

OPPLA

Markus Pahlow

July 6, 2023

Please note: This manual uses angle brackets <> to denote placeholders, e.g., `oppla_<DATE>.tar.bz2` could be expanded to `oppla_2020-04-14.tar.bz2`, and square brackets [] denote optional input.

Contents

1	Obtaining and Installing OPPLA	1
2	Running OPPLA	2
2.1	Prerequisites	2
2.1.1	The initial-conditions file	2
2.1.2	The forcing file	2
2.2	Command-line options	3
2.3	Namelist	3
3	Adding a Plankton Group	12
3.1	Add a new namelist to <code>pfn</code>	12
3.2	Add a profile to the initial-conditions file <code>ifn</code>	12
3.3	Add the new group (species) to the plankton community	12
4	The R <code>oppla</code> package	12
4.1	Creating and modifying forcing files	12
4.2	Reading and processing model output in R	13
4.2.1	Reading model output	13
4.2.2	Time slices and initial-condition files	13
4.2.3	Time averages and climatologies	13
4.2.4	Vertical integrals	13
4.2.5	Level and tile plots	14
4.2.6	Namelist files	14
5	The Matlab <code>oppla</code> packages	14
5.1	Reading model output	14
5.2	Climatologies	14
5.3	Oppla database (odb) functions	15
6	Parameter estimation with a genetic algorithm	15
6.1	Suggested procedure	16

1 Obtaining and Installing OPPLA

You can get `oppla` either as a tar file or via git. If you have a tar file, execute

```
tar xf oppla_<DATE>.tar.bz2
```

which unpacks all files into a folder named `oppla`. You can also obtain `oppla` from our gitlab repository with

```
git clone https://git.geomar.de/markus-pahlow/oppla.git [<OPPLA>]
```

which creates the folder `oppla` (or `<OPPLA>` if specified) and installs all files there. Make sure that `oppla` (or `<OPPLA>`) does not exist, as this will fail if `oppla` (or `<OPPLA>`) exists and is not an empty folder.

Oppla depends on the NetCDF and UDUNITS2 libraries, which must be installed prior to compiling oppla. If these libraries are installed but `make` complains that it cannot find them, try adding the absolute path(s) of their folder(s) to the `LIBRARY_PATH` and `DYLD_LIBRARY_PATH` or `LD_LIBRARY_PATH` environment variables in your shell's initialisation file (e.g., `~/.profile`, `~/.bash_profile`, or `~/.zprofile`). There you should also set the environment variable `FC` to the name (path) of your Fortran compiler.

Assuming that `oppla` was unpacked/downloaded to the folder `oppla` inside your home directory, the code can be compiled and the R and Matlab `oppla` packages (and dependencies) installed with

```
cd ~/oppla
make
```

You need to run `make` after every update of the Fortran or R code, e.g., after `git pull`. The R and Matlab packages will be installed only if `make` finds R or `matlab` in your `$PATH`.

2 Running OPPLA

2.1 Prerequisites

An `oppla` simulation requires at least 3 files, a control file (Fortran namelist format, default: `oppla_control.nml`, Section 2.3), an initial-conditions file (NetCDF format, Section 2.1.1), and a file with physical boundary conditions (forcing data, Section 2.1.2). All namelists can be in the control file but some, namely those defining the plankton functional types (namelists `dic`, `bac`, `det`, `dom`, `cfo`, and `cmo`) can also go into a separate parameter file (`pfn`). The name of this parameter file can be passed via command-line option `-p` (Section 2.2) or by setting `pfn` in namelist files (Table 2). The contents of the namelists are detailed in Section 2.3 below.

2.1.1 The initial-conditions file

This is a NetCDF file with datasets defining the initial conditions. It can be created from `oppla` output in R with `oppla::oppla_icf()`. An `oppla` output file can also be used directly as an initial-conditions file, e.g., to continue a simulation. The name of the initial-conditions file is stored in the Fortran variable `ifn`, which can be provided with command-line option `-i` or in namelist files. The initial-conditions file contains datasets for longitude, latitude, time, and depth, whose actual names can be specified in namelist `start` (Table 7). Longitude and latitude are scalar datasets. Time can be a scalar or a vector. If it is a vector, the time slice for the initial conditions can be selected with parameter `tin` in namelist `timing` (Table 3). The depth dataset is a vector defining the depth structure of the 1-D grid. An optional depth edges dataset specifies the boundaries between the grid cells. If missing, the depth edges are calculated assuming that they lie midway between adjacent grid-cell centres. The depth structures of the initial-conditions and the forcing-data file must match exactly.

The remaining datasets are the initial conditions for the state variables of the model. Each of these is a vector or time-depth matrix with the same number of depth levels as the depth dataset. The datasets are associated with the state variables via namelist `coco` (Table 9). Datasets not associated in namelist `coco` are ignored. Thus, it is possible to use the same initial-conditions file for simulations with different plankton communities. Namelist `coco` also links the functional types with their specific parameter sets via the species entries in the namelists `bac`, `det`, `dom`, `cfo`, and `cmo`, each of which can occur multiple times in the control or parameter file (Fig. 1).

2.1.2 The forcing file

This is another NetCDF file with datasets defining the boundary conditions, i.e. temperature, salinity, vertical eddy-diffusion coefficients and velocities, and surface and bottom boundary conditions and fluxes (surface irradiance, wind, etc., see Table 6). Constant surface boundary conditions specified in namelist `envi` (Table 8) will be used if not provided in the forcing file. The forcing file must contain datasets for depth, time, latitude, longitude, and the depth dataset must match that in the initial-conditions file. The names of these are specified in namelist `physics`, where also the physical forcing datasets are selected (Table 6). Alkalinity is a special case, as it can be provided in the forcing file, as either surface or profile (forcing) data in namelist `physics`, but it can be also a state variable, which is activated by specifying `'alkalinity'` in the `dima` vector in namelist `coco` (Table 9) and adding alkalinity to the initial-conditions file. Only one of these is allowed.

2.2 Command-line options

The general syntax for an oppla simulation is

`<path to oppla> [<options>] [<control file>]`

where `<path to oppla>` is the full relative or absolute path (including the executable name, `oppla`) to where the executable resides, `<options>` can be any combination of the options shown in the following table, and the `<control file>` (default: `oppla_control.nml`) is a Fortran namelist file. Note that a `<control file>` must be present in the current working directory and that its name must be `oppla_control.nml` if it is not named in the command line.

Table 1: Command-line options and corresponding namelist variables in the `<control file>`.

