Canadian Technical Report of
Hydrography and Ocean Sciences 226

2003

The CASCO4b User's Guide

by

Charles G. Hannah, Daniel R. Lynch[1], Keston Smith[1]

Ocean Sciences Division
Maritimes Region
Fisheries and Oceans Canada

Bedford Institute of Oceanography
P.O. Box 1006
Dartmouth, N.S.
Canada B2Y 4A2

[1]Dartmouth College, Hanover, N.H.

Cat. No. Fs 97-18/226E   ISSN 0711-6764

Correct citation for this publication:

Hannah, C.G., D.R. Lynch and K. Smith, 2003. The CASCO4b User's Guide. Can. Tech. Rep. Hydrogr. Ocean Sci. 226: vii+80 p.

Table of Contents

List of Figures v

List of Tables vi

Abstract/Résumé vii

1 Introduction 1

2 Care and Feeding of CASCO4b 2

2.1 Hydrodynamics Input (*.ins) . . . 3

2.2 Velocity Errors . . . 6

2.2.1 The UneXplained Velocities (*.uxv) file . . . 6

2.3 Elevation Errors . . . 7

2.3.1 The UneXplained Elevation (*.uxe) file . . . 7

2.4 Inversion Parameters . . . 8

2.5 Friction . . . 11

2.6 Vertical Grid . . . 11

3 CASCO Output 12

3.1 Perturbation Boundary Conditions . . . 12

3.1.1 The CASCO Boundary Condition file (*.cbc) . . . 12

3.2 Cost Function Information . . . 13

3.3 Final Error . . . 15

3.4 Other . . . 15

4 User Defined Subroutines 16

4.1 Friction . . . 16

4.2 Vertical Grid . . . 16

4.3 Output . . . 19

5 Examples 23

5.1 Mode A with velocity forcing . . . 23

5.2 Mode A with elevation forcing . . . 35

5.3 Mode B . . . 41

6 Model Theory and Implementation 51

6.1 Discrete Equations for the Forward Model . . . 51

6.2 Optimal Fit of Model to Data . . . 54

6.3 Gradient Descent Algorithm . . . 55

6.4 Adjoint Equations . . . 57

6.5 Covariance Matrices . . . 58

6.5.1 Elevation . . . 58

6.5.2 Velocity . . . 58
6.5.3 Boundary Conditions . . . 58
6.5.4 Construction of W_ρ . . . 59

7 Software Design and Implementation Notes **61**
7.1 Control Structure . . . 62
7.2 Data Structures . . . 62
7.3 Important Data Items . . . 67
7.4 Legacy Features . . . 68
7.5 Timing . . . 69
7.5.1 Model Time . . . 69
7.5.2 Boundary Condition Time . . . 69
7.5.3 Measurement Time . . . 71
7.6 Other . . . 73

8 Final Comments **74**

References **75**

A Source Code and Test Cases **76**

List of Figures

1 Schematic showing the major modules and the data flow in CASCO4 . . 2
2 A sample *.ins file 5
3 The first few lines of a sample *.cbc file 14
4 Definition of lin_stress. 17
5 Definition of lin_enz. 18
6 Definition of vertgrids2. 19
7 Definition of outputs2. 21
8 Definition of outputm2. 22
9 Test case geometry 25
10 Mode A truth, the steady state 27
11 Truth velocity time series for Mode A 28
12 Residual (unexplained) velocity time series after inversion 28
13 Inverse solution for Mode A with velocity forcing 29
14 Inverse solution error for Mode A with velocity forcing 30
15 Inverse solution boundary conditions 31
16 Mode A: Cost function as a function of iteration 32
17 Elevation observations (truth) for Mode A 36
18 Residual (unexplained) elevations after inversion 36
19 Mode A inverse solution with elevation forcing 37
20 Mode A inverse solution error with elevation forcing 38
21 Inverse solution boundary conditions 39
22 Cost function as a function of iteration 40
23 Mode B truth 44
24 Mode B truth. Elevation timeseries at the 6 sampling locations 45
25 Mode B truth. Velocity timeseries at the 6 sampling locations 45
26 Mode B residual (unexplained) elevation after inversion 46
27 Mode B residual (unexplained) velocity after inversion 46
28 Mode B inverse solution 47
29 Mode B inverse solution error 48
30 Inverse solution boundary conditions 49
31 Cost function as a function of iteration 50
32 The structure of CASCO4b 63
33 The structure of SACO based on CASCO modules 65
34 CASCO4b genealogy 66
35 Definition of the model time in CASCO 70
36 Definition of the boundary condition time in CASCO 70
37 The relationship between model time and observation time 72

List of Tables

1 The input data stream for CASCO4b. 3
2 Input parameters related to the inversion process in CASCO4b. 10
3 List of user controlled files and subroutines 16
4 Parameters for the Mode A test case. 24
5 Parameters for the Mode B test case 41
6 Summary of Finite Element Matrices 53
7 The include files which define the data families. 64
8 Functions which return useful numbers. 73

Abstract

Hannah, C.G., D.R. Lynch and K. Smith, 2003. The CASCO4b User's Guide. Can. Tech. Rep. Hydrogr. Ocean Sci. 226: vii+80 p.

CASCO4b is an inverse model whose purpose is to determine the optimal time-varying elevation boundary conditions from velocity and elevation observations in the interior of the model domain. This is done by minimizing the misfit between the modelled and observed velocity and elevation. The model is based on a finite element implemention of the three dimensional, linear, barotropic equations of motion and the adjoint method. CASCO4b contains two important improvements over its predecessor CASCO3e: it allows for the inversion of both elevation and velocity, whereas CASCO3e only allowed for velocity observations; and 2) the coastal boundary condition (no flow normal to the coast) has been implemented. This report is a self-contained description of CASCO4b and includes a complete description of the input and output structure, three examples with the complete set of input files and plots of the output, a description of the theory, and details of the implementation in FORTRAN code.

Résumé

Hannah, C.G., D.R. Lynch and K. Smith, 2003. The CASCO4b User's Guide. Can. Tech. Rep. Hydrogr. Ocean Sci. 226: vii+80 p.

CASCO4b est un modèle inverse dont le but est de déterminer les conditions de frontière optimales et d'élévation variables dans le temps à partir des observations de vitesse et d'élévation à l'intérieur du domaine du modèle. Ceci est fait en minimisant la différence entre la vitesse et l'élévation modelées et observées. Le modèle est basé sur une implémention éléments finis des équations tridimensionnelles, linéaires et barotropiques du mouvement, et de la méthode d'adjoint. CASCO4b contient deux améliorations importantes par rapport à son prédécesseur CASCO3e: il permet l'inversion de l'élévation et de la vitesse, tandis que CASCO3e prenait en compte seulement les observations de vitesse; et 2) la condition de frontière côtière (écoulement nul normal à la côte) a été introduite. Ce rapport est une description exhaustive de CASCO4b et inclut une description complète de la structure d'entrée et de sortie, trois exemples avec l'ensemble complet des fichiers d'entrée et des sorties graphiques, une description de la théorie, et des détails de l'implémentation en code de Fortran.

1 Introduction

CASCO is an inverse model whose purpose is to determine the optimal time-varying elevation boundary conditions from velocity and elevation observations in the interior of the domain. This is done by minimizing the misfit between the modelled and observed velocity and elevation.

CASCO has 3 major components: SACO, the linearized forward model; MOODY the exact adjoint of SACO; and the code which uses these two models in an iterative fashion to construct the optimal time-varying elevation boundary conditions. This is illustrated in Figure 1.

The forward portion of the model (SACO) is a linearization of a fully nonlinear 3D circulation model (QUODDY; Lynch et al. (1996)). Its inversion is achieved by standard descent algorithms using an exact algebraic adjoint (MOODY) and strong dynamical constraints. The cost function is a weighted least squares blend of velocity mismatch, elevation mismatch and boundary elevation size, slope, and tendency. The control parameters are the open-water elevation boundary conditions. Solution is achieved in the time domain, as a complement to the frequency-domain tidal inversion (TRUXTON; Lynch et al. (1998)). The intended use of CASCO is following the removal of a best prior circulation estimate which accounts for tide, local baroclinicity and the direct response to local wind forcing. The remaining subtidal velocity signal is then inverted to provide far-field elevation forcing.

The version of CASCO described herein is CASCO4b released in November 2002. CASCO4b has two improvements over its predecessor, CASCO3e: 1) it allows for the inversion of both elevation and velocity, whereas CASCO3e only allowed for velocity observations; and 2) the coastal boundary condition (no flow normal to the coast) has been implemented. The development and applications of CASCO3e are described by Hannah and Lynch (1999), Lynch and Hannah (2001), and Lynch and Naimie (2002).

This document is intended to be a self-contained description of CASCO4b. Sections 2 - 5 provide a complete description of the input and output structure of CASCO4b and three examples with the complete set of input files and plots of the output. These should help new users to get started and help to verify that the code is working in a new environment. Sections 6 and 7 contain descriptions of the theory and implementation that should help the advanced user to modify the code. Appendix A contains a list of the files that constitute the distribution of CASCO4b.

CASCO4 is written in FORTRAN77. The code attempts to be portable. So far it has been compiled and run successfully on the following operating systems: SGI, Linux on an Intel processor, and Sun Solaris. The development of CASCO, and the distribution discussed herein, assumes the existence of a unix-like operating system: for features such as long file names, shell scripts, and tools such as awk, grep and make. As well this document assumes that the user has some experience with finite element circulation

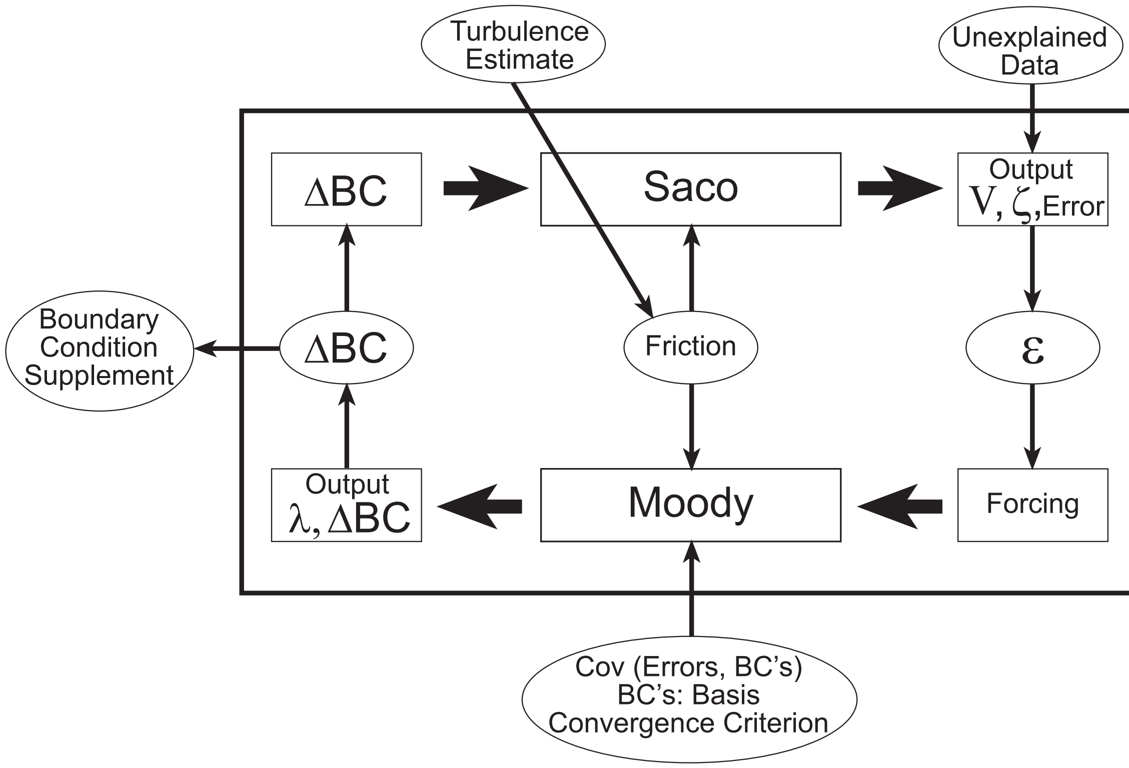

Figure 1: Schematic showing the major modules and the data flow in CASCO4. The ovals are the primary data objects and the rectangles are processes which act on the data. ΔBC are the perturbation boundary conditions and ε represents the errors (or misfits between the observations and model results.

models, in particular the QUODDY family (QUODDY4, FUNDY5, TRUXTON6) and the standard file types associated with those models.

The model names such as QUODDY, FUNDY and CASCO are taken from geographical features around the Gulf of Maine; they are not acronyms. Nevertheless we have capitalized the names because capitalizing model names seems to be the accepted convention.

2 Care and Feeding of CASCO4b

The inputs for CASCO4b fall into five primary categories:

1. The standard inputs for the hydrodynamics models, including mesh files and time information (Section 2.1);
2. The velocity errors (data) which need to be minimized (Section 2.2);

ϵ_1, ϵ_2	Convergence criteria
I_{max}	Maximum number of iterations
$W0$, $W1$, $W2$	Regularization terms
V_{noise}	Expected rms noise level for velocity (m/s)
E_{noise}	Expected rms noise level for elevation (m)
Δt_{bc}	Time step for the boundary conditions (hours)
method	Code for the descent method
file.ins	Hydrodynamics input file
file.uxv	Velocity input file
file.uxe	Elevation input file

Table 1: The input data stream for CASCO4b.

3. The elevation errors (data) which need to be minimized (Section 2.3);
4. The parameters which control the inversion process such as the expected errors, the regularization terms and the convergence criteria (Section 2.4);
5. The bottom friction and vertical eddy viscosity (Section 2.5);
6. The specification of the vertical grid (Section 2.6).

CASCO4b can be run with elevation data and velocity data together or either data type separately.

Table 1 shows the input stream required by CASCO4b, ignoring inputs required by the output routines which come after. The symbols are defined in the rest of this section. The examples in Section 5 will illustrate the complete data stream, including data required by the sample output routines.

2.1 Hydrodynamics Input (*.ins)

The basic input to the hydrodynamics models (SACO and MOODY) is the *.ins file (file.ins in Table 1), a sample of which is shown in Figure 2. In short:

- Line 2 is a comment line.
- Line 4 defines the mesh name. The required files are test6.nod, test6.ele, test6.bat, test6.lev, where test6 is the mesh name.
- Line 6 defines the name of boundary element file.
- Line 8 defines the start time of the simulation. The start time is read using a formatted read statement that allows for a single space after COLD-START and then

the date in DD MM YYYY seconds format. The FORTRAN read statement has the following format (i2,1x,i2,1x,i4,1x,f21.12). Note that all CASCO runs require a COLD-START.

- Line 10 defines the echo file where basic diagnostic information about the hydrodynamics is written.
- Lines 13 defines some scale factors which are best set to 1.0.
- Lines 14 to 17 define the latitude, the minimum depth, the simulation end time and the time step for the hydrodynamics mode. The simulation end time is read with an unformatted read statement, therefore Line 16 requires 3 integers (day, month, and year) and a floating point number (seconds). The time in the example uses the same format as required by Line 8.
- Line 18 defines the parameter THETA which controls the timing stepping algorithm in the wave equation, THETA=0.75 is a good value.
- Line 19 defines another wave equation parameter (τ_0) and the reader is referred to the literature; $\tau_0 = 0.0002$ is standard.
- Line 20 defines the number of vertical nodes.
- Line 21 defines the parameter (NLBS) which controls how bottom friction is handled. If NLBS = 0 then the linear bottom friction parameter is constant and taken as the value specified in Line 23. Otherwise the user subroutine lin_stress is called. (See Sections 2.5 and 4.)
- Line 22 defines the minimum vertical eddy viscosity.
- Line 23 defines the minimum value of the linear bottom friction coefficient.
- Line 24 defines the time weighting factor for the solution of the vertical structure of the velocity: 0.5 is a good value. In CASCO4b this variable is not used. The time weighting factor is hardcoded to be 0.5. See Section 7.4 for more details.

The lines with text inside the {}, such as {Comment:}, are comment lines that describe the data lines. They are defined by the QUODDY4 standard.

The CASCO subroutines which collect the basic information for the hydrodynamics models are derived directly from QUODDY4.1. As such the basic input file (*.ins) is based on the *.inq file from QUODDY4.1. The *.ins file consists of lines 1-17, 19,20, 23, 24, 26-28 from the QUODDY4.1 *.inq file. The interpretation of line 21 has changed.

```
1.     {Comment:}
2.     testH: forward model truth: through flow 3 day rampup in time
3.     {Case name:}
4.     test6
5.     {Boundary element incidence list:}
6.     test6.4sides.bel
7.     {Initial condition file:}
8.     COLD-START 06 04 1995 0.0
9.     {Echo file:}
10.    testH1.echo
11.    {Simulation parameters:} 10 -   20 04 1995 0.0
12.    SI UNITS            [units]
13.    1.00  1.00  1.00  [x, y, and z scaling factor]
14.    43.500            [degree latitude]
15.    10.0              [minimum depth]
16.    15 04 1995 0.0 [end date (d m y) and time (sec) of simulation]
17.    200            [time step (seconds)]
18.    0.75           [THETA]
19.    0.0002         [TAU0]
20.    11             [number of vertical nodes]
21.    0           [nonlinear bottom stress factor, NLBS]
22.    0.0100      [minimum vertical viscosity/diffusivity]
23.    0.0005      [minimum bottom stress coefficient]
24.    0.50        [time weighting factor for vertical stress/diffusion]
```

Figure 2: A sample *.ins file. The column of line numbers was added for presentation and is not part of the file format.

2.2 Velocity Errors

The CASCO convention is that errors are unexplained observations, defined as Observation - Model. At initialization the 'Model' is zero so the errors are the observations read from an external file.

The file format used for the velocity input (and output) is the *.uxv (UneXplained Velocities). The file format assumes that the velocity observations have been integrated over some vertical interval. The upper and lower limits of the vertical interval and the observed water depth is required for each data record. The format was constructed for ADCP data but current meter data can be added by assuming integration over some small interval (e.g. 1 m).

2.2.1 The UneXplained Velocities (*.uxv) file

The *.uvx file is a specialized version of the generic m3d format. The format is:

0) an arbitrary number of comment lines.
1) XXXX - the first required line
2) mesh name - e.g. test6
3) comment
4) Year - e.g. 1995
5) 8 - the number of columns
6) An arbitrary number of data records.

Each data record (a line) consists of 8 items:

- time: decimal days in UTC0 format (days from beginning of the year). This is yrday0_utc in the USGLOBEC Georges Bank Data Thesaurus.
- xobs, yobs: horizontal positions in same coordinate system as the FEM mesh. Units: meters.
- uobs, vobs: horizontal velocity in E/N coordinates. Units: m/s.
- zminobs: upper limit of the observed water column. Units: m.
- zmaxobs: lower limit of the observed water column
- zbotobs: observed water depth.

Notes:

1. The velocities are partial vertical averages, so the portion of the water column which was sampled must be part of the data stream.
2. zminobs, zmaxobs and zbotobs are defined in a coordinate system which is 0 at the surface of the ocean and positive downwards. The values are typically positive. This is the way in which observational data is typically reported. This is the opposite to the vertical coordinate used in the hydrodynamics modules (SACO and MOODY).
3. Observed water depths and model water depths are generally not the same, Therefore, CASCO uses the observed water depth to scale the model depths so that CASCO samples the same fraction of the water column as was observed.
4. The time is with respect to a particular year, where the year is given in the header. Year boundaries are not a problem. Let days in the previous year be negative and days in the following year be greater than 365 (or 366 Year is a leap year).
5. CASCO will reject data which falls outside the mesh and the time window.
6. The file format is identical to the TRUXTON input and the routines which read the file are the same. An exception is that CASCO4b stops if the mesh name in the *.uxv file is not identical to that in the *.ins file.
7. The data does not have to be sorted with respect to time, as CASCO sorts the data internally. However the output will be sorted.

2.3 Elevation Errors

For the pressure variable the decision was made to deal with elevation, the deviation from mean sea level, rather than the pressure at the bottom of the water column. The convention is again that the errors are unexplained observations, defined as Observation - Model. At initialization the 'Model' is zero so the errors are the observations read from an external file.

The file format used for the elevation input (and output) is the *.uxe file (UneXplained Elevation).

2.3.1 The UneXplained Elevation (*.uxe) file

The *.uxe file is a specialized version of the generic m3d format. The format is:

0) an arbitrary number of comment lines.

1) XXXX - the first required line

2) mesh name - e.g. test6

3) comment

4) Year - e.g. 1995

5) 4 - the number of columns

6) An arbitrary number of data records.

Each data record (a line) consists of 4 items:

- time: decimal days in UTC0 format (days from beginning of the year). This is yrday0_utc in the USGLOBEC Georges Bank Data Thesaurus.
- xobs, yobs: horizontal positions in same coordinate system as the FEM mesh. Units: meters.
- eobs: elevation observation. Units: m/s.

Notes:

1. The elevation is assumed to be the deviation about mean sea level.
2. The time is with respect to a particular year, where the year is given in the header. Year boundaries are not a problem. Let days in the previous year be negative and days in the following year be greater than 365 (or 366 Year is a leap year).
3. CASCO4 will reject data which falls outside the mesh and the time window.
4. CASCO4 stops if the mesh name in the *.uxe file is not identical to that in the *.ins file.
5. The data does not have to be sorted with respect to time, as CASCO4 sorts the data internally. However the output will be sorted.

2.4 Inversion Parameters

The inversion process requires parameters distinct from those required by the hydrodynamics. These include information on

- The expected value of the errors associated with the input velocity time series.
- The expected value of the errors associated with the input elevation time series.

- The information for construction of the regularization terms (required for an underdetermined system).
- The convergence criteria.
- The descent algorithm.

The 10 parameters required by CASCO4a (Table 1), are shown in Table 2 with more details. Further details are given below.

The cost function (Ω) which is minimized in CASCO4 is

$$\Omega = \frac{1}{2}\left\{\rho^*[W_\rho]\rho + \sum_{k=1}^{N_\delta}\delta_k^*[W_\delta]\delta_k + \sum_{k=1}^{N_\epsilon}\epsilon^*[W_\epsilon]\epsilon_k\right\} \tag{1}$$

where ρ are the boundary elevations, W_ρ contains the regularization terms, δ_k are the velocity errors, W_δ is the covariance matrix for the velocity errors, ϵ_k are the elevation errors, W_ϵ is the covariance matrix for the elevation errors,

The construction of the covariance arrays for the velocity errors and the elevation errors assumes that the errors are uncorrelated and equal. The resulting covariance matrix $[W_\delta]$ is diagonal and requires a single input, the velocity noise level V_{noise},

$$[W_\delta] = \frac{1}{N_\delta}\frac{1}{V_{noise}^2}[I] \tag{2}$$

where N_δ is the number of velocity observations (u and v pairs) and $[I]$ is the identity matrix.

