OCS Study MMS 94-0020

Technical Manual for a Coupled Sea-Ice/Ocean Circulation Model (Version 1)

Katherine S. Hedström
Institute of Marine and Coastal Sciences
Rutgers University

Technical Manual for a Coupled Sea-Ice/Ocean Circulation Model (Version 1)

Katherine S. Hedström
Institute of Marine and Coastal Sciences
Rutgers University

April 1994

This study was funded by the Alaska Outer Continental Shelf Region of the Minerals Management Service, U.S. Department of the Interior, Anchorage, Alaska, through Cooperative Agreement **14-35-0001-30675** with Rutgers University, Institute of Marine and Coastal Sciences.

This document was prepared with LaTeX and **xfig**.

Acknowledgments for SPEM version 3.0

I would like to thank Dale Haidvogel, whose patience in describing the SPEM to me made this document possible, and who made many helpful suggestions while I was writing it. I would also like to thank him for writing up the model physics and for allowing me to include that description here. Thanks go to Hsiao-Ming Hsu for his careful reading of an early version of this user's guide.

Aike Beckmann has been very helpful in my efforts to upgrade this document, both in contributing information about model changes and in carefully checking what I have written.

Acknowledgments for SPEM version 3.9

John Wilkin and Jim Mansbridge have been a pleasure to work with on the masking and rotated mixing tensors, and have even provided me with the material to include here. Jim and John also provided the Tasman Sea example, while Anthony Macks and Jason Middleton provided the upwelling/downwelling example. Thanks to the Usenet community for providing great tools like **perl**, **patch**, **cpp**, **rcs**, and **imake** to aid in software development (and to make it more fun). Cathy Lascara talked me into trying **imake** with SPEM which has been well worth the trouble.

Thanks again to Dale Haidvogel for giving me the time to rewrite this the way I wanted to, not to mention the freedom to make some of these changes to SPEM. Rowan Hughes was kind enough to read this manual critically and to make a large number of grammatical corrections. Thanks especially to the SPEM user community for making this effort worthwhile.

Development and testing of the SPEM model has been funded by grants from the Office of Naval Research (SC-53789), the National Science Foundation (OCE 90-12754-01), the Minerals Management Service (14-35-0001-30675), and the National Aeronautics and Space Administration (GC-R-261348-006-C).

Abstract

A coupled sea-ice/ocean circulation model is described. Principal components of the coupled model include the semi-spectral primitive equation model of Haidvogel et al. (1991) and the dynamic-thermodynamic sea ice model of Hibler (1979). The resulting coupled model is applied to the Western Arctic Ocean. This user's manual describes the equations and algorithms used in both the ice and ocean components, the methods of coupling, and configuration details for the Western Arctic.

Contents

List of Figures

1 Introduction

A coupled sea-ice/ocean circulation model is described. Principal components of the coupled model include the semi-spectral primitive equation model of Haidvogel et al. [12] and the dynamic-thermodynamic sea ice model of Hibler [20]. The resulting coupled model is applied to the Western Arctic Ocean. This user's manual describes the equations and algorithms used in both the ice and ocean components, the methods of coupling, and configuration details for the Western Arctic.

Sections 2 and 3 describe the SPEM physics and numerical techniques and are an update of the description in Haidvogel, Wilkin, and Young [12]. Section 4 lists the SPEM subroutines, functions and variables. Section 5 describes the support programs which are needed to provide SPEM with data files. Section 6 describes the Lagrangian floats and how to use them, including the plotting post-processor **fltplt**. Section 7 describes John Wilkin's postprocessing programs **readhist**, **shorthist**, and **plothist**. The ice model equations are presented in §8, which is derived from Hibler [21] and Hibler [22]. The subroutines and variables for the ice model as well as the coupling to the ocean model are described in §9. The process of configuring the coupled model for a particular application is described in §10, including a discussion of the Western Arctic application.

The principle attributes of the ocean model are as follows:

General

- Primitive equations with temperature, salinity, and an equation of state.
- Hydrostatic and Boussinesq approximations.
- Optional Lagrangian floats ("water parcels").

Horizontal

- Orthogonal-curvilinear coordinates.
- Arakawa C grid.
- Closed basin, periodic channel, doubly periodic, or user supplied boundary conditions.
- Masking of land areas.

Vertical

- Sigma (bottom following) coordinate.
- Chebyshev modal expansion.
- Rigid lid on top.

Mixing options

- Horizontal along-sigma Laplacian and biharmonic viscosity and diffusion.
- Horizontal Laplacian diffusion along constant z or *in situ* density surfaces.
- Horizontal free-slip or no-slip boundaries.
- Vertical harmonic viscosity and diffusion with a spatially variable coefficient.
- Vertical convective mixing when statically unstable.

The principle attributes of the ice model are as follows:

General

- Momentum equations with a non-linear viscous-plastic rheology.

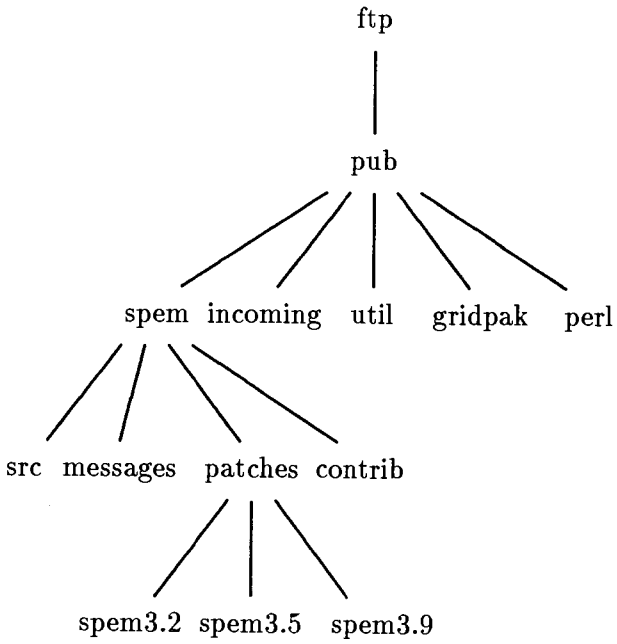

Figure 1: Tree structure of anonymous ftp directory

coast0.lines.Z	A high-resolution coastline file for **xcoast** and **plothist** (compressed).
coast1.lines.Z	A low-resolution coastline file for **xcoast** and **plothist** (compressed).
maskedge.tar.Z	A program for finding the input to **mask** from the bathymetry mask on the grid. It is written in C++ and requires **libg++**.

Another directory of interest is pub/util, which contains:

fimake.tar.Z	Fortran **imake** configuration files. This will be updated once I have read the **imake** book (DuBois [10]).
fperl.tar.Z	The perl scripts which I use with Fortran, described in Appendix E.
spemlib.tar.Z	Library used by SPEM and **gridpak**.
spemlib patches	If in doubt about your patchlevel, just get the whole spemlib again.
xpots	An X11 application for generating a sequence of numbers from zero to one. Will be used by a future version of **gridpak**.

The Unix convention is that a filename ending in .Z has been compressed with the Unix **compress** utility. The corresponding **uncompress** is included with many Unix systems, but its source code is also available as part of the gnu **gzip** package. Likewise, a .tar ending designates a file which was created with the Unix **tar** (tape archive) utility. The steps in unpacking these files are:

```
% uncompress gridpak.tar.Z
% tar xvf gridpak.tar
```

Note that both .tar and .Z files are binary and must be retrieved using binary mode in ftp.

Included in the **spemlib** tar file is the **mud2** elliptic solver. **mud2** is copyrighted by NCAR and they request that you not make any changes to it without their permission.

If you are unable to acquire the code in this fashion feel free to contact me at:

```
      common // var1, var2
 44   format(//)
```

and the new Fortran 90 string concatenation:

```
mystring = 'Hello, ' // 'World!'
```

3. Macro replacement. One feature of **cpp** is that you can define macros and have it perform replacements. The code:

```
#define REAL double precision
      REAL really_long_variable, second_long_variable
```

becomes

```
      double precision really_long_variable, second_long_variable
```

and you run the risk of creating lines which are longer than 72 characters in length.

- I have started declaring all variables as being of type **BIGREAL** or **SMALLREAL** and then defining them to be **real*8** and **real*4** respectively (except on a Cray). This usually works for variables, but some compilers do not like

```
real*8 function vmin(var1, var2)
```

If you run into one of these finicky compilers, you will just have to fix each **BIGREAL** function by hand.

- Through the years there have been several file formats for the grid file. Make sure that your grid-generation software and your version of **getgrid.F** are consistent. Likewise, make sure that you don't try to read a single precision binary file from a double precision SPEM program.

- We have also changed versions of **mud2** and are now using version 3.0. The argument list has changed in that **iparm** is now dimensioned differently. Make sure you are using the appropriate version of **mud2** for your code, in **gridpak** as well as in SPEM.

- The Lagrangian floats have not been fixed up to do the right thing near masked regions. If a float drifts near a land mask it will be horizontally interpolated with zeroes from inside the masked region, even with the bilinear option. It is not obvious how to fix this.

- The plotting has been set up to work with the masking. Some variables are plotted on ψ points, which has the disadvantage that one gridbox wide peninsulas will not be masked since there are no special values internal to the peninsula. Other variables are plotted on ρ points—the problem with that is the values end half a gridbox from masked regions and sometimes extend half a gridbox outside the domain. This can be confusing if you don't expect it.

- The vertical sigma coordinate was chosen as being a sensible way to handle variations in the water depth. It has been used with success when the maximum and minimum depths differ by a factor of twenty or less, and the depth variations are well resolved by the horizontal grid (see Appendix C). On the other hand, it is not at all difficult to get into a parameter range in which SPEM is ill-behaved, for instance it has been known to have trouble with a depth range spanning three orders of magnitude.

2 Ocean Model Formulation

2.1 Equations of motion

The primitive equations in Cartesian coordinates can be written:

$$\frac{\partial u}{\partial t} + \vec{v} \cdot \nabla u - fv = -\frac{\partial \phi}{\partial x} + \mathcal{F}_u + \mathcal{D}_u \tag{1}$$

$$\frac{\partial v}{\partial t} + \vec{v} \cdot \nabla v + fu = -\frac{\partial \phi}{\partial y} + \mathcal{F}_v + \mathcal{D}_v \tag{2}$$

$$\frac{\partial T}{\partial t} + \vec{v} \cdot \nabla T = \mathcal{F}_T + \mathcal{D}_T \tag{3}$$

$$\frac{\partial S}{\partial t} + \vec{v} \cdot \nabla S = \mathcal{F}_S + \mathcal{D}_S \tag{4}$$

$$\rho = \rho(T, S, P) \tag{5}$$

$$\frac{\partial \phi}{\partial z} = \frac{-\rho g}{\rho_o} \tag{6}$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0, \tag{7}$$

where, in standard notation:

$$(u, v, w) = \text{the } (x, y, z) \text{ components of vector velocity } \vec{v}$$
$$\rho_o + \rho(x, y, z, t) = \text{total } \textit{in situ} \text{ density}$$
$$T(x, y, z, t) = \text{potential temperature}$$
$$S(x, y, z, t) = \text{salinity}$$
$$P = \text{total pressure } P \approx -\rho_o g z$$
$$\phi(x, y, z, t) = \text{dynamic pressure } \phi = (P/\rho_o)$$
$$f(x, y) = \text{Coriolis parameter}$$
$$g = \text{acceleration of gravity}$$
$$(\mathcal{F}_u, \mathcal{F}_v, \mathcal{F}_T, \mathcal{F}_S) = \text{forcing terms}$$

and

$$(\mathcal{D}_u, \mathcal{D}_v, \mathcal{D}_T, \mathcal{D}_S) = \text{diffusive terms.}$$

Equations (1) and (2) express the momentum balance in the x- and y-directions, respectively. The time evolution of the perturbation temperature and salinity fields, $T(x, y, z, t)$ and $S(x, y, z, t)$, are governed by the advective-diffusive equations (3) and (4). The equation of state is given by equation (5). In the Boussinesq approximation, density variations are neglected in the momentum equations except in their contribution to the buoyancy force in the vertical momentum equation (6). Under the hydrostatic approximation, it is further assumed that the vertical pressure gradient balances the buoyancy force. Lastly, equation (7) expresses the continuity equation for an incompressible fluid. For the moment, the effects of forcing and dissipation will be represented by the schematic terms $\mathcal{F}$ and $\mathcal{D}$, respectively. The horizontal and vertical mixing will be described more fully in §3.13.

make the coordinate transformation:

$$\hat{x} = x$$
$$\hat{y} = y$$
$$\sigma = 1 + 2(z/h)$$
$$z = h/2(\sigma - 1)$$

and

$$\hat{t} = t.$$

In the stretched system, the vertical coordinate σ spans the range $-1 \leq \sigma \leq 1$; we are therefore left with level upper ($\sigma = 1$) and lower ($\sigma = -1$) bounding surfaces. The chain rules for this transformation are:

$$\left(\frac{\partial}{\partial x}\right)_z = \left(\frac{\partial}{\partial x}\right)_\sigma + (1-\sigma)\frac{h_x}{h}\frac{\partial}{\partial \sigma}$$
$$\left(\frac{\partial}{\partial y}\right)_z = \left(\frac{\partial}{\partial y}\right)_\sigma + (1-\sigma)\frac{h_y}{h}\frac{\partial}{\partial \sigma}$$

and

$$\frac{\partial}{\partial z} = \frac{2}{h}\frac{\partial}{\partial \sigma}.$$

As a trade-off for this geometric simplification, the dynamic equations become somewhat more complicated. The resulting dynamic equations are, after dropping the carats:

$$\frac{\partial u}{\partial t} - fv + \vec{v}\cdot\nabla u = -\frac{\partial \phi}{\partial x} + (1-\sigma)\left(\frac{g\rho}{2\rho_o}\right)\frac{\partial h}{\partial x} + \mathcal{F}_u + \mathcal{D}_u \tag{8}$$

$$\frac{\partial v}{\partial t} + fu + \vec{v}\cdot\nabla v = -\frac{\partial \phi}{\partial y} + (1-\sigma)\left(\frac{g\rho}{2\rho_o}\right)\frac{\partial h}{\partial y} + \mathcal{F}_v + \mathcal{D}_v \tag{9}$$

$$\frac{\partial T}{\partial t} + \vec{v}\cdot\nabla T = \mathcal{F}_T + \mathcal{D}_T \tag{10}$$

$$\frac{\partial S}{\partial t} + \vec{v}\cdot\nabla S = \mathcal{F}_S + \mathcal{D}_S \tag{11}$$

$$\rho = \rho(T, S, P) \tag{12}$$

$$\frac{\partial \phi}{\partial \sigma} = \left(\frac{-gh\rho}{2\rho_o}\right) \tag{13}$$

$$\frac{\partial}{\partial x}(hu) + \frac{\partial}{\partial y}(hv) + h\frac{\partial \Omega}{\partial \sigma} = 0 \tag{14}$$

where

$$\vec{v} = (u, v, \Omega)$$
$$\vec{v}\cdot\nabla = u\frac{\partial}{\partial x} + v\frac{\partial}{\partial y} + \Omega\frac{\partial}{\partial \sigma}.$$

The vertical velocity in sigma coordinates is

$$\Omega(x, y, \sigma, t) = \frac{1}{h}\left[(1-\sigma)u\frac{\partial h}{\partial x} + (1-\sigma)v\frac{\partial h}{\partial y} + 2w\right].$$

the equations of motion (8)-(14) can be re-written (see, e.g., Arakawa and Lamb [4]) as:

$$\frac{\partial}{\partial t}\left(\frac{hu}{mn}\right)+\frac{\partial}{\partial \xi}\left(\frac{hu^2}{n}\right)+\frac{\partial}{\partial \eta}\left(\frac{huv}{m}\right)+\frac{\partial}{\partial \sigma}\left(\frac{hu\Omega}{mn}\right)$$
$$-\left\{\left(\frac{f}{mn}\right)+v\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right)-u\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right)\right\}hv=$$
$$-\left(\frac{h}{n}\right)\frac{\partial \phi}{\partial \xi}+(1-\sigma)\left(\frac{gh\rho}{2\rho_o n}\right)\frac{\partial h}{\partial \xi}+\mathcal{F}_u+\mathcal{D}_u \tag{19}$$

$$\frac{\partial}{\partial t}\left(\frac{hv}{mn}\right)+\frac{\partial}{\partial \xi}\left(\frac{huv}{n}\right)+\frac{\partial}{\partial \eta}\left(\frac{hv^2}{m}\right)+\frac{\partial}{\partial \sigma}\left(\frac{hv\Omega}{mn}\right)$$
$$+\left\{\left(\frac{f}{mn}\right)+v\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right)-u\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right)\right\}hu=$$
$$-\left(\frac{h}{m}\right)\frac{\partial \phi}{\partial \eta}+(1-\sigma)\left(\frac{gh\rho}{2\rho_o m}\right)\frac{\partial h}{\partial \eta}+\mathcal{F}_v+\mathcal{D}_v \tag{20}$$

$$\frac{\partial}{\partial t}\left(\frac{hT}{mn}\right)+\frac{\partial}{\partial \xi}\left(\frac{huT}{n}\right)+\frac{\partial}{\partial \eta}\left(\frac{hvT}{m}\right)+\frac{\partial}{\partial \sigma}\left(\frac{h\Omega T}{mn}\right)=\mathcal{F}_T+\mathcal{D}_T \tag{21}$$

$$\frac{\partial}{\partial t}\left(\frac{hS}{mn}\right)+\frac{\partial}{\partial \xi}\left(\frac{huS}{n}\right)+\frac{\partial}{\partial \eta}\left(\frac{hvS}{m}\right)+\frac{\partial}{\partial \sigma}\left(\frac{h\Omega S}{mn}\right)=\mathcal{F}_S+\mathcal{D}_S \tag{22}$$

$$\rho=\rho(T,S,P) \tag{23}$$

$$\frac{\partial \phi}{\partial \sigma}=-\left(\frac{gh\rho}{2\rho_o}\right) \tag{24}$$

$$\frac{\partial}{\partial \xi}\left(\frac{hu}{n}\right)+\frac{\partial}{\partial \eta}\left(\frac{hv}{m}\right)+\frac{\partial}{\partial \sigma}\left(\frac{h\Omega}{mn}\right)=0. \tag{25}$$

All boundary conditions remain unchanged.

