# Documentation of DEMENTpy v1.0

Dr. Bin Wang
Department of Ecology and Evolutionary Biology
University of California Irvine
bwang7@uci.edu
or wbwenwu@gmail.com
Twitter: @bioatmo_sphere

November 1st, 2019

# 1 Structure

DEMENTpy is an individual-based model of microbial systems that is spatially explicit and trait-based, which is built from the bottom up from gene and metabolism through community all the way up to system-level emergent functions.

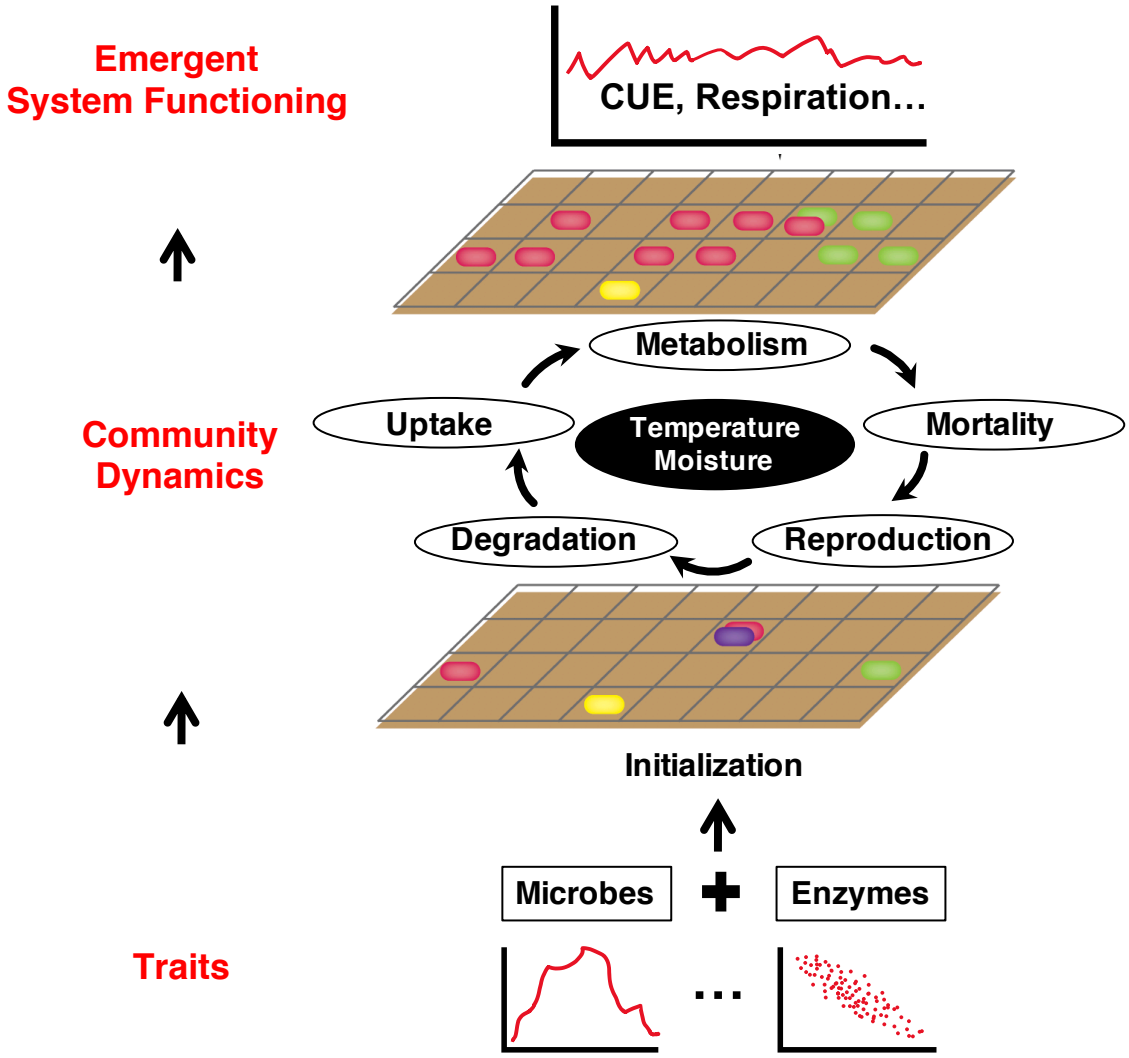


**Figure 1:** DEMENTpy Structure from traits through community to system-level emergent processes.

### 1.1 Modelling Unit

The modelling unit in DEMENTpy is an individual taxon, which is illustrated as below:

## 2 Community Initialization

This is the foremost step in DEMENTpy, as only after this can a microbial community that can behave be initiated. This step also completely embodies the trait-based approach. In detail, a community of being spatially explicit is created based on various traits under a series of assumptions. These traits and assumptions are elaborated below with examples.

***Example code can be displayed showing how trait values are drawn from distributions and assigned to different individuals.***

It must be noteworthy that community initialization goes beyond the very beginning model setup that just initializes a community to start the simulation with. DEMENTpy is built with mimicking the Grassland systems in Southern California, CA. In each new year, microbial community is reinitialized. This reinitialization serves to capturing new community while new litters input into the systems in a new year. To achieve this reinitialization, we calculate cumulative frequency of every taxa over the past year in terms of their abundance, based on which a new spatially explicit microbial community for a new year is derived from the microbial community pool. These steps are executed in DEMENTpy using the repopulation () method in the grid.py module.

Please be aware that this method of reinitialization is different from R-based DEMENT, which instead uses biomass-based frequency calculation.

```
def repopulation(self,output,day,mic_reinit):

    """
    deal with reinitialization of microbial community, subsrates, and
    monomers  on the grid
    ------------------------------------------------------------------------
    Parameters:
        output: an instance of the Output class, in which a variable
                referring to taxon-specific total mass over the grid of
                every iteration is used--MicrobesSeries_repop
        pulse: the pulse index
        day:   the day index
    """
    # reinitialize substrates and monomers
    self.Substrates = self.Substrates_init
    self.Monomers   = self.Monomers_init

    # reinitialize microbial community
    if mic_reinit == 1.00:

        New_microbes = self.Init_Microbes.copy() #NOTE copy()!! bloody lesson
        #fb = self.fb[0:self.n_taxa]
        #max_size_b = self.max_size_b
        #max_size_f = self.max_size_f

        # cumulative abundance; note the column index
        # option 1: mass-based.
        #cum_abundance = output.MicrobesSeries_repop.iloc[:,(day+2-self.cycle):(day+2)].sum(axis=1)
        # option 2: abundance-based
        cum_abundance = output.Taxon_count_repop.iloc[:,(day+2-self.cycle):(day+2)].sum(axis=1)

        # account for different cell mass sizes of bacteria and fungi
        #if sum(fb==1) == 0: # no fungal taxa
        #    frequencies = cum_abundance/cum_abundance.sum()
        #else:
        #    cum_abundance.loc[fb==1] = cum_abundance[fb==1]*max_size_b/max_size_f
        #    frequencies = cum_abundance/cum_abundance.sum()

        # Switched to taxon abundance-based, so no more adjustments
        frequencies = cum_abundance/cum_abundance.sum()
        frequencies = frequencies.fillna(0)

        probs = pd.concat([frequencies,1-frequencies],axis=1,sort=False)

        # Randomly assign microbes to each grid box based on prior densities
        choose_taxa = np.array([0]* self.gridsize * self.n_taxa).reshape(self.n_taxa,self.gridsize)
        for i in range(self.n_taxa):
            # Alternative 1
            choose_taxa[i,:] = np.random.choice([1,0],self.gridsize,replace=True,p=probs.iloc[i,:])

        # Note order='F'
        New_microbes.loc[np.ravel(choose_taxa,order='F')==0] = 0

        # reinitialize the microbial community
        self.Microbes = New_microbes
```

**Figure 2:** Repopulation method in the grid.py module

# 3 Processes

DEMENTpy is a mechanistically explicit model that encompasses processes including degradation of substrates, uptake of monomers, metabolism, mortality, and reproduction. These processes are introduced as below following the order of implementation in the model.

