A general mathematical framework for understanding the behavior of heterogeneous stem cell regeneration
Python 移植项目说明

目录

1. 项目索引:

1.1 系统框架之外的数学细节部分
RAND.py: 对应原 cpp 程序 Random.cpp，没有用到系统框架的接口。
1.2 系统框架代码
Cell.py: 细胞对象
CellSystem.py: 模拟实验的主体系统对象
CustomClass.py: 用户可以自定义的对象虚类
CustomFunc.py: 含有两种迭代系统更新时使用的函数虚类
Environment.py: 环境对象
Identifiers.py: 实现字符串和矩阵/数组索引对应的对象，对用户透明
ParaCellsError.py: 报错机制，对用户透明
SystemTools.py: 后续可扩展的对系统操作的工具集
UpdateTools.py: 后续可扩展的对细胞迭代函数集合
1.3 项目主体部分
demo.py: 实现系统框架在本项目的应用实现
1.4 结果输出文件
output.txt: 实验数据分布和原 cpp 代码运行结果一致

2. demo.py 代码说明及系统框架用法

2.1 实验参数设定（9 至 34 行）

```
# 基础设定
RANDMAX = 23371.0
RANDPI = 3.1415926

beta0 = 0.12
kappa0 = 0.2
theta0 = 1e3
eta = 60
p_a1 = 5.8
p_a2 = 2.2
p_a3 = 3.75
p_a4 = 2.0
p_b1 = 4.0
```

```
mu0 = 2e-3
tau = 20

alpha1 = 1.65
alpha2 = 1.56

dt = 0.25
T0 = 50
T1 = 100
ntpr = 4
ntpx = 1
INITCELL = 1024  # 初始细胞数
MAXCELL = 1024  # 最大细胞数
```

对应原项目中的 par.dat 和 md.in 参数文件，这里把数据直接写在代码里，省去从文件读入的环节。

目前在 SystemTools.py 中新加了按一定格式读入初始细胞数、最大细胞数、时间间隔、初始时间和结束时间的函数。其他各项目各不相同的实验参数如需从文件读入需要单独编写程序。

2.2 确定细胞和环境的属性名称和个数（36 至 54 行）

```
# 设置细胞属性和环境参数
X0 = 'X0'
X1 = 'X1'
ProfQ = 'ProfQ'
Type = 'Type'
age = 'age'
cellAttributes = [X0, X1, ProfQ, Type, age]

t = 't'
Prolif = 'Prolif'
NumCell = 'NumCell'
NumPoolCell = 'NumPoolCell'
N0 = 'N0'
N1 = 'N1'
N2 = 'N2'
N3 = 'N3'
N4 = 'N4'
N5 = 'N5'
environmentParameters = [t, Prolif, NumCell, NumPoolCell,
N0, N1, N2, N3, N4, N5]
```

代码中为了利用 ide 的自动补全特性把各个字符串都赋值给了变量名，并用变量名构成 list，也可以直接使用字符串本身构成 list。cellAttributes 列表中都是每个细胞共有的属性，在将其传入 CellSystem 的构造函数时会按列表顺序生成等列数的矩阵（细胞池）；environmentParameters 列表中是环境的参数（在此项目中是统计用途的参数），之后也需要传入 CellSystem 的构造函数，同时在实例化对象中按顺序生成环境数组。

使用系统框架时首先要确定计算的模型，将实际问题转化成如上的结构，确定细胞和环境有哪些属性，在第一步写出两个 list，方便之后构造系统对象。

2.3 数学计算函数（57 至 86 行）

（对应于原 cpp 代码的同名函数，不属于系统框架使用部分，在后面迭代函数中使用）

```
def fdeathrate():
    mu = mu0
    return mu


def fbeta(N0 , x1, x2):
    theta = theta0 * (1.0 + pow(p a4 * x2, 6.0) / (1.0 +
pow(p a4 * x2, 6.0)))
    beta = beta0 * (1.0 / (1.0 + N0  / theta)) * (
           (p a1 * x1 + pow(p a2 * x1, 6.0)) / (1 +
pow(p a3 * x1, 6.0)))
    return beta


def fkappa(x1):
    kappa = kappa0 * 1.0 / (1.0 + pow(p b1 * x1, 6.0))
    return kappa


def GetnextEpi(i, t , x1, x2):
    if i == 0:
        phi = 0.08 + 1.0 * pow(alpha1 * x1, 1.8) / (1 +
pow(alpha1 * x1, 1.8))
    elif i == 1:
        f = 1.0 / (1 + math.exp(-(t  - T0) / 1000.0))
        phi = 0.08 + (1.0 + f * 0.4 / (1 + pow(2.5 * x1,
6))) * pow(alpha2 * x2, 1.8) / (1 + pow(alpha2 * x2,
1.8))

    a = eta * phi
    b = eta * (1 - phi)
    z = RAND.BetaDistribution(a, b)

    return z
```

