windows are opened, they have the same size, shape, placement, and options selected as when they were previously closed.

12. Speed increases for the PDB queries may be improved through optimization and testing. Currently the speed at which some bottom and sound velocity queries execute seem slower that the READB. This is not a hard statement of fact, just an impression gained through development.

13. Once model data is firmed up, implementing a PDB strategy to cache this data would be viable. Annex E Experimental PDB Model Data Query (developed on DRDC cruise Q325) may help. This query used a table structure similar to the PDB and was hard coded for location, date and model type. However, it does work and may be used to help with the details of writing a more generic model management system for the PDB.

14. Implement a PASTET help system, especially if this tool will be used by others outside of the DRDC Modelling Group.

15. Continuation of the refactoring work that was begun at DRDC in 2012, to develop PASTET into a more robustly object-oriented software architecture.

References

[1] Call-up 015, order no. W7707-4500991706, under Standing Offer Contract #W7707-098179/001/HAL entitled *Maintenance, Development, and Enhancements of the System Test Bed and Environment Modeling Manager and Associated Software*. This call up is entitled *Implementation of nonlinear sensitivity measures in PASTET* and dated 2012-11-29.

[2] Eclipse Software Development Environment, http://www.eclipse.org/

[3] Model–view–controller, http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

[4] Jordan Pronk, *Software Design Document — Portable Acoustic Sensitivity Transmission Estimation Tool (PASTET)*, DRDC internal document, 3 May 2012, 27 pp.

[5] *Numerical Recipes: The Art of Scientific Computing*, Third Edition. http://www.nr.com/

This page intentionally left blank.

List of symbols/abbreviations/acronyms/initialisms

ASCII	American Standard Code for Information Interchange
BP	Bottom Profile
CCW	Counter clockwise
CFAV	Canadian Forces Auxiliary Vessel
CGI	Common Gateway Interface
CNOOFS	Canadian Newfoundland Operational Ocean Forecast System
CORA	Centre for Operational Research and Analysis
CSA	Contractual Scientific Authority
CW	Clockwise
DND	Department of National Defence
DRDC	Defence Research & Development Canada
DREnet	Defence Research Establishment Network
EMM	Environmental Modelling Manager
EOF	Empirical Orthogonal Function
ETOPO1/2/5	Electronic Topography (database), the number is the resolution in arc-minutes
FTP	File Transfer Protocol
GNU	A system of open-source, user-supported software, related to Unix (but G Not Unix)
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
LOB	Line of Bearing
MIME	Multipurpose Internet Mail Extensions
NCOM	Naval Coastal Ocean Model
NetCDF	Network Common Data Form
NaN	Not a Number (a flag for a floating point numerical error).
nm	Nautical Mile
PASTET	Portable Acoustic Sensitivity and Transmission Evaluation Tool
PDB	Production Database (successor to READB)
PHP	PHP: Hypertext Preprocessor

PLEIADES	DRDC-Atlantic sonar platform built on the STB
PPT	Parts Per Thousand
PWGSC	Public Works and Government Services, Canada
READB	Rapid Environmental Assessment Database
SA	Scientific Authority
SE	Signal Excess
SQL	Structured Query Language
SSP	Sound Speed Profile
STB	System Test Bed
TL	Transmission Loss
TSD	Temperature Salinity Depth
XBT	Expendable Bathythermograph

Annex A Monte_Carlo_methods.h

```
/*
 * Monte_Carlo_methods.h
 *
 *  Created on: 2013-01-21
 *      Author: Terry J. Deveau
 */

#ifndef MONTE_CARLO_METHODS_H_
#define MONTE_CARLO_METHODS_H_

static const int MIN_SAMPLES_MONTE_CARLO = 5;
static const int MIN_RUNS_MONTE_CARLO = 6;
static const int MAX_RUNS_MONTE_CARLO = 64;

class Monte_Carlo_methods
{
	friend class SSP_EOF;
	friend class Bathym_PS;
	friend class Range_depth_shifting;
public:
	bool toggle_on; // true if the method is currently toggled on for use
	bool selected; // true if the user has opted to use the method, when
activated
	const int min_samples; // minimum number of samples to allow activation
	const int min_runs; // minimum value allowed for n_runs
	const int max_runs; // maximum value allowed for n_runs
private:
	bool activated; // true if the method is available for use
	int n_runs; // number of runs that the method will use

public:
	bool set_n_runs( const int & _n_runs = MIN_RUNS_MONTE_CARLO )
	{
		if ( _n_runs < min_runs || _n_runs > max_runs )
			return false;

		n_runs = _n_runs;
		return true;
	}

	int get_n_runs( void ) const { return n_runs; }

	virtual bool activate( void )
	{
		activated = true;
		return activated;
	}

	virtual void deactivate( void )
	{
		activated = false;
	}

	bool get_is_activated( void ) const { return activated; }
	Monte_Carlo_methods( const int & _n_runs = MIN_RUNS_MONTE_CARLO,
			const int & _min_runs = MIN_RUNS_MONTE_CARLO,
			const int & _max_runs = MAX_RUNS_MONTE_CARLO,
			const int & _min_samples = MIN_SAMPLES_MONTE_CARLO)
	:	toggle_on(false),
		selected(false),
		activated(false),
		n_runs(_n_runs),
		min_runs(_min_runs),
		max_runs(_max_runs),
		min_samples(_min_samples)
	{}

	virtual ~Monte_Carlo_methods( void ) {}
};


#endif /* MONTE_CARLO_METHODS_H_ */
```

