DRAFT User's Manual for an S-Coordinate Primitive Equation Ocean Circulation Model (SCRUM) Version 3.0

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

November 11, 1997

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

Acknowledgments

The SPEM model had a free-surface version written by Dale Haidvogel and myself, which was never adequately well-behaved. We asked Tony Song to try to fix it and Tony made many changes, including the introduction of the vertical s-coordinate. John Wilkin therefore named it SCRUM, the S-Coordinate Rutgers University Model. Hernan Arango has since made even more changes, cleaning up the numerics and the code, writing new NetCDF I/O routines, etc. Bob Chant has also contributed to the SCRUM effort, providing the Smolarkiewicz advection scheme. Bernard Barnier and Anne-Marie Treguier convinced us to try the vertical finite difference approach which has proven to be numerically much more stable than either the original spectral scheme or the finite elements that Tony introduced. John Wilkin, Aike Beckmann and Dale Haidvogel provided the rotated mixing tensors for the horizontal viscosity/diffusion. Bill Large provided us with the code for their planetary boundary layer option and Scott Durski put it into SPEM. I owe all these people and the rest of the SPEM/SCRUM community, especially Hernan Arango for making his notes available and for the comments describing all the subroutines and variables in the model.

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.

The **Prism** debugger from Thinking Machines has been invaluable.

Development and testing of the SPEM model has been funded by 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).

Development and testing of the SCRUM model has been funded by the Minerals Management Service (14-35-01-96-CT030818) and the Office of Naval Research (N00014-93-1-0758, N00014-95-1-0457, and N00014-93-1-0197).

This is Contribution #97-10 of the Institute of Marine and Coastal Sciences, Rutgers University.

Abstract

The S-Coordinate Rutgers University Model (SCRUM), authored by Dr. Hernan Arango et al. of the Institute of Marine and Coastal Sciences at Rutgers University, is one approach to regional and basin-scale ocean modeling. This user's manual for SCRUM describes the model equations and algorithms, as well as additional user configurations necessary for specific applications.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This user's manual for the S-Coordinate Rutgers University Model (SCRUM) describes the model equations and algorithms, as well as additional user configurations necessary for specific applications. Some initial tests of SCRUM are described in Song and Haidvogel [33], while others will be described in Haidvogel and Beckmann [12].

The principle attributes of the model are as follows:

General

- Primitive equations with potential temperature, salinity, and an equation of state.
- Hydrostatic and Boussinesq approximations.
- Optional Smolarkiewicz advection scheme on tracers (potential temperature, salinity, etc.).

Horizontal

- Orthogonal-curvilinear coordinates.
- Arakawa C grid.
- Closed basin, periodic, prescribed, radiation, and reduced-physics open boundary conditions.
- Masking of land areas.

Vertical

- s (terrain-following) coordinate.
- Free surface.
- Tridiagonal solve with implicit treatment of vertical viscosity and diffusivity.

Mixing options

- Horizontal Laplacian and biharmonic viscosity and diffusion along constant s, z or *in situ* density surfaces.
- Horizontal free-slip or no-slip boundaries.

- Vertical harmonic viscosity and diffusion with a spatially variable coefficient, with options to compute the coefficients with Large et al. [18], Mellor-Yamada [22], or Pacanowski-Philander [24] mixing schemes.

Implementation

- Dimensional in MKS units.
- Fortran 77 and common extensions.
- Separate CM Fortran version which ought to be easily modified for Fortran 95 compilers.
- Runs under UNIX, requires the C preprocessor.
- All I/O is done in NetCDF [27], requires the NetCDF library.
- Optimized for vector (Cray) computers.
- Pre- and post-processing graphics package available which uses the NCAR graphics libraries.

Chapters 2 and 3 describe the model physics and numerical techniques and are an update of the description in Song and Haidvogel [33]. Chapter 4 lists the model subroutines, functions and variables. Chapter 5 describes the support programs which are needed to provide SCRUM with data files. As distributed, SCRUM is ready to run with a number of example problems. The process of configuring SCRUM for a particular application is described in Chapter 6, including a discussion of a few example applications. Finally, Chapter 7 describes Hernan Arango's plotting programs **cnt**, **ccnt**, **sec**, and **csec**.

1.1 Acquiring the SCRUM code

The version of the model described in this document is available over Internet via the file transfer protocol (ftp). The user is ftp or anonymous with the password being your electronic mail address. The files are on ahab.rutgers.edu (IP number 128.6.142.5). When connected, you will be in the ftp directory. The directory structure is shown in Fig. 1.1. Everyone has write permission in the pub/incoming directory. To get to the SCRUM source code, type 'cd pub/scrum/tars' and then 'get scrum3_f77.tar.gz'. It might be more convenient to access these files through our web site:

```
http://marine.rutgers.edu/po/index.html
```

The UNIX convention is that a filename ending in .gz has been compressed with the gnu **gzip** utility. The corresponding **gunzip** also comes with it. Likewise, a .tar ending designates a file created with the Unix **tar** (tape archive) utility. The steps in unpacking these files are:

```
% gunzip gridpak.tar.gz
% tar xvf gridpak.tar
```

or

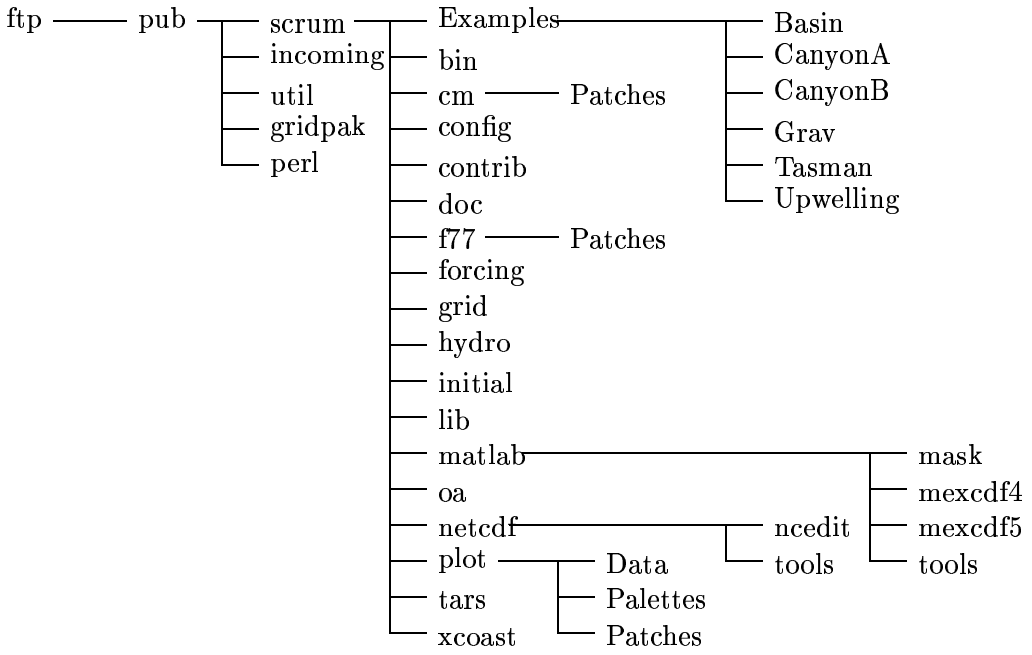

Figure 1.1: Tree structure of anonymous ftp directory

```
% zcat gridpak.tar.gz | tar xvf -
```

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

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

Hernan G. Arango
Institute of Marine and Coastal Sciences
P.O. Box 231
New Brunswick, NJ 08903
(908)-932-3704

Internet: arango@ahab.rutgers.edu

1.2 The SCRUM email list

We maintain an electronic mailing list of SCRUM users. If you would like to be added (or taken off) the easiest thing to do is to send email to majordomo@imcs.rutgers.edu with a message containing the word "help". Other possible messages include "subscribe scrum" and "unsubscribe scrum", both with the obvious meanings. Hernan Arango, Dale Haidvogel and I use the list to send out announcements of new model versions, patches to old versions, and news of SCRUM meetings. There are even occasional announcements of job opportunities. The email list is a mail alias on imcs.rutgers.edu such that all mail sent

to scrum@imcs.rutgers.edu will go to everyone on the list. We strongly recommend that you subscribe to this list if you use SCRUM (and learn to use **patch**!).

1.3 Future plans

Our group is continuing to explore new directions in ocean modeling, but SCRUM 3.0 is meant to be a relatively stable model. Our plans include:

- We have a 2-dimensional model which uses spectral finite elements in the horizontal [15]. The corresponding 3-dimensional model has been written and is being tested on more and more realistic problems. Ask Mohamed Iskandarani (mohamed@ahab.rutgers.edu) for more information.
- Coupling to Paul Budgell's versions of the Hibler sea-ice model. This coupled model exists and is being tested. Please ask Dale Haidvogel or myself for more information.
- Improvements to gridpak, time permitting. The wish list is growing faster than progress is being made.
- Lagrangian floats from SPEM. This is just a matter of time.
- Multi-threaded parallel version of SCRUM for shared memory machines such as those produced by SGI and Cray.

1.4 Warnings and bugs

SCRUM is not a large program by some standards, but it is still complex enough to require some effort to use effectively. Section 6.1 attempts to describe what the user is responsible for—please read it carefully.

More specific things to be wary of include:

- It is recommended that you use 64 bits of precision rather than 32 bits.
- The code must be run through the C preprocessor before it is compiled. This can occasionally be dangerous, especially with the newer ANSI C versions of **cpp**. Potential problems are listed in Appendix D.
- I have started declaring all variables as being of type **BIGREAL** and then defining them to be **real*8** (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.

- The SCRUM grid generation software was originally developed for use with SPEM. Through the years there have been several file formats for the SPEM grid file. Make sure that your grid generation software is recent enough to create a NetCDF grid file as opposed to a binary **fort.3**. The unformatted binary files created portability problems with different architectures and with single vs. double precision.

- The vertical s 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 value of the stretching parameter, **THETA_S**, is between zero and five. It is also desirable to have the depth variations be well resolved by the horizontal grid. For realistic problems we often fail to resolve the bathymetric slopes and we then resort to bathymetric smoothing. This is something of an art and we do not yet know how much smoothing is required.

Chapter 2

Model Formulation

2.1 Equations of motion

The primitive equations in Cartesian coordinates can be written:

$$\frac{\partial u}{\partial t} + \vec{v} \cdot \nabla u - fv = -\frac{\partial \phi}{\partial x} + \mathcal{F}_u + \mathcal{D}_u \tag{2.1}$$

$$\frac{\partial v}{\partial t} + \vec{v} \cdot \nabla v + fu = -\frac{\partial \phi}{\partial y} + \mathcal{F}_v + \mathcal{D}_v \tag{2.2}$$

$$\frac{\partial T}{\partial t} + \vec{v} \cdot \nabla T = \mathcal{F}_T + \mathcal{D}_T \tag{2.3}$$

$$\frac{\partial S}{\partial t} + \vec{v} \cdot \nabla S = \mathcal{F}_S + \mathcal{D}_S \tag{2.4}$$

$$\rho = \rho(T, S, P) \tag{2.5}$$

$$\frac{\partial \phi}{\partial z} = \frac{-\rho g}{\rho_o} \tag{2.6}$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0. \tag{2.7}$$

The variables are shown in Table 2.1. Equations (2.1) and (2.2) express the momentum balance in the x- and y-directions, respectively. The time evolution of the potential temperature and salinity fields, $T(x, y, z, t)$ and $S(x, y, z, t)$, are governed by the advective-diffusive equations (2.3) and (2.4). The equation of state is given by equation (2.5). 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 (2.6). Under the hydrostatic approximation, it is further assumed that the vertical pressure gradient balances the buoyancy force. Lastly, equation (2.7) 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 §3.9.

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

Table 2.1: The variables used in the description of the ocean model

2.2 Vertical boundary conditions

The vertical boundary conditions can be prescribed as follows:

$$
\begin{aligned}
\text{top } (z = \zeta(x,y,t)) \qquad & \nu \tfrac{\partial u}{\partial z} = \tau_s^x(x,y,t) \\
& \nu \tfrac{\partial v}{\partial z} = \tau_s^y(x,y,t) \\
& \kappa_T \tfrac{\partial T}{\partial z} = \tfrac{Q_T}{\rho_o c_P} + \tfrac{1}{\rho_o c_P} \tfrac{dQ_T}{dT}(T - T_{\text{ref}}) \\
& \kappa_S \tfrac{\partial S}{\partial z} = \tfrac{(E-P)S}{\rho_o} \\
& w = \tfrac{\partial \zeta}{\partial t} \\
\text{and bottom } (z = -h(x,y)) \qquad & \nu \tfrac{\partial u}{\partial z} = \tau_b^x(x,y,t) \\
& \nu \tfrac{\partial v}{\partial z} = \tau_b^y(x,y,t) \\
& \kappa_T \tfrac{\partial T}{\partial z} = 0 \\
& \kappa_S \tfrac{\partial S}{\partial z} = 0 \\
& -w + \vec{v} \cdot \nabla h = 0.
\end{aligned}
$$

The surface boundary condition variables are defined in Table 2.2. Since Q_T is a strong

Variable	Description
τ_s^x, τ_s^y	surface wind stress
Q_T	surface heat flux
$E - P$	evaporation minus precipition
T_{ref}	surface reference temperature

Table 2.2: The variables used in the vertical boundary conditions for the ocean model

function of the surface temperature, it is also prudent to include a correction term for the change in Q as the surface temperature drifts away from the reference temperature that

was used in computing Q_T. On the variable bottom, $z = -h(x, y)$, the horizontal velocity components are constrained to accommodate a prescribed bottom stress which is a sum of linear and quadratic terms:

$$\tau_b^x = (\gamma_1 + \gamma_2\sqrt{u^2 + v^2})u$$
$$\tau_b^y = (\gamma_1 + \gamma_2\sqrt{u^2 + v^2})v$$

The vertical heat and salt flux may also be prescribed at the bottom, although they are usually set to zero.

2.3 Horizontal boundary conditions

As distributed, the model can easily be configured for a periodic channel, a doubly periodic domain, or a closed basin. Code is also included for open boundaries which may or may not work for your particular application. Appropriate boundary conditions are provided for u, v, T, S, and ζ. At every timestep the subroutines **bcs2d** and **bcs3d** are called to fill in the necessary boundary values.

The model domain is logically rectangular, but it is possible to mask out land areas on the boundary and in the interior. Boundary conditions on these masked regions are straightforward, with a choice of no-slip or free-slip walls.

If biharmonic friction is used, a higher order boundary condition must also be provided. The model currently has this built into the code where the biharmonic terms are calculated. The high order boundary conditions used are $\frac{\partial}{\partial x}\left(\frac{h\nu}{mn}\frac{\partial^2 u}{\partial x^2}\right) = 0$ on the eastern and western boundaries and $\frac{\partial}{\partial y}\left(\frac{h\nu}{mn}\frac{\partial^2 u}{\partial y^2}\right) = 0$ on the northern and southern boundaries. The boundary conditions for v, T, and S are similar. These boundary conditions were chosen because they preserve the property of no gain or loss of volume-integrated momentum, temperature, or salt.

2.4 s (stretched vertical) coordinate system

From the point of view of the computational model, it is highly convenient to introduce a stretched vertical coordinate system which essentially "flattens out" the variable bottom at $z = -h(x, y)$. Such "s" coordinate systems have long been used, with slight appropriate modification, in both meteorology and oceanography (e.g., Phillips [25] and Freeman et al. [8]). To proceed, we make the coordinate transformation:

$$\hat{x} = x$$
$$\hat{y} = y$$
$$s = s(x, y, z)$$
$$z = z(x, y, s)$$

and

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

See Appendix B for the form of s used here. In the stretched system, the vertical coordinate s spans the range $-1 \leq s \leq 0$; we are therefore left with level upper ($s = 0$) and lower ($s = -1$) bounding surfaces. The chain rules for this transformation are:

$$\left(\frac{\partial}{\partial x}\right)_z = \left(\frac{\partial}{\partial x}\right)_s - \left(\frac{1}{H_z}\right)\left(\frac{\partial z}{\partial x}\right)_s \frac{\partial}{\partial s}$$

$$\left(\frac{\partial}{\partial y}\right)_z = \left(\frac{\partial}{\partial y}\right)_s - \left(\frac{1}{H_z}\right)\left(\frac{\partial z}{\partial y}\right)_s \frac{\partial}{\partial s}$$

$$\frac{\partial}{\partial z} = \left(\frac{\partial s}{\partial z}\right)\frac{\partial}{\partial s} = \frac{1}{H_z}\frac{\partial}{\partial s}$$

where

$$H_z \equiv \frac{\partial z}{\partial s}$$

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

$$\frac{\partial u}{\partial t} - fv + \vec{v}\cdot\nabla u = -\frac{\partial \phi}{\partial x} - \left(\frac{g\rho}{\rho_o}\right)\frac{\partial z}{\partial x} - g\frac{\partial \zeta}{\partial x} + \mathcal{F}_u + \mathcal{D}_u \tag{2.8}$$

$$\frac{\partial v}{\partial t} + fu + \vec{v}\cdot\nabla v = -\frac{\partial \phi}{\partial y} - \left(\frac{g\rho}{\rho_o}\right)\frac{\partial z}{\partial y} - g\frac{\partial \zeta}{\partial y} + \mathcal{F}_v + \mathcal{D}_v \tag{2.9}$$

$$\frac{\partial T}{\partial t} + \vec{v}\cdot\nabla T = \mathcal{F}_T + \mathcal{D}_T \tag{2.10}$$

$$\frac{\partial S}{\partial t} + \vec{v}\cdot\nabla S = \mathcal{F}_S + \mathcal{D}_S \tag{2.11}$$

$$\rho = \rho(T, S, P) \tag{2.12}$$

$$\frac{\partial \phi}{\partial s} = \left(\frac{-gH_z\rho}{\rho_o}\right) \tag{2.13}$$

$$\frac{\partial H_z}{\partial t} + \frac{\partial (H_z u)}{\partial x} + \frac{\partial (H_z v)}{\partial y} + \frac{\partial (H_z \Omega)}{\partial s} = 0 \tag{2.14}$$

where

$$\vec{v} = (u, v, \Omega)$$

$$\vec{v}\cdot\nabla = u\frac{\partial}{\partial x} + v\frac{\partial}{\partial y} + \Omega\frac{\partial}{\partial s}.$$

The vertical velocity in s coordinates is

$$\Omega(x, y, s, t) = \frac{1}{H_z}\left[w - (1+s)\frac{\partial \zeta}{\partial t} - u\frac{\partial z}{\partial x} - v\frac{\partial z}{\partial y}\right]$$

and

$$w = \frac{\partial z}{\partial t} + u\frac{\partial z}{\partial x} + v\frac{\partial z}{\partial y} + \Omega H_z.$$

In the stretched coordinate system, the vertical boundary conditions become:

$$\begin{aligned}
\text{top } (s{=}\ 0) \qquad & \left(\tfrac{\nu}{H_z}\right)\tfrac{\partial u}{\partial s} = \tau_s^x(x,y,t) \\
& \left(\tfrac{\nu}{H_z}\right)\tfrac{\partial v}{\partial s} = \tau_s^y(x,y,t) \\
& \left(\tfrac{\kappa_T}{H_z}\right)\tfrac{\partial T}{\partial s} = \tfrac{Q_T}{\rho_o c_P} + \tfrac{1}{\rho_o c_P}\tfrac{dQ}{dT}(T - T_{\text{ref}}) \\
& \left(\tfrac{\kappa_S}{H_z}\right)\tfrac{\partial S}{\partial s} = \tfrac{(E-P)S}{\rho_o} \\
& \Omega = 0 \\
\text{and bottom } (s=-1) \qquad & \left(\tfrac{\nu}{H_z}\right)\tfrac{\partial u}{\partial s} = \tau_b^x(x,y,t) \\
& \left(\tfrac{\nu}{H_z}\right)\tfrac{\partial v}{\partial s} = \tau_b^y(x,y,t) \\
& \left(\tfrac{\kappa_T}{H_z}\right)\tfrac{\partial T}{\partial s} = 0 \\
& \left(\tfrac{\kappa_S}{H_z}\right)\tfrac{\partial S}{\partial s} = 0 \\
& \Omega = 0.
\end{aligned}$$

Note the simplification of the boundary conditions on vertical velocity that arises from the s coordinate transformation.

2.5 Horizontal curvilinear coordinates

In many applications of interest (e.g., flow adjacent to a coastal boundary), the fluid may be confined horizontally within an irregular region. In such problems, a horizontal coordinate system which conforms to the irregular lateral boundaries is advantageous. It is often also true in many geophysical problems that the simulated flow fields have regions of enhanced structure (e.g., boundary currents or fronts) which occupy a relatively small fraction of the physical/computational domain. In these problems, added efficiency can be gained by placing more computational resolution in such regions.

The requirement for a boundary-following coordinate system and for a laterally variable grid resolution can both be met, for suitably smooth domains, by introducing an appropriate orthogonal coordinate transformation in the horizontal. Let the new coordinates be $\xi(x,y)$ and $\eta(x,y)$, where the relationship of horizontal arc length to the differential distance is given by:

$$(ds)_\xi = \left(\frac{1}{m}\right) d\xi \tag{2.15}$$

$$(ds)_\eta = \left(\frac{1}{n}\right) d\eta \tag{2.16}$$

Here, $m(\xi,\eta)$ and $n(\xi,\eta)$ are the scale factors which relate the differential distances $(\Delta\xi, \Delta\eta)$ to the actual (physical) arc lengths. Appendix B.1 contains the curvilinear version of several common vector quantities.

Denoting the velocity components in the new coordinate system by

$$\vec{v} \cdot \hat{\xi} = u \tag{2.17}$$

and

$$\vec{v} \cdot \hat{\eta} = v \tag{2.18}$$

the equations of motion (2.8)-(2.14) can be re-written (see, e.g., Arakawa and Lamb [1]) as:

$$\begin{aligned}
&\frac{\partial}{\partial t}\left(\frac{H_z u}{mn}\right) + \frac{\partial}{\partial \xi}\left(\frac{H_z u^2}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{H_z uv}{m}\right) + \frac{\partial}{\partial s}\left(\frac{H_z u\Omega}{mn}\right) \\
&\quad - \left\{\left(\frac{f}{mn}\right) + v\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right) - u\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right)\right\} H_z v = \\
&\quad - \left(\frac{H_z}{n}\right)\left(\frac{\partial \phi}{\partial \xi} + \frac{g\rho}{\rho_o}\frac{\partial z}{\partial \xi} + g\frac{\partial \zeta}{\partial \xi}\right) + \frac{H_z}{mn}\left(\mathcal{F}_u + \mathcal{D}_u\right)
\end{aligned} \tag{2.19}$$

$$\begin{aligned}
&\frac{\partial}{\partial t}\left(\frac{H_z v}{mn}\right) + \frac{\partial}{\partial \xi}\left(\frac{H_z uv}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{H_z v^2}{m}\right) + \frac{\partial}{\partial s}\left(\frac{H_z v\Omega}{mn}\right) \\
&\quad + \left\{\left(\frac{f}{mn}\right) + v\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right) - u\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right)\right\} H_z u = \\
&\quad - \left(\frac{H_z}{m}\right)\left(\frac{\partial \phi}{\partial \eta} + \frac{g\rho}{\rho_o}\frac{\partial z}{\partial \eta} + g\frac{\partial \zeta}{\partial \eta}\right) + \frac{H_z}{mn}\left(\mathcal{F}_v + \mathcal{D}_v\right)
\end{aligned} \tag{2.20}$$

$$\frac{\partial}{\partial t}\left(\frac{H_z T}{mn}\right) + \frac{\partial}{\partial \xi}\left(\frac{H_z uT}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{H_z vT}{m}\right) + \frac{\partial}{\partial s}\left(\frac{H_z \Omega T}{mn}\right) = \frac{H_z}{mn}\left(\mathcal{F}_T + \mathcal{D}_T\right) \tag{2.21}$$

$$\frac{\partial}{\partial t}\left(\frac{H_z S}{mn}\right) + \frac{\partial}{\partial \xi}\left(\frac{H_z uS}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{H_z vS}{m}\right) + \frac{\partial}{\partial s}\left(\frac{H_z \Omega S}{mn}\right) = \frac{H_z}{mn}\left(\mathcal{F}_S + \mathcal{D}_S\right) \tag{2.22}$$

$$\rho = \rho(T, S, P) \tag{2.23}$$

$$\frac{\partial \phi}{\partial s} = -\left(\frac{gH_z \rho}{\rho_o}\right) \tag{2.24}$$

$$\frac{\partial}{\partial t}\left(\frac{H_z}{mn}\right) + \frac{\partial}{\partial \xi}\left(\frac{H_z u}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{H_z v}{m}\right) + \frac{\partial}{\partial s}\left(\frac{H_z \Omega}{mn}\right) = 0. \tag{2.25}$$

Since z is a linear function of ζ, equation (2.25) can be rewritten as:

$$\frac{\partial}{\partial t}\left(\frac{\zeta}{mn}\right) + \frac{\partial}{\partial \xi}\left(\frac{H_z u}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{H_z v}{m}\right) + \frac{\partial}{\partial s}\left(\frac{H_z \Omega}{mn}\right) = 0. \tag{2.26}$$

All boundary conditions remain unchanged.

Chapter 3

Numerical Solution Technique

3.1 Vertical and horizontal discretization

3.1.1 Horizontal grid

In the horizontal (ξ, η), a traditional, centered, second-order finite-difference approximation is adopted. In particular, the horizontal arrangement of variables is as shown in Fig. 3.1. This is equivalent to the well known Arakawa "C" grid, which is well suited for problems with horizontal resolution that is fine compared to the first radius of deformation (Arakawa and Lamb [1]).

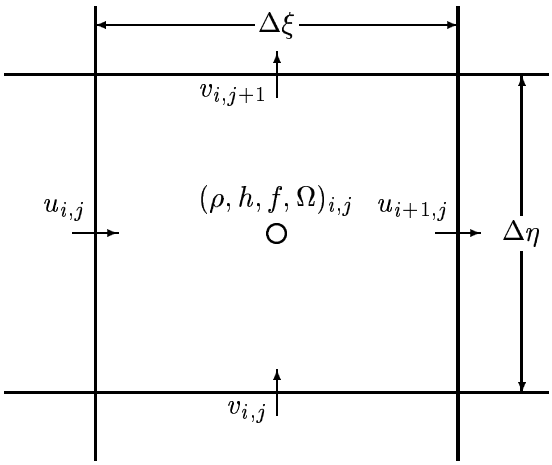

Figure 3.1: Placement of variables on an Arakawa C grid

3.1.2 Vertical grid

The vertical discretization also uses a second-order finite-difference approximation. Just as we use a staggered horizontal grid, the model was found to be more well-behaved with a

staggered vertical grid. The vertical grid is shown in Fig. 3.2.

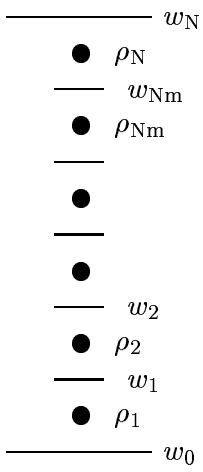

Figure 3.2: Placement of variables on staggered vertical grid

3.2 Masking of land areas

SCRUM has the ability to work with interior land areas, although the computations occur over the entire model domain. One grid cell is shown in Fig. 3.1 while several cells are shown in Fig. 3.3, 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.

3.2.1 Velocity

At the end of every timestep, the values of many variables within the masked region are set to zero by multiplying by the mask for either the u, v or ρ points. This is appropriate for the v points **E** and **L** in Fig. 3.3, 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. Likewise, the term $\frac{\partial}{\partial\xi}\nu\frac{\partial v}{\partial\xi}$ in the v equation must be corrected at the mask boundaries.

This is accomplished by having a fourth mask array defined at the ψ points, in which the values depend on whether or not free-slip boundaries are desired. In the case of free-slip, the value of $\frac{\partial u}{\partial\eta}$ is simply set to zero at points **D** and **K**. For no-slip boundaries, we count on the values inside the land (point **G**) having been zeroed out. For point **D**, the image point at **G** should contain minus the value of u at point **A**. The desired value of $\frac{\partial u}{\partial\eta}$ is therefore $2u_{\mathbf{A}}$ while instead we have simply $u_{\mathbf{A}}$. In order to achieve the correct result, we multiply

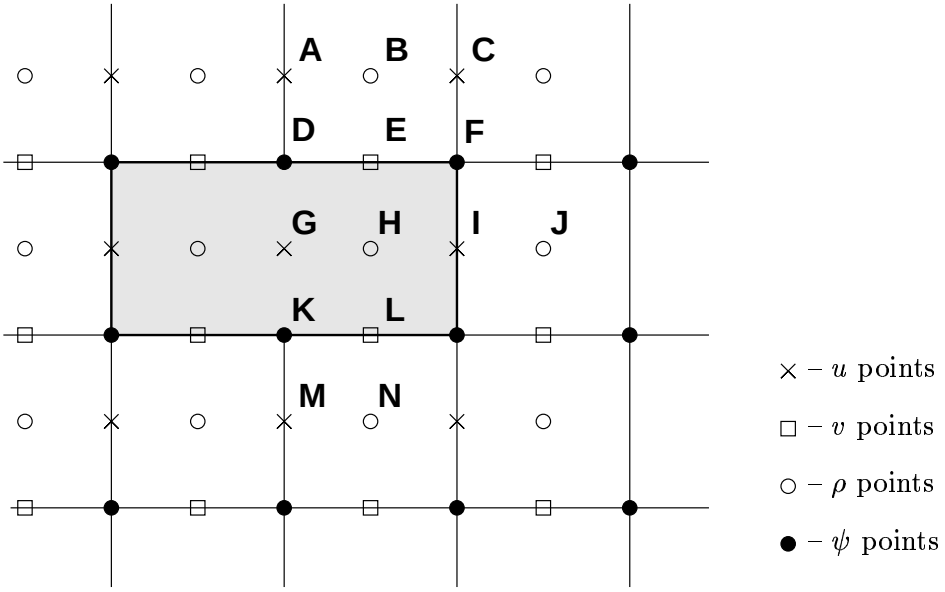

Figure 3.3: Masked region within the domain

by a mask which contains the value 2 at point **D**. It also contains a 2 at point **K** so that $\frac{\partial u}{\partial \eta}$ there will acquire the desired value of $-2u_{\mathbf{M}}$.

