ministerie van verkeer en waterstaat

**koninklijk nederlands meteorologisch instituut**

# A primitive equation model for the Equatorial Pacific

M.A.F. Allaart and A. Kattenberg

Technical reports; TR-124

Technische rapporten; TR-124

de bilt, 1990

KNMI

# A Primitive Equation Model for the Equatorial Pacific

Marc Allaart and Arie Kattenberg

April 17, 1990

# Contents


**1 Overview** 1
- 1.1 Introduction . . . 1
- 1.2 Correctness . . . 1
- 1.3 Derivation, assumptions and approximations . . . 2
  - 1.3.1 Momentum equation . . . 2
  - 1.3.2 Equation of state; $\rho = \rho(p, T, S)$ . . . 3
  - 1.3.3 Heat equation . . . 3

**2 Model algorithms** 4
- 2.1 Equations . . . 4
- 2.2 Boundary conditions and forcings . . . 5
- 2.3 Time integration . . . 5
  - 2.3.1 Coriolis terms . . . 5
  - 2.3.2 Surface gravity waves . . . 6
  - 2.3.3 Solution sequence . . . 7
- 2.4 Choice of the grid . . . 8
- 2.5 The implicit step . . . 9
  - 2.5.1 Setting up the matrix . . . 9
  - 2.5.2 Solution method . . . 9
- 2.6 Advection . . . 10
- 2.7 Horizontal diffusion . . . 11
- 2.8 Vertical diffusion . . . 12
- 2.9 Windstress and friction . . . 13
- 2.10 Heat- and freshwater fluxes . . . 14
- 2.11 Equation of state . . . 15

**3 Description of the program** 16
- 3.1 Introduction . . . 16
- 3.2 Language . . . 16
- 3.3 General outline . . . 17
- 3.4 Files . . . 17
- 3.5 Arrays . . . 20
- 3.6 Constants . . . 23
- 3.7 Description of the subroutines . . . 25
  - 3.7.1 Program MAIN . . . 25
  - 3.7.2 Subroutine READC . . . 26
  - 3.7.3 Subroutine CCOORD . . . 27

3.7.4 Subroutine MATRIX . . . . . 27
3.7.5 Subroutine ZERO . . . . . 27
3.7.6 Subroutine HEIGHT . . . . . 28
3.7.7 Subroutine TMPHD . . . . . 28
3.7.8 Subroutine TMPVD . . . . . 28
3.7.9 Subroutine TMPADV . . . . . 28
3.7.10 Subroutine TMPE . . . . . 29
3.7.11 Subroutine SLTHD . . . . . 29
3.7.12 Subroutine SLTVD . . . . . 29
3.7.13 Subroutine SLTADV . . . . . 29
3.7.14 Subroutine SLTE . . . . . 29
3.7.15 Subroutine DENS . . . . . 29
3.7.16 Subroutine MOMHD . . . . . 29
3.7.17 Subroutine MOMVD . . . . . 30
3.7.18 Subroutine VXADV . . . . . 30
3.7.19 Subroutine VYADV . . . . . 30
3.7.20 Subroutine DELTAV . . . . . 30
3.7.21 Subroutine IMPL1 . . . . . 30
3.7.22 Subroutine IMPL2 . . . . . 31
3.7.23 Subroutine EXPLCT . . . . . 31
3.7.24 Subroutine DIAGN . . . . . 31
3.7.25 Subroutine RESULT . . . . . 32
3.7.26 Subroutine PTOT . . . . . 32
3.7.27 Subroutine MMASK . . . . . 32
3.7.28 Subroutine INITW . . . . . 32
3.7.29 Subroutine DAY . . . . . 33
3.7.30 Subroutine TMPINI . . . . . 33
3.7.31 Subroutine SLTINI . . . . . 33
3.7.32 Subroutine INTERP . . . . . 34
3.7.33 Subroutine FRIEZE . . . . . 34
3.7.34 Subroutine REREAD . . . . . 34
3.7.35 Subroutine PACK . . . . . 34
3.7.36 Subroutine PACKUT . . . . . 35
3.7.37 Subroutine PACKIN . . . . . 35
3.8 Grids and Sub-grids. . . . . 37
Table of symbols used . . . . . 40
References . . . . . 41
Figure captions . . . . . 41

**Abstract**

This report describes and documents a version of the Pacific Equatiorial Ocean Model that was developed at KNMI during 1988 and 1989.

It is a multilevel gridpoint model with variable horizontal resolution (25 ... 400 km horizontally, 20 ... 2200 m vertically). The primitive equations for momentum, heat and salt/fresh water balance are solved in a model of a the region bounded by the latitudes $\pm 29°$ and 'realistic' meridional land-boundaries; the coastline of Middle and South America and Autralia on the other side. After a brief discussion of the physics, the model equations and the numerical solution techniques are given in chapter 2. Details of the computer code and implementation are presented in chapter 3.

# Chapter 1

# Overview

## 1.1 Introduction

We have written a computer program for making three dimensional simulations of the Pacific Ocean near the equator on climatological timescales. Our objective was to obtain a well documented state-of-the-art program for research and further development.

Before we started we studied an existing program, developed by Maier-Reimer and used by Latif (Latif 1982). The mathematical background of our model is similar to this. Details are very different, however, as will be pointed out.

The model was developed with coupled ocean-atmosphere simulations in mind, to study large scale coupled anomalies in oceanic and atmospheric circulations, related to the ENSO phenomenon. The realisation of this project is an ongoing activity on a timescale of several years and the ocean model is continually being adapted and developed further. Nevertheless it was found desirable and feasable to document and describe one stage (fall 1989) of the model development in this report.

## 1.2 Correctness

General Circulation Models, like the present one, can only simulate reality in an approximative sense. One can doubt some of the approximations that were made to derive the equations that we solve. Additional difficulties arise when these equations are discretized on a suitable space-time grid. Sub grid scale processes must be parameterized in terms of coarsely gridded bulk parameters. Also, errors are introduced by taking a finite timestep. Furthermore, complicated programs like this can contain unknown bugs.

We used the following methods to convince ourselves of the correctness of this program:

- The program is stable for a long integration period (at least 50 modelyears).
- A (N-S) symmetric windfield and initial condition lead to a symmetric result.
- The magnitude of the velocities, the shapes and the time-dependency of the resulting current system seem reasonable, as do the heat fluxes needed to obtain the right sea surface temperatures. (such a remark has only limited value; no suitable observational data to verify are available).

Further verification and testing is an ongoing activity, as is further development of the model. Additional tests that are considered are an experiment showing the dispersion of waves and a more critical analysis of the currents that are produced.

## 1.3 Derivation, assumptions and approximations

We will point out here how the equations used in the program (see section 2.1) were derived and what assumptions and approximations were made. References in square brackets [ ] are to Gill's book "Ocean Atmosphere Dynamics" (Gill 1982). Table I at the end of this report gives a summary of symbols that are used throughout the report (and their meaning).

### 1.3.1 Momentum equation

We start with Newton's second law of motion in a rotating frame of reference, [4.5.5]. Introducing the effects of (spatially varying 'eddy') viscosity a variant of [4.5.14] can be derived:

$$\frac{\mathrm{D}\vec{u}}{\mathrm{D}t} + 2\vec{\Omega} \times \vec{u} = -\frac{1}{\rho}\vec{\nabla}p - \vec{g} + \vec{\nabla} \cdot (\nu\vec{\nabla})u. \tag{1.1}$$

Subsequently a suitable coördinate system is chosen, one with the $z$-axis in the direction of the gradient of the geopotential, [4.5.10]. $x$ is in the East-West direction, $y$ in the North-South direction, both in the plane of constant geopotential (i.e. the plane of the sea- surface at rest). Because of *earth curvature* these coördinates introduce an error that increases with latitude, which is ignored. Similar to [4.5.11], [4.5.12] and [4.5.13], which are in flux form, we obtain:

$$\frac{\partial u}{\partial t} = 2\Omega_z v - 2\Omega_y w - \frac{1}{\rho}\frac{\partial p}{\partial x} + (\vec{\nabla} \cdot \nu\vec{\nabla})u - \vec{\nabla} \cdot u\vec{u}, \tag{1.2}$$

$$\frac{\partial v}{\partial t} = -2\Omega_z u + 2\Omega_x w - \frac{1}{\rho}\frac{\partial p}{\partial y} + (\vec{\nabla} \cdot \nu\vec{\nabla})v - \vec{\nabla} \cdot v\vec{u}, \tag{1.3}$$

$$\frac{\partial w}{\partial t} = 2\Omega_y u - 2\Omega_x v - \frac{1}{\rho}\frac{\partial p}{\partial z} + (\vec{\nabla} \cdot \nu\vec{\nabla})w - \vec{\nabla} \cdot w\vec{u} - \mathrm{g}. \tag{1.4}$$

The components of the earth rotation vector are now:

$$\begin{aligned} \Omega_x &= 0 \\ \Omega_y &= \Omega\cos(\vartheta) \\ \Omega_z &= \Omega\sin(\vartheta) \end{aligned}$$

Where $\vartheta$ is the latitude, a function of $y$ only. The next step is to make the *hydrostatic approximation*, i.e. the horizontal scale of the motion is assumed large compared with the vertical scale. For such motions two terms on the right hand side of equation 1.4, the pressure term and the acceleration of gravity, exceed the others by some orders of magnitude. Therefore the pressure can be calculated assuming that these two terms balance exactly. A result of this is an infinite vertical sound speed, i.e.: changes in the surface pressure are immediately communicated to all lower levels. This is reasonable because the total depth divided by the actual sound speed yields a time interval that is small compared to the timestep. The vertical velocity $w$ can, in stead of from 1.4, be computed from $u$ and $v$ using the continuïty equation.

The next approximation that is made is called the *Boussinesq approximation*, [6.4.11]. It holds when the vertical scale of the motion is small compared to the scale height. In this approximation the density is assumed to be constant, except where density variations give rise to buoyancy forces. These forces enter in the horizontal pressure gradient terms. As a result the equation for the conservation of mass implies conservation of volume; so the total volume of seawater is constant (The error involved is less than 1%).

The last approximation that we make to the momentum equation is the deletion of the term $\Omega_y w$ from equation 1.2 and the term $\Omega_x w$ from 1.3. This approximation is reasonable according to a scaling argument that is given on page 449 of Gill's book (Gill 1982).

At the surface and bottom *boundaries* a simple drag law introduces the forcing by the wind and the frictional influence of the bottom. At the vertical walls that surround the model domain 'no-slip' conditions (that allow momentum transfer 'through' the walls) are imposed on the momentum equations.

The resulting equations can be found in section (2.1).

### 1.3.2 Equation of state; $\rho = \rho(p, T, S)$

The density should be a function of temperature, salinity and pressure. We have developed a parameterisation of the density that is quadratic in temperature and linear in salinity. The density is independent of pressure, hence static incompressibility of water is assumed, although there is no real good reason to do so: this assumption must be investigated further. The obvious way to introduce it is to make the thermal expansion coëfficient a function of the depth.

### 1.3.3 Heat equation

The heat equation [4.4.6]:

$$\rho C_p \frac{DT}{Dt} - \alpha T \frac{Dp}{Dt} = \vec{\nabla} \cdot (\kappa \vec{\nabla} T - \vec{F}_{\text{rad}}) + Q_H \tag{1.5}$$

is simplified by assuming that the *thermal expansion coëfficient ($\alpha$) is zero.*
$\alpha = 0$
$Q_H$ (internal heating) $= 0$
$\frac{dQ}{dz} \neq 0$ at surface and bottom

This results in the following equation:

$$\frac{dT}{dt} = \vec{\nabla} \cdot \chi \vec{\nabla} T - \vec{\nabla} \cdot T\vec{w} \text{ (body)}, \tag{1.6}$$

and for the top layer where we assume that all radiative energy is deposited:

$$\frac{dT}{dt} = \frac{1}{\rho C_p} \frac{Q}{\Delta z} \text{ (forcing)} \tag{1.7}$$

$\chi = \frac{\kappa}{\rho C_p}$

Like in the momentum equation an (eddy) diffusivity is added to give the equation for the bulk of the water. At the bottom layer an additional Newtonian cooling term is introduced to keep the lower layers cool (diffusion and advection tend to heat up the lower layers in the long term). The top layer can exchange heat with the surroundings through a similar Newtonian heating term or by means of a more sophisticated flux-forcing scheme. The vertical walls that surround the domain are 'thermally insulated'.

# Chapter 2

# Model algorithms

## 2.1 Equations

We will solve the following equations for each layer in the model (boundary conditions and forcings will be shown in the next section.).

First the two horizontal components of the momentum equation:

$$\frac{\partial u}{\partial t} = fv - \frac{1}{\rho}\frac{\partial}{\partial x}[p + g\rho s] + (\vec{\nabla}\nu\vec{\nabla})u - (\vec{u}\cdot\vec{\nabla})u \tag{2.1}$$

$$\frac{\partial v}{\partial t} = -fu - \frac{1}{\rho}\frac{\partial}{\partial y}[p + g\rho s] + (\vec{\nabla}\nu\vec{\nabla})v - (\vec{u}\cdot\vec{\nabla})v \tag{2.2}$$

These equations give the tendencies in the horizontal velocities at a fixed point in space. On the right hand side we find:

- coriolis force;
- pressure gradient forces. Pressure due to deviations from a flat ocean surface (s, 'dynamic height') are separated out in the pressure term;
- diffusive (eddy) momentum transport;
- momentum advection.

Then the hydrostatic condition:

$$\frac{\partial p}{\partial z} = -g\rho \tag{2.3}$$

a remnant of the third component of the momentum equation, $w$ now being calculated with:

$$\vec{\nabla}\cdot\vec{u} = 0 \quad \text{or} \quad \frac{\partial w}{\partial z} = -\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y} \tag{2.4}$$

the continuïty equation for constant density.

$$\frac{\partial S}{\partial t} = (\vec{\nabla}\kappa\cdot\vec{\nabla})S - (\vec{u}\cdot\vec{\nabla})S \tag{2.5}$$

is the local change in salinity, and similarly

$$\frac{\partial T}{\partial t} = (\vec{\nabla}\kappa\cdot\vec{\nabla})T - (\vec{u}\cdot\vec{\nabla})T \tag{2.6}$$

is the local change in temperature (internal heating sources are ignored). On the righthand sides we see transports, due to respectively:

- eddy diffusivity
- advection

Finally, the equation of state (see section 2.11) is:

$$\rho = \rho_0 + \mathcal{A}(\mathrm{T} - \mathrm{T}^*)^2 + \mathcal{B}\mathrm{S} \tag{2.7}$$

## 2.2 Boundary conditions and forcings

At the vertical walls surrounding the domain 'no-slip' conditions are imposed and transport of heat and salt is not allowed. Instead of a rigid-lid condition at the surface, vertical motion is included using the boundary conditions:

$$w_{surface} = \frac{\mathrm{ds}}{\mathrm{dt}}, \quad w_{bottom} = 0. \tag{2.8}$$

Windforcing in the top layer and frictional drag in the bottom layer are introduced in the horizontal momentum equations with an extra stress term at the r.h.s:

$$+\frac{1}{\rho}\frac{\partial \sigma_x}{\partial z} \text{ in (1.1) and } +\frac{1}{\rho}\frac{\partial \sigma_y}{\partial z} \text{ in (1.2)} \tag{2.9}$$

the parameterizations for wind- and bottom stress $\vec{\sigma}$ will be discussed later (2.9).

