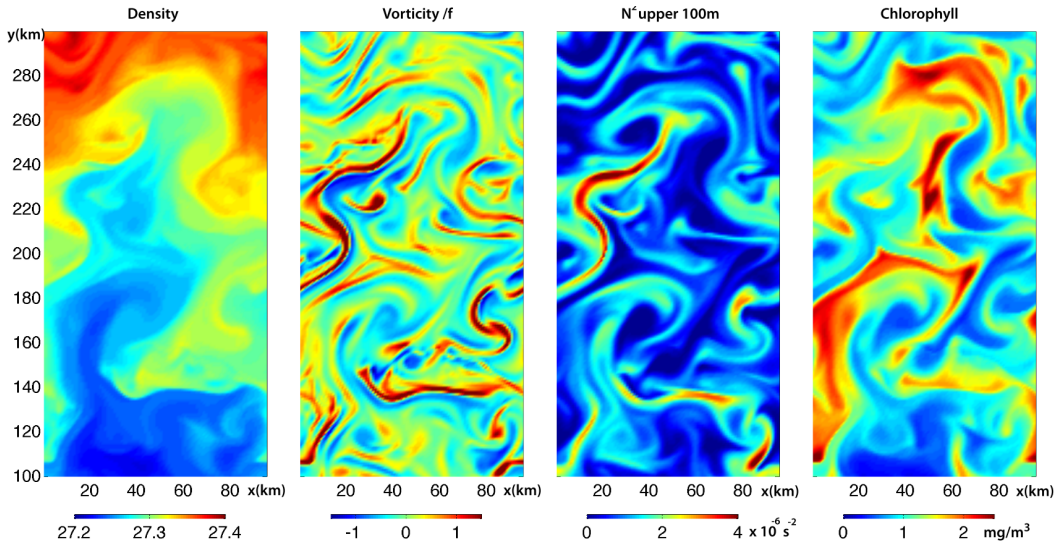
User Manual for PSOM



Ocean and Environmental Processes

Dr. Amala Mahadevan's Lab

Compiled by M. Dever

September 27, 2019

Contents

1 Introduction **2**

2 Downloading PSOM **2**

3 Setting-up PSOM **2**
3.1 Fortran compiler and bash_profile 2
3.2 `optfile` and `configure.sh` 3

4 Running PSOM **3**

5 Test Cases **4**
5.1 Wiggles 4
5.2 NA 4
5.3 Shelfbreak 4

6 Setting up your PSOM simulation **5**
6.1 Create your experiment directory 5
6.2 Defining your model grid 5
6.3 cppdefs.h 7
6.4 namelist 8
6.5 Defining the initial conditions 8
6.6 Wind stress 9
6.7 Surface Heat Fluxes 10

7 Particle tracking in PSOM **10**
7.1 Non-sinking particles 10
7.2 Sinking particles 11
7.3 Reading unformatted binary output files 11
7.4 Customizing Particle-tracking in PSOM 12
7.4.1 Parameters in `namelist` 12
7.4.2 `particles.f90` 12

A Appendix A **14**

B Appendix B - particle_open_bin.m **18**

C Appendix C - particle_bin2csv **20**

1 Introduction

PSOM, pronounced "soam" (the nectar derived from the churning of the oceans in Indian mythology), stands for Process Study Ocean Model. It is a versatile, three-dimensional, non-hydrostatic, computational fluid dynamical model for oceanographic (as well as other) applications (Mahadevan et al., 1996a,b). The model uses the finite volume method on a structured grid with boundary fitted coordinates (topography conforming sigma grid in the vertical, and boundary conforming in the horizontal). The model has a free-surface. It can be used for large- and small-scale phenomena and can be run in hydrostatic or non-hydrostatic mode (Mahadevan, 2006). It uses a highly efficient solution procedure that exploits the skewness arising from the small geometrical aspect (depth to length scale) ratio of the ocean to speed up the solution of the non-hydrostatic pressure, which is solved by the multigrid method.

The model has been used for a number of process studies, including investigation of the vertical transport of nutrients for phytoplankton production (Mahadevan and Archer, 2000) and the dynamics of submesoscale processes (Mahadevan and Tandon, 2006; Mahadevan, Tandon and Ferrari, 2010). Since the non-hydrostatic model is well-posed with open boundaries, it can be used as a nested high-resolution model with time-varying boundary conditions applied from a coarser resolution general circulation model (Mahadevan and Archer, 1998). The model is thus ideally suited for high-resolution, limited-region modeling studies.

2 Downloading PSOM

1. Download the latest version of PSOM from GitHub. Users are strongly encouraged to track the modifications and improvements made to PSOM, and to submit them for review through GitHub.