Option	Variable	Namelist	Default	
<code>-d data</code>	<code>dfn</code>	<code>files</code>		physical forcing data file in NetCDF format
<code>-i init</code>	<code>ifn</code>	<code>files</code>		file with initial conditions (NetCDF)
<code>-o output</code>	<code>ofn*</code>	<code>files</code>		output file name (NetCDF)
<code>-p param</code>	<code>pfn</code>	<code>files</code>	<code><control file></code>	plankton parameter file
<code>-c or -r</code>				resume suspended simulation (initial conditions are read from the end of the output file; option <code>-i</code> and <code>ifn</code> in namelist files (Table 2) are ignored)
<code>-f</code>				overwrite output file if it exists
<code>-m</code>	<code>soma</code>	<code>fluxes</code>	<code>.FALSE.</code>	write source and flux matrices to output file
<code>-v</code>	<code>.NOT.quiet</code>	<code>vode</code>	<code>quiet = .TRUE.</code>	show DVODE errors and warnings

*It is recommended to specify `ofn` as a relative path. If `ofn` is an absolute path, `dfn`, `ifn`, `pfn` and the `<control file>` should also be specified as absolute paths as otherwise `oppla` cannot determine their locations relative to `ofn`.

Pressing ‘CTRL-C’ terminates program execution after the next output step (`dto` in namelist `timing`). Pressing ‘CTRL-C’ a second time terminates execution as soon as possible. To resume a simulation suspended with ‘CTRL-C’, just append `-r` to the command line. To extend a completed simulation, use `-r` and increase `end` in namelist `timing` (Table 3).

2.3 Namelists

The control file is in Fortran namelist format. It can contain entries for all other command line options in its namelists `files` (Table 2) and `start` (Table 7). Command-line options override settings in the control file. Namelist `vode` (Table 4) is for DVODE settings and `physics` (Table 6) for selecting specific datasets, bottom boundary concentrations and surface fluxes in the physical forcing data file. The “Symbol” columns in Tables 15 and 14 (namelists `cfo` and `cmo`) refer to the symbols used in equations.pdf.

Table 2: Namelist files in subroutine `plankton_init` (module `plankton`), to be placed in the control file.

Parameter	Default	Description
<code>dfn*</code>		physical forcing data file in NetCDF format
<code>ifn*</code>		file with initial conditions (NetCDF); this can be the output from a previous run, e.g., for spin-up
<code>ofn*</code>		output file name; should be a relative path if any of the other file names are
<code>pfn*</code>	<code><control file></code>	file with namelists <code>parphy</code> , <code>parzoo</code> , <code>pardon</code> , <code>parfrac</code>

*File names may start with a `~`, a `\`, or an environment variable, e.g., `$HOME`, but this is not recommended for `ofn` as it results in an absolute path, see also note below Table 1.

Table 3: Namelist timing in subroutine plankton_init (module plankton), to be placed in the control file.

Parameter	Default	Description
dto	1 h	output time step
start	0 d	start of simulation relative to the origin of the forcing file time units
end	—	end of simulation relative to the origin of the forcing file time units
tin	0 d	time of initial conditions relative to the origin of the initial-conditions file (ifn) time units
timeout	0 h	max. time allowed for one dvoid step

Initial conditions

Namelist coco

Parameter namelists

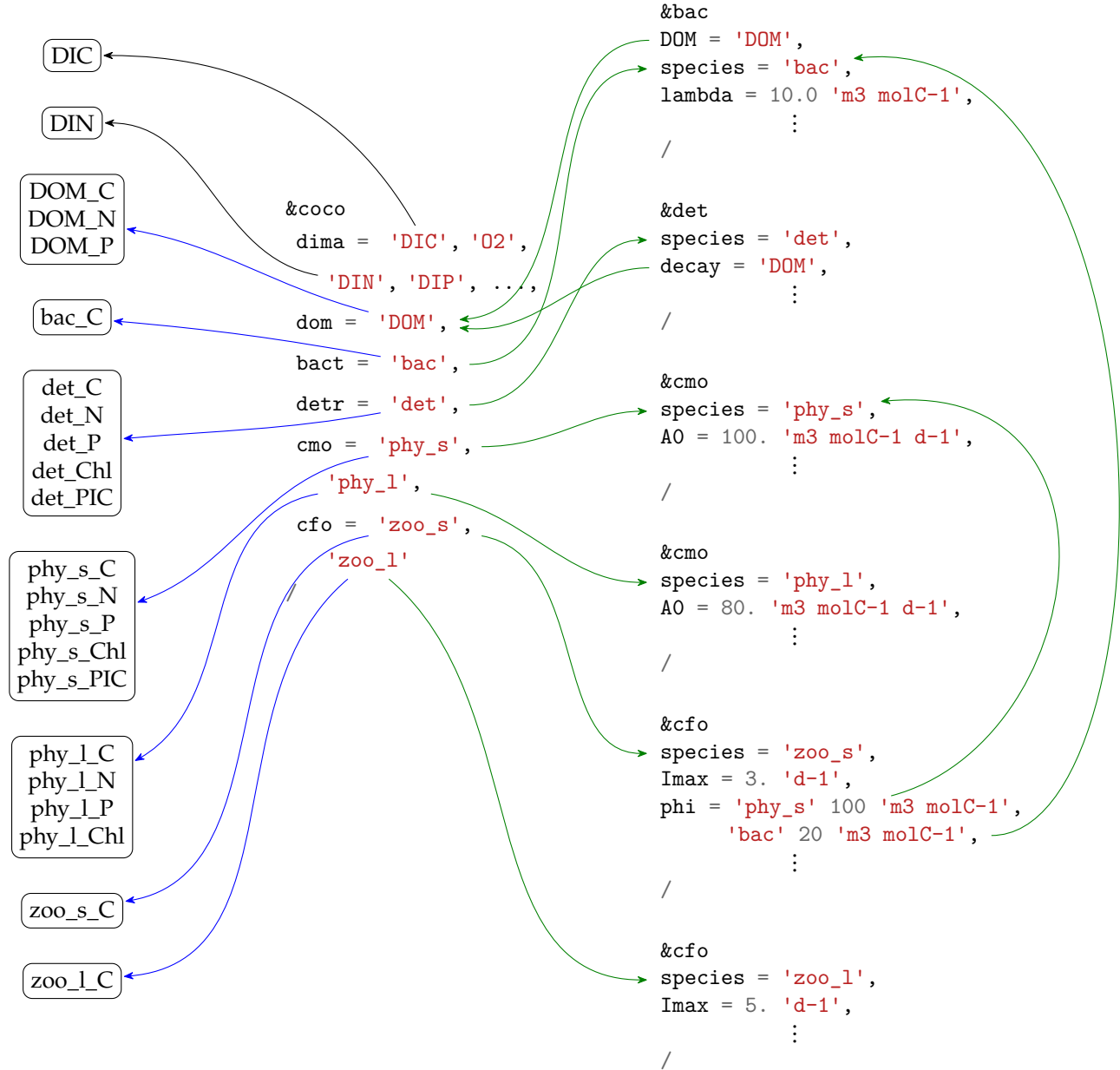


Figure 1: Namelist coco selects (groups of) state variables from datasets in the initial-conditions file (left) and associates the functional groups with their parameters via the species entries of the corresponding parameter namelists (right). The species entries also establish links among functional types, e.g., between the zooplankton food preferences (phi) and the corresponding prey types.

