FILE.

Institute of
Oceanographic Sciences
Deacon Laboratory

INTERNAL DOCUMENT No. 323

The Equation of State algorithms used by the OCCAM model

A C Coward

1993

Natural Environment Research Council

INSTITUTE OF OCEANOGRAPHIC SCIENCES
DEACON LABORATORY

INTERNAL DOCUMENT No. 323

The Equation of State algorithms used by the OCCAM model

A C Coward

1993

Wormley
Godalming
Surrey GU8 5UB UK
Tel +44-(0)428 684141
Telex 858833 OCEANS G
Telefax +44-(0)428 683066

DOCUMENT DATA SHEET

AUTHOR COWARD, A.C.	*PUBLICATION DATE* 1993
TITLE The Equation of State algorithms used by the OCCAM model.	
REFERENCE Institute of Oceanographic Sciences Deacon Laboratory, Internal Document, No. 323, 46pp. (Unpublished manuscript)	

ABSTRACT

The Ocean Circulation and Climate Advanced Modelling project (OCCAM) is an NERC Community Project which forms part of the UK contribution to the World Ocean Circulation Experiment (WOCE).

The aim of OCCAM Core Project 7 is to develop a global ocean model which will form the central 'test-bed' for algorithms developed by associated projects.

This document details the Equation of State algorithm which will be used to calculate density from potential temperature and salinity values derived by the numerical model.

KEYWORDS

NUMERICAL MODELLING

PROJECT - OCCAM

ISSUING ORGANISATION

Institute of Oceanographic Sciences
Deacon Laboratory
Wormley, Godalming
Surrey GU8 5UB. UK.

Director: Colin Summerhayes DSc

Telephone Wormley (0428) 684141
Telex 858833 OCEANS G.
Facsimile (0428) 683066

CONTENTS

1. INTRODUCTION

An important part of the Ocean Circulation and Climate Advanced Modelling project (OCCAM) is the development of a global ocean model. The final form of the model is, as yet, undecided but it will certainly require an accurate and computationally efficient method of calculating density from temperature and salinity values. Within a Bryan-Cox-Semtner ocean general circulation model (Bryan 1969, Semtner 1974, Cox 1984) such calculations are done at least three times for each grid node within the model. Precise methods require many hundreds of floating point calculations to derive each value. However, Webb (1992) has examined the available equations of state and compared the use of polynominal approximations with more complex schemes.

Following the recommendations of Webb (1992), the Cox polynominal equation of state will be used in the OCCAM model but the coefficients will be calculated using an improved method. This improved method uses the UNESCO equation of state (EOS80), (Fofonoff and Millard (1983), UNESCO (1981) and Gill (1982)) and accurate methods for converting model depth to pressure and *in situ* temperature to potential temperature.

To be precise, a FORTRAN77 program has been adapted to calculate the coefficients of a third order polynominal in potential temperature (°C) and salinity (parts per part) such that the polynominal will return values for density fitted to a least squares criterion. A different set of coefficients is derived for each model level.

1.1 Earlier work

This work uses as a starting point the subroutines and utility program supplied with the GFDL Modular Ocean Model (MOM) version 1 (Pacanowski, Dixon and Rosati (1990)). The equation of state supplied with the MOM code (state.F) employs the Cox polynominal approximation to the UNESCO equation of state as standard. A utility program, denscoef.F, is also supplied which will calculate the nine coefficients for each model level and construct the data statements to be included in state.F. The nine coefficients, $\beta_1, \ldots\ldots, \beta_9$, are employed within the model to calculate the density anomaly (relative to a reference density, $\rho_{0,k}$) using the polynominal equation:

$$\begin{aligned}\left(\rho-\rho_{0,k}\right) \;=\;& \beta_{1,k}\left(t-t_{o,k}\right)+\beta_{2,k}\left(s-s_{o,k}\right)+\beta_{3,k}\left(t-t_{o,k}\right)^2 \\ &+\beta_{4,k}\,(t-t_{o,k})\,(s-s_{o,k})+\beta_{5,k}\left(s-s_{o,k}\right)^2 \\ &+\beta_{6,k}\,(t-t_{o,k})^3+\beta_{7,k}\,(t-t_{o,k})(s-s_{o,k})^2 \\ &+\beta_{8,k}\left(t-t_{o,k}\right)^2\left(s-s_{0,k}\right)+\beta_{9,k}\left(s-s_{o,k}\right)^3\end{aligned} \qquad (1)$$

where $t_{o,k}$, $s_{o,k}$ and $r_{o,k}$ are 'reference' values for each level k.

2. The GFDL scheme

The program denscoef.F calculates the values for t_o, s_o, ρ_o and $\beta_1, \ldots\ldots, \beta_9$ in the following manner:
For each level:

1. Ranges of *in situ* t and s values are set for each model level. The current settings are shown in table 1 (page 8).
2. The ranges are divided into 2 x kx *in situ* temperature values and kx salinity values, where kx is a parameter set within the program (currently kx=5).
3. Either the Knudsen equation of state or the UNESCO equation of state is then used to calculate the density at each of the 2 x kx x kx points within the domain. The depth in metres of the level is used to approximate pressure for these calculations.
4. Each density is converted to a σ_t value.
5. Each *in situ* temperature is converted to a potential temperature (th) value using the Fofonoff and Froese (1958) equation.
6. The mean temperature (t_1) and salinity (s_o) are calculated and the σ_t (tot) value for seawater with these properties is calculated as in step 3.
7. The mean *in situ* temperature is converted to potential temperature (t_o) and t_o, s_o and tot are subtracted from each of the corresponding th, s and σ values.
8. A least squares procedure is then used to derive the nine polynominal coefficients as shown in equation 1.
9. The coefficients are then rescaled so that they are valid when used with salinity values given in model units. Model units are: (N.S.U.-35.)/1000., so this step involves subtracting 0.035 from the value of s_o and multiplying each coefficient by 10^3 for each power of s.
10. At the same stage the coefficients are rescaled so that density values in g/cm^3 are calculated instead of σ_t. If ρ is measured in g/cm^3 then σ_t = 1000 x (ρ - 1). Since the

right hand side of the polynominal is, at this stage, $(\sigma - \sigma_0)$, the rescaling is achieved by multiplying each coefficient by 10^{-3}.

11. The values of t_0, s_0 and the nine coefficients are written out in data statements to a file, dncoef.h. This file will be included in state.F by the C preprocessing stage at compilation. The values of σ_0 are also written to dncoef.h within FORTRAN comment statements.
12. State.F may now be fed values of potential temperature (°C) and salinity in model units and will return the deviation in g/cm^3 from the reference density for the relevant level.

3. The OCCAM scheme

The program, denscoef.F has been changed in order to incorporate the recommendations of Webb (1992). The changes have been made in the style recommended by the original authors and all the original features of the program have been retained in the form of '#ifdef' preprocessor constructions. The currently recommended method is now the default but the previous default options may be recovered using the 'gfdl_den' compile-time option. Several enhancements have been made for the current work.

Firstly, the linear scheme used to convert model depth to pressure has been replaced by a more accurate scheme using an iterative inverse of the algorithm due to Saunders (1981).

Secondly, the *in situ* to potential temperature conversion is carried out by direct integration using the Bryden equation for the adiabatic lapse rate (Bryden 1973) and a pressure increment of 1 decibar.

Thirdly, the curve-fitting procedure has been improved. As described above the polynominal approximation fixes the density at (t_0, s_0) to be ρ_0. Removing this constraint and making ρ_0 one of the unknown coefficients gives a better overall fit at the expense of introducing a small error at the central point. The reference density is not used in the main ocean model so the routine, state.F can be used without modification.

Finally, in the OCCAM version, the output procedure has been modified to write all relevant data into data statements within the include file. This includes two new arrays, z0 and rho0, each dimensioned with km elements. z0 contains the depth in centimetres and rho0 contains the new reference density of the relevant model level. These new arrays are not used in state.F within the Cox/MOM code but are required by any other applications which require the actual density value. The introduction of these arrays requires a minor modification to state.F :

the line:

```
common /cstate/ to(km), so(km), c(km,9)
```

should be changed to:

```
common /cstate/ to(km), so(km), c(km,9), z0(km), rho0(km).
```

The new OCCAM version of denscoef.F, is listed in Appendix I. In a SUNOS UNIX environment this can be compiled by the command:

```
f77 -o eqstat denscoef.F
```

On other systems it may be necessary to explicitly invoke the C-preprocesor. For example:

```
cc -P denscoef.F
mv denscoef.i denscoef.f
f77 -o eqstat denscoef.f
```

produces the same result. Similarly, use the following command in a SUNOS environment to compile the original GFDL default options:

```
f77 -D gfdl_den -o eqstat denscoef.F
```

A listing of state.F is given in Appendix III and the format of the dncoef.h file, which is produced by denscoef.F and included in state.F at compilation, is shown in Appendix II.