2. Unzip the file into a directory (e.g., `V1.0-master`)

3. Go to the "`V1.0-master/code`" directory and follow the instructions detailed below.

NOTE : Hereafter, filepaths are always specified with respect to the directory "`V1.0-master/code`".

3 Setting-up PSOM

3.1 Fortran compiler and bash_profile

Before setting-up PSOM, it is important to make sure that the compiler that will be used to compile PSOM is properly installed. Once the compiler is properly installed, the file `.bash_profile` should be edited to include the proper paths. In the

example below, the `.bash_profile` includes 3 different paths where the compiler could be located (*/opt/intel/composer_xe_2015/bin*, */usr/local*, and */opt/intel/bin*) .

```
#bash_profile

# PSOM environment variables
export DYLD_LIBRARY_PATH=/opt/intel/lib
export PATH=$PATH\:/opt/intel/composer_xe_2015/bin:/usr/local:/opt/intel/
    bin
```

3.2 `optfile` and `configure.sh`

Before being able to compile PSOM, a few necessary steps must be taken:

Step 1: Update the `optfile`
Edit the line "*fcomp*=..." to specify the proper compiler (e.g., ifort, pgf95, etc.). If compiler is modified, the user is encouraged to scan through the ./optfile to make other necessary modifications (e.g., activation of gotm library, define_parallel flag, etc.).

Step 2: Run `configure.sh`

```
 sh tools/configure.sh
```

Running `configure.sh` will search for makedepend (which is necessary to create the makefile). The script will stop if makedepend is not installed. The makefile file will only be created if makedepend can be called. Therefore, the user should download makedepend, which is free and widely accessible. For instance, on mac, you can simply type (if you have macport installed):

```
sudo port install makedepend
```

If you chose to use netcdf, an attempt to use nc-config will be tried. If nc-config is present, it will be used in order to set the links to netcdf libraries appropriately; If not, you can complete this step yourself by linking the executable to the netcdf libraries. To do so, edit ./optfile before going to the next step.

To summarize, This step has 2 effects: (1) modify `optfile`, and (2) customize `namelist`. Failures in running `configure.sh` can likely be avoided by correctly installing makedepend and/or netcdf libraries.

4 Running PSOM

After properly setting-up PSOM, you can simply follow the "clean, compile, run" sequence:

clean: This step is only necessary if a new experiment is compiled. If re-compiling the same experiment, there is no need to run `clean.sh`. `clean.sh` is ran using:

```
sh tools/clean.sh
```

compile: Running `compile.sh` will (1) create the makefile in the folder mkfile, and (2) compile the fortran code. A failure during step 4 most likely indicates that there is an error in the fortran code. `compile.sh` is ran using:

```
# To compile PSOM using default setup (cf. superceding trick)
sh tools/compile.sh

# To compile a specific experiment, such as the wiggle test case
sh tools/compile.sh wiggle
```

run: Runs the executable generated by the compiling step. It uses the `namelist` file to define the experiment's parameters (e.g., time step, output path, etc.; see Section 6.4).

```
# To run the wiggle experiment
/exe/nh_wiggle < wiggle/namelist_wiggle
```

5 Test Cases

5.1 Wiggles

```
- wiggle       : This is a testcase with a wiggling front over a flat
  topography.
```

5.2 NA

```
- NA           : This is a much more complex simulation of three fronts
  that go unstable. This casse has particles and tracers for biology.
```

5.3 Shelfbreak

```
- shelfbreak   : Simulation of the Middle Atlantic Bight shelfbreak with a
   shelf front and a shelfbreak font. The topography includes a sharp
  slope at the break. It shows how to use the "user" namelist.
```

6 Setting up your PSOM simulation

6.1 Create your experiment directory

For every experiment you want to conduct, create a directory that will contain the source files specific to this experiment. As an example, let's say you want to create an experiment named "my_experiment". First, create a directory `my_experiment/`, in which you will create two subdirectories `my_experiment/inc` and `my_experiment/src`. You can either create these directories manually, or you can run:

```
# Copies the template directory
cp -r expe_template my_experiment
```

Whether you are a user or a developer, **you are strongly invited to leave untouched the files contained in `model/`**. This directory is designed to contain the latest version of the model, which is common to every user at a given time. For every routine that will be specific to `my_experiment`, a new subroutine should be created in `my_experiment/src/`. This can be achieved using the following command:

```
# Copies the initial conditions subroutine
cp model/src/ini_st.f90 my_experiment/src
```

The compiling step (see Section 4) includes a superseding procedure that will take into account the new version of `ini_st.f90` (in `my_experiment/src/`) and disregard the standard version found in `model/src/`. More precisely, it will create the makefile based on this new state of the model (in `./mkfile`), compile and create the executable `exe/nh_my_experiment`. More details on `compile.sh` may be found by running:

```
# Provides more information on compile.sh
sh tools/compile.sh -help
```

Superseding trick
When compiling, priority is given to the routines present in the experiment folder, over the source code present in `model/src/`. For example, if the wiggle experiment is compiled, any routines located in `wiggle/src/` will be used over the matching routines located in `model/src/`.