Table 4: Namelist vode in subroutine dcode:ode:read → read_ode (module dcode), to be placed in the control file.

Parameter	Default	Description
atol	1×10^{-9}	absolute tolerance; if atol < 0 then -atol is multiplied with the initial values
rtol	1×10^{-6}	relative tolerance for one time step (dto/nto)
mxatol	10	maximum absolute tolerance
mxrtol	10	maximum relative tolerance
itask	4	task index for DVODE 0: overshoot and interpolate 1: limited overshooting (don't overshoot sunrise and sunset)
jsv	1	how to treat the Jacobian 0: don't save Jacobian 1: save Jacobian
h0	0	size of first step
hmax	0	maximum allowed step size
hmin	0	minimum allowed step size
meth	1	integration method 0: implicit Adams method 1: method based on backward differentiation formulas
miter	0	corrector iteration method 0: functional iteration (no Jacobian) 1: chord iteration with user-supplied full Jacobian 2: chord iteration with internally-generated full Jacobian 3: chord iteration with internally-generated diagonal Jacobian 4: chord iteration with user-supplied banded Jacobian 5: chord iteration with internally-generated banded Jacobian 6: chord iteration with user-supplied sparse Jacobian 7: chord iteration with internally-generated sparse Jacobian
mxstep	500	maximum number of steps for one call to dcode
mxhnil	0	maximum number of messages printed per problem
maxord	12	maximum order: ≤ 5 for non-stiff problems, ≤ 12 for stiff problems
quiet	F	whether to suppress DVODE error messages F corresponds to command-line option -v T corresponds to command-line option -q
constr	T	whether to constrain concentrations and fractions explicitly

Table 5: Namelist start in subroutine innf (module onf), to be placed in the control file.

Parameter	Default	Description
time	'Time'	name of time dataset in dfn
depth	'Depth'	name of depth dataset in dfn
lat	'latitude'	name of latitude dataset in dfn
lon	'longitude'	name of longitude dataset in dfn
svgroup	' '	name of state-variables group in ifn and ofn

Table 6: Namelist physics in subroutine innf (module onf) \Rightarrow states%open, to be placed in the control file.

Parameter	Default	Description																		
advect	'CD'	name of advection scheme; 'CD': modified central differences, 'upwind' (default), or 'dummy': no advection																		
bottom*		types and names of bottom-boundary datasets; the types must be names of state variables and the names are the corresponding datasets in file dfn																		
clsbot	.FALSE.	flag indicating whether the bottom of the model domain is closed																		
depth	'Depth'	name of depth variable in file dfn																		
forcing*		types and names of gridded (depth-time) forcing datasets in file dfn; types can be 'Alkalinity' [†] , 'Salinity', 'Temperature', 'VDC': vertical eddy-diffusion coefficients, 'Velocity': vertical velocities																		
fPAR	0.43	PAR fraction of SWR																		
lat	'latitude'	name of latitude attribute in file dfn																		
lon	'longitude'	name of longitude attribute in file dfn																		
profile*		types and names of temporally constant vertical profiles in file dfn; types can be 'Alkalinity' [†] , 'Salinity', 'Temperature', 'VDC': vertical eddy-diffusion coefficients, 'Velocity': vertical velocities																		
surface*		types and names of surface and latitude datasets: <table><thead><tr><th>type</th><th>description</th></tr></thead><tbody><tr><td>'avgPAR' or 'avgSWR'</td><td>average daytime surface PAR or SWR[‡] (W m⁻²)</td></tr><tr><td>'Ice'</td><td>surface ice cover fraction (1)</td></tr><tr><td>'Wind'</td><td>wind speed at 10 m (m s⁻¹)</td></tr><tr><td>'Pressure'</td><td>surface pressure (Pa)</td></tr><tr><td>'CO2'</td><td>atmospheric pCO₂ (ppm)</td></tr><tr><td>'Alkalinity'[†]</td><td>surface alkalinity (mmol m⁻³)</td></tr><tr><td>'law'</td><td>light attenuation coefficient of water (m⁻¹)</td></tr><tr><td>'Latitude'[§]</td><td>latitude (°N)</td></tr></tbody></table> and the names are the corresponding datasets in file dfn	type	description	'avgPAR' or 'avgSWR'	average daytime surface PAR or SWR [‡] (W m ⁻²)	'Ice'	surface ice cover fraction (1)	'Wind'	wind speed at 10 m (m s ⁻¹)	'Pressure'	surface pressure (Pa)	'CO2'	atmospheric pCO ₂ (ppm)	'Alkalinity' [†]	surface alkalinity (mmol m ⁻³)	'law'	light attenuation coefficient of water (m ⁻¹)	'Latitude' [§]	latitude (°N)
type	description																			
'avgPAR' or 'avgSWR'	average daytime surface PAR or SWR [‡] (W m ⁻²)																			
'Ice'	surface ice cover fraction (1)																			
'Wind'	wind speed at 10 m (m s ⁻¹)																			
'Pressure'	surface pressure (Pa)																			
'CO2'	atmospheric pCO ₂ (ppm)																			
'Alkalinity' [†]	surface alkalinity (mmol m ⁻³)																			
'law'	light attenuation coefficient of water (m ⁻¹)																			
'Latitude' [§]	latitude (°N)																			
surflx*		types and names of surface-flux datasets; the types must be names of state variables and the names are the corresponding datasets in file dfn																		
time	'Time'	name of time variable in file dfn																		

*List whose elements have two components each, a type and a name. The type is the kind of data and the name is the name of a dataset in the physical forcing file. For example, the following selects datasets Temperature, D, and dswrf_NCEP in the forcing file and associates them with Temperature, VDC, and avgSWR, respectively, in oppla:

```
&physics
  forcing = 'Temperature' 'Temperature', 'VDC' 'D',
  surface = 'avgSWR' 'dswrf_NCEP'
/
```

[†]Only if alkalinity is not a state variable.