The corner point **F** is treated in the same way as points **D** and **K**. This can be changed in **get_mask** if desired.

3.2.2 Temperature, salinity and surface elevation

The handling of masks by the temperature, salinity and surface elevation equations is similar to that in the momentum equations, and is in fact simpler. Values of T, S and ζ inside the land masks, such as point **H** in Fig. 3.3, 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.

3.3 Conservation properties

SCRUM conserves the first moments of u, v, S, and T. This is accomplished by using the flux form of the momentum and tracer equations. It is also necessary to be careful when averaging between the velocity and tracer grids, for instance to obtain u at ρ points. The semi-discrete form of the dynamic equations (2.19)–(2.22) is:

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

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

$$-\frac{\overline{H_z}^{\xi}}{\overline{n}^{\xi}}\delta_{\xi}\phi-\frac{g\overline{H_z}^{\xi}}{\rho_o\overline{n}^{\xi}}\overline{\rho}^{\xi}\delta_{\xi}z-\frac{g\overline{H_z}^{\xi}}{\rho_o\overline{n}^{\xi}}(\rho_o+\overline{\rho}^{\xi})\delta_{\xi}\zeta+\mathcal{D}_u+\mathcal{F}_u \tag{3.1}$$

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

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

$$-\frac{\overline{H_z}^{\eta}}{\overline{m}^{\eta}}\delta_{\eta}\phi-\frac{g\overline{H_z}^{\eta}}{\rho_o\overline{m}^{\eta}}\overline{\rho}^{\eta}\delta_{\eta}z-\frac{g\overline{H_z}^{\eta}}{\rho_o\overline{m}^{\eta}}(\rho_o+\overline{\rho}^{\eta})\delta_{\eta}\zeta+\mathcal{D}_v+\mathcal{F}_v \tag{3.2}$$

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

$$+\delta_{s}\left(\overline{T}^{s}\frac{H_z\Omega}{mn}\right)=\mathcal{D}_T+\mathcal{F}_T \tag{3.3}$$

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

$$+\delta_{s}\left(\overline{S}^{s}\frac{H_z\Omega}{mn}\right)=\mathcal{D}_S+\mathcal{F}_S \tag{3.4}$$

$$\rho=\rho(S,T,P) \tag{3.5}$$

$$\phi(s)=-\frac{g}{\rho_o}I_s^0 H_z\rho \tag{3.6}$$

Here δ_{ξ}, δ_{η} and δ_s denote simple centered finite-difference approximations to $\partial/\partial\xi$, $\partial/\partial\eta$ and $\partial/\partial s$ with the differences taken over the distances $\Delta\xi$, $\Delta\eta$ and Δs, respectively. $\overline{(\quad)}^{\xi}$,

$\overline{(\quad)}^{\eta}$ and $\overline{(\quad)}^{s}$ represent averages taken over the distances $\Delta\xi$, $\Delta\eta$ and Δ_s. I_s^0 indicates a second-order vertical integral computed as a sum from level s to the surface at $s = 0$.

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 [1]; however, their scheme also conserves enstrophy.

The continuity equation will be discussed below in §3.7.

3.4 Vertical viscosity and diffusion

The $\mathcal{D}_u$, $\mathcal{D}_v$, $\mathcal{D}_T$ and $\mathcal{D}_S$ terms in equations (3.1)–(3.4) represent both horizontal and vertical mixing processes. The horizontal options will be covered in §3.9. The vertical viscosity and diffusion terms have the form:

$$\frac{\partial}{\partial s}\left(\frac{\kappa}{H_z mn}\frac{\partial \phi}{\partial s}\right) \tag{3.7}$$

where ϕ represents one of u, v, T or S. This is timestepped using a semi-implicit Crank-Nicholson scheme with a weighting of 0.5 on the old timestep and 0.5 on the new timestep. Specifically, the equation of motion for ϕ can be written as:

$$\frac{\partial (H_z \phi)}{\partial t} = mnR_\phi + \frac{\partial}{\partial s}\left(\frac{\kappa}{H_z}\frac{\partial \phi}{\partial s}\right) \tag{3.8}$$

where R_ϕ represents all of the forcing terms other than the vertical viscosity or diffusion. Since we want the diffusion term to be evaluated partly at the current timestep n and partly at the next timestep $n+1$, we introduce the parameter λ and rewrite equation (3.8) as:

$$\frac{\partial (H_z \phi)}{\partial t} = mnR_\phi + (1-\lambda)\frac{\partial}{\partial s}\left(\frac{\kappa}{H_z}\frac{\partial \phi^n}{\partial s}\right) + \lambda\frac{\partial}{\partial s}\left(\frac{\kappa}{H_z}\frac{\partial \phi^{n+1}}{\partial s}\right). \tag{3.9}$$

The discrete form of equation (3.9) is:

$$\begin{aligned}\frac{H_{z_k}^{n+1}\phi_k^{n+1} - H_{z_k}^n \phi_k^n}{\Delta t} = mnR_\phi &+ \frac{(1-\lambda)}{\Delta s^2}\left[\frac{\kappa_k}{H_{z_k}}(\phi_{k+1}^n - \phi_k^n) - \frac{\kappa_{k-1}}{H_{z_{k-1}}}(\phi_k^n - \phi_{k-1}^n)\right] \\ &+ \frac{\lambda}{\Delta s^2}\left[\frac{\kappa_k}{H_{z_k}}(\phi_{k+1}^{n+1} - \phi_k^{n+1}) - \frac{\kappa_{k-1}}{H_{z_{k-1}}}(\phi_k^{n+1} - \phi_{k-1}^{n+1})\right]\end{aligned} \tag{3.10}$$

where k is used as the vertical level index. This can be reorganized so that all the terms involving ϕ^{n+1} are on the left and all the other terms are on the right. The equation for ϕ_k^{n+1} will contain terms involving the neighbors above and below (ϕ_{k+1}^{n+1} and ϕ_{k+1}^{n-1}) which leads to a set of coupled equations with boundary conditions for the top and bottom. The general form of these equations is:

$$A_k\phi_{k+1}^{n+1} + B_k\phi_k^{n+1} + C_k\phi_{k-1}^{n+1} = D_k \tag{3.11}$$

where the boundary conditions are written into the coefficients for the end points. In this case the coefficients become:

$$A(1) = 0 \tag{3.12}$$

$$A(2:\mathbf{N}) = -\frac{\lambda \Delta t\, \kappa_{k-1}}{\Delta s^2 H^{n+1}_{z_{k-1}}} \tag{3.13}$$

$$B(1) = H^{n+1}_{z_1} + \frac{\lambda \Delta t\, \kappa_1}{\Delta s^2 H^{n+1}_{z_1}} \tag{3.14}$$

$$B(2:\mathbf{Nm}) = H^{n+1}_{z_k} + \frac{\lambda \Delta t\, \kappa_k}{\Delta s^2 H^{n+1}_{z_k}} + \frac{\lambda \Delta t\, \kappa_{k-1}}{\Delta s^2 H^{n+1}_{z_{k-1}}} \tag{3.15}$$

$$B(\mathbf{N}) = H^{n+1}_{z_\mathbf{N}} + \frac{\lambda \Delta t\, \kappa_\mathbf{Nm}}{\Delta s^2 H^{n+1}_{z_\mathbf{Nm}}} \tag{3.16}$$

$$C(1:\mathbf{Nm}) = -\frac{\lambda \Delta t\, \kappa_k}{\Delta s^2 H^{n+1}_{z_k}} \tag{3.17}$$

$$C(\mathbf{N}) = 0 \tag{3.18}$$

$$D(1) = H^n_{z_1}\phi^n_1 + \Delta t\, mn R_{\phi_1} + \frac{\Delta t(1-\lambda)}{\Delta s^2}\frac{\kappa_1}{H^n_{z_1}}(u^n_2 - u^n_1) - \frac{\Delta t}{\Delta s}\tau_b \tag{3.19}$$

$$D(2:\mathbf{Nm}) = H^n_{z_k}\phi^n_k + \Delta t\, mn R_{\phi_k} + \tag{3.20}$$

$$\frac{\Delta t(1-\lambda)}{\Delta s^2}\left[\frac{\kappa_k}{H^n_{z_k}}(u^n_{k+1} - u^n_k) - \frac{\kappa_{k-1}}{H^n_{z_{k-1}}}(u^n_k - u^n_{k-1})\right] \tag{3.21}$$

$$D(\mathbf{N}) = H^n_{z_\mathrm{N}}\phi^n_\mathrm{N} + \Delta t\, mn R_{\phi_\mathrm{N}} - \frac{\Delta t(1-\lambda)}{\Delta s^2}\frac{\kappa_\mathbf{Nm}}{H^n_{z_\mathbf{Nm}}}(u^n_\mathrm{N} - u^n_\mathrm{Nm}) + \frac{\Delta t}{\Delta s}\tau_s \tag{3.22}$$

This is a standard tridiagonal system for which the solution procedure can be found in any standard reference such as Press et al. [26].

3.5 Depth-integrated equations

The depth average of a quantity A is given by:

$$\overline{A} = \frac{1}{D}\int_{-1}^{0} H_z A ds \tag{3.23}$$

where the overbar indicates a vertically averaged quantity and

$$D \equiv \zeta(\xi, \eta, t) + h(\xi, \eta) \tag{3.24}$$

is the total depth of the water column. The vertical integral of equation (2.19) is:

$$\frac{\partial}{\partial t}\left(\frac{D\overline{u}}{mn}\right)+\frac{\partial}{\partial \xi}\left(\frac{D\overline{uu}}{n}\right)+\frac{\partial}{\partial \eta}\left(\frac{D\overline{uv}}{m}\right)-\frac{Df\overline{u}}{mn}$$
$$-\left[\overline{vv}\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right)-\overline{uv}\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right)\right]D=-\frac{D}{n}\left(\frac{\partial\overline{\phi_2}}{\partial \xi}+g\frac{\partial \zeta}{\partial \xi}\right)$$
$$+\frac{D}{mn}\left(\overline{\mathcal{F}}_u+\overline{\mathcal{D}}_{h_u}\right)+\frac{1}{mn}\left(\tau_s^{\xi}-\tau_b^{\xi}\right) \tag{3.25}$$

where ϕ_2 includes the $\frac{\partial z}{\partial \xi}$ term, $\overline{\mathcal{D}}_{h_u}$ is the horizontal viscosity and the vertical viscosity only contributes through the upper and lower boundary conditions. The corresponding vertical integral of equation (2.20) is:

$$\frac{\partial}{\partial t}\left(\frac{D\overline{v}}{mn}\right)+\frac{\partial}{\partial \xi}\left(\frac{D\overline{uv}}{n}\right)+\frac{\partial}{\partial \eta}\left(\frac{D\overline{vv}}{m}\right)+\frac{Df\overline{v}}{mn}$$
$$+\left[\overline{uv}\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right)-\overline{uu}\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right)\right]D=-\frac{D}{m}\left(\frac{\partial\overline{\phi_2}}{\partial \eta}+g\frac{\partial \zeta}{\partial \eta}\right)$$
$$+\frac{D}{mn}\left(\overline{\mathcal{F}}_v+\overline{\mathcal{D}}_{h_v}\right)+\frac{1}{mn}\left(\tau_s^{\eta}-\tau_b^{\eta}\right). \tag{3.26}$$

We also need the vertical integral of equation (2.26). Using the vertical boundary conditions on Ω we get:

$$\frac{\partial}{\partial t}\left(\frac{\zeta}{mn}\right)+\frac{\partial}{\partial \xi}\left(\frac{D\overline{u}}{n}\right)+\frac{\partial}{\partial \eta}\left(\frac{D\overline{v}}{m}\right)=0. \tag{3.27}$$

The presence of a free surface introduces waves which propagate at a speed of $\sqrt{gh}$. These waves usually impose a more severe timestep limit than any of the internal processes. We have therefore chosen to solve the full equations by means of a split timestep. In other words, the depth integrated equations (3.25), (3.26), and (3.27) are integrated using a short timestep and the values of $\overline{u}$ and $\overline{v}$ are used to replace those found by integrating the full equations on a longer timestep. A diagram of the timestepping is shown in Fig. 3.4.

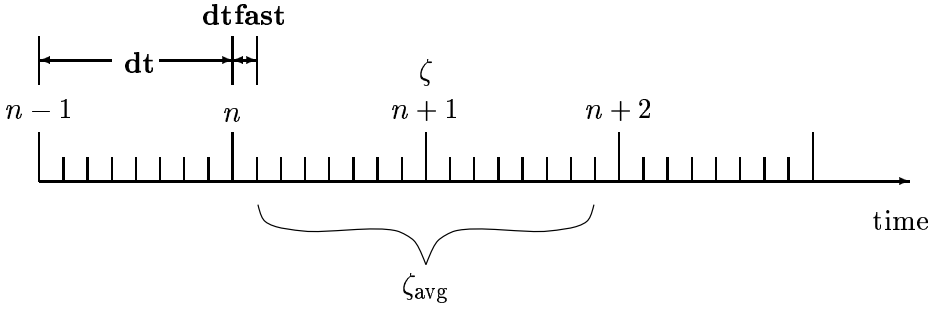

Figure 3.4: The split timestepping used in the model.

Some of the terms in equations (3.25) and (3.26) are updated on the short timestep while others are not. The contributions from the slow terms are computed once per long timestep and stored. If we call these terms $R_{u_{\rm slow}}$ and $R_{v_{\rm slow}}$, equations (3.25) and (3.26) become:

$$\frac{\partial}{\partial t}\left(\frac{D\overline{u}}{mn}\right)+\frac{\partial}{\partial \xi}\left(\frac{D\overline{u}\,\overline{u}}{n}\right)+\frac{\partial}{\partial \eta}\left(\frac{D\overline{u}\,\overline{v}}{m}\right)-\frac{Df\overline{u}}{mn}$$
$$-\left[\overline{v}\,\overline{v}\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right)-\overline{u}\,\overline{v}\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right)\right]D=R_{u_{\rm slow}}+g\frac{\partial \zeta}{\partial \xi}+\frac{D}{mn}\mathcal{D}_{\overline{u}}-\frac{1}{mn}\tau_b^{\xi} \tag{3.28}$$

$$\frac{\partial}{\partial t}\left(\frac{D\overline{v}}{mn}\right)+\frac{\partial}{\partial \xi}\left(\frac{D\overline{u}\,\overline{v}}{n}\right)+\frac{\partial}{\partial \eta}\left(\frac{D\overline{v}\,\overline{v}}{m}\right)+\frac{Df\overline{v}}{mn}$$
$$+\left[\overline{u}\,\overline{v}\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right)-\overline{u}\,\overline{u}\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right)\right]D=R_{v_{\rm slow}}+g\frac{\partial \zeta}{\partial \eta}+\frac{D}{mn}\mathcal{D}_{\overline{v}}-\frac{1}{mn}\tau_b^{\eta}. \tag{3.29}$$

When timestepping the model, we compute the right-hand-sides for equations (3.1) and (3.2) as well as the right-hand-sides for equations (3.28) and (3.29). The vertical integral of the 3-D right-hand-sides are obtained and then the 2-D right-hand-sides are subtracted. The resulting fields are the slow forcings $R_{u_{\rm slow}}$ and $R_{v_{\rm slow}}$. This was found to be the easiest way to retain the baroclinic contributions of the non-linear terms such as $\overline{uu}-\overline{u}\,\overline{u}$.

The model is timestepped from time n to time $n+1$ by using short timesteps on equations (3.28), (3.29) and (3.27). A trapezoidal-leapfrog timestepping is used. In practice, we actually timestep all the way to time $(n+2)$ – **dtfast** and then average the values of $\overline{u}$, $\overline{v}$ and ζ. The averages are used to replace the values at time $n+1$. These time averages damp out certain instabilities which would otherwise grow to dominate the solution.

3.6 Time stepping: internal velocity modes, temperature, and salinity

The momentum equations (3.1) and(3.2) are advanced by computing all the terms except the vertical viscosity and then using the implicit scheme described in §3.4 to find the new values for u and v. The depth-averaged component is then removed and replaced by the $\overline{u}$ and $\overline{v}$ computed as in §3.5. A third-order Adams-Bashforth timestepping is used when computing the right-hand-side terms (see Appendix A). The temperature and salinity equations (3.3) and (3.4) are also advanced as in §3.4. There is also an option to advect the temperature and salinity using a Smolarkiewicz scheme as described in [31] and [32].

3.7 Determination of the vertical velocity and density fields

Having obtained a complete specification of the u, v, T, and S fields at the next time level by the methods outlined above, the vertical velocity and density fields can be calculated. The vertical velocity is obtained by combining equations (2.26) and (3.27) to obtain:

$$\frac{\partial}{\partial \xi}\left(\frac{H_z u}{n}\right)+\frac{\partial}{\partial \eta}\left(\frac{H_z v}{m}\right)+\frac{\partial}{\partial s}\left(\frac{H_z \Omega}{mn}\right)-\frac{\partial}{\partial \xi}\left(\frac{D\overline{u}}{n}\right)-\frac{\partial}{\partial \eta}\left(\frac{D\overline{v}}{m}\right)=0. \tag{3.30}$$

Solving for $H_z\Omega/mn$ and using the semi-discrete notation of §3.3 we obtain:

$$\frac{H_z\Omega(s)}{mn} = \int \left[\delta_\xi \left(\frac{\overline{u}\overline{D}^\xi}{\overline{n}^\xi} \right) + \delta_\eta \left(\frac{\overline{v}\overline{D}^\eta}{\overline{m}^\eta} \right) - \delta_\xi \left(\frac{u\overline{H_z}^\xi}{\overline{n}^\xi} \right) - \delta_\eta \left(\frac{v\overline{H_z}^\eta}{\overline{m}^\eta} \right) \right] ds. \tag{3.31}$$

The integral is actually computed as a sum from the bottom upwards and also as a sum from the top downwards. The value used is a linear combination of the two, weighted so that the surface down value is used near the surface while the other is used near the bottom.

The density is obtained from temperature and salinity via an equation of state. SCRUM provides a choice of a nonlinear equation of state $\rho = \rho(T, S, z)$ or a linear equation of state $\rho = \rho(T)$. The nonlinear equation of state has been modified and now corresponds to the UNESCO equation of state as derived by Jackett and McDougall [16]. It computes *in situ* density as a function of potential temperature, salinity and pressure.

Warning: although we have used it quite extensively, McDougall (personal communication) claims that the single-variable ($\rho = \rho(T)$) equation of state is not dynamically appropriate as is. He has worked out the extra source and sink terms required, arising from vertical motions and the compressibility of water. They are quite complicated and we have not implemented them to see if they alter the flow.

3.8 The pressure gradient terms

The pressure gradient terms in equations (2.19) and (2.20) are written in the form

$$H_z \nabla\phi + \frac{g\rho H_z}{\rho_o} \nabla z + g H_z \nabla\zeta \tag{3.32}$$

This is the form traditionally used in sigma-coordinate models to account for the horizontal differences being taken along surfaces of constant s. This form can be shown to lead to significant errors when $|\nabla h|$ is large (Haney [13]; and Beckmann and Haidvogel [3]).

The pressure ϕ is computed by a vertical integration of the density field using equation (2.24). Prior to the integration, a horizontal average of the density, $\overline{\rho}(z)$, is subtracted from ρ. As discussed by Haney [13] and McCalpin [20], $\overline{\rho}$ 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.

3.9 Horizontal friction and diffusion

In Chapter 2 the diffusive terms were written simply as $\mathcal{D}_u, \mathcal{D}_v, \mathcal{D}_T$, and $\mathcal{D}_S$. The vertical component of these terms was described in §3.4. Here we describe SCRUM's options for representing the horizontal component of these terms.

3.9.1 Laplacian

The Laplacian of a scalar ϕ in curvilinear coordinates is (see Batchelor [2], Appendix 2):

$$\nabla^2\phi = \nabla \cdot \nabla\phi = mn \left[\frac{\partial}{\partial\xi}\left(\frac{m}{n}\frac{\partial\phi}{\partial\xi} \right) + \frac{\partial}{\partial\eta}\left(\frac{n}{m}\frac{\partial\phi}{\partial\eta} \right) \right] \tag{3.33}$$

This term in SCRUM is multiplied by $\frac{\nu_2 H_z}{mn}$ and becomes

$$\left[\frac{\partial}{\partial\xi}\left(\frac{\nu_2 H_z m}{n}\frac{\partial\phi}{\partial\xi}\right)+\frac{\partial}{\partial\eta}\left(\frac{\nu_2 H_z n}{m}\frac{\partial\phi}{\partial\eta}\right)\right] \tag{3.34}$$

where ϕ is any of u, v, T, and S. This form guarantees that the term does not contribute to the volume-integrated equations, except when using no-slip boundaries in the momentum equations.

3.9.2 Biharmonic

The biharmonic operator is $\nabla^4 = \nabla^2\nabla^2$; the corresponding term is computed using a temporary variable Y:

$$Y = \frac{mn}{H_z}\left[\frac{\partial}{\partial\xi}\left(\frac{\nu_4 H_z m}{n}\frac{\partial\phi}{\partial\xi}\right)+\frac{\partial}{\partial\eta}\left(\frac{\nu_4 H_z n}{m}\frac{\partial\phi}{\partial\eta}\right)\right] \tag{3.35}$$

and is

$$-\left[\frac{\partial}{\partial\xi}\left(\frac{H_z m}{n}\frac{\partial Y}{\partial\xi}\right)+\frac{\partial}{\partial\eta}\left(\frac{H_z n}{m}\frac{\partial Y}{\partial\eta}\right)\right] \tag{3.36}$$

where ϕ is once again any of u, v, T, and S. Note that u and v are treated as independent scalar quantities rather than as a vector. The complete Laplacian operator on a vector quantity $\vec{u}$ contains additional terms, including v terms in the u equation and vice versa. These extra terms were found to be small in a test problem and have been left out of the model.

3.9.3 Rotated mixing tensors

Both the Laplacian and biharmonic terms above operate on surfaces of constant s and can contribute substantially to the vertical mixing. However, the oceans are thought to mix along constant density surfaces so this is not entirely satisfactory. Therefore, the option of using rotated mixing tensors for the Laplacian and biharmonic operators has been added. Options exist both to diffuse on constant z surfaces (**MIX_GP_UV**, **MIX_GP_TS**) and constant *in situ* density surfaces (**MIX_EN_UV**, **MIX_EN_TS**).

The horizontal Laplacian diffusion operator is computed by finding the three components of the flux of the quantity ϕ. The ξ and η components are locally horizontal, rather than along the s surface. These fluxes are:

$$F^\xi = \nu_2 m\frac{\partial\phi}{\partial\xi} - \underbrace{\frac{\nu_2}{H_z}\left[m\frac{\partial z}{\partial\xi}\underbrace{+S_x}_{\texttt{MIX\_EN}}\right]\frac{\partial\phi}{\partial s}}_{\texttt{MIX\_GP}} \tag{3.37}$$

$$F^\eta = \nu_2 n\frac{\partial\phi}{\partial\eta} - \underbrace{\frac{\nu_2}{H_z}\left[n\frac{\partial z}{\partial\eta}\underbrace{+S_y}_{\texttt{MIX\_EN}}\right]\frac{\partial\phi}{\partial s}}_{\texttt{MIX\_GP}} \tag{3.38}$$

$$F^s = -\underbrace{\frac{1}{H_z}\left[m\frac{\partial z}{\partial \xi}\underbrace{+S_x}_{\texttt{MIX\_EN}}\right]F^\xi}_{\texttt{MIX\_GP}} - \underbrace{\frac{1}{H_z}\left[n\frac{\partial z}{\partial \eta}\underbrace{+S_y}_{\texttt{MIX\_EN}}\right]F^\eta}_{\texttt{MIX\_GP}} \tag{3.39}$$

where

$$S_x = \frac{\frac{\partial \rho}{\partial x}}{\frac{\partial \rho}{\partial z}} = \frac{\left[m\frac{\partial \rho}{\partial \xi} - \frac{m}{H_z}\frac{\partial z}{\partial \xi}\frac{\partial \rho}{\partial s}\right]}{\frac{1}{H_z}\frac{\partial \rho}{\partial s}}$$

$$S_y = \frac{\frac{\partial \rho}{\partial y}}{\frac{\partial \rho}{\partial z}} = \frac{\left[n\frac{\partial \rho}{\partial \eta} - \frac{n}{H_z}\frac{\partial z}{\partial \eta}\frac{\partial \rho}{\partial s}\right]}{\frac{1}{H_z}\frac{\partial \rho}{\partial s}}$$

No flux boundary conditions are easily imposed by setting

$$\begin{aligned} F^\xi &= 0 \qquad \text{at } \xi \text{ walls} \\ F^\eta &= 0 \qquad \text{at } \eta \text{ walls} \\ F^s &= 0 \qquad \text{at } s = -1, 0 \end{aligned}$$

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

$$\frac{\partial}{\partial \xi}\left(\frac{H_z F^\xi}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{H_z F^\eta}{m}\right) + \frac{\partial}{\partial s}\left(\frac{H_z F^s}{mn}\right) \tag{3.40}$$

The biharmonic rotated mixing tensors are computed much as the non-rotated biharmonic mixing. We define a temporary variable Y based on equation (3.40):

$$Y = \frac{mn}{H_z}\left[\frac{\partial}{\partial \xi}\left(\frac{H_z F^\xi}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{H_z F^\eta}{m}\right) + \frac{\partial}{\partial s}\left(\frac{H_z F^s}{mn}\right)\right]. \tag{3.41}$$

We then build up fluxes of Y as in equations (3.37)–(3.39). We then apply equation (3.40) to these Y fluxes to obtain the biharmonic mixing tensors.

Chapter 4

Details of the Code

4.1 Main subroutines

A flow chart for the main program is shown in Fig. 4.1. The boxes refer to subroutines which are described as follows:

couple Couples 3-D and 2-D momentum fields. It computes and removes the vertical means from the newly computed 3-D velocities and replaces those means with the more accurate 2-D velocities.

depth2d Computes the evolving total depth of the water column that is associated with the 2-D momentum equations. It also computes the coefficients used in the advection and viscosity of 2-D momentum.

depth3d Computes the evolving depths of the model grid and its associated vertical transformation metric H_z. It also computes the coefficients which contain H_z and are used in the horizontal advection of momentum and tracers and in the horizontal mixing.

frc2drhs Computes the forcing terms (**rufrc**,**rvfrc**) for the 2-D momentum equations which are held constant over the short time steps. These forcing terms contains the vertically integrated terms from the 3-D momentum equations which are not considered in the 2-D equations.

initial Does everything that needs to be done to start up the model run. It reads initial parameters and $u, v, T, S,$ and ζ fields from disk or calls **ana_initial**. It then calculates the remaining initial fields and opens the restart file. The flow chart for **initial** is shown in Fig. 4.2.

omega Calculates the scaled vertical velocity $H_z\Omega/mn$ according to equation (3.31).

prsgrd Calculates the horizontal pressure gradients according to equation (3.32).

rho_eos Calculates the density anomaly, ρ, using the equation of state (§3.7).

set_vbc Sets the vertical boundary conditions for momentum and tracers.

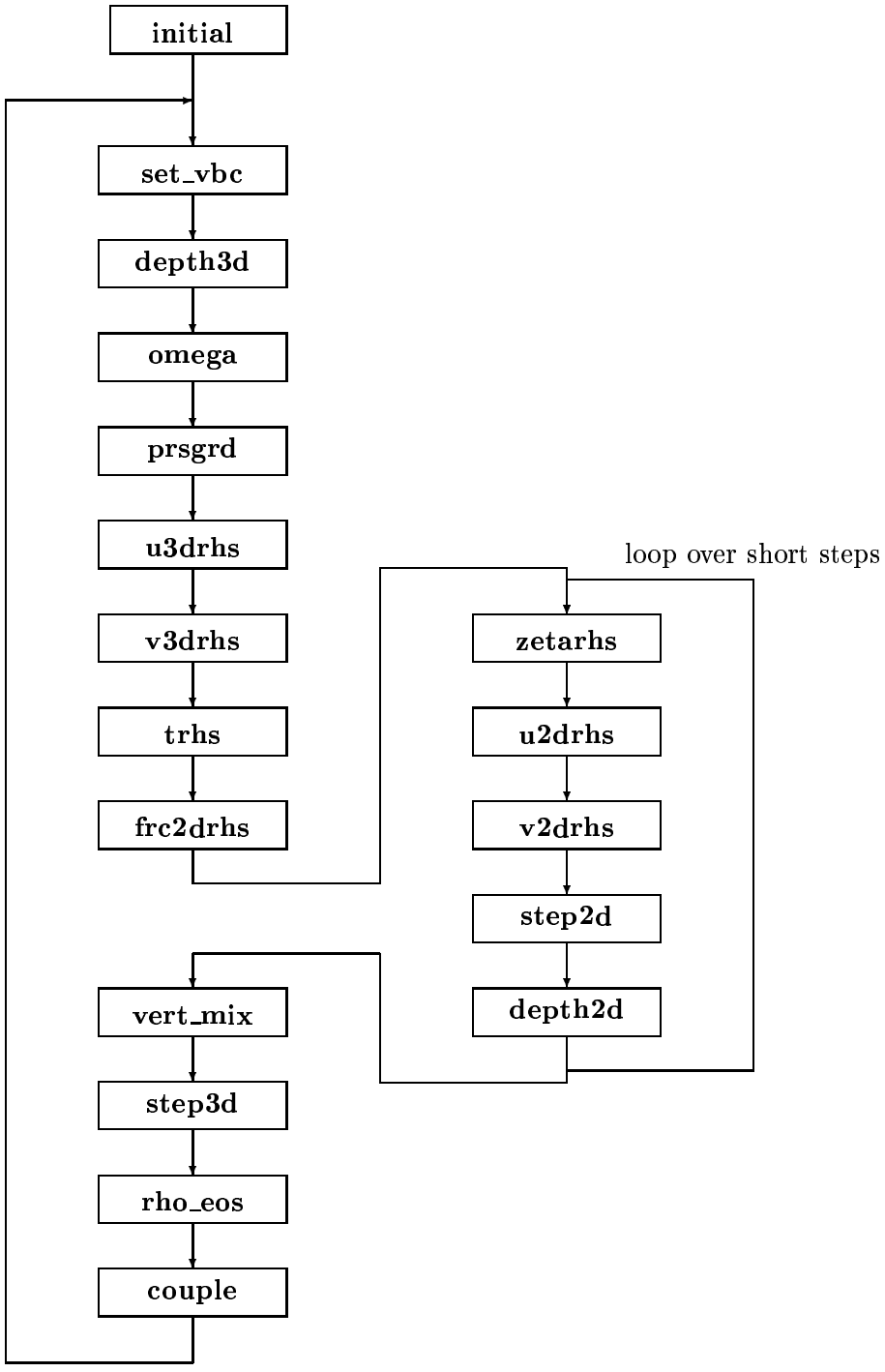

Figure 4.1: Flow chart of the model main program.