6.2 Defining your model grid

The grid size is defined in `size.h`. If you wish to modify the grid size, you must first copy `size.h` into your experiment's directory:

```
# Copies the grid file
cp model/inc/size.h my_experiment/inc
```

Grids used in previous experiments are listed in this file and commented out. If your grid appears in a commented line, Comment the uncommented line and uncomment the one you want. Be aware that if your grid set requires more than 2Go, you might experience compilation issues. If so, you may fix the issue by editing `tools/genmakefilel` to replace the default compiling options by:

```
fflags_o="-fpp -real-size 64 -mcmodel medium -shared-intel -stand 03 -u"
fflags_e="-fpp -real-size 64 -mcmodel medium -shared-intel -stand 03 -u"
```

If your grid set does not appear in `my_experiment/inc/size.h`, you can create the required line. Defining the model grid is not straight-forward, because of the multi-grid solver mgrid <**FIX ME: (See Section)**>. The multi-grid solver is used to allow the reuse of array space in `mgrid.f90` <**FIX ME: insert link to function?**>. Although this issue could now be circumvented by making use of f90's dynamic allocation of memory, the code was originally in fortran77, explaining the need for space re-allocation. A step-by-step approach to defining your own grid is provided below:

1. Choose grid dimensions NI, NJ, and NK (i.e., the number of grid cells in x, y, and z directions) such that the grid can be subdivided a maximum number of times by a factor of 2 to form "$ngrid$" levels of grid. For example, choosing $NI = 48$, $NJ = 24$, and $NK = 32$ constrains the grid levels to 4 (i.e., $ngrid = 4$), because:

$$\begin{aligned} NI &: 48; 24; 12; 6; 3 && \text{(5 grid levels)} \\ NJ &: 24; 12; 6; 3 && \text{(4 grid levels)} \\ NK &: 32; 16; 8; 4; 2 && \text{(5 grid levels)} \end{aligned}$$

   The number of grid points possible for a specific $ngrid$ can be computed by multiplying prime numbers (2, 3, 5, 7, etc.) by $2^{ngrid-1}$. Table 1 lists some of the most commonly used number of grid points, depending on the number of grid levels $ngrid$.

2. Compile `tools/preproc.f90`:

```
# Compiles preproc.f90 (e.g., using ifort)
ifort preproc.f90 -o preproc
```

3. Runs `preproc.f90` and fill the values that are asked:

```
# Runs preproc
./preproc
```

4. Copy/Paste the last line the program provides in `my_experiment/inc/size.h`. Below is an example for $NI = 96$, $NJ = 160$, and $NK = 32$ (hence $ngrid = 5$, see Table 1):

```
./preproc
 number of grid levels in mgrid, ngrid =
```

Table 1: Number of grid points associated with a specific number of grid levels $ngrid$. These numbers can be computed by multiplying prime numbers (2, 3, 5, 7, etc.) by $2^{ngrid-1}$. Each experiment's number of grid levels is set by the minimum $ngrid$ associated with NI, NJ, and NK.

$ngrid$	Number of grid points (NI, NJ, or NK)							
4	16	24	40	56	88	104	136	152
5	32	48	80	112	176	208	272	304
6	64	96	160	224	352	416	544	608
7	128	192	320	448	704	832	1088	1216

```
 5
  input the grid info
  NI =
 96
  NJ =
 160
  NK =
 32
  Number of grid points on fine grid: nx,ny,nz   96   160   32
 m, ntint, ntout, nbc(m)  1       491520       539784       47104
 m, ntint, ntout, nbc(m)  2        61440        73800       11776
 m, ntint, ntout, nbc(m)  3         7680        10920        2944
 m, ntint, ntout, nbc(m)  4          960         1848         736
 m, ntint, ntout, nbc(m)  5          120          384         184
 Copy the following line to size.h
 INTEGER, PARAMETER :: NI=96, NJ=160, NK = 32, ngrid=5, maxout
     =626736, maxint=561720, int1=491520
```

6.3 cppdefs.h

This file defines the different options to be used in the experiment. Again, it is recommended to copy this file into the experiment folder (e.g., `my_experiment/inc/` before making any modifications. To include (exclude) an option, use #define (#undef) $option\_name$. The file includes 13 options:

$runtracmass$: placeholder

$periodic\_ew$: placeholder

$periodic\_ns$: placeholder

$allow\_particle$: If defined, allows the seeding of particles in the experiment. Please refer to section <**FIX ME: ref to particle section**>for a detailed explanation on particle seeding.

