Governing equations of rotating stratified flow (version 2)

The Boussinesq approximation (also known as the Boussinesq-Oberbeck approximation) is a special case of the low-Mach number equations for small density and temperature variations with gravity included. Traditionally, the Boussinesq approximation has two main components: (i) density fluctuations resulting from motion result principally from thermal (as opposed to pressure) effects, and (ii) perturbation density is important only in the gravity term. Let

$$\rho_t = \rho_0 + \rho'(\mathbf{x}, t), \tag{1}$$

$$P_t = \bar{P}(y) + P'(\mathbf{x}, t), \tag{2}$$

where ρ_t and P_t are the total density and pressure, and $\mathbf{x} = (x, y, z)$ is the coordinate vector. It has been observed that $\rho' \ll \rho_0$, and $P' \ll \bar{P}$, where symbol "$\ll$" implies ~ 100 times smaller for ocean flows. Hydrostatic balance solution gives

$$\nabla \bar{P} = -\rho_0 g \hat{\mathbf{y}}, \tag{3}$$

where $\hat{\mathbf{y}}$ is the unit vector in the vertical direction.

For low-Mach number (< 0.3), the conservation equation of mass is

$$\nabla \cdot \mathbf{u} = 0, \tag{4}$$

where $\mathbf{u} = (u, v, w)$ is the velocity vector. The momentum conservation equations are

$$\rho_t(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u}) = -\nabla P_t - \rho_t f \hat{\mathbf{y}} \times \mathbf{u} - \rho_t g \hat{\mathbf{y}} + \mu \nabla^2 \mathbf{u}, \tag{5}$$

where $f = 2\Omega$ is twice the frame rotation rate, g is the gravitational acceleration, and μ is the viscosity. The momentum conservation equations can be simplified using the hydrostatic balance solution (3) and the Boussinesq approximation

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p - f \hat{\mathbf{y}} \times \mathbf{u} - g\theta \hat{\mathbf{y}} + \nu \nabla^2 \mathbf{u}, \tag{6}$$

where p is the effective pressure, ν is the kinematic viscosity, and $\theta = \rho'/\rho_0$ is the non-dimensional density.

In order to closure the conservation equations for mass (4) and momentum (6), the transportation equation of density fluctuation (or, internal energy, salinity) is needed

$$\frac{\partial \theta}{\partial t} + \mathbf{u} \cdot \nabla \theta = \kappa \nabla^2 \theta, \tag{7}$$

where κ is the diffusion coefficient of density.
P.S.
The dimensional equations can be referred to Coleman, Ferziger & Spalart [1], Slinn & Riley [3], Smith & Waleffe [4], and Shirgaonkar & Lele [2].

References

[1] G. N. Coleman, J. H. Ferziger, and P. R. Spalart. Direct simulation of the stably stratified turbulent Ekman layer. *J. Fluid Mech.*, 244:677–712, 1992.

[2] A. A. Shirgaonkar and S. K. Lele. On the extension of the boussinesq approximation for inertia dominated flows. *Phys. Fluids.*, 18(066601):1–12, 2006.

[3] D. N. Slinn and J. J. Riley. A model for the simulation of turbulent boundary layers in an incompressible stratified flow. *J. Comput. Phys.*, 144(2):550–602, 1998.

[4] L. M. Smith and F. Waleffe. Generation of slow large scales in forced rotating stratified turbulence. *J. Fluid Mech.*, 451:145–168, 2002.

Description of parameter file for "channelflow" code (including rotation in the y-direction, scalar equation for stratification)