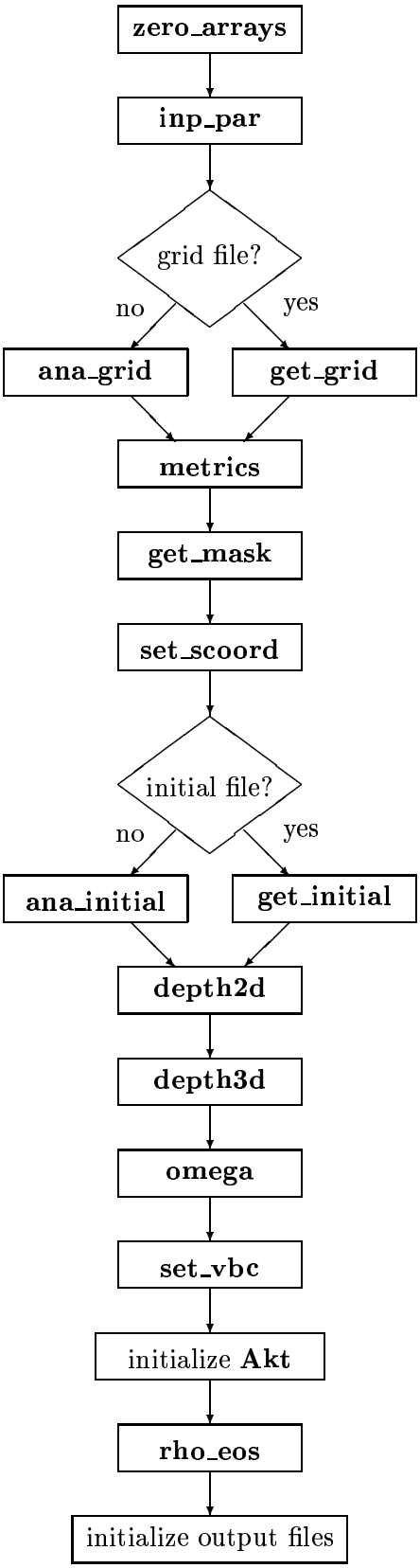

Figure 4.2: Flow chart of the **initial** subroutine.

step2d Time steps free-surface and 2-D momentum equations. Also does the time-averaging described in §3.5.

step3d Time steps the 3-D momentum and tracers (usually potential temperature and salinity), using the tridiagonal solver described in §3.4

trhs Calculates and stores contributions to the right-hand-side of the tracer equations (2.21) and 2.22), where the advective terms have been moved to the right-hand-side.

u2drhs Calculates and stores contributions to the right-hand-side of equation (3.28), where all the terms other than $\frac{\partial}{\partial t}(\frac{Du}{mn})$ have been moved to the right-hand-side.

u3drhs Calculates and stores contributions to the right-hand-side of equation (2.19), where all the terms other than $\frac{\partial}{\partial t}(\frac{H_z u}{mn})$ have been moved to the right-hand-side.

v2drhs Calculates and stores contributions to the right-hand-side of equation (3.29), where all the terms other than $\frac{\partial}{\partial t}(\frac{Dv}{mn})$ have been moved to the right-hand-side.

v3drhs Calculates and stores contributions to the right-hand-side of equation (2.20), where all the terms other than $\frac{\partial}{\partial t}(\frac{H_z v}{mn})$ have been moved to the right-hand-side.

vert_mix Computes the vertical mixing coefficients for momentum (**Akv**) and tracers (**Akt**).

zetarhs Computes the right-hand-side of equation (3.27), where all the terms other than $\frac{\partial}{\partial t}(\frac{\zeta}{mn})$ have been moved to the right-hand-side.

4.2 Other subroutines and functions

Initialization

ana_grid Sets up an analytic grid.

ana_initial Sets up analytic initial conditions.

blkdat Initializes some variables and parameters stored in common blocks.

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

get_grid Reads in the curvilinear coordinate arrays as well as f and h from a grid netCDF file.

get_initial Reads initial fields from disk—either restart or initializing from a climatology.

get_mask Reads in the mask arrays from the grid netCDF file. It also adjusts **pmask** as required for the free-slip/no-slip boundary conditions as described in §3.2.

inp_par Reads in input model parameters from standard input. It also writes out these parameters to standard output and calls **checkdefs**.

metrics Computes the metric term combinations which do not depend on the surface elevation and therefore remain constant in time.

set_scoord Sets and initializes relevant variables associated with the vertical transformation to nondimensional s-coordinate described in Appendix B.

zero_arrays Initializes (zeroes out) various arrays.

NetCDF I/O

def_avg Creates the SCRUM averages NetCDF file and defines its dimensions, attributes, and variables.

def_his Creates the SCRUM history NetCDF file and defines its dimensions, attributes, and variables.

def_rst Creates the SCRUM restart NetCDF file and defines its dimensions, attributes, and variables.

def_station Creates the SCRUM station NetCDF file and defines its dimensions, attributes, and variables.

get_date Gets today's date, day of the week and time called. It uses Sun's intrinsic **date** routine by default.

lenstr Returns the character position of the last non-blank character in a "string" after removing the leading blank characters, if any. Should not be called with a literal string argument.

opencdf Opens an existing NetCDF file, inquires about its contents, and checks for consistency with model dimensions.

wrt_avg Writes SCRUM time-averaged fields into the averages NetCDF file.

wrt_his Writes requested SCRUM fields at requested levels into the history NetCDF file.

wrt_rst Writes SCRUM fields into the restart NetCDF file.

wrt_station Writes out data into the stations NetCDF file.

Forcing fields The file **analytic.F** contains analytical formulations for computing various forcings and initializations. For more realistic problems these fields are read from NetCDF files.

ana_bmflux Computes analytic kinematic bottom momentum flux.

ana_btflux Computes analytic kinematic bottom flux of tracer type variables.

ana_clima Computes analytic climatology fields.

ana_smflux Computes analytic kinematic surface momentum flux (wind stress).

ana_srflux Computes analytic kinematic surface shortwave radiation.

ana_sst Computes analytic sea surface temperature and dQdSST which are used in the surface heat flux correction.

ana_stflux Computes analytic kinematic surface flux of tracer type variables.

get_bmflux Reads bottom momentum flux (bottom stress) from the forcing NetCDF file, and then linearly time-interpolates to current model time.

get_btflux Reads bottom flux of tracer type variables from the forcing NetCDF file, and then linearly time-interpolates to current model time.

get_clima Reads in climatological fields from the climatology NetCDF file, and then linearly time-interpolates to current model time.

get_cycle Determines relevant parameters for time cycling of data from the forcing NetCDF file. For instance, you may wish to use monthly means for each year of a multi-year run.

get_smflux Reads surface momentum flux (wind stress) from the forcing NetCDF file, and then linearly time-interpolates to current model time.

get_srflux Reads shortwave radiation flux from the forcing NetCDF file, and then linearly time-interpolates to current model time.

get_sst Reads sea surface temperature and surface net heat flux sensitivity to sea surface temperature from the forcing NetCDF file and then linearly time-interpolates to current model time.

get_stflux Reads surface flux of tracer type variables from the forcing NetCDF file, and then linearly time-interpolates to current model time.

Horizontal mixing The horizontal mixing routines have options for doing Laplacian or biharmonic mixing, along surfaces of constant s, z, or *in situ* density, as described in §3.9. The horizontal mixing of 2-D momentum is computed in **u2drhs** and **v2drhs**.

get_rhosxe Computes the isopycnal slopes (nondimensional) used in the mixing tensor rotation relative to geopotential surfaces.

shapd Applies a 2-D Shapiro filter to array **a**.

t3dmix Computes horizontal mixing of tracer type variables.

u3dmix Computes horizontal mixing of the 3-D momentum component in the ξ-direction.

v3dmix Computes horizontal mixing of the 3-D momentum component in the η-direction.

Vertical mixing The model contains a variety of methods for computing the vertical mixing coefficients **Akt** and **Akv**, including an analytic formula.

ana_vmix Computes analytic vertical mixing coefficients for momentum and tracers.

bv_freq Computes the squared Brunt-Väisälä frequency at w-points $N^2 = -\frac{g}{\rho_o}\frac{\partial \rho}{\partial z}$.

lmd_vmix Computes vertical mixing coefficients for momentum and tracers at the ocean interior using the Large, McWilliams and Doney [18] mixing scheme.

lmd_bldepth Determines the oceanic planetary boundary layer depth, **hbl**, as the shallowest depth where the bulk Richardson number is equal to the critical value, **Ric**.

lmd_blmix Sets the vertical mixing coefficients within the boundary layer.

lmd_swfrac Computes the fraction of solar shortwave flux penetrating to specified depth (times Zscale) due to exponential decay in Jerlov water type.

lmd_wscale Computes the turbulent velocity scale for momentum and tracers using a 2-D lookup table as a function of **ustar** and **zetahat**.

my25_vmix Computes vertical mixing coefficients for momentum and tracers using the Mellor and Yamada [22] mixing level 2.5 scheme with modifications described in Galperin et al. [10].

my25_q Solves the prognostic equation for turbulent energy variables used in the Mellor-Yamada level 2.5 turbulent closure.

my2_vmix Computes vertical mixing coefficients for momentum and tracers using the Mellor and Yamada [22] mixing level 2 scheme which is based on Richardson number-dependent stability functions.

pp_vmix Computes vertical mixing coefficients for momentum and tracers using the Pacanowski and Philander [24] mixing scheme which is based on the Richardson number.

ri_number Computes the gradient Richardson number for the vertical mixing schemes.

trisolver Solves the PDE $A\phi(k-1) + B\phi(k) + C\phi(k+1) = D$ for field ϕ using the tridiagonal solver, also known as Thomas algorithm (Richtmeyer and Morton [28]).

Bottom boundary-layer model The model has an optional bottom boundary layer based on Styles and Glenn [34].

ana_bsedim Computes analytic bottom sediment grain size and density.

ana_wwave Computes analytic wind induced wave amplitude, direction and period.

get_bsedim Reads initial sediment grain size and density from the forcing NetCDF file.

get_wwave Reads wind induced wave amplitude, direction and period from the forcing NetCDF file, and then linearly time-interpolates to current model time.

sg_bbl96 Computes kinematic bottom momentum stress using Styles and Glenn [34] bottom boundary layer formulation.

sg_ubab Computes maximum wave bottom velocity and excursion from wind induced wave amplitude and period by solving the linear wave dispersion relation for a given wave number.

Boundary conditions The files **bcs2d.F** and **bcs3d.F** contain horizontal boundary conditions for the various 2-D and 3-D variables, respectively. The boundary routines are also called to specify boundary conditions on $\nabla^2\phi$ for the horizontal biharmonic operator on the field ϕ.

ana_t3dbc Computes analytic boundary conditions for tracer type variables.

ana_u2dbc Computes analytic boundary conditions for 2-D u-type variables.

ana_u3dbc Computes analytic boundary conditions for 3-D u-type variables.

ana_v2dbc Computes analytic boundary conditions for 2-D v-type variables.

ana_v3dbc Computes analytic boundary conditions for 3-D v-type variables.

ana_w3dbc Computes analytic boundary conditions for 3-D w-type variables.

ana_zetabc Computes analytic boundary conditions for free-surface type variables.

inflow Processes prescribed inflow open boundary conditions from climatology data.

t3dbc Boundary conditions for 3-D tracer type variables.

u2dbc Boundary conditions for 2-D u-type variables.

u3dbc Boundary conditions for 3-D u-type variables.

v2dbc Boundary conditions for 2-D v-type variables.

v3dbc Boundary conditions for 3-D v-type variables.

w3dbc Boundary conditions for 3-D w-type variables.

xtrbry Extracts and loads data into boundary field arrays. These boundary field arrays are used in the treatment of the open boundaries via radiation conditions.

zetabc Boundary conditions for free-surface type variables.

Other

abratio Calculates the ratio of the thermodynamic expansion coefficients for potential temperature and salinity, alpha/beta, at horizontal and vertical w-points from a polynomial expression (Jackett and McDougall [16]).

alfabeta Computes thermal expansion and saline contraction coefficients as a function of potential temperature, salinity, and pressure from a polynomial expression (Jackett and McDougall [16]).

ana_diag Computes customized diagnostics.

ana_meanRHO Analytical mean density anomaly (**rhobar**).

crash Dies in the manner appropriate for your computer (**stop** or **call exit**). It also closes any open NetCDF files.

day_code computes a code for the day of the week, given the date. This code is good for dates after January 1, 1752 AD, the year the Gregorian calendar was adopted in Britain and the American colonies.

diag Computes various diagnostic fields, such as the volume averaged kinetic and potential energies.

smol_adv Evaluates horizontal and vertical advection terms for tracers using the Smolarkiewicz [30] advection scheme. It uses an upstream advection scheme with a second corrective upstream step to reduce the implicit diffusion. An anti-diffusion velocity is computed and used in the second pass through the advection operator.

smol_adiff Computes the "anti-diffusion velocity" used to suppress the numerical diffusion that is associated with the upstream differencing operator for advection.

smol_ups Computes a first-order upstream differencing operator for the 3-D advection of a tracer (scalar) field.

wvelocity Computes vertical velocity (w) from the model vertical velocity ($\Omega H_z/mn$).

4.3 C preprocessor variables

Before it can be compiled, the model must be run through the C preprocessor **cpp**, as described in Appendix D. The C preprocessor has its own variables, which may be defined either with an explicit **#define** command or with a command line option to **cpp**. We have chosen to define these variables in an include file, **cppdefs.h**, except for some machine-dependent ones, which are defined in the appropriate Makefiles. These variables allow you to conditionally compile sections of the code. For instance, if **MASKING** is not defined then the masking code will not be seen by the compiler, and the masking variables will not be declared. These **cpp** variables can be grouped into several categories:

momentum terms

BODYFORCE Define to apply the surface stresses as a body force.

CURVGRID Define to compute the extra non-linear terms which arise when using curvilinear coordinates.

UV_ADV Define to compute the momentum advection terms.

UV_COR Define to compute the Coriolis term.

UV_PRS Define to compute the horizontal pressure gradient term.

UV_VIS2 Define to compute the horizontal Laplacian viscosity.

UV_VIS4 Define to compute the horizontal biharmonic viscosity.

tracers

DIAGNOSTIC Define if it is a diagnostic calculation in which the tracer fields do not change in time.

NONLIN_EOS Define to use the nonlinear equation of state.

QCORRECTION Define to use the net heat flux correction.

SALINITY Define if salinity is used as one of the tracers.

SMOLARKIEWICZ Define to compute Smolarkiewicz advection.

TS_ADV Define to compute the tracer advection terms.

TS_DIF2 Define to compute the horizontal Laplacian diffusion.

TS_DIF4 Define to compute the horizontal biharmonic diffusion.

general model configuration

AVERAGES Define to write out time-averaged model fields.

RMDOCINC Define to remove documentation in include files with the C preprocessor.

SOLVE2D Define to solve the 2-D primitive equations.

SOLVE3D Define to solve the 3-D primitive equations.

STATIONS Define to write out time-series information at specific points in the model.

TIME_AVG Define to average over short timesteps as described in §3.5.

analytic fields

ANA_BMFLUX Define for an analytic bottom momentum stress.

ANA_BSEDIM Define for an analytic bottom sediment grain size and density.

ANA_BSFLUX Define for an analytic bottom salt flux.

ANA_BTFLUX Define for an analytic bottom heat flux.

ANA_CLIMA Define for an analytic climatology.

ANA_DIAG Define for customized diagnostics.

ANA_GRID Define for an analytic model grid set-up.

ANA_INITIAL Define for analytic initial conditions.

ANA_MEANRHO Define for an analytic mean density anomaly.

ANA_SMFLUX Define for an analytic kinematic surface momentum stress.

ANA_SRFLUX Define for an analytic kinematic surface shortwave radiation.

ANA_SSFLUX Define for an analytic kinematic surface freshwater flux.

ANA_SST Define for an analytic SST and $\partial Q/\partial$SST.

ANA_STFLUX Define for an analytic kinematic surface heat flux.

ANA_VMIX Define for analytic vertical mixing coefficients.

ANA_WWAVE Define for an analytic wind induced wave field.

analytic boundary conditions

ANA_T3DBC Define for analytic boundary conditions on tracers.

ANA_U2DBC Define for analytic boundary conditions on the 2-D u-momentum.

ANA_U3DBC Define for analytic boundary conditions on the 3-D u-momentum.

ANA_V2DBC Define for analytic boundary conditions on the 2-D v-momentum.

ANA_V3DBC Define for analytic boundary conditions on the 3-D v-momentum.

ANA_W3DBC Define for analytic boundary conditions on the 3-D w-momentum.

ANA_W3DBC Define for analytic boundary conditions on the free surface.

horizontal mixing

CLIMAT_TS_MIXH Define for horizontal diffusion of a tracer minus its climatology.

MIX_GP_TS Define for diffusion along constant z (geopotential) surfaces.

MIX_EN_TS Define for diffusion along constant ρ (epineutral) surfaces.

MIX_GP_UV Define for viscosity along constant z (geopotential) surfaces.

MIX_EN_UV Define for viscosity along constant ρ (epineutral) surfaces.

MIX_S_TS Define for diffusion along constant s surfaces.

MIX_S_UV Define for viscosity along constant s surfaces.

vertical mixing

CLIMAT_TS_MIXV Define for vertical diffusion of a tracer minus its climatology.

BVF_MIXING Define to activate Brunt-Väisälä frequency mixing.

LMD_MIXING Define to activate Large/McWilliams/Doney interior closure.

LMD_CONVEC Define to add convective mixing due to shear instabilities.

LMD_DDMIX Define to add double-diffusive mixing.

LMD_KPP Define to add boundary layer mixing from a local K-Profile Parameterization (KPP).

LMD_RIMIX Define to add diffusivity due to shear instabilities.

MY2_MIXING Define to activate Mellor/Yamada Level-2 closure.

MY25_MIXING Define to activate Mellor/Yamada Level-2.5 closure.

Q_DIF2 Define for horizontal Laplacian diffusion of q.

Q_DIF4 Define for horizontal biharmonic diffusion of q.

PP_MIXING Define to activate Pacanowski/Philander closure.

SG_BBL96 Define to activate Styles/Glenn bottom boundary layer formulation.

boundary conditions

EW_PERIODIC Define for periodic boundaries in the **i** direction.

NS_PERIODIC Define for periodic boundaries in the **j** direction.

OBC_EAST Define for an open boundary on the "east" (**i** = **L**) edge.

OBC_NORTH Define for an open boundary on the "north" (**j** = **M**) edge.

OBC_SOUTH Define for an open boundary on the "south" ($\mathbf{j} = \mathbf{1}$) edge.

OBC_WEST Define for an open boundary on the "west" ($\mathbf{i} = \mathbf{1}$) edge.

detailed open boundary conditions

OBC_INFLOW Define to process inflow conditions from climatology data.

OBC_FSGRADIENT Define for a gradient condition on the free surface.

OBC_FSGRAVITY Define for gravity-wave radiation of the free surface.

OBC_FSORLANSKI Define for an Orlanski radiation condition on the free surface.

OBC_FSMORLANSKI Define for a modified Orlanski radiation condition on the free surface.

OBC_FSPRESCRIBE Define to prescribe the free surface boundary from data.

OBC_M2GRADIENT Define for a gradient condition on the 2-D momentum.

OBC_M2GRAVITY Define for gravity-wave radiation of the 2-D momentum.

OBC_M2ORLANSKI Define for an Orlanski radiation condition on the 2-D momentum.

OBC_M2MORLANSKI Define for a modified Orlanski radiation condition on the 2-D momentum.

OBC_M2PRESCRIBE Define to prescribe the 2-D momentum boundary from data.

OBC_M2REDUCED Define for reduced physics on 2-D momentum at the boundary.

OBC_M3GRADIENT Define for a gradient condition on the 3-D momentum.

OBC_M3ORLANSKI Define for an Orlanski radiation condition on the 3-D momentum.

OBC_M3MORLANSKI Define for a modified Orlanski radiation condition on the 3-D momentum.

OBC_M3PRESCRIBE Define to prescribe the 3-D momentum boundary from data.

OBC_TCLIMA Define for relaxation to the tracer climatology.

OBC_TORLANSKI Define for an Orlanski radiation condition on tracers.

OBC_TMORLANSKI Define for a modified Orlanski radiation condition on tracers.

OBC_TPRESCRIBE Define to prescribe the tracer boundary from data.

OBC_TREDUCED Define for reduced physics on tracers at the boundary.

general

MASKING Define if there is land in the domain to be masked out.

CLIMATOLOGY Define to define the climatology arrays.

NUDGING Define for nudging to climatology.

precision These variables were introduced so that one code could be used for Crays and workstations, all using 64 bit precision.

DBLEPREC For double precision arithmetic.

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.

FLoaT This is used so that the correct intrinsic is called, either **float**, **dfloat** or **real**.

model test problems One of these can be defined to obtain an example test problem.

BASIN Define for the "Big Bad Basin" example.

CANYON_A Define for the Canyon A (homogeneous) example.

CANYON_B Define for the Canyon B (stratified) example.

GRAV_ADJ Define for the gravitational adjustment example.

OVERFLOW Define for the overflow example.

SEAMOUNT Define for the seamount example.

TASMAN_SEA Define for the Tasman Sea example.

UPWELLING Define for the upwelling/downwelling example described in §6.2.

command line These are defined as a command line option in some of the **Makefiles** since they are machine dependent.

NO_EXIT This will determine whether your program ends with a **stop** command or by calling **exit**. I prefer **exit** on a Sun and **stop** on an IBM RS/6000. The RS/6000 will not properly close files when using **call exit** so it is possible to lose some of your output unless you use **stop**.

Note that the **exit** subroutine on many computers does not require an argument. The Sun **exit** subroutine uses the integer argument value as the return code from SCRUM for use by the shell under which SCRUM is run.

AIX Most versions of **cpp** which are supplied by the vendor have some variables automatically defined. For instance, on a SparcStation, **sun**, **unix**, and **sparc** will all be defined. However, the RS/6000 **cpp** does not define anything useful to check for so I have the RS/6000 Makefile define **AIX**. This is used because both the SGI and the IBM RS/6000 will continue to compute if some variables have become **NaN**. In order to stop the calculation, we check for **NaN** as the error from **diag**, but the method of checking varies from one system to another. Another system-dependent component of SCRUM is in the implementation of **get_date**.

4.4 Important parameters

The following is a list of the important parameters in the model. The rest of the parameters defined in **param.h** are derived from **L**, **M**, and **N**.

L Number of grid points in the ξ-direction.

M Number of grid points in the η-direction.

N Number of grid points in the vertical.

NS Maximum number of output station points.

NT Number of tracer fields. Often **NT** = 2 for potential temperature and salinity.

There are a lot of parameters defined in **pconst.h** to represent literal constants of type **BIGREAL**. It is much safer to use the parameters when these values are needed as subroutine arguments. The names for the constants were chosen based on the following "rules":

- Use a prefix of **c** for whole real numbers (**c0** for zero and **c1** for 1.0).
- Use a prefix of **p** for non repeating fractions (**p5** for 0.5).
- Use a prefix of **r** for reciprocals (**r3** for 1.0/3.0 and **r10** for 0.1 which could also be **p1**).
- Combine use of the prefix and **e** for scientific notation (**c1e4** for $1.0e+4$ and **c1em4** for $1.0e-4$).
- Use names when appropriate (**pi** for $\pi = 3.14159265\ldots$).

4.5 Include files and the variables within them

SCRUM has a fair number of include files which contain common blocks. The common blocks contain all of the global variables in the model. All dimensional variables are given in MKS units.

ocean.h The main time-dependent model fields.

time	SCRUM time since initialization.
u	3-D velocity component in the ξ-direction.
v	3-D velocity component in the η-direction.
w	$H_z\Omega/mn$, scaled velocity component in the σ-direction.
t	tracer variables (usually potential temperature and salinity).
rho	perturbation density ρ (total density = $\rho_o + \rho$).
rhobar	horizontal average of density, must be a function of z only (see §3.8).
phix	ξ-component of the baroclinic pressure gradient.

phie η-component of the baroclinic pressure gradient.

zeta free surface ζ at the previous and current short time levels.

ubar 2-D velocity component in the ξ-direction at the previous and current short time levels.

vbar 2-D velocity component in the η-direction at the previous and current short time levels.

ubaravg 2-D velocity component in the ξ-direction time-averaged over all short timesteps, at the previous, current, and future long time levels.

vbaravg 2-D velocity component in the η-direction time-averaged over all short timesteps, at the previous, current, and future long time levels.

zetaavg free surface ζ time-averaged over all short timesteps, at the previous, current, and future long time levels.

This file also contains the time-averaged fields used for the monthly-mean computations:

avgrho average potential density anomaly.

avgt average tracer type variables (usually potential temperature and salinity).

avgtime average SCRUM time since initialization.

avgu2d average 2-D velocity component in the ξ-direction.

avgu3d average 3-D velocity component in the ξ-direction.

avgv2d average 2-D velocity component in the η-direction.

avgv3d average 3-D velocity component in the η-direction.

avgw3d average s-coordinate ($\Omega H_z/mn$) vertical velocity.

avgzeta average free surface elevation.

bblm.h Fields required by the bottom boundary layer of Styles and Glenn [34]. It computes the bottom stress and the hydraulic roughness length z_o due to the combined effects of waves and currents.

Ab wave bottom excursion amplitude.

Awave wind induced wave amplitude at ρ-points.

Cr non-dimensional function that determines the relative importance of currents and wind induced waves on the bottom stress at ρ-points.

Dwave wind induced wave direction (radians) at ρ-points.

Pwave wind induced wave period at ρ-points.

Sdens sediment grain density at ρ-points.

Ssize sediment grain diameter at ρ-points.

Ub maximum wave bottom horizontal velocity.

UstarC time-averaged near-bottom friction current magnitude at ρ-points.

sg_Kiter maximum number of iterations in the computation of bottom wavenumber.

sg_Keps convergence criterion for the computation of bottom wavenumber via the Newton-Raphson method.

sg_Siter maximum number of iterations in the computation of stress due to bottom friction velocity.

sg_Seps convergence criterion for the computation of the stress due to bottom friction velocity via the Newton-Raphson method.

sg_alpha free parameter indicating the constant stress region of the wave boundary layer.

sg_brlmin minimum allowed bottom roughness length.

sg_nu kinematic viscosity of seawater.

climat.h This file contains the climatology arrays.

tclm tracer climatology at the current time-step.

nudgcof time-scale (1/sec) coefficients for nudging towards climatology.

tclima work array containing two time levels of the tracer climatology read from disk.

ttclm time of the tracer climatology fields read from disk.

forces.h This file contains the surface and bottom forcing arrays. It also contains work arrays used in time-averaging these fields when they are read from a file. The non-work arrays in this file are:

srflx kinematic surface shortwave solar radiation flux at ρ-points.

stflx kinematic surface flux of tracer type variables at ρ-points.

dqdt kinematic surface net heat flux sensitivity to the sea surface temperature.

sst sea surface temperature used when computing Q, called T_ref in §2.2.

sustr kinematic surface momentum flux (wind stress) in the ξ-direction at u-points.

svstr kinematic surface momentum flux (wind stress) in the η-direction at v-points.

btflx kinematic bottom flux of tracer type variables at ρ-points.

bustr kinematic bottom momentum flux (bottom stress) in the ξ-direction at u-points.

bvstr kinematic bottom momentum flux (bottom stress) in the η-direction at v-points.

wwag wind induced wave amplitude gridded data.

wwap wind induced wave amplitude point data.

wwdg wind induced wave direction (radians) gridded data.

wwdp wind induced wave direction (radians) point data.

wwpg wind induced wave period gridded data.

wwpp wind induced wave period point data.

grid.h This file contains the arrays for the vertical and horizontal grid information.

Cd_r first derivative of the $C(s)$ curves ($\partial C(s)/\partial s$) at vertical ρ-points.

