Veros-BGC Documentation

Release 0.1.3+0.g9edd102.dirty

The Veros Team

Aug 09, 2021

USAGE

Warning: Veros-BGC is not yet compatible with Veros v1.x.x. The last compatible version is v0.2.3.

Veros-BGC adds a full-fledged NPZD (Nutrients-Phytoplankton-Zooplankton-Detritus) loop on top of the Veros ocean model, with support for custom tracers and additional rules.

Veros-BGC is based on MOBI by Andreas Schmittner, Oregon State University.

Steffen Randrup created Veros-BGC as a part of his Master's thesis.

CHAPTER

ONE

BASIC USAGE

First, install Veros-BGC:

```
$ pip install veros-bgc
```

To get started with a new setup, you can use `bgc_global_4deg` as a template:

```
$ veros copy-setup bgc_global_4deg
```

To enable Veros-BGC on a new setup, you will have to register it as a Veros plugin. Add the following to your setup definition:

```
import veros_bgc

class MySetup(VerosSetup):
    __veros_plugins__ = (veros_bgc,)
```

This registers the plugin for use with Veros. Then, you can use *the Veros-BGC settings* to configure Veros-BGC. The most important settings is `enable_npzd`, which acts as a master switch for Veros-BGC:

```
class MySetup(VerosSetup):
    # ...

    def set_parameter(self, vs):
        vs.enable_npzd = True
```

See also:

All new *settings* and *variables* defined by Veros-BGC in their respective sections.

CHAPTER

TWO

CUSTOM RULES AND TRACERS

The biogeochemistry module for Veros is designed for allowing construction of user defined ecosystems. Base systems are made available for a basic Nutrients-Plankton-Zooplankton-Detritus, NPZD, system, which can optionally be extended by a basic carbon cycle. Enabling the biogeochemistry module and activating the basic NPZD system can be done by setting `enable_npzd = True`.

Ecosystems created with the biogeochemistry module are extensible and any components of them are in principle replaceable. This is handled by three principles: Representation of model tracers by classes, representation of interactions between tracers as rules, and separation of component creation from activation.

2.1 Tracer classes

All model tracers in the biogeochemistry module are created as instances of classes inheriting from a base class `NPZD_tracer`. This class itself inherits from numpy.ndarray, which allows using it like any other Veros variable. The concentration (or appropriate unit) of the tracer within each cell in Veros' grid is stored in the corresponding cell in the tracer grid. The base class defines attributes for operations which may apply to any tracer.

Instances of this class must be created with a numpy array backing the tracer values. Preferably this array was created in variables.py. Additionally a name must be supplied. This name will uniquely identify the tracer during simulation. Optional arguments may be supplied: `transport` By default this value is True. When `transport` is true, the tracer is transported according to the selected transportation scheme. Setting a value for for `sinking_speed` will cause the tracer to be included in calculations of sinking tracers. And setting `light_attenuation` will block downward shortwave radiation proportionally to the concentration of the tracer. This class may itself be used for tracers, which should not express any further features such as the nutrients phosphate.

```
NPZD_tracer(vs.po4, 'po4')
```

Tracers which should express actionable features such as grazing, primary production must implement certain methods. Methods to implement are `mortality` for mortality, `recycle` for recycling, `potential_growth` for primary production. In addition to this methods should be supplied a list of functions representing limiting in growth by nutrients. `grazing` for grazing. This method should return dictionaries for grazing, digestion, excretion, and sloppy feeding. Where the keys are names of the tracers, which have been grazed upon. It is possible to add additional actionable methods by editing npzd.py.

2.1.1 Predefined tracers

A number of classes for tracers have been predefined. These classes can be instantiated with different parameters to defined tracers with varying properties. For example creating tracers for coccolithophores and phytoplankton can be done like

```
coccolithophores = Phytoplankton(np.zeros((3,vs.nx, vs.ny, vs.nz)), 'coccolithophores',
                                  light_attenuation=1,
                                  growth_parameter=0.9,
                                  recycling_rate=0.8,
                                  mortality_rate=0.7)



phytoplankton = Phytoplankton(vs.phytoplankton, 'phytoplankton',
                              light_attenuation=vs.light_attenuation_phytoplankton,
                              growth_parameter=vs.maximum_growth_rate_phyto,
                              recycling_rate=vs.fast_recycling_rate_phytoplankton,
                              mortality_rate=vs.specific_mortality_phytoplankton)
```

Base tracer

class `veros_bgc.core.npzd_tracers.`**`NPZD_tracer`**(*input_array*, *name*, *sinking_speed=None*, *light_attenuation=None*, *transport=True*, *description=None*)

Bases: `numpy.ndarray`

Class for npzd tracers to store additional information about themselves.