Hao Lu 11/02/2007, revised on 2/26/2008, 3/5/2008

1. Filename is "condition.param"

2. **[Example]**

```
1.  nu = 0.25
2.  dtmax = 0.1
3.  CFLmin = 0.1
4.  CFLmax = 0.9
5.  T0 = 0.0
6.  T1 = 10.0
7.  dT = 1.0
8.  Omega = 0.0
9.  Usurface = 0.0
10. isurfp = 0
11. Up0 = 0.0
12. Up = 0.0
13. Op0 = 0.0
14. Op = 0.0
15. kpx = 0.0
16. kpz = 0.0
17. ihyper = 0
18. nuu = 0.0
19. iscalar = 0
20. g = 0.0
21. kappa = 1.0
22. BC = DbDt
23. bottomvalue = 0.0
24. topvalue = 1.0
25. sbound = 0
26. smax = 1.0
27. smin = 0.0
28. dPdx = 0.0
29. iforce = 0
30. kfmin = 3.5
31. kf = 4.0
32. kfmax = 4.5
33. ef = 0.4
34. forcingtime = 0.0
35. iLES = 0
36. baseflow = PlaneCouette
37. timestepping = CNRK2
38. initstepping = SMRK2
39. nonlinearity = Rotational
40. dealiasing = DealiasXZ
41. taucorrection = true
42. constraint = PressureGradient
43. Ubulk = 0.0
44. Lx = 2.0
45. Ly = 2.0
46. Lz = 2.0
```

3. There are 46 parameters totally; <u>many of them can be not included in a "condition.param" file if using the default values</u>. The order is arbitrary, but the format "name = value" must be followed for each parameter.

4. **[Explanation] (R: double precision real number; I: integer number)**

nu = 0.25: R; kinetic viscosity; the default value is 0.25

dtmax = 0.1: R; maximum time step for time marching; the default value is 0.1

CFLmin = 0.1 : R; minimum CFL number; the default value is 0.1

CFLmax = 0.9 : R; maximum CFL number; the default value is 0.9

T0 = 0.0 : R; initial time; the default value is 0.0
T1 = 10.0 : R; final time; the default value is 10.0
dT = 1.0 : R; time step to output assessments (such as spectrum); the default value is 1.0

Omega = 0.0 : R; rotation rate in the y-direction; the default value is 0.0

Usurface = 0.0 : R; mean velocity on top surface; the default value is 0.0;
isurfp = 0 : I; identification of adding perturbation on top; the default value is 0; isurfp=0: no perturbation;
  isurfp=1: u=Usurface+Up0*sin(Op0*t)+Up*sin(Op*t-kpx*x*2π/Lx-kpz*z*2π/Lz);
  isurfp=2: v=Up0*sin(Op0*t)+Up*sin(Op*t-kpx*x*2π/Lx-kpz*z*2π/Lz);
  isurfp=3: w=Up0*sin(Op0*t)+Up*sin(Op*t-kpx*x*2π/Lx-kpz*z*2π/Lz);
Up0 = 0.0 : R; perturbation velocity (zero mode); the default value is 0.0;
Up = 0.0 : R; perturbation velocity; the default value is 0.0;
Op0 = 0.0 : R; frequency for zero mode; the default value is 0.0;
Op = 0.0 : R; frequency; the default value is 0.0;
kpx = 0.0: R; the default value is 0.0
kpz = 0.0: R; the default value is 0.0

ihyper = 0: I; identification of using hyper-viscosity; if ihyper=0, no hyper-viscosity is included, if ihyper=1, hyper-viscosity will be included in momentum equations; the default value is 0
nuu = 0.0: R; magnitude of hyper-viscosity; the default value is 0.0

iscalar = 0 : I; identification of including scalar equation; the default value is 0; if iscalar=0, then scalar equation is not included; if iscalar=1, then scalar equation is included, but scalar does not effect momentum equation; if iscalar=2, then scalar equation is included, and scalar effect momentum equation (stratification). Note, set iscalar=0 if you don't want to compute a scalar equation, and it will save more than 25% CPU cost.

g = 0.0 : R; value of gravitational acceleration, which describes the density fluctuation effect in momentum equation; the default value is 0.0;

kappa = 1.0 : R; diffusion coefficient for scalar equation; the default value is 1.0