2.4 迭代函数 1（为细胞初始化的函数）（89 至 93 行）

```
# 所有细胞初始化函数
class CellInitial(CellMethod):
    def run(self, cell: Cell):
        cell.setAttr(X0, random.uniform(0, 1))
        cell.setAttr(X1, random.uniform(0, 0.1))
```

在目前的系统框架中，为了保证迭代函数格式的统一性和使用的方便性，在 CustomFunc.py 中提供了两种（但实际用法类似）的迭代函数虚类：CellMethod 和 CellEnvMethod，两者都没有构造函数（使用 python 默认的类构造函数），且都只有一个需要用户实现的 run 函数，区别在于前者（如上例使用）的 run 函数参数只有一个 cell 对象，在函数内只能调用 Cell.py 中 class Cell 的接口；后者（2.5 中使用）的 run 函数参数有

cell 和 env 函数，额外可以调用 Environment.py 中 class Env 的接口。

注 1：在实际编写时，要注意逻辑上迭代函数都是对每个细胞均操作一遍，细胞池内有多少细胞就调用多少遍——所以如果只希望对系统的 Environment 操作一次的话，将操作语句单独编写在迭代函数外面（见 2.6）。

注 2：对于 Cell、Env、CellSystem 各个接口用法已经在代码中有详细注释，此处暂且省略细节，2.5 中会对一些编写迭代函数时重要且常用的接口进行说明。

如上的 CellInitial 函数主要功能是给每个细胞的随机变量属性 X1、X2 赋初值，分别为 0-1 均匀分布和 0-0.1 均匀分布的随机数。实际上这里并没有真正意义上的“迭代”去更改细胞数量，只是对细胞池的更新，这是因为默认生成的细胞池矩阵中所有数据都是 0，实际使用中首先要将那些需要有初始值的属性进行类似 2.4 中的处理。

2.5 迭代函数 2（涉及细胞增殖/分化的主要迭代函数）（96 至 151 行）

```
# 主要的迭代函数
class CellTypeDecisionFunc(CellEnvMethod):
    def run(self, cell: Cell, env: Env):
        mu = fdeathrate() * dt
        beta = fbeta(env.getAttr(N0), cell.getAttr(X0),
cell.getAttr(X1)) * dt
        kappa = fkappa(cell.getAttr(X0)) * dt

        cell.setAttr(Type, 0)

        if int(cell.getAttr(ProfQ)) == 0:
            rand = RAND.Value_()
            if rand < kappa:
                cell.setAttr(Type, 3)
                cell.remove()
            else:
                if rand < kappa + beta:
                    cell.setAttr(Type, 4)
                    cell.setAttr(ProfQ, 1)
                    cell.setAttr(age, 0)

        if int(cell.getAttr(ProfQ)) == 1:
            rand = RAND.Value ()
            if rand < mu:
                cell.setAttr(Type, 2)
                cell.remove()
            else:
                if cell.getAttr(age) < tau:
                    cell.incrAttr(age, dt)

                else:
                    # 有丝分裂
                    cell.setAttr(Type, 1)
                    cell.proliferate()

                    # 修改子细胞属性
```

```
                # 用 setDaughtersAttr()不加第三个参数默认将子
细胞 1、2 的数值全部修改
                cell.setDaughtersAttr(X0, GetnextEpi(0,
env.getAttr(t), cell.getAttr(X0),

cell.getAttr(X1)))
                cell.setDaughtersAttr(X1, GetnextEpi(1,
env.getAttr(t), cell.getAttr(X0),

cell.getAttr(X1)))
                cell.setDaughtersAttr(ProfQ, 0)
                cell.setDaughtersAttr(age, 0)

        if int(cell.getAttr(Type)) == 3:
            env.incrAttr(N1, 1)
        elif int(cell.getAttr(Type)) == 4 or
int(cell.getAttr(Type)) == 0:
            env.incrAttr(N0, 1)
            if int(cell.getAttr(ProfQ)) == 0:
                env.incrAttr(N5, 1)
            else:
                env.incrAttr(N4, 1)
        elif int(cell.getAttr(Type)) == 1:
            env.incrAttr(N0, 2)
            env.incrAttr(N2, 1)
        elif int(cell.getAttr(Type)) == 2:
            env.incrAttr(N3, 1)
```

