

Technical Manual for a Coupled Sea-Ice/Ocean
Circulation Model (Version 2)

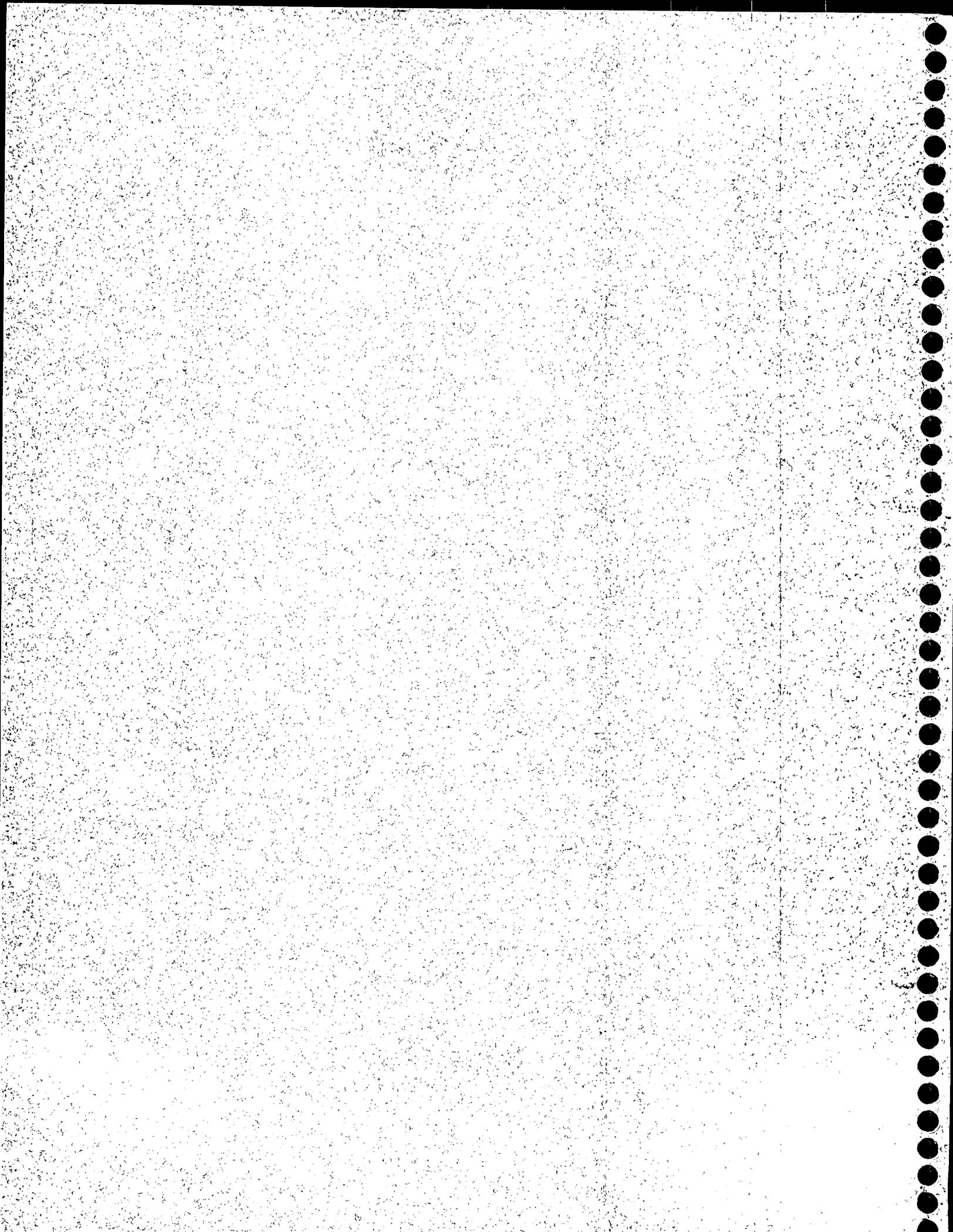
Katherine S. Hedström
Institute of Marine and Coastal Sciences
Rutgers University

U.S. Department of the Interior
Minerals Management Service
Anchorage, Alaska

Contract No. 14-35-01-96-CT-30818

U.S. Department of the Interior
Minerals Management Service
Alaska Outer Continental Shelf Region

MMS



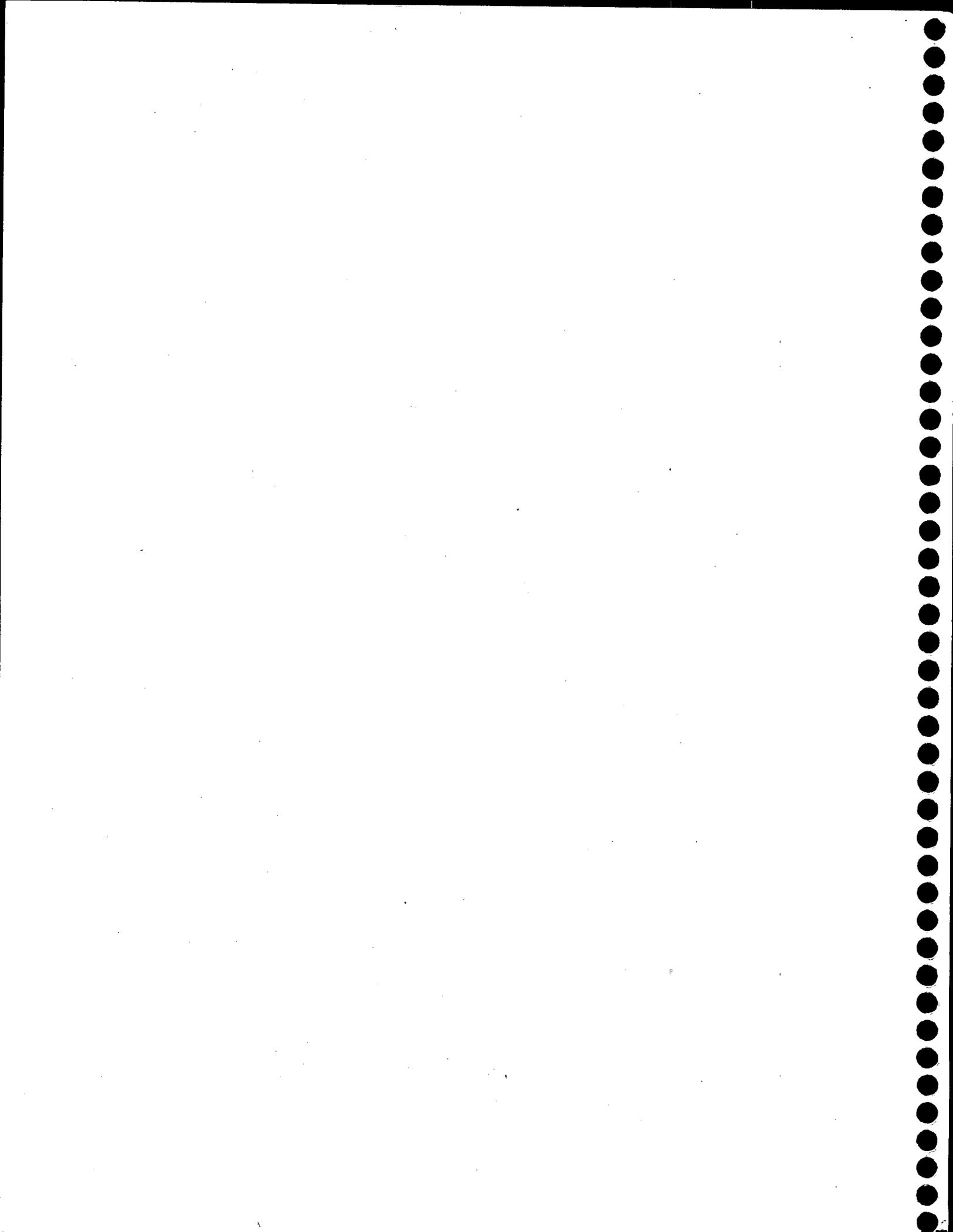
Technical Manual for a Coupled Sea-Ice/Ocean Circulation Model (Version 2)

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

June 2000

This study was funded by the Alaska Outer Continental Shelf Region of the Minerals Management Service, U.S. Department of the Interior, Anchorage, Alaska, through Contract **14-35-01-96-CT-30818** with Rutgers University, Institute of Marine and Coastal Sciences.

The opinions, findings, conclusions, or recommendations expressed in this report or product are those of the authors and do not necessarily reflect the views of the U.S. Department of the Interior, nor does mention of trade names or commercial products constitute endorsement or recommendation for use by the Federal Government.



This document was prepared with L^AT_EX 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 and provided me with its description. 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.

Bill Hibler first came up with the viscous-plastic rheology we are using. Paul Budgell has rewritten the dynamic sea-ice model, improving the solution procedure and making the water-stress term implicit in time. We are very grateful that he is allowing us to use his version of the code. The sea-ice thermodynamics is derived from Sirpa Häkkinen's implementation of the Mellor-Kantha scheme. She was kind enough to allow us to start with her code.

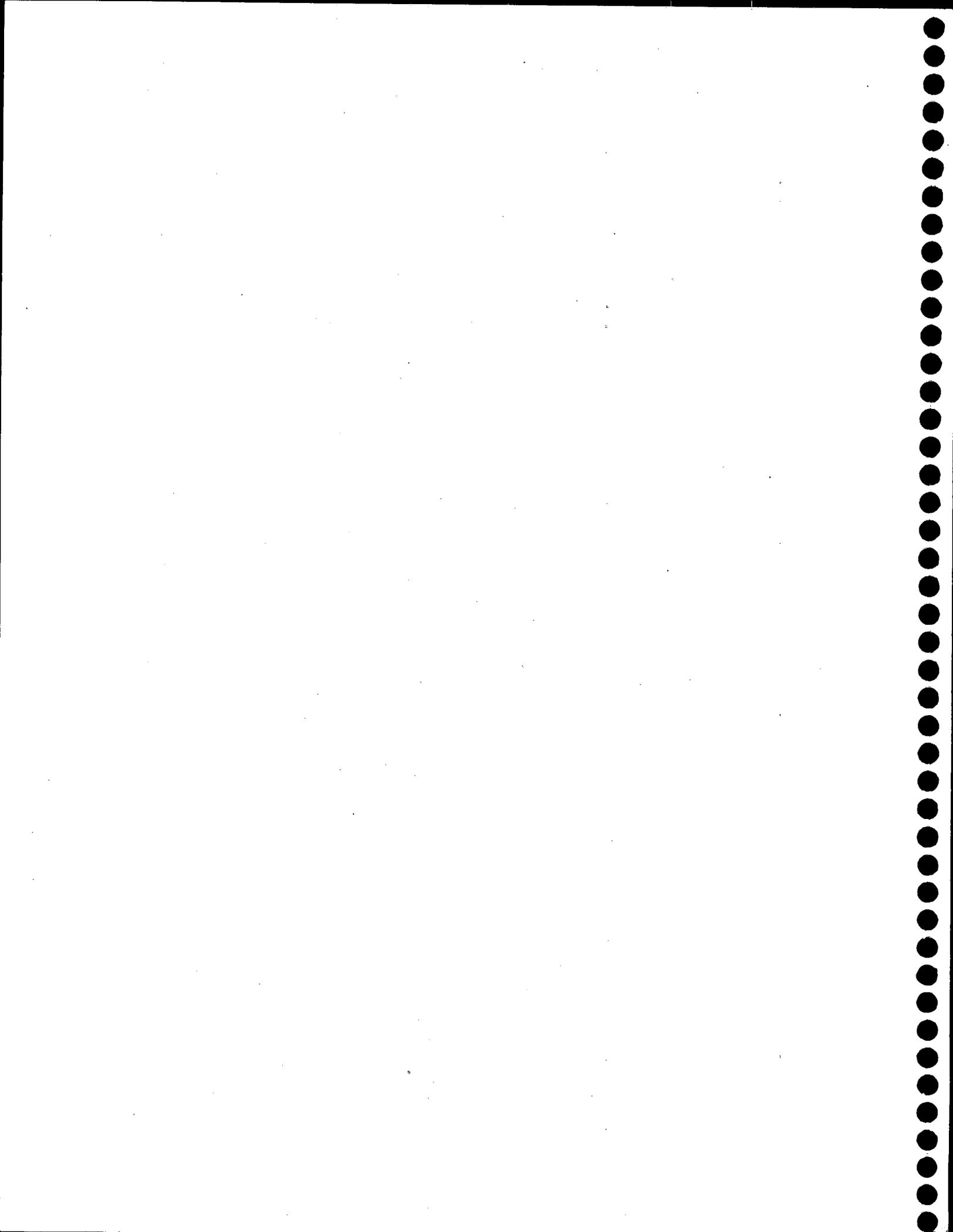
Thanks to the Usenet community for providing great tools like **perl**, **patch**, **cpp**, **rcs**, and **imake** to aid in software development (and to make it more fun). Cathy Lascara talked me into trying **imake** with SPEM which has been well worth the trouble.

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).

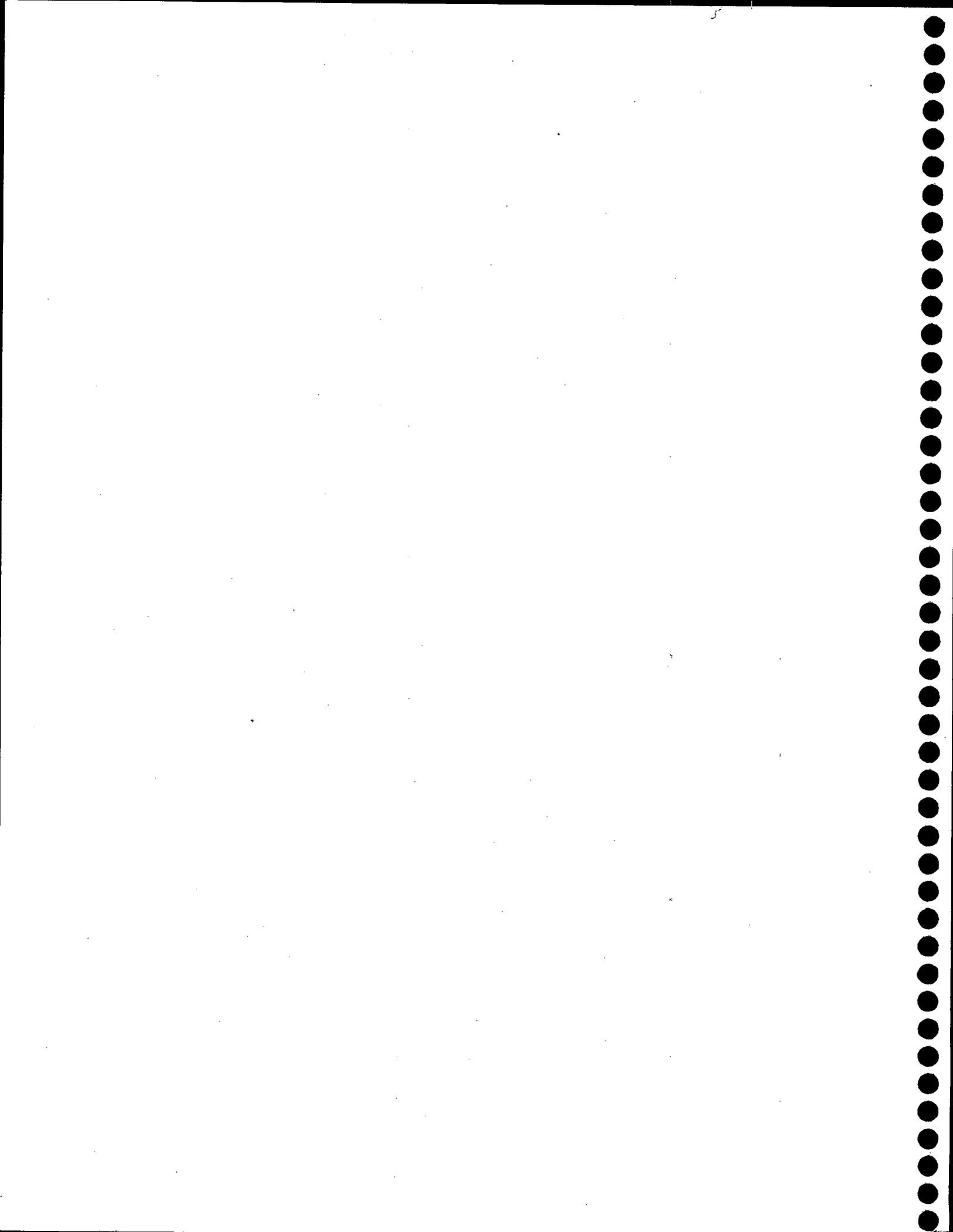
UNIX is a registered trademark of the Open Group.
Sun is a trademark of Sun Microsystems, Inc.
SGI is a trademark of Silicon Graphics, Inc.

This is Contribution #2000-07 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. It also describes the sea-ice model that is based on Hibler's viscous-plastic dynamics and the Mellor-Kantha thermodynamics.



Contents

1	Introduction	1
1.1	Acquiring the SCRUM code	2
1.2	The SCRUM email list	3
1.3	Future plans	3
1.4	Warnings and bugs	4
2	Ocean Model Formulation	5
2.1	Equations of motion	5
2.2	Vertical boundary conditions	6
2.3	Horizontal boundary conditions	7
2.4	s (stretched vertical) coordinate system	7
2.5	Horizontal curvilinear coordinates	9
3	Numerical Solution Technique	11
3.1	Vertical and horizontal discretization	11
3.1.1	Horizontal grid	11
3.1.2	Vertical grid	11
3.2	Masking of land areas	12
3.2.1	Velocity	12
3.2.2	Temperature, salinity and surface elevation	13
3.3	Conservation properties	13
3.4	Vertical viscosity and diffusion	14
3.5	Depth-integrated equations	15
3.6	Time stepping: internal velocity modes, temperature, and salinity	17
3.7	Advection schemes	17
3.7.1	Smolarkiewicz	17
3.7.2	Third-order Upwind	19
3.8	Determination of the vertical velocity and density fields	20
3.9	The pressure gradient terms	20
3.10	Horizontal friction and diffusion	20
3.10.1	Laplacian	21
3.10.2	Biharmonic	21
3.10.3	Rotated mixing tensors	21
3.11	Vertical mixing schemes	22
3.11.1	Brunt-Väisälä frequency scheme	23
3.11.2	Pacanowski-Philander	23
3.11.3	Mellor-Yamada	23
3.11.4	The Large, McWilliams and Doney parameterization	25
3.12	Open boundary conditions	28
3.12.1	Gradient boundary condition	28
3.12.2	Radiation boundary condition	28
4	Details of the Code	31
4.1	Main subroutines	31
4.2	Other subroutines and functions	34
4.3	C preprocessor variables	38
4.4	Important parameters	42

5	Support Programs for Initialization	43
5.1	Grid generation	43
5.1.1	ezgrid	43
5.1.2	gridpak	43
5.2	Masking	43
5.2.1	The scrum_mask program	43
5.3	Objective Analysis	44
5.4	Forcing fields	47
5.5	Initial and climatology fields	47
6	Configuring SCRUM for a Specific Application	49
6.1	Configuring SCRUM	49
6.1.1	cppdefs.h and checkdefs.F	50
6.1.2	Model domain	50
6.1.3	x, y grid	51
6.1.4	ξ, η grid	51
6.1.5	Initial conditions	51
6.1.6	Equation of state	51
6.1.7	Boundary conditions	53
6.1.8	Model forcing	53
6.1.9	scrum.in	53
6.1.10	User variables and subroutines	60
6.2	Upwelling/Downwelling Example	60
6.2.1	cppdefs.h	60
6.2.2	Model domain	61
6.2.3	ana_grid	61
6.2.4	Initial conditions and the equation of state	61
6.2.5	Boundary conditions	61
6.2.6	Model forcing	61
6.2.7	scrum.in	62
6.2.8	Output	62
6.3	North Atlantic example	65
6.3.1	cppdefs.h	65
6.3.2	Model domain	70
6.3.3	gridpak	71
6.3.4	Initial conditions	71
6.3.5	Boundary conditions	71
6.3.6	Forcing	71
6.3.7	Climatology	71
6.3.8	scrum.in	75
6.3.9	Output	75
7	Plotting Programs for Postprocessing	81
8	Ice Model Formulation	85
8.1	Model structure	85
8.2	Horizontal curvilinear coordinates	87
8.3	Numerical Scheme	89
8.4	Horizontal boundary conditions	90
8.5	Thermodynamics	90
8.5.1	Ocean surface boundary conditions	94

8.5.2	Frazil ice formation	95
8.5.3	Differences from Mellor and Kantha	96
9	Description of the Ice Model and the Coupling	97
9.1	Ice model structure	97
9.1.1	Thermodynamic subroutines	97
9.1.2	Initialization	99
9.1.3	Forcing fields	99
9.1.4	Other subroutines	100
9.2	Coupling strategy	100
9.3	C preprocessor variables	100
A	Model Timestep	103
B	The vertical <i>s</i>-coordinate	105
C	Horizontal curvilinear coordinates	107
D	Viscosity and Diffusion	109
D.1	Horizontal viscosity	109
D.2	Horizontal Diffusion	109
D.3	Vertical Viscosity and Diffusion	109
E	Radiant heat fluxes	111
E.1	Shortwave radiation	111
E.2	Longwave radiation	111
E.3	Sensible heat	111
E.4	Latent heat	111
F	The C preprocessor	113
F.1	File inclusion	113
F.2	Macro substitution	113
F.3	Conditional inclusion	114
F.4	C comments	115
F.5	Potential problems	115
F.6	Modern Fortran	116
G	The patch program	117
H	Makefiles	119
H.1	imake	119
H.2	Your Makefile	120
I	Perl scripts for Fortran	123
I.1	redo	123
I.2	findent	124
I.3	relabel	124
I.4	unenddo	124
I.5	ifspace	125
I.6	sfmakedepend	125

List of Figures

1	Tree structure of anonymous ftp directory	3
2	Placement of variables on an Arakawa C grid	11
3	Placement of variables on staggered vertical grid	11
4	Masked region within the domain	12
5	The split timestepping used in the model.	16
6	Flow chart of the model main program.	32
7	Flow chart of the initial subroutine.	33
8	Small grid with masked regions	44
9	The scrump_mask program in action	47
10	The whole grid.	52
11	The upwelling/downwelling bathymetry.	66
12	Surface velocities after one day, showing the flow to the left of the wind (southern hemisphere).	67
13	Constant ξ slices of the u, v, w and Ω fields at day 1.	68
14	Constant ξ slices of the T, S (tracer), kinetic energy and Ertel potential vorticity at day 1.	69
15	The North Atlantic grid.	72
16	The raw bathymetry from etopo5	73
17	The smoothed North Atlantic bathymetry.	74
18	The annual mean surface elevation for year 10.	79
19	Diagram of the different locations where ice melting and freezing can occur.	91
20	Diagram of internal ice temperatures and fluxes. The hashed layer is the snow.	91
21	Flow chart for the sea-ice model.	98
22	Flow chart for the coupled ice-ocean model.	101
23	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 $b = 0.4$	106
24	Creating Makefiles	121

List of Tables

1	The variables used in the description of the ocean model	5
2	The variables used in the vertical boundary conditions for the ocean model	6
3	Variables used in the ice momentum equations	86
4	Variables used in the ice thermodynamics	92
5	Ocean surface variables	94
6	Frazil ice variables	96
7	Variables used in computing the incoming radiation and latent and sensible heat	112

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 [54], while others are described in Haidvogel and Beckmann [19]. This manual also describes the sea-ice model that we are using, derived from that of Hibler [22]. It was rewritten and coupled to SCRUM by Paul Budgell.

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 third-order upwind advection scheme.
- Optional Smolarkiewicz advection scheme for tracers (potential temperature, salinity, etc.).
- Option for point sources and sinks.

Horizontal

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

Vertical

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

Ice

- Hibler viscous-plastic dynamics.
- Mellor-Kantha thermodynamics.
- Orthogonal-curvilinear coordinates.
- Arakawa C grid.
- Smolarkiewicz or third-order upwind advection of tracers.
- Optional ridging scheme.

Mixing options

- Horizontal Laplacian and biharmonic viscosity and diffusion along constant s , z or 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. [28], Mellor-Yamada [36], or Pacanowski-Philander [40] mixing schemes.

Implementation

- Dimensional in meter, kilogram, second (MKS) units.
- Fortran 77 and common extensions.
- Runs under UNIX, requires the C preprocessor.
- All input and output is done in NetCDF [47] (Network Common Data Format), requires the NetCDF library.
- Optimized for vector computers.
- Pre- and post-processing graphics package available which uses the NCAR (National Center for Atmospheric Research) graphics libraries.

Chapters 2 and 3 describe the model physics and numerical techniques and are an update of the description in Song and Haidvogel [54]. Chapter 4 lists the model subroutines and functions. 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. Chapter 7 describes Hernan Arango's plotting programs `cnt`, `ccnt`, `sec`, and `csec`. Chapter 8 describes the ice equations while chapter 9 describes the ice subroutines and the coupling procedure.