Annex B SSP_EOF.h

```
/*
 * SSP_EOF.h
 *
 *  Created on: 2013-01-21
 *      Author: Terry J. Deveau
 */

#ifndef SSP_EOF_H_
#define SSP_EOF_H_

#include <vector>
#include "Monte_Carlo_methods.h"
#include "../pastet_structs.h"
#include "../GraphPlotting.h"
#include <gsl/gsl_qrng.h>
#include <gsl/gsl_randist.h>

static const int DEF_SSP_EOFS = 4;
static const int MIN_SSP_EOFS = 3;
static const int MAX_SSP_EOFS = 64;
static const double DEF_CUM_EVAL_THRESHOLD = 0.0;
static const int MIN_CUM_EVAL_THRESHOLD = 0.0;
static const int MAX_CUM_EVAL_THRESHOLD = 1.0;

class SSP_EOF : public Monte_Carlo_methods
{
private:
       std::vector<int> meanN;
       std::vector<double> meanDepth;
       std::vector<double> meanSpeed;
       std::vector<double> *collSpeed;
       std::vector<double> *eigenvectors;
       std::vector<double> *eigenvalues_norm;
       std::vector<double> *eigenvalues_unnorm;
       std::vector<double> *SSP_variations;
       int n_coll, n_points, n_evals;
       double maxDepth;
       int n_eofs;
       double threshold;
       int n_eofs_use;

       gsl_rng * gsl_rng_data;

       void eof_analysis( void );

public:

       const std::vector<double> *depthPoints;
       const std::vector<double> * const *speedPoints;

       bool activate( SENSJOB* const &pSJ, ProtoPASTET* const &pPP  );
       void deactivate( void );
       int get_n_eofs( void ) const { return n_eofs; }
       bool set_n_eofs( const int & _n_eofs = DEF_SSP_EOFS )
	{
		if ( _n_eofs < MIN_SSP_EOFS || _n_eofs > MAX_SSP_EOFS )
			return false;

		n_eofs = _n_eofs;
		return true;
	}
	double get_threshold( void ) const { return threshold; }
	bool set_threshold( const double & _threshold = DEF_CUM_EVAL_THRESHOLD )
	{
		if ( _threshold < MIN_CUM_EVAL_THRESHOLD || _threshold >
						MAX_CUM_EVAL_THRESHOLD )
			return false;

		threshold = _threshold;
		return true;
	}

	friend void GraphPlotting::load_plotSSP (Widget &soundSpeedPlot, SENSJOB*
					const &pSJ, ProtoPASTET* const &pPP );

	SSP_EOF( const int & _n_runs = MIN_RUNS_MONTE_CARLO,
			const int & _min_runs = MIN_RUNS_MONTE_CARLO,
			const int & _max_runs = MAX_RUNS_MONTE_CARLO,
			const int & _min_samples = MIN_SAMPLES_MONTE_CARLO,
			const int & _n_eofs = DEF_SSP_EOFS,
			const double & _threshold = DEF_CUM_EVAL_THRESHOLD )
	: Monte_Carlo_methods( _n_runs, _min_runs, _max_runs, _min_samples )
	{
		depthPoints = &meanDepth;
		speedPoints = &SSP_variations;

		n_eofs = _n_eofs;
		threshold = _threshold;
		collSpeed = NULL;
		eigenvectors = NULL;
		eigenvalues_norm = NULL;
		eigenvalues_unnorm = NULL;
		SSP_variations = NULL;
		n_points = 0;
		n_coll = 0;
		n_evals = 0;
		maxDepth = 0.0;

	// GNU Scientific Library (gsl) used to obtain Gaussian random deviates
		gsl_rng_data = gsl_rng_alloc ( gsl_rng_taus );
		gsl_rng_set ( gsl_rng_data, (unsigned long int) time(0) );
	}

	~SSP_EOF( void )
	{
		gsl_rng_free (gsl_rng_data);
	}
};


#endif /* SSP_EOF_H_ */
```

