A Computer Program for Calculating Frequencies and Modal Structures of Free Coastal-Trapped Waves

by

John L. Wilkin

Woods Hole Oceanographic Institution
Woods Hole, Massachusetts  02543

December 1987

Technical Report

Funding was provided by the National Science Foundation under grant Number OCE 85-21837.

Reproduction in whole or in part is permitted for any purpose of the United States Government. This report should be cited as: Woods Hole Oceanog. Inst. Tech. Rept., WHOI-87-53.

Approved for publication; distribution unlimited.

Approved for Distribution:

Robert C. Beardsley, Chairman
Department of Physical Oceanography
A Computer Program for Calculating Frequencies
and Modal Structures of Free Coastal-Trapped Waves

John L. Wilkin
Woods Hole Oceanographic Institution
Woods Hole, Massachusetts 02543

Abstract

A listing and full documentation is presented for a FORTRAN computer pro-
gram which computes the dispersion curves and across-shelf modal struc-
tures of free coastal-trapped waves in a coastal channel. The three velocity
components, mass transport streamfunction, and density and pressure per-
turbation fields are computed. The solution procedure used (horizontal finite
differences on a staggered grid and an expansion in the vertical in terms of
modified Chebyshev polynomials) makes the solution compatible (without
interpolation) with the numerical scheme employed in the Haidvogel et al.
(1988) primitive equation ocean circulation model.
Contents

1 Introduction .................................................. 3

2 Theory and Problem Formulation ............................. 4
   2.1 Governing Equation and Boundary Conditions .......... 4
   2.2 Transformation to Vertical Sigma Coordinates .......... 6
   2.3 Computing the Other Variables .......................... 7

3 Numerical Discretization and Solution ...................... 7
   3.1 Chebyshev Polynomial Basis Set .......................... 7
   3.2 Horizontally Staggered Grid .............................. 8
   3.3 Implementation .......................................... 9

4 How to Run the Program and what it will tell you .......... 10
   4.1 Units ..................................................... 10
   4.2 Altering the Grid Resolution ............................. 10
   4.3 Across-shelf Coordinate Stretching ....................... 10
   4.4 Depth Profile ............................................ 11
   4.5 Stratification ............................................ 11
   4.6 Input ...................................................... 11
      4.6.1 Interactive .......................................... 11
      4.6.2 Non-Interactive ...................................... 12
   4.7 Output .................................................... 13
   4.8 An example for you to try ................................ 14

5 Performance and Testing ...................................... 15

6 Using CTWEIG in Conjunction with Another Program ....... 19
   6.1 Saving the Output from CTWEIG ......................... 19
   6.2 Subroutine CTWAVE ....................................... 19
   6.3 Getting the Phase Right .................................. 21

7 References .................................................. 23

8 Listing of Program CTWEIG .................................. 24
1 Introduction

This report describes the design and operation of a computer program which finds the eigenfrequencies and corresponding across-shelf modal structures of the velocity, pressure, and density perturbation of coastal-trapped waves (CTWs). Numerical solutions to this problem have previously been obtained by Wang and Mooers (1976), Huthnance (1978) and Brink (1982) and the computer code used in the latter study is available to the oceanographic community in Brink and Chapman (1987); why then do we need another?

Dr. Dale Haidvogel of Johns Hopkins University has recently developed a four-dimensional \((x, y, z, t)\) primitive equation ocean circulation model which incorporates arbitrary stratification, variable bottom topography and irregular horizontal geometry. These features make the model particularly suitable for studies of processes on the continental shelf and slope. Since any transient low-frequency forcing on the continental shelf will generate CTWs, a knowledge of the across-shelf structures of CTWs is desirable to aid the interpretation of any numerical experiment in coastal oceanography.

Haidvogel's model uses a horizontally staggered finite-difference grid on which different field variables are specified at physically different "half" grid points. It also uses a spectral representation of the vertical dependence which is fundamentally different from the finite-difference representation of Brink and Chapman (1987). Interpolating the across-shelf structures returned by the program of Brink and Chapman (1987) onto a grid compatible with Haidvogel's numerical scheme would create undesirable numerical errors. The program described here was therefore written to be compatible with Haidvogel's model. The CTW eigenvalue problem is solved by resonance iteration using centered finite differences on a staggered grid in the horizontal. The horizontal coordinate can be stretched arbitrarily making it possible to refine the grid resolution at any across-shelf location. A modified sigma coordinate system and Chebyshev spectral representation is employed in the vertical. The program is less general than that of Brink and Chapman (1987) in three regards:

- A rigid-lid surface boundary condition is employed (as in Haidvogel et al., 1988).
- It does not compute long-wave properties.
- A solid wall is placed at the offshore boundary (as in Haidvogel et al., 1988).

The code could easily be modified to relax the latter point to incorporate a more realistic offshore boundary condition.

The CTW eigenvalue problem is formulated in section 2 with a description of the horizontal \((\eta)\) and vertical \((\sigma)\) coordinate transformations. Section 3 contains a brief outline of the numerical discretization and solution procedure. Instructions for running the program are presented in section 4 and the results of tests of its performance are summarized in section 5. Section 6 demonstrates how the modal structures can be recomputed and utilized by an application program.
2 Theory and Problem Formulation

2.1 Governing Equation and Boundary Conditions

The dynamics of coastal-trapped wave propagation are governed by the linearized, inviscid, hydrostatic Boussinesq equations which may be written

\[
\frac{\partial}{\partial t} \left( \frac{u}{mn} \right) - \frac{f}{mn} v = -\frac{1}{n} \frac{\partial \phi}{\partial \xi} \tag{1}
\]

\[
\frac{\partial}{\partial t} \left( \frac{v}{mn} \right) + \frac{f}{mn} u = -\frac{1}{m} \frac{\partial \phi}{\partial \eta} \tag{2}
\]

\[
\frac{\partial \phi}{\partial z} = -\frac{\rho g}{\rho_0} \tag{3}
\]

\[
\frac{\partial \rho}{\partial t} + w \frac{\partial \rho}{\partial z} = 0 \tag{4}
\]

\[
mn \left( \frac{\partial}{\partial \xi} \left( \frac{u}{n} \right) + \frac{\partial}{\partial \eta} \left( \frac{v}{m} \right) \right) + \frac{\partial w}{\partial z} = 0 \tag{5}
\]

where \(u\) and \(v\) are the velocity components in the direction of the generalized orthogonal curvilinear coordinates \(\xi\) and \(\eta\) respectively, \(w\) is the vertical \((z)\) velocity component, \(t\) is time, \(f\) the Coriolis frequency, \(\phi = p/\rho_0\) where \(p\) is the perturbation pressure, and the density is

\[
\rho_{\text{total}} = \rho_0 + \bar{\rho}(z) + \rho(x, y, z, t) \tag{6}
\]

It is assumed that \(\rho_0 \gg \bar{\rho} \gg \rho\).

The metric coefficients \(m\) and \(n\) of the orthogonal curvilinear coordinate system relate differential distances in the \(\xi, \eta\) directions to actual physical arc lengths

\[
(ds)_\xi = \frac{d\xi}{m} \quad \text{and} \quad (ds)_\eta = \frac{d\eta}{n} \tag{7}
\]

A line element of length \(ds\) therefore satisfies

\[
ds^2 = \frac{d\xi^2}{m^2} + \frac{d\eta^2}{n^2} \tag{8}
\]

In terms of cartesian coordinates \(x, y\) the line element may also be written

\[
ds^2 = dx^2 + dy^2 \tag{9}
\]

\[
= \left( \frac{\partial x}{\partial \xi} d\xi + \frac{\partial x}{\partial \eta} d\eta \right)^2 + \left( \frac{\partial y}{\partial \xi} d\xi + \frac{\partial y}{\partial \eta} d\eta \right)^2 \tag{10}
\]

which simplifies to

\[
ds^2 = \left[ \left( \frac{\partial x}{\partial \xi} \right)^2 + \left( \frac{\partial y}{\partial \xi} \right)^2 \right] d\xi^2 + \left[ \left( \frac{\partial x}{\partial \eta} \right)^2 + \left( \frac{\partial y}{\partial \eta} \right)^2 \right] d\eta^2 \tag{11}
\]
provided \( \frac{\partial x}{\partial \xi} \frac{\partial x}{\partial \eta} + \frac{\partial y}{\partial \xi} \frac{\partial y}{\partial \eta} = 0 \), i.e. \( \xi \) and \( \eta \) are orthogonal. Comparison of (8) and (11) shows how \( m \) and \( n \) can be computed from cartesian coordinates defined as functions of \( \xi \) and \( \eta \).

\[
m = \left( \left( \frac{\partial x}{\partial \xi} \right)^2 + \left( \frac{\partial y}{\partial \xi} \right)^2 \right)^{-\frac{1}{2}}
\]

(12)

\[
n = \left( \left( \frac{\partial x}{\partial \eta} \right)^2 + \left( \frac{\partial y}{\partial \eta} \right)^2 \right)^{-\frac{1}{2}}
\]

(13)

Defining the Brunt-Väisälä frequency profile \( N(x) \) by \( N^2(x) = -\frac{\xi}{\rho_0 \frac{\partial}{\partial z}} \) and seeking periodic solutions of the form \( e^{-i\omega t} \), equations (1) through (4) can be manipulated to give expressions for the three velocity components

\[
u = \frac{1}{f^2 - \omega^2} \left( i \omega n \frac{\partial \phi}{\partial \eta} + f m \frac{\partial \phi}{\partial \xi} \right)
\]

(14)

\[
v = \frac{1}{f^2 - \omega^2} \left( i \omega n \frac{\partial \phi}{\partial \eta} + f m \frac{\partial \phi}{\partial \xi} \right)
\]

(15)

\[
w = \omega \frac{1}{N^2} \frac{\partial \phi}{\partial z}
\]

(16)

which when substituted into (5) give a single equation for the pressure

\[
m n \frac{\partial}{\partial \eta} \left( \frac{n}{m} \frac{\partial \phi}{\partial \eta} \right) + (f^2 - \omega^2) \frac{\partial}{\partial z} \left( \frac{1}{N^2} \frac{\partial \phi}{\partial z} \right) + m n \frac{\partial}{\partial \xi} \left( \frac{m}{n} \frac{\partial \phi}{\partial \xi} \right) = 0
\]

(17)

Solutions to (17) which represent waves of fixed wavenumber \( k \) travelling along a straight coast are obtained by adopting rectangular \( \xi, \eta \) coordinates which lie parallel to cartesian coordinates \( x, y \) aligned along the shelf and offshore respectively. In such a system the metric coefficients \( m, n \) are independent of \( \eta, \xi \) respectively so that \( m = (\frac{\partial x}{\partial \xi})^{-1}, n = (\frac{\partial x}{\partial \eta})^{-1} \), and the line element in the \( x \) direction, \( dx \), becomes \( dx = \frac{1}{m} \frac{\partial \xi}{\partial \xi} \). The operator \( m \frac{\partial}{\partial \xi} \) is then simply \( \frac{\partial}{\partial z} \) which for progressive waves of the form \( e^{i(kz - \omega t)} \) becomes \( ik \). Equation (17) then simplifies to

\[
n \frac{\partial}{\partial \eta} \left( \frac{n}{m} \frac{\partial \phi}{\partial \eta} \right) + (f^2 - \omega^2) \frac{\partial}{\partial z} \left( \frac{1}{N^2} \frac{\partial \phi}{\partial z} \right) - k^2 \phi = 0
\]

(18)

The boundary conditions on (18) are

- no flow through the coast or the solid wall placed at the offshore extent of the numerical domain \( (v = 0 \text{ at } \eta = 0, \eta_{max}) \)

- no vertical velocity at the rigid lid \( (w = 0 \text{ at } z = 0) \)

- no flow through the bottom \( (n \frac{\partial}{\partial \eta} v + w = 0 \text{ at } z = -h) \)

where the depth \( h \) is assumed to vary in \( \eta \) only. Using equations (14) through (16) for the velocity components, these boundary conditions may be restated in terms of the pressure \( \phi \) as

\[
n \frac{\partial}{\partial \eta} \left( \frac{n}{m} \phi \right) + f \frac{k}{\omega} \phi = 0 \text{ at } \eta = 0, \eta_{max}
\]

(19)
\[
\frac{\partial \phi}{\partial z} = 0 \quad \text{at} \quad z = 0 \quad (20)
\]
\[
n\frac{\partial h}{\partial \eta} \left( n \frac{\partial \phi}{\partial \eta} + \frac{f k}{\omega} \phi \right) + \frac{f^2 - \omega^2}{N^2} \frac{\partial \phi}{\partial z} = 0 \quad \text{at} \quad z = -h \quad (21)
\]

For a given \( h, N^2, f, \) and wavenumber \( k, \) there is a sequence of solutions to (18) subject to (19) through (21) at discrete eigenfrequencies \( \omega_1, \omega_2, \ldots, \omega_n \) which are free coastal-trapped waves.

### 2.2 Transformation to Vertical Sigma Coordinates

It is computationally convenient to adopt a stretched vertical coordinate which effectively flattens out the variable bottom topography at \( z = -h(\eta). \) The new vertical coordinate \( \sigma \) is defined by

\[
\sigma = 1 + 2 \frac{z}{h(\eta)} \quad (22)
\]

so that the surface becomes \( \sigma = 1 \) and the bottom \( \sigma = -1. \) The differential operators in the original coordinates \( (\eta, z) \) transform to the following operators in the new sigma coordinates \( (\eta, \sigma). \)

\[
\frac{\partial}{\partial z} \rightarrow \frac{2}{h} \frac{\partial}{\partial \sigma} \quad (23)
\]
\[
\frac{\partial}{\partial \eta} \rightarrow \frac{\partial}{\partial \eta} + \frac{1 - \sigma}{h} \frac{\partial h}{\partial \eta} \frac{\partial}{\partial \sigma} \quad (24)
\]

from which it is clear that the simplification in the geometry of the problem is gained at the expense of complicating the equations to be solved.