BC = DbDt : ={DbDt, DbNt, NbDt, NbNt}; boundary condition for scalar equation; the default value is DbDt; upper "D", "N" mean Dirichlet and Neumann, lower "b", "t" mean bottom and top

bottomvalue = 0.0: R; value of scalar boundary condition at bottom; the default value is 0.0
topvalue = 1.0: R; value of scalar boundary condition at top; the default value is 1.0

sbound = 0: I; set scalar bound or not; if sbound=1 or 3, s≤smax; if sbound=2 or 3, s≥smin; otherwise, no bound; the default value is 0 (no bound)
smax = 0.0: R; maximum value of scalar; the default value is 1.0
smin = 0.0: R; minimum value of scalar; the default value is 0.0

dPdx = 0.0: R; mean pressure gradient; the default value is 0.0

iforce = 0: I; identification of including Gaussian forcing; the default value is 0; if iforce=0, then Gaussian forcing is off; if iforce=1, then Gaussian forcing is on; Gaussian forcing range is kfmin<k<kfmax, the peak wave-number is kf, the energy input rate is ef, and until forcingtime

- kfmin = 3.5: R; the default value is 3.5
- kf = 4.0: R; the default value is 4.0
- kfmax = 4.5: R; the default value is 4.5
- ef = 0.4: R; the default value is 0.4
- forcingtime = 0.0: R; the default value is 0.0 (won't stop forcing if forcingtime ≤0.0)

iLES = 0: I; identification of LES (SGS) model; the default value is 0; if iLES=0, then no LES modeling; if iLES=others, different SGS models will be included according to this id number.

(We suggest using the default values for the following parameters)

baseflow = PlaneCouette: ={Zero, PlaneCouette, Parabolic}; base flow; the default value is PlaneCouette

timestepping = CNRK2: ={CNFE1, CNAB2, CNRK2, SMRK2, SBDF1, SBDF2, SBDF3, SBDF4}; time stepping scheme; the default value is CNRK2; currently, rotation and stratification is included only using CNRK2

initstepping = SMRK2: ={CNFE1, CNAB2, CNRK2, SMRK2, SBDF1, SBDF2, SBDF3, SBDF4}; initial time stepping scheme; the default value is SMRK2; not very serious, can be arbitrary or just use the default value

nonlinearity = Rotational: ={Rotational, Convection, Divergence, SkewSymmetric, Alternating, Alternating_, LinearAboutField, LinearAboutProfile}; method to include nonlinear term; the default value is Rotational; Rotational form is more accurate than Convection form and Divergence form for many cases, and Rotational form is much faster than SkewSymmetric form

dealiasing = DealiasXZ: ={NoDealiasing, DealiasXZ, DealiasY, DealiasXYZ, DealiasX}; de-aliasing flag; the default value is DealiasXZ; NoDealiasing is not accurate, de-aliasing flag in the y-direction does not work (code dumped), however, de-aliasing in the y-direction is always employed

taucorrection = true: ={true, false}; tau correction or not; the default value is true;

constraint = PressureGradient : = {PressureGradient, BulkVelocity}; flow constraint to be enforced; the default value is PressureGradient

Ubulk = 0.0 : R; bulk velocity value; the default value is 0.0

Lx = 2.0 : R; box dimension in the x-direction; the default value is 2.0
Ly = 2.0 : R; box dimension in the y-direction; the default value is 2.0
Lz = 2.0 : R; box dimension in the z-direction; the default value is 2.0

5. Flags to run this program: there are one optional flag to run the program, and both flags are integer numbers. For example, “run 2”. Let’s use a general form “run i” to explain the details.

I	Plot3D (ParaView)	TecPlot X (X>9)	U^+ in wall coordinates
±1	Yes	No	No
±2	No	Yes	No
±3	Yes	Yes	No
±4	Yes	No	Yes
±5	No	Yes	Yes
±6	Yes	Yes	Yes
±7	No	No	Yes
Otherwise (or not set)	No	No	No

If i<0, then program won’t save velocity/pressure/scalar data. For example, set i=−8 (or no input of i) in order to turn off everything.