Similarily the elevation errors are assumed to be uncorrelated and equal. The resulting covariance matrix $[W_\epsilon]$ is diagonal and requires a single input, the elevation noise level E_{noise},

$$[W_\epsilon] = \frac{1}{N_\epsilon}\frac{1}{E_{noise}^2}[I] \tag{3}$$

where N_ϵ is the number of elevation observations.

For CASCO4 the assumption about uncorrelated errors is part of the core code. It does not allow for a more general (i.e. nondiagonal) covariance structure. However the assumption about the equal error levels are made at the input level and can be changed if the user wants to assign different error levels to different observations. The user would be required to provide the code to bring the required data into the model and fill the covariance arrays W_δ and W_ϵ.

Problems in coastal ocean are always underdetermined with more unknowns than independent observations. As such, regularization, or penalty, terms (W_ρ) are required to control the solution The regularization used here contains terms which penalize the amplitude, slope and tendency (time derivative) of the elevation boundary conditions.

Error Fields	
V_{rms}	Expected value of the velocity error (m/s)
E_{rms}	Expected value of the elevation error (m)
Descent Algorithm	
method	1 = steepest descent
	2 = conjugate gradient
Regularization	
$W0$	$W1 \sim < \rho >^{-2}$
$W1$	$W1 \sim < \partial\rho/\partial s >^{-2} \sim (g/f)^2 V_g^{-2}$
$W2$	$W2 \sim < \partial\rho/\partial t >^{-2}$
Δt_{bc}	time step for the boundary conditions (hours)
Convergence	
ϵ_1	Convergence when $\Omega \leq \epsilon_1 \Omega_0$
ϵ_2	Convergence when $\Delta\Omega \leq \epsilon_2 \Omega$
I_{max}	maximum number of iterations

Table 2: Input parameters related to the inversion process in CASCO4b.

There is a second control on the time domain obtained by specifying a time interval for the boundary conditions (Δt_{BC}) which is greater than the time step in the hydrodynamic model. This control mechanism has two effects, it limits the time variability and it reduces the size of the solution space.

The construction of W_ρ requires the parameters $W0$, $W1$, $W2$ which penalize amplitude, slope and tendency, respectively. The scaling for these parameters are given in Table 2, where ρ is the boundary elevation, $\partial\rho/\partial n$ is the slope along the boundary, $\partial\rho/\partial t$ is the time derivative, and $< \cdots >$ is the expected value. $W1$ can be related to the expected size of boundary inflows via the geostrophic relationship: $fV_g = g\partial\rho/\partial s$, where V_g is the expected size of the boundary inflows and $\partial\rho/\partial s$ is the slope along the boundary. Thus

$$W1 \sim \left(\frac{fV_g}{g}\right)^{-2} \sim 10^{10}/V_g^2 \tag{4}$$

It may be useful to scale V_g to the observed velocity data variance, adjusted for the noise level: ${V_g}^2 \sim V_{signal}^2 - V_{noise}^2$.

In CASCO4b there are two algorithms available for finding the minimum of the cost function: steepest descent and the conjugate gradient method. The conjugate gradient method generally finds the minimum in fewer iterations and is usually preferred. The selection between the two methods is done at run time (see Table 2).

Convergence of the CASCO4b iteration is achieved when any of the following conditions is reached:

- The objective function is small relative to its initial value: $\Omega \leq \epsilon_1 \Omega_0$;

- The most recent change in the objective function is small relative to its current value: $\Delta\Omega \leq \epsilon_2 \Omega$;

- The iteration count exceeds a maximum, I_{max}.

More sophisticated convergence rules are required. One possibility is to consider the convergence of the cost due to the boundary conditions. Notice however that the cost due to the boundary conditions is not required to decrease monitonically, only the total cost (Ω) is required to do that.

2.5 Friction

There are two friction parameters in CASCO: the linearized bottom friction parameter (κ), and the vertical eddy viscosity (N_z). The bottom friction parameter can vary horizontally, $\kappa(x, y)$, and the vertical eddy viscosity can vary in all three dimensions, $N_z(x, y, z)$. In the present implementation, they are both independent of time.

In general, $\kappa(x, y)$ and $N_z(x, y, z)$ are taken from a run of the full nonlinear model, and are brought into CASCO via the user subroutines lin_stress and lin_enz. However they can also be set to the minimum values specified in the *.ins file. This is controlled as follows. If the parameter NLBS = 0 (line 21 in *.ins) then $\kappa = \kappa_{min}$, where κ_{min} is defined on line 23 of the *.ins file. If NLBS = 1 then lin_stress is called to fill $\kappa(x, y)$. The calling subroutine get_friction applies the filter $\kappa = \max(\kappa, \kappa_{min})$ after calling lin_stress. Values of NLBS other than 0 and 1 cause CASCO4 to stop.

The vertical eddy viscosity is handled in a similar fashion. The parameter CLOSURE is specified as a parameter in the file friction.h. If CLOSURE = 'CONSTANT' then $N_z(x, y, z) = N_{min}$. Where N_{min} is determined by line 22 in the *.ins file. If CLOSURE = 'SPATIAL' then $N(x, y, z)$ is specified by the user subroutine lin_enz. The calling subroutine get_friction applies the filter $N_z = \max(N_z, N_{min})$ after calling lin_enz. Values of CLOSURE other than 'CONSTANT' and 'SPATIAL' cause CASCO4 to stop.

These two user subroutines are discussed further in Section 4. The input data stream for lin_stress and lin_enz have not been dealt with. It is assumed that the users will modify the CASCO input stream to meet the need.

2.6 Vertical Grid

The vertical grid is defined by the user subroutine vertgrids2 (Section 4). The number of vertical nodes (NNV) is defined in the *.ins file and is passed to vertgrids2. If CASCO is being used in concert with a nonlinear circulation model to construct a nonlinear inverse, then perhaps the vertical grid should be taken from the nonlinear circulation model. In development, it has been simpler to use the standard subroutine sinegrid, with its controlling parameter dzbl (the distance between the lower, or upper, two nodes) specified in vertgrids2. Thus the vertical grid information does not appear in the data stream.

3 CASCO Output

The outputs from CASCO fall into four primary categories:

1. The answer, the perturbation boundary conditions which minimize the errors (Section 3.1).
2. Information about the cost function and the convergence of the solution (Section 3.2)
3. The final errors at the observation locations (Section 3.3).
4. Information about the forward and adjoint solutions (Section 3.4).

3.1 Perturbation Boundary Conditions

When CASCO4b is finished, the optimal boundary conditions are written to a file specified by the user. The format of the file is part of the CASCO specification and the file is written out without regard to the user subroutines.

3.1.1 The CASCO Boundary Condition file (*.cbc)

The file for CASCO boundary condition output (*.cbc) contains time series for the elevation at the boundary nodes of the mesh, along with information on the start time, the time step for the timeseries, and the total number of time points in the file. The unit of elevation is metres. A sample *.cbc file is shown in Fig. 3

line 1: CASENAME

line 2: comment line

line 3: Start time for the timeseries, in Human Units: day month year seconds (3 integers, one real, all separated by one blank). Seconds is the elapsed time since the day began.

line 4: number of time points, boundary condition time step Δt_{bc}

line 5: number of boundary nodes

line 6: to the end of the file: One line per datum. Each line contains the following: time step #, boundary node number, corresponding global node number, elevation. Data are grouped by time step i.e. the first set of lines gives all boundary conditions at time 1, the next set of lines is time 2, etc.

For an array F(i,j) with
i=boundary node number, i=1,nr
j=time index, j=1,nt
ig(i)=global node number corresponding to boundary node i
there will be nt*nr lines, written by

```
do j=1,nt
do i=1,nr
write (*,*) j, i, ig(i), F(i,j)
enddo
enddo
```

Note that the duration of the timeseries is (nt-1)*(dt).

3.2 Cost Function Information

Let Ω_{bc}, Ω_v, Ω_e be the costs associated with the associated with the boundary conditions, the velocity errors, and the elevation errors respectively. Then the total cost $\Omega = \Omega_{bc} + \Omega_v + \Omega_e$, and from (1)

$$\Omega_{bc} = \{\rho^*[W_\rho]\rho\} \tag{5}$$

$$\Omega_v = \sum_{k=1}^{N} \delta_k^*[W_\delta]\delta_k \tag{6}$$

and

$$\Omega_e = \sum_{k=1}^{N} \epsilon_k^*[W_\epsilon]\epsilon_k \tag{7}$$

The factor of 1/2 in the formal definition of the cost function has been ignored in CASCO4b output as it was in CASCO3e.

The rms velocity error V_{error} is defined from

$$V_{error}^2 = \sum_{k=1}^{N} \delta_k^*\delta_k \tag{8}$$

and the weighted rms velocity error is $\Omega_v^{1/2}$. The rms elevation error E_{error} is defined from

$$E_{error}^2 = \sum_{k=1}^{N} \epsilon_k^*\epsilon_k \tag{9}$$

```
bank150
perturbation bcs:  dbc_best(nr,ntbc)
           10 04 1990 3600.0
           25 3600.000
           116
           1           1         1083   -2.0237034E-03
           1           2         1046   -4.8881918E-03
           1           3         1012   -7.2865048E-03
           1           4          978   -2.6548486E-02
           1           5          943   -2.2535568E-02
           1           6          907   -2.5906080E-02
           1           7          872   -7.0271669E-03
           1           8          838   -2.5983905E-02
           1           9          802    8.2576023E-03
           1          10          765   -7.2515397E-03
           1          11          730    3.5073619E-02
etc.
```

Figure 3: The first few lines of a sample *.cbc file. The mesh is bank150 with 116 boundary nodes. The timeseries comprises 25 1-hour points, which therefore lasts 24 hours, starting at 0100 hours on April 10, 1990:

and the weighted rms velocity elevation is $\Omega_e^{1/2}$.

A standard file has not been created to hold the information about the cost function and convergence criteria and the end of each iteration. To retain flexibility and make it easy to change our mind about the convergence criteria, the information about the cost function and the convergence criteria is written to standard output. If the standard output is saved to a file, it is easy to extract the required information using the unix utility `grep`.

Assume that the standard output has been saved to the file `log`. Then a list of the cost function at the end of each iteration can be retrieved by `grep COST: log`. The output has 7 columns: the string 'COST:', the iteration count, Ω, $\Omega_v^{1/2}$, V_{error}, $\Omega_e^{1/2}$, E_{error}. The first line is an information line which described the columns. The second line gives the initial values.

Information about the costs associated with the boundary conditions, the velocity misfit and the elevation misfit can be retrieved by `grep "COST-bcs COST-vel COST-elev:"`. The three numerical columns are Ω_{bc}, Ω_v and Ω_e, respectively.

Information about progress towards the convergence criteria can be obtained by `grep`

`CONV log`. There are 5 lines for each iteration. The first line, labelled `CONVER1` shows progress toward the ϵ_1 criterion (Table 2), the second column is the iteration count, the third Ω/Ω_0, and the fourth ϵ_1. The second line, labelled `CONVER2`, shows progress towards the ϵ_2 criterion, the third column is $\Delta\Omega/\Omega$, and the fourth is ϵ_2. The next three lines are information about the relative change in Ω_{bc}, Ω_v, and Ω_e which are not yet used. The sixth line reports whether convergence has been achieved.

Information about whether the iteration count has exceeded the maximum value is not reported explicitly. However the iteration count is reported.

3.3 Final Error

The final error, that portion of the observations which cannot be explained by the perturbation boundary conditions, is written to files specified by the user. This action is independent of the user subroutines. The file formats are the *.uxe and *.uxv formats.

3.4 Other

The standard output from CASCO should be saved to a file (the log file) while CASCO is running. The output contains the convergence and cost function information (see above) and lots of other diagnostic output. Both the log file and the echo file (Section 2.1) should be checked when CASCO crashes.

The user subroutines distributed with CASCO4b can write output time series of elevation and (depth-integrated) velocity for both the forward and inverse model runs. This can generate a lot of output since files are generated during each iteration.

CASCO4b writes some diagnostic information to fortran logical unit numbers without defining the file names. This results is the system generating default file names which vary between compilers.

The timing information related to the perturbation boundary conditions is written to logical unit 81 (in Gregorian data format). Information about how to map the hydrodynamic model time to the boundary condition time is written to logical unit 80.

For example the information about the measurement functionals for the velocity observations and the velocity covariance matrix (W_δ) are written to logical unit 22.

The information on the measurement functionals for the elevation observations can be write to a file by uncommented the relevant lines in `saco_basis_elev`.

Array Sizes	
CASCO.DIM	Include file that defines the array sizes
User Subroutines	
lin_stress	bottom friction
lin_enz	vertical eddy viscosity
vertgrids2	vertical nodes
outputs2	output for SACO runs
outputm2	output from MOODY runs
Auxillary Output Subroutines	
ts1_saco_write	time series of elevation and velocity
ts1_moody_write	time series of adjoint elevation and velocity
domega2_write	writes $\partial\Omega/\partial\rho$
domega1_write	writes $\partial\Omega/\partial\xi$

Table 3: List of user controlled files and subroutines

4 User Defined Subroutines

The list of files and subroutines which the user needs to control is given in Table 3. The file `CASCO.DIM` contains the definitions of the the array sizes and needs to be modified by the user. The user subroutines are defined further in the rest of this section. Some auxillary output subroutines which are part of the distribution are also listed.

4.1 Friction

As described above the user needs to specify two friction parameters: the linearized bottom friction coefficient κ and the vertical eddy viscosity N_z. These can be spatial constants or defined by the user subroutines lin_stress and lin_enz.

If NLBS = 1 then the subroutine lin_stress is called to specify $\kappa(x, y)$.

If CLOSURE = 'SPATIAL' then the subroutine lin_enz is called to specify $N_z(x, y, z)$.

Figures 4 and 5 contain the definitions of lin_stress and lin_enz. After calling lin_stress and lin_enz, CASCO3 applies the minimums given in the *.ins file.

4.2 Vertical Grid

The distribution of the vertical nodes is determined by the user subroutine vertgrids2. The subroutine is called once during the initialization phase and the number of vertical nodes (NNV) as defined in the *.ins file and is passed. vertgrids2 is defined in Figure 6.

If CASCO is being used in concert with a nonlinear circulation model to construct

```
C*************************************************************************
C Read linearized bottom friction coefficients from data files.
      SUBROUTINE lin_stress(kd,seckd,ncall,ak,akmin,nn)
         INCLUDE 'CASCO.DIM'
         integer ncall
         integer kd
         real    seckd
         real    ak(nndim)
         real    akmin
         integer nn
C
C Index range:  I=1,NN. where NN - number of horizontal nodes
C   ak(I): linear bottom friction parameter.
C   akmin: minimum value for the bottom friction parameter.
C   kd, seckd: the Gregorian day and elapsed seconds in the day
C   ncall = 0 at first call
C   ncall <>0 otherwise.
C Called once (at startup).
C
C*************************************************************************
```

Figure 4: Definition of lin_stress.

```
C*************************************************************************
C Read vertical eddy viscosity coefficients from data files.
      SUBROUTINE lin_enz(kd,seckd,ncall,enz,enzmin,nn,nnv)
C
C Problem.  Q4 (and thus SACO) use the vertical eddy viscosity
C               as a vertical element based quantity.
C           F5 (and thus TRUXTON) use a vertical node based
C               vertical eddy viscosity.
C           Thus a different en.s3r is required for the different
C           applications.  This code does not distinguish since
C           in Q4 ENZM is dimensioned ENZM(NNDIM,NNVDIM) and the
C           same reading code can serve both masters *IF* the
C           input is formatted correctly.  That is, if in Q4 applications
C           the element values are written to the first NNV-1 positions
C           and a zero is inserted at NNV.
C
         INCLUDE 'CASCO.DIM'
         integer ncall
         integer kd
         real    seckd
         real    enz(nndim,nnvdim)
         real    enzmin
         integer nn, nnv
C
C Index range:  I=1,NN, where NN - number of horizontal nodes
C               J=1,NNV, where NNV - number of vertical nodes
C   enz(I,J): vertical eddy viscosity.
C   ekmin: minimum value for the  vertical eddy viscosity.
C   kd, seckd: Gregorian day and elapsed seconds in the day.
C   ncall = 0 at first call
C   ncall <>0 otherwise.
C Called once (at startup).
C*************************************************************************
```

Figure 5: Definition of lin_enz.

```
C*****************************************************************
      SUBROUTINE VERTGRIDS2(NN,NNV,HDOWN,ZETA,Z)
C Include file for parameter statements
      INCLUDE 'CASCO.DIM'
C Fixed global variables
      INTEGER NN, NNV
      REAL HDOWN(NNDIM), ZETA(NNDIM)
      REAL Z(NNDIM,NNVDIM)
C-----------------------------------------------------------------
C This subroutine must be modified by the user to assign the vertical
C   grid under each horizontal node.  There must be assignments to
C   specify the location of the nodes in the 3-D array Z(I,J):
C
C Index range:  I=1,NN, J=1,NNV
C          NN - number of horizontal nodes
C          NNV - number of vertical nodes
C
C Z and J are positive upward:
C          Bottom:  Z=-HDOWN; J=1
C          Surface: Z= ZETA ; J=NNV
C
C  HDOWN(I) is the water depth (positive down)
C  ZETA(I)  is the elevation and should be identically zero for CASCO3.
C  Called once (at startup).
C-----------------------------------------------------------------
```

Figure 6: Definition of vertgrids2.

a nonlinear inverse, then perhaps the vertical grid should be taken from the nonlinear circulation model. However this has not been an issue during development.

4.3 Output

In the same fashion as QUODDY, SACO calls a user defined output subroutine, outputs2, at every time step, and MOODY calls outputm2. Samples of these routines are included with the distribution of CASCO4b. Brief definitions are contained in Figures 7 and 8.

When constructing the output routines the user must bear in mind that the routines are called at every time step of every CASCO iteration. This can generate a lot of data. As well, on each iteration the subroutines must either overwrite the data files from the previous iteration or create unique file names on the fly. As well, the SACO output routine (outputs2) must be able to distinguish between being inside and outside the iteration loop. The variable `casco_iter` provides this information.

The output routines provided with the CASCO4b distribution provide the option of either writing a standard set of files (with unique names) every CASCO iteration or writing nothing. outputs2 has the ability to distinguish between calls during the CASCO iteration procedure and those during the final SACO run.

A useful auxillary routine, used by the sample output routines, is ts1_write which writes time series of the basic variables (ζ, $\overline{u}$, $\overline{v}$) at specified nodes.

```
C*********************************************************************
      SUBROUTINE OUTPUTS2(ITER,KD,SECKD,DELT,MESH,FILINQ,FILICQ,
     & NN,NNV,NE,X,Y,IN,HDOWN, AK, ENZM,
     & ZETA,UBAR,VBAR,Z,U,V,ZETAOLD,UBAROLD,VBAROLD, casco_iter)
C
C This user subroutine acts as an interface for the user to output
C the results at a particular time in the simulation.
C
C Index range:  I=1,NN, J=1,NNV, K=1,NE, L=1,NEV
C   NN - number of horizontal nodes
C   NE - number of horizontal elements
C   NNV - number of vertical nodes
C   NEV - number of vertical elements (NEV=NNV-1)
C
C   KD,SECKD - Gregorian calendar day and seconds since the beginning
C     of that calendar day.
C   ITER - Iteration number for the current time step
C   DELT - Time step size in seconds
C   X(I),Y(I) - horizontal cartesian nodal coordinates
C   HDOWN(I) - horizintal nodal bathymetry
C   IN(I,K) - horizontal triangular element incidence list
C   ZETA(I) -  free surface elevation
C   UBAR(I),VBAR(I) - vertically averaged velocity
C   Z(I,J) - nodal coordinate locations in 3-D
C   U(I,J),V(I,J) - nodal values of the X and Y components of velocity.
C   ENZM(I,L) elemental values of the vertical eddy viscosities
C   ZETAOLD(I) - free surface elevation
C   UBAROLD(I),VBAROLD(I) - vertically averaged velocity
C   MESH - Name of the mesh
C   FILINQ - Name if the .inq file being used to control the simulation
C   FILICQ - Name of the .icq4 file used to initialize the simulation
C
C   ITER = 0 --> initialization purposes
C   ITER > 0 --> after each time step
C   ITER = -1 --> end of simulation
C   casco_iter - the casco iteration count. (< 0 --> casco finished)
C*********************************************************************
```

Figure 7: Definition of outputs2.

```
C*********************************************************************
      subroutine outputm2(iter,kd,seckd,delt,nn,nnv,ne,x,y,in,hdown,
     & zeta,ubar,vbar,z,u,v,zetaold,DomegaDxi,DomegaDrho,lrvs,nr,
     & nsteps,ntbc,casco_iter)
C
C Output of the adjoint variables.
c
c index range:  i=1,nn, j=1,nnv, k=1,ne, l=1,nev
c   nn - number of horizontal nodes
c   ne - number of horizontal elements
c
c   kd,seckd - gregorian calendar day and seconds since the beginning
c     of that calendar day.
c   iter - iteration number for the current time step
c   delt - time step size in seconds
c   x(i),y(i) - horizontal cartesian nodal coordinates
c   hdown(i) - horizintal nodal bathymetry
c   in(i,k) - horizontal triangular element incidence list
c   zeta(i) -  free surface elevation
c   ubar(i),vbar(i) - vertically averaged velocity
c   z(i,j) - nodal coordinate locations in 3-d
c   u(i,j),v(i,j) - nodal values of the x and y components of velocity.
c   zetaold(i) - free surface elevation
C   DomegaDxi - gradient of cost function in full time space.
C   DomegaDrho - gradient of cost function in reduced time space.
C   lrvs  - map from boundary node space to global node space
C   nr    - number of elevation boundary nodes
c   nsteps - number of hydrodynamic model time steps
C   ntbc  - number of boundary condition time steps
c   iter = 0 --> initialization purposes
c   iter > 0 --> after each time step
c   iter = -1 --> end of simulation
c   casco_iter - the casco iteration count. (< 0 --> casco finished)
C*********************************************************************
```

Figure 8: Definition of outputm2.

5 Examples

There are three examples based on the simple examples from Lynch and Hannah (2001). The first one reproduces the Mode A experiment using velocity observations for the inversion. The second one uses the same forward solution but uses elevation observations for the inversion. The third example uses the Mode B experiment from Lynch and Hannah (2001) to illustrate the use of velocity and elevation observations in the same experiment and the use of the land boundary condition.

