

```

1 | #include "cppdefs.h"
2 | -----
3 | !BOP
4 | !
5 | ! MODULE: eqstate --- the equation of state \label{sec:eqstate}
6 | !
7 | ! INTERFACE:
8 | !   MODULE eqstate
9 | !
10 | ! DESCRIPTION:
11 | !   Computes the density,  $\rho$ , and buoyancy from the
12 | !   salinity,  $S$ , the temperature,  $\Theta$ , and the thermodynamic
13 | !   pressure,  $P$ , according to an \emph{equation of state},
14 | !   \begin{equation}
15 | !     \label{DefEOS}
16 | !     \rho = \hat{\rho}(S, \Theta, P)
17 | !   \end{equation}
18 | !   \point
19 | !
20 | !   The following remark on the thermodynamic interpretation of
21 | !   density, temperature, and pressure is useful here. If  $\Theta$  is
22 | !   identified with the in-situ temperature, and  $P$  with the in-situ
23 | !   pressure, then  $\rho$  will be the in-situ density. On the
24 | !   other hand, if  $P$  is identified with the surface pressure, and
25 | !    $\Theta$  with the potential temperature, the same equation of
26 | !   state, \eq{DefEOS}, will yield  $\rho$  as the potential
27 | !   density. Note that the quantity  $\sigma_t$  found in the GOTM output
28 | !   is simply computed from  $\rho - 1000 \text{ kg m}^{-3}$ , and may
29 | !   therefore adopt different meanings.
30 | !
31 | !
32 | !   At present, two different models for the equation of state ("modes"),
33 | !   and four different "methods" how to evaluate the equation of state
34 | !   are implemented.
35 | !
36 | !   Modes:
37 | !   \begin{enumerate}
38 | !     \item The UNESCO equation of state according to \cite{FofonoffMillard83}
39 | !     \item The \cite{JACKETTea05} equation of state
40 | !   \end{enumerate}
41 | !   Methods:
42 | !   \begin{enumerate}
43 | !     \item the full equation of state --- including pressure effects
44 | !     \item the full equation of state --- without pressure effects
45 | !     \item the linearised equation of state
46 | !     \item a general linear form of the equation of state
47 | !   \end{enumerate}
48 | !
49 | ! USES:
50 | !   IMPLICIT NONE
51 | !
52 | !   default: all is private.
53 | !   private
54 | !
55 | ! !PUBLIC MEMBER FUNCTIONS:
56 | !   public init_eqstate, eqstate1, eos_alpha, eos_beta, unesco, rho_feistel
57 | !
58 | !   interface init_eqstate
59 | !     module procedure init_eqstate_nml
60 | !     module procedure init_eqstate_yaml
61 | !   end interface
62 | !
63 | !
64 | ! !REVISION HISTORY:
65 | !   Original author(s): Hans Burchard & Karsten Bolding
66 | !
67 | !EOP
68 | !
69 | !   private data members
70 | !   integer, public          :: eq_state_method, eq_state_mode
71 | !   REALTYPE                :: T0, S0, p0, dtr0, dsr0
72 | !   logical                  :: init_linearised
73 | !
74 | ! -----
75 | !
76 | !   contains
77 | ! -----
78 | !
79 | !BOP
80 | !
81 | ! !IROUTINE: Read the namelist {\tt eqstate}
82 | !
83 | ! !INTERFACE:
84 | !   subroutine init_eqstate_nml(namlst)
85 | !
86 | ! !DESCRIPTION:
87 | !   Here, the namelist {\tt eqstate} in the namelist file {\tt gotmrun.nml}
88 | !   is read.
89 | !
90 | ! !USES:
91 | !   IMPLICIT NONE
92 | !
93 | ! !INPUT PARAMETERS:
94 | !   integer, intent(in)          :: namlst
95 | !
96 | ! !REVISION HISTORY:
97 | !   Original author(s): Hans Burchard & Karsten Bolding
98 | !