$rhoonly$: If defined, only the density field rho is used. The density field is stored in the salinity array (s; see `evalrho_rho.f90`). If not defined, rho is computed from the salinity (s) and temperature (T) fields (see `evalrho_sT.f90`).

$relaxation$: placeholder

$bottom\_thickness$: placeholder

$file\_output$: placeholder

$file\_output\_cdf$: placeholder

$file\_output\_bin$: placeholder

$gotm\_call$: placeholder

$implicit$: placeholder

$parallel$: placeholder

6.4 namelist

This file defines key parameters relating to the experiment (e.g., grid resolution, time step, diffusion, output, ...). Again, it is recommended to copy this file into the experiment folder (e.g., `my_experiment/` before making any modifications. Each parameter in the file is either self-explanatory or include a short description as a comment.

¡¡¡¡¡¡¡ Updated upstream

6.5 Defining the initial conditions

Initial conditions can be specified either in the corresponding subroutines, or from an input file. The former approach is used in the Shelfbreak test-case (see Section 5.3), where the temperature and salinity distributions are determined from analytical expressions in DO-loops, and only requires a limited knowledge of the model grid. The latter approach can sometimes be more practical, especially when using available data products to initialize the model. However, this approach requires mapping the data used to initialize the experiment to the pre-defined model grid.

The horizontal grid is relatively straightforward to determine, given the grid size specified in `my_experiment/inc/size.h` (i.e., NI and NJ), and the grid resolution specified in `my_experiment/namelist` (i.e., dx and dy). The location of each grid point can be computed using the following equations:

$$x(i) = -dx/2 + idx; \qquad i = (0, 1, 2 \ldots NI, NI + 1) \tag{1}$$

$$y(j) = -dy/2 + jdy; \qquad j = (0, 1, 2 \ldots NJ, NJ + 1) \tag{2}$$

The initial conditions in the temperature and salinity: Talk abour the rhoonly flag, the way to set initial conditions, etc...

¡¡¡¡¡¡¡ Updated upstream

6.6 Wind stress

To specify a customized wind forcing, the code in `wind_stress.f90` can be modified. The wind stress at the surface is prescribed to the model through the variables $stress\_top\_x$, $stress\_top\_y$, and $stress\_top$. The dimensions of these three variables are (NI,NJ) (see `header.f90`). The surface wind stress can be read from a file:

```
! Import the wind stress time series for model forcing
if (step.eq.1) then
  open(unit=17, file='youfilefullpath.in')
  do i=1,nsteps
    read(17,fmt="(F5.10,F5.10)") stressxTS(i),stressyTS(i)
  end do
  close (17)
  PRINT*,"Read wind stress"
end if
stress_top_x = stressxTS(step)
stress_top_y = stressyTS(step)
```

or specified as a constant:

```
stress_top_x = 0.05d0
stress_top_y = 0.01d0
PRINT*,"Read Wind Stress"
```

If your domain includes solid boundary (i.e., no periodicity), it is recommended to damp the surface wind stress close to the boundaries, to avoid upwelling/downwelling. Below is an example of wind stress damping at the north/south boundaries using a tanh profile. A similar approach can be used in the east/west direction.

```
! Apply a tanh profile in the meridional direction
ycenter = 0.5*(yc(NJ)+yc(1)) ! Find the middle of the domain
ywindmin = 10.0          ! Starts damping 10 km from southern boundary
ywindmax = yc(NJ)-10.d0        ! Starts damping 10 km from northern
    boundary
edge = 0.06             ! tightnness of the padding in the wind stress

do j=1,NJ
  if (yc(j).lt.yc(NJ/2)) then
    stressprofile(j) = 0.5*(tanh(edge*(yc(j)-ywindmin)*PI)+1.d0)
  else
    stressprofile(j) = -0.5*(tanh(edge*(yc(j)-ywindmax)*PI)-1.d0)
  end if
end do

do j=1,NJ
  do i=1,NI
```

```
    stress_top_x(i,j) = stressxTS(step)*stressprofile(j)
    stress_top_y(i,j) = stressyTS(step)*stressprofile(j)
  end do
end do
```

6.7 Surface Heat Fluxes

7 Particle tracking in PSOM

PSOM offers the option to release and track particles ”online” (i.e., as part of the numerical simulation). To activate this option, the allow_particle options must be defined in `inc/cppdefs.h` by including (See Section 6.3):

```
#define allow_particle
```

The particle tracking code has been written for a rectangular grid only, and cannot be used “as is” for a non-rectangular model grid.