Annex C SSP_EOF.cpp

```
/*
 * SSP_EOF.cpp
 *
 * Created January, 2013
 * Created by Terry J. Deveau (TJD)
 *
 * Based upon an IDL program, ssp_svd_temp
 *
 * The code below is extracted from the IDL program that was used as a template
 * in developing the "activate" member function, and other related functions.
 *
       max_ssps=500
       max_dpths=500

; vector of starting indexes into tmparr1 and tmparr2 for start of the raw profiles
       ssp_indices=lonarr(max_ssps)

       tmparr1[line_indx] ; depth of mu, sigma, and raw profiles
       tmparr2[line_indx] ; speed of mu, sigma, and raw profiles
       ssp_mean_index ; index to tmparr1 and tmparr2 for start of mu profile
       ssp_min_index ; index to tmparr1 and tmparr2 for start of mu-sigma profile
       ssp_indices[i] ; index to tmparr1 and tmparr2 for start of I'th raw profile
       n_ssps ; total number of raw profiles
       ssp_indices[n_ssps] ; end of last raw profile index plus 1

       ;*****determine depth and speed for mean SSP from PASTET output file***

       dpth_mean_vec1=float(tmparr1[ssp_mean_index:ssp_min_index-3L])
       spd_mean_vec1=float(tmparr2[ssp_mean_index:ssp_min_index-3L])
       if dpth_mean_vec1[0] ne 0. then begin
           dpth_mean_vec1=[0.,dpth_mean_vec1]
           spd_mean_vec1=[spd_mean_vec1[0],spd_mean_vec1]
           speed_at_zero=spd_mean_vec1[0] ; NB: SHOULD BE AFTER "endif" not before
       endif
       mx_mean_depth=max(dpth_mean_vec1)

       ;*****interpolate speed and depth to 1-m intervals ***

       interpolating_depths=findgen(fix(mx_mean_depth)+1) ; vector of depths in
                                                  consecutive whole meters
       interpolating_speeds=interpol(spd_mean_vec1,
                                  dpth_mean_vec1,interpolating_depths)
       dpth_mean_vec1=interpolating_depths
       spd_mean_vec1=interpolating_speeds
       n_out=n_elements(dpth_mean_vec1)
       ssp_array=fltarr(n_ssps+1,n_out)
       for j=0,n_ssps-1 do begin
           dpth_vec1=float(tmparr1[ssp_indices[j]:ssp_indices[j+1]-3L])
           max_depth=max(dpth_vec1,index_max)

           spd_vec1=float(tmparr2[ssp_indices[j]:ssp_indices[j+1]-3L])

           ; ***********match depths at maximum for the given SSP ***********
           speed_at_max_depth=spd_vec1[index_max]
```

```
            if max_depth+10. lt mx_mean_depth then begin
                match_depth=fix(max_depth+0.49)
                match_speed=interpolating_speeds[match_depth]
                delta_speed=match_speed-speed_at_max_depth
                spd_vec1=spd_vec1+delta_speed
                spd_vec1=[spd_vec1,interpolating_speeds[match_depth+1:*]]
                dpth_vec1=[dpth_vec1,interpolating_depths[match_depth+1:*]]
            endif
            if dpth_vec1[0] ne 0. then begin
                dpth_vec1=[0.,dpth_vec1]
                spd_vec1=[spd_vec1[0],spd_vec1]
            endif
            delta_speed_at_zero=speed_at_zero-spd_vec1[0]
            spd_vec1=spd_vec1+delta_speed_at_zero

            ;**********interpolate SSP to depth vector
            outspds=interpol(spd_vec1,dpth_vec1,dpth_mean_vec1)
            ssp_array[j+1,*]=outspds
        endfor

        ssp_mean=spd_mean_vec1

        ssp_array[0,*]=dpth_mean_vec1

        c_arr=fltarr(n_out,n_ssps)
        for j1=0,n_ssps-1 do
               for j2=0,n_out-1 do
                      c_arr[j2,j1] = ssp_array[j1+1,j2]-ssp_mean[j2]

        ;perform EOF analysis on differences from mean
        eof_analysis, c_arr,n_out,n_ssps,eofs,coeff,eigvls,local_var,eigvls1

        eig_cum=total(eigvls,/cumulative)
        threshold=0.98
        a_eig_cum=where(eig_cum ge threshold,count)

        if count gt 0 then
               n_eofs=a_eig_cum+1 else n_eofs=n_ssps
        ;***or just use a fixed number (e.g. 4)
        ;print, eigvls[0]+eigvls[1]+eigvls[2]+eigvls[3] ;gives the variance of the
first four EOFs

        n_eofs=min([4,n_ssps])
        ssps_out=fltarr(n_mc+1,n_out)
        ssps_out[0,*]=dpth_mean_vec1

        seed=5L
        randoms=randomn(seed,n_mc,n_eofs)

        for j=0,n_mc-1 do begin
               for k=0,n_eofs-1 do begin
                      tmp=randoms[j,k]*sqrt(eigvls1[k])*eofs[*,k]
                      ssps_out[j+1,*]=ssps_out[j+1,*]+tmp
               endfor
               ssps_out[j+1,*]=ssps_out[j+1,*]+ssp_mean
        endfor
        n_out=n_out+1

        ;***write outputs to a file to check
        ssps_out2=fltarr(n_mc+1,n_out-1)
        tempdep=findgen(n_out-1)*max(dpth_mean_vec1)/float(n_out-2)
        for j=1,n_mc do
               ssps_out2[j,*]=interpol(ssps_out[j,*],dpth_mean_vec1,tempdep)
        ssps_out=fltarr(n_mc+1,n_out)

        ssps_out[*,0:n_out-2]=ssps_out2
        ssps_out[0,0:n_out-2]=tempdep
        ssps_out[0,n_out-1]=dpth
        for l=1,n_mc do
               ssps_out[l,n_out-1]=ssps_out[l,n_out-2]
        ssp_mean=interpol(ssp_mean,dpth_mean_vec1,tempdep)
        ssp_mean=[ssp_mean,tmparr2[ssp_min_index-3L]]
        write_csv,filename+'_MC.csv',float(round(10.*ssps_out))/10.
*/

#include <stdio.h>
#include <stdlib.h>
#include "../pastet.h"
#include "../GraphPlotting.h"
#include "SSP_EOF.h"

bool SSP_EOF::activate( SENSJOB* const &pSJ, ProtoPASTET* const &pPP )
{
        deactivate();
        bool rtn = true;
        // perform class SSP_EOF activation.

        int i, j, k, i_coll;
        int nRadials = 0;
        SSPRadial* pR = NULL;

        // get input data from one of the available SSP filter sources
        if ( pSJ->soundSpeedAveragingMethod == VERTEX_AVERAGING_METHOD )
        {
               nRadials = pPP->ulNRadialsSSPVrtx;
               pR = pPP->SSPVrtxRadials;
        }
        else if ( pSJ->soundSpeedAveragingMethod == SAMPLING_AVERAGING_METHOD )
        {
               nRadials = pPP->ulNRadialsSSPSamp;
               pR = pPP->SSPSampRadials;
        }
        else if ( pSJ->soundSpeedAveragingMethod == GRID_AVERAGING_METHOD )
        {
               nRadials = pPP->ulNRadialsSSPGrid;
               pR = pPP->SSPGridRadials;
        }
        else if ( pSJ->soundSpeedAveragingMethod == UNIFORM_AVERAGING_METHOD )
        {
               nRadials = pPP->ulNRadialsSSPUnfm;
               pR = pPP->SSPUnfmRadials;
        }

        n_coll = 0; // number of SSPs in the collection
        n_points = 0; // maximum number of depth points in any SSP in the collection
        int i_points = 0; // index of the SSP with the maximum number of points

        // find the SSP in the input data with the maximum number of points
        for ( i=0; i<nRadials; ++i )
```

```
                for ( j=0; j<pR[i].numSSPs; ++j )
                {
                        const SSPPoint *const pt = pR[i].SSPs[j].points;
                        // number of points in this SSP
                        const int npts = pR[i].SSPs[j].numPoints;
                        if ( npts > n_points )
                        {
                                n_points = npts;