Cd_w first derivative of the $C(s)$ curves ($\partial C(s)/\partial s$) at vertical w-points.

Cs_r set of s-curves used to stretch the vertical coordinate lines that follow the topography at vertical ρ-points.

Cs_w set of s-curves used to stretch the vertical coordinate lines that follow the topography at vertical w-points.

D total water column depth at ρ-points at current and previous short time levels.

Hz first s-derivative of the z-coordinate ($\partial z/\partial s$) at ρ-points.

Tcline width of surface or bottom boundary layer in which higher vertical resolution is required during stretching.

angler angle (radians) between ξ-axis and East at ρ-points.

ds non-dimensional vertical grid spacing (**ds**=1/**N**).

el length of domain in the η-direction.

h bottom depth at ρ-points.

hc s-coordinate parameter, **hc**=**min**(**hmin**,**Tcline**).

hmax maximum depth of bathymetry.

hmin minimum depth of bathymetry.

latr latitude (degrees North) at ρ-points.

lonr longitude (degrees East) at ρ-points.

ods reciprocal of vertical grid spacing (**ods**=1/**ds**).

sc_r s-coordinate independent variable, $[-1 < s < 0]$ at vertical ρ-points

sc_w s-coordinate independent variable, $[-1 < s < 0]$ at vertical w-points.

spherical logical switch indicating spherical grid configuration.

theta_s s-coordinate surface control parameter θ, $[0 < \theta < 20]$.

theta_b s-coordinate bottom control parameter b, $[0 < b < 1]$.

z_r actual depths at horizontal ρ-points and vertical ρ-points.

z_w actual depths at horizontal ρ-points and vertical w-points.

xl length of domain in the ξ-direction.

xp x-coordinates at ψ-points.

xr x-coordinates at ρ-points.

yp y-coordinates at ψ-points.

yr y-coordinates at ρ-points.

iounits.h I/O filenames and flags for writing fields to history files.

Nlev number of levels to write out in history file.

Lev levels to write out in history file.

aparnam input assimilation parameters file name.

assname input assimilation file name.

avgname output averages file name.

clmname input climatology file name.

fltname output floats file name.

fposnam input initial floats positions file name.

frcname input forcing fields file name.

grdname input grid file name.

hisname output history file name.

ininame input initial conditions file name.

ispos i-index for station positions.

jspos j-index for station positions.

nstation number of output stations.

rstname output restart file name.

sposnam input station positions file name.

staname output station data file name.

stdinp unit number for standard input (often 5).

stdout unit number for standard output (often 6).

usrout unit number for generic user output.

usrname user input/output generic file name.

wrtAKS switch to write out vertical diffusion coefficient for salinity.

wrtAKT switch to write out vertical diffusion coefficient for potential temperature.

wrtAKV switch to write out vertical viscosity coefficient.

wrtHBL switch to write out depth of mixed layer.

wrtO switch to write out Ω vertical velocity.

wrtU switch to write out 3-D u-momentum component.

wrtT switch to write out tracer type variables.

wrtUBAR switch to write out 2-D u-momentum component.

wrtV switch to write out 3-D v-momentum component.

wrtVBAR switch to write out 2-D v-momentum component.

wrtW switch to write out w-momentum component.

wrtZ switch to write out free surface elevation.

mask.h The 2-D mask arrays.

rmask mask at ρ-points (0=Land, 1=Sea).

pmask slipperiness mask at ψ-points (0=Land, 1=Sea, 1-gamma2=boundary).

umask mask at u-points (0=Land, 1=Sea).

vmask mask at v-points (0=Land, 1=Sea).

metrics.h Horizontal grid metrics and their combinations.

Dmon_p compound term, Dm/n at ψ-points at current and previous short time levels.

Dnom_p compound term, Dm/n at ψ-points at current and previous short time levels.

Duon_u compound term, $D\overline{u}/n$ at u-points at current and previous short time levels.

Dvom_v compound term, $D\overline{v}/m$ at v-points at current and previous short time levels.

Huon_u compound term, $H_z u/n$ at u-points.

Hvom_v compound term, $H_z v/m$ at v-points.

Hzmon_p compound term, $H_z m/n$ at ψ-points.

Hznom_p compound term, $H_z n/m$ at ψ-points.

dmde η-derivative of inverse metric factor m, $\frac{\partial}{\partial \eta}(\frac{1}{m})$.

dndx ξ-derivative of inverse metric factor n, $\frac{\partial}{\partial \xi}(\frac{1}{n})$.

f Coriolis parameter.

fomn compound term, $f/(mn)$ at ρ-points.

pm coordinate transformation metric m associated with differential distances in ξ.

pmon_p compound term, m/n at ψ-points.

pmon_r compound term, m/n at ρ-points.

pmon_u compound term, m/n at u-points.

pn coordinate transformation metric n associated with differential distances in η.

pnom_p compound term, n/m at ψ-points.

pnom_r compound term, n/m at ρ-points.

pnom_v compound term, n/m at v-points.

mixing.h The horizontal and vertical viscosity/diffusion coefficients. Some of the vertical mixing schemes have additional variables for their internal use.

Akt spatially variable vertical mixing coefficient for tracers.

Akt_bak background vertical mixing coefficient for tracers.

Akv spatially variable vertical mixing coefficient for momentum.

Akv_bak background vertical mixing coefficient for momentum.

at1 time weight for current time level vertical mixing coefficients to avoid instability of the implicit scheme.

at2 time weight for previous time level vertical mixing coefficients to avoid instability of the implicit scheme.

rinavfl logical switch to activate the spatial averaging of gradient Richardson number.

tnu2 lateral Laplacian constant mixing coefficient for tracer type variables.

tnu4 lateral biharmonic constant mixing coefficient for tracer type variables.

uvnu2 lateral Laplacian constant mixing coefficient for momentum.

uvnu4 lateral biharmonic constant mixing coefficient for momentum.

ncscrum.h A plethora of variables referring to objects in the various I/O NetCDF files.

obc.h Arrays used in the open boundary conditions.

tebry eastern boundary condition data for tracer type variables.

tnbry northern boundary condition data for tracer type variables.

tsbry southern boundary condition data for tracer type variables.

tsbry western boundary condition data for tracer type variables.

uebry eastern boundary condition data for 3-D velocity component in the ξ-direction.

unbry northern boundary condition data for 3-D velocity component in the ξ-direction.

usbry southern boundary condition data for 3-D velocity component in the ξ-direction.

usbry western boundary condition data for 3-D velocity component in the ξ-direction.

vebry eastern boundary condition data for 3-D velocity component in the η-direction.

vnbry northern boundary condition data for 3-D velocity component in the η-direction.

vsbry southern boundary condition data for 3-D velocity component in the η-direction.

vsbry western boundary condition data for 3-D velocity component in the η-direction.

rhs.h The right-hand-side arrays.

rt right-hand-side of the tracer equations.

rtold right-hand-side of the tracer equations at previous two time levels.

ru right-hand-side of the 3-D u-momentum equation.

rubar right-hand-side of the 2-D u-momentum equation.

rufrc right-hand-side forcing term for the 2-D u-momentum equation.

ruold right-hand-side of the 3-D u-momentum equation at previous two long time levels.

rustr surface and bottom u-momentum stresses.

rv right-hand-side of the 3-D v-momentum equation.

rvbar right-hand-side of the 2-D v-momentum equation.

rvfrc right-hand-side forcing term for the 2-D v-momentum equation.

rvold right-hand-side of the 3-D v-momentum equation at previous two long time levels.

rvstr surface and bottom v-momentum stresses.

rzeta right-hand-side of the free surface equation.

scalars.h A large number of scalars of all sorts.

R0 background constant density anomaly used in linear equation of state.

Scoef saline contraction coefficient in linear equation of state.

S0 background salinity value used in analytic fields.

Tcoef thermal expansion coefficient in linear equation of state.

T0 background potential temperature value used in analytic fields.

abc1 constant factor (**abc1**=23/12) used in the 3rd-order Adams-Bashforth time stepping.

abc2 constant factor (**abc2**=16/12) used in the 3rd-order Adams-Bashforth time stepping.

abc3 constant factor (**abc3**=5/12) used in the 3rd-order Adams-Bashforth time stepping.

adv_ord number of iterations or passes through advection operator in the Smolarkiewicz scheme.

dstart time stamp assigned to model initialization (usually a Calendar day, such as modified Julian Day).

dt size of 3-D primitive equations timestep.

dtfast size of 2-D primitive equations timestep.

dtods timestep factor, **dtods**=**dt**/**ds**.

dtods2 timestep factor, **dtods2**=**dt**/(**ds***__ds__).

dtsfast size of 2-D primitive equations timestep between consecutive steps.

dummy1 scalar dummy variable.

dummy2 scalar dummy variable.

dummy3 scalar dummy variable.

fracdt timestep factor, **fracdt**=2/**dt**.

g acceleration due to gravity.

gorho0 gravity divided by mean density, **gorho0**=g/ρ_o.

gamma2 slipperiness variable, either 1.0 (free slip) or -1.0 (no slip).

ibcLapQ switch associated with boundary conditions for the Laplacian of Mellor-Yamada turbulent energy variables.

ibcLapT switch associated with boundary conditions for the Laplacian of tracer type variables.

ibcLapU switch associated with boundary conditions for the Laplacian of the velocity component in the ξ-direction.

ibcLapV switch associated with boundary conditions for the Laplacian of the velocity component in the η-direction.

ibcQ switch associated with boundary conditions for the Mellor-Yamada turbulent energy variables.

ibcT switch associated with boundary conditions for the tracer type variables.

ibcTa switch associated with boundary conditions for the tracer type variables in Smolarkiewicz advection scheme.

ibcU switch associated with boundary conditions for the velocity component in the ξ-direction.

ibcUa switch associated with boundary conditions for the anti-diffusion velocity in the ξ-direction.

ibcUavg switch associated with boundary conditions for the time-averaged velocity component in the ξ-direction.

ibcV switch associated with boundary conditions for the velocity component in the η-direction.

ibcVa switch associated with boundary conditions for the anti-diffusion velocity in the η-direction.

ibcVavg switch associated with boundary conditions for the time-averaged velocity component in the η-direction.

ibcW switch associated with boundary conditions for the velocity component in the s-direction.

ibcWa switch associated with boundary conditions for the anti-diffusion velocity in the s-direction.

ibcZ switch associated with boundary conditions for the free surface elevation.

ibcZavg switch associated with boundary conditions for the time-averaged free surface elevation.

icavg current number of time-records accumulated in output time-averaged arrays.

iic timestep counter for 3-D primitive equations.

iif timestep counter for 2-D primitive equations.

jjf timestep counter component (1 or 2) used in the trapezoidal timestep correction of the 2-D primitive equations.

knew pointer to current short time level for variables associated with 2-D primitive equations.

kold pointer to previous short time level for variables associated with 2-D primitive equations.

krhs pointer to the time level used to calculate the right-hand term in the 2-D primitive equations.

kstp pointer to the time level to which the current changes are added in the 2-D primitive equations.

lambda Crank-Nicolson scheme weight for upper and lower time levels (usually, **lambda**=0.5).

lcycle logical switch used to recycle time records in output restart file. If **.true.**, only the latest two restart time records are maintained. If **.false.**, all restart field are saved every **nrst** timesteps without recycling.

ldefhis logical switch used to create the history file. If **.true.**, a new history file is created. If **.false.**, data is appended to an existing history file.

levbfrc shallowest level to apply bottom momentum stress as a bodyforce.

levsfrc deepest level to apply surface momentum stress as a bodyforce.

lnew pointer to future $(n+1)$ long time level for variables associated with the time averaging.

lnow pointer to current (n) long time level for variables associated with the time averaging.

lold pointer to previous $(n-1)$ long time level for variables associated with the time averaging.

lwrthis logical switch to activate the writing of fields to the SCRUM history file.

mnew pointer to current long time level for variables associated with turbulent energy equations.

mold pointer to previous long time level for variables associated with turbulent energy equations.

mrhs pointer to the time level used to calculate the right-hand term in the turbulent energy equations.

mstp pointer to the time level to which the current changes are added in the turbulent energy equations.

navg number of timesteps between storage of time-averaged fields.

ndtfast number of timesteps for 2-D equations between each **dt**.

ninfo number of timesteps between print of single line information to standard output.

nmix_en number of timesteps between computations of isopycnal slopes used in the rotated mixing tensor.

nrecavg number of time records written in averages file.

nrechis number of time records written in history file.

nrecrst number of time records written in restart file.

nrecsta number of time records written in stations file.

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

nrst number of timesteps between storage of restart fields.

nsta number of timesteps between storage of station data.

ntimes ending timesteps in evolving the 3-D primitive equations in the current run.

ntsavg starting timestep for accumulation of output time-averaged fields.

ntstart starting timestep in evolving the 3-D primitive equations; usually 1, if not a restart run.

nwrt number of timesteps between writing of fields into output history file.

qdtfac timestep factor for turbulent energy equations.

rdrg linear bottom drag coefficient.

rdrg2 quadratic bottom drag coefficient.

rho0 mean density ρ_o.

rnudg inverse time scale (days) of the nudging towards climatology.

rstflag logical switch which indicates whether or not the model is being restarted.

tdays SCRUM time since initialization (days).

tfacfast time stepping variable for the leap-frog trapezoidal correction of the 2-D primitive equations.

trapfast logical switch activated during the leap-frog trapezoidal timestep correction of the 2-D primitive equations.

wall1 logical switch for side 1 ($i = 1$), **.true.** if it is a wall, **.false.** if it is open.

wall2 logical switch for side 2 ($j = 1$), **.true.** if it is a wall, **.false.** if it is open.

wall3 logical switch for side 3 ($i = \mathbf{L}$), **.true.** if it is a wall, **.false.** if it is open.

wall4 logical switch for side 4 ($j = \mathbf{M}$), **.true.** if it is a wall, **.false.** if it is open.

strings.h

Coptions activated C-preprocessing options.

title title of model run.

work.h Work arrays.

a2d – h2d utility 2-D arrays used as temporary storage.

a3d – h3d utility 3-D arrays used as temporary storage.

4.6 Statement functions

We use the following statement functions (inline functions) throughout the code to more easily keep track of factors of two.

avg.h

av2 average of the two arguments.

av4 average of the four arguments.

Chapter 5

Support Programs for Initialization

5.1 Grid generation

On startup, SCRUM either reads a NetCDF file or calls **ana_grid** to find the location of the grid points, the grid metrics, the bathymetry, the land/sea mask, and the Coriolis parameter f. If you won't be using **ana_grid**, the grid file must be generated before SCRUM can be run, either with **ezgrid** or with the programs in **gridpak**. The version of **ezgrid** which produces a NetCDF file is available in

```
ftp://ahab.rutgers.edu/pub/gridpak/ezgrid.shar
```

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 **cppdefs.h**. If **UPWELLING** is not defined then the important parameters are:

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, $f = f_o + \beta y$.

In either case you will have to also set the name of the gridfile, **grdname**, near the top of the **ezgrid.F** file. Once these parameters are set to your chosen values, compile and run it:

```
make ezgrid
ezgrid
```

This should create a binary NetCDF file called **grdname**.

5.1.2 gridpak

SCRUM 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

SCRUM now supports the masking of land areas, for which it requires some new input arrays. These arrays are read from the grid NetCDF file—there is no current option for creating an analytic mask. The mask is defined on ρ-points; see Fig. 5.1 for an example of a small domain with an isolated island and a promontory adjacent to the boundary. There

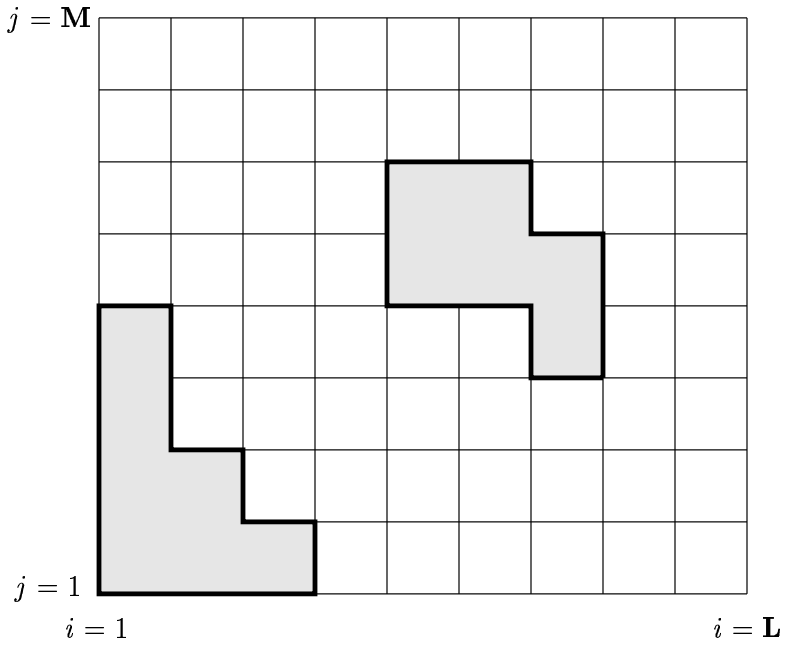

Figure 5.1: Small grid with masked regions

are also arrays for the mask on u-points, v-points, and ψ-points which are derived from the ρ-point mask. The ψ-point mask also depends on the free-slip/no-slip option chosen as described in §3.2.

The programs in **gridpak** find the ρ-point mask based on the bathymetry dataset. Elevations at or above sea level are assumed to be in the land mask. You may choose to edit this mask, so Hernan Arango has written a **Matlab** tool called **scrum_mask**. It is an interactive tool which requires **Matlab** as well as **mexcdf** for reading and writing NetCDF files from **Matlab**. It is available in

`ftp://ahab.rutgers.edu/pub/scrum/matlab/mask/`
`ftp://ahab.rutgers.edu/pub/scrum/tars/scrum3_matlab.tar.gz`

and includes a **README** file. An example of its use is shown in Fig. 5.2. This represents the same mask as in Fig. 5.1, with a circle for each ρ-point, including the boundary "image" points. The darker circles are land. Notice that I have made the "image" points have the same mask value as the points they mirror.

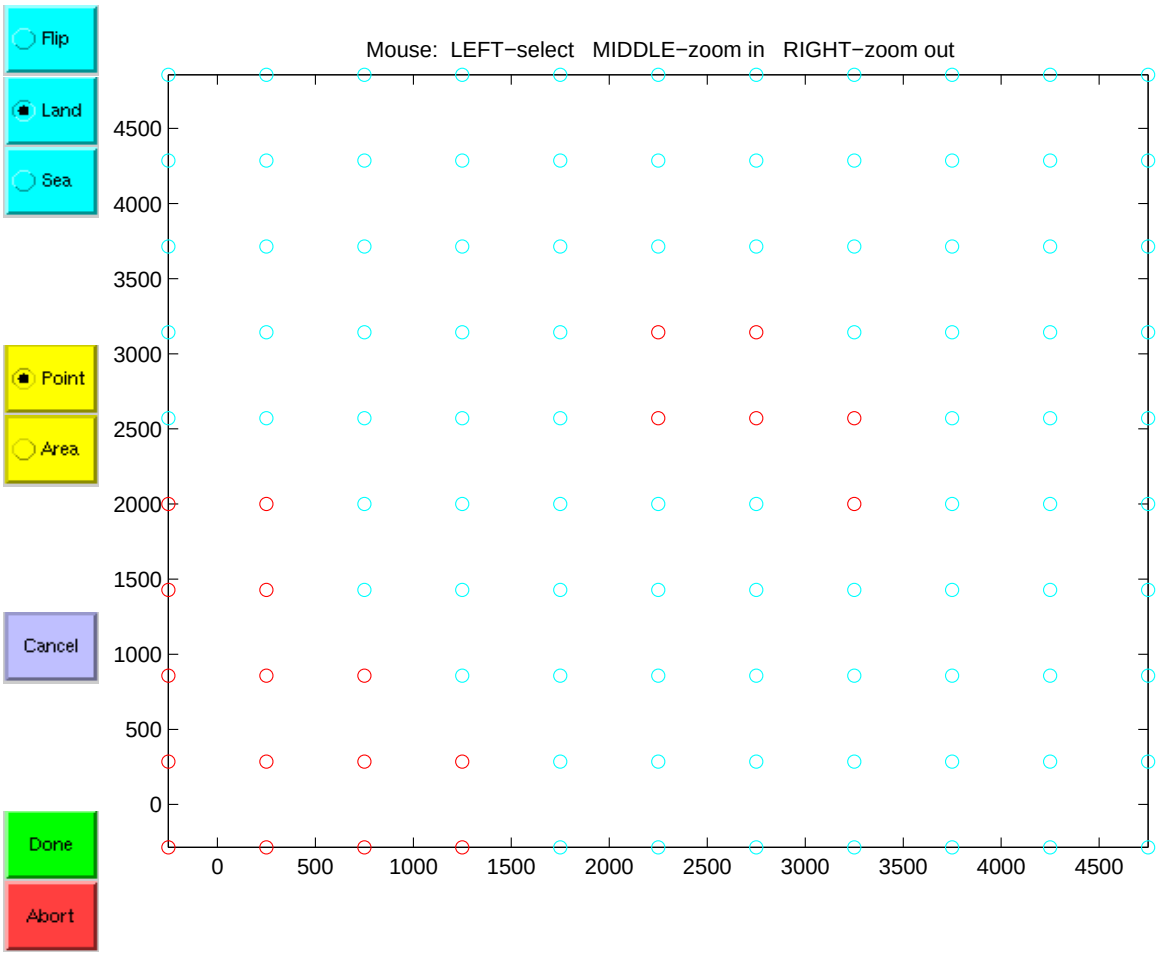

Figure 5.2: The **scrum_mask** program in action

5.3 Objective Analysis

[This section was contributed by Hernan Arango.]

The objective analysis (**oa**) package described here can be used to prepare initial, climatology, update, and forcing fields for SCRUM. It maps oceanographic and atmospheric data to a specified application grid. Currently, it processes the following fields: *in situ* temperature, potential temperature, *in situ* density anomaly, salinity, sigma-t, sound speed, dynamic

height, surface net heat flux (Q), surface freshwater flux, precipitation rate, evaporation rate, incoming solar shortwave radiation, surface momentum (wind) stress components, sea surface temperature (SST), and surface net heat flux sensitivity to SST ($\partial Q/\partial \mathrm{SST}$).

This **oa** package is derived from an earlier program which Hernan Arango and Carlos Lozano wrote at Harvard University in 1993. The basic algorithm used by this package is described in Carter and Robinson [6]. A comprehensive description of this methodology can also be found in Gadin [9], Bretherton et al. [5], McWilliams et al. [21], Daley [7], Bennett [4], and others.

Given observations $s_i = s(\mathbf{x}_i, t_i)$ at location $\mathbf{x}_i, t_i, i = 1, \ldots N$ an estimate ϕ_E of a scalar ϕ is derived for location $\mathbf{x}$ and time t. A linear unbiased estimate is given by:

$$\phi_E(\mathbf{x}, t) = \overline{\phi}(\mathbf{x}, t) + \sum_i w_i (s_i - \overline{s_i})$$

for arbitrary w_i since $\overline{\phi_E} = \overline{\phi}$. The associated variance of error is:

$$\overline{e^2(w)} = \overline{(\phi - \phi_E(w))^2}$$

with $w = (w_1, \ldots w_N)$. The overbar denotes an expected or ensamble mean value. The minimizer w_*:

$$\overline{e^2(w_*)} \leq \overline{e^2(w)}$$

is

$$w_* = \mathbf{A}^{-1} p$$

with minimum error variance (Gauss-Markov):

$$\overline{e}_*^2 = \overline{e^2(w_*)} = \overline{(\phi - \overline{\phi})^2} - p' \mathbf{A}^{-1} p$$

Here, for conveniance, matrix notation has been used. $s = [s_1, \ldots s_N]$ is a column correlation vector, $p = (\phi - \overline{\phi})(s - \overline{s})$, and $\mathbf{A}$ is the covariance matrix:

$$\mathbf{A} = \overline{(s - \overline{s})(s - \overline{s})'}$$

where the prime denotes a transpose.

Notice that $\mathbf{A}$ is symmetric. In what follows, excluding pathological cases, $\mathbf{A}$ is assumed to be positive definite. The best linear estimate ϕ_* is then:

$$\phi_*(\mathbf{x}, t) = \overline{\phi(\mathbf{x}, t)} + p' \mathbf{A}^{-1} (s - \overline{s})$$

with error $\overline{e_*^2}$.

The essential information required is statistical; namely the spatial-temporal mean of the scalar and observations, the covariance between observations, and the covariance between the scalar and the observations.

The observations can be of different types, and different from the scalar which you are trying to find. Their usefulness is measured by the fractional reduction of error:

$$\frac{p' \mathbf{A}^{-1} p}{\overline{(\phi - \overline{\phi})^2}}$$

In this package it is assumed that the covariance of the scalar is homogeneous in space and homogeneous and isotropic in time:

$$C\left((\mathbf{x_1}, t_1), (\mathbf{x_2}, t_2)\right) = C\left(\mathbf{x_1} - \mathbf{x_2}, |t_2 - t_1|\right)$$

and errors at two different locations and times are uncorrelated:

$$E\left((\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2)\right) = E(\mathbf{x}_1, t_1)\delta(\mathbf{x_2} - \mathbf{x_1})\delta(t_2 - t_1).$$

Currently, an analytical, isotropic, Gaussian correlation function is assumed:

$$C(\mathbf{x_1} - \mathbf{x_2}, |t_2 - t_1|) = \mathcal{C}(|\mathbf{x_1} - \mathbf{x_2}|, |t_2 - t_1|)$$

with

$$\mathcal{C}(\mathbf{r}, \tau) = \exp\left[-\left(\frac{\tau}{\tau_o}\right)^2\right] G(\mathbf{r})$$

$$G(\mathbf{r}) = \left[1 - \left(\frac{\mathbf{r}}{a}\right)^2\right] \exp\left[-\left(\frac{\mathbf{r}}{b}\right)^2\right]$$

where τ_o is the time decorrelation scale, a is the zero crossing distance, and b is the spatial decorrelation scale.

This package uses a local solution to the **oa** equations. That is, only **nnce** influsential observations are considered at each mapped grid point. This method is practical because it avoids inverting large matrices when the number of observations is large. Observations that are too far apart in space and time from the mapped point contribute very little to the estimate, as one might expect.

It is available from:

`ftp://ahab.rutgers.edu/pub/scrum/tars/scrum3_oa.tar.gz`

and includes a **README** file.

5.4 Forcing fields

There are options for calling either **ana_smflux** or **get_smflux** to get the surface momentum forcing. If you do not have an analytic formulation for this, you will have to create a NetCDF forcing file which contains the surface momentum fluxes. It can either contain one point value or a 2-D field of values. Likewise, the field can be constant in time or contain values for a series of times. It is even possible to have a limited number of fields which get cycled over in time. For instance, you can provide 12 monthly mean fields and tell it to cycle over these in a multi-year run.

The other forcing fields are treated in the same way and are also contained in the NetCDF forcing file. These include surface and bottom heat and salt fluxes, the $\partial Q/\partial T$ and $T_{\rm ref}$ terms from §2.2, the incoming shortwave radiation use by the Large et al. mixing scheme, and the wave information used by the Styles and Glenn bottom boundary layer.

An example program which creates the forcing NetCDF file is provided by the files in

`ftp://ahab.rutgers/pub/scrum/forcing`

This program reads a file produced by the **oa** package.

5.4.1 Initial and climatology fields

The model will either read its initial fields from a NetCDF file or it will compute them in **analytical.F**. If it is not computing them, the routine **get_initial** will read a history file or a file produced by the **initial** program. This program in turn is expecting to read the output of the **oa** program. The **initial** program is in

```
ftp://ahab.rutgers/pub/scrum/initial
```

The model has the option of reading in 3-D climatology fields from a climate NetCDF file. This file contains the 3-D climatologies for the tracers, perhaps at a number of times. The subroutine **get_clima** will read this file and do any necessary time interpolations. The climate file is also produced by the **initial** program. The climatology could also be used for the boundary conditions, both for the tracer values on inflow or for prescribed boundary conditions. In this case it would make more sense to only store the 2-D arrays. We do not yet have the software for handling these 2-D arrays, but it would be a straightforward modification to the **initial** program.

Chapter 6

Configuring SCRUM for a Specific Application

This chapter describes the parts of SCRUM for which the user is responsible when configuring it for a given application. Section 6.1 describes the process in a generic fashion while §6.2 and §6.3 step through the application of SCRUM to upwelling/downwelling and wind-driven North Atlantic problems, respectively. As distributed, SCRUM is ready to run quite a few examples, where the C preprocessor flags determine which is to be executed. Some of these examples are described as noted and some will be described in Haidvogel and Beckmann [12]. The examples are:

BASIN This is a rectangular, flat-bottomed basin with double-gyre wind forcing. When run, it produces a western boundary current flowing into a central "Gulf Stream" which goes unstable and generates eddies. The goal is to run adiabatically to study the homogenization of potential vorticity. It takes a long time and caused difficulties for SPEM 3 so we call it the "Big Bad Basin".

CANYON_A The canyon is a periodic channel with a steep shelf along one wall, where the shelf contains a steep canyon. There is a periodic forcing which causes the water to oscillate along the channel. The rotation and the shelf lead to non-zero mean flows, especially near the canyon. Version A is homogeneous and can be executed with a 2-D model. See Haidvogel and Beckmann [11] for a description of the canyon problems and the gravitational adjustment problem.

CANYON_B This is like Canyon A, except that it is stratified.