1.1 Acquiring the SCRUM code

The version of the model described in this document is a merger between SCRUM 4.0 and a sea-ice model. SCRUM has since been replaced by ROMS as our ocean model of choice. If you would like to obtain this code, please contact me (kate@imcs.rutgers.edu). Version 3.0 of SCRUM is available from <ftp://ahab.rutgers.edu/pub>. When connected, you will be in the ftp directory. The directory structure is shown in Fig. 1. Everyone has write permission in the pub/incoming directory. To get to the SCRUM source code, type 'cd pub/scrump/tars' and then 'get scrump3.f77.tar.gz'. Some version 4.0 files are under pub/arango. It might be more convenient to access the version 3.0 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

```
% gunzip < 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
71 Dudley Road
New Brunswick, NJ 08903
(732)-932-3704

Internet: arango@imcs.rutgers.edu

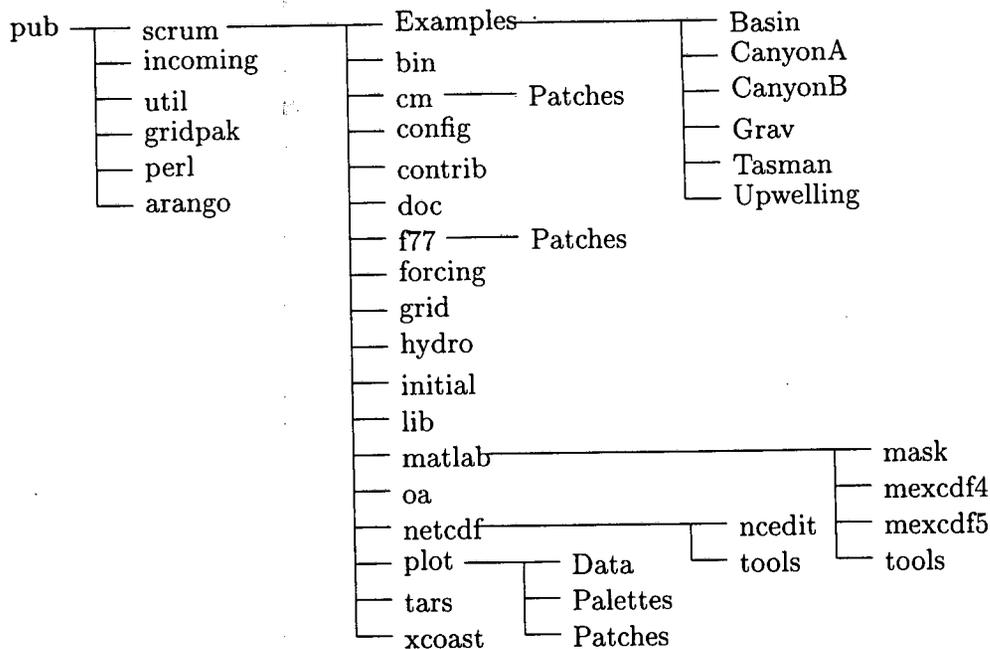


Figure 1: Tree structure of anonymous ftp directory

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. Our plans include:

- We have a 2-dimensional model which uses spectral finite elements in the horizontal [24]. The corresponding 3-dimensional model has been written and is being tested on more and more realistic problems. Ask Mohamed Iskandarani (mohamed@imcs.rutgers.edu) for more information.
- Alternate versions of the ice thermodynamics. Possible options are those of Ebert and Curry[10] and Doug Martinson. On the ice dynamics, Bruno Tremblay is planning to convert his model to a curvilinear version for parallel computers.
- Lagrangian floats from SPEM. This is just a matter of time.
- ROMS is a multi-threaded parallel version of SCRUM for shared memory machines such as those produced by SGI and Sun. Plans are to add an MPI option for distributed memory systems. ROMS plans also include nesting, vertical splines, and data assimilation. For the latest news on ROMS and the various support programs, please contact Hernan Arango (arango@imcs.rutgers.edu).

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 F. We have the source code for a version of **cpp** which is known to work at <ftp://ahab.rutgers.edu/pub/util/cpp.tar.gz>.
- 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 in turn changes the shape of the basin and leads to its own set of problems, such as altered sill depths. Also, the currents will react to the change in shelf slope—you are now solving a different problem.

2 Ocean Model Formulation

2.1 Equations of motion

The primitive equations in Cartesian coordinates can be written:

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

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

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

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

$$\rho = \rho(T, S, P) \quad (5)$$

$$\frac{\partial \phi}{\partial z} = \frac{-\rho g}{\rho_0} \quad (6)$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0. \quad (7)$$

The variables are shown in Table 2.1.

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
$h(x, y)$	bottom depth
ν, κ	horizontal viscosity and diffusivity
K_m, K_T, K_S	vertical viscosity and diffusivity
P	total pressure $P \approx -\rho_0 g z$
$\phi(x, y, z, t)$	dynamic pressure $\phi = (P/\rho_0)$
$\rho_0 + \rho(x, y, z, t)$	total <i>in situ</i> density
$S(x, y, z, t)$	salinity
t	time
$T(x, y, z, t)$	potential temperature
u, v, w	the (x, y, z) components of vector velocity \vec{v}
x, y	horizontal coordinates
z	vertical coordinate
$\zeta(x, y, t)$	the surface elevation

Table 1: The variables used in the description of the ocean model

Equations (1) and (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 (3) and (4). The equation of state is given by equation (5). In the Boussinesq approximation, density variations are neglected in the momentum equations except in their contribution to the buoyancy force in the vertical momentum equation (6). Under the hydrostatic approximation, it is further assumed that the vertical pressure gradient balances the buoyancy force. Lastly, equation (7) expresses the continuity equation for an incompressible fluid. For the moment, the effects of forcing and dissipation will be represented by the schematic terms \mathcal{F} and \mathcal{D} , respectively. The horizontal and vertical mixing will be described more fully in §3.10.

2.2 Vertical boundary conditions

The vertical boundary conditions can be prescribed as follows:

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

Variable	Description
$E - P$	evaporation minus precipitation
γ_1, γ_2	linear and quadratic bottom stress coefficients
Q_T	surface heat flux
τ_s^x, τ_s^y	surface wind stress
τ_b^x, τ_b^y	bottom stress
T_{ref}	surface reference temperature

Table 2: The variables used in the vertical boundary conditions for the ocean model

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

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

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 for u 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 [43] and Freeman et al. [13]). To proceed, we make the coordinate transformation:

$$\begin{aligned}\hat{x} &= x \\ \hat{y} &= y \\ s &= s(x, y, z) \\ z &= z(x, y, s)\end{aligned}$$

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:

$$\begin{aligned}\left(\frac{\partial}{\partial x}\right)_z &= \left(\frac{\partial}{\partial x}\right)_s - \left(\frac{1}{H_z}\right) \left(\frac{\partial z}{\partial x}\right)_s \frac{\partial}{\partial s} \\ \left(\frac{\partial}{\partial y}\right)_z &= \left(\frac{\partial}{\partial y}\right)_s - \left(\frac{1}{H_z}\right) \left(\frac{\partial z}{\partial y}\right)_s \frac{\partial}{\partial s} \\ \frac{\partial}{\partial z} &= \left(\frac{\partial s}{\partial z}\right) \frac{\partial}{\partial s} = \frac{1}{H_z} \frac{\partial}{\partial s}\end{aligned}$$

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 hats:

$$\frac{\partial u}{\partial t} - fv + \vec{v} \cdot \nabla u = -\frac{\partial \phi}{\partial x} - \left(\frac{g\rho}{\rho_0}\right) \frac{\partial z}{\partial x} - g \frac{\partial \zeta}{\partial x} + \mathcal{F}_u + \mathcal{D}_u \quad (8)$$

$$\frac{\partial v}{\partial t} + fu + \vec{v} \cdot \nabla v = -\frac{\partial \phi}{\partial y} - \left(\frac{g\rho}{\rho_0}\right) \frac{\partial z}{\partial y} - g \frac{\partial \zeta}{\partial y} + \mathcal{F}_v + \mathcal{D}_v \quad (9)$$

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

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

$$\rho = \rho(T, S, P) \quad (12)$$

$$\frac{\partial \phi}{\partial s} = \left(\frac{-gH_z \rho}{\rho_0}\right) \quad (13)$$

$$\frac{\partial H_z}{\partial t} + \frac{\partial(H_z u)}{\partial x} + \frac{\partial(H_z v)}{\partial y} + \frac{\partial(H_z \Omega)}{\partial s} = 0 \quad (14)$$

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:

$$\begin{aligned} \text{top } (s=0) \quad & \left(\frac{K_m}{H_z}\right) \frac{\partial u}{\partial s} = \tau_s^x(x, y, t) \\ & \left(\frac{K_m}{H_z}\right) \frac{\partial v}{\partial s} = \tau_s^y(x, y, t) \\ & \left(\frac{K_T}{H_z}\right) \frac{\partial T}{\partial s} = \frac{Q_T}{\rho_0 c_P} + \frac{1}{\rho_0 c_P} \frac{dQ}{dT} (T - T_{\text{ref}}) \\ & \left(\frac{K_S}{H_z}\right) \frac{\partial S}{\partial s} = \frac{(E-P)S}{\rho_0} \\ & \Omega = 0 \end{aligned}$$

$$\begin{aligned} \text{and bottom } (s=-1) \quad & \left(\frac{K_m}{H_z}\right) \frac{\partial u}{\partial s} = \tau_b^x(x, y, t) \\ & \left(\frac{K_m}{H_z}\right) \frac{\partial v}{\partial s} = \tau_b^y(x, y, t) \\ & \left(\frac{K_T}{H_z}\right) \frac{\partial T}{\partial s} = 0 \\ & \left(\frac{K_S}{H_z}\right) \frac{\partial S}{\partial s} = 0 \\ & \Omega = 0. \end{aligned}$$

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:

$$(ds)_\xi = \left(\frac{1}{m}\right) d\xi \quad (15)$$

$$(ds)_\eta = \left(\frac{1}{n}\right) d\eta \quad (16)$$

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 C contains the curvilinear version of several common vector quantities.

Denoting the velocity components in the new coordinate system by

$$\vec{v} \cdot \hat{\xi} = u \quad (17)$$

and

$$\vec{v} \cdot \hat{\eta} = v \quad (18)$$

the equations of motion (8)-(14) can be re-written (see, e.g., Arakawa and Lamb [2]) as:

$$\begin{aligned} \frac{\partial}{\partial t} \left(\frac{H_z u}{mn} \right) + \frac{\partial}{\partial \xi} \left(\frac{H_z u^2}{n} \right) + \frac{\partial}{\partial \eta} \left(\frac{H_z uv}{m} \right) + \frac{\partial}{\partial s} \left(\frac{H_z u \Omega}{mn} \right) \\ - \left\{ \left(\frac{f}{mn} \right) + v \frac{\partial}{\partial \xi} \left(\frac{1}{n} \right) - u \frac{\partial}{\partial \eta} \left(\frac{1}{m} \right) \right\} H_z v = \\ - \left(\frac{H_z}{n} \right) \left(\frac{\partial \phi}{\partial \xi} + \frac{g\rho}{\rho_0} \frac{\partial z}{\partial \xi} + g \frac{\partial \zeta}{\partial \xi} \right) + \frac{H_z}{mn} (\mathcal{F}_u + \mathcal{D}_u) \quad (19) \end{aligned}$$

$$\begin{aligned} \frac{\partial}{\partial t} \left(\frac{H_z v}{mn} \right) + \frac{\partial}{\partial \xi} \left(\frac{H_z uv}{n} \right) + \frac{\partial}{\partial \eta} \left(\frac{H_z v^2}{m} \right) + \frac{\partial}{\partial s} \left(\frac{H_z v \Omega}{mn} \right) \\ + \left\{ \left(\frac{f}{mn} \right) + v \frac{\partial}{\partial \xi} \left(\frac{1}{n} \right) - u \frac{\partial}{\partial \eta} \left(\frac{1}{m} \right) \right\} H_z u = \\ - \left(\frac{H_z}{m} \right) \left(\frac{\partial \phi}{\partial \eta} + \frac{g\rho}{\rho_0} \frac{\partial z}{\partial \eta} + g \frac{\partial \zeta}{\partial \eta} \right) + \frac{H_z}{mn} (\mathcal{F}_v + \mathcal{D}_v) \quad (20) \end{aligned}$$

$$\frac{\partial}{\partial t} \left(\frac{H_z T}{mn} \right) + \frac{\partial}{\partial \xi} \left(\frac{H_z u T}{n} \right) + \frac{\partial}{\partial \eta} \left(\frac{H_z v T}{m} \right) + \frac{\partial}{\partial s} \left(\frac{H_z \Omega T}{mn} \right) = \frac{H_z}{mn} (\mathcal{F}_T + \mathcal{D}_T) \quad (21)$$

$$\frac{\partial}{\partial t} \left(\frac{H_z S}{mn} \right) + \frac{\partial}{\partial \xi} \left(\frac{H_z u S}{n} \right) + \frac{\partial}{\partial \eta} \left(\frac{H_z v S}{m} \right) + \frac{\partial}{\partial s} \left(\frac{H_z \Omega S}{mn} \right) = \frac{H_z}{mn} (\mathcal{F}_S + \mathcal{D}_S) \quad (22)$$

$$\rho = \rho(T, S, P) \quad (23)$$

$$\frac{\partial \phi}{\partial s} = - \left(\frac{g H_z \rho}{\rho_o} \right) \quad (24)$$

$$\frac{\partial}{\partial t} \left(\frac{H_z}{mn} \right) + \frac{\partial}{\partial \xi} \left(\frac{H_z u}{n} \right) + \frac{\partial}{\partial \eta} \left(\frac{H_z v}{m} \right) + \frac{\partial}{\partial s} \left(\frac{H_z \Omega}{mn} \right) = 0. \quad (25)$$

Since z is a linear function of ζ , equation (25) can be rewritten as:

$$\frac{\partial}{\partial t} \left(\frac{\zeta}{mn} \right) + \frac{\partial}{\partial \xi} \left(\frac{H_z u}{n} \right) + \frac{\partial}{\partial \eta} \left(\frac{H_z v}{m} \right) + \frac{\partial}{\partial s} \left(\frac{H_z \Omega}{mn} \right) = 0. \quad (26)$$

All boundary conditions remain unchanged.

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. 2. 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 [2]).

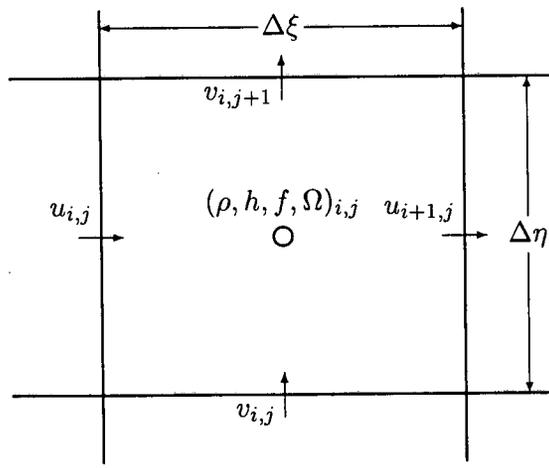


Figure 2: Placement of variables on an Arakawa C grid

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.

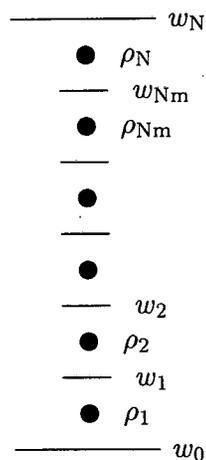


Figure 3: Placement of variables on staggered vertical grid

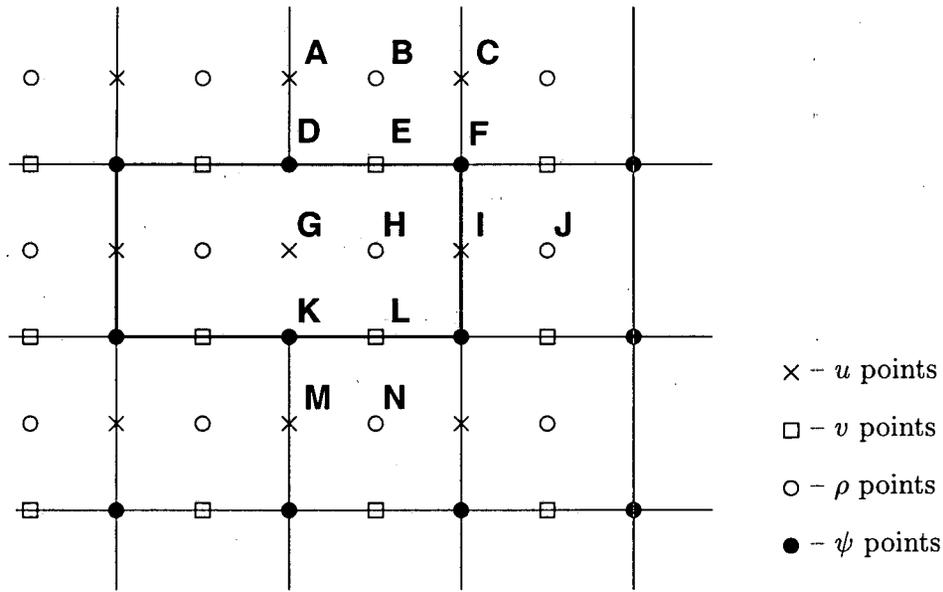


Figure 4: Masked region within the domain

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. 2 while several cells are shown in Fig. 4, 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. 4, 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 $2u_A$ while instead we have simply u_A . In order to achieve the correct result, we multiply 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 $-2u_M$.

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. 4, 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 (19)–(22) is:

$$\begin{aligned} \frac{\partial}{\partial t} \left(\frac{u \overline{H_z^\xi}}{\overline{m^\xi \overline{n^\xi}} \right) + \delta_\xi \left\{ \overline{u^\xi \left(\frac{u \overline{H_z^\xi}}{\overline{n^\xi}} \right)^\xi} \right\} + \delta_\eta \left\{ \overline{u^\eta \left(\frac{v \overline{H_z^\eta}}{\overline{m^\eta}} \right)^\xi} \right\} + \delta_s \left\{ \overline{u^s \left(\frac{H_z \Omega}{mn} \right)^\xi} \right\} \\ - \overline{\left\{ \frac{f}{mn} + \overline{v^\eta} \delta_\xi \left(\frac{1}{n} \right) - \overline{u^\xi} \delta_\eta \left(\frac{1}{m} \right) \right\} H_z \overline{v^\eta}}^\xi = \\ - \frac{\overline{H_z^\xi}}{\overline{n^\xi}} \delta_\xi \phi - \frac{g \overline{H_z^\xi}}{\rho_o \overline{n^\xi}} \overline{\rho^\xi} \delta_\xi z - \frac{g \overline{H_z^\xi}}{\rho_o \overline{n^\xi}} (\rho_o + \overline{\rho^\xi}) \delta_\xi \zeta + \mathcal{D}_u + \mathcal{F}_u \quad (27) \end{aligned}$$

$$\begin{aligned} \frac{\partial}{\partial t} \left(\frac{v \overline{H_z^\eta}}{\overline{m^\eta \overline{n^\eta}} \right) + \delta_\xi \left\{ \overline{v^\xi \left(\frac{u \overline{H_z^\xi}}{\overline{n^\xi}} \right)^\eta} \right\} + \delta_\eta \left\{ \overline{v^\eta \left(\frac{v \overline{H_z^\eta}}{\overline{m^\eta}} \right)^\eta} \right\} + \delta_s \left\{ \overline{v^s \left(\frac{H_z \Omega}{mn} \right)^\eta} \right\} \\ + \overline{\left\{ \frac{f}{mn} + \overline{v^\eta} \delta_\xi \left(\frac{1}{n} \right) - \overline{u^\xi} \delta_\eta \left(\frac{1}{m} \right) \right\} H_z \overline{u^\xi}}^\eta = \\ - \frac{\overline{H_z^\eta}}{\overline{m^\eta}} \delta_\eta \phi - \frac{g \overline{H_z^\eta}}{\rho_o \overline{m^\eta}} \overline{\rho^\eta} \delta_\eta z - \frac{g \overline{H_z^\eta}}{\rho_o \overline{m^\eta}} (\rho_o + \overline{\rho^\eta}) \delta_\eta \zeta + \mathcal{D}_v + \mathcal{F}_v \quad (28) \end{aligned}$$

$$\frac{\partial}{\partial t} \left(\frac{hT}{mn} \right) + \delta_\xi \left(\frac{u \overline{H_z^\xi} \overline{T^\xi}}{\overline{n^\xi}} \right) + \delta_\eta \left(\frac{v \overline{H_z^\eta} \overline{T^\eta}}{\overline{m^\eta}} \right) + \delta_s \left(\overline{T^s} \frac{H_z \Omega}{mn} \right) = \mathcal{D}_T + \mathcal{F}_T \quad (29)$$

$$\frac{\partial}{\partial t} \left(\frac{hS}{mn} \right) + \delta_\xi \left(\frac{u \overline{H_z^\xi} \overline{S^\xi}}{\overline{n^\xi}} \right) + \delta_\eta \left(\frac{v \overline{H_z^\eta} \overline{S^\eta}}{\overline{m^\eta}} \right) + \delta_s \left(\overline{S^s} \frac{H_z \Omega}{mn} \right) = \mathcal{D}_S + \mathcal{F}_S \quad (30)$$

$$\rho = \rho(S, T, P) \quad (31)$$

$$\phi(s) = -\frac{g}{\rho_o} \Gamma_s^0 H_z \rho \quad (32)$$

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{(\)^\xi}$, $\overline{(\)^\eta}$ and $\overline{(\)^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 [2]; however, their scheme also conserves enstrophy.

The continuity equation will be discussed below in §3.8.

3.4 Vertical viscosity and diffusion

The \mathcal{D}_u , \mathcal{D}_v , \mathcal{D}_T and \mathcal{D}_S terms in equations (27)–(30) represent both horizontal and vertical mixing processes. The horizontal options will be covered in §3.10. The model has several options for computing the vertical coefficients; these will be described in §3.11. The vertical viscosity and diffusion terms have the form:

$$\frac{\partial}{\partial s} \left(\frac{K}{H_z mn} \frac{\partial \phi}{\partial s} \right) \quad (33)$$

where ϕ represents one of u , v , T or S , and K is the corresponding vertical viscous or diffusive coefficient. 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:

$$\frac{\partial(H_z \phi)}{\partial t} = mnR_\phi + \frac{\partial}{\partial s} \left(\frac{K}{H_z} \frac{\partial \phi}{\partial s} \right) \quad (34)$$

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 (34) as:

$$\frac{\partial(H_z \phi)}{\partial t} = mnR_\phi + (1 - \lambda) \frac{\partial}{\partial s} \left(\frac{K}{H_z} \frac{\partial \phi^n}{\partial s} \right) + \lambda \frac{\partial}{\partial s} \left(\frac{K}{H_z} \frac{\partial \phi^{n+1}}{\partial s} \right). \quad (35)$$

The discrete form of equation (35) is:

$$\begin{aligned} \frac{H_{z_k}^{n+1} \phi_k^{n+1} - H_{z_k}^n \phi_k^n}{\Delta t} = mnR_\phi + \frac{(1 - \lambda)}{\Delta_s^2} \left[\frac{K_k}{H_{z_k}} (\phi_{k+1}^n - \phi_k^n) - \frac{K_{k-1}}{H_{z_{k-1}}} (\phi_k^n - \phi_{k-1}^n) \right] \\ + \frac{\lambda}{\Delta_s^2} \left[\frac{K_k}{H_{z_k}} (\phi_{k+1}^{n+1} - \phi_k^{n+1}) - \frac{K_{k-1}}{H_{z_{k-1}}} (\phi_k^{n+1} - \phi_{k-1}^{n+1}) \right] \end{aligned} \quad (36)$$

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:

$$A_k \phi_{k+1}^{n+1} + B_k \phi_k^{n+1} + C_k \phi_{k-1}^{n+1} = D_k \quad (37)$$

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

$$A(1) = 0 \quad (38)$$

$$A(2 : N) = -\frac{\lambda \Delta t K_{k-1}}{\Delta s^2 H_{z_{k-1}}^{n+1}} \quad (39)$$

$$B(1) = H_{z_1}^{n+1} + \frac{\lambda \Delta t K_1}{\Delta s^2 H_{z_1}^{n+1}} \quad (40)$$

$$B(2 : Nm) = H_{z_k}^{n+1} + \frac{\lambda \Delta t K_k}{\Delta s^2 H_{z_k}^{n+1}} + \frac{\lambda \Delta t K_{k-1}}{\Delta s^2 H_{z_{k-1}}^{n+1}} \quad (41)$$

$$B(N) = H_{z_N}^{n+1} + \frac{\lambda \Delta t K_{Nm}}{\Delta s^2 H_{z_{Nm}}^{n+1}} \quad (42)$$

$$C(1 : Nm) = -\frac{\lambda \Delta t K_k}{\Delta s^2 H_{z_k}^{n+1}} \quad (43)$$

$$C(N) = 0 \quad (44)$$

$$D(1) = H_{z_1}^n \phi_1^n + \Delta t mn R_{\phi_1} + \frac{\Delta t(1-\lambda) K_1}{\Delta s^2 H_{z_1}^n} (\phi_2^n - \phi_1^n) - \frac{\Delta t}{\Delta s} \tau_b \quad (45)$$

$$D(2 : Nm) = H_{z_k}^n \phi_k^n + \Delta t mn R_{\phi_k} + \quad (46)$$

$$\frac{\Delta t(1-\lambda)}{\Delta s^2} \left[\frac{K_k}{H_{z_k}^n} (\phi_{k+1}^n - \phi_k^n) - \frac{K_{k-1}}{H_{z_{k-1}}^n} (\phi_k^n - \phi_{k-1}^n) \right] \quad (47)$$

$$D(N) = H_{z_N}^n \phi_N^n + \Delta t mn R_{\phi_N} - \frac{\Delta t(1-\lambda) K_{Nm}}{\Delta s^2 H_{z_{Nm}}^n} (\phi_N^n - \phi_{Nm}^n) + \frac{\Delta t}{\Delta s} \tau_s \quad (48)$$

This is a standard tridiagonal system for which the solution procedure can be found in any standard reference, such as Press et al. [44].

3.5 Depth-integrated equations

The depth average of a quantity A is given by:

$$\bar{A} = \frac{1}{D} \int_{-1}^0 H_z A ds \quad (49)$$

where the overbar indicates a vertically averaged quantity and

$$D \equiv \zeta(\xi, \eta, t) + h(\xi, \eta) \quad (50)$$

is the total depth of the water column. The vertical integral of equation (19) is:

$$\begin{aligned} \frac{\partial}{\partial t} \left(\frac{D\bar{u}}{mn} \right) + \frac{\partial}{\partial \xi} \left(\frac{D\bar{u}\bar{u}}{n} \right) + \frac{\partial}{\partial \eta} \left(\frac{D\bar{u}\bar{v}}{m} \right) - \frac{Df\bar{v}}{mn} \\ - \left[\bar{v}\bar{v} \frac{\partial}{\partial \xi} \left(\frac{1}{n} \right) - \bar{u}\bar{v} \frac{\partial}{\partial \eta} \left(\frac{1}{m} \right) \right] D = -\frac{D}{n} \left(\frac{\partial \bar{\phi}_2}{\partial \xi} + g \frac{\partial \zeta}{\partial \xi} \right) \\ + \frac{D}{mn} (\bar{\mathcal{F}}_u + \bar{\mathcal{D}}_{h_u}) + \frac{1}{mn} (\tau_s^\xi - \tau_b^\xi) \end{aligned} \quad (51)$$

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

of equation (20) is:

$$\begin{aligned} \frac{\partial}{\partial t} \left(\frac{D\bar{v}}{mn} \right) + \frac{\partial}{\partial \xi} \left(\frac{D\bar{u}\bar{v}}{n} \right) + \frac{\partial}{\partial \eta} \left(\frac{D\bar{v}\bar{v}}{m} \right) + \frac{Df\bar{u}}{mn} \\ + \left[\bar{u}\bar{v} \frac{\partial}{\partial \xi} \left(\frac{1}{n} \right) - \bar{u}\bar{u} \frac{\partial}{\partial \eta} \left(\frac{1}{m} \right) \right] D = -\frac{D}{m} \left(\frac{\partial \bar{\phi}_2}{\partial \eta} + g \frac{\partial \zeta}{\partial \eta} \right) \\ + \frac{D}{mn} (\bar{\mathcal{F}}_v + \bar{\mathcal{D}}_{h_v}) + \frac{1}{mn} (\tau_s^\eta - \tau_b^\eta). \end{aligned} \quad (52)$$

We also need the vertical integral of equation (26). Using the vertical boundary conditions on Ω we get:

$$\frac{\partial}{\partial t} \left(\frac{\zeta}{mn} \right) + \frac{\partial}{\partial \xi} \left(\frac{D\bar{u}}{n} \right) + \frac{\partial}{\partial \eta} \left(\frac{D\bar{v}}{m} \right) = 0. \quad (53)$$

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 (51), (52), and (53) are integrated using a short timestep and the values of \bar{u} and \bar{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. 5.

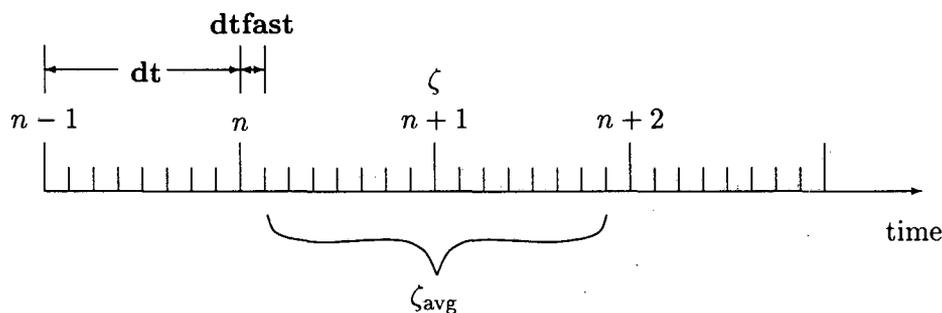


Figure 5: The split timestepping used in the model.

Some of the terms in equations (51) and (52) 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 $R_{u_{\text{slow}}}$ and $R_{v_{\text{slow}}}$, equations (51) and (52) become:

$$\begin{aligned} \frac{\partial}{\partial t} \left(\frac{D\bar{u}}{mn} \right) + \frac{\partial}{\partial \xi} \left(\frac{D\bar{u}\bar{u}}{n} \right) + \frac{\partial}{\partial \eta} \left(\frac{D\bar{u}\bar{v}}{m} \right) - \frac{Df\bar{v}}{mn} \\ - \left[\bar{v}\bar{v} \frac{\partial}{\partial \xi} \left(\frac{1}{n} \right) - \bar{u}\bar{v} \frac{\partial}{\partial \eta} \left(\frac{1}{m} \right) \right] D = R_{u_{\text{slow}}} - \frac{gD}{n} \frac{\partial \zeta}{\partial \xi} + \frac{D}{mn} \mathcal{D}_{\bar{u}} - \frac{1}{mn} \tau_b^\xi \end{aligned} \quad (54)$$

$$\begin{aligned} \frac{\partial}{\partial t} \left(\frac{D\bar{v}}{mn} \right) + \frac{\partial}{\partial \xi} \left(\frac{D\bar{u}\bar{v}}{n} \right) + \frac{\partial}{\partial \eta} \left(\frac{D\bar{v}\bar{v}}{m} \right) + \frac{Df\bar{u}}{mn} \\ + \left[\bar{u}\bar{v} \frac{\partial}{\partial \xi} \left(\frac{1}{n} \right) - \bar{u}\bar{u} \frac{\partial}{\partial \eta} \left(\frac{1}{m} \right) \right] D = R_{v_{\text{slow}}} - \frac{gD}{m} \frac{\partial \zeta}{\partial \eta} + \frac{D}{mn} \mathcal{D}_{\bar{v}} - \frac{1}{mn} \tau_b^\eta. \end{aligned} \quad (55)$$

When timestepping the model, we compute the right-hand-sides for equations (27) and (28) as well as the right-hand-sides for equations (54) and (55). 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 $R_{u_{\text{slow}}}$ and $R_{v_{\text{slow}}}$. This was found to be the easiest way to retain the baroclinic contributions of the non-linear terms such as $\overline{u\bar{u}} - \bar{u}\bar{u}$.

The model is timestepped from time n to time $n+1$ by using short timesteps on equations (54), (55) and (53). Equation (53) is timestepped first, so that an estimate of the new D is available for the time rate of change terms in equations (54) and (55). A third-order predictor-corrector timestepping is used. In practice, we actually timestep all the way to time $(n+2) - \text{dtfast}$ and then average the values of \bar{u} , \bar{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 (27) and (28) 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 \bar{u} and \bar{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 (29) and (30) are also advanced as in §3.4. There is also an option to advect the temperature and salinity using a Smolarkiewicz scheme; this affects the time stepping as described in §3.7.1.

3.7 Advection schemes

Thus far, the advection scheme presented here is a centered second-order scheme. This scheme is known to have some unfortunate properties in the presence of strong gradients, such as large over- and under-shoots of tracers, leading to the need for large amounts of horizontal smoothing. SCRUM also provides two alternative advection schemes with better behavior in many situations. At present, the alternatives are only implemented in the full 3-D engine of the model.

3.7.1 Smolarkiewicz

Smolarkiewicz has written a series of papers ([50], [51], [52] and [53]) on an advection scheme known as MPDATA (Multidimensional Positive Definite Advection Transport Algorithm). MPDATA is meant to be applied to positive-definite tracer fields only. It uses a first-order upwind scheme to obtain a first estimate of the updated field, followed by one or more “antidiffusion” passes to obtain the final updated field. Note that we add an offset to the temperature field before advecting with this scheme, since ocean temperatures can be negative (or we could use the Kelvin scale instead).

Recall that the advection of a tracer ψ has an equation of the form

$$\frac{\partial H_z \psi}{\partial t} = -\frac{\partial}{\partial \xi} F^\xi - \frac{\partial}{\partial \eta} F^\eta - \frac{\partial}{\partial s} F^s. \quad (56)$$

where we have introduced the advective fluxes:

$$F^\xi = \frac{H_z u \psi}{n} \quad (57)$$

$$F^\eta = \frac{H_z v \psi}{m} \quad (58)$$

$$F^s = \frac{H_z \Omega \psi}{mn} \quad (59)$$

Rather than computing the centered average for these terms, we use the “upwind” or “donor cell” value for ψ . For instance:

$$F_{i,j,k}^\xi = \frac{\overline{H_z}^\xi}{\overline{n}^\xi} [\max(0, u_{i,j,k})\psi_{i-1,j,k} + \min(0, u_{i,j,k})\psi_{i,j,k}]. \quad (60)$$

This picks up the value of ψ to the left if u is positive, otherwise it picks up the value to the right (F^ξ is located at a u point on the grid).

Using these fluxes, we compute an upwind estimate of the new tracer:

$$\psi^* = \psi^n - \frac{\Delta t mn}{H_z} \left(\frac{\partial F^\xi}{\partial \xi} + \frac{\partial F^\eta}{\partial \eta} + \frac{\partial F^s}{\partial s} \right) \quad (61)$$

This estimate is used to create an ‘‘antidiffusive’’ velocity:

$$\tilde{u} = \frac{1}{2} \left(\frac{|u|}{m} - \Delta t u^2 \right) \frac{m}{\psi^*} \frac{\partial \psi^*}{\partial \xi} - \frac{1}{2} \Delta t u \left(v \frac{n}{\psi^*} \frac{\partial \psi^*}{\partial \eta} + w \frac{1}{\psi^*} \frac{\partial \psi^*}{\partial z} \right) \quad (62)$$

$$\tilde{v} = \frac{1}{2} \left(\frac{|v|}{n} - \Delta t v^2 \right) \frac{n}{\psi^*} \frac{\partial \psi^*}{\partial \eta} - \frac{1}{2} \Delta t v \left(u \frac{m}{\psi^*} \frac{\partial \psi^*}{\partial \xi} + w \frac{1}{\psi^*} \frac{\partial \psi^*}{\partial z} \right) \quad (63)$$

$$\tilde{w} = \frac{1}{2} (|w| \Delta z - \Delta t w^2) \frac{1}{\psi^*} \frac{\partial \psi^*}{\partial z} - \frac{1}{2} \Delta t w \left(u \frac{m}{\psi^*} \frac{\partial \psi^*}{\partial \xi} + v \frac{n}{\psi^*} \frac{\partial \psi^*}{\partial \eta} \right) \quad (64)$$

In SCRUM, these velocities are set to zero if the first term is smaller than the second term (there is no such check in the ice MPDATA code, nor is there mention of such a thing in the Smolarkiewicz papers). Using these velocities in another upwind iteration leads to a second estimate of the new tracer fields:

$$\psi^{n+1} = \psi^* - \frac{\Delta t mn}{H_z} \left(\frac{\partial F^\xi(\tilde{u})}{\partial \xi} + \frac{\partial F^\eta(\tilde{v})}{\partial \eta} + \frac{\partial F^s(\tilde{w})}{\partial s} \right) \quad (65)$$

In fact, this correction step can be repeated any number of times in order to improve the solution.

Thus far, we have described the original MPDATA scheme which maintains the positive definite nature of the tracer. However, it does not prevent under- and overshoots in a tracer such as salinity, in which the initial range would be say 34 to 36 PSU. An optional nonoscillatory modification to MPDATA is described in Smolarkiewicz [53]. It uses ideas from the FCT (flux-corrected transport) scheme to limit the antidiffusive fluxes. This modification is based on multiplying the antidiffusive fluxes by a coefficient C such that $0 \leq C \leq 1$ and the resulting tracer value is bounded by $\psi^{\min} \leq \psi^{n+1} \leq \psi^{\max}$.

First, we need to define ψ^{\min} and ψ^{\max} .

$$\begin{aligned} \psi_{i,j,k}^{\min} &= \min(\psi_{i,j,k}, \psi_{i-1,j,k}, \psi_{i+1,j,k}, \psi_{i,j-1,k}, \psi_{i,j+1,k}, \psi_{i,j,k-1}, \psi_{i,j,k+1}, \\ &\quad \psi_{i,j,k}^*, \psi_{i-1,j,k}^*, \psi_{i+1,j,k}^*, \psi_{i,j-1,k}^*, \psi_{i,j+1,k}^*, \psi_{i,j,k-1}^*, \psi_{i,j,k+1}^*) \\ \psi_{i,j,k}^{\max} &= \max(\psi_{i,j,k}, \psi_{i-1,j,k}, \psi_{i+1,j,k}, \psi_{i,j-1,k}, \psi_{i,j+1,k}, \psi_{i,j,k-1}, \psi_{i,j,k+1}, \\ &\quad \psi_{i,j,k}^*, \psi_{i-1,j,k}^*, \psi_{i+1,j,k}^*, \psi_{i,j-1,k}^*, \psi_{i,j+1,k}^*, \psi_{i,j,k-1}^*, \psi_{i,j,k+1}^*) \end{aligned}$$

Then, we define the β -ratios:

$$\begin{aligned} (\beta \uparrow)^{-1} (\psi_{i,j,k}^{\max} - \psi_{i,j,k}^*) &= m \Delta t [\max(0, \tilde{u}_{i,j,k}) \psi_{i-1,j,k}^* - \min(0, \tilde{u}_{i+1,j,k}) \psi_{i+1,j,k}^*] \\ &\quad + n \Delta t [\max(0, \tilde{v}_{i,j,k}) \psi_{i,j-1,k}^* - \min(0, \tilde{v}_{i,j+1,k}) \psi_{i,j+1,k}^*] \\ &\quad + \frac{\Delta t}{\Delta z} [\max(0, \tilde{w}_{i,j,k-1}) \psi_{i,j,k-1}^* - \min(0, \tilde{w}_{i,j,k}) \psi_{i,j,k+1}^*] \\ (\beta \downarrow)^{-1} (\psi_{i,j,k}^* - \psi_{i,j,k}^{\min}) &= m \Delta t [\max(0, \tilde{u}_{i+1,j,k}) \psi_{i,j,k}^* - \min(0, \tilde{u}_{i,j,k}) \psi_{i,j,k}^*] \\ &\quad + n \Delta t [\max(0, \tilde{v}_{i,j+1,k}) \psi_{i,j,k}^* - \min(0, \tilde{v}_{i,j,k}) \psi_{i,j,k}^*] \\ &\quad + \frac{\Delta t}{\Delta z} [\max(0, \tilde{w}_{i,j,k}) \psi_{i,j,k}^* - \min(0, \tilde{w}_{i,j,k-1}) \psi_{i,j,k}^*] \end{aligned}$$

Finally, the flux-limited form of the antidiffusion velocities is:

$$\tilde{u}' = \min(1, \beta_{\downarrow i-1,j,k}, \beta_{\uparrow i,j,k}) \max(0, \tilde{u}) + \min(1, \beta_{\uparrow i-1,j,k}, \beta_{\downarrow i,j,k}) \min(0, \tilde{u}) \quad (66)$$

$$\tilde{v}' = \min(1, \beta_{\downarrow i,j-1,k}, \beta_{\uparrow i,j,k}) \max(0, \tilde{v}) + \min(1, \beta_{\uparrow i,j-1,k}, \beta_{\downarrow i,j,k}) \min(0, \tilde{v}) \quad (67)$$

$$\tilde{w}' = \min(1, \beta_{\downarrow i,j,k}, \beta_{\uparrow i,j,k+1}) \max(0, \tilde{w}) + \min(1, \beta_{\uparrow i,j,k}, \beta_{\downarrow i,j,k+1}) \min(0, \tilde{w}) \quad (68)$$

These velocities are used in equation (65) to produce the flux-limited estimate of ψ^{n+1} .

3.7.2 Third-order Upwind

There is a class of third-order upwind advection schemes, both one-dimensional (Leonard [30]) and two-dimensional (Rasch [45]). This scheme is known as UTOPIA (Uniformly Third-Order Polynomial Interpolation Algorithm). Applying flux limiters to UTOPIA is explored in Thuburn [58], although it is not implemented in SCRUM. The two-dimensional formulation in Rasch contains terms of order $u^2\psi$ and $u^3\psi$, including cross terms ($uv\psi$). The terms which are nonlinear in velocity have been dropped in SCRUM, leaving one extra upwind term in the computation of the advective fluxes:

$$F^\xi = \frac{H_z u}{n} \left(\psi - \gamma \frac{\partial^2 \psi}{\partial \xi^2} \right) \quad (69)$$

$$F^\eta = \frac{H_z v}{m} \left(\psi - \gamma \frac{\partial^2 \psi}{\partial \eta^2} \right) \quad (70)$$

The second derivative terms are centered on a ρ point in the grid, but are needed at a u or v point in the flux. The upstream value is used [see equation (60)]. The value of γ in the model is $\frac{1}{8}$ while that in Rasch [45] is $\frac{1}{6}$.

Because the third-order upwind scheme is designed to be two-dimensional, it is not used in the vertical (though one might argue that we are simply performing one-dimensional operations here). Instead, we use a centered fourth-order scheme in the vertical when the third-order upwind option is turned on:

$$F^s = \frac{H_z w}{mn} \left[-\frac{1}{16} \psi_{i,j,k-1} + \frac{9}{16} \psi_{i,j,k} + \frac{9}{16} \psi_{i,j,k+1} - \frac{1}{16} \psi_{i,j,k+2} \right] \quad (71)$$

One advantage of UTOPIA over MPDATA is that it can be used on variables having both negative and positive values. Therefore, it can be used on velocity as well as scalars (is there a reference for this?). For the u -velocity, we have:

$$F^\xi = \left(u - \gamma \frac{\partial^2 u}{\partial \xi^2} \right) \left[\frac{H_z u}{n} - \gamma \frac{\partial^2}{\partial \xi^2} \left(\frac{H_z u}{n} \right) \right] \quad (72)$$

$$F^\eta = \left(u - \gamma \frac{\partial^2 u}{\partial \eta^2} \right) \left[\frac{H_z v}{m} - \gamma \frac{\partial^2}{\partial \eta^2} \left(\frac{H_z v}{m} \right) \right] \quad (73)$$

$$F^s = \frac{H_z w}{mn} \left[-\frac{1}{16} u_{i,j,k-1} + \frac{9}{16} u_{i,j,k} + \frac{9}{16} u_{i,j,k+1} - \frac{1}{16} u_{i,j,k+2} \right] \quad (74)$$

while for the v -velocity we have:

$$F^\xi = \left(v - \gamma \frac{\partial^2 v}{\partial \xi^2} \right) \left[\frac{H_z u}{n} - \gamma \frac{\partial^2}{\partial \eta^2} \left(\frac{H_z u}{n} \right) \right] \quad (75)$$

$$F^\eta = \left(v - \gamma \frac{\partial^2 v}{\partial \eta^2} \right) \left[\frac{H_z v}{m} - \gamma \frac{\partial^2}{\partial \eta^2} \left(\frac{H_z v}{m} \right) \right] \quad (76)$$

$$F^s = \frac{H_z w}{mn} \left[-\frac{1}{16} v_{i,j,k-1} + \frac{9}{16} v_{i,j,k} + \frac{9}{16} v_{i,j,k+1} - \frac{1}{16} v_{i,j,k+2} \right] \quad (77)$$

In all these terms, the second derivatives are evaluated at an upstream location.

3.8 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 (26) and (53) to obtain:

$$\frac{\partial}{\partial \xi} \left(\frac{H_z u}{n} \right) + \frac{\partial}{\partial \eta} \left(\frac{H_z v}{m} \right) + \frac{\partial}{\partial s} \left(\frac{H_z \Omega}{mn} \right) - \frac{\partial}{\partial \xi} \left(\frac{D\bar{u}}{n} \right) - \frac{\partial}{\partial \eta} \left(\frac{D\bar{v}}{m} \right) = 0. \quad (78)$$

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

$$\frac{H_z \Omega}{mn} = \int \left[\delta_\xi \left(\frac{\bar{u} D^\xi}{\bar{n}^\xi} \right) + \delta_\eta \left(\frac{\bar{v} D^\eta}{\bar{m}^\eta} \right) - \delta_\xi \left(\frac{u \bar{H}_z^\xi}{\bar{n}^\xi} \right) - \delta_\eta \left(\frac{v \bar{H}_z^\eta}{\bar{m}^\eta} \right) \right] ds. \quad (79)$$

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 [25]. 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.

3.9 The pressure gradient terms

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

$$H_z \nabla \phi + \frac{g \rho H_z}{\rho_0} \nabla z + g H_z \nabla \zeta \quad (80)$$

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 [20]; and Beckmann and Haidvogel [5]).

The pressure ϕ is computed by a vertical integration of the density field using equation (24). Prior to the integration, a horizontal average of the density, $\bar{\rho}(z)$, is subtracted from ρ . As discussed by Haney [20] and McCalpin [33], $\bar{\rho}$ 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.

3.10 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.

3.10.1 Laplacian

The Laplacian of a scalar ψ in curvilinear coordinates is (see Appendix C):

$$\nabla^2 \psi = \nabla \cdot \nabla \psi = mn \left[\frac{\partial}{\partial \xi} \left(\frac{m}{n} \frac{\partial \psi}{\partial \xi} \right) + \frac{\partial}{\partial \eta} \left(\frac{n}{m} \frac{\partial \psi}{\partial \eta} \right) \right] \quad (81)$$

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

$$\left[\frac{\partial}{\partial \xi} \left(\frac{\nu_2 H_z m}{n} \frac{\partial \psi}{\partial \xi} \right) + \frac{\partial}{\partial \eta} \left(\frac{\nu_2 H_z n}{m} \frac{\partial \psi}{\partial \eta} \right) \right] \quad (82)$$

where ψ 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.

3.10.2 Biharmonic

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

$$Y = \frac{mn}{H_z} \left[\frac{\partial}{\partial \xi} \left(\frac{\nu_4 H_z m}{n} \frac{\partial \psi}{\partial \xi} \right) + \frac{\partial}{\partial \eta} \left(\frac{\nu_4 H_z n}{m} \frac{\partial \psi}{\partial \eta} \right) \right] \quad (83)$$

and is

$$- \left[\frac{\partial}{\partial \xi} \left(\frac{H_z m}{n} \frac{\partial Y}{\partial \xi} \right) + \frac{\partial}{\partial \eta} \left(\frac{H_z n}{m} \frac{\partial Y}{\partial \eta} \right) \right] \quad (84)$$

where ψ 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.

3.10.3 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 to diffuse on constant z surfaces (**MIX_GP_UV**, **MIX_GP_TS**), constant *in situ* density surfaces (**MIX_EPI_UV**, **MIX_EPI_TS**), and constant potential density surfaces (**MIX_ISO_UV**, **MIX_ISO_TS**)

The horizontal Laplacian diffusion operator is computed by finding the three components of the flux of the quantity ψ . The ξ and η components are locally horizontal, rather than along the s surface. Here, **MIX_EPI** represents both the epineutral and isopycnal options. The diffusive

fluxes are:

$$F^\xi = \nu_2 \left[m \frac{\partial \psi}{\partial \xi} - \underbrace{\left(m \frac{\partial z}{\partial \xi} + S_x \right)}_{\text{MIX_EPI}} \right] \frac{\partial \psi}{\partial s} \quad (85)$$

MIX_GP

$$F^\eta = \nu_2 \left[n \frac{\partial \psi}{\partial \eta} - \underbrace{\left(n \frac{\partial z}{\partial \eta} + S_y \right)}_{\text{MIX_EPI}} \right] \frac{\partial \psi}{\partial s} \quad (86)$$

MIX_GP

$$F^s = - \underbrace{\frac{1}{H_z} \left(m \frac{\partial z}{\partial \xi} + S_x \right)}_{\text{MIX_GP}} F^\xi - \underbrace{\frac{1}{H_z} \left(n \frac{\partial z}{\partial \eta} + S_y \right)}_{\text{MIX_GP}} F^\eta \quad (87)$$

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

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

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

$$\frac{\partial}{\partial \xi} \left(\frac{H_z F^\xi}{n} \right) + \frac{\partial}{\partial \eta} \left(\frac{H_z F^\eta}{m} \right) + \frac{\partial}{\partial s} \left(\frac{H_z F^s}{mn} \right) \quad (88)$$

The biharmonic rotated mixing tensors are computed much as the non-rotated biharmonic mixing. We define a temporary variable Y based on equation (88):

$$Y = \frac{mn}{H_z} \left[\frac{\partial}{\partial \xi} \left(\frac{H_z F^\xi}{n} \right) + \frac{\partial}{\partial \eta} \left(\frac{H_z F^\eta}{m} \right) + \frac{\partial}{\partial s} \left(\frac{H_z F^s}{mn} \right) \right] \quad (89)$$

We then build up fluxes of Y as in equations (85)–(87). We then apply equation (88) to these Y fluxes to obtain the biharmonic mixing tensors.

3.11 Vertical mixing schemes

SCRUM contains a variety of methods for setting the vertical viscous and diffusive coefficients. The choices range from simply choosing fixed values to the KPP and Mellor-Yamada turbulence closure schemes. See Large [27] for a review of surface ocean mixing schemes. Many schemes have a background molecular value which is used when the turbulent processes are assumed to be small (such as in the interior).

3.11.1 Brunt-Väisälä frequency scheme

One of the simplest schemes is to set the vertical diffusion coefficients to be large when the water column is vertically unstable. The vertical viscosity is uniform:

$$K_m = K_{m_{\text{background}}} \quad (90)$$