Applying the transformed operators (23) and (24) to (18) through (21) and dropping the carats yields the set of equations to be solved.

\[
n \frac{\partial}{\partial \eta} \left( n \frac{\partial \phi}{\partial \eta} + (1 - \sigma) \frac{n \partial h}{h} \frac{\partial \phi}{\partial \sigma} \right) + (1 - \sigma) \frac{n \partial h}{h} \frac{\partial}{\partial \eta} \left( n \frac{\partial \phi}{\partial \eta} + (1 - \sigma) \frac{n \partial h}{h} \frac{\partial \phi}{\partial \sigma} \right) + \frac{f^2 - \omega^2}{N^2} \frac{\partial^2 \phi}{\partial \sigma^2} - k^2 \phi = 0 \quad (25)
\]

subject to boundary conditions

\[
n \frac{\partial \phi}{\partial \eta} + (1 - \sigma) \frac{n \partial h}{h} \frac{\partial \phi}{\partial \sigma} + \frac{f k}{\omega} \phi = 0 \quad \text{at} \quad \eta = 0, \eta_{\text{max}} \quad (26)
\]
\[
\frac{\partial \phi}{\partial \sigma} = 0 \quad \text{at} \quad \sigma = 1 \quad (27)
\]
\[
n \frac{n^2 \partial h}{\partial \eta} \frac{\partial \phi}{\partial \eta} + \frac{2}{h} \left[ \left( \frac{n \partial h}{\partial \eta} \right)^2 + \frac{f^2 - \omega^2}{N^2} \right] \frac{\partial \phi}{\partial \sigma} + n \frac{\partial h}{\partial \eta} \frac{f k}{\omega} \phi = 0 \quad \text{at} \quad \sigma = -1 \quad (28)
\]

Note that the Brunt-Väisälä frequency, assumed to be a function of \( z \) only, will vary in both \( \eta \) and \( \sigma \) as a result of the coordinate mapping.
A vertical velocity in sigma coordinates (employed in Haidvogel’s primitive equation model) may also be defined

\[ \Omega(\xi, \eta, \sigma, t) = \frac{1}{h} \left[ (1 - \sigma) n \frac{\partial h}{\partial \eta} v + 2w(\xi, \eta, z, t) \right] \]  

(29)

in terms of which the top and bottom boundary conditions are simply \( \Omega = 0 \) at \( \sigma = \pm 1 \).

2.3 Computing the Other Variables

Applying the \( \sigma \)-transformation to (14) through (16) yields equations from which the alongshelf velocity \( u \), across-shelf velocity \( v \) and vertical velocity \( w \) can be evaluated. The vertical velocity in \( \sigma \) coordinates (29) is evaluated directly from the solutions for \( v \) and \( w \). The density \( \rho \) is computed from (3). The program outputs both the total pressure (barotropic component included) and the baroclinic pressure (surface pressure removed, i.e. \( \phi - \phi|_{\sigma=1} \)). It is the latter which is consistent with the definition used by Haidvogel et al. (1988) and is stored as variable PPHI in the common block /SOL/. The transport streamfunction \( \psi \), defined as

\[ n \frac{\partial \psi}{\partial \eta} = -h \bar{u} ; \quad m \frac{\partial \psi}{\partial \xi} = ik \psi = h \bar{v} \]  

(30)

where the overbar denotes a depth average, is computed by depth-averaging the solution for \( u \).

3 Numerical Discretization and Solution

3.1 Chebyshev Polynomial Basis Set

The vertical (\( \sigma \)) dependence of the field variables is represented as an expansion in the polynomial set \( P_k(\sigma) \)

\[ \phi(\eta, \sigma) = \sum_{k=0}^{N} \hat{\phi}_k(\eta) P_k(\sigma) \]  

(31)

The \( P_k(\sigma) \) are a modified form of the Chebyshev polynomials of the first kind \( (T_k(\sigma)) \) described by Abramowitz and Stegun (1972, chapter 22).

\[ P_k(\sigma) = \begin{cases} 
T_0(\sigma) & k = 0 \\
T_k(\sigma) & k \geq 1, \ k \ odd \\
T_k(\sigma) + \frac{1}{k^2 - 1} & k \geq 2, \ k \ even
\end{cases} \]

(32)

The numerical technique employed does not explicitly solve for the polynomial coefficients \( \hat{\phi}_k \) but instead solves for the actual variable values \( \phi_n \) at the “collocation” points (or equivalent grid points) \( \sigma_n \) corresponding to the locations of the extrema of the Chebyshev basis set

\[ \sigma_n = \cos(\pi(n - N)/N) \quad 0 \leq n \leq N \]  

(33)

The expansion coefficients \( \hat{\phi}_k \) can be recovered from the variable values at the \( N+1 \) collocation points \( \phi(\sigma_n) \) by a simple matrix transformation. Defining \( \hat{\phi} \) to be a vector of the expansion
coefficients and \( \vec{\phi} \) to be a vector of the variable values at the collocation points, it may be seen that a matrix \( F \) defined as

\[
F = \begin{pmatrix}
P_0(-1) & P_1(-1) & \cdots & P_N(-1) \\
P_0(\sigma_1) & P_1(\sigma_1) & \cdots & P_N(\sigma_1) \\
\vdots & \vdots & \ddots & \vdots \\
P_0(1) & P_1(1) & \cdots & P_N(1)
\end{pmatrix}
\]  

(34)

satisfies \( \vec{\phi} = F\vec{\phi} \) and hence that \( \vec{\phi} = F^{-1}\vec{\phi} \). It is now straightforward to compute matrices which will operate on a vector of variable values at the collocation points to produce vectors of derivatives or integrals of those values.

Consider the matrix \( D \)

\[
D = \begin{pmatrix}
P_0'(-1) & P_1'(-1) & \cdots & P_N'(-1) \\
P_0'(\sigma_1) & P_1'(\sigma_1) & \cdots & P_N'(\sigma_1) \\
\vdots & \vdots & \ddots & \vdots \\
P_0'(1) & P_1'(1) & \cdots & P_N'(1)
\end{pmatrix}
\]  

(35)

where the primes denote differentiation with respect to \( \sigma \). A differentiating matrix \( C_{DZ} \) can be formed by taking the product \( C_{DZ} = D \cdot F^{-1} \) since

\[
\frac{\partial}{\partial \sigma} \vec{\phi} = \sum_{k=0}^{N} \frac{\partial}{\partial \sigma} P_k(\sigma) \phi_k = D\vec{\phi} = D \cdot F^{-1}\vec{\phi} = C_{DZ}\vec{\phi}
\]  

(36)

Forming the matrix \( C_{\text{INT}} \)

\[
C_{\text{INT}} = \begin{pmatrix}
\int_{-1}^{1} P_0 \, d\sigma & \int_{-1}^{1} P_1 \, d\sigma & \cdots & \int_{-1}^{1} P_N \, d\sigma \\
\int_{\sigma_1}^{1} P_0 \, d\sigma & \int_{\sigma_1}^{1} P_1 \, d\sigma & \cdots & \int_{\sigma_1}^{1} P_N \, d\sigma \\
\vdots & \vdots & \ddots & \vdots \\
\int_{1}^{1} P_0 \, d\sigma & \int_{1}^{1} P_1 \, d\sigma & \cdots & \int_{1}^{1} P_N \, d\sigma
\end{pmatrix} F^{-1}
\]  

(37)

produces an integrating operator

\[
\int_{\sigma_n}^{1} \phi(\sigma) \, d\sigma = C_{\text{INT}}\vec{\phi}
\]  

(38)

### 3.2 Horizontally Staggered Grid

The numerical scheme of Haidvogel et al. (1988) employs an Arakawa C horizontal grid where the variables \( u, v, \psi \) and \( \Omega, \phi, \rho, h \) are specified at physically different half grid points (Figure 1).

In order that the solution obtained here be compatible with this scheme it is necessary to follow the same convention for specifying where the variables are defined. This requires that the grid spacing be uniform in the stretched \( \eta \) coordinate (i.e. \( \Delta \eta = \text{constant} \)) and that the coast (\( \eta = 0 \)) coincide with the points at which \( v \) and \( \psi \) are defined. Thus the notation \( \phi_{k,j} \) refers to the
value of $\phi$ at $\eta_j = (j - \frac{1}{2})\Delta\eta$, $\sigma = \sigma_k$ while $v_{k,j}$ refers to $v$ at $\eta_j = (j - 1)\Delta\eta$, $\sigma = \sigma_k$. Since $n = \left(\frac{3y}{\partial n}\right)^{-1}$, if $\Delta\eta$ is chosen to be unity the metric coefficients $n$ will have units $\text{length}^{-1}$ and represent the reciprocal of the actual distance separating two adjacent grid points$^1$. Since the CTW eigenvalue problem is only two-dimensional, the alongshelf displacement of the $u$ and $\psi$ points on an Arakawa grid is not explicitly accommodated and must be accounted for using the known $e^{i(kz-\omega t)}$ alongshelf structure when using the CTW solutions in conjunction with a three-dimensional model. Horizontal derivatives are approximated by using simple finite differences centered about the half grid points.

3.3 Implementation

The first step in solving the free wave eigenvalue problem for a particular wavenumber/frequency pair $(k, \omega)$ is the formulation of a matrix equation for a combined vector of all $\phi_{k,j}$ values. The set of equations at each horizontal grid point $j = 1, 2 \ldots M-1$ has the form

$$A_j \tilde{\phi}_{j-1} + B_j \tilde{\phi}_j + C_j \tilde{\phi}_{j+1} = 0$$

(39)

where $\tilde{\phi}_j$ is a vector of values of $\phi$ at the $k = 0, 1 \ldots N$ collocation levels, at $\eta = \eta_j$. Each of the square matrices $A_j, B_j$ and $C_j$ is therefore of dimension $N+1$. A discretized form of the bottom boundary condition (28) is entered in the first row of each matrix; the surface boundary

$^1$Conversely, setting the metric coefficients $n$ to unity everywhere and computing $\Delta\eta$ as the channel width divided by the number of grid points would produce a $\Delta\eta$ corresponding to a conventional finite difference grid spacing having units length.
condition (27) is entered in the last row of each matrix; and the governing equation (25) in the remaining rows. Since the grid points at which the $\phi$ values are defined do not coincide with the channel walls, discretization of the boundary conditions (26) at the channel walls leads to two matrix equations of the form

$$B_0 \tilde{\phi}_0 + C_0 \tilde{\phi}_1 = 0$$  \hspace{1cm} (40)

$$A_M \tilde{\phi}_{M-1} + B_M \tilde{\phi}_M = 0$$  \hspace{1cm} (41)

for the vectors of $\phi$ values on either side of the walls at $\eta = 0$ and $\eta = \eta_{max}$. These two sets of equations complete a block tri-diagonal system of equations which is solved using subroutine NBTRPD, a double precision version of a subroutine taken from appendix B of Anderson et al. (1984). Should you wish to apply a different offshore boundary condition than that used here (e.g. the condition used by Brink and Chapman (1987) that $\frac{\partial u}{\partial \eta} = 0$ at $\eta = \eta_{max}$), then the section of the computer code in subroutine PHIEQN corresponding to (41) must be altered. The appropriate section is indicated in the program listing on page 40.

The equations, as stated, are homogeneous. To obtain a nontrivial solution, a single element in the right-hand-side vector is assigned a nonzero value. If the chosen wavenumber/frequency pair is close to that of a free wave, a strong response to this weak forcing will occur. The program finds the frequency corresponding to a specified wavenumber by calculating a degree of resonance

$$RI = \int_0^\eta_{max} \int_0^{\eta_{max}} \frac{\partial \tilde{\phi}}{\partial \eta} \tilde{n}^{-1} d\eta \, dz$$  \hspace{1cm} (42)

and iterating on values of $\omega$ until $RI$ takes a local maximum. The alternative of specifying $\omega$ and searching for the corresponding $k$ is available. The mode number will be the number of zero crossings in $\phi$ along the bottom.

4 How to Run the Program and what it will tell you

4.1 Units

The program assumes SI units throughout. If you haven’t yet converted it’s time you did.

4.2 Altering the Grid Resolution

The grid resolution is specified by the first PARAMETER statement in each program module. $M$ is the number of horizontal finite difference grid points and $N$ is one less than the number of vertical Chebyshev modes. To alter the grid resolution simply alter the first PARAMETER statement in every program module. (There are 7 of them.)

4.3 Across-shelf Coordinate Stretching

The mapping between the cartesian coordinate $y$ and the generalized curvilinear coordinate $\eta$ is specified by a function subprogram YF(ETA). Calls to YF are made to initialize an array ($YN$)
of the values of $y$ at the $2M-1$ half grid points $\eta = 0, \frac{1}{2} \Delta \eta, \Delta \eta \ldots (M-1) \Delta \eta$. Using the relation $n = (\frac{\partial y}{\partial \eta})^{-1}$, these values are differenced to give the metric coefficients $n$ which are entered in array PN. In the version of the program presented in section 8 the coordinate mapping is simply $y = \frac{\eta}{\eta_{max}} \times$ (channel width) so that the $n$ factors are constant and the grid spacing in $y$ is uniform. Should you wish to employ the grid stretching facility of the program to refine the grid spacing at a particular across-shelf location, this is easily done.

For example, examine the listing of FUNCTION YF(ETA) on pages 31-32. Two lines are commented out. The first is $y = \frac{1}{2} L [1 + \tan(\frac{\eta}{4}(2 \frac{n}{\eta_{max}} - 1))]$ which refines the grid spacing in the middle of the channel. The second is the mapping $y = L(e^{n/\eta_{max}} - 1)/(e - 1)$ which refines the grid spacing near the coast at $\eta = 0$.

### 4.4 Depth Profile

The across-shelf profile of the depth (in meters) is entered in array H by a series of calls to function subprogram HF(Y). You will need to write your own version of HF to specify the topography you wish to study. Note that the argument of HF is not $\eta$ but rather the cartesian coordinate $y$. This eliminates any necessity for changes in HF should you choose to run the program for a variety of horizontal coordinate mappings.

### 4.5 Stratification

The vertical profile of the Brunt-Väisälä frequency squared ($N^2$) in the deep ocean is determined by another function subprogram that you must write yourself and link to the main program. Function BVFSQ(Z) returns the value of $N^2$ (in rad$^2$s$^{-2}$) as a function of the vertical cartesian coordinate $z$.

### 4.6 Input

The program can be run in two modes, interactive and noninteractive.

**4.6.1 Interactive**

In interactive mode, FORTRAN unit ITERM is used for terminal input and FORTRAN unit JTERM for terminal output. ITERM and JTERM are assigned their values in the first executable statements in the program. In the version of the program presented in section 8, they are assigned the values 5 and 6 respectively. The interactive mode is invoked by executing the program and typing T (for terminal) followed by a carriage return. Thereafter you will be prompted for the remaining input in free format and reminded of the units required. This mode is useful for becoming familiar with the input required and is *reasonably* forgiving if you enter something foolish, but is by no means idiot proof. But since you’re a coastal oceanographer you’re probably pretty smart and can figure it out. For a line by line description of the input … read on.
4.6.2 Non-Interactive

In noninteractive mode, FORTRAN unit ITERM is the input data file. The first record in the file must be F (for file). This will suppress the output of the interactive prompts.

