

Jul 10, 2020

Generalized mathematical model of cancer heterogeneity

DOI

<https://dx.doi.org/10.17504/protocols.io.bhzej73e>

**Generalized
mathematical
model of cancer
heterogeneity**

Motohiko Naito¹

¹Ronin



Motohiko Naito

Create & collaborate more with a free account

Edit and publish protocols, collaborate in communities, share insights through comments, and track progress with run records.

Create free account

OPEN  ACCESS



DOI: <https://dx.doi.org/10.17504/protocols.io.bhzej73e>

External link: <https://www.biorxiv.org/content/10.1101/2020.05.25.115725v1.full#ref-14>

Protocol Citation: Motohiko Naito 2020. Generalized mathematical model of cancer heterogeneity. **protocols.io**
<https://dx.doi.org/10.17504/protocols.io.bhzej73e>

Manuscript citation:
<https://doi.org/10.1101/2020.05.25.115725>

License: This is an open access protocol distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited

Protocol status: Working

We use this protocol and it's working

Created: June 28, 2020

Last Modified: July 10, 2020

Protocol Integer ID: 38662

Keywords: cancer, heterogeneity, mathematical model, generalized mathematical model of cancer growth, symmetry by cancer clone, conventional cancer growth property, cancer growth, cancer heterogeneity, monte carlo simulation, monte carlo simulations of the solution, cancer clone, stochastic process, stochastic differential equation, field of oncology, oncology, cancer, advances in oncology, mathematical modeling, generalized mathematical model, useful tool for oncologist,

Abstract

The number of reports on mathematical modeling related to oncology is increasing with advances in oncology. Even though the field of oncology has developed significantly over the years, oncology-related experiments remain limited in their ability to examine cancer. To overcome this limitation, in this study, a stochastic process was incorporated into conventional cancer growth properties to obtain a generalized mathematical model of cancer growth. Further, an expression for the violation of symmetry by cancer clones that leads to cancer heterogeneity was derived by solving a stochastic differential equation. Monte Carlo simulations of the solution to the derived equation validate the theories formulated in this study. These findings are expected to provide a deeper understanding of the mechanisms of cancer growth, with Monte Carlo simulation having the potential of being a useful tool for oncologists.

Attachments



[Generalized mathemat...](#)

293KB

Guidelines

References (Sort Version)

1. Funao, N. The R tips 2nd ed. Ohmsha, Tokyo; 2013. (written in Japanese, this book is a kind of textbook for the beginner of R)
2. https://www.jst.go.jp/crest/math/ja/suugakujuku/archive/text/4_Teramae.pdf (written in Japanese)
3. Rossant, C. Ipython interactive computing and visualization Cookbook. Chapter 13: Stochastic dynamical system. Packet Publishing Ltd., Birmingham; 2014
4. https://matplotlib.org/examples/pylab_examples/barchart_demo.html

References (Detailed Protocol)

- (14) Yamazaki H, Naito M, Ghani FI, et al. characterization of cell properties of CD24 and CD26-positive human malignant mesothelioma cells. *Biochem Biophys Res Commun*. 2012; **419**(3): 529–536.
- (16) Feller W. *An introduction to probability theory and its applications*. 3rd ed. Wiley: New Jersey; 1968.
- (17) Mikosch T. *Elementary stochastic calculus with finance in view*. World Science Publishing: Singapore; 1998.

Materials

Software

- Python, matplotlib
- R Software

Safety warnings

 Refer to the Safety Data Sheet (SDS) for any hazards and safety warnings.

Before start

The R software was used to analyze data and simulate cancer cell growth. In addition, R was used to create all graphs except **Fig. 2(a)** and **Fig. 3(a)** (see case 'Detailed Protocol'), which were drawn using Python, matplotlib.

- 1 There's a short protocol available that contains all used codes, and a detailed version that is based on the preprint.