$$K_s = \begin{cases} C_o & \text{if } Ri_g < 0, \\ K_{s_{\text{background}}} & \text{if } Ri_g = 0, \\ \min(\nu_{\text{max}}, \max(\nu_{\text{min}}, C/\sqrt{Ri_g})) & \text{if } Ri_g > 0. \end{cases} \quad (91)$$

Where K_s applies to any scalar. The constants in this expression are $C_o = 1.0$, $C = 10^{-7}$, $\nu_{\text{min}} = 3 \times 10^{-5}$ and $\nu_{\text{max}} = 4 \times 10^{-4}$, and the gradient Richardson number is

$$Ri_g = \frac{N^2}{\left(\frac{\partial u}{\partial z}\right)^2 + \left(\frac{\partial v}{\partial z}\right)^2}. \quad (92)$$

3.11.2 Pacanowski-Philander

Pacanowski and Philander [40] developed a vertical mixing parameterization based on measurements in the equatorial oceans:

$$K_m = \frac{\nu_o}{(1 + aRi_g)^n} + K_{m_{\text{background}}} \quad (93)$$

$$K_s = \frac{K_m}{(1 + aRi_g)} + K_{s_{\text{background}}} \quad (94)$$

The constants here are $\nu_o = .01$, $n = 2$ and $a = 5$. The values of K_m and K_s can get very large for negative values of Ri_g , so we choose to limit these values to that obtained for $Ri_g = 0$.

3.11.3 Mellor-Yamada

One of the more popular closure schemes is that of Mellor and Yamada [36], [37]. They actually present a hierarchy of closures of increasing complexity. SCRUM provides only the "Level 2.5" closure with the Galperin et al. [14] modifications as described in Allen et al. [1]. This closure scheme adds two prognostic equations, one for the turbulent kinetic energy ($\frac{1}{2}q^2$) and one for the turbulent kinetic energy times a length scale (q^2l).

The turbulent kinetic energy equation is:

$$\frac{D}{Dt} \left(\frac{q^2}{2} \right) - \frac{\partial}{\partial z} \left[K_q \frac{\partial}{\partial z} \left(\frac{q^2}{2} \right) \right] = P_s + P_b - \xi_d \quad (95)$$

where P_s is the shear production, P_b is the buoyant production and ξ_d is the dissipation of turbulent kinetic energy. These terms are given by

$$P_s = K_m \left[\left(\frac{\partial u}{\partial z} \right)^2 + \left(\frac{\partial v}{\partial z} \right)^2 \right], \quad (96)$$

$$P_b = K_s N^2, \quad (97)$$

$$\xi_d = \frac{q^3}{B_1 l} \quad (98)$$

where B_1 is a constant. One can also add a traditional horizontal Laplacian or biharmonic diffusion (\mathcal{D}_q) to the turbulent kinetic energy equation. The form of this equation in the model coordinates

becomes

$$\begin{aligned} \frac{\partial}{\partial t} \left(\frac{H_z q^2}{mn} \right) + \frac{\partial}{\partial \xi} \left(\frac{H_z u q^2}{n} \right) + \frac{\partial}{\partial \eta} \left(\frac{H_z v q^2}{m} \right) + \frac{\partial}{\partial s} \left(\frac{H_z \Omega q^2}{mn} \right) - \frac{\partial}{\partial s} \left(\frac{K_q}{mn H_z} \frac{\partial q^2}{\partial s} \right) = \\ \frac{2H_z K_m}{mn} \left[\left(\frac{\partial u}{\partial z} \right)^2 + \left(\frac{\partial v}{\partial z} \right)^2 \right] + \frac{2H_z K_s}{mn} N^2 - \frac{2H_z q^3}{mn B_1 l} + \frac{H_z}{mn} \mathcal{D}_q. \end{aligned} \quad (99)$$

The vertical boundary conditions are:

$$\begin{aligned} \text{top } (z = \zeta(x, y, t)) \quad & \frac{H_z \Omega}{mn} = 0 \\ & \frac{K_q}{mn H_z} \frac{\partial q^2}{\partial s} = \frac{B_1^{2/3}}{\rho_o} \left[(\tau_s^\xi)^2 + (\tau_s^\eta)^2 \right] \\ & H_z K_m \left(\frac{\partial u}{\partial z}, \frac{\partial v}{\partial z} \right) = \frac{1}{\rho_o} (\tau_s^\xi, \tau_s^\eta) \\ & H_z K_s N^2 = \frac{Q}{\rho_o c_P} \\ \text{and bottom } (z = -h(x, y)) \quad & \frac{H_z \Omega}{mn} = 0 \\ & \frac{K_q}{mn H_z} \frac{\partial q^2}{\partial s} = \frac{B_1^{2/3}}{\rho_o} \left[(\tau_b^\xi)^2 + (\tau_b^\eta)^2 \right] \\ & H_z K_m \left(\frac{\partial u}{\partial z}, \frac{\partial v}{\partial z} \right) = \frac{1}{\rho_o} (\tau_b^\xi, \tau_b^\eta) \\ & H_z K_s N^2 = 0 \end{aligned}$$

The equation is timestepped much like the model tracer equations, including an implicit solve for the vertical operations and an option for using the third-order upwind advection.

There is also an equation for the turbulent length scale l :

$$\frac{D}{Dt} (l q^2) - \frac{\partial}{\partial z} \left[K_l \frac{\partial l q^2}{\partial z} \right] = l E_1 (P_s + P_b) - \frac{q^3}{B_1} \tilde{W} \quad (100)$$

where \tilde{W} is the wall proximity function:

$$\tilde{W} = 1 + E_2 \left(\frac{l}{kL} \right)^2 \quad (101)$$

$$L^{-1} = \frac{1}{\zeta - z} + \frac{1}{H + z} \quad (102)$$

The form of this equation in the model coordinates becomes

$$\begin{aligned} \frac{\partial}{\partial t} \left(\frac{H_z q^2 l}{mn} \right) + \frac{\partial}{\partial \xi} \left(\frac{H_z u q^2 l}{n} \right) + \frac{\partial}{\partial \eta} \left(\frac{H_z v q^2 l}{m} \right) + \frac{\partial}{\partial s} \left(\frac{H_z \Omega q^2 l}{mn} \right) - \frac{\partial}{\partial s} \left(\frac{K_q}{mn H_z} \frac{\partial q^2 l}{\partial s} \right) = \\ \frac{H_z}{mn} l E_1 (P_s + P_b) - \frac{H_z q^3}{mn B_1} \tilde{W} + \frac{H_z}{mn} \mathcal{D}_{ql}. \end{aligned} \quad (103)$$

where \mathcal{D}_{ql} is the horizontal diffusion of the quantity $q^2 l$.

Given these solutions for q and l , the vertical viscosity and diffusivity coefficients are:

$$K_m = ql S_m + K_{m\text{background}} \quad (104)$$

$$K_s = ql S_h + K_{s\text{background}} \quad (105)$$

$$K_q = ql S_q + K_{q\text{background}} \quad (106)$$

and the stability coefficients S_m , S_h and S_q are found by solving

$$S_s [1 - (3A_2B_2 + 18A_1A_2)G_h] = A_2 [1 - 6A_1B_1^{-1}] \quad (107)$$

$$S_m [1 - 9A_1A_2G_h] - S_s [G_h(18A_1^2 + 9A_1A_2)G_h] = A_1 [1 - 3C_1 - 6A_1B_1^{-1}] \quad (108)$$

$$G_h = \min\left(-\frac{l^2N^2}{q^2}, 0.028\right). \quad (109)$$

$$S_q = 0.41S_m \quad (110)$$

The constants are set to $(A_1, A_2, B_1, B_2, C_1, E_1, E_2) = (0.92, 0.74, 16.6, 10.1, 0.08, 1.8, 1.33)$. The quantities q^2 and q^2l are both constrained to be no smaller than 10^{-8} while l is set to be no larger than $0.53q/N$.

3.11.4 The Large, McWilliams and Doney parameterization

The vertical mixing parameterization introduced by Large, McWilliams and Doney [28] is a versatile first order scheme which has been shown to perform well in open ocean settings. Its design facilitates experimentation with additional or modified representations of specific turbulent processes.

Surface boundary layer The Large, McWilliams and Doney scheme (LMD) matches separate parameterizations for vertical mixing of the surface boundary layer and the ocean interior. A formulation based on boundary layer similarity theory is applied in the water column above a calculated boundary layer depth h_{sbl} . This parameterization is then matched at the interior with schemes to account for local shear, internal wave and double diffusive mixing effects.

Viscosity and diffusivities at model levels above a calculated surface boundary layer depth (h_{sbl}) are expressed as the product of the length scale h_{sbl} , a turbulent velocity scale w_x and a non-dimensional shape function.

$$\nu_x = h_{sbl}w_x(\sigma)G_x(\sigma) \quad (111)$$

where σ is a non-dimensional coordinate ranging from 0 to 1 indicating depth within the surface boundary layer. The x subscript stands for one of momentum, temperature and salinity.

Surface Boundary layer depth The boundary layer depth h_{sbl} is calculated as the minimum of the Ekman depth, estimated as,

$$h_e = 0.7u_* / f \quad (112)$$

(where u_* is the friction velocity $u_* = \sqrt{\tau_x^2 + \tau_y^2 / \rho}$), the Monin-Obukhov depth:

$$L = u_*^3 / (\kappa B_f) \quad (113)$$

(where $\kappa = 0.4$ is von Karman's constant and B_f is the surface buoyancy flux), and the shallowest depth at which a critical bulk Richardson number is reached. The critical bulk Richardson number (Ri_c) is typically in the range 0.25–0.5. The bulk Richardson number (Ri_b) is calculated as:

$$Ri_b(z) = \frac{(B_r - B(d))d}{|\vec{V}_r - \vec{V}(d)|^2 + V_t^2(d)} \quad (114)$$

where d is distance down from the surface, B is the buoyancy, B_r is the buoyancy at a near surface reference depth, \vec{V} is the mean horizontal velocity, \vec{V}_r the velocity at the near surface reference depth and V_t is an estimate of the turbulent velocity contribution to velocity shear.

The turbulent velocity shear term in this equation is given by LMD as,

$$V_t^2(d) = \frac{C_v(-\beta_T)^{1/2}}{Ri_c \kappa} (c_s \epsilon)^{-1/2} d N w_s \quad (115)$$

where C_v is the ratio of interior N to N at the entrainment depth, β_T is ratio of entrainment flux to surface buoyancy flux, c_s and ϵ are constants, and w_s is the turbulent velocity scale for scalars. LMD derive (115) based on the expected behavior in the pure convective limit. The empirical rule of convection states that the ratio of the surface buoyancy flux to that at the entrainment depth be a constant. Thus the entrainment flux at the bottom of the boundary layer under such conditions should be independent of the stratification at that depth. Without a turbulent shear term in the denominator of the bulk Richardson number calculation, the estimated boundary layer depth is too shallow and the diffusivity at the entrainment depth is too low to obtain the necessary entrainment flux. Thus by adding a turbulent shear term proportional to the stratification in the denominator, the calculated boundary layer depth will be deeper and will lead to a high enough diffusivity to satisfy the empirical rule of convection.

turbulent velocity scale To estimate w_x (where x is m - momentum or s - any scalar) throughout the boundary layer, surface layer similarity theory is utilized. Following an argument by Troen and Mahrt [59], Large et al. estimate the velocity scale as

$$w_x = \frac{\kappa u_*}{\phi_x(\zeta)} \quad (116)$$

where ζ is the surface layer stability parameter defined as z/L . ϕ_x is a non-dimensional flux profile which varies based on the stability of the boundary layer forcing. The stability parameter used in this equation is assumed to vary over the entire depth of the boundary layer in stable and neutral conditions. In unstable conditions it is assumed only to vary through the surface layer which is defined as ϵh_{sbl} (where ϵ is set at 0.10). Beyond this depth ζ is set equal to its value at ϵh_{sbl} .

The flux profiles are expressed as analytical fits to atmospheric surface boundary layer data. In stable conditions they vary linearly with the stability parameter ζ as

$$\phi_x = 1 + 5\zeta \quad (117)$$

In near-neutral unstable conditions common Businger-Dyer forms are used which match with the formulation for stable conditions at $\zeta = 0$. Near neutral conditions are defined as

$$-0.2 \leq \zeta < 0 \quad (118)$$

for momentum and,

$$-1.0 \leq \zeta < 0 \quad (119)$$

for scalars. The non dimensional flux profiles in this regime are,

$$\phi_m = (1 - 16\zeta)^{1/4} \quad (120)$$

$$\phi_s = (1 - 16\zeta)^{1/2} \quad (121)$$

In more unstable conditions ϕ_x is chosen to match the Businger-Dyer forms and with the free convective limit. Here the flux profiles are

$$\phi_m = (1.26 - 8.38\zeta)^{1/3} \quad (122)$$

$$\phi_s = (-28.86 - 98.96\zeta)^{1/3} \quad (123)$$

The shape function The non-dimensional shape function $G(\sigma)$ is a third order polynomial with coefficients chosen to match the interior viscosity at the bottom of the boundary layer and Monin-Obukhov similarity theory approaching the surface. This function is defined as a 3rd order polynomial.

$$G(\sigma) = a_0 + a_1\sigma + a_2\sigma^2 + a_3\sigma^3 \quad (124)$$

with the coefficients specified to match surface boundary conditions and to smoothly blend with the interior,

$$a_0 = 0 \quad (125)$$

$$a_1 = 1 \quad (126)$$

$$a_2 = -2 + 3\frac{\nu_x(h_{sbl})}{hw_x(1)} + \frac{\partial_x \nu_x(h)}{w_x(1)} + \frac{\nu_x(h)\partial_\sigma w_x(1)}{hw_x^2(1)} \quad (127)$$

$$a_3 = 1 - 2\frac{\nu_x(h_{sbl})}{hw_x(1)} - \frac{\partial_x \nu_x(h)}{w_x(1)} - \frac{\nu_x(h)\partial_\sigma w_x(1)}{hw_x^2(1)} \quad (128)$$

where $\nu_x(h)$ is the viscosity calculated by the interior parameterization at the boundary layer depth.

Countergradient flux term The second term of the LMD scheme's surface boundary layer formulation is the non-local transport term γ which can play a significant role in mixing during surface cooling events. This is a redistribution term included in the tracer equation separate from the diffusion term and is written as

$$-\frac{\partial}{\partial z} K\gamma. \quad (129)$$

LMD base their formulation for non-local scalar transport on a parameterization for pure free convection from Mailhôt and Benoit [32]. They extend this parameterization to cover any unstable surface forcing conditions to give

$$\gamma_T = C_s \frac{\overline{wT_0} + \overline{wT_R}}{w_T(\sigma)h} \quad (130)$$

for temperature and

$$\gamma_S = C_s \frac{\overline{wS_0}}{w_S(\sigma)h} \quad (131)$$

for salinity (other scalar quantities with surface fluxes can be treated similarly). LMD argue that although there is evidence of non-local transport of momentum as well, the form the term would take is unclear so they simply specify $\gamma_m = 0$.

The interior scheme The interior scheme of Large, McWilliams and Doney estimates the viscosity coefficient by adding the effects of several generating mechanisms: shear mixing, double-diffusive mixing and internal wave generated mixing.

$$\nu_x(d) = \nu_x^s + \nu_x^d + \nu_x^w \quad (132)$$

Shear generated mixing The shear mixing term is calculated using a gradient Richardson number formulation, with viscosity estimated as:

$$\nu_x^s = \begin{cases} \nu_0 & Ri_g < 0, \\ \nu_0[1 - (Ri_g/Ri_0)^2]^3 & 0 < Ri_g < Ri_0, \\ 0 & Ri_g > Ri_0. \end{cases} \quad (133)$$

where ν_0 is 5.0×10^{-3} , $Ri_0 = 0.7$.

Double diffusive processes The second component of the interior mixing parameterization represents double diffusive mixing. From limited sources of laboratory and field data LMD parameterize the salt fingering case ($R_\rho > 1.0$)

$$\nu_s^d(R_\rho) = \begin{cases} 1 \times 10^{-4} [1 - (\frac{R_\rho - 1}{R_\rho^0 - 1})^2]^3 & \text{for } 1.0 < R_\rho < R_\rho^0 = 1.9, \\ 0 & \text{otherwise.} \end{cases} \quad (134)$$

$$\nu_\theta^d(R_\rho) = 0.7\nu_s^d \quad (135)$$

For diffusive convection ($0 < R_\rho < 1.0$) LMD suggest several formulations from the literature and choose the one with the most significant impact on mixing (Fedorov [11]).

$$\nu_\theta^d = (1.5^{-6})(0.909 \exp(4.6 \exp[-0.54(R_\rho^{-1} - 1)])) \quad (136)$$

for temperature. For other scalars,

$$\nu_s^d = \begin{cases} \nu_\theta^d(1.85 - 0.85R_\rho^{-1})R_\rho & \text{for } 0.5 \leq R_\rho < 1.0, \\ \nu_\theta^d 0.15R_\rho & \text{otherwise.} \end{cases} \quad (137)$$

Internal wave generated mixing Internal wave generated mixing serves as the background mixing in the LMD scheme. It is specified as a constant for both scalars and momentum. Eddy diffusivity is estimated based on the data of Ledwell et al. [29]. While Peters et al. [42] suggest eddy viscosity should be 7 to 10 times larger than diffusivity for gradient Richardson numbers below approximately 0.7. Therefore LMD use

$$\nu_m^w = 1.0 \times 10^{-4} m^2 s^{-1} \quad (138)$$

$$\nu_s^w = 1.0 \times 10^{-5} m^2 s^{-1} \quad (139)$$

3.12 Open boundary conditions

Currently, SCRUM has two open boundary conditions options; a gradient condition and a radiation condition. These options are available independently for each of the four sides and for each field category. The four categories are 2-D momentum, 3-D momentum, free surface, and 3-D tracers.

3.12.1 Gradient boundary condition

This boundary condition is extremely simple and consists of setting the gradient of a field to zero at the edge. The outside value is set equal to the closest interior value. It is probably too simple to be useful in realistic problems.

3.12.2 Radiation boundary condition

In realistic domains, open boundary conditions can be extremely difficult to get right. There can be situations where incoming flow and outgoing flow happen along the same boundary or even at the same horizontal location. Orlanski [39] proposed a radiation scheme in which a local phase velocity is computed and used to radiate things out (if it is indeed going out). This works well for a wave propagating normal to the boundary, but has problems when waves approach the boundary at an angle. Raymond and Kuo [46] have modified the scheme to account for propagation in all three directions. In SCRUM, only the two horizontal directions are accounted for:

$$\frac{\partial \psi}{\partial t} = - \left(C_x \frac{\partial \psi}{\partial \xi} + C_y \frac{\partial \psi}{\partial \eta} \right) \quad (140)$$

where

$$C_x = \frac{F \frac{\partial \psi}{\partial \xi}}{\left(\frac{\partial \psi}{\partial \xi}\right)^2 + \left(\frac{\partial \psi}{\partial \eta}\right)^2} \quad (141)$$

$$C_y = \frac{F \frac{\partial \psi}{\partial \eta}}{\left(\frac{\partial \psi}{\partial \xi}\right)^2 + \left(\frac{\partial \psi}{\partial \eta}\right)^2} \quad (142)$$

$$F = -\frac{\partial \psi}{\partial t} \quad (143)$$

These terms are evaluated at the closest interior point in a manner consistent with the timestepping scheme used. The phase velocities are limited so that the local CFL condition is satisfied. They are then applied to the boundary point using equation (140), again using a consistent timestepping scheme. Raymond and Kuo give the form used for centered differencing and a leapfrog timestep while SCRUM uses one-sided differences.

The radiation approach is appropriate for waves leaving the domain. A check is made to see which way the phase velocity is headed. If it is entering the domain, a zero gradient condition is applied.

4 Details of the Code

4.1 Main subroutines

The main program is in **scrum.F**. It calls the initialization routines and then calls either **main3d** or **main2d**. A flow chart for **main3d** is shown in Fig. 6. The boxes refer to subroutines which are described as follows:

- 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 ($R_{u_{\text{slow}}}$, $R_{v_{\text{slow}}}$) 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. 7.
- inp_par** Reads in input model parameters from standard input. It also writes out these parameters to standard output and calls **checkdefs**.
- omega** Calculates the scaled vertical velocity $H_z\Omega/mn$ according to equation (79).
- prsgrd** Calculates the horizontal pressure gradients according to equation (80).
- rho_eos** Calculates the density anomaly, ρ , using the equation of state (§3.8).
- set_vbc** Sets the vertical boundary conditions for momentum and tracers.
- step2d** Time steps free-surface and 2-D momentum equations. Also does the time-averaging described in §3.5 and calls the 2-D boundary condition routines.
- step3d** Time steps the 3-D momentum and tracers (usually potential temperature and salinity), using the tridiagonal solver described in §3.4. 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. It also calls the boundary condition routines.
- trhs** Calculates and stores contributions to the right-hand-side of the tracer equations (21 and 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 (54), 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 (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 (55), where all the terms other than $\frac{\partial}{\partial t}(\frac{Dv}{mn})$ have been moved to the right-hand-side.

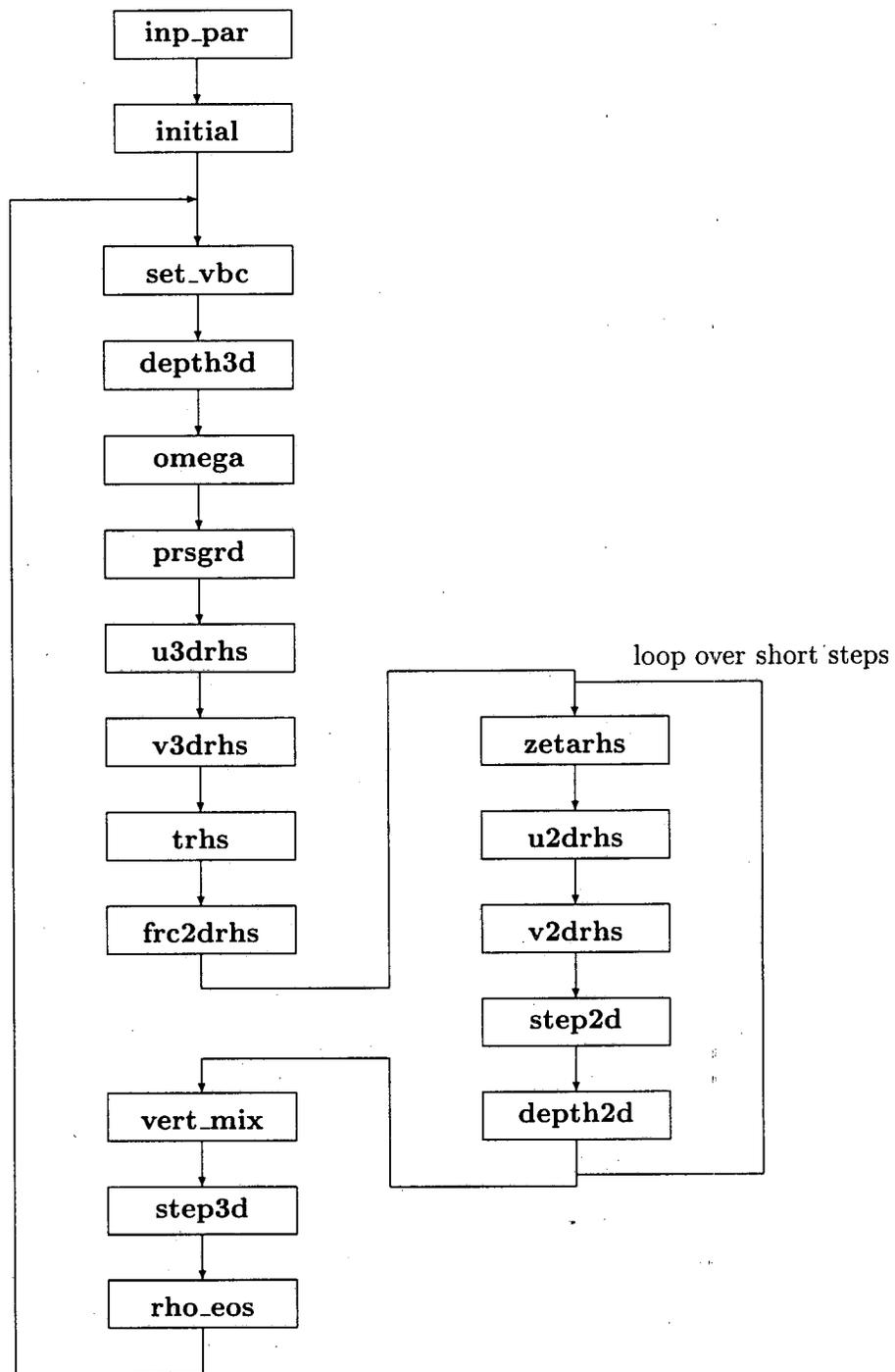


Figure 6: Flow chart of the model main program.

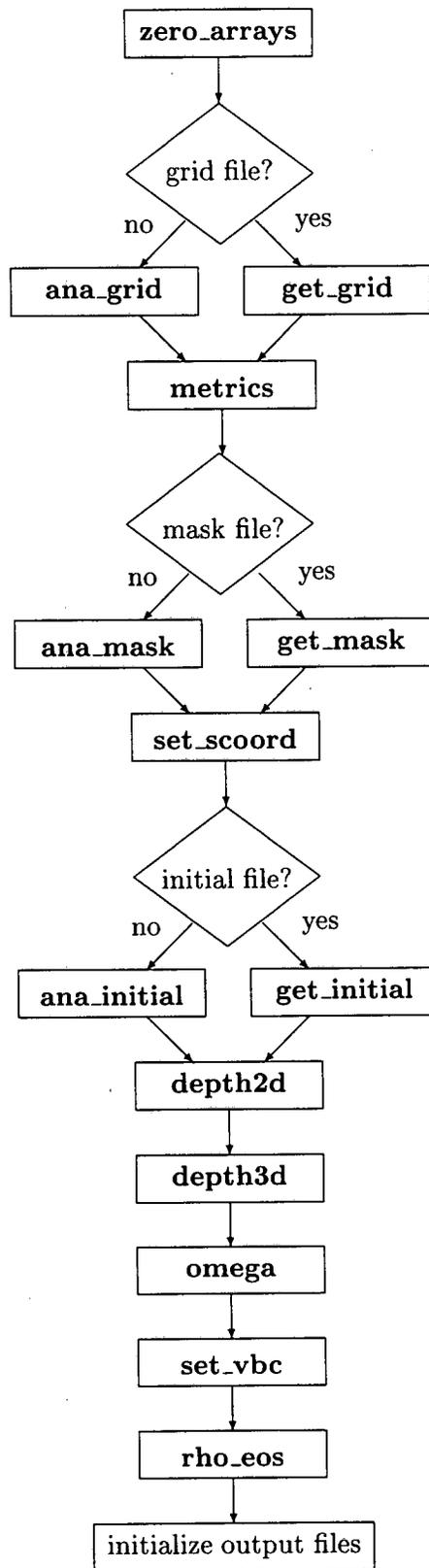


Figure 7: Flow chart of the initial subroutine.

- v3drhs** Calculates and stores contributions to the right-hand-side of equation (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 (53), 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.
- ana_mask** Sets up an analytic mask.
- blkdat** Initializes some variables and parameters stored in common blocks.
- checkdefs** Reports on which C preprocessor variables have been **#defined** 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.
- 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_smflux Computes analytic kinematic surface momentum flux (wind stress).

ana_srflux Computes analytic kinematic surface shortwave radiation.

ana_ssh Computes analytic sea surface height and $dQdSST$ which are used in the surface heat flux correction.

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.

ana_tclima Computes analytic tracer climatology fields.

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_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_ssh Reads sea surface height 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.

get_tclima Reads climatology of tracer type variables from the climatology NetCDF file, and then linearly time-interpolates to current model time.

set_nudgcof Set nudging coefficients time-scales (1/s).

Horizontal mixing The horizontal mixing routines have options for doing Laplacian or biharmonic mixing, along surfaces of constant s , z , or density, as described in §3.10. The horizontal mixing of 2-D momentum is computed in **u2drhs** and **v2drhs**.

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

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

set_hmixing Sets horizontal mixing coefficients. If requested, it also scales horizontal mixing by the grid size and enhances horizontal mixing in the sponge areas.

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

smagorinsky This routine computes horizontal mixing coefficients for momentum using Smagorinsky parameterization (Tag et al. [57]).

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 $N^2 = -\frac{g}{\rho_0} \frac{\partial \rho}{\partial z}$.

lmd_vmix Computes vertical mixing coefficients for momentum and tracers at the ocean interior using the Large, McWilliams and Doney [28] 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_wscales 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 [36] mixing level 2.5 scheme with modifications described in Galperin et al. [14].

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

pp_vmix Computes vertical mixing coefficients for momentum and tracers using the Pacanowski and Philander [40] 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 [48]).

Bottom boundary-layer model The model has an optional bottom boundary layer based on Styles and Glenn [56].

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 [56] 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 .

inflow Processes prescribed inflow open boundary conditions from climatology data.

obc_volcons Computes integral mass flux across all the open boundaries. Then, it corrects 2D velocities across to enforce global mass conservation.

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

ab_ratio Calculates the ratio of the thermodynamic expansion coefficients for potential temperature and salinity, α/β , at horizontal and vertical w -points from a polynomial expression (Jackett and McDougall [25]).

alfabeta Computes thermal expansion and saline contraction coefficients as a function of potential temperature, salinity, and pressure from a polynomial expression (Jackett and McDougall [25]).

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 [50] 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 F. 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:

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.

GS_FRONT Define for idealized Gulf Stream front example.

MUNK Define for Stommel/Munk wind driven ocean basin.

OVERFLOW Define for the overflow example.

RIVERPLUME Define for the river plume example.

SEAMOUNT Define for the seamount example.

SOLITON Define for the equatorial Rossby soliton example.

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

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_GSCHEME Define for third-order upwind advection scheme.

UV_PRS Define to compute the horizontal pressure gradient term.

UV_PSOURCE Define for point sources/sinks.

UV_VIS2 Define to compute the horizontal Laplacian viscosity.

UV_VIS4 Define to compute the horizontal biharmonic viscosity.

WJ_PRS Define for weighted Jacobian pressure gradient.

tracers

DIAGNOSTIC Define for a diagnostic calculation in which the tracer fields do not change in time.

ICE Define to use ice component of the model (see §8.
ICE_THERMO Define for ice thermodynamics.
NONLINE_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.
TS_GSCHEME Define for third-order upwind advection scheme.
TS_PSOURCE Define for point sources/sinks.

general model configuration

AVERAGES Define to write out time-averaged model fields.
RMDOCINC Define to remove documentation in include files with the C preprocessor.
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_DIAG Define for customized diagnostics.
ANA_GRID Define for an analytic model grid set-up.
ANA_INITIAL Define for analytic initial conditions.
ANA_MASK Define for an analytic mask.
ANA_MEANRHO Define for an analytic mean density anomaly.
ANA_PSOURCE Define for analytic point sources.
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_SSH Define for an analytic sea surface height.
ANA_SST Define for an analytic SST and $\partial Q/\partial SST$.
ANA_STFLUX Define for an analytic kinematic surface heat flux.
ANA_TCLIMA Define for an analytic tracer climatology.
ANA_VMIX Define for analytic vertical mixing coefficients.
ANA_WWAVE Define for an analytic wind induced wave field.

horizontal mixing of momentum

MIX_GP_UV Define for viscosity along constant z (geopotential) surfaces.
 MIX_EPI_UV Define for viscosity along constant *in situ* density (epineutral) surfaces.
 MIX_ISO_UV Define for viscosity along constant potential density (isopycnal) surfaces.
MIX_S_UV Define for viscosity along constant s surfaces.
SMAGORINSKY Define for Smagorinsky mixing parameterization.
VIS_GRID Define for horizontally variable viscosity coefficient.

horizontal mixing of tracers

DIF_GRID Define for horizontally variable diffusion coefficient.
MIX_GP_TS Define for diffusion along constant z (geopotential) surfaces.
 MIX_EPI_TS Define for diffusion along constant *in situ* density (epineutral) surfaces.
 MIX_ISO_TS Define for diffusion along constant potential density (epineutral) surfaces.
MIX_S_TS Define for diffusion along constant s surfaces.

vertical mixing

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_NONLOCAL Define to add convective nonlocal transport.
 LMD_RIMIX Define to add diffusivity due to shear instabilities.
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 .
 Q_GSCHEME Define for third-order upwind advection 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.
INFLOW2D Define to use SSH climatology as 2D inflow data.
INFLOW3D Define to process 3D inflow from tracer climatology.
NS_PERIODIC Define for periodic boundaries in the j direction.

detailed eastern open boundary conditions—other sides have similar

EAST_FSGRADIANT Define for a gradient condition on the free surface.
EAST_M2GRADIANT Define for a gradient condition on the 2-D momentum.

EAST_M2RADIATION Define for a radiation condition on the 2-D momentum.
EAST_M3GRADIENT Define for a gradient condition on the 3-D momentum.
EAST_M3RADIATION Define for a radiation condition on the 3-D momentum.
EAST_TGRADIENT Define for a gradient condition on the tracers.
EAST_TRADIATION Define for a radiation condition on the tracers.

general

MASKING Define if there is land in the domain to be masked out.
M2NUDGING Define for nudging to 2-D momentum data.
M3NUDGING Define for nudging to 3-D momentum data.
TCLIMATOLOGY Define for processing the tracer climatology arrays.
TNUDGING Define for nudging to tracer climatology.
ZCLIMATOLOGY Define for processing the sea surface height climatology arrays.
ZNUDGING Define for nudging to sea surface height climatology.

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

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.

DBLEPREC For double precision arithmetic.

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

OUT_DOUBLE For double precision output.

NF_FOUT Either **nf_double** or **nf_real**, depending on **OUT_DOUBLE**.

NF_FTYPE Either **nf_double** or **nf_real**, depending on model precision.

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**.

isalt	Index into tracer arrays for salinity.
itemp	Index into tracer arrays for temperature.
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.
Nsrc	Maximum number of point sources/sinks.
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 $\pi = 3.14159265\dots$).

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, $f = f_0 + \beta y$.

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 `scrum_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 or computed in `ana_mask`. The mask is defined on ρ -points; see Fig. 8 for an example of a small domain with an isolated island and a promontory adjacent to the boundary. There 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 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

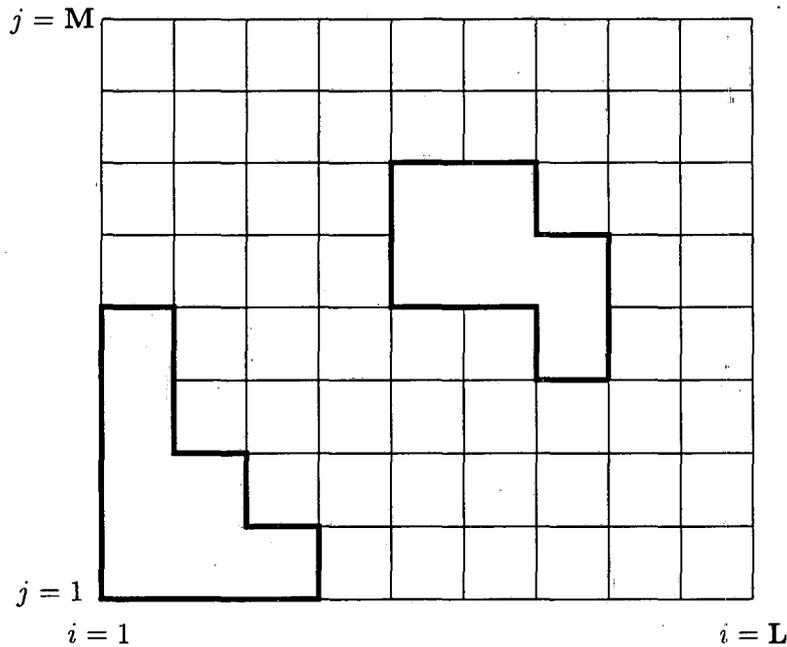


Figure 8: Small grid with masked regions

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. 9. This represents the same mask as in Fig. 8, with a circle for each ρ -point, including the boundary “image” points. The red circles are land while the blue circles are ocean. Notice that I have made the “image” points have the same mask value as the points they mirror.

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 ($\partial Q/\partial \text{SST}$).

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 [8]. A comprehensive description of this methodology can also be found in Gandin [15], Bretherton et al. [7], McWilliams et al. [34], Daley [9], Bennett [6], and others.

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

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

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 $w = (w_1, \dots, w_N)$. The overbar denotes an expected or ensemble 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 convenience, matrix notation has been used. $s = [s_1, \dots, s_N]$ 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:

$$C((\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2)) = C(\mathbf{x}_1 - \mathbf{x}_2, |t_2 - t_1|)$$

and errors at two different locations and times are uncorrelated:

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

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

$$C(\mathbf{x}_1 - \mathbf{x}_2, |t_2 - t_1|) = C(|\mathbf{x}_1 - \mathbf{x}_2|, |t_2 - t_1|)$$

with

$$C(\mathbf{r}, \tau) = \exp \left[- \left(\frac{\tau}{\tau_0} \right)^2 \right] G(\mathbf{r})$$

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

where τ_0 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 **nnc** influential 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/scrump/tars/scrump3_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 field, 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 snapshots 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_{ref} terms from §2.2, the incoming shortwave radiation used by the Large et al. mixing scheme, and the wave information used by the Styles and Glenn bottom boundary layer. The ice thermodynamics also requires forcing fields such as air temperature and cloud fraction.

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

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

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

5.5 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/scrump/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.

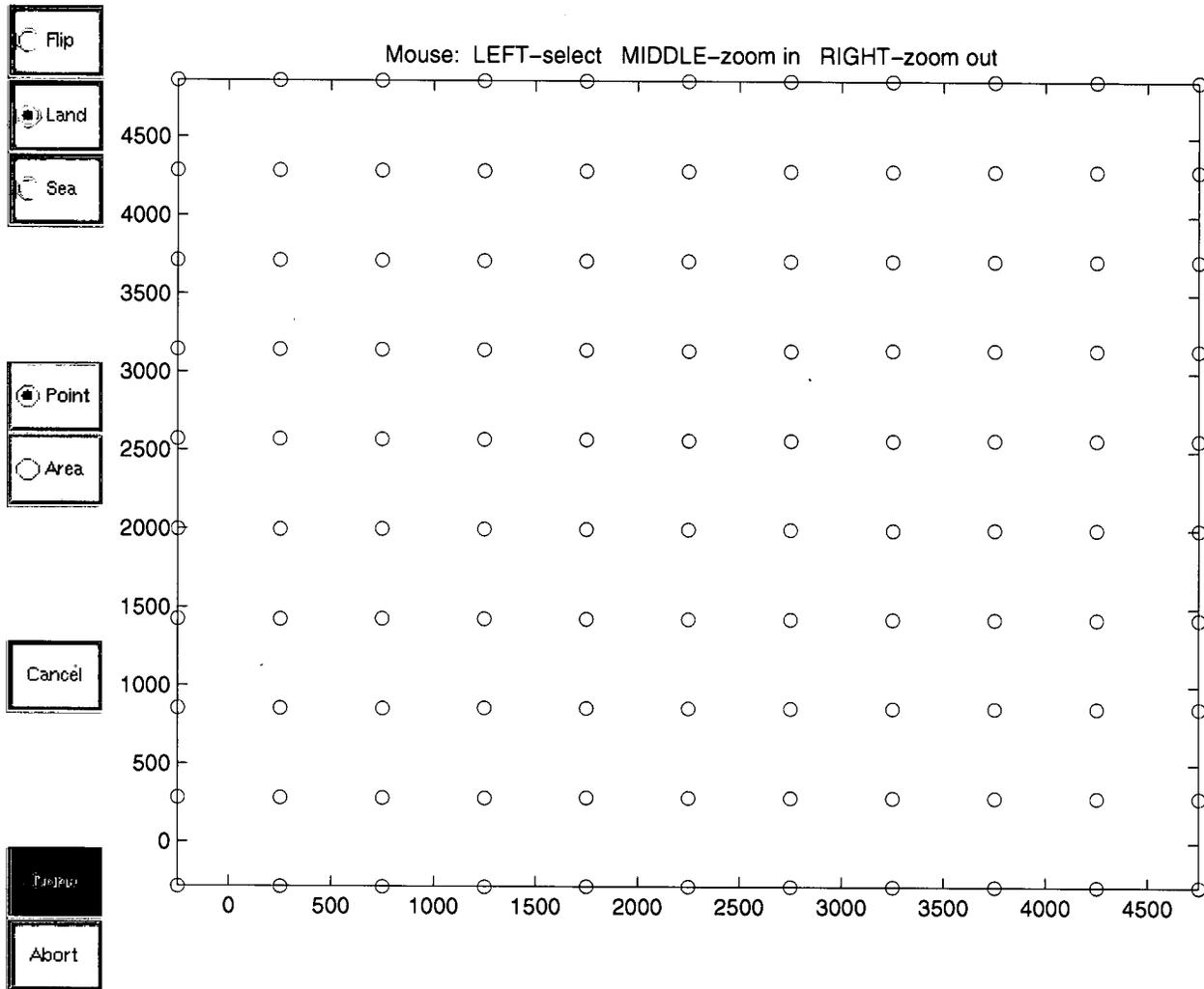
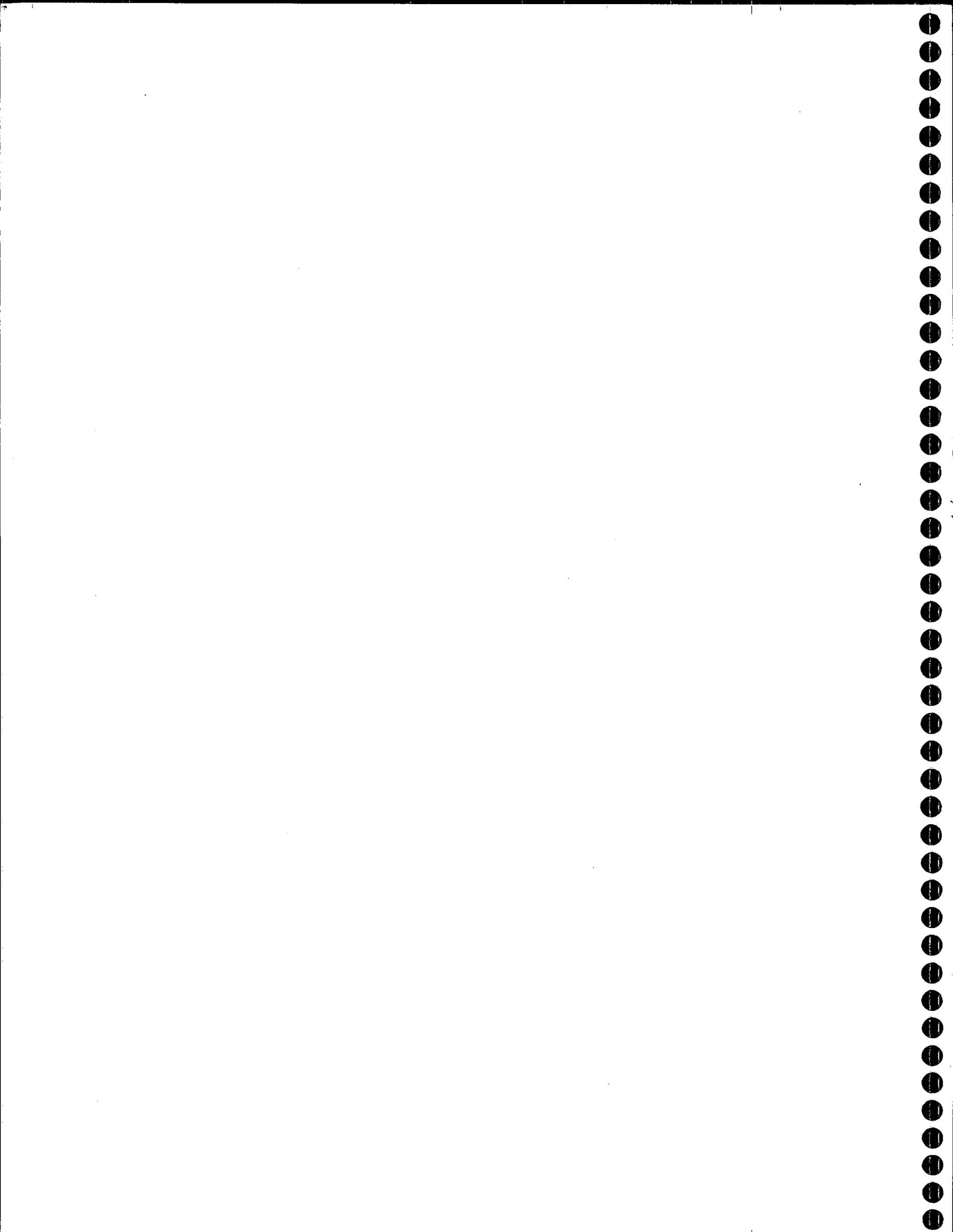


Figure 9: The **scrum_mask** program in action



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 in Haidvogel and Beckmann [19]. Some of them are listed here:

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 [18] 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 [5] and McCalpin [33] for more information.

UPWELLING The upwelling/downwelling example was contributed by Anthony Macks and Jason Middleton [31] 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/scrump/tars/scrump3_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 **scrump.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.

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 $L \geq 2$, $M \geq 2$, $N \geq 2$ and $NT \geq 1$. **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 [61]. 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. 2. The overall grid is shown in Fig. 10. 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 tracer climatology fields also require appropriate values if they are to be used, and are provided by `ana_tclima` or `get_tclima`. Likewise, the surface height climatology is read by `get_ssh` or provided by `ana_ssh`.

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 [25] and a linear $\rho(T, S)$. The linear form is $\rho = \mathbf{R0} + \mathbf{Tcoef} \cdot T + \mathbf{Scoef} \cdot S$ or $\rho = \mathbf{R0} + \mathbf{Tcoef} \cdot T$, 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`.

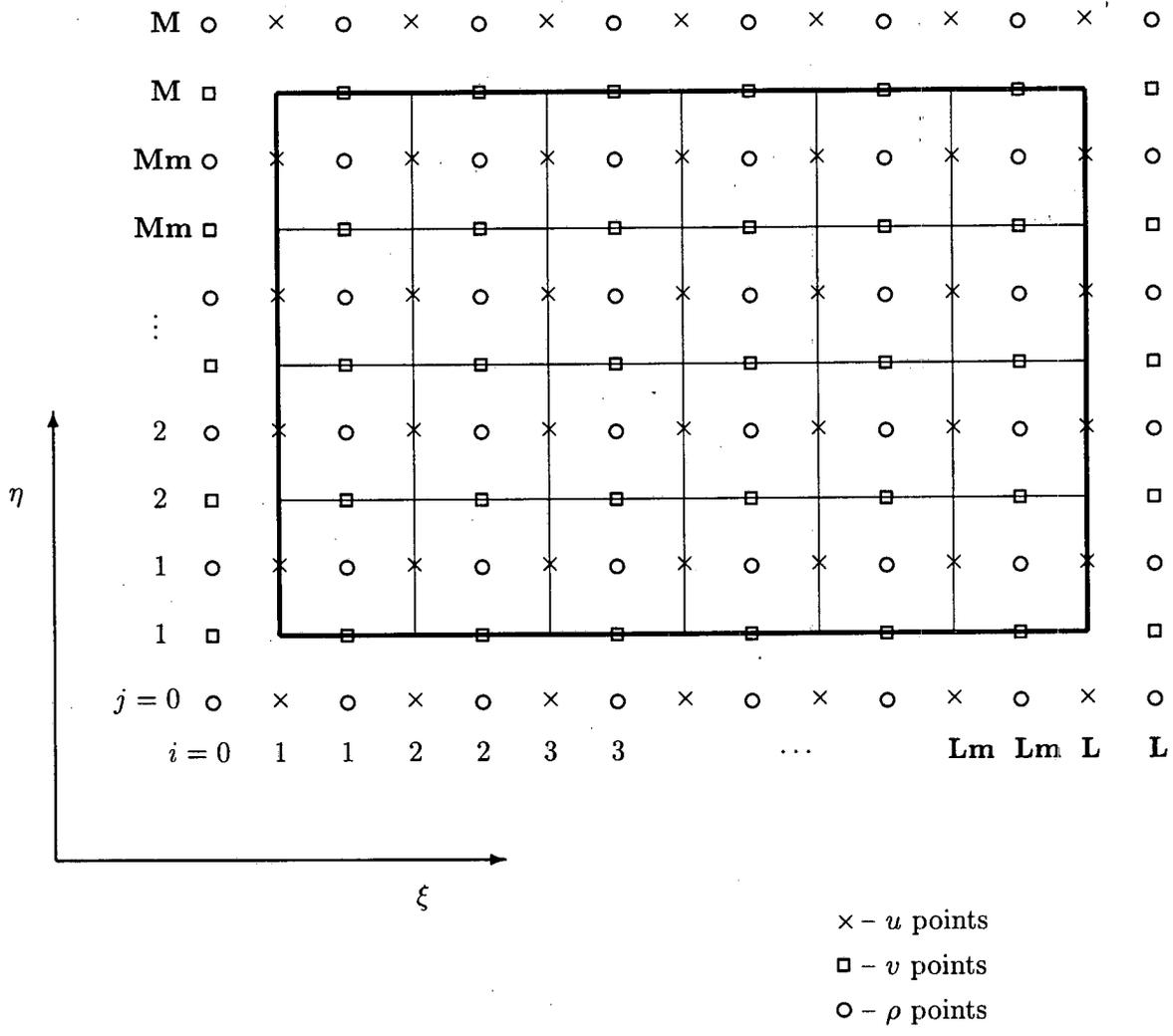


Figure 10: The whole grid.

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 $u, v, \bar{u}, \bar{v}, T, S$ and ζ . They are currently configured for a closed basin, a periodic channel, a doubly periodic domain or an open domain with radiation conditions. The open boundary conditions are evolving; the latest efforts are in the ROMS model.

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 used in the North Atlantic simulations in sponge layers along the northern and southern boundaries. Set the climatologies in **ana_tclima** or in a file read by **get_tclima**, set **NUDGING** in **cppdefs.h** and also set the array **nudgcof** in **set_nudgcof.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.

nrreci Record number of the ice restart file to read as the initial conditions (coupled ice model only).

nrst Number of timesteps between writing of restart fields.

nwrst 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 Isopycnal thicknesses.

Kdiff Isopycnal mixing thickness diffusivity for each tracer variable. A value is expected for each of the **NT** tracers. These isopycnal thicknesses are used when the Gent/McWilliams isopycnic mixing is activated (not currently implimented).

6 Horizontal viscosity coefficients.

uvnu2 Constant mixing coefficient for the horizontal Laplacian viscosity.

uvnu4 Constant mixing coefficient for the horizontal biharmonic viscosity.

7 Vertical mixing coefficients for tracers.

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

8 Vertical mixing coefficient for momentum.

akv_bak Background vertical mixing coefficient for momentum.

9 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.

10 Bottom drag coefficients.

rdrng Linear bottom drag coefficient.

rdrng2 Quadratic bottom drag coefficient.

Zo Bottom roughness.

11 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 body force. Used when the C-preprocessor option **BODYFORCE** is defined.

levbfrc Shallowest level to apply bottom momentum stresses as a body force. Used when the C-preprocessor option **BODYFORCE** is defined.

12 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 .

13 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.

14 Nudging/relaxation time scales.

Znudg Time scale (days) of nudging towards sea surface height climatology.

M2nudg Time scale (days) of nudging towards 2-D momentum climatology.

M3nudg Time scale (days) of nudging towards 3-D momentum climatology.

Tnudg Time scale (days) of nudging towards tracer climatology. A value is expected for each of the **NT** tracers.

15 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.

16 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 ($i = L$), **.true.** if it is a wall, **.false.** if it is open.

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

17 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.

18 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.

19 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.

20 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:

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

Enter values in ascending numerical order.

21 String with a maximum of eighty characters.

title Title of the model run.

22 String with a maximum of eighty characters.

rstname Output restart file name (NetCDF).

23 String with a maximum of eighty characters.

hisname Output history file name (NetCDF).

24 String with a maximum of eighty characters.

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

25 String with a maximum of eighty characters.

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

26 String with a maximum of eighty characters.

fltname Name of the file containing the float output (NetCDF). Not implemented yet.

27 String with a maximum of eighty characters.

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

28 String with a maximum of eighty characters.

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

29 String with a maximum of eighty characters.

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

30 String with a maximum of eighty characters.

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

- 31 String with a maximum of eighty characters.
assname Name of the file containing the assimilation fields (NetCDF).
- 32 String with a maximum of eighty characters.
aparnam Name of the file containing the assimilation parameters (ASCII).
- 33 String with a maximum of eighty characters.
sposnam Name of the file containing the stations positions (ASCII).
- 34 String with a maximum of eighty characters.
fposnam Name of the file containing the initial drifter positions (ASCII).
- 35 String with a maximum of eighty characters.
iiname Name of the ice input file (coupled ice model only).
- 36 String with a maximum of eighty characters.
irstnam Name of the ice restart file (coupled ice model only).
- 37 String with a maximum of eighty characters.
ihisnam Name of the ice history file (coupled ice model only).
- 38 String with a maximum of eighty characters.
iavgnam Name of the ice averages file (coupled ice model only).
- 39 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  Kdiff[1:NT] (m2/s)
   0.d0        0.d0
6  UVNU2 (m^2/s),  UVNU4 (m^4/s)
   10.d0       0.d0
7  AKT_BAK[1:NT] (m^2/s)
   1.0d-5     1.0d-5
8  AKV_BAK (m^2/s)
   1.0d-4
9  AKQ_BAK (m^2/s)  Q2NU2 (m^2/s),  Q2NU4 (m^4/s)
   1.0d-4         20.d0         1.0d+07
10 RDRG (m/s),      RDRG2,      Zo (m)