```

```

99 |EOP
100 |
101 | !LOCAL VARIABLES:
102 |   namelist /eqstate/ eq_state_mode,eq_state_method,T0,S0,p0,dtr0,dsr0
103 |
104 | -----
105 |BOC
106 |   LEVEL1 'init_eqstate_nml'
107 |
108 |   init_linearised = .true.
109 |   read(namlst,nml=eqstate,err=80)
110 |   LEVEL2 'done.'
111 |   return
112 |   80 FATAL 'I could not read "eqstate" namelist'
113 |   stop 'init_eqstate'
114 | end subroutine init_eqstate_nml
115 |EOC
116 |
117 | -----
118 |BOP
119 |
120 | !IROUTINE: Read the yaml configuration {\tt eqstate}
121 |
122 | !INTERFACE:
123 |   subroutine init_eqstate_yaml(branch)
124 |
125 | !DESCRIPTION:
126 |
127 | !USES:
128 |   use yaml_settings
129 |   IMPLICIT NONE
130 |
131 | !INPUT PARAMETERS:
132 |   class (type_settings), intent(inout) :: branch
133 |
134 | !REVISION HISTORY:
135 |   Original author(s): Hans Burchard & Karsten Bolding
136 |
137 |EOP
138 |
139 | !LOCAL VARIABLES:
140 |   integer, parameter :: rk = kind(_ONE_)
141 |   class (type_settings), pointer :: twig
142 |
143 | -----
144 |BOC
145 |   LEVEL1 'init_eqstate_yaml'
146 |   call branch%get(eq_state_mode, 'method', 'method', default=2, &
147 |     options=(/option(1, 'UNESCO', 'UNESCO'), option(2, 'Jackett et al. (2005)', 'Jackett'))/)
148 |   call branch%get(eq_state_method, 'form', 'formulation', &
149 |     options=(/option(1, 'full', 'full'), option(2, 'full with buoyancy frequency based on surface press
150 |     ure', 'full-pot'), &
151 |     option(3, 'linearized at T0,S0,p0', 'linear'), option(4, 'linearized at T0,S0,p0,dtr0,dsr0', 'linea
152 |     r_custom'))/, default=1)
153 |   twig => branch%get_child('linear')
154 |   call twig%get(T0, 'T0', 'reference temperature', 'Celsius', &
155 |     minimum=-2._rk, default=10._rk)
156 |   call twig%get(S0, 'S0', 'reference salinity', 'psu', &
157 |     minimum=0._rk, default=35._rk)
158 |   call twig%get(p0, 'p0', 'reference pressure', 'Pa', &
159 |     default=0._rk)
160 |   call twig%get(dtr0, 'dtr0', 'thermal expansion coefficient', 'kg/m^3/K', &
161 |     default=-0.17_rk)
162 |   call twig%get(dsr0, 'dsr0', 'saline expansion coefficient', 'kg/m^3/psu', &
163 |     default=0.78_rk)
164 |
165 |   init_linearised = .true.
166 |   LEVEL2 'done.'
167 |   end subroutine init_eqstate_yaml
168 |EOC
169 |
170 | -----
171 |BOP
172 |
173 | !IROUTINE: Select an equation of state
174 |
175 | !INTERFACE:
176 |   REALTYPE function eqstate1(S,T,p,g,rho_0)
177 |
178 | !DESCRIPTION:
179 |   Calculates the in-situ buoyancy according to the selected method.
180 |   {\tt S} is salinity $S$ in psu, {\tt T} is
181 |   potential temperature $\theta$ in $^\circ$C (ITS-90), {\tt p} is
182 |   gauge pressure (absolute pressure - 10.1325 bar), {\tt g} is the
183 |   gravitational acceleration in m/s^2 and {\tt rho_0} the reference
184 |   density in kg/m^3. {\tt eqstate1} is the in-situ-density
185 |   in kg/m^3.