The input records should be

1. 'F' for file as explained above.

2. F,YL,YLS
   
   • F is the Coriolis frequency \((rad\ s^{-1})\).
   
   • YL is the channel width \(L\) in meters. In general \(L\) will be required by function subprogram yf(eta) when computing the \(y(\eta)\) coordinate mapping. To minimize the effect of having a wall rather than an open boundary at \(\eta_{max}\), \(L\) should be large enough that the region of shelf topography occupies no more than about \(\frac{1}{3}L\), i.e. \(\frac{2}{3}\) of the channel is flat bottom ocean.
   
   • YLS is the distance \(L_s\) from the coast to the region of flat bottom ocean (in meters). It represents a typical across-shelf length scale and is used only in computing the stratification parameter \(S = N^2_{max} h^2_{max}/f^2 L_s^2\).

3. Either 'W' if you wish to specify the wavenumber and search for a corresponding free wave frequency, or 'K' to specify frequency and search for a corresponding wavenumber.

4. Either XX the wavenumber \((m^{-1})\) or W the frequency \((rad\ s^{-1})\) depending on the option selected above.

5. WXRNG,DW
   
   • WXRNG is a starting guess at the wave frequency \((rad\ s^{-1})\) or wavenumber \((m^{-1})\). The program searches for the free wave solution nearest to this value.
   
   • DW is the initial search increment \((rad\ s^{-1}\ or\ m^{-1})\).

6. WTOL is the fractional tolerance to which the solution is determined. When the difference between two successive frequency estimates \(|\omega_{new} - \omega_{old}|\) is less than \(WTOL \times \omega_{new}\) (or similarly for \(k\)), convergence is deemed to have occurred. It should be noted that the discretized equations are ill-conditioned near a resonant wavenumber/frequency pair so that specifying a value of WTOL which is less than roughly \(10^{-4}\) is likely to cause an "arithmetic fault" or "divide by zero" error in subroutine NBTRIPD. In general, WTOL = \(10^{-3}\) is quite adequate for accurately determining the solution.

7. ITMAX is the maximum number of iterations to be allowed.

8. The next set of records is a list of the field variables to be output. The options, to be entered one per line with a blank line to end the list, are:
• P for the pressure $\phi = p/\rho_0$ (barotropic component included)
• U for alongshelf velocity $u$
• V for across-shelf velocity $v$
• W for vertical velocity $w$
• WSIG for vertical velocity $\Omega$ in sigma coordinates
• PSI for streamfunction $\psi$
• RHO for density perturbation $\rho$
• PHI for "baroclinic" pressure $\phi - h|_{\sigma=1}$
• or simply ALL to output all of the above.
• PLT to create modal structure plots (see the listing of subroutine CTWPLLOT for more information).
• don’t forget the blank line

9. NKPTS. This input is only required if the frequency search option was selected. It is the number of $k, \omega$ points to calculate on the dispersion curve.

10. RKMAX the maximum value of the wavenumber if a dispersion curve is being computed. This input is not required if NKPTS = 1.

4.7 Output

The output device (in addition to the terminal if interactive mode is used) is FORTRAN file IDS\text{SK} which is currently set to 7. This can be changed by altering the assignment statement IDS\text{SK} = 7 following the ITERM and JTERM assignments on page 25 of the program listing.

The program outputs a summary of the input parameters and then information on the wavenumber/frequency search including $\omega, k, c$ (phase speed) and the degree of resonance $RI$. Then the requested variable solutions are output beginning at $\eta = 0$ or $\eta = \frac{1}{2} \Delta \eta$ according to the half grid points on which they are specified, with each row being a profile of the values from $\sigma = -1$ (bottom) to $\sigma = 1$ (surface). It should be noted that the across-shelf structures of $v, w$ and $\Omega$ are in quadrature with all the other variables. If the solutions for $u, \psi, \rho$ and $\phi$ are taken to be real-valued, then the solutions as output for $v, w$ and $\Omega$ must be multiplied by $i = \sqrt{-1}$, to obtain the correct phase relationship. This is explained in more detail in section 6.3. If a dispersion curve is to be computed, frequency search information for each point will be output followed by a summary of the solution.

The solutions have all been normalized such that

$$1 = \int_{-h}^{0} \phi^2|_{\eta=0} \, dz + \int_{0}^{\infty} \frac{\partial h}{\partial \eta} \phi^2|_{\sigma=-h} \, d\eta = \int_{-1}^{1} \frac{1}{2} \frac{\partial h}{\partial \eta} \phi^2|_{\eta=0} \, d\sigma + \int_{0}^{\eta_{max}} \frac{\partial h}{\partial \eta} \phi^2|_{\sigma=-1} \, d\eta$$

13
where $\phi$ is the *total* pressure (i.e. barotropic component included). An alternative normalization, which gives a solution having unit energy flux, can be enabled by restoring several lines of code presently commented out in subroutine UV (see the listing at the bottom of page 42).

### 4.8 An example for you to try

The version of CTWEIG presented in section 8 has routines *bvfsq, hf* and *yf* which describe an ocean with constant $N^2$, a linearly sloping shelf and constant metric factors $n$. Given the input parameters:

\[
\begin{align*}
T & \quad 1.0E-04, \quad 300.0E+03, \quad 100.0E+03 \\
W & \quad 1.0E-06 \\
& \quad 3.6E-06, \quad 0.2E-06 \\
& \quad 0.001 \\
10 & \quad (a \ blank \ line) \\
9 & \quad 2.0E-05
\end{align*}
\]

the program should output:

**Solution to the coastal-trapped wave eigenvalue problem in a coastal channel.**

Numerical resolution is $M = 31$ horizontal finite difference grid points and $7$ vertical Chebyshev modes, i.e. $N = 6$

- Coriolis frequency = $1.000000D-04$ rad/s
- Channel width = $3.000000D+02$ km
- Brunt-Vaisala frequency profile (rad/s)$^{**2}$:
  - Depth (m) $\quad -4.00000D+03 \quad -3.7321D+03 \quad -3.0000D+03 \quad -2.0000D+03 \quad -1.0000D+03 \quad -2.6795D+02 \quad 0.0000D+00$
  - $N^{**2}$ $\quad 6.2500D-06 \quad 6.2500D-06 \quad 6.2500D-06 \quad 6.2500D-06 \quad 6.2500D-06 \quad 6.2500D-06$
  - $S = (NH/$fl$)^{**2} = 1.0000D+00$

<table>
<thead>
<tr>
<th>Iteration</th>
<th>omega</th>
<th>k</th>
<th>c</th>
<th>int p**2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.6000000000D-06</td>
<td>1.0000000000D-06</td>
<td>3.600000D+00</td>
<td>6.24415D+19</td>
</tr>
<tr>
<td>2</td>
<td>3.8000000000D-06</td>
<td>1.0000000000D-06</td>
<td>3.800000D+00</td>
<td>1.15553D+21</td>
</tr>
<tr>
<td>3</td>
<td>4.0000000000D-06</td>
<td>1.0000000000D-06</td>
<td>4.000000D+00</td>
<td>3.81217D+20</td>
</tr>
<tr>
<td>4</td>
<td>3.8792070324D-06</td>
<td>1.0000000000D-06</td>
<td>3.87921D+00</td>
<td>5.33247D+22</td>
</tr>
<tr>
<td>5</td>
<td>3.8727478751D-06</td>
<td>1.0000000000D-06</td>
<td>3.87275D+00</td>
<td>3.58216D+23</td>
</tr>
</tbody>
</table>
Successful convergence

followed by a listing of the modal structures and a summary of the dispersion relation:

<table>
<thead>
<tr>
<th>wavenumber (/m)</th>
<th>frequency (rad/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00000E-06</td>
<td>3.86915E-06</td>
</tr>
<tr>
<td>3.37500E-06</td>
<td>1.28297E-05</td>
</tr>
<tr>
<td>5.75000E-06</td>
<td>2.11489E-05</td>
</tr>
<tr>
<td>8.12500E-06</td>
<td>2.88451E-05</td>
</tr>
<tr>
<td>1.05000E-05</td>
<td>3.53146E-05</td>
</tr>
<tr>
<td>1.28750E-05</td>
<td>4.12433E-05</td>
</tr>
<tr>
<td>1.52500E-05</td>
<td>4.65388E-05</td>
</tr>
<tr>
<td>1.76250E-05</td>
<td>5.12663E-05</td>
</tr>
<tr>
<td>2.00000E-05</td>
<td>5.55006E-05</td>
</tr>
</tbody>
</table>

5 Performance and Testing

The accuracy of the program has been checked by comparisons with known solutions for three cases:

1. barotropic shelf waves in a coastal channel with exponential topography

2. baroclinic Kelvin waves in a flat bottom ocean

3. the numerical results of Huthnance (1978) for a shelf of constant slope in an ocean of constant $N^2$.

Table 1 compares the computed and theoretical phase speeds of barotropic shelf waves for the case $h(y) = 75 \exp(2\lambda y)$ meters, $\lambda = 2 \times 10^{-5} m^{-1}$, $f = 10^{-4} s^{-1}$, $k = 10^{-5} m^{-1}$, and $N^2 = 10^{-10} s^{-2}$ in a channel of width 100 km using $M=51$ and $N=3$. Note that the program cannot accommodate $N^2 \equiv 0$ but barotropic conditions can be achieved by choosing $N^2$ sufficiently small that $S = N^2_{\text{max}} h^2_{\text{max}} / f^2 L^2 \ll 1$.

Table 2 presents the computed solutions for the phase speeds and for the pressure and alongshelf velocity at the grid point nearest to the coast for baroclinic Kelvin waves with $N^2 = 6.25 \times 10^{-5} s^{-2}$, $h = 4000 m$ and $f = 10^{-4} s^{-1}$. The values were computed with a grid resolution of $M=51$ and $N=9$ in a channel 100 km wide. The percent errors show that the computed solutions are in excellent agreement with theoretical values.
<table>
<thead>
<tr>
<th>Mode</th>
<th>phase speed ((ms^{-1}))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Computed</td>
</tr>
<tr>
<td>1</td>
<td>2.6931</td>
</tr>
<tr>
<td>2</td>
<td>0.9018</td>
</tr>
<tr>
<td>3</td>
<td>0.4283</td>
</tr>
</tbody>
</table>

Table 1: Comparison of phase speeds computed with the present model to theoretical values for barotropic shelf waves in a coastal channel with \(h(y) = 75 \exp(2\lambda y) \text{ meters, } \lambda = 2 \times 10^{-6}m^{-1}, f = 10^{-4}s^{-1} \) and \( k = 10^{-5}m^{-1} \).

Huthnance’s (1978) results were computed using the offshore boundary condition that the shelf abuts a semi-infinite ocean of constant depth whereas the present model assumes a solid wall at the offshore boundary of the numerical domain. As a result, the phase speeds computed by the program (using a channel four times as wide as the shelf) are slightly less than those computed by Huthnance. The comparison is made in Table 3 which includes, for completeness, the solutions computed using Brink and Chapman’s program. To verify that the solutions are in good agreement, the alongshelf and across-shelf velocity components for the mode 2, \( S = 1 \) case and the across-shelf velocity component for the mode 2, \( S = 10 \) case are plotted in Figure 2 along with the corresponding plots from Huthnance (1978).

For a grid resolution of \( M=51 \) and \( N=9 \) the program requires 9.5 seconds of CPU time per iteration on a MicroVAX II. For a reasonable starting guess it will converge to a fractional tolerance of 0.001 in roughly 6 iterations. This compares very favorably with Brink and Chapman’s program which typically takes 9 iterations to converge for the same initial guess and tolerance and requires 21 seconds of CPU time per iteration on a grid of 25 horizontal and 17 vertical points.

<table>
<thead>
<tr>
<th>Mode</th>
<th>( c )</th>
<th>% error</th>
<th>( \phi(y = 1\text{km}, \sigma = 1) ) ( \times 10^{-2} )</th>
<th>% error</th>
<th>( u(y = 1\text{km}, \sigma = 1) ) ( \times 10^{-3} )</th>
<th>% error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10.07</td>
<td>0.008</td>
<td>2.2139</td>
<td>0.003</td>
<td>2.1995</td>
<td>0.001</td>
</tr>
<tr>
<td>2</td>
<td>5.034</td>
<td>0.028</td>
<td>2.1917</td>
<td>0.017</td>
<td>4.3552</td>
<td>0.006</td>
</tr>
<tr>
<td>3</td>
<td>3.360</td>
<td>0.132</td>
<td>2.1705</td>
<td>0.004</td>
<td>6.4662</td>
<td>0.038</td>
</tr>
</tbody>
</table>

Table 2: Solutions for phase speed \((c)\), pressure \((\phi)\) and alongshelf velocity \((u)\) of baroclinic Kelvin waves for \( N^2 = 6.25 \times 10^{-8}s^{-2}, h = 4000 \text{ m, } f = 10^{-4}s^{-1} \) and \( k = 10^{-7}m^{-1} \) with percent errors compared to theoretical values.
Figure 2: (a) Across-shelf velocity component for the case Mode 2, $S = 1$ in Table 3. (b) Alongshelf velocity component for Mode 2, $S = 1$. (c) Across-shelf velocity component for Mode 2, $S = 10$. Left: After Huthnance, 1978* (nondimensional units). Right: Solution obtained using the present model with $M=81$, $N=9$ and with a channel width four times that of the shelf (units: (a) $10^{-6} m s^{-1}$; (b) $10^{-3} m s^{-1}$; (c) $10^{-6} m s^{-1}$). While the contour intervals differ due to a different normalisation procedure, the features of the solution are clearly the same.