7.1 Non-sinking particles

While key particle parameters are set in `namelist` (e.g., particle number, frequency of outputs, etc.; see Section 7.4), the seeding and advection of particles is controlled by the code included in `particles.f90`. The file includes the following subroutines:

open_parti_files: This subroutine creates the output files where the particle characteristics will be saved. The number of output files is specified in `namelist` and must be a factor of the total number of particles (NPR). Increasing the number of output files reduced the size of the individual files, which proves to be useful when dealing with a very large number of particles. The output files are unformatted binary files.

save_parti: This subroutine loops through all the particles and writes the specified variables. The number of variables saved is important, as it must be known to read the unformatted binary output files.

ini_particles: This subroutine is called when the model timestep matches the particle initialization timestep specified in `namelist`. This is where the seeding of particle is defined. By default, all particles are released below the surface layer in the middle of the model domain. To personalize the release of particles, see Section 7.4.2.

get_parti_vel: This subroutine interpolates the physical model’s velocity field onto the particles’ positions (using *interp_trilinear*; see below). The *get_parti_vel* subroutine also interpolates variables of interest onto the particles position (e.g., salinity, temperature, density, vorticity, etc.).

parti_forward: This subroutine extrapolates the position of a particle at the next timestep t+1 using a 2^{nd} order Adams-Bashforth scheme. As an example, the position of the particle in the zonal direction is computed using:

$$(i,j,k)_{t+1} = (i,j,k)_t + dtf \times \frac{1}{2}[3(u,v,w)_{t+1} - (u,v,w)_t]$$

Where (i,j,k) is the position of the particle in the model space, dtf is the non-dimensional model time step, (u,v,w) is the non-dimensional velocity field at the particle's location, and the subscripts represent the timestep. The corresponding code appears in `particles.f90` as (e.g., for the particle position in the zonal direction):

```
1 ! Assign i-position to particle.
2 parti(i)%i = parti(i)%i + 0.5d0 * dtf * (3d0 * parti(i)%u - parti(
    i)%u0)
3
```

At t=0, the velocities are assumed to be zero (set in *ini_particles*, and the 2^{nd} order Adams-Bashforth scheme simplifies to a one-step Euler scheme.

interp_trilinear This subroutine is used to interpolate 3D model variables onto a particle's position using a trilinear interpolation technique (e.g., velocities, density, etc.).

interp_bilinear This subroutine is used to interpolate 2D model variables onto a particle's position using a bilinear interpolation technique (e.g., depth of water column).

7.2 Sinking particles

The subroutine *get_parti_vel* includes the possibility to prescribe a vertical sinking velocity to the particles (set to 0 m/s by default). The prescribed velocity must be scaled appropriately to match the scaling used by PSOM. This requires information about the function used to compute the thickness of the model cells in the vertical, defined in `findzall.f90`. If this function is modified, the code in `particles.f90` **must be changed accordingly** (See Appendix A for important information). Although not implemented in the code, a horizontal velocity (e.g., to simulate swimming behavior) could also be easily prescribed to the particles following a similar method.

7.3 Reading unformatted binary output files

Particle-tracking output files are written as unformatted binary files. Information on how the output file is built is therefore required to be able to access the data. Two MATLAB routines to import or convert the particle-tracking output are provided with the model code:

- `particle_open_bin.m`: Imports the particle-tracking data into MATLAB as a 3D matrix (Nbr of particles, model time, # of recorded variables; see Appendix B). WARNING: This routine SHOULD NOT BE USED FOR LARGE FILES otherwise the 3D matrix will become too large and will crash MATLAB.

- `particle_bin2csv.m`: Converts the particle-tracking data into a CSV-file (see Appendix C). This can be helpful when trying to import the particle-tracking data into another software (i.e., into an SQL database).

7.4 Customizing Particle-tracking in PSOM

7.4.1 Parameters in `namelist`

To set up a particle-tracking experiment in PSOM, the allow_particle option must be defined in `inc/cppdefs.h` (See Section 6.3):

```
#define allow_particle
```

Four key variables related to particle-tracking are set in the `namelist` file:

1. The total number of Particles (NPR). NPR must be a multiple of the number of output files (see below).

2. The time step at which the particles are released ($ini\_particle\_time$). $ini\_particle\_time$ **must be** greater than 0, or than $pickup\_step$ if pickup files are used to initialize the experiment (see Section 6.4 <**FIX ME: Refer to section about namelist and pick up files**>). If $ini\_particle\_time$ = $pickup\_step$, no particle output file will be written.