[§]Latitude changes on a daily time scale only.

[‡]SWR will be converted to PAR by multiplying with fPAR.

Table 7: Namelist fluxes in subroutine plankton_init (plankton.f90), to be placed in the control file (optional).

Parameter	Default	Description
flux_depth	all	vector of (approximate) depths for which to write vertical fluxes to ofn
flux_name	'flux'	name of the flux 3D state-depth-time array of physical fluxes written to ofn
soma	.FALSE.	flag indicating whether to write the flux and source arrays (flux and soma) to ofn
soma_name	'soma'	name of the soma array written to ofn
state_name	'States'	name of the dimension for state-variables in the source matrix soma

Table 8: Namelist envi in plankton_init (module plankton), to be placed in the control file.

Parameter	Default	Description
atmprs	1013.25 hPa	atmospheric pressure ^x
bottom	0	name-value-units vector* of constant bottom boundary conditions
co2	370 ppm	CO ₂ concentration in air ^x
daylen	0.5	day length (only for artificial light cycle)
daylength	'Brock81'	day-length function 'Brock81': day length after Brock (1981) 'Forsythe95': day length incl. twilight after Forsythe et al. (1995)
dicy	''	diurnal light cycle '' or 'none': constant light 'Brock81': natural light cycle after Brock (1981) 'MesoAqua': light cycle for MesoAqua experiments 'LD': fixed rectangular light-dark cycle
fsfalk	0	name-value-units vector* of alkalinity surface-flux factors
kag	—	aggregation kernel (used only if not set in namelist det)
lchl	16 m ² (g.Chl) ⁻¹	light-attenuation coefficient of Chl
lacw	0.04 m ⁻¹	light-attenuation coefficient of water
lapon	16 m ² (mol.N) ⁻¹	light-attenuation coefficient of PON
latdeg	0 °N	latitude ^x
londeg	0 °E	longitude ^x
nbfish	0	number of boxes (layers) with fish
PAR	0 W m ⁻²	surface irradiance ^x
potalk	—	potential alkalinity [†]
ptl	0.8333 degrees	twilight parameter for day-length function 'Forsythe95'
salinity	—	salinity ^x
temperature	—	temperature ^x
totalk	—	total alkalinity [†]
vdc	0 m ² s ⁻¹	vertical eddy-diffusion coefficient ^x
ice	0	surface ice cover fraction ^x
wind	0 m s ⁻¹	wind speed at 10m ^x
velocity	0 m s ⁻¹	vertical velocity ^x

^xused only if not in forcing file

[†]used only if alkalinity is not a state variable and not in forcing file

*each element has three parts: name, value, units

Table 9: Namelist coco (community composition) in plankton_init (module plankton), to be placed in the control file.

Parameter	Default	Description
dima	''	vector of dissolved-matter types, e.g., 'DIC', 'DIN', 'DIP', 'O2', ...
dom	''	name of dissolved-organic matter type
bact	''	vector of bacteria species names
detr	''	vector of detritus-compartment names
cmo	''	vector of phytoplankton and diazotroph species names
cfo	''	vector of zooplankton species names

The names in dima must be names of datasets in the initial-conditions file ifn. The names in the other entries are base-names of such datasets. For example, if one of the species names in cmo is 'phy', the variable species in one of the cmo namelists (Table 14) must be set to 'phy', and ifn must contain four datasets named 'phy_C', 'phy_N', 'phy_P', and 'phy_Ch1'. For base-names in dom and detr there should be three datasets named, e.g., 'det_C', 'det_N', and 'det_P', and for bact and cfo it is one, e.g., 'zoo_C'. See also Fig. 1.

Table 10: Namelist dic in subroutine indic (module dic), to be placed in parameter file pfn.

Parameter	Default	Description
niter	2	number of iterations for pH calculation
pH	8.1	initial guess for pH
pHfun	'Munhoven 2013'	function for calculating pH 'Follows 2006': Follows, Ito & Dutkiewicz (2006) 'Munhoven 2013': Munhoven (2013)
u10	0 ms ⁻¹	wind speed at 10m
rfrnsi	1 mol mol ⁻¹	N:Si Redfield ratio
rfrnp	16 mol mol ⁻¹	N:P Redfield ratio
pvfun	'LM86'	piston velocity function 'LM86': Liss & Merlivat (1986) 'W92': Wanninkhof (1992) 'WG99': Wanninkhof & McGillis (1999)

Table 11: Namelist dom in subroutine bac:dom:read → dom_read (module bac), to be placed in parameter file pfn.

Parameter	Default	Description
ldlc	0 m ² d ⁻¹ W ⁻¹	abiotic light-dependent lability decay rate of DOC
ldln	0 m ² d ⁻¹ W ⁻¹	abiotic light-dependent lability decay rate of DON
trlc	0 d ⁻¹	transformation rate from refractory to labile DOC
trln	0 d ⁻¹	transformation rate from refractory to labile DON
fT	'Eppley'	temperature function
Q10	1.89	Q ₁₀ of the temperature function
Tref	27 °C	reference temperature

Table 12: Namelist(s) bac in subroutine bac:bacteria:read \rightarrow bac_read (module bac), to be placed in file pfn.

Parameter	Default	Description
species	' '	species identifier for IO
DOM	'DOM'	name of labile DOM compartment
fT	'Eppley'	temperature function
fTlam	.FALSE.	flag whether saturation of DOC uptake is temperature dependent
ggem	0.3	maximum gross growth efficiency of bacteria
QN	0.16 mol N (mol.C) ⁻¹	N:C ratio
QP	0.01 mol P (mol.C) ⁻¹	P:C ratio
Q10	1.89	Q ₁₀ of the temperature function
sticky	0	stickiness (susceptibility to aggregation)
Tref	27 °C	reference temperature
Vmax	5 d ⁻¹	maximum DOC uptake rate
lambda	0.1 m ³ molC ⁻¹	Ivlev constant for DOC uptake
ADIM	1 m ³ (mol.C) ⁻¹ d ⁻¹	affinity for DIN and DIP

Table 13: Namelist(s) det in subroutine det:detritus:read \rightarrow det_read (module det), to be placed in file pfn.