Table 1

k	depth	tmin	tmax	smin	smax
1	10.35e2	-2.000	29.000	28.5000	37.0000
2	32.35e2	-2.000	29.000	28.5000	37.0000
3	57.25e2	-2.000	29.000	28.5000	37.0000
4	86.00e2	-2.000	29.000	28.5000	37.0000
5	120.15e2	-2.000	29.000	28.5000	37.0000
6	162.15e2	-2.000	29.000	28.5000	37.0000
7	216.30e2	-2.000	29.000	28.5000	37.0000
8	290.05e2	-2.000	19.000	33.7000	36.6000
9	393.50e2	-2.000	19.000	33.7000	36.6000
10	532.00e2	-2.000	14.000	34.0000	35.8000
11	700.00e2	-2.000	14.000	34.0000	35.8000
12	887.50e2	-2.000	11.000	34.1000	35.7000
13	1087.50e2	-1.000	9.000	34.2000	35.3000
14	1295.50e2	-1.000	7.000	34.4000	35.1000
15	1508.50e2	-1.000	7.000	34.5000	35.1000
16	1725.50e2	-1.000	7.000	34.5000	35.1000
17	1945.50e2	-1.000	7.000	34.5000	35.0000
18	2167.50e2	-1.000	7.000	34.6000	35.0000
19	2391.50e2	-1.000	7.000	34.6000	35.0000
20	2617.00e2	-1.000	7.000	34.6000	35.0000
21	2843.50e2	-1.000	7.000	34.6000	35.0000
22	3071.00e2	-1.000	7.000	34.6000	35.0000
23	3299.50e2	-1.000	7.000	34.6000	35.0000
24	3529.00e2	-1.000	7.000	34.6000	35.0000
25	3759.00e2	-1.000	7.000	34.6000	35.0000
26	3989.50e2	-1.000	7.000	34.6000	35.0000
27	4220.50e2	-1.000	7.000	34.6000	35.0000
28	4452.00e2	-1.000	7.000	34.6000	35.0000
29	4684.00e2	-1.000	7.000	34.6000	35.0000
30	4916.50e2	0.000	7.000	34.6000	35.0000
31	5149.50e2	0.000	7.000	34.6000	35.0000
32	5382.50e2	0.000	7.000	34.6000	35.0000

REFERENCES

BRYDEN, H.L. 1973 New polynomials for thermal expansion, adiabatic temperature gradient and potential temperature of sea water.
Deep Sea Research, 20, p 401-408.

FOFONOFF, N.P. & FROESE, C. 1958 As shown in FOFONOFF, N.P. 1962 The Sea: Vol. 1. (Ed. M. Hill.) Interscience, New York, p17.

FOFONOFF, N.P. & MILLARD, R.C. Jnr. 1983 Alogrithms for computation of fundamental properties of seawater.
UNESCO Technical Papers in Marine Science, 44, 53pp.

GILL, A.E. 1982 Atmosphere-Ocean dynamics.
Academic Press, New York, 662pp.

PACANOWSKI, R.C., DIXON K., & ROSATI, A. 1990 The GFDL Modular Ocean Model Users Guide, version 1.0.
GFDL Group Techinical Report No. 2.

SAUNDERS, P. M. 1981 Practical Conversion of Pressure to Depth.
Journal of Physical Oceanography, 11 (4), p 573-574.

UNESCO. 1981 Tenth report of the joint panel on oceanographic tables and standards.
UNESCO Technical Papers in Marine Science, 36, 25pp.

WEBB, D.J. 1992 The equation of state algorithms used by the FRAM model.
Institute of Oceanographic Sciences Deacon Laboratory, Internal Document No. 313, 34pp.

APPENDIX I:

denscoef.F

```
c     subroutine eqstat
      program eqstat
c
c     due to the simple UNIX linker, only one main program may be in the
c     directory at one time. To run this program, wipe out the
c     the subroutine call & uncomment the program eqstat line. all other
c     main programs must be either removed from the directory or
c     commented out also!
c     note: this is not a problem if "makefiles" are used for compiling
c     & linking.
c
c=======================================================================
c
c     calculate coefficients for "MOM" density computations
c
c       This program calculates the 9 or 10 coefficients of a third
c     order polynomial approximation to the equation of state for sea
c     water.
c       The program yields coefficients that will compute density as a
c     function of temperature, and salinity, at predetermined depths,
c     as used in the subroutine "state" of the GFDL ocean model.
c     More specifically, the densities calculated from the ploynomial
c     formula are in the form of sigma anomalies.  The method is taken
c     from that described by Bryan & Cox (1972).
c       By default, the program uses the equation of state set by the
c     Joint Panel on Oceanographic Tables & Standards (UNESCO, 1981)
c     an described by Gill (1982).  An option exists to use the older
c     Knudsen-Ekman equation of state, as described by Fofonoff (1962),
c     if the user prefers.
c       Subroutine "lsqsl2" performs the iterative least-squares
c     polynomial fitting for the overdetermined system.  The algorithm
c     is outlined by Hanson and Lawson (1969), and the code looks as if
c     it has not been altered since that time.
c
c     references:
c
c        Bryan, K. & M. Cox. 1972  An approximate equation of state for
c          numerical models of ocean circulation.
c          J. Phys. Oceanogr., 2, 510-514, 1972.
```

```
c        Fofonoff, N. 1962  The Sea: Vol 1, (ed. M. Hill).
c          Interscience, New York, pp 3-30.
c        Gill, A.  1982.  Atmosphere-Ocean Dynamics: International
           Geophysical Series No. 30.
c          Academic Press, London, pp 599-600.
c        Hanson, R., & C. Lawson. 1969   Extensions and applications of
c          the Householder algorithm for solving linear least squares
c          problems.  Math. Comput., 23, pp 787-812.
c        UNESCO. 1981  10th report of the joint panel on oceanographic
           tables and standards.
c          UNESCO Tech. Papers in Marine Sci. No. 36, Paris.
c
c
c    ifdef options:
c
c     Default: use the "reference equation of state" as in Webb (1992)
c     Ref: Webb, D.J. 1992  "The equation of state algorithms used by
c     the FRAM model".  Institute of Oceanographic Sciences Internal
c     Document No. 313.
c
c    "gdfl_den"
c     Revert to the original "GFDL" scheme as supplied with MOM version
c     1.0, released December 1991.
c    "knudsen"
c     To over-ride the default of using the UNESCO equation of state
c     and to instead employ the Knudsen-Ekman formula.
c    "insitu"
c     If the user desires the polynomial approximations to calculate
c     density as a function of in situ temperature, salinity, and depth,
c     then the ifdef option "insitu" must be defined.  Otherwise, the
c     default assumption is that potential temperatures will be used (as
c     in the ocean model code).
c    "extras"
c     If the user wishes to have a detailed report of the inputs and
c     results of the curve fitting processes written to the standard
c     output unit (stdout), then the ifdef option "extras" should be
c     defined.  The default is for a rather short summary to be written.
c
c    inputs:
c      The user needs only to specify the number of model levels "km"
c     and the model layer thicknesses [cm] "dzt(1..km)".  This
c     information can be entered below via the same "dzt" data statement
c     contained in the "thick.h" file used in the compilation of the
c     GFDL ocean model.  The parameter "km" and constants "c0", "c1",
c     "c2" and "p5" can be set by including the "param.h" file from the
c     model as well.
c
```

```
c
c-----------------------------------------------------------------------
c
      implicit double precision (a-h,o-z)
c
c-----------------------------------------------------------------------
c
#include "param.h"
c
      dimension dzt(km)
c
      parameter (kx = 5, kxx = 2*kx, kk = kx*kxx,
#ifdef gfdl_den
     $  kcolm=9 )
#else
     $  kcolm=10 )
#endif
      parameter (krdim = kk+4*kcolm, ksdim = kk+8*kcolm )
c
      dimension a(kk,kcolm),sigma(kk),sigman(kk),c(kk,kcolm),x(kcolm),
     $          sb(ksdim), r(krdim)
      dimension tmin(km), smin(km), tmax(km), smax(km),
     $          z(km), dd(km), ss(km), ab(kcolm+4,km), ts(33,4),
     $          ta(kxx), sa(kxx), tp(kk), sp(kk), th(kk)
c
      real realz
c
      double precision mpercm
      data  mpercm / 1.0d-2 /
c
#include "thick.h"
c
c  enter bounds for polynomial fit: at 33 levels from sfc to 8000 m.
c          ts(k,1)=lower bnd of t at z=(k-1)*250 meters
c          ts(k,2)=upper bnd of t          "
c          ts(k,3)=lower bnd of s          "
c          ts(k,4)=upper bnd of s          "
c
c      The user should review the appropriateness of the "ts" values set
c     below, and modify them if the intended modelling application could
c     be expected to yield temperature and salinity values outside of
c     the "ts" ranges set by default.
c
      data (ts(k,1),k=1,33) / 4*-2.0, 15*-1.0, 14*0.0 /
      data (ts(k,2),k=1,33) / 29.0, 19.0, 14.0, 11.0, 9.0, 28*7.0 /
      data (ts(k,3),k=1,33) / 28.5, 33.7, 34.0, 34.1, 34.2, 34.4,
     $                        2*34.5, 15*34.6, 10*34.7 /
      data (ts(k,4),k=1,33) / 37.0, 36.6, 35.8, 35.7, 35.3, 2*35.1,
     $                        26*35.0 /
```

```
#ifndef gfdl_den
      data xlat/30.0/
c     xlat    = Reference latitude used by the depth-to-pressure
c               function: FNPZ.
#endif
c
c     z       = model levels (midpoint of model layers)
c     tmin, tmax, smin, smax = minimum and maximum in situ temperature
c               and salinity values which define the ranges to be used
c               when computing the polynomials at each model level
c     dd, ds  = increment between temperature and salinity values at
c               each model level to be used in constructing array of
c               temperature, salinity and density for curve fitting
c     ta, sa  = in situ temperature and salinity values available for
c               constructing array of data for curve fitting at each
c               model level
c     tp, sp  = in situ temperature and salinity values constructed from
c               all combinations of ta & sa
c     th      = potential temperature values associated with "tp" at a
c               given level and salinity
c     t1, s1, tot1, th1 = level mean insitu temp., salinity, density,
c               and potential temp. used in polynomial fitting
c     tot     = density (in sigma units) calculate from t1 and s1 at a
c               given model level
c     sigma   = insitu densities (in sigma units) calculated from "tp"
c               and "sp" values
c     sigman  = insitu density anomalies at a given level (formed by
c               subracting "tot" from sigma)
c     tanom, sanom = temperature and salinity anomalies used in loading
c               array "a" for use in lsqsl2 curve fitting
c     x       = the 10 polynomial coefficients. This includes a constant
c               for the polynominal which means that the polynominal is
c               not constrained to pass through tot1 at (th1,s1). This
c               constant is subtracted from the level reference density
c               after the least-squares procedure. The polynominal used
c               by state and statec is thus unchanged but should be a
c               better fit than that given by the original procedure.
c     r, sb   = used only in lsqsl2
c
c=======================================================================
c

c     calculate depths of levels from dzt (converting dzt from cm
c     to meters) - the maximum allowable depth is 8000 meters
c
      z(1)= p5 * dzt(1) * mpercm
      do 100 k=2,km
        z(k) = z(k-1) + p5 * (dzt(k)+dzt(k-1)) * mpercm
100   continue
```

```
c
c     set the temperature and salinity ranges to be used for each model
c     level when performing the polynomial fitting
c
      do 200 k=1,km
        realz = z(k)/250.0
        i = ifix (realz) + 1
        tmin(k) = ts(i,1)
        tmax(k) = ts(i,2)
        smin(k) = ts(i,3)
        smax(k) = ts(i,4)
200   continue
c
c   write out model depths and ranges of temperatures & salinities over
c   which the polynomial approximations are computed.
c
      write (stdout,9060)
      write (stdout,9061) (z(i),tmin(i),tmax(i),smin(i),smax(i),i=1,km)
      write (stdout,9062)
c
c   set temperature and salinity increments to be used in creating
c   curve fitting array at each level (twice as many temperature values
c   than salinity values)
c
      fkx = kx
      do 300 k=1,km
        dd(k) = (tmax(k)-tmin(k)) / (c2*fkx-c1)
        ss(k) = (smax(k)-smin(k)) / (fkx-c1)