186 |   For {\tt eq_state_method}=1, the UNESCO equation of state is used,
187 |   for {\tt eq_state_method}=2, the \cite{JACKETTTea05} equation
188 |   of state is used. Here, some care is needed, since the UNESCO equation
189 |   used bar for pressure and the \cite{JACKETTTea05} uses dbar for pressure.
190 |   For values of
191 |   {\tt eq_state_method} ranging from 1 to 4, one of the following methods
192 |   will be used.
193 |
194 |   \begin{enumerate}
195 |     \item the full equation of state for sea water
196 |       including pressure dependence.

```

```

195 !      \item the equation of state for sea water
196 !          with the pressure evaluated at the sea surface as
197 !          reference level. This is the choice
198 !          for computations based on potential temperature and density.
199 !      \item a linearised equation of state.
200 !          The parameters {\tt T0},
201 !          {\tt S0} and {\tt p0} have to be specified in the namelist.
202 !      \item a linear equation of state with prescribed {\tt rho0}, {\tt T0},
203 !          {\tt S0}, {\tt dtr0}, {\tt dsr0} according to
204 !          \begin{equation}
205 !              \label{eosLinear}
206 !              \rho = \rho_0 + \text{\tt dtr0} (T - T_0)
207 !                      + \text{\tt dsr0} (S - S_0)
208 !          \end{equation}
209 !      \end{enumerate}
210 !
211 !
212 !
213 ! !USES:
214 ! IMPLICIT NONE
215 !
216 ! !INPUT PARAMETERS:
217 ! REALTYPE,intent(in)          :: S,T,p
218 ! REALTYPE,optional,intent(in) :: g,rho_0
219 !
220 ! !REVISION HISTORY:
221 ! Original author(s): Hans Burchard & Karsten Bolding
222 !
223 !EOP
224 !
225 ! !LOCAL VARIABLES:
226 ! REALTYPE          :: x
227 ! REALTYPE, save    :: rh0,dtr,dsr
228 ! REALTYPE          :: dTT,dSS
229 ! logical           :: press
230 !
231 !-----
232 !BOC
233 select case (eq_state_mode)
234 case(1)
235     select case (eq_state_method)
236     case (1)
237         press=.true.
238         x=unesco(S,T,p,press)
239     case (2)
240         press=.false.
241         x=unesco(S,T,p,press)
242     case (3)
243         if (init_linearised) then
244             press=.true. ! This allows for choosing potentials other than p=0
245             dTT=0.001
246             dSS=0.001
247             rh0= unesco(S0,T0,p0,press)
248             dtr=(unesco(S0,T0+0.5*dTT,p0,press) &
249                 -unesco(S0,T0-0.5*dTT,p0,press))/dTT
250             dsr=(unesco(S0+0.5*dSS,T0,p0,press) &
251                 -unesco(S0-0.5*dSS,T0,p0,press))/dSS
252             init_linearised=.false.
253         end if
254         x=rh0+dtr*(T-T0)+dsr*(S-S0)
255     case (4)
256         x=rho_0+dtr0*(T-T0)+dsr0*(S-S0)
257     case default
258     end select
259 case(2)
260     select case (eq_state_method)
261     case (1)
262         press=.true.
263         x=rho_feistel(S,T,p*10.,press)
264     case (2)
265         press=.false.
266         x=rho_feistel(S,T,p*10.,press)
267     case (3)
268         if (init_linearised) then
269             press=.true. ! This allows for choosing potentials other than p=0
270             dTT=0.001
271             dSS=0.001
272             rh0= rho_feistel(S0,T0,p0*10.,press)
273             dtr=(rho_feistel(S0,T0+0.5*dTT,p0*10.,press) &
274                 -rho_feistel(S0,T0-0.5*dTT,p0*10.,press))/dTT
275             dsr=(rho_feistel(S0+0.5*dSS,T0,p0*10.,press) &
276                 -rho_feistel(S0-0.5*dSS,T0,p0*10.,press))/dSS
277             init_linearised=.false.