3 Numerical Solution Technique (SPEM)

3.1 Vertical spectral method

The vertical (σ) dependence of the model variables is represented as an expansion in a finite polynomial basis set $P_k(\sigma)$; that is, we set

$$b(\xi,\eta,\sigma) = \sum_{k=0}^{N} P_k(\sigma)\hat{b}_k(\xi,\eta) \tag{26}$$

where the arbitrary variable b, introduced here for convenience, is any of u, v, ρ, T, S, ϕ, or Ω. The only restriction placed on the form of the basis polynomials is that $\frac{1}{2}\int_{-1}^{1} P_k(\sigma)d\sigma = \delta_{0k}$, i.e., only the lowest order polynomial may have a non-zero vertical integral. This isolates the external mode (or depth-averaged component of the field) in the amplitude of the $k = 0$ polynomial. In practice, the spectral technique does not explicitly evaluate the polynomial coefficients $\hat{b}_k$ but rather the actual variable values at "collocation" points (or equivalent grid points) σ_n, chosen to correspond to the location of the extrema of the highest order polynomial. The collocation point values b_n are thus defined as

$$b_n = b(\sigma_n) = \sum_{k=0}^{N} P_k(\sigma_n)\,\hat{b}_k \qquad 0 \le n \le N \tag{27}$$

and will be functions of (ξ,η). The polynomial coefficients $\hat{b}_k$ can be recovered from the collocation point values b_n by a linear matrix transformation. Consider a matrix F whose elements are

$$F_{nk} = P_k(\sigma_n) \tag{28}$$

and let b and $\hat{b}$ be column vectors whose elements are b_n and $\hat{b}_k$ respectively. Then

$$b = F\hat{b} \tag{29}$$

and hence

$$\hat{b} = F^{-1}b. \tag{30}$$

It is now straightforward to represent vertical differentiation and integration in terms of matrix operators. Consider a matrix R whose elements are

$$R_{nk} = \frac{\partial P_k(\sigma_n)}{\partial\sigma}. \tag{31}$$

Then the matrix $C_{DZ} = RF^{-1}$ will perform differentiation of the model field (b) in the vertical direction. This is denoted in the following subsections by the δ_σ operation, since

$$\delta_\sigma b = \frac{\partial P_k(\sigma_n)}{\partial\sigma}\hat{b}_k = R\hat{b} = RF^{-1}b = C_{DZ}b. \tag{32}$$

Similarly, consider a matrix S whose elements are

$$S_{nk} = \int_{\sigma_n}^{1} P_k(\sigma)\,d\sigma. \tag{33}$$

Then the matrix $C_{INT} = SF^{-1}$ will perform vertical integration, denoted in the following sections by the I_σ^1 operation, since

$$I_\sigma^1 b = \int_\sigma^1 b\,d\sigma = \sum_{k=0}^{N}\int_{\sigma_n}^{1} P_k(\sigma)\,d\sigma\,\hat{b}_k = S\hat{b} = SF^{-1}b = C_{INT}b. \tag{34}$$

By default, the SPEM uses the Chebyshev polynomials. The matrix operators F, R and S and collocation levels σ_n for this basis set are given in Appendix B.

including two land cells. The process of defining which areas are to be masked is described in §5.2, while this section describes how the masking affects the computation of the various terms in the equations of motion.

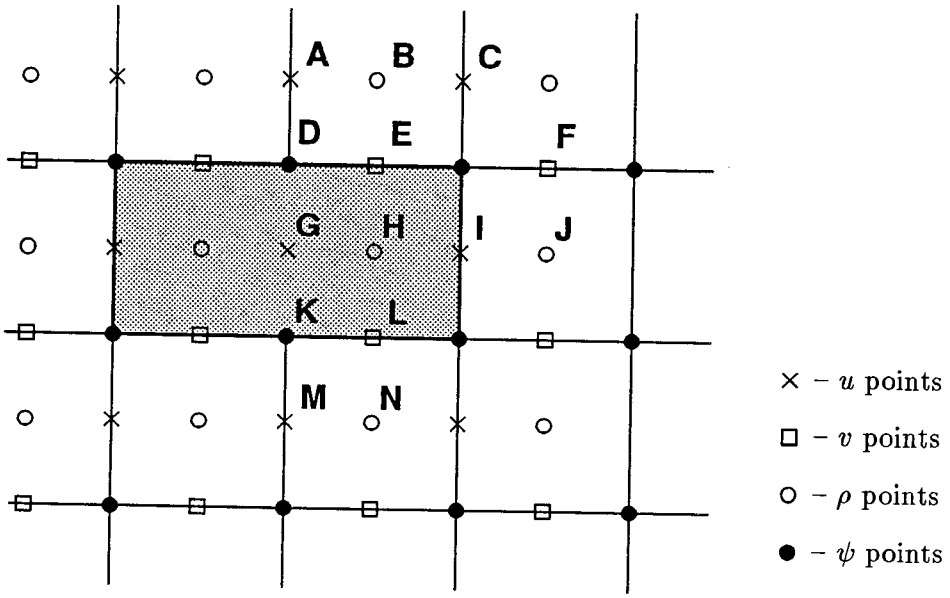

Figure 3: Masked region within the domain

3.4.1 Velocity equations

At the end of every timestep, the values of many variables within the masked region are set to zero (subroutine **zeroland**). This is appropriate for the v points **E** and **L** in Fig. 3, since the flow in and out of the land should be zero. It is likewise appropriate for the u point at **I**, but is not necessarily correct for point **G**. The only term in the u equation that requires the u value at point **G** is the horizontal viscosity, which has a term of the form $\frac{\partial}{\partial \eta}\nu\frac{\partial u}{\partial \eta}$. Since point **G** is used in this term by both points **A** and **M**, it is not sufficient to replace its value with that of the image point for **A**. Instead, the term $\frac{\partial u}{\partial \eta}$ is computed and the values at points **D** and **K** are replaced with the values appropriate for either free-slip or no-slip boundary conditions (subroutine **slipbound**). Likewise, the term $\frac{\partial}{\partial \xi}\nu\frac{\partial v}{\partial \xi}$ in the v equation must be corrected at the mask boundaries.

The fourth-order pressure gradient option (see §3.11) requires values at ρ points inside masked areas, such as point **H**. We have chosen to avoid this problem by reducing to second-order accuracy at the mask boundaries (subroutine **pcorrect**).

3.4.2 Temperature and salinity equations

The handling of masks by the temperature and salinity equations is similar to that in the momentum equations, and is in fact simpler. Values of T and S inside the land masks, such as point **H** in Fig. 3, are set to zero after every timestep. This point would be used by the horizontal diffusion term for points **B**, **J**, and **N**. This is handled by setting the first derivative terms at points **E**, **I**, and **L** to zero, to be consistent with a no-flux boundary condition.

Note that the equation of state must be able to handle $T = S = 0$ since this is the value inside masked regions.

$$\frac{\partial}{\partial t}\left(\frac{v\overline{h}^{\eta}}{\overline{m}^{\eta}\overline{n}^{\eta}}\right)+\delta_{\xi}\left\{\overline{v}^{\xi}\overline{\left(\frac{u\overline{h}^{\xi}}{\overline{n}^{\xi}}\right)}^{\eta}\right\}+\delta_{\eta}\left\{\overline{v}^{\eta}\overline{\left(\frac{v\overline{h}^{\eta}}{\overline{m}^{\eta}}\right)}^{\eta}\right\}+\frac{\overline{h}^{\eta}\overline{\Omega}^{\eta}}{\overline{m}^{\eta}\overline{n}^{\eta}}\frac{\partial v}{\partial\sigma}+\overline{v\delta_{\sigma}\left\{\frac{h\Omega}{mn}\right\}}^{\eta}$$

$$+\overline{\left\{\frac{fh}{mn}+\overline{v}^{\eta}\,h\delta_{\xi}\left(\frac{1}{n}\right)-\overline{u}^{\xi}\,h\delta_{\eta}\left(\frac{1}{m}\right)\right\}\overline{u}^{\xi}}^{\eta}=$$

$$-\frac{\overline{h}^{\eta}}{\overline{m}^{\eta}}\delta_{\eta}\phi+(1-\sigma)\frac{g\overline{h}^{\eta}\overline{\rho}^{\eta}}{2\rho_{o}\overline{m}^{\eta}}\delta_{\eta}h+\mathcal{D}_{v}+\mathcal{F}_{v} \tag{39}$$

$$\frac{\partial}{\partial t}\left(\frac{hT}{mn}\right)+\delta_{\xi}\left(\frac{u\overline{h}^{\xi}\overline{T}^{\xi}}{\overline{n}^{\xi}}\right)+\delta_{\eta}\left(\frac{v\overline{h}^{\eta}\overline{T}^{\eta}}{\overline{m}^{\eta}}\right)$$

$$+\frac{h\Omega}{mn}\frac{\partial T}{\partial\sigma}+T\delta_{\sigma}\left(\frac{h\Omega}{mn}\right)=\mathcal{D}_{T}+\mathcal{F}_{T} \tag{40}$$

$$\frac{\partial}{\partial t}\left(\frac{hS}{mn}\right)+\delta_{\xi}\left(\frac{u\overline{h}^{\xi}\overline{S}^{\xi}}{\overline{n}^{\xi}}\right)+\delta_{\eta}\left(\frac{v\overline{h}^{\eta}\overline{S}^{\eta}}{\overline{m}^{\eta}}\right)$$

$$+\frac{h\Omega}{mn}\frac{\partial S}{\partial\sigma}+S\delta_{\sigma}\left(\frac{h\Omega}{mn}\right)=\mathcal{D}_{S}+\mathcal{F}_{S} \tag{41}$$

$$\rho=\rho(S,T,P) \tag{42}$$

$$\phi(\sigma)=-\left(\frac{gh}{2\rho_{o}}\right)I_{\sigma}^{1}\rho \tag{43}$$

$$\delta_{\xi}\left\{\frac{u\overline{h}^{\xi}}{\overline{n}^{\xi}}\right\}+\delta_{\eta}\left\{\frac{v\overline{h}^{\eta}}{\overline{m}^{\eta}}\right\}+\delta_{\sigma}\left\{\frac{h\Omega}{mn}\right\}=0. \tag{44}$$

Here δ_{ξ} and δ_{η} denote simple centered finite-difference approximations to $\partial/\partial\xi$ and $\partial/\partial\eta$, with the differences taken over the distances $\Delta\xi$ and $\Delta\eta$, respectively; $\overline{(\quad)}^{\xi}$ and $\overline{(\quad)}^{\eta}$ represent averages taken over the distances $\Delta\xi$ and $\Delta\eta$; δ_{σ} represents a vertical derivative evaluated according to the prescription described in §3.1; and I_{σ}^{1} indicates the analogous vertical integral. The vertical advection terms are actually more complicated, as described in §3.5.

This method of averaging was chosen because it internally conserves first moments in the model domain, although it is still possible to exchange mass and energy through the open boundaries. The method is similar to that used in Arakawa and Lamb [4]; however, their scheme also conserves enstrophy.

3.7 Time stepping: depth-integrated flow ($k=0$ mode)

In continuous form, the horizontal momentum and continuity equations (19), (20) and (25) can be written:

$$\frac{\partial}{\partial t}\left(\frac{hu}{mn}\right)=R_{u}-\left(\frac{h}{n}\right)\frac{\partial\phi}{\partial\xi} \tag{45}$$

$$\frac{\partial}{\partial t}\left(\frac{hv}{mn}\right)=R_{v}-\left(\frac{h}{m}\right)\frac{\partial\phi}{\partial\eta} \tag{46}$$

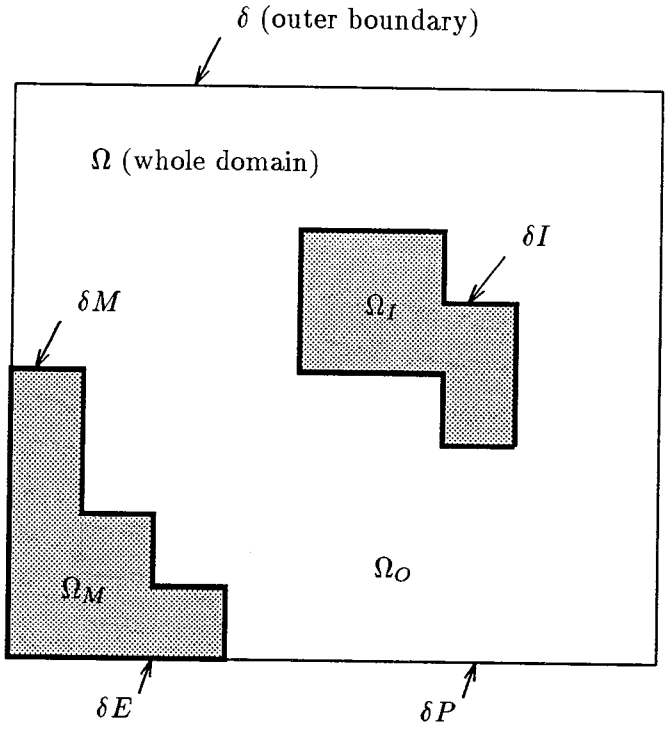

Figure 4: Domain regions and boundaries

3.8 Masking and the vorticity equation

The **mud2** elliptic solver accommodates singly- or doubly-periodic, mixed-derivative, and specified (Dirichlet) boundary conditions, but these boundary conditions must be applied on the edges of the rectangular computational domain. In order to extend the elliptic solver to accommodate masked land areas contiguous to the perimeter of the domain, or islands within the domain, modifications to the procedure for solving equation (55) are required. The capacitance matrix method is one such algorithm.

3.8.1 Capacitance matrix method

The description of the capacitance matrix algorithm presented here is from Wilkin et al. [40] and closely follows that of Milliff [30]. The notation used in this section to describe the subregions and boundaries of the computational domain is shown in Fig. 4. The full domain is denoted by Ω and is comprised of various subdomains: Ω_O, the *ocean* region; Ω_M, the *mainland* (which is any land area contiguous to the perimeter); and Ω_I, the *islands*. The perimeter of the computational domain is denoted by δ and is comprised of δP, the *perimeter* of the ocean region, and δE, the *exterior* of the mainland region. Land regions are separated from the ocean by boundaries δM of the *mainland* region, and δI for the *islands*. Each of the N_I islands will be represented by an interior Ω_I and a boundary δI. In addition, δC denotes all *coastlines* within the interior of the computational domain, i.e., $\delta C = \delta M + \sum_{i=1}^{N_I} \delta I$; and δO denotes the physical boundary of the multiply-connected *ocean*, i.e., $\delta O = \delta P + \delta C$.

The elliptic problem to be solved may be stated as

$$\mathcal{L}\psi = Z \tag{56}$$

in the region Ω_O, subject to $\psi(\delta O)$ being prescribed. Here, $\mathcal{L}$ is the two-dimensional differential operator corresponding to equation (55). The boundary conditions on solid land boundaries are

$\mathbf{K}^{-1}$ is referred to as the *capacitance matrix*. At each model timestep, the matrix multiplication (64) must be performed to evaluate $\mathbf{w}$. However, it is impractical to store the N_B influence functions ψ_{G_n} and multiply these by $\mathbf{w}$ to get ψ_H at every timestep. Rather, the right hand side of equation (56) is augmented with a vorticity forcing field

$$Z_H = \sum_{n=1}^{N_B} Z_{G_n} w_n \tag{65}$$

consisting of the weights w_n distributed along the boundary δC. Then the solution to

$$\mathcal{L}\psi = Z + Z_H \tag{66}$$

in Ω, subject to boundary conditions on δ, will be precisely $\psi_P + \psi_H$. Equation (66) can be solved directly with **mud2** because the boundary conditions are prescribed only on the perimeter of the computational domain δ.

3.8.2 Island circulation

In §3.8 it was assumed that k_i, the value of ψ on each island, could be specified *a priori*. In general, this is not the case as the transport between the islands will evolve with time in response to the model forcing and is determined by the condition that the surface pressure must be continuous following any closed path around each island.