300   continue
c
c   loop over all model levels
c
      do 400 k=1,km
      write(6,'(a,i3,$)') 'Start level: ',k
c
        do 340 i=1,kxx
          fi = i
          ta(i) = tmin(k) + (fi-c1)*dd(k)
          sa(i) = smin(k) + (fi-c1)*ss(k)
340     continue
c
c   load the "kxx" cominations of the 2*"kx" insitu temp. and "kx"
c   salinity values into "tp" and "sp"
c
        do 360 i=1,kxx
          do 350 j=1,kx
            ka = kx*i + j - kx
            tp(ka) = ta(i)
            sp(ka) = sa(j)
350       continue
```

```
360     continue
c
        t1  = c0
        s1  = c0
        tot = c0
        th1 = c0
        fkk = kk
c
c   calculate insitu density "sigma" for each t,s combintion at
c   this depth "d"
c
        do 370 ka=1,kk
#ifndef gfdl_den
c Convert depth to pressure accurately
          d=fnpz(z(k),xlat)
#else
          d = z(k)
#endif
          s = sp(ka)
          t = tp(ka)
c
#ifdef knudsen
c   "knuekm" returns density (in sigma units) from insitu temperature,
c   salinity, & depth (pressure) using the Knudsen-Ekman formula
c
          call knuekm(t,s,d,densit)
c
          sigma(ka) = densit
#else
c   "unesco" returns density (kg per m**3) from insitu temperature,
c   salinity, & depth (pressure) using the UNESCO equation of state
c
          call unesco(t,s,d,densit)
c
          sigma(ka) = densit - 1.0d3
#ifdef gfdl_den
     $                                 + 2.5d-2
c (Note original denscoef.F added 2.5d-2 here to convert to "old" sigma)
#endif
#endif
c
c   "potem" returns potential temp. from from insitu temperature,
c   salinity, & depth (pressure)
c
#ifdef gfdl_den
          call potem(t,s,d,theta)
#else
c   ..so does "tadiab" but more accurately
          theta = tadiab(t,s,d,0.0d0,1.0d0)
#endif
```

```
c
          th(ka) = theta
          t1 = t1 + tp(ka)
          s1 = s1 + sp(ka)
          tot = tot + sigma(ka)
          th1 = th1 + th(ka)
370     continue
c
c   form layer averages "t1", "s1", "th1", and "tot1", and compute
c   reference density "tot" from "t1" and "s1" at this depth "d"
c
        t1 = t1/fkk
        s1 = s1/fkk
        th1 = th1/fkk
        tot1 = tot/fkk
#ifdef knudsen
c
c   "knuekm" returns density from insitu temp., salinity, & depth
c   (pressure) using the Knudsen-Ekman formula
c
        call knuekm (t1, s1, d, densit)
c
        tot = densit
#else
c
c   "unesco" returns density from insitu temp., salinity, & depth
c   (pressure) using the UNESCO equation of state
c
        call unesco (t1, s1, d, densit)
        tot = densit - 1.0d3
#ifdef gfdl_den
     $                               + 2.5d-2
c (Note original denscoef.F added 2.5d-2 here to convert to "old" sigma)
#endif
#endif
c
#ifdef extras
c
c define "extras" for voluminous printout of calculation info.
c
        write (stdout,'(a49)')
     $  ' insitu temperatures used in polynomial fit & avg'
        write (stdout, 9071) kk, (tp(ka),ka=1,kk)
        write (stdout, 9072) t1, k
        write (stdout,'(a40)')
     $  ' salinities used in polynomial fit & avg'
        write (stdout, 9071) kk, (sp(ka),ka=1,kk)
        write (stdout, 9072) s1, k
        write (stdout,'(a53)')
     $   ' densities (sigma units) used in polynomial fit & avg'
         write (stdout, 9071) kk, (sigma(ka),ka=1,kk)
         write (stdout, 9072) tot1, k
         write (stdout,'(a54)')
     $   ' density calculated from level avg insitu t & salinity'
         write (stdout, 9072) tot, k
         write (stdout,'(a52)')
     $   ' potential temperatures used in polynomial fit & avg'
         write (stdout, 9071) kk, (th(ka),ka=1,kk)
         write (stdout, 9072) th1, k
#ifdef insitu
         write (stdout,'(a47)')
     $   ' >> insitu temps were used in polynomial fit <<'
#else
         write (stdout,'(a50)')
     $   ' >> potential temps were used in polynomial fit <<'
#endif
c
#endif
#ifndef insitu
c
c define insitu if using insitu temperatures (removes this line)
c
         t1 = th1
c
#endif
c
c   begin loading "ab" array with level averages
c
         ab(1,k) = z(k)
         ab(2,k) = tot
         ab(3,k) = t1
         ab(4,k) = s1
c
         do 380 ka=1,kk
#ifndef insitu
c
c define insitu (removes this line) if using insitu temperatures
c
           tp(ka) = th(ka)
#endif
c
c   create anomalies for temperature, salinity & density and
c   load work array "a" with the anomalies and their products
c
           tanom = tp(ka) - t1
           sanom = sp(ka) - s1
           sigman(ka) = sigma(ka) - tot
```

```
          a(ka,1)  = tanom
          a(ka,2)  = sanom
          a(ka,3)  = tanom * tanom
          a(ka,4)  = tanom * sanom
          a(ka,5)  = sanom * sanom
          a(ka,6)  = a(ka,3)  * tanom
          a(ka,7)  = a(ka,5)  * tanom
          a(ka,8)  = a(ka,3)  * sanom
          a(ka,9)  = a(ka,5)  * sanom
#ifndef gfdl_den
          a(ka,10)= c1
#endif
380     continue
c
c     set the arguments used in call to "lsqsl2"
c     ndim = first dimension of array a
c     nrow =number of rows of array a
c     ncol = number of columns of array a
c     in = option number of lsqsl2
c     itmax = number of iterations
c
        ndim = 50
        nrow = kk
        ncol = kcolm
        in = 1
        itmax = 4
c
        it = 0
        ieq = 2
        irank = 0
        eps = 1.0e-7
        nhdim = kcolm
c
c   LSQL2 is  a Jet Propulsion Laboratory subroutine that computes the
c   least squares fit in an iterative manner for overdetermined systems.
c   Find vector x such that ax-sigman is minimised.
c
        call lsqsl2 (ndim, a, nrow, ncol, sigman, x, irank, in, itmax,
     $              it, ieq, enorm, eps, nhdim, h, c, r, sb)
c
#ifdef extras
        write (stdout, 9081) k, (x(i),i=1,kcolm)
        write (stdout, 9082) tot
        write (stdout, 9062)
c
#endif
        do 390 i=1,ncol
          ab(i+4,k) = x(i)
390     continue
```

```
c
      write(6,'(a,i3)') ' End of level: ',k
 400  continue
c
      nn = ncol + 4
      write (stdout, 9091)
      write (stdout, 9092) ((ab(i,j),i=1,nn),j=1,km)
      write (stdout, 9093)
c
c     write data statements to unit 50 ==> "dncoef.h"
c
      open (50,file='dncoef.h')
c
      write(50,9501)
c
c  Now rescale the coefficients so that the polynominal can be used with
c  model salinities (i.e. adjust s0 & multiply coefficients of s by
c  1000) and rescale so that density is given in g/cm3 instead of
c  sigma_t (i.e. divide all coefficients by 1000). E.g. Coefficient of
c  (t-t0)*(s-s0)**2 will be multiplied by: 0.001*1.*1000.**2=1000.
c
      do 500 k=1,km
c  convert sig0 to g/cm3 and subtract constant:
                          ab(2,k) = 1.e-3 * ab(2,k)
#ifndef gfdl_den
     $                                        - (1.e-3 * ab(14,k)) + 1.
#endif
c  rescale so to model units:
                          ab(4,k) = 1.e-3 * ab(4,k) - 0.035
c  adjust coefficients:
c   1. (t-to):
                          ab(5,k) = 1.e-3 * ab(5,k)
c   2. (t-to)**2:
                          ab(7,k) = 1.e-3 * ab(7,k)
c   3. (t-to)**3:
                          ab(10,k) = 1.e-3 * ab(10,k)
c   4. (s-so)**2:
                          ab( 9,k) = 1.e+3 * ab( 9,k)
c   5. (t-to)*(s-so)**2:
                          ab(11,k) = 1.e+3 * ab(11,k)
c   6. (s-so)**3:
                          ab(13,k) = 1.e+6 * ab(13,k)
c   7. scaling factors equate to unity for all other coefficients
c
500   continue
#ifdef gfdl_den
c
```

```
c   write out "to" & "so" data statements
c
      do 600 nx=3,4
        if (nx .eq. 3) write(50,9502)
        if (nx .eq. 4) write(50,9503)
        n = 0
        do 590 ii=1,99
          is = n+1
          ie = n+5
          if (ie .lt. km) then
            write(50,9510) (ab(nx,i),i=is,ie)
            n = ie
          else
            ie = km
            n = ie-is+1
            if (n .eq. 1) write(50,9511) (ab(nx,i),i=is,ie)
            if (n .eq. 2) write(50,9512) (ab(nx,i),i=is,ie)
            if (n .eq. 3) write(50,9513) (ab(nx,i),i=is,ie)
            if (n .eq. 4) write(50,9514) (ab(nx,i),i=is,ie)
            if (n .eq. 5) write(50,9515) (ab(nx,i),i=is,ie)
            goto 600
          endif
590     continue
600   continue
#endif
c
c  write out data statement for each level
c  (z0, rho0, to, so & 9 coefficients)
      do 700 k=1,km
#ifdef gfdl_den
        write(50,9521) k
#else
        write(50,9521) k,k,k,k,k
        write(50,9524) (ab(i,k),i=1,4)
#endif
        write(50,9522) (ab(i,k),i=5,8)
        write(50,9522) (ab(i,k),i=9,12)
        write(50,9523) ab(13,k)
700   continue
c
      write (50,9531)
      write (50,9532) (i, z(i),tmin(i),tmax(i),smin(i),smax(i),i=1,km)
      write (50,9533)
      do 800 k=1,km
        ab(2,k) = ab(2,k) * 1.e3
#ifndef gfdl_den
     $                   - 1.e3
#endif
```

```
800   continue
      write (50,9534) (ab(2,k),k=1,km)
      write (50,9535)
      close (50)
c
c =====================================================================
c
      stop
c
 9060 format(///6x,'level   tmin      tmax      smin       smax',/)
 9061 format(5x,f5.0,4f10.3)
 9062 format(///)
 9091 format(//,
     $' calculating coefficients for "mom" density computations'/
     $'   z    sig0    t    s       x1           x2           ',
     $'x3           x4           x5           x6           x7           x8',