Note: Inhenrits from numpy.ndarray to make it work seamless with array operations

Parameters

- **input_array** (`numpy.ndarray`) – Numpy array backing data
- **name** (`str`) – Identifier for the tracer, which must be unique within a given configuration
- **sinking_speed** (`numpy.ndarray`, optional) – Numpy array for how fast the tracer sinks in each cell
- **transport** (`bool` = True, optional) – Whether or not to include the tracer in physical transport
- **light_attenuation** (`numpy.ndarray`, optional) – Factor for how much light is blocked

name
Identifier for the tracer, which must be unique within a given configuration

description
Description of the tracer represented by the class

transport
Whether or not to include the tracer in physical transport

sinking_speed
If set: how fast the tracer sinks in each cell

Type `numpy.ndarray`, optional

`light_attenuation`
If set: Factor for how much light is blocked

Type `numpy.ndarray`, optional

Recyclable tracer

class `veros_bgc.core.npzd_tracers.`**`Recyclable_tracer`**(*input_array*, *name*, *recycling_rate=0*, ***kwargs*)
Bases: `veros_bgc.core.npzd_tracers.NPZD_tracer`

A recyclable tracer

This would be tracer, which may be a tracer like detritus, which can be recycled

Parameters

- **`input_array`** (`numpy.ndarray`) – Numpy array backing data
- **`name`** (`str`) – Identifier for the tracer, which must be unique within a given configuration
- **`recycling_rate`** – A factor scaling the recycling by the population size
- **`**kwargs`** – All named parameters accepted by super class

`recycling_rate`
A factor scaling the recycling by the population size

`+ All attributes held by super class`

Plankton

class `veros_bgc.core.npzd_tracers.`**`Plankton`**(*input_array*, *name*, *mortality_rate=0*, ***kwargs*)
Bases: `veros_bgc.core.npzd_tracers.Recyclable_tracer`

Class for plankton object, which is both recyclable and displays mortality

This class is intended as a base for phytoplankton and zooplankton and not as a standalone class

Note: Typically, it would desirable to also set light attenuation

Parameters

- **`input_array`** (`numpy.ndarray`) – Numpy array backing data
- **`name`** (`str`) – Identifier for the tracer, which must be unique within a given configuration
- **`mortality_rate`** – Rate at which the tracer is dying in mortality method
- **`**kwargs`** – All named parameters accepted by super class

`mortality_rate`
Rate at which the tracer is dying in mortality method

`+ All attributes held by super class`

Phytoplankton

class `veros_bgc.core.npzd_tracers.`**`Phytoplankton`**(*input_array*, *name*, *growth_parameter=0*, ***kwargs*)

Bases: *`veros_bgc.core.npzd_tracers.Plankton`*

Phytoplankton also has primary production

Parameters

- **`input_array`** (`numpy.ndarray`) – Numpy array backing data
- **`name`** (`str`) – Identifier for the tracer, which must be unique within a given configuration
- **`growth_parameter`** – Scaling factor for maximum potential growth
- **`**kwargs`** – All named parameters accepted by super class

`growth_parameter`

Scaling factor for maximum potential growth

`+ All attributes held by super class`

Zooplankton

class `veros_bgc.core.npzd_tracers.`**`Zooplankton`**(*input_array*, *name*, *max_grazing=0*, *grazing_saturation_constant=1*, *grazing_preferences={}*, *assimilation_efficiency=0*, *growth_efficiency=0*, *maximum_growth_temperature=20*, ***kwargs*)

Bases: *`veros_bgc.core.npzd_tracers.Plankton`*

Zooplankton displays quadratic mortality rate but otherwise is similar to ordinary phytoplankton

Parameters

- **`input_array`** (`numpy.ndarray`) – Numpy array backing data
- **`name`** (`str`) – Identifier for the tracer, which must be unique within a given configuration
- **`max_grazing`** – Scaling factor for maximum grazing rate
- **`grazing_saturation_constant`** – Saturation in Michaelis-Menten
- **`grazing_preferences`** – Dictionary of preferences for grazing on other tracers
- **`assimilation_efficiency`** – Fraction of grazed material ingested
- **`growth_efficiency`** – Fraction of ingested material resulting in growth
- **`maximum_growth_temperature`** (= *20*) – Temperature in Celsius where increasing temperature no longer increases grazing
- **`**kwargs`** – All named parameters accepted by super class