5.1 Mode A with velocity forcing

Consider the idealized mid-latitude shelf regime, illustrated in Figure 9. The length scale is of order 10^2 km. Bathymetry is 1-D, of order 100 m, with uniform cross-shelf slope $= 1.8 \times 10^{-3}$. Horizontal resolution is uniform at 10 km. There are 10 vertical elements everywhere, with sinusoidal Δz tapering to 1 meter at top and bottom. Temporal resolution is of order 3 minutes. Open-water boundaries surround the computational domain. The vertical mixing parameters reflect linearization within a tidal regime, with mean speeds of order 10 cm/s at the bottom, 25 cm/s aloft.

Six moorings are deployed on a regular 60×40 km grid as shown; each is capable of sampling and vertically averaging the entire water column, with error. The observations at these locations will be inverted to obtain the pressure signal on the boundary.

A forward model run of SACO, forced by boundary conditions, constitutes truth. The most basic mode is along-shelf throughflow, mode A of Lynch and Hannah (2001). The SACO "truth" is shown in Figure 10. This solution was started from rest, with BC's ramped up linearly over 72 hours and held constant thereafter. The slow spinup is effectively quasi-steady, as illustrated in Figure 11. The effects of topography and friction are visible as horizontal gradients but they are not dominant; and small, damped inertial oscillations are similarly present. Effectively we have a nearly-2D (x,z) mode. The parameters for SACO are given in the upper part of Table 4.

For the inversion, the velocity noise is presumed totally uncorrelated, with RMS size $V_{noise} = 3$ cm/s. The expected boundary condition slope is 10^{-6}, compatible with the observed 10 cm/s velocity. The time scale for boundary conditions $\Delta t_{bc} = 72$ hours; and the overall length of simulation is 216 hours. Penalties for boundary condition size and tendency are zero, the latter consistent with the prior smoothing by the 72-hour time-step. These and all other case parameters are summarized in Table 4.

For this example, no noise is added to the data. The resulting inversion is a measure of the accuracy of the inversion for this case. (See Lynch and Hannah (2001) for a discussion of accuracy and precision as it relates to inverse models). The residual unexplained velocity data is reduced to a fraction of a cm/s, Figure 12. Note that the inertial oscillations are left in the residual, a consequence of the heavy temporal smoothing ($\Delta t_{bc} = 72$

Hydrodynamic Parameters (*.ins)	
mesh name	test6
bel file	test6.4sides.bel
start time (dd mm yyyy s)	06 04 1995 0.0
echo file	testH1.echo
Latitude	$43.5° N$
h_{min}	10 m
stop time (dd mm yyyy s)	20 04 1995 0.0
time step	200 s
THETA	0.75
τ_0	2.0×10^{-4} s^{-1}
NNV	11
NLBS	0
k_{min}	0.0005 m/s
N_{min}	0.0100 m^2/s
EPSN	0.5
Hidden Hydrodynamic Parameters	
Δz_{bl}	1 m
CLOSURE	CONSTANT
Inversion Parameters	
ϵ_1	0.001
ϵ_2	0.01
I_{max}	100
W_0	0.0
W_1	10^{12}
W_2	0.0
V_{noise}	0.03 m/s
E_{noise}	0.01 m
Δt_{bc}	72 hours

Table 4: Parameters for the Mode A test case.

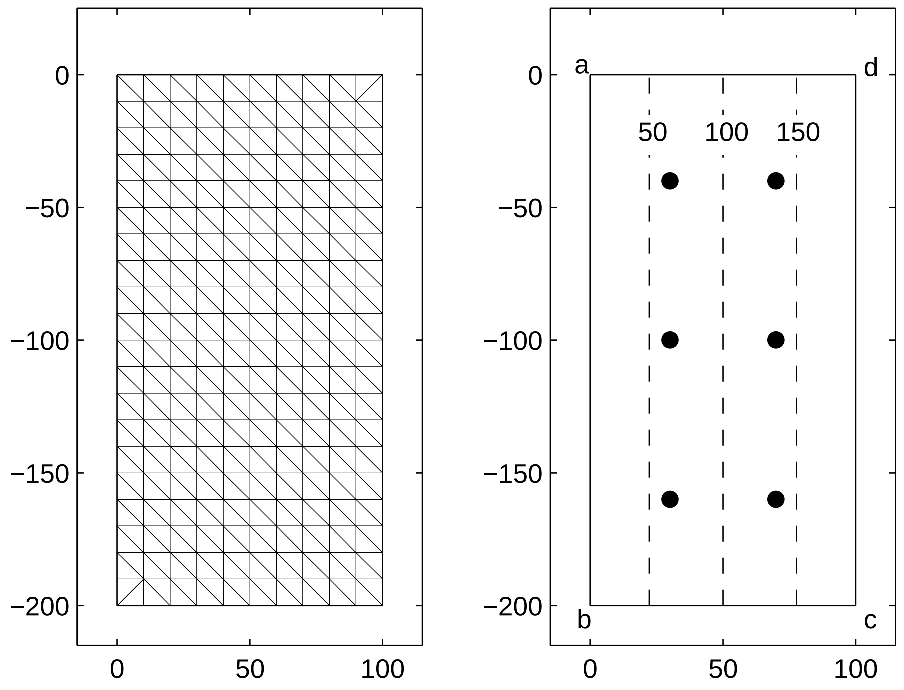

Figure 9: Test case geometry. Left: 231-node triangular mesh. Right: bathymetry (dash contours) and mooring locations (dots). Corners are indicated for later reference.

hr). Figure 13 shows the complete solution at the end of the simulation. Over much of the domain, the solution is accurate; there are notable departures. At the northeast corner, flow is incorrectly entering the domain, and at the northwest corner too much flow is entering the domain. More serious is the downstream boundary, where a nearly impermeable wall has been artificially created, the inverse solution is totally bogus there. Figure 14 shows the discrepancy with truth. Accuracy is high over much of the domain including the entrance region, and well below the anticipated noise level in the interior. The zones of obvious error are well above the noise level. Generally, interpolation accuracy is high, as is extrapolation to portions of the boundaries. But the balance of the boundary is grossly inaccurate. These are patterns of mean error or bias (Lynch and Hannah, 2001).

Examination of the true and inverse elevation boundary conditions is instructive. Figure 15 displays both at four points in time. Compared with truth, the inverse solution has a tendency to smear the corners and flatten the outflow area. The corner smearing lessens the mean square slope, while still producing the same aggregate effects at the moorings. The downstream artificial 'wall' has little dynamical interaction with the flow at the moorings, so the preferred solution is flat there, to satisfy the regularization.

The cost function is plotted in Figure 16. It is clear that this solution has reduced the BC cost below that of the true BC's via the smoothing perturbations described above, at little cost to the velocity error at the moorings. Essentially, where the BC's exert little influence over the observations, the regularization terms dominate. These BC's in turn create the most serious extrapolation errors. The cost of the elevation errors is identically zero, since there were no elevation observations.

The details of the solution in the regions with large errors are noticeably different in CASCO4b compared with CASCO3e. These changes result from the changes in the details of the descent algorithm in CASCO4b (Section 7). Since the velocity misfit is well below the specified error level, the program is clearly rummaging around trying to minimize the regularization terms.

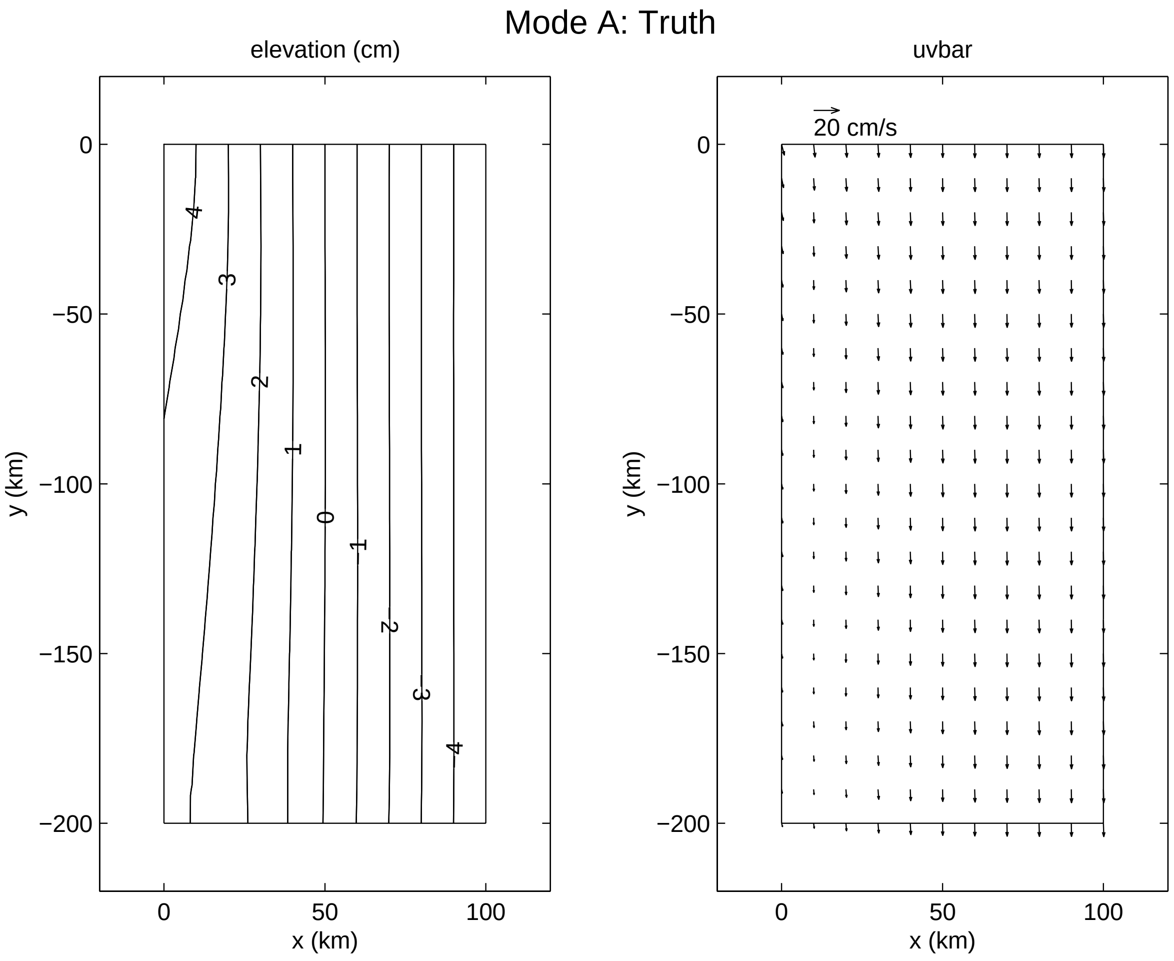

Figure 10: Mode A truth, the steady state.

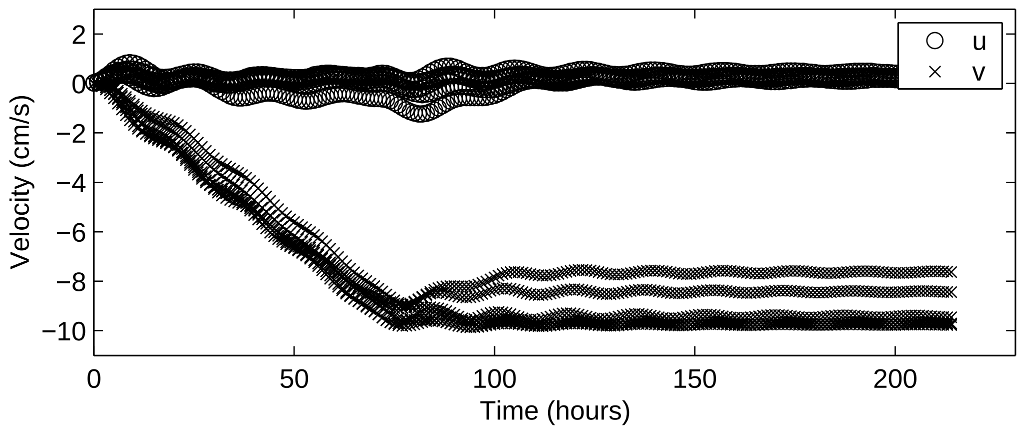

Figure 11: Truth velocity time series for Mode A at the 6 sampling stations shown in Fig. 9. Small, damped inertial oscillations are superposed upon the quasi-steady solution. The sampling is perfect, i.e. no observational error.

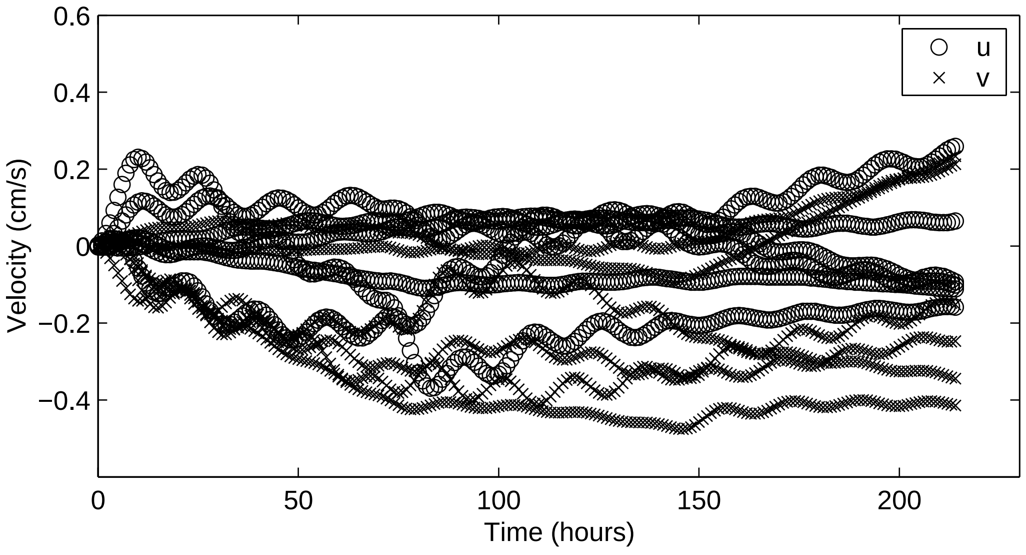

Figure 12: Residual (unexplained) velocity time series after inversion of Mode A using velocity observations.

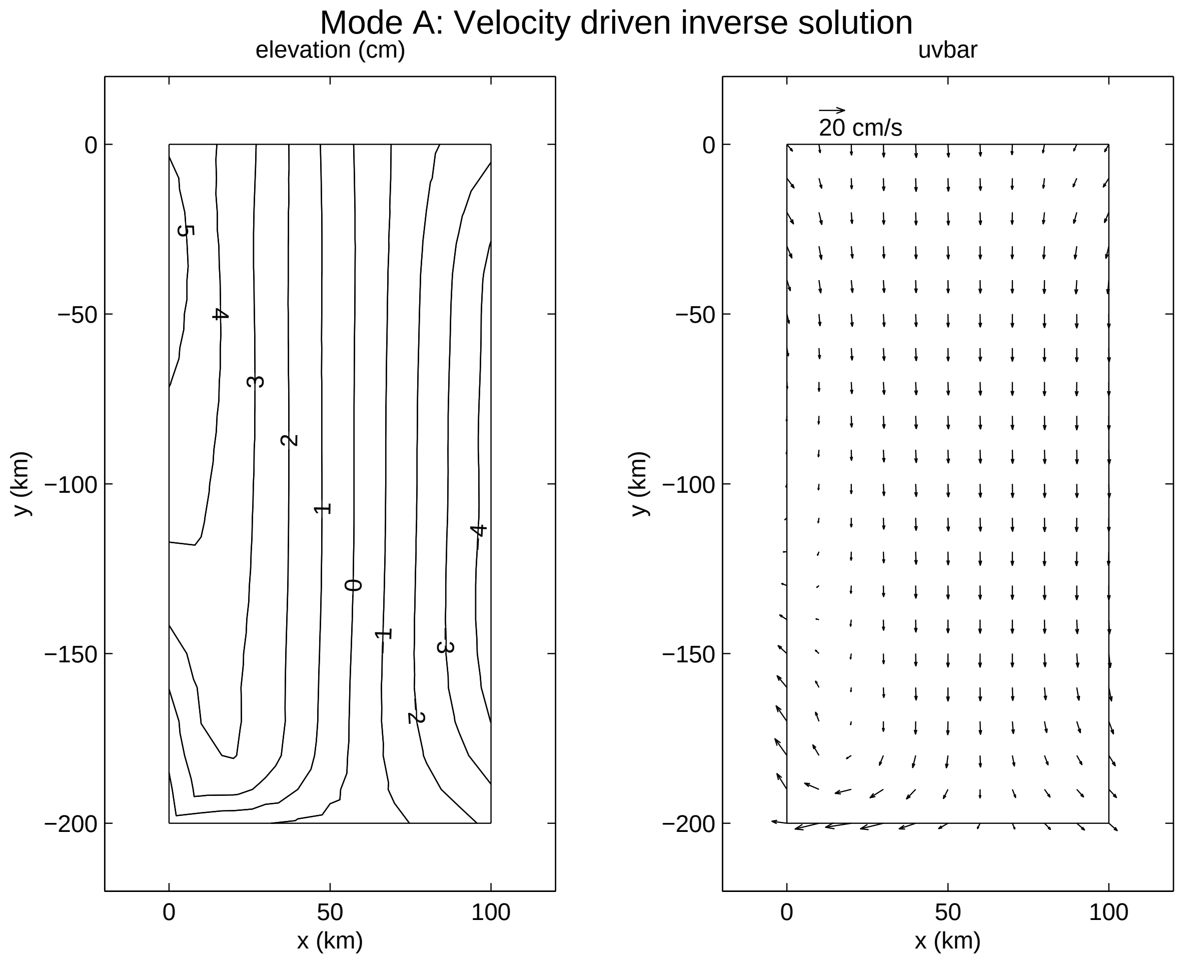

Figure 13: Inverse solution for Mode A with velocity forcing.

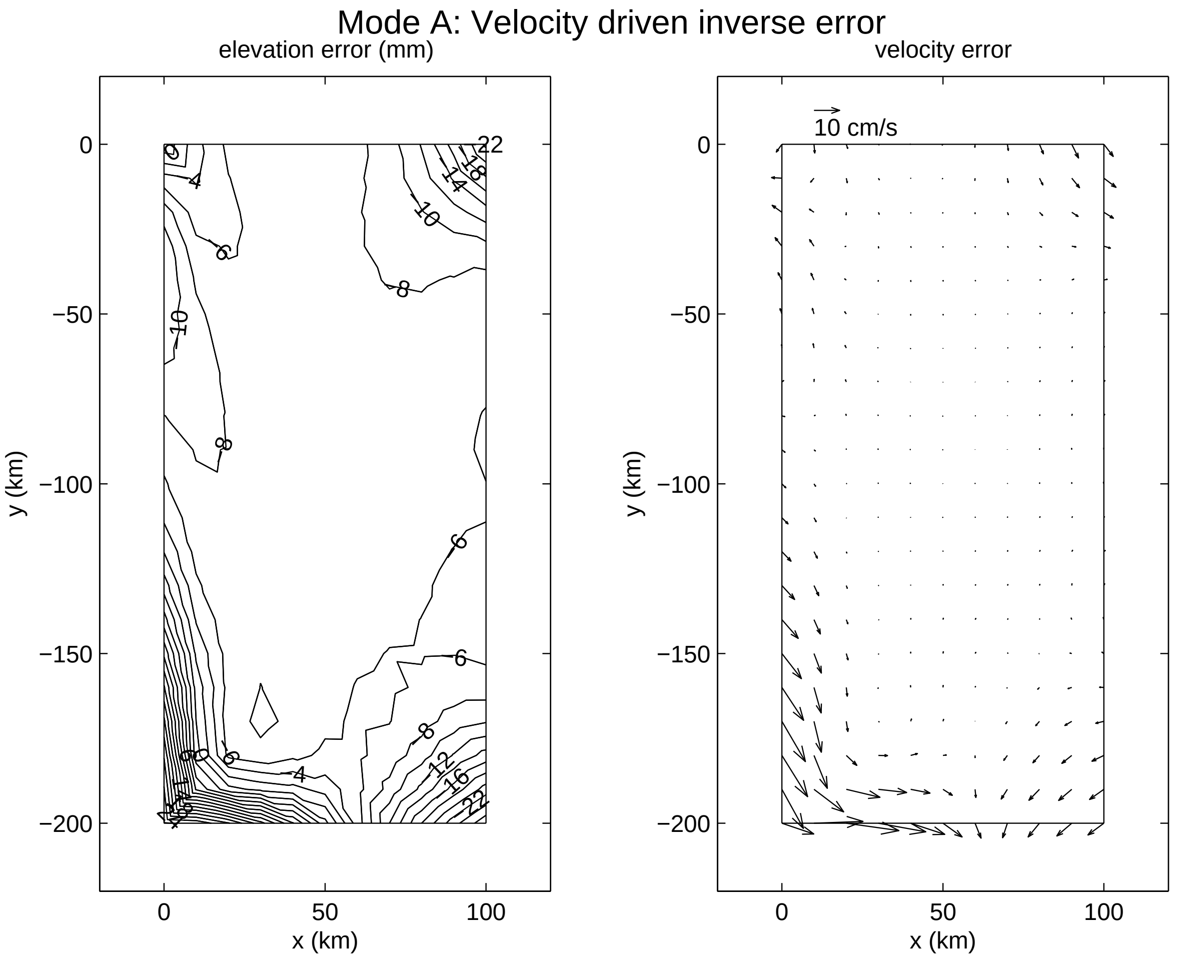

Figure 14: Inverse solution error for Mode A with velocity forcing. The difference between the 'Truth' solution (Fig. 10) and the inverse solution (Fig. 13) is plotted. Notice that the elevation error is plotted in mm whereas the solution is in cm.