     $'            x9',/)
#ifdef gfdl_den
 9092 format(//,f5.0,f8.4,f5.1,f6.2,9e12.5)
#else
 9092 format(//,12(e11.5,1x))
#endif
 9093 format(//,
     $' === a new "dncoef.h" has been created by this program === ')

#ifdef extras
 9071 format(/' kk = # of pts going into interpltn =',i4,/
     $       (1x,5e14.7))
 9072 format(5x,' avg =',e14.7,' for level ',i4,/)
#ifdef gfdl_den
 9081 format(' model level ',i3,': before scaling (x(i),i=1,9)='/
     $        1x,5e14.7,/,1x,4e14.7)
#else
 9081 format(' model level ',i3,': before scaling (x(i),i=1,10)='/
     $        1x,5e14.7,/,1x,5e14.7)
#endif
 9082 format(' reference sigma, about which density anomalies are ',
     $       'computed'/1x,e14.7)
#endif
c
#ifdef gfdl_den
 9501 format('c===================== include file "dncoef.h"',
     $  ' ========================='/'c'/'c'/,
     $  'c     normalized temperatures, salinities and',
     $  ' coefficients'/'c     generated by program "eqstat" ',
     $  'which fits 3rd order polynomials'/'c     to the equation ',
     $  'of state for each model level.'/'c')
```

```
#else
 9501 format('c===================== include file "dncoef.h"',
     $  ' ========================'/'c'/'c'/,
     $  'c     normalized temperatures, salinities and coefficients'/
     $  'c     generated by program "eqstat" which fits 3rd order '/
     $  'c     polynomials to the equation of state for each model ',
     $        'level.'/'c'/
     $  'c     The polynominal returns density deviations in g/cm**3 ',
     $        'from '/
     $  'c     "rho0" for each level. The polynominal is in powers of'/
     $  'c     (t-to) and (s-so), where t is potential temperature, '/
     $  'c     measured in degrees C and s is salinity measured in '/
     $  'c     model units. The arrays rho0 and z0, where z0 contains'/
     $  'c     the depth in centimetres of each level, are stored for '/
     $  'c     reference.'/'c')
#endif
 9502 format(6x,'data to /',67x,i9)
 9503 format(6x,'data so /',67x,i9)
 9510 format(5x,'$',8x,5(f10.7,','))
 9511 format(5x,'$',8x,f10.7,'/',/'c')
 9512 format(5x,'$',8x,f10.7,',',f10.7,'/',/'c')
 9513 format(5x,'$',8x,2(f10.7,','),f10.7,'/',/'c')
 9514 format(5x,'$',8x,3(f10.7,','),f10.7,'/',/'c')
 9515 format(5x,'$',8x,4(f10.7,','),f10.7,'/',/'c')
#ifdef gfdl_den
 9521 format(6x,'data (c(',i2,',n),n=1,9)/')
#else
 9521 format(6x,'data z0(',i2,'), rho0(',i2,'), to(',i2,'), so(',i2,')'
     +       ', (c(',i2,',n),n=1,9)/')
#endif
 9522 format(5x,'$',9x,4(e13.7,','))
 9523 format(5x,'$',9x,e13.7,'/',/,'c')
#ifndef gfdl_den
 9524 format(5x,'$',9x,f10.5,'E02,',2(f13.7,','),e13.7,',')
#endif
 9531 format('c  the above coefficients were calculated using program ',
     $       '"eqstat"',
#ifdef gfdl_den
     $       /'c  compiled with the "gfdl_den" option and',
#endif
#ifdef knudsen
     $       /'c  employing the Knudsen-Ekman equation of state.',
#else
     $       /'c  employing the UNESCO equation of state.',
#endif
     $       /'c  They are valid for the following depths and',
     $        ' T and S ranges'
     $       /'c',t7,'k',t14,'depth',t27,'tmin',t37,
     $        'tmax',t52,'smin',t62,'smax')
 9532 format('c',t5,i3,t12,f7.2,'e2',t25,f7.3,t35,f7.3,t50,f7.4,
     $       t60,f7.4)
 9533 format( 'c'/
     $        'c  the 3rd order polynomial will return density ',
     $           'departures [gm/cm**3] as'/
     $        'c  a function of',
#ifdef insitu
     $              ' insitu ',
#else
     $              ' potential ',
#endif
     $       'temperature [deg C] & salinity [model units]'/
     $       'c'/
     $       'c       k level reference densities (in sigma units):')
 9534 format('c  ',8f8.4)
 9535 format('c')
c
      end
c
        subroutine knuekm (t, s, d, rho)
c=======================================================================
c     this subroutine calculates the density of seawater using the
c     Knudsen-Ekman equation of state.
c
c     input [units]:
c       in-situ temperature (t): [degrees centigrade]
c       salinity (s): [per mil]
c       depth (d): [meters of depth, to approximate pressure]
c     output [units]:
c       density (rho): sigma units
c
c     reference:
c        Fofonoff, N., The Sea: Vol 1, (ed. M. Hill). Interscience,
c          New York, 1962, pp 3-30.
c
c-----------------------------------------------------------------------
c
      implicit double precision (a-h,o-z)
c
c=======================================================================
c
      t2 = t*t
      t3 = t2*t
      s2 = s*s
      s3 = s2*s
      f1 = -1.0d0 * (t - 3.98d0)**2 * (t + 2.83d2) /
     $      (5.0357d2*(t + 6.726d1))
```

```
       f2 = t3*1.0843d-6 - t2*9.8185d-5 + t*4.786d-3
       f3 = t3*1.6670d-8 - t2*8.1640d-7 + t*1.803d-5
       fs = s3*6.76786136d-6 - s2*4.8249614d-4 + s*8.14876577d-1
c
       sigma= f1 + (fs + 3.895414d-2)*
      $      (1.0d0 - f2 + f3*(fs - 2.2584586d-1))
c
       a= d*1.0d-4*(1.055d2 + t*9.50d0 - t2*1.58d-1 - d*t*1.5d-4)  -
      $   (2.27d2 + t*2.833d1 - t2*5.51d-1 + t3*4.0d-3)
       b1 = (fs - 2.81324d1)*1.d-1
       b2 = b1 * b1
       b  = -b1* (1.473d2 - t*2.72d0 + t2*4.0d-2 - d*1.0d-4*
      $     (3.24d1 - 0.87d0*t + 2.0d-2*t2))
       b  = b + b2*(4.5d0 - 1.0d-1*t - d*1.0d-4*(1.8d0 - 6.0d-2*t))
       co = 4.886d3/(1.0d0 + 1.83d-5*d)
c
       alpha = d*1.0d-6*(co + a + b)
c
       rho = (sigma + alpha)/(1.d0 - 1.0d-3*alpha)
c
       return
       end
         subroutine lsqsl2
      1(ndim,a,d,w,b,x,irank,in,itmax,it,ieq,enorm,eps1
      2,nhdim,h,aa,r,s)
c
c      this routine is a modification of lsqsol. march, 1968. r. hanson.
c      linear least squares solution
c
c      this routine finds x such that the euclidean length of
c      (*) ax-b is a minimum.
c
c      here a has k rows and n columns, while b is a column vector with
c      k components.
c
c      an orthogonal matrix q is found so that qa is zero below the main
c      diagonal.
c      suppose that rank (a)=r
c      an orthogonal matrix s is found such that
c      qas=t is an r x n upper triangular matrix whose last n-r columns
c      are zero.
c      the system tz=c (c the first r components of qb) is then
c      solved. with w=sz, the solution may be expressed
c      as x = w + sy, where w is the solution of (*) of minimum euclid-
c      ean length and y is any solution to (qas)y=ty=0.
c
c      iterative improvements are calculated using residuals and
c      the above procedures with b replaced by b-ax, where x is an
c      approximate solution.
```

```
c
      implicit double precision (a-h,o-z)
c
      double precision sj,dp,up,bp,aj
      logical erm
      integer d,w
c
c     in=1 for first entry.
c                 a is decomposed and saved. ax-b is solved.
c     in = 2 for subsequent entries with a new vector b.
c     in=3 to restore a from the previous entry.
c     in=4 to continue the iterative improvement for this system.
c     in = 5 to calculate solutions to ax=0, then store in the array h.
c     in  =  6   do not store a  in aa.  obtain  t = qas, where t is
c     min(k,n) x min(k,n) and upper triangular. now return.do not obtain
c     a solution.
c     no scaling or column interchanges are performed.
c     in  =  7   same as with  in = 6  except that soln. of min. length
c               is placed into x. no iterative refinement.  now return.
c     column interchanges are performed. no scaling is performed.
c     in  =  8    set addresses. now return.
c
c     options for computing  a matrix product   y*h  or  h*y are
c     available with the use of the entry points  myh and mhy.
c     use of these options in these entry points allow a great saving in
c     storage required.
c
c
      dimension a(ndim,ndim),b(1),aa(d,w),s(1), x(1),h(nhdim,nhdim),r(1)
c     d = depth of matrix.
c     w = width of matrix.
      k=d
      n=w
      erm=.true.
c
c     if it=0 on entry, the possible error message will be suppressed.
c
      if (it.eq.0) erm=.false.
c
c     ieq = 2      if column scaling by least max. column length is
c     to be performed.
c
c     ieq = 1       if scaling of all components is to be done with
c     the scalar max(abs(aij))/k*n.
c
c     ieq = 3 if column scaling as with in =2 will be retained in
c     rank deficient cases.
c
```

```
c     the array s must contain at least max(k,n) + 4n + 4min(k,n) cells
c        the   array r must contain k+4n s.p. cells.
c
      data eps2/1.d-16/
c     the last card controls desired relative accuracy.
c     eps1  controls  (eps) rank.
c
      isw=1
      l=min0(k,n)
      m=max0(k,n)
      j1=m
      j2=n+j1
      j3=j2+n
      j4=j3+1
      j5=j4+1
      j6=j5+1
      j7=j6+1
      j8=j7+n
      j9=j8+n
      lm=l
      if (irank.ge.1.and.irank.le.l) lm=irank
      if (in.eq.6) lm=l
      if (in.eq.8) return
c
c     return after setting addresses when in=8.
c
      go to (10,360,810,390,830,10,10), in
c
c     equilibrate columns of a (1)-(2).
c
c     (1)
c
   10 continue
c
c     save data when in = 1.
c
      if (in.gt.5) go to 30
      do 20 j=1,n
      do 20 i=1,k
   20 aa(i,j)=a(i,j)
   30 continue
      if (ieq.eq.1) go to 60
      do 50 j=1,n
      am=0.e0
      do 40 i=1,k