STEP CASE

Short Version and Codes 10 steps

Abstract of the theory to simulate the growth of cancer cells

- 2 Number of the cancer cells at time t :

$$N(t) = n_0 e^{\lambda t} \pm e^{\lambda t} \sqrt{2n_0 \lambda} \int_0^t e^{-\lambda s} dBs. \quad (\text{Eq. 1})$$

where, $N(0)=n_0$, λ is a parameter of the growth of cancer cells, Bs denote the brownian motion function.

- 3 In general, if appropriate experiments or the observations of cancer patients' status are performed, then the mean and variance of cancer growth are obtained. $\hat{m}$, $\hat{v}$, and $\hat{t}$ are defined to be the actual values of mean, variance, and time, respectively.

Note

Note that these three symbols with the hat mark are not algebraic variables but the actual experimental data obtained.

- 4 Thus, following equation are obtained:

$$n_0 = \frac{\hat{m}^2}{\hat{v} + \hat{m}}, \lambda = \frac{1}{\hat{t}} \log_e \left(\frac{\hat{v} + \hat{m}}{\hat{m}} \right)$$

In conclusion, if appropriate experiments or the observations of cancer patients' status are performed, then the number of cancer cells can be solved.

- 5 The interpretations of Eq. (1) are continued. Eq. (1) apparently consists of two equations. Thus, Eq. (1) indicates that cancer growth follows two different formulae.

$N(t) = n_0 e^{\lambda t} + \gamma(t) e^{\lambda t}$, where $\gamma(t) = \sqrt{2n_0 \lambda} \int_0^t e^{-\lambda s} dBs$ is defined as the Ryu pathway and $N(t) = n_0 e^{\lambda t} - \gamma(t) e^{\lambda t}$ as the Ken pathway.

Design of the numerical solution and Monte Carlo simulation

6

Note

This section describes the outline of the numerical solution and simulation conducted in the preprint. The detailed programs written for these calculations are available in the next sections.

The numerical solution of $dN(t) = \lambda N(t)dt + \sigma dBt$ is obtained by employing the Euler–Maruyama method:

n partitions of the interval $(0, t]$ are defined as $(t_{i-1}, t_i]$, $(i = 1, 2, \dots, n)$, where $\Delta t_i := t_{i+1} - t_i$.

Then,

$$\Delta N_i := N_{i+1}(t) - N_i(t) = \lambda \Delta t_i + \sigma \Delta B t_i,$$

where $\Delta B t_i$ is a random number that follows a normal distribution with *mean* = 0 and *sd* = $\sqrt{\Delta t_i}$.

7

By solving the above equation, the numerical solution for $N(t)$ is obtained:

$$N(t) = n_0 + \sum_i \Delta N_i.$$

This calculation is itself the Monte Carlo method. Based on the

law of large numbers, repeating this calculation numerous times can reveal the population mean of cancer growth. This Monte Carlo calculation was repeated 100 times to simulate cancer growth; this is referred to as Monte Carlo simulations.

Code of growth simulation

8

First of all, because #(sentence, or number) is automatically ignored by R software as well as Python software, # is usually used to comment to human.

Note

Note that the code mentioned below has a random number in it. Therefore, even if you run this code with no arrangement, it is not possible to obtain the exact figures presented in this preprint.

Note

The codes shown in Figs 2 and 3 are similar. Thus, only the code shown in Fig. 3 is presented in this manuscript (if you want the data of Fig 2, then you only exchange the value of lam, sigma, and n (appeared first time)).

If you want to re-produce the exactly same data on your PC, then you can get it to type the below code at first:

```
> set.seed (any integral, e.g. 29 )  
# code fixing the random number
```

Fig 3 code

9

Note

Figures can be found in the case 'Detailed Protocol' or in the [preprint](#).