Parameter	Default	Description
species	'det'	species identifier for IO
aggregates	.FALSE.	does this group represent aggregates?*
decay	' '	name of compartment receiving decayed detritus constituents; if left empty, detritus C, N, P are directly remineralised to DIC, DIN, DIP
fT	'Eppley'	temperature function for decay rates
kag	-1 d ⁻¹	aggregation kernel
omax	15	saturation (Ω) above which aragonite precipitates
remiC	0 d ⁻¹	detritus C decay rate
formPIC	0 d ⁻¹	detritus PIC formation rate to DIC for $\Omega > \text{omax}$
remiPIC	0 d ⁻¹	detritus PIC decay rate to DIC (remiPIC > 0 activates detritus-PIC)
remiN	0 d ⁻¹	detritus N decay rate
remiP	0 d ⁻¹	detritus P decay rate
remiChl	0 d ⁻¹	detritus Chl decay rate (remiChl > 0 activates detritus-Chl)
sink	0 m d ⁻¹	sinking velocity
sticky	0	stickiness (susceptibility to aggregation)
*can be .TRUE. in one group at most		

Table 14: Namelist(s) cmo for phytoplankton and diazotrophs in subroutine cmo:read → cmo_read (module cmo), to be placed in file pfn.

Parameter	Symbol	Default or Units		Description
species		' '		species identifier for IO
A0	A_0		$\text{m}^3 \text{mol}^{-1} \text{d}^{-1}$	potential nutrient affinity
alpha	α	0	$\text{m}^2 \text{W}^{-1} \text{mol} (\text{g.chl})^{-1} \text{d}^{-1}$	light affinity
DOM		'DOM'		name of labile DOM compartment
FON	F_0^N	0	$\text{mol N} (\text{mol.C})^{-1} \text{d}^{-1}$	potential N_2 fixation rate
fF	f_F	0		degree of N_2 fixation
fPIC	f_{PIC}	0		calcification factor
fT		'Eppley'		temperature function
fTalpha		.FALSE.		is α temperature dependent?
fTNF		' '		temperature function for N_2 fixation
fTRC		.TRUE.		is R_M^{Chl} temperature dependent?
pa		'dynamic'		type of photo-acclimation 'dynamic': fast Chl dynamics (2.13a) 'slow': slow Chl dynamics (2.13b) ' ': no Chl dynamics
QON	Q_0^N	0.04	$\text{mol N} (\text{mol.C})^{-1}$	N subsistence quota (minimum N:C ratio)
QOP	Q_0^P	0.001	$\text{mol P} (\text{mol.C})^{-1}$	P subsistence quota (minimum P:C ratio)
Q10		1.89		Q_{10} of the temperature function
RC	R_M^{Chl}	0.1	d^{-1}	cost of Chl maintenance
rdl		1		daylength parameter
sticky		0		stickiness (susceptibility to aggregation)
Tmax		60	$^{\circ}\text{C}$	upper temperature limit for function fT
Tref		27	$^{\circ}\text{C}$	reference temperature
V0	V_0	5	d^{-1}	potential C, N, P acquisition rate
zC	ζ^{Chl}		$\text{mol C} (\text{g.Chl})^{-1}$	cost of photosynthesis
zF	ζ^F	2	$\text{mol C} (\text{mol.N})^{-1}$	cost of N_2 fixation
zN	ζ^N	0.6	$\text{mol C} (\text{mol.N})^{-1}$	cost of biosynthesis

Table 15: Namelist(s) `cfo` for zooplankton in subroutine `cfo:read` → `cfo_read` (module `cfo`), to be placed in parameter file `pfn`.

Parameter	Symbol	Default	Description
<code>species</code>		<code>' '</code>	species identifier for IO
<code>beta</code>	β	0.2	digestion (assimilation) coefficient
<code>ca</code>	c_a	0.1	cost of assimilation coefficient
<code>cf</code>	c_f	0.1	cost of foraging coefficient
<code>E_{max}</code>	E_{\max}	0.99	maximum assimilation efficiency
<code>fhRm</code>	f_{hib}^R	1	hibernation factor for maintenance respiration (diapause mortality $\approx 0.003 \text{ d}^{-1}$, Heath et al., 2000)
<code>fhphi</code>	f_{hib}^ϕ	1	hibernation factor for feeding; set <code>fhphi</code> = 0 to make this group stop feeding at hibernation depth (Hirche, 1996)
<code>C_{max}</code>	ϕ_0	1 $\text{m}^3 (\text{mol.C})^{-1}$	potential prey capture coefficient
<code>phi</code>	ϕ	—	vector of food preferences
<code>I_{max}</code>	I_{\max}	1 d^{-1}	maximum ingestion rate
<code>R_m</code>	R_M	0 d^{-1}	rate of maintenance respiration
<code>mort</code>	M_{fish}	0 d^{-1}	surface mortality (due to fish)
<code>QN</code>	Q^N	0.16 mol N (mol.C) $^{-1}$	N:C ratio
<code>QP</code>	Q^P	0.01 mol P (mol.C) $^{-1}$	P:C ratio
<code>px</code>		1	ability to egest particles (faecal pellets)
<code>motile</code>		<code>.TRUE.</code>	flag indicating motile species
<code>forage</code>		<code>'cfo'</code>	foraging function: <code>'cfo'</code> or <code>'switch'</code>
<code>egest</code>		<code>'det'</code>	name of target-compartment for egestion
<code>fT</code>		<code>'Eppley'</code>	temperature function
<code>DOM</code>		<code>'DOM'</code>	name of labile DOM compartment
<code>Q10</code>		1.89	Q_{10} of the temperature function
<code>Tref</code>		27 °C	reference temperature
<code>fde</code>	f_{diss}^X	0	dissolved fraction of egestion
<code>svm</code>		<code>' '</code>	seasonal vertical migration type: <code>' '</code> = <code>'static'</code> or <code>'dynamic'</code>
<code>doya</code>	doy_{asc}	—	day of year of ascent for svm (only for <code>svm</code> \neq <code>'dynamic'</code>)
<code>doyd</code>	doy_{des}	—	day of year of descent for svm (only for <code>svm</code> \neq <code>'dynamic'</code>)
<code>d0dvm</code>	d_{night}	0 m	night-time (upper) depth of diel vertical migration
<code>depdvm</code>	d_{day}	0 m	day-time (lower) depth of diel vertical migration
<code>dt_{dvm}</code>	Δt_{dvm}	0 d	time window for DVM around sunrise and sunset
<code>d0svm</code>	d_{summer}	0 m	upper depth of seasonal vertical migration
<code>depsvm</code>	d_{winter}	0 m	lower depth of seasonal vertical migration
<code>dt_{svm}</code>	Δt_{svm}	0 d	temporal spread around <code>doya</code> , <code>doyd</code>
<code>dt_{svma}</code>	Δt_{svm}	0 d	temporal spread around <code>doya</code>
<code>dt_{svmd}</code>	Δt_{svm}	0 d	temporal spread around <code>doyd</code>
<code>vdvm</code>	v_{dvm}	0 m d^{-1}	velocity of diel vertical migration
<code>vsvm</code>	v_{svm}	0 m d^{-1}	seasonal vertical migration velocity (15–20 m d^{-1} , Heath, 1999)

3 Adding a Plankton Group

Adding a new plankton group involves three steps, (1) adding a namelist to the parameter file `pfn`, (2) adding a dataset with the initial profile to the initial-conditions file `ifn`, and (3) adding the new group (species) to the plankton community in namelist `coco`.