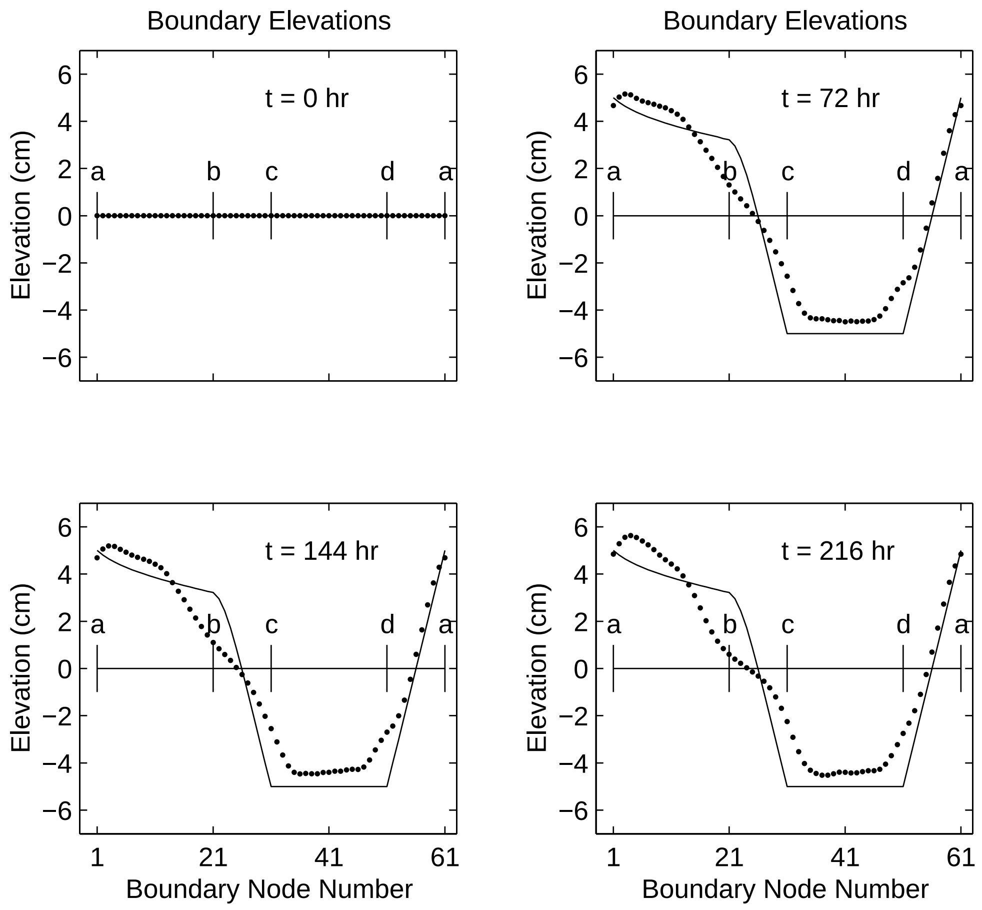

Figure 15: Inverse solution boundary conditions (dots) for Mode A with velocity forcing. Also shown are the truth BC's (line). Initial conditions are zero for both truth and inverse solutions. The a, b, c, d represent the location of the corners of the mesh as shown in Fig. 9.

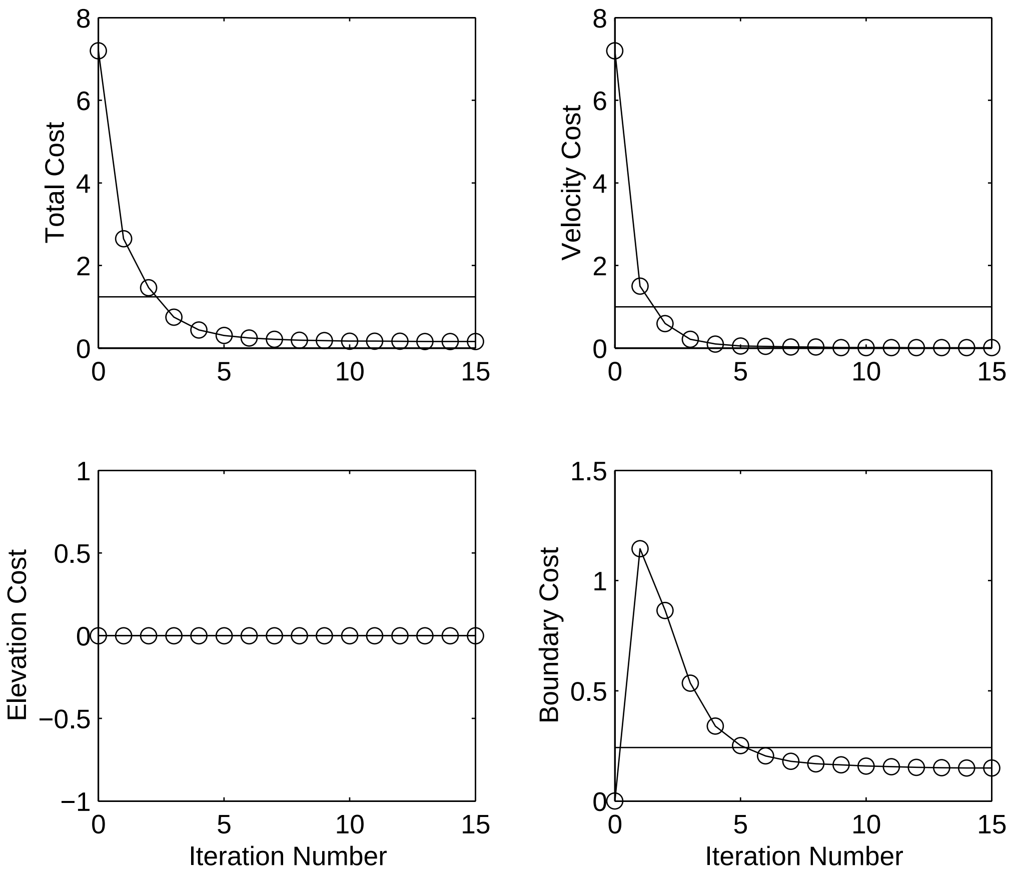

Figure 16: Cost function plotted as a function of iteration number (upper left). The components of the cost function are also shown. The horizontal lines represent the cost contributions of the true BC's and the prior estimates of velocity and elevation noise. Since there is no noise in the sampling, and a perfect inverse exists, the velocity contribution is driven toward zero. The regularization results in abandoning the true BC's in favor of smoothing, decreasing the BC cost below the truth at the expense of a small nonzero velocity cost. Since there are no elevation observations the elevation cost is identically zero.

The complete shell script for running the inversion above is given here. The script file name is `casco_ModeA_basecase_justvel` (November 2002 distribution).

```
#!/bin/csh
# script to run casco to compute the optimal elevation bcs and
# then run saco to compute the final estimate of the linear solution.

#Input files
  set ins_file = ../inputs/testH1.ins        # saco input file
  set inp_vel_error = ../inputs/inverse1.adcp_out.n0.uxv
  set inp_elev_error = none
  set samp_nodes_list = ../inputs/sample_node.list
  set samp_dt  = 1800.0      # time step for sampling nodes.

#Output files
  set cbc_final = final_bcs.cbc    # final boundary condition time series
  set final_error_casco = final_error.uxv       #final error from casco
  set final_eerror_casco = final_error.uxe       #final error from casco
  set final_error_saco  = final_error_saco.uxv  #final error from saco
  set final_eerror_saco  = final_eerror_saco.uxe  #final error from saco
  set samp_for_root     = samp_saco      #root file name - saco time series
  set samp_adj_root     = samp_moody     #root file name - moody time series

#CASCO parameters
  set cutoff1 = 0.001      # cutoff on cost function
  set cutoff2 = 0.01       # cutoff on change in cost function
  set itermax = 30         # maximum number of iterations
  set w0      = 0.0        # weight for elevations
  set w1      = 1.0E12     # weight for elevation slopes
  set w2      = 0.0        #time weight
  set bcs_tscale = 72.0    # smoothing time scale for bcs (hours)
  set exp_rms_vel_err = 0.03   # expected rms velocity error (m/s)
  set exp_rms_elev_err = 0.01  # expected rms elevation error (m)
  set method_descent = 2   # 1 --> steepest descent
                           # 2 --> conjugate gradient

#The executable program
  set casco_exe = ../src/casco4b

$casco_exe  <<  EOD
$cutoff1 $cutoff2
$itermax
$w0 $w1 $w2
$exp_rms_vel_err
$exp_rms_elev_err
```

```
$bcs_tscale
$method_descent
$ins_file
$inp_vel_error
$inp_elev_error
$samp_nodes_list
$samp_dt
$samp_adj_root
$samp_nodes_list
$samp_dt
$samp_for_root
$cbc_final
$final_error_casco
$final_eerror_casco
$samp_nodes_list
$samp_dt
$samp_for_root
$final_error_saco
$final_eerror_saco
EOD
```

The first half of the script defines the variables which are used to feed CASCO4b. The second half runs CASCO4b and feeds it the required input.

The first line specifies that the script is a c-shell script, this defines the syntax used to define the variables. Note that # defines a comment.

The three lines after the comment `#Input files` define the input files: the ins file, the uxv file, the uxe file, and a sample node list. In this case there are no elevation observations and the dummy file name `none` is used to indicate this. CASCO4b understands `none` and `NONE`. The sample node list is required by the user subroutines `outputs2` and `outputm2`, it defines the nodes where time series of elevation and depth-averaged velocity will be reported. The output is written to a *.ts1 file. The variable `samp_dt` defines the time step for the output time series.

The five lines after `#Output files` define the basic output files. The first three file names are required: the cbc file, the final error at the observation locations (from the final iteration), and the error from the final SACO run. The final errors should be identical and this provides a consistency check. The last two lines are part of the user subroutines. `samp_for_root` is the root file name for the time series output for SACO (the forward model) during the iteration process. In this case the time series output during iterations 4 and 51 (for example) will be written to the files `samp_saco.04.ts1` and `samp_saco.51.ts1`. `samp_for_root` is the root file name for the time series output for MOODY (the adjoint model). In this case the time series output during iterations 4 and 51 (for example) will be written to the files `samp_moody.04.ts1` and `samp_moody.51.ts1`.

The nine lines after `#CASCO Parameters` define the inversion parameters reported in Table 4.

The line after `#The executable program` defines the location and name of the program which will be run.

The lines after `#Run CASCO` actually run the code. The first line executes the program. The `<< EOD` is a c-shell feature which says "read the following lines, until you find the string 'EOD', and treat the lines as if they were typed in from the keyboard." The pair of EOD's can be replaced by any other string, just beware of spaces. The string $cutoff1 says 'put the value of the variable `cutoff1` here.' So the line `$cutoff1 $cutoff2` is replaced by 0.001 0.01 before being given to CASCO4b.

The first eight lines are the CASCO4b input stream with the order as given in Table 1. The next six lines supply the time series sampling information for the output subroutines. Clearly this will change as the output subroutines develop. The time series output can be suppressed by setting the root file name (`samp_for_root` and/or `samp_adj_root` to the string `none`. This requires that the next two lines in the input stream be deleted. See the output subroutines for more details.

The next two lines are the file names for the perturbation boundary conditions and the final error.

The next three lines feed the time series sampling information to the output subroutines for the final SACO run (after the iteration process is complete).

The next to last line is the file name for the final error from the final SACO run.

The final line (EOD) closes the input stream.

5.2 Mode A with elevation forcing

The second test case is the same as the first except that elevation rather than pressure is used for the observations. The shell script for running this test case is practically identical to the previous one. The script file name is `casco_ModeA_basecase_justelev`. The only differences are in the input file section, where the elevation input file has a name and the velocity input file is now `none`. The input file section of the script is shown here

```
#Input files
  set ins_file = ../inputs/testH1.ins        # saco input file
  set inp_vel_error = none
  set inp_elev_error = ../inputs/inverse1.adcp_out.n0.uxe
  set samp_nodes_list = ../inputs/sample_node.list
  set samp_dt  = 1800.0      # time step for sampling nodes.
```

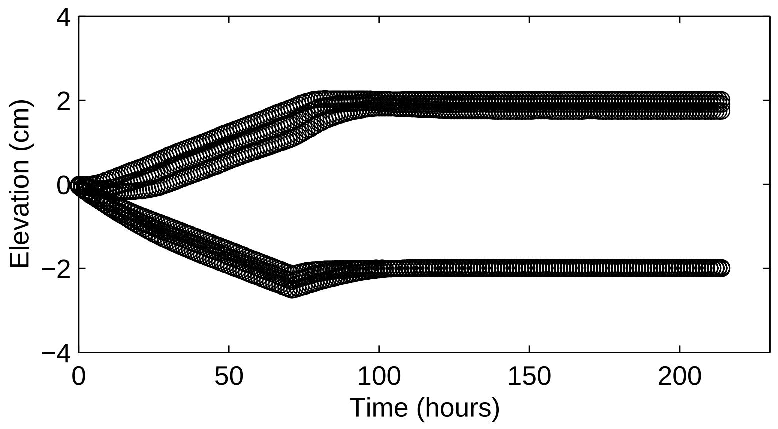

Figure 17: Elevation observations (truth) for Mode A at the 6 sampling locations (Fig. 9. The sampling is perfect, i.e. no observational error.

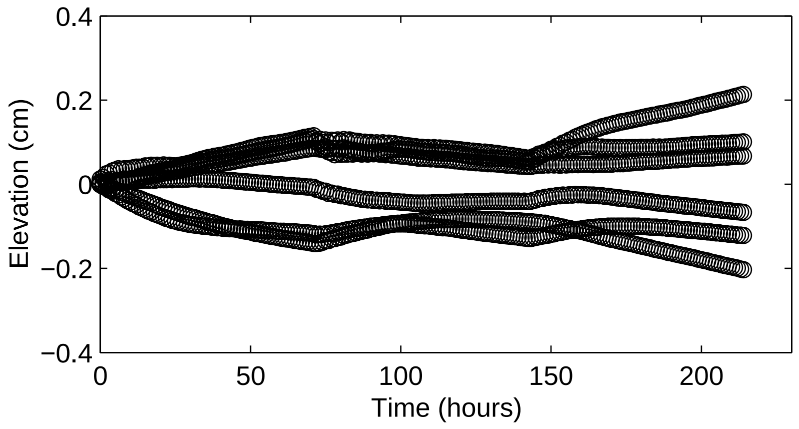

Figure 18: Residual (unexplained) elevations after inversion of Mode A with elevation forcing.

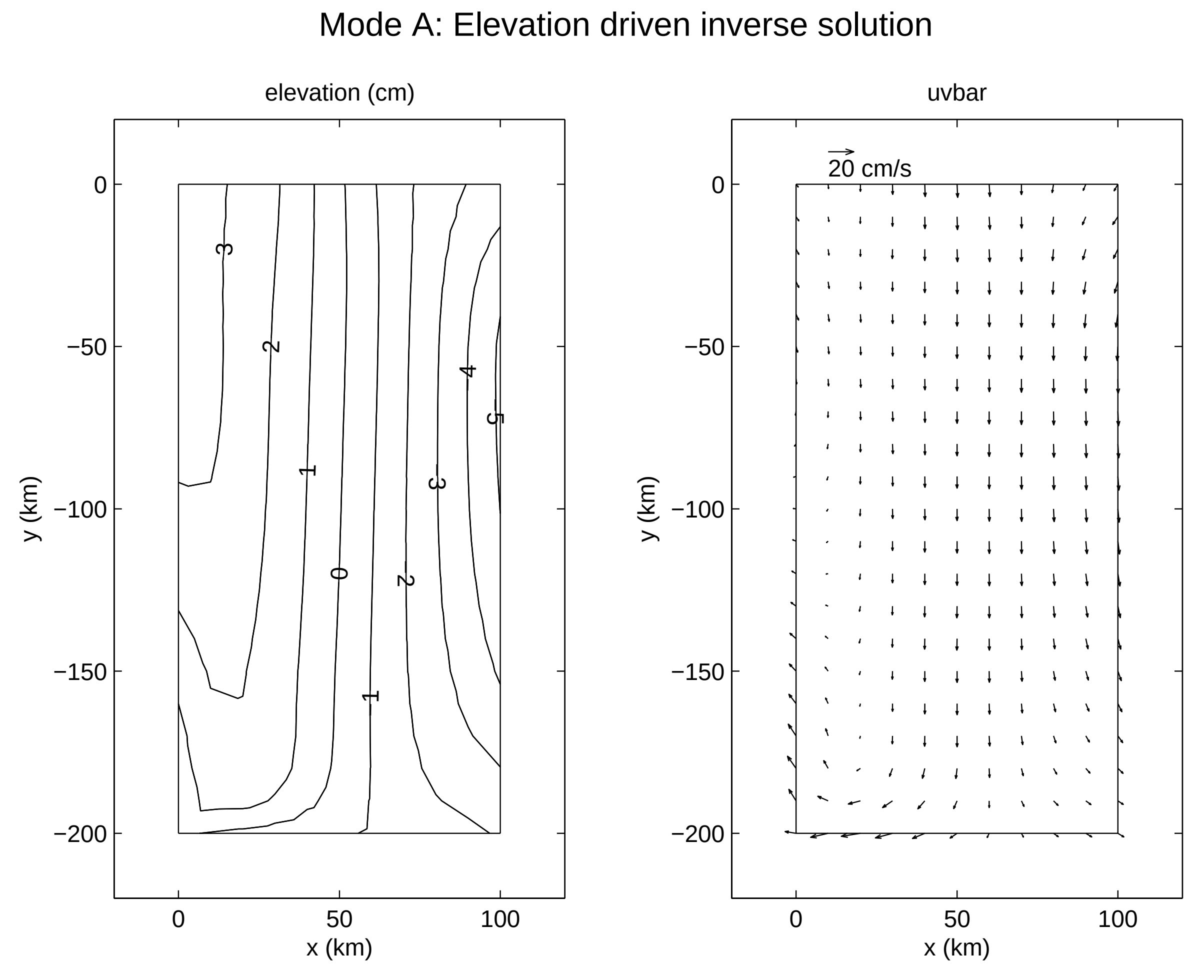

Figure 19: Mode A inverse solution with elevation forcing.

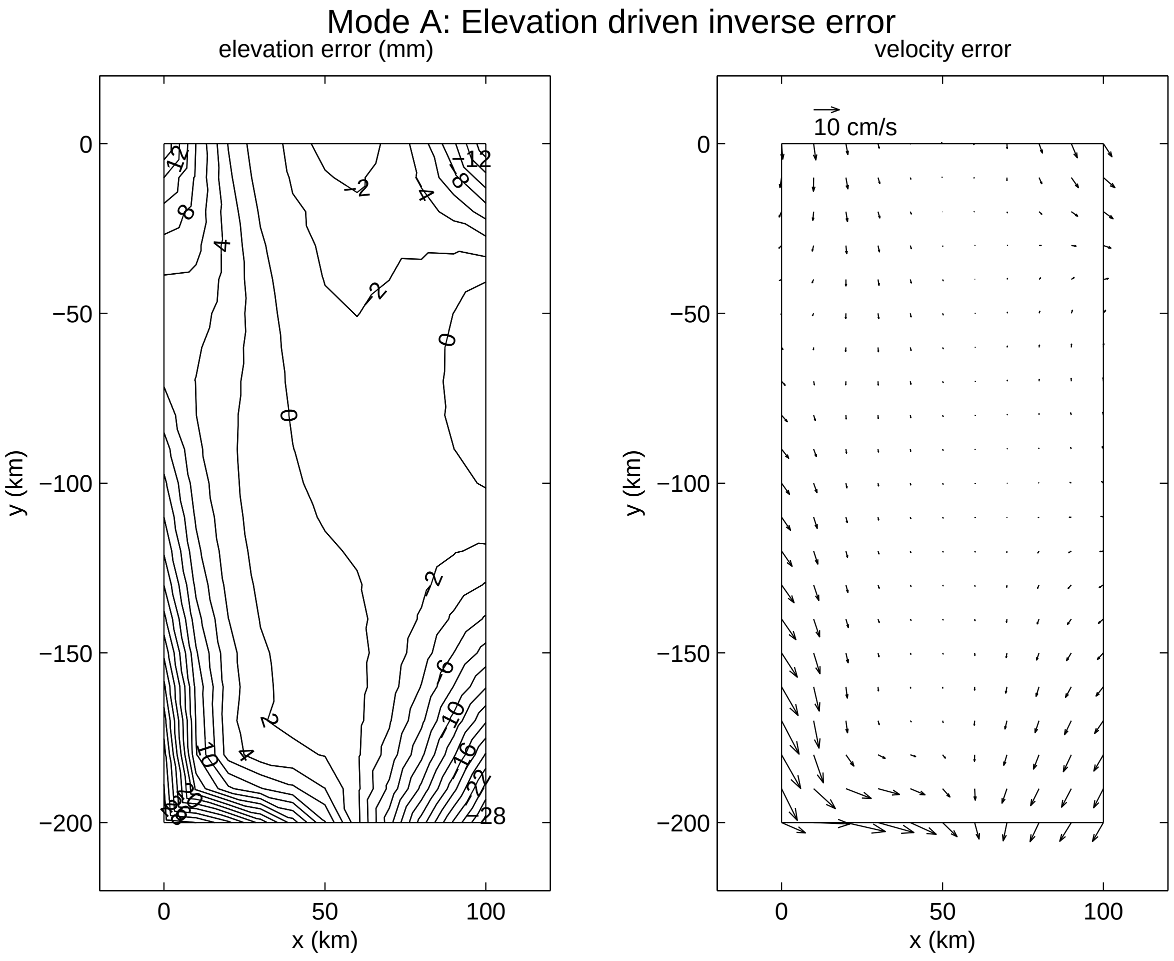

Figure 20: Mode A inverse solution error with elevation forcing. The difference between the truth and the inverse solution is plotted. Notice that the elevation error is plotted in mm (compared with cm for the solution in Fig. 19.)

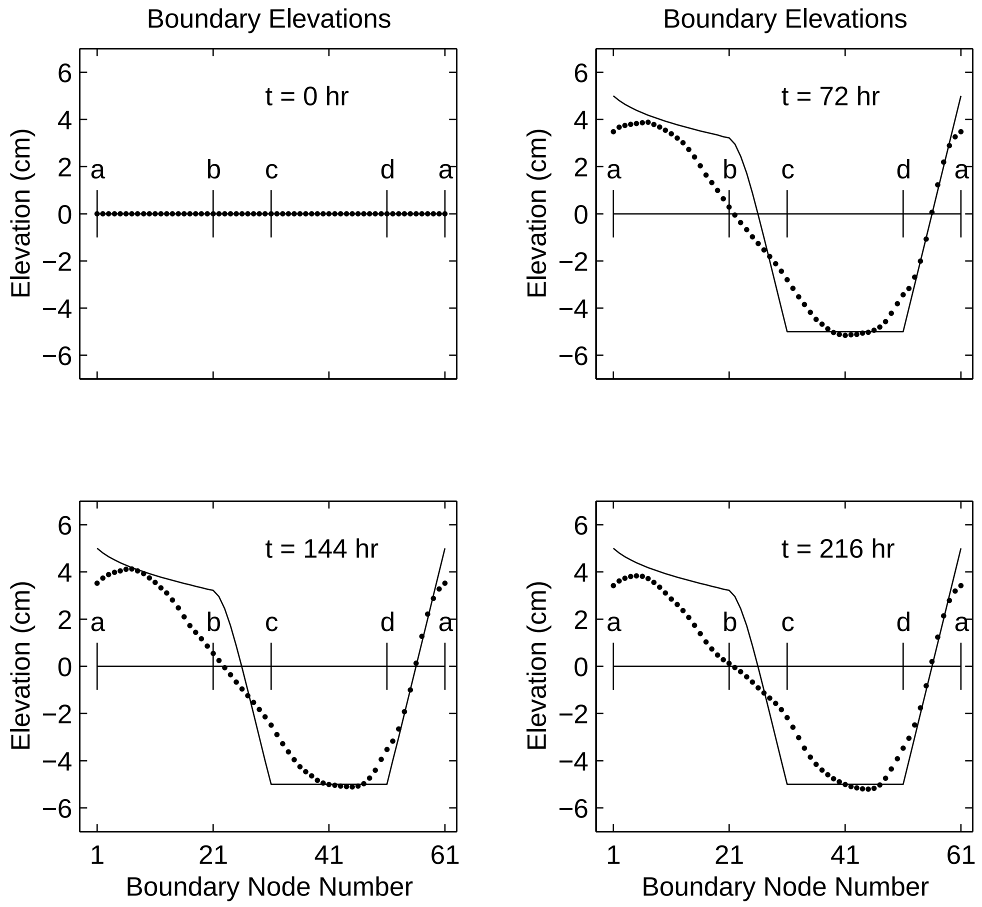

Figure 21: Inverse solution boundary conditions at four points in time. Also shown are the steady-state truth BC's. Initial conditions are zero for both truth and inverse solution. The a, b, c, d represent the location of the corners of the mesh as shown in Fig. 9.