*Reprinted with permission of the American Meteorological Society
\[ S = \frac{N_{\text{max}}^2 h_{\text{max}}^2}{f^2 L_s^2} \]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>.288 (.310) [.312]</td>
<td>.104 (.108) [.111]</td>
<td>.060 (.062) [.047]</td>
</tr>
<tr>
<td>1</td>
<td>.397 (.422) [.421]</td>
<td>.173 (.176) [.176]</td>
<td>.112 (.114) [.115]</td>
</tr>
<tr>
<td>10</td>
<td>1.010 (1.045) [1.013]</td>
<td>.491 (.509) [.484]</td>
<td>.312 (.339) [.295]</td>
</tr>
</tbody>
</table>

Table 3: Comparison of the nondimensional long-wave phase speeds \( c/(fL_s) \) obtained with the present model (using \( M=81 \) grid points and 10 vertical modes, i.e. \( N=9 \)) for a shelf with constant slope and constant \( N^2 \) to those of Huthnance (shown in parentheses) and Brink and Chapman (shown in brackets).
6 Using CTWEIG in Conjunction with Another Program

6.1 Saving the Output from CTWEIG

Every user is likely to have in mind a different application for program CTWEIG. Should you wish to simply compute and output CTW frequencies and modal structures, you're all set. However, if you want to save the modal structures and input them into another program, for example the Haidvogel primitive equation model, you need some way of saving or recomputing the output from CTWEIG.

One way to make the modal structures available to another program is to run CTWEIG, output all the solutions to a file, and have your application program read the file. Quite apart from being rather clumsy, this would involve a lot of input/output and would diminish the precision of the solution unless you increased the number of significant figures in the output format.

6.2 Subroutine CTWAVE

A better way is to use a frequency/wavenumber pair previously determined by running CTWEIG as the input to a subroutine which recomputes the modal structures. The following subroutine, CTWAVE, will do this.

```
subroutine ctwave

parameter ( m=31, n=6,
               1 mm=m-1, nm=n-1, mp=m+1, np=n+1 )
implicit double precision (a-h,o-z)
dimension p(0:n,0:m)
common /abc/ a(0:n,0:n,0:m),b(0:n,0:n,0:m),c(0:n,0:n,0:m)
double precision kx
common /const/ kx,w,f,y,l,yls
common /dchb/ pi,sig(0:n),cp(0:n,0:n),cf(0:n,0:n),cd(0:n,0:n),
               1 cdz(0:n,0:n),cint(0:n,0:n),cdzs(0:n,0:n),
               2 cdzdz(0:n,0:n)
common /deta/ deta,desq
common /geom/ h(0:m),pn(0:m),yn(0:2*mm)
common /sol/ uu(0:n,mm),vv(0:n,m),ww(0:n,mm),
               1 wsiw(0:n,mm),pphi(0:n,mm),rrho(0:n,mm),ppsi(m)
c compute the Chebyshev transform matrices
call chbssetd
```
do 1 k=0,n
do 1 kk=0,n
   cdzs(k,kk) = 0.d0
1  cdzs(k,kk) = 0.d0
do 2 k=0,n
do 2 kk=0,n
do 3 i=0,n
do 3 k=0,n
do 3 kk=0,n
3  cdzs(k,kk) = cdzs(k,kk) + cdzs(k,i)*cdzs(i,kk)
call phieqn(p)
call nbtripd(a,b,c,p,1,m+1,np)
call response(p(0,1),pint)
write(6,10) pint
10 format(' PINT =',1pd10.4)
call uv(p)
return
end

CTWAVE must be linked to the routines BVFSQ, CHBSETD, DOTP, INVMTX, PHIEQN, RESPONSE, UV, and NBTRIPD and its associated routines, listed in section 8. By calling the same routines used by CTWEIG, CTWAVE computes the Chebyshev transformation matrices, sets up and solves the discretized governing equation, and computes all the modal structures and stores them in common block /SOL/. Naturally, the module in your application program which calls CTWAVE should enter in the appropriate common blocks the same metric factors, depth profile, channel dimensions, wavenumber and Coriolis parameter that you used to compute the wave frequency.

For example, to recompute the modal structures of the example case presented in section 4.8, the initialization routine would include the following lines:

parameter ( m=31, n=6,
            mm=m-1, nm=n-1, mp=m+1, np=n+1 )
double precision deta,desq,kx,omega,f,yl,yls
double precision uu,vv,ww,wsig,pphi,rrho,ppsi,hctw,pnctw,ynctw
common /data/ deta,desq
common /const/ kx,omega,f,yl,yls
common /sol/ uu(0:n,mm),vv(0:n,m),ww(0:n,mm),
               wsig(0:n,mm),pphi(0:n,mm),rrho(0:n,mm),ppsi(m)
common /geom/ hctw(0:m),pnctw(0:m),ynctw(0:2*mm)

Set wave and environmental parameters and grid spacing constants

\[ kx = 1.0 \times 10^{-5} \]
\[ \omega = 3.4031639538d-05 \]
\[ f = 1.0 \times 10^{-4} \]
\[ yl = 300.0 \times 10^{03} \]
\[ yls = 100.0 \times 10^{03} \]
\[ det = 1.0d0 \]
\[ desq = det \ast det \]

Set topography, coordinate mapping. Compute pn scale factors

\[
\begin{align*}
do & \quad 90 \quad j=1,mm \\
    & \quad ynctw(2*j-2) = yf(\text{dble}(j-1)) \\
    & \quad \mathbf{90} \quad \text{ynctw}(2*j-1) = y\text{f}(\text{ble}(j) - 0.5d0) \\
    & \quad \text{ynctw}(2*mm) = yf(\text{dble}(mm)) \\
    & \quad \text{do} \quad 100 \quad j=1,mm \\
    & \quad \text{pnctw}(j) = det/(yn(2*j)-yn(2*j-2)) \\
    & \quad \mathbf{100} \quad \text{hctw}(j) = hf(yn(2*j-1)) \\
    & \quad \text{pnctw}(0) = \text{pnctw}(1) \\
    & \quad \text{pnctw}(mm) = \text{pnctw}(mm) \\
    & \quad \text{hctw}(0) = hfctw(yn(0) - 0.5d0*det/\text{pn}(0)) \\
    & \quad \text{hctw}(mm) = hfctw(yn(2*mm) + 0.5d0*det/\text{pn}(mm))
\end{align*}
\]

Set up ctw structures

\text{call ctwave}

6.3 Getting the Phase Right

Before using the modal structures, the solutions as output for \( v \), \( w \) and \( \Omega \) must be multiplied by \( i = \sqrt{-1} \) to obtain the correct phase relative to the other variables. This is because the solution for \( \phi \) (and similarly for \( u \), \( \psi \), and \( \rho \)) is computed such that

\[ \phi(x, \eta_j, \sigma_k, t) = Re[P\phi(k,j)\exp(i(kz - \omega t))] = P\phi(k,j) \cos(kx - \omega t) \]

whereas the solution for \( v \) (and similarly \( w \) and \( \Omega \)) is computed such that

\[ v(x, \eta_j, \sigma_k, t) = Re[iVV(k,j)\exp(i(kz - \omega t))] = -V\phi(k,j) \sin(kx - \omega t) \]

An example application is the set of function subprograms UCTW, VCTW... listed below which compute \( u \), \( v \ldots \) for a CTW in a uniform coastal channel as a function of position and time.

\begin{verbatim}
real function uctw(x,j,k,t)
parameter ( m=31, n=6,
\end{verbatim}
1    mm=m-1, nm=n-1, mp=m+1, np=n+1

double precision kx,omega,f,yl,yls,uu,vv,ww,wsig,pphi,rrho,ppsi
common /const/ kx,omega,f,yl,yls
common /sol/ uu(0:n,mm),vv(0:n,m),ww(0:n,mm),
wsig(0:n,mm),pphi(0:n,mm),rrho(0:n,mm),ppsi(m)
uctw = sngl(uu(k,j)*dcos(kx*x-omega*t))
return

entry vctw(x,j,k,t)
vctw = -sngl(vv(k,j)*dsin(kx*x-omega*t))
return

c
entry wctw(x,j,k,t)
wctw = -sngl(wsig(k,j)*dsin(kx*x-omega*t))
return

c
entry rhoctw(x,j,k,t)
rhoctw = sngl(rrho(k,j)*dcos(kx*x-omega*t))
return

c
entry phictw(x,j,k,t)
phictw = sngl(pphi(k,j)*dcos(kx*x-omega*t))
return

c
entry psictw(x,j,t)
psictw = sngl(ppsi(j)*dcos(kx*x-omega*t))
return
end
7 References


Acknowledgements

I am grateful to Dale Haidvogel for supplying the Chebyshev routines and to Dave Chapman and Ken Brink for many helpful discussions during the development and debugging of the code. Dave deserves special thanks for never tiring of saying "It has to work. Just keep looking and you'll find the bug in there somewhere." This work was supported by the National Science Foundation under grant OCE85-21837. Computer facilities at the National Center for Atmospheric Research in Boulder, Colorado, were used during part of the program development. NCAR is funded by the National Science Foundation.
Listing of Program CTWEIG

0001 c This is a program for solving the coastal-trapped wave eigenvalue problem
0002 c for arbitrary stratification and bottom topography by resonance iteration.
0003 c A rigid-lid surface boundary condition is employed.
0004 c A (slippery) wall is placed at the offshore boundary.
0005 c The governing equation in terms of pressure is solved using a finite
0006 c difference approximation in the horizontal and an expansion in Chebyshev
0007 c polynomials in the vertical.
0008 c
0009 c The user must supply:
0010 c (i) function subprogram BVFSQ to generate values of the Brunt-Vaisala
0011 c frequency squared as a function of depth (see the example function
0012 c for a full explanation)
0013 c (ii) the depth profile and an array of values of the cartesian
0014 c coordinate \( y \) mapping the physical locations of the across-shelf
0015 c grid points to their location in terms of the generalized across-
0016 c shelf coordinate \( \eta \). All these values are stored in the common
0017 c block /GEOM/ and their determination is described in the Technical
0018 c Report. Routines are provided which will compute
0019 c all these variables if the depth can expressed by a function
0020 c subprogram HF(Y) and if the dependence of the cartesian coordinate
0021 c \( y \) on the generalized coordinate \( \eta \) can be expressed by a function
0022 c subprogram YF(ETA). No modifications to the latter function
0023 c need be made if the user does not wish to use the grid stretching
0024 c capability of the program to refine grid resolution near particular
0025 c across-shelf locations.
0026 c
0027 c SI units are used throughout.
0028 c
0029 c The program presents the option of specifying wavenumber and searching
0030 c for a corresponding free wave frequency or vice versa. Array WKRNG
0031 c contains the three current estimates of the parameter being searched
0032 c (frequency or wavenumber).
0033 c Array RII contains the corresponding values of the degree of resonance RI.
0034 c RI = integral (p=2) dx dz. The program seeks to find WKRNG values
0035 c which bracket a maximum of RI, i.e. WKRNG(1) < WKRNG(2) < WKRNG(3) ;
0036 c RII(1) < RII(2) and RII(3) < RII(2). Then the bracket is narrowed using
0037 c a quadratic search until two consecutive guesses are within the specified
0038 c tolerance.
0039 c
0040 c \( M \) is the number of finite difference grid points in the horizontal
0041 c \( N+1 \) is the number of Chebyshev modes
0042 c To alter the program resolution, the following PARAMETER statement and those
0043 c corresponding to it in the other program segments must be altered.
0044 c
0045 c WARNING: Two non-ANSI-standard FORTRAN features have been used in the
0046 c program. They are (1) the use of names exceeding six characters
0047 c for some subroutines and common blocks and (2) the use of
0048 c exclamation points for end of line comments. Since many compilers
0049 c do support these features I have left them in for clarity in
0050 c understanding how the code works. If your compiler will not
0051 c accept them the changes required are straightforward.
0052 c
0053 c Direct any questions/problems (and reports of errors!) to:
0054 c John Wilkin
0055 c W.H.O.I., Woods Hole, NASS 02543
0056 c (617) 548-1400 ext 2731

24
program ctewig
parameter ( m=31, n=6, 
m-1, mm=m-1, mm=n-1, mp=m+1, np=n+1 )
parameter ( nwpvts=500 )
imPLICIT double precision (a-h,o-z)
dimension p(0:n,0:m)
dimension wkrng(3), rii(3)
dimension lbl(20)
character*1 reply
character*4 ivar(0:10)
logical wrch, plot
common /abc/ a(0:n,0:n,0:m), b(0:n,0:n,0:m), c(0:n,0:n,0:m)
double precision kx
common /const/ kx, w, yl, yls
common /dchb/ pi, sig(0:n), cp(0:n,0:n), cf(0:n,0:n), cd(0:n,0:n), 
                1 cdz(0:n,0:n), cint(0:n,0:n), cdzs(0:n,0:n), 
                2 ccdzsdz(0:n,0:n)
common /data/ deta, desq
common /geom/ h(0:m), pn(0:m), yn(0:2+mm)
common /idev/ iterm, jterm, idak
logical term, pout, vout, wout, wsgout, rhoout,
         1 phiout, psaiout
common /logical/ term, pout, vout, wout, wsgout, rhoout,
         1 phiout, psaiout
common /sol/ uu(0:n,mm), vv(0:n,m), ww(0:n,mm), 
            1 wsig(0:n,mm), pphi(0:n,mm), rrho(0:n,mm), ppsi(m)
real wavenum, freq
common /disp/ nkpts, wavenum(0:nwpvts), freq(0:nwpvts,2)
0088 c
0089 c Compute the Chebyshev transform matrices
0090 call chbsedt
0091 do k=0,n
0092    do k=0,n
0093        cdzs(k,kk) = 0.d0
0094        ccdzsdz(k,kk) = 0.d0
0095    do 2 k=0,n
0096        ccdzsdz(k,kk) = (1.d0-sig(k)) * cdz(k,kk)
0097    do 3 i=0,n
0098        ccdzsdz(k,kk) = ccdzsdz(k,kk) + cdzs(k,i)*cdzs(i,kk)
0099    do 3 kk=0,n
0100        c
0101          3 c
0102 c Input
0103     iterm = 5
0104     jterm = 6
0105     idak = 7
0106     read(iterm, '(ll)') term
0107     if(term) write(jterm, 10)
0108      5 format(' Enter Coriolis freq (rad/s), channel width YL', 
0110            , ' and shelf width YLS (metres)' )
0111     1 read(iterm, *, err=5) f, yl, yls
0112     15 if(term) write(jterm, 20)
0113      20 format(' Enter: W to search frequency for given wavenumber', ).
' K to search wavenumber for given frequency'
read(item,'(a)',err=15)reply
if((reply.eq.'W').or.(reply.eq.'W'))then
  warch = .true.
else if((reply.eq.'k').or.(reply.eq.'K'))then
  warch = .false.
else
  go to 15
endif
30 if(warch)then
35 if(term)write(jterm,40)
40 format(' Enter wavenumber ( units /m )')
read(item,*,err=35)kx
if(term)write(jterm,50)
50 format(' Enter initial estimate of omega and search'
1 ' increment ( units /s )')
else
55 if(term)write(jterm,60)
60 format(' Enter frequency ( units /s )')
read(item,*,err=55)w
if(term)write(jterm,70)
70 format(' Enter initial estimate of wavenumber and search'
1 ' increment ( units /m )')
endif
read(item,*,err=30)wkrng(2),dw
wkrng(3) = wkrng(2) + dw
75 if(term)write(jterm,80)
80 format(' Enter fractional tolerance ( dw/w or dk/k ) to which to'
1 ' determine frequency or wavenumber')
85 if(term)write(jterm,90)
90 format(' Enter maximum number of iterations')
read(item,*,err=85)itmax
95 if(term)write(jterm,100)
100 format(' Enter the variables which are to be output upon'',
1 ' convergence.',',,
2 ' Options are: P, U, V, W, WSIG (w in sigma coord),',',
3 ' PSI, RHO, PHI (baroclinic pressure)',',,
4 ' or simply ALL to output everything.',',,
5 ' PLT creates plots of all variables.',',,
6 ' Enter one per line, ending with a blank line.')
nout = 0
1001 continue
if(ivar(nout).ne.' ')then
  c do while(ivar(nout).ne.' ')
  nout = nout + 1
  read(item,'(a4)',end=110)ivar(nout)
  c end do
  go to 1001
endif
110 continue
120 i=1,nout-1
125 if((ivar(i).eq.'ALL').or.(ivar(i).eq.'all'))then
  pout=.true.
  uout=.true.
  vout=.true.
wout=true.
wsout=true.
phiout=true.
psiout=true.
rhout=true.