3.1 Add a new namelist to `pfn`

Decide which functional type the new group belongs to and select the corresponding namelist type:

Functional Type	Namelist Type
bacteria	bac
dissolved organic matter*	dom
detritus	det
phytoplankton, diazotrophs	cmo
zooplankton	cfo

*Note that you can have only 1 DOM group. The other types have a parameter species for identifying the group in namelist `coco` and the initial-conditions and output files. Each group must have a unique species identifier.

3.2 Add a profile to the initial-conditions file `ifn`

Here is an example for adding another zooplankton group to a start file already containing an initial profile for a zooplankton group named “zoo”, which belongs to type `cfo`. Since the `cfo` type allows only C tracers, the initial profile for zoo is stored in a dataset named “zoo_C”. A new group named “mezoo” can be added in R like this:

```
library(oppla)
nc <- onc_info(filename = ifn, write = TRUE)
zooC <- RNetCDF::var.get.nc(ncfile = nc$self, variable = "zoo_C")
RNetCDF::var.def.nc(ncfile = nc$self, varname = "mezoo_C", vartype = "NC_DOUBLE",
dimensions = nc$var$zoo_C$dimensions)
RNetCDF::att.put.nc(ncfile = nc$self, variable = "mezoo_C", name = "units", type = "NC_CHAR",
value = nc$var$zoo_C$units)
RNetCDF::var.put.nc(ncfile = nc$self, variable = "mezoo_C", data = zooC) #write zoo_C to mezoo_C
RNetCDF::close.nc(nc$self)
```

The species parameter in the new `cfo` namelist must be specified as `species = 'mezoo'`.

3.3 Add the new group (species) to the plankton community

The group must be activated by listing its species name in the corresponding species vector in namelist `coco`. In the above example, this means adding `'mezoo'` to the vector `cfo` in namelist `coco`.

4 The R `oppla` package

The `oppla` R package is normally installed during the make process. The package can be loaded with `library(oppla)` or `require(oppla)`. After upgrading R, you may have to delete the file `R/.oppla_installed` before issuing make again to install `oppla` and its dependencies.

4.1 Creating and modifying forcing files

The function `oppla_forcing_foci()` creates a forcing file from FOCI output. Surface and bottom data, e.g., surface irradiance or wind speed, can be added by passing a list of surface data sets as the surface argument of `oppla_forcing_foci()` or with the function `oppla_forcing_add_ts()`. For example, to add surface alkalinity to an existing forcing file (`"LS_FOCI.nc"`), first prepare a data frame alkalinity with a column `"time"` of class `POSIXct` and a column with the alkalinity data (class `units`). Then,

```
library(oppla)
oppla_forcing_add_ts(forcing_file = "LS_FOCI.nc", new_data = alkalinity)
```

4.2 Reading and processing model output in R

Oppla provides the oppla R package with functions for reading, manipulating and writing namelist files and for reading model output into R. Several R scripts in the R folder are samples for reading data and creating forcing files which may serve as templates for other needs. The following instructions assume that the current working directory is the folder where the simulation was done.

4.2.1 Reading model output

The `oppla_read()` function works reliably with the (default) option `forcing = TRUE` only when the oppla output file is in the location where it was produced. Use `forcing = FALSE` if the output file was copied or moved. Oppla does not write the concentrations of CO_2 , HCO_3^- , and CO_3^{2-} into the output file, but these can be added with the function `oppla_DIC()`. This will only work with the default setting (`forcing = TRUE`) for `oppla_read()`.

```
library(oppla)
oo1 <- oppla_read(file = "output_1.nc") #read state variables
oo2 <- oppla_read(file = "output_2.nc") #read state variables
oo1_dic <- oppla_dic(oppla = oo1) #add CO2, HCO3-, CO32- columns
## the following two commands only work with command-line option -m
## or if soma = T in namelist start
of <- oppla_flux(data = oo1) #read physical (vertical) fluxes
os <- oppla_soma(data = oo1) #read fluxes among state variables (soma)
```

4.2.2 Time slices and initial-condition files

```
ts1 <- oppla_slice(oppla = oo1, date = oo1$date[150]) #extract time slice
... #modify time slice
oppla_icf(oppla = ts1, filename = "ic_ts1.nc") #create initial-conditions file
## create an initial-conditions file for a new depth structure from the first time slice in oo1
nc <- onc_info(filename = "new_forcing.nc", close = TRUE)
depth_new <- units::set_units(nc$dim$depth$vals, nc$dim$depth$units)
oppla_icf(oppla = oo1, filename = "ic_new.nc", depth = depth_new)
```

`oppla_slice()` is a convenience function to facilitate manipulating initial conditions. Function `oppla_icf()` also has an optional `date` argument, which allows creating initial-conditions files from oppla output. Finally, oppla can restart directly from an output file via option `-i` (parameter `ifn` in namelist files, Table 2) and parameter `tin` in namelist timing (Table 3).

4.2.3 Time averages and climatologies

```
oomon <- oppla_monthly(data = oo1) #monthly averages
oomct <- oppla_monthly(data = oo1, avc = "month") #monthly climatology
ooday <- oppla_daily(data = oo1) #daily averages
```

4.2.4 Vertical integrals

```
ooi <- oppla_int(data = oo1, depth = units::set_units(200, "m")) #vertical integrals to 200 m
ooimon <- oppla_int(data = oo1, avc = c("month", "year")) #monthly averages to 100 m
```

Note that `depth` should be specified as a `units` object, otherwise `oppla_int()` assumes `m`.