3. The number of output files to generate ($parti\_file\_num$). Increasing the number of files logically decreases the file size. This is especially useful when dealing with a very large number of particles, or when writing particles' position at high frequency. The number of file must be a factor of NPR (see above).

4. The frequency of particle outputs ($parti\_outfreq$) in number of time steps.

7.4.2 `particles.f90`

a. Particle seeding

The seeding and tracking of the particles in PSOM are controlled by subroutines located in `particles.f90`. To personalize the seeding of particles in the model, the code in the subroutine $ini\_particles$ should be altered. By default, the particles are released below the surface layer, in the middle of the model domain:

```
! User-defined particle positioning.
 DO ip=1, NPR
   parti(ip)%i=REAL(NI)/2d0 ! mid-domain in x
   parti(ip)%j=REAL(NJ)/2d0 ! mid-domain in y
   parti(ip)%k=REAL(NK)-5.  ! sub-surface cell
=======
=======
\subsection{Initial conditions (\texttt{ini\_$*$.f90})}
>>>>>>> Stashed changes
>>>>>>> Stashed changes

   ! Converts model grid to distances (i,j,k) ==> (x,y,z)
   parti(ip)%x = parti(ip)%i * dx
   parti(ip)%y = parti(ip)%j * dy
   ! Assign z-position to particle based on sigma level
   ! Calculate the scaled z-depth
  CALL sigma2z(parti(ip)%i,parti(ip)%j,parti(ip)%k,swap1)
   parti(ip)%z = swap1 * DL
 ENDDO
```

b. Sinking velocity

A sinking velocity can be prescribed to the particles (default = 0 m/s) by modifying the following line of code in *get_parti_vel*:

```
! Then, specify the sinking velocity (in m/s), including the scaling
    factors
 parti(ip)%wsink= -0d0/86400d0/WL*parti(ip)%wzf*EPS   ! 0 m/day
```

A Appendix A

Prescribing Sinking Velocity to Particles

M.Dever

July 18, 2017

How is sinking prescribed

The sinking velocity in the model (w_{sink}) is prescribed in the subroutine `get_parti_vel` in `particles.f90`. To match the physical model's velocity, w_{sink} must be scaled appropriately using:

```
parti(ip)%wsink= -1d0/86400d0/WL*parti(ip)%wzf*EPS   ! lm/day
```

where WL non-dimensionalizes the vertical velocity, EPS is the Rossby number, and wzf is the coefficient scaling the vertical velocity with the size of the the k-cell in the k-space.

More on wzf
wzf can be though of as $1/\Delta z$. It is the inverse of the (normalized) grid spacing in the vertical **at the cell faces**. The equivalent metric at the cell centers is wz.

Error on the vertical displacement

Motivation

The coefficient wzf needs to be determined for each particle position. The original way of determining wzf relied on (1) the linear interpolation of the cell-centered vertical grid spacing wz, and (2) the trilinear interpolation used in the particle code (see `particles.f90`).

```
! Compute wz at face grids using linear interpolation
wzf = 0.5d0*(wz(:,:,0:NK) + wz(:,:,1:NK+1))

! Compute wzf at the paricle's location using trilinear interpolation
CALL interp_trilinear(dic,djc,dkc,wzf(ic:ic+1,jc:jc+1,kfc:kfc+1),parti(ip)%wzf)
```

Issues

The approach to determine the wzf coefficient onto the particle position in the vertical introduced some error in the vertical position of the particle. This error was identified by comparing the depth of a particle sinking at a constant rate as computed by the `particles.f90` routine, with the theoretical depth based on the sinking rate, the time elapsed, and the release depth:

$$model = w_{sink} \times (t - t_0) + z_0$$

Figure 1 shows the error on the particle's vertical positioning with respect to the particle's depth.The error increases between the cell centers and faces (where the velocity is underestimated; see Figure 1), and decreases between cell faces and centers (where the velocity is overestimated). The error therefore oscillates and grows as k-cells become thicker. Future model variables (i.e., current velocities) will be interpolated onto the erroneous particle position, therefore introducing some cumulative effect in the errors associated with this method. Such errors are hard to quantify.