278         end if
279         x=rh0+dtr*(T-T0)+dsr*(S-S0)
280     case (4)
281         x=rho_0+dtr0*(T-T0)+dsr0*(S-S0)
282     case default
283     end select
284 case default
285 end select
286
287 eqstate1=-g*(x-rho_0)/rho_0
288
289 return
290 end function eqstate1
291 !EOC
292

```

```

293 |-----
294 !BOP
295 !
296 ! !ROUTINE: Compute thermal expansion coefficient
297 !
298 ! !INTERFACE:
299 !   REALTYPE function eos_alpha(S,T,p,g,rho_0)
300 !
301 ! !DESCRIPTION:
302 !   Computes the thermal expansion coefficient defined by
303 !   \begin{equation}
304 !     \label{eosAlpha}
305 !     \alpha =
306 !       - \frac{1}{\rho_0}
307 !       \left( \frac{\partial \rho_{is}}{\partial T} \right)_S
308 !   =
309 !       \frac{1}{g}
310 !       \left( \frac{\partial B_{is}}{\partial T} \right)_S
311 !     ,
312 !   \end{equation}
313 !   where $B_{is}$ denotes the in-situ buoyancy. The computation is carried
314 !   out by a finite difference approximation of \eq{eosAlpha},
315 !   requiring two evaluations of the equation of state.
316 !   Note, that comparing \eq{eosAlpha} with \eq{eosLinear} it follows that
317 !   $\alpha = - \text{\tt dtr0}/\rho_0$.
318 !
319 ! !USES:
320 !   IMPLICIT NONE
321 !
322 ! !INPUT PARAMETERS:
323 !   REALTYPE,intent(in)           :: S,T,p
324 !   REALTYPE,optional,intent(in)  :: g,rho_0
325 !
326 ! !REVISION HISTORY:
327 !   Original author(s): Lars Umlauf
328 !
329 !EOP
330 !
331 ! !LOCAL VARIABLES:
332 !
333 !   REALTYPE,parameter           :: delta = 0.01
334 !   REALTYPE                     :: buoy_a,buoy_b
335 |-----
336 !BOC
337
338     buoy_a = eqstate1(S,T+0.5*delta,p,g,rho_0)
339     buoy_b = eqstate1(S,T-0.5*delta,p,g,rho_0)
340
341     eos_alpha = (buoy_a - buoy_b) / (g*delta)
342
343   end function eos_alpha
344 !EOC
345
346
347 |-----
348 !BOP
349 !
350 ! !ROUTINE: Compute saline contraction coefficient
351 !
352 ! !INTERFACE:
353 !   REALTYPE function eos_beta(S,T,p,g,rho_0)
354 !
355 ! !DESCRIPTION:
356 !   Computes the saline contractioncoefficient defined by
357 !   \begin{equation}
358 !     \label{eosBeta}
359 !     \beta =
360 !       \frac{1}{\rho_0}
361 !       \left( \frac{\partial \rho_{is}}{\partial S} \right)_T
362 !   =
363 !       - \frac{1}{g}
364 !       \left( \frac{\partial B_{is}}{\partial S} \right)_T
365 !     ,
366 !   \end{equation}
367 !   where $B_{is}$ denotes the in-situ buoyancy. The computation is carried
368 !   out by a finite difference approximation of \eq{eosBeta},
369 !   requiring two evaluations of the equation of state.
370 !   Note, that comparing \eq{eosBeta} with \eq{eosLinear} it follows that
371 !   $\beta = \text{\tt dsr0}/\rho_0$.
372 !
373 !
374 ! !USES:
375 !   IMPLICIT NONE
376 !
377 ! !INPUT PARAMETERS:
378 !   REALTYPE,intent(in)           :: S,T,p
379 !   REALTYPE,optional,intent(in)  :: g,rho_0
380 !