A consequence of the rigid-lid approximation is that the surface pressure is not carried explicitly in the model. Therefore, its effect on the circulation around each island is inferred by an integration of the depth-averaged momentum balance. The depth-average of equations (45) and (46) may be written

$$\frac{\partial \overline{\mathbf{u}}}{\partial t} = \overline{\mathbf{R}}_{\mathbf{u}} - \frac{1}{\rho_o}\nabla p_s \tag{67}$$

where $\overline{\mathbf{R}}_{\mathbf{u}}$ now represents all other terms in the balance except the gradient of the surface pressure p_s. Around any closed path $\mathbf{s}$

$$\oint \nabla p_s \cdot d\mathbf{s} = 0, \tag{68}$$

which implies, from equation (67)

$$\frac{\partial \Gamma_i}{\partial t} = \oint_i \overline{\mathbf{R}}_{\mathbf{u}} \cdot d\mathbf{s}, \tag{69}$$

where

$$\Gamma_i = \oint_i \overline{\mathbf{u}} \cdot d\mathbf{s} \tag{70}$$

is the circulation following the chosen path around island i. Equation (69) is integrated at each timestep to predict the value of Γ_i around each island at the new time level.

Let $\hat{\psi}$ denote the solution obtained by the capacitance matrix algorithm (66) and $\hat{\Gamma}_i$ the corresponding island circulation values. Substitution of (51) and (52) into (70) gives

$$\hat{\Gamma}_i = \oint_i -\frac{1}{h}\frac{\partial \hat{\psi}}{\partial r} d\mathbf{s} \tag{71}$$

where r is directed inward, normal to the path of integration. The discrepancy between Γ_i and $\hat{\Gamma}_i$ can be amended by adding to $\hat{\psi}$ a weighted sum of island Green's functions ψ_i, defined as the solution to

$$\mathcal{L}\psi_i = 0 \tag{72}$$

in Ω_O, subject to $\psi_i = 1$ on island i and zero on all other islands and the mainland. The ψ_i are closely related to the influence functions and are readily computed by the capacitance matrix method.

3.11 The pressure gradient terms

The pressure gradient terms in equations (19) and (20) are written in the form

$$h\nabla\phi + (1-\sigma)h\frac{\partial}{\partial\sigma}\left(\frac{\phi}{h}\right)\nabla h$$

or

$$h\nabla\phi + (1-\sigma)\left(\frac{gh\rho}{2\rho_o}\right)\nabla h.$$

This is the form traditionally used in sigma-coordinate models to account for the horizontal differences being taken along surfaces of constant σ. This form can be shown to lead to significant errors when $|\nabla h|$ is large (Haney [16]; and Beckmann and Haidvogel [6]).

These errors can be greatly reduced if these terms are calculated to fourth-order accuracy (McCalpin [27]). The fourth-order form for the derivative operator is:

$$\frac{\partial\phi}{\partial\xi}_{i-\frac{1}{2}} = \frac{-\phi_{i+1} + 27\phi_i - 27\phi_{i-1} + \phi_{i-2}}{24\Delta\xi} + O(\Delta\xi^4) \tag{78}$$

and the fourth-order accurate averaging operator is:

$$\overline{\phi}^{\xi} = \frac{1}{16}(-\phi_{i+1} + 9\phi_i + 9\phi_{i-1} - \phi_{i-2}) + O(\Delta\xi^4) \tag{79}$$

Rather than trying to maintain fourth-order accuracy at horizontal boundaries by using one-sided differences, SPEM drops to second-order accuracy at all horizontal boundaries (at the domain edges and near masked land areas).

In both the second- and fourth-order calculations of the pressure gradient, the pressure ϕ is computed by a vertical integration of the density field using equation (24). Prior to the integration, a horizontal average of the density, $\overline{\rho}(z)$, is subtracted from ρ. As discussed by Haney [16] and McCalpin [27], $\overline{\rho}$ does not contribute to the pressure gradient in the analytic equations. However, when numerically computing its contribution to the pressure gradient, the error from this term can be unacceptably large.

3.12 Computation of the surface pressure

In timestepping the model from t to $t+\Delta t$, most of the terms in the momentum equations are computed directly. However, since the SPEM has a rigid lid, the effects of the surface pressure gradient are not immediately known. We bybass calculating the surface pressure directly by calculating the curl of the momentum equations and finding the depth-integrated streamfunction, as described in §3.7. However, once the timestep is complete, it is possible to infer the surface pressure gradient components since then all other terms are known in the momentum balance. The 2-D fields of pressure gradient are computed when needed and are stored in each history record.

3.13 Friction and diffusion

In §2 the diffusive terms were written simply as $\mathcal{D}_u, \mathcal{D}_v, \mathcal{D}_T$, and $\mathcal{D}_S$. Here we describe how these terms are represented in SPEM. In fact, they are the sum of vertical and horizontal terms, where the horizontal term has several options.

3.13.1 Vertical smoothing

The vertical diffusion term is identical for all four equations of motion. In the model coordinates it is represented as:

$$\frac{4}{hmn}\frac{\partial}{\partial\sigma}\left(\nu\frac{\partial\phi}{\partial\sigma}\right). \tag{80}$$

$$F^\sigma = \nu\epsilon\frac{4}{h^2}\frac{\partial\phi}{\partial\sigma} + \underbrace{\underbrace{\left[(1-\sigma)\frac{m}{h}\frac{\partial h}{\partial\xi} - \underbrace{\frac{2S_x}{h}}_{\texttt{MIX\_EN}}\right]}_{\texttt{MIX\_GP}} F^\xi 1}$$
$$+ \underbrace{\underbrace{\left[(1-\sigma)\frac{n}{h}\frac{\partial h}{\partial\eta} - \underbrace{\frac{2S_y}{h}}_{\texttt{MIX\_EN}}\right]}_{\texttt{MIX\_GP}} F^\eta} \tag{87}$$

where

$$S_x = \frac{\frac{\partial\rho}{\partial x}}{\frac{\partial\rho}{\partial z}} = \frac{\left[m\frac{\partial\rho}{\partial\xi} + (1-\sigma)\frac{m}{h}\frac{\partial h}{\partial\xi}\frac{\partial\rho}{\partial\sigma}\right]}{\frac{2}{h}\frac{\partial\rho}{\partial\sigma}}$$

$$S_y = \frac{\frac{\partial\rho}{\partial y}}{\frac{\partial\rho}{\partial z}} = \frac{\left[n\frac{\partial\rho}{\partial\eta} + (1-\sigma)\frac{n}{h}\frac{\partial h}{\partial\eta}\frac{\partial\rho}{\partial\sigma}\right]}{\frac{2}{h}\frac{\partial\rho}{\partial\sigma}}$$

and ϵ is the ratio of vertical to horizontal diffusion coefficients. No flux boundary conditions are easily imposed by setting

$$\begin{aligned} F^\xi &= 0 \qquad \text{at } \xi \text{ walls} \\ F^\eta &= 0 \qquad \text{at } \eta \text{ walls} \\ F^\sigma &= 0 \qquad \text{at } \sigma = \pm 1 \end{aligned}$$

Finally, the flux divergence is calculated and is added to the right-hand-side term for the field being computed:

$$\frac{\partial}{\partial\xi}\left(\frac{hF^\xi}{n}\right) + \frac{\partial}{\partial\eta}\left(\frac{hF^\eta}{m}\right) + \frac{\partial}{\partial\sigma}\left(\frac{hF^\sigma}{mn}\right) \tag{88}$$

These options are currently available for only the T and S equations. There are plans to also provide them for the u and v equations—ask me about the status of this update.

3.14 Convective adjustment

If desired, the density field is checked for occurrences of hydrostatic instability, which the model will then attempt to remove. It searches through the water column using a simple conservative mixing scheme. The variables T, S and optionally u and v are vertically mixed and the new ρ is then calculated from the equation of state.

There are actually two different schemes, depending on the equation of state being used. For the linear equation of state, two vertically adjacent ρ values are compared and then mixed if the shallower one is denser. For the fully nonlinear equation of state, this is not appropriate because it does not account for the effects of compressibility. Instead, the Brunt-Väisälä frequency is computed for the point midway between collocation points using the formula from McDougall [28]. Mixing occurs for negative values of N^2.

4 Details of the SPEM Code

4.1 Main subroutines

A flow chart for the main program is shown in Fig. 5. The boxes refer to subroutines which are described as follows:

bcs Horizontal boundary conditions are imposed upon the u, v, T and S fields. Currently supported options are a doubly-periodic domain, periodic channel, and closed basin with either free-slip or no-slip walls. This routine can be modified by the user to provide open boundaries of your favorite type.

init Does everything that needs to be done to start up the model run. It reads initial parameters and u, v, T, S, and ψ fields from disk or calls **peminit**. It then calculates the remaining initial fields, creates plots of the initial fields, and opens the restart file. The flow chart for **init** is shown in Fig. 6.

omega Calculates the vertical velocity Ω according to equation (77).

prsgrd Calculates the baroclinic pressure gradients, with an entry for calculating surface pressure gradients (§3.11).

rhocal Calculates ρ using the equation of state (§3.10).

srhs Calculates and stores contributions to the right hand side of equation (22), where the advective terms have been moved to the right hand side.

step_init Sets some variables for the next timestep. This could be used to update the wind forcing or the boundary conditions.

tmstp Performs the timestep on T, S and the timestep on the 'baroclinic part' of u and v, via entry points **tstp**, **sstp**, **ustp**, and **vstp**, respectively. The entry **vrtstp** performs the timestep on Z, solves for ψ, and completes the time step on u and v.

trhs Calculates and stores contributions to the right hand side of equation (21), where the advective terms have been moved to the right hand side.

urhs Calculates and stores contributions to the right hand side of equation (19), where all the terms other than $\frac{\partial}{\partial t}(\frac{hu}{mn})$ have been moved to the right hand side.

vrhs Calculates and stores contributions to the right hand side of equation (20), where all the terms other than $\frac{\partial}{\partial t}(\frac{hv}{mn})$ have been moved to the right hand side.

vrtrhs Calculates the R_Z term in equation (54).

4.2 Other subroutines and functions

Initialization

chbset Sets up the Chebyshev polynomial functions.

checkdefs Reports on which C preprocessor variables have been **#defined** and checks their consistency.

getgrid Reads in the curvilinear coordinate arrays as well as f and h from a grid generation program.

histio Reads or writes the history file header.

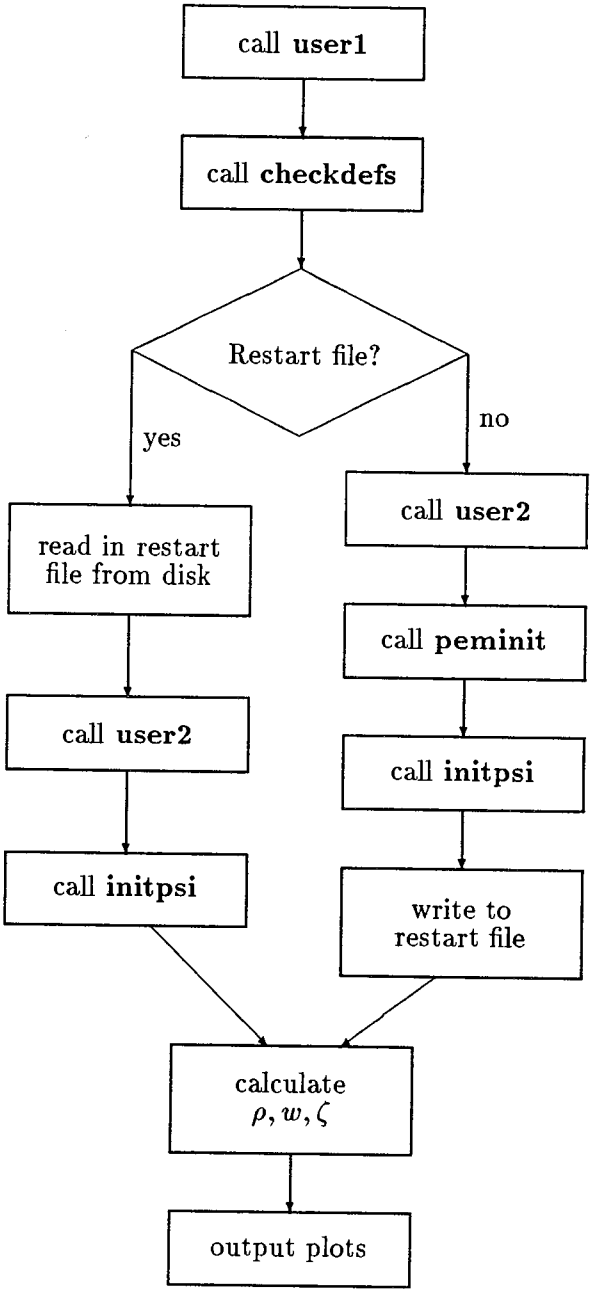

Figure 6: Flow chart for **init**

mixuv Vertically mixes u and v in regions of Richardson number instability (currently a place-holder).

pcorrect Replaces the fourth-order pressure gradient with the second-order value next to the mask boundaries. Note that the fourth-order value is incorrect because it uses points which are interior to the land.

slipbound Applies the horizontal boundary conditions for the u and v equations around masked land areas. These boundary conditions are for either free-slip or no-slip boundaries—more work would have to be done to make this work with partial-slip boundaries.

zeroland Sets the values of a field inside the land masks to some value. This value is usually either zero or a special value for the plots.

Plotting

botplt Plots the bottom topography in slice plots.

cpmpxy Replaces the **conpack** mapping routine so that the slice and slab contour plots come out in the curvilinear coordinate system.

cpshift Makes **conpack** calls in such a way as to emulate our old modified **conrec**. It will optionally shift the contour lines by half a contour interval. See the version 3.0 or 3.2 NCAR graphics manual [7].

fill Fills an array for plotting horizontal sections of the model fields—either a level of constant sigma or the amplitude of a vertical mode.

getxxyy Fills the **xs**, **ys**, **xv**, **yv**, **xb**, and **yb** arrays for **cpmpxy** and **vvumxy**.

label Puts labels on plots. Changes the **set** plotting environment.

peplt Creates slab plots of $u, v, \psi, \rho, T, S, w, \zeta, \vec{v}, \phi$ and calls **pltslcyz** and **pltslcxz** for the slice plots.

ploth Plots bottom topography, its gradient, and a resolution ratio (see Appendix C).

pltperim Draws the outline of the model domain in x, y coordinates, including mask edges.

pltslcxz Plots slices on surfaces of constant η.

pltslcyz Plots slices on surfaces of constant ξ.

slablab Labels the horizontal slab plots.

slabplt Plots a 2-D array as a horizontal slab.

sliceplt Gathers together some of the repeated code for vertical slice plots.

vvcurv Makes vector calls in such a way as to emulate our old modified **velvct**, allowing velocity vectors to be plotted in x-y space instead of in ξ-η space. It requires NCAR Graphics 3.2—otherwise you will still have to use a modified **velvct**, as described in Appendix F.

vvumxy User mapping for the vector calls. Contains the old **fx** and friends if you do not define **NCARG_32**.

windplt Plots the wind-stress vector field.

xyzplot Fills an array for plotting horizontal sections of the model fields on a level of constant z.

yzplot Fills an array for plotting vertical sections of the model fields along constant ξ or η surfaces.

masking

MASKING Define if you want to mask out islands and peninsulas within the domain.

ISLANDS Define if you want the model to compute ψ on any islands.

periodic boundaries

DBPER Define if your domain is doubly-periodic.

PERCHAN Define if your domain is a periodic channel.

PERINTG Define if your periodic channel transport is not prescribed and must be computed by the model.

precision These variables were introduced so that one code could be used for Crays and workstations, all using 64 bit precision. The challenge was in linking to single precision graphics libraries on the workstations.

DOUBLE Define this to add calls to **sngl** within the plotting subroutines.

BIGREAL This is the type of all floating point variables used in the model computations. It *must* be defined to be something, such as **real** or **real*8**. If this is set to **double precision** you should also use a compiler option for extending source lines past 72 characters in width.

SMALLREAL This is the type of all floating point variables used by the NCAR plotting routines. It *must* be defined to be something, such as **real**.

rhsdefs.h These are the **cpp** variables which are used in the right-hand-side subroutines and in **rhocal**. Putting them into a separate include file reduces the number of routines which have to be recompiled after a change.

CLIMAT_TS_MIXH Specifies whether or not to subract T_{clm} from T before computing the horizontal diffusion term. Also applies to the S equation. This can be useful when using horizontal diffusion along σ-surfaces over steep bathymetry.

CLIMAT_TS_MIXV Specifies whether or not to subract T_{clm} from T before computing the vertical diffusion term. Also applies to the S equation. This is useful if you have a vertical temperature gradient which you want to maintain in the presence of vertical diffusion, without including surface heat fluxes.

CLIMAT_TS_RLX Specifies whether or not to relax (nudge) the T and S fields to their climatologies. Nudging is a primitive form of data assimilation. If you have a spatially variable nudging constant (**rdmp**) you can nudge only along a boundary, for instance, in which case it is called a sponge.

CLIMAT_UV_RLX Specifies whether or not to relax (nudge) the u and v fields to their climatologies.

MIX_GP_TS Specifies whether or not to rotate the tensors for horizontal diffusion of T and S to lie along surfaces of constant geopotential (z), rather than surfaces of constant σ.