GRAV_ADJ The gravitational adjustment problem takes place in a long narrow domain which is initialized with dense water at one end and light water at the other. At time zero, the water is released and it generates two propagating fronts as the light water rushes to fill the top and the dense water rushes to fill the bottom. This configuration was used to test various advection schemes.

OVERFLOW This configuration is similar to the GRAV_ADJ problem, but is initialized with dense water in the shallow part of a domain with a sloping bottom.

SEAMOUNT The seamount test was used to test the pressure gradient errors. It has an idealized seamount in a periodic channel. See Beckmann and Haidvogel [3] and McCalpin [20] for more information.

TASMAN_SEA This was used to test the masking in SPEM, especially the streamfunction solve around an island. It is a square, flat-bottomed box with one island and wind forcing to produce three asymmetric gyres. It was designed for an early draft of Wilkin et al. [37].

UPWELLING The upwelling/downwelling example was contributed by Anthony Macks and Jason Middleton [19] and consists of a periodic channel with shelves on each side. There is along-channel wind forcing and the Coriolis term leads to upwelling on one side and downwelling on the other side. If you run it for several days, you end up with dense water over light water.

The input files for the SCRUM examples are included in the file:

```
ftp://ahab.rutgers.edu/pub/scrum/tars/scrum3_examples.tar.gz
```

This file also contains sample output NetCDF files and plot files and is quite large.

6.1 Configuring SCRUM

The three main files you need to change in SCRUM are **scrum.in**, **cppdefs.h**, and **analytical.F**. These provide the input, set the options you want, and provide analytic formulas for various fields, respectively. If more realistic fields are desired, you will have to provide other input files as well, for instance for the grid and the wind forcing.

6.1.1 cppdefs.h and checkdefs.F

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 **cppdefs.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
```

We use the first of these.

When configuring SCRUM for your problem, it is recommended that you add a new **cpp** variable for it. New **cpp** variables can be added to **cppdefs.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 simple ice test is called **MMS_BOX**:

```
#ifdef MMS_BOX
# define EW_PERIODIC
# define NS_PERIODIC
# define ICE
      :
#endif  /* MMS_BOX */
```

If it becomes necessary to update to a newer version of SCRUM, it is simple to find the parts of the code which belong to the Arctic version and copy them to the new SCRUM (if you get behind on the patches).

For each new **cpp** variable, it is recommended that you also add the appropriate code to **checkdefs.F**, such as:

```
#ifdef ICE
      write(stdout,20) 'ICE',
     &                 'Coupled sea-ice model.'
      is=lenstr(Coptions)+1
      Coptions(is:is+4)=' ICE,'
#endif /* ICE */
```

Note that the number "4" on the **Coptions** line must be set according to the length of the string you are adding. In this case 4 is for "ICE,", including the comma.

6.1.2 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 in **param.h** which specify the grid size and one parameter for the number of tracers:

L	Number of finite-difference points in ξ.
M	Number of finite-difference points in η.
N	Number of finite-difference points in the vertical.
NT	Number of tracers.

There are no constraints on these except $\mathbf{L} \geq 2$, $\mathbf{M} \geq 2$, $\mathbf{N} \geq 2$ and $\mathbf{NT} \geq 1$. **L** and **M** should be at least 3 if the domain is periodic in that direction.

6.1.3 x, y grid

The subroutine **get_grid** or **ana_grid** is called by **initial** to set the grid arrays, the bathymetry, and the Coriolis parameter. Most of the simple test problems have their grid information specified in **ana_grid** in the file **analytical.F**. More realistic problems require a NetCDF grid file, produced by the grid generation programs described in Wilkin and Hedstrom [36]. The variables which are read by **get_grid** are:

xl, el, spherical, f, h, pm, pn, x_rho, y_rho, lon_rho, lat_rho, angle.

If the grid is curved, **get_grid** will also read:

dndx, dmde.

6.1.4 ξ, η grid

Before providing initial conditions and boundary conditions, the user must understand the model grid. The fields are laid out on an Arakawa C grid as in Fig. 3.1. The overall grid is shown in Fig. 6.1. The thick outer line shows the position of the model boundary. The points inside this boundary are those which are advanced in time using the model physics. The points on the boundary and those on the outside must be supplied by the boundary conditions.

The three-dimensional model fields are carried in three-dimensional arrays, except the tracers where the fourth array index tells which tracer is being referred to. For instance, **itemp** $= 1$ refers to potential temperature while **isalt** $= 2$ refers to salinity. The integers i, j, and k are used throughout the model to index the three spatial dimensions:

i Index variable for the ξ-direction.
j Index variable for the η-direction.
k Index variable for the σ-direction. $k = 1$ refers to the bottom while $k =$ **N** refers to the surface.

6.1.5 Initial conditions

The initial values for the model fields are provided by either **ana_initial** or **get_initial**. **get_initial** is also used to read a restart file if the model is being restarted from a previous run.

Also in **initial**, **rho_eos** is called to initialize the density field. **rho_eos** in turn calls **ana_meanRHO** to initialize the **rhobar** array. **rhobar** is a function of z only and should be more or less the horizontal average of the density field. It is subtracted from ρ, before ρ is vertically integrated in **prsgrd**, to reduce the errors in the pressure gradient terms.

The climatology fields also require appropriate values if they are to be used, and are provided by **ana_clima** or **get_clima**.

6.1.6 Equation of state

The equation of state is defined in the subroutine **rho_eos**. Two versions are provided in SCRUM: a nonlinear $\rho = \rho(T, S, z)$ from Jackett and McDougall [16] and a linear $\rho(T, S)$. The linear form is $\rho =$ **R0** $+$ **Tcoef** $\cdot\, T +$ **Scoef** $\cdot\, S$ or $\rho =$ **R0** $+$ **Tcoef** $\cdot\, T$, depending on whether or not **SALINITY** is defined. Specify which equation of state you would like to use by setting the **NONLIN_EOS** C preprocessor flag in **cppdefs.h**. The linear coefficients **R0**, **Tcoef** and **Scoef** are set in **scrum.in**.

6.1.7 Boundary conditions

The horizontal boundary conditions are provided by the subroutines in **bcs3d** and **bcs2d**. They are called every timestep and provide the boundary values for the fields $u, v, \overline{u}, \overline{v}, T, S$ and ζ. They are currently configured for a closed basin, a periodic channel, or a doubly periodic domain. There are also a number of options for open boundaries. You will have to try them or ask an expert which is the best for your application.

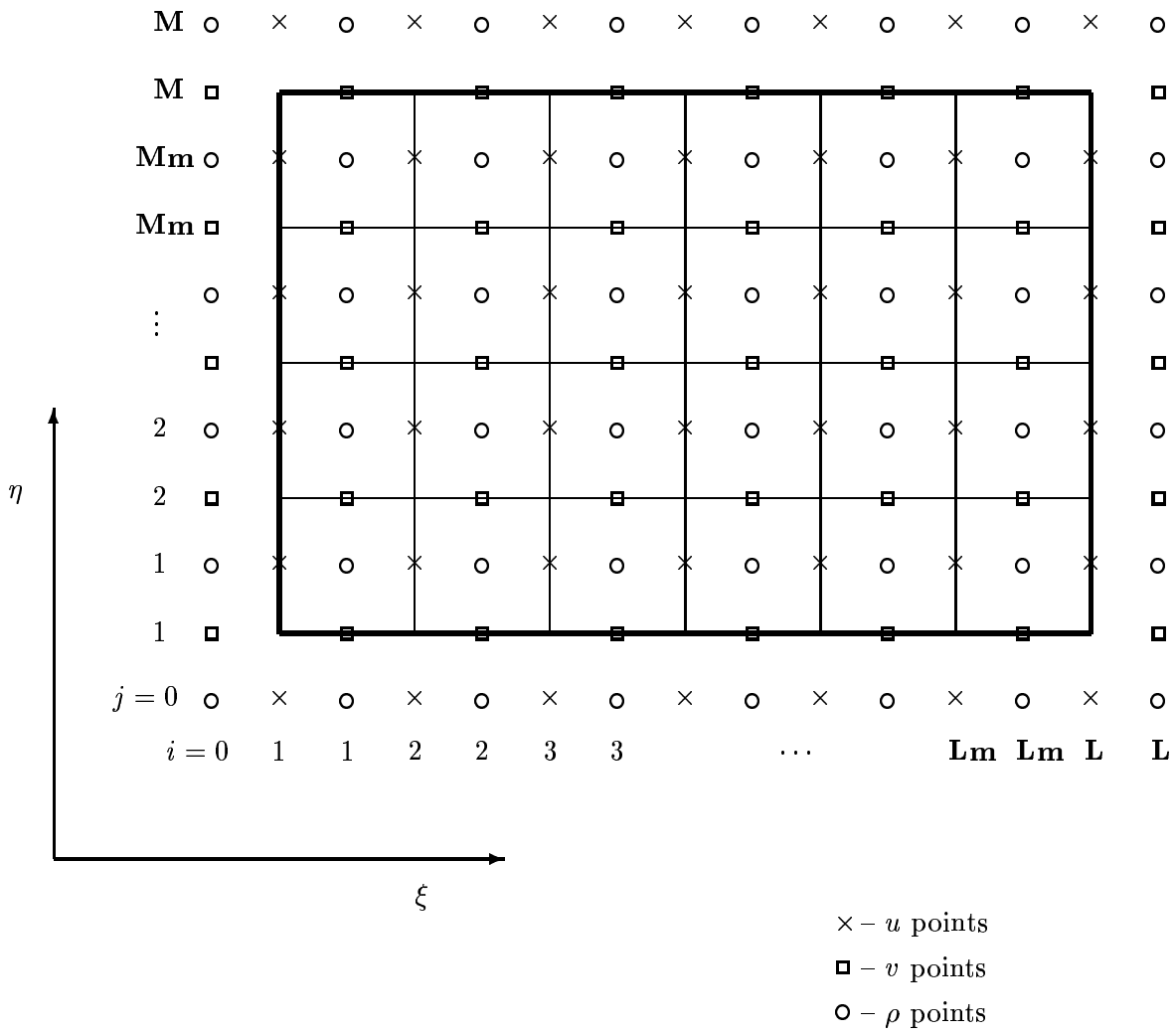

Figure 6.1: The whole grid.

6.1.8 Model forcing

(a) Winds and thermal fluxes

There are two different ways to apply a wind forcing: as a surface momentum flux in the vertical viscosity term, or as a body force over the upper water column. In the past, our vertical resolution was relatively coarse and the vertical viscosity would have to have been unreasonably large for us to resolve the surface Ekman layer. If that is your situation, define **BODYFORCE** in **cppdefs.h** and provide a value for **levsfrc** in **scrum.in**. The forcing is applied over the levels from **levsfrc** to **N**. The above caution about vertical resolution also applies to the surface fluxes of T and S, although **BODYFORCE** only refers to wind stress, not the surface tracer fluxes.

More recently, we have been setting the vertical s-coordinate parameters to retain some resolution near the surface and to apply the fluxes as boundary conditions to the vertical viscosity/diffusivity. In either case, the surface and bottom fluxes are either defined analytically or read from the forcing file. You must either edit the appropriate parts of **analytical.F** or create a NetCDF forcing file in the format expected by **get_smflux**, **get_stflux** and their friends. Note that it is quite common to put the wind stress into the forcing file while having an analytic bottom stress.

(b) Climatology

One way to force the model is via a nudging to the tracer climatologies. This was is used in the North Atlantic simulations in sponge layers along the northern and southern boundaries. Set the climatologies in **ana_clima** or in a file read by **get_clima**, set **NUDGING** in **cppdefs.h** and also set the array **nudgcof** in **initial.F**.

6.1.9 scrum.in

SCRUM expects to read a number of variables on standard in. It is easiest to prepare an input file and then run SCRUM as:

```
scrum < scrum.in > scrum.out &
```

The input is organized as pairs of lines, the first with a number and then some text which is ignored, the second with the values for the set of variables for which the first line provides the key. The pairs of lines can be in any order but are usually sorted numerically. The number 99 signals the end of these pairs and the rest of the input file contains comments for the user. The input pairs are as follows:

1 Time-stepping parameters.

ntimes Number of timesteps to evolve the 3-D equations in the current run. This is actually the total number, including any previous segments of the same run. For instance, if you already did a three-month run and wish to continue for another three months, set **ntimes** to the number of steps needed for six months. If you don't like this and would prefer to have the behavior of the SPEM variable **ntmes**, modify **main.F** so that:

```
do iic=ntstart,ntimes
```

becomes

```
do iic=ntstart,ntimes-1+ntstart
```

dt Timestep in seconds for the 3-D equations.

ndtfast Number of timesteps for the 2-D equations to be executed each **dt**.

2 Input/Output parameters. SCRUM has several possible output files. The output files include a restart file, a history file, an averages file, and a station file. The restart file often contains only two records with the older record being overwritten during the next write. The history file can contain a subset of the restart fields, for instance just the surface elevation and the surface temperature. The averages file contains time-averages of the model fields, for instance montly means, or yearly means, depending on **navg**. The station file contains timeseries for specified points, possibly quite frequently since each record is small.

nrrec Record number of the restart file to read as the initial conditions.

nrst Number of timesteps between writing of restart fields.

nwrt Number of timesteps between writing fields into the history file.

ntsavg Starting timestep for the accumulation of output time-averaged data. For instance, you might want to average over the last day of a thirty-day run.

navg Number of timesteps between writing time-averaged data into the averages file.

nsta Number of timesteps between writing data into stations file.

ninfo Number of timesteps between printing a single line of diagnostic information to the standard output.

ldefhis Logical switch used to create the history file. If **.true.**, a new history file is created. If **.false.** and **nlev** > 0, data is appended to an existing history file.

lcycle Logical flag used to recycle time records in the restart file. If **.true.**, only the latest two restart time records are retained. If **.false.**, all restart fields are saved every **nrst** timesteps without recycling.

3 Laplacian horizontal mixing of tracers.

tnu2 Constant mixing coefficient for the horizontal Laplacian diffusion of each tracer. A value is expected for each of the **NT** tracers.

4 Biharmonic horizontal mixing of tracers.

tnu4 Constant mixing coefficient for the horizontal biharmonic diffusion of each tracer. A value is expected for each of the **NT** tracers.

5 Horizontal viscosity coefficients.

uvnu2 Constant mixing coefficient for the horizontal Laplacian viscosity.

uvnu4 Constant mixing coefficient for the horizontal biharmonic viscosity.

6 Vertical mixing coefficients for tracers.

akt_bak Background vertical mixing coefficient for the tracers. A value is expected for each of the **NT** tracers.

7 Vertical mixing coefficient for momentum.

akv_bak Background vertical mixing coefficient for momentum.

8 Mellor-Yamada Level 2.5 parameters.

akq_bak Background vertical mixing coefficient for turbulent kinetic energy.

q2nu2 Constant mixing coefficient for the horizontal Laplacian diffusion of turbulent kinetic energy.

q2nu4 Constant mixing coefficient for the horizontal biharmonic diffusion of turbulent kinetic energy.

9 Bottom drag coefficients.

rdrg Linear bottom drag coefficient.

rdrg2 Quadratic bottom drag coefficient.

10 Various parameters.

nmix_en Number of timesteps between computations of isopycnal slopes used in the rotated mixing tensor.

adv_ord Order of advection scheme when using Smolarkiewicz advection. A value of **adv_ord** = 2 is recommended to suppress the diffusive nature of the "upwind" scheme. A value of **adv_ord** = 1 will yield the standard "upwind" advection.

levsfrc Deepest level to apply surface momentum stresses as a bodyforce. Used when the C-preprocessor option BODYFORCE is defined.

levbfrc Shallowest level to apply bottom momemtum stresses as a bodyforce. Used when the C-preprocessor option BODYFORCE is defined.

11 Vertical s-coordinates parameters.

theta_s s-coordinate surface control parameter, $[0 < \mathbf{theta\_s} < 20]$.

theta_b s-coordinate bottom control parameter, $[0 < \mathbf{theta\_b} < 1]$.

Tcline Width of the surface or bottom boundary layer in which higher vertical resolution is required during stretching.

WARNING: Users need to experiment with these parameters. We have found out that the model goes unstable with high values of **theta_s**. With steep and very tall topography, it is recommended that you use **theta_s** ≤ 3.0.

12 Mean Density and time stamp.

rho0 Mean density used in the Boussinesq approximation.

dstart Time stamp assigned to model initialization (days). Usually a Calendar linear coordinate, like modified Julian day. For example:

dstart = 10200 corresponds to May 1, 1996

It is called modified Julian day because an offset of 2440000 needs to be added.

rnudg Time scale (days) of nudging towards climatology at the interior and at the boundaries.

13 Linear equation of state parameters.

R0 Background density value used in the linear equation of state.

T0 Background potential temperature constant used in **analytical.F**.

S0 Background salinity constant used in **analytical.F**.

Tcoef Thermal expansion coefficient in the linear equation of state.

Scoef Saline contraction coefficient in the linear equation of state.

14 Slipperiness parameters.

gamma2 Slipperiness variable, either 1.0 (free slip) or -1.0 (no slip).

wall1 Logical switch for side 1 ($i = 1$), **.true.** if it is a wall, **.false.** if it is open.

wall2 Logical switch for side 2 ($j = 1$), **.true.** if it is a wall, **.false.** if it is open.

wall3 Logical switch for side 3 ($i = \mathbf{L}$), **.true.** if it is a wall, **.false.** if it is open.

wall4 Logical switch for side 4 ($j = \mathbf{M}$), **.true.** if it is a wall, **.false.** if it is open.

15 Logical switches to activate the writing of fields associated with the momentum equations into the NetCDF history file:

wrtU Write out 3-D u-velocity component.

wrtV Write out 3-D v-velocity component.

wrtW Write out 3-D w-velocity component.

wrtO Write out 3-D Ω vertical velocity.

wrtUBAR Write out 2-D u-velocity component.

wrtVBAR Write out 2-D v-velocity component.

wrtZ Write out free-surface.

16 Logical switches to activate the writing of fields associated with the tracer equations into the NetCDF history file. A value is expected for each of the **NT** tracers.

wrtT Write out tracer type variables: potential temperature, salinity, etc.

17 Logical switches to activate the writing of other fields into the NetCDF history file:

wrtRHO Write out density anomaly.

wrtAKV Write out vertical viscosity coefficient.

wrtAKT Write out vertical diffusion coefficient for temperature.

wrtAKS Write out vertical diffusion coefficient for salinity.

wrtHBL Write out depth of the planetary boundary layer.

18 Number and Levels to output:

nlev Number of levels to write out to the history file for each activated 3-D field. If **nlev** < 0, all model levels are written out. IF **nlev** $= 0$, the history file will not be created.

lev If **nlev** > 0, levels to write out to the history file. **nlev** values are expected:

$$1 \leq \mathbf{lev}(1 : \mathbf{nlev}) \leq \mathbf{N}$$

Enter values in ascending numerical order.

19 String with a maximum of eighty characters.

title Title of the model run.

20 String with a maximum of eighty characters.

rstname Output restart file name (NetCDF).

21 String with a maximum of eighty characters.

hisname Output history file name (NetCDF).

22 String with a maximum of eighty characters.

avgname Name of the file for the averaged model fields (NetCDF).

23 String with a maximum of eighty characters.

staname Name of the file for the station output (NetCDF).

24 String with a maximum of eighty characters.

fltname Name of the file containing the float output (NetCDF).

25 String with a maximum of eighty characters.

grdname Name of the file containing the grid data (NetCDF).

26 String with a maximum of eighty characters.

ininame Name of the file containing the initial conditions. It can be a SCRUM restart file (NetCDF).

27 String with a maximum of eighty characters.

frcname Name of the file containing the forcing fields (NetCDF).

28 String with a maximum of eighty characters.

clmname Name of the file containing the climatology fields (NetCDF).

29 String with a maximum of eighty characters.

assname Name of the file containing the assimilation fields (NetCDF).

30 String with a maximum of eighty characters.

aparnam Name of the file containing the assimilation parameters (ASCII).

31 String with a maximum of eighty characters.

sposnam Name of the file containing the stations positions (ASCII).

32 String with a maximum of eighty characters.

fposnam Name of the file containing the initial drifter positions (ASCII).

33 String with a maximum of eighty characters.

usrname User's generic input file name.

An example input file without the trailing comments is:

```
1     NTIMES,     DT (s),    NDTFAST
       1800       240.d0       20
2     NRREC,   NRST,   NWRT,  NTSAVG,   NAVG,   NSTA,  NINFO,  LDEFHIS,  LCYCLE
        0       360     360      1       360      1      1        T        T
3     TNU2[1:NT]  (m^2/s)
      5.d0        5.d0
4     TNU4[1:NT]  (m^4/s)
       1.0d+07     1.0d+07
5     UVNU2 (m^2/s),   UVNU4 (m^4/s)
       10.d0           0.d0
6     AKT_BAK[1:NT] (m^2/s)
      1.0d-5    1.0d-5
7     AKV_BAK (m^2/s)
      1.0d-4
8     AKQ_BAK (m^2/s)  Q2NU2 (m^2/s),   Q2NU4 (m^4/s)
      1.0d-4             20.d0           1.0d+07
9     RDRG (m/s),      RDRG2
      4.5E-04          0.d0
```

```
10   NMIX_EN,    ADV_ORD,   LEVSFRC,   LEVBFRC
       1           2          1          1
11   THETA_S,   THETA_B,    TCLINE (m)
      3.d0      0.d0         50.d0
12   RHO0 (Kg/m^3),  DSTART (days),   RNUDG (days)
     1025.d0         0.d0             0.d0
13   R0 (Kg/m^3)  T0 (deg C),  S0 (PSU),     TCOEF,      SCOEF
     1026.9524      10.d0        35.d0     -1.67e-04   7.62e-04
14   GAMMA2,  WALL1,  WALL2,  WALL3,   WALL4
       1.d0     T       F       F       F
15   wrtU,    wrtV,   wrtW,   wrtO,   wrtUBAR,  wrtVBAR,  wrtZ
       T        T       T       F       F         F         T
16   wrtT(1:NT)    (temperature, salinity, etc.)
       T        T
17   wrtRHO,  wrtAKV, wrtAKT, wrtAKS, wrtHBL
       F        F       F       F       F
18   NLEV,  LEV(1:NLEV)  in ascending order (if NLEV<0, all levels are saved)
      -1     1 3 5
19   TITLE (a80)
Scrum 3.0
20   RSTNAME (a80): SCRUM output restart file name, if any.
scrum_rst.nc
21   HISNAME (a80): SCRUM output history file name, if any.
scrum_his.nc
22   AVGNAME (a80): SCRUM output averages file name, if any.
scrum_avg.nc
23   STANAME (a80): SCRUM output stations file name, if any.
scrum_sta.nc
24   FLTNAME (a80): SCRUM output floats file name, if any.
scrum_flt.nc
25   GRDNAME (a80): SCRUM input grid file name, if any.
scrum_grd.nc
26   ININAME (a80): SCRUM input initial conditions file name, if any.
scrum_ini.nc
27   FRCNAME (a80): SCRUM input forcing fields file name, if any.
scrum_frc.nc
28   CLMNAME (a80): SCRUM input climatology fields file name, if any.
scrum_clm.nc
29   ASSNAME (a80): SCRUM input assimilation fields file name, if any.
scrum_ass.nc
30   APARNAM (a80): SCRUM input assimilation parameters file name, if any.
assimilation.dat
31   SPOSNAM (a80): SCRUM input station positions file name, if any.
stations.dat
32   FPOSNAM (a80): SCRUM input initial floats positions file name, if any.
```

```
floats.dat
33   USRNAME (a80): USER's input/output generic file name, if any.
/dev/null
99   END of input data
```

6.1.10 User variables and subroutines

It is possible for the user to add new variables and common blocks appropriate to a given application. It is also possible to add new subroutines, for instance to read in river inflow data. If you create new source files they will have to be added to the **Makefile** or the **Imakefile** (see §F). Also, any new **#include** statements will have to be listed in the **Makefile** dependencies. The simplest way to add them is to run **make depend**.

6.2 Upwelling/Downwelling Example

One application for which SCRUM has been configured is a wind-driven upwelling and downwelling example, described in Macks and Middleton [19]. There is a shelf on each wall of a periodic channel and an along-channel wind forcing, which drives upwelling at one wall and downwelling at the other. This problem depends on the Ekman layer, so a surface stress is used with vertical viscosity. The Ekman depth is estimated to be 9 m if $A_v = 0.01 m^2/s$, so the vertical grid spacing must resolve this. The maximum depth is 150 m and our choice of the vertical grid parameters leads to a surface Δz of 4.0 m.

6.2.1 cppdefs.h

The C preprocessor variable **UPWELLING** has been introduced to make sure that we can **#define UPWELLING** and have a consistent upwelling configuration of the model. This is done in part within **cppdefs.h** by

```
#ifdef UPWELLING
#define UV_ADV
#undef  UV_VIS2
#define UV_PRS
#define UV_COR
#define TS_ADV
#undef  TS_DIF2
#undef        NONLIN_EOS
#undef        SALINITY
#undef        CURVGRID
#define       EW_PERIODIC
#undef        NS_PERIODIC
#define TIME_AVG
#undef  BODYFORCE
#define ANA_GRID
#define ANA_INITIAL
#define ANA_MEANRHO
```

```
#define ANA_SMFLUX
#define ANA_STFLUX
#define ANA_SSFLUX
#define ANA_BTFLUX
#define ANA_BSFLUX
#define ANA_VMIX
#endif  /* UPWELLING */
```

Here we have declared that we want a periodic channel but no masking. There is neither salinity nor climatology. The momentum equations have the Coriolis and pressure gradients, but no horizontal viscosity. The only term in the tracer equation is the advection.

6.2.2 Model domain

The flow does not vary in x, so **L** can be small. Set the values for **L**, **M**, **N** and **NT** in **param.h**:

$$\begin{aligned}\mathbf{L} &= 42\\ \mathbf{M} &= 81\\ \mathbf{N} &= 16\\ \mathbf{NT} &= 2.\end{aligned}$$

6.2.3 ana_grid

For this geometry one has a choice of using the grid-generation programs described in Wilkin and Hedstrom [36], or of using **ana_grid** to create the grid analytically. The **ana_grid** subroutine in **analytical.F** was modified to produce a bathymetry with a shelf on both walls of the channel when **UPWELLING** is defined. The fluid depth ranges from 27 m on the shelves to 150 m in the center of the channel. The horizontal grid spacing is uniform at 1 km and the Coriolis parameter f is set to a constant value suitable for Sydney, Australia.

6.2.4 Initial conditions and the equation of state

We would like the initial conditions to be a motionless fluid with an exponential stratification. **ana_initial** is configured accordingly.

The stratification can be provided by either T or S, or by both T and S. For simplicity we will only have an active temperature field and we will use the linear equation of state by setting **NONLIN_EOS** to **#undef** in **rhsdefs.h**. We want the density to be 26.35 at the bottom and 24.22 at the top with an e-folding scale of 50 meters. The initial temperature is set to $14 + 8\mathrm{e}^{z/50}$ in **ana_initial**. The linear equation of state parameters are set in **scrum.in**.

Since density does not depend on salinity, we have a choice of how to handle the second tracer. We can either use it as a passive tracer or not timestep on it at all by setting $\mathbf{NT} = 1$. We will use it as a passive tracer and initialize it to be a function of y.

We have set **ana_meanRHO** to the desired initial density field. The climatology fields are not used and need not be initialized.

6.2.5 Boundary conditions

The periodic channel options have already been chosen in **cppdefs.h**. We do not have to do anything else.

6.2.6 Model forcing

In this problem we want to resolve the surface Ekman layer and to use a surface wind stress rather than a body force. We want the amplitude of the wind to ramp up with time so we modify **ana_smflux** accordingly. The wind will build to an amplitude of 0.1 Pascals / ρ_o, or $10^{-4} m^2 s^{-2}$.

We need to edit **ana_vmix** to make sure that the vertical viscosity **Akv** is set to the value we want. This must be large at the surface ($10^{-2} m^2 s^{-1}$) to create a thick Ekman layer, but has been chosen to decrease with depth. We also need to check that **ana_sbflux**, **ana_stflux**, etc. are set correctly.

6.2.7 scrum.in

The model has been set up to run for one day with an internal timestep of 120 s and an external timestep of 12 s. We will write history and restart records every 1/4 day. The value of the linear bottom friction coefficient **rdrg** is set to 4.5×10^{-4} and the channel walls are set to be free-slip.

6.2.8 Output

The model writes some information to standard out, after setting **ninfo** to 72:

```
SCRUM input parameters:

      720  ntimes      Number of timesteps to evolve 3-D equations.
   120.00  dt          Timestep size (s) for 3-D equations.
       10  ndffast     Number of timesteps for 2-D equations between each DT.
        0  nrrec       Number of restart records to read from disk.
      180  nrst        Number of timesteps between storage of restart fields.
      180  nwrt        Number of timesteps between writing fields into
                       history file.
       72  ninfo       Number of timesteps between print of information
                       to standard output.
        T  ldefhis     Switch to create a new history NetCDF file.
        T  lcycle      Switch to recycle time-records in restart NetCDF file.
0.000E+00  tnu2(1)     Horizontal, Laplacian mixing coefficient (m^2/s)
                       for tracer 1.
0.000E+00  tnu2(2)     Horizontal, Laplacian mixing coefficient (m^2/s)
                       for tracer 2.
0.000E+00  uvnu2       Horizontal, Laplacian mixing coefficient (m^2/s)
                       for momentum.
```

```
 0.000E+00  Akt_bak(1)  Background vertical mixing coefficient (m^2/s)
                        for tracer 1.
 0.000E+00  Akt_bak(2)  Background vertical mixing coefficient (m^2/s)
                        for tracer 2.
 1.000E-05  Akv_back    Background vertical mixing coefficient (m^2/s)
                        for momentum.
 4.500E-04  rdrg        Linear bottom drag coefficient (m/s).
 0.000E+00  rdrg2       Quadratic bottom drag coefficient.
 3.000E+00  theta_s     S-coordinate surface control parameter.
 0.000E+00  theta_b     S-coordinate bottom  control parameter.
   50.0000  Tcline      S-coordinate surface/bottom layer width (m) used
                        in vertical coordinate stretching.
 1000.0000  rho0        Mean density (kg/m^3) used in Boussinesq approximation.
    0.0000  dstart      Time stamp assigned to model initialization (days).
    0.0000  T0          Background potential temperature (Celsius) constant.
    0.0000  S0          Background salinity (PSU) constant.
   30.3795  R0          Background density (kg/m^3) used in linear Equation
                        of State.
-2.800E-01  Tcoef       Thermal expansion coefficient (1/Celsius).
 0.000E+00  Scoef       Saline contraction coefficient (1/PSU).
      1.00  gamma2      Slipperiness variable: free-slip (1.0) or
                                               no-slip (-1.0).
         F  wall1       Boundary for side 1 (i=1): wall/open (T/F).
         T  wall2       Boundary for side 2 (j=1): wall/open (T/F).
         F  wall3       Boundary for side 3 (i=L): wall/open (T/F).
         T  wall4       Boundary for side 4 (j=M): wall/open (T/F).
         T  wrtU        Write out 3D U-momentum component (T/F).
         T  wrtV        Write out 3D V-momentum component (T/F).
         T  wrtW        Write out W-momentum component (T/F).
         T  wrtO        Write out omega vertical velocity (T/F).
         T  wrtUBAR     Write out 2D U-momentum component (T/F).
         T  wrtVBAR     Write out 2D V-momentum component (T/F).
         T  wrtZ        Write out free-surface (T/F).
         T  wrtT(1)     Write out tracer 1 (T/F).
         T  wrtT(2)     Write out tracer 2 (T/F).
         T  wrtRHO      Write out density anomaly (T/F).
         F  wrtAKV      Write out vertical viscosity coefficient (T/F).
         F  wrtAKT      Write out vertical T-diffusion coefficient (T/F).
         F  wrtAKS      Write out vertical S-diffusion coefficient (T/F).
        16  Nlev        Number of levels to write out.
            Lev         Levels to write out:
                        01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16

Upwelling/Downwelling Example on a Periodic Double Shelf Channel
```

```
Output/Input Files:

           Output Restart File:  scrum_rst.nc
           Output History File:  scrum_his.nc
         Input/Output USER File:  /dev/null

Activated C-preprocessing Options:

 ANA_BSFLUX        Analytical kinematic bottom salt flux.
 ANA_BTFLUX        Analytical kinematic bottom heat flux.
 ANA_GRID          Analytical grid set-up.
 ANA_INITIAL       Analytical initial conditions.
 ANA_MEANRHO       Analytical mean density anomaly.
 ANA_SMFLUX        Analytical kinematic surface momentum flux.
 ANA_SSFLUX        Analytical kinematic freshwater (E-P) flux.
 ANA_STFLUX        Analytical kinematic surface heat flux.
 ANA_VMIX          Analytical vertical mixing coefficients.
 DBLEPREC          Double precision arithmetic.
 EW_PERIODIC       East-West periodic boundaries.
 MIX_GP_TS         Mixing of tracers along geopotential surfaces.
 MIX_GP_UV         Mixing of momentum along geopotential surfaces.
 SOLVE2D           Solving 2D Primitive Equations.
 SOLVE3D           Solving 3D Primitive Equations.
 TIME_AVG          Time averaging over two short timestep cycles.
 TS_ADV            Advection of tracers.
 UPWELLING         Upwelling/Downwelling Example.
 UV_ADV            Advection of momentum.
 UV_COR            Coriolis term.
 UV_PRS            Hydrostatic pressure gradient term.