3D numerical simulation using channelflow 1.1.2 for rotating turbulence

Input and initialization

Time marching

dns.advance(u, q, n)

Output and end

```
// Define gridsize
  const int Nx=32;
  const int Ny=33;
  const int Nz=32;

  // Define box size
  const Real Lx=1.75*pi;
  const Real a= -1.0;
  const Real b=  1.0;
  const Real Lz=1.2*pi;

  // Define flow parameters
  const Real Reynolds = 400.0;
  const Real nu = 1.0/Reynolds;
  const Real dPdx = 0.0;
  const Real Ubulk = 0.0;

  // Define integration parameters
  const int n = 50;      // take fifty steps between printouts
  const Real dt = 0.03125;  // integration timestep
  const Real T  = 100.0; // integrate from t=0 to t=T

  // Define DNS parameters
  DNSFlags flags;
  flags.baseflow     = PlaneCouette;
  flags.timestepping = SBDF3;
  flags.initstepping = SMRK2;
  flags.nonlinearity = Rotational;
  flags.dealiasing   = DealiasXZ;
  flags.taucorrection = true;
  flags.constraint  = PressureGradient; // enforce constant pressure gradient
  flags.dPdx = dPdx;
  flags.Ubulk = Ubulk;

  // Define size and smoothness of initial disturbance
  Real spectralDecay = 0.5;
  Real magnitude  = 0.3;
  int kxmax = 3;
  int kzmax = 3;

  // Construct data fields: 3d velocity and 1d pressure
  cout << "building velocity and pressure fields..." << flush;
  FlowField u(Nx,Ny,Nz,3,Lx,Lz,a,b);
  FlowField q(Nx,Ny,Nz,1,Lx,Lz,a,b);
  cout << "done" << endl;

  // Perturb velocity field
  u.addPerturbations(kxmax,kzmax,1.0,spectralDecay);
  u *= magnitude/L2Norm(u);

  // Construct Navier-Stoke integrator, set integration method
  cout << "building DNS..." << flush;
  DNS dns(u, nu, dt, flags);
  cout << "done" << endl;
```

Data can be loaded from a binary or asc file.

DNSFlags.baseflow: PlaneCouette - Ubase=y; Parabolic - Ubase=1-y^2
Zero - Ubase=0

DNSFlags.timestepping & initstepping:
CNFE1, CNAB2, CNRK2, SMRK2, SBDF1, SBDF2, SBDF3, SBDF4

DNSFlags.nonlinearity: the way to compute the nonlinear term in the NS equations. Rotational, Convection, Divergence, SkewSymmetric ...

DNSFlags.dealiasing: NoDealiasing, DealiasY, DealiasXZ, DealiasXYZ

DNSFlags.constraint: enforcing bluk velocity or mean pressure constraint

update velocity (u) and pressue (q)

tau solver

calcuate effective pressue; add rotation term

calculate diffusion in physical space:
diff2(uk_, Rxk_, Pyk_);
diff2(vk_, Ryk_, Pyk_);
diff2(wk_, Rzk_, Pyk_)