对应原 cpp 代码中 System.cpp 内 CSystem::SystemUpdate 的第一个 for 循环。

常用接口说明:

2.5.1 cell 常用接口

Cell.setAttr(属性名，值)：将指定属性设置为给定值。

Cell.getAttr(属性名)：获取属性值。

Cell.incrAttr(属性名，值)：将指定属性增加给定值。

Cell.remove()：调用该接口，会在内部将细胞的“结果细胞数”（resultCellNum）修改为 0，将该细胞的索引添加到待删除细胞列表，在完成对每个细胞的计算后将该列表中所有细胞删除。

Cell.proliferate()：调用该接口，系统框架内部将“结果细胞数”修改为 2，并将细胞目前的状态复制到 cell 对象的子细胞 1 和子细胞 2，此后可以修改子细胞 1，2 的属性，在迭代函数运行完成后，将子细胞 1 覆盖到原先细胞位置，将子细胞 2 粘连到矩阵尾部。

注 1：remove()和 proliferate()的底层区别在于：前者在运行迭代循环时每次只是记录需要移除的细胞索引，迭代函数末尾一次性统一处理；后者在循环中每次即时处理，并且没有记录。

注 2：目前的系统结构设计，在逻辑上来说对 cell 调用增值函数 proliferate 之前细胞不存在子细胞，也就是不能对子细胞的属性进行更改；在底层上看，为了节省内存空间，调用 proliferate 前两个子细胞都是 None，调用后才会复制原来的细胞属性占用内存。所以一定要注意调用 proliferate 函数后再对子细胞操作。

Cell.setDaughtersAttr(属性名，值，*子细胞编号：default 0*)：将子细胞的指定属性改为给定值。子细胞编号默认为 0，如果省略该参数则是对两个子细胞的属性统一修改，也可以传

入参数 1 或 2 单独修改子细胞 1 或子细胞 2。

2.5.2 env 常用接口

Env.setAttr(属性名，值):

Env.getAttr(属性名):

Env.incAttr(属性名，值):

以上三个函数类比 Cell 的对应同名接口用法。

其他接口见代码注释和函数说明字符

2.6 主函数（154 至 192 行）

2.6.1 将系统对象实例化，创建模拟实验环境

```
system = CellSystem(INITCELL, cellAttributes,
environmentParameters, maxCellNum=MAXCELL)
```

一下是 CellSystem 构造函数原型:

```
CellSystem(self, initCellNum: int, cellAttr: list, envParam: list,
maxCellNum: int = 100,customObj: CustomClass = None, dT:float = 0, T0:float
= 0, T1:float = 0):
```

*第一个参数是初始化系统的细胞数，决定了系统刚开始有多少细胞。

另外，系统提供了 CellSystem.addCells(个数，*迭代函数对象：default None*)接口直接添加一定数量的细胞，同时可以用迭代函数初始化这些新添加的细胞。

以及 CellSystem.addCellsFromMatrix(numpy.ndarray 对象，*迭代函数对象：default None*)可以直接传入一个 numpy 的 ndarray 对象，添加到细胞池内。(注意行列数要匹配，否则会报错)，如果有大量的外部数据，可以通过该方法组成细胞池，只需将外部数据读入成 np.ndarray 类型，这也是目前 python 从外部读数据获得结果的最常见类型。

*第二个、第三个参数分别是 2.2 中写出的细胞属性和环境参数列表。

*第四个参数是设置的最大细胞数，不过如果不使用 UpdateTools.py 的 PoolPadding 工具，这个最大细胞数不会影响系统运行。

*第五个参数（可以不使用）需要传入一个实例化的自定义对象（继承 CustomClass），会在系统的构造函数中生成一个与细胞池矩阵每一行对应的自定义对象数组，逻辑上属于细胞的一部分，每个细胞都有一个独立的此类对象，初始时所有对象均为传入构造函数的对象的拷贝。

