
description: | API documentation for modules: morphogenDiffusionABM.

lang: en

classoption: onside geometry: margin=1in papersize: a4

linkcolor: blue links-as-notes: true ...

Module `morphogenDiffusionABM`

Agent-based modeling of morphogen diffusion in extracellular cavities and interstitial spaces.

The morphogens are the agents. The class agent has following attributes - id, type, name, pos, track, bState, bdTime, resTime, bdCount, isBound, diffCoef, dist, jump. Another class called 'simulateABM' contains methods to initialize the agents, update the states at each simulation step, and record the observations.

Classes

Class `agent`

```
class agent(  
    id  
)
```

Generates agents and stores their attributes.

Attributes ----- `id`, `type`, `name`, `pos`, `track`, `bState`, `bdTime`,
`resTime`, `bdCount`, `isBound`, `diffCoef`, `dist`, `jump`

Class `simulateABM`

```
class simulateABM(  
    steps,  
    gsize,  
    dir,  
    boundDiff=0.5,  
    recepDens=200  
)
```

Contains methods to run the simulations.

Attributes ----- `steps`, `grid`, `agents`, `gS`, `dir`, `jumpArr`, `boundDiff`, `boundDist`, `recepDens`,
`receptorPos`, `membranePos`, `allowedPos`, ...

Methods

Method `customPDF`

```
def customPDF(  
    self,  
    r,  
    D=10,  
    tau=0.01  
)
```

Generates distribution of probabilities for given jump distances

Args ----- `self`

`r` - array of jump distance (numpy.array)

`D` - diffusion coefficient (in $\mu\text{m}^2/\text{s}$)

`tau` - duration of the simulation step (in s). default = 0.01 (10 ms)

Returns ----- Array of probabilities of jump distances

Method `drawCircle`

```
def drawCircle(
    self,
    pos,
    r
)
```

Returns radial positions at a given distance from the given position

Args ----- `self`

`pos` - [x,y] for given position

`r` - radius of the circle

Returns ----- A list of positions on the circle.

Method `getMemPos`

```
def getMemPos(
    self,
    dir=None
)
```

Detects edges (membrane positions) in the grid.

Args ----- `self`

`dir` - default = None, provide a directory path to save a .tif image of the membrane positions

Modifies ----- Attribute: `membranePos`

Method `getNeighbours`

```
def getNeighbours(
    self,
    pos,
    r,
    any=0
)
```

Updates the `self.neighbours` with neighbours of given position.

Args ----- `self`

`pos` - given position [x,y]

`r` - distance from pos

`any` - boolean

```
0- only take positions at distance r
1- take all positions within radius r
```

Modifies ----- Attributes: `neighbours`

Method `grid2D`

```
def grid2D(
    self,
    gsize,
    gridImg=None,
    nrw=0
)
```

Makes the 2D grid for simulation

Args ----- `self`

`gsize` - size of the grid

`gridImg` - input custom grid array (a `numpy.array` object)

nrw - Narrowness (An integer to make the grid narrower or wider.

nrw=0 means no change in initial grid

nrw>0 means make the grid narrower, **nrw<0** means make the grid wider

1 nrw unit changes the width of extracellular space by 2 pixels or 20 nm.)

Modifies ----- Class [simulateABM](#)

Attributes: **grid**, **membranePos**, **allowedPos**

Method initialize

```
def initialize(  
    self,  
    num=1,  
    name='sailor',  
    diffCoef=10,  
    bindSth=0,  
    oneSide=0,  
    resTime=0,  
    eqFrac=0.5  
)
```

Initializes the agents for the simulation

Args ----- **num** - number of agents

name - name of agent

diffCoef - diffusion coefficient

bindSth - bindSth (integer - 0, 1, ...). A receptor with the radius of bindSth is detected for binding.

oneSide - boolean (default = 0),

0 to initiate agents at random positions,

1 to initiate agents on one side of the grid

resTime - average residence time for the agent

eqFrac - initial bound fraction (0 means all agents are free, 1 means all agents are bound, default = 0.5)

Modifies ----- Class [agent](#)

Attributes: **name**, **id**, **track**, **bState**, **pos**, **resTime**, **bdTime**, **bdCount**, **isBound**, **diffCoef**, **dist**, **jump**, ...

Class [simulateABM](#)

Attributes: **receptorPos**, **agents**

Method observe

```
def observe(  
    self,  
    step,  
    fig,  
    cmap  
)
```

Generates plots of agent tracks

Args ----- **self**

step - simulation step number

fig - plt.figure (blank figure to draw the plot)

cmap - colormap to color different agents

Generates ----- a plot of tracks at self.dir

Method `plotBoundFrac`

```
def plotBoundFrac(  
    self,  
    step,  
    fig  
)
```

To plot the bound fraction.

Method `update`

```
def update(  
    self,  
    step  
)
```

To update the agents at each simulation step

Args ----- `self`

`step` - simulation step number (int)

Modifies ----- Class `agent`

Attributes: `bState`, `pos`, `isBound`, `bdCount`, `jump`

Class `simulateABM`

Attributes: `agents`, `neighbours`