```

```

4.5E-04 /      0.d0      0.d0
11  NMIX_EN,    ADV_ORD,  LEVSFRC,  LEVBFRC
      1          2          1          1
12  THETA_S,   THETA_B,   TCLINE (m)
      3.d0      0.d0      50.d0
13  RHO0 (Kg/m^3),  DSTART (days)
      1025.d0    0.d0
14  ZNUDG (days), M2NUDG (days), M3NUDG(days), TNUDG[1:NT] (days)
      0.d0      0.d0      0.d0      0.d0      0.d0
15  R0 (Kg/m^3)  TO (deg C),  S0 (PSU),    TCOEF,      SCOEUF
      1026.9524  10.d0      35.d0      -1.67e-04  7.62e-04
16  GAMMA2,    WALL1,    WALL2,    WALL3,    WALL4
      1.d0      T        F        F        F
17  wrtU,     wrtV,     wrtW,     wrtO,     wrtUBAR,  wrtVBAR,  wrtZ
      T        T        T        F        F        F        T
18  wrtT(1:NT) (temperature, salinity, etc.)
      T        T
19  wrtRHO,   wrtAKV,   wrtAKT,   wrtAKS,   wrtHBL
      F        F        F        F        F
20  NLEV,    LEV(1:NLEV) in ascending order (if NLEV<0, all levels are saved)
      -1      1 3 5
21  TITLE (a80)
Scrum 4.0
22  RSTNAME (a80): SCRUM output restart file name, if any.
scrum_rst.nc
23  HISNAME (a80): SCRUM output history file name, if any.
scrum_his.nc
24  AVGNAM (a80): SCRUM output averages file name, if any.
scrum_avg.nc
25  STANAME (a80): SCRUM output stations file name, if any.
scrum_sta.nc
26  FLTNAME (a80): SCRUM output floats file name, if any.
scrumflt.nc
27  GRDNAME (a80): SCRUM input grid file name, if any.
scrum_grd.nc
28  ININAME (a80): SCRUM input initial conditions file name, if any.
scrum_ini.nc
29  FRCNAME (a80): SCRUM input forcing fields file name, if any.
scrum_frc.nc
30  CLMNAME (a80): SCRUM input climatology fields file name, if any.
scrum_clm.nc
31  ASSNAME (a80): SCRUM input assimilation fields file name, if any.
scrum_ass.nc
32  APARNAM (a80): SCRUM input assimilation parameters file name, if any.
assimilation.dat
33  SPOSNAM (a80): SCRUM input station positions file name, if any.
stations.dat
34  FPOSNAM (a80): SCRUM input initial floats positions file name, if any.
floats.dat
35  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 §H). 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 [31]. 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 $A_v = 0.01m^2/s$, 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`:

```
L = 42
M = 81
N = 16
NT = 2.
```

6.2.3 ana_grid

For this geometry one has a choice of using the grid-generation programs described in Wilkin and Hedstrom [61], 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 $14 + 8e^{z/50}$ 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 `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_0 , or $10^{-4} m^2 s^{-2}$.

We need to edit `ana_vmix` to make sure that the vertical viscosity A_{kv} 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).
0.000E+00	Znudg	Nudging/relaxation inverse time scale (1/days) for free-surface.
0.000E+00	M2nudg	Nudging/relaxation inverse time scale (1/days) for 2D momentum.
0.000E+00	M3nudg	Nudging/relaxation inverse time scale (1/days) for 3D momentum.
0.000E+00	Tnudg(1)	Nudging/relaxation inverse time scale (1/days) for tracer 1.

0.000E+00	Tnudg(2)	Nudging/relaxation inverse time scale (1/days) for tracer 2.
0.0000	TO	Background potential temperature (Celsius) constant.
0.0000	SO	Background salinity (PSU) constant.
30.3795	RO	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. 11 to 14.

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

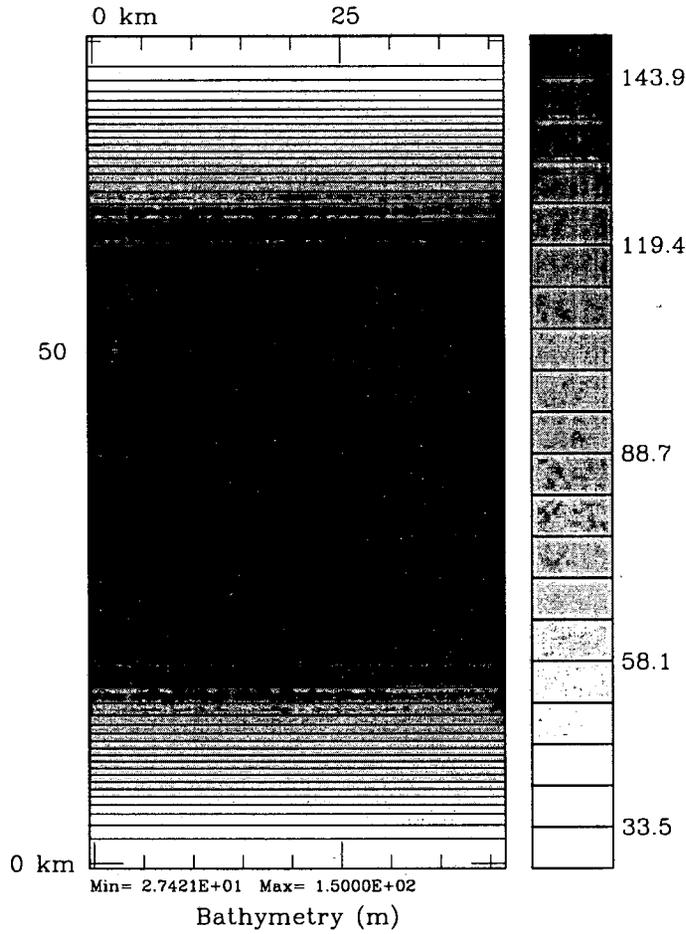
```
:
# if defined DAMEE_B || defined DAMEE_S
#define UV_ADV
#define UV_GSCHEME
#undef UV_VIS2
#undef UV_VIS4
#define UV_PRS
#define UV_COR
#undef MIX_GP_UV
#define TS_ADV
#define TS_GSCHEME
#undef TS_DIF2
#undef TS_DIF4
#undef MIX_GP_TS
#undef SMOLARKIEWICZ
#define NONLIN_EOS
#define SALINITY
#undef 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
```

SCRUM 3.0

Wind-Driven Upwelling/Downwelling Test over a Periodic Channel

Theta s=3, Theta b=0, Tcline=50; nu(v,t)=(0,0) m²/s

0.00 Day



Monday - April 28, 1997 - 5:11:01 PM
scrum_his.nc

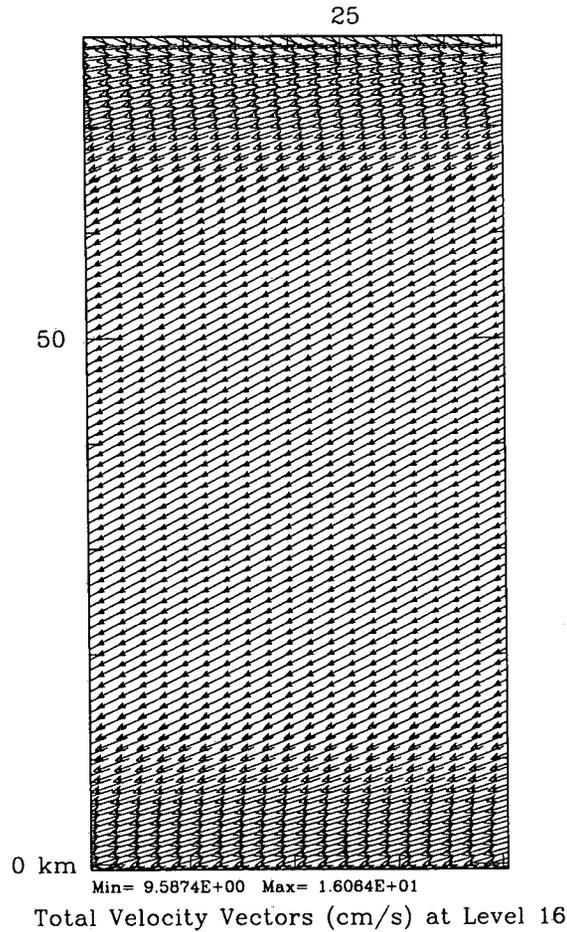
Figure 11: The upwelling/downwelling bathymetry.

SCRUM 3.0

Wind-Driven Upwelling/Downwelling Test over a Periodic Channel

Theta s=3, Theta b=0, Tcline=50; nu(v,t)=(0,0) m²/s

1.00 Day



Monday - April 28, 1997 - 4:18:11 PM
scrue_his.nc

Figure 12: Surface velocities after one day, showing the flow to the left of the wind (southern hemisphere).

SCRUM 3.0

Wind-Driven Upwelling/Downwelling Test over a Periodic Channel

Theta s=3, Theta b=0, Tcline=50; nu(v,t)=(0,0) m²/s

1.00 Day

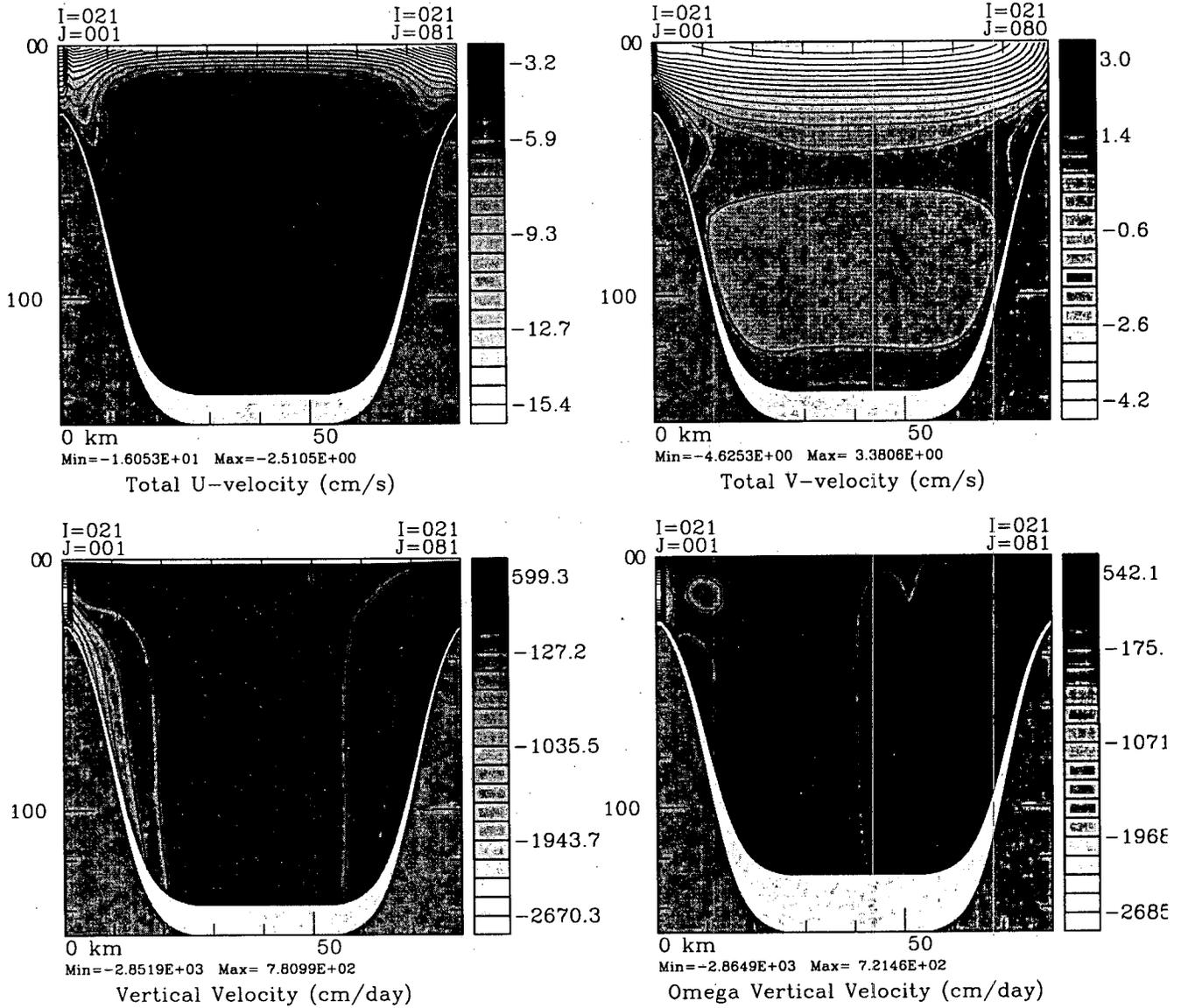


Figure 13: Constant ξ slices of the u, v, w and Ω fields at day 1.

SCRUM 3.0

Wind-Driven Upwelling/Downwelling Test over a Periodic Channel

Theta s=3, Theta b=0, Tcline=50; nu(v,t)=(0,0) m²/s

1.00 Day

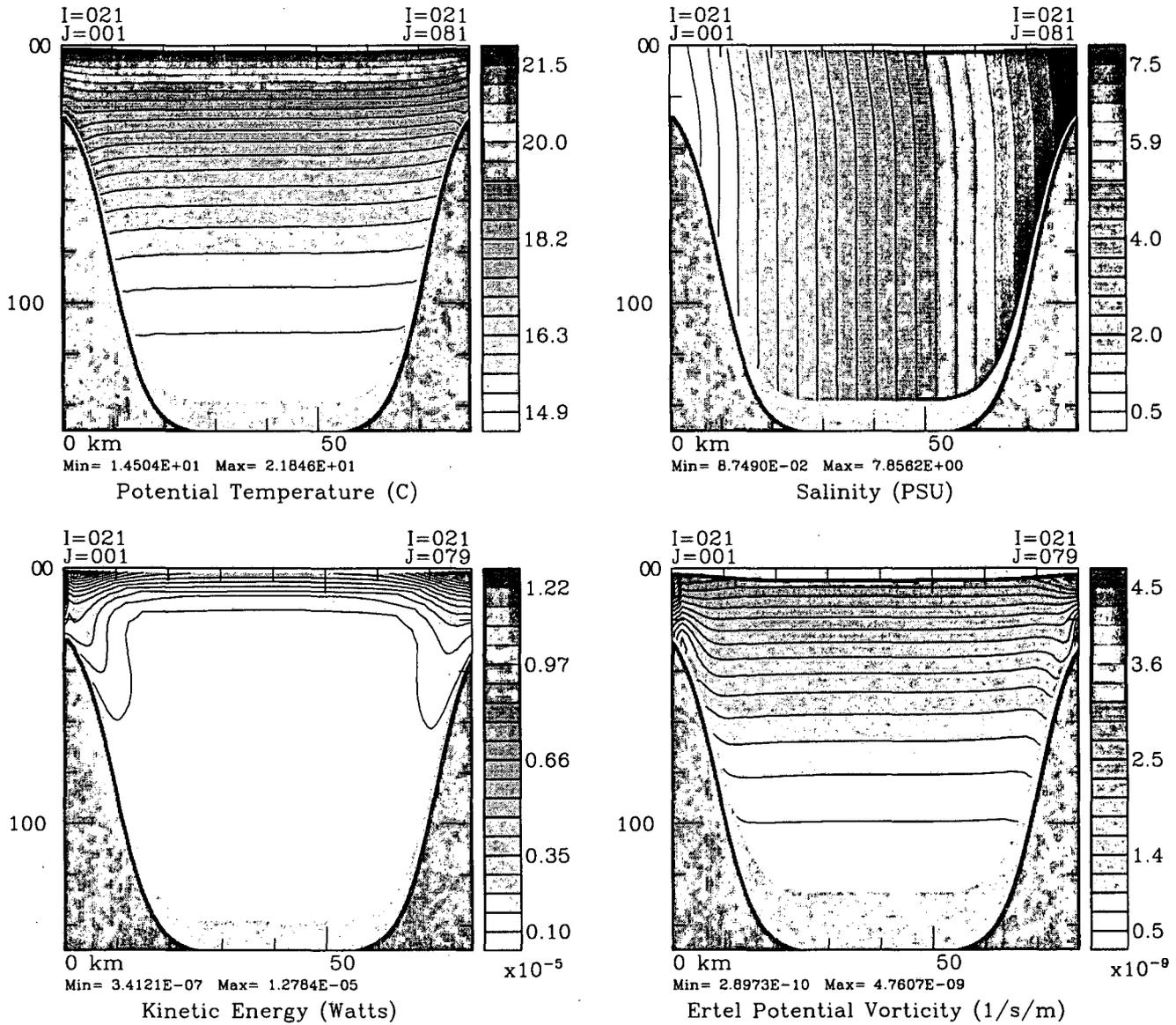


Figure 14: Constant ξ slices of the T , S (tracer), kinetic energy and Ertel potential vorticity at day 1.