Heating and fresh water forcing (originating from precipitation and evaporation) at the surface and cooling at the bottom can be introduced with Newtonian forcing terms at the r.h.s of equations 2.5 and 2.6 for those layers:

$$+\frac{\mathrm{A_{surf}}}{\Delta z_{\mathrm{surf}}}(\Gamma_{\mathrm{forc}} - \Gamma_{\mathrm{surf}}) \text{ or } +\frac{\mathrm{A_{bott}}}{\Delta z_{\mathrm{bott}}}(\Gamma_{\mathrm{forc}} - \Gamma_{\mathrm{bott}}) \tag{2.10}$$

with $\Gamma$ either 'T' or 'S' and $\Delta z$ a layer thickness.

## 2.3 Time integration

We will now describe a method to solve the equations 2.1, 2.2, 2.4 and 2.8. (Equations 2.5, 2.6 and 2.7 will be considered in section 2.6)

The forcing terms at the surface and bottom, described in section 2.2 are included in the description below; restriction of these terms to the relevant layer is implied.

We will start off with the coriolis force.

### 2.3.1 Coriolis terms

The time derivatives in equations 2.1 and 2.2 are changed into discrete timesteps, $\tau$; We will use a two-time-level algorithm, denoting with '$^n$' and '$^o$' the new and old timelevels. Intermediate moments in time will be discussed; they are denoted '$^a$' and '$^b$'. If no time level is indicated on a term, it is slowly varying and a precise choice is not relevant.

First we re-write the equations to single out the coriolis terms:

$$u^n = u^* + \tau f v^a \tag{2.11}$$

$$v^n = v^* - \tau f u^a \tag{2.12}$$

With $u^*$ and $v^*$ (intermediate parts of the velocity):

$$u^* = u^o - \frac{\tau}{\rho}\frac{\partial p}{\partial x} - \tau g\frac{\partial s^b}{\partial x} + \frac{\tau}{\rho}\frac{\partial \sigma_x}{\partial z} + \tau(\vec{\nabla}\nu\vec{\nabla})u - \tau(\vec{u}\cdot\vec{\nabla})u \tag{2.13}$$

$$v^* = v^o - \frac{\tau}{\rho}\frac{\partial p}{\partial y} - \tau g\frac{\partial s^b}{\partial y} + \frac{\tau}{\rho}\frac{\partial \sigma_y}{\partial z} + \tau(\vec{\nabla}\nu\vec{\nabla})v - \tau(\vec{u}\cdot\vec{\nabla})v \tag{2.14}$$

We define '$^a$' with a linear interpolation:

$$u^a \equiv \alpha u^o + (1-\alpha)u^n \tag{2.15}$$

with $\alpha = 0$ yielding an 'implicit' and $\alpha = 1$ an explicit scheme.

Substituting 2.15 in 2.11 and 2.12 gives:

$$u^n[1+\tau^2 f^2(1-\alpha)^2] = [u^* + \tau f\alpha v^o] + \tau f(1-\alpha)[v^* - \tau f\alpha u^o] \tag{2.16}$$

$$v^n[1+\tau^2 f^2(1-\alpha)^2] = [v^* - \tau f\alpha u^o] - \tau f(1-\alpha)[u^* + \tau f\alpha v^o] \tag{2.17}$$

Conservation of kinetic energy is possible by requiring:

$$(u^n)^2 + (v^n)^2 = (u^o)^2 + (v^o)^2 \tag{2.18}$$

from equations 2.16 and 2.17, when $u^*$ , $v^*$ are set to $u^o$, $v^o$. The result is that energy will only be conserved if $\alpha = 0.5$. (Latif uses in his program $\alpha = 0.0$). For stationary solutions, where all time-derivatives vanish, this choice is irrelevant. Time dependent solutions may be affected.

### 2.3.2 Surface gravity waves

We continue with solving equations 2.16, 2.17, 2.4 and 2.8, using 2.13 and 2.14. Index '$^b$' in 2.13 and 2.14 indicates an interpolation in the timestep similar to 2.15:

$$s^b \equiv \beta s^o + (1-\beta)s^n \tag{2.19}$$

Labelling a common factor in 2.16 and 2.17 as $\gamma$:

$$\gamma = [1+\tau^2 f^2(1-\alpha)^2]^{-1} \tag{2.20}$$

We can rewrite 2.16 and 2.17 as:

$$u^n = \gamma[u^+ - \tau g(1-\beta)\frac{\partial s^n}{\partial x}] + \gamma\tau f(1-\alpha)[v^+ - \tau g(1-\beta)\frac{\partial s^n}{\partial y}] \tag{2.21}$$

$$v^n = \gamma[v^+ - \tau g(1-\beta)\frac{\partial s^n}{\partial y}] - \gamma\tau f(1-\alpha)[u^+ - \tau g(1-\beta)\frac{\partial s^n}{\partial x}] \tag{2.22}$$

with:

$$u^+ = u^o - \frac{\tau}{\rho}\frac{\partial p^o}{\partial x} - \tau g\beta\frac{\partial s^o}{\partial x} + \frac{\tau}{\rho}\frac{\partial \sigma^o{}_x}{\partial z} + \tau f\alpha v^o + \tau(\vec{\nabla}\nu\vec{\nabla})u^o - \tau(\vec{u}^o\cdot\vec{\nabla})u^o \tag{2.23}$$

$$v^+ = v^o - \frac{\tau}{\rho}\frac{\partial p^o}{\partial y} - \tau g\beta\frac{\partial s^o}{\partial u} + \frac{\tau}{\rho}\frac{\partial \sigma^o{}_y}{\partial z} - \tau f\alpha u^o + \tau(\vec{\nabla}\nu\vec{\nabla})v^o - \tau(\vec{u}^o\cdot\vec{\nabla})v^o \tag{2.24}$$

We now take equations 2.4 and 2.8, and change the continuous timestep into a discrete one. The surface velocity needed in 2.8 can be obtained by vertical integration of $\frac{\partial w}{\partial z}$ from depth 0 to depth D (where $w(D) = w_{\text{bottom}} \equiv 0$). We could calculate 2.8 at any moment in the timestep, that is,

make an interpolation like 2.15 and 2.19 but that complicates the program unnecesarily. For u and v the values at the new time level will be used:

$$s^n = s^o - \tau \int_0^D [\frac{\partial u^n}{\partial x} + \frac{\partial v^n}{\partial y}]\, dz \tag{2.25}$$

Now we insert 2.21 and 2.22, reorder and split $s^n$ for notational convenience in two terms ($s^n = s^e + s^i$):

$$s^e = s^o - \tau \int_0^D [\frac{\partial}{\partial x}(\gamma u^+ + \gamma\tau f(1-\alpha)v^+) + \frac{\partial}{\partial y}(\gamma v^+ - \gamma\tau f(1-\alpha)u^+)]\, dz \tag{2.26}$$

$$s^i = \tau^2 g D(1-\beta)[\frac{\partial}{\partial x}(\gamma\frac{\partial s^n}{\partial x}) + \frac{\partial}{\partial x}(\gamma\tau f(1-\alpha)\frac{\partial s^n}{\partial y}) + \frac{\partial}{\partial y}(\gamma\frac{\partial s^n}{\partial y}) - \frac{\partial}{\partial y}(\gamma\tau f(1-\alpha)\frac{\partial s^n}{\partial x})] \tag{2.27}$$

We simplify equation 2.26 by interchanging intergration and differentiation.

$\beta = 1$ will result in a straightforward explicit formulation,

$\beta = 0$ will result in a completely implicit alorithm.

Because the classical CFL-criterium is violated:

$$\tau^2 g D / \min(\Delta x, \Delta y) > 1 \tag{2.28}$$

explicit formulation may be unstable, and we will have to make another choice. Latif uses in his program $\beta = 0.0$; experiments show that the algorithm remains stable for $0.0 < \beta < 0.5$, and for stationary solutions the results are not affected by $\beta$. Possibly the dispersion of waves can be 'tuned' choosing an optimal value for $\beta$. Tests must prove this. We have chosen $\beta = 0.4$.

When a spatial discretisation is introduced, and $\vec{s}$ denotes a one dimensional vector of all surface elevations, the implicit equation 2.27 can be written as a matrix equation, in symbolic notation:

$$\vec{s}^n - \vec{s}^e = \vec{\vec{M}} \cdot \vec{s}^n \tag{2.29}$$

and so we can write ($\vec{\vec{I}}$ is the identity matrix):

$$\vec{s}^n = (\vec{\vec{I}} - \vec{\vec{M}})^{-1} \cdot \vec{s}^e \tag{2.30}$$

### 2.3.3 Solution sequence

Now we can step equations 2.16, 2.17, 2.4 and 2.8 forward in time in the following way:

1. - compute $u^+$, $v^+$ (2.23), (2.24)
2. - $\int u^+ dz$, $\int v^+ dz$
3. - compute $s^e$ (2.27)
4. - compute $s^n$ (2.30)
5. - compute $u^n$, $v^n$ (2.21), (2.22)
6. - derive $w^n$ (2.4)
7. - recompute $s^n$ (2.8)
8. - salinity S (2.5)
9. - temperature T (2.6)
10. - density $\rho$ (2.7)
11. - pressure $p$ (2.3)

## 2.4 Choice of the grid

The two grids most used in ocean modelling are the Arakawa C-grid and the Arakawa E-grid. (Arakawa, 1966). For both grids a version of the program exists. The grid we use is rectangular, but not equidistant, so for every spatial derivative we will have to specify the "current $dx$ or $dy$".

C-grid

| | $x_{i-2}$ | $x_{i-1}$ | $x_i$ | $x_{i+1}$ | $x_{i+2}$ |
|---|---|---|---|---|---|
| $y_{j+2}$ | $s_{i-2,j+2}$ | $u_{i-1,j+2}$ | $s_{i,j+2}$ | $u_{i+1,j+2}$ | $s_{i+2,j+2}$ |
| $y_{j+1}$ | $v_{i-2,j+1}$ | | $v_{i,j+1}$ | | $v_{i+2,j+1}$ |
| $y_j$ | $s_{i-2,j}$ | $u_{i-1,j}$ | $s_{i,j}$ | $u_{i+1,j}$ | $s_{i+2,j}$ |
| $y_{j-1}$ | $v_{i-2,j-1}$ | | $v_{i,j-1}$ | | $v_{i+2,j-1}$ |
| $y_{j-2}$ | $s_{i-2,j-2}$ | $u_{i-1,j-2}$ | $s_{i,j-2}$ | $u_{i+1,j-2}$ | $s_{i+2,j-2}$ |

We will now show what equation 2.27 looks like when it is discretized on this grid. For simplicity all indices '$^n$' are left out, and the representation of the equation is split in two stages; the "continuïty part" 2.31, and the "$u-v$ part" 2.32. ($u$ and $v$ are intermediate symbols, defined here):

The continuïty part is trivial:

$$s^i = -\tau D[(\frac{u_{i+1,j} - u_{i-1,j}}{x_{i+1} - x_{i-1}}) + (\frac{v_{i,j+1} - v_{i,j-1}}{y_{j+1} - y_{j-1}})] \tag{2.31}$$

The $u-v$ parts seem trivial too;

$$u_{i+1,j} = -\tau g(1-\beta)\gamma_{i+1,j}[s^u_{i+1,j} + \tau f_{i+1,j}(1-\alpha)s^v_{i+1,j}] \tag{2.32}$$

(Showing one term out of four), but they are not: The $s^u$ and $s^v$ are not defined on the gridpoint indicated, so we will have to use an interpolation.

$$\begin{aligned}(x_{i+2} - x_i)(y_{i+1} - y_{i-1})s^v_{i+1,j} \quad &= \quad s^v_{i+2,j+1}(x_{i+1} - x_i)(y_j - y_{j-1}) + \\ &+ \quad s^v_{i+2,j-1}(x_{i+1} - x_i)(y_{j+1} - y_j) + \\ &+ \quad s^v_{i,j+1}(x_{i+2} - x_{i+1})(y_j - y_{j-1}) + \\ &+ \quad s^v_{i,j-1}(x_{i+2} - x_{i+1})(y_{j+1} - y_j).\end{aligned} \tag{2.33}$$

the symbols $s^u$ and $s^v$ are shorthand for:

$$s^u_{i,j} = \frac{s_{i+1,j} - s_{i-1,j}}{x_{i+1} - x_{i-1}} \tag{2.34}$$

and

$$s^v_{i,j} = \frac{s_{i,j+1} - s_{i,j-1}}{y_{j+1} - y_{j-1}}. \tag{2.35}$$

E-grid

| | $x_{i-2}$ | $x_{i-1}$ | $x_i$ | $x_{i+1}$ | $x_{i+2}$ |
|---|---|---|---|---|---|
| $y_{j+2}$ | | | $s_{i,j+2}$ | | |
| $y_{j+1}$ | | $s_{i-1,j+1}$ | $v_{i,j+1}$ | $s_{i+1,j+1}$ | |
| $y_j$ | $s_{i-2,j}$ | $u_{i-1,j}$ | $s_{i,j}$ | $u_{i+1,j}$ | $s_{i+2,j}$ |
| $y_{j-1}$ | | $s_{i-1,j-1}$ | $v_{i,j-1}$ | $s_{i+1,j-1}$ | |
| $y_{j-2}$ | | | $s_{i,j-2}$ | | |

The continuïty part is exactly the same as with the C-grid, but the $u - v$ parts are now defined on the gridponts indicated.

One can see the E-grid as two interlaced C-grids. There are indeed two independent solutions for the advection-diffusion equation and they tend to drift apart, we will call this effect the E-grid waves.