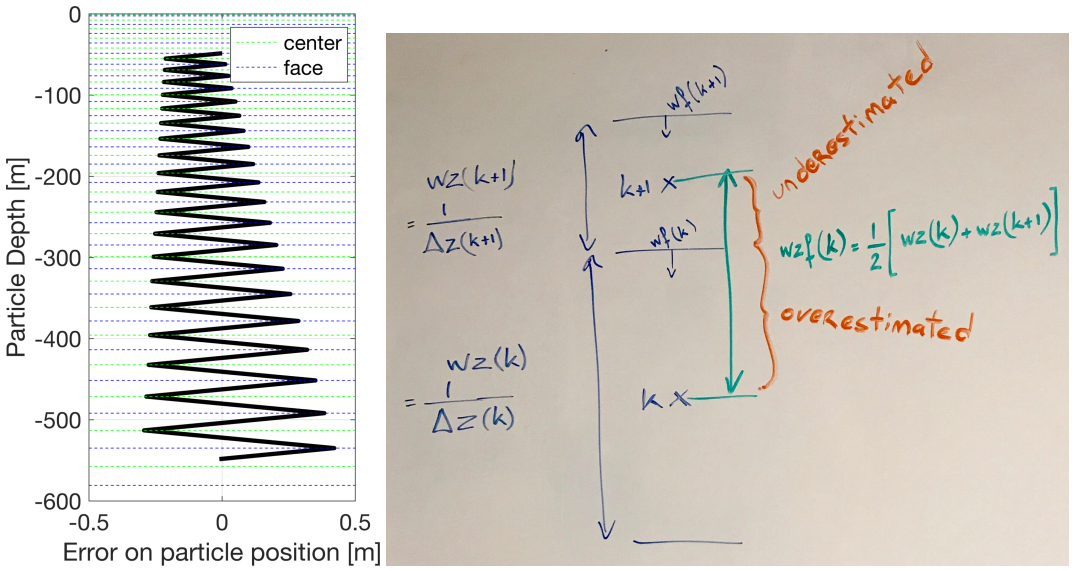


Figure 1: Error on particle vertical position due to the interpolation of wzf onto the particle position. Error is maximum at model grid cell centers and at grid cell faces. Error grows with depth, as grid cells become thicker, and is independent of the sinking velocity. Right panel shows the important variables and highlights the limitations of the original method.

What about the horizontal?
The error in the vertical described here arises from the fact that cell dimensions change with depth. A similar issue will therefore be present in the horizontal when using a non-rectangular grid. For a non-rectangular grid, uxf and vyf should be computed using a similar approach than the method outlined below. For a rectangular grid, the linear interpolation of ux and vy onto the faces is adequate.

New approach

Instead of relying on an interpolation of the discrete values of wzf, the continuous function that determines the depth of the k-levels (i.e., the vertical grid spacing), is used to exactly determine the value of wzf at the particle's location. The function is defined in the routine `findz_topmoves.f90`.

```
zc(i,j,k)= (exp(pfac*xfac)-1.d0)*epm1inv*(D(i,j)+dztop) -dztop
```

which can be re-written in terms of set parameters as:

$$zc(i,j,k) = \left(\frac{D(i,j) + dztop}{e^{pfac} - 1}\right) e^{pfac} e^{\frac{-pfac(k-0.5)}{NK-1}} + C \qquad (1)$$

where zc is the depth of the cell-center, $D(i,j)$ is the dimensionless depth of the 0-th face z (at cell centers in x and y), $dztop$ is the dimensionless thickness of the uppermost cell, $pfac$ is the vertical stretching parameter used to define the sigma levels, NK is the number of vertical levels, and C is a constant.

What is the constant C?
The value of C is irrelevant in this context, as the difference between two z-levels is the quantity we are ultimately interested in.

The equation for the difference between two z-levels at k and k+1 (i.e., Δz) can thus be derived (Figure 2):

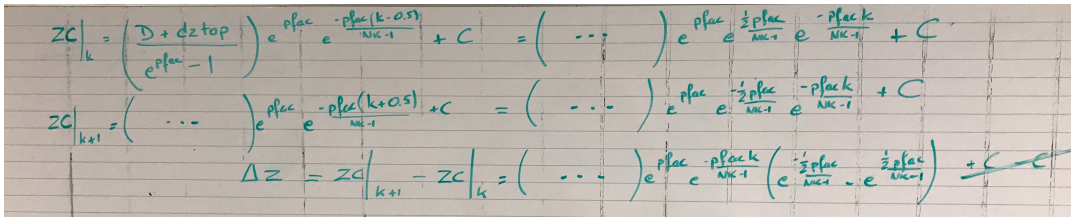


Figure 2: Equations used to compute Δz at k-faces in `particles.f90`

The variable wzf can now be computed exactly at the particle's position by taking the inverse of Δz. Figure 3 shows the error (in meters, as well as in percentage of the particle's depth) on the particle vertical position after modifying the approach to compute wzf. The error grows linearly with depth, with a slope of 1.735×10^{-4} m per meter (i.e., 17.35 cm at 1000 m deep).