注 1：继承 CustomClass 的类需要实现一个无参的构造函数，这是由于随着新的细胞的产生，自定义对象也随之增加，使得细胞和自定义对象一一对应，此过程需要调用其无参构造函数。虽然在此处无参构造可能不够灵活，但用户还可以在构造函数之外添加任何其他方法，并且在迭代函数中均可以调用。(通过 Cell.getCustomObj()可以获得本细胞的自定义对象，通过 Cell.getDaughtersObj()可以获得指定编号子细胞或两个子细胞的自定义对象)

*其他参数（dT，T0，T1）均为可选参数，目前只是为用户提供一种存储这些参数的途径，CellSystem 内部还没有设计对应的接口，这三个参数为 public 参数，可以直接通过“.”进行访问和修改。本案例中实际用到时间迭代循环时都是在类的外部实现的。

2.6.2 实例化迭代函数

```
# 实例化迭代函数
cellInitial = CellInitial()
cellDecision = CellTypeDecisionFunc()
poolPadding = PoolPadding()
```

迭代函数在实现上本质是类（见 2.4、2.5），类内部存在一个需要用户实现的 run()方

法，在实际系统运行时的原理是：每次迭代时生成一个代表本轮循环细胞的 Cell 对象和 Env 对象（如果函数继承的是 CellEnvMethod 才会生成 Env 对象），然后用其作为参数运行 run()方法。

CellSystem.updateSystem()（具体见 2.6.3）函数的第一个参数需要传入一个用户实现的迭代函数的实例，因此需要将所有迭代函数类实例化，具体方法如上，只需[实例变量名 = 迭代函数类名()]调用 python 类的默认构造函数。

注 1：此处 PoolPadding()类来自 UpdateTools.py

2.6.3 初始化所有细胞的随机数

```
# 初始化细胞数值
system.updateSystem(cellInitial, fate decision=False)

system.setEnvParam(NumCell, INITCELL)
```

首先调用 2.4 中编写的函数为所有细胞随机数值初始化：

CellSystem.updateSystem()接受两个参数：首先是一个继承自基类 AbstractMethod（CellMethod 和 CellEnvMethod 共同的父类）的迭代函数对象；第二个参数 fate_decision 默认值为 True，也就是说，如果不传入这个参数，默认会在迭代过程中运行检查每个细胞“结果细胞数”并控制细胞移除和增长的部分代码，如果修改为 False 则会跳过从而节省一部分时间。这里我们的 cellInitial 函数不对细胞数做修改，故传入 False。

注 1：如果函数内部没有调用 Cell 类的 remove()和 proliferate()接口，即使 fate_decision 设置为 True 也不会对细胞数量做修改；反之，如果设置为 False 则无论如何不会更改细胞数量，因为运行时会跳过检查“结果细胞数”的部分。

在赋初值之后，调用 CellSystem.setEnvParam()修改环境参数。（CellSystem 常用接口在 3. 中详细说明）

2.6.4 时间循环部分

```
while Time <= T1:
    system.setEnvParam(t, Time)
    system.setEnvParam(N0, 0)
    system.setEnvParam(N1, 0)
    system.setEnvParam(N2, 0)
    system.setEnvParam(N3, 0)
    system.setEnvParam(N4, 0)
    system.setEnvParam(N5, 0)
    system.setEnvParam(NumPoolCell, system.getCellNum())
    # 进行迭代
    system.updateSystem(cellDecision)
    # 每次迭代后的处理
    system.setEnvParam(Prolif, system.getEnvParam(N0) /
system.getEnvParam(NumPoolCell))
    system.setEnvParam(NumCell,
                       system.getEnvParam(NumCell) *
system.getEnvParam(Prolif))
    system.setEnvParam(NumPoolCell, system.getCellNum())
    system.updateSystem(poolPadding)
    Time += dt
    step += 1
    if step % ntpx == 0:
```

```
        for i in range(system.getEnvAttrNum() - 1):
            f.write("%.4f " % system.getEnv()[i])
        f.write('\n')
f.close()
```