4.2.5 Level and tile plots

`oppla_level()` and `oppla_tile()` can produce one or several level or tile plots on the same colour scale.

```
## oppla_level uses levelplot() from package lattice:
oppla_level(column = phyt_C, data = ooday, ylim = c(200, 0))
oppla_level(column = DIN / DIP, data = list(oo1, oo2), ylim = c(200, 0))
## oppla_tile uses ggplot() and geom_tile() from package ggplot2:
oppla_tile(column = DIN / DIP, data = list(oo1, oo2), ylim = c(200, 0))
```

Monthly climatologies of oppla results and observations. First create the climatologies, ensuring that both model and data climatologies contain the column to be plotted. Note that the depth edges in the data climatology need not match those in the model results. The following example assumes that the data in `mydata.rds` contains a column named `Chlorophyll` and the model was configured with two phytoplankton groups, `diaz` and `phyt`.

```
data <- readRDS(file = "mydata.rds")
model <- oppla_read(file = "oppla_out.nc")
dmc <- op_aggregate(data = data, edges = (0:20) * 10, datecols = "month") #data climatology
mmc <- oppla_ave(data = model, avc = "month") #model climatology
mmc[j = Chlorophyll := diaz_Ch1 + phyt_Ch1] #add Chlorophyll column to model climatology
oppla_tile(column = Chlorophyll, data = list(data = dmc, model = mmc), ylim = c(200, 0))
```

4.2.6 Namelist files

Fortran namelists can be read into named lists of lists with `nml_read()`, modified with `nml_set()`, and written to files with `nml_write()`. Values of namelist variables can be extracted with `nml_get()`.

```
library(oppla)
nml <- nml_read(file = "oppla_control.nml")
v <- list(A0 = list("cmo", 1, "A0"), # parameter A0 from first namelist cmo
phi = list("cfo", 1, "phi", 1:3)) # first 3 phi values from first namelist cfo
p <- nml_get(nml = nml, vars = v)
p$A0 <- units::as_units(150, units(p$A0))
p$phi[2] <- units::as_units(50, units(p$phi[2]))
nml1 <- nml_set(nml = nml, vars = v, values = p)
nml_write(nml = nml1, file = "oppla_1_control.nml")
```

5 The Matlab oppla packages

The make process installs the Matlab packages `oppla`, `nml`, `odb`, and `eos80` by adding the path to `oppla/MATLAB` to your startup.m, creating it if it did not exist already.

5.1 Reading model output

```
oo1 = oppla.read('output_1.nc'); %state variables, input and forcing files, and soma tables and
oo2 = oppla.read('output_2.nc'); %flux arrays if present in the output files
```

5.2 Climatologies

```
co1 = oppla.cmt(oo1); %climatologies for all state variables
co2 = oppla.cmt(oo2, 'DIC', 'DIN'); %climatologies for DIC and DIN
```

5.3 Oppla database (odb) functions

The odb format is used to store data along with meta-data in a profile-oriented way. An odb database is a Matlab structure with fields `cruise` (cell array of cruise names), `station` (cell array of station names), `file` (cell array of file names of the original data), `prf`, `rec`, and one field for each type of data (Fig. 2). The `rec` field links the data to each other and to the profiles. It is a structure array with fields `depth` and `prf`, where `depth` is the depth of the record and `prf` an index into the `prf` field of the main odb structure. Each data field has fields `rec`, `val`, `units`, where `val` and `units` are the observed values and `units` and `rec` is an index into the `prf` and `depth` vectors inside the `rec` field of the main odb structure. The `prf` field is another structure array with fields `crs` (index into the `crs` field of the main odb structure), `stn` (index into station), `file` (index into file), `date`, `lat`, and `lon`. Thus, each data entry is linked to a record, which associates the data with a depth and a profile, which in turn links the record to meta-data (cruise, station, original file, date, latitude, longitude).

Fields of odb databases are stored as individual variables in mat-files, so should be assigned to a variable when loaded.

```
lsd = load('LS_WOD.mat'); % load odb database
ls_chl = odb.get(lsd, 'Chlorophyll'); % extract all Chlorophyll data
odb.contour(lsd, 'Chlorophyll'); % filled contour plot of all Chlorophyll data
```

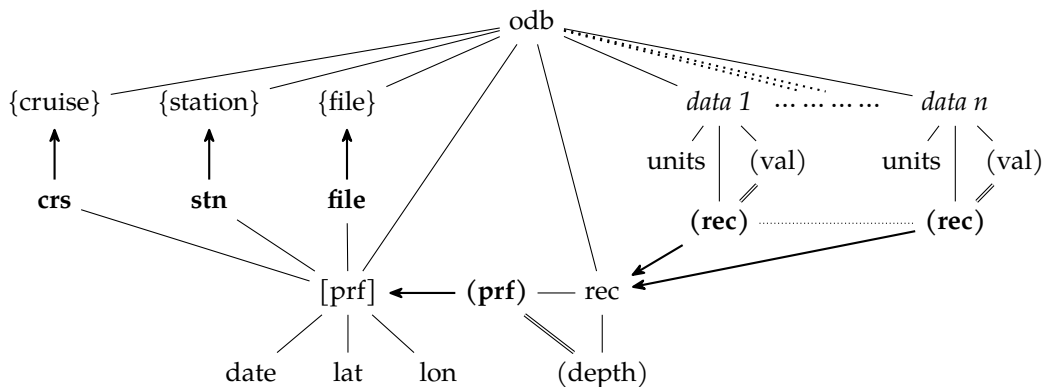


Figure 2: Oppla database (odb) structure. Field names in {} are cell arrays, names in [] are structure arrays, and names in () are vectors. **Bold type** and thick arrows indicate indexing, e.g., the (**rec**) fields of the data fields are index vectors into the (**prf**) and (**depth**) arrays of the rec structure. The (**rec**) and (**val**) data fields have the same size, as do the (**prf**) and (**depth**) fields of rec. Matching elements of the (**rec**) vectors identify data from the same records.

6 Parameter estimation with a genetic algorithm

The suite of tools in the GA folder can be used to apply a genetic algorithm (GA) for parameter estimation. The actual GA is implemented in `GA/gafort.f90`, which is compiled into `GA/gafort`. The executable `gafort` reads its parameters from namelist `ga` in the file `gafort.in` and calls the R-function `getObs()` from `GA/costfun.R` and the bash script `GA/costfun.sh`, which prepare a climatology of observations, execute a set of oppla simulations, and calculate the costs assessing the differences between the observed and simulated climatologies. The observed climatology and the model simulations are based on two files which must be present in the same working directory where you execute the `gafort` program: the oppla control file and the default parameter file. The names of these two files are specified in namelist `ga` in the file `gafort.in` as `oppla_file` and `default_param_file`, respectively. The `default_param_file` is used as a template for generating the individual parameter input files for oppla. It is also used as the parameter set for the first individual in the first generation at the start of a GA application.