```

#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:

```

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. 15 and the unsmoothed bathymetry is shown in Fig. 16. 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 advised 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. 17. 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**. After trying a number of options, we ended up with walls to the north and south with nudging regions (see below).

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 Gibraltar. We edited **set_nudgcof.F** to set the **nudgcof** array accordingly.

North Atlantic DAMEE #4

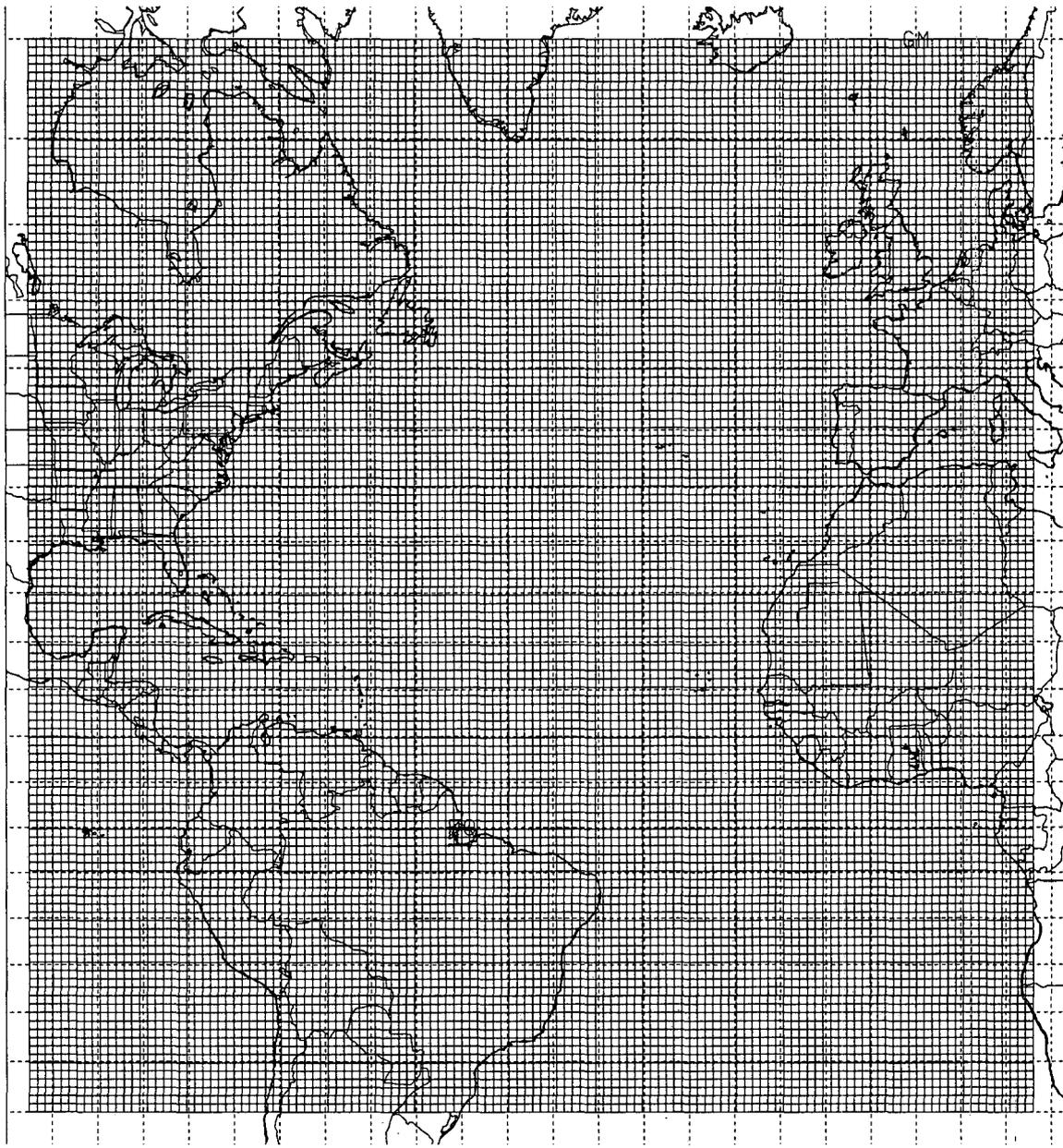


Figure 15: The North Atlantic grid.

Bottom Topography

MIN DEPTH = 200.000

MAX DEPTH = 5500.0

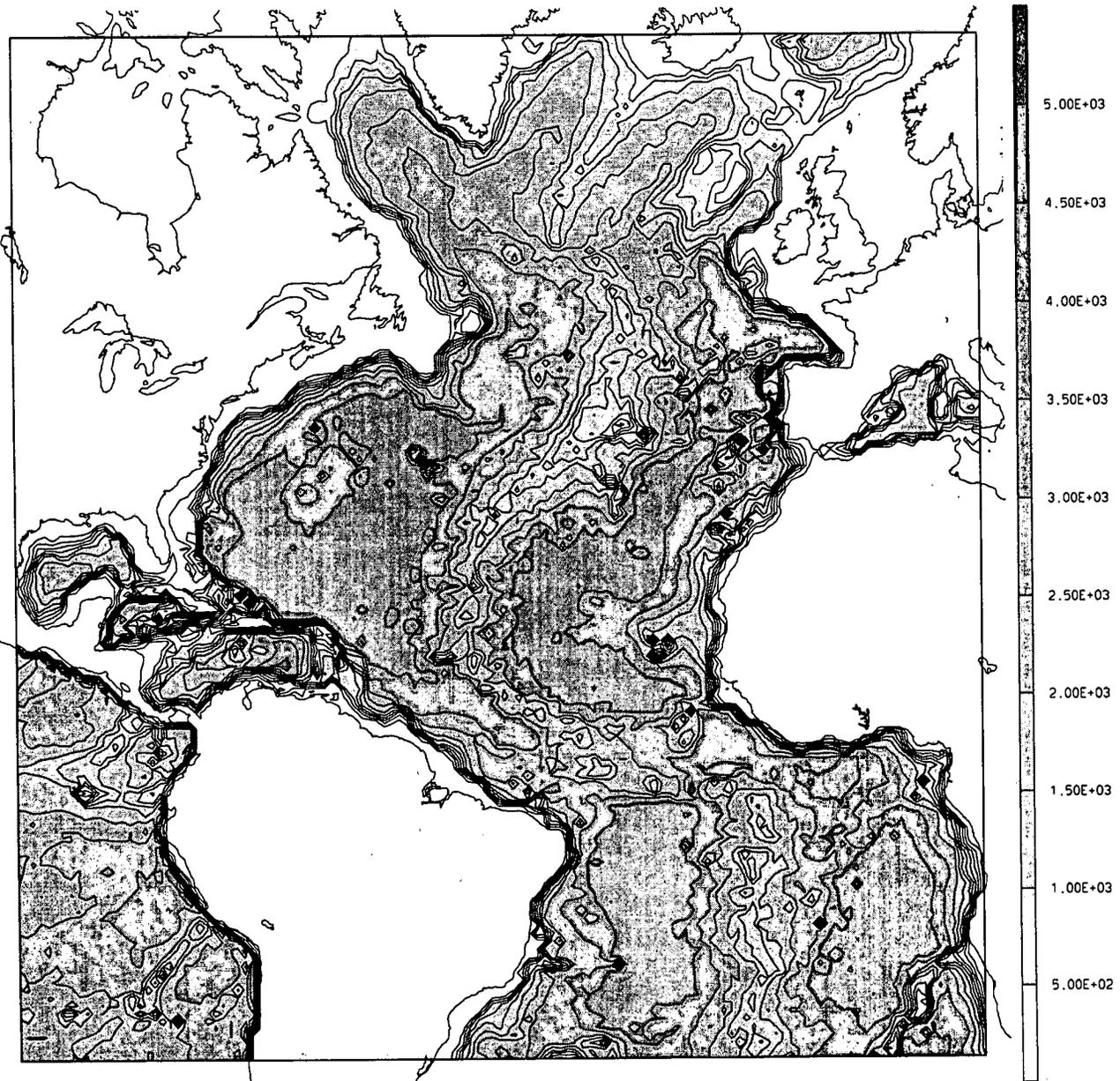


Figure 16: The raw bathymetry from etopo5.

Bottom Topography

MIN DEPTH = 200.000

MAX DEPTH = 5500.0

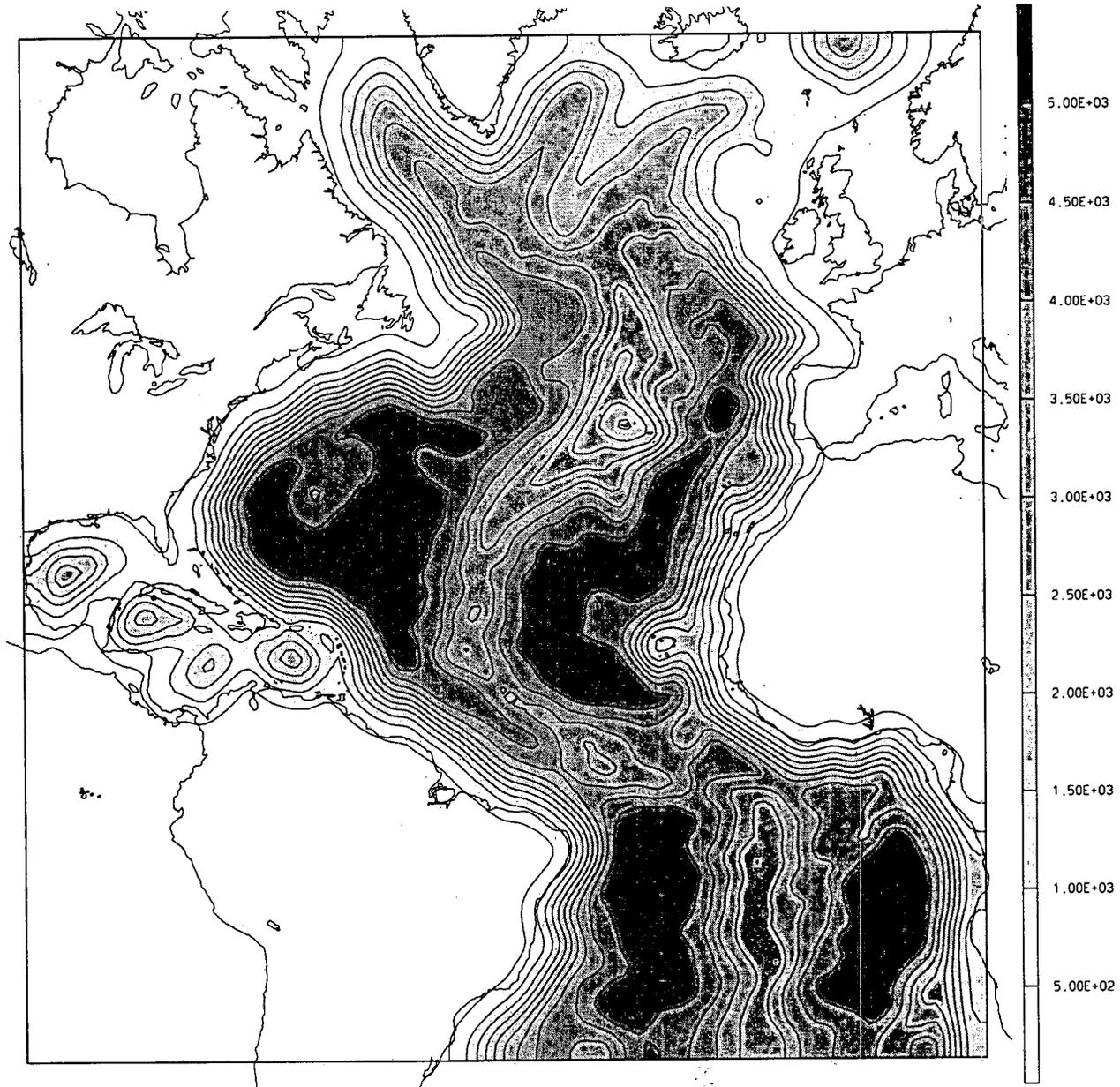


Figure 17: The smoothed North Atlantic bathymetry.

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 is turned off (see above) since the third-order upwind scheme is providing the smoothing. 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-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	rdrgr	Linear bottom drag coefficient (m/s).
0.000E+00	rdrgr2	Quadratic bottom drag coefficient.
18	levsfrc	Deepest level to apply surface stress as a body force.
1	levbfrc	Shallowest level to apply bottom stress as a body force.
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_GSCHEME Shchepetkin-McWilliams G-Scheme Advection of tracer.
 UV_ADV Advection of momentum.
 UV_COR Coriolis term.
 UV_GSCHEME Shchepetkin-McWilliams G-Scheme Advection of momentum.
 UV_PRS Hydrostatic pressure gradient term.

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_TCLIMA - Read climatology of tracer 01 for time = 0.0000E+00
 GET_TCLIMA - 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. 18.

ROMS 1.0

North Atlantic Damee #4: Run 27

LMD mixing, G-Scheme, $\nu(v,t)=(0,0)$, $\Theta=(5,0.4)$, $N=20$

Sea surface elevation

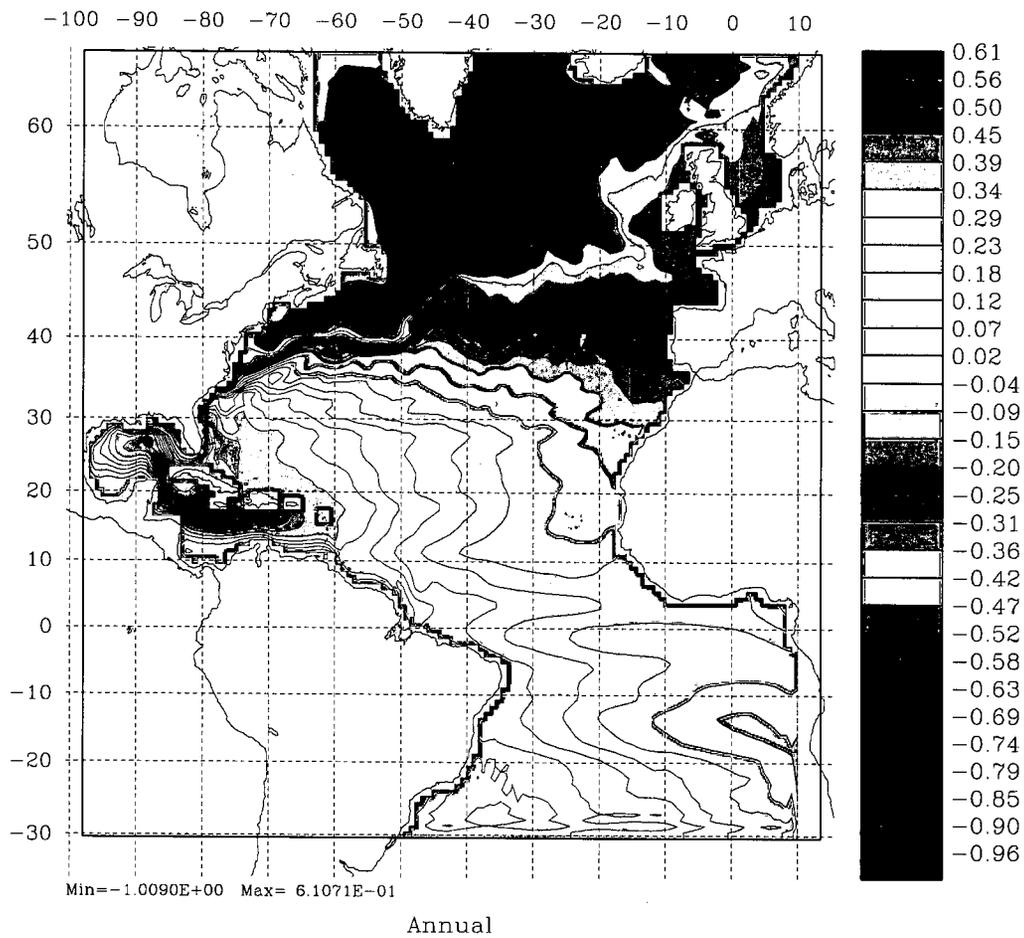
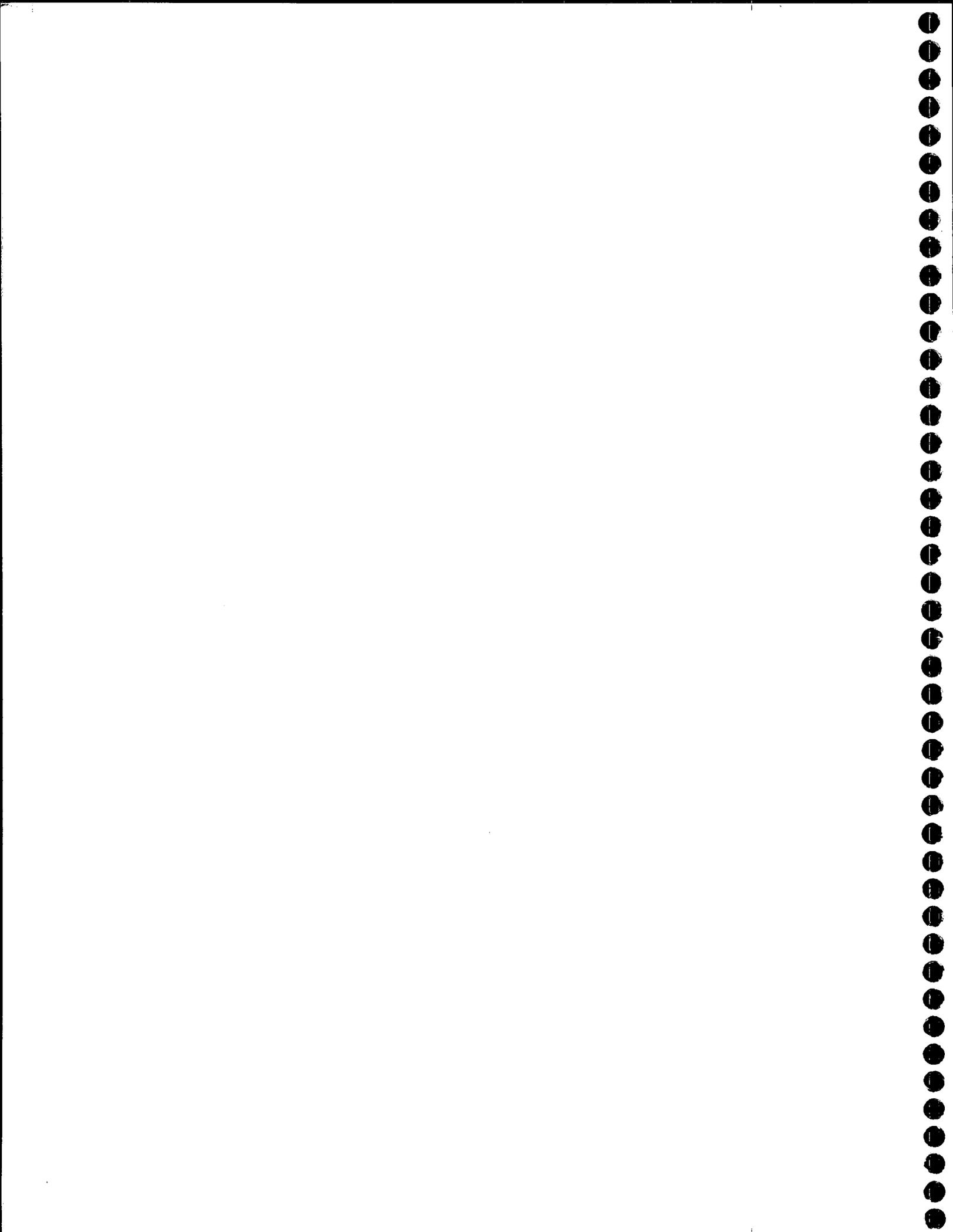


Figure 18: The annual mean surface elevation for year 10.



7 Plotting Programs for Postprocessing

Hernan Arango has provided SCRUM with some programs for creating plots from the NetCDF history and restart files. Some example plots are shown in §6. 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 an 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 an option of finding equal-spaced points along a straight line through the curvilinear grid.

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

```
1996 -1 : year and starting year-day (use yearday<0, for no time label)
SCRUM 3.0
Coarse Arctic ocean with Budgell ice dynamics
ice thermodynamics

8      NFIELDS: number of fields to plot. Line below, field(s) types:
42,45,46,47,48,49,50,121,122,123,124 field identification:
FLDID(1:NFIELDS)
1      NLEVELS: number of levels and/or depths to plot (0 for all levels)
20     levels (>0) or depths (<0) to plot: FLDLEV(1:NLEVELS)
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
0.0    PMIN : field minimum value for color palette (0.0 for default)
0.0    PMAX : field maximum value for color palette (0.0 for default)
1      ICNT : draw contours between color bands: 0=no, 1=yes
0.0    ISOVAL : iso-surface value to process (see below)
1.2    VLWD : vector line width (1.0 for default)
2.0    VLSCL : vector length scale (1.0 for default)
1      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)
10.0   LGRID : Desired longitude/latitude grid spacing (degrees)
4      IPROJ : map projection (see below).
-60.0  PLON : projection Pole longitude (west values are negative).
```

```

90.0  PLAT  : projection Pole latitude (south values are negative).
0.0   ROTA  : projection rotation angle (clockwise; degrees).
0     LMSK  : flag to color mask land: [0] no, [1] yes
-1    NPAGE : number of plots per page (currently 1, 2, or 4)
T     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.
T     CST    : logical switch to read and plot coastlines and islands.
50.0 50.0  : bottom and top map latitudes (south values are negative).
-110.0 80.0 : left and right map longitudes (west values are negative).
/d2/kate/plot/varid.dat
/d1/arango/scrum3.0/plot/Palettes/gs1.pal
/d2/kate/plot/default.cnt
ice_rst.nc
scrum_rst_1.nc
/d2/kate/arctic/gridpak/arctic_grid_2.nc
/u1/coasts/coast.dat

```

c

```

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

```

c

*** Above FILENAMES:

```

1st line: input; variables ID file.
2nd line: input; color palette file.
3rd line: input; contour parameters.
4th line: input; primary NetCDF file.
5th line: input; secondary NetCDF file.
6th line: input; grid NetCDF file.
7th 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)

```

*** IPROJ: Map Projections option. Some of the values for the projection Pole and rotation angle are suggested.

Check the NCAR manual for details.

- [1] Cylindrical equidistant: PLON=0, PLAT=0, ROTA=0
- [2] Mercator: PLON=?, PLAT=0, ROTA=0
- [3] Lambert conformal conic: PLON=?, PLAT=?, ROTA=?
- [4] Stereographic azimuthal: PLON=?, PLAT=90 or -90, ROTA=0

*** IVINC, JVINC: vector grid sampling. If either value is negative, the velocity vectors are drawn at PSI-points. Otherwise, if both values are positive, the vectors are drawn at interior RHO-points.

*** LGRID: Longitude/latitude grid spacing. The grid is drawn at LGRID spacing starting from specified map lower corner. If LGRID is negative, the latitude labels in the map are rotated 90 degrees to avoid label congestion, if any.

*** NPAGE: Number of plots per page. 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).
- :
- :

As you can see, there are comments describing what needs to be done. Please see the variable ID file for the complete list of fields which can be plotted—this list changes as Hernan adds the ability to plot new fields. Also, check your default.cnt file for other vector and contour parameters. The palette file includes two number systems, one in the scale from 0 to 255 and the other from 0 to 1. The SCRUM plotting uses the first while the SEOM plotting uses the second.

8 Ice Model Formulation

Hibler [22] has described a model for the simulation of sea ice circulation and thickness. The model has been tested over a seasonal cycle and the results are also shown in that article. This report was derived from the documentation for the sea ice model written by Hibler [23] since the dynamical equations are much the same. The model itself has been rewritten by Paul Budgell and is now implemented on an orthogonal, curvilinear Arakawa C-grid, has a new elliptic solver, and the nonlinear advection of momentum has been omitted. The thermodynamics are derived from Mellor and Kantha [35] (MK89 below). Sirpa Häkkinen allowed us to use her implementation of MK89; we obtained it from the Norwegians, who call it "hakkis".

8.1 Model structure

The overall structure consists of two principal components—the momentum equations and the ice continuity equations. The momentum balance includes air and water stress, Coriolis force, internal ice stress, inertial forces and ocean tilt as shown in equations (144) and (145):

$$M \frac{\partial u}{\partial t} = M f v - M g \frac{\partial \zeta_w}{\partial x} + \tau_a^x + \tau_w^x + \mathcal{F}_x \quad (144)$$

$$M \frac{\partial v}{\partial t} = -M f u - M g \frac{\partial \zeta_w}{\partial y} + \tau_a^y + \tau_w^y + \mathcal{F}_y. \quad (145)$$

The nonlinear advection terms have been omitted, since they are usually much smaller than the others. Nonlinear formulas are used for both the ocean-ice and air-ice surface stress:

$$\bar{\tau}_a = \rho_a C_a |\vec{V}_{10}| \vec{V}_{10} \quad (146)$$

$$C_a = \frac{1}{2} C_d [1 - \cos(2\pi \min(h_i + .1, .5))] \quad (147)$$

$$\bar{\tau}_w = \rho_w C_w |\vec{v}_w - \vec{v}| (\vec{v}_w - \vec{v}). \quad (148)$$

The symbols used in these equations along with the values for the constants are listed in Table 3.

A key component of the momentum balance is the force due to the internal ice stress (\mathcal{F}_x and \mathcal{F}_y). This force is based on a constitutive law which relates the ice stress to the strain rate and ice strength (equation (149)). For this model, a viscous-plastic behavior is used. Rigid plastic behavior is approximated in this law by allowing the ice to flow in a plastic manner for normal strain rates and to creep in a linear viscous manner for small strain rates. The treatment of ice as a viscous-plastic fluid was largely motivated by the desire to avoid the complexities associated with elastic-plastic behavior under flow. The stress-strain relationship is given by

$$\sigma_{ij} = 2\eta \epsilon_{ij} + (\zeta - \eta) \epsilon_{kk} \delta_{ij} - \frac{P}{2} \delta_{ij}. \quad (149)$$

The viscous-plastic terms (151) and (152) are found by taking the divergence of the stress tensor:

$$(\mathcal{F}_x, \mathcal{F}_y) = \nabla \cdot \sigma \quad (150)$$

with the result that

$$\mathcal{F}_x = \frac{\partial}{\partial x} \left[(\eta + \zeta) \frac{\partial u}{\partial x} + (\zeta - \eta) \frac{\partial v}{\partial y} - P/2 \right] + \frac{\partial}{\partial y} \left[\eta \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right) \right] \quad (151)$$

$$\mathcal{F}_y = \frac{\partial}{\partial y} \left[(\eta + \zeta) \frac{\partial v}{\partial y} + (\zeta - \eta) \frac{\partial u}{\partial x} - P/2 \right] + \frac{\partial}{\partial x} \left[\eta \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right) \right] \quad (152)$$

Variable	Value	Description
$A(x, y, t)$		ice concentration
$\alpha(A)$		ridging function
C_a		nonlinear air drag coefficient
C_d	2.2×10^{-3}	air drag coefficient
C_w	10×10^{-3}	water drag coefficient
$(\mathcal{D}_h, \mathcal{D}_s, \mathcal{D}_A)$		diffusion terms
$\epsilon_{ij}(x, y, t)$		strain rate tensor
$\eta(x, y, t)$		nonlinear shear viscosity
e	2	eccentricity of the elliptical yield curve
$(\mathcal{F}_x, \mathcal{F}_y)$		internal ice stress
$f(x, y)$		Coriolis parameter
g	9.8 m s^{-2}	acceleration of gravity
H		Heaviside function
$h_i(x, y, t)$		ice thickness of ice-covered fraction
h_o	1 m	ice cutoff thickness
$h_s(x, y, t)$		snow thickness on ice-covered fraction
$M(x, y, t)$		ice mass (density times thickness)
$P(x, y, t)$		ice pressure or strength
(P^*, C)	$(2.75 \times 10^4, 20)$	ice strength parameters
(S_h, S_s, S_A)		thermodynamic terms
$\sigma_{ij}(x, y, t)$		stress tensor
$\vec{\tau}_a$		air stress
$\vec{\tau}_w$		water stress
(u, v)		the (x, y) components of ice velocity \vec{v}
$(\vec{V}_{10}, \vec{v}_w)$		10 meter air and surface water velocities
(ρ_a, ρ_w)	$(1.3 \text{ kg m}^{-3}, 1025 \text{ kg m}^{-3})$	air and water densities
$\zeta(x, y, t)$		nonlinear bulk viscosity
$\zeta_w(x, y, t)$		height of the ocean surface

Table 3: Variables used in the ice momentum equations

where the nonlinear viscosities are given by

$$\zeta = \frac{P}{2 [(\epsilon_{11}^2 + \epsilon_{22}^2)(1 + 1/e^2) + 4e^{-2}\epsilon_{12}^2 + 2\epsilon_{11}\epsilon_{22}(1 - 1/e^2)]^{1/2}} \quad (153)$$

and

$$\eta = \frac{\zeta}{e^2}. \quad (154)$$

The "pressure gradient" term is also modeled as a term in the internal ice stress. This term represents the resistance which ice has to being compressed (ice strength) and is a function of ice thickness and concentration:

$$P = P^* Ah_i \exp[-C(1 - A)]H(-\nabla \cdot \vec{v}). \quad (155)$$

The Heaviside function guarantees that the ice has no strength when the flow is divergent (Gray and Killworth [16]).

The second major component of the model consists of continuity equations describing the evolution of the ice thickness characteristics. Three parameters are calculated: the ice thickness h_i , the snow thickness h_s , and the compactness, A , which is defined as the fraction of area covered by thick ice. Note that Hibler's h_I variable is equivalent to our Ah_i combination—his h_I is the average thickness over the whole gridbox while our h_i is the average thickness over the ice-covered fraction of the gridbox. The continuity equations describing the evolution of these parameters (equations (156)–(158)) also include thermodynamic terms (S_h , S_s and S_A), which will be described in §8.5:

$$\frac{\partial Ah_i}{\partial t} = -\frac{\partial(uAh_i)}{\partial x} - \frac{\partial(vAh_i)}{\partial y} + S_h + \mathcal{D}_h \quad (156)$$

$$\frac{\partial Ah_s}{\partial t} = -\frac{\partial(uAh_s)}{\partial x} - \frac{\partial(vAh_s)}{\partial y} + S_s + \mathcal{D}_s \quad (157)$$

$$\frac{\partial A}{\partial t} = -\frac{\partial(uA)}{\partial x} - \frac{\partial(vA)}{\partial y} + S_A + \mathcal{D}_A \quad 0 \leq A \leq 1. \quad (158)$$

The first two equations represent the conservation of ice and snow. Equation 158 is discussed in some detail in MK89, but represents the advection of ice blocks in which no ridging occurs as long as there is any open water. An optional ridging term can be added (Gray and Killworth [17]):

$$\frac{\partial A}{\partial t} = -\frac{\partial(uA)}{\partial x} - \frac{\partial(vA)}{\partial y} - A\alpha(A)\nabla \cdot \vec{v}H(-\nabla \cdot \vec{v}) + S_A + \mathcal{D}_A \quad 0 \leq A \leq 1. \quad (159)$$

where $\alpha(A)$ is an arbitrary function such that $\alpha(0) = 0$, $\alpha(1) = 1$, and $0 \leq \alpha(A) \leq 1$. The ridging term leads to an increase in h_i under convergent flow as would be produced by ridging. The function $\alpha(A)$ should be chosen so that it is near zero until the ice concentration is large enough that ridging is expected to occur, then should increase smoothly to one.

8.2 Horizontal curvilinear coordinates

Applying the curvilinear transformation used in §2.5 and described in Appendix C, we use a transformation to an orthogonal curvilinear coordinate system. Denoting the velocity components in the new coordinate system by

$$\vec{v} \cdot \hat{\xi} = u \quad (160)$$

and

$$\vec{v} \cdot \hat{\eta} = v \quad (161)$$

the equations of motion (144), (145), (156)–(158) can be re-written (see, e.g., Arakawa and Lamb [2]):

$$\frac{\partial u}{\partial t} = fv - gm \frac{\partial \zeta_w}{\partial \xi} + \frac{1}{M} (\tau_a^\xi + \tau_w^\xi + \mathcal{F}_\xi) \quad (162)$$

$$\frac{\partial v}{\partial t} = -fu - gn \frac{\partial \zeta_w}{\partial \eta} + \frac{1}{M} (\tau_a^\eta + \tau_w^\eta + \mathcal{F}_\eta) \quad (163)$$

$$\frac{\partial Ah_i}{\partial t} = -mn \left[\frac{\partial}{\partial \xi} \left(\frac{Ah_i u}{n} \right) + \frac{\partial}{\partial \eta} \left(\frac{Ah_i v}{m} \right) \right] + S_h \quad (164)$$

$$\frac{\partial Ah_s}{\partial t} = -mn \left[\frac{\partial}{\partial \xi} \left(\frac{Ah_s u}{n} \right) + \frac{\partial}{\partial \eta} \left(\frac{Ah_s v}{m} \right) \right] + S_s \quad (165)$$

$$\frac{\partial A}{\partial t} = -mn \left[\frac{\partial}{\partial \xi} \left(\frac{Au}{n} \right) + \frac{\partial}{\partial \eta} \left(\frac{Av}{m} \right) \right] + S_A \quad (166)$$

S_h , S_s and S_A remain unchanged.

The viscous-plastic terms can be derived from equation (149). In curvilinear coordinates the strain rate tensor can be written as:

$$\epsilon_{11} = m \frac{\partial u}{\partial \xi} + vmn \frac{\partial}{\partial \eta} \left(\frac{1}{m} \right) \quad (167)$$

$$\epsilon_{12} = e_{21} = \frac{1}{2} \left[\frac{m}{n} \frac{\partial (nv)}{\partial \xi} + \frac{n}{m} \frac{\partial (mu)}{\partial \eta} \right] \quad (168)$$

$$\epsilon_{22} = n \frac{\partial v}{\partial \eta} + umn \frac{\partial}{\partial \xi} \left(\frac{1}{n} \right) \quad (169)$$

In curvilinear coordinates the divergence of a symmetric tensor \mathbf{T} is:

$$\begin{aligned} \nabla \cdot \mathbf{T} = & \hat{\xi} \left[m \frac{\partial \mathbf{T}_{11}}{\partial \xi} + n \frac{\partial \mathbf{T}_{12}}{\partial \eta} + \mathbf{T}_{11} mn \frac{\partial}{\partial \xi} \left(\frac{1}{n} \right) \right. \\ & \left. + 2\mathbf{T}_{12} mn \frac{\partial}{\partial \eta} \left(\frac{1}{m} \right) - \mathbf{T}_{22} mn \frac{\partial}{\partial \xi} \left(\frac{1}{n} \right) \right] \\ & + \hat{\eta} \left[m \frac{\partial \mathbf{T}_{12}}{\partial \xi} + n \frac{\partial \mathbf{T}_{22}}{\partial \eta} - \mathbf{T}_{11} mn \frac{\partial}{\partial \eta} \left(\frac{1}{m} \right) \right. \\ & \left. + 2\mathbf{T}_{12} mn \frac{\partial}{\partial \xi} \left(\frac{1}{n} \right) + \mathbf{T}_{22} mn \frac{\partial}{\partial \eta} \left(\frac{1}{m} \right) \right] \quad (170) \end{aligned}$$

A more general expression for derivatives of tensors is given by Aris [3] in terms of Christoffel symbols.

The viscous-plastic terms become:

$$\begin{aligned}
\mathcal{F}_\xi = & m \frac{\partial}{\partial \xi} \left[(\zeta - \eta) m n \frac{\partial}{\partial \xi} \left(\frac{u}{n} \right) \right] + m \frac{\partial}{\partial \xi} \left[(\zeta - \eta) m n \frac{\partial}{\partial \eta} \left(\frac{v}{m} \right) \right] - \frac{m}{2} \frac{\partial P}{\partial \xi} \\
& + 2m \frac{\partial}{\partial \xi} \left[\eta m \frac{\partial u}{\partial \xi} + \eta v m n \frac{\partial}{\partial \eta} \left(\frac{1}{m} \right) \right] + n \frac{\partial}{\partial \eta} \left[\eta \frac{m}{n} \frac{\partial (nv)}{\partial \xi} + \eta \frac{n}{m} \frac{\partial (mu)}{\partial \eta} \right] \\
& + 2\eta m^2 n \frac{\partial u}{\partial \xi} \frac{\partial}{\partial \xi} \left(\frac{1}{n} \right) + 2\eta v m^2 n^2 \frac{\partial}{\partial \eta} \left(\frac{1}{m} \right) \frac{\partial}{\partial \xi} \left(\frac{1}{n} \right) + 2\eta m^2 \frac{\partial (nv)}{\partial \xi} \frac{\partial}{\partial \eta} \left(\frac{1}{m} \right) \\
& + 2\eta n^2 \frac{\partial (mu)}{\partial \eta} \frac{\partial}{\partial \eta} \left(\frac{1}{m} \right) - 2\eta m n^2 \frac{\partial v}{\partial \eta} \frac{\partial}{\partial \xi} \left(\frac{1}{n} \right) - 2\eta u m^2 n^2 \left[\frac{\partial}{\partial \xi} \left(\frac{1}{n} \right) \right]^2
\end{aligned} \tag{171}$$

$$\begin{aligned}
\mathcal{F}_\eta = & n \frac{\partial}{\partial \eta} \left[(\zeta - \eta) m n \frac{\partial}{\partial \xi} \left(\frac{u}{n} \right) \right] + n \frac{\partial}{\partial \eta} \left[(\zeta - \eta) m n \frac{\partial}{\partial \eta} \left(\frac{v}{m} \right) \right] - \frac{n}{2} \frac{\partial P}{\partial \eta} \\
& + m \frac{\partial}{\partial \xi} \left[\eta \frac{m}{n} \frac{\partial (nv)}{\partial \xi} + \eta \frac{n}{m} \frac{\partial (mu)}{\partial \eta} \right] + 2n \frac{\partial}{\partial \eta} \left[\eta n \frac{\partial v}{\partial \eta} + \eta u m n \frac{\partial}{\partial \xi} \left(\frac{1}{n} \right) \right] \\
& - 2\eta m^2 n \frac{\partial u}{\partial \xi} \frac{\partial}{\partial \eta} \left(\frac{1}{m} \right) - 2\eta v m^2 n^2 \left[\frac{\partial}{\partial \eta} \left(\frac{1}{m} \right) \right]^2 + 2\eta m^2 \frac{\partial (nv)}{\partial \xi} \frac{\partial}{\partial \xi} \left(\frac{1}{n} \right) \\
& + 2\eta n^2 \frac{\partial (mu)}{\partial \eta} \frac{\partial}{\partial \xi} \left(\frac{1}{n} \right) + 2\eta m n^2 \frac{\partial v}{\partial \eta} \frac{\partial}{\partial \eta} \left(\frac{1}{m} \right) + 2\eta u m^2 n^2 \frac{\partial}{\partial \eta} \left(\frac{1}{m} \right) \frac{\partial}{\partial \xi} \left(\frac{1}{n} \right)
\end{aligned} \tag{172}$$

8.3 Numerical Scheme

The coupled nonlinear equations of the model are treated as an initial value problem using energy-conserving finite difference techniques. Under this initial value approach, values of all parameters at all grid points are required to start the integration, and boundary values of velocity are required thereafter. The forcing fields for the model consist of wind stress fields as well as surface winds for the thermodynamic quantities, air temperature, dew point temperature, cloud cover, and snow precipitation. It also needs near-surface ocean currents and sea-surface elevation for computing the ocean stress term and the surface tilt term. The inertial terms are retained in the dynamical equations, while the nonlinear advection is assumed to be small enough to neglect. Because of the very strong ice interaction term, explicit integration of the momentum equations would force timesteps to be extremely small (of the order of a second or less). The ice thickness equations, on the other hand, can be explicitly integrated over much longer timesteps. To avoid this severe timestep limitation due to the ice interaction, the momentum equations are integrated implicitly. This implicit integration does, however, require a relaxation solution of a set of simultaneous equations at each timestep. It is currently implemented using calls to either the **ESSL** library or the **NSPCG** library. We are using **NSPCG** because it is freely available.

Using this implicit scheme, the inertial terms will normally be significant for short timesteps (of the order of 1 hour or less) but insignificant for long timesteps. Although not a formal stability requirement, it is wise to choose timesteps that are small compared to the variability of the ice forcing. The effect of different timestep magnitudes on the momentum balance is discussed in

Appendix B of Hibler [22]. This appendix also contains some examples which may help the user decide on timestep magnitudes for particular applications. Because of this implicit treatment of the momentum equation, the only formal stability requirement is a Courant-Friedrichs-Lewy condition for the advection terms in the thickness equations:

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

A key feature of the numerical scheme is a staggered grid, known as the "Arakawa C grid", where the velocity is defined on the sides of a grid box and variables such as thickness and viscosity are defined at the center of the grid box, as shown in Fig. 2.

Because of the strong ice interaction (which in this model is dissipative in nature) the momentum equations are essentially parabolic in form and hence have few numerical instability problems over long-term integrations. (While there are few numerical problems, it should be emphasized that the dissipative ice interaction terms are highly nonlinear and can lead to unstable flow fields in the absence of water drag. Such a feature is a physical characteristic of plastic flow and not a numerical artifact). However, in principle it is possible for the ice interaction to be very small even though there may be a finite ice mass. Under this situation, the momentum advection terms could cause nonlinear instabilities. To ensure against such situations, Hibler put a lower limit on the bulk viscosity parameter (and hence indirectly shear viscosity as well). It is never allowed to drop below 1.0×10^7 kg/s, a value which negligibly modifies the ice drift.

Nonlinear instabilities over long-term integrations can also arise from the nonlinear advection terms in the thickness continuity equations. To avoid this problem, the advection terms in equations (156)–(158) can be computed with a choice of either Smolarkiewicz's MPDATA ([53]) or third-order upwind (Leonard [30]). See §3.7 for a description of these schemes. It may be possible to omit the diffusivity from these equations if the forcing fields are sufficiently smooth.

8.4 Horizontal boundary conditions

As mentioned above, initial conditions at all points and ice velocities at the boundaries are thereafter required to initiate the integration of the system of equations forward in time. The most natural boundary condition is to take the ice velocity to be zero on the boundaries. This can be done either at a land boundary or at an ocean location where there is no ice. Note that the boundary condition does not affect the ice motion in such circumstances since in the absence of ice the strength is zero. More generally, as long as the ocean boundaries are removed from the ice edge, the coupled nature of the model will cause a natural ice edge boundary condition to be created. However, it is also possible to form an "open" boundary condition by setting the strength equal to zero near a boundary. These gridboxes are called "outflow cells" in Hibler [22]. In the Arctic simulations, these outflow cells are used at the open edge near Greenland.

The boundary conditions on the momentum equations are to set u and v to zero at all boundaries, including islands. This is accomplished by the elliptic solver during the implicit timestep.

The ice and snow thickness and ice concentration equations have no-flux boundary conditions imposed along the mask boundaries. The outflow cells contain a radiation condition if the velocity is outward and no change if the velocity is inward.

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

8.5 Thermodynamics

The thermodynamics used is based on the algorithm described in Mellor and Kantha [35], who have a useful description of the various melting and freezing processes, plus the coupling to a full

three-dimensional ocean. Their form of equations (156) and (158) is:

$$\frac{\partial Ah_i}{\partial t} + \frac{\partial(uAh_i)}{\partial x} + \frac{\partial(vAh_i)}{\partial y} = \frac{\rho_o}{\rho_i} [A(W_{io} - W_{ai}) + (1 - A)W_{ao} + W_{fr}] \quad (173)$$

$$\frac{\partial A}{\partial t} + \frac{\partial(uA)}{\partial x} + \frac{\partial(vA)}{\partial y} = \frac{\rho_o A}{\rho_i h_i} [\Phi(1 - A)W_{ao} + (1 - A)W_{fr}] \quad 0 \leq A \leq 1. \quad (174)$$

Here, the W variables are the freeze or melt rates as shown in Fig. 19 and Table 4. The frazil ice growth W_{fr} will be discussed further in §8.5.2—note that it contributes to changes in A as well as to changes in h_i . The other term that contributes to A is W_{ao} . This term includes a factor Φ which Mellor and Kantha set to different values depending on whether ice is melting or freezing:

$$\Phi = 4.0 \quad W_{ao} \geq 0 \quad (175)$$

$$\Phi = 0.5 \quad W_{ao} < 0 \quad (176)$$

$$(177)$$

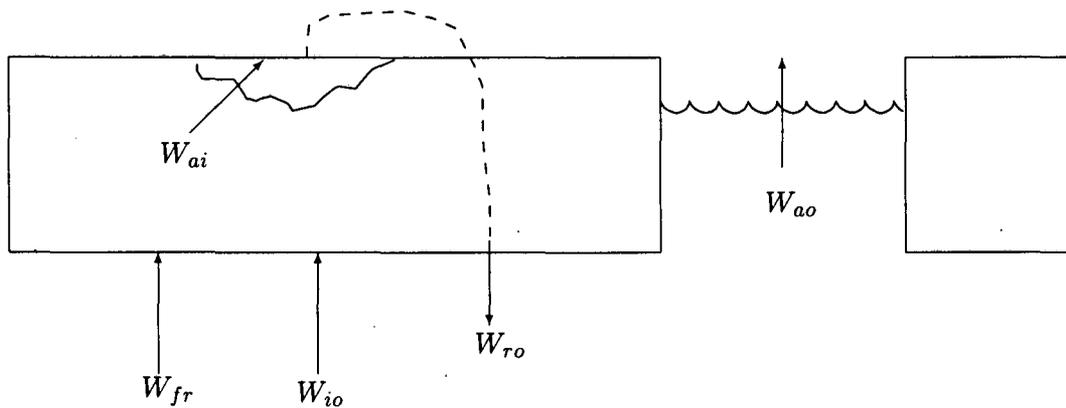


Figure 19: Diagram of the different locations where ice melting and freezing can occur.

Figure 20 shows the locations of the ice and snow temperatures and the heat fluxes. The temperature profile is assumed to be linear between adjacent temperature points. The interior of the ice contains "brine pockets", leading to a prognostic equation for the temperature T_1 .

The surface flux to the air is:

$$Q_{ai} = -H\downarrow - LE\downarrow - \epsilon_s LW\downarrow - (1 - \alpha_s)SW\downarrow + \epsilon_s \sigma (T_3 + 273)^4 \quad (178)$$

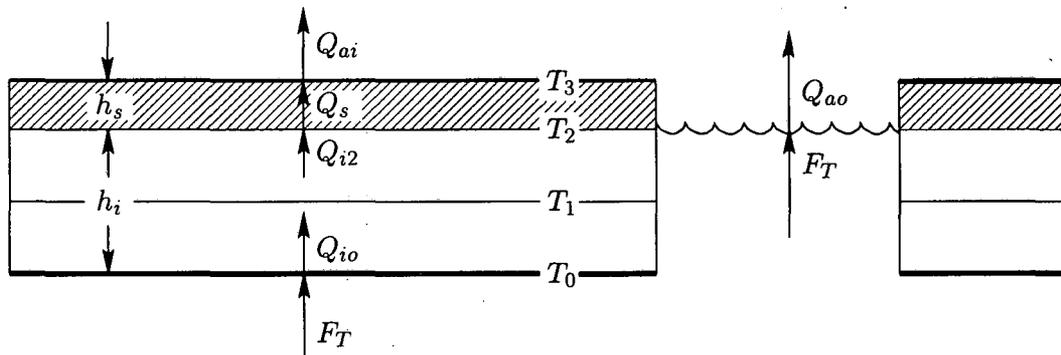


Figure 20: Diagram of internal ice temperatures and fluxes. The hashed layer is the snow.

Variable	Value	Description
α_w	0.10	shortwave albedo of water
α_i	0.50	shortwave albedo of ice
α_s	0.75	shortwave albedo of snow
C_k		snow correction factor
C_{pi}	2093 J kg ⁻¹ K ⁻¹	specific heat of ice
C_{po}	3990 J kg ⁻¹ K ⁻¹	specific heat of water
ϵ_w	0.97	longwave emissivity of water
ϵ_i	0.97	longwave emissivity of ice
ϵ_s	0.99	longwave emissivity of snow
$E(T, r)$		enthalpy of the ice/brine system
$F_T \uparrow$		heat flux from the ocean into the ice
$H \downarrow$		sensible heat
i_w		fraction of the solar heating transmitted through a lead into the water below
k_i	2.04 W m ⁻¹ K ⁻¹	thermal conductivity of ice
k_s	0.31 W m ⁻¹ K ⁻¹	thermal conductivity of snow
L_i	302 MJ m ⁻³	latent heat of fusion of ice
L_s	110 MJ m ⁻³	latent heat of fusion of snow
$LE \downarrow$		latent heat
$LW \downarrow$		incoming longwave radiation
m	-0.054°C/PSU	coefficient in linear $T_f(S) = mS$ equation
Φ		contribution to A equation from freezing water
Q_{ai}		heat flux out of the snow/ice surface
Q_{ao}		heat flux out of the ocean surface
Q_{i2}		heat flux up out of the ice
Q_{io}		heat flux up into the ice
Q_s		heat flux up through the snow
r		brine fraction in ice
ρ_i	910 m ³ /kg	density of ice
S_i	5 PSU	salinity of the ice
$SW \downarrow$		incoming shortwave radiation
σ	5.67×10^{-8} W m ⁻² K ⁻⁴	Stefan-Boltzmann constant
T_0		temperature of the bottom of the ice
T_1		temperature of the interior of the ice
T_2		temperature at the upper surface of the ice
T_3		temperature at the upper surface of the snow
T_f		freezing temperature
$T_{melt.i}$	mS_i	melting temperature of ice
$T_{melt.s}$	0° C	melting temperature of snow
W_{ai}		melt rate on the upper ice/snow surface
W_{ao}		freeze rate at the air/water interface
W_{fr}		rate of frazil ice growth
W_{io}		freeze rate at the ice/water interface
W_{ro}	W_{ai}	rate of run-off of surface melt water

Table 4: Variables used in the ice thermodynamics

The formulas for sensible heat, latent heat, and incoming longwave and shortwave radiations are the same as in Parkinson and Washington [41] and are shown in Appendix E. The sensible heat is a function of T_3 , as is the heat flux through the snow Q_s . Setting $Q_{ai} = Q_s$, we can solve for T_3 by setting $T_3^{n+1} = T_3^n + \Delta T_3$ and linearizing in ΔT_3 . The temperature T_3 is found by an iterative solution of the surface heat flux balance (using the previous value of T_1 in equation 186). As in Parkinson and Washington, if T_3 is found to be above the melting temperature, it is set to T_{melt} and the extra energy goes into melting the snow or ice:

$$W_{ai} = \frac{Q_{ai} - Q_{i2}}{\rho_o L_3} \quad (179)$$

$$L_3 \equiv [E(T_3, 1) - E(T_1, R_1)] \quad (180)$$

Note that $L_3 = (1 - r)L_i$ plus a small sensible heat correction. We are not storing water on the surface in melt pools, so everything melted at the surface is assumed to flow into the ocean ($W_{ro} = W_{ai}$).

Inside the ice there are brine pockets in which there is salt water at the *in situ* freezing temperature. It is assumed that the ice has a uniform overall salinity of S_i and that the freezing temperature is a linear function of salinity. The brine fraction r is given by

$$r = \frac{S_i m}{T_1}$$

The enthalpy of the combined ice/brine system is given by

$$E(T, r) = r(L_i + C_{po}T) + (1 - r)C_{pi}T \quad (181)$$

Substituting in for r and differentiating gives:

$$\frac{\partial E}{\partial T} = -\frac{S_i m L_i}{T_1^2} + C_{pi} \quad (182)$$

Inside the snow, we have

$$Q_s = \frac{k_s}{h_s}(T_2 - T_3) \quad (183)$$

The heat conduction in the upper part of the ice layer is

$$Q_{I2} = \frac{2k_i}{h_i}(T_1 - T_2) \quad (184)$$

These can be set equal to each other to solve for T_2

$$T_2 = \frac{T_3 + C_k T_1}{1 + C_k} \quad (185)$$

where

$$C_k \equiv \frac{2k_i h_s}{h_i k_s}$$

Substituting into (184), we get:

$$Q_s = Q_{I2} = \frac{2k_i}{h_i} \frac{(T_1 - T_3)}{(1 + C_k)} \quad (186)$$

Note that in the absence of snow, C_k becomes zero and we recover the formula for the no-snow case in which $T_3 = T_2$.

Variable	Value	Definition
b	3.0	factor
\dot{E}		evaporation
k	0.4	von Karman's constant
ν	$1.8 \times 10^{-6} m^2 s^{-1}$	kinematic viscosity of seawater
\dot{P}		precipitation
Pr	13.0	molecular Prandtl number
Pr_t	0.85	turbulent Prandtl number
S_0		surface salinity
τ_{io}		stress on the ocean from the ice
τ_{ao}		stress on the ocean from the wind
T		internal ocean temperature
u_τ		friction velocity $ \tau_{io} ^{1/2} \rho_o^{-1/2}$
z_0		roughness parameter

Table 5: Ocean surface variables

At the bottom of the ice, we have

$$Q_{I0} = \frac{2k_i}{h_i} (T_0 - T_1) \quad (187)$$

The difference between Q_{I0} and Q_{I2} goes into the enthalpy of the ice:

$$\rho_i h_i \left[\frac{\partial E}{\partial t} + \vec{v} \cdot \nabla E \right] = Q_{I0} - Q_{I2} \quad (188)$$

We can use the chain rule to obtain an equation for timestepping T_1 :

$$\rho_i h_i \frac{\partial E}{\partial T} \left[\frac{\partial T_1}{\partial t} + \vec{v} \cdot \nabla T_1 \right] = Q_{I0} - Q_{I2} \quad (189)$$

where

$$\begin{aligned} Q_{I0} - Q_{I2} &= \frac{2k_i}{h_i} \left[(T_0 - T_1) - \frac{(T_1 - T_3)}{1 + C_k} \right] \\ &= \frac{2k_i}{h_i} \left[T_0 + \frac{T_3 - (2 + C_k)T_1}{1 + C_k} \right] \end{aligned}$$

8.5.1 Ocean surface boundary conditions

The ocean receives surface stresses from both the atmosphere and the ice, according to the ice concentration:

$$K_m \frac{\partial u_w}{\partial z} = \frac{A}{\rho_o} \tau_{io}^x + \frac{1 - A}{\rho_o} \tau_{ao}^x \quad (190)$$

$$K_m \frac{\partial v_w}{\partial z} = \frac{A}{\rho_o} \tau_{io}^y + \frac{1 - A}{\rho_o} \tau_{ao}^y \quad (191)$$

where the relevant variables are in table 5.

The surface ocean is assumed to be at the freezing temperature for the surface salinity ($T_0 = mS$) where we use the salinity from the uppermost model point at $z = -\frac{1}{2}\Delta z$. From this, we can obtain a vertical temperature gradient for the upper ocean to use in the heat flux formula:

$$\frac{F_T}{\rho_o C_{p0}} = -C_{Tz} (T_0 - T) \quad z \rightarrow 0 \quad (192)$$

where

$$C_{T_z} = \frac{u_\tau}{P_{rt} k^{-1} \ln(-z/z_0) + B_T} \quad (193)$$

$$B_T = b \left(\frac{z_0 u_\tau}{\nu} \right)^{1/2} P_{rt}^{2/3} \quad (194)$$

Once we have a the value for F_T , we can use it to find the ice growth rates:

$$W_{io} = \frac{1}{\rho_o L_o} (Q_{io} - F_T) \quad (195)$$

$$W_{ao} = \frac{1}{\rho_o L_o} (Q_{ao} - F_T) \quad (196)$$

$$(197)$$

where

$$L_o \equiv [E(T_0, 1) - E(T_1, r_1)] \quad (198)$$

The ocean model receives the following heat and salt fluxes:

$$F_T = A Q_{io} + (1 - A) Q_{ao} - W_o L_o \quad (199)$$

$$F_S = (W_o - A W_{ro})(S_i - S_o) + (1 - A) S_o (\dot{P} - \dot{E}) W_o \quad \equiv A W_{io} + (1 - A) W_{ao} \quad (200)$$

8.5.2 Frazil ice formation

Following Steele et al. [55], we check to see if any of the ocean temperatures are below freezing at the end of each timestep. If so, frazil ice is assumed to form, changing the local temperature and salinity. The ice that forms is assumed to instantly float up to the surface and add to the ice layer there. We assume balances in the mass, heat, and salt before and after the ice is formed:

$$m_{w_1} = m_{w_2} + m_i \quad (201)$$

$$m_{w_1} (C_{pw} T_1 + L) = m_{w_2} (C_{pw} T_2 + L) + m_i C_{pi} T_2 \quad (202)$$

$$m_{w_1} S_1 = m_{w_2} S_2. \quad (203)$$

The variables are defined in Table 6. Defining $\gamma = m_i/m_{w_2}$ and dropping terms of order γ^2 leads to:

$$T_2 = T_1 + \gamma \left[\frac{L}{C_{pw}} + T_1 \left(1 - \frac{C_{pi}}{C_{pw}} \right) \right] \quad (204)$$

$$S_2 = S_1 (1 + \gamma). \quad (205)$$

We also want the final temperature and salinity to be on the freezing line, which we approximate as:

$$T_f = mS + nz. \quad (206)$$

We can then solve for γ :

$$\gamma = \frac{-T_1 + mS_1 + nz}{\frac{L}{C_{pw}} + T_1 \left(1 - \frac{C_{pi}}{C_{pw}} \right) - mS_1}. \quad (207)$$

The ocean is checked at each depth k and at each timestep for supercooling. If the water is below freezing, the temperature and salinity are adjusted as in equations (204) and (205) and the ice above is thickened by the amount:

$$\Delta h = \gamma_k \Delta z_k \frac{\rho_w}{\rho_i}. \quad (208)$$

Variable	Value	Definition
C_{pi}	$1994 \text{ J kg}^{-1} \text{ K}^{-1}$	specific heat of ice
C_{pw}	$3987 \text{ J kg}^{-1} \text{ K}^{-1}$	specific heat of water
γ	m_i/m_{w2}	fraction of water that froze
L	$3.16e5 \text{ J kg}^{-1}$	latent heat of fusion
m_i		mass of ice formed
m_{w1}		mass of water before freezing
m_{w2}		mass of water after freezing
m	-0.0543	constant in freezing equation
n	7.59×10^{-4}	constant in freezing equation
S_1		salinity before freezing
S_2		salinity after freezing
T_1		temperature before freezing
T_2		temperature after freezing

Table 6: Frazil ice variables

8.5.3 Differences from Mellor and Kantha

We have tried to modify the **hakkis** model to more closely follow MK89. However, there are also ways in which we have deviated from it.

- Add advection of snow.
- Add lateral melting of snow when ice is melting laterally.
- We iterate on the solution of T_3 .
- We took a shortcut in the solution of S_0, T_0 for the surface heat and salt fluxes. We also apply them differently to the ocean model.
- We added various limiters:
 - Ice concentration: $A \geq A_{\min}, A_{\min} = 0.02$.
 - Ice thickness: $h_i \geq h_{\min}, h_{\min} = 0.1$.
 - Brine fraction: $r \leq r_{\max}, r_{\max} = 0.2$

9 Description of the Ice Model and the Coupling

9.1 Ice model structure

The flow of the main program for the ice model is shown in Fig. 21. The overall structure essentially consists of two components—the momentum equations and the ice continuity equations. The momentum balance includes air and water stresses, Coriolis force, internal ice stress, inertial forces and ocean tilt (equations (144) and (145)). There is a choice of rheologies (form of internal ice stress) including viscous-plastic (used here), free drift, Mohr-Coulomb, and cavitating fluid. A semi-implicit timestep is done on the momentum equations which are solved by calling a solver from the **NSPCG** library. The ice viscosities are non-linear functions of the velocity, so the solution is iterated several times, with the viscosities being recomputed each iteration.

The ice continuity consists of advection (done in **ice_mpdata** or **ice_advect**), and thermodynamics (**run_mk** and **ice_mk**). We have also added the diffusion of the ice tracer variables in order to obtain a smoother solution. This may be optional if the forcing fields are sufficiently smooth.

The main program calls **ice_step**, which in turn calls:

ice_advect Compute the ice advection according to a third-order upwind advection scheme. The ice ridging and ice diffusion also happen here if they are enabled.

ice_bctrans Boundary conditions for A , h_i and h_s .

ice_bcuv Boundary conditions for ice u and v .

ice_cavitating Compute the ice viscosities according to a cavitating fluid rheology.

ice_freedrift Compute the ice parameters to model free drift (no internal ice stress contribution).

ice_gencoef Generate coefficients for the iterative solution to the ice momentum equations.

ice_mohrcoulomb Compute the ice viscosities according to a Mohr-Coulomb fluid rheology.

ice_mpdata Compute the advection of a scalar according to a Smolarkiewicz advection scheme.

ice_rhs Gather the right-hand-side terms for the ice momentum equations prior to using the solver.

ice_solver_NSPCG Solve the implicit ice momentum equations using the **NSPCG** library.

run_mk Compute the changes in ice thickness and concentration due to the thermodynamics. Also produce the surface heat and salt forcing for the ocean model.

ice_viscplast Compute the ice viscosities according to a viscous-plastic fluid rheology.

9.1.1 Thermodynamic subroutines

The thermodynamic subroutines used in the model are:

ice_mk Compute the net energy budget and change of thickness for ice and snow.

ice_frazil Compute the frazil ice growth due to supercooled water in the ocean (called from **main3d** after the ocean timestep).

rads Compute the heat flux budgets over the water.

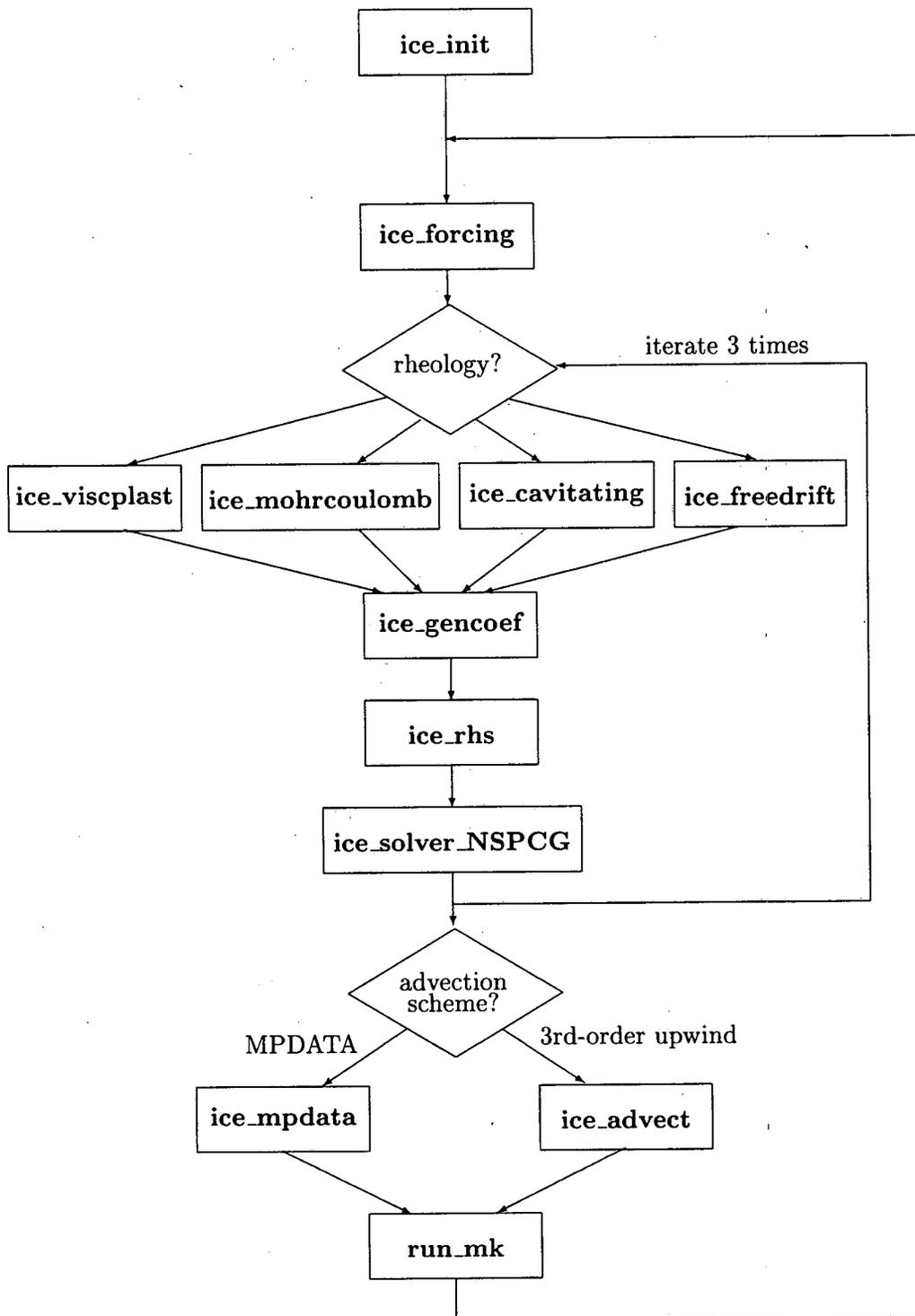


Figure 21: Flow chart for the sea-ice model.

9.1.2 Initialization

During initialization, the following routines are called:

def_ice_avg Create the netCDF ice averages file.

def_ice_his Create the netCDF ice history file.

def_ice_rst Create the netCDF ice restart file.

freeze Make sure that no water temperatures are below freezing during initialization. It does not form frazil ice.

get_ice Read a record from the netCDF ice restart file.

hakkblkdat Block data initializing some parameters for the ice thermodynamics.

ice_blkdat Block data initializing some parameters for the Smolarkiewicz advection routine.

ice_init Initialize the ice model, either by reading a history file or by setting the initial values.

ice_user1 Initialize some ice variables.

init_hakkis Initialize some ice thermodynamic variables.

init_ice Initialize some more ice variables.

9.1.3 Forcing fields

The ice model requires some extra forcing fields. These fields require their own routines for reading them from the netCDF forcing files:

get_airp Reads surface air pressure from the forcing NetCDF file, and then linearly time-interpolates to current model time.

get_airt Reads surface air temperature from the forcing NetCDF file, and then linearly time-interpolates to current model time.

get_cloud Reads cloud fraction from the forcing NetCDF file, and then linearly time-interpolates to current model time.

get_dewt Reads surface dewpoint temperature from the forcing NetCDF file, and then linearly time-interpolates to current model time.

get_lrflux Reads incoming longwave radiation from the forcing NetCDF file, and then linearly time-interpolates to current model time.

get_precip Reads precipitation from the forcing NetCDF file, and then linearly time-interpolates to current model time.

get_wind Reads surface winds from the forcing NetCDF file, and then linearly time-interpolates to current model time.

9.1.4 Other subroutines

The other subroutines in the ice model include:

ice_forcing Computes the forcing fields due to exchange of momentum by the ice and ocean.

smooth Smooths a field with a five-point Laplacian filter.

wrt_ice_avg Writes a record to the netCDF ice averages file.

wrt_ice_his Writes a record to the netCDF ice history file.

wrt_ice_rst Writes a record to the netCDF ice restart file.

9.2 Coupling strategy

The flow chart for the coupled ice-ocean model is shown in Fig. 22. The ice model is stepped first since it provides the surface heat and salt fluxes to the ocean model. After the ocean model is timestepped, the ocean temperatures are checked for supercooled water—if any is found it is converted into frazil ice and the ocean temperature and salinity are adjusted to be at freezing. The ocean equation of state is computed after this conversion.

9.3 C preprocessor variables

The ice model has two C preprocessor variables in **cppdefs.h**:

ICE Define to use ice component of the model.

ICE_THERMO Define for ice thermodynamics.

The ice model requires new forcing fields to be read or computed:

ANA_AIRT Define for an analytic air temperature.

ANA_CLOUD Define for an analytic cloud fraction.

ANA_DEWT Define for an analytic dew-point temperature.

ANA_LRFLUX Define for an analytic incoming longwave radiation.

ANA_SNOW Define for an analytic snow fall rate.

ANA_SLP Define for an analytic sea-level pressure.

ANA_WIND Define for analytic wind fields.

There are also some C preprocessor variables in **icedefs.h**:

NSPCG Use the **nspcg** library for the implicit solver. Either this or **ESSL** must be turned on.

ESSL Use the **essl** library for the implicit solver.

ANA_ICE_INIT Initialize the ice fields analytically as opposed to reading them from a file.

ICE_MOMENTUM Compute the ice momentum equations.

ICE_ADVECT Advect the ice tracers.

ice_GSCHEME Use a third-order upwind advection scheme for the tracers. This must be defined for either of these to take effect:

ICE_DIFFUSION Add a Laplacian diffusion on the ice tracers.

ICE_RIDGING Add an ice ridging scheme from Gray and Killworth [16].

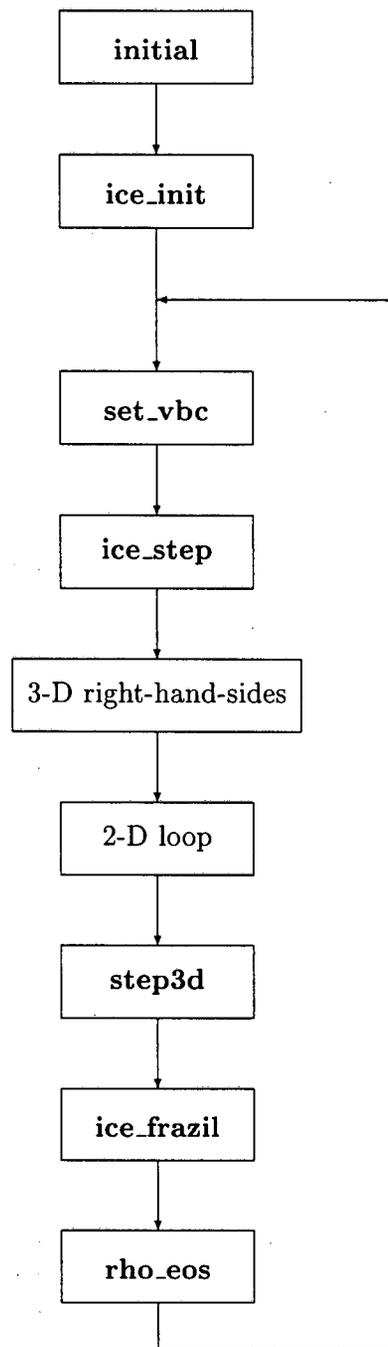


Figure 22: Flow chart for the coupled ice-ocean model.

A Model Timestep

Numerical timestepping uses a discrete approximation to:

$$\frac{\partial \phi(t)}{\partial t} = \mathcal{F}(t) \quad (209)$$

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:

$$\frac{\phi(t + \Delta t) - \phi(t)}{\Delta t} = \mathcal{F}(t) \quad (210)$$

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 $O(\Delta t^2)$:

$$\frac{\phi(t + \Delta t) - \phi(t - \Delta t)}{2\Delta t} = \mathcal{F}(t). \quad (211)$$

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 $\mathcal{F}^*(t + \Delta t)$:

$$\frac{\phi(t + \Delta t) - \phi(t)}{\Delta t} = \frac{1}{2} [\mathcal{F}(t) + \mathcal{F}^*(t + \Delta t)]. \quad (212)$$

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:

$$\frac{\phi(t + \Delta t) - \phi(t)}{\Delta t} = \alpha \mathcal{F}(t) + \beta \mathcal{F}(t - \Delta t) + \gamma \mathcal{F}(t - 2\Delta t) \quad (213)$$

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:

$$\frac{\phi(t + \Delta t) - \phi(t)}{\Delta t} = \phi' + \frac{\Delta t}{2} \phi'' + \frac{\Delta t^2}{6} \phi''' + \dots \quad (214)$$

where

$$\mathcal{F}(t) = \phi' \quad (215)$$

$$\mathcal{F}(t - \Delta t) = \phi' - \Delta t \phi'' + \frac{\Delta t^2}{2} \phi''' + \dots \quad (216)$$

$$\mathcal{F}(t - 2\Delta t) = \phi' - 2\Delta t \phi'' + 2\Delta t^2 \phi''' + \dots \quad (217)$$

We find that the coefficients are:

$$\alpha = \frac{23}{12} \quad (218)$$

$$\beta = -\frac{4}{3} \quad (219)$$

$$\gamma = \frac{5}{12} \quad (220)$$

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.

B The vertical s -coordinate

Following Song and Haidvogel [54], the vertical coordinate has been chosen to be:

$$z = \zeta(1 + s) + h_c s + (h - h_c)C(s), \quad -1 \leq s \leq 0 \quad (221)$$

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:

$$C(s) = (1 - b) \frac{\sinh(\theta s)}{\sinh \theta} + b \frac{\tanh[\theta(s + \frac{1}{2})] - \tanh(\frac{1}{2}\theta)}{2 \tanh(\frac{1}{2}\theta)} \quad (222)$$

where θ and b are surface and bottom control parameters. Their ranges are $0 < \theta \leq 20$ and $0 \leq b \leq 1$, respectively. Equation (221) 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:

$$z = (\zeta + h)(1 + s) - h \quad (223)$$

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 $\theta \leq 5$.
- Some problems turn out to be sensitive to the value of θ used.

Figure 23 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:

$$H_z \equiv \frac{\partial z}{\partial s} = (\zeta + h) + (h - h_c) \frac{\partial C(s)}{\partial s}. \quad (224)$$

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

$$\frac{\partial C(s)}{\partial s} = (1 - b) \frac{\cosh(\theta s)}{\sinh \theta} \theta + b \frac{\coth(\frac{1}{2}\theta)}{2 \cosh^2[\theta(s + \frac{1}{2})]} \theta. \quad (225)$$

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 .

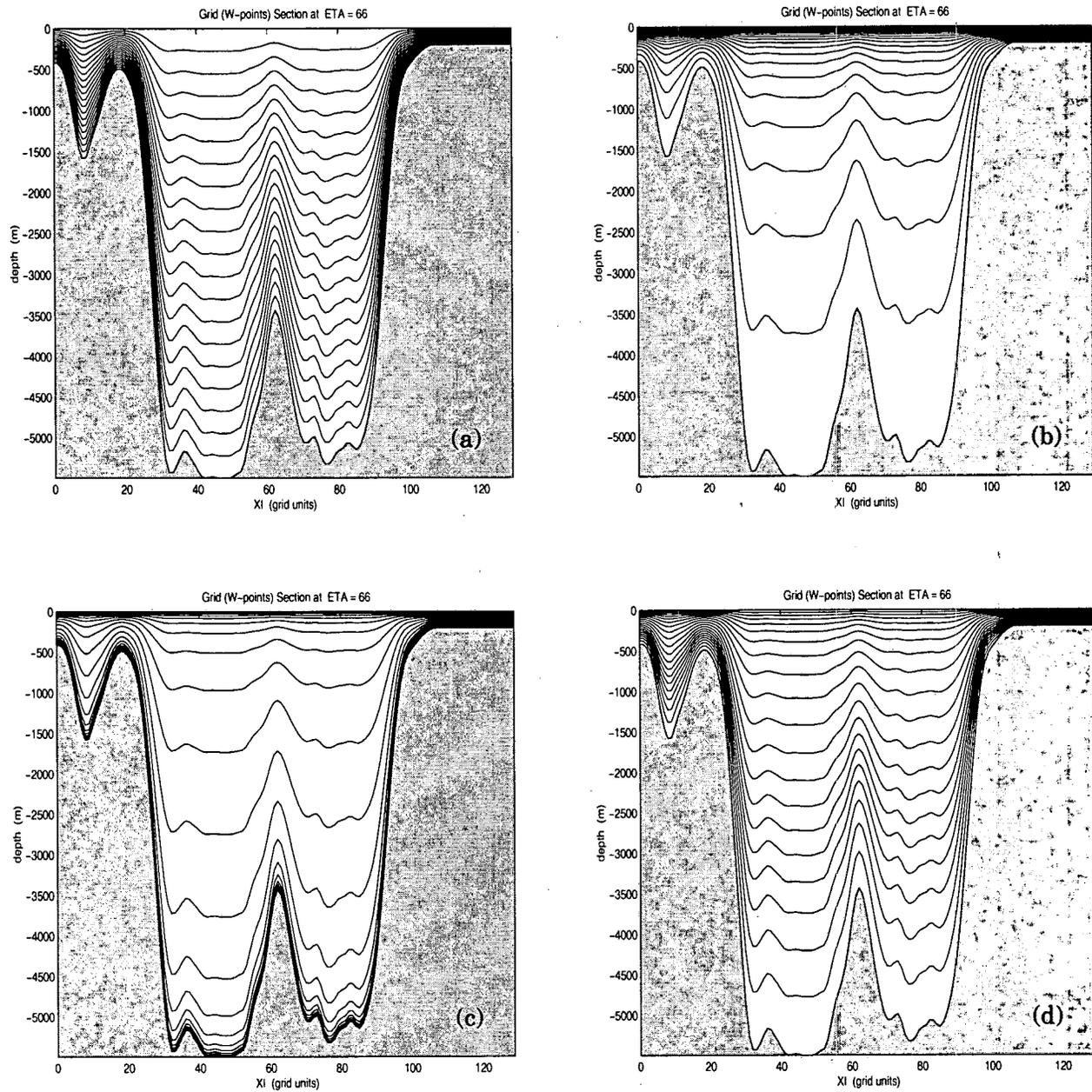


Figure 23: The s -surfaces for the North Atlantic with (a) $\theta = 0.0001$ and $b = 0$, (b) $\theta = 8$ and $b = 0$, (c) $\theta = 8$ and $b = 1$. (d) The actual values used in this domain were $\theta = 5$ and $b = 0.4$.

C 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:

$$(ds)_\xi = \left(\frac{1}{m}\right) d\xi \quad (226)$$

$$(ds)_\eta = \left(\frac{1}{n}\right) d\eta \quad (227)$$

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, [4]):

$$\nabla\phi = \hat{\xi}m \frac{\partial\phi}{\partial\xi} + \hat{\eta}n \frac{\partial\phi}{\partial\eta} \quad (228)$$

$$\nabla \cdot \vec{a} = mn \left[\frac{\partial}{\partial\xi} \left(\frac{a}{n} \right) + \frac{\partial}{\partial\eta} \left(\frac{b}{m} \right) \right] \quad (229)$$

$$\nabla \times \vec{a} = mn \begin{vmatrix} \hat{\xi}_1 & \hat{\xi}_2 & \hat{k} \\ \frac{\partial}{\partial\xi} & \frac{\partial}{\partial\eta} & \frac{\partial}{\partial z} \\ \frac{a}{m} & \frac{b}{n} & c \end{vmatrix} \quad (230)$$

$$\nabla^2\phi = \nabla \cdot \nabla\phi = mn \left[\frac{\partial}{\partial\xi} \left(\frac{m}{n} \frac{\partial\phi}{\partial\xi} \right) + \frac{\partial}{\partial\eta} \left(\frac{n}{m} \frac{\partial\phi}{\partial\eta} \right) \right] \quad (231)$$

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

D Viscosity and Diffusion

D.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

$$\Delta t < t_{damp} = \frac{\Delta x^2}{\pi^2\nu_2} \quad \text{or} \quad \frac{\Delta x^4}{\pi^4\nu_4} \quad (232)$$

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

$$\Delta x < L_{BL} = \left(\frac{\nu_2}{\beta}\right)^{\frac{1}{3}} \quad \text{and} \quad \left(\frac{\nu_4}{\beta}\right)^{\frac{1}{5}} \quad (233)$$

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 .

D.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.

D.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, as described in §3.11.

E Radiant heat fluxes

As was seen in §8.5, the model thermodynamics requires fluxes of latent and sensible heat and long-wave and shortwave radiation. We follow the lead of Parkinson and Washington [41] in computing these terms.

E.1 Shortwave radiation

The Zillman equation for radiation under cloudless skies is:

$$Q_o = \frac{S \cos^2 Z}{(\cos Z + 2.7)e \times 10^{-5} + 1.085 \cos Z + 0.10} \quad (234)$$

where the variables are as in Table 7. The cosine of the zenith angle is computed using the formula:

$$\cos Z = \sin \phi \sin \delta + \cos \phi \cos \delta \cos HA. \quad (235)$$

The declination is

$$\delta = 23.44^\circ \times \cos [(172 - \text{day of year}) \times 2\pi/365] \quad (236)$$

and the hour angle is

$$HA = (12 \text{ hours} - \text{solar time}) \times \pi/12. \quad (237)$$

The correction for cloudiness is given by

$$SW\downarrow = Q_o(1 - 0.6c^3). \quad (238)$$

An estimate of the cloud fraction c will be provided by Jennifer Francis ([12]).

E.2 Longwave radiation

The clear sky formula for incoming longwave radiation is given by:

$$F\downarrow = \sigma T_a^4 \{1 - 0.261 \exp[-7.77 \times 10^{-4}(273 - T_a)^2]\} \quad (239)$$

while the cloud correction is given by:

$$LW\downarrow = (1 + 0.275c) F\downarrow. \quad (240)$$

E.3 Sensible heat

The sensible heat is given by the standard aerodynamic formula:

$$H\downarrow = \rho_a c_p C_H V_{wg} (T_a - T_{sfc}). \quad (241)$$

E.4 Latent heat

The latent heat depends on the vapor pressure and the saturation vapor pressure given by:

$$e = 611 \times 10^{a(T_d - 273.16)/(T_d - b)} \quad (242)$$

$$e_s = 611 \times 10^{a(T_{sfc} - 273.16)/(T_{sfc} - b)} \quad (243)$$

Variable	Value	Description
(<i>a</i> , <i>b</i>)	(9.5, 7.66)	vapor pressure constants over ice
(<i>a</i> , <i>b</i>)	(7.5, 35.86)	vapor pressure constants over water
<i>c</i>		cloud cover fraction
C_E	1.75×10^{-3}	transfer coefficient for latent heat
C_H	1.75×10^{-3}	transfer coefficient for sensible heat
c_p	$1004 \text{ J kg}^{-1} \text{ K}^{-1}$	specific heat of dry air
δ		declination
<i>e</i>		vapor pressure in pascals
e_s		saturation vapor pressure
ϵ	0.622	ratio of molecular weight of water to dry air
<i>HA</i>		hour angle
<i>L</i>	$2.5 \times 10^6 \text{ J kg}^{-1}$	latent heat of vaporization
<i>L</i>	$2.834 \times 10^6 \text{ J kg}^{-1}$	latent heat of sublimation
ϕ		latitude
Q_o		incoming radiation for cloudless skies
q_s		surface specific humidity
q_{10m}		10 meter specific humidity
ρ_a		air density
<i>S</i>	1353 W m^{-2}	solar constant
σ	$5.67 \times 10^{-8} \text{ W m}^{-2} \text{ K}^{-4}$	Stefan-Boltzmann constant
T_a		air temperature
T_d		dew point temperature
T_{sfc}		surface temperature of the water/ice/snow
V_{wg}		geostrophic wind speed
<i>Z</i>		solar zenith angle

Table 7: Variables used in computing the incoming radiation and latent and sensible heat

The vapor pressures are used to compute specific humidities according to:

$$q_{10m} = \frac{\epsilon e}{p - (1 - \epsilon)e} \quad (244)$$

$$q_s = \frac{\epsilon e_s}{p - (1 - \epsilon)e_s} \quad (245)$$

The latent heat is also given by a standard aerodynamic formula:

$$LE\downarrow = \rho_a L C_E V_{wg} (q_{10m} - q_s). \quad (246)$$

Note that these need to be computed independently for the ice-covered and ice-free portions of each gridbox since the empirical factors *a* and *b* and the factor *L* differ depending on the surface type.

F 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 [26]. More practical advice on using **cpp** is given by Hazard [21].

F.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**.

F.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.

F.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
```