Vertical S-coordinate System:

  level    S-coord   at hmin    over slope    at hmax

  16        0.00        0.00        0.00        0.00
  15       -0.06       -1.71       -2.87       -4.02
  14       -0.12       -3.43       -5.78       -8.12
  13       -0.19       -5.14       -8.77      -12.39
  12       -0.25       -6.86      -11.89      -16.92
  11       -0.31       -8.57      -15.18      -21.80
  10       -0.38      -10.28      -18.71      -27.14
   9       -0.44      -12.00      -22.54      -33.08
   8       -0.50      -13.71      -26.74      -39.76
```

```
   7       -0.56      -15.42      -31.40      -47.37
   6       -0.62      -17.14      -36.62      -56.09
   5       -0.69      -18.85      -42.52      -66.20
   4       -0.75      -20.57      -49.27      -77.97
   3       -0.81      -22.28      -57.02      -91.76
   2       -0.88      -23.99      -66.00     -108.01
   1       -0.94      -25.71      -76.46     -127.21
   0       -1.00      -27.42      -88.71     -150.00

 MAIN - started time-stepping SCRUM:

 Day =      0.100000  avgKE =  7.090495E-19  avgPE =  1.697521E-08
 Day =      0.200000  avgKE =  5.151371E-18  avgPE =  1.697524E-08
 Day =      0.300000  avgKE =  1.904604E-17  avgPE =  1.697527E-08
 Day =      0.400000  avgKE =  4.972789E-17  avgPE =  1.697529E-08
 Day =      0.500000  avgKE =  1.058779E-16  avgPE =  1.697532E-08
 Day =      0.600000  avgKE =  1.973340E-16  avgPE =  1.697534E-08
 Day =      0.700000  avgKE =  3.345132E-16  avgPE =  1.697535E-08
 Day =      0.800000  avgKE =  5.280177E-16  avgPE =  1.697536E-08
 Day =      0.900000  avgKE =  7.890830E-16  avgPE =  1.697537E-08
 Day =      1.000000  avgKE =  1.130497E-15  avgPE =  1.697536E-08

 Main - number of time records written in history file: 0005
        number of time records written in restart file: 0002

Main Done.
```

NetCDF history and restart files are also created, containing the model fields at the requested times. We have asked it to save both history and restart records every 1/4 day. In this case, the restart file has been told to "cycle", or to write over the second last record. The restart file at the end of the run contains the fields at 3/4 day and 1 day. The history file contains records for 0, 1/4, 1/2, 3/4, and 1 day. Plots can be made from either file, using the plotting software described in §7. Selected frames from such plots are shown in Fig. 6.2 to 6.5.

6.3 North Atlantic example

The upwelling/downwelling examples is one in which all the start-up fields are defined analytically. The other extreme is one in which everything is read from files, as in our North Atlantic simulations.

6.3.1 cppdefs.h

The C preprocessor variable **DAMEE_B** has been introduced to make sure that we can **#define DAMEE_B** and have a consistent configuration of the model. This is done in part within **cppdefs.h** by

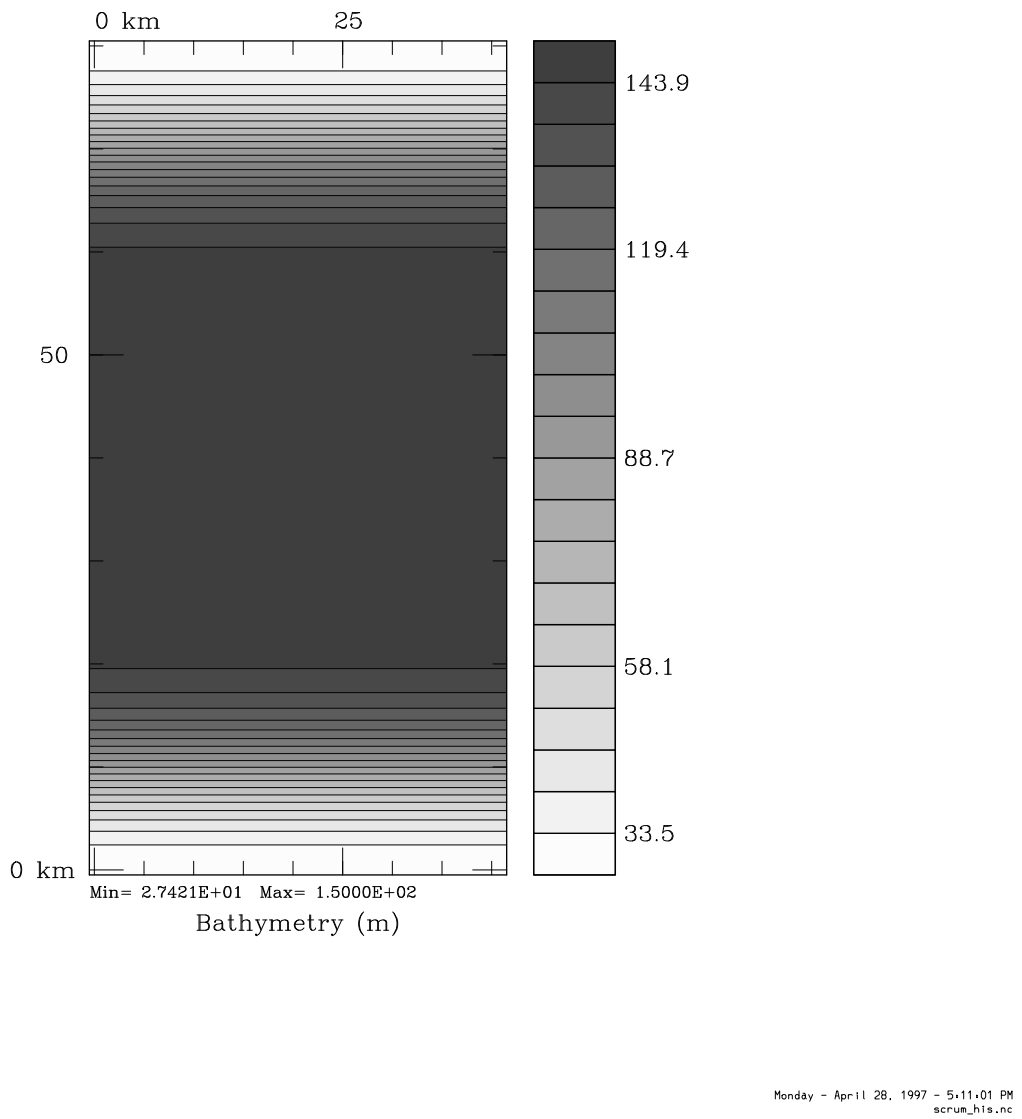

Figure 6.2: The upwelling/downwelling bathymetry.

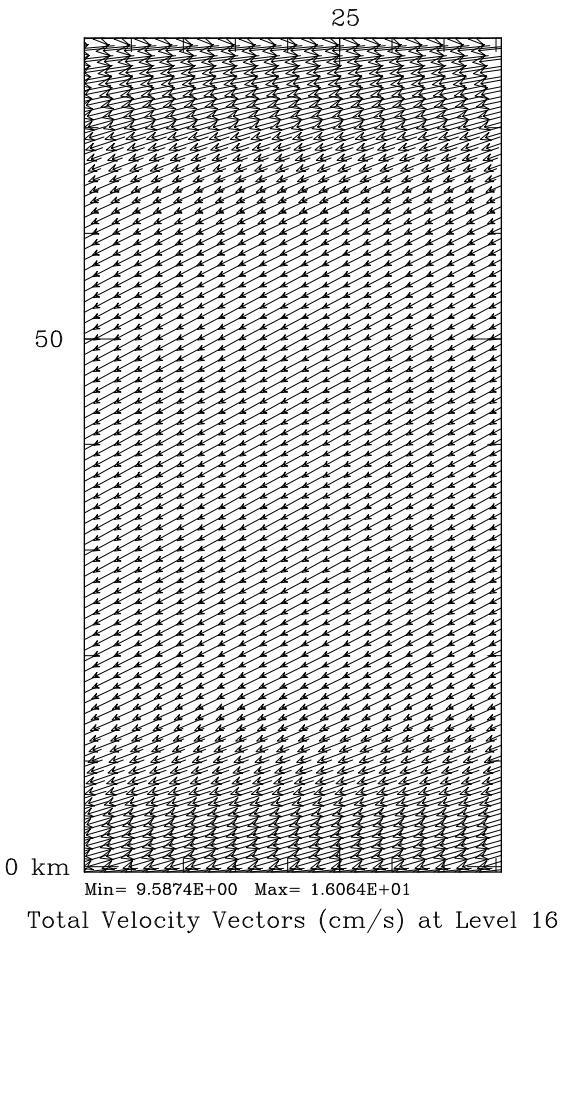

Figure 6.3: Surface velocities after one day, showing the flow to the left of the wind (southern hemisphere).

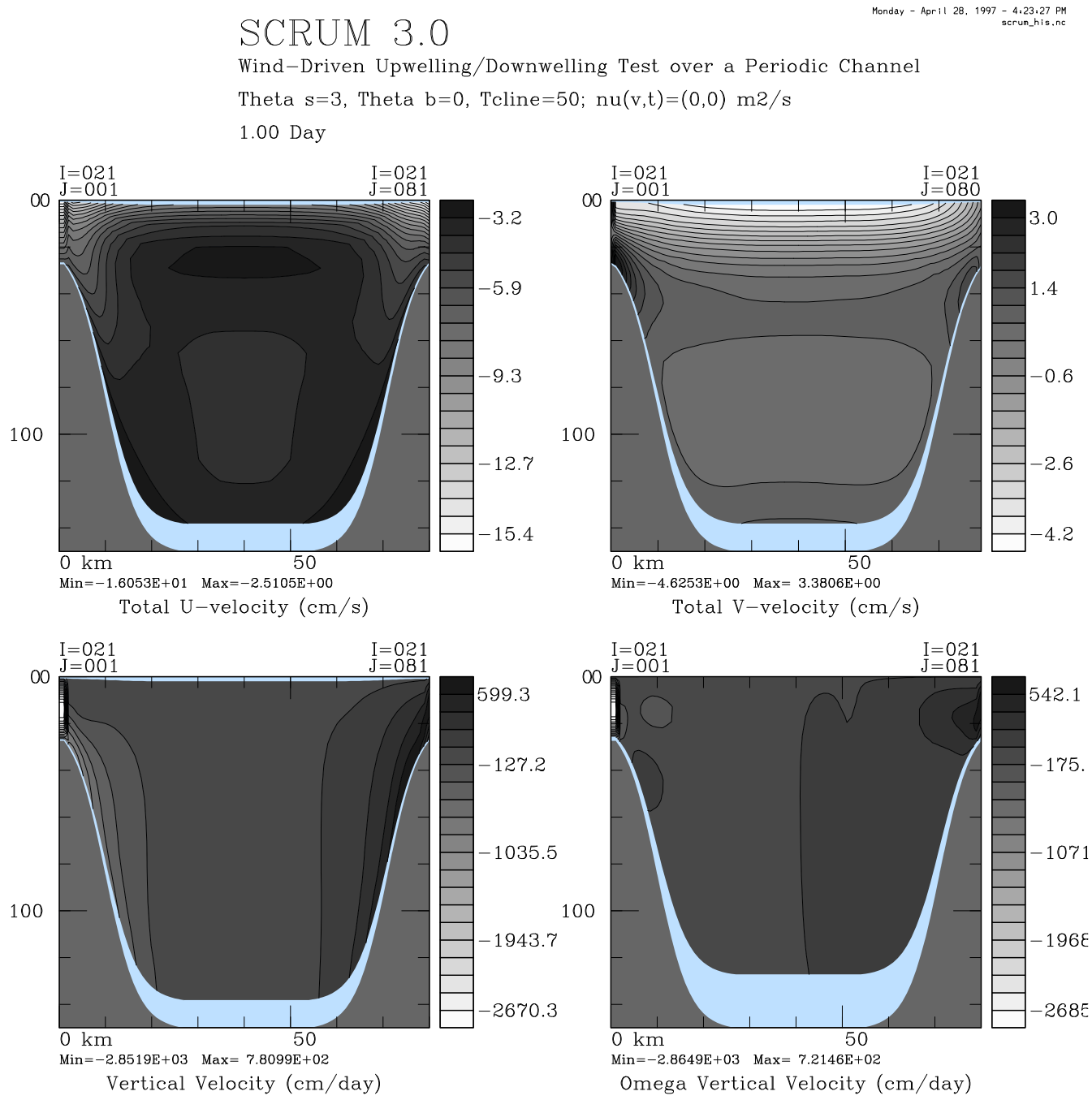

Figure 6.4: Constant ξ slices of the u, v, w and Ω fields at day 1.

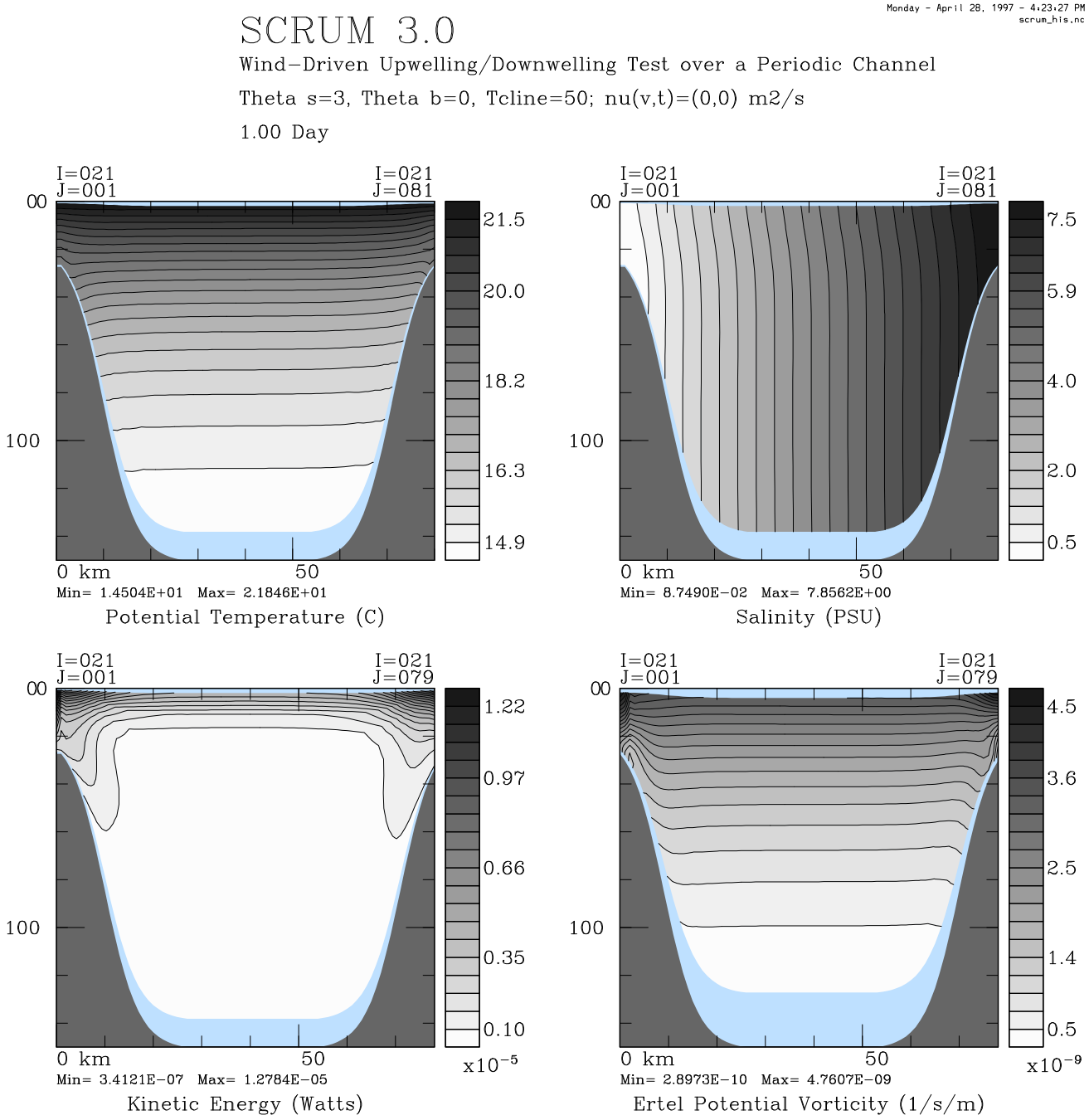

Figure 6.5: Constant ξ slices of the T, S (tracer), kinetic energy and Ertel potential vorticity at day 1.

```
!
!  Select options for horizontal mixing of MOMENTUM:
!
#undef  MIX_S_UV        /* mixing along constant S-surfaces */
# ifndef MIX_S_UV
#define MIX_GP_UV       /* mixing on geopotential (constant Z) surfaces */
#undef  MIX_EN_UV       /* mixing on epineutral (constant RHO) surfaces */
# endif /* !MIX_S_UV */
!
!  Select options for horizontal mixing of TRACERS:
!
#undef  CLIMAT_TS_MIXH  /* apply horizontal mixing to tracer-climatology */
#undef  CLIMAT_TS_MIXV  /* apply vertical mixing to tracer-climatology */
#undef  MIX_S_TS        /* mixing along constant S-surfaces */
# ifndef MIX_S_TS
#define MIX_GP_TS       /* mixing on geopotential (constant Z) surfaces */
#undef  MIX_EN_TS       /* mixing on epineutral (constant RHO)
surfaces */ # endif /* !MIX_S_TS */
         :
# if defined DAMEE_B || defined DAMEE_S
#define UV_ADV
#define UV_VIS2
#undef  UV_VIS4
#define UV_PRS
#define UV_COR
#define TS_ADV
#define TS_DIF2
#undef  TS_DIF4
#undef  SMOLARKIEWICZ
#define NONLIN_EOS
#define SALINITY
#undef  DIAGNOSTIC
#define QCORRECTION
#define CURVGRID
#define AVERAGES
#undef  STATIONS
#undef  OBC_EAST
#undef  OBC_WEST
#define OBC_NORTH
#define OBC_SOUTH
#undef  EW_PERIODIC
#undef  NS_PERIODIC
#undef  INFLOW
#define OBC_TPRESCRIBE
#define MASKING
```

```
#define TIME_AVG
#define BODYFORCE
#undef  BVF_MIXING
#undef  PP_MIXING
#define LMD_MIXING
#define LMD_RIMIX
#define LMD_CONVEC
#undef  LMD_DDMIX
#define LMD_KPP
#define CLIMATOLOGY
#define NUDGING
#define ANA_MEANRHO
#undef  ANA_SMFLUX
#undef  ANA_SSFLUX
#undef  ANA_STFLUX
#undef  ANA_SRFLUX
#define ANA_BSFLUX
#define ANA_BTFLUX
#undef  ANA_V2DBC
# endif /* DAMEE_B || DAMEE_S */
```

Here, we have declared that we want a closed basin (not periodic), masking, salinity, and the non-linear equation of state. We want Laplacian viscosity and diffusion along constant z-surfaces and the full non-linear, curvilinear momentum equations.

We also added the DAMEE flags to **checkdefs.F**:

```
#ifdef DAMEE_B
      write(stdout,20) 'DAMEE_B',
     &                 'North Atlantic DAMEE Big Domain Application.'
      is=lenstr(Coptions)+1
      Coptions(is:is+9)=' DAMEE_B,'
      iexample=iexample+1
#endif /* DAMEE_B */
#ifdef DAMEE_S
      write(stdout,20) 'DAMEE_S',
     &                 'North Atlantic DAMEE Small Domain Application.'
      is=lenstr(Coptions)+1
      Coptions(is:is+9)=' DAMEE_S,'
      iexample=iexample+1
#endif /* DAMEE_S */
```

6.3.2 Model domain

A large number of horizontal grid points was chosen to resolve the domain at less than one degree. Values for **L**, **M**, **N**, and **NT** are:

$$\mathbf{L} = 129$$

M = 129
N = 20
NT = 2.

6.3.3 gridpak

The grid has uniform spacing on a Mercator projection so that both Δx and Δy get smaller as you get farther from the equator. The grid was chosen to go from 30° S to 65° N and was generated with **sqgrid**. We then found the latitude and longitude values with **tolat** and interpolated the **etopo5** bathymetry to the grid with **bathtub**. The grid is shown in Fig. 6.6 and the unsmoothed bathymetry is shown in Fig. 6.7. It is clear that the unsmoothed bathymetry contains some incredibly steep regions. We have not pushed SCRUM to see what its steepness limit is, but we also ran SPEM in this configuration and its elliptic solver requires substantial smoothing at this resolution. We were adviced by Bernard Barnier to retain the shallow island arc in the Caribbean. We also had some bad experiences with shelves that disappeared into the land mask, such as at Cape Hatteras and the Iberian peninsula. We filled in the Pacific and the Mediterranean and did some unspeakable hacking to **bathsuds** to obtain the bathymetry shown in Fig. 6.8. We then ran **sphere** to obtain the values of m and n suitable for a spherical Earth and ran Hernan Arango's mask editing tool **scrum_mask**.

6.3.4 Initial conditions

We would like the initial conditions to be a motionless fluid with temperature and salinity fields from the Levitus 1994 February mean climatology. We prepared a NetCDF file with zero u, v and ζ fields. The T and S fields were interpolated from the Levitus fields—we tried several different interpolation/extrapolation techniques, including the **oa** program described in §5.3.

An analytic function for the mean density was added to **ana_meanRHO** for this problem:

```
# elif defined DAMEE_B || defined DAMEE_S
          do k=1,N
            do j=0,M
              do i=0,L
                rhobar(i,j,k)=30.5-0.004*z_r(i,j,k)-
        &                           c4*exp(z_r(i,j,k)/2000.0)
              enddo
            enddo
          enddo
```

6.3.5 Boundary conditions

The non-periodic option has already been chosen by not defining **EW_PERIODIC** or **NS_PERIODIC** in **cppdefs.h**. We have defined **OBC_NORTH**, **OBC_SOUTH** and **OBC_TREDUCED** to prescribe tracers on the northern and southern boundaries.

6.3.6 Forcing

The forcing is provided by surface momentum, heat and salt fluxes from the COADS dataset. We apply the heat flux correction (**#define QCORRECTION**), which is also provided in COADS. We use the **oa** program to put the values onto the model grid for each of the twelve monthly means.

6.3.7 Climatology

We used the same Levitus temperature and salinity fields for the climatology as for the initial conditions. The DAMEE problem was specified to have nudging to the climatology at the northern and southern boundaries, as well as at the Straits of Gibralter. We editted **initial.F** to set the **nudgcof** array accordingly.

6.3.8 scrum.in

We use an internal timestep of 2160 s and an external timestep of 108 s. The horizontal viscosity and diffusion coefficients are 5000 and 1000, respectively. The stretching parameters are $\theta = 5$, $b = .4$ and $h_c = 200m$.

6.3.9 Output

The model writes out information to standard out:

```
 SCRUM input parameters:

    144000  ntimes      Number of timesteps to evolve 3-D equations.
   2160.00  dt          Timestep size (s) for 3-D equations.
        20  ndffast     Number of timesteps for 2-D equations between each DT.
         0  nrrec       Number of restart records to read from disk.
       400  nrst        Number of timesteps between storage of restart fields.
      1200  nwrt        Number of timesteps between writing fields into
                        history file.
         1  ntsavg      Starting timestep for the accumulation of output
                        time-averaged data.
      1200  navg        Number of timesteps between writing of time-averaged
                        data into averages file.
         1  ninfo       Number of timesteps between print of information
                        to standard output.
         T  ldefhis     Switch to create a new history NetCDF file.
         T  lcycle      Switch to recycle time-records in restart NetCDF file.
 1.000E+03  tnu2(1)     Horizontal, Laplacian mixing coefficient (m^2/s)
                        for tracer 1.
 1.000E+03  tnu2(2)     Horizontal, Laplacian mixing coefficient (m^2/s)
                        for tracer 2.
 5.000E+03  uvnu2       Horizontal, Laplacian mixing coefficient (m^2/s)
```

```
                    for momentum.
1.000E-05  Akt_bak(1)  Background vertical mixing coefficient (m^2/s)
                    for tracer 1.
1.000E-05  Akt_bak(2)  Background vertical mixing coefficient (m^2/s)
                    for tracer 2.
1.000E-04  Akv_back    Background vertical mixing coefficient (m^2/s)
                    for momentum.
3.000E-04  rdrg        Linear bottom drag coefficient (m/s).
0.000E+00  rdrg2       Quadratic bottom drag coefficient.
       18  levsfrc    Deepest level to apply surface stress as a bodyforce.
        1  levbfrc    Shallowest level to apply bottom stress as a bodyforce.
5.000E+00  theta_s     S-coordinate surface control parameter.
4.000E-01  theta_b     S-coordinate bottom  control parameter.
 200.0000  Tcline      S-coordinate surface/bottom layer width (m) used
                    in vertical coordinate stretching.
1000.0000  rho0        Mean density (kg/m^3) used in Boussinesq approximation.
  30.0000  dstart      Time stamp assigned to model initialization (days).
   0.0000  T0          Background potential temperature (Celsius) constant.
   0.0000  S0          Background salinity (PSU) constant.
     1.00  gamma2      Slipperiness variable: free-slip (1.0) or
                                          no-slip (-1.0).
        T  wall1       Boundary for side 1 (i=1): wall/open (T/F).
        T  wall2       Boundary for side 2 (j=1): wall/open (T/F).
        T  wall3       Boundary for side 3 (i=L): wall/open (T/F).
        T  wall4       Boundary for side 4 (j=M): wall/open (T/F).
        T  wrtU        Write out 3D U-momentum component (T/F).
        T  wrtV        Write out 3D V-momentum component (T/F).
        T  wrtW        Write out W-momentum component (T/F).
        F  wrtO        Write out omega vertical velocity (T/F).
        T  wrtUBAR     Write out 2D U-momentum component (T/F).
        T  wrtVBAR     Write out 2D V-momentum component (T/F).
        T  wrtZ        Write out free-surface (T/F).
        T  wrtT(1)     Write out tracer 1 (T/F).
        T  wrtT(2)     Write out tracer 2 (T/F).
        F  wrtRHO      Write out density anomaly (T/F).
        T  wrtAKV      Write out vertical viscosity coefficient (T/F).
        T  wrtAKT      Write out vertical T-diffusion coefficient (T/F).
        T  wrtAKS      Write out vertical S-diffusion coefficient (T/F).
        T  wrtHBL      Write out depth of mixed layer (T/F).