                                i_points = n_coll;
                        }
                        ++n_coll;
                }

        // obtain the depth grid and copy the speed data to an internal array.
        // also prepare to calculate the mean profile of the input SSPs --
        // ignoring any "mean" profile that may already have been calculated
        // somewhere else for this data set, since it may not be a true arithmetic
        // mean.

        meanN.assign(n_points,0);
        meanSpeed.assign(n_points,0.0);
        collSpeed = new std::vector<double>[n_coll];
        i_coll = 0;

        for ( i=0; i<nRadials; ++i )
                for ( j=0; j<pR[i].numSSPs; ++j, ++i_coll )
                {
                        const SSPPoint *const pt = pR[i].SSPs[j].points;
                        const int npts = pR[i].SSPs[j].numPoints;
                        for ( k = 0; k < npts; ++k )
                        {
                                if ( i_coll == i_points )
                                        meanDepth.push_back( pt[k].depth );
                                meanN[k] += 1;
                                meanSpeed[k] += pt[k].speed;
                                collSpeed[i_coll].push_back( pt[k].speed );
                        }
                }

        maxDepth = meanDepth[n_points-1];

        // Now finalize the mean speed at every depth point in the mean vector
        for ( k = 0; k < n_points; ++k )
                if ( meanN[k] > 1 ) meanSpeed[k] /= (double)meanN[k];

// use the relative speeds of the mean vector to finish of the tails of any
// speed vectors in the collection that don't have data to maxDepth. Also,
// transform each SSP in the collection into a deviant vector from the mean SSP

        for ( i = 0; i < n_coll; ++i )
        {
                j = collSpeed[i].size();
                const double end_speed_delta =  collSpeed[i][j-1] - meanSpeed[j-1];

// TJD: the original IDL code subtracts from each SSP the difference between its
// sound speed at the surface and the mean sound speed at the surface. This makes
// it impossible to later reconstruct SSPs variations from the EOFs that have
// anything other than the mean sound speed at the surface.

//#define MATCH_TO_IDL_SOURCE_PIN_SURFACE_SPEED
```

```
#ifdef MATCH_TO_IDL_SOURCE_PIN_SURFACE_SPEED
             const double surface_speed_delta = meanSpeed[0] - collSpeed[i][0] +
                                                              end_speed_delta;
             for ( k = 0; k < j; ++k )
                   collSpeed[i][k] += surface_speed_delta - end_speed_delta -
                                                                 meanSpeed[k];
             for ( k = j; k < n_points; ++k )
                   collSpeed[i].push_back( surface_speed_delta );
#else
             for ( k = 0; k < j; ++k ) collSpeed[i][k] -= meanSpeed[k];
             for ( k = j; k < n_points; ++k )
                    collSpeed[i].push_back( end_speed_delta );
#endif // MATCH_TO_IDL_SOURCE_PIN_SURFACE_SPEED
       }

       SSP_EOF::eof_analysis();

       std::vector<double>eig_cum = *eigenvalues_norm;
       for ( i = 1; i < n_evals; ++i )
             eig_cum[i] += eig_cum[i-1];

       int count = 0;
       for ( i = 0; i < n_evals; ++i )
       {
             if ( eig_cum[i] < threshold )
                   ++count;
       }

// TJD: the IDL code calculated the number of EOFs to use by counting how
// many it took to accumulate a hard-coded threshold in the eigenvalues sum,
// but then it ignored that count and used a hard-coded number (4).
// Here, the hard-coded number is replaced by a class member variable (n_eofs)
// that defaults to a fixed number, but can be specified either at object
// instantiation or afterwards. Also, the threshold idea is retained, with a
// default likewise adjustable, and which will over-ride n_eofs if exceeding
// the threshold requires a higher count.

       n_eofs_use = n_eofs;
       if ( count > n_eofs_use ) n_eofs_use = count;
       if ( n_eofs_use > n_evals ) n_eofs_use = n_evals;

       SSP_variations = new std::vector<double>[n_runs];
       for ( i=0; i<n_runs; ++i )
       {
             for ( k=0; k<n_points; ++k )
                   SSP_variations[i].push_back( meanSpeed[k] );
             for ( j=0; j<n_eofs_use; ++j )
             {

// TJD: this EOF-based approach to generating SSP variations is incomplete as it
// stands. The IDL code uses random Gaussian deviates, with sigma = 1.0, to weight
// the sqrt(eigenvalue)*eigenvector summation, which is then applied to the mean.
// The problem is that the original SSP set does not represent all ranges of
// eigenvalue weightings, but only a restricted subset. In particular, the randomly
// positive and negative variations in the Gaussian deviates will cause a strong
// eigenvector to appear in its mirror-reversed form in some of the synthesized
// SSPs. Switching to a uniformly positive random deviate does not fully solve
// the problem, because the real SSP may in fact use a negative weight on some
// of these eigenvectors, so the positive weight may be the one that generates
// the mirror reversed version, but at least both normal and mirror-reversed
// versions of a strong eigenvector will not simultaneously be represented in
// the synthesized SSP variations. More work is required to base the weightings
// on the actual distribution of eigenvalue weightings in the original SSPs.

//#define MATCH_TO_IDL_SOURCE_GAUSSIAN_DEVIATES
#ifdef MATCH_TO_IDL_SOURCE_GAUSSIAN_DEVIATES
// GNU Scientific Library (gsl) used for Gaussian random deviates with sigma = 1.0
                    double weight = gsl_ran_gaussian( gsl_rng_data, 1.0 )
#else
// GNU Scientific Library (gsl) used for uniform random deviates from 0.0 - 1.0
                    double weight = gsl_rng_uniform( gsl_rng_data )
#endif // MATCH_TO_IDL_SOURCE_GAUSSIAN_DEVIATES
                           * sqrt( eigenvalues_unnorm->at(j) );
                    for ( k=0; k<n_points; ++k )
                           SSP_variations[i][k] += weight * eigenvectors[j][k];
             }
      }

// The IDL program has additional code here to ensure that the SSP_variations
// have a final depth point at the bottom depth, but in our case that was already
// taken care of during the depth filtering operation.

      if ( rtn ) rtn = Monte_Carlo_methods::activate();
      return rtn;
}

void SSP_EOF::deactivate( void )
{
      meanN.clear();
      meanDepth.clear();
      meanSpeed.clear();
      delete [] collSpeed;
      collSpeed = NULL;
      delete [] eigenvectors;
      eigenvectors = NULL;
      delete eigenvalues_norm;
      eigenvalues_norm = NULL;
      delete eigenvalues_unnorm;
      eigenvalues_unnorm = NULL;
      delete [] SSP_variations;
      SSP_variations = NULL;
      n_points = 0;
      n_coll = 0;
      n_evals = 0;
      maxDepth = 0.0;

      Monte_Carlo_methods::deactivate();
}
```