## 2.5 The implicit step

### 2.5.1 Setting up the matrix

The matrix $\vec{\vec{M}}$ in 2.29 is constructed by sorting the equations by s-term used. For example we will get for the $s_{i+2,j}$ term:

$$\tau^2 g D(1-\beta)\gamma_{i+2,j}\frac{1}{x_{i+2}-x_i}\frac{1}{x_{i+1}-x_{i-1}} \tag{2.36}$$

The position of the matrix elements 2.36 in the matrix M depends on how we arrange the s-points into a vector. The method currently used is the following:

1. From west to east.
2. From south to north.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | · | · | 13 | 14 | 15 | 16 | etc. | | | |
| | · | · | 7 | 8 | 9 | 10 | 11 | 12 | · | · |
| $y\uparrow$ | · | · | 1 | 2 | 3 | 4 | 5 | 6 | · | · |
| | · | · | · | · | · | · | · | · | · | · |
| | · | · | · | · | · | · | · | · | · | · |

$\underrightarrow{x}$

The dots represent land-points that are not included in the computations, if they were included they would remain unchanged.
The matrix contains one row for each numbered s-point. On this row there are 9 nonzero coëfficients for the terms 2.36.

### 2.5.2 Solution method

The equation to be solved here is 2.30 that reads in a simplified form:

$$\vec{A} = \vec{\vec{N}}^{-1} \cdot \vec{B} \tag{2.37}$$

The matrix $\vec{\vec{N}}$ is well conditioned and can be inverted with Gauss-elimination without pivoting. For reasons of efficiency (discussed later) the inverse matrix $\vec{\vec{N}}^{-1}$ can not be computed and stored during calculations. We will use techniques called L-U decomposition (Forsythe and Moler 1967) and forward and backward substitution (Press et.al. 1986).
2.37 can be written as:

$$\vec{\vec{N}} \cdot \vec{A} = \vec{B} \tag{2.38}$$

L-U decomposition is a method to write an arbitrary (nonsingular) matrix as a product of two matrices:

$$\vec{\vec{L}} \cdot \vec{\vec{U}} \cdot \vec{A} = \vec{B} \tag{2.39}$$

Where L is zero in its upper triangle; and U is zero in its lower triangle. The diagonal elements of L are all one.

If we now introduce vector $\vec{Y}$ and write 2.39 as:

$$\vec{\vec{U}} \cdot \vec{A} = \vec{Y} \tag{2.40}$$

$$\vec{\vec{L}} \cdot \vec{Y} = \vec{B} \tag{2.41}$$

The last one is computed by forward substitution:

$$Y_j = B_j - \sum_{i=1}^{j} L_{i,j} Y_i \tag{2.42}$$

And finally 2.40 is solved by backward substitution:

$$A_j = (Y_j - \sum_{i=j+1}^{n} U_{i,j} A_i)/U_{j,j} \tag{2.43}$$

## 2.6 Advection

Equations 2.1, 2.2, 2.5 and 2.6 show advection terms, that have the form:

$$\ldots(\vec{\nabla} \cdot \vec{u})B, \tag{2.44}$$

where B is a scalar. These terms result from a transformation from the frame of reference of the water to the frame of reference of the fixed grid. All the fundamental properties of the water should be advected, in our case, $u$, $v$, T and S. From a computational point of view, T and S are different from $u$ and $v$ because the T or S-points are located on a different sub-grid, also 2.44 is nonlinear when B is $u$ or $v$.

Numerically solving the advection equation (or the advection- diffusion equation) is a complicated matter and much has been published about it (e.g. Roache, 1982). Term 2.44 should conserve many different properties; for example any integer power of B, so our task is to construct an algorithm with the following properties:

- Threedimensional
- Stable
- 2 - time level

- Conserves $B^n$ for as many n as possible.

Our choice is the "second upwind differencing" or "donor cell" method, a little modified for $u$ and $v$. This method is stable, threedimensional, 2-time level and shows a local conservation of B. A distinct advantage of this method is the easy handling of the "land-points", in the case of u and v. For T and S, we have to make sure that the "land-points" do not act as eternal sources or sinks. The method works like this for $\Gamma$ (T or S):

gridpoints:

$$\Gamma_{i-2} \quad U_{i-1} \quad \Gamma_i \quad U_{i+1} \quad \Gamma_{i+2}$$

$$\begin{aligned}(\Gamma_i^n - \Gamma_i^o)/\tau \quad &= \frac{\Gamma_{i+2}^o - \Gamma_i^o}{x_{i+1} - x_{i-1}} \min(u_{i+1}^{*o}, 0) + \\ &+ \frac{\Gamma_i^o - \Gamma_{i-2}^o}{x_{i+1} - x_{i-1}} \max(u_{i-1}^{*o}, 0).\end{aligned} \tag{2.45}$$

For u and v we use on the E-grid a 4-point interpolation like this:

$$\begin{array}{ccccc} & u_{i-1,j+1} & & u_{i+1,j+1} & \\ u_{i-2,j} & u^*_{i-1,j} & u_{i,j} & u^*_{i+1,j} & u_{i+2,j} \\ & u_{i-1,j-1} & & u_{i+1,j-1} & \end{array}$$

On the C-grid a 2-point interpolation is sufficient to obtain a suitable $u^*$ or $v^*$.
It is unclear to us what choice Latif makes in his program. We have looked for another method that conserves $B^2$ as well, (amounting to conservation of kinetic and potential energy) and we found at least one (the Lax-Wendroff method). Its generalisation to three dimensions is not straightforward, however, and implementation of this method awaits further study.

## 2.7 Horizontal diffusion

There are two reasons for including horizontal diffusion in the program. One is a (very arbitrary) parametrisation of the (turbulent) viscosity and conductivity, and the other is the numerical stability of the program. In this model $\kappa_h$ is not a function of the coordinates, so diffusion is modelled as:

$$\frac{\partial \mathrm{B}}{\partial t} = \kappa_h \nabla^2 \mathrm{B} \tag{2.46}$$

where B is a scalar.

We use an 8-point diffusion operator that conserves B.

$$\begin{array}{ccccc} & & \mathrm{B}_{i,j+2} & & \\ & \mathrm{B}_{i-1,j+1} & & \mathrm{B}_{i+1,j+1} & \\ \mathrm{B}_{i-2,j} & & \mathrm{B}_{i,j} & & \mathrm{B}_{i+2,j} \\ & \mathrm{B}_{i-1,j-1} & & \mathrm{B}_{i+1,j-1} & \\ & & \mathrm{B}_{i,j-2} & & \end{array}$$

$$\begin{aligned}&\nabla^2 \mathrm{B}(x_{i+1} - x_{i-1})(y_{j+1} - y_{j-1}) = \\ &\quad \kappa_c(\mathrm{B}_{i+2,j} - \mathrm{B}_{i,j})(y_{j+1} - y_{j-1})/(x_{i+2} - x_i) + \\ &\quad + \kappa_c(\mathrm{B}_{i-2,j} - \mathrm{B}_{i,j})(y_{j+1} - y_{j-1})/(x_i - x_{i-2}) +\end{aligned} \tag{2.47}$$

$$
\begin{aligned}
&+\kappa_c(\mathrm{B}_{i,j+2}-\mathrm{B}_{i,j})(x_{i+1}-x_{i-1})/(y_{j+2}-y_j)+\\
&+\kappa_c(\mathrm{B}_{i,j-2}-\mathrm{B}_{i,j})(x_{i+1}-y_{i-1})/(y_j-y_{j-2})+\\
&+1/4\kappa_e(\mathrm{B}_{i+1,j+1}-\mathrm{B}_{i,j})(x_{i+1}-x_i)(y_{j+1}-y_j)/\{(x_{i+1}-x_i)^2+(y_{j+1}-y_j)^2\}+\\
&+1/4\kappa_e(\mathrm{B}_{i+1,j-1}-\mathrm{B}_{i,j})(x_{i+1}-x_i)(y_j-y_{j-1})/\{(x_{i+1}-x_i)^2+(y_j-y_{j-1})^2\}+\\
&+1/4\kappa_e(\mathrm{B}_{i-1,j+1}-\mathrm{B}_{i,j})(x_i-x_{i-1})(y_{j+1}-y_j)/\{(x_i-x_{i-1})^2+(y_{j+1}-y_j)^2\}+\\
&+1/4\kappa_e(\mathrm{B}_{i-1,j-1}-\mathrm{B}_{i,j})(x_i-x_{i-1})(y_j-y_{j-1})/\{(x_i-x_{i-1})^2+(y_j-y_{j-1})^2\}.
\end{aligned}
$$

For testing purposes the program contains two variables for $\kappa_h$, one for the first four terms in 2.48; $\kappa_c$ (C-grid value) and one for the last four terms; (E-grid value) (the last four terms are absent in the C-grid version).

## 2.8 Vertical diffusion

The vertical diffusion is modeled using 'K-theory'. The strength of the diffusion is determined by the stability of one layer with respect to an adjacent layer. This is expressed in the Richardson number, defined as:

$$
R_i=\frac{-g(\frac{\partial\rho}{\partial z})}{\rho[(\frac{\partial u}{\partial z})^2+(\frac{\partial v}{\partial z})^2]} \tag{2.48}
$$

$R_i > 0.25$ implies stability,
$R_i < 0.25$ implies instability.

How to model strong turbulent diffusion is not yet clear. A very interesting approach is the transilient turbulence (Stull, Kraus, 1987), that we are testing in the context of this ocean model. K-theory, that we use here, is not really suited for strong turbulence. For not-so-strong diffusion we have a rough approximation (Pacanowski, Philander, 1981), and a more recent modification of their result (Peters, Gregg, Toole, 1981), showing a much stronger diffusion. In order to make the formulation simpler we introduce here the so called Richardson function :

$$
R_f\equiv(1+5R_i)^{-1} \tag{2.49}
$$

$\kappa$ and $\nu$ are now expressed as a function of $R_f$:

$$
\kappa=\kappa_0+\kappa_1R_f^{2.5}+\kappa_2R_f^{16} \tag{2.50}
$$

$$
\nu=\nu_0+\nu_1R_f^{1.5}+\nu_2R_f^{16} \tag{2.51}
$$

$\nu_0 = 2.0\cdot10^{-5}$ m$^2$/s
$\nu_1 = 5.0\cdot10^{-4}$ m$^2$/s
$\nu_2 = 640$ m$^2$/s
$\kappa_0 = 1.0\cdot10^{-6}$ m$^2$/s
$\kappa_1 = 5.0\cdot10^{-4}$ m$^2$/s
$\kappa_2 = 80$ m$^2$/s
The value of $\kappa_2$ is not certain; Because Kraus claims turbulent conductivity has 1/8 of the value of turbulent viscosity (Kraus, 1987) we took the value cited above.

Since we have an explicit scheme the value of $\nu$ or $\kappa$ is restricted to: $\nu$ (or $\kappa$) $< 1/2(\Delta z)^2/\tau$.

A vertical slice of the grid looks like this:

$$\begin{array}{llll} T & u,v & T & (k+2) \\ w & R & w & (k+1) \\ T & u,v & T & (k) \\ w & R & w & (k-1) \\ T & u,v & T & (k-2) \end{array}$$

The equation for vertical diffusion:

$$\frac{\partial B}{\partial t} = \frac{\partial}{\partial z}[\nu \frac{\partial B}{\partial z}] \tag{2.52}$$

looks like this in discrete form:

$$\Delta B_k = \tau[\nu_{k+1}\frac{B_{k+2} - B_k}{z_{k+2} - z_k} - \nu_{k-1}\frac{B_k - B_{k-2}}{z_k - z_{k-2}}]\frac{1}{z_{k+1} - z_{k-1}} \tag{2.53}$$

This equation conserves B, independent of the value of $\nu$. For the viscosity equation, ($u$,$v$-subgrids) we will find the $\nu$-values on the R-subgrid, The richardson number is computed by first defining the density on the T-subgrid, and then interpolating that to the $u$,$v$-subgrid.

For the thermal- conductivity equation $u$,$v$ are interpolated to the T-subgrid, the rest is straightforward. We have verified the numerical stability of the resulting scheme. Due to the strong nonlinearity occasional chaotic behavior may arise, without affecting the resulting structure, however.

## 2.9 Windstress and friction

The windstress, according to 2.9, in equations 2.1 and 2.2 looks like this:

$$\frac{\partial u}{\partial t} = \frac{1}{\rho_s}\frac{\partial \sigma_x}{\partial z} \tag{2.54}$$

$$\frac{\partial v}{\partial t} = \frac{1}{\rho_s}\frac{\partial \sigma_y}{\partial z} \tag{2.55}$$

Where $\sigma_x$ and $\sigma_y$ are defined as:

$$\sigma_x = C_D \rho_a (d_x^2 + d_y^2 + 4\Sigma^2)^{1/2} d_x, \tag{2.56}$$

$$\sigma_y = C_D \rho_a (d_x^2 + d_y^2 + 4\Sigma^2)^{1/2} d_y, \tag{2.57}$$

with

$$d_x = u_{\text{wind}} - u_{\text{sea surface}}, \tag{2.58}$$

$$d_y = v_{\text{wind}} - v_{\text{sea surface}}. \tag{2.59}$$

$C_D = 0.0012$
$\rho_a = 1.2$ kg/m$^3$
$\rho_s = 1025$ kg/m$^3$
$\Sigma = 3.0$ m/s

The windfields used are monthly averaged windfields. Because windstress is quadratic in the windspeeds, $\Sigma$ is introduced to account for the variability of the real wind around the monthly averaged windstress (Gill 1986).

According to S.Weber (personal communication), 2.54, 2.55 can also be used for bottom friction; 2.56, 2.57 reduce to:

$$\sigma_x = C_D \rho_s (u^2 + v^2)^{1/2} u \tag{2.60}$$

$$\sigma_y = C_D \rho_s (u^2 + v^2)^{1/2} v. \tag{2.61}$$

Coastlines are not handeled in this way; here the "no-slip boundary condition" is used. Land-points are considered to be sea-points with u and v forced to zero. Horizontal diffusion allows momentum transfer through the coastlines. Experiments show that the results are not really different from "free slip boundaries".

## 2.10 Heat- and freshwater fluxes

Heat forcing is described by eq. 1.7; in discrete form this is:

$$\Delta T = \frac{\tau}{\Delta z} \frac{Q}{\rho C_p} \tag{2.62}$$

$\Delta T$ change in temperature in one timestep
$Q$ is the heatflux,
$C_p$ is the specific heat of water (4000J/kg/K),
$\tau$ is the timestep (7200 s),
$\Delta z$ is the thickness of the first layer (20m)

Haney (1971) parameterised the heatflux in terms of the difference between the air temperature and the SST (sea surface temperature):

$$\frac{Q}{\rho C_p} = A(T_{\text{air}} - T_{\text{SST}}) \tag{2.63}$$

$A = 3.0 \cdot 10^{-6}$m/s
We can implement this formula by setting $T_{\text{air}}$ to the observed SST and $T_{\text{SST}}$ to the temperature in the upper layer. The temperature in the upper layer will relax to the observed SST in a time $\frac{\Delta z}{A}$, which is about 77 days.