## 3.1 Degradation

## 3.2 Uptake

## 3.3 Metabolism

DEMENTpy now explicitly calculates metabolic production of transporters, enzymes, and osmolytes. However, it is noteworthy that for each of these three categories, it is still far from being explicit, especially for osmolyte, which, though it is assumed to have differing genes within each individual taxon and among different taxa, has constant stoichiometry without knowing specific osmotic compounds.

As regards enzyme, different taxa produce different enzymes that have differing kinetic parameters. These different enzymes still have the same stoichiometry, but entail differing metabolic costs for production by different taxa.

Similar to enzyme, ...

## 3.4 Mortality

Mortality is implemented in DEMENTpy as both a deterministic and a stochastic process. Firstly, a microbial cell dies when its mass reaches a threshold value. In the current version of DEMENTpy the threshold value is assumed to be a constant among taxa. In addition, microbial cells die from a stochastic process, which is, atop a basal mortality probability (differentiated between bacteria and fungi), constrained by drought intensity and drought tolerance, and functional group. These two processes are executed with the deterministic process preceding the stochastic one. The mortality probability is calculated following:

Parameters involved in this mortality process include:

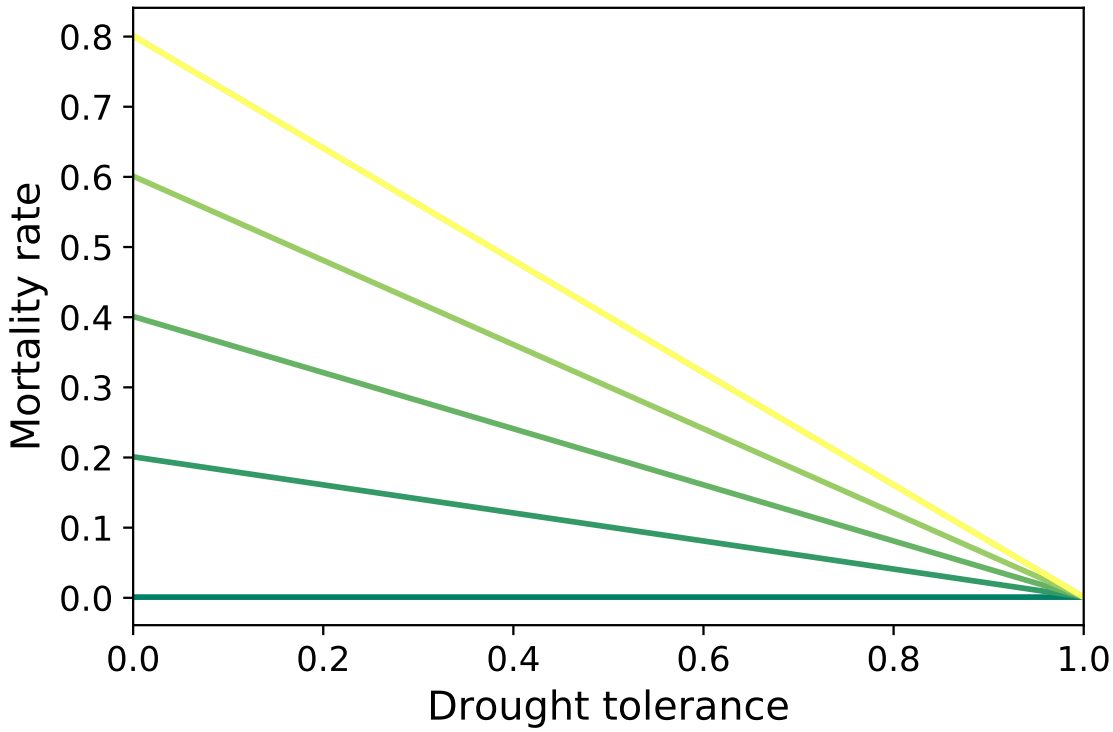


**Figure 3:** Microbial mortality as a function of drought and tolerance.

### 3.5 Reproduction

- -

## 4 Simulation Protocol

–

## 5 DEMENTpy Programming Structure

## 6 Running DEMENTpy

DEMENTpy is open source project, of which the code is archived on GitHub at: https://github.com/bioatmosphere/DEMENTpy.