`max_grazing`

Scaling factor for maximum grazing rate

`grazing_saturation_constant`

Saturation in Michaelis-Menten

`grazing_preferences`

Dictionary of preferences for grazing on other tracers

`assimilation_efficiency`

Fraction of grazed material ingested

growth_efficiency
Fraction of ingested material resulting in growth

maximum_growth_temperature
Temperature in Celsius where increasing temperature no longer increases grazing

+ All attributes held by super class

2.1.2 Extending tracers

The biogeochemistry tracers make use of the object oriented nature of Python to allow easy extensibility. Tracers which exhibit nearly identical behavior can be created via extension. For example the `Zooplankton` class overrides the mortality function defined by the `Plankton` class

```
class Zooplankton(Plankton):

  # ...

  @veros_method(inline=True)
  def mortality(self, vs):
      """
      Zooplankton mortality is modelled with a quadratic mortality rate
      """
      return self.mortality_rate * self ** 2
```

By using this approach you only have to focus on the differences between tracers.

2.2 Rules

Creating your tracers as objects does not in itself add any time evolution to the system. You must also specify the interaction between the tracers. This is done by creating rules. A rule specifies the flow from one tracer to another. An ecosystem can be defined as a collection of rules each specifying part of the flow between tracers.

Rules consist of a function describing the interaction, the name of the source tracer and the name of the sink tracer. The function itself may be used in several rules. The rule function has access to any variable stored in the Veros object. This includes results of the methods described in the previous section. An example rule could look like

```
@veros_method(inline=True)
def recycling_to_po4(vs, source, sink):
  return {source: -vs.recycled[source], sink: vs.redfield_ratio_PN * vs.recycled[source]}
```

The function returns a dictionary. The keys of the dictionary must be names of the tracers, which are affected by the rule. The values are numpy arrays corresponding to the change in the tracer. The return dictionary is not strictly required to contain two keys. If a rule only represents part of an interaction, just one key can be included. Any number of entries in the dictionary will be processed, but a rule is intended to represent a flow between two tracers. The rule should then be registered with the names of the source and sink to make it available for use in Veros.

```
register_npzd_rule(vs, 'recycle_phytoplankton_to_po4', (recycling_to_po4, 'phytoplankton
↪', 'po4'))
```

The rule is registered with the Veros object as the first argument followed by a unique name for the rule and a tuple consisting of the rule function, the name of the source, and the name of the sink. Those two names will be passed as arguments to the function. The rule name is used for selecting the rule for activation. The tuple may also be replaced by a list containing names of other rules. This collection of rules may later be activated using just the name the list was registered with.

2.2.1 Optional arguments

Rules can also be registered with optional arguments.

The `label` argument specifies a displayed name which is shown in the graph generated by the biogeochemistry diagnostics.

`boundary` may take 3 values. 'SURFACE', 'BOTTOM' or None (default). If 'SURFACE' the rule only applies to the top layer of the grid. 'BOTTOM' means the rule only applies to the cells immediately above the bottom. None means the rule applies to the entire grid.

`group` specifies in which of three execution locations the rule will be applied. The 'PRIMARY' group is the default group. Rules in this group will be evaluated several times in a loop. The number of times specified by the ratio between `vs.dt_tracer` and `vs.dt_bio`. The result of the rule will be time stepped and added to the tracer concentrations. The 'PRE' group is evaluated once per tracer time step before the 'PRIMARY' loop. The results of these rules are not time stepped before adding to the result to the relevant tracers. The 'POST' group is evaluated once before the 'PRIMARY' rules. Time stepping is left out of 'PRE' and 'POST' rules in order to allow them to clean up or reuse results from other rules.

2.2.2 Difference between rules and tracer classes

The difference between rules and classes and their methods is, that the tracer objects themselves do not modify tracer concentrations. Only the rules should influence the time evolution of the tracers. The results of the methods may be used in rules.

2.3 Activation

In order to use the created classes and rules. They must be activated. Tracers are activated by register_npzd_data. Rules are activated by adding their names to npzd_selected_rules for example.

```
detritus = Recyclable_tracer(vs.detritus, 'detritus',
                             sinking_speed=dtr_speed,
                             recycling_rate=vs.remineralization_rate_detritus)
register_npzd_data(vs, detritus)
```

This adds a tracer with the name 'detritus' to the model which sinks and a recycling method.