MIX_GP_UV Specifies whether or not to rotate the tensors for horizontal diffusion of u and v to lie along surfaces of constant geopotential (z). (Avaliable soon).

MIX_EN_TS Specifies whether or not to rotate the tensors for horizontal diffusion of T and S to lie along epineutral surfaces (constant *in situ* density).

MIX_EN_UV Specifies whether or not to rotate the tensors for horizontal diffusion of u and v to lie along epineutral surfaces (constant *in situ* density). (Avaliable soon).

N Number of baroclinic polynomials in the vertical.

islandp Maximum number of elements in any of the masks, used to dimension many of them. (When Fortran 90 becomes common, we will be able to dynamically allocate exactly the right amount of memory for each array.)

lablen The first **lablen** characters in **ident** are used to label the plots.

max_no_isle Maximum number of "free" islands for which the circulation must be computed; currently set to twenty.

n_cm Number of points on the mask boundaries, also the dimension of the capacitance matrix. Called N_B in §3.8.

nflt Number of Lagrangian floats.

nifree Number of islands for which the circulation has to be determined. This should be set to 1 for a periodic channel if **PERINTG** is defined.

nwrk Size of the work space used by **mud2**.

4.5 Common blocks and the variables within them

All dimensional variables are given in MKS units.

/ocean/ The main time-dependent model fields.

- **time** time.
- **u** velocity component in ξ-direction.
- **v** velocity component in η-direction.
- **w** Ω, velocity component in σ-direction.
- **rho** perturbation density (total density $= \rho_o + \rho$).
- **t** temperature T.
- **psi** horizontal transport streamfunction ψ.
- **vrt** vorticity of depth-averaged flow field Z.
- **dwds** vertical derivative of Ω, really $\frac{\partial}{\partial\sigma}\left(\frac{h\Omega}{mn}\right)$ or $\frac{h}{mn}\frac{\partial\Omega}{\partial\sigma}$.
- **rhobar** horizontal average of density, must be a function of z only (see §3.11).

/chb/ Polynomial transformation matrices.

- **pi** $\pi = 3.14159265\ldots$
- **sig** array of sigma values $\sigma(k)$.
- **cp** matrix of orthogonal polynomials $F = P_k(\sigma_n)$ (equation 28). Multiply a mode amplitude vector by **cp** to get the vector of values at the σ_n locations, $b(\sigma_n)$.
- **cf** inverse of the matrix of orthogonal polynomials, F^{-1} (equation 30). Multiply a $b(\sigma_n)$ vector by **cf** to get the mode amplitude vector.
- **cd** derivatives of $P_j(\sigma_i)$ (equation 31).
- **cdz** derivative transform matrix C_{DZ} (equation 32). Multiply a vertical vector by **cdz** to get the vector of derivatives.
- **cint** integral transform matrix C_{INT} (equation 34). Multiply a vertical vector by **cint** to get the vector of integrals, integrating down from the surface.

second index specifies the field according to the key:

$$
\begin{aligned}
1 &= \xi \\
2 &= \eta \\
3 &= \sigma \\
4 &= \tfrac{d\xi}{dt}(\xi,\eta,\sigma) = mu \\
5 &= \tfrac{d\eta}{dt}(\xi,\eta,\sigma) = nv \\
6 &= \Omega(\xi,\eta,\sigma) \\
7 &= T(\xi,\eta,\sigma) \\
8 &= S(\xi,\eta,\sigma) \\
9 &= \rho(\xi,\eta,\sigma)
\end{aligned}
$$

kptime Interval between disk writes of float information.

nfltrrec Number of float restart records to read in from disk during initialization.

/grdpts/ x, y (physical space) grid point locations, used in plotting the model fields.

xp x coordinates of the ψ points.

yp y coordinates of the ψ points.

xr x coordinates of the ρ points.

yr y coordinates of the ρ points.

xmin minimum x value on the ρ grid.

ymin minimum y value on the ρ grid.

xmax maximum x value on the ρ grid.

ymax maximum y value on the ρ grid.

xpmin minimum x value on the ψ grid.

ypmin minimum y value on the ψ grid.

xpmax maximum x value on the ψ grid.

ypmax maximum y value on the ψ grid.

xl length of domain in x-direction, **xl** = **xpmax** − **xpmin**.

el length of domain in y-direction, **el** = **ypmax** − **ypmin**.

/hmn/ Combinations of h, m, and n to save computations at the expense of storage.

mhon_u mh over n at u points.

mhon_r mh over n at ρ points.

mhon_p mh over n at ψ points.

nhom_v nh over m at v points.

nhom_r nh over m at ρ points.

nhom_p nh over m at ψ points.

homn_u h over mn at u points.

homn_v h over mn at v points.

homn_r h over mn at ρ points.

homn_p h over mn at ψ points.

hon_u h over n at u points.

sphie η-component of the surface pressure gradient.

nspr number of steps between calculating the surface pressure gradients.

/plt/ Plotting flags and parameters.

zplot array of specific z levels to be plotted if **zslab** is **true**.

ztop the z-level of the top of the slice plots.

zbot the z-level of the bottom of the slice plots.

nplvl number of levels to be plotted.

npl array of specific σ levels to be plotted, from 0 to **N**, if **zslab** is **false**. If **plot_modes** is **true**, then this is the list of specific modes to be plotted.

nslcyz number of y-z (constant ξ) cross sections to be plotted.

nslyz array of specific ξ-locations of the y-z cross sections.

nslcxz number of x-z (constant η) cross sections to be plotted.

nslxz array of specific η-locations of the x-z cross sections.

Lplt largest value of i in the plots, either **Lm** or **L**, depending on whether the domain is periodic in ξ.

Mplt largest value of j in the plots, either **Mm** or **M**, depending on whether the domain is periodic in η.

plot_u logical switch to turn on plotting of the u field.

plot_v logical switch to turn on plotting of the v field.

plot_w logical switch to turn on plotting of the w field.

plot_rho logical switch to turn on plotting of the ρ field.

plot_psi logical switch to turn on plotting of the ψ field.

plot_s logical switch to turn on plotting of the S field.

plot_t logical switch to turn on plotting of the T field.

plot_vort logical switch to turn on plotting of the Z field.

plot_vec logical switch to turn on vector plotting of the $\vec{v}$ field.

plot_spr logical switch to turn on vector plotting of the surface pressure gradient.

plot_dwds logical switch to turn on plotting of a measure of the horizontal divergence.

plot_curl logical switch to turn on plotting of the vertical component of the curl field.

plot_wind logical switch to turn on vector plotting of the wind stress field.

plot_h logical switch to turn on plotting of the bathymetry and its slope.

plot_igf logical switch to turn on plotting of the island Green's function for each island.

plot_init logical switch to turn on plotting of the initial fields.

plot_modes logical flag for plotting mode amplitudes rather than levels in physical space. This is only used if **zslab** is **false**.

zslab determines whether the level (slab) plots are of planes of constant z or constant σ; **true** for plots at a constant z.

rbarsub logical flag to specify whether $\overline{\rho}$ is subtracted out before ρ is plotted.

/pltloc/ Lowlevel plotting variables.

fl1 left coordinate of first quadrant (upper left) plots.

dt timestep Δt .

cfact0 weighting factor for the right-hand-side arrays.

cfact1 weighting factor for the right-hand-side arrays.

cnb weighting factor for $\mathcal{F}(t - \Delta t)$.

cnc weighting factor for $\mathcal{F}^*(t + \Delta t)$.

dts size of the timestep between **lstp** and **lnew**.

ntmes number of steps in current run.

nrst number of steps between storage of restart fields.

nrrec number of restart records to read from disk, the last is used as the initial conditions.

nplt number of steps between calls to subroutine **peplt** (for plots).

nmix number of steps between vertical convective mixings.

nmixplot the number of mixing events is stored for each horizontal point. This field is periodically plotted and reinitialized to zero. **nmixplot** is the frequency of this plotting.

ncorr number of steps between timestep corrections.

iic timestep counter.

jjc timestep component counter.

lnew one of several time level indices (equal to either 1 or 2) used for the last array index in the model fields. **lnew** points to the current time level.

lold pointer to the previous time level.

lstp pointer to the time level to which the current changes are added.

ldis pointer to the time level used in the dissipative terms (not **lrhs** for numerical stability reasons).

lrhs pointer to the time level used to calculate the right-hand-side terms.

ltrp pointer to the time level of the right-hand-side arrays (**ru**, **rv**, etc.) in which the changes are being added.

lab1 pointer to a time level for the right-hand-side arrays.

lab2 pointer to a time level for the right-hand-side arrays.

/user/ Arrays, variables, and constants defined by the user as being relevant to a specific application.

tanh_time function of time which gets updated every timestep.

/vrtadv/ Needed for first moment conservation of the vertical advection operator.

corrfunc correction function—a second order polynomial which goes to zero at the top and bottom and is used in the first-moment conservation algorithm.

/wrk/ Temporary work/storage arrays.

rt stores right hand side of T equation.

ru stores right hand side of u equation.

rv stores right hand side of v equation.

rz stores right hand side of vorticity equation.

/cmudif/ More information for u viscosity in the η-direction.

sign_udif array of values which are either $+1$ or -1 and specify which direction the land point is from the water point in the **cmux** arrays.

/cmv/ Mask of v points that fall on land.

lmask_v number of v points that fall on land.

mask_v array of pointers to these land points, including those on the boundary.

/cmvx/ Mask for v viscosity in the ξ-direction.

lmask_vx number of v points which are next to a land boundary in the ξ-direction.

mask_vx array of pointers to the boundary $\frac{\partial v}{\partial \xi}$ points, which are at ψ points.

mask_vvel array of pointers to the v boundary points in the water.

/cmvdif/ More information for v viscosity in the ξ-direction.

sign_vdif array of values which are either $+1$ or -1 and specify which direction the land point is from the water point in the **cmvx** arrays.

/cmpint/ Mask of ψ points which fall inside land.

lmask_pint number of ψ points which are completely surrounded by land.

mask_pint array of pointers to these land points.

/cmpsib/ Mask boundary points.

n_cm_chk number of interior mask boundary points, equal to **n_cm** but needed as a variable by the **mask** program.

mask_psib array of pointers to the mask boundary points.

/cmpsibp/ Indices of the separate land chunks in **mask_psib**.

mask_psib_p for each island and peninsula, this contains the beginning and ending indices of that land bit within **mask_psib** and **g_cm**.

/cilowpx/ This and the next several common blocks are used for computing, in turn, the circulation around each "free" island.

ilow_px index in **mask_pox** and **mask_plx** of the first point for each island.

iup_px index in **mask_pox** and **mask_plx** of the last point for each island.

ilow_pe index in **mask_poe** and **mask_ple** of the first point for each island.

iup_pe index in **mask_poe** and **mask_ple** of the last point for each island.

/cmpox/ More circulation arrays.

mask_pox array of pointers to ocean ψ points that are in the $\pm\xi$-direction sections of a path around the islands.

mask_plx array of pointers to land ψ points that are in the $\pm\xi$-direction sections of a path around the islands.

mask_poe array of pointers to ocean ψ points that are in the $\pm\eta$-direction sections of a path around the islands.

mask_he array of pointers to ρ points just above the **mask_phie** points, for arrays dimensioned (0:L, 0:M).

mask_hme array of pointers to ρ points just below the **mask_phie** points, for arrays dimensioned (0:L, 0:M).

/cap/ The capacitance matrix.

g_cm the ψ value on each of the boundary points, set in **bcpsi** on peninsulas and fixed islands.

capinv the capacitance matrix.

/igf/ Island Green's function arrays.

grnfnc Green's function for each island.

cigfinv inverse of the island interaction matrix, which gives the circulation around each island due to unit ψ around another island.

circ circulation there ought to be around each island, consistent with the physics in the model.

circadd circulation which must be added to each island by applying the island Green's function.

/wndstr/ Wind stress arrays.

wndx wind stress in the ξ-direction.

wnde wind stress in the η-direction.

/wml/ Extra wind stress arrays for applying the wind stress as a body force over the upper water column.

wmlwgtu weighting factor multiplied by the ξ-component of wind stress when adding in its contribution.

wmlwgtv weighting factor multiplied by the η-component of wind stress when adding in its contribution.

mldsigu number of collocation points over which the ξ-component of wind stress is added.

mldsigv number of collocation points over which the η-component of wind stress is added.

4.6 Statement functions

These statement functions are defined in various include files. Some, such as **akt**, may be constant in time and expensive to calculate, and may be replaced with arrays.

av2.h Averages to more easily keep track of factors of two.

av2 average of the two arguments.

av4 average of the four arguments.

fld.h Initial fields.

ufld initial u values.

vfld initial v values.

sfld initial S values.

Larry Wall has written a program to take the output of **diff** and automatically apply it to the old version of a file to create the new version. This program is called **patch** and is available from all the gnu archive sites. If the output of **diff** has been saved in the file **spem.patch.20** then **patch** would be used as follows:

```
patch < spem.patch.20
```

As **patch** updates the files, it leaves the original of **file** in **file.orig**. If it gets confused for some reason (if you modified the lines of code **patch** wants to change or I send out a bad patch file) it will create a **file.rej** file. I often check to see if any **.rej** files get created—these can be used to patch **file** by hand and can then be deleted.

4.8 Makefiles

One of the software development tools which comes with the UNIX operating system is called **make**. **make** has many uses, but is most commonly used to keep track of how a large program should be compiled. You provide it with a list of your source files and instructions on how to compile them. It will check the relative ages of the source and object files, only compiling those for which the object file is out of date. It is assumed that **make** will be used to compile SPEM and its related programs. See Oram and Talbott [31] for a description of **make** that is easier to read than the **man** page.

The file in which you provide **make** with its commands is usually called **Makefile**. Several **Makefiles** are provided with SPEM, one for each brand of computer to which I have easy access. These **Makefiles** have become quite complex, but are organized into several sections:

suffix rules These are lines of the form **.F.o:**, followed by a rule telling **make** how to make a file called **foo.o** from **foo.F**. This particular rule is used extensively and comes in two forms, depending on whether or not the compiler will invoke the C preprocessor (**cpp**) for you. If the compiler does not invoke **cpp** then **make** will do so, creating an intermediate **foo.f** file.

macros These are lines of the form **CFT = f77**, which in this case allows you to use one name (**$(CFT)**) for the compiler even though each compiler has a different name. The macros in the **Makefiles** are defined in two separate sections:

machine dependent These macros give the name of the compiler and sensible flags for that compiler, etc.

project dependent These macros depend on the project but not the computer, such as the list of source files used to build the executable.

rules These are lines of the form:

```
ezgrid: ezgrid.o
<TAB>$(CFT) -o ezgrid ezgrid.o
```

where **ezgrid** is the target to be compiled, and **ezgrid** depends on **ezgrid.o**. **make** will first check to ensure that **ezgrid.o** is up to date and then execute the commands on the following lines (which must start with a <TAB> character).

dependencies These lines tell **make** which object files must be rebuilt when an include file is modified. Also, if the C processor is being invoked specifically by the suffix rule for **.F.o**, creating an intermediate **.f** file, then **make** must be told to recreate the **.f** file after its include files are modified. The dependency lines are generated automatically by a **perl**

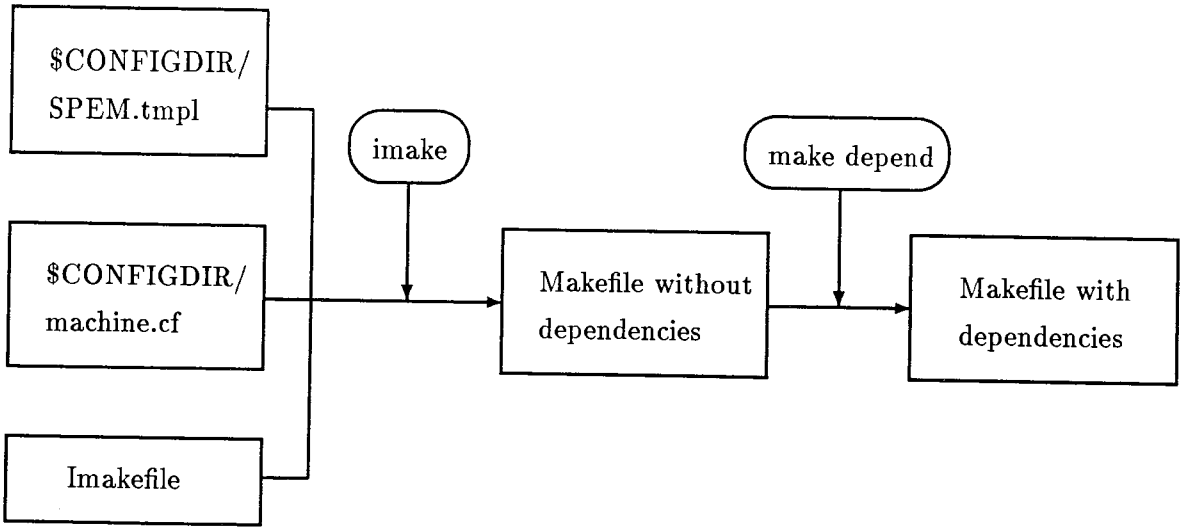

Figure 7: Creating **Makefiles**

4.9.1 Your Makefile

If you are using one of the environments for which **Makefiles** are supplied, you are set, although you may want to check the compile flags. Otherwise, you have the choice of copying and modifying an existing **Makefile** or using **imake** and creating your own configuration file. In either case, you will need to know certain things about your environment such as:

- The name of the Fortran compiler.
- The compiler options you wish to use.
- How to link to the NCAR graphics libraries, if they exist and you wish to use them.
- Whether or not the Fortran compiler will invoke **cpp** and how to tell either the compiler or **make** to do so.
- What file extension the compiler requires.