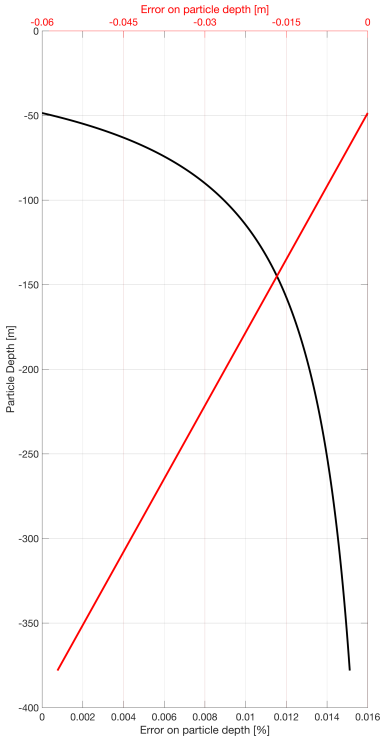


Figure 3: Error on particle vertical position after the approach used to compute wzf was modified.

What about the horizontal?
A slightly different equation than the one depicted in Figure 2 must be used if computing wz. While wzf must be computed using the depth of the z-cell centers zc, wz must be computed using the depths of the z-cell faces zf. this is achieved by using k instead of $(k - 0.5)$ in Equation 1.

B Appendix B - particle_open_bin.m

```
clear

% ** This code imports the particle-tracking data directly from bin output
% files.
% ** The file path has to be specified.
% ** output is a 3D matrix where the 1st dimension is the particle number,
% the 2nd dimension is the model time, and the 3rd diemnsion is the
% recorded variables.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% WARNING
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This routine SHOULD NOT BE USED FOR LARGE FILES otherwise MATLAB matrices
% will become too large and will crash MATLAB.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PARAMETERS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% number of files to import into matlab
number_of_files = 1;
% filepath
path = 'specify the path for the output files here';
% Number of variables recorded (see particles.f90)
number_of_variables = 20;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CORE CODE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Loops through the nummber of files to open
for filenum = 1:number_of_files

% Create fullpath
filename = ['op.parti-',num2str(filenum,'%03.f'),'.bin'];
fullpath = [path,'/',filename];

% Display the file being extracted
disp(filename);

```

```
% Open the file
fileID = fopen(fullpath);

% Extract the data (refer to particles.f90 to confirm that number)
A = fread(fileID,[number_of_variables Inf],'double');
A = A';

% Finds the number of particle per file using the ID numbers
if filenum == 1
partnum = max(A(:,1));
end

% write a matrix of dimensions:
% # of particles x timestep x recorded variables
for Np = (filenum-1)*partnum+1:(filenum-1)*partnum+partnum
ind = find(A(:,1) == Np);
data(Np,:,:) = A(ind,:);
end; clear Np ind

fclose(fileID);
clear A fileID filename path fullpath ans

end; clear filenum partnum
```

C Appendix C - particle_bin2csv

```
clear

% This code converts the binary file ouput from the model into CSV files.
% This can be useful to open in other programs (e.g., import in a SQL
% database)

%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PARAMETERS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% number of files to import into matlab
number_of_files = 1;
% filepath of files to import
pathin = 'specify the path of the files to import here';
% Number of variables recorded (see particles.f90)
number_of_variables = 20;
% filepath of the csv-file to be written
pathout = 'specify the path where you would like to write the csv-files';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CORE CODE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Loops through the number of files to open
for filenum = 1:number_of_files

    % Create fullpath
    filename = ['op.parti-',num2str(filenum,'%03.f'),'.bin'];
    fullpath = [pathin,'/',filename];

    % Display the file being extracted
    disp(['Converting ',filename,' into a CSV file ...']);

    % Open the file
    fileID = fopen(fullpath);

    % Extract the data (refer to particles.f90 to confirm that number)
    A = fread(fileID,[number_of_variables Inf],'double');
    A = A';

    % Remove all the zeros recorded
    ind = find(A(:,1)==0);
    if isempty(ind)~=1
        warning(['Missing ',num2str(length(ind)),' records...!'])
```

```
        A(ind,:) = [];
    end; clear ind


    % Re-write the file as CSV.
    csvwrite([pathout,'/op.parti-',num2str(filenum,'%03.f'),'.csv'],A)

    fclose(fileID);
    clear A partnum fileID filename path fullpath

end; clear filenum
```