Annex D EOFanalysis.cpp

```
/*
 * EOFanalysis.cpp
 *
 * Created January, 2013
 * Created by Terry J. Deveau (TJD)
 *
 * Based upon two IDL programs, eof_analysis and EIGENQL
 *
 * See extensive comments below for more info on the IDL sources.
 *
*/

/*
; Copyright (c) 2001, Forschungszentrum Juelich GmbH ICG-II
; All rights reserved.
; Unauthorized reproduction prohibited.
;
;+
; NAME:
;       eof_analysis
;
; PURPOSE:
;       This procedure performs empirical orthogonal functions (EOF) analysis.
;
; CATEGORY:
;       MATH
;
; CALLING SEQUENCE:
;       eof_analysis, array, m, n, eofs, coeff, eigenvalues, local_var,
;           eigenvalues1
;
; INPUTS:
;       array:  (m,n)-array (double) on which eof_analysis is performed
;       m:      number of elements in the n observation vectors (e.g. points in
space)
;       n:      number of observations (e.g. points in time)
;
; OUTPUTS:
;
; TJD: the below has been edited to conform to IDL array(column,row) terminology.
;
;       m>n:        ! first calculate normalized eigenvectors of the covariance
;                         matrix of transpose(A) instead of A !
;
;       eofs:       (m,n)-array containing the n EOFs (n rows of the array)
;                         (spatial loading patterns, each of size m)
;       coeff:      (n,n)-array containing the n EOF-coefficients (n rows of the
;                         array) belonging to the n EOFs (time coefficients)
;       eigenvalues:(n)-vector containing the n normalized Eigenvalues (explained
;                         variances) of the covariance matrix belonging to the EOFs
;       local_var:  (m,n)-array containing the spatial distributions of variance
;                         from the n EOFs (each of size m)
;       eigenvalues1:(n)-vector containing the n unnormalized Eigenvalues
;                         of the covariance matrix belonging to the EOFs
;
```

```
;       m<=n:
;
;       eofs:       (m,m)-array containing the m EOFs (m rows of the array)
;                        = Eigenvectors of the covariance-matrix (spatial loading
;                        patterns, each of size m)
;       coeff:      (m,n)-array containing the n EOF-coefficients (n rows of the
;                        array) belonging to the m EOFs (time coefficients)
;       eigenvalues:(m)-vector containing the m normalized Eigenvalues (explained
;                        variances) of the covariance matrix belonging to the EOFs
;       local_var:  (m,m)-array containing the spatial distributions of variance
;                        from the m EOFs (each of size m)
;       eigenvalues1:(m)-vector containing the n unnormalized Eigenvalues
;                        of the covariance matrix belonging to the EOFs
;
;
; MODIFICATION HISTORY:
;       Written by:     Olaf Stein November 2001
;
;       12/03/02 ; replacement of PCOMP by EIGENQL
;       13/03/02 : introduction of local_var
;       January 2013 : comments revised by Terry J. Deveau
;-
;
;
; TJD: the code below is extracted from the IDL program and was used as a
; template in developing the C++ member function that follows the comments.
;
; Please note that in IDL, array indexing is ARRAY[columnIndex,rowIndex]
; but in C/C++ array indexing is ARRAY[rowIndex][columnIndex]
;
; This is especially important to understand when converting IDL code to
; C/C++ for the cases where the # and ## IDL matrix operators are being used
; as the IDL definition of these operators is based on array(columns,rows).
;
; eof_analysis, array[m,n], m, n, eofs[,], coeff[,],
;             eigenvalues[], local_var[,], eigenvalues1[]

if m gt n then begin
  eofs = dblarr(m,n)
endif else begin
  eofs = dblarr(m,m)
  array2 = array
endelse

for k=0,m-1 do begin
  mn = mean(array[k,*],/double)
  array[k,*] = array[k,*]-mn
endfor

if m gt n then begin
  a1 = dblarr(n,n)
  array2 = dblarr(m,n)
  for i=0,n-1 do begin
    for j=0,n-1 do begin
      a1[i,j] = -1.D0/double(n)
      if i eq j then a1[i,j] = 1.D0+a1[i,j]
    endfor
  endfor

  array2 = a1 ## array
  array2 = transpose(array2)

  array3 = array2 # transpose(array2)

  array3[*,*] = array3[*,*]/double(n)

endif else begin
  a1 = dblarr(n,n)
  for i=0,n-1 do begin
    for j=0,n-1 do begin
      a1[i,j] = -1.D0/double(n)
      if i eq j then a1[i,j] = 1.D0+a1[i,j]
    endfor
  endfor

  array2 = array # a1

  array3 = array2 # transpose(array2)

  array3[*,*] = array3[*,*]/double(m)

endelse

eigenvalues = EIGENQL(array3, EIGENVECTORS = eofs_small, RESIDUAL = residual,
                                                                    /double)

eigenvalues1 = eigenvalues
eigenvalues = eigenvalues/total(eigenvalues)

if m gt n then begin

  local_var = dblarr(m,n)
  eofs_small[*,n-1] = 0.D0
  array2 = transpose(array2)

  for i=0,n-2 do begin
    eofs_small[*,i] = eofs_small[*,i]/norm(eofs_small[*,i],/double)
  endfor

  eofs = eofs_small ## array2

  for i=0,n-2 do begin
    eofs[*,i] = eofs[*,i]/norm(eofs[*,i],/double)
  endfor

  coeff = transpose(eofs) # array2

endif else begin

  local_var = dblarr(m,m)

  for i=0,m-1 do begin
; TJD: the line commented out below was how it was written when I received the file
;      I commented it out and replaced it below, believing it to be a mistake
;   if norm(eofs_small[i,*],/double) gt 0 then begin
    if norm(eofs_small[*,i],/double) gt 0 then begin
      eofs[*,i] = eofs_small[*,i]/norm(eofs_small[*,i],/double)
    endif
  endfor
  coeff = transpose(eofs) # array

endelse

for j=0,m-1 do begin
  total_var = total(eofs[j,*]*eofs[j,*]*eigenvalues[*],/double)
  local_var[j,*] =  eofs[j,*]*eofs[j,*]*eigenvalues[*]/total_var
endfor


; $Id: eigenql.pro,v 1.19 2004/01/21 15:54:52 scottm Exp $
;
; Copyright (c) 1996-2004, Research Systems, Inc.  All rights reserved.
;       Unauthorized reproduction prohibited.
;+
; NAME:
;       EIGENQL
;
; PURPOSE:
;      This function computes the eigenvalues and eigenvectors of an
;      N by N real, symmetric array using Householder reductions and
;      the QL method with implicit shifts. The result is a vector
;      containing the eigenvalues.  The eigenvectors are returned
;      in a separate keyword parameter.
;
; CATEGORY:
;       Linear Algebra / Eigensystems
;
; CALLING SEQUENCE:
;       Eigenvalues = Eigenql(A)
;
; INPUTS:
;       A:    An N by N symmetric array of type float or double.
;
; OUTPUTS:
;      Function value = Eigenvalues = the computed eigenvalues, ordered