The biggest changes to the **Makefile** result from the way **cpp** is executed. If your compiler does it for you, start from the Sun files, otherwise start from the IBM RS/6000 files. Sometimes it is better to tell **make** to execute **cpp** even if the compiler will do it. For instance, the old Cray debugger became confused about line numbers if it did not have the intermediate **.f** files to work with.

You will also have to edit the **Makefile** or **Imakefile** if you add source files to SPEM. In this case, you will have to add the new files to the **OBJS** and **SRCS** macros, and make sure that the dependencies are listed correctly for the new files.

4.10 spemlib

SPEM uses several utility routines which don't have to be recompiled each time **L** and **M** are changed, and can therefore be put into a library. Also, some companies provide their own versions of the BLAS routines. You can chose to not compile the **spemlib** BLAS routines (by editing the **spemlib Makefile**) and instead link to the system BLAS library (but see note below).

The routines which are currently in this library are:

comf Needed by **sepx4**, from **fishpak**.

genbun Needed by **sepx4**, from **fishpak**.

5 Support Programs for Initialization

5.1 Grid generation

On startup, SPEM reads a file on unit **igrid** to find the location of the grid points, the bathymetry, and the Coriolis parameter f. This file must be generated before SPEM can be run, either with **ezgrid** or with the programs in **gridpak**.

5.1.1 ezgrid

ezgrid was written to generate a uniform rectangular grid with a simple bathymetry. It has two modes, one for the upwelling example, and one for rectangular basins; the mode is determined by the **UPWELLING** switch in **spemdefs.h**. If **UPWELLING** is not defined then the important parameters are:

L must be consistent with the SPEM **L** dimension.

M must be consistent with the SPEM **M** dimension.

xl basin length in the ξ-direction.

el basin width in the η-direction.

h0 bottom depth.

f0, beta Coriolis parameter with the β-plane approximation, $f = f_o + \beta y$.

igrid grid file unit number, set to 3 by default.

Once these parameters are set to your chosen values, compile and run it:

```
make ezgrid
ezgrid
```

This should create a binary file on unit **igrid**.

5.1.2 gridpak

SPEM has been designed to be used with curvilinear orthogonal grids for boundary-following domains, etc., so there are situations in which you want a more flexible grid-generation program than **ezgrid**. We have been working on a suite of programs called **gridpak**, including **xcoast**, an interactive boundary drawing program. See §1.1 for instructions on obtaining **gridpak** and its documentation.

5.2 Masking

5.2.1 The mask program

SPEM now supports the masking of land areas, for which it requires some new input files. The first of these contains all the arrays used in the horizontal boundary conditions (and plotting) and is generated by the **mask** program, which was written by Jim Mansbridge. Its input is an ASCII file containing the i, j pairs for the boundary ψ points and is easiest to describe with an example. Figure 8 shows a small domain with an isolated island and a promontory adjacent to the boundary. The corresponding **mask.in** file is:

The free islands must be listed before the other types of land mass in the description file. The current version of **mask** treats fixed islands and promontories the same, except for checking to make sure that promontories are attached to the boundary at some point. The fixed island mode is recommended even for promontories, since fixed islands are reported in the output of **mask_psib_p**.

Once **mask** has been told what type of land to expect, the following lines contain the i, j pairs defining that land object. The pairs specify the coordinates of the ψ points on the land boundary, ordered as if you were moving around the masked region with land to your left. It is not necessary to list every ψ point, but only the corners of the "staircase" where the boundary changes direction. Interior islands must start and end at the same point while promontories, including fixed islands, may start and end at the domain boundary.

The process of preparing the input file for **mask** can be tedious, so the **maskedge** program was written to find the input file from an array of zeros and ones. This program is on ahab in the **gridpak** directory. See its **README** file for more information.

Before compiling **mask**, edit **spem3.9.h** to set the parameter **islandp**. If it is too small then **mask** will complain, but there is no way to know beforehand how big it should be. A conservative estimate would be $\mathbf{L} \times \mathbf{M}/2$. When **mask** is finished, it reports on the minimum value for **islandp**.

Once the input file has been prepared it can be used as follows:

```
make mask
mask < mask.in
```

assuming that the input file has been called **mask.in**. The output of **mask** is a binary file on unit 24 (often called **fort.24**) and some text on **stdout** (standard out). The text output tells you what values to use for **islandp**, **n_cm**, and **nifree** in **spem3.9.h**. The text output for the above mask is:

```
# Useful comment, describing this small example mask

 The dimension of the SPEM capacitance matrix must be n_cm =    18
 The number of free islands must be nifree =     1
 The minimum possible value of islandp =    21

Land chunks within g_cm: mask_psib_p(i, j)

Free islands:
Island   1 has   12 points,      starting at   1

Fixed islands:
Island   2 has   6 points,      starting at   13
```

The report on **mask_psib_p** tells you the location of each island within the **g_cm** array. This is useful for specifying the ψ value on fixed islands within **bcpsi**.

Note: **mask** has a **VERBOSE** mode in which it writes out *much* more information to **stdout**. If this is what you really want, uncomment the line with:

```
#define VERBOSE
```

in the **mask.F** file.

There are two extra options in **mask** for models other than SPEM. If you #define **ICE**, **mask** will write another mask file appropriate to our version of the Hibler [20] ice model. Also, if you #define **FREE_SURFACE**, **mask** will not write out the capacitance matrix arrays since they are not needed in models with a free surface.

6 Lagrangian Floats

6.1 General description

SPEM has optional Lagrangian floats for simulating the path taken by water parcels. The float information is stored in an array and the floats are advected at each timestep by the model velocity fields. The float positions and values of T, S, ρ, and $\vec{u}$ at these positions are written out to a float history file at user-specified intervals. Plots and float statistics may be created from this history file. It should be noted that although water parcels have certain conservation properties, these are not necessarily maintained by the model floats.

The particles are advected using a fourth-order Runge-Kutta timestepping scheme. For this, the fluid velocity at the particle position is needed. The particles can be located between the grid points so a horizontal interpolation is done using the neighboring grid points. There is a choice of a bilinear interpolation using the four nearest grid points or a bicubic interpolation using the four by four square of grid points surrounding the particle. The algorithm for the bicubic interpolation is

$$A(x_f, y_f) = \sum_{i=1,4} \sum_{j=1,4} A(x_i, y_j) \frac{\prod_{k=1,4}^{k\neq i}(x_f - x_k) \prod_{l=1,4}^{l\neq j}(y_f - y_l)}{\prod_{k=1,4}^{k\neq i}(x_i - x_k) \prod_{l=1,4}^{l\neq j}(y_j - y_l)} \tag{89}$$

where (x_f, y_f) is the position of the particle and (x_i, y_j) are the positions of the 16 neighboring grid points. The interpolation is done in the horizontal planes, in sigma space, of the collocation points. The vertical interpolation to the float depth is then done using the polynomial modes of the model.

At each timestep, each particle is checked to see if it is inside the model domain. If a particle leaves, it is reinitialized at a fixed location which is specified by the user (at **xnew**, **ynew**, and **znew**) and a message is printed to **stdout** (unit 6). If you find that particles are often leaving the box other than through open boundaries, then your timestep may be too long.

Warning: the particles are not checked for entry into the masked regions. Also, field values from inside the masked regions will be used if the floats approach the masks. This problem is worse for the bicubic interpolation and is worse for the T, S, and ρ fields than for the velocities.

6.1.1 bcw

In the model the vertical velocity array **w** ends one-half grid point to the inside of the edge of the model domain. To allow the same interpolation scheme in all of the velocity fields it was decided to add exterior points to the **w** array. The exterior values are determined in the subroutine **bcw** by a linear extrapolation from the interior values of the array. These exterior values are used only in the particle tracking subroutines.

6.2 To use floats

The changes which need to be made to the code to use floats are as follows:

6.2.1 Makefile

Check the **Makefile** to be sure that **FLTOBJS** is not set to a NULL list.

6.2.2 spemdefs.h

You must

```
#define FLOATS
```

nlayer One possible float release strategy is to put out an array of floats at each of several depths. All floats at one depth would then logically belong together and are put in the same plot in the 'default' plots. The number of these float groupings is given by the parameter **nlayer**. **nlayer** should be set to 1 if a different release strategy is used.

fltplt.F

gridfile For plots without a ρ slab, the program only needs to read in the grid file (unit 3) and the float file (unit 12). If you want ρ slabs you should have a history file available. This may have a header (with **ident**, climatology, etc). If **gridfile** is **true** then **fltplt** will try to read the grid file, otherwise it will read in the header.

fpday Number of float records per day. Calculate it from the model timestep **dt** and the frequency of saving float records **kptime**. That is,

$$\mathbf{fpday} = \frac{86400}{\mathbf{dt} \times \mathbf{kptime}}.$$

Character quality The quality of the characters drawn by **plotchar** is set by a call to **pcseti**. The best is 0, the cheapest is 2.

lwidth Sets the width of the outline drawn by pltperim and the width of the float tracks. Makes no difference on some hardware.

Block data Initially, just make sure there are the proper number of values (**nlayer**) for each limit array. If you want to set the color range on the plots, the limits can be set in the **block data**. To use your limits, type 'n' in response to the question "Should the range of colors be calculated from the chosen floats?". If this is too much trouble, then comment out the whole **block data**.

Colors Each of the five plots (z, w, T, S, ρ) has its own color bar, defined in **tzplt**, **twplt**, **ttplt**, **tsplt** and **trhoplt**. To change a color bar, set the parameter **ncol** in the appropriate subroutine to the number of colors you want, then fill the **rgb** array. (Note that some compilers have a default limit to the number of allowed continuation lines).

The foreground and background colors have been set to black and white, respectively, in the main program. Change the calls to **gscr** if you would like them to be different.

6.3.2 Running fltplt

fltplt will interactively ask you a number of questions, most of which should be obvious. Some which aren't are:

- "Would you like the default plots?" Answering "yes" will tell it to plot all the floats in each group together. For each of the **nlayer** groups, there will be a plot for each of z, w, T, S and ρ. There will be no density slabs and no more questions.
- "Would you like to specify a range of float IDs?" This is so you can ask for floats 121 to 140 without typing in 121, 122, 123, ... It will assume that all floats to be plotted have consecutive ID's.
- "Should the range of colors be calculated from the chosen floats?" If not, the limits set in the blockdata will be used. The limits will be that of the group to which the first float to be plotted belongs.

6.4.3 Example plots

There are many options when running fltplt. To get an overview of what the floats do, let's plot the central float for each group as well as the real drifter tracks. This plot is shown in Fig. 9 and is discussed in detail in Hedstrom et al. [19].

7 Useful Programs for Postprocessing

John Wilkin has provided SPEM with some programs for reading and plotting the standard history files. They include the **spem3.9.h** file, link to SPEM's **histio** and other subroutines, and are therefore easier to maintain than completely separate programs would be.

7.1 readhist

readhist reads the header and restart files from a SPEM run and provides a summary of their contents. It asks you for the unit numbers **ihdr** and **iin**, and reads your response from **stdin.** First, it summarizes the header file and then asks you to specify the i, j, k triplet defining a point in the model. It will then report on that point for each record in the history file. An example of using **readhist** is as follows:

```
Enter header file unit number:
8
Enter history file unit number:
9

SPEM 3.9 - flat-bottom basin
           with islands
 L =   97 M =   65 N =   4
 xl =      1800000.0000000 rdmp =   0.
 pm (0, 0) =      5.3333333333333D-05 dhde (Lm, M) =   0.
 xp (1, 1) =   0. yp (L, M) =      1200000.0000000
 pi =      3.1415926535898 csum (N) =   -0.13333333333333
 iprm =   1  1  1  1  1  3  2  6  6  97  65  1  6  3  117693
111082  1
 fprm =   0.    96.000000000000  0.    64.000000000000
1.00000000000000D-06 0.
 mgopt =   0  2  1  1  1
 xmin =    -9375.00 ymax =     1.20938E+06
 dt =      1080.0000000000
Enter i,j,k for point values
4 4 4
Writing point values at (i,j,k):  4  4  4
 record   1
 time =   0. seconds
          0. days
 psi(i,j)   =   0.
 u(i,j,k,1) =   0.
 v(i,j,k,1) =   0.
 t(i,j,k,1) =      5.0000000000000
 record   2
 time =      172800.00000000 seconds
             2.0000000000000 days
 psi(i,j)   =    -288814.05390403
 u(i,j,k,1) =     7.1815545337800D-03
 v(i,j,k,1) =    -5.0854005679659D-03
 t(i,j,k,1) =     5.0000000000000
 end of file in iput
```

it choose a sensible value. On the other hand, if you know exactly what you want, it is possible to specify the range of values to be contoured. If you have a variable which ranges from 0.0 to 1.0 and you want to have a contour interval of 0.1:

```
8              number contour levels (0 for default)
.1    .9       contour level min, max (reqd if n levels > 0)
```

If it is a color plot it will use ten colors. Eight colors will be used for values between 0.1 and 0.9, plus a color for all values less than 0.1 and a color for all values greater than 0.9. If the highest value in your field is 0.66, some of the colors will not be used. What we often do for the animated sequences is to run **plothist** once, asking for the default contour levels, decide which range we really want, and to run **plothist** again specifying the desired range with a suitable number of contours.

The last line in **plot.in** sets three variables which are only used for color plots:

```
-1 t 8       color scheme, label at side (true), nominal
         no. color regions
```

There are currently four color schemes in the distributed version of **plothist**, chosen with the first of these parameters, where the sign specifies whether black outlines will be drawn around each contour. The color schemes each have 17 colors and are defined in subroutine **defclr** within **cpsfill.F**. More colors may be added by changing **ncolor** everywhere in **cpsfill.F** and by adding more lines to the **rgb** data statements. Watch out for the 19 continuation line limit in the Fortran 77 standard (some compilers have the option of allowing more continuation lines). Other color schemes can also be added to **defclr**.

The other two variables in the above line specify the location of the color bar and the number of color regions. The latter is only used if you asked for the default number of contours above. **conpack** will chose anything from the number you provide (**ncontr**) to $2 \times$ **ncontr** $- 1$. If you choose too large a number it will run out of colors.

One other new feature in **plothist** is the ability to plot coastlines from the **xcoast** data file. For this option you will need to supply the **proj.h** file from **gridpak** so that it knows what map projection to use. Also, it checks the **XCOASTDATA** environment variable in order to locate the coastline file. You can set this with a line such as

```
setenv XCOASTDATA /home/moi/xcoast/xcoastdata
```

in your **.login** file (for **csh** users). Note that the coastline drawing option is quite slow since it does not use the most efficient algorithm.

8 Ice Model Formulation

Hibler [20] has described a model for the simulation of sea ice circulation and thickness. The model has been tested over a seasonal cycle and the results are also shown in that article. This report was derived from the documentation for the sea ice model written by Hibler [21] and describes the model in its current form. In this documentation frequent reference will be made to Hibler [20] which describes the model, and will hereafter referred to as JPO79.