Note that the `oppla_file` must specify the file names for the initial conditions (`ifn` in namelist files) and the physical forcing (`dfn` in namelist files) and contain an additional namelist `costfun` with the variables `cut_date` and `max_depth` giving the cut-off date after which and the depth above which the simulation results are used for calculating the model climatology.

The actual cost function is implemented in the R-function `costfun()` in `GA/costfun.R`. The script `GA/costfun.R` is an example for the Labrador Sea, comparing the simulations with observations stored in `R/LS_WOD.rds`. A new `costfun.R` must be created for different areas and datasets, providing the two functions `getObs(file, oppla_inp)` and `costfun(pop_size, cost_file, error_file, model_file, oppla_inp, ...)`, adhering to the same argument lists. The script `costfun.R` and any files accessed by the functions in it must also be present in the working directory from where you execute the `gafort` program.

The script `GA/costfun.sh` is the main interface between `gafort` and the cost function. It first calls `make_inp()` to generate a set of `oppla` parameter input files from parameter sets provided by `gafort` in `oppla_gar_<ng>.nml`, where `ng` is the current generation number. Then `GA/costfun.sh` executes the simulations for these parameter input files for the current generation. Finally, it calls `costfun()` to calculate costs for all individuals and write them to a file (`oppla_cost_GA<n>.csv`) which is then read by `gafort` to generate the parameter sets for the next generation.

6.1 Suggested procedure

1. Create and enter a new folder (GA in the following) for the parameter estimation.
2. Prepare a control file with the names of the initial-condition and physical-forcing files in namelist files and with the namelist `costfun` with entries:

`cut_date` first date in the output file to be used for the cost function

`max_depth` model and data are used down to this depth for the cost function

`param_match` (optional) pairs of parameter names to be set to equal values; it should have the format

`'<name 1>' '<name 1a>', '<name 2>' '<name 2a>', ...`

where `<name 1>` is an element in `param_name` in namelist `ga` in `gafort.in`, e.g., the entry

`param_match 'cfo-2-phi-3' 'cfo-3-phi-3'`

causes `make_inp()` to set `'cfo-3-phi-3'` to the same value as `'cfo-2-phi-3'` in all parameter files.

3. Place the initial-condition and physical-forcing files in GA.
4. Prepare a default parameter file (`default_param_file` in step 6 below) and place it in GA.
5. Prepare (or copy the example) script `costfun.R` and any files used by it, e.g., files with observed data, to GA.

The script `costfun.R` must provide two functions:

- `getObs(file, file_zoo, oppla_inp)`
- `costfun(pop_size, cost_file, error_file, model_file, oppla_inp, ...)`

6. Prepare the `gafort.in` file based on `<oppla>/GA/gafort.in` (`<oppla>` is the `oppla` installation folder).

- entries `oppla_file` and `default_param_file` in namelist `ga` specify the control and default parameter files, specified as `<control>` and `<param>`, respectively, in step 7 below
- `param_name`, `parmin`, and `parmax` specify the names and upper and lower limits of the parameters to be optimised; the names of the parameters should have the format

`'<namelist name>-<namelist index>-<variable name>[-<variable index>]'`

e.g., the scalar variable `alpha` in the first or only `cmo` namelist is `'cmo-1-alpha'` and the third element in the array variable `phi` in the second `cfo` namelist is `'cfo-2-phi-3'`

7. Do a trial simulation and test the `getObs()` and `constfun()` functions in R:

```
mkdir -p fitness1
<oppla>/oppla <control> -p <param> -o fitness1/test.nc #trial simulation
R                                     #start R

obs <- getObs()
costfun(pop_size = 1, cost_file = "cost_test.csv", model_file = "test.nc",
oppla_inp = "<control>", obs = obs)
```

8. Execute the `GA/gafort` executable by specifying the full path to its location inside `<oppla>`.

References

- Brock, T. D. (1981). Calculating solar radiation for ecological studies. *Ecol. Model.* 14(1–2): 1–19.
- Follows, M. J., Ito, T. & Dutkiewicz, S. (2006). On the solution of the carbonate chemistry system in ocean biogeochemistry models. *Ocean Model.* 12: 290–301. doi: 10.1016/j.ocemod.2005.05.004.
- Forsythe, W. C., Rykiel, E. J., Stahl, R. S., Wu, H.-i. & Schoolfield, R. M. (1995). A model comparison for daylength as a function of latitude and day of year. *Ecol. Model.* 80(1): 87–95. doi: 10.1016/0304-3800(94)00034-f.
- Heath, M. R., Astthorsson, O. S., Dunn, J., Ellertsen, B., Gaard, E., Gislason, A., Gurney, W. S. C., Hind, A. T., Irigoien, X., Melle, W., Niehoff, B., Olsen, K., Skreslet, S. & Tande, K. S. (2000). Comparative analysis of *Calanus finmarchicus* demography at locations around the Northeast Atlantic. *ICES J. mar. Sci.* 57: 1562–1580. ISSN: 1054-3139. doi: 10.1006/jmsc.2000.0950.
- Heath, M. (1999). The ascent migration of *Calanus finmarchicus* from overwintering depths in the Faroe–Shetland Channel. *Fisheries Oceanography* 8(Suppl. 1): 84–99. doi: 10.1046/j.1365-2419.1999.00013.x.
- Hirche, H. J. (1996). Diapause in the marine copepod, *Calanus finmarchicus* — A review. *Ophelia* 44: 129–143. ISSN: 0078-5326. doi: 10.1080/00785326.1995.10429843.
- Liss, P. S. & Merlivat, L. (1986). Air-sea gas exchange rates. Introduction and synthesis. In: *The Role of Air-Sea Exchange in Geochemical Cycling*. Ed. by P. Buat-Ménard. Vol. 185. NATO ASI Series C: Mathematical and Physical Sciences. Reidel, Dordrecht, pp. 113–129.
- Munhoven, G. (2013). Mathematics of the total alkalinity-pH equation — pathway to robust and universal solution algorithms: the SolveSAPHE package v1.0.1. *Geosci. Model Dev.* 6: 1367–1388. ISSN: 1991-959X. doi: 10.5194/gmd-6-1367-2013.
- Wanninkhof, R. (1992). Relationship Between Wind Speed and Gas Exchange Over the Ocean. *J. Geophys. Res.* 97(C5): 7373–7382. doi: 10.1029/92JC00188.
- Wanninkhof, R. & McGillis, W. R. (1999). A cubic relationship between air-sea CO₂ exchange and wind speed. *Geophys. Res. Lett.* 26(13): 1889–1892.