   40 am= max(am,abs(a(i,j)))
c
c      s(m+n+1)-s(m+2n) contains scaling for output variables.
```

```
c
      n2=j2+j
      if (in.eq.6) am=1.d0
      s(n2)=1.d0/am
      do 50 i=1,k
   50 a(i,j)=a(i,j)*s(n2)
      go to 100
   60 am=0.d0
      do 70 j=1,n
      do 70 i=1,k
   70 am= max(am,abs(a(i,j)))
      am=am/float(k*n)
      if (in.eq.6) am=1.d0
      do 80 j=1,n
      n2=j2+j
   80 s(n2)=1.d0/am
      do 90 j=1,n
      n2=j2+j
      do 90 i=1,k
   90 a(i,j)=a(i,j)*s(n2)
c     compute column lengths with d.p. sums finally rounded to s.p.
c
c     (2)
c
  100 do 110 j=1,n
      n7=j7+j
      n2=j2+j
  110 s(n7)=s(n2)
c
c      s(m+1)-s(m+ n) contains variable permutations.
c
c     set permutation to identity.
c
      do 120 j=1,n
      n1=j1+j
  120 s(n1)=j
c
c     begin elimination on the matrix a with orthogonal matrices .
c
c     ip=pivot row
c
      do 250 ip=1,lm
c
c
      dp=0.d0
      km=ip
      do 140 j=ip,n
      sj=0.d0
      do 130 i=ip,k
      sj=sj+a(i,j)**2
  130 continue
      if (dp.gt.sj) go to 140
      dp=sj
      km=j
      if (in.eq.6) go to 160
  140 continue
c
c     maximize (sigma)**2 by column interchange.
c
c     supress column interchanges when in=6.
c
c
c     exchange columns if necessary.
c
      if (km.eq.ip) go to 160
      do 150 i=1,k
      a1=a(i,ip)
      a(i,ip)=a(i,km)
  150 a(i,km)=a1
c
c     record permutation and exchange squares of column lengths.
c
      n1=j1+km
      a1=s(n1)
      n2=j1+ip
      s(n1)=s(n2)
      s(n2)=a1
      n7=j7+km
      n8=j7+ip
      a1=s(n7)
      s(n7)=s(n8)
      s(n8)=a1
  160 if (ip.eq.1) go to 180
      a1=0.d0
      ipm1=ip-1
      do 170 i=1,ipm1
      a1=a1+a(i,ip)**2
  170 continue
      if (a1.gt.0.d0) go to 190
  180 if (dp.gt.0.d0) go to 200
c
c     test for rank deficiency.
c
  190 if (dsqrt(dp/a1).gt.eps1) go to 200
      if (in.eq.6) go to 200
      ii=ip-1
      if (erm) write (6,1140) irank,eps1,ii,ii
      irank=ip-1
      erm=.false.
      go to 260
c
c     (eps1) rank is deficient.
c
  200 sp=dsqrt(dp)
c
c     begin front elimination on column ip.
c
c     sp=sqroot(sigma**2).
c
      bp=1.d0/(dp+sp*abs(a(ip,ip)))
c
c     store beta in s(3n+1)-s(3n+1).
c
      if (ip.eq.k) bp=0.d0
      n3=k+2*n+ip
      r(n3)=bp
      up=dsign(dble(sp)+abs(a(ip,ip)),dble(a(ip,ip)))
      if (ip.ge.k) go to 250
      ipp1=ip+1
      if (ip.ge.n) go to 240
      do 230 j=ipp1,n
      sj=0.d0
      do 210 i=ipp1,k
  210 sj=sj+a(i,j)*a(i,ip)
      sj=sj+up*a(ip,j)
      sj=bp*sj
c
c     sj=yj now
c
      do 220 i=ipp1,k
  220 a(i,j)=a(i,j)-a(i,ip)*sj
  230 a(ip,j)=a(ip,j)-sj*up
  240 a(ip,ip)=-sign(sp,a(ip,ip))
c
      n4=k+3*n+ip
      r(n4)=up
  250 continue
      irank=lm
  260 irp1=irank+1
      irm1=irank-1
      if (irank.eq.0.or.irank.eq.n) go to 360
      if (ieq.eq.3) go to 290
c
c     begin back processing for rank deficiency case
c      if irank is less than n.
```

```
c
      do 280 j=1,n
      n2=j2+j
      n7=j7+j
      l=min0(j,irank)
c
c     unscale columns for rank deficient matrices when ieq.ne.3.
c
      do 270 i=1,l
  270 a(i,j)=a(i,j)/s(n7)
      s(n7)=1.d0
  280 s(n2)=1.d0
  290 ip=irank
  300 sj=0.d0
      do 310 j=irp1,n
      sj=sj+a(ip,j)**2
  310 continue
      sj=sj+a(ip,ip)**2
      aj=dsqrt(sj)
      up=dsign(aj+abs(a(ip,ip)),dble(a(ip,ip)))
c
c     ip th element of u vector calculated.
c
      bp=1.d0/(sj+abs(a(ip,ip))*aj)
c
c     bp = 2/length of u squared.
c
      ipm1=ip-1
      if (ipm1.le.0) go to 340
      do 330 i=1,ipm1
      dp=a(i,ip)*up
      do 320 j=irp1,n
      dp=dp+a(i,j)*a(ip,j)
  320 continue
      dp=dp/(sj+abs(a(ip,ip))*aj)
c
c     calc. (aj,u), where aj=jth row of a
c
      a(i,ip)=a(i,ip)-up*dp
c
c     modify array a.
c
      do 330 j=irp1,n
  330 a(i,j)=a(i,j)-a(ip,j)*dp
  340 a(ip,ip)=-dsign(aj,dble(a(ip,ip)))
c
c     calc. modified pivot.
c
```

```
c
c     save beta and ip th element of u vector in r array.
c
      n6=k+ip
      n7=k+n+ip
      r(n6)=bp
      r(n7)=up
c
c     test for end of back processing.
c
      if (ip-1) 360,360,350
  350 ip=ip-1
      go to 300
  360 if (in.eq.6) return
      do 370 j=1,k
  370 r(j)=b(j)
      it=0
c
c     set initial x vector to zero.
c
      do 380 j=1,n
  380 x(j)=0.d0
      if (irank.eq.0) go to 690
c
c     apply q to rt. hand side.
c
  390 do 430 ip=1,irank
      n4=k+3*n+ip
      sj=r(n4)*r(ip)
      ipp1=ip+1
      if (ipp1.gt.k) go to 410
      do 400 i=ipp1,k
  400 sj=sj+a(i,ip)*r(i)
  410 n3=k+2*n+ip
      bp=r(n3)
      if (ipp1.gt.k) go to 430
      do 420 i=ipp1,k
  420 r(i)=r(i)-bp*a(i,ip)*sj
  430 r(ip)=r(ip)-bp*r(n4)*sj
      do 440 j=1,irank
  440 s(j)=r(j)
      enorm=0.d0
      if (irp1.gt.k) go to 510
      do 450 j=irp1,k
  450 enorm=enorm+r(j)**2
      enorm=sqrt(enorm)
      go to 510
  460 do 480 j=1,n
```

```
      sj=0.d0
      n1=j1+j
      ip=s(n1)
      do 470 i=1,k
  470 sj=sj+r(i)*aa(i,ip)
c
c     apply at to rt. hand side.
c     apply scaling.
c
      n7=j2+ip
      n1=k+n+j
  480 r(n1)=sj*s(n7)
      n1=k+n
      s(1)=r(n1+1)/a(1,1)
      if (n.eq.1) go to 510
      do 500 j=2,n
      n1=j-1
      sj=0.d0
      do 490 i=1,n1
  490 sj=sj+a(i,j)*s(i)
      n2=k+j+n
  500 s(j)=(r(n2)-sj)/a(j,j)
c
c     entry to continue iterating.  solves tz = c = 1st irank
c     components of qb .
c
  510 s(irank)=s(irank)/a(irank,irank)
      if (irm1.eq.0) go to 540
      do 530 j=1,irm1
      n1=irank-j
      n2=n1+1
      sj=0.
      do 520 i=n2,irank
  520 sj=sj+a(n1,i)*s(i)
  530 s(n1)=(s(n1)-sj)/a(n1,n1)
c
c     z calculated.  compute x = sz.
c
  540 if (irank.eq.n) go to 590
      do 550 j=irp1,n
  550 s(j)=0.d0
      do 580 i=1,irank
      n7=k+n+i
      sj=r(n7)*s(i)
      do 560 j=irp1,n
      sj=sj+a(i,j)*s(j)
  560 continue
      n6=k+i
      do 570 j=irp1,n
  570 s(j)=s(j)-a(i,j)*r(n6)*sj
  580 s(i)=s(i)-r(n6)*r(n7)*sj
c
c     increment for x of minimal length calculated.
c
  590 do 600 i=1,n
  600 x(i)=x(i)+s(i)
      if (in.eq.7) go to 750
c
c     calc. sup norm of increment and residuals
c
      top1=0.d0
      do 610 j=1,n
      n2=j7+j
  610 top1= max(top1,abs(s(j))*s(n2))
      do 630 i=1,k
      sj=0.d0
      do 620 j=1,n
      n1=j1+j
      ip=s(n1)
      n7=j2+ip
  620 sj=sj+aa(i,ip)*x(j)*s(n7)
  630 r(i)=b(i)-sj
      if (itmax.le.0) go to 750
c
c     calc. sup norm of x.
c
      top=0.d0
      do 640 j=1,n
      n2=j7+j
  640 top= max(top,abs(x(j))*s(n2))
c
c     compare relative change in x with tolerance eps .
c
      if (top1-top*eps2) 690,650,650
  650 if (it-itmax) 660,680,680
  660 it=it+1
      if (it.eq.1) go to 670
      if (top1.gt..25*top2) go to 690
  670 top2=top1
      go to (390,460), isw
  680 it=0
  690 sj=0.d0
      do 700 j=1,k
      sj=sj+r(j)**2
  700 continue
      enorm=dsqrt(sj)
      if (irank.eq.n.and.isw.eq.1) go to 710
      go to 730
  710 enm1=enorm
c
c     save x array.
c
      do 720 j=1,n
      n1=k+j
  720 r(n1)=x(j)
      isw=2
      it=0
      go to 460
c
c     choose best solution
c
  730 if (irank.lt.n) go to 750
      if (enorm.le.enm1) go to 750
      do 740 j=1,n
      n1=k+j
  740 x(j)=r(n1)
      enorm=enm1
c
c     norm of ax - b located in the cell enorm .
c
c
c     rearrange variables.
c
  750 do 760 j=1,n
      n1=j1+j
  760 s(j)=s(n1)
      do 790 j=1,n
      do 770 i=j,n
      ip=s(i)
      if (j.eq.ip) go to 780
  770 continue
  780 s(i)=s(j)
      s(j)=j
      sj=x(j)
      x(j)=x(i)
  790 x(i)=sj
c
c     scale variables.
c
      do 800 j=1,n
      n2=j2+j
  800 x(j)=x(j)*s(n2)
      return
c
```

```
c     restore a.
c
  810 do 820 j=1,n
      n2=j2+j
      do 820 i=1,k
  820 a(i,j)=aa(i,j)
      return
c
c     generate solutions to the homogeneous equation ax = 0.
c
  830 if (irank.eq.n) return
      ns=n-irank
      do 840 i=1,n
      do 840 j=1,ns
  840 h(i,j)=0.d0
      do 850 j=1,ns
      n2=irank+j
  850 h(n2,j)=1.d0
      if (irank.eq.0) return
      do 870 j=1,irank
      do 870 i=1,ns
      n7=k+n+j
      sj=r(n7)*h(j,i)
      do 860 k1=irp1,n
  860 sj=sj+h(k1,i)*a(j,k1)
      n6=k+j
      bp=r(n6)
      dp=bp*r(n7)*sj
      a1=dp
      a2=dp-a1
      h(j,i)=h(j,i)-(a1+2.*a2)
      do 870 k1=irp1,n
      dp=bp*a(j,k1)*sj
      a1=dp
      a2=dp-a1
  870 h(k1,i)=h(k1,i)-(a1+2.*a2)
c
c     rearrange rows of solution matrix.
c
      do 880 j=1,n
      n1=j1+j
  880 s(j)=s(n1)
      do 910 j=1,n
      do 890 i=j,n
      ip=s(i)
      if (j.eq.ip) go to 900
  890 continue
  900 s(i)=s(j)
```

```
      s(j)=j
      do 910 k1=1,ns
      a1=h(j,k1)
      h(j,k1)=h(i,k1)
  910 h(i,k1)=a1
      return
c
 1140 format (/'warning. irank has been set to',i4,'  but(',1pe10.3,1 ')
      rank is',i4,'.  irank is now taken as ',i4)
      end
      subroutine potem (t, s, p, theta)
c
c=======================================================================
c     this subroutine calculates potential temperature as a function
c     of in-situ temperature, salinity, and pressure.
c
c     input [units]:
c       in-situ temperature (t): [degrees centigrade]
c       salinity (s): [per mil]
c       pressure (p): [decibars, approx. as meters of depth]
c     output [units]:
c       potential temperature (theta): [degrees centigrade]
c
c     references:
c        based on Fofonoff and Froese (1958) as shown in ...
c        Fofonoff, N., The Sea: Vol 1, (ed. M. Hill). Interscience,
c          New York, 1962, page 17, table iv.
c
c-----------------------------------------------------------------------
c
      implicit double precision (a-h,o-z)
c
c=======================================================================
c
      b1    = -1.60d-5*p
      b2    = 1.014d-5*p*t
      t2    = t*t
      t3    = t2*t
      b3    = -1.27d-7*p*t2
      b4    = 2.7d-9*p*t3
      b5    = 1.322d-6*p*s
      b6    = -2.62d-8*p*s*t
      s2    = s*s
      p2    = p*p
      b7    = 4.1d-9*p*s2
      b8    = 9.14d-9*p2
      b9    = -2.77d-10*p2*t
      b10   = 9.5d-13*p2*t2
      b11   = -1.557d-13*p2*p
      potmp = b1+b2+b3+b4+b5+b6+b7+b8+b9+b10+b11
      theta = t-potmp
c
      return
      end
      subroutine unesco (t, s, pin, rho)
c
c=======================================================================
c     this subroutine calculates the density of seawater using the
c     standard equation of state recommended by unesco(1981).
c
c     input [units]:
c       in-situ temperature (t): [degrees centigrade]
c       salinity (s): [practical salinity units]
c       pressure (pin): [decibars, approx. as meters of depth]
c     output [units]:
c       density(rho): kilograms per cubic meter
c
c     references:
c
c        Gill, A. 1982   Atmosphere-Ocean Dynamics: International
c         Geophysical Series No. 30. Academic Press, London, pp 599-600.
c        UNESCO. 1981  10th report of the joint panel on oceanographic
           tables and standards.
           UNESCO Tech. Papers in Marine Sci. No. 36, Paris.
c
c-----------------------------------------------------------------------
c
      implicit double precision (a-h,o-z)
c
c=======================================================================
c
      c1p5 = 1.5d0
c
c  convert from depth [m] (decibars) to bars
      p = pin * 1.0d-1
c
      rw =     9.99842594d2 + 6.793952d-2*t - 9.095290d-3*t**2
     $        + 1.001685d-4*t**3 - 1.120083d-6*t**4 + 6.536332d-9*t**5
c
      rsto =   rw + (8.24493d-1 - 4.0899d-3*t + 7.6438d-5*t**2
     $        - 8.2467d-7*t**3 + 5.3875d-9*t**4) * s
     $       + (-5.72466d-3 + 1.0227d-4*t - 1.6546d-6*t**2) * s**c1p5
     $       + 4.8314d-4 * s**2
c
      xkw =     1.965221d4 + 1.484206d2*t - 2.327105d0*t**2 +
     $          1.360477d-2*t**3 - 5.155288d-5*t**4
```

```
c
      xksto =   xkw + (5.46746d1 - 6.03459d-1*t + 1.09987d-2*t**2
     $         - 6.1670d-5*t**3) * s
     $        + (7.944d-2 + 1.6483d-2*t - 5.3009d-4*t**2) * s**c1p5
c
      xkstp =   xksto + (3.239908d0 + 1.43713d-3*t + 1.16092d-4*t**2
     $         - 5.77905d-7*t**3) * p
     $        + (2.2838d-3 - 1.0981d-5*t - 1.6078d-6*t**2) * p * s
     $        + 1.91075d-4 * p * s**c1p5
     $        + (8.50935d-5 - 6.12293d-6*t + 5.2787d-8*t**2) * p**2
     $        + (-9.9348d-7 + 2.0816d-8*t + 9.1697d-10*t**2) * p**2 * s
c
      rho =    rsto / (1.0d0 - p/xkstp)
c
      return
      end
#ifndef gfdl_den
      function fnpz(z,xlat)
      implicit double precision (a-h,o-z)
      parameter (mloop=30,mconv=5,eps=1d-6)
c
c  function to calculate pressure in decibars from depth in metres using
c  an iterative inverse of saunders algorithm (function fnpz).  iterates
c  until the error is zero, a limit cycle is detected of 'mloop'
c  iterations reached.  error exit if error > eps.  array pa used to
c  detect a limit cycle.
c
c  check value fnpz = 10302.423165          - cray 64-bit
c                   = 10302.4231650052      - ieee 64-bit.
c
      dimension pa(mconv)
c
      p  = z
      ia = 0
      do 20 i=1,30
      zz = fnzp(p,xlat)
c zero error
      if(z.eq.zz)goto 50
      ee = z - zz
      ea = abs(ee)
c save new best value
      if(ia.eq.0.or.ea.lt.ep)then
        ia = 1
        ep = ea
        pa(ia) = p
c look for limit cycle
      elseif(ea.eq.ep)then
        do 40 j=1,ia
```

```
         if(p.eq.pa(j))goto 50
   40    continue
         if(ia.lt.mconv)then
           ia = ia + 1
           pa(ia) = p
         endif
       endif
c correct p and loop
       p  = p + ee
  20   continue
c
       if(ea.gt.eps)then
       print *,'subroutine fnpz.  iteration has not converged after',
      &        ' 30 iterations'
       print *,'object depth =',z
       print *,'latest p = ',p,'.   corresponding z = ',zz
       print *,'minimum error = ',ea
       print *,'number of corresponding ps =',ia
       print *,'pa array',(pa(k),k=1,ia)
       stop
       endif
c
       p = pa(ia)
  50   fnpz = p
       return
       end

       function fnzp(pin,xlat)
       implicit double precision (a-h,o-z)
c
c  function to transform pressure to depth using the method of
c  p.m.saunders, 1981.  journal of physical oceanography, 11, 573-574.
c
c  input:  pin = pressure in decibars ("oceanographic" pressure
c                equals absolute pressure minus one atmosphere).
c          xlat= latitude in degrees.
c
c  output: fnzp = depth in metres.
c
c  check value: fnzp = 9712.478325455        - cray 64-bit
c                    = 9712.4783254538,      - ieee 64-bit.
c  for: pin=10000.0, xlat=30.0.
c
       data in/0/
       save in
c
c 1.   calculate constants
c
```

```
       if(in.eq.0)then
         in = 1
         pi = 3.141592654d0
         radian = pi/180d0
         g1 =  9.780318d0
         g2 =  9.780318d0*(5.3024d-3 - 5.9d-6*4.0e0)
         g3 = -9.780318d0*5.9d-6 * 4.0d0
c
c  a1 = specific volume at (t=0,s=35,p=0) times 10**5
c  rk = constant coeficient
c  ra = term proportional to p
c  rb = term proportional to p**2
c
         s = 35.0d0
         c1p5 = 1.5d0
         a1 = 1d5/(9.99842594d2 + 8.24493d-1*s
     &           - 5.72466d-3*s**c1p5 + 4.8314d-4*s**2)
         rk = 1.965221d4 + 5.46746d1*s + 7.944d-2*s**c1p5
         ra = 3.239908d0 + 2.2838d-3*s + 1.91075d-4*s**c1p5
         rb = 8.50935d-5 - 9.9348d-7*s
         dd = sqrt(ra*ra - 4.0d0*rk*rb)
         c1 = 0.5d0/rb
         c2 = ra/rk
         c3 = rb/rk
         c4 = ra/(2.0d0*rb*dd)
         c5 = 2.0d0*rb/(ra - dd)
         c6 = 2.0d0*rb/(ra + dd)
         c7 = 0.5d0*2.226d-6
      endif
c
c 2.  calculate gravity
c
      x  = sin(radian*xlat)**2
      gs = (g3*x + g2)*x + g1
c  convert from pressure in decibars to bars
      p  = pin*1.0d-1
c
c 3.  integrate specific volume
c
      r1 = a1*(p - c1*log((c3*p + c2)*p+1.0d0) + c4*log((1.0d0 + c5*p)
     &        /(1.0d0 + c6*p)))
      fnzp = r1/(gs + c7*pin)
c
      return
      end

      function tadiab(tt,ss,p0,p1,dpp)
      implicit double precision (a-h,o-z)
```

```
c
c  subroutine to calculate the final temperature of water moved
c  adiabatically from an initial temperature tt, salinity ss and
c  pressure p0, to a final pressure p1.
c
c  the integral equation is solved by direct integration with a pressure
c  increment dpp - using the bryden equation for the adiabatic lapse
c  rate (subrouitne atg).
c
c    t = initial temperature in degrees centigrade.
c    s = salinity in nsu.
c    p0= initial pressure in decibars.
c    p1= final pressure in decibars.
c    dpp=pressure step.
c    tadiab = final temperature in degrees centigrade.
c
c  pressures are "oceanographic" pressures, equal to absolute pressures
c  minus one atmosphere.
c  tests with dpp values ranging from 1 to 128 decibars showed the most
c  accurate results were obtained with dpp equal to 1.
c
c  check value: tadiab = 43.26663196648         - cray 64-bit
c                      = 43.266631967051,       - ieee 64-bit.
c  for: t=40.0, s=40.0, p0=0.0, p1=10000.0, dpp=1.0.
c
      if(p0.lt.0.0d0 .or. p0.gt.20000.0d0
     &   .or.p1.lt.0.0d0 .or. p1.gt.20000.0d0)then
        print *,' subroutine tadiab stopping - pressures out of range'
        print *,' pressures p0 and p1 = ',p0,p1
        print *,' allowed range has min of 0.0, max of 20,000'
        stop
      endif
c
      dp = sign(dpp,p1-p0)
      p  = p0
      t  = tt
      tb = t  - atg(p0,t,ss)*dp
c
 10   ta = tb + 2.0d0*atg(p,t,ss)*dp
      p  = p  + dp
      tb = t
      t  = ta
      test = (p - p1)*(p - dp - p1)
      if(test.gt.0d0)goto 10
      tadiab = ((p1 - p + dp)*t + (p - p1)*tb)/dp
      return
      end
      function atg(p,t,s)
      implicit double precision (a-h,o-z)
c
c  adiabatic temperature gradient deg c per decibar
c  ref: bryden,h., 1973, deep-sea res., 20, 401-408
c  units:
c        pressure        p        decibars
c        temperature     t        deg celcius (ipts-68)
c        salinity        s        (pss-78)
c        adiabatic       atg      degrees celcius per decibar
c
c  pressure is "oceanographic" pressure equal to absolute pressure
c  minus one atmosphere.
c
c  check value: atg = 3.2559758                          - cray 64-bit
c      .            = 3.2559758000000d-04 deg c/dbar - ieee 64-bit.
c  for: p=10000.0, t=40.0, s=40.0.
c
      ds = s-35d0
      atg = (((-2.1687d-16*t + 1.8676d-14)*t - 4.6206d-13)*p
     &    + ((2.7759d-12*t - 1.1351d-10)*ds + ((-5.4481d-14*t
     &    + 8.733d-12)*t - 6.7795d-10)*t + 1.8741d-8))*p
     &    + (-4.2393d-8*t + 1.8932d-6)*ds
     &    + ((6.6228d-10*t - 6.836d-8)*t + 8.5258d-6)*t + 3.5803d-5
      return
      end
#endif
```