381 ! !REVISION HISTORY:
382 !   Original author(s): Lars Umlauf
383 !
384 !EOP
385 !
386 ! !LOCAL VARIABLES:
387 !
388 !   REALTYPE,parameter           :: delta = 0.01
389 !   REALTYPE                     :: buoy_a,buoy_b
390 |-----

```

```

391 !BOC
392
393     buoy_a    = eqstate1(S+0.5*delta,T,p,g,rho_0)
394     buoy_b    = eqstate1(S-0.5*delta,T,p,g,rho_0)
395
396     eos_beta = -(buoy_a - buoy_b) / (g*delta)
397
398 end function eos_beta
399 !EOC
400
401 -----
402 !
403 !BOP
404 !
405 ! !IROUTINE: The UNESCO equation of state
406 !
407 ! !INTERFACE:
408     REALTYPE function unesco(S,T,p,UNPress)
409 !
410 ! !DESCRIPTION:
411 ! Computes the in-situ density in \eq{DefEOS} according to the
412 ! UNESCO equation of state for sea water (see \cite{FofonoffMillard83}).
413 ! The pressure
414 ! dependence can be switched on ({\tt UNPress=.true.}) or off
415 ! ({\tt UNPress=.false.}). {\tt S} is salinity $$ in psu, {\tt T} is
416 ! potential temperature $\theta$ in $^{\circ}$C (ITS-90), {\tt p} is
417 ! gauge pressure (absolute pressure - 10.1325 bar) and
418 ! {\tt unesco} is the in-situ density in kg\,m$^{-3}$.
419 ! The check value is {\tt unesco(35,25,1000) = 1062.53817} .
420 !
421 ! !USES:
422     IMPLICIT NONE
423 !
424 ! !INPUT PARAMETERS:
425     REALTYPE, intent(in)           :: S,T,p
426     LOGICAL, intent(in)            :: UNPress
427 !
428 ! !REVISION HISTORY:
429 ! Original author(s): Hans Burchard & Karsten Bolding
430 !
431 !EOP
432 !
433 ! !LOCAL VARIABLES:
434     REALTYPE           :: x,K
435     REALTYPE           :: T2,T3,T4,T5,S15,S2,S3,p2
436 !
437 !-----
438 !BOC
439     T2 = T*T
440     T3 = T*T2
441     T4 = T2*T2
442     T5 = T*T4
443     S15= S**1.5
444     S2 = S*S
445     S3 = S*S2
446
447     x=999.842594+6.793952e-02*T-9.09529e-03*T2+1.001685e-04*T3
448     x=x-1.120083e-06*T4+6.536332e-09*T5
449     x=x+S*(0.824493-4.0899e-03*T+7.6438e-05*T2-8.2467e-07*T3)
450     x=x+S*5.3875e-09*T4
451     x=x+sqrt(S3)*(-5.72466e-03+1.0227e-04*T-1.6546e-06*T2)
452     x=x+4.8314e-04*S2
453
454     if ((UNPress).and.(p.gt.0)) then
455         p2=p*p
456         K= 19652.21
457             +148.4206      *T      -2.327105      *T2      &
458             + 1.360477E-2*T3      -5.155288E-5      *T4      &
459             + 3.239908      *p      +1.43713E-3      *T *p      &
460             + 1.16092E-4      *T2*p      -5.77905E-7      *T3*p      &
461             + 8.50935E-5      *p2      -6.12293E-6      *T *p2      &
462             + 5.2787E-8      *T2*p2      &
463             + 54.6746      *S      -0.603459      *T      *S      &
464             + 1.09987E-2      *T2      *S      -6.1670E-5      *T3      *S      &
465             + 7.944E-2      *S15+1.6483E-2      *T      *S15      &
466             - 5.3009E-4      *T2      *S15+2.2838E-3      *p      *S      &
467             - 1.0981E-5      *T *p      *S      -1.6078E-6      *T2*p      *S      &
468             + 1.91075E-4      *p      *S15-9.9348E-7      *p2*S      &
469             + 2.0816E-8      *T *p2*S      +9.1697E-10      *T2*p2*S      &
470         x=x/(1.-p/K)
471     end if
472
473     unesco=x
474     return
475 end function unesco
476 !EOC
477 !-----
478 !BOP
479 !