8.1 Model structure

The overall structure consists of two principal components—the momentum equations and the ice continuity equations. The momentum balance includes air and water stress, Coriolis force, internal ice stress, inertial forces and ocean tilt as shown in equations (90) and (91) (from JPO79).

$$M\frac{\partial u}{\partial t} + M\vec{v}\cdot\nabla u = Mfv - Mg\frac{\partial H}{\partial x} + \tau_a^x + \tau_w^x + \mathcal{F}_x \tag{90}$$

$$M\frac{\partial v}{\partial t} + M\vec{v}\cdot\nabla v = -Mfu - Mg\frac{\partial H}{\partial y} + \tau_a^y + \tau_w^y + \mathcal{F}_y \tag{91}$$

Nonlinear boundary layers for both the ocean-ice and air-ice surface stress are used.

$$\vec{\tau}_a = \rho_a C_a |\vec{V}_g|(\vec{V}_g \cos\phi + \mathbf{k}\times\vec{V}_g \sin\phi) \tag{92}$$

$$\vec{\tau}_w = \rho_w C_w |\vec{v}_w - \vec{v}|\,[(\vec{v}_w - \vec{v})\cos\theta + \mathbf{k}\times(\vec{v}_w - \vec{v})\sin\theta] \tag{93}$$

A key component of the momentum balance is the force due to the internal ice stress ($\mathcal{F}_x$ and $\mathcal{F}_y$). This force is based on a constitutive law which relates the ice stress to the strain rate and ice strength (equation (94)). For this model, a viscous-plastic behavior is used. Rigid plastic behavior is approximated in this law by allowing the ice to flow in a plastic manner for normal strain rates and to creep in a linear viscous manner for small strain rates. The treatment of ice as a viscous-plastic fluid was largely motivated by the desire to avoid the complexities associated with elastic-plastic behavior under flow. The stress-strain relationship is given by

$$\sigma_{ij} = 2\eta\epsilon_{ij} + (\zeta - \eta)\epsilon_{kk}\delta_{ij} - \frac{P}{2}\delta_{ij}. \tag{94}$$

The viscous-plastic terms (96) and (97) are found by taking the divergence of the stress tensor:

$$(\mathcal{F}_x, \mathcal{F}_y) = \nabla\cdot\boldsymbol{\sigma} \tag{95}$$

$$\mathcal{F}_x = \frac{\partial}{\partial x}\left[(\eta+\zeta)\frac{\partial u}{\partial x} + (\zeta-\eta)\frac{\partial v}{\partial y} - P/2\right] + \frac{\partial}{\partial y}\left[\eta\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right] \tag{96}$$

$$\mathcal{F}_y = \frac{\partial}{\partial y}\left[(\eta+\zeta)\frac{\partial v}{\partial y} + (\zeta-\eta)\frac{\partial u}{\partial x} - P/2\right] + \frac{\partial}{\partial x}\left[\eta\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right] \tag{97}$$

where the nonlinear viscosities are given by

$$\zeta = \frac{P}{2\left[(\epsilon_{11}^2 + \epsilon_{22}^2)(1 + 1/e^2) + 4e^{-2}\epsilon_{12}^2 + 2\epsilon_{11}\epsilon_{22}(1 - 1/e^2)\right]^{1/2}} \tag{98}$$

$$\eta = \frac{\zeta}{e^2}. \tag{99}$$

$$(P^*, C) = (2.75 \times 10^4, 20) = \text{ice strength parameters}$$
$$(\vec{\tau}_a, \vec{\tau}_w) = \text{air and water stresses}$$
$$(\vec{V}_g, \vec{v}_w) = \text{air and water velocities}$$
$$C_a = \text{nonlinear air drag coefficient}$$
$$C_w = 5.5 \times 10^{-3} = \text{water drag coefficient}$$
$$(\phi, \theta) = (25^o, 25^o) = \text{air and water turning angles}$$
$$(\rho_a, \rho_w) = (1.3\, kg\, m^{-3}, 1000\, kg\, m^{-3}) = \text{air and water densities}$$
$$(\mathcal{F}_x, \mathcal{F}_y) = \text{viscous-plastic terms}$$
$$(S_h, S_A) = \text{thermodynamic terms}$$

and

$$(\mathcal{D}_h, \mathcal{D}_A) = \text{ice diffusion terms.}$$

8.2 Numerical Scheme

The coupled nonlinear equations of the model are treated as an initial value problem using energy conserving finite difference techniques. Under this initial value approach, values of all parameters at all grid points are required to start the integration, and boundary values of velocity are required thereafter. The forcing fields for the model consist of geostrophic wind fields and near-surface ocean currents which are used for both the ocean stress term and to estimate a geostrophic surface tilt. Both inertial and momentum advection terms are retained in the dynamical equations. Because of the very strong ice interaction term, explicit integration of the momentum equations would force timesteps to be extremely small (of the order of a second). The ice thickness equations, on the other hand, can be explicitly integrated over timesteps of several days. To avoid this severe timestep limitation due to the ice interaction, the momentum equations are integrated implicitly. This implicit integration does, however, require a relaxation solution of a set of simultaneous equations at each timestep. It is currently implemented as a line over-relaxation (SLOR) in the x-direction for the u equation and in the y-direction for the v equation.

Using this implicit scheme, the inertial terms will normally be significant for short timesteps (of the order of 1 hour or less) but insignificant for long timesteps. Although not a formal stability requirement, it is wise to choose timesteps that are small compared to the variability of the ice forcing because the nonlinear terms are treated in a semi-implicit manner in the momentum balance. The effect of different timestep magnitudes on the momentum balance is discussed in Appendix B of JPO79. This Appendix also contains some examples which may help the user decide on timestep magnitudes for particular applications. Because of this implicit treatment of the momentum equation, the only formal stability requirement is a Courant-Friedrichs-Lewy condition for the advection terms in the thickness equations:

$$\Delta t \leq \Delta x[2(u^2 + v^2)]^{1/2}$$

A key feature of the numerical scheme is a staggered grid, known as the "Arakawa B-grid", where the velocity is defined at the corner of a grid box and variables such as thickness and viscosity are defined at the center of the grid box, as shown in Fig. 10.

In the time-marching scheme (equations 21–24, JPO79) ice thickness and velocity are temporally staggered. In particular, the thickness characteristics are considered to be defined at a temporal location halfway between the ice velocities. This scheme (formally called the forward-backward procedure) efficiently integrates coupled equations of this kind (e.g. Mesinger and Arakawa [29]).

Because of the strong ice interaction (which in this model is dissipative in nature) the momentum equations are essentially parabolic in form and hence have few numerical instability problems over

The thickness and concentration equations have no-flux boundary conditions imposed along the mask boundaries. The open northern edge has the thickness specified inside the outflow cells, while the original model did an average of the neighboring cells to find appropriate values in the outflow cells.

The primary characteristic of the outflow cells is that the ice strength goes to zero there. The values of P, ζ, and η are all set to zero in outflow cells after their values are computed.

8.4 Thermodynamics

To integrate this model over very long time intervals requires a thermodynamic code specifying ice growth rates as a function of thickness and time of year. The ice is assumed to have a thickness distribution where the ice-covered water contains seven different thicknesses, each of the same area, such that the average thickness over the cell is h. Since the dynamical equations only follow the ice concentration and average thickness, the ice is redistributed into the seven thickness bins at each timestep. This thickness distribution is a simplification of that used in Hibler [22]. The ice thickness affects the heat lost through the ice and also affects the surface temperature since there is assumed to be a balance between heat lost at the surface and heat diffused from the bottom of the ice (which is at the freezing temperature). The heat budget is computed for each of the seven thicknesses as well as the open water fraction.

In the present code, growth rates are computed from the ice thickness, air temperature, wind speed, and a radiation balance. There is also a simple ocean mixed layer to represent the ocean's contribution to the heat budget. The thermal balance used is described in Hibler [22] and is similar to that of Manabe et al. [26] and Parkinson and Washington [32]. It uses a constant ice conductivity and a linear vertical temperature profile as shown in Fig. 11. The effects of snow cover are approximated by changing the surface albedo of the ice to "snow" values for below freezing surface temperatures.

The basic surface heat balance equation in the presence of an ice cover is (where fluxes into the surface are considered positive):

$$(1-\alpha)\mathcal{F}_S + \mathcal{F}_L + \mathcal{D}_1|V_g|(T_a - T_o) +$$

$$\mathcal{D}_2|V_g|[q_a(T_a) - q_s(T_o)] - \mathcal{D}_3 T_o^4 + \frac{K}{H}(T_w - T_o) = 0. \tag{105}$$

The variables used here are:

α = surface albedo

T_o = surface temperature of the ice

T_a = air temperature

T_w = water temperature

K = ice conductivity

H = ice thickness (set to at least .05 m)

q_a = specific humidity of the air

q_s = specific humidity at the ice surface

$(\mathcal{F}_S, \mathcal{F}_L)$ = incoming shortwave and longwave radiation

$(\mathcal{D}_1, \mathcal{D}_2)$ = bulk sensible and latent heat transfer coefficients

$\mathcal{D}_3$ = Stefan-Boltzmann constant times the surface emissivity.

where A_1 is a constant and $A_2(T_o)$ is a nonlinear function of T_o. Using a Newton-Raphson procedure equation 106 is solved for T_o. If T_o is warmer than 273.16 K, it is set equal to 273.16 K. The growth rate is then calculated from

$$f_B(H) = \frac{-[A_1 + A_2(T_o) + F_o]}{Q_1} \tag{107}$$

where

Q_1 = volumetric heat of fusion of ice

$$Q_1 = 3.02 \times 10^8 \; J\, m^{-3}\, K^{-1}$$

F_o = constant oceanic heat flux into the mixed layer from the deep ocean

$$F_o = 0.$$

In the case of open water, equation 107 is also used to estimate the growth rate, with the modification that the albedo is set equal to 0.1, T_o is set equal to T_w, and the latent heat transfer coefficient $\mathcal{D}_2$ has a different value (see above). In any given grid cell the water temperature is calculated from the mixed layer equations, and in the presence of ice is always equal to 271.2 K.

9 Description of the Ice Model and the Coupling

9.1 Model structure

The flow of the main program is shown in Fig. 12. The overall structure essentially consists of two components—the momentum equations and the ice continuity equations. The momentum balance includes air and water stresses, Coriolis force, internal ice stress, inertial forces and ocean tilt (equations (1) and (2)) and is computed in **form** and **relax_new**. The ice continuity consists of the advection and diffusion (both done in **advect**), and thermodynamics (**growth**).

The timestepping of the ice dynamics is semi-implicit. An initial estimate of the future state is obtained by first linearizing the non-linear terms about the prior time level. A subsequent correction step is performed to produce more accurate estimates of the fields. **form** is called to compute the nonlinear viscosities for each estimate of the ice velocities. However, I believe that **form** would only need to be called twice per timestep.

advect Advect and diffuse a scalar quantity such as ice thickness or concentration. The diffusion coefficient is built in and depends on the grid spacing variable **deltax**.

form Set up some of the arrays used by **relax_new**, including the ice viscosities, ice strength, wind and water stresses, and ice mass. These are quantities which change from one timestep to the next, but not for each iteration of the relaxation scheme.

growth Modify the ice thickness and concentration to reflect ice melt and growth. Then limit the ice concentration to lie between **a22** and 1.0.

ice_init Initialize the ice model, either by reading a history file or by setting the initial values. If a restart file is not read, **ice_init** calls **relax_new** to get an estimate of the ice velocities which is in balance with the wind and water stresses.

ice4 This is the main subroutine for the ice model. It calls **form**, **relax_new**, **advect** and **growth**.

relax_new Solve the coupled u and v equations using an implicit timestep. The solution technique uses multiple line relaxations in the x-direction for the u equation followed by multiple line relaxations in the y-direction for the v equation.

9.1.1 Other subroutines

The other subroutines in the ice model include:

budget Compute a heat budget for either open water or the ice. The change in ice thickness is computed and depends on the air temperature, the wind speed, and the radiation balance. In the case of ice cover, the surface temperature of the ice is also found (iteratively). **budget** is called by **groatb**.

diffus_ice Diffuse the ice thickness and concentration. This is called by **advect**.

groatb Finds the temperature of the ocean mixed layer (from **yneg**) and the wind speed, then calls **budget** for the open ocean plus once each for seven layers of ice. Then the total change in ice volume is computed from the sum of these eight budgets. Called by **growth**.

icecor Find the Coriolis parameter (**fcor**) at ice velocity points. Currently uses the map projection information, but could just as well average the SPEM f values.

icein Read ice history files.

iceout Write ice history files.

maskinit Initialize the mask arrays, including the outflow cell array.

mean Find the mean ice volume of non-outflow cells surrounding outflow cells. Not currently being used.

plast Finds the viscosities **zeta** and **eta**.

rnegt Finds **yneg**, the amount of extra heat left over after all the ice is melted.

xmaxm Finds the maximum absolute value of an array.

9.2 Coupling strategy

The flow chart for the coupled ice-ocean model is shown in Fig. 13. Several functions were introduced to input the wind and thermal forcing and to couple between the ice and ocean subroutines. The subroutine which is the main program and which calls the ice and ocean subroutines is called **driver**.

There are extra variables for passing information between the ice and ocean components of the model. For instance, the ocean model has the ocean velocity (**u**), but the ice model also needs the ocean velocity for computing the stresses (**gwatx**). The ocean velocities are defined on an Arakawa C-grid while the ice model needs the information on an Arakawa B-grid. The velocities are horizontally averaged when **gwatx** is being found from **u** to account for the difference in grid locations. The thermodynamic variables are all co-located on the grid, but the array indices range from 0 to **L** in SPEM while they range from 1 to **NX1** in the ice code.

from_ice Pass the wind stress and salt flux information to the ocean model. Also sets the surface ocean temperatures to be at least as warm as the freezing point.

intept First of several routines for horizontally interpolating the forcing fields to the model grid points, all derived from **intep**, all assuming that the model grid points are uniformly spaced. This one interpolates the thermal fields to h points.

intept0 Set up the interpolation arrays for the thermal field interpolations.

intepw Interpolate the wind forcing to B-grid u points.

intepw0 Set up the interpolation arrays for the wind field interpolations.

spem_hflux Find the heat flux between the ocean and the atmosphere.

spem_stress Find the wind force at ice u points by calling **intepw** to do the horizontal interpolation. Also do the (linear) time interpolation between consecutive wind records and the rotation from geostrophic winds to surface winds.

spem_thermo Find the thermal fields at h points by calling **intept** to do the horizontal interpolation. Also do the (linear) time interpolation between consecutive records.

to_ice Pass variables to the ice from the ocean, including surface velocities (at $\mathbf{N}-1$), surface salinity and "mixed-layer" depth.

to_spem Pass the ice concentration and thickness to the ocean model.

wdrag Find the momentum stress on the water from the ice and the air.

9.3 Model Variables

The model has array dimension parameters:

NX The number of grid points in the x-direction, equivalent to **L** in SPEM.

NY The number of grid points in the y-direction, equivalent to **M** in SPEM.

NX1 NX + 1.

NY1 NY + 1.

MX The number of x grid points in the wind forcing field.

MY The number of y grid points in the wind forcing field.

MX1 MX + 1, x dimension of the thermal forcing fields.

MY1 MY + 1, y dimension of the thermal forcing fields.

The model variables include:

Main variables These variables each have three time levels, specified by the third index. A value of 1 represents the current time and a value of 2 represents the previous timestep. The third time level of **uice** and **vice** is used in the timestepping algorithm while the third index of **heff** and **area** is used as work space for the diffusion and the computation of **yneg**.

uice The x-component of the ice velocity.

vice The y-component of the ice velocity.

heff Mean ice thickness per unit area.

area Ice concentration, or fraction of area covered by thick ice.

Momentum variables

amass The ice mass, **heff** times the ice density.

fcor The Coriolis parameter.

ffcor The Coriolis parameter times the ice mass.

eta Nonlinear shear viscosity.

zeta Nonlinear bulk viscosity.

press The ice pressure or strength.

forcex The terms in the u momentum equation which do not depend on the current velocity.

forcey The terms in the y momentum equation which do not depend on the current velocity.

uicec Extra time level of u, for timestepping the momentum equations.

vicec Extra time level of v, for timestepping the momentum equations.

Wind/water stress variables

dairn The air stress $\rho_a C_a |\vec{V}_g|$.

dwatn The water stress $\rho_w C_w |\vec{u}_i - \vec{u}_w|$.

drags Symmetric component of water stress, **dwatn** · **coswat**.

iicice **iic**, timestep counter.

it1 1, starting value for timestep counter.

it2 **ntmes**, ending value for timestep counter.

icode **rstflag**, flag telling whether or not this run started by reading a restart file, 0 for no restart, 1 for restart.

Scalar variables

time_ice Ice time, in hours.

deltax Grid spacing in the x-direction, currently 20 km.

deltay Grid spacing in the y-direction, currently 20 km.

deltat Ice timestep.

delt Ocean timestep, **dt**.

deltt Ice timestep, either 1 or 2 times **deltat** (like **dts**).

icount Day counter for the monthly diagnostics.

idia Day counter for the surface forcing fields.

IDAY1 Day counter.

IDAY2 Day counter.

TIME1 Time counter for surface stress.

TIME2 Time counter for surface stress.

TIME3 Time counter for surface thermal fields.

TIME4 Time counter for surface thermal fields.

itice Ice timestep counter.

idtice Number of ice timesteps per day.

icnt Some sort of initialization flag for **spem_stress**.

icnt2 Some sort of initialization flag for **spem_thermo**.

lad Determines the type of timestep in advect, 1 for leapfrog, 2 for Euler backward. Currently set to 2.

iin_ice Input unit number for ice history files.

iout_ice Output unit number for ice history files.

maskio Unit number for the ice mask file (containing **heffm**).

Parameter variables

ho Thickness which divides thick ice from thin ice, 1 m in the current model.

a22 Minimum allowed ice concentration, currently 15%.

error Error tolerance for relaxation scheme, currently 2×10^{-4}.

Monthly mean variables

mean_uice Mean ice u velocity.

mean_vice Mean ice v velocity.

heff_mean Mean effective ice thickness.

area_mean Mean ice concentration.

10 Configuring the Coupled Model for a Specific Application

This section describes the parts of the model for which the user is responsible when configuring it for a given application. Section 10.1 describes the process in a generic fashion for the ocean. Section 10.2 lists some of the concerns for using the ice model in an alternate application. Finally, §10.3 steps through the application of the coupled model to the Western Arctic.

10.1 Configuring the ocean model

10.1.1 spemdefs.h and rhsdefs.h

For each of the **cpp** variables described in §4.3, decide whether or not you want it to be defined. Each defined variable should have a line of the form:

```
#define SOME_VAR
```

Note that any undefined variable need not be mentioned, but we leave placeholders for them in **spemdefs.h** and **rhsdefs.h** as a reminder that they are meaningful. These placeholders can be in any of the following forms:

```
#undef SOME_VAR1
c #define SOME_VAR2
! #define SOME_VAR3
/* #define SOME_VAR4 */
```

The last form is a C comment which will be deleted by the C preprocessor. However, **cpp** will not delete these C comments when invoked with the **-P** option, so one of the other forms is recommended.