Rules which have been registered with register_npzd_rule are activated by selecting them with select_npzd_rule. select_npzd_rule accepts rule names. If the name represents a collection of rules, each rule in the collection is activated.

```
# activate the npzd_basic_phytoplankton_grazing rule
select_npzd_rule(vs, 'npzd_basic_phytoplankton_grazing')

# a list of rules, which have been registered with a single name
# may be activated collectively from that name

register_npzd_rule(vs, 'group_npzd_basic', [
      'npzd_basic_phytoplankton_grazing',
      'npzd_basic_phytoplankton_mortality',
```

(continues on next page)

(continued from previous page)

```
      'npzd_basic_phytoplankton_fast_recycling',
      'npzd_basic_phytoplankton_primary_production',
      'npzd_basic_zooplankton_grazing',
      'npzd_basic_zooplankton_mortality',
      'npzd_basic_zooplankton_sloppy_feeding',
      'npzd_basic_detritus_remineralization',
      'npzd_basic_detritus_grazing',
      'npzd_basic_detritus_bottom_remineralization'
])

select_npzd_rule(vs, 'group_npzd_basic')  # This activates all the rules in the␣
↪collection
```

The example setup file for biogeochemistry demonstrates how a configuration file can be used to activate rules.

CHAPTER

THREE

SETUP GALLERY

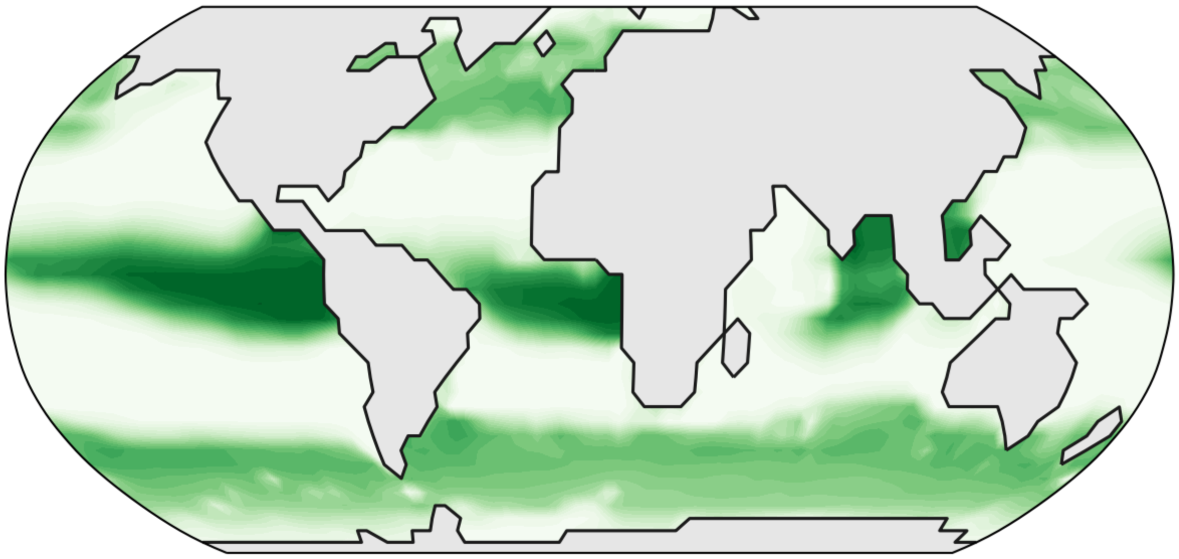
This page gives an overview of the available model setups. To copy the setup file and additional input files (if applicable) to the current working directory, you can make use of the **`veros copy-setup`** command.

Example:

```
$ veros copy-setup bgc_global_4deg
```

3.1 Configurations with Biogeochemistry

Global four-degree model with BioGeoChemistry



3.1.1 Global four-degree model with BioGeoChemistry

`class veros_bgc.setup.bgc_global_4deg.GlobalFourDegreeBGC`(*state=None*, *override=None*, *plugins=None*)

Bases: `veros.veros.VerosSetup`

Global 4 degree model with 15 vertical levels and biogeochemistry.