Another approach is forcing the model with such a heatflux that $T_{\text{SST}}$ remains equal to the observed SST. Now the heatflux $Q$ is the computed quantity. We can accomplish this by setting $A$ to a value so that $T_{\text{upper layer}}$ will relax to the SST in one timestep:

$$A = \frac{\Delta z}{\tau} \tag{2.64}$$

We can compute the error in the simulated SST by substituting 2.63 in 2.64:

$$Q = \rho C_p \frac{\Delta z}{\tau} T_{\text{error}} \tag{2.65}$$

For the extreme case of $Q = 1000$ W/m$^2$ this results in $T_{\text{error}} = 0.1$°C.

To avoid warming up of the lowest layer by diffusion and advection the model actively cools the lowest layer. This mimicks the supply of cold water from high-latitude deep water formation regions that flows equatorward and wells up to form the (sub)tropical thermocline in the real ocean.

$$\frac{\partial T}{\partial t} = \frac{A}{\Delta z}(T_{\text{bottom}} - T_{\text{lowest layer}}) \tag{2.66}$$

$\Delta z$ is the thickness of the lowest layer (2200 m),
$A = 1.0 \cdot 10^{-3}$m/s and
$T_{\text{bottom}}$ must not be higher than the lowest temperature in the initial condition for stability reasons. The coastlines are thermally isolated.

For the salt flux at the surface a similar forcing scheme was chosen:

$$\frac{\partial S}{\partial t} = \frac{A}{\Delta z}(S_{\text{forcing}} - S_{\text{upper layer}}) \tag{2.67}$$

$\Delta z$ is the thickness of the upper layer (20 m),
$S_{\text{forcing}}$ is the observed surface salinity from Levitus' atlas.

The fresh water flux can be calculated as:

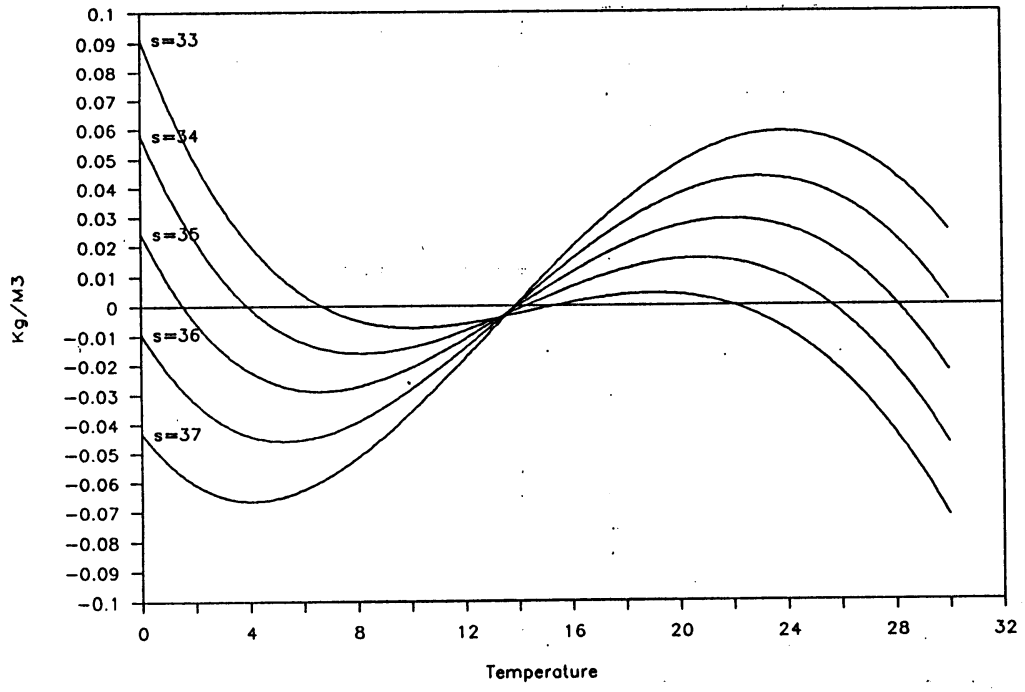
$$\frac{\partial F}{\partial t} = -\frac{\Delta z}{S}\frac{\partial S}{\partial t} \quad [\text{m/s}]$$

## 2.11 Equation of state

The density of seawater as a function of temperature and salinity is published by the UNESCO (UNESCO, 1981). The UNESCO approximation is probably a Chebyshev- polynomal approximation to the observed values. This approximation limits the absolute error to a certain value, but we are not so much interested in an exact value as in a well defined first derivative. So we made a second-order Chebyshev polynomal approximation in temperature, and a first order Chebyshev polynomial approximation in salinity, to the UNESCO function with the following result:

$$\rho = \rho_0 + \mathcal{A}(\mathrm{T} - \mathrm{T}^*)^2 + \mathcal{B}\mathrm{S} \tag{2.68}$$

With $\rho_0$ = 1001.32630 kg/m$^3$, $\mathcal{A}$ = -0.00471 kg/m$^3$K$^2$, $\mathcal{B}$ = 0.77390 kg/m$^3$ per permille and $\mathrm{T}^* = -7.73508$°C. It is a truncated Chebychev polynomial that is fit to the full equation, given by Gill [App. three] around 14°C and 35 permille salinity. It gives less than 0.1 permille error in the ranges 0–32°C and 33–37 permille salinity. Below the *density error* is shown.



# Chapter 3

# Description of the program

## 3.1 Introduction

In this chapter we will describe the computer program itself. At this moment (August 1989) two versions of the program exist and this paper refers to both of them.

Version 18.4: Arakawa E-grid.
Version 19.2: Arakawa C-grid.

## 3.2 Language

An attempt was made to write the program in ansi fortran77. A few exception to this rule are:

### update format

The source code is written as a series of Cray- (or Cyber-) 'update' input decks. After processing by update (a batch editor) the source can be compiled. The main task of update is to include the common-blocks at the start of almost every program unit. On other computer systems (like Unix systems, vax/vms, Unisys A-series and IBM pc compatibles), the common- deck at the start of the source code must be written to a separate file and the "*CALL COMMON" directive must be changed into an appropriate "INCLUDE" compiler directive.

### namelist i/o

Namelist i/o is a technique used to simplify the control of the program and it is widely supported. It allows the use of defaults for all variables and the user should only specify the variables for which the default is not good. On computer systems where namelist i/o is not implemented, these statements must be replaced by normal read and write statements and so the user must specify the values of all (about 30) variables.

### message

Is a subroutine that writes a message to the dayfile. The call to message can be deleted without affecting the program. On a Convex computer it can be replaced by a write operation to unit zero.

## 3.3 General outline

The flow structure of the program is simple. It could have been written as one long program unit. In order to make it more readable and to decrease compile time it was split up into a number of subroutines that are called in a fixed order from the main program. All communication between the program units goes through the common blocks that are declared at the start of the these program units.

One can subdivide the program into three parts. The initialization part, the stepping part and the freeze part.

### Stepping

Within the program a fundamental time-step exist. Normally many steps are made during one run. For every step a sequence of subroutines is called that do the actual calculations. The order in which they are called is relevant. The subroutine that prints some results (a diagnosis of the behavior of the program) is not (necessarily) called at every time-step.

### Initialization

The initialization part is a series of subroutines that initializes a run. Some arrays that are calculated once and not changed later, are calculated here. Also the arrays describing the current status of the ocean (velocities, temperatures) are initialized, either by reading results from a previous run (frozen data), by reading observational data, or from an (internal) crude guess.

### Freeze part

At the end of the run the ocean status can be written to a file ("frozen"). This file can be used to initialize a subsequent run. Also a file with input to a post-processor program can be written.

## 3.4 Files

The program can use up to 9 files. The next table is a short summary of their use.

| unit | filename | r/w | type | use |
|---|---|---|---|---|
| 1 | LTMP | r | b/p | sea-water temps |
| 1 | LSLT | r | b/p | salinity values |
| 1 | LFSU | r | b/p | winds/SST |
| 1 | LMASK | r | b/p | land/sea mask |
| 2 | LPACK | w | p | packing results |
| 3 | LREAD | r | b | read frozen file |
| 4 | LWRITE | w | b | write frozen file |
| 5 | input | r | s | namelist/read |
| 6 | output | w | s | diagnosis |
| 10 | LFSU | r | r | winds/SST |
| 11 | DIAGN | w | s | diagnostic output |
| 14 | — | r/w | r | (Q-flux/W-flux) |
| 15 | — | r/w | r | (SST-anomalies) |

r=read only, w=write only, r/w=read or write. b=binary, p=packed, s=sequential (characters), r=random access.

**Input** file used in MAIN
This file should contain two *namelist-groups* called $INPUT. All variables in *common* blocks /CONST/ and /FILES/ can be changed.

The two cases, a fresh start of the model and a restart of a 'frozen' run are treated separately:

**No frozen file**
(This is the case when variable LREAD is not given a value in the first *group*). The defaults for all variables are set by READC. The user must change all variables for which the default is not good in the first *group*. The second *group* must be left empty.

**Frozen file**
The filename for the frozen file (LREAD) must be specified in the first *group*, no other variables should be changed here. The defaults for all variables are read from the frozen file. The user must change all variables for which the default is not good in the second *group*.

**Output** file used in MAIN, RESULT, +others
First, a copy of the *namelist-group* $INPUT is written and after that, it is used as a log-file for the program. Many subroutines can write to this file, the diagnostic information is written from subroutine DIAGN.

File **LREAD** used in REREAD
If the filename in LREAD is not 'NONE' subroutine REREAD will read the frozen file LREAD. This file must be written by subroutine FRIEZE (see later) and contains six records:

1. a copy of array VX
2. a copy of array VY
3. a copy of array TMP
4. a copy of array SLT
5. a copy of array SURF
6. a copy of all variables in /CONST/ and /FILES/

This information is sufficient to resume the calculation from the point where the run was frozen.

File **LWRITE** used in FRIEZE
If the filename in LWRITE is not 'NONE' subroutine FRIEZE will write the information described above to file LWRITE.

File **LTMP** used in TMPINI
If the file name in LTMP is not 'NONE' subroutine TMPINI will read 13 records from file LTMP. The records should contain initial temperatures at the 13 program levels. Starting location is 149.5° East, −30.5° North, incrementing with 1° East 1° North. These data will be horizontally interpolated to the program grid. Alternative code is supplied for reading *packed* files.

File **LSLT** used in SLTINI
Identical to LTMP (see above), salinities in permille.

File **LMASK** used in MMASK
If the filename in LMASK is not 'NONE' subroutine MMASK will read a land-sea mask for the file LMASK. The mask is interpolated to the program grid and stored in array SEA(v 18.4) or SEAX and SEAY(v19.2). The coordinates are the same as used in LTMP and LSLT. The file should contain numbers less than 18.5 for land points and numbers greater than 18.5 for sea-points. (Normally the number of the lowest level in the levitus data is used, level 19 corresponds to a depth of 1000 meter.)

Alternative code is supplied for reading *packed* files.

File **LFSU** used in INITW
If the filename in LFSU is not 'NONE' subroutine INITW will open a random access file. This file contains in each record:

1. Sea surface temperature, [°C]
2. Wind-field (West to East) [m/s]
3. Wind-field (South to North) [m/s]

For each month in the period 1970-1983 there is one record. The last year contains climatological data.

Start 152°/−30°, increment 2°/2°. (wind field)
Start 152°/−29°, increment 2°/2°. (sst field)

If LFSU is not 'NONE' routine DAY computes the correct record numbers, reads those records, interpolates them to the program grid and then interpolates them to the correct fraction of the current month.

File **LPACK** used in PACK
At the end of the run some information about the state of the ocean is *packed* into file LPACK, for further post-processing. Any array with two dimensional data can be *packed* and send to the post processor. See the description of PACK for what information is *packed* at the current version.

File **DIAGN** used in DIAGN
This file contains diagnostic output. See the description of DIAGN for what information is supplied the current version.

Unit 14
Unit 14 is reserved for a random access file containing (computed) heatfluxes and fresh water fluxes.

Unit 15

Unit 15 is reserved for a random access file containing (computed) temperature anomalies.

Packed files

The packing of data in files is a technique used to send two- dimensional data from one computer to another in an efficient way. Every value is *packed* in two bytes, both containing 5 significant bits. A readable subset of the ascii character-set is used. Packing -and writing- is performed by subroutine PACKIN, reading and unpacking is done by subroutine PACKUT.

**WARNING:** The Unisys A-series computers use an internal ebcdic character-set and another version of PACKUT must be used on this computer.

## 3.5 Arrays

This chapter will describe all arrays occurring in the the common blocks, sorted per common block. More information about the grids and sub-grids used, see chapter 3.8. In this paper we will use the following abbreviations:

nx : the number of points in the West-East direction.
: (version 18.4: half this number)
ny : the number of points in the South-North direction.
nz : the number of levels (bottom → surface)

Common block // All real arrays.

* VX(nx,ny,nz) V-sub-grid [m/s]
  West → East water velocity.
* VXI(nx,ny,nz) T-sub-grid [m/s]
  VX horizontally interpolated to T-sub-grid.(V18.4 only)
* VY(nx,ny,nz) V-sub-grid [m/s]
  South → North water velocity.
* VYI(nx,ny,nz) T-sub-grid [m/s]
  VY horizontally interpolated to T-sub-grid.(V18.4 only)
* VZ(nx,ny,nz) Z-sub-grid [m/s]
  Bottom → Top water velocity.
* VZI(nx,ny,nz) R-sub-grid [m/s]
  VZ horizontally interpolated to R-sub-grid.(V18.4 only)
* VXT(nx,ny,nz) V-sub-grid [$m/s^2$]
  Acceleration, vx.
* VYT(nx,ny,nz) V-sub-grid [$m/s^2$]
  Acceleration, vy.
* TMP(nx,ny,nz) T-sub-grid [°C]
  Temperature.
* SLT(nx,ny,nz) T-sub-grid [permille]
  Salinity.
* PRS(nx,ny,nz) T-sub-grid [Pa]
  Pressure anomaly due to density anomalies.

(Surface elevation is not included here).