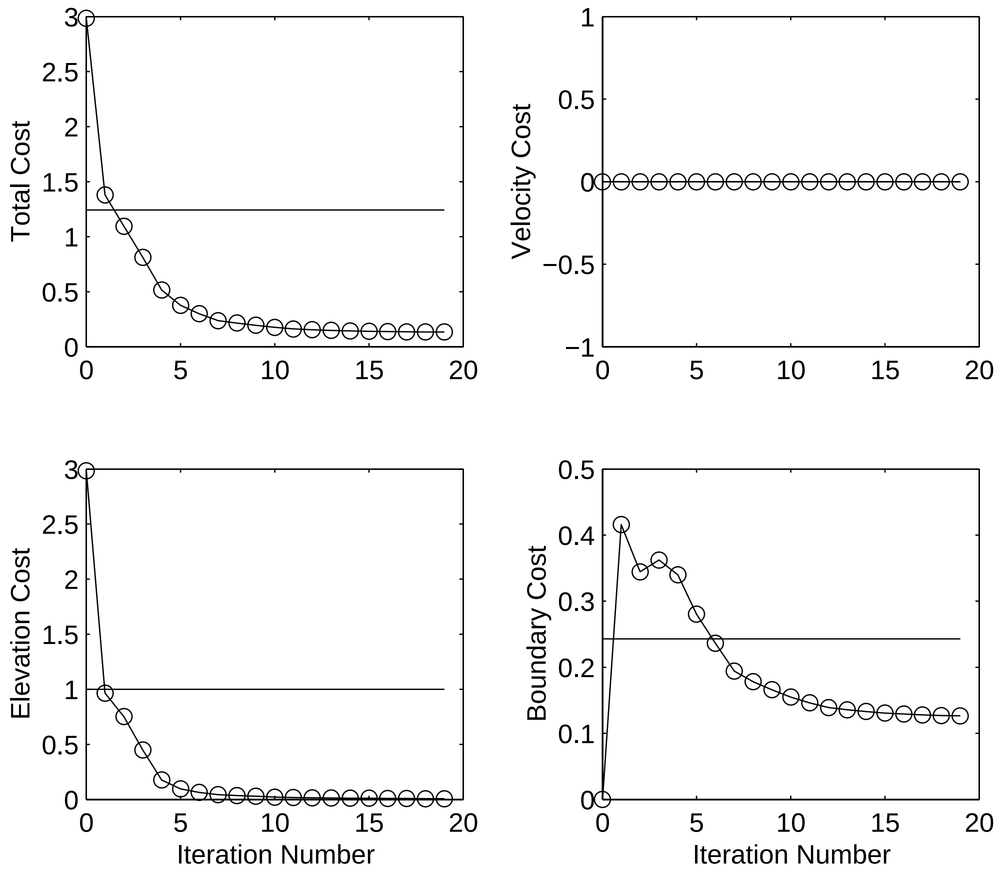

Figure 22: Cost function as a function of iteration. The horizontal bars represent the cost contributions of the prior estimate of the velocity and elevation noise and the true BC's. The velocity cost is identically zero since there are no velocity observations. Since there is no noise in the sampling, and a perfect inverse exists, the elevation contribution is driven toward zero. The regularization results in abandoning the true BC's in favor of smoothing, decreasing the BC cost below the truth at the expense of a small nonzero elevation costs.

The solution is presented without discussion. We simply note that while the solution inside the convex hull of the observation locations is quite good, the ability of the model to extrapolate elevation field outside the observation locations is worse than when the velocity observations were used.

5.3 Mode B

The final test case is Mode B from Lynch and Hannah (2001). This case demonstrates the use of both velocity and elevation observations and the land boundary condition.

The parameters for the hydrodynamic model are provided in Table 5. The only difference from Mode A test case is the boundary element (bel) file. The inversion parameters are identical to those in Table 4.

Hydrodynamic Parameters (*.ins)	
mesh name	test6
bel file	test6.3sides.bel
start time (dd mm yyyy s)	06 04 1995 0.
echo file	testI1.echo
Latitude	$43.5°N$
h_{min}	10 m
stop time (dd mm yyyy s)	20 04 1995 0.
time step	200 s
THETA	0.75
τ_0	2.0×10^{-4} s^{-1}
NNV	11
NLBS	0
k_{min}	0.0005 m/s
N_{min}	0.0100 m^2/s
EPSN	0.5

Table 5: Hydrodynamics model parameters for the Mode B test case. The only difference from Mode A test case is the boundary element (bel) file.

The complete shell script for running the inversion is given here. The script file name is `casco_ModeB_basecase_both`.

```
#!/bin/csh
# script to run casco to compute the optimal elevation bcs and
# then run saco to compute the final estimate of the linear solution.

#Input files
```

```
  set ins_file = ../inputs/testI1.ins       # saco input file
  set inp_vel_error = ../inputs/inverse1.modeB.land.uxv
  set inp_elev_error = ../inputs/inverse1.modeB.land.uxe
  set samp_nodes_list = ../inputs/sample_node.list
  set samp_dt  = 1800.0      # time step for sampling nodes.
#Output files
  set cbc_final = final_bcs.cbc    # final boundary condition time series
  set final_error_casco = final_error.uxv       #final error from casco
  set final_eerror_casco = final_error.uxe       #final error from casco
  set final_error_saco  = final_error_saco.uxv  #final error from saco
  set final_eerror_saco  = final_eerror_saco.uxe  #final error from saco
  set samp_for_root     = samp_saco      #root file name saco output
  set samp_adj_root     = samp_moody     #root file name moody output

#Casco parameters
  set cutoff1 = 0.001      # cutoff on cost function
  set cutoff2 = 0.01       # cutoff on change in cost function
  set itermax = 100        # maximum number of iterations
  set itermax = 30         # maximum number of iterations
  set w0      = 0.0        # weight for elevations
  set w1      = 1.0E12     # weight for elevation slopes
  set w2      = 0.0        #time weight
  set bcs_tscale = 72.0    # smoothing time scale for bcs (hours)
  set exp_rms_vel_err = 0.03   # expected rms velocity error (m/s)
  set exp_rms_elev_err = 0.01  # expected rms elevation error (m)
  set method_descent = 2

  set casco_exe = ../src/casco4b

$casco_exe  <<  EOD
$cutoff1 $cutoff2
$itermax
$w0 $w1 $w2
$exp_rms_vel_err
$exp_rms_elev_err
$bcs_tscale
$method_descent
$ins_file
$inp_vel_error
$inp_elev_error
$samp_nodes_list
$samp_dt
$samp_adj_root
$samp_nodes_list
$samp_dt
```

```
$samp_for_root
$cbc_final
$final_error_casco
$final_eerror_casco
$samp_nodes_list
$samp_dt
$samp_for_root
$final_error_saco
$final_eerror_saco
EOD
```

The truth solution (Fig. 23) and the standard figures are provided without discussion. See Lynch and Hannah (2001) for a discussion of the velocity only inversion of mode B.

The meeting of the land with the downstream open boundary is not as smooth as one would wish. Downstream is defined relative to shelf-wave propagation. It is hard to see but in the lower left corner of Fig. 28 and Fig. 29 there are larger than desired velocity vectors parallel to the coast. We believe that the following explanation holds. The downstream boundary does not affect the solution at the observation locations, therefore it is entirely determined by the regularization. In the solutions with four wet boundaries the regularization provides some smoothness from one boundary segment to the next. However the introduction of the land boundary on the left weakens the smoothness constraints in the lower left corner. This results in larger elevation gradients and large velocities in that corner. Clearly a better downstream boundary condition is required.

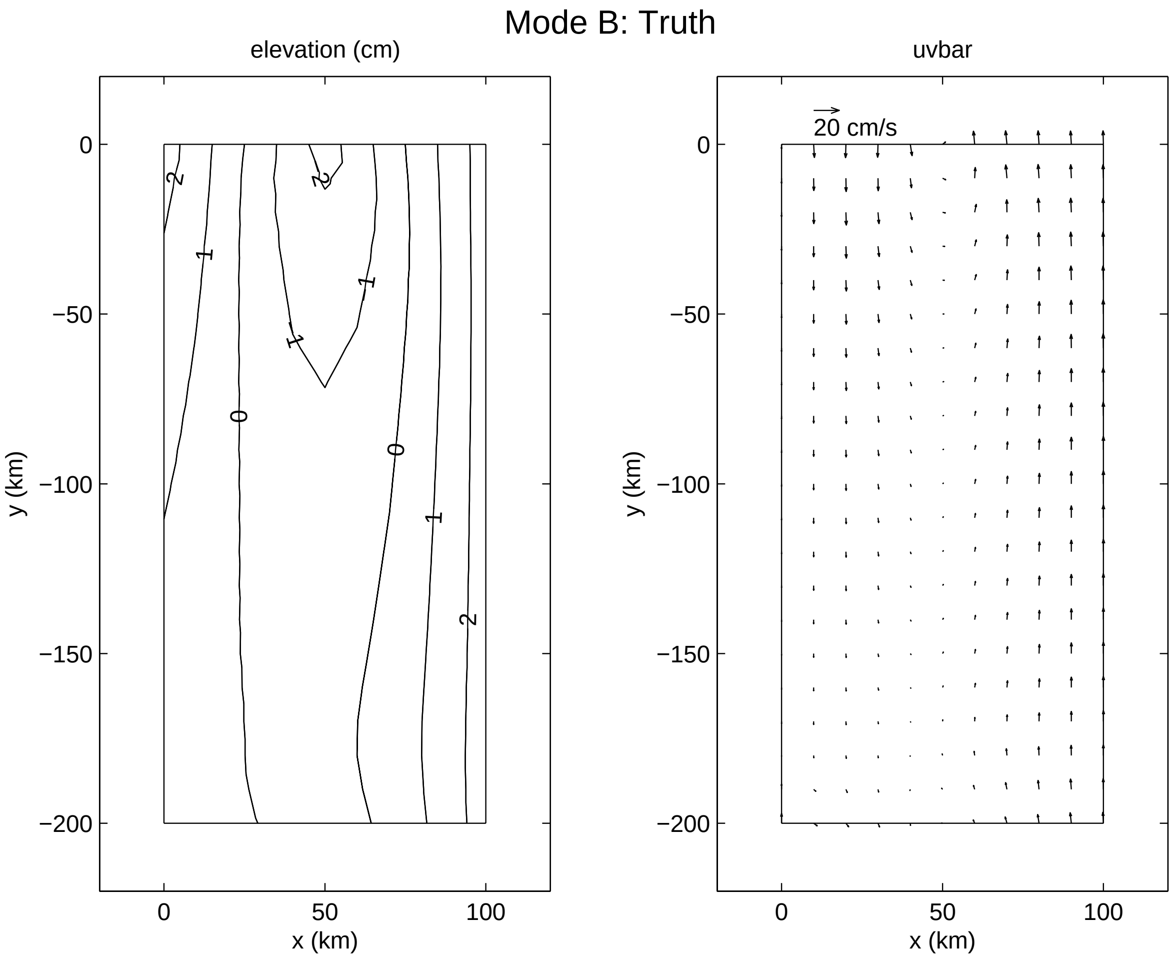

Figure 23: Mode B truth

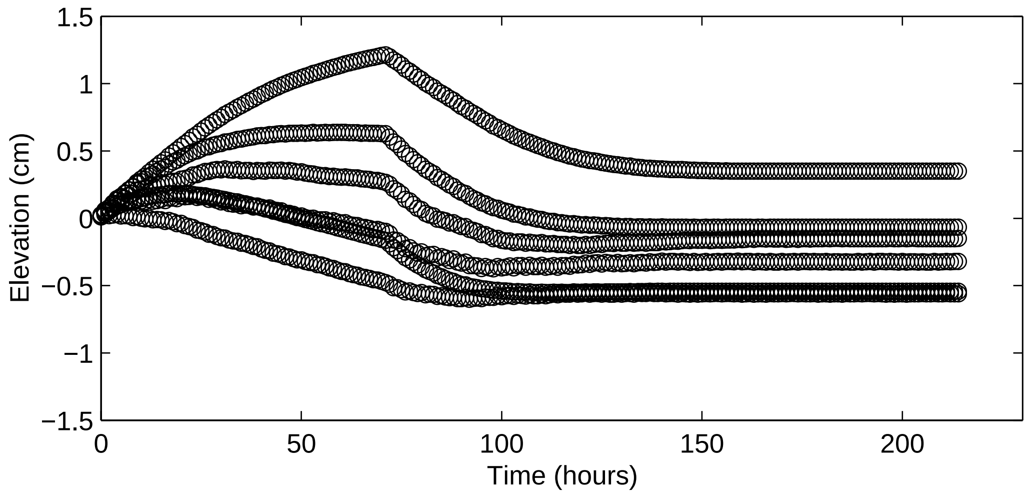

Figure 24: Mode B truth. Elevation timeseries at the 6 sampling locations shown in shown in Fig. 17. The sampling is perfect, i.e. no observational error.

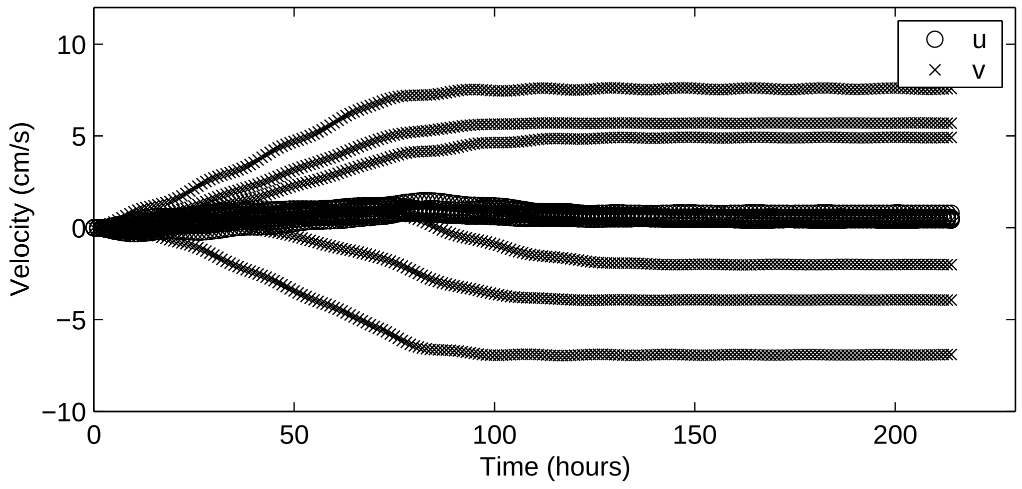

Figure 25: Mode B truth. Velocity timeseries at the 6 sampling locations shown in shown in Fig. 17. The sampling is perfect, i.e. no observational error.

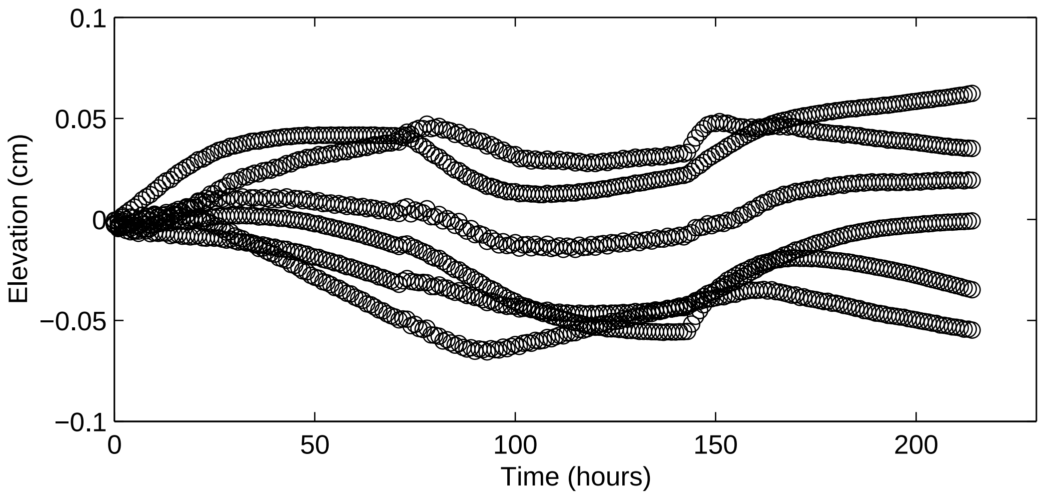

Figure 26: Mode B residual (unexplained) elevation after inversion.

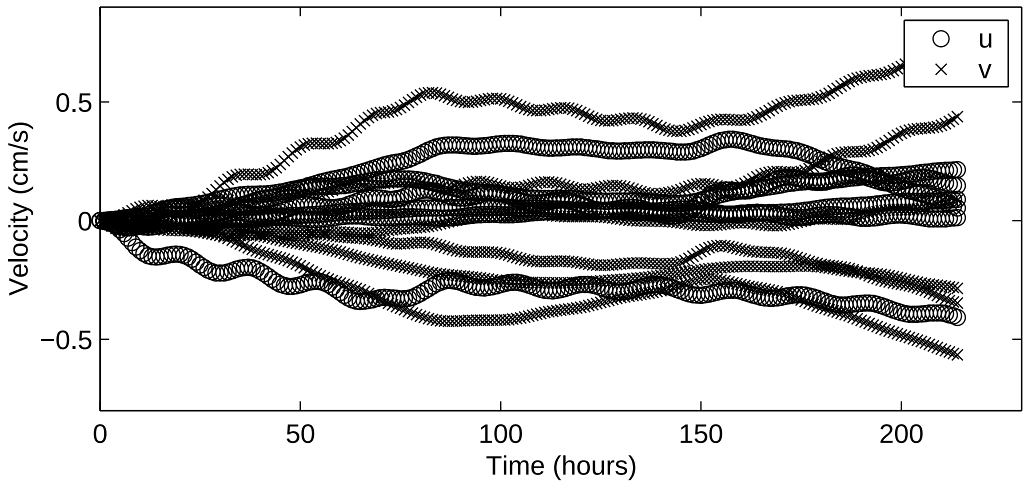

Figure 27: Mode B residual (unexplained) velocity after inversion.