dendif

if((ivar(i).eq."PLT'').or.(ivar(i).eq."plt''))plot=true.
if((ivar(i).eq."P'').or.(ivar(i).eq."p'"))pout=true.
if((ivar(i).eq."U'').or.(ivar(i).eq."u'"))uout=true.
if((ivar(i).eq."V'').or.(ivar(i).eq."v'"))vout=true.
if((ivar(i).eq."W'').or.(ivar(i).eq."w'"))wout=true.
if((ivar(i).eq."WSIG'').or.(ivar(i).eq."wsig'"))wsout=true.
if((ivar(i).eq."PHI'').or.(ivar(i).eq."phi'"))phiout=true.
if((ivar(i).eq."PSI'').or.(ivar(i).eq."psi'"))psioout=true.

120
if((ivar(i).eq."RHO'').or.(ivar(i).eq."rho'"))rhout=true.

deta is unity, grid dimensions are incorporated in pn scale factors
deta = 1.d0
desq = deta*deta

c
Set topography and coordinate mapping and compute pn scale factors

c from yn(eta)
do 140 j=1,mm

yn(2*j-2) = yf(dfble(j-1))
yn(2*j-1) = yf(dfble(j)*0.5d0)
yn(2*mm) = yf(dfble(mm))
do 150 j=1,mm

pn(j) = deta/(yn(2*j)-yn(2*j-2))
h(j) = hf(yn(2*j-1))
pn(0) = pn(1)
pn(m) = pn(mm)
h(0) = hf(yf(yn(0)*0.5d0*deta/pn(0)))
h(m) = hf(yf(yn(2*mm)*0.5d0*deta/pn(m))
call out

c
Get wavenumber range info for computing dispersion curve
(this option only available if frequency search was selected)
if(warch)then

185 if(term)write(jterm,190)
190 format('/','Enter the number of frequency/wavenumber pairs you',
191 1 ' wish to compute',/',' (If >1 a dispersion curve will',
192 2 ' be calculated)')
2013 read(itype,*,err=185)nkpts
2014 if(nkpts.gt.1)then
2015 195 if(term)write(jterm,200)
2016 200 format('/','Enter maximum value of the wavenumber axis')
2017 read(itype,*,err=200)rkmax
2018 rkinc = (rkmax-kx)/float(nkpts-1)
2019 wavenum(0) = 0.
2020 freq(0,1) = 0.
2021 wavenum(1) = sngl(kx)
2022 do 210 i=2,nkpts
2023 210 wavenum(i) = wavenum(i-1) + sngl(rkinc)
2024 210 wavenum(i) = wavenum(i-1) + sngl(rkinc)
2024 210 wavenum(i) = wavenum(i-1) + sngl(rkinc)
2025 endif
2026 else
2027 nkpts = 1
2028 endif

27
Loop on wavenumber values if full dispersion curve is being computed

do 300 ik=1,nkpts

Solve for first estimate of wkrng, wkrng(2)
call out2
ncount = 1
if(warch) then
  w = wkrng(2)
else
  kx = wkrng(2)
endif
call phieqn(p)
call nbtripd(a,b,c,p,1,m+1,np)
call response(p0,p1,pint)
rii(2) = pint
call out3(ncount,pint)

Solve for second guess, wkrng(3)
call phieqn(p)
call nbtripd(a,b,c,p,1,mp,np)
call response(p0,p1,pint)
rii(3) = pint
call out3(ncount,pint)

if(rii(3).gt.rii(2)) then ! search toward increasing wkrng
  nstep = 1
  continue
else
  kx = wkrng(3)
endif
call phieqn(p)
call nbtripd(a,b,c,p,1,mp,np)
call response(p0,1,pint)
rii(3) = pint
call out3(ncount,pint)

if(rii(3).gt.rii(2)).and.(ncount.le.itmax) then
  do while((rii(3).gt.rii(2)).and.(ncount.le.itmax))
enddo
endif

Repeat until RI decreases, at which point a local maximum in
RI will have been bracketed
ncount = ncount + 1
wkrng(1) = wkrng(2)
wkrng(2) = wkrng(3)
wkrng(3) = wkrng(2) + dble(nstep) * dw
if(warch.and.(wkrng(3).ge.f)) call crash4
rii(1) = ri(2)
rii(2) = ri(3)
if(warch) then
  w = wkrng(3)
else
  kx = wkrng(3)
endif
call phieqn(p)
call nbtripd(a,b,c,p,1,mp,np)
call response(p0,1,pint)
rii(3) = pint
call out3(ncount,pint)
nstep = nstep + 1
end do

else ! search toward decreasing wkrng
  ncount = ncount + 1
  wkrng(1) = wkrng(2) - dw
  if(wsrch) then
    w = wkrng(1)
  else
    kx = wkrng(1)
  endif

  call phieqn(p)
  call nbtripd(a,b,c,p,i,mp,np)
  call response(p(0,1),pint)
  rii(1) = pint
  call out3(ncount,pint)
  nstep = 2
  continue
  if((rii(1).gt.rii(2)).and.(ncount.lt.itmax)) then
    do while ((rii(1).gt.rii(2)).and.(ncount.lt.itmax))
      c Repeat until RI decreases, at which point a local maximum in
      c RI will have been bracketed
      ncount = ncount + 1
      wkrng(3) = wkrng(2)
      wkrng(2) = wkrng(1)
      wkrng(1) = wkrng(2) - dble(nstep) * dw
      if(wkrng(1).le.0.) call crash(5)
      rii(3) = rii(2)
      rii(2) = rii(1)
      if(wsrch) then
        w = wkrng(1)
      else
        kx = wkrng(1)
      endif
      call phieqn(p)
      call nbtripd(a,b,c,p,i,mp,np)
      call response(p(0,1),pint)
      rii(1) = pint
      call out3(ncount,pint)
      nstep = nstep + 1
      end do
    go to 1003
  endif
endif

if(ncount.ge.itmax) call crash(3)

The solution is now bracketed with wkrng(2) being the best estimate so far.  Perform a quadratic search until the solution is bracketed to the desired tolerance.

continue

if((dmin1((wkrng(3)-wkrng(2))/wkrng(2),(wkrng(2)-wkrng(1)))/
  wkrng(2)).gt.wtol ) .and. (ncount.lt.itmax ) then
  do while((dmin1((wkrng(3)-wkrng(2))/wkrng(2),(wkrng(2)-wkrng(1)))/
    wkrng(2)).gt.wtol ) .and. (ncount.lt.itmax )
    ncount = ncount + 1
    call qsearch(wkrng,rii,wknew)
if (wsrch) then
    w = wknew
else
    kx = wknew
endif

call phieqn(p)
call nbtripd(a,b,c,p,1,mp,np)
call response(p(0,1),pint)
ri = pint
call out3(ncount,pint)
if (ri.lt.rii(2)) then ! Did not find new best solution
    if (wckill.lt.wkkrng(2)) then ! replace lower bracket value
        wkkrng(1) = wknew
        rii(1) = ri
    else ! replace upper bracket value
        wkkrng(3) = wknew
        rii(3) = ri
    endif
else ! Did find new best solution
    if (wckill.lt.wkkrng(2)) then ! replace upper bracket
        wkkrng(3) = wkkrng(2)
        rii(3) = rii(2)
        wkkrng(2) = wknew
        rii(2) = ri
    else ! replace lower bracket
        wkkrng(1) = wkkrng(2)
        rii(1) = rii(2)
        wkkrng(2) = wknew
        rii(2) = ri
    endif
endif
c end do
go to 1004
endif
if (ncount.eq.itsmax) call crash(6)
c Sucessful convergence
if (wsrch) then
    w = wkkrng(2)
else
    kx = wkkrng(2)
endif
call phieqn(p)
call nbtripd(a,b,c,p,1,mp,np)
call response(p(0,1),pint)
c if (ik.eq.1) then
    call out1(ncount,pint)
call uv(p)
call out4(p)
if (pout) then
    call out2
    call out3(ncount,pint)
endif
if (plot) call ctwplot(p)
c endif
c
Record dispersion point; set new wavenumber and initial frequency guess

if(wrch) then
  freq(ik,1) = snsl(w)
wfrng(2) = w + (w-freq(ik-1,1)) / (kx-wavenum(ik-1)) * 
wfrng(3) = wfrng(2) + dw
  kx = dble(wavenum(ik+1))
endif

End loop on dispersion curve wavenumber values
continue

Output dispersion curve
if(nkpts.gt.1) then
  if(term)write(jterm,400)
  write(idsk,400)
  format(/,' Dispersion relation: f(wavenumber (/m) frequency (rad/s))/')
  do 410 i=1,nkpts
    if(term)write(jterm,420)wavenum(i),freq(i,1)
    write(idsk,420)wavenum(i),freq(i,1)
    format(1pe14.5,10x,1pe14.5)
    call displot
  end
endif
call exit
end

double precision function bvfsq(z)
implicit double precision (a-h,o-z)

bvfsq = 1.d-10 ! barotropic
bvfsq = 1.25e-06 ! Huthnance case S=0.2
bvfsq = 6.25e-06 ! Huthnance case S=1
bvfsq = 6.25e-05 ! Huthnance case S=10
bvfsq = 5.e-05+dexp(1.25e-03*z) ! exponential stratification
return
end

double precision function hf(y)
implicit double precision (a-h,o-z)

hf = 75.d0+dexp(2.d0*2.d-05*y) ! exponential shelf
hf = 4000.d0 ! Kelvin waves
hf = dmax1(10.d0,dmin1(10.d0+0.0399d0*y,4000.d0)) ! linear slope (Huthnance)
return
end

double precision function yf(eta)
parameter ( m=31, n=6,
1 mm=m-1, nm=n-1, mp=m+1, np=n+1)
implicit double precision (a-h,p-z)
double precision kx
common /const/ kx,w,f,y1,yls
yf = eta/dble(mm)*y1
yf = 0.5d0*y1*(1.d0+dtnu(datan(1.d0)*(2.d0+eta/dble(mm)-1.d0)))
yf = y1*(dexp(eta/dble(mm))-1.d0)/(dexp(1.d0)-1.d0)
return
end
subroutine chbsetd
  parameter ( m=31, n=6, 
    i mm=m-1, nm=n-1, mp=m+1, np=n+1) 
  c Modification to Dale Haidvogel's version of chbset:
  c keep cp,cf etc double precision
  c double precision pi,sig,cp,cf,cd,cdz,cint
  common /dchb/ pi,sig(0:n),cp(0:n,0:n),cf(0:n,0:n),cd(0:n,0:n),
    cdz(0:n,0:n),cint(0:n,0:n),cdzs(0:n,0:n),
    cdzsdz(0:n,0:n)
  double precision dsig(0:n),dcp(0:n,0:n),dcf(0:n,0:n),
    dcd(0:n,0:n),dcdz(0:n,0:n),dcdt(0:n,0:n),
    dac,ds,dt
  Need value of pi
  dpi = 4.d0*datan(1.d0)
  Location of cheb points
  do 200 k=0,n
    200 dsig(k) = dcos( -dpi*dble(n-k)/dble(n) )
  Integrals of t(z) over -1:z+1
  dcsum(0) = 2.d0
  do 203 k = 1,n,2
    203 dcsum(k) = 0.d0
  do 204 k = 2,n,2
    204 dcsum(k) = -2.d0 / ( dble(k+k)-1.d0 )
  Values of p(z)
  np1=n+1
  do 208 i = 0,n
    dac = -dacos( dsig(i) )
    dcp(i,0) = 1.d0
    do 206 j = 1,n
      206 dcp(i,j) = dcos( dble(j)*dac ) - 0.5d0*dcsum(j)
  continue
  call invmtx(np1,np1,dcp,dcf,dt)
  Integrals of p(z)
  do 215 i = 0,n
    dcwrk(i,0) = 1.d0 - dsig(i)
    dcwrk(i,1) = 0.5d0 - 0.5d0*dsig(i)*dsig(i)
    dac = -dacos( dsig(i) )
    do 211 j=2,n
      211 dcwrk(i,j) = -1.d0/(dble(j+j)-1.d0)
    1 ( dcwrk((j+1)*dac)/(2.d0*(j+1)) )
    2 - dcwrk((j-1)*dac)/(2.d0*(j-1))
    3 -0.5d0*dcsum(j)*(1.d0-dsig(i))
  continue
  Integral transform matrix
  do 220 j=0,n
    do 220 i=0,n
      220 dcnt(i,j) = 0.d0
    do 225 k=0,n

do 222 i=0,n
do 222 j=0,n
222 dcint(i,j) = dcint(i,j) + dcwsk(i,k)*dcf(k,j)
225 continue

c
Derivatives of t(x)

ds = 1.d0
do 230 j=0,n
ds = -ds
dcd(0,j) = ds*dble(j*j)
dcd(n,j) = dble(j*j)
230 continue