对应原 cpp 代码中 CSystem::SystemUpdate 内部出去第一个 for 循环其余部分。

由于原程序中 Ntemp 的值和每次迭代后 N0 的值相等，故为了不调用全局变量加快运行速度删去了这个 Ntemp 变量。

```
    _N0 = 0;          // Number of cells in the pool after cell fate decision.
    _N1 = 0;          // Number of cells removed from resting phase
    _N2 = 0;          // Number of cells undergoing mitosis
    _N3 = 0;          // Number of cells removed from the proliferating phase
    _N4 = 0;          // Number of cells remain unchanged at the proliferating phase
    _N5 = 0;          // Number of cells remain unchanged at the resting phase
```

```
while Time <= T1:
    system.setEnvParam(t, Time)
    system.setEnvParam(N0, 0)
    system.setEnvParam(N1, 0)
    system.setEnvParam(N2, 0)
    system.setEnvParam(N3, 0)
    system.setEnvParam(N4, 0)
    system.setEnvParam(N5, 0)
    system.setEnvParam(NumPoolCell, system.getCellNum())
```

用 CellSystem. setEnvParam()在每次时间循环开始时初始化统计变量 N0 至 N5。

此处的系统当前时间 t 也被写在了环境参数内，属于环境的一部分。

```
# 进行迭代
    system.updateSystem(cellDecision)
```

在每次时间循环处理统计变量后调用 2.5 编写的迭代函数进行主要的计算，这里省略了第二个参数，默认让 fate_decision 代码工作。

```
  _Prolif = 1.0*Ntemp/_NumPoolCell;
  _NumCell =_NumCell * _Prolif;
```

```
# 每次迭代后的处理
    system.setEnvParam(Prolif, system.getEnvParam(N0) /
system.getEnvParam(NumPoolCell))
    system.setEnvParam(NumCell,
                       system.getEnvParam(NumCell) *
system.getEnvParam(Prolif))
    system.setEnvParam(NumPoolCell, system.getCellNum())
```

以上代码相互对应，是每次迭代后的计算部分

```
for (i=1; i<=Ntemp; i++)
{
    if(Rand()<p0 && k<_MaxNumCell)
    {
        k=k+1;
        (*this)(k)._X[0] = X1[i];
        (*this)(k)._X[1] = X2[i];
        (*this)(k)._ProfQ = PQ[i];
        (*this)(k)._age = age[i];
    }
    if (k==_MaxNumCell)
    {
        break;
    }
}
_NumPoolCell = k;
```

这里的控制细胞总数的操作用 UpdateTools.py 中的工具代替。

```
system.updateSystem(poolPadding)
```

3. CellSystem 的主要接口说明

getCellNum()：返回当前细胞数（细胞池行数）
getCellAttrNum()：返回当前细胞属性数（细胞池列数）
getEnvAttrNum()：返回环境参数个数（环境数组长度）
getPool()：返回整个细胞池
getEnv()：返回环境数组
getMaxCellNum()：返回最大细胞数
getObjs()：返回所以细胞的自定义对象
addCells(个数，*迭代函数对象：default None*)：见 2.6.1
addCellsFromMatrix(numpy.ndarray 对象，*迭代函数对象：default None*)：见 2.6.1
addCellAttr(属性名，*迭代函数对象：default None*)：增加细胞属性（给细胞池加入一列），如果有需要为新属性做初始化可以传入一个迭代函数对象，这里的迭代过程和上面两个增加细胞个数的操作类似，不会对细胞进行增殖或死亡/移除。
addEnvParam(参数名，值)：为环境增加一个参数，并设定初始值。
setCellAttr(细胞索引，属性名，值)：为指定位置的细胞修改一个属性的值。
incrCellAttr(细胞索引，属性名，值)：为指定位置的细胞一个属性增加给定值。
setEnvParam(参数名，值)：为环境的一个参数指定新值。
incrEnvParam(参数名，值)：为环境的一个参数增加给定值。
getCellAttr(细胞索引，属性名)：获取指定细胞的一个属性值。
getEnvParam(参数名)：获取指定环境参数值。
updateSystem(*迭代函数对象：default None，fate_decision：default True*)：见 2.6.3
sumAttr(属性名)：返回细胞池指定属性值的和（列求和）
printCells()、printEnvAttr()、printAll()：在终端输出结果

注 1：如果希望在文件中输出实验结果，可以参考本项目中，获取细胞池或属性数组输出到文件中。