;             as specified, stored in an N element vector.
;
; KEYWORD PARAMETERS:
;    ABSOLUTE:   If set, order eigenvalues by their absolute value (magnitude),
;                otherwise sort by signed value.
;    ASCENDING:  If set to a non-zero value, eigenvalues are returned in
;                ascending order (smallest to largest). If not
;                set or set to zero, eigenvalues are returned in descending
;                order (largest to smallest). The eigenvectors are
;                correspondingly reordered.
;
;       DOUBLE:  If set to a non-zero value, computations are done in
;                double precision arithmetic.
;
; EIGENVECTORS:  The computed eigenvectors, an N x N array.  The ith
;              row, (*,i), corresponds to the ith eigenvalue. If this named
;                variable is not supplied, eigenvectors are not computed.
;
;    OVERWRITE:  If set to a non-zero value, the input array is used for
;                internal storage and its previous contents are overwritten,
;                saving memory if the original array values are no longer
;                required.
;
```

```
;     RESIDUAL:  Use this keyword to specify a named variable which returns
;                the residuals for each eigenvalue/eigenvector(lambda/x) pair.
;                The residual is based on the definition Ax - (lambda)x = 0
;                and is an array of the same size as A and the same type as
;                RESULT. The rows of this array correspond to the residuals
;                for each eigenvalue/eigenvector pair.
;                NOTE: If the OVERWRITE keyword is set to a non-zero value,
;                      this keyword has no effect.
;
; EXAMPLE:
;       Define an N by N real, symmetric array.
;         a = [[ 5.0,  4.0,  0.0, -3.0], $
;              [ 4.0,  5.0,  0.0, -3.0], $
;              [ 0.0,  0.0,  5.0, -3.0], $
;              [-3.0, -3.0, -3.0,  5.0]]
;
;       Compute the eigenvalue/eigenvector pairs.
;       The resulting array has 5 columns and 4 rows.
;
;     Eigenvalues = EIGENQL(a, EIGENVECTORS=evecs, RESIDUAL = residual)
;
;     PRINT, Eigenvalues
;        12.0915       6.18661      1.0000       0.721870
;     PRINT, evecs
;        -0.554531    -0.554531    -0.241745     0.571446
;         0.342981     0.342981    -0.813186     0.321646
;         0.707107    -0.707107 -2.58096e-08      0.00000
;         0.273605     0.273605     0.529422     0.754979
;
;       The accuracy of each eigenvalue/eigenvector (lamda/x) pair may be
;       checked by printing the residual array. This array is the same size
;       as A and the same type as RESULT. All residual values should be
;       floating-point zeros.
;
;     print, residual
;
; MODIFICATION HISTORY:
;           Written by:  GGS, RSI, January 1996
;           Modified:    DMS, RSI, August 1996
;                        Added ABSOLUTE, and reorganized calling sequence.
;
;-
;
; The EIGENQL function computes the eigenvalues and eigenvectors of an n-by-n
; real, symmetric array using Householder reductions and the QL method with
; implicit shifts.
;
; This routine is written in the IDL language. Its source code can be found
; in the file eigenql.pro in the lib subdirectory of the IDL distribution.
;
; This routine depends on the computational subroutines TRIRED, TRIQL, and
; TRIQL_NOVEC. It also uses SORT, ROTATE, and TRANSPOSE, in addition to
; mundane IDL intrinsic routines.
;
; Note: If you are working with complex inputs, use the LA_EIGENQL function
instead.
;
; Syntax
; Result = EIGENQL( A [, /ABSOLUTE] [, /ASCENDING] [, /DOUBLE]
;  [, EIGENVECTORS=variable] [, /OVERWRITE | , RESIDUAL=variable] )
```

```
;
; Return Value
; This function returns an n-element vector containing the eigenvalues.
;
; Arguments
; A
; An n-by-n symmetric single- or double-precision floating-point array.
;
; Keywords
; ABSOLUTE
; Set this keyword to sort the eigenvalues by their absolute value (their
; magnitude) rather than by their signed value.
;
; ASCENDING
; Set this keyword to return eigenvalues in ascending order (smallest to largest).
; If not set or set to zero, eigenvalues are returned in descending order
; (largest to smallest). The eigenvectors are correspondingly reordered.
;
; DOUBLE
; Set this keyword to force the computation to be done in double-precision
arithmetic.
;
; EIGENVECTORS
; Set this keyword equal to a named variable that will contain the computed
; eigenvectors in an n-by-n array. The ith row of the returned array contains the
; ith eigenvalue. If no variable is supplied, the array will not be computed.
;
; OVERWRITE
; Set this keyword to use the input array for internal storage and to overwrite
; its previous contents.
;
; RESIDUAL
; Use this keyword to specify a named variable that will contain the residuals
; for each eigenvalue/eigenvector (l/x) pair. The residual is based on the
; definition Ax - (l)x = 0 and is an array of the same size as A and the same
; type as Result. The rows of this array correspond to the residuals for each
; eigenvalue/eigenvector pair.
;
; Note: If the OVERWRITE keyword is set, the RESIDUAL keyword has no effect.
;

FUNCTION EigenQL, Array, Eigenvectors=Eigenvecs, Absolute=absolute, $
             Ascending = Ascending, Double = Double, $
                Overwrite = Overwrite, Residual = Residual

  ;This function computes the eigenvalues and eigenvectors (optionally) of a
  ;symmetric array.

  ON_ERROR, 2

  Type = SIZE(Array)

  if Type[Type[0]+1] ne 4 and Type[Type[0]+1] ne 5 then $
    MESSAGE, "Input array must be float or double."

  if Type[1] ne Type[2] then $
    MESSAGE, "Input must be an N by N array."

  Symmetric = WHERE(Array ne TRANSPOSE(Array), nErrors)
  if nErrors ne 0 then $
```

```
    MESSAGE, "Input array must be symmetric."

  if N_ELEMENTS(Double) eq 0 then Double = (Type[Type[0]+1] eq 5)

  if KEYWORD_SET(Overwrite) then Eigenvecs = TEMPORARY(Array) $
  else Eigenvecs = Array

  ;Compute tridiagonal form.
  TRIRED, Eigenvecs, EigenValues, EigenIndex, Double = Double

  if ARG_PRESENT(Eigenvecs) then begin
    ;Compute the eigenvalues and eigenvectors(stored row-major).
    TRIQL, EigenValues, EigenIndex, Eigenvecs, Double = Double
  endif else begin
    ;Compute the only the eigenvalues.
    TRIQL_NOVEC, EigenValues, EigenIndex, Double = Double
  endelse

  if KEYWORD_SET(absolute) then  EigenIndex = SORT(ABS(EigenValues)) $