do 235 i=1,nm
dcd(i,0) = 0.d0
dcd(i,1) = 1.d0
dac = -dacos( dsig(i) )
do 232 j=2,n
dcd(i,j) = dble(j) * dsin(j*dac) / dsin(dac)
235 continue

c
Derivative transform matrix

do 240 j=0,n
do 240 i=0,n
240 dcdz(i,j) = 0.d0
do 241 k=0,n
do 242 i=0,n
do 242 j=0,n
242 dcdz(i,j) = dcdz(i,j) + dcd(i,k)*dcf(k,j)
241 continue

c
Modification to Haidvogel's version of chbset:
keep cp,cf etc double precision
pi = dpi
do 300 k=0,n
300 dsig(k) = dsig(k)
do 301 i=0,n
do 301 j=0,n
cp(i,j) = dcp(i,j)
cf(i,j) = dcf(i,j)
cd(i,j) = dcd(i,j)
cdz(i,j) = dcdz(i,j)
301 dcint(i,j) = dcint(i,j)
return
end

c
subroutine crash(ier)
common / idev / iterm,jterm,idsk
logical term,pout,uout,vout,wout,wsout,rhouout,
1 phiout,psiout
common /logical/ term,pout,uout,vout,wout,wsout,rhouout,
1 phiout,psiout
common /disp/ nkpts
if (ier.eq.3)then
write(idsk,13)
if(term)write(jterm,13)
13 format(' ERROR: maximum number of iterations exceeded',/,
1 ' without bracketing maximum')
else if(ier.eq.4) then
  write(idsk,14)
  if(term)write(jterm,14)
  format(' ERROR seeking bracket : omega > f')
else if(ier.eq.5) then
  write(idsk,15)
  if(term)write(jterm,15)
  format(' ERROR seeking bracket : omega or k < 0')
else if(ier.eq.6) then
  write(idsk,16)
  if(term)write(jterm,16)
  format(' ERROR: maximum number of iterations exceeded',/, 1
  while reducing bracket')
endif
if(nkpts.gt.1)call displot
call exit
return
end
c double precision function dotp(n,v1,l1,v2,l2)
c
Computes the inner product of two vectors
double precision v1(1),v2(1)
i1 = 1
i2 = 1
dotp = 0.d0
do 10 i=1,n
dotp = dotp + v1(i1) * v2(i2)
i1 = i1 + l1
i2 = i2 + l2
10 return
end
c subroutine invmtx(n,ndim,a,v,d)
implicit double precision (a-h,o-z)
dimension a(ndim,1),v(ndim,1)
dimension ip(140),ir(140)
d1=1.d0
do 270 j=1,n
do 211 i=1,n
d=1.d1
vmax=0.d0
if(ip(j)) = 201,203,201
203 do 205 i=1,n
if(ip(j)) = 205,206,205
206 tst=v(i,j)
vh=dabs(tst)
202 vmax=vh
202 k=1
202 l=j
202 205 continue
201 continue
ip(1)=k
ir(m)=1
204 continue
pvt=v(k,1)
v(k,1)=d1
do 215 j=1,n
hold=v(k,j)
v(k,j)=v(1,j)
215 v(1,j)=hold/pvt
do 221 i=1,n
if(i.eq.1) go to 221
hold=v(i,1)
v(i,1)=0.d0
212 v(i,j)=v(i,j)-v(1,j)*hold
221 continue
211 continue
do 209 j=1,n
m=n-j+1
l=ir(m)
k=ip(l)
if(k.eq.1) go to 209
d=d
208 continue
209 continue
return
end

Subroutine out

Subroutines to output results.

0 : Output input parameters
1 : Signal convergence
2 : Header for iteration results
3 : Output result of particular guess
4 : Output requested variable solutions

WARNING: Some FORTRAN compilers will not accept the syntax of some
of the write statements in this subroutine despite their
conforming to ANSI standard. The intent is clear so I'm
sure that, should you strike any problems, you can figure
out the necessary changes.

parameter ( m=31, n=6,
1 mm=m-1, mmn=m-1, mp=m+1, np=n+1)
implicit double precision (a-h,o-z)
double precision    kx
common /const/ kx,w,f,y1,yls
common /dchbb/ pi,sig(0:n),cp(0:n,0:n),cf(0:n,0:n),cd(0:n,0:n),
cdz(0:n,0:n),cint(0:n,0:n),cdzs(0:n,0:n),
cdzsdz(0:n,0:n)
common /deta/ deta,desq
common /geom/ h(0:m),pn(0:m),yn(0:2*mm)
common / idev / iterm,jterm,idak
0028 logical term,pout,uout,vout,wout,wsgout,rhout,
0029 1 phiout,psiout
0030 common / logical/ term,pout,uout,vout,wout,wsgout,rhout,
0031 1 phiout,psiout
0032 common / sol/ uu(0:n,m),vv(0:n,m),ww(0:n,mm),
0033 1 wsig(0:n,mm),pphi(0:n,mm),rrho(0:n,mm),ppsi(m)
0034 dimension p0(0:n,0:m)
0035 z(k,j) = 0.5d0 * h(j) * (sig(k)-1.d0)
0036 mtmp = m
0037 ntmp = n
0038 if(term)write(jterm,10)mtmp,ntmp,ntmp+1,ntmp
0039 write(idisk,10)mtmp,ntmp,ntmp+1,ntmp
0040 10 format(' Solution to the coastal-trapped wave eigenvalue',
0041 1 ' problem in a coastal channel',',', 'Numerical resolution',',
0042 2 ' is N =',i3,,' horizontal finite difference grid points',',
0043 3 ' and',i3,,' vertical Chebyshev modes, i.e. N =','i3,/
0044 if(term)write(jterm,20)f,y1,1.d+03
0045 write(idisk,20)f,y1,1.d+03
0046 20 format(' Coriolis frequency =',1pd15.5, ', rad/s',',
0047 1 ' Channel width =',1pd15.5, ', km',')
0048 if(term)write(jterm,30)
0049 write(idisk,30)
0050 30 format(' Brunt-Vaisala frequency profile ( (rad/s)**2 ) :',')
0051 if(term)write(jterm,40)0.5d0*h(mm)*(sig(k)-1.d0),k=0,n
0052 write(idisk,40)0.5d0*h(mm)*(sig(k)-1.d0),k=0,n
0053 40 format(' Depth (m) ',10(1pd12.4),')')
0054 if(term)write(jterm,50)(bvfsq(z(k,mm))),k=0,n
0055 write(idisk,50)(bvfsq(z(k,mm))),k=0,n
0056 50 format(' N**2 ',10(1pd12.4),')')
0057 c
0058 c Determine Nmax**2 and compute stratification parameter S
0059 bvmax = 0.0d0
0060 do 60 k=0,n
0061 60 bvmax = dmax1( bvmax, bvfsq(z(k,mm)) )
0062 s = bvmax*(h(mm)/y1)**2
0063 if(term)write(jterm,70)s
0064 write(idisk,70)s
0065 70 format(4d15.4)
0066 return
0067 c
0068 entry out1(ncount,pint)
0069 if(term)write(jterm,80)
0070 write(idisk,80)
0071 80 format(4d15.4)
0072 if(term)write(jterm,200)
0073 write(idisk,200)
0074 if(term)write(jterm,300)ncount,w,kx,w/kx,pint
0075 write(idisk,300)ncount,w,kx,w/kx,pint
0076 write(idisk,90)
0077 90 format(4d15.4)
0078 1 z = 0 eta = inf
0079 2 1 = integri p(eta=0)**2 dz + integri p(z=-h)**2 dh/deleta
0080 3 z = -h eta = 0
0081 4 Variables to be output are:
0082 c90 format(' The solution is normalized such that: ',)
0083 c 1 y=ymax z=0
0084 c
2 ' energy flux = intgrl (pu) dz dy = 1 m**5 s**-2 ',/
3 ' y=ymin z=-h ',/,
4 ' Variables to be output are:' ,/)
0087 if(pout) write(idsk,110)
0088 if(phiout) write(idsk,120)
0089 if(uout) write(idsk,130)
0090 if(vout) write(idsk,140)
0091 if(wout) write(idsk,150)
0092 if(wsigout) write(idsk,160)
0093 if(rhoout) write(idsk,170)
0094 if(psiout) write(idsk,180)
0095 110 format(' p pressure (barotropic component included) ')
0096 120 format(' phi baroclinic pressure ')
0097 130 format(' u alongshelf velocity ')
0098 140 format(' v across-shelf velocity ')
0099 150 format(' w vertical velocity ')
0100 160 format(' wsig vertical velocity (sigma coordinates) ')
0101 170 format(' rho density perturbation ')
0102 180 format(' psi streamfunction ')
0103 return
0104 c
e ntry out2
0105 if(term) write(jterm,200)
0107 write(idsk,200)
0108 200 format(,' Iteration omega',15X,'k',14X,'c',8X,'int p**2')
0109 return
0110 c
e ntry out3(ncount,pint)
0112 if(term) write(jterm,300) ncount,w,kx,w/kx,pint
0113 write(idsk,300) ncount,w,kx,w/kx,pint
0114 300 format(i6,4x,2(1dp18.10),2(1dp13.5))
0115 return
0116 c
e ntry out4(p)
0118 c
0119 if(pout) then
0120 if(term) write(jterm,310)
0121 write(idsk,310)
0122 310 format(,'5x','y(km)',6x,'depth',7x,'pressure solution p ')
0123 if(term) write(jterm,410)
0124 write(idsk,410)
0125 do 320 j=1,mm
0126 if(term) write(jterm,430) (yn(2*j-1)/1000.),h(j),
0127 1 (p(k,j),k=0,n)
0128 write(idsk,430) (yn(2*j-1)/1000.),h(j),(p(k,j),k=0,n)
0129 if(term) write(jterm,330)
0130 330 format(,' Solutions for other variables written to disk.' ,/)
0131 endif
0132 if(phiout) then
0133 write(idsk,340)
0134 340 format(,'5x','y(km)',6x,'depth',7x,'baroclinic ' ,
0135 1 'pressure perturbation ')
0136 write(idsk,410)
0137 do 350 j=1,mm
0138 write(idsk,430) (yn(2*j-1)/1000.),h(j),(pphi(k,j),k=0,n)
0139 endif
0140 if(uout) then
write(idsk, 400)
format(15x, 'y(km)', 6x, 'depth', 7x, 'alongshelf velocity, u')
write(idsk, 410)
format(28x, 'sigma_0 (bottom).........to sigma_N (surface)', /)
do 420 j = 1, mm
write(idsk, 430) (yn(2*j-1)/1000., h(j), (uu(k,j), k = 0, n))
format(10.2, 2x, f10.2, 2x, 9(1pd12.4), /, 24x, 9(1pd12.4))
endif
if(vout) then
write(idsk, 440)
format(15x, 'y(km)', 6x, 'depth', 7x, 'across-shelf velocity, v')
write(idsk, 410)
do 450 j = 1, mm
htmp = 0.5d0*(h(j)+h(j-1))
write(idsk, 430) (yn(2*j-2)/1000., htmp, (vv(k,j), k = 0, n))
endif
if(wout) then
write(idsk, 460)
format(15x, 'y(km)', 6x, 'depth', 7x, 'vertical velocity', 1
'(z coordinates)')
write(idsk, 410)
do 470 j = 1, mm
write(idsk, 430) (yn(2*j-1)/1000., h(j), (ww(k,j), k = 0, n))
endif
if(wsgout) then
write(idsk, 480)
format(15x, 'y(km)', 6x, 'depth', 7x, 'vertical velocity', 1
'(sigma coordinates)')
write(idsk, 410)
do 490 j = 1, mm
write(idsk, 430) (yn(2*j-1)/1000., h(j), (wsg(k,j), k = 0, n))
endif
if(rhoout) then
write(idsk, 500)
format(15x, 'y(km)', 6x, 'depth', 7x, 'density perturbation, rho')
write(idsk, 410)
do 510 j = 1, mm
write(idsk, 430) (yn(2*j-1)/1000., h(j), (rrho(k,j), k = 0, n))
endif
if(psiout) then
write(idsk, 520)
format(15x, 'y(km)', 6x, 'depth', 7x, 'streamfunction, psi')
do 530 j = 1, mm
htmp = 0.5d0*(h(j)+h(j-1))
write(idsk, 430) (yn(2*j-2)/1000., htmp, ppsi(j))
endif
return
end

SUBROUTINE PHIEQN(p)
C Computes the block tridiagonal system of equations resulting from
C a finite-difference/chebyshev polynomial approximation to the
C coastal-trapped wave eigenvalue problem
C parameter ( m=31, n=6,
C 1, mm=m-1, nn=n-1, mp=m+1, np=n+1)
C implicit double precision (a-h,p-z)
dimension p(0:n,0:m),wrk(0:n,0:n)
common /abc/ a(0:n,0:n,0:m),b(0:n,0:n,0:m),c(0:n,0:n,0:m)
double precision kx
common /const/ kx,w,f,yl,yls
common /dchb/ pi,sig(0:n),cp(0:n,0:n),cf(0:n,0:n),cd(0:n,0:n),
 1 cdz(0:n,0:n),cint(0:n,0:n),cdzs(0:n,0:n),
 2 cdzsdz(0:n,0:n)
common /deta/ deta,desq
common /geom/ h(0:m),pn(0:m),yn(0:2*mm)

z(k,j) = 0.5d0 * h(j) * (sig(k)-1.d0)

  c
  do 10 j=1,mm
  c
  Enter term one of the governing equation
  tmpa1 = 0.5d0*pn(j)/desq*(pn(j)+pn(j-1))
  tmpa2 = -tmpa1*(h(j)-h(j-1))/(h(j)+h(j-1))
do 20 k=0,n
      do 30 kk=0,n
  30 a(k,kk,j) = tmpa2 * cdzs(k,kk)
do 20 k=0,n
      do 30 kk=0,n
  30 tmpb1 = 0.5d0*pn(j)/desq*
  1 ( (pn(j+1)+pn(j))/h(j+1)-h(j))/h(j+1)+h(j) )
  2 (pn(j)+pn(j-1))/h(j)+h(j-1) )
tmpb2 = 0.5d0*pn(j)/desq*(pn(j+1)+2.d0*pn(j)+pn(j-1))
do 40 k=0,n
do 50 kk=0,n
  50 b(k,kk,j) = tmpb1 * cdzs(k,kk)
do 40 k=0,n
      do 50 kk=0,n
  50 tmpc1 = 0.5d0*pn(j)/desq*(pn(j)+pn(j))
tmpc2 = tmpc1*(h(j+1)-h(j))/h(j+1)+h(j))
do 60 k=0,n
do 70 kk=0,n
c(k,kk,j) = tmpc2 * cdzs(k,kk)
do 60 k=0,n
c(k,k,j) = c(k,k,j) + tmpc1
  c
  Add in term two of the governing equation
  tmpa = 0.25d0*pn(j)*2/desq*(h(j+1)-h(j-1))/h(j)
do 80 k=0,n
      do 80 kk=0,n
  80 a(k,kk,j) = a(k,kk,j) - tmpa * cdzs(k,kk)
tmpb = 0.25d0*pn(j)*2/desq*(h(j+1)-h(j-1))/h(j))**2
do 90 k=0,n
      do 90 kk=0,n
  90 b(k,kk,j) = b(k,kk,j) + tmpb * cdzs(k,kk)
tmpc = tmpa
do 100 k=0,n
      do 100 kk=0,n
  100 c(k,kk,j) = c(k,kk,j) + tmpc * cdzs(k,kk)
  c
  Add in term three of the governing equation
  tmpb1 = (f**2-w**2)*4.d0/h(j)**2
do 110 k=0,n
      tmpb2 = tmpb1/bvfgs(z(k,j))
do 110 kk=0,n
  110 wrk(k,kk) = tmpb2 * cdz(k,kk)
do 120 i=0,n
do 120 k=0,n