F.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.

F.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.

F.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.

G 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.

H 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 [38] 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 I). 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.

H.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.
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. 24. 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.

H.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.

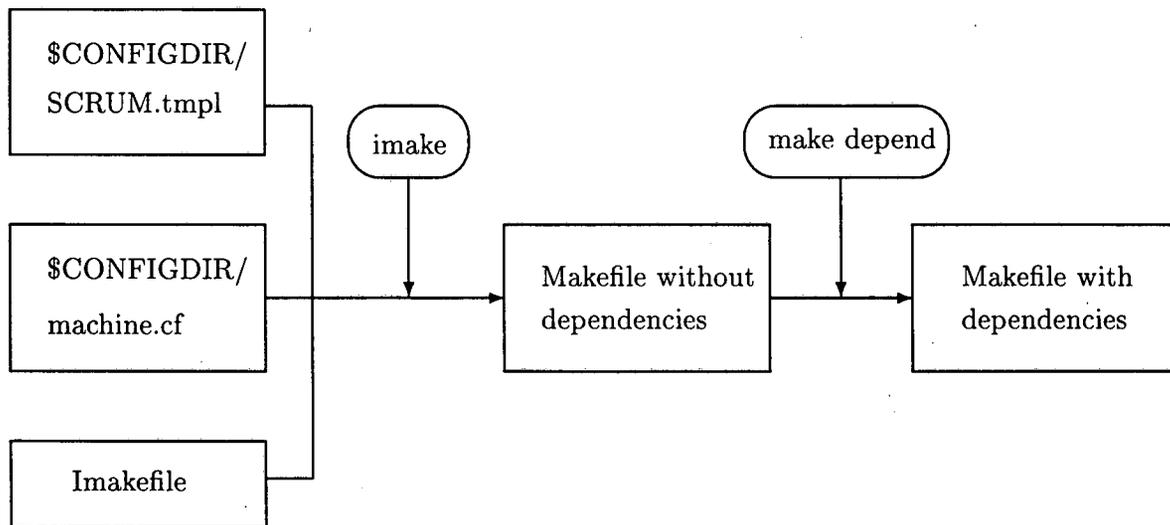


Figure 24: Creating Makefiles

- 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.

I 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. [60], while a more tutorial approach is given by Schwartz [49]. **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.

I.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.

I.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.

I.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 (**cs**h), 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.

I.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.

I.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
```

I.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 §H.

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**.

References

- [1] J. S. Allen, P. A. Newberger, and J. Federiuk. Upwelling circulation on the oregon continental shelf. part i: Response to idealized forcing. *J. Phys. Oceanogr.*, 25:1843–1866, 1995.
- [2] A. Arakawa and V. R. Lamb. *Methods of computational physics*, volume 17, pages 174–265. Academic Press, 1977.
- [3] R. Aris. *Vectors, tensors and the basic equations of fluid mechanics*. Prentice-Hall, Englewood Cliffs, NJ, 1962.
- [4] G. K. Batchelor. *An introduction to fluid dynamics*. Cambridge University Press, 1967.
- [5] 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.
- [6] A. F. Bennett. *Inverse methods in physical oceanography*. Cambridge University Press, 1992.
- [7] 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.
- [8] E. F. Carter and A. R. Robinson. Analysis models for the estimation of oceanic fields. *J. Atmos. Ocean. Tech.*, 4:49–74, 1987.
- [9] R. Daley. *Atmospheric data analysis*, chapter 5. Cambridge University Press, 1991.
- [10] E. E. Ebert and J. A. Curry. An intermediate one-dimensional thermodynamic sea ice model for investigating ice-atmosphere interactions. *J. Geophys. Res.*, 98:10085–10109, 1993.
- [11] K. N. Fedorov. Layer thicknesses and effective diffusivities in the diffusive thermocline convection in the ocean. In J. C. J. Nihoul and B. M. Jamart, editors, *Small-scale turbulence and mixing in the ocean*, pages 471–479. Elsevier, New York, 1988.
- [12] J. A. Francis and A. Schweiger. A new window opens on the arctic. *Trans. Amer. Geophys. Un.*, 81:77–83, 2000.
- [13] 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.
- [14] 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.
- [15] L. S. Gandin. *The objective analysis of meteorological fields*. Hydrometeorological Publishing House, Leningrad, 1963. English translation: Israel Program for Scientific Translations, Jerusalem, 1965.
- [16] J. M. N. T. Gray and P. D. Killworth. Stability of the viscous-plastic sea ice rheology. *J. Phys. Oceanogr.*, 25:971–978, 1995.
- [17] J. M. N. T. Gray and P. D. Killworth. Sea ice ridging schemes. *J. Phys. Oceanogr.*, 26:2420–2428, 1996.
- [18] D. B. Haidvogel and A. Beckmann. Numerical models of the coastal ocean. *The Sea*, 10:457–482, 1998.

- [19] D. B. Haidvogel and A. Beckmann. *Numerical Ocean Circulation Modeling*. Imperial College Press, 1999.
- [20] R. L. Haney. On the pressure gradient force over steep topography in sigma coordinate ocean models. *J. Phys. Oceanogr.*, 21:610-619, 1991.
- [21] W. P. Hazard. Using cpp to aid portability. *Computer Language*, 8(11):49-54, 1991.
- [22] W. D. Hibler, III. A dynamic thermodynamic sea ice model. *J. Phys. Oceanogr.*, 9:815-846, 1979.
- [23] W. D. Hibler, III. Documentation for a two-level dynamic thermodynamic sea ice model. Technical report, USACRREL, Hanover, NH, 1980. Special Report 80-8.
- [24] 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.
- [25] D. R. Jackett and T. J. McDougall. Stabilization of hydrographic data. *J. Atmos. Ocean. Tech.*, 12:381-389, 1995.
- [26] B. W. Kernighan and D. M. Ritchie. *The C Programming Language*. Prentice Hall, Englewood Cliffs, New Jersey 07632, second edition, 1988.
- [27] W. G. Large. Modeling and parameterization ocean planetary boundary layers. In E. P. Chassignet and J. Verron, editors, *Ocean Modeling and Parameterization*, pages 81-120. Kluwer Academic Publishers, 1998.
- [28] 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.
- [29] J. R. Ledwell, A. J. Wilson, and C. S. Low. Evidence for slow mixing across the pycnocline from an open-ocean tracer-release experiment. *Nature*, 364:701-703, 1993.
- [30] B. P. Leonard. A stable and accurate convective modelling procedure based on quadratic upstream interpolation. *Comput. Method Appl. Mech. Eng.*, 19:59-98, 1979.
- [31] A. Macks and J. Middleton. Numerical modelling of wind-driven upwelling and downwelling. University of New South Wales, 1993.
- [32] J. Mailhôt and R. Benoit. A finite-element model of the atmospheric boundary layer suitable for use with numerical weather prediction models. *J. Atmos. Sci.*, 39:2249-2266, 1982.
- [33] J. 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.
- [34] 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.
- [35] G. L. Mellor and L. Kantha. An ice-ocean coupled model. *J. Geophys. Res.*, 94:10,937-10,954, 1989.
- [36] G. L. Mellor and T. Yamada. A hierarchy of turbulence closure models for planetary boundary layers. *J. Atmos. Sci.*, 31:1791-1806, 1974.
- [37] G. L. Mellor and T. Yamada. Development of a turbulence closure model for geophysical fluid problems. *Rev. Geophys. Space Phys.*, 20:851-875, 1982.

- [38] A. Oram and S. Talbott. *Managing Projects with make*. O'Reilly & Associates, Inc., Sebastopol, CA, 1991.
- [39] I. Orlanski. A simple boundary condition for unbounded hyperbolic flows. *J. Comp. Phys.*, 21(3):251-269, July 1976.
- [40] R. C. Pacanowski and G. H. Philander. Parameterization of vertical mixing in numerical models of tropical oceans. *J. Phys. Oceanogr.*, 11:1443-1451, 1981.
- [41] C. L. Parkinson and W. M. Washington. A large-scale numerical model of sea ice. *J. Geophys. Res.*, 84:6565-6575, 1979.
- [42] H. Peters, M. C. Gregg, and J. M. Toole. On the parameterization of equatorial turbulence. *J. Geophys. Res.*, 93:1199-1218, 1988.
- [43] N. A. Phillips. A coordinate system having some special advantages for numerical forecasting. *J. Meteorology*, 14(2):184-185, 1957.
- [44] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes, The Art of Scientific Computing*. Cambridge University Press, 1986.
- [45] P. J. Rasch. Conservative shape-preserving two-dimensional transport on a spherical reduced grid. *Mon. Wea. Rev.*, 122:1337-1350, 1994.
- [46] W. H. Raymond and H. L. Kuo. A radiation boundary condition for multi-dimensional flows. *Quart. J. R. Met. Soc.*, 110:535-551, 1984.
- [47] 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.
- [48] 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.
- [49] R. L. Schwartz. *Learning perl*. O'Reilly & Associates, Inc., Sebastopol, CA, 1993. the llama book.
- [50] P. K. Smolarkiewicz. A simple positive definite advection scheme with small implicit diffusion. *Mon. Wea. Rev.*, 111:479-486, 1983.
- [51] P. K. Smolarkiewicz. A fully multidimensional positive definite advection transport algorithm with small implicit diffusion. *J. Comp. Phys.*, 54:325-362, 1984.
- [52] P. K. Smolarkiewicz and T. L. Clark. The multidimensional positive definite advection transport algorithm: further development and applications. *J. Comp. Phys.*, 67:396-438, 1986.
- [53] P. K. Smolarkiewicz and W. W. Grabowski. The multidimensional positive definite advection transport algorithm: non-oscillatory option. *J. Comp. Phys.*, 86:355-375, 1990.
- [54] 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.
- [55] M. Steele, G. L. Mellor, and M. G. McPhee. Role of the molecular sublayer in the melting or freezing of sea ice. *J. Phys. Oceanogr.*, 19:139-147, 1989.
- [56] 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.

- [57] P. Tag. A comparison of several forms of eddy viscosity parameterization in a two-dimensional cloud model. *J. Appl. Meteor.*, 18:1429-1441, 1979.
- [58] J. Thuburn. Multidimensional flux-limited advection schemes. *J. Comp. Phys.*, 123:74-83, 1996.
- [59] I. B. Troen and L. Mahrt. A simple model of the atmospheric boundary layer; sensitivity to surface evaporation. *Boundary-Layer Meteor.*, 37:129-148, 1986.
- [60] L. Wall, T. Christiansen, and R. L. Schwartz. *Programming perl*. O'Reilly & Associates, Inc., Sebastopol, CA, second edition, 1996. the camel book.
- [61] J. Wilkin and K. Hedstrom. User's manual for an orthogonal curvilinear grid-generation package. Institute for Naval Oceanography, 1991.

The Department of the Interior Mission



As the Nation's principal conservation agency, the Department of the Interior has responsibility for most of our nationally owned public lands and natural resources. This includes fostering sound use of our land and water resources; protecting our fish, wildlife, and biological diversity; preserving the environmental and cultural values of our national parks and historical places; and providing for the enjoyment of life through outdoor recreation. The Department assesses our energy and mineral resources and works to ensure that their development is in the best interests of all our people by encouraging stewardship and citizen participation in their care. The Department also has a major responsibility for American Indian reservation communities and for people who live in island territories under U.S. administration.

The Minerals Management Service Mission



As a bureau of the Department of the Interior, the Minerals Management Service's (MMS) primary responsibilities are to manage the mineral resources located on the Nation's Outer Continental Shelf (OCS), collect revenue from the Federal OCS and onshore Federal and Indian lands, and distribute those revenues.

Moreover, in working to meet its responsibilities, the **Offshore Minerals Management Program** administers the OCS competitive leasing program and oversees the safe and environmentally sound exploration and production of our Nation's offshore natural gas, oil and other mineral resources. The **MMS Royalty Management Program** meets its responsibilities by ensuring the efficient, timely and accurate collection and disbursement of revenue from mineral leasing and production due to Indian tribes and allottees, States and the U.S. Treasury.

The MMS strives to fulfill its responsibilities through the general guiding principles of: (1) being responsive to the public's concerns and interests by maintaining a dialogue with all potentially affected parties and (2) carrying out its programs with an emphasis on working to enhance the quality of life for all Americans by lending MMS assistance and expertise to economic development and environmental protection.