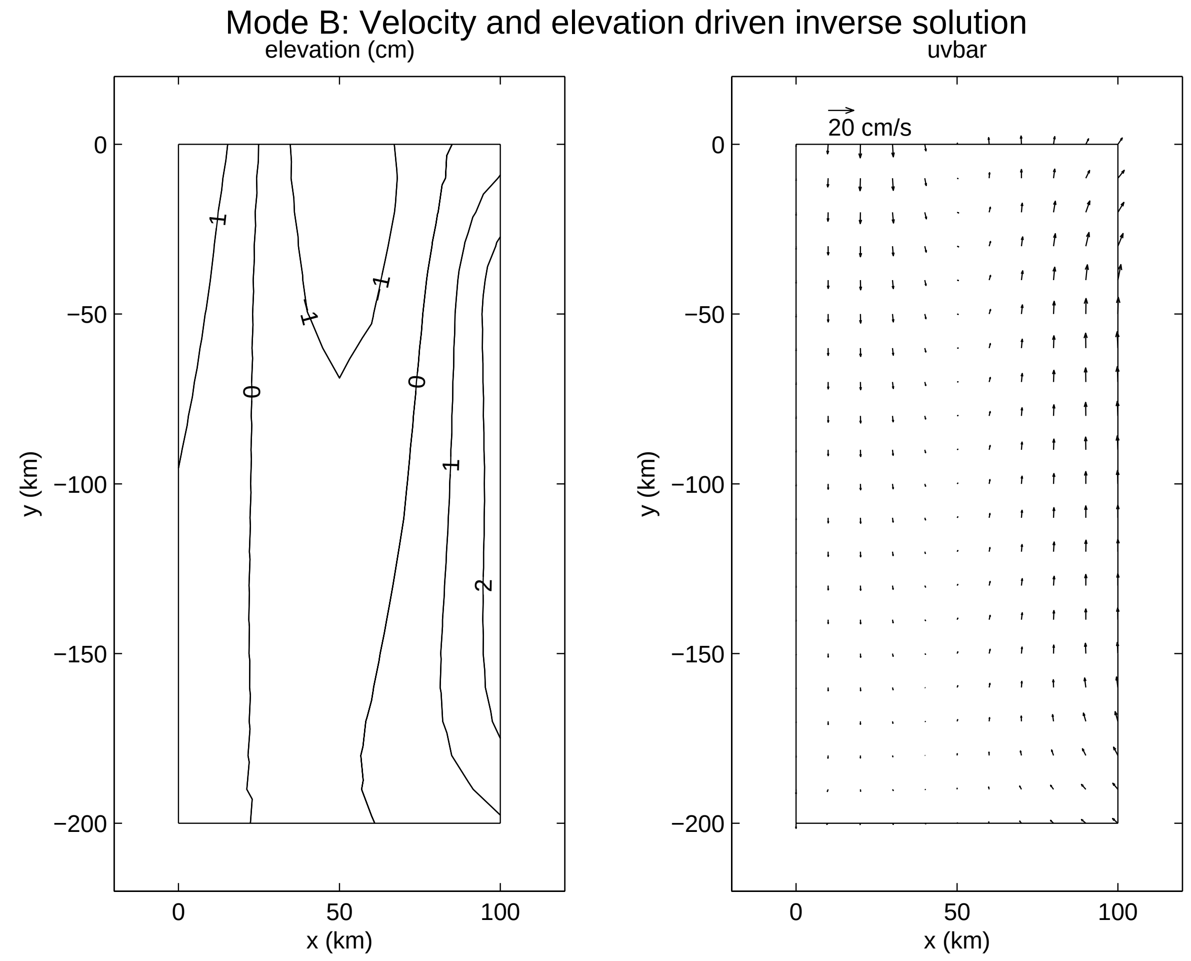

Figure 28: Mode B inverse solution

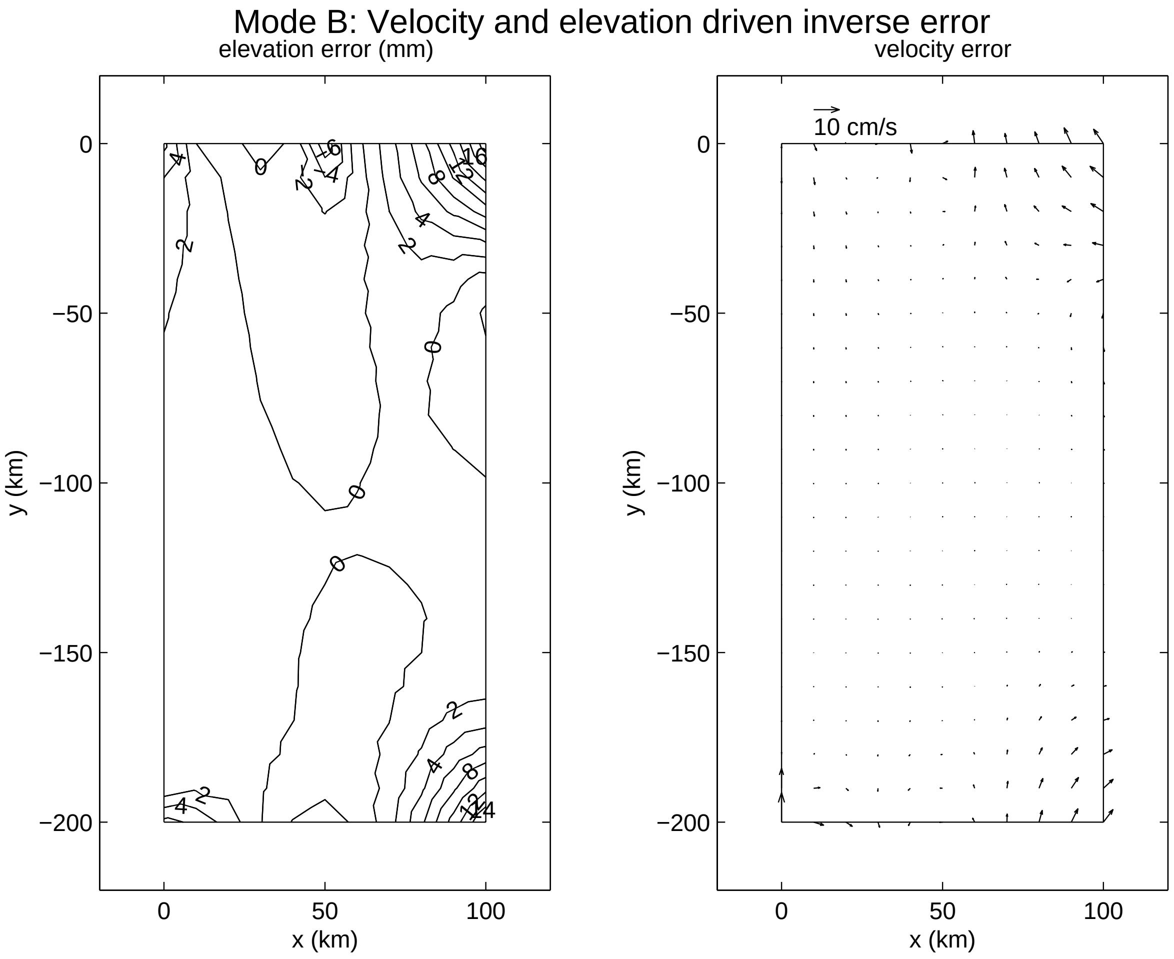

Figure 29: Mode B inverse solution error. The difference between the 'Truth' and the inverse solution is plotted. Notice that the elevation error is plotted mm (compared with cm for the solution in Fig. 28.)

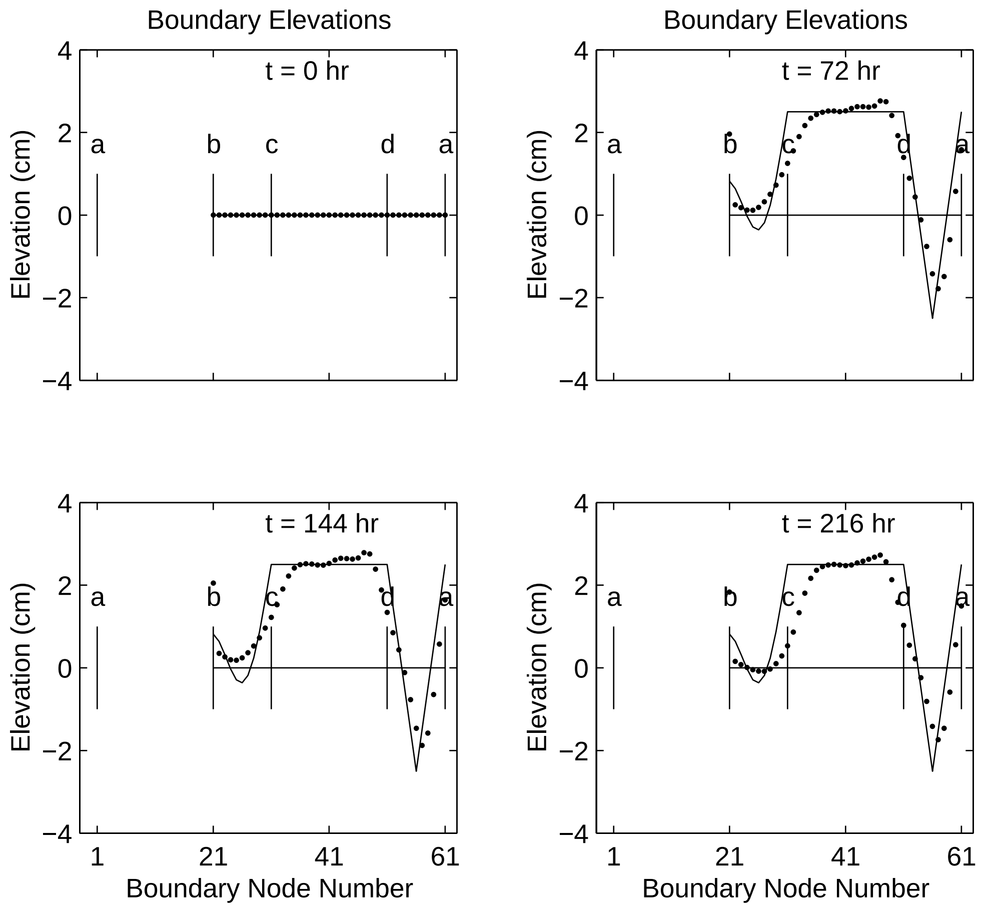

Figure 30: Inverse solution boundary conditions (dots) at four points in time. Also shown are the truth BC's (line). Initial conditions are zero for both truth and inverse solution. The a, b, c, d represent the location of the corners of the mesh as shown in Fig. 9. Elevation values are not shown for the part of the boundary that is land (a to b).

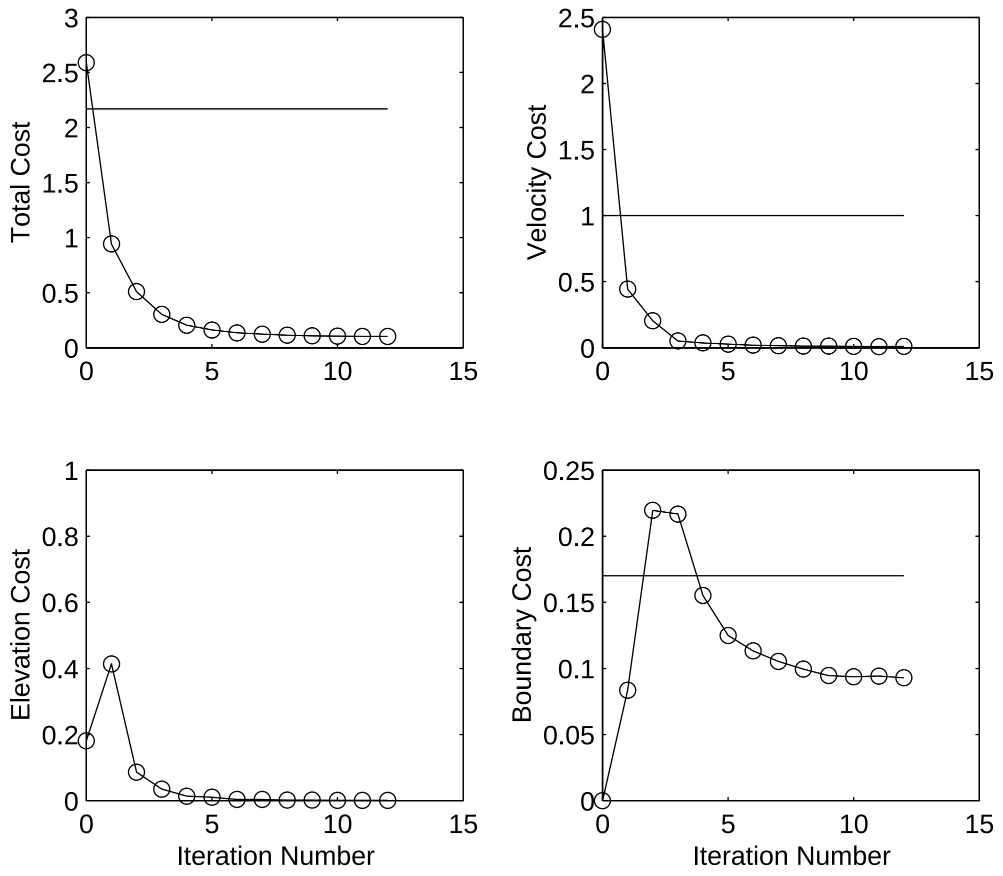

Figure 31: Cost function as a function of iteration. The horizontal bars represent the cost contributions of the the prior estimate of the velocity and elevation noise and the true BC's. The expected elevation cost is 1 and is hidden in the frame of the plot. Since there is no noise in the sampling, and a perfect inverse exists, the velocity and elevation contributions are driven toward zero. The regularization results in abandoning the true BC's in favor of smoothing, decreasing the BC cost below the truth at the expense of a small nonzero velocity and elevation costs.

6 Model Theory and Implementation

We solve the linearized 3-D shallow water equations:

$$\frac{\partial \zeta}{\partial t} + \nabla \cdot \left(h\overline{V} \right) = 0 \tag{10}$$

$$\frac{\partial V}{\partial t} + f \times V + g\nabla\zeta - \frac{\partial}{\partial z}\left(N\frac{\partial V}{\partial z} \right) = 0 \tag{11}$$

with surface and bottom boundary conditions

$$N_z \frac{\partial V}{\partial z} = 0 \quad (z = 0) \tag{12}$$

$$= \kappa V \quad (z = -h) \tag{13}$$

The vertical viscosity and linear bottom slip coefficient ($N_z(x,y,z)$, $\kappa(x,y)$) are specified exogenously. These equations are rearranged as in Lynch and Werner (1991), replacing equation 10 with

$$\frac{\partial^2 \zeta}{\partial t^2} + \tau_0 \frac{\partial \zeta}{\partial t} - \nabla \cdot \left(gh\nabla\zeta + f \times q + \kappa V_{-h} - \tau_0 q \right) = 0 \tag{14}$$

(q is the vertical integral of V). Initial conditions are known. Open water BC's are posed in terms of an unknown ζ distribution. This is adjusted to minimize velocity errors.

6.1 Discrete Equations for the Forward Model

These equations are discretized on linear finite elements (triangular in (x, y); terrain-following bilinear in z). Semi-implicit time-stepping is used, leading to a 3-level-in-time version of equation 14 for surface elevation plus a conventional 2-level momentum equation. In discrete form, we have

Initial Condition:

$$(\zeta,\ V)_{-1} = (\tilde{\zeta},\ \tilde{V})_{-1}$$
$$(\zeta,\ V)_0 = (\tilde{\zeta},\ \tilde{V})_0$$

Dynamic:

$$[M]\,\zeta_{k+1} = [A]\,\zeta_k + [B]\,\zeta_{k-1} + [C]\,q_k + [D]\,vb_k + [E_{k+1}]\,\rho + R_{k+1} \qquad k = [0, N-1] \quad (15)$$

$$[N]\,(V_{k+1} + V_k) = [F]\,V_k + [Z^*]\,[G]\,(\zeta_{k+1} + \zeta_k) + S_{k+1} \qquad k = [0, N-1] \qquad (16)$$

$$vb_k = [Y]\,V_k \qquad k = [0, N-1] \qquad (17)$$

$$q_k = [Z]\,V_k \qquad k = [0, N-1] \qquad (18)$$

Errors:

$$\delta_k = \tilde{d}_k - [P_k]\,V_k - [Q_k]\,V_{k-1} \qquad k = [1, N] \qquad (19)$$

$$\epsilon_k = \tilde{e}_k - [T_k]\,\zeta_k - [U_k]\,\zeta_{k-1} \qquad k = [1, N] \qquad (20)$$

X^* indicates the transpose of X; and $\tilde{Y}$ indicates data; and definitions are as follows:

$(\zeta,\ V)_k$: Vectors of (Elevations, Velocities) at time k (State variables in forward model)

q_k: transport vector (vertical integral of V_k)

vb_k: bottom velocity (sampled from V_k)

$(\tilde{\zeta},\ \tilde{V})_k$: IC's; *assumed known with certainty*

$(\tilde{e},\ \tilde{d})_k$: (Elevation, Velocity) observations in time window $[k-1,\ k]$

$(\epsilon,\ \delta)_k$: (Elevation, Velocity) errors (mismatch between observation and model)

$[Y]$, $[Z]$: samples 2-D fields $(vb,\ q)$ from 3-D V field.

$[P]$, $[Q]$: measurement functional; interpolates model V to space-time observation points

$[T]$, $[U]$: measurement functional; interpolates model ζ to space-time observation points

ρ: the control variables (Type I BC's);

$[E_k]$: projects ρ onto the simulation as elevation BC's at time k.

R, S: Known forcing (BC's, wind, BPG)

and the matrices are obtained with standard finite element assembly and are summarized in Table 6.

Let $W(x,y)$ be the basis for ζ and $\phi(x,y,z)$ be the basis for V_x and V_y. Let $<>$ indicate integration over x,y and $<\overline{\cdots}>$ indicate integration over the 3-d finite element. From Lynch and Werner (1991), equation 27 we have the Wave Equation matrices:

$$M_{ij} = \langle W_j\ W_i \rangle \left(1+\frac{\tau_0\Delta t}{2}\right)+\frac{\theta\Delta t^2}{2}\langle gh\nabla W_j\cdot\nabla W_i\rangle \qquad (21)$$

$$A_{ij} = 2\langle W_j\ W_i\rangle - (1-\theta)\Delta t^2\langle gh\nabla W_j\cdot\nabla W_i\rangle \qquad (22)$$

$$B_{ij} = \langle W_j\ W_i \rangle \left(-1+\frac{\tau_0\Delta t}{2}\right)-\frac{\theta\Delta t^2}{2}\langle gh\nabla W_j\cdot\nabla W_i\rangle \qquad (23)$$

$$C_{ij} = \Delta t^2\left\{\ \tau_0\left\langle W_j\tfrac{\partial W_i}{\partial x}\right\rangle - f\left\langle W_j\tfrac{\partial W_i}{\partial y}\right\rangle,\ \ \tau_0\left\langle W_j\tfrac{\partial W_i}{\partial y}\right\rangle + f\left\langle W_j\tfrac{\partial W_i}{\partial x}\right\rangle\ \right\} \qquad (24)$$

$$D_{ij} = -\Delta t^2\left\{\ \left\langle \kappa W_j\tfrac{\partial W_i}{\partial x}\right\rangle,\ \ \left\langle \kappa W_j\tfrac{\partial W_i}{\partial y}\right\rangle\ \right\} \qquad (25)$$

The Momentum Equation matrices are from LW91, equations 28-32:

$$N_{ij} = \qquad (26)$$

$$\frac{1}{2}\begin{bmatrix} \left\langle \overline{\tfrac{2}{\Delta t}\phi_j\ \phi_i + N_z\tfrac{\partial\phi_j}{\partial z}\tfrac{\partial\phi_i}{\partial z}} + \kappa\left(\phi_j\phi_i\right)|_{z=-h}\right\rangle & -f\left\langle\overline{\phi_j\ \phi_i}\right\rangle \\ f\left\langle\overline{\phi_j\ \phi_i}\right\rangle & \left\langle \overline{\tfrac{2}{\Delta t}\phi_j\ \phi_i + N_z\tfrac{\partial\phi_j}{\partial z}\tfrac{\partial\phi_i}{\partial z}} + \kappa\left(\phi_j\phi_i\right)|_{z=-h}\right\rangle \end{bmatrix}$$

$$F_{ij} = \begin{bmatrix} \left\langle\overline{\tfrac{2}{\Delta t}\phi_j\ \phi_i}\right\rangle & 0 \\ 0 & \left\langle\overline{\tfrac{2}{\Delta t}\phi_j\ \phi_i}\right\rangle \end{bmatrix} \qquad (27)$$

$$G_{ij} = \frac{1}{2}\begin{bmatrix} -\left\langle g\tfrac{\partial W_j}{\partial x}\ W_i\right\rangle \\ -\left\langle g\tfrac{\partial W_j}{\partial y}\ W_i\right\rangle \end{bmatrix} \qquad (28)$$

$$Z_{ij} = \overline{\phi_j(x_i,y_i,z)} \qquad (29)$$

$$Y_{ij} = \phi_j|_{z=-h(x_i,y_i)} \qquad (30)$$

Note that $[Z]$ performs vertical averages; and $[Y]$ samples the velocity basis at the bottom. Y is binary in our case.

Table 6: Summary of Finite Element Matrices

6.2 Optimal Fit of Model to Data

We seek the least squares fit:

$$\text{Minimize } \tfrac{1}{2}\Big\{\rho^*[W_\rho]\rho + \sum_{k=1}^{N} \delta_k^*[W_\delta]\delta_k + \sum_{k=1}^{N} \epsilon_k^*[W_\epsilon]\epsilon_k\Big\}$$

subject to the forward model constraints. To enforce these, we introducing the Lagrange Multipliers $\tilde{\lambda}$, $\tilde{\gamma}$, $\tilde{\nu}$, $\tilde{\mu}$, λ, and γ and minimize the functional Ω

$$\begin{aligned}
\Omega &= \frac{1}{2}\Big\{\rho^*[W_\rho]\rho + \sum_{k=1}^{N} \delta_k^*[W_\delta]\delta_k + \sum_{k=1}^{N} \epsilon_k^*[W_\epsilon]\epsilon_k\Big\} \\
&+ \sum_{k=1}^{N} \tilde{\gamma}_k^* \left([P_k]\,V_k + [Q_k]\,V_{k-1} - \tilde{d}_k + \delta_k\right) + \sum_{k=1}^{N} \tilde{\lambda}_k^* \left([T_k]\,\zeta_k + [U_k]\,\zeta_{k-1} - \tilde{e}_k + \epsilon_k\right) \\
&+ \sum_{k=1}^{N} \tilde{\nu}_k^* \left([Z]\,V_{k-1} - q_{k-1}\right) + \sum_{k=1}^{N} \tilde{\mu}_k^* \left([Y]\,V_{k-1} - vb_{k-1}\right) \\
&+ \sum_{k=1}^{N} \lambda_k^* \left([M]\,\zeta_k - [A]\,\zeta_{k-1} - [B]\,\zeta_{k-2} - [C]\,q_{k-1} - [D]\,vb_{k-1} - [E_k]\,\rho - R_k\right) \\
&+ \sum_{k=1}^{N} \gamma_k^* \left([N]\,(V_k + V_{k-1}) - [F]\,V_{k-1} - [Z^*]\,[G]\,(\zeta_k + \zeta_{k-1}) - S_k\right)
\end{aligned}$$

The gradient of Ω provides the first-order conditions for a minimum. The forward model is recovered by setting the gradient with respect to the Lagrange multipliers to zero. The other gradient terms are:

$$\begin{aligned}
\frac{\partial\Omega}{\partial\zeta_k} &= \lambda_k^*\,[M] - \lambda_{k+1}^*\,[A] - \lambda_{k+2}^*\,[B] - (\gamma_k^* + \gamma_{k+1}^*)\,[Z^*]\,[G] \\
&\quad + \tilde{\lambda}_k^*\,[T_k] + \tilde{\lambda}_{k+1}^*\,[U_{k+1}] \qquad (31) \\
\frac{\partial\Omega}{\partial V_k} &= (\gamma_k^* + \gamma_{k+1}^*)\,[N] - \gamma_{k+1}^*\,[F] + \tilde{\nu}_{k+1}^*\,[Z] + \tilde{\mu}_{k+1}^*\,[Y] \\
&\quad + \tilde{\gamma}_k^*\,[P_k] + \tilde{\gamma}_{k+1}^*\,[Q_{k+1}] \qquad (32) \\
\frac{\partial\Omega}{\partial q_k} &= -\lambda_{k+1}^*\,[C] - \tilde{\nu}_{k+1}^* \qquad (33) \\
\frac{\partial\Omega}{\partial vb_k} &= -\lambda_{k+1}^*\,[D] - \tilde{\mu}_{k+1}^* \qquad (34) \\
\frac{\partial\Omega}{\partial\delta_k} &= \delta_k^*\,[W_\delta] + \tilde{\gamma}_k^* \qquad (35) \\
\frac{\partial\Omega}{\partial\epsilon_k} &= \epsilon_k^*\,[W_\epsilon] + \tilde{\lambda}_k^* \qquad (36)
\end{aligned}$$

$$\frac{\partial \Omega}{\partial \rho} \;=\; \rho^* [W_\rho] - \sum_{k=1}^{N} \lambda_k^* [E_k] \tag{37}$$

Setting the first six of these to zero gives the **Adjoint Model**:

$$\begin{aligned}(\gamma_k^* + \gamma_{k+1}^*) [N] - \gamma_{k+1}^* [F] + \tilde{\nu}_{k+1}^* [Z] + \tilde{\mu}_{k+1}^* [Y] = \\ \delta_k^* [W_\delta] [P_k] + \delta_{k+1}^* [W_\delta] [Q_{k+1}] \qquad k = [1, N]\end{aligned} \tag{38}$$

$$\begin{aligned}\lambda_k^* [M] - \lambda_{k+1}^* [A] - \lambda_{k+2}^* [B] = \\ (\gamma_k^* + \gamma_{k+1}^*) [Z^*] [G] + \epsilon_k^* [W_\epsilon] [T_k] + \epsilon_{k+1}^* [W_\epsilon] [U_{k+1}] \qquad k = [1, N]\end{aligned} \tag{39}$$

$$\tilde{\nu}_{k+1}^* = -\lambda_{k+1}^* [C] \qquad k = [0, N-1] \tag{40}$$

$$\tilde{\mu}_{k+1}^* = -\lambda_{k+1}^* [D] \qquad k = [0, N-1] \tag{41}$$

with **Terminal Conditions**

$$\delta_k, \epsilon_k, \lambda_k, \gamma_k = 0 \qquad k > N \tag{42}$$

(Note that $\tilde{\lambda}$ and $\tilde{\gamma}$ have been eliminated.)