480 ! !IROUTINE: The \cite{JACKETTea05} equation of state
481 !
482 ! !INTERFACE:
483     REALTYPE function rho_feistel(s,th,p,UNPress)
484 !
485 ! !DESCRIPTION:
486 ! Computes the in-situ density in \eq{DefEOS} according to the
487 ! \cite{JACKETTea05} equation of state for sea water, which is based
488 ! on the Gibbs potential developed by \cite{FEISTEL03}. The pressure

```

```

489! dependence can be switched on ({\tt UNPress=.true.}) or off
490! ({\tt UNPress=.false.}). {\tt s} is salinity $$ in psu, {\tt th} is
491! potential temperature $\theta$ in $\circ$C (ITS-90), {\tt p} is
492! gauge pressure (absolute pressure - 10.1325 dbar) and
493! {\tt rho\_feistel} is the in-situ density in kg\,m$^{-3}$.
494! The check value is {\tt rho\_feistel(20,20,1000) = 1017.728868019642} .
495!
496! !USES:
497! IMPLICIT NONE
498!
499! !INPUT PARAMETERS:
500! REALTYPE, intent(in)          :: s,th,p
501! LOGICAL, intent(in)           :: UNPress
502!
503! !REVISION HISTORY:
504! Original author(s): Hans Burchard & Karsten Bolding
505!
506!EOP
507!
508! !LOCAL VARIABLES:
509! REALTYPE                      :: th2,sqrts,pth,anum,aden
510!
511!-----
512!BOC
513
514th2 = th*th
515sqrts = sqrt(s)
516
517
518anum =          9.9984085444849347d+02 + &
519          th*( 7.3471625860981584d+00 + &
520          th*(-5.3211231792841769d-02 + &
521          th* 3.6492439109814549d-04)) + &
522          s*( 2.5880571023991390d+00 - &
523          th* 6.7168282786692355d-03 + &
524          s* 1.9203202055760151d-03)
525
526aden =          1.0000000000000000d+00 + &
527          th*( 7.2815210113327091d-03 + &
528          th*(-4.4787265461983921d-05 + &
529          th*( 3.3851002965802430d-07 + &
530          th* 1.3651202389758572d-10))) + &
531          s*( 1.7632126669040377d-03 - &
532          th*( 8.8066583251206474d-06 + &
533          th2* 1.8832689434804897d-10) + &
534          sqrts*( 5.7463776745432097d-06 + &
535          th2* 1.4716275472242334d-09))
536
537
538
539if((UNPress).and.(p.gt.0.d0)) then
540
541    pth = p*th
542
543    anum = anum +          p*( 1.1798263740430364d-02 + &
544          th2* 9.8920219266399117d-08 + &
545          s* 4.6996642771754730d-06 - &
546          p*( 2.5862187075154352d-08 + &
547          th2* 3.2921414007960662d-12))
548
549    aden = aden +          p*( 6.7103246285651894d-06 - &
550          pth*(th2* 2.4461698007024582d-17 + &
551          p* 9.1534417604289062d-18))
552
553end if
554
555rho_feistel = anum/aden
556
557
558
559! Note: this function should always be run in double precision
560! (since rho is returned rather than sigma = rho-1.0d3)
561
562return
563end function rho_feistel
564!EOC
565
566end module eqstate
567
568!-----
569! Copyright by the GOTM-team under the GNU Public License - www.gnu.org
570!-----

```