* RIF(nx,ny,nz) R-sub-grid [-]
  Richardson function.(v18.4 only)
* RFX(nx,ny,nz) [-]
  Richardson function.(v19.2 only)
* RFY(nx,ny,nz) [-]
  Richardson function.(v19.2 only)
* QQQ(nx,ny,nz) T-sub-grid
  general scratch array (contains density at some point).
* SEA(nx,ny) V-sub-grid [-]
  land/sea mask (land=0, sea=1)(v18.4 only)
* SEAX(nx,ny) U-sub-grid [-]
  land/sea mask (land=0, sea=1)(v19.2 only)
* SEAY(nx,ny) V-sun-grid [-]
  land/sea mask(land=0, sea=1)(v19.2 only)
* TSUR(nx,ny) T-sub-grid [°C]
  Atmosphere (or top layer-) temperature.
* TBOT(nx,ny) T-sub-grid [permille]
  Bottom (or lowest layer-) temperature.
* SSUR(nx,ny) T-sub-grid [°C]
  Top layer salinity.
* SBOT(nx,ny) T-sub-grid [permille]
  Lowest layer salinity.
* VX0(nx,ny) U-sub-grid [/s]
  Scratch array.
* VY0(nx,ny) V-sub-grid [/s]
  Scratch array.
* VX1(nx,ny) U-sub-grid [/s]
  VX, vertically integrated.
* VY1(nx,ny) V-sub-grid [/s]
  VY, vertically integrated.
* WINDX(nx,ny) U-sub-grid [m/s]
  wind velocity(W → E)
* WINDY(nx,ny) V-sub-grid [m/s]
  wind velocity(S → N)
* SURF(nx,ny) T-sub-grid [m]
  Surface elevation (0=sea-level)
* SUR1(nx,ny) T-sub-grid [m]
  Surface elevation, scratch.

Common block /MTRX/ All real arrays.

Used in the implicit step (IMPL2).

* PGL(nx*ny,2*nx) [-]
  Decomposed matrix.

* B(nx*ny) [m]
surface elevation vector, old.
* X(nx*ny) [m]
surface elevation vector, new.

Common block /INTER/One real array.
* AR(jx,jy)
Original fields (before interpolation)

Common block /COORD/ All real arrays, except MDY.
//////// version 18.4 ////////

| | | |
|---|---|---|
| * XX(nx*2) | [m] | x position grid-points. |
| * DX2(nx*2) | [m] | XX(i+1)-XX(i-1) |
| * OX2(nx*2) | [/m] | 1/DX2 |
| * DX1(nx*2) | [m] | XX(i+1)-XX(i) |
| * OX1(nx*2) | [/m] | 1/DX1 |
| * XG(nx*2) | [°] | x position in degrees (0=centre) |
| * YY(ny) | [m] | y position grid-points. |
| * DY2(ny) | [m] | YY(i+1)-YY(i-1) |
| * OY2(ny) | [/m] | 1/DY2 |
| * DY1(ny) | [m] | YY(i+1)-YY(i) |
| * OY1(ny) | [/m] | 1/DY1 |
| * YG(ny) | [°] | y position in degrees (0=equator) |
| * ZZ(nz*2) | [m] | height above bottom. |
| * DZV(nz) | [m] | distance between V-sub-grid levels. |
| * OZV(nz) | [/m] | 1/DZV |
| * DZZ(nz) | [m] | distance between Z-sub-grid levels. |
| * OZZ(nz) | [/m] | 1/DZZ |
| * DNE(nx,ny) | [/m] | Special arrays for ... |
| * DSE(nx,ny) | [/m] | ...horizontal diffusion. |
| * DNW(nx,ny) | [/m] | |
| * DSW(nx,ny) | [/m] | |
| * MDY(ny) | [-] | 1=odd / 0=even numbered row. |
| * FIN(ny) | [-] | $\gamma$ for coriolis force. |
| * FCR(ny) | [/s] | $f$ for coriolis force. |
| * BTF(ny) | [-] | $\tau\gamma f$ for coriolis force. |

Common block /COORD/ All real arrays.
//////// version 19.2 ////////

| | | |
|---|---|---|
| * XX(mx) | [m] | x coordinate, u,s sub-grids. |

| | | |
|---|---|---|
| * XG(nx) | [m] | position in degrees east. |
| * DX1S(nx) | [m] | XX(i*2)-XX(i*2-1) |
| * OX1S(nx) | [/m] | 1/DX1S |
| * DX1U(nx) | [m] | XX(i*2+1)-XX(i*2) |
| * OX1U(nx) | [/m] | 1/DX1U |
| * DX2S(nx) | [m] | XX(i*2)-XX(i*2-2) |
| * OX2S(nx) | [/m] | 1/DX2S |
| * DX2U(nx) | [m] | XX(i*2+1)-XX(i*2-1) |
| * OX2U(nx) | [/m] | 1/DX2U |
| * YY(my) | [m] | y coordinate, v,s sub-grids. |
| * YG(ny) | [m] | position in degrees north. |
| * DY1S(ny) | [m] | YY(i*2)-YY(i*2-1) |
| * OY1S(ny) | [/m] | 1/DY1S |
| * DY1V(ny) | [m] | YY(i*2+1)-YY(i*2) |
| * OY1V(ny) | [/m] | 1/DY1V |
| * DY2S(ny) | [m] | YY(i*2)-YY(i*2-2) |
| * OY2S(ny) | [/m] | 1/DY2S |
| * DY2V(ny) | [m] | YY(i*2+1)-YY(i*2-1) |
| * OY2V(ny) | [/m] | 1/DY2V |
| * ZZ(nz*2) | [m] | height above bottom. |
| * DZV(nz) | [m] | distance between V-sub-grid levels. |
| * OZV(nz) | [/m] | 1/DZV |
| * DZZ(nz) | [m] | distance between Z-sub-grid levels. |
| * OZZ(nz) | [/m] | 1/DZZ |
| * B1U(ny) | [-] | $\gamma$ for coriolis force.(U-sub-grid) |
| * B1V(ny) | [-] | $\gamma$ for coriolis force.(V-sub-grid) |
| * B2U(ny) | [-] | $\tau\gamma f$ for coriolis force.(U-sub-grid) |
| * B2V(ny) | [-] | $\tau\gamma f$ for coriolis force.(V-sub-grid) |
| * F3U(ny) | [/s] | $f$ for coriolis force.(U-sub-grid) |
| * F3V(ny) | [/s] | $f$ for coriolis force.(V-sub-grid) |

## 3.6 Constants

Common block /FILES/ (All type character*40)

| | |
|---|---|
| * LREAD | Filename of frozen file to be read;<br>Default = 'NONE', no file read. |
| * LWRITE | Filename of frozen file to be written;<br>Default = 'NONE', no file written. |
| * LMASK | Filename of file containing land/sea mask.<br>Default = 'NONE', no file read. |
| * LFSU | Filename of file containing SST/wind-fields;<br>Default = 'NONE', no file read. |
| * LTMP | Filename of file containing initial sea water temps. |

Default = 'NONE', no file read.

* LSLT — Filename of file containing initial salinities.
Default = 'NONE', no file read.

* LPACK — Filename of file containing packed results.
Default = 'NONE', no file written.

| Common block /CONST/ | | r=real, i=integer. | | | Used in |
|---|---|---|---|---|---|
| * OMEGA | $\Omega$ | r | $7.292 \cdot 10^{-05}$ | [/s] | CCOORD |
| | | Angular velocity of earth. | | | |
| * REARTH | $R_0$ | r | $6.378 \cdot 10^{+06}$ | [m] | CCOORD |
| | | Equatorial radius of earth. | | | |
| * PI | $\pi$ | r | 3.1415... | [-] | CCOORD |
| | | Guess what. | | | |
| * GRAV | $g$ | r | 9.782 | [m/s$^2$] | CCOORD,DELTAV, MATRIX |
| | | Gravitational acceleration near equator. | | | |
| * RHO | $\rho_0$ | r | 1025.0 | [Kg/m$^3$] | (many) |
| | | Average sea-water density. | | | |
| | | (Correct for $T \sim 17°C$,$S \sim 35$,$P \sim 0$) | | | |
| * CP | $C_p$ | r | 4000.0 | [J/Kg/K] | DIAGN |
| | | Specific heat of sea water. | | | |
| * RFV0 | $\nu_v$ | r | $2.0 \cdot 10^{-05}$ | [m$^2$/s] | MOMVD |
| * RFV1 | | r | $5.0 \cdot 10^{-04}$ | [m$^2$/s] | MOMVD |
| * RFV2 | | r | $6.4 \cdot 10^{+02}$ | [m$^2$/s] | MOMVD |
| * RFVM | | r | 0.53 | [- ] | MOMVD |
| | | Three constants in vertical viscosity. | | | |
| * RFT0 | $\kappa_v$ | r | $1.0 \cdot 10^{-06}$ | [m$^2$/s] | TMPVD |
| * RFT1 | | r | $5.0 \cdot 10^{-04}$ | [m$^2$/s] | TMPVD |
| * RFT2 | | r | $8.0 \cdot 10^{+01}$ | [m$^2$/s] | TMPVD |
| * RFTM | | r | 0.60 | [- ] | TMPVD |
| | | Three constants in vertical conductivity. | | | |
| * AH | $\nu_h$ | r | 5000.0 | [m$^2$/s] | MOMHD,TMPVD |
| | | Hor. diffusion constant, (v19.2 only). | | | |
| * AHC | $\nu_h$ | r | 5000.0 | [m$^2$/s] | MOMHD,TMPHD |
| | | Hor. diffusion constant, C-grid.(v18.4 only) | | | |
| * AHE | $\nu_h$ | r | 5000.0 | [m$^2$/s] | MOMHD,TMPHD |
| | | Hor. diffusion constant, E-grid.(v18.4 only) | | | |
| * ASUR | $A_s$ | r | $1.0 \cdot 10^{-04}$ | [m/s] | TMPVD |
| | | Newtonian heat exchange rate (surface). | | | |
| * ABOT | $A_b$ | r | $1.0 \cdot 10^{-02}$ | [m/s] | TMPVD |
| | | Newtonian heat exchange rate (bottom). | | | |
| * T0 | $T_0$ | r | $-7.73508$ | [°C] | DENS |
| * GAMMA | $\gamma$ | r | $-0.00471$ | [Kg/m$^3$/K$^2$] | DENS |

| | | | | | |
|---|---|---|---|---|---|
| | | Two constants describing the density as function of the temperature as a quadratic function. Correct for salinity=35 permille. | | | |
| * BETA | $\beta$ | r | +0.77390 | [Kg/m$^3$/prom] | DENS |
| | | Coefficient of the salinity in the equ. of state. | | | |
| * RHM | $\rho_m$ | r | $-1.0 \cdot 10^{-04}$ | [Kg/m$^3$/m] | MOMHD,TMPHD |
| | | minimum density gradient used in calculation of the richardson function. | | | |
| * RHOAIR | $\rho_a$ | r | 1.2 | [Kg/m$^3$] | MOMVD |
| | | Density of air, used in wind-stress. | | | |
| * SIGMA2 | $4\Sigma^2$ | r | 36.0 | [m/s] | MOMVD |
| | | Extra wind speed to correct for unresolved wind-speed variability ($\Sigma$=3m/s). | | | |
| * CD | $C_d$ | r | 0.0012 | [-] | MOMVD |
| | | Drag coefficient. | | | |
| * LCLIM | | i | 1 | [-] | DAY |
| | | Climatology/actual value switch. | | | |
| * NFST | | i | 1 | [-] | MAIN,REREAD |
| | | Number of the first iteration. | | | |
| * NSTP | | i | 400 | [-] | MAIN,REREAD |
| | | Number of last iteration. | | | |
| * NPRT | | i | 10 | [-] | MAIN,RESULT |
| | | Every NPRT iterations diagnostic output is made. | | | |
| * DT | $\tau$ | r | 7200.0 | [s] | (many) |
| | | Length of time-step. | | | |
| * A0 | $\alpha$ | r | 0.5 | [-] | CCOORD,DELTAV |
| | | Interpolation entry for coriolis force. Specify between 0 and 0.5. | | | |
| * B0 | $\beta$ | r | 0.4 | [-] | DELTAV,EXPLCT |
| | | Interpolation entry for gravity waves. Specify between 0 and 0.5. | | | |
| * CHECK | | r | $\pi$ | [-] | REREAD |
| | | Checksum for frozen file. | | | |

## 3.7 Description of the subroutines

In the following, we refer with 'loop.nn' (nn an integer) to a FORTRAN DO loop, or DO loop nest, associated with statement label nr. nn in the source text.

### 3.7.1 Program MAIN

From here most subprograms are called. The order in which the subprograms are called is significant. We can distinguish between three parts: the initialization part, the stepping part, and the freeze part.

Initialization part.
All variables in /CONST/ and /FILES/ are set to their defaults (READC).

A namelist-group is read to see if this is an initial run or a continued run.
In the latter case, REREAD is called to read de values in /CONST/ and /FILES/ from a previous run.
Then subroutine CCOORD is called to create the coordinates in /COORD/.
The sea is initialized to a zero state. (MMASK, TMPINI, SLTINI are called, arrays VX, VY, SURF, SUR1, are set to zero).
INITW is called to initialize the wind field file.
Now REREAD is called again to read the current state.
Another namelist-group is read, so the user can change some of the variables in /CONST/ and/or FILES. (Warning: do not change DT, as it has already been used!)
Now the namelist-group is printed.
And PGL is initialized in MATRIX.
De density (QQQ), and richardson function (RIF, or RFX and RFY) are computed (DENS).
The variable LAST is set to a ridiculous value, so in the first step the wind-fields will be re-interpolated.

Stepping part. loop.11
Here a number of subroutines is called in a cyclic way to perform a (large) number of time-steps. At the end of a run this cycle is interrupted at the point where we need to store the least information to resume the calculations.
First, a new value for the height of the sea level is computed in HEIGHT.
At this point, sometimes (depending on NPRT) diagnostic output is written to files OUTPUT and DIAGN.(DIAGN,RESULT)
And again sometimes (at the start of a new day) new wind-fields are calculated (DAY).
Then the temperature equation is handled. (TMPHD, TMPVD, TMPADV, TMPE)
Then the same calculations are done for the salinity (SLTHD, SLTVD, SLTADV, SLTE).
The density (QQQ) the pressure (PRS) and the richardson functions (RIF, or RFX and RFY) are calculated from the new temperatures and salinities (DENS).
Now the explicit part of the momentum equations is done.
(MOMHD, MOMVD, VXADV, VYADV, DELTAV) (vertical diffusion, horizontal diffusion, advection, pressure)
The implicit step is called to compute new VX, VY, values (IMPL1, IMPL2, EXPLCT).
The computation of VZ and SURF are done in HEIGHT, at the start of this loop.

Freeze part.
At the end of the run two more things need to be done.
FRIEZE is called to write the current state of the ocean to file LWRITE, so that calculations can resume in a subsequent run.
PACK is called to write some data to file LPACK, that can be processed by a post-processor program.

### 3.7.2 Subroutine READC

This subroutine initialises a number of constants to a preset default value. De constants and their values are tablulated in section 3.6.