The last gradient vector $\partial\Omega/\partial\rho$ (equation 37) gives the **direction of steepest descent**. It is set to zero by iteration. During iteration it directs the improvement in the boundary conditions ρ.

6.3 Gradient Descent Algorithm

We search for the minimum Ω in the parameter space ρ. From any given point ρ^0, there are 2 decisions: which direction to move in (ρ^1) and how far to go in that direction (θ) before changing direction:

$$\rho = \rho^0 + \theta\rho^1 \tag{43}$$

Any selection of ρ^1 will produce new errors

$$\delta = \delta^0 + \theta\delta^1 \tag{44}$$

$$\epsilon = \epsilon^0 + \theta\epsilon^1 \tag{45}$$

with (δ^0, ϵ^0) the current errors and (δ^1, ϵ^1) the change in the errors achieved with $\theta = 1$. In turn the objective function will become a simple function of θ. The **optimal** θ is obtained by simple minimization:

$$\theta = -\frac{\rho^{1*}[W_\rho]\rho^0 + \delta^{1*}[W_\delta]\delta^0 + \epsilon^{1*}[W_\epsilon]\epsilon^0}{\rho^{1*}[W_\rho]\rho^1 + \delta^{1*}[W_\delta]\delta^1 + \epsilon^{1*}[W_\epsilon]\epsilon^1} \tag{46}$$

This minimizes Ω along the search direction ρ^1.

The direction of descent ρ^1 is selected according to the one of two methods: **Method of Steepest Descent** or **Conjugate Gradient Method**(Press et al., 1986; Golub and VanLoan, 1983). In the steepest descent method, ρ^1 is simply the negative of the gradient of the cost function, $\rho^1 = -\partial\Omega/\partial\rho$. In the CGM, we identify a sequence of gradients $g_l \equiv \partial\Omega/\partial\rho_l$ and directions h_l, with l indicting iteration number. The direction is computed as a blend of the current gradient and the previous direction:

$$h_{l+1} = -g_{l+1} + \gamma h_l \tag{47}$$

$$\gamma = \frac{(g_{l+1} - g_l) \cdot g_{l+1}}{g_l \cdot g_l} \tag{48}$$

ρ^1 is then computed as an increment of arbitrary length in this direction:

$$\rho^1_{l+1} \parallel h_{l+1} = (-g_{l+1} + \gamma h_l) \tag{49}$$

The method reduces to the steepest descent method if γ is arbitrarily set to zero. Initially, $h_0 = -g_0$ is sufficient to get things started.

The gradient descent algorithm is summarized as follows:

- Given ρ^0, δ^0, ϵ^0, and the gradient $\frac{\partial\Omega}{\partial\rho}$, compute the direction of descent ρ^1
- A single forward-model calculation with $\theta = 1$ is sufficient to calculate δ^1 and ϵ^1
- Compute the optimal value of θ
- By superposition, update ρ, δ, and ϵ

This completes the new, improved forward solution. It is followed by an adjoint solution forced by the new values of δ and ϵ, resulting in a new gradient $\frac{\partial\Omega}{\partial\rho}$, and the cycle continued to convergence. A suitable convergence rule is the achievement of vanishingly small values of the gradient, thereby satisfying the final first-order condition for an optimal solution. Note that only a single forward and adjoint solution is required per iteration.

6.4 Adjoint Equations

There is a natural correspondence among the prime (forward) and dual (adjoint) variables:

$$\zeta \Longleftrightarrow \lambda \tag{50}$$
$$V \Longleftrightarrow \gamma \tag{51}$$
$$q \Longleftrightarrow \nu \tag{52}$$
$$vb \Longleftrightarrow \mu \tag{53}$$

It is useful to recast the adjoint equations by a change of nomenclature:

$$\begin{aligned}(V_k^* + V_{k+1}^*)\,[N] - V_{k+1}^*\,[F] + q_{k+1}^*\,[Z] + vb_{k+1}^*\,[Y] = \\ \delta_k^*\,[W_\delta]\,[P_k] + \delta_{k+1}^*\,[W_\delta]\,[Q_{k+1}] \qquad k=[1,N]\end{aligned} \tag{54}$$

$$\begin{aligned}\zeta_k^*\,[M] - \zeta_{k+1}^*\,[A] - \zeta_{k+2}^*\,[B] = \\ (V_k^* + V_{k+1}^*)\,[Z^*]\,[G] + \epsilon_k^*\,[W_\epsilon]\,[T_k] + \epsilon_{k+1}^*\,[W_\epsilon]\,[U_{k+1}] \qquad k=[1,N]\end{aligned} \tag{55}$$

$$q_{k+1}^* = -\zeta_{k+1}^*\,[C] \qquad k=[0,N-1] \tag{56}$$

$$vb_{k+1}^* = -\zeta_{k+1}^*\,[D] \qquad k=[0,N-1] \tag{57}$$

with Terminal Conditions

$$\delta_k, \epsilon_k, \zeta_k, V_k = 0 \qquad k > N \tag{58}$$

and Gradient

$$\frac{\partial \Omega}{\partial \rho} = \rho^*\,[W_\rho] - \sum_{k=1}^{N} \zeta_k^*\,[E_k] \tag{59}$$

In this form, the strong correspondence among forward and adjoint equation sets is more visible. Essentially all the algorithms for solving the forward model can be recycled for the adjoint, with due regard for the detailed differences (e.g. reversal of first derivatives and the $\zeta :: V$ coupling).

6.5 Covariance Matrices

6.5.1 Elevation

Elevation measurements are assumed to be sampled at a point in (x, y, t). Modeled elevation is sampled in the same way.

Elevation errors are assumed to be homogeneous and independent. W_ϵ is configured to compute the *weighted mean squared elevation error*,

$$\epsilon^*[W_\epsilon]\epsilon = \frac{1}{N_{obs}} \sum \epsilon_i^2 / E_{noise}^2$$

where E_{noise}^2 is the expected value of the i^{th} squared error (the elevation noise variance), and N_{obs} is the number of elevation observations. Accordingly, $[W_\epsilon]$ is diagonal:

$$[W_\epsilon] = \frac{1}{N_{obs}} \frac{1}{E_{noise}^2} [I]$$

6.5.2 Velocity

Velocity measurements are assumed to be sampled at a point in (x, y, t) and averaged over a vertical interval (z_1, z_2). Modeled velocity is sampled in the same way. To adjust for discrepancies between model and observed bathymetries, the vertical averaging is done in a bathymetry-normalized vertical coordinate.

Velocity errors are assumed to be homogeneous and independent. W_δ is configured to compute the *weighted mean squared velocity error*,

$$\delta^*[W_\delta]\delta = \frac{1}{N_{obs}} \sum (\delta_x^2 + \delta_y^2)_i / V_{noise}^2$$

where V_{noise}^2 is the expected value of the i^{th} squared error (the velocity noise variance), and N_{obs} is the number of velocity observations. (Number of ADCP points). Accordingly, $[W_\delta]$ is diagonal:

$$[W_\delta] = \frac{1}{N_{obs}} \frac{1}{V_{noise}^2} [I]$$

6.5.3 Boundary Conditions

W_ρ is configured to compute a weighted sum of mean squared BC size, slope, and tendency. The weight factors are w_0, w_1 and w_2 respectively. This is written as

$$\rho^*[W_\rho]\rho = \frac{1}{\int dt} \frac{1}{\oint ds} \int \oint \left[w_0 \rho^2 + w_1 \left(\frac{\partial \rho}{\partial s} \right)^2 + w_2 \left(\frac{\partial \rho}{\partial t} \right)^2 \right] ds\, dt$$

Note that this measure is normalized with respect to the boundary length $\oint ds$ and the time span $\int dt$. It represents a weighted average of the boundary signals, not a summation or integration.

The ideal value of w_0 would be 1/[the expected size**2 of a boundary elevation]; for w_1, 1/[expected boundary slope**2]; and for w_2, 1/[expected boundary tendency**2]. It is useful to relate w_1 to the expected size of geostrophic boundary inflows V_g:

$$V_g = \frac{g}{f}\frac{\partial \zeta}{\partial s}$$

Thus w_1 scales with $[fV_g/g]^{-2}$, i.e. roughly $10^{10}/{V_g}^2$. It is useful to scale V_g to the observed velocity data variance V_{obs}^2, adjusted for the noise level:

$${V_g}^2 \sim V_{obs}^2 - V_{noise}^2$$

6.5.4 Construction of W_ρ

The construction of the boundary covariance proceeds as follows. $\rho(s,t)$ is represented in the separate linear bases $\phi_i(s)$ and $\psi_j(t)$:

$$\rho(s,t) = \sum \rho_{ik}\phi_i(s)\psi_k(t)$$

ρ_{ik} represents the value of the boundary condition at spatial node i and time level k. The spatial basis $\phi_i(s)$ is identical to the FEM basis for $\zeta(x,y)$ along the boundary. The temporal basis $\psi_k(t)$ represents linear interpolation among equally-spaced time points k, but these points in boundary condition time are separated by a larger time step Δt_ρ than that used for $\zeta(t)$ in the simulation. This represents an extra form of temporal smoothing.

Expanding ρ in its bases, we have the quadratic form for $\rho^*[W_\rho]\rho$:

$$\rho^*[W_\rho]\rho = \sum_{ijkl}\rho_{ik}\left[\frac{1}{\int dt}\frac{1}{\oint ds}\left\{w_0\oint \phi_i\phi_j ds\int \psi_k\psi_l dt\right.\right.$$
$$\left.\left.+w_1\oint \frac{\partial \phi_i}{\partial s}\frac{\partial \phi_j}{\partial s}ds\int \psi_k\psi_l dt + w_2\oint \phi_i\phi_j ds\int \frac{\partial \psi_k}{\partial s}\frac{\partial \psi_l}{\partial s}dt\right\}\right]\rho_{jl}$$

It is useful to recognize standard FEM matrices $[\mathcal{M}]$, the boundary-integral mass matrix, and $[\mathcal{K}]$, the negative of the FEM boundary Laplacian:

$$\mathcal{K}_{ij} = \oint \frac{\partial \phi_i}{\partial s}\frac{\partial \phi_j}{\partial s}\, ds \qquad\qquad \mathcal{M}_{ij} = \oint \phi_i\phi_j\, ds$$

Analogous matrices $[\mathcal{L}]$ and $[\mathcal{N}]$ comprise integrals of the temporal bases:

$$\mathcal{L}_{kl} = \int \frac{\partial \psi_k}{\partial t} \frac{\partial \psi_l}{\partial t}\, dt \qquad\qquad \mathcal{N}_{kl} = \int \psi_k \psi_l \, dt$$

All integrals in W_ρ are evaluated using the Trapezoidal Rule (nodal quadrature). With linear bases, and assuming uniform Δt_ρ, we have the following properties:

- $[\mathcal{M}]$ and $[\mathcal{N}]$ are diagonal; $[\mathcal{K}]$ and $[\mathcal{L}]$ are symmetric and sparse, with at most three nonzero entries per row.
- $\mathcal{N}_{kk} = \Delta t_\rho$ for 'interior' $k = [2, N_{t-1}]$. For the endpoints $k = 1$ and $k = N_t$, $\mathcal{N}_{kk} = \Delta t_\rho/2$.
- Each row of $[\mathcal{L}]$ represents the (negative) centered second time-difference:

$$[\mathcal{L}]\rho_{i,k} = \frac{-1}{\Delta t_\rho}\delta_k^2 \rho_{i,k} = \frac{-\rho_{i,k-1} + 2\rho_{i,k} - \rho_{i,k+1}}{\Delta t_\rho}$$

  with homogeneous Neumann initial and terminal conditions $\frac{\partial \rho}{\partial t} = 0$:

$$[\mathcal{L}]\rho_{i,1} = \frac{\rho_{i,1} - \rho_{i,2}}{\Delta t_\rho} \qquad\qquad (\rho_{i,0} = \rho_{i,1})$$

$$[\mathcal{L}]\rho_{i,N_t} = \frac{-\rho_{i,N_t-1} + \rho_{i,N_t}}{\Delta t_\rho} \qquad\qquad (\rho_{i,N_t+1} = \rho_{i,N_t})$$

With these simplifications, it is only necessary to store two small arrays to represent W_ρ:

- $W_{\rho s}$ contains the time-invariant spatial smoothing operator:

$$[W_{\rho s}] = \frac{1}{\int dt} \frac{1}{\oint ds} \left(w_0[\mathcal{M}] + w_1[\mathcal{K}] \right) \Delta t_\rho$$

  with $\int dt = \Delta t_\rho (N_t - 1)$. The dimension of $W_{\rho s}$ is $(N_b,\ N_b)$ since it represents time-invariant spatial smoothing. (N_b is the number of boundary nodes.) Because it is sparse, its storage requires only $3N_b$ entries ($2N_b$ if symmetry is exploited).

  $W_{\rho t}$ contains the essential time-smoothing information. Because of the simplicity of uniform Δt_ρ, this matrix is also time-invariant, and diagonal, keyed to the spatial boundary node number:

$$[W_{\rho t}] = \frac{1}{\int dt} \frac{1}{\oint ds} w_2 [\mathcal{M}] \frac{1}{\Delta t_\rho}$$

The general space-time operation $\rho^*[W_\rho]\rho$ is achieved by

$$\rho^*[W_\rho]\rho = \sum_{k=1}^{N_t} \epsilon_k \sum_{i,j=1}^{N_b} \rho(i,k) W_{\rho s}(i,j)\rho(j,k)$$

$$+ \sum_{i=1}^{N_b} W_{\rho t}(i,i) \sum_{k=1}^{N_t} \rho(i,k) \left[-\rho(i,k-1) + 2\rho(i,k) - \rho(i,k+1)\right]$$

The first operation represents the spatial smoothing. The endpoint factor ϵ_k implements trapezoidal rule integration in time: $\epsilon_k = 0.5$ for $k = 1$ or N_t; otherwise $\epsilon_k = 1$. The second operation represents temporal smoothing. The end point conditions $\rho_{i,0} = \rho_{i,1}$ and $\rho_{i,N_t+1} = \rho_{i,N_t}$ are implied.

Similarly, the space-time operation $[W_\rho]\rho$ (by symmetry, the same as $\{\rho^*[W_\rho]\}^*$) is achieved by

$$[W_\rho]\rho = \epsilon_k \sum_{j=1}^{N_b} W_{\rho s}(i,j)\rho(j,k)$$

$$+W_{\rho t}(i,i) \left[-\rho(i,k-1) + 2\rho(i,k) - \rho(i,k+1)\right]$$

for all (i,k), with the same endpoint conditions.

7 Software Design and Implementation Notes

This section describes the design of the CASCO software. The purpose is provide insight into the way the software is structured so that future developers can find their way about in the code.

Conceptually CASCO consists of three parts: a linear forward model (SACO); the adjoint of SACO, called MOODY; and the support code which uses SACO and MOODY in an iterative procedure to compute the elevation boundary conditions which minimize the unexplained errors (velocities and elevations). This is the picture illustrated in Fig. 1.

The purpose of SACO is take the current estimate of the elevation boundary conditions, compute the velocities, elevations and the errors. The purpose of MOODY is to take the errors and compute the gradient of the cost function (Ω) with respect to the boundary elevations (the control variables). The support code has four functions: 1) use the gradient information ($\partial\Omega/\partial\rho$) to estimate new boundary conditions for SACO; 2) coordinate the iteration procedure using SACO and MOODY; 3) determine whether the iteration has converged; and 4) deal with the input and output.

SACO is a linearization of QUODDY4. The model is 3-d barotropic with user specified vertical eddy viscosity and bottom friction. The only forcing terms are the elevation boundary conditions. The wind and baroclinic forcing have been removed.

7.1 Control Structure

The structure of CASCO4b is shown in Fig. 32. The actual source code (casco4b.f) looks much the same. The basic flow is as follows. All the input information is assembled and the covariance matrices (W_ρ, W_ϵ, W_δ) calculated. Then there is an iteration loop which uses MOODY to calculate the gradient of the cost function, computes trial boundary conditions, uses those boundary conditions to run SACO, estimates the optimal step size, adjusts the solution and decides whether convergence has occurred. Finally there is a final run of SACO and the answers are written out.

The code is based on the QUODDY4 source code. The main QUODDY module was split into 3 modules (startup, station_calc, saco_run), and the elevation and velocity calculation modules were stripped down the linear barotropic core (elevations4, verticals5).

The key design decision was to write the program so that the main module of CASCO never sees the data arrays. It simply calls a series of modules which do the work (call these the upper modules). The data is passed between the upper modules via common blocks (discussed below). The upper modules call the next level of working modules (such as stationaryc4, elevations4, verticals5) in the way they were called before in QUODDY4.

The details of the calculation of the velocities and elevations have changed from QUODDY4. The most important change is that the wave equation and velocity calculations are now explicitly matrix operations. This was done to simplify the construction of the adjoint model, and has led to the creation of several new stationary arrays. As well, the wave equation has been linearized, and elevation, ζ, is now a prognostic variable (rather than total depth), and the friction terms (bottom stress coefficient and vertical eddy viscosity) are now constant in time,

As a result of these changes, the user subroutines which define the vertical grid (`vertgrids2`), the bottom friction parameters (`lin_stress`), and the vertical eddy viscosity (`lin_enz`) are only called during the initialization phase.

Notice that an independent SACO can be constructed using a subset of the subroutines shown in Fig. 32. Figure 33 shows the structure of SACO, which is very close the source code for the version of SACO that was used to calculate the 'TRUTH' solutions in Section 5.

There are lots of new subroutines and most of them are stored one routine per file. The genealogy (who calls whom) is given in Fig. 34. As the subroutines stabilize they can be packaged into a smaller number of files.

7.2 Data Structures

An attempt was made to group the data into natural families (data structures) and define one or two common blocks per family. Each family is defined in an include file

CASCO4b Main Module

```
iter = 0
call casco_input               !Get the casco user inputs.
itermax = max_iterations       !Establish the maximum # of iterations.
method = descent_code()        !Get descent method.
call startup                   !Read the mesh files and such.
call station_calc              !Calculate all the constant and common arrays.
call get_friction(iter)        !Read the friction values.
call set_times                 !Precalculate all the timing information.
call getdata_moody             !Get the forcing data.
call measure_func              !Compute the measurement functionals.
call get_errvar                !Compute the error variances.
call get_Wrho                  !Build weight function for the elevation bcs.
call initial_conditions        !Set the initial conditions.
call statistics(iter)          !Compute initial value of the cost function.
converged = .false.
do while ((iter.lt.itermax).and.(.not.converged))
   iter = iter + 1
   call moody_run(iter)        !Compute the gradient of the cost function.
   if method == 1 then         !Compute the trial bcs for the saco run.
       call trial_bcs(iter)    !Gradient descent.
   else if method == 2
       call trial_bcs_cgm(iter)    !Conjugate gradient method.
   else
       STOP due to error
   endif
   call saco_run(iter)         !Compute velocity errors based on trial bcs
   call adjust_soln(iter)      !Compute optimum step size, bcs & errors
   call statistics(iter)
   converged = convergence(iter)   !Test for convergence.
end while
call write_cbc                 !write the perturbation bcs
call write_error               !write the final errors (version 1).
call dbc_best_to_dbc           !Move the bcs around.
call saco_run(-1)              !Do the final saco run.
call error_to_best             !Move errors to arrays to write from.
call write_error               !Write the final errors (version 2).
```

Figure 32: The structure of CASCO4b

Table 7: The include files which define the data families.

CASCO.DIM	All the array size specifications
casco_params.h	The casco specific input parameters
constants.h	All the physical constants and time stepping parameters
covariance.h	The variance and covariance of the elevation and velocity errors
deltabcs.h	The perturbation boundary conditions calculated by MOODY
errors.h	The velocity and elevation errors
friction.h	The bottom friction and vertical eddy viscosity information
gradcost.h	The gradient of the cost function
mesh.h	All the input related to mesh geometry and boundary conditions
misc.h	Miscellaneous stuff
numbers.h	Definition of functions which return useful numbers (not used yet)
observes.h	The time, location and values of the unexplained observations which are the forcing for casco.
saco.h	The prognostic variables for SACO
sacotime.h	Timing information for the forward run and the mapping from the model time points to the boundary condition time points.
startstop.h	The start and stop times for the model runs
statarrays.h	The stationary arrays calculated by stationaryq4
statistics.h	The current values of the basic statistical measures
weights.h	The interpolation functions for the observations which define the measurement functionals
wrho.h	The weight function for the elevation boundary conditions

(*.h, Table 7). The name of the common block is usually the same as the name of the include file (but not always).

The include files contain the variable declarations, common blocks, and documentation for the data structures. The documentation includes assigning responsibility for filling the data structures.

All the data is hidden from the main part of CASCO. The common blocks are used to pass data between the upper modules (defined as those that appear in Figure 32). There is one family per include file so that modules only see data that they need. The upper modules communicate with the next level by a combination of common blocks and subroutine arguments.

SACO Main Module

```
call startup              !Read the mesh files and such.
call station_calc         !Calculate all the constant and common arrays.
iter = 1
call get_friction(iter)   !Read the friction values.
call set_times            !Precalculate all the timing information.
call getdata_moody        !Get the observations.
call measure_func         !Compute the measurement functionals.
iter = -1
call saco_run(iter)       !Run the forward model.
```

Figure 33: The structure of SACO based on CASCO modules. To sucessfully run SACO using this structure the user will need to modify the subroutines that provide the elevation boundary conditions (bcs2) and write the output (outputs2).