  else EigenIndex = SORT(EigenValues)

  if KEYWORD_SET(Ascending) eq 0 then $ ;Descending order?
    EigenIndex = ROTATE(EigenIndex, 5)  ;Reverse the indices

  EigenValues = EigenValues[EigenIndex]         ;Permute the eigenvalues

  if ARG_PRESENT(Eigenvecs) then begin
    Eigenvecs = Eigenvecs[*, EigenIndex]        ;and the eigenvectors.

    ;If input array is overwritten, then finish.
    if KEYWORD_SET(Overwrite) then RETURN, Eigenvalues

    if ARG_PRESENT(Residual) then begin ;Compute eigenvalue/vector residuals.
      if Double eq 0 then Residual = FLTARR(Type[1], Type[2]) $
      else Residual = DBLARR(Type[1], Type[2])
      for k = 0, Type[2]-1 do $
        Residual[*,k] = Array ## Eigenvecs[*,k] - $
                        EigenValues[k] * Eigenvecs[*,k]
    endif
  endif
  RETURN, Eigenvalues
END
*/

#include <stdio.h>
#include <math.h>
#include <vector>
#include "../_NumericalRecipes/nr.h"
#include "SSP_EOF.h"

static double nr_row_L2_norm( const NRMat<double> &a, const int &row )
{
	double norm = 0.0;
	int nCols = a.ncols();
	for ( int j=0; j<nCols; ++j )
		norm += a[row][j]*a[row][j];
	return sqrt( norm );
}
```

```
void SSP_EOF::eof_analysis( void )
{
	int i, j, k;
	n_evals = n_points > n_coll ? n_coll : n_points;

// This loop to subtract the mean speed at each depth from all the speeds at
// that depth is necessary only because of the short SSP tail extension that was
// done after the mean was subtracted that did not include any tail extensions.
	NRMat<double> deltaSpeed(n_coll,n_points);
	for ( k = 0; k < n_points; ++k )
	{
		double avgSpeed = 0.0;
		for ( i = 0; i < n_coll; ++i )
			avgSpeed += collSpeed[i][k];
		avgSpeed /= (double) n_points;
		for ( i = 0; i < n_coll; ++i )
			deltaSpeed[i][k] = collSpeed[i][k] - avgSpeed;
	}

	NRMat<double> a1(n_coll,n_coll);
	for ( i=0; i<n_coll; ++i )
	  for ( j=0; j<n_coll; ++j )
	    a1[i][j] = (double)( i == j ) - 1.0/(double)n_coll;

	NRMat<double> array2( 0.0, n_coll, n_points );
	for ( i=0; i<n_coll; ++i )
	  for ( k=0; k<n_points; ++k )
		for ( j=0; j<n_coll; ++j )
			array2[i][k] += a1[i][j]*deltaSpeed[j][k];

	NRMat<double> eofs_small( 0.0, n_evals, n_evals );
	for ( i=0; i<n_evals; ++i )
	  for ( j=0; j<n_evals; ++j )
	  {
		for ( k=0; k<n_points; ++k )
			eofs_small[i][j] += array2[i][k]*array2[j][k];
		eofs_small[i][j] /= (double)n_evals;
	  }
	NRVec<double> evals( n_evals );
	NRVec<double> scratchPad( n_evals );

// IDL equivalent: TRIRED, Eigenvecs, EigenValues, EigenIndex, Double = Double
	NR::tred2( eofs_small, evals, scratchPad );

// IDL equivalent: TRIQL, EigenValues, EigenIndex, Eigenvecs, Double = Double
	NR::tqli( evals, scratchPad, eofs_small );

	// get sorted index of eigenvalues in ASCENDING sequence
	NRVec<int> EigenIndex( n_evals );
	NR::indexx( evals, EigenIndex );

	// Create the rank vector for sorting in-place.
	// Also calculate the eigenvalue normalization factor in the same loop.
	NRVec<int> EigenRank( n_evals );
	double esum = 0.0;
	for( i = 0; i < n_evals; ++i )
	{
		EigenRank[EigenIndex[i]] = i;
		esum += evals[i];
	}
```

```
        eigenvalues_norm = new std::vector<double>;
        eigenvalues_unnorm = new std::vector<double>;
        // Permute the eigenvalues and eigenvectors.
        // Also normalize the permuted eigenvalues in the same loop.
        for( i = n_evals-1; i >= 0; --i )
        {
                k = EigenIndex[i];
                if ( k != i )
                {
                        double temp = evals[i];
                        evals[i] = evals[k];
                        evals[k] = temp;
                        for ( j=0; j<n_evals; ++j )
                        {
                                temp = eofs_small[i][j];
                                eofs_small[i][j] = eofs_small[k][j];
                                eofs_small[k][j] = temp;
                        }
                        j = EigenRank[i];
                        EigenIndex[j] = k;
                        EigenRank[k] = j;
                }
// store the eigenvalues in the class member in DESCENDING sequence
                eigenvalues_norm->push_back( evals[i] / esum );
                eigenvalues_unnorm->push_back( evals[i] );
        }

        eigenvectors = new std::vector<double>[n_evals];
        double norm;
        if ( n_points > n_coll )
        {
// TJD: IDL code zeroed-out the least eigenvector. Unclear why. Not doing so here.
// In the IDL code, rotation had already been performed. Here it
// hasn't yet, so the least is first here, not last, until later.
//              for ( k=0; k<n_evals; ++k )
//                      eofs_small[0][k] = 0.0;
//              for ( i=1; i<n_evals; ++i )
                for ( i=0; i<n_evals; ++i )
                {
                        norm = nr_row_L2_norm( eofs_small, i );
                        if ( norm > 0.0 )
                                for ( j=0; j<n_evals; ++j )
                                        eofs_small[i][j] /= norm;
                }

                NRMat<double> eofs( 0.0, n_evals, n_points );
                for ( i=0; i<n_evals; ++i )
                        for ( k=0; k<n_points; ++k )
                                for ( j=0; j<n_evals; ++j )
                                        eofs[i][k] += eofs_small[i][j]*array2[j][k];

// TJD: IDL code zeroed-out the least eigenvector. Unclear why. Not doing so here.
//              for ( i=0; i<n_evals-1; ++i )
                for ( i=0; i<n_evals; ++i )
                {
                        norm = nr_row_L2_norm( eofs, i );
                        if ( norm == 0.0 ) norm = 1.0;
                        for ( k=0; k<n_points; ++k )
// store the eigenvectors in the class member in DESCENDING eigenvalue sequence
```

```
                           eigenvectors[n_evals-1-i].push_back( eofs[i][k] / norm );
              }
       }
       else
       {
              for ( i=0; i<n_evals; ++i )
              {
// TJD: the lines below differ from the IDL code, because I believe
// the IDL code has a mistake there. See the comments at the top.
                     norm = nr_row_L2_norm( eofs_small, i );
                     if ( norm == 0.0 ) norm = 1.0;
                     for ( j=0; j<n_evals; ++j )
// store the eigenvectors in the class member in DESCENDING eigenvalue sequence
                           eigenvectors[n_evals-1-i].push_back( eofs_small[i][j] /
norm );
              }
       }

}
```