### 3.7.3 Subroutine CCOORD

This subroutine sets up the coordinates of the program in common block /COORD/ DXKM are the distances between grid-points (x-direction) in km.
The last values near the centre of the ocean (sometimes referred to as the "dateline" (positioned at X0G).
DYKM are the distances between grid-points (y-direction) in km.
The last values near the equator (positioned at Y0G=0).
ZM are the depth values (in m) of the layers in the model.
The first value is the deepest, start of first v-sub-grid layer,
The second value is the start of the first t-sub-grid layer.
DOM is the factor used to calculate m's from km's.
X0G is the position of the centre of the ocean (degrees east).
Y0G is the position of the centre of the ocean (degrees north).
CIRCLE is the number of degrees in a full circle.

loop.10: X (west-east) positions in meters, the space derivatives, and the position in degrees from X0G.
loop.20: Y (south-north) positions in meters, the space derivatives, and the position in degrees from Y0G.
loop.25: Coriolis force related functions.
loop.30: Z (bottom-surface) position in meters and its space derivatives.
loop.40: Special functions for horizontal diffusion. (v18.4 only)

### 3.7.4 Subroutine MATRIX

This subroutine sets up the matrix PGL in common block /MTRX/ See equation 2.33.
From the second index of PGL, the first is subtracted, so that the diagonal elements have second index 0. Because only the near-diagonal elements are nonzero, we do not have to store most of the array.

loop.20: Clear array PGL.
loop.30: Creates the matrix in PGL.
loop.50: L-U decomposition of matrix PGL.

### 3.7.5 Subroutine ZERO

An efficient routine to set a multi dimensional array to zero. In the next version of FORTRAN this will be a single statement.

| Parameters : | | in/out | |
|---|---|---|---|
| 1 | X(N1,N2,N3) | o | Array involved. |
| 2 | N1 | i | First dimension. |
| 3 | N2 | i | Second dimension, if none, specify 1. |
| 4 | N3 | i | Third dimension, if none, specify 1. |

loop.11: Here all the work is done.

### 3.7.6 Subroutine HEIGHT

VZ is computed from VX and VY, with the continuity equation 2.4.
VX, VY, VZ are horizontally interpolated to the other sub-grid
(VXI, VYI, VZI) in version 18.4. Bi-linear interpolation is used.
Then a new value for the surface elevation is computed 2.8.

loop.10: Calculation of VZ with de continuity equation.
loop.30: VZ is interpolated to VZI.
loop.40: VX is interpolated to VXI.
loop.50: VY is interpolated to VYI.
loop.60: SURF is calculated from VZ surface.

### 3.7.7 Subroutine TMPHD

Horizontal temperature diffusion.
QQQ is used here to store the time derivative of TMP.
The equations used are 2.46, 2.48. The only complication is the land-sea mask.
We want the ocean to be thermally isolated from the "walls" around it, and so we must multiply every heat-flow with the land-sea mask. The implementation of this feature is somewhat arbitrary.
Temperature is conserved.

loop.10: C-grid diffusion, (east/west and north/south).
loop.20: E-grid diffusion, (ne/se and nw/sw). (v18.4 only)

### 3.7.8 Subroutine TMPVD

Vertical temperature diffusion.
QQQ is now the time derivative of TMP.
Near the surface and the bottom a Newtonian heating/cooling is computed (2.62, 2.63, 2.65), with a fixed coefficient (ASUR, ABOT).
A Haney forcing can be simulated by specifying the correct value for ASUR.
Temperature is conserved.
From RIF, or RFX and RFY AK ($\kappa$) is computed (2.50), and Richardson number dependent mixing is done (2.53).

loop.10: Surface heat-exchange.
loop.20: Bottom heat-exchange.
loop.30: Vertical diffusion, Richardson number dependent.

### 3.7.9 Subroutine TMPADV

Temperature advection.
See note on QQQ above.
The advection scheme is well documented by 2.46.
Temperature is transferred from one cell to the adjacent cells, and so conservation of temperature is assured.

loop.10: $\partial T/\partial x * v_x$
loop.20: $\partial T/\partial y * v_y$

| | | |
|---|---|---|
| loop.30: | $\partial T/\partial z * v_z$ | Body |
| loop.40: | $\partial T/\partial z * v_z$ | Bottom |
| loop.50: | $\partial T/\partial z * v_z$ | Surface |

### 3.7.10 Subroutine TMPE

Computes new temperatures from the derivatives accumulated in QQQ.

loop.30: Array TMP is updated.

### 3.7.11 Subroutine SLTHD

### 3.7.12 Subroutine SLTVD

### 3.7.13 Subroutine SLTADV

### 3.7.14 Subroutine SLTE

Salinity calculations.
Completely identical to the temperature equations.
One can consider writing a general advection/diffusion equation solver, this would make the program a little shorter, but I feel, would damage the simplicity of the current program.

### 3.7.15 Subroutine DENS

Density and Pressure, and Richardson function.
First the density anomaly is calculated from the temperature and the salinity with the equation of state (2.68). Anomaly means the value of $\rho_0$ is not included. Warning: $\rho_0$ is about, but not by definition 1000.

Then the pressure anomaly is calculated from the density anomaly. This pressure can only be used to calculate its horizontal derivatives. The absolute value, as well as its vertical derivative have no meaning. The pressure due to the surface elevation (SURF) is not included.
The Richardson function is calculated (2.48,2.49).

loop.110: Density is calculated.
loop.210: Surface layer pressure (assuming constant density).
loop.220: Pressure other layers, (density interpolated).
loop.310: Richardson number, RFX or RIF.
loop.410: Richardson number, RFY.(v19.2 only)

### 3.7.16 Subroutine MOMHD

Momentum horizontal diffusion.
See notes by TMPHD. No problems with the land-sea mask here, momentum is allowed to flow freely through the "walls".
The results (an accelleration) are stored in VXT, VYT.

loop.10: C-grid diffusion, VX.
loop.20: C-grid diffusion, VY.
loop.30: E-grid diffusion, VX. (v18.4 only)
loop.40: E-grid diffusion, VY.(v18.4 only)

### 3.7.17 Subroutine MOMVD

Momentum vertical diffusion.
Wind-stress is calculated by 2.52-2.57.
Bottom friction with 2.58, 2.59.
The results are added to VXT, VYT.

loop.100: Richardson mixing.
loop.300: Wind stress.
loop.400: Bottom friction.

### 3.7.18 Subroutine VXADV

### 3.7.19 Subroutine VYADV

Momentum advection.
The advected quantity is the momentum itself (or better, the velocity, density is assumed to be constant) VX and VY.
The advecting speed is the horizontally interpolated speed (VXI,VYI). Other definitions of VXI and VYI may lead to instabilities in the longer term (E-grid waves). I think conservation of momentum is only guaranteed if VXI, VYI, VZI obey the continuity equation, and it is not clear that they do.
Time derivatives are collected in VXT and VYT.

| | | |
|---|---|---|
| loop.10: | $\partial v/\partial x * v_x$ | |
| loop.20: | $\partial v/\partial y * v_y$ | |
| loop.30: | $\partial v/\partial z * v_z$ | Body |
| loop.40: | $\partial v/\partial z * v_z$ | Bottom layer |
| loop.50: | $\partial v/\partial z * v_z$ | Surface layer |

### 3.7.20 Subroutine DELTAV

Pressure forces.
Forces due to pressure (PRS) gradients are calculated here.
Then the explicit pressure gradient due to the surface elevation SURF, also added, and so is the explicit coriolis force. Now new speeds are calculated from the time derivatives VXT, VYT. At this point we completed calculation of $u^+$ and $v^+$ in (2.23,2.24).
At the end of this routine VXY and VYT contain the new velocity values.

loop.100: Pressure gradient. (PRS)
loop.200: Explicit surface pressure. ($\beta$)
loop.300: Multiply with land-sea mask.
loop.400: Explicit coriolis force. ($\alpha$)
loop.500: Multiply with the land-sea mask and time step; add old values.

### 3.7.21 Subroutine IMPL1

The implicit step, part one.
In fact, equation 2.23 is solved here. In order to make computations a little easier integration and

differentiation are interchanged in 2.23.
$S^e$ is stored in SUR1.

zero: Initialize VX1 and VY1, before integration.
loop.10: Horizontal integration of VXT and VYT.
loop.30: Coriolis rotation. $(1 - \alpha)$
loop.40: Take gradient.

### 3.7.22 Subroutine IMPL2

Implicit step, part two.
Equation 2.27 is solved here, with the method described in section 2.8.
$S^n$ and $S^e$ are stored in SUR1.

loop.10: B is collected from SUR1.
loop.20: Multiply by lower triangle 2.42.
loop.30: Multiply by upper triangle 2.43.
loop.40: Scatter result to SUR1.

### 3.7.23 Subroutine EXPLCT

Implicit step, part three.
Maybe the name of this subroutine is a little confusing.
The new velocities are calculated 2.21, 2.22. To clarify the subroutine it is split into two parts, the first part calculates the changes in velocities due to the surface elevation (SUR1), the second part handles the coriolis force.

loop.100: Surface pressure gradient $(1 - \beta)$
loop.200: Coriolis force $(1 - \alpha)$

### 3.7.24 Subroutine DIAGN

This subroutine is called every NSTP steps to print some diagnostic information.
Examples of such output, for two spinup experiments (40 year 'climatological' forcing followed by forcing with 14 year observed data), with- and without 'active' salinity, are shown in graphical representation in figures 1 – 6. For this purpose the South equatorial- Equatorial under- and North equatorial countercurrents are defined as eastward flows faster than 5 cm/s between 24°S and 3°S, 3°S and 3°N, 3°N and 24°N respectively.

Parameter:

1 ISTP i Step number

Output (printed)

| pos | loop | var. | value. |
|---|---|---|---|
| 1 | - | ISTP | Step number. |
| 2 | 110 | RMS | RMS of the sea-level (tilting of the ocean). |
| 3 | 210 | ENK | Kinetic energy near the equator. |

| | | | |
|---|---|---|---|
| 4 | 310 | H18 | Average height of the 18°C isotherm |
| 5 | 320 | TAV | Average temperature above 1050 meter. |
| 6 | 410 | Q1 | Average surface heat-flux. |
| 7 | 420 | Q3 | Average bottom heat-flux. |
| 8 | 510 | CUR1 | South equatorial current flux. |
| 9 | 520 | CUR2 | Equatorial undercurrent flux |
| 10 | 530 | CUR3 | North equatorial counter current flux. |

### 3.7.25 Subroutine RESULT

Alternative for DIAGN.
Development aid.

### 3.7.26 Subroutine PTOT

Print part of a 2 dimensional staggered field.
Development aid.

### 3.7.27 Subroutine MMASK

Initializes the land-sea-mask
The land-sea mask is a real array containing 0.0 at land points, and 1.0 at sea points.
In version 18.4 there is one land-sea mask (SEA) located at the v-sub-grid. and one located at the v-sub-grid. If LMASK is 'NONE' only the "walls" at the edges of the grid are masked out, a rectangular ocean results, and the message 'FAKING LAND-SEA MASK' is printed.
If LMASK is not 'NONE', from the file LMASK ocean depth values from the LEVITUS data-set are read. All points deeper than 1000 meter (LEVITUS level 19) are considered sea-points, all others land-points.
X0,Y0 is the coordinate of the first grid-point in the data-file.
XI,YI are increment values, all in degrees.
Interpolation to the program grid is performed in subroutine INTERP. The data is read and unpacked in subroutine PACKUT, this also prints a message.

| | |
|---|---|
| zero: | Set all points to 0.0 |
| loop.10: | Set all points, except the walls, to 1.0 |
| loop.30: | Set land-points to 0.0 (SEA or SEAX) |
| loop.40: | Set land points to 0.0 (SEAY, v19.2 only) |

### 3.7.28 Subroutine INITW

Initializes SST and WIND FIELD data.
If LFSU is 'NONE' this routine generates a simple model wind field in arrays WINDX and WINDY. The message 'FAKING WINDFIELD' is produced. If LFSU contains a filename, this file is assumed to be a random access file containing observed SSTs and winds. This file is opened to unit 10. A message specifying the opened file is produced.
The file is read in routine DAY.

### 3.7.29 Subroutine DAY

If LFSU is 'NONE', this subroutine performs no function at all. The values in array TSUR are initialized in TMPINI, and not changed again. The values in WINDX and WINDY are initialized in INITW, and not changed either.

If LFSU is not 'NONE', new wind fields and SSTs are computed. The parameter of this subroutine contains a day number. From this day number two record numbers are computed, and two interpolation constants.
If the two records needed are not yet in memory, they are read from the random file, all data is interpolated to the program grid.
Finally all data is interpolated to the correct day. It now seems that reinterpolating the wind field once a day is not good enough, future versions may do this every time step.

loop.10: Linear interpolation to specified date.

### 3.7.30 Subroutine TMPINI

Read the initial temperature values.
If LTMP is 'NONE', the message 'FAKING INITIAL TEMPERATURES' is printed, and average values for the temperature at each level are used (array TZ). Otherwise, a message specifying the name of the file is produced, and 13 levels LEVITUS temperatures are read from file LTMP, and interpolated (INTERP) to the program grid.

So this system works in the least obvious way, vertical interpolation is done elsewhere, (to create 13 levels), horizontal interpolation is done here.
X0, Y0 are the coordinates of the first (south-west) grid-point in the data-file.
XI, YI are the increments, all in degrees.

Independent of the value of LTMP, the initial surface (TSUR) and bottom (TBOT) temperatures are copied from the highest and lowest layer. TSUR can be modified by DAY, TBOT remains unchanged.

loop.10: Data read and interpolated.
loop.20: Faked initial temperatures are copied.
loop.50: TSUR and TBOT are given a value.

### 3.7.31 Subroutine SLTINI

Reads the initial salinity values.
See TMPINI, above. (variable=LSLT).
In this case SBOT as well as SSUR remain unchanged during program execution.

loop.10: Data read and interpolated.
loop.20: Salinity values are all set to 35.0.
loop.50: SSUR and SBOT are given a value.

### 3.7.32 Subroutine INTERP

This subroutine interpolates external fields to the program grid.

parameters:

| | | |
|---|---|---|
| 1 | A1(JX,JY) | Input array, grid-points at X0+i*XI, 1 < i < JX.<br>grid-points at Y0+j*YI, 1 < j < JY |
| 2 | A2(MX,NY) | Output array. |
| 3 | JX | See above. |
| 4 | JY | See above. |
| 5 | M | Staggering, 0=T-sub-grid, 1=V-sub-grid. |
| 6 | X0 | West-most X-coordinate, degrees. |
| 7 | XI | Increment X-coordinate, degrees. |
| 8 | Y0 | South-most Y-coordinate, degrees. |
| 9 | YI | Increment Y-coordinate, degrees. |

MX and NY are known (parameter statements in the common-deck).
Before the bi-linear interpolation is done error checking is performed on X0, XI, Y0, YI. If one of them is incorrect, the message 'INTERPOLATION ERROR', is printed, and the program stopped. The coordinate arrays XG and YG are used here, so be sure READC is called before this subroutine.

loop.10: Remove landpoints from data (CRAY only).
loop.20: Bi-linear interpolation.