Scrum 3.0 - North Atlantic Damee 4: Annual Levitus 0.75 resolution


 Output/Input Files:
```

```
           Output Restart File:  scrum_rst.nc
         Output Averages File:  scrum_avg.nc
              Input Grid File:  damee_grid_4.nc
        Input Initial File:  damee_lev_4feb.nc
        Input Forcing File:  frc_coads_4.nc
      Input Climatology File:  damee_clm_4L.nc
      Input/Output USER File:  /dev/null

Activated C-preprocessing Options:

 ANA_BSFLUX       Analytical kinematic bottom salt flux.
 ANA_BTFLUX       Analytical kinematic bottom heat flux.
 ANA_MEANRHO      Analytical mean density anomaly.
 AVERAGES         Writing out time-averaged fields.
 BODYFORCE        Momentum stresses as body-forces.
 CURVGRID         Orthogonal curvilinear grid.
 DAMEE_B          North Atlantic DAMEE Big Domain Application.
 DBLEPREC         Double precision arithmetic.
 LMD_CONVEC       LMD convective mixing due to shear instability.
 LMD_MIXING       Large/McWilliams/Doney interior mixing.
 LMD_KPP          Large/McWilliams/Doney boundary layer mixing.
 LMD_RIMIX        LMD diffusivity due to shear instability.
 MASKING          Land/Sea masking.
 MIX_GP_TS        Mixing of tracers along geopotential surfaces.
 MIX_GP_UV        Mixing of momentum along geopotential surfaces.
 NONLIN_EOS       Non-linear Equation of State for seawater.
 NUDGING          Nudging toward climatology.
 OBC_NORTH        Open North boundary edge.
 OBC_SOUTH        Open South boundary edge.
 OBC_TREDUCED     Tracers, boundary reduced physics condition.
 QCORRECTION      Surface net heat flux correction.
 SALINITY         Using salinity.
 SOLVE2D          Solving 2D Primitive Equations.
 SOLVE3D          Solving 3D Primitive Equations.
 TCLIMATOLOGY     Processing tracer climatology data.
 TIME_AVG         Time averaging over two short timestep cycles.
 TNUDGING         Nudging toward tracer climatology.
 TS_ADV           Advection of tracers.
 TS_DIF2          Laplacian mixing of tracers.
 UV_ADV           Advection of momentum.
 UV_COR           Coriolis term.
 UV_PRS           Hydrostatic pressure gradient term.
 UV_VIS2          Laplacian mixing of momentum.
```

```
Vertical S-coordinate System:

  level    S-coord   at hmin    over slope    at hmax

  20        0.00        0.00        0.00        0.00
  19       -0.05      -10.00      -20.03      -30.05
  18       -0.10      -20.00      -43.30      -66.60
  17       -0.15      -30.00      -71.92     -113.84
  16       -0.20      -40.00     -108.94     -177.89
  15       -0.25      -50.00     -158.64     -267.27
  14       -0.30      -60.00     -226.50     -393.01
  13       -0.35      -70.00     -318.60     -567.19
  12       -0.40      -80.00     -439.47     -798.94
  11       -0.45      -90.00     -588.95    -1087.90
  10       -0.50     -100.00     -759.64    -1419.28
   9       -0.55     -110.00     -938.48    -1766.95
   8       -0.60     -120.00    -1112.90    -2105.81
   7       -0.65     -130.00    -1277.09    -2424.19
   6       -0.70     -140.00    -1433.59    -2727.18
   5       -0.75     -150.00    -1591.00    -3032.00
   4       -0.80     -160.00    -1761.00    -3361.99
   3       -0.85     -170.00    -1956.64    -3743.28
   2       -0.90     -180.00    -2192.17    -4204.35
   1       -0.95     -190.00    -2483.64    -4777.28
   0       -1.00     -200.00    -2850.00    -5500.00

     GET_INITIAL - Processing initial conditions  for time = 0.0000E+00
     GET_SRFLUX  - Read solar shortwave radiation for time =    345.0
     GET_STFLUX  - Read surface flux of tracer 01 for time =    345.0
     GET_STFLUX  - Read surface flux of tracer 02 for time =    345.0
     GET_SMFLUX  - Read surface momentum stresses for time =    345.0
     GET_CLIMA   - Read climatology of tracer  01 for time = 0.0000E+00
     GET_CLIMA   - Read climatology of tracer  02 for time = 0.0000E+00

MAIN - started time-stepping SCRUM:

     GET_SRFLUX  - Read solar shortwave radiation for time =    15.00
     GET_STFLUX  - Read surface flux of tracer 01 for time =    15.00
     GET_STFLUX  - Read surface flux of tracer 02 for time =    15.00
     GET_SMFLUX  - Read surface momentum stresses for time =    15.00

Day =      0.025000  avgKE =  9.587033E-16  avgPE =  7.947536E-07
Day =      0.050000  avgKE =  9.608381E-16  avgPE =  7.947534E-07
Day =      0.075000  avgKE =  9.583708E-16  avgPE =  7.947537E-07
                   :
```

It also writes out NetCDF files for restart, history, and monthly averages. Plots can be made from all three of these files; an example plot is shown in Fig. 6.9.

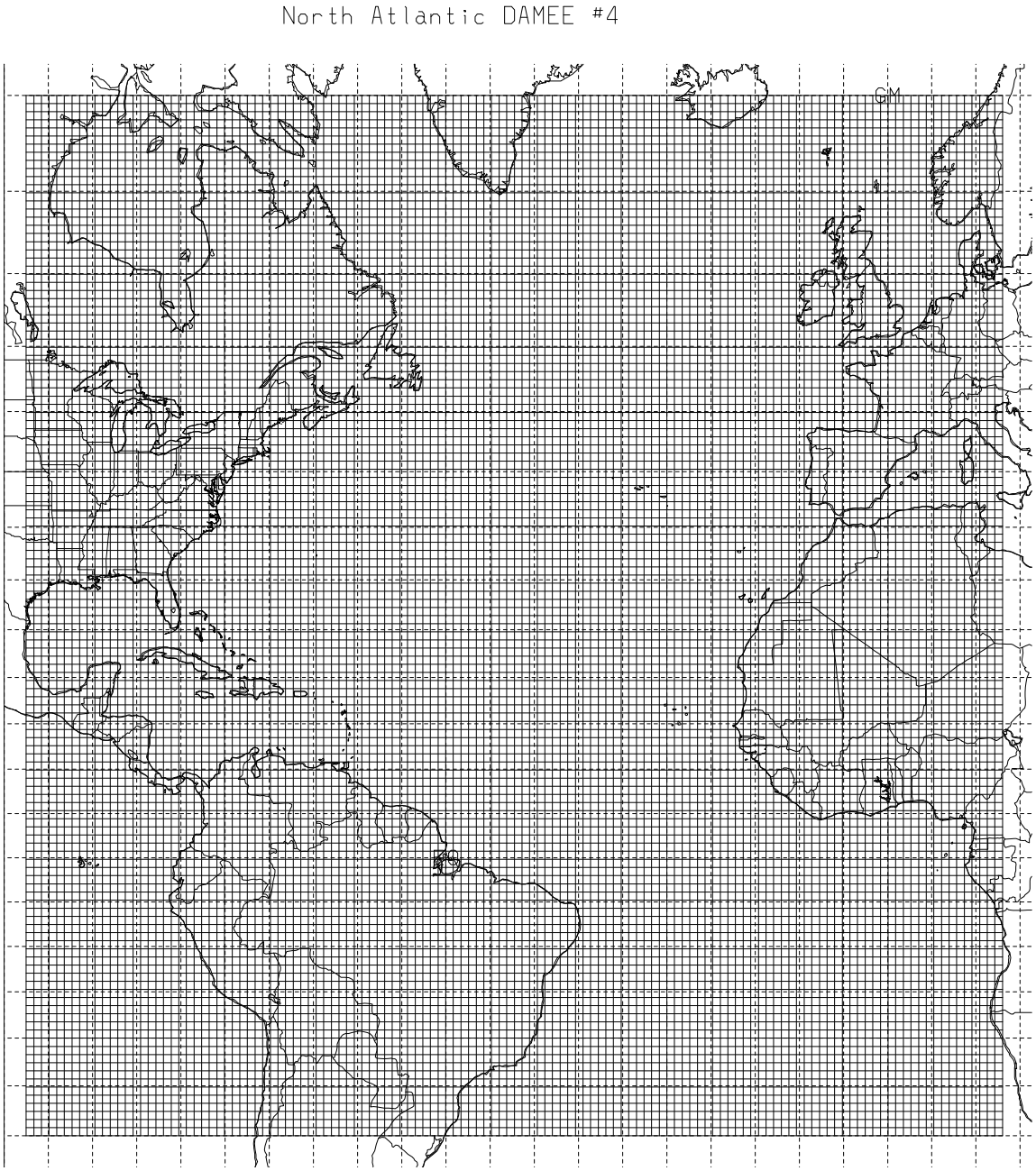

Figure 6.6: The North Atlantic grid.

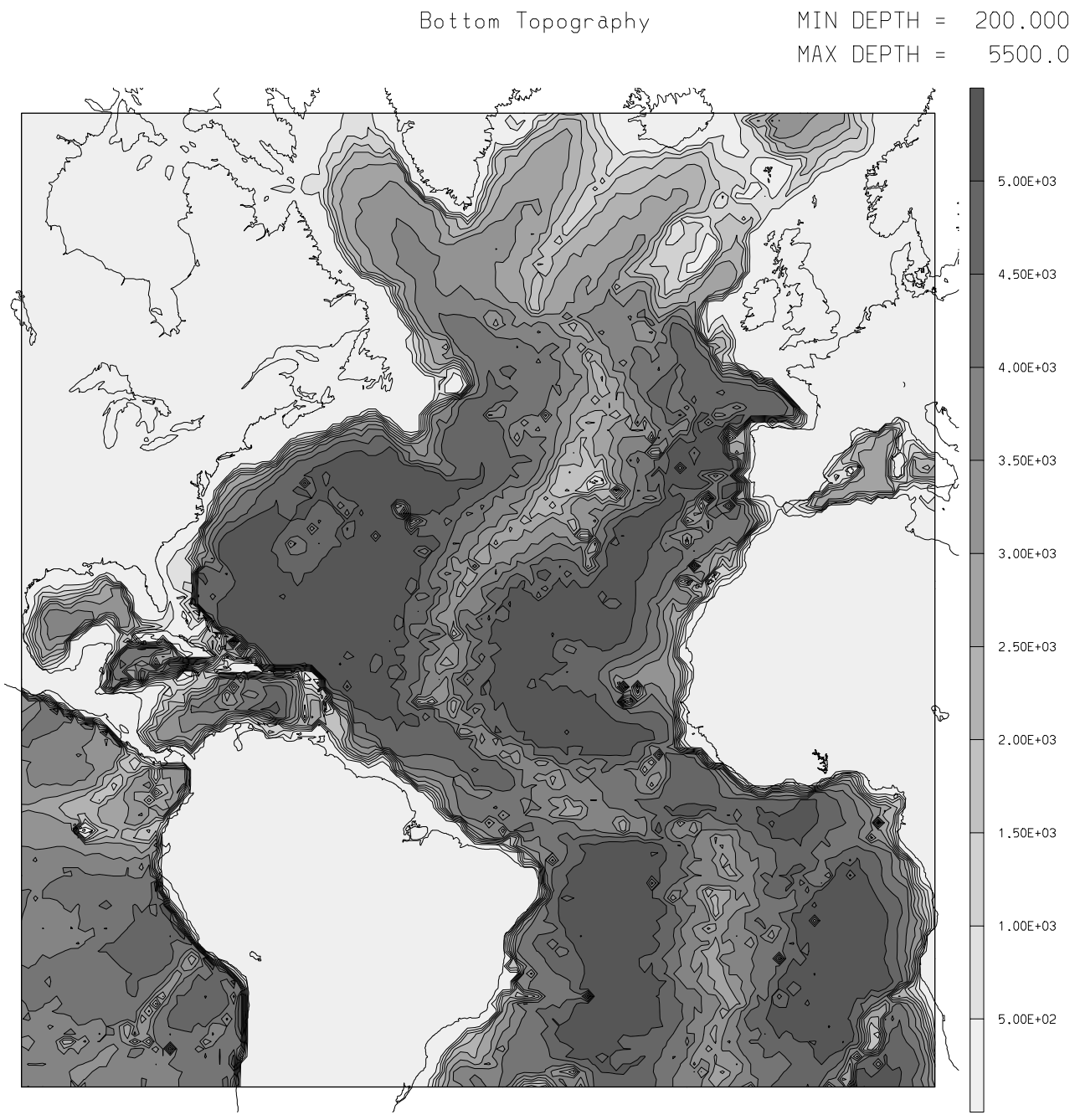

Figure 6.7: The raw bathymetry from **etopo5**.

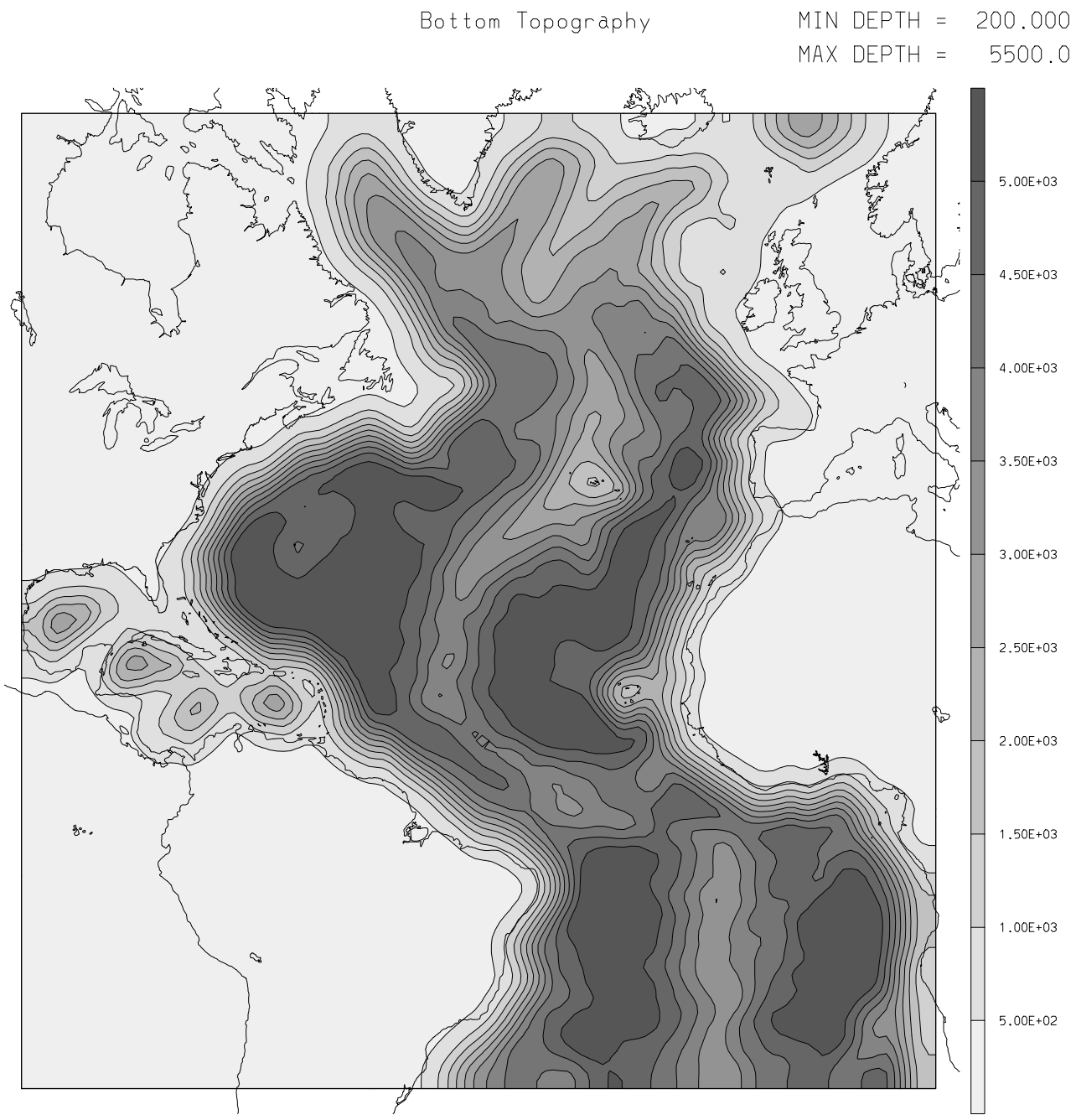

Figure 6.8: The smoothed North Atlantic bathymetry.

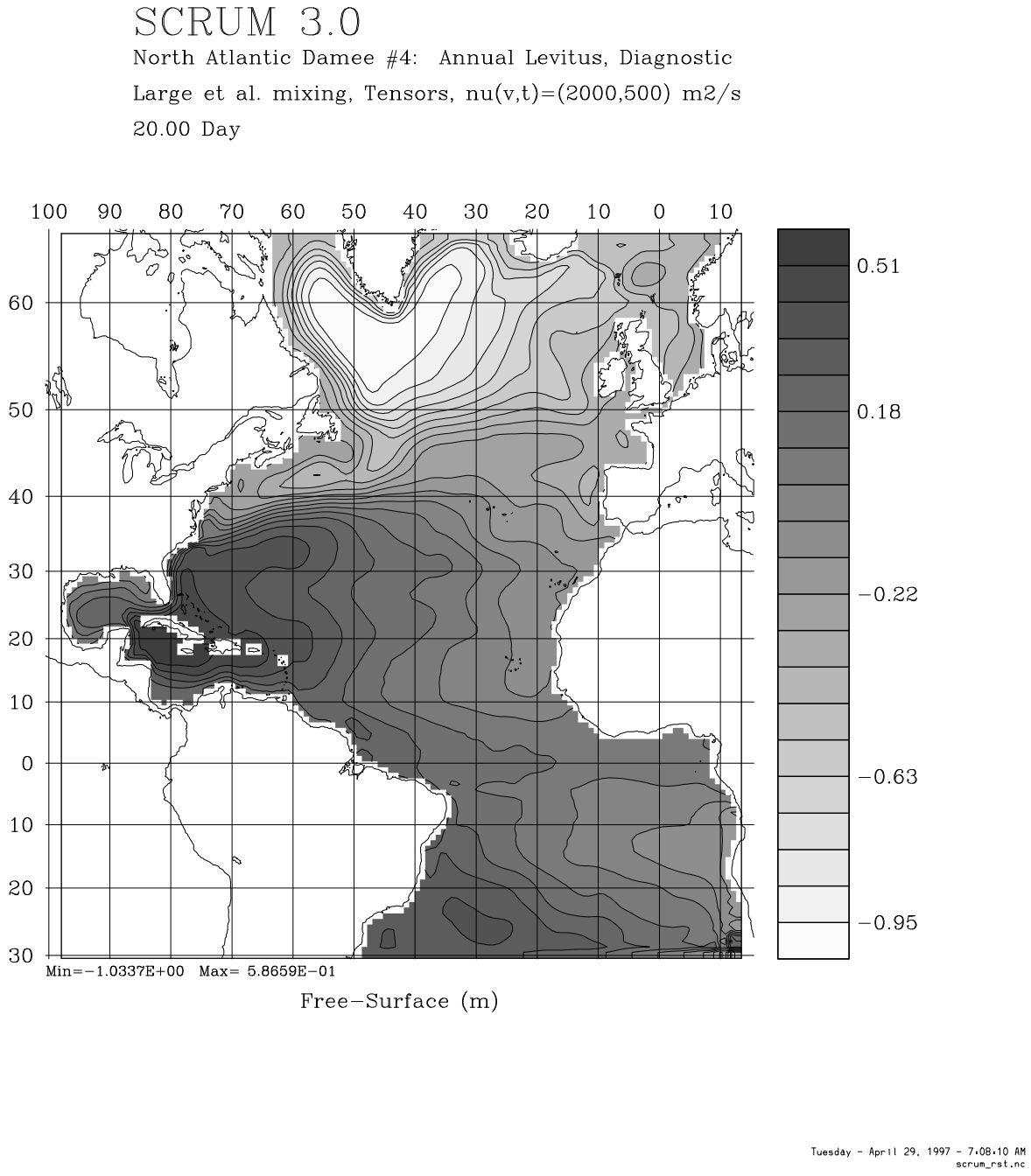

Figure 6.9: The surface elevation for day 20.

Chapter 7

Plotting Programs for Postprocessing

Hernan Arango has provided SCRUM with some programs for creating plots from the NetCDF history and restart files. There are four plotting programs:

cnt creates black-and-white plots of the horizontal fields, including constant depth plots of the 3-D fields.

ccnt creates color plots of the horizontal fields, including constant depth plots of the 3-D fields.

sec creates black-and-white plots of vertical slices through the 3-D fields. It includes on option of finding equal-spaced points along a straight line through the curvilinear grid.

csec creates color plots of vertical slices through the 3-D fields. It includes on option of finding equal-spaced points along a straight line through the curvilinear grid.

All of these program come with example input files. For instance, the input file for **cnt** is called **cnt.in** and is as follows:

```
1996 -1  :  year and starting year-day (use yearday<0, for no time label)
SCRUM 3.0
Form Streess Test over Steep Shelf/Slope with Across-Shelf Canyon
Canyon A: Homogeneous Case
Vnu=20 m^2/s, Tnu=20 m^2/s
2     NFIELDS: number of fields to plot. Line below, field(s) types:
1,2            field identification: FLDID(1:NFIELDS)
1     NLEVELS: number of levels and/or depths to plot (0 for all levels)
1,2,3,4,5      levels (>0) or depths (<0) to plot: FLDLEV(1:NLEVELS)
0     FRSTD  : first day to plot
0     LASTD  : last day to plot
0     DSKIP  : plot every other DSKIP days (0.0 plot at its own time frequency)
0     VINTRP : vertical interpolation scheme: 0=linear, 1:cubic splines
```

```
1.2   VLWD   : vector line width (1.0 for default)
4.0   VLSCL  : vector length scale (1.0 for default)
2     IVINC  : vector grid sampling in the X-direction (1 for default)
1     JVINC  : vector grid sampling in the Y-direction (1 for default)
0     IREF   : secondary or reference field option (see below)
25    IDOVER : overlay field identification (for IREF=1,2 only)
1     LEVOVER: level of the overlay field (set to 0 if same as current FLDLEV)
0.0   RMIN   : overlay field minimum value to consider (0.0 for default)
0.0   RMAX   : overlay field maximum value to consider (0.0 for default)
1     LGRID  : Desired longitude/latitude grid spacing (degrees)
1     IPROJ  : map projection: [1] cyl. equidistant, [2] Mercator, [3] Lambert
2     NPAGE  : number of plots per page (currently 1, 2, or 4)  (see below).
F     READGRD: logical switch to read in positions from grid NetCDF file.
F     PLTLOGO: logical switch draw Logo.
T     WRTHDR : logical switch to write out the plot header titles.
T     WRTBLAB: logical switch to write out the plot bottom title.
T     WRTRANG: logical switch to write out data range values and CI.
T     WRTFNAM: logical switch to write out input primary filename.
T     WRTDATE: logical switch to write out current date.
F     CST    : logical switch to read and plot coastlines and islands.
0.0 0.0      : bottom and top map latitudes (south values are negative).
0.0 0.0      : left and right map longitudes (west values are negative).
/home/arango/scrum3.0/plot/Data/default.cnt
scrum_his.nc
scrum_his.nc
scrum_grd.nc
/dev/null

c
c======================================================================
c  Copyright (c) 1996 Rutgers University                             ===
c======================================================================
c

 *** Above FILENAMES:

            1nd line: input; contour parameters.
            2rd line: input; primary NetCDF file.
            3th line: input; secondary NetCDF file.
            4th line: input; grid NetCDF file.
            5th line: input; coastlines file.

 *** IREF:  Secondary or reference field option:
           -1   Overlay horizontal grid
            0   no secondary or reference field to plot
```

```
            1   plot field overlay from primary file
            2   plot field overlay from secondary file
            3   primary - secondary file (field subtraction)
            4   Day0 - DayN (field subtraction)

 *** NPAGE: Set this parameter to a negative value (-1, -2, or -4) to
            activate preservation of the plot aspect ratio.


  Plotting Fields classification: (* derived fields)