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.) JASCO Applied Sciences 32 Troop Ave, Dartmouth, NS, Canada, B3B 1Z1	2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.) UNCLASSIFIED (NON-CONTROLLED GOODS) DMC A REVIEW: GCEC DECEMBER 2012

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

Implementation of nonlinear sensitivity measures in PASTET: Final Report

4. AUTHORS (last name, followed by initials – ranks, titles, etc. not to be used)

Deveau, T.J.

5. DATE OF PUBLICATION (Month and year of publication of document.) December 2013	6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.) 74	6b. NO. OF REFS (Total cited in document.) 5

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Contract Report

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

Defence R&D Canada – Atlantic
9 Grove Street
P.O. Box 1012
Dartmouth, Nova Scotia B2Y 3Z7

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.) W7707-098179/001/HAL
10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.) 00532	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.) DRDC Atlantic CR 2013-086

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

Unlimited

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.))

Unlimited

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.) DRDC Atlantic has been developing a research-level acoustic prediction system in support of tactical decision aids and improving operator effectiveness, the System Test Bed (STB). The Portable Acoustic Sensitivity and Transmission Estimation Tool (PASTET) has been, and is being, developed as an adjunct to the STB and extension of existing acoustic propagation modeling capabilities. The first objective of this call-up is to include in PASTET the capability for using an Empirical Orthogonal Function (EOF) analysis of sound speed profiles (SSP) to provide the basis for a nonlinear estimate of the sensitivity of transmission loss (TL) to SSP variance, using a Monte Carlo methodology. This is an extension of the present capability for linear estimation of TL sensitivity to SSP standard deviation. As a second objective, this call-up also included the capability in PASTET for a nonlinear estimate of the TL sensitivity to bathymetry variability, through a power spectrum analysis of fluctuations and a Monte Carlo methodology. A third objective, also using a Monte Carlo methodology, is to include an option to handle spatial field shifting, which finds the correlation between a reference acoustic field and its perturbed variations. In addition to the three principal objectives, this call-up is tasked with fixing a number of software defects that have previously been identified in PASTET. This report describes the work that was done during the call-up, the issues that were encountered, and the results that were produced. It includes recommendations for additional work.
14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)