### 3.7.33 Subroutine FRIEZE

This subroutine freezes the ocean dynamics and stores all necessary information to continue the program on file LWRITE.

This information is:

VX, VY, TMP, SLT, SURF, and /CONST/

A message is printed.
The name of this subroutine may seem odd to you, but "freeze" is not allowed on the UNISYS A-series computers.

### 3.7.34 Subroutine REREAD

This subroutine reads the information from a file created by FRIEZE (above) to enable continued calculation in separate runs. All information is copied back to the original locations,except the filenames LREAD, LWRITE and LPACK, which remain unchanged.

Also the contents for variable CHECK is tested. If not correct, the program is stopped.

### 3.7.35 Subroutine PACK

This subroutine packs some data to file LPACK. This data can be send to another computer for post-processing.
If LPACK is 'NONE' this subroutine performs no function.

Data Currently packed :

| rec | loop | array | value. |
|---|---|---|---|
| 1 | 12 | SURF | Surface elevation. |
| 2 | 21 | TMP | Temperature-equator. |
| 3 | 23 | TMP | Temperature-dateline. |
| 4 | 25 | TMP | Temperature-surface. |
| 5 | 31 | VX | West-Eastvelocity-equator. |
| 6 | 33 | VX | West-Eastvelocity-dateline. |
| 7 | 35 | VX | West-Eastvelocity-surface. |
| 8 | 61 | VX | West-Eastflux integrated in Z direction. |
| 9 | 61 | VY | South-Northflux integrated in Z direction. |
| 10 | 64 | VX | West-Eastflux integrated in Y direction. |
| 11 | 64 | VZ | Bottom-Surfaceflux integrated in Y direction. |
| 12 | 67 | VY | South-northflux integrated in X direction. |
| 13 | 67 | VZ | Bottom-Surfaceflux integrated in X direction. |
| 14 | 71 | RIF | Richardson function, equator. |
| 15 | 73 | RIF | Richardson function, dateline. |
| 16 | 81 | TMP | Depth of 18° isotherm. |

The array AR (in /INTER/) is used as scratch array.

### 3.7.36 Subroutine PACKUT

Reads a file (UNIT=1) with packed information, unpacks the data scales it and copies it to a specified two dimensional array.
For every call this subroutine prints one line output.

| | parameters: | in/out | |
|---|---|---|---|
| 1 | AR(NXMAX,NYMAX) | o | Array involved. |
| 2 | NXMAX | i | X-dimension of array AR. |
| 3 | NYMAX | i | Y-dimension of array AR. |
| 4 | NX | o | Number of points read (X). |
| 5 | NY | o | Number of points read (Y). |
| 6 | ILEV | o | See PACKIN. |
| 7 | IPH | o | See PACKIN. |
| 8 | TITH | o | Field identification. |
| 9 | IERR | o | 0 - no error.<br>1 - end of file.<br>2 - unexpected end of file.<br>3 - data header error.<br>4 -Arrays too small. |

loop.11: Unpacking + scaling.

### 3.7.37 Subroutine PACKIN

Packs and writes in formation from a specified two dimensional

array to a file with UNIT=2.
For every call this routine prints one line of output.

| | parameters: | in/out | |
|---|---|---|---|
| 1 | AR(IXDIM,IYDIM) | i | Array involved. |
| 2 | IXDIM | i | True X-dimension of array AR. |
| 3 | IYDIM | i | True Y-dimension of array AR. |
| 4 | NX | i | Number of points to be written (X). |
| 5 | NY | i | Number of points to be written (Y). |
| 6 | ILEV | i | Any integer. |
| 7 | IPH | i | Any integer. |
| 8 | TITH | i | Field identification. |

loop.10: Scaling of data.
loop.20: Packing.

## 3.8 Grids and Sub-grids.

The Arakawa E-grid is used in version 18.4 of the program. It can be seen as two overlaid Arakawa C-grids, so two more or less independent solutions to the equations exist. The grids are coupled by the "E-grid" loops in the horizontal diffusion.

There is no time-staggering, so the grids are completely identical in every time-step.

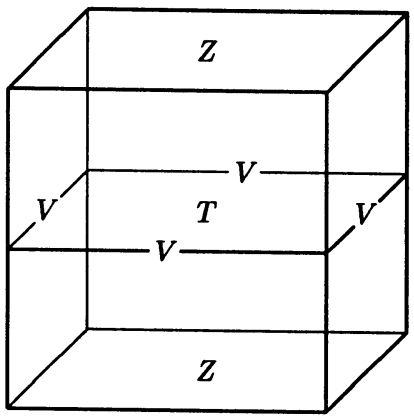
Four so called sub-grids exist. Points located on the same sub-grid are defined on exactly the same locations in space. Points located on different sub-grids are defined on different locations in space.

| | | |
|---|---|---|
| V-sub-grid | 3-dim arrays | VX, VY, VXT, VYT. |
| | 2-dim arrays | VX0, VX1, VY0, VY1, WINDX, WINDY. |
| T-sub-grid | 3-dim arrays | TMP, PRS, SLT, QQQ, VXI, VYI. |
| | 2- dim arrays | TSUR, TBOT, SSUR, SBOT. |
| Z-sub-grid | 3-dim array | VZ. |
| R-sub-grid | 3-dim array | RIF/VZI. |

The V-sub-grid and the R sub-grid only differ in (Z) levels, so the 2-dimensional arrays belong to both.

The T-sub-grid and the Z sub-grid also only differ in level.

The position of the sub-grids with respect to each other can easily be seen in the following drawing. The box represents part of the ocean. The T-sub-grid points are located at the centre of the box, and represent the value for the whole box. The V-sub-grid points are located at centre of the vertical edges, and represent fluxes in and out of the box. The Z-sub-grid points are located at the centre of the top and the bottom, and represent vertical fluxes.



## Horizontal cross-section of the Arakawa E-grid.

In the figure below, V,T represent sub-grids.

| IY/IM | | | | | | |
|---|---|---|---|---|---|---|
| 7/1 | $T_{1,7}$ | $V_{1,7}$ | $T_{2,7}$ | $V_{2,7}$ | $T_{3,7}$ | $V_{3,7}$ |
| 6/0 | $V_{1,6}$ | $T_{1,6}$ | $V_{2,6}$ | $T_{2,6}$ | $V_{3,6}$ | $T_{3,6}$ |
| 5/1 | $T_{1,5}$ | $V_{1,5}$ | $T_{2,5}$ | $V_{2,5}$ | $T_{3,5}$ | $V_{3,5}$ |
| 4/0 | $V_{1,4}$ | $T_{1,4}$ | $V_{2,4}$ | $T_{2,4}$ | $V_{3,4}$ | $T_{3,4}$ |
| 3/1 | $T_{1,3}$ | $V_{1,3}$ | $T_{2,3}$ | $V_{2,3}$ | $T_{3,3}$ | $V_{3,3}$ |
| 2/0 | $V_{1,2}$ | $T_{1,2}$ | $V_{2,2}$ | $T_{2,2}$ | $V_{3,2}$ | $T_{3,2}$ |
| 1/1 | $T_{1,1}$ | $V_{1,1}$ | $T_{2,1}$ | $V_{2,1}$ | $T_{3,1}$ | $V_{3,1}$ |
| IX | 1 | 1 | 2 | 2 | 3 | 3 |
| II | 1 | 2 | 3 | 4 | 5 | 6 |

This is a part of the ocean containing 6 * 7 = 42 grid-points
3 dimensional arrays are stored at only half the number of grid-points, so we have 21 T-points.
For symmetry reasons the right-most grid-column is never used, and the number of points in the y-direction must be odd, so the corners of the used grid are always T-points.

Formulas for computation in the grid:

IM = MOD(IY,1)  1 = odd rows, 0 = even rows.

T-sub-grid and Z-sub-grid:
real position: XG(IX*2-IM)
V-point to the east: V(IX-IM+1,IY,IZ)
V-point to the west: V(IX-IM ,IY,IZ)

V- sub-grid and R-sub-grid:
real position: XG(IX*2+IM-1)
T-point to the east: T(IX+IM ,IY,IZ)
T-point to the west: T(IX+IM-1,IY,IZ)

## Horizontal cross-section of the Arakawa C-grid.

In the figure below, U,V,T represent sub-grids.

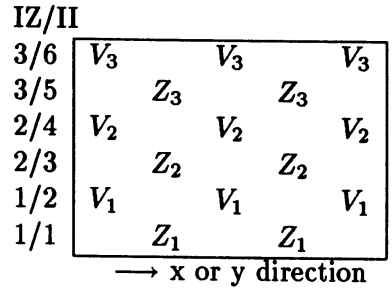
| IY/II | | | | | | |
|---|---|---|---|---|---|---|
| 4/7 | $T_{1,4}$ | $U_{1,4}$ | $T_{2,4}$ | $U_{2,4}$ | $T_{3,4}$ | $U_{3,4}$ |
| 3/6 | $V_{1,3}$ | | $V_{2,3}$ | | $V_{3,3}$ | |
| 3/5 | $T_{1,3}$ | $U_{1,3}$ | $T_{2,3}$ | $U_{2,3}$ | $T_{3,3}$ | $U_{3,3}$ |
| 2/4 | $V_{1,2}$ | | $V_{2,2}$ | | $V_{3,2}$ | |
| 2/3 | $T_{1,2}$ | $U_{1,2}$ | $T_{2,2}$ | $U_{2,2}$ | $T_{3,2}$ | $U_{3,2}$ |
| 1/2 | $V_{1,1}$ | | $V_{2,1}$ | | $V_{3,1}$ | |
| 1/1 | $T_{1,1}$ | $U_{1,1}$ | $T_{2,1}$ | $U_{2,1}$ | $T_{3,1}$ | $U_{3,1}$ |
| IX | 1 | 1 | 2 | 2 | 3 | 3 |
| II | 1 | 2 | 3 | 4 | 5 | 6 |

For symmetry reasons the right-most grid-column (U-points only), and the top column (not indicated, V-points only) are not used, so the corners of the used grid are always T-points.

DX1S(1) gives the distance between $U_{1,1}$ and $T_{1,1}$.
DX1U(1) gives the distance between $T_{2,1}$ and $U_{1,1}$.
DX2S(2) gives the distance between $U_{2,1}$ and $U_{1,1}$.
DX2U(1) gives the distance between $T_{2,1}$ and $T_{1,1}$.

## Vertical cross section.

In the figure below, V,Z represent sub-grids.




V-sub-grid and T-sub-grid:
Real depth: ZZ(IZ*2+1)
Layer thickness: DZV(IZ) (from V(IZ+1) to V(IZ))

Z-sub-grid and R-sub-grid:
Real depth: ZZ(IZ*2)
Layer thickness: DZZ(IZ) (from Z(IZ+1) TO Z(IZ))

Note that the $Z_1$ values are always zero.

The Z-sub-grid counts from 1(dummy) to NZ(surface)
The V-sub-grid counts from 1(real) to NZ-1, the NZ value is located at the surface and unused.

TABLE I

Symbols used:

| | | |
|---|---|---|
| $u$ | Water velocity x-direction (west $\rightarrow$ east) | [m/s] |
| $v$ | Water velocity y-direction (south $\rightarrow$ north) | [m/s] |
| $w$ | Water velocity z-direction (bottom $\rightarrow$ surface) | [m/s] |
| $\vec{u}$ | Three dimensional water velocity | [m/s] |
| T | Temperature | [°C] |
| S | Salinity | [permille] |
| s | Surface elevation relative to $D$ | [m] |
| $\vec{g}$ | Gravity acceleration | [m/s$^2$] |
| $\rho$ | Density | [kg/m$^3$] |
| $\tau$ | Timestep | [s] |
| $\vec{\sigma}$ | Windstress or bottom friction vector | [Pa] |
| $\Sigma$ | Pseudo windstress due to variability | [m$^2$/s$^2$] |
| P | Total pressure | [Pa] |
| $p$ | Pressure relative to $D$ | [Pa] |
| $R_0$ | Earth radius | [m] |
| $\Omega$ | Earth rotation | [s$^{-1}$] |
| $f$ | $2\Omega \sin(y/R_0)$ | [s$^{-1}$] |
| $\gamma$ | $[1+\tau^2 f^2(1-\alpha)^2]^{-1}$ | [-] |
| $R_i$ | Richardson number | [-] |
| $R_f$ | Richardson function | [-] |
| $\kappa$ | Eddy coëfficient for diffusivity | [m$^2$/s] |
| $\nu$ | Eddy coëfficient for viscosity | [m$^2$/s] |
| $D$ | Total depth | [m] |
| $\mathcal{A}$ | Thermal expansion parameter | [kg/m$^3$K$^2$] |
| $\mathcal{B}$ | Salinity expansion coëfficient | [kg/m$^3$permille] |
| $A$ | Coupling constant Newtonian forcing scheme | [m/s] |
| $C_p$ | Heat capacity of oceanwater | [J/kg/K] |
| $C_D$ | Drag coëfficient | [-] |

# References

Arakawa,A., 1966: J.Comput.Phys., 1, 119-143.

Forsythe,G.E., Mole,C.B., 1967: "Computer Solution of Linear Algebraic Systems", (Engewood Cliffs, N.J.: Prentice Hall).

Gill,A.E., 1982: "Atmosphere-Ocean Dynamics", Academic Press, NY.

Gill,A.E., 1986: "The relationship between surface wind and surface stress", Ocean Modelling, 67, 9. -Unpublished Manuscript-

Haney,R.L., 1971: J.Phys.Oceanogr., 1, 241-248.

Kraus,E.B., 1987: J.Geophys.Res., 92, (c13), 14242-14250.

Kattenberg, A., Allaart, M.A.F.: 1989: "The Effect of Salinity on a Tropical Pacific Ocean Model", Ocean Modelling, 83, 2. -Unpublished Manuscript-

Latif,M., 1982: J.Phys.Oceanogr., 17, 246-263.

Pacanowski,R.C., Philander,S.G.H., 1981: J.Phys.Oceanogr., 11, 1443-1451.

Peters,H., Gregg,M.C., Toole,J.M., 1988: J.Geophys.Res., 93, (G2), 1199-1218.

Press,W.H., Flannery,B.P., Teukosky,S.A., Vetterling,W.T., 1986: "Numerical Recipes", Cambridge University Press, Cambridge.