39
do 120 kk=0,n
120   b(k,kk,j) = b(k,kk,j) + cdz(k,i)*wrk(i,kk)
   c
   c  Add in term four of the governing equation
101   tmpb = kx**2
102   do 130 k=0,n
103   130   b(k,k,j) = b(k,k,j) - tmpb
   c
104   10    continue
   c
   c  Wall boundary conditions
105   eta = 0
106   tmpb1 = -0.5d0/deta*(pn(1)+pn(0)) + 0.5d0*f*kx/w
107   tmpb2 = 0.5d0/deta*(pn(1)+pn(0))*(h(1)-h(0))/(h(1)+h(0))
108   do 140 k=0,n
109   140   do 150 kk=0,n
110   150       b(k,kk,0) = tmpb2 * cdzs(k,kk)
111   140       b(k,k,0) = b(k,k,0) + tmpb1
112   140   tmpc1 = 0.5d0/deta*(pn(1)+pn(0)) + 0.5d0*f*kx/w
113   140   tmpc2 = tmpb2
114   140   do 160 k=0,n
115   160       c(k,kk,0) = tmpc2 * cdzs(k,kk)
116   160       c(k,k,0) = c(k,k,0) + tmpc1
   c
117   c****  eta = etamax
118   c  It is this section of code, which computes the matrices a(*,*),
119   c  and b(*,*), that must be altered to implement a different
120   c  offshore boundary condition. Details on how to do this are included
121   c  in the technical report.
122   tmpa1 = -0.5d0/deta*(pn(m)+pn(mm)) + 0.5d0*f*kx/w
123   tmpa2 = 0.5d0/deta*(pn(m)+pn(mm))*(h(m)-h(mm))/(h(m)+h(mm))
124   do 180 k=0,n
125   180       do 190 kk=0,n
126   190           a(k,kk,m) = tmpa2 * cdzs(k,kk)
127   190           a(k,k,m) = a(k,k,m) + tmpa1
128   190       tmpb1 = 0.5d0/deta*(pn(m)+pn(mm)) + 0.5d0*f*kx/w
129   190       tmpb2 = tmpa2
130   190       do 200 k=0,n
131   200           b(k,kk,m) = tmpb2 * cdzs(k,kk)
132   200           b(k,k,m) = b(k,k,m) + tmpb1
   c****
   c
133   c  Surface boundary condition
134   c  do 220 j=0,m
135   220       do 220 j=1,mm
136   220           a(n,kk,j) = 0.d0
137   220           b(n,kk,j) = cdz(n,kk)
138   220       c(n,kk,j) = 0.d0
139   c
140   c  Bottom boundary condition
141   c  do 230 j=1,mm
142   230       tmpa = -0.25d0*pn(j)**2/desq*(h(j+1)-h(j-1))
143   230       a(0,0,j) = tmpa
do 240 kk=1,n
a(0,kk,j) = 0.0d0
240 tmpb1 = 2.0d0/h(j)*((0.5d0*pn(j))/deta*(h(j+1)-h(j-1))**2 +
1 (f**2-w**2)/bvfsq(z(0,j))
tmpb2 = 0.5d0*pn(j)/deta * f*kx/w * (h(j+1)-h(j-1))
do 250 kk=0,n
250 b(0,kk,j) = tmpb1 * cdz(0,kk)
b(0,0,j) = b(0,0,j) + tmpb2
tmpc = -tmpa
c(0,0,j) = tmpc
do 260 kk=1,n
c(0,kk,j) = 0.0d0
c 230 continue
c 240 c 
 250 c Set the right hand side vector p
 260 c 
do 280 j=0,m
 280 p(k,j) = 0.0d0
p(n/4,m/4) = 1.0d0
p(n/4,3*m/4) = 1.0d0
return
end
0001 c subroutine qsearch(wkng,rii,wknew)
0002 c Subroutine to perform a quadratic search by finding the freq/wavnum wknew
0003 c corresponding to the point where a parabola passing through the points
0004 c wkng(1),rl(1) ; wkng(2),rl(2) ; wkng(3),rl(3) takes a minimum where
0005 c rl(j) = 1 / ( rii(j) ) . The reciprocal of the rii values is taken
0006 c because the singularity at the resonance peak is of the approximate form:
0007 c rii = 1/(wk-wko)**2 where wko is the resonant freq/wavnum. Therefore
0008 c 1/rii is approximately parabolic in shape and its minimum (at wko) is well
0009 c estimated by a quadratic search.
0010 c double precision wknew,wkng(3),rii(3),rl(3),x1,x2,x3
0011 c do 10 k=1,3
0012 10 rl(k) = 1.0d0 / rii(k)
x1 = ( wkng(3) - wkng(1)) * ( rl(2) - rl(1))
x2 = ( wkng(2) - wkng(1)) * ( rl(3) - rl(1))
x3 = ( wkng(3) + wkng(1)) * x1 - ( wkng(2) + wkng(1)) * x2
wknew = 0.5d0 * x3 / ( x1 - x2)
return
end
subroutine response(p,pint)
0002 c Evaluates the integral of p**2 dy dz
0003 c parameter ( m=31, n=6, 
0004 c implicit double precision (a-h,p-z)
0005 c dimension p(0:n,1)
0006 common /dchb/ pi,sig(0:n),cp(0:n,0:n),cf(0:n,0:n),cd(0:n,0:n),
0007 1 cdxs(0:n,0:n),cint(0:n,0:n),cdszs(0:n,0:n),
0008 2 cdxszs(0:n,0:n)
0009 common /geom/ h(0:m),pn(0:m),yn(0:2*mm)
0010 pint = 0.0d0
0011 do 10 j=1,mm
tmp = 1.0d0/ln(j)/h(j)
do 10 kk=0,n
10    pint = pint + tmp*cint(0,kk)*p(kk,j)**2
return
end

subroutine uv(p)

Computes the solutions for the other variables by differencing the solution for p.
parameter ( m=31, n=6, 
1 mm=m-1, nm=n-1, mp=m+1, np=n+1)
implicit double precision (a-h,p-z)
dimension p(0:n,0:m)
dimension wrk(0:n,0:n)
double precision kx
common /const/ kx,w,f,y,f,yls
common /data/ delta,desq
common /dchb/ pi,sig(0:n),cp(0:n,0:n),cf(0:n,0:n),cd(0:n,0:n), 
1 cdz(0:n,0:n), cint(0:n,0:n), cdzs(0:n,0:n), 
2 cdzsiz(0:n,0:n)
common /geom/ 
1 h(0:m),pp(0:n,m),yn(0:2**mm)
common /sol/ uu(0:n,mm),vv(0:n,m),ww(0:n,mm), 
1 wsig(0:n,mm),pphi(0:n,mm),rrho(0:n,mm),ppsi(m)
z(k,j) = 0.5d0 * h(j) * (s(k)-1.d0)

Normalize the pressure solution
do 70 k=0,n
wrk(k,0) = ( 0.5d0*( p(k,0)+p(k,1) ) )**2
pnorm = 0.25d0*(h(0)+h(1)) * dotp(np,cint(0,0),np,wrk(0,0),1)
do 80 j=1,mm
pnorm = pnorm + 0.5d0*(h(j+1)-h(j-1))*p(0,j)**2

Compute the u (alongshelf) velocity component
tmp3 = w*kk/f
tmp0 = f/(f+f-w*w)
do 100 j = 1,mm

tmp1 = 0.5d0 * p(n)/delta
tmp2 = (h(j+1)-h(j-1))/h(j) * tmp1
do 100 k=0,n
uu(k,j) = dotp(np,cdz(k,0),np,p(0,j),1)
do 100 k=0,n
uu(k,j) = tmp0 * (-tmp1 * (p(k,j+1)-p(k,j-1)) - 
1 tmp2 * (1.0d0-sig(k)) * uu(k,j) - 
2 tmp3 * p(k,j) )

Alternative normalization: adjust amplitude so that energy

flux = int y=0,1 z=0,-h (pu) dz dy = 1.0 m**5 s**-2
To implement, uncomment the next 14 lines of code and switch
format 90 in subroutine out.

pnorm = 0.d0
do 105 j=1,mm
do 106 k=0,n
wrk(k,0) = p(k,j)*uu(k,j)
42
c105  pnorm = pnorm + 0.5d0*h(j)/pn(j)*deta*
0054  c 1    dotp(np,cint(0,0),np,wrk(0,0),1)
0055  c pnorm = 1.0d0/dsqrt(dabs(pnorm))
0056  c do 107 k=0,n
0057  c p(k,0) = pnorm * p(k,0)
0058  c107  p(k,m) = pnorm * p(k,m)
0059  c do 108 j=1,mm
0060  c do 108 k=0,n
0061  c p(k,j) = pnorm * p(k,j)
0062  c108  uu(k,j) = pnorm * uu(k,j)
0063  c
0064  c Compute the v (across-shelf) velocity component
0065  tmp0 = 0.5d0*f*kx/w
0066  tmp1 = w/(f**2-w**2)
0067  c do 110 j=2,mm
0068  c do 110 j=1,m
0069  c110  tmp2 = 0.5d0*(pu(j)+pu(j-1))/deta
0070  tmp3 = tmp2*(h(j)-h(j-1))/(h(j)+h(j-1))
0071  c do 110 k=0,n
0072  c tmp4 = dotp(np,cdz(k,0),np,p(0,j),1) +
0073  c110  dotp(np,cdz(k,0),np,p(0,j-1),1)
0074  c110  vv(k,j) = tmp1*(tmp2*(p(k,j)-p(k,j-1))+tmp3*(1.0d0-sig(k))
0075  c110  *tmp4 + tmp0*(p(k,j)+p(k,j-1))
0076  c
0077  c Compute ww, the vertical z coordinate velocity component
0078  do 130 j=1,mm
0079  do 130 k=0,n
0080  c130  ww(k,j) = 2.0d0*w / ( h(j)*bvfsq(z(k,j)) ) *
0081  c130  dotp(np,cdz(k,0),np,p(0,j),1)
0082  c
0083  c Compute wsig, the vertical sigma coordinate velocity component
0084  tmp0 = w/(f**2-w**2)
0085  tmp1 = f*kx/w
0086  c do 140 j=1,mm
0087  c140  tmp2 = 0.5d0*pu(j)/deta
0088  tmp3 = tmp2 * (h(j)+h(j-1))/h(j)
0089  c do 140 k=0,n
0090  c140  wsig(k,j) = 2.0d0/h(j)*ww(k,j) + tmp3*(1.0d0-sig(k))*tmp0*
0091  c140  *tmp2*(p(k,j+1)-p(k,j-1)) + tmp3*(1.0d0-sig(k))
0092  c140  *dotp(np,cdz(k,0),np,p(0,j),1) +
0093  c140  tmp1*p(k,j))
0094  c
0095  c Compute the density perturbation
0096  rho0 = 1000.0d0
0097  g = 9.81d0
0098  c do 150 j=1,mm
0099  c150  tmp = -2.0d0*rho0 / (g*h(j))
0100  c do 150 k=0,n
0101  c150  rrho(k,j) = tmp * dotp(np,cdz(k,0),np,p(0,j),1)
0102  c
0103  c Compute the transport streamfunction
0104  c do 160 j=1,m
0105  c160  ppsi(j) = h(j)/kx*0.5d0*dotp(np,cint(0,0),np,vv(0,j),1)
0106  c
0107  c compute the baroclinic pressure perturbation
0108  c do 170 j=1,mm
0109  c do 170 k=0,n

43.
pphi(k,j) = p(k,j) - p(n,j)

return
end

Subroutines to solve block tri-diagonal system of equations without pivoting strategy with the dimensions of the block matrices being n x n (n is any integer greater than 1).