 [  1]  IDUTOT  total velocity component in the XI-direction (cm/s).
 [  2]  IDVTOT  total velocity component in the ETA-direction (cm/s).
*[  3]  IDTVEC  total velocity vectors (cm/s).
*[  4]  IDTMAG  total velocity vector magnitude (cm/s).
                 :
                 :
 [120]  IDSVSE  Wind stress in the ETA-direction, observation error.
```

As you can see, there are comments which describe what needs to be done. Please see the comments at the bottom of the input files for the rest of the fields that can be plotted—this list changes as Hernan adds the code to plot new fields.

Appendix A

Model Timestep

Numerical timestepping uses a discrete approximation to:

$$\frac{\partial \phi(t)}{\partial t} = \mathcal{F}(t) \tag{A.1}$$

where ϕ represents one of u, v, T, S or ζ and $\mathcal{F}(t)$ represents all the right-hand-side terms. The simplest approximation is the Euler timestep:

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

where you predict the next ϕ 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 $\mathrm{O}(\Delta t^2)$:

$$\frac{\phi(t+\Delta t) - \phi(t-\Delta t)}{2\Delta t} = \mathcal{F}(t). \tag{A.3}$$

This timestep is more accurate, but it is unconditionally unstable with respect to diffusion. Also, the even and odd timesteps tend to diverge in a computational mode. This computational mode can be damped by taking correction steps. SCRUM's timestep on the depth-integrated equations is a leapfrog step with a trapezoidal correction on every step, which uses a leapfrog step to obtain an initial guess of $\phi(t+\Delta t)$. We will call the right-hand-side terms calculated from this initial guess $\mathcal{F}^*(t+\Delta t)$:

$$\frac{\phi(t+\Delta t) - \phi(t)}{\Delta t} = \frac{1}{2}\left[\mathcal{F}(t) + \mathcal{F}^*(t+\Delta t)\right]. \tag{A.4}$$

This leapfrog-trapezoidal timestep is stable with respect to diffusion and it strongly damps the computational mode. However, the right-hand-side terms are computed twice per timestep.

The timestep on SCRUM's full 3-D fields is done with a third-order Adams-Bashforth step. It uses three time-levels of the right-hand-side terms:

$$\frac{\phi(t+\Delta t) - \phi(t)}{\Delta t} = \alpha\mathcal{F}(t) + \beta\mathcal{F}(t-\Delta t) + \gamma\mathcal{F}(t-2\Delta t) \tag{A.5}$$

where the coefficients α, β and γ are chosen to obtain a third-order estimate of $\phi(t+\Delta t)$. We use a Taylor series expansion:

$$\frac{\phi(t+\Delta t)-\phi(t)}{\Delta t} = \phi' + \frac{\Delta t}{2}\phi'' + \frac{\Delta t^2}{6}\phi''' + \cdots \tag{A.6}$$

where

$$\mathcal{F}(t) = \phi' \tag{A.7}$$

$$\mathcal{F}(t-\Delta t) = \phi' - \Delta t\phi'' + \frac{\Delta t^2}{2}\phi''' + \cdots \tag{A.8}$$

$$\mathcal{F}(t-2\Delta t) = \phi' - 2\Delta t\phi'' + 2\Delta t^2\phi''' + \cdots \tag{A.9}$$

We find that the coefficients are:

$$\alpha = \frac{23}{12} \tag{A.10}$$

$$\beta = -\frac{4}{3} \tag{A.11}$$

$$\gamma = \frac{5}{12} \tag{A.12}$$

The model carries one time level for the physical fields and three time levels of the right-hand-side information. The initial fields are read in but the right-hand-sides are not stored; an Euler timestep is used for the first two steps to get things going.

Appendix B

The vertical s-coordinate

Following Song and Haidvogel [33], the vertical coordinate has been chosen to be:

$$z = \zeta(1+s) + h_c s + (h - h_c)C(s), \qquad -1 \le s \le 0 \tag{B.1}$$

where h_c is either the minimum depth or a shallower depth above which we wish to have more resolution. $C(s)$ is defined as:

$$C(s) = (1-b)\frac{\sinh(\theta s)}{\sinh\theta} + b\frac{\tanh[\theta(s+\frac{1}{2})] - \tanh(\frac{1}{2}\theta)}{2\tanh(\frac{1}{2}\theta)} \tag{B.2}$$

where θ and b are surface and bottom control parameters. Their ranges are $0 < \theta \le 20$ and $0 \le b \le 1$, respectively. Equation (B.1) leads to $z = \zeta$ for $s = 0$ and $z = h$ for $s = -1$.

Some features of this coordinate system:

- It is a generalization of the σ-coordinate system. Letting θ go to zero and using L'Hopital's rule, we get:
$$z = (\zeta + h)(1+s) - h \tag{B.3}$$
which is the σ-coordinate.
- It has a linear dependence on ζ and is infinitely differentiable in s.
- The larger the value of θ, the more resolution is kept above h_c.
- For $b = 0$, the resolution all goes to the surface as θ is increased.
- For $b = 1$, the resolution goes to both the surface and the bottom equally as θ is increased.
- For $\theta \neq 0$ there is a subtle mismatch in the discretization of the model equations, for instance in the horizontal viscosity term. We recommend that you stick with "reasonable" values of θ, say $\theta \le 5$.
- Some problems turn out to be sensitive to the value of θ used.

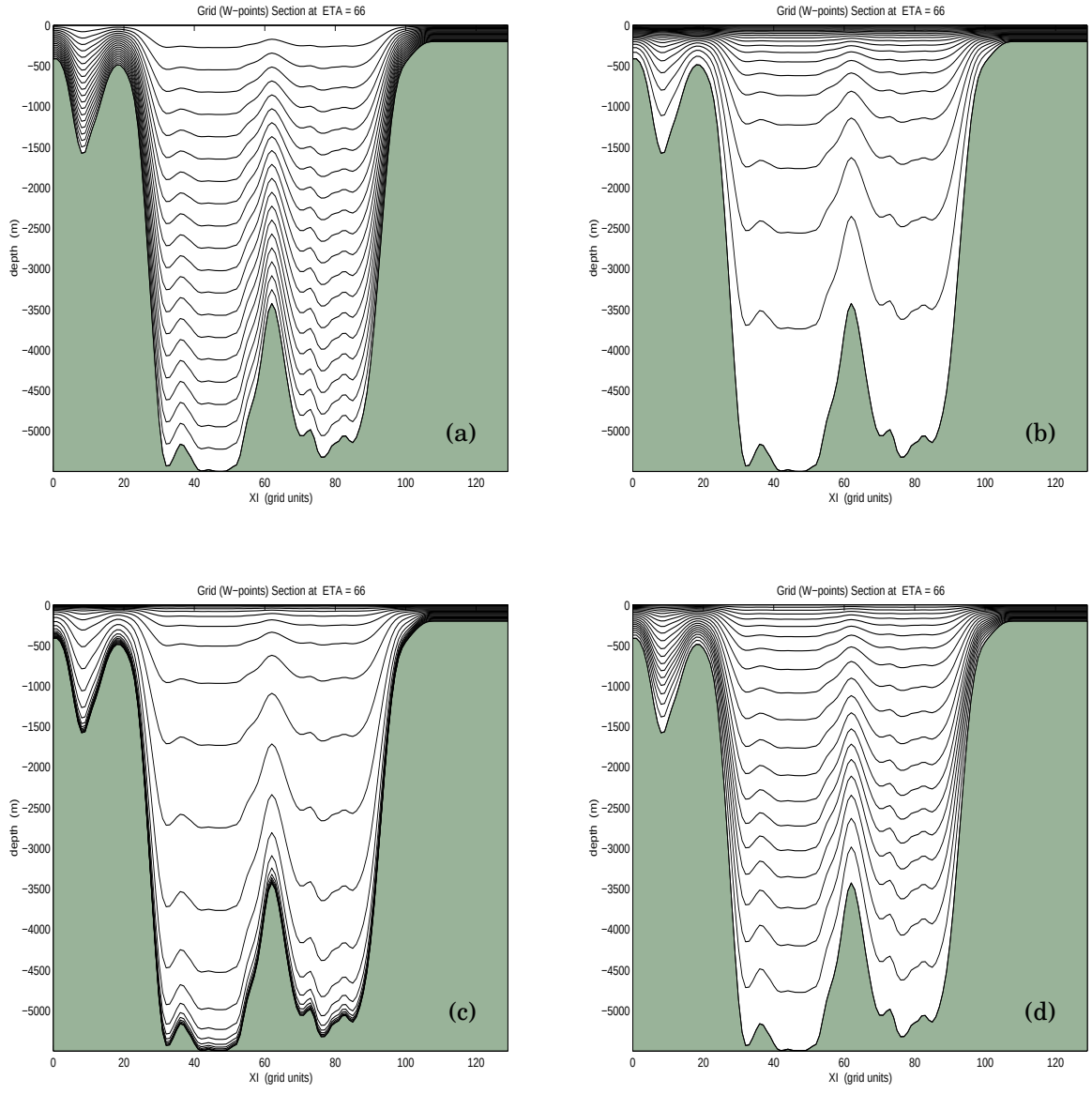

Figure B.1: The s-surfaces for the North Atlantic with (a) $\theta = 0.0001$ and $b = 0$, (b) $\theta = 8$ and $b = 0$, (c) $\theta = 8$ and $b = 1$. (d) The actual values used in this domain were $\theta = 5$ and $b = 0.4$.

Figure B.1 shows the s-surfaces for several values of θ and b for one of our domains. It was produced by a Matlab tool written by Hernan Arango which is available from our web site (see §1.1).

We find it convenient to define:

$$H_z \equiv \frac{\partial z}{\partial s} = (\zeta + h) + (h - h_c)\frac{\partial C(s)}{\partial s}. \tag{B.4}$$

The derivative of $C(s)$ can be computed analytically:

$$\frac{\partial C(s)}{\partial s} = (1-b)\frac{\cosh(\theta s)}{\sinh\theta}\theta + b\frac{\coth(\frac{1}{2}\theta)}{2\cosh^2[\theta(s+\frac{1}{2})]}\theta. \tag{B.5}$$

However, we choose to compute H_z discretely as $\Delta z/\Delta s$ since this leads to the vertical sum of H_z being exactly the total water depth D.

B.1 Horizontal curvilinear coordinates

The requirement for a boundary-following coordinate system and for a laterally variable grid resolution can both be met (for suitably smooth domains) by introducing an appropriate orthogonal coordinate transformation in the horizontal. Let the new coordinates be $\xi(x, y)$ and $\eta(x, y)$ where the relationship of horizontal arc length to the differential distance is given by:

$$(ds)_\xi = \left(\frac{1}{m}\right) d\xi \tag{B.6}$$

$$(ds)_\eta = \left(\frac{1}{n}\right) d\eta \tag{B.7}$$

Here, $m(\xi, \eta)$ and $n(\xi, \eta)$ are the scale factors which relate the differential distances $(\Delta\xi, \Delta\eta)$ to the actual (physical) arc lengths.

It is helpful to write the equations in vector notation and to use the formulas for div, grad, and curl in curvilinear coordinates (see Batchelor, Appendix 2, [2]):

$$\nabla\phi = \hat{\xi} m \frac{\partial\phi}{\partial\xi} + \hat{\eta} n \frac{\partial\phi}{\partial\eta} \tag{B.8}$$

$$\nabla \cdot \vec{a} = mn \left[\frac{\partial}{\partial\xi}\left(\frac{a}{n}\right) + \frac{\partial}{\partial\eta}\left(\frac{b}{m}\right)\right] \tag{B.9}$$

$$\nabla \times \vec{a} = mn \begin{vmatrix} \frac{\hat{\xi}_1}{m} & \frac{\hat{\xi}_2}{n} & \hat{k} \\ \frac{\partial}{\partial\xi} & \frac{\partial}{\partial\eta} & \frac{\partial}{\partial z} \\ \frac{a}{m} & \frac{b}{n} & c \end{vmatrix} \tag{B.10}$$

$$\nabla^2\phi = \nabla \cdot \nabla\phi = mn \left[\frac{\partial}{\partial\xi}\left(\frac{m}{n}\frac{\partial\phi}{\partial\xi}\right) + \frac{\partial}{\partial\eta}\left(\frac{n}{m}\frac{\partial\phi}{\partial\eta}\right)\right] \tag{B.11}$$

where ϕ is a scalar and $\vec{a}$ is a vector with components a, b, and c.

Appendix C

Viscosity and Diffusion

C.1 Horizontal viscosity

The horizontal viscosity and diffusion coefficients are scalars which are read in from **scrum.in**. Several factors to consider when choosing these values are:

spindown time The spindown time on wavenumber k is $\frac{1}{k^2\nu_2}$ for the Laplacian operator and $\frac{1}{k^4\nu_4}$ for the biharmonic operator. The smallest wavenumber corresponds to the length $2\Delta x$ and is $k = \frac{\pi}{\Delta x}$, leading to

$$\Delta t < t_{damp} = \frac{\Delta x^2}{\pi^2\nu_2} \quad \text{or} \quad \frac{\Delta x^4}{\pi^4\nu_4} \tag{C.1}$$

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

$$\Delta x < L_{BL} = \left(\frac{\nu_2}{\beta}\right)^{\frac{1}{3}} \quad \text{and} \quad \left(\frac{\nu_4}{\beta}\right)^{\frac{1}{5}} \tag{C.2}$$

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.

C.2 Horizontal Diffusion

We have chosen anything from zero to the value of the horizontal viscosity for the horizontal diffusion coefficient. One common choice is an order of magnitude smaller than the viscosity.

C.3 Vertical Viscosity and Diffusion

SCRUM stores the vertical mixing coefficients in arrays with three spatial dimensions called **Akv** and **Akt**. **Akt** also has a fourth dimension specifying which tracer, so that temperature and salt can have differing values. Both **Akt** and **Akv** are stored at w-points in the

model; horizontal averaging is done to obtain **Akv** at the horizontal u and v-points. The values for these coefficients can be set in a number of ways, depending on the flags chosen in **cppdefs.h**:

ANA_VMIX An analytic formula must be defined in **ana_vmix** in **analytical.F**.

BVF_MIXING This is a simple formula where **Akt** is proportional to $1/N$, where N is the Brunt-Väisälä frequency. This was provided by Bernard Barnier, who used it in his modeling of the Atlantic Ocean.

PP_MIXING This specifies the Pacanowski and Philander [24] scheme, which depends on the local Richardson number.

MY2_MIXING This specifies the Mellor and Yamada [22] level 2 closure.

MY25_MIXING This specifies the Mellor and Yamada [22] level 2.5 closure. It requires the model to track an additional tracer, q.

LMD_MIXING This specifies the Large et al. [18] planetary boundary layer.

Appendix 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. I have put this one in pub/util/cpp.tar.gz on the ahab.rutgers.edu ftp site since it supports the **#elif** construct. 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-sun-solaris2.5/2.7.2/cpp
```

where **sparc** is the architecture, **sun** is the manufacturer, **solaris2.5** is the operating system and version, and **2.7.2** is **gcc**'s version number.

This Appendix describes the C preprocessor as used in SCRUM with the Fortran language. A more complete description is given by Kernighan and Ritchie [17]. More practical advice on using **cpp** is given by Hazard [14].

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 SCRUM 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 SCRUM as a reasonably portable way to get 64-bit precision:

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

It is customary to use uppercase for **cpp** macros—the C preprocessor is case sensitive.

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  MASKING
```

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

D.3 Conditional inclusion

It is possible to control which parts of the code are seen by the Fortran compiler by the use of **cpp**'s conditional inclusion. For example, the statements

```
#ifdef MASKING
# include "mask.h"
#endif /* MASKING */
        :
# ifdef MASKING
c
c  Apply Land/Sea mask: slipperiness.
c
      do j=1,M
        do i=2,Lm
          Uflux(i,j)=Uflux(i,j)*pmask(i,j)
        enddo
      enddo
# endif /* MASKING */
```

will not be in the Fortran source code if **MASKING** has not been defined. Likewise, **#ifndef** tests for a macro being undefined:

```
      #ifndef RMDOCINC
      c  rmask       Mask at RHO-points (0=Land, 1=Sea).
      c  pmask       Slipperiness mask at PSI-points (0=Land, 1=Sea,
      c                                                 1-gamma2=boundary).
      c  umask       Mask at U-points (0=Land, 1=Sea).
      c  vmask       Mask at V-points (0=Land, 1=Sea).
      c
      c=======================================================================
      #endif
```

There are also **#else** and **#elif** (else if) statements, although **#elif** is newer and is not supported by all versions of **cpp**. An example using **#else** and **#elif** is shown:

```
#if defined BASIN
      parameter (L=181, M=141, N=12, NT=1)
#elif defined CANYON_A
      parameter (L=66, M=49, N=10, NT=1)
#elif defined CANYON_B
      parameter (L=66, M=49, N=15, NT=1)
           :
#elif defined UPWELLING
      parameter (L=42, M=81, N=16, NT=2)
#else
      parameter (L=???, M=???, N=??, NT=?)
#endif
```

Actually, **#ifdef** is a restricted version of the more general test

```
      #if     expression
```

where "expression" is a constant integer value. Zero evaluates to **false** and everything else is considered **true**. Compound expressions may be built using the C logical operators:

```
        &&          logical and
        ||          logical or
        !           logical not
```

These symbols would be used as in the following example:

```
      #if defined CANYON_A || defined CANYON_B
            do j=0,M
              do i=0,L
                yc=c32000-c16000*(sin(pi*xr(i,j)/xl))**24
                h(i,j)=c20+p5*(hmax-c20)*(c1+tanh((yr(i,j)-yc)/c10000))
              enddo
            enddo
      #endif
```

D.4 C comments

The C preprocessor will also delete C language comments starting with /* and ending with */ as in:

```
#endif  /* MASKING */
```

When mixed with Fortran code, it is safer to use a Fortran comment.

D.5 Potential problems

The use of the C preprocessor is not entirely free of problems, but many can be worked around or avoided by using the Der Mouse version of **cpp**.

1. Apostrophes in Fortran comments. **cpp** does not know that it is in a comment and some versions will complain about unmatched apostrophes in the following:

   ```
   c  Some useful comment about Green's functions.
   ```

   The **gnu** version of **cpp** (which comes with **gcc**) has a **-traditional** option which makes it more appropriate for use with Fortran.

2. C++ comments. Some of the newer versions of **cpp** will remove C++ comments which go from '//' to the end of the line. Some perfectly reasonable Fortran lines contain two consecutive slashes, such as:

   ```
       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.

Also, make sure that your macros will not be found anywhere else in the code. I used to use **#define DOUBLE** for double precision until it was pointed out to me that **DOUBLE PRECISION** is perfectly valid Fortran. The macro processor would turn this into **1 PRECISION** since something that is defined has a value of 1.

D.6 Modern Fortran

I started working on these ocean models before 1990, much less before Fortran 90 compilers were generally available. Fortran 90's **kind** feature would be a better way to handle the **BIGREAL** type declarations. On the other hand, Fortran 90 does not include conditional compilation. However, it is deemed useful enough that the Fortran 2000 committee has a draft document describing how Fortran might support conditional compilation. We *might* start using this in about ten years.

Appendix E

The patch program

We sometimes discover things in SCRUM which we would like to modify, either to fix bugs, or to add new features. Hernan Arango keeps track of these changes and periodically sends patches to the list of known SCRUM users so that they can update their versions. By sending out these changes rather than the whole updated model, people can acquire bug fixes and still retain the changes they have made to SCRUM for their own applications.

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 **scrum.patch.20** then **patch** would be used as follows:

```
patch < scrum.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) 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.

Appendix F

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 SCRUM and its related programs. See Oram and Talbott [23] 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 SCRUM, 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** script, which searches the source files for **#include** directives (see Appendix G). Note that if you add your own source files with **#include** statements, you will need to rerun **make depend** to update the dependency list. If your files are not in the list of SCRUM sources, they will have to be added to the **depend:** entry in the **Makefile/Imakefile** first.

F.0.1 imake

Since it is difficult to keep consistent **Makefiles** for several different computers, it was suggested that we try **imake**, which is distributed with the X window system. It helps you to separate the system dependent parts of a **Makefile** into configuration files (kept in a central location) and project dependent parts called **Imakefiles**. Then, when you want to make SCRUM on a Cray computer, you combine the SCRUM **Imakefile** with the Cray configuration file to create the appropriate **Makefile**. This is done by the shell script **fmkmf**, which takes the computer type as its argument:

```
fmkmf Cray
```

This will generate **Makefile.Cray**. If you provide an unrecognized computer type then the **generic.cf** file will be used. The list of recognized computers is growing and currently includes:

Alpha DEC Alpha running OSF/1.

CM2 Connection Machine.

CF90 Cray with **f90** and UNICOS.

Cray Cray with **cf77** and UNICOS.

F90 The NAG Fortran 90 compiler in free format style.

HP Hewlett-Packard 9000/700 family.

Gnu Gnu Fortran.

NAG The NAG Fortran 90 compiler in fixed (old) format style.

RS6000 IBM RS/6000 with **xlf** and AIX.

RS6000old IBM RS/6000 with an older **xlf** and AIX.

SGI SGI with IRIX and **f77**.

Solaris Sun Sparc with Solaris 2.x.

Sun Sun Sparc with SunOS 4.x.

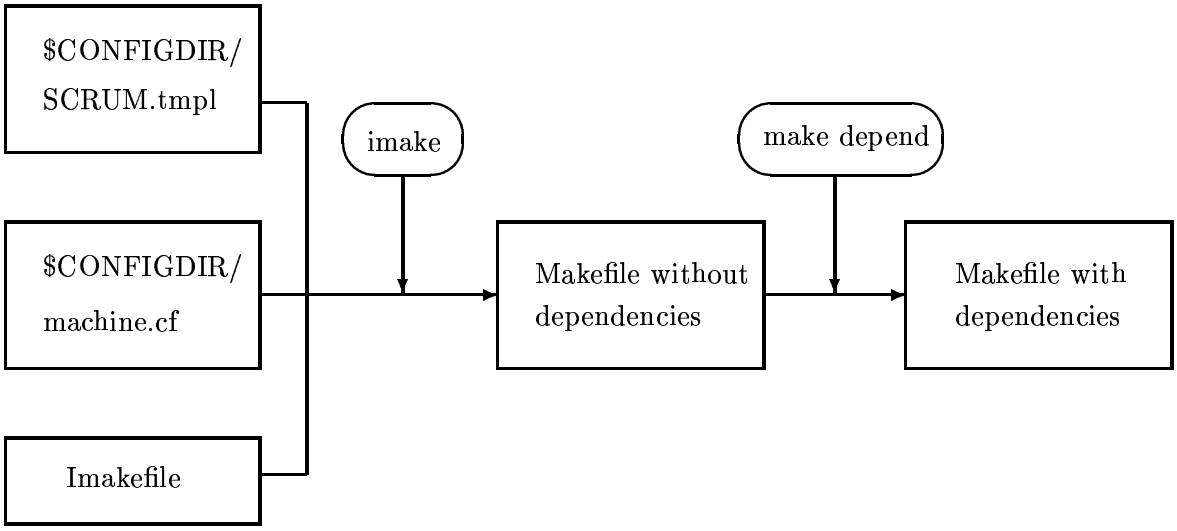

Figure F.1: Creating **Makefiles**

Titan Kubota Titan 3000 with **fc**.

It would be possible to have a different configuration file for each Cray you use, or for different versions of the Sun compiler.

The **fmkmf** script executes **imake** followed by **make depend**, as shown in Fig. F.1. It requires the **$CONFIGDIR** variable to be set to where the configuration files are kept. Also, the **make depend** phase executes a **perl** script that requires the **perl** program. The configuration files and the **fmkmf** script are distributed as described in §1.1 and **perl** is available from all the **gnu** archive sites.

We could have chosen to have the **fmkmf** script try to determine which type of computer it was being run on, and use the appropriate configuration file. However, it may be that we decide to run SCRUM on a computer which does not have **imake** or **perl**. This way, we can generate the **Makefile** on a computer which has the necessary support programs and then just transfer the **Makefile** along with the SCRUM code.

F.0.2 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 old 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 SCRUM. 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.

Appendix G

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 et al. [35], while a more tutorial approach is given by Schwartz [29]. **Perl** itself is available from your nearest **CPAN** archive site, for instance:

```
http://www.perl.com/CPAN/
```

I have several **Perl** scripts which I find useful when working with Fortran programs, and which are available from:

```
http://marine.rutgers.edu/po/perl.html
```

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 SCRUM, but have been known to do the wrong thing. Some of the scripts become confused when part of an **if** or **do** statement is inside an **#ifdef** clause. The following will parse as two nested do loops, only one of which is terminated:

```
#ifdef EW_PERIODIC
      do i=1,Lm
#else
      do i=0,L
#endif
        :
      enddo
```

It is also extremely dangerous to run **relabel** on an arithmetic **if**.

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

G.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.

G.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
```

There is an option to change the number of spaces for each level of indenting. To get an indent of four instead of two, use:

```
findent -n 4 < file.F > file.new
```

See the comments at the top of the code for the more obscure options.

G.3 relabel

relabel was written by Sverre Froyen to replace the numbered Fortran labels with new sequentially ordered labels. It was the first of these scripts and helped me to write the rest. **relabel** is used as follows:

```
relabel < file.F > file.new
```

It does have some known bugs, however:

- No computed goto.
- No assigned goto or assign.
- No arithmetic if.
- No new-lines inside the parenthesis immediately following a read/write.
- Others not yet discovered.

All the source files in SPEM have been run through **redo**, **findent**, and **relabel**. In the C shell (**csh**), a series of files can be processed at once:

```
ahab% foreach file (*.F)
foreach? redo < $file > $file.red
foreach? findent < $file.red > $file.fin
foreach? relabel < $file.fin > $file.rel
foreach? echo $file done
foreach? end
```

You can then rename **$file.rel** to **$file.F** and get rid of the temporary files *after* checking to make sure that all the files still look sensible.

G.4 unenddo

unenddo will turn all **do-enddo** loops into **do-continue** loops to comply with the Fortran 77 standard. It is used as follows:

```
unenddo < file.F > file.new
```

unenddo replaces **enddo** statements with labelled **continue** statements. It starts numbering these statements at 2000 assuming that existing labels use only three digits. If desired, **unenddo** can be told to start labelling with a different number by modifying the **$label_no_start** variable.

G.5 ifspace

When I am feeling particularly contentious I also run **ifspace** on the code. It will convert

```
if(i.eq.0.or.j.eq.0)   then
```

to

```
if (i .eq. 0 .or. j .eq. 0) then
```

Use as follows:

```
ifspace < file.F > file.new
```

G.6 sfmakedepend

The other **Perl** script I use with Fortran modifies the **Makefile** to include dependency information, much like the X11 program **makedepend**. I originally wrote **fmakedepend** which was used with traditional Fortran include statements. I later wrote a variant of it for use with the C preprocessor, called **sfmakedepend**. The latest version of **sfmakedepend** does the job of both programs and also searches for the dependencies introduced by Fortran 90 modules. It is used by the **Makefiles** described in §F.

It recursively searches for Fortran style includes, for instance if **file.f** has the statement:

```
include 'commons.h'
```

the line

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

will be added to the bottom of the **Makefile**. This tells **make** that **file.o** depends on **commons.h** as well as **file.f**, and to recompile **file.f** whenever **commons.h** is modified. It likewise searches source files 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.

When using the C preprocessor, you can ask it to search directories other than the current directory. Likewise, **sfmakedepend** can be instructed to search other directories with **-I dir** options. Note that it is legal to have more than one **-I dir** option as in:

```
sfmakedepend -I /usr/local/include -I /home/me/include *.F
```

Fortran 90 introduces some interesting dependencies. Two compilers I have access to (NAG **f90** and IBM **xlf**) produce a private **my_module.mod** file if you define **module My_Module** in file **mod.f90**. This file is used by the compiler when you use the module as a consistency check (type-safe programming). If **foo.f90** uses that module, you will need the following dependency information:

```
foo.o: my_module.mod
my_module.mod: mod.o
```

This says that before compiling **foo.f90** we need to have the file **my_module.mod**. This file in turn depends on **mod.o**, so that **mod.f90** must be compiled before **foo.f90**. The sgi is similar except that it uses the file **MY_MODULE.kmo** to store the private module information. Use **sfmakedepend -g** on the SGI.

Rather than creating extra module files, the Cray and Parasoft compilers store the module information in the object file and then files which use the modules need to be compiled with extra flags pointing to the module object files. For instance, if **foo.f90** uses **My_Module** which was defined in **mod.f90**, then you will need to compile **mod.f90** first and provide the Cray compiler with the extra option **-p mod.o** when compiling **foo.f90**. When using the Cray, use **sfmakedepend -c** to get the dependency information:

```
foo.o: mod.o
        $(CFT) $(FFLAGS) -c -p mod.o foo.f90
```

$(CFT) and **$(FFLAGS)** are assumed to be previously defined as the name of the compiler and the compiler options, respectively.

Note: These f90 module dependencies can confuse some versions of **make**, especially of the System V variety. We use gnu **make** because it can follow these chained dependencies and do the right thing.

sfmakedepend assumes that all the files using and defining modules are in the same directory and are all in the list of files to be searched. It seems that the industry has not settled on a practical way to deal with a separate modules directory, anyway.

I sometimes include non-existent files as a compile time consistency check:

```
#ifndef PLOTS
#include "must_define_PLOTS"       /* bogus include */
#endif
```

This program warns about include files it can't find, but not if there is a "bogus" on the same line.

See the comments at the top of **sfmakedepend** for up-to-date information on the options. I may someday get inspired to use a newer version of the **getopt** routine and rename the options to have names like **-SGI** and **-Cray**.

Bibliography

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

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

[3] 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.

[4] A. F. Bennett. *Inverse methods in physical oceanography.* Cambridge University Press, 1992.

[5] F. P. Bretherton, R. E. Davis, and C. B. Fandry. A technique for objective analysis and design of oceanographic experiments applied to mode-73. *Deep Sea Res.*, 23:559–582, 1976.

[6] E. F. Carter and A. R. Robinson. Analysis models for the estimation of oceanic fields. *J. Atmos. Ocean. Tech.*, 4:49–74, 1987.

[7] R. Daley. *Atmospheric data analysis*, chapter 5. Cambridge University Press, 1991.

[8] 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.

[9] L. S. Gadin. *The objective analysis of meteorological fields.* Hydrometeorological Publishing House, Leningrad, 1963. English translation: Israel Program for Scientific Translations, Jerusalem, 1965.

[10] B. Galperin, L. H. Kantha, S. Hassid, and A. Rosati. A quasi-equilibrium turbulent energy model for geophysical flows. *J. Atmos. Sci.*, 45:55–62, 1988.

[11] D. B. Haidvogel and A. Beckmann. Numerical models of the coastal ocean. *The Sea*, 1997. in press.

[12] D. B. Haidvogel and A. Beckmann. *Numerical Ocean Circulation Modeling.* Imperial College Press, 1998. partially written.

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

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

[15] M. Iskandarani, D.B. Haidvogel, and J.P. Boyd. A staggered spectral element model with applications to the oceanic shallow water equations. *Int. J. Num. Meth. Fl.*, 20:393–414, 1995.

[16] D. R. Jackett and T. J. McDougall. Stabilization of hydrographic data. *J. Atmos. Ocean. Tech.*, 12:381–389, 1995.

[17] B. W. Kernighan and D. M. Ritchie. *The C Programming Language.* Prentice Hall, Englewood Cliffs, New Jersey 07632, second edition, 1988.

[18] W. G. Large, J. C. McWilliams, and S. C. Doney. Oceanic vertical mixing: a review and a model with a nonlocal boundary layer parameterization. *Rev. Geophys.*, 32:363–403, 1994.

[19] Anthony Macks and Jason Middleton. Numerical modelling of wind-driven upwelling and downwelling. University of New South Wales, 1993.

[20] John D. McCalpin. A comparison of second-order and fourth-order pressure gradient algorithms in a σ-coordinate ocean model. *Int. J. Num. Meth. Fl.*, 18:361–383, 1994.

[21] J. C. McWilliams, W. B. Owens, and B. L. Hua. An objective analysis of the polymode local dynamics experiment. part i: general formalism and statistical model selection. *J. Phys. Oceanogr.*, 16:483–504, 1986.

[22] G. L. Mellor and T. Yamada. A hierarchy of turbulence closure models for planetary boundary layers. *J. Atmos. Sci.*, 31:1791–1806, 1974.

[23] A. Oram and S. Talbott. *Managing Projects with make.* O'Reilly & Associates, Inc., Sebastopol, CA, 1991.

[24] R. C. Pacanowski and G. H. Philander. Parameterization of vertical mixing in numerical models of tropical oceans. *J. Phys. Oceanogr.*, 11:1443–1451, 1981.

[25] N. A. Phillips. A coordinate system having some special advantages for numerical forecasting. *J. Meteorology*, 14(2):184–185, 1957.

[26] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes, The Art of Scientific Computing.* Cambridge University Press, 1986.

[27] R. Rew, G. Davis, S. Emmerson, and H. Davies. *NetCDF User's Guide.* Unidata, University Corporation for Atmospheric Research, Boulder, Colorado, 1996. Version 2.4.

[28] R. D. Richtmeyer and K. W. Morton. *Difference Methods for Initial-Value Problems.* Interscience Publishers, J. Wiley and Sons, New York, New York, second edition, 1967.

[29] R. L. Schwartz. *Learning perl.* O'Reilly & Associates, Inc., Sebastopol, CA, 1993. the llama book.

[30] P. K. Smolarkiewicz. A simple positive definite advection scheme with small implicit diffusion. *Mon. Wea. Rev.*, 111:479–486, 1983.

[31] P. K. Smolarkiewicz and W. W. Grabowski. The multidimensional positive definite advection transport algorithm: non-oscillatory option. *J. Comp. Phys.*, 86:355–375, 1990.

[32] P. K. Smolarkiewicz and L. G. Margolin. On forward-in-time differencing for fluids: extension to a curvilinear framework. *Mon. Wea. Rev.*, 121:1847–1859, 1993.

[33] Y. Song and D. B. Haidvogel. A semi-implicit ocean circulation model using a generalized topography-following coordinate system. *J. Comp. Phys.*, 115(1):228–244, 1994.

[34] R. Styles and S. M. Glenn. Observation and modeling of sediment transport events in the middle atlantic bight. In *8th International conference on Physics of estuaries and coastal seas*, 1996. submitted.

[35] L. Wall, T. Christiansen, and R. L. Schwartz. *Programming perl.* O'Reilly & Associates, Inc., Sebastopol, CA, second edition, 1996. the camel book.

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

[37] 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.*, 20:649–662, 1995.