New **cpp** variables can be defined in **spemdefs.h** and then used in the code with an **#ifdef** statement. This is a simple way to keep track of pieces that you add for your application. For instance, my Arctic boundary conditions are labelled with **ARC_OPEN**:

```
#ifdef ARC_OPEN
c  Arctic pseudo-open boundary conditions
                :
                :
#endif  /* ARC_OPEN */
```

If it becomes necessary to update to a newer version of SPEM it is simple to find the parts of the code which belong to the Arctic version and copy them to the new SPEM.

10.1.2 Model domain

One of the first things the user must decide is how many grid points to use, and can be afforded. There are three parameters which specify the grid size:

- **L** Number of finite-difference points in ξ.
- **M** Number of finite-difference points in η.
- **N** Number of baroclinic vertical modes.

Since the model uses the **mud2** elliptic solver you are not free to pick arbitrary values for **L** and **M**. For a non-periodic domain the following relations must be satisfied:

$$\mathbf{L} = \mathbf{nxl} \times 2^{(\mathrm{nxstep}-1)} + 1 \tag{108}$$

$$\mathbf{M} = \mathbf{nyl} \times 2^{(\mathrm{nystep}-1)} + 1 \tag{109}$$

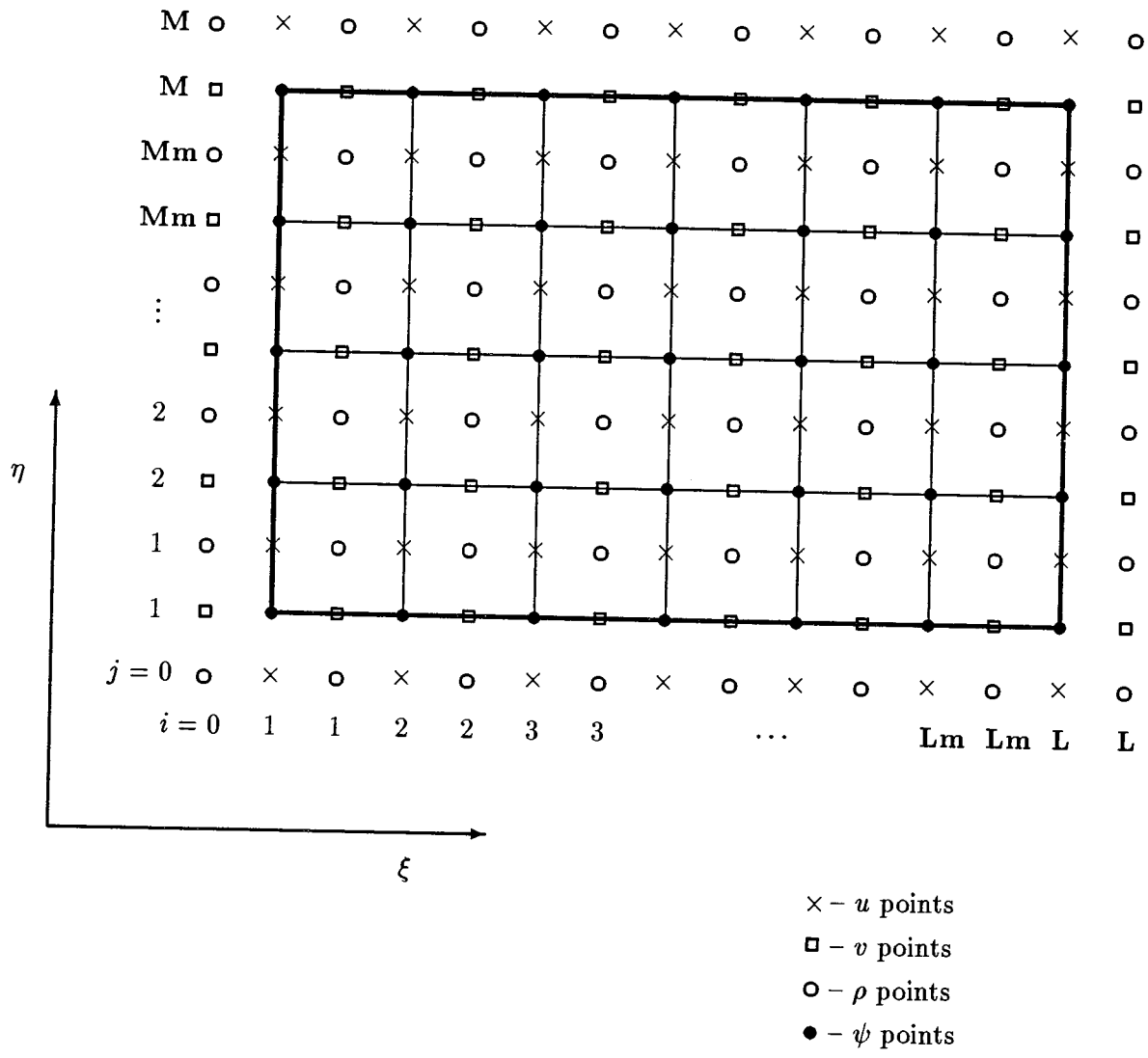

Figure 14: The whole grid

(b) Climatology

One way to force the model is via a nudging to the climatology. This was used in the Coastal Transition Zone simulations described by Haidvogel et al. [14]. For that problem, nudging on the u and T equations was used, making sure that the two climatology fields were in approximate geostrophic balance. This nudging served to drive a baroclinic along-channel flow.

(c) Barotropic mode

The barotropic mode can be forced by the boundary conditions on the ψ equation. This was used in some seamount simulations where **bcpsi** was configured to drive an oscillating flow past a seamount (see Haidvogel et al. [13]).

10.1.9 user1

As many of the user choices as possible are put into the subroutines **user1** and **user2**. The variables in **user1** are not written to the restart file and can safely be initialized before the restart file is read. In fact, **nrrec** determines whether or not the restart file will be read and must be initialized at the start of the model run. The variables in **user1** are described in §4.5 and include the I/O unit numbers, the number of timesteps to take, which plots to make, and which terms in equations 19–22 to compute.

10.1.10 user2

user2 contains variables which can be updated after the restart file is read. One variable which is set in **user2** is the timestep **dt**. For simple wave-like problems the timestep is limited by the CFL condition, such that $c\frac{\Delta t}{\Delta x} < 1$, where c is the phase speed of the wave. However, when horizontal and vertical diffusion are included, the stability issue becomes more complicated and the timestep constraint becomes one of the considerations when deciding on suitable values of the viscous and diffusion coefficients (especially the vertical component).

The choice of diffusive terms is set in **user1**, but the horizontal diffusion coefficients are set in **user2**. Several factors to consider when choosing these values are:

spindown time The spindown time on wavenumber k is $\frac{1}{k^2\nu}$ for the Laplacian operator and $\frac{1}{k^4\nu}$ for the biharmonic operator. The smallest wavenumber corresponds to the length $2\Delta x$ and is $k = \frac{\pi}{\Delta x}$, leading to

$$\Delta t < t_{damp} = \frac{\Delta x^2}{\pi^2\nu} \quad \text{or} \quad \frac{\Delta x^4}{\pi^4\nu} \tag{112}$$

This time should be short enough to damp out the numerical noise which is being generated but long enough on the larger scales to retain the features you are interested in. This time should also be resolved by the model timestep.

boundary layer thickness The western boundary layer has a thickness proportional to

$$\Delta x < L_{BL} = \left(\frac{\nu}{\beta}\right)^{\frac{1}{3}} \quad \text{and} \quad \left(\frac{\nu}{\beta}\right)^{\frac{1}{5}} \tag{113}$$

for the Laplacian and biharmonic viscosity, respectively. We have found that the model typically requires the boundary layer to be resolved with at least one grid cell. This leads to coarse grids requiring large values of ν.

user2 also reads **ident** from unit 5 (**stdin**). It is a string which is used to identify the experiment—the first **lablen** characters of **ident** are used to label the plots. The free-slip/no-slip variable **slip_flag** is initialized and the mask arrays are read if they are needed.

10.2.4 icebcs.h

The ice thickness and concentration are specified along the northern boundary after every timestep. This is done in both **ice_init** and in **growth** so the code was put into the **icebcs.h** include file. Other exterior values for both thickness and concentration are set to zero by the land masks.

10.2.5 Forcing fields

The forcing fields are read in by subroutines which are hard-coded for the Arctic application. The subroutines for interpolating these fields to the model grid are also specific to the Arctic datasets.

10.3 Arctic example

The Arctic application is a relatively high resolution (20 km) computation done in the Western Arctic with masking of part of Alaska, Siberia, and some islands, as shown in Fig. 15. It is a basin with a prescribed inflow at the Bering Strait and outflow across the northern boundary. We will force the system with wind and thermal forcing and include a coupled ice model as described in §9.2.

10.3.1 spemdefs.h

We have chosen the following C preprocessor variables in **spemdefs.h**:

```
#define S_CALC
#undef PERINTG
#undef PERCHAN
#undef DBPER
#define MASKING
#define ISLANDS
#define CLIMATOLOGY
#define FOURTHORDER
```

Here we have declared that we want a basin, masking, climatology, and the fourth-order pressure gradient. We have also added a new C preprocessor variable to place around our boundary conditions:

```
#define ARC_OPEN
```

The boundary conditions will be described below.

10.3.2 Model domain

The number of gridpoints was chosen to be a compromise between wanting as fine a resolution as possible and wanting the job to finish in a finite amount of time. It also must require less than 128 megabytes of storage to be run on some of our computers. Values for **L**, **M**, and **N** are:

$$\mathbf{L} = \mathbf{NX} = 129$$
$$\mathbf{M} = \mathbf{NY} = 129$$
$$\mathbf{N} = 7.$$

10.3.3 gridpak

For this geometry **ezgrid** is not an option and so we used the grid-generation programs collectively called **gridpak**. A flow chart of these programs and the order in which they are used is shown in Fig. 16. These programs will be described in the **gridpak** manual. The only program which is used in a non-standard way is **bathsuds**, which is where whatever required "cheating" is done to the bathymetry field. In this case, we have clipped the bathymetry to lie between 50 and 500 meters deep. We have also filled in a continental shelf along the Canadian edge and along the open northern edge and smoothed the resulting field. A conformal conic map projection was chosen for the grid generation. Once the grid was created, the x, y values were converted to a Lambert equal area projection to correspond to the projection used for the forcing fields. The grid is not quite orthogonal in this projection, but the metric factors m and n were computed from the orthogonal grid in spherical coordinates.

10.3.4 Initial conditions and the equation of state

We would like the initial conditions to be motionless fluid with a realistic stratification. The statement functions **ufld**, **vfld**, and **psifld** are all set to zero in **fld.h**.

The Levitus T and S fields were horizontally averaged and approximated by the functions:

$$\begin{aligned} t_o &= .6 - 5.2e^{z/300} + 4.0e^{z/600} \\ s_o &= 34.9 - 4.7e^{z/200} \end{aligned}$$

These approximate fields are used to initialize the model fields and the T and S climatologies and are shown in Fig. 17 and 18. The **rhobar** field is initialized from the initial density field rather than from a statement function.

The ice initial conditions are set in **ice_init** and are also chosen to correspond to no flow. The ice concentration is set at 1.0, representing 100% ice cover. The ice thickness is set to 1.65 m, the thickness of 1.5 m of water.

10.3.5 Boundary conditions

The non-periodic options have been chosen in **spemdefs.h**. We would like to have a flow of .8 Sverdrups in through the Bering Strait and an equivalent flow out through the northern boundary. This flow is specified by the boundary conditions on ψ in **bcpsi**. ψ is set to zero along the Canadian and Alaskan coasts, linearly decreasing across the Bering Strait, constant along the Siberian coast, and increasing across the remaining northern boundary. Since there is a promontory attached to the Siberian coast, it is also necessary to set the corresponding elements of **g_cm**. The ψ boundary values are ramped up over the first couple weeks with the function:

$$\frac{1}{2}\left[1 + \tanh\left(\frac{t - t_o}{\alpha}\right)\right]$$

The distribution of flow through the northern boundary is not well known and is estimated from the results of a coupled ice-ocean simulation done by Hibler and Bryan [23]. The model reads in a series of values from zero to one, one for each ψ point on the boundary. These values were produced by the **xpots** program which can be found in the **pub/util** directory on ahab.rutgers.edu.

The **bcs** subroutine must also be modified to provide the appropriate barotropic flow normal to the boundaries. The baroclinic modes still perceive the boundaries as closed. All of the changes to **bcpsi** and **bcs** are surrounded by C preprocessor conditionals:

```
#ifdef ARC_OPEN
        :
#endif
```

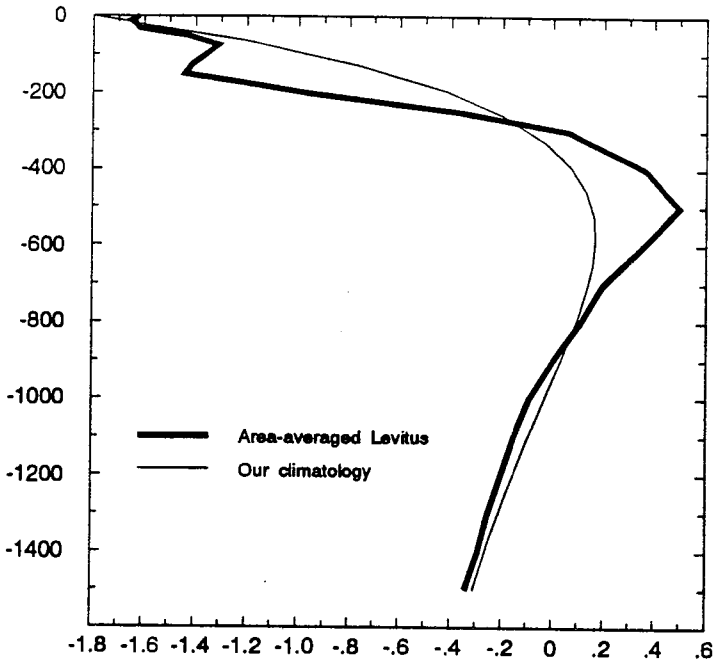

Figure 17: The area-averaged Levitus temperature profile and the temperature climatology

The entire northern boundary is set to contain outflow cells in **maskinit**. The ice will be set to zero there, increasing the ice flow out of the domain. The ice thickness and concentration along that boundary are also reset to the initial values at each timestep in **icebcs.h**.

10.3.6 Forcing

The wind and thermal forcing fields are read from disk on a coarse regular grid and then interpolated and rotated to the model coordinate system. The subroutines **intept**, **intepw**, **spem_stress**, and **spem_thermo** take care of these operations. The daily forcing fields are linearly interpolated in time at each timestep.

The subroutine **windinit** is not used to initialize the forcing fields, but is used to set up the arrays for using a body force over a variable number of vertical nodes. New arrays were added for treating the body force on T and S in the same fashion as on u and v. **zmldep** is set to -51 meters to insure that the forcing over the shelf is barotropic (for model stability).

The other forcing which we have been forced to use in the coupled model is a nudging of the T and S fields back to climatology. A nudging timescale of 100 days is sufficient to get rid of an instability in Kotsebue Sound. The instability seems to be due to the ice being blown out of the sound and new ice forming there, leading to local changes in the temperature and salinity fields which the model is unable to handle.

10.3.7 user1

The terms to be used in the equations of motion are specified by the flags in **user1**. We want everything turned on except biharmonic viscosity and vertical viscosity so **uflags** and **vflags** 2, 4, 5, 6, 7 and 8 are **true** while 1 and 3 are **false**. In the temperature and salinity equations, flags 2, 4 and 5 are **true**.

A Model Timestep

Numerical timestepping uses a discrete approximation to

$$\frac{\partial u(t)}{\partial t} = \mathcal{F}(t) \tag{114}$$

where $\mathcal{F}(t)$ represents all the right-hand-side terms. The simplest approximation is the Euler timestep:

$$\frac{u(t+\Delta t) - u(t)}{\Delta t} = \mathcal{F}(t) \tag{115}$$

where you predict the next u value based only on the current fields. This method is accurate to first order in Δt; however, it is unconditionally unstable with respect to advection.

The leapfrog timestep is accurate to $O(\Delta t^2)$:

$$\frac{u(t+\Delta t) - u(t-\Delta t)}{2\Delta t} = \mathcal{F}(t). \tag{116}$$

This timestep is more accurate, but it is unconditionally unstable with respect to diffusion. The diffusion terms must be lagged ($\mathcal{F}(t-\Delta t)$), in which case they are first-order accurate. Also, the even and odd timesteps tend to diverge in a computational mode. This computational mode can be damped by taking an occasional correction step. SPEM has a choice of correction steps, both of which use a leapfrog step to obtain an initial guess of $u(t+\Delta t)$. We will call the right-hand-side terms calculated from this initial guess $\mathcal{F}^*(t+\Delta t)$.