These routines were obtained from the book "Computational Fluid Mechanics and Heat Transfer" by D.A. Anderson et.al., 1984, McGraw-Hill. This version converted to double precision and tested 4/1/87 by John Wilkin.

subroutine nbtridp(a,b,c,d,il,iu,order)
implicit double precision (a-h,p-z)
dimension a(1),b(1),c(1),d(1)
integer order,ordsq

a = sub diagonal matrix
b = diagonal matrix
c = sup diagonal matrix
d = right hand side vector
il = lower value of index for which matrices are defined
iu = upper value of index for which matrices are defined
(solution is sought for btrid(a,b,c)*x=d for indices of x between il and iu inclusive. solution is over-written in d vector).
order = order of a,b,c matrices and length of d vector at each point denoted by index i (order can be any integer greater than 1)
ordsq = order**2

forward elimination
i = il
iomat = 1+(i-1)*ordsq
iovec = 1+(i-1)*order
call ludeco(b(iomat),order)
call lusolv(b(iomat),d(iovec),d(iovec),order)
do 100 j=1,order
   iomatj = iomat+(j-1)*order
   call lusolv(b(iomat),c(iomatj),c(iomatj),order)
100 continue

i = i+1
i amat = 1+(i-1)*ordsq
iovec = 1+(i-1)*order
imat = i amat-ordsq
iovec = iovec-order
call mulput(a(iamat),d(iovec),d(iovec),order)
do 300 j=1,order
   iomatj = iomat+(j-1)*order
   imatj = imat+(j-1)*order
   call mulput(a(iomat),c(iimatj),b(iomatj),order)
300 continue
call ludeco(b(iomat),order)
call lusolv(b(iomat),d(iovec),d(iovec),order)
if(i.eq.iu) go to 500
do 400 j=1,order
imatj = imat+(j-1)*order
call lusolv(b(iomat),c(imatj),c(imatj),order)
400 continue
go to 200
500 continue

c
back substitution
i = iu
600 continue
i = i-1
imat = 1+(i-1)*ordsq
iovec = 1+(i-1)*order
ilvec = ilvec+order
call mulput(c(iomat),d(ilvec),d(iovec),order)
if(i.gt.il) go to 600
return
end

subroutine to calculate l-u decomposition of a given matrix a and
store the result in a. (no pivoting strategy is employed)

subroutine ludeco(a,order)
implicit double precision (a-h,p-z)
integer order
dimension a(order,1)
do 8 jc=2,order
8       a(i,jc) = a(i,jc)/a(i,1)
jrc = 1
10     continue
jrc = jrc+1
jrjcml = jrjc-1
jrjcp1 = jrjc+1
do 14 jr=jrjc,order
    sum = a(jr,jrjc)
17     do 12 jm=1,jrjcml
12     sum = sum - a(jr,jm)*a(jm,jrjc)
20     a(jr,jrjc) = sum
21     if(jrjc.eq.order) return
do 18 jc=jrjcp1,order
22     sum = a(jrjc,jc)
23     do 16 jm=1,jrjcml
16     sum = sum - a(jrjc,jm)*a(jm,jc)
26     a(jrjc,jc) = sum/a(jrjc,jrjc)
27     go to 10
28     end

subroutine to multiply a vector b by a matrix a, subtract result
from another vector c and store result in c. thus vector c is
overwritten.

subroutine mulput(a,b,c,order)
implicit double precision (a-h,p-z)
dimension a(1),b(1),c(1)
integer order
do 200 jr=1,order
sum = 0.0
   do 100 jc=1,order
      ia = jr+(jc-1)*order
      sum = sum + a(ia)*b(jc)
   100   c(jr) = c(jr) - sum
   return
   end

 subroutine to solve linear algebraic system of equations a*c=b and
store results in vector c. matrix a is input in l-u decomposition
form. (no pivoting strategy has been employed to compute the l-u
decomposition of the matrix a).

 subroutine lusolv(a,b,c,order)
 implicit double precision (a-h,p-z)
 integer order
 dimension a(order,1),b(1),c(1)

 c first l(inv)*b
 c(j) = c(j)/a(j,1)
do 14 jr=2,order
   jrm1 = jr-1
   sum = b(jr)
do 12 jm=1,jrm1
   sum = sum - a(jr,jm)*c(jm)
   c(jr) = sum/a(jr,jr)
14    c(jr) = c(jr) + sum
12    return
100   end

 subroutine ctwplot(p)
 c Since the plotting software available to different users varies widely,
 c this version of the program presented for distribution only has
dummy plotting subroutines. However, if you have access to the N.C.A.R.
System Plot Package routines you can replace this dummy subroutine with
those listed below. Even if you can't use those routines directly, you
c may find it useful to look at them to see how to convert the solutions
computed in sigma coordinates to regular z coordinates.
 return
 entry displot
 return
 end
subroutine ctwpplot(p)

Plots across-shelf structures of all variables using plotting routines from the N.C.A.R System Plot Package.

Plots are in eta, z coordinates.

To use a different graphics software package you will need to replace the calls to routines SET, PWRITY, PERIM, CONREC and FRAME.

It is routine CONREC which actually creates the contour plot of array PLT dimensioned NZ by NM. When interpolating from eta, sigma coordinates to eta, z coordinates, routine YZPLOT flags z points deeper than the water depth with the value 1.e+35 which the contouring routine CONREC ignores when drawing contours.

parameter ( m=31, n=6,
1 mm=m-1, mm=n-1, mp=m+1, np=n+1 )

parameter ( nz=201 )

implicit double precision ( a-h, p-y )

dimension p(0:n,0:m)

double precision  kx

common /const/ kx,w,f,yf,yls

common /data/ data, desq

common /dchb/ pi,sig(0:n),cp(0:n,0:n),cf(0:n,0:n),cd(0:n,0:n),
1 cdz(0:n,0:n),cint(0:n,0:n),cdzs(0:n,0:n),
2 cdzsdx(0:n,0:n)

common /geom/ h(0:m),pn(0:m),yn(0:2*mm)

common /idev/ / iterm, jterm, idsk

logical term, devout, tout, wout, wsgout, rhou, plot,
1 phiout, psiout, plot

common /logical/ / term, devout, tout, wout, wsgout, rhou, plot,
1 phiout, psiout

common /sol/ uu(0:n,m),vv(0:n,m),ww(0:n,mm),
1 wsig(0:n,mm),pphi(0:n,mm),rrho(0:n,mm),ppsi(m)

dimension x(m,0:n),label(100),lbl2(10)

real pltd(mm,nx)

real spval, epsval, cntmin, cntmax, cnttinc

common /conre1/ioffp, spval, epsval, cntmin, cntmax, cnttinc, nonzero

character*40 lbl1

iofp=1

spval=1.e+35

if(term)write(iterm,5)

format( enter a label for the modal structures plot',/,
1 ' ---- 40 uppercase characters max ----->')

read(iterm,6)lbl1

format(a)

encode(40,7,1lbl2)lbl1

format(a40)

call pwrity(512,190,1lbl2,40,1,0,0)

if(term)write(jterm,8)

format( plotting across-shelf structures ....... )

encode(30,9,label)sngl(w),sngl(kx)

format(7h0MEGA= ,ipe9.3,5h K=,ipe9.3)

call pwrity(512,250,label,30,1,0,0)

encode(30,10,label)sngl(t),sngl(y1)

format(7h F= ,ipe9.3,5h YL=,ipe9.3)

call pwrity(512,220,label,30,1,0,0)

call set(0.05,.95,.3,.8,1.,float(mm),1.,float(nx),1)
encode(30,11,label)
11 format(30h ALONGSHELF VELOCITY )
call pwrite(512,980,label,30,1,0,0)
call perim(1,10,1,10)
call yzplot(uu,plt,-h(m),0.d0)
call conrec(plt,mm,mm,nx,0.,0.,0.,1,-1,-682)
call frame
encode(30,21,label)
21 format(30h ACROSS-SHELF VELOCITY )
call pwrite(512,980,label,30,1,0,0)
call pwrite(512,190,lb12,40,1,0,0)
call perim(1,10,1,10)
call yzplot(vv,plt,-h(m),0.d0)
call conrec(plt,mm,mm,nx,0.,0.,0.,1,-1,-682)
call frame
encode(30,31,label)
31 format(30h VERTICAL VELOCITY (Z COORD) )
call pwrite(512,980,label,30,1,0,0)
call pwrite(512,190,lb12,40,1,0,0)
call perim(1,10,1,10)
call yzplot(ww,plt,-h(m),0.d0)
call conrec(plt,mm,mm,nx,0.,0.,0.,1,-1,-682)
call frame
encode(30,41,label)
41 format(30h VERTICAL VELOCITY (SIG COORD))
call pwrite(512,980,label,30,1,0,0)
call pwrite(512,190,lb12,40,1,0,0)
call perim(1,10,1,10)
call yzplot(wsig,plt,-h(m),0.d0)
call conrec(plt,mm,mm,nx,0.,0.,0.,1,-1,-682)
call frame
encode(30,51,label)
51 format(30h PRESSURE )
call pwrite(512,980,label,30,1,0,0)
call pwrite(512,190,lb12,40,1,0,0)
call perim(1,10,1,10)
call yzplot(p,plt,-h(m),0.d0)
call conrec(plt,mm,mm,nx,0.,0.,0.,1,-1,-682)
call frame
encode(30,61,label)
61 format(30h BAROCLINIC PRESSURE )
call pwrite(512,980,label,30,1,0,0)
call pwrite(512,190,lb12,40,1,0,0)
call perim(1,10,1,10)
call yzplot(pphi,plt,-h(m),0.d0)
call conrec(plt,mm,mm,nx,0.,0.,0.,1,-1,-682)
call frame
encode(30,71,label)
71 format(30h DENSITY PERTURBATION )
call pwrite(512,980,label,30,1,0,0)
call pwrite(512,190,lb12,40,1,0,0)
call perim(1,10,1,10)
call yzplot(rrho,plt,-h(m),0.d0)
call conrec(plt,mm,mm,nx,0.,0.,0.,1,-1,-682)
call frame
return
end
subroutine yzplot(var,plt,zbot,ztop)

This subroutine expands a solution defined at the Chebyshev
collocation points onto a rectangular grid of nz by mm points
implicit double precision (a-h,p-z)
parameter ( m=31, n=6,
1
parameter ( nz=201 )
common /dchb/ pi,sig(0:n),cp(0:n,0:n),cf(0:n,0:n),cd(0:n,0:n),
1
2
cdx(0:n,0:n),cint(0:n,0:n),cdzs(0:n,0:n),
common /geom/ h(0:m)
dimension csum(0:n),wgt(0:n),amp(0:n,mm),var(0:n,m)
real plt(mm,nz)

compute depth increments
dh=(ztop-zbot)/dble(nz-1)

integrals of t(x) over -1:x:1
csum(0) = 2.d0
do 100 k=1,n,2
100  csum(k) = 0.d0
do 110 k=2,n,2
110  csum(k) = -2.d0 / ( dble(k*k)-1.d0 )

transform to amplitude space
do 120 j=1,mm
do 120 k=0,n
120  amp(k,j) = dotp(np,cf(k,0),np,var(0,j),1)

interpolate values from amp into plt
do 130 k=1,nx
130  zz = zbot + dble(k-1)*dh
do 140 j=1,mm
140  ss = 1.d0 + (2.d0*zz/h(j))
    if( ss.1t.(-1.d0) ) then
    plt(j,k) = 1.e+35
    else

    arg = - dacos(ss)
    arg = -dble(acos(angl(ss)))
    wgt(0) = 1.d0
    do 150 i=1,n
    wgt(i) = dcos( dble(i)*arg ) - 0.5d0*csum(i)
    plt(j,k) = sngl(dotp(np,amp(0,j),1,wgt(0),1))
150  end if
7447  140 continue
7448  130 continue
7449  return
7450  end

subroutine displot

Plots dispersion curve using NCAR System Plot Package routines
parameter ( nwppts=500 )
double precision xx,w,f,yl,yls
common /const/ xx,w,f,yl,yls
common / idev / iterm, jterm, idsk
logical
  term, pout, uout, vout, wout, wsgout, rhoout,
  phiout, psiout, plot
common / logical/
  term, pout, uout, vout, wout, wsgout, rhoout,
  phiout, psiout
common / disp/
  nkpts, wavenum, (nwvpts+1), freq(nwvpts+1, 2)
dimension
  label(100), lb12(10)
character
 eryl 
if(term) write(iterm, 200)
  200 format(' Enter a label for the dispersion plot', /, 
  1       ' <---- 40 uppercase characters max ------->')
  read(iterm, 210) lb11
  210 format(a)
  encode(40, 220, lb12) lb11
  220 format(a40)
call pwrty(450, 900, lb12, 40, 1, 0, 0)
do 230 i = 1, nkpts+1
  freq(i, 2) = smgl(f)
call displa(2,1,1)
call anotat(21HAWENUMBER (/METRES$), 17HFREQUENCY (/SEC$), 
  1       1 , 1, -1, 0.)
call ezmyy(wavenum, freq, nwvpts+1, 2, nkpts+1, 
  1       21HCTW DISPERSION CURVES$)
call frame
  return
end
A listing and full documentation is presented for a FORTRAN computer program which computes the dispersion curves and across-shelf modal structures of free coastal-trapped waves in a coastal channel. The three velocity components, mass transport streamfunction, and density and pressure perturbation fields are computed. The solution procedure used (horizontal finite differences on a staggered grid and an expansion in the vertical in terms of modified Chebyshev polynomials) makes the solution compatible (without interpolation) with the numerical scheme employed in the Haidvogel et al. (1988) primitive equation ocean circulation model.
Attn: Stella Sanchez-Wade
Documents Section
Scripps Institution of Oceanography
Library, Mail Code C-075C
La Jolla, CA 92039

Hancock Library of Biology & Oceanography
Alan Hancock Laboratory
University of Southern California
University Park
Los Angeles, CA 90089-0371

Gifts & Exchanges
Library
Bedford Institute of Oceanography
P.O. Box 1006
Dartmouth, NS, B2Y 4A2, CANADA

Office of the International Ice Patrol
c/o Coast Guard R & D Center
Avery Point
Groton, CT 06340

Library
Physical Oceanographic Laboratory
Nova University
8000 N. Ocean Drive
Dania, FL 33304

NOAA/EDIS Miami Library Center
4301 Rickenbacker Causeway
Miami, FL 33149

Library
Skidaway Institute of Oceanography
P.O. Box 13687
Savannah, GA 31416

Institute of Geophysics
University of Hawaii
Library Room 252
2525 Correa Road
Honolulu, HI 96822

Library
Chesapeake Bay Institute
4800 Atwell Road
Shady Side, MD 20876

MIT Libraries
Serial Journal Room 14E-210
Cambridge, MA 02139

Director, Ralph M. Parsons Laboratory
Room 48-311
MIT
Cambridge, MA 02139

Marine Resources Information Center
Building E38-320
MIT
Cambridge, MA 02139

Library
Lamont-Doherty Geological Observatory
Colombia University
Palisades, NY 10964

Library
Serials Department
Oregon State University
Corvallis, OR 97331

Pell Marine Science Library
University of Rhode Island
Narragansett Bay Campus
Narragansett, RI 02882

Working Collection
Texas A&M University
Dept. of Oceanography
College Station, TX 77843

Library
Virginia Institute of Marine Science
Gloucester Point, VA 23062

Fisheries-Oceanography Library
151 Oceanography Teaching Bldg.
University of Washington
Seattle, WA 98195

Library
R.S.M.A.S.
University of Miami
4600 Rickenbacker Causeway
Miami, FL 33149

Maury Oceanographic Library
Naval Oceanographic Office
Bay St. Louis
NSTL, MS 39522-5001