APPENDIX II

state.F

```
      subroutine state (t, s, rho)
#ifdef multitasking
cfpp$ noconcur r
#endif
c
c=======================================================================
c
c     state computes one row of normalized densities by using a 3rd
c     order polynomial fit to the knudsen formula, for each level
c     subtract normalizing constants from temperature and salinity
c     and compute polynomial approximation of knudsen density.
c
c     note.. for precision purposes, there is a depth dependent
c     constant subtracted from the density returned by this routine.
c     so... this routine should be used only for horizontal gradients
c     of density.
c
c     inputs:
c
c     t = the input row of temperatures
c     s = the input row of salinities (units: (ppt-35)/1000)
c
c     output:
c
c     rho = normalized densities
c
c=======================================================================
c
c
#include "param.h"
c
      dimension t(imt,km), s(imt,km), rho(imt,km)
      common /cstate/ to(km), so(km), c(km,9), z0(km), rho0(km)
c
#include "dncoef.h"
c
c
```

```
c-----------------------------------------------------------------------
c     statement function
c-----------------------------------------------------------------------
c
      dens (tq, sq, k) = (c(k,1) + (c(k,4) + c(k,7)*sq)*sq +
     $                   (c(k,3) + c(k,8)*sq + c(k,6)*tq)*tq)*tq +
     $                   (c(k,2) + (c(k,5) + c(k,9)*sq)*sq)*sq
c
c-----------------------------------------------------------------------
c
      do 100 k=1,km
        do 90 i=1,imt
          rho(i,k) = dens (t(i,k)-to(k), s(i,k)-so(k), k)
90      continue
100   continue
c
      return
c
c
c
c
      entry statec (t, s, rho, ind)
c
c
c=======================================================================
c
c     statec computes, for one row, the normalized densities by using
c     a 3rd order polynomial fit to the knudsen formula. For
c     purposes of checking vertical stability between adjacent
c     levels, the reference depth for pressure dependence in
c     the knudsen formula must be held constant. that level is
c     determined by "ind".
c
c     inputs:
c
c     t   = the input row of temperatures
c     s   = the input row of salinities (units: (ppt-35)/1000)
c     ind = 1 for comparing levels 1 to 2, 3 to 4, etc.
c           (coefficients for the lower of the 2 levels are used)
c           2 for comparing levels 2 to 3, 4 to 5, etc.
c           (coefficients for the lower of the 2 levels are used)
c
c     output:
c
c     rho = normalized densities
c
c=======================================================================
c
```

```
c
      if (ind .lt. 1 .or. ind .gt. 2) then
        write (stderr,99)
        stop '1 statec'
      endif
c
      do 200 l=1,km,2
        if (ind .eq. 1) then
          k = min(l+1,km)
        else
          k = 1
        endif
        do 190 i=1,imt
          rho(i,l) = dens (t(i,l)-to(k), s(i,l)-so(k), k)
190     continue
200   continue
c
      do 300 l=2,km,2
        if (ind .eq. 1) then
          k = 1
        else
          k = min(l+1,km)
        endif
        do 290 i=1,imt
          rho(i,l) = dens (t(i,l)-to(k), s(i,l)-so(k), k)
290     continue
300   continue
      return
   99 format(/' error => bad "ind" in statec: ind =',i10)
      end
```

APPENDIX III:

dncoef.h

The include file dncoef.h now appears in the following form:

```
c===================== include file "dncoef.h" =======================
c
c     normalized temperatures, salinities and coefficients
c     generated by program "eqstat" which fits 3rd order
c     polynomials to the equation of state for each model level.
c
c     The polynominal returns density deviations in g/cm**3 from
c     "rho0" for each level. The polynominal is in powers of
c     (t-to) and (s-so), where t is potential temperature,
c     measured in degrees C and s is salinity measured in
c     model units. The arrays rho0 and z0, where z0 contains
c     the depth in centimetres of each level, are stored for
c     reference.
c
      data z0( 1), rho0( 1), to( 1), so( 1), (c( 1,n),n=1,9)/
     $            10.35000E02,    1.0245946,    13.4986130,-.2250000E-02,
     $          -.2017008E-03,0.7730203E+00,-.4930029E-05,-.2021526E-02,
     $          0.1678596E+00,0.3608601E-07,0.3776118E-02,0.3602963E-04,
     $          0.1609481E+01/
c
      data z0( 2), rho0( 2), to( 2), so( 2), (c( 2,n),n=1,9)/
     $            32.35000E02,    1.0246937,    13.4956607,-.2250000E-02,
     $          -.2021070E-03,0.7728720E+00,-.4923108E-05,-.2019249E-02,
     $          0.1681032E+00,0.3601443E-07,0.3770950E-02,0.3599568E-04,
     $          0.1609324E+01/
c
.
.
.
c
      data z0(32), rho0(32), to(32), so(32), (c(32,n),n=1,9)/
     $          5382.50000E02,    1.0518755,     2.9330675,-.2000000E-03,
     $          -.2294241E-03,0.7561387E+00,-.3894801E-05,-.2015824E-02,
     $          0.2060329E+00,0.3214992E-07,0.3008361E-02,0.3937013E-04,
     $          0.1602931E+01/
```

Brook Road, Wormley, Godalming
Surrey, GU8 5UB,
United Kingdom
Telephone +44 (0) 428-684141
Facsimile +44 (0) 428-683066
Telex 858833 OCEANS G

Natural
Environment
Research
Council