CASCO: Who calls Whom

```
startup:  echosets3, initializes3, gday, up_date
get_friction:  lin_stress, lin_enz
station_calc:
    stationaryc2:  sprsbld1, ematbuild, bansoltr, sprsrowzap, cthomas

set_times:  get_sacotime, gday_utc0, get_bcstime, time_basis
getdata_moody:  getvelobs, getelevobs, nobservations, nelevobservations
    getvel_obs:  getobs_8c, basis2d, utc0_gday, indexx, sortDRL
    getelev_obs:  getobs_4c, basis2d, utc0_gday, indexx, sortDRL
measure_func:  saco_basis:  basis2d, poival3d
    vertweights :  poival3d

get_Wrho:  BuildWrho3:  sprsbldw1
statistics:  nobservations, nelevobservations, nelev_bcs, rms_2vec,
    rwms_2vec, rms_1vec, dWDELTAd, xWRHOx, xWRHOy

moody_run:  up_date2, EMATpreMULT, vertavg, xWRHO
    elevationm3:  sprspremltadd, bansoltr
outputm2:  ts1_moody_write, Domega2_write, Domega1_write
verticalm4:  sprspremlt, cthomas

trial_bcs:  sup_norm2d
trial_bcs_cgm:  sup_norm2d, directionCGM
saco_run:  bcs2, dmy, vertavg, up_date
    elevations4:  sprsmltadd, bcs2, EMATMULT, bansoltr
    outputs2:  ts1_saco_write
    saco_uverror:  poival3d
    verticals5:  sprsmlt, cthomas

adjust_soln:  sup_norm2d,sup_norm1d,nobservations,nelevobservations,nelev_bcs
    stepsize5:  d1WDELTAd2, dWDELTAd, xWrhox, xWrhoy

error_to_best:  nobservations, nelevobservations
dbc_best_to_dbc:  nelev_bcs
write_cbc:  dmy, cstring, nelev_bcs
write_error:  write_uverror, write_eerror
    write_uverror:  gday_utc0
    write_eerror:  gday_utc0
```

Figure 34: CASCO4b genealogy. Some routines that do not call other routines are not listed.

7.3 Important Data Items

Here we discuss the important data items, where they are initialized and where they are updated.

Observations The data structure for the observations is defined in observes.h

The observations are read in by getdata_moody (casco4b).

Errors The data structure for the errors is defined in errors.h

The errors are initialized in initial_conditions.

The trial error fields (UERROR, VERROR, EERROR) are updated during the SACO run (outputs2, saco_uverror, saco_eerror).

The best estimate of the error fields (uerr_best, verr_best, eerr_best) are calculated in adjust_soln.

The final error is written to a file by write_error.

Measurement functionals The data structure for measurement functionals are defined in weights.h. The velocity calculations are done by saco_basis and vertweights. The elevation calculations are done by saco_basis_elev. The subroutine measure_func acts as a wrapper for saco_basis and saco_basis_elev and hides the details from casco4b.

Boundary Conditions The trial elevation boundary conditions are passed to SACO in an array DBC (see deltabcs.h). DBC is the responsibility of trial_bcs.

The best estimate of the elevation boundary conditions (DBC_BEST) is calculated in adjust_soln after the SACO run.

The final boundary conditions (DBC_BEST) are written to a file in write_cbc (casco4b).

Related parameters are

DELTTBCS the time interval for the boundary conditions in DBC (calculated in set_times based on the averaging time scale specified by the user).

NTBC the number of time levels (for the bcs) that MOODY passes to SACO. This is calculated in set_times.

Error Covariance The error covariance structure is assumed and diagonal and in the present implementation is assumed constant (Section 2.4). The expected rms velocity error and elevation error are specified by the user and the error variances are calculated in get_errvar (CASCO4b). The data structures are defined in covariance.h.

BC Covariance The boundary condition covariance is defined by the three numbers W0, W1, W2 and the calculation in BuildWrho3 (get_Wrho). W0 provides amplitude control, W1 provides slope control, and W2 provides tendency control.

The data structures are defined in wrho.h.

Friction The friction parameters are user specified parameters defined by calls to user subroutines: bottom friction (lin_ak), vertical eddy viscosity (lin_enz),

By assumption the friction is constant in time.

The horizontal viscosity has been eliminated.

Vertical Grid The vertical node positions (ZMID) are computed using using vertgrids2 (initializes3).

The vertical grid is fixed in time. I think all references to ZOLD and ZNEW have been removed.

One consequence of fixing the vertical grid is that ZMID(I,NNV) cannot be used to determine the elevation.

7.4 Legacy Features

This section discusses several odd features which may cause some confusion. These features are a legacy from the development pathway taken with CASCO. While annoying, these are not bugs.

The following features are a result of the fact that the CASCO input routines are based on the QUODDY4.1 input routines and the conversion has not been pushed to completion.

- The *.lev file is required even though CASCO does not use baroclinic pressure gradients.
- The bottom friction parameters NLBS and AKMIN are in the *.ins file. However their only use is in get_friction1, where NLBS determines whether to use AKMIN for the bottom friction parameter or call the user subroutine lin_stress. The friction related parameters should probably be relegated entirely to the user subroutines. However the present structure parallels the QUODDY structure and has proven very useful for development purposes.
- The variable EPSN is the time weighting factor for the solution of the vertical structure of the velocity. It is specified in line 24 of the *.ins file (Fig. 2). In CASCO3e and CASCO4b this variable is not used. In subroutine stationaryc2 `ESPN=0.5` is specified. This input parameter is required for the software to run but it is ignored.

These types of features should be removed as the software develops.

One item that was on this list for CASCO3b (Hannah and Lynch, 1999) was the variable `CLOSURE`. This variable has been moved into the include file `friction.h`. This makes it much easier to find and change.

The variable `scale_DoDr` (see Section 7.6) has been moved out of the subroutine `casco_input.f` and into the include file `casco_params.h`. This should make it much easier to find and change.

The subroutine `statistics` computes the cost function statistics. In the current version the cost function components and the weighted mean square errors are computed separately, even though they are closely related (Section 3.2). This is simply a residual from earlier versions and should be cleaned up in the future.

7.5 Timing

7.5.1 Model Time

Time nodes in the Forward and Adjoint Models are represented with the same universal time index k used in the equations presented herein. k increases in the natural (forward time) direction. The unknown Forward and Adjoint variables are at $k = [1, N]$. The Forward Model Initial Conditions (ICs) are at $k = [-1, 0]$. The Adjoint Model Terminal Conditions (TCs) are at $k = [N + 1, N + 2]$. The model timestep DT is uniform. Time steps are numbered in the natural manner. The first time step goes from k=0 to k=1; etc. Generally, time step k ends at time point k, time tk=t0+k*DT. This is summarized in Fig. 35. The crude notation such as t0 and DT is used in the text to allow for easy reference to the ASCII graphics in Figs. 35 - 37.

The forward model stops when the time at the end of the current time step is greater than or equal to the user-specified stop time. The time check is after the time step has been completed. Thus the Forward Model will generally run beyond the requested stop time by a fraction of a time step. This is turn determines the location of the Terminal Conditions for the Adjoint Model, since the time nodes coincide exactly.

The real world time is kept track of by two numbers: `kd`, `seckd`. `kd` is the day counter and `seckd` counts the elapsed seconds in the current day.

7.5.2 Boundary Condition Time

The MOODY boundary conditions which are sent to SACO are defined at a larger timestep than the model time step (identical in both SACO and MOODY). Let the boundary condition time counter be `j`. Figure 36 shows the relationship between `j` and the universal time counter `k`.

```
       t=t0                t=tk=t0+k*DT
       |                   |
       |                   |
k: -1  0   1   2   3   k-1 k   k+1 N-1 N   N+1 N+2
    |  |                                    |   |
    |  |                                    |   |
    ICs                                     TCs

  k = 0 at the beginning of the forward model run
  k = 1 at the end of the first forward model time step
  k = N at the end of the last forward model time step

  k = N+1 at the beginning of the adjoint (backwards) model run
  k = N at the end of the first adjoint model time step
  k = 1 at the end of the last adjoint model time step
```

Figure 35: Definition of the model time in CASCO. `k` is the universal time index. ICs are the initial conditions and TCs are the terminal conditions.

```
k: -1  0   1   2   3   k-1 k   k+1 N-1 N   N+1 N+2
       |                               |
       |                               |
j:     1               j         j+1   Jmax

  j = 1    coincides with k = 0
  j = Jmax coincides with k = N
```

Figure 36: Definition of the boundary model time in CASCO. `k` is the universal time index (Fig. 35) and `j` is the boundary condition time index.

The mapping between the model time (`k`) to the boundary condition time (`j`) is computed in `set_times` as follows. For each k we need:

1. the next `j` in the list i.e. min(`j`) s.t. time(`k`) $\leq$ time(`j`).
2. theta defined as follows
   - let the time from `j`-1 to `j` be DTBCS
   - let the time from `j`-1 to `k` be DT
   - let theta = DT/DTBCS
   - thus if time(`k`) = time(`j`) then THETA = 1

We have insisted on

- time(`k`=0) = time(`j`=1)
- time(`k`=N) = time(`j`=Jmax)
- For time(`k`) < time(`j`=1) we set `j` = 1, THETA = 1.
- For time(`k`) $\leq$ time(`j`=Jmax) we set `j` = NTBC, THETA = 1.

The mapping from the model time (`k`) to the boundary condition time (`j`) is written to fortran logical units 80 and 81 by subroutine `set_times`.

The boundary condition time and the model time are only guaranteed to coincide at the endpoints. Linear interpolation is used in all cases to map the boundary elevations between the two time bases.

7.5.3 Measurement Time

The mapping of the time of the observations to the universal model time `k` is the function of the subroutine `measure_func`. The actual work is done by `saco_basis` (velocity) and `saco_basis_elev` (elevation). The information is stored in the variables `BASISOBS` and `EBASISOBS` which are defined in `weights.h`. The description below uses the velocity variable names but the description is applies to both types of observations.

In general the observation time is not at the `k`'s but fall in the intervals between. These are the time elements, which are numbered in the natural sense of the forward model:

- time element #1 ends at `k` = 1
- time element #2 ends at `k` = 2

- time element N ends at `k` = N

Assume that the model (either forward or adjoint) has just completed a time step,

- In the forward model the time element just completed is te = `k`
- In the adjoint model the time element just completed is te = `k`+1

The mapping between `k` and the observation time is shown in Fig. 37.

```
k: -1  0   1   2     3     k-1   k    k+1  N-1  N    N+1  N+2
                            |     |
                            |     |
                           1-X    X

Time weights for observation which occurred in time element k:
    X = BASISOBS(6,i) = fraction time step completed when the
             observation occurred.
    k = BASISOBS(5,i) = time element in which observation occurred.
```

Figure 37: The relationship between the universal model time `k` and the time of the observations. `BASISOBS` is an array defined for the velocity observations (`observes.h`). A similar array, called `EBASISOBS`, exists for the elevation observations.

The interpolation of the forward model results (SACO) to the observation time is done by linear interpolation. For observation i, the model velocity is $V_{k-1} * (1 - X) + V_k * X$ where X = BASISOBS(6,i) and `k` = BASISOBS(5,i) (see Fig. 37).

For the inverse model (MOODY), the contributions to the adjoint forcing at time `k` come from errors which occur in in time elements `k` and `k`+1. Let E_i be an error that occurs in time element `k`, then

$\delta_k = E_i * X_i|_k$ be the contribution from all errors which occur in time element k.

and

$\delta_{k+1} = E_i * X_i|_{k+1}$ be the contribution from all errors which occur in time element k+1.

where the δ_k appear in the original formalation (Sections 6.1 and 6.2). Then the total contribution to the adjoint forcing (the right-hand-side of the equations or RHS) at time `k` is

$$\text{RHS}(k) = E_i * X_i|_k + E_i * (1 - X_i)|_{k+1}$$

Table 8: Functions which return useful numbers.

nelev_bcs	number of elevation boundary nodes (NR)
nbc_time_steps	number of time steps in the boundary condition time series
ntime_steps	number of time steps in a single run of saco or moody
nnodes	number of horizontal nodes in the mesh (NN)
nelems	number of horizontal elements (NE)
nobservations	number of velocity observations
nelevobservations	number of elevation observations
max_iterations	maximum number of iterations
descent_code	The code to determine the descent algorithm

From the point of view of the observations, an error E_i which occurs in time element `k` contributes to the adjoint forcing at two different times `k` and `k`-1. The contributions to the forcing are

$$\begin{aligned}\mathrm{RHS}(k) &= E_i * X_i \\ \mathrm{RHS}(k-1) &= E_i * (1 - X_i)\end{aligned}$$

7.6 Other

We have tried to implement a policy of only providing subroutines with the data they actually need (data hiding). Often a subroutine needs access to a number, such as the number of boundary nodes (nr), but not any of the other data in the data family that nr is associated with. To deal with this we have implemented functions which return many of the important numbers in the code. Two examples are `max_iteration` and `descent_code` in the main casco4b module (Fig. 32). These functions are listed in Table 8 and are collected together in the file `numbers.f`.

The details of the gradient descent algorithm was changed for CASCO4b. In CASCO3b the magnitude of the trial boundary conditions was scaled with the amplitude of the cost function (smaller steps with smaller cost). In CASCO4b the amplitude of the trial boundary conditions is constant. The scaling is as follows. The trial boundary conditions are scaled so that the largest element is 1. Then the trial boundary conditions are multiplied by the parameter `scale_DoDr`. In the distribution `scale_DoDr = 0.1`, which means that the largest element of the trial boundary conditions is 10 cm. This parameter is set in the file `casco_params.h`.

We think that what is really desired is to have the trial boundary conditions generate elevations and velocities in the SACO run that are of the same order as the misfits. This will reduce the likelihood of roundoff errors dominating the descent algorithm.

A determined attempt was made to use explicit typing of all the variables. This allows one to find typing errors by compiling with the -u option (which finds all the untyped variables).

8 Final Comments

This document provides the user with complete description of the input and output structure of CASCO4b and three examples with the complete set of input files, output files and plots of the output. These should help new users to get started and help to verify that the code is working in a new environment. This document also provided a description of the theory and implementation that should help the advanced user to modify the code.

CASCO4b is a straight forward implementation of the equations presented in Section 6. We believe that the coding of CASCO4b is correct. However there may be bugs hiding in corners that we have not explored.

Very few extras have been provided to make the code easy to use. The directions for future enhancements await some practical experience with the model in real applications.

There are many areas where CASCO could be improved. These include

- Convergence rules (Section 2.4).
- Details of the estimate of the trial boundary conditions (Section 7.6).
- Improved covariance models (Section 2.4).
- Improved boundary conditions in regions that do not influence the observations (Section 5).
- Allowing for nondiagonal covariance structure (next paragraph).

There is one area for future development that will require both theoretical developments and practical experience. In their one experiment with a short boundary condition time step (3 hr), Lynch and Hannah (2001) found that inertial oscillations developed in both the forward and inverse models. They were difficult to control and were only removed after several hundred iterations. The CASCO3e User Guide (Hannah and Lynch, 1999) also provides an example of this problem in the example of the use of the weight functions to provide temporal smoothing. This example (Section 5.2 of Hannah and Lynch (1999)) was not provided in this document.

Lynch and Naimie (2002) found that a practical way around this problem for their application was to write the boundary condition time series as a convolution of the wind stress. Another possibility is to run CASCO with a coarse boundary condition time step as a first cut and then refine the time step.

We believe that the heart of the problem is that the misfits (or errors) are presented to the inverse model as a series of impulses at the observation time and these impulses excite inertial oscillations. One way forward would be smear the misfits in time. The most general way to implement this would be to allow for non-diagonal covariance structure for the misfits, which would require revisions to the code of the inverse model. There may be other useful methods.

Acknowledgements - We thank Chris Naimie for his efforts on the software engineering of QUODDY4. His work made the development of CASCO much easier than could have been. We also thank Chris for the measurement functional code. This work was supported in part by the U.S. National Science Foundation (NSF), the U.S. National Oceanic and Atmospheric Administration (NOAA).

References

Golub, G. and VanLoan, C. (1983). *Matrix Computations.* The Johns Hopkins University Press.

Hannah, C. and Lynch, D. (1999). Casco3 User Guide. Dartmouth College Numerical Methods Lab Report NML-99-6, November 1999. http://www-nml.dartmouth.edu/Publications/internal_reports/NML-99-6.html.

Lynch, D. and Hannah, C. (2001). Inverse model for limited-area hindcasts on the continental shelf. *J. Atmos. Ocean. Tech.*, 18:962–981.

Lynch, D., Ip, J., Naimie, C., and Werner, F. (1996). Comprehensive coastal circulation model with application to the Gulf of Maine. *Contin. Shelf Res.*, 16:875–906.

Lynch, D. and Naimie, C. (2002). Hindcasting the Georges Bank circulation, Part II: wind-band inversion. *Contin. Shelf Res.*, 22:2191–2224.

Lynch, D., Naimie, C., and Hannah, C. (1998). Hindcasting the Georges Bank Circulation: Part I, Detiding. *Contin. Shelf Res.*, 18:607–639.

Lynch, D. and Werner, F. (1991). Three-dimensional hydrodynamics on finite elements, Part II: Nonlinear time-stepping model. *Int. J. Num. Methods Fluids*, 12:507–533.

Press, W. H., Flannery, B., Teukolsky, S., and Vetterling, W. (1986). *Numerical Recipes: The Art of Scientific Computing.* Cambridge University Press.

A Source Code and Test Cases

All of the software and data files used to make the runs described in this document are contained in the distribution of the Casco4b code. This distribution can be found under Casco in the Software section of the web page of the Numerical Methods Lab at Dartmouth College (try `http://www-nml.dartmouth.edu/circmods/gom.html` or `http://www-nml.dartmouth.edu/Software`).

The matlab scripts which make the plots are also included. Some of the scripts make use of the OPNML matlab tool box written by Brian Blanton, which can be found at `http://www.opnml.unc.edu`. The version used are the Matlab5 versions.

Below is the text of the file `readme.contents` included with the November 2002 distribution of Casco4b.

```
*****************************************************************
Distribution of Casco4b   November 2002

**Source Code
  casco - An inverse model for estimating the optimal time
          varying barotropic boundary conditions from velocity errors.

  saco  - An independent version of the linear forward model embedded
  in casco.  The only forcing is the pressure boundary conditions.
          This code was used to create the TRUTH for the test cases
          (modeA and modeB).

**Directories
   ./bcs_forward/      - boundary conditions for the test problems
   ./inputs/           - forcing data and input
   ./mesh/             - mesh files for the test problems
   ./mfiles/           - matlab file for plotting the casco output
   ./run/              - the home of the test cases
   ./src/casco4b       - the casco4b source code
   ./src/casco4b/saco3 - source code for saco3(uses code from casco)

**Useful Information Files
   ./bcs_forward/readme.bcs - describes the elevation bc forcing files
   ./inputs/Readme.inputs   - describes the contents of the inputs directory
   ./readme.contents   -    - this file
   ./run/Readme.basecase    - describes the scripts for running the test cases

**************************************************
**Contents of the directories

./bcs_forward: boundary conditions for the test problems
```

```
readme.bcs
test6.testH.atw.ak0.0005.resbcs.3sides.s2r
test6.testH.atw.ak0.0005.resbcs.s2r
test6.testI.atw.ak0.0005.resbcs.3sides.s2r
test6.testI.atw.ak0.0005.resbcs.s2r


./inputs: observational files for test problems from the casco4b manual
inverse1.adcp_out.n0.0.uxe
inverse1.adcp_out.n0.0.uxv
inverse1.adcp_out.n0.uxe
inverse1.adcp_out.n0.uxv
inverse1.modeB.land.uxe
inverse1.modeB.land.uxv
Readme.inputs
sample_node.list
testH1.ins
testI1.ins


./mesh: Mesh geometry and bc type specification files
test6.3sides.bel
test6.4sides.bel
test6.bat
test6.bnd
test6.ele
test6.lev
test6.nod
testgeom.m
testgeom.ps

./mfiles: Matlab files for plotting casco output.
  The plotting routines should be called from within
  the "run/" directory.

edges2chains.m
plot_cbc_s2r.m
plot_contourmat.m
plot_cost_function2.m
plot_cost_function.m
plot_final_elevation_error.m
plot_final_velocity_error.m
plot_input_elevation_data.m
plot_input_velocity_data.m
plot_inverse_error.m
```

```
plot_inverse_solution.m
plot_truth.m
read_cbc.m
read_m3d.m
read_uxe.m
scattered_contour.m
suptitle.m

./run: The working directory from which the test cases can be run
casco_ModeA_basecase_justelev : script to run ModeA with elevation forcing
casco_ModeA_basecase_justvel  : script to run ModeA with velocity forcing
casco_ModeA_basecase_both     : script to run ModeA with elevation  and
         velocity forcing
casco_ModeB_basecase_both     : script to run ModeB with elevation and
         velocity forcing
casco_ModeB_basecase_justelev : script to run ModeB with elevation forcing
casco_ModeB_basecase_justvel  : script to run ModeB with velocity forcing
cleanup : script to delete output files
do_the_manual.m : runs the test cases and plots the output
Readme.basecase : information
saco_ModeA_basecase : script to run modeA TRUTH
saco_ModeB_basecase : script to run modeB TRUTH
sample_output/    the output from the runs shown in the manual
test6.3sides.bel    : bel file for modeB
test6.4sides.bel    : bel file for modeA
test6.bat           : mesh files required for model runs
test6.bnd
test6.ele
test6.lev
test6.nod


run/sample_output:  Sample output from three test cases presented
    in the manual.
run/sample_output/modeA_truth : Output from modeA truth run
run/sample_output/modeB_truth : Output from modeB truth run
run/sample_output/modeA_justelev: Output from modeA elevation inversion
run/sample_output/modeA_justvel : Output from modeA velocity inversion
run/sample_output/modeB_both : Output from modeB elevation and
                                                     velocity inversion


src: source code for casco4b and saco3
adjust_soln.f
bcs_saco_moody.f
```

```
BuildWrho3.f
casco1_fixsubs.f
casco2_sprspak.f
casco4a.f
CASCO.DIM
casco.info
casco_info.f
casco_input.f
casco_params.h
constants.h
convergence2.f
covariance.h
dbc_best_to_dbc.f
deltabcs.h
domega_write.f
elevationm4.f
elevations4.f
EMATpreMULT.f
errors.h
error_to_best.f
FEMPAKA.f
friction.h
get_bcstime.f
getdata_moody4.f
getelevobs.f
get_errvar.f
get_friction1.f
get_nsteps.f
getobs_4c.f
getobs_8c.f
get_sacotime.f
getvelobs.f
get_Wrho.f
gradcost.h
indexx.f
initial_conditions.f
inverse_parameters.h
lin_frict.f
main.tex
makefile
measure_func.f
mesh.h
misc.f
misc.h
moody_run.f
```

```
newnorms.f
numbers.f
numbers.h
observes.h
outputm2.f
outputs2.f
reorderinbe.f
rms.f
saco_basis_elev.f
saco_basis.f
saco.h
saco_run3.f
sacotime.h
set_times.f
startstop.h
start_subs.f
startup.f
statarrays.h
stationaryc3.f
station_calc2.f
statistics.f
statistics.h
stepsize5.f
sup_norm.f
time_basis.f
time_manip.f
trial_bcs_cgm.f
trial_bcs.f
vertgrids2.f
verticalm4.f
verticals5.f
weights.h
wrho.h
write_cbc.f
write_eerror.f
write_error.f
write_uverror.f


src/saco3: source code for saco3(uses code from casco)
bcs2.f
makefile
saco3.f
```