dealiasing

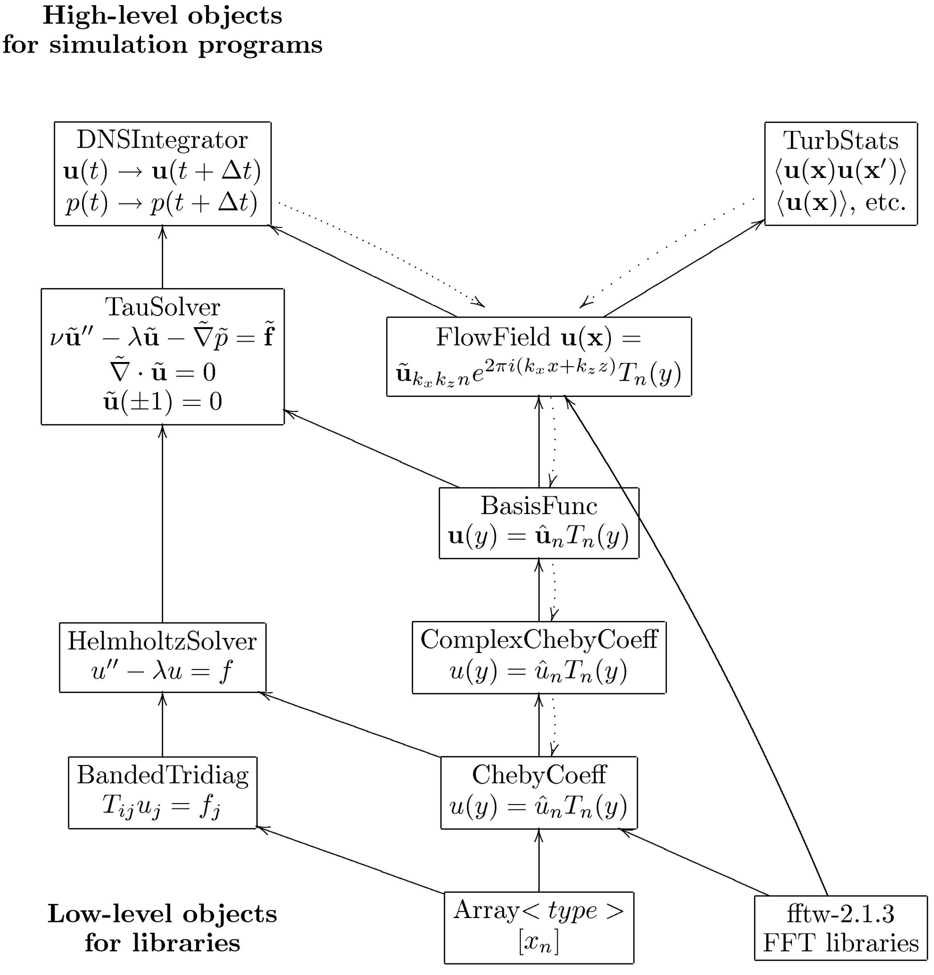
calculate the Nonlinear term according nonlinearity flag:
navierstokesNL(u, ..., flags.nonlinearity)

DNSAlgorithm has been cloned by RungeKuttaDNS, MultistepDNS, and CNABstyleDNS. Thus advance(...) will be directed to one of them according to DNSFlags.timestepping.

```
void DNS::advance(FlowField& u, FlowField& q, int Nsteps) {
  assert(main_algorithm_);

  // Error check
  if (!main_algorithm_->full() && !init_algorithm_) {
    cerr << "DNS::advance(u,q,Nsteps) : the main algorithm is uninitialized,\n"
  << "and the initialization algorithm is not set. This should not be\n"
  << "possible. Please submit a bug report (see documentation)."
   << endl;
    exit(1);
  }
  int n=0;
  while (!main_algorithm_->full() && n<Nsteps) {
    init_algorithm_->advance(u,q,1);
    main_algorithm_->push(u);
    if (main_algorithm_->full()) {
      delete init_algorithm_;
      init_algorithm_ = 0;
    }
    ++n;
  }

  main_algorithm_->advance(u,q,Nsteps-n);
}
```


High-level objects
for simulation programs
DNSIntegrator
$\mathbf{u}(t) \rightarrow \mathbf{u}(t+\Delta t)$
$p(t) \rightarrow p(t+\Delta t)$
TurbStats
$\langle \mathbf{u}(\mathbf{x})\mathbf{u}(\mathbf{x}') \rangle$
$\langle \mathbf{u}(\mathbf{x}) \rangle$, etc.
TauSolver
$\nu\tilde{\mathbf{u}}'' - \lambda\tilde{\mathbf{u}} - \tilde{\nabla}\tilde{p} = \tilde{\mathbf{f}}$
$\tilde{\nabla} \cdot \tilde{\mathbf{u}} = 0$
$\tilde{\mathbf{u}}(\pm 1) = 0$
FlowField $\mathbf{u}(\mathbf{x}) =$
$\tilde{\mathbf{u}}_{k_x k_z n} e^{2\pi i(k_x x + k_z z)} T_n(y)$
BasisFunc
$\mathbf{u}(y) = \hat{\mathbf{u}}_n T_n(y)$
HelmholtzSolver
$u'' - \lambda u = f$
ComplexChebyCoeff
$u(y) = \hat{u}_n T_n(y)$
BandedTridiag
$T_{ij} u_j = f_j$
ChebyCoeff
$u(y) = \hat{u}_n T_n(y)$
Low-level objects
for libraries
Array< *type* >
$[x_n]$
fftw-2.1.3
FFT libraries