Reference: Steffen Ole Randrup Kristensen. (2019). Extending the Veros climate simulator with biochemistry. Model design and AMOC collapse, MSc, 67p. https://sid.erda.dk/share_redirect/CVvcrowL22/Thesis/SteffenRandrup_MSc_thesis.pdf.

CHAPTER

FOUR

AVAILABLE SETTINGS

`VerosState.enable_npzd = False`

`VerosState.recycled = {}`
Amount of recycled material [mmol/m^3] for NPZD tracers

`VerosState.mortality = {}`
Amount of dead plankton [mmol/m^3] by species

`VerosState.net_primary_production = {}`
Primary production for each producing plankton species

`VerosState.plankton_growth_functions = {}`
Collection of functions calculating growth for plankton by species

`VerosState.limiting_functions = {}`
Collection of functions calculating limits to growth for plankton by species

`VerosState.npzd_tracers = {}`
Dictionary whose values point to veros variables for npzd tracers

`VerosState.npzd_rules = []`
List of active rules in primary loop of BGC

`VerosState.npzd_pre_rules = []`
List of rules to executed in the pre loop of BGC

`VerosState.npzd_post_rules = []`
Rules to be executed after primary bio loop

`VerosState.npzd_available_rules = {}`
Every rule created is stored here, can be individual rules or collections of rules

`VerosState.npzd_selected_rule_names = []`
name of selected rules

`VerosState.npzd_export = {}`
Exported material from npzd tracers by sinking

`VerosState.npzd_import = {}`
Imported material from npzd tracers from layer above. Takes same value as npzd_export scaled by level differences. Sea surface is 0

`VerosState.zprefs = {}`
Preference for zooplankton to graze on named tracers

`VerosState.npzd_transported_tracers = []`
List of NPZD tracers which are transported

`VerosState.npzd_advection_derivatives = {}`
Stores derivates of advection term for tracers

`VerosState.temporary_tracers = {}`
Temporary copy of npzd_tracers for biogeochemistry loop

`VerosState.light_attenuation_phytoplankton = 0.047`
Light attenuation of phytoplankton

`VerosState.light_attenuation_water = 0.04`
Light attenuation of water [1/m]

`VerosState.light_attenuation_ice = 5.0`
Light attenuation of ice [1/m]

`VerosState.remineralization_rate_detritus = 0`
Remineralization rate of detritus [1/sec]

`VerosState.bbio = 0`
the b in b ** (c*T)

`VerosState.cbio = 0`
the c in b ** (c*T)

`VerosState.maximum_growth_rate_phyto = 0.0`
Maximum growth rate parameter for phytoplankton in [1/sec]

`VerosState.maximum_grazing_rate = 0`
Maximum grazing rate at 0 deg C [1/sec]

`VerosState.fast_recycling_rate_phytoplankton = 0`
Fast-recycling mortality rate of phytoplankton [1/sec]

`VerosState.saturation_constant_N = 0.7`
Half saturation constant for N uptake [mmol N / m^3]

`VerosState.saturation_constant_Z_grazing = 0.15`
Half saturation constant for Z grazing [mmol/m^3]

`VerosState.specific_mortality_phytoplankton = 0`
Specific mortality rate of phytoplankton

`VerosState.quadric_mortality_zooplankton = 0`
Quadric mortality rate of zooplankton [1/ (mmol N ^2 s)]

`VerosState.assimilation_efficiency = 0`
Effiency with which ingested prey is converted growth in zooplankton, range: [0,1]

`VerosState.zooplankton_growth_efficiency = 0`
Zooplankton growth efficiency, range: [0,1]

`VerosState.wd0 = 0.0`
Sinking speed of detritus at surface [m/s]

`VerosState.mwz = 1000`
Depth below which sinking speed of detritus remains constant [m]

`VerosState.mw = 2.3148148148148148e-07`
Increase in sinking speed with depth [1/sec]

`VerosState.zprefP = 1`
Zooplankton preference for grazing on Phytoplankton

`VerosState.zprefZ = 1`
Zooplankton preference for grazing on other zooplankton

`VerosState.zprefDet = 1`
Zooplankton preference for grazing on detritus

`VerosState.redfield_ratio_PN = 0.0625`
Refield ratio for P/N

`VerosState.redfield_ratio_CP = 113.6`
Refield ratio for C/P

`VerosState.redfield_ratio_ON = 10.6`
Redfield ratio for O/N

`VerosState.redfield_ratio_CN = 7.1`
Redfield ratio for C/N

`VerosState.trcmin = 1e-13`
Minimum npzd tracer value

`VerosState.u1_min = 1e-06`
Minimum u1 value for calculating avg J

`VerosState.zooplankton_max_growth_temp = 20.0`
Temperature (C) for which zooplankton growth rate no longer grows with temperature