The first choice of correction steps is a trapezoidal step:

$$\frac{u(t+\Delta t) - u(t)}{\Delta t} = \frac{1}{2}\left[\mathcal{F}(t) + \mathcal{F}^*(t+\Delta t)\right] \tag{117}$$

and the second choice is a third-order Adams-Bashforth:

$$\frac{u(t+\Delta t) - u(t)}{\Delta t} = \mathcal{F}(t) - \frac{1}{4}\mathcal{F}(t-\Delta t) + \frac{1}{4}\mathcal{F}^*(t+\Delta t). \tag{118}$$

SPEM uses a leapfrog timestep with a correction every **ncorr** timesteps. The type of correction step is determined by the value of **stpflag** (**true** for a third-order Adams-Bashforth).

The model carries two time levels for many of the physical fields, differentiated by the last index in the arrays. The initial fields are only read in for one time level; an Euler timestep is used for the first step to get things going. A correction step is made if you are using trapezoidal corrections (you would need to obtain $\mathcal{F}(-\Delta t)$ to make a third-order correction).

A restart record is written every **nrst** timesteps. If you were to start the model from one of these restart records you would start with an Euler step. Therefore, to give repeatable results, the model treats the first step after each record is written as if it were the first step.

B Chebyshev Polynomial Basis Functions

As described in §3.1, the $P_k(\sigma)$ expansion functions used in the vertical spectral method can be chosen somewhat arbitrarily. Here, we review the form of the matrix operators when the $P_k(\sigma)$ are a modified form of the Chebyshev polynomials of the first kind ($T_k(\sigma)$; see, e.g., [1] and [15]). The $T_k(\sigma)$ are defined as

$$T_k(\sigma) = \cos[k\cos^{-1}(\sigma)], \qquad -1 \le \sigma \le 1$$

where $-\pi \le cos^{-1}(\sigma) \le 0$. We then set

$$P_k(\sigma) = \begin{cases} T_0(\sigma) & k = 0 \\ T_k(\sigma) & k \ge 1, \quad k \text{ odd} \\ T_k(\sigma) + \frac{1}{k^2-1} & k \ge 2, \quad k \text{ even.} \end{cases} \tag{119}$$

which then conform to the required integral property that only $P_o(\sigma)$ have a non-zero vertical integral. The collocation points σ_n are located at the extrema of the highest order polynomial $P_N(\sigma)$; hence,

$$\sigma_n = \cos[\pi(n-N)/N], \qquad 0 \le n \le N. \tag{120}$$

The first eight modified Chebyshev polynomials and the collocation levels for $N = 8$ are shown in Fig. 19.

The matrix F (see equation (28)) may now be evaluated from (119) and (120). Substituting (119) into (31) gives the elements of matrix R:

$$R_{nk} = \frac{\partial P_k(\sigma)}{\partial \sigma} = \frac{k\sin(k\theta_n)}{\sin\theta_n} \tag{121}$$

where $\theta_n = \cos^{-1}(\sigma_n)$. Similarly, (119) and (33) give the elements of matrix S:

$$S_{nk} = \int_{\sigma_n}^{1} P_k(\sigma)d\sigma = \begin{cases} (1-\sigma_n) & k = 0 \\ \frac{1}{2}(1-\sigma_n^2) & k = 1 \\ \left[\frac{\cos(k+1)\sigma}{2(k+1)} - \frac{\cos(k-1)\sigma}{2(k-1)}\right]_{\cos^{-1}(\sigma_n)}^{0} & k > 1, \text{odd} \\ \left[\frac{\cos(k+1)\sigma}{2(k+1)} - \frac{\cos(k-1)\sigma}{2(k-1)}\right]_{\cos^{-1}(\sigma_n)}^{0} & \\ \quad +(1-\sigma_n)\frac{1}{k^2-1} & k > 1, \text{even.} \end{cases} \tag{122}$$

Finally, the differentiating and integrating operators C_{DZ} and C_{INT}, respectively, are computed by first inverting F, and then calculating the products $C_{DZ} = RF^{-1}$ and $C_{INT} = SF^{-1}$.

C Bathymetric Steepness

SPEM and other sigma coordinate models are known to have some trouble in the presence of steep bathymetry, especially if the grid spacing is not fine enough. Unfortunately, we do not know, in general, the relationship between steep bathymetry, stratification, horizontal and vertical resolution, and whether the model will run stably. However, two plots have been added to **ploth** to guide the user in determining which areas of the domain are likely to lead to bathymetric problems. The first of these plots is simply the bottom slope, or $|\nabla h|$. The maximum for the whole domain is written in the corner of the plot.

If you find that the model is ill-behaved and you suspect a problem with steep bathymetry, then there are two ways to improve on the situation. One is to filter the bathymetry until it is no longer "too steep". The other is to increase the horizontal grid spacing until you resolve the slope "well enough". A measure of how well the slope has to be resolved is obtained from the "slope parameter" r, defined in Beckmann and Haidvogel [6]:

$$r = \frac{\Delta h}{2\overline{h}} = \frac{h_i - h_{i-1}}{h_i + h_{i-1}}.$$

r ranges from -1 to 1, although we actually plot $|r|$ and print the maximum value (r_{max}) in the corner of the plot. Beckmann (personal communication) recommends satisfying the relationship

$$r_{max}\,\mathrm{N} < 1 \tag{123}$$

as long as all of the vertical modes are needed to represent the vertical structure in the model. This advice is based on his experience with the second-order pressure gradient—his experience with the fourth-order scheme is limited, although he expects (123) to still be an appropriate relationship. By and large, the fourth-order pressure gradient option should be used with variable bathymetry.

McCalpin [27] has investigated the pressure gradient error as a function of the Burger number defined as

$$S = \frac{N_o H_o}{f_o L}$$

where H_o is the deep-water depth and L is the length scale of the bathymetry variations (seamount width). If ρ' is the size of the perturbation density $(\rho - \overline{\rho})$ then it is best to keep

$$\begin{aligned} S\,\rho' &< 0.25 \quad \text{second order} \\ S\,\rho' &< 1.0 \quad \text{fourth order} \end{aligned}$$

The results also improve as the horizontal resolution is increased.

Warning: John Wilkin designed a filter for reducing the slope parameter while trying to leave as many small-r features as possible. When trying this out, it was discovered that the remaining $2\Delta x$ signal in the bathymetry generated lots of $2\Delta x$ noise in the flow fields. Having small values of r does not guarantee that the model will produce smooth results.

D The C preprocessor

The C preprocessor, **cpp**, is a standalone program which comes with most C compilers. On many UNIX systems it is not in the default path, but in **/lib** or in **/usr/lib**. If you do not have a C preprocessor then there are several versions available via anonymous ftp. For instance, **ftp.uu.net** has two in the **/published/oreilly/nutshell/imake** directory—I have built and used the one from Der Mouse on a Cray. One also comes with **gcc**, the **gnu** C compiler. If you build this compiler, **cpp** will have a path such as

```
/usr/local/lib/gcc-lib/sparc/2.4.5/cpp
```

where **sparc** is the architecture and **2.4.5** is the version number.

This Appendix describes the C preprocessor as used in SPEM with the Fortran language. A more complete description can is given by Kernighan and Ritchie [25]. More practical advice on using **cpp** is given by Hazard [17].

D.1 File inclusion

Placing common blocks in smaller files, which are then included in each subroutine, is the easiest way to make sure that the common blocks are declared consistently. Many Fortran compilers support an include statement where the compiler replaces the line

```
include 'file.h'
```

with the contents of **file.h**; **file.h** is assumed to be in the current directory. The C preprocessor has an equivalent include statement:

```
#include "file.h"
```

We are using the C preprocessor style of include because many of the SPEM include files are not pure Fortran and must be processed by **cpp**.

D.2 Macro substitution

A macro definition has the form

```
#define    name          replacement text
```

where **name** would be replaced with "replacement text" throughout the rest of the file. This is used in SPEM as a semi-portable way to get 64-bit precision:

```
#define  BIGREAL     real*8
```

It is also possible to define macros with arguments, as in

```
#define av2(a1,a2)        (.5 * ((a1) + (a2)))
```

although this is riskier than the equivalent statement function

```
av2(a1,a2) = .5 * (a1 + a2)
```

The statement function is preferable because it allows the compiler to do type checking and because you don't have to be as careful about using enough parentheses.

The third form of macro has no replacement text at all:

```
#define  PLOTS
```

In this case, **PLOTS** will evaluate to **true** in the conditional tests described below.

E perl scripts for Fortran

perl is a computer language, invented by Larry Wall, for manipulating text and other useful things. It is fully described in Wall and Schwartz [38], while a more tutorial approach is given by Schwartz [35]. **perl** itself is available from your nearest **gnu** archive site.

I have several **perl** scripts which I find useful when working with Fortran programs, and are available from ahab.rutgers.edu, as described in §1.1. It is not necessary to know **perl** to use these scripts, but it must be installed on your system. To use these scripts, simply place them somewhere in your path and make sure that they are executable:

```
chmod 755 relabel
```

(on a UNIX machine).

The following scripts modify your source code and usually work on the style of Fortran in SPEM, but have been known to do the wrong thing. Some of the scripts become confused when part of an **if** statement is inside an **#ifdef** clause as is done in **main.F**. It is also extremely dangerous to run **relabel** on an arithmetic **if** (as in **invmtx**).

Do not delete your original code before checking the new code.

E.1 redo

This program reformats **do** loops and was written to convert

```
      do 10 i=1,20
10       sum = sum+i
```

to

```
      do 10 i=1,20
         sum = sum+i
10    continue
```

The **-E** option tells it to use **enddo** instead of **continue** as in

```
      do  i=1,20
         sum = sum+i
      enddo
```

redo is used as follows:

```
redo < file.F > file.new
```

redo was written so that **findent** would work on SPEM.

E.2 findent

findent will indent your Fortran code two spaces for **do** loops and **if** statements. It will not correct lines which extend beyond 72 characters, but will print out a warning for each one. It assumes that each **do** loop ends with a **continue** or **enddo**. **findent** is used as follows:

```
findent < file.F > file.new
```

E.6 sfmakedepend

We started using the C preprocessor with SPEM version 3.5 and **fmakedepend** no longer produced the appropriate dependency information. **sfmakedepend** was created to rectify this situation and is used by the **Makefiles** described in §4.8. It searches source files with names ending in **.F** for C style includes such as

```
#include "commons.h"
```

and adds the corresponding dependencies to the **Makefile**. It has several options, including **-s**, required for Fortran compilers which will not invoke the C preprocessor for you. In this case the above dependency line would become

```
file.o: commons.h
file.f: commons.h
```

letting **make** know that the C preprocessor must be rerun on **file.F** whenever **commons.h** is updated.

The other options are for compilers which not only do not execute the C preprocessor, but require your source files to have a different extension. For instance, the NAG compiler assumes that files with the **.f90** extension will use the new Fortran 90 free format where statements may start in column 1. In this case, **make** executes **cpp** and creates a file with the proper extension

```
cpp -P < file.F > file.f90
```

and the dependencies become

```
file.o: commons.h
file.f90: commons.h
```

F Modifications to old velvct

Before NCAR graphics version 3.2, a number of changes had to be made to the NCAR graphics [8] routine **velvct** in order for the vector plots to work in SPEM's curvilinear coordinates. This is no longer necessary if you have the new NCAR package, but is still retained as an option when **NCARG_32** is not defined.

First we will describe the changes to **velvct** and then show the output of

```
diff -c velvct.dist velvct.F
```

where the **-c** option tells **diff** to include some context lines around the changes.

The most important change allows the plots to be made in the curvilinear coordinates instead of rectangular coordinates. This is done by commenting out the statement functions

```
FX(X,Y) = X
FY(X,Y) = Y
```

and

```
MXF(XX,YY,UU,VV,SFXX,SFYY,MXX,MYY) = MXX + IFIX(SFXX*UU)
MYF(XX,YY,UU,VV,SFXX,SFYY,MXX,MYY) = MYY + IFIX(SFYY*VV)
```

and providing the functions **fx**, **fy**, **mxf**, and **myf**.

Another change to **velvct** allows the vectors to protrude outside of the model boundary. Add the lines

```
call gqclip(IER,ICLP,IAR)
call gsclip(0)
```

before the line

```
C DRAW THE VECTORS
```

This will save the current clipping state and disable the GKS clipping. The two lines

```
CALL GQCLIP(IER,ICLP,IAR)
CALL GSCLIP(0)
```

a few lines down should be commented out so that the saved clipping state is not overwritten.

Note that turning clipping off may not be a good idea. If you use the zoom option in **plothist** for the vector plots, vectors will be drawn all over the screen. You can always opt to leave this part of **velvct** alone, and never get vectors drawn outside of the bounding box defined by your call to **set**.

F.1 Context Diffs for velvct

What follows is a Unix context difference file. It starts with a header specifying which files are being compared. The lines which differ are shown with a few surrounding lines for context and are marked with an exclamation point. New lines are marked with a plus sign while deleted lines are marked with a minus sign. The lines from the distributed **velvct.f** are shown first, followed by the lines from the modified version.

```
diff -c velvct.dist velvct.F
*** velvct.dist Fri Jan 26 09:24:17 1990
--- velvct.F Mon Mar 19 09:52:58 1990
***************
```

```
C     TURN OFF CLIPPING SO ARROW CAN BE DRAWN
C
! c   CALL GQCLIP(IER,ICLP,IAR)
! c   CALL GSCLIP(0)
      CALL DRWVEC (28768,608,28768+LEN,608,LABEL,10)
C
C     RESTORE CLIPPING
```

Literature Cited

[1] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions.* Number 55 in Applied Mathematics Series. National Bureau of Standards, 1972.

[2] J. Adams. *MUDPACK: multigrid software for linear elliptic partial differential equations, version 3.0.* National Center for Atmospheric Research, Boulder, Colorado, March 1991. Scientific Computing Division User Doc.

[3] J. R. Adams, R. Garcia, B. Gross, J. Hack, D. Haidvogel, and V. Pizzo. Applications of multigrid software in the atmospheric sciences. *Mon. Wea. Rev.*, 120:1447–1458, 1992.

[4] A. Arakawa and V. R. Lamb. *Methods of computational physics*, volume 17, pages 174–265. Academic Press, 1977.

[5] G. K. Batchelor. *An introduction to fluid dynamics.* Cambridge University Press, 1967.

[6] A. Beckmann and D. B. Haidvogel. Numerical simulation of flow around a tall, isolated seamount. part i: Problem formulation and model accuracy. *J. Phys. Oceanogr.*, 23:1736–1753, 1993.

[7] F. Clare and D. Kennison. *NCAR graphics guide to new utilities, version 3.00.* National Center for Atmospheric Research, Boulder, Colorado, October 1989.

[8] F. Clare, D. Kennison, and R. Lackman. *NCAR graphics user's guide, version 2.0.* National Center for Atmospheric Research, Boulder, Colorado, August 1987.

[9] K. Dowd. *High Performance Computing.* O'Reilly and Associates, Inc., Sebastopol, CA, 1993.

[10] P. DuBois. *Software Portability with imake.* O'Reilly & Associates, Inc., Sebastopol, CA, 1993.

[11] N. G. Freeman, A. M. Hale, and M. B. Danard. A modified sigma equations' approach to the numerical modeling of great lake hydrodynamics. *J. Geophys. Res.*, 77(6):1050–1060, 1972.

[12] D. Haidvogel, J. Wilkin, and R. Young. A semi-spectral primitive equation ocean circulation model using vertical sigma and orthogonal curvilinear horizontal coordinates. *J. Comp. Phys.*, 94:151–184, 1991.

[13] D. B. Haidvogel, A. Beckmann, D. C. Chapman, and R.-Q. Lin. Numerical simulation of flow around a tall, isolated seamount. part ii: Resonant generation of trapped waves. *J. Phys. Oceanogr.*, 1993. accepted for publication.

[14] D. B. Haidvogel, A. Beckmann, and K. S. Hedstrom. Dynamical simulations of filament formation and evolution in the coastal transition zone. *J. Geophys. Res.*, 96:15017–15040, 1991.

[15] D. B. Haidvogel and T. Zang. The accurate solution of poisson's equation by expansion in chebyshev polynomials. *J. Comp. Phys.*, 30(2):167–180, 1979.

[16] R. L. Haney. On the pressure gradient force over steep topography in sigma coordinate ocean models. *J. Phys. Oceanogr.*, 21:610–619, 1991.

[17] W. P. Hazard. Using cpp to aid portability. *Computer Language*, 8(11):49–54, 1991.

[36] Ralph Shapiro. Smoothing, filtering, and boundary effects. *Rev. Geophys. Space Phys.*, 8(2):359–387, 1970.

[37] A. S. Thorndike, D. A. Rothrock, G. A. Maykut, and R. Colony. The thickness distribution of sea ice. *J. Geophys. Res.*, 80(33):4501–4513, 1975.

[38] L. Wall and R. L. Schwartz. *Programming perl.* O'Reilly and Associates, Inc., Sebastopol, CA, 1991. the camel book.

[39] J. Wilkin and K. Hedstrom. User's manual for an orthogonal curvilinear grid-generation package. Institute for Naval Oceanography, 1991.

[40] J. L. Wilkin, J. Mansbridge, and K. S. Hedstrom. An application of the capacitance matrix method to accomodate masked land areas and island circulations in a primitive equation ocean model. *Int. J. Num. Meth. Fl.*, 1994. submitted.