Roache,P.J., 1982: "Computational Fluid Dynamics", Hermosa, Albuquerque.

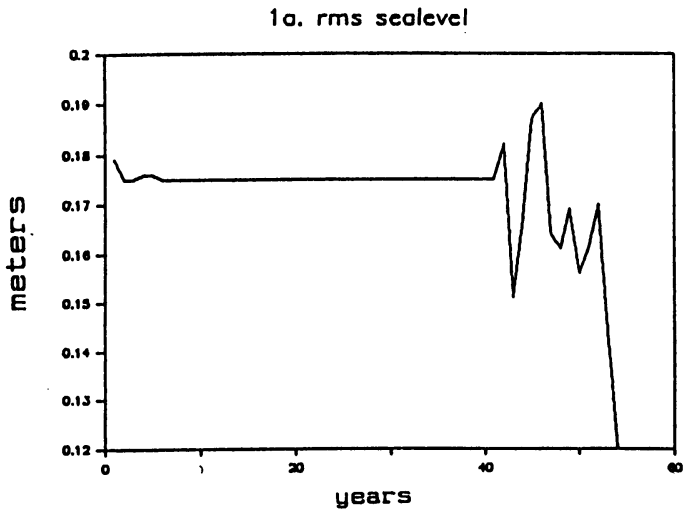
Stull,R.B., Kraus,E.B., 1987: J.Geophys.Res., 92, (C10), 10745-10755.

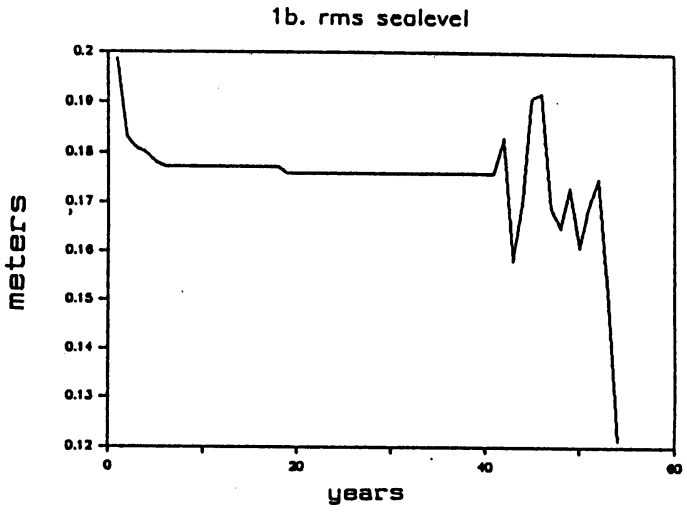
UNESCO, 1981: Technical Papers in Marine Sci., 4361, UNESCO, Paris.

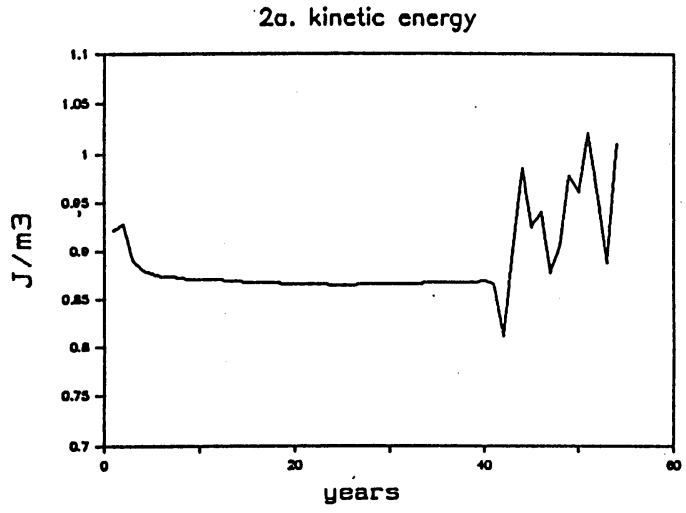
# Figure Captions

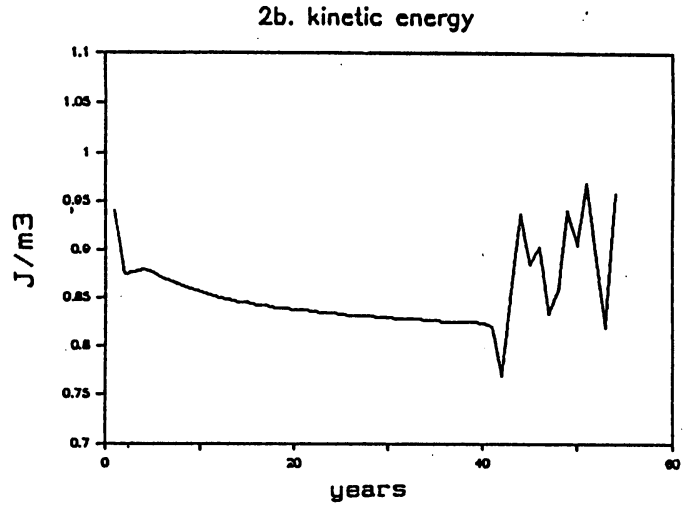
Figures 1–6 give a graphical representation of the output of the "DIAGN" routine (section 3.7.24) for two preliminary experiments, as an example. The horizontal axis in all the frames labels model-years; the experiments consisted of 40 yr forcing with 'climatological forcing' ('spinup'), followed by forcing with 14 years 'observed data'. The left-hand frames, labeled 'a' refer to a run with the complete model as described in this report; the right-hand frames, labeled 'b' refer to a run where salinity advection and diffusion are switched off. The signals on the vertical axes are discussed in section 3.7.24.

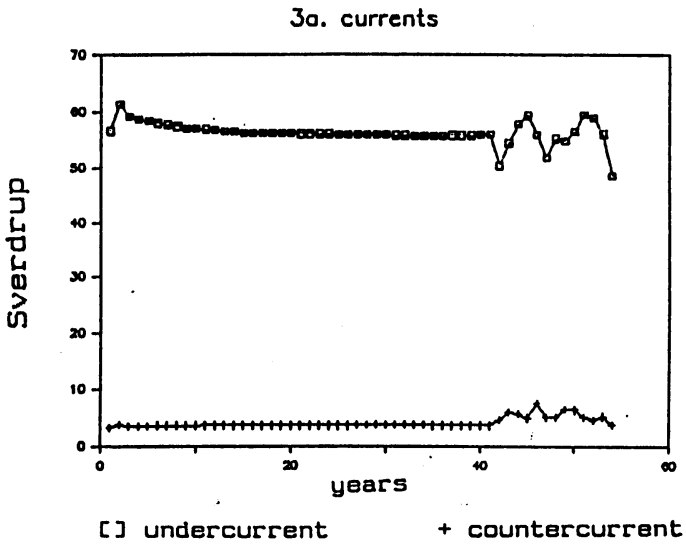
The experiments are discussed elsewhere, by Kattenberg and Allaart (1989).



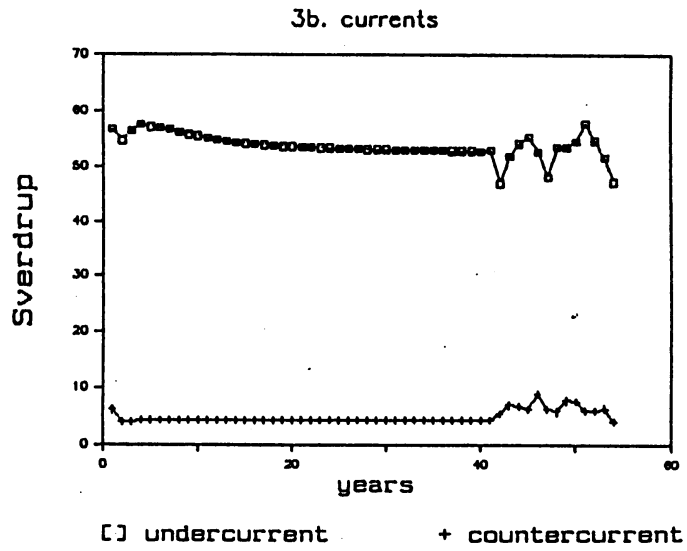



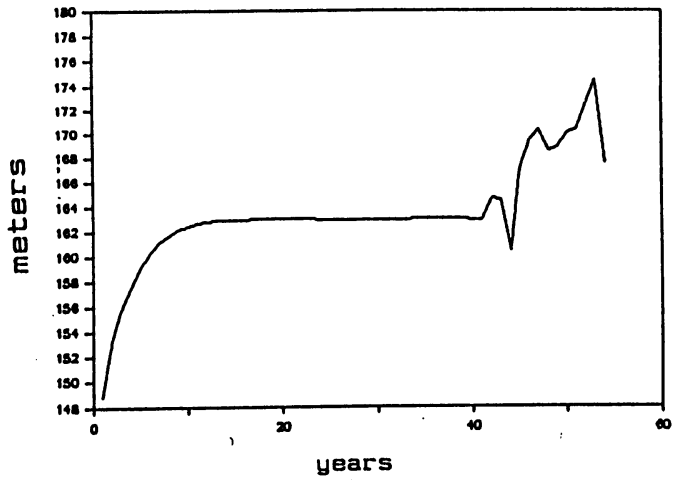




active salinity

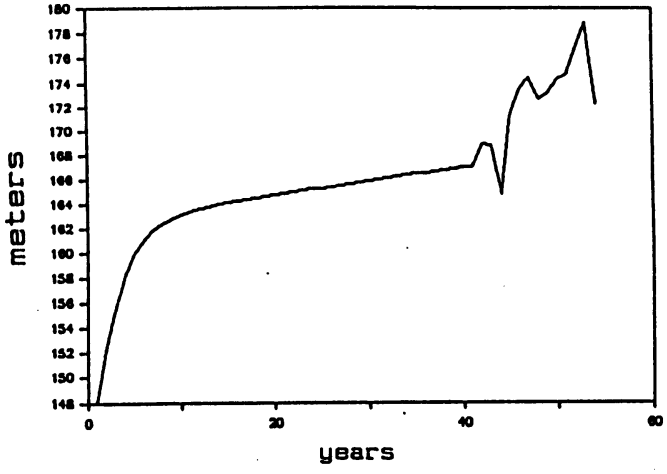



constant salinity

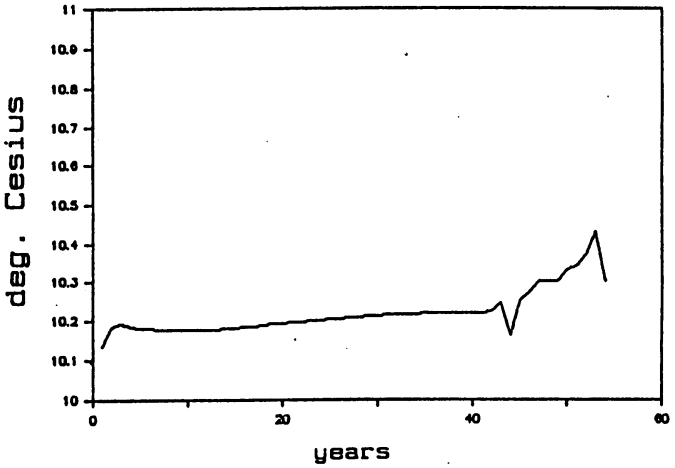
4a. depth 18c



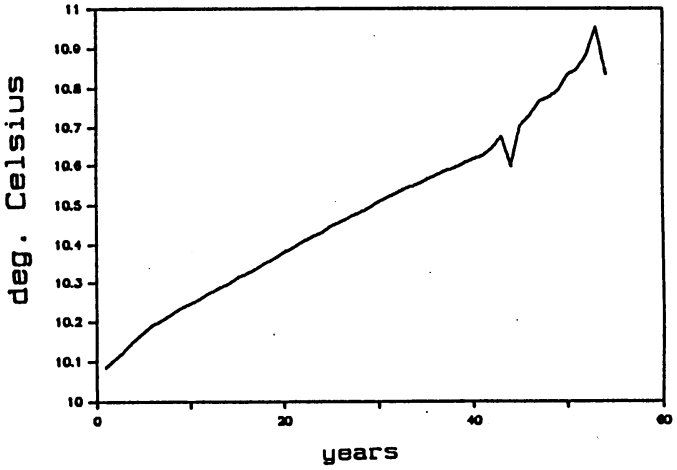

4b. depth 18c



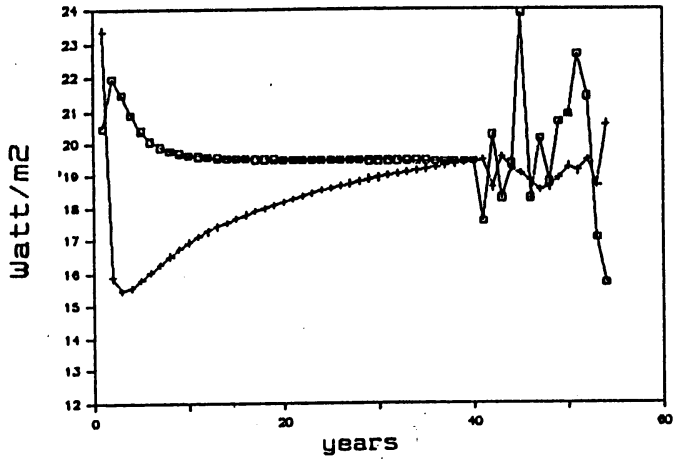

5a. temp. above 1050 m




5b. temp. above 1050 m



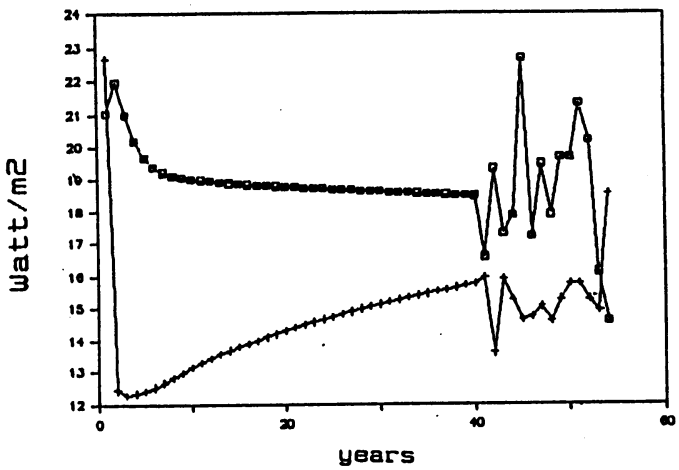

6a. heatflux




[] surface flux + bottom flux

active salinity

6b. heatflux




[] surface flux + bottom flux

constant salinity