`VerosState.capr = 0.022`
Carbonate to carbon production ratio

CHAPTER FIVE

AVAILABLE VARIABLES

Attributes:

: Time-dependent

: Included in snapshot output by default

: Written to restart files by default

5.1 Conditional variables

5.1.1 enable_npzd

`VerosState.`**`bottom_mask`**

Units

Dimensions xt, yt, zt

Type `int8`

Attributes

Bottom mask

`VerosState.`**`phytoplankton`**

Units mmol/m^3?

Dimensions xt, yt, zt, timesteps

Type `float`

Attributes

Concentration of phytoplankton in grid box

`VerosState.`**`zooplankton`**

Units mmol/m^3?

Dimensions xt, yt, zt, timesteps

Type `float`

Attributes

Concentration of zooplankton in grid box

`VerosState.`**`detritus`**

Units mmol/m^3?

Dimensions xt, yt, zt, timesteps

Type `float`

Attributes

Concentration of detritus in grid box

`VerosState.`**`po4`**

Units mmol/m^3?

Dimensions xt, yt, zt, timesteps

Type `float`

Attributes

Concentration of phosphate in grid box

`VerosState.`**`swr`**

Units W/m^3?

Dimensions xt, yt

Type `float`

Attributes

Incomming solar radiation at sea level

`VerosState.`**`rctheta`**

Units 1

Dimensions yt

Type `float`

Attributes

Effective vertical coordinate for incoming solar radiation

`VerosState.`**`dayfrac`**

Units 1

Dimensions yt

Type `float`

Attributes

Fraction of day with sunlight

`VerosState.`**`excretion_total`**

Units mmol/m^3 / s

Dimensions xt, yt, zt

Type `float`

Attributes

Zooplankton grazing causes excretion. This stores the total excreted amount for all consumed tracers

5.1.2 enable_carbon

`VerosState.`**`dic`**

Units mmol/m^3

Dimensions xt, yt, zt, timesteps

Type `float`

Attributes

Concentration of inorganic carbon ions and molecule

`VerosState.`**`alkalinity`**

Units mmol/m^3

Dimensions xt, yt, zt, timesteps

Type `float`

Attributes

Combined bases and acids

`VerosState.`**`atmospheric_co2`**

Units ppmv

Dimensions xt, yt

Type `float`

Attributes

Atmospheric co2 concentration

`VerosState.`**`cflux`**

Units mmol/m^2/s

Dimensions xt, yt

Type `float`

Attributes

Flux of CO2 over the ocean-atmosphere bounday

`VerosState.`**`wind_speed`**

Units m/s

Dimensions xt, yt

Type `float`

Attributes

Just used for debugging. Please ignore

`VerosState.`**`hSWS`**

Units 1

Dimensions xt, yt

Type `float`

Attributes

[H] in Sea water sample

`VerosState.pCO2`

Units ?ppmv/atm?

Dimensions xt, yt

Type `float`

Attributes

Partial CO2 pressure

`VerosState.dpCO2`

Units ?ppmv/atm?

Dimensions xt, yt

Type `float`

Attributes

Difference in ocean CO2 pressure and atmospheric

`VerosState.co2star`

Units ?ppmv?

Dimensions xt, yt

Type `float`

Attributes

Adjusted CO2 in ocean

`VerosState.dco2star`

Units ?ppmv?

Dimensions xt, yt

Type `float`

Attributes

Adjusted CO2 difference

`VerosState.rcak`

Units 1

Dimensions xt, yt, zt

Type `float`

Attributes

Calcite is redistributed after production by dissolution varying by depth

CHAPTER

SIX

BIOGEOCHEMISTRY DIAGNOSTIC

This module monitors total phosphate and produces interaction graphs for the biogeochemistry module

class `veros_bgc.diagnostics.npzd_monitor.`**`NPZDMonitor`**(*setup*)

Bases: `veros.diagnostics.diagnostic.VerosDiagnostic`

Diagnostic monitoring nutrients and plankton concentrations

`name = 'npzd'`

`output_frequency = None`

Frequency (in seconds) in which output is written

`save_graph = False`

Whether or not to save a graph of the selected dynamics

INDEX