#Generating the function that plots the Ryu pathway of the JMM CV cell lines.

```
###
```

```
# jmmryucv ()
```

```
###
```

```
jmmryucv <-function(){
```

```
#fixing the time and parameters
```

```
dt <- 0.01
```

```
# time interval
```

```
TT <-250
```

```
# end time. TT is 50 in some figures.
```

```
lam <-2.44      # growth rate: lambda, one of the parameters of this equation
```

```
sigma <-18.73*sqrt(dt)
```

```
# sigma is the coefficient of the Brownian motion function.
```

```
n <-188.87
```

```
#substituting the initial number of cancer cells, another parameter of this equation
```

```
#making the list of results
```

```
nn<-n
```

```
tt <- 0.0
```

```
#main body of this calculation
```

```
for(i in 1:TT){
```

```
#calculating the increment. The second term says that "Create the one random number  
that follows normal distribution and multiply it by sigma."
```

```
dn <-(lam*n)*dt+sigma*rnorm(1)
```

```
n <-dn+n
```

```
# if variable i is integral, the latest results are added to the list nn and tt
```

```
if(i %% 1==0){
```

```
nn<-rbind(nn,n)
```

```
tt<-rbind(tt,i);
```

```
}  
}  
#plot the results of the calculations  
matplot(tt,nn,type="l",ylab="Number of Cancer cells", xlab="Time",col="blue",  
ylim=c(0,10000))  
}  
#Generating the function that plots the Ken pathway of JMM CV cell lines  
####  
# jmmkencv ()  
####  
  
jmmkencv <-function(){  
  dt <- 0.01  
  TT <-250  
  lam <-2.44  
  sigma <-18.73*sqrt(dt)  
  n <-188.87  
  nn<=n  
  tt <- 0.0  
  for(i in 1:TT){  
    dn <- (lam*n)*dt-sigma*rnorm(1)  
    n <-dn+n  
    if(i %% 1==0){  
      nn<-rbind(nn,n)  
      tt<-rbind(tt,i);  
    }  
  }  
  matplot(tt,nn,type="l",ylab="Number of Cancer cells",  
  xlab="Time",col="blue",ylim=c(0,1e+05)) # 1e+05 =10,000 at R  
}  
  
# Generating the function that shows the two pathways in one sheet  
  
####  
  
#jmmcvplot ()  
####  
  
jmmcvplot <-function(){  
  jmmkencv()  
  par(new=TRUE)  
  jmmryucv()
```

```
}
```

Calculation codes of the JMM sh-CD24 cell lines, which is basically the same as that of the JMM CV cell lines.

```
####
```

```
# jmmsd24ryu()
```

```
####
```

```
jmmsd24ryu <-function(){
```

```
dt <- 0.01
```

```
TT <-250
```

```
lam <-2.33
```

```
sigma <-21.11*sqrt(dt)
```

```
n <-134.22
```

```
nn<-n
```

```
tt <- 0.0
```

```
for(i in 1:TT){
```

```
dn <-(lam*n)*dt-sigma*rnorm(1)
```

```
n <-dn+n
```

```
if(i %% 1==0){
```

```
nn<-rbind(nn,n)
```

```
tt<-rbind(tt,i);
```

```
}
```

```
}
```

```
matplot(tt,nn,type="l",ylab="Number of Cancer cells",
```

```
xlab="Time",col="red",ylim=c(0,1e+05))
```

```
}
```

```
####
```

```
# jmmsd24ken()
```

```
####
```

```
jmmsd24ken <-function(){
```

```
dt <- 0.01
```

```
TT <-250
```

```
lam <-2.33
```

```
sigma <-21.11*sqrt(dt)
```

```
n <-134.22
```

```
nn<-n
```

```
tt <- 0.0
```

```
for(i in 1:TT){
```

```
dn <-(lam*n)*dt+sigma*rnorm(1)
```

```
n <-dn+n
```

```
if(i %% 1==0){
  nn<-rbind(nn,n)
  tt<-rbind(tt,i);
}
}
matplot(tt,nn,type="l",ylab="Number of Cancer cells",
xlab="Time",col="red",ylim=c(0,1e+05))
}

####
# jmmsd24plot()
####

jmmcd24plot <-function(){
  jmmsd24ken()
  par(new=TRUE)
  jmmsd24ryu()
}

# Calculation codes of the JMM sh-CD26 cell lines
####
# jmmsd26ryu()
####

jmmcd26ryu <-function(){
  dt <- 0.01
  TT <-250
  lam <-2.75
  sigma <-9.74*sqrt(dt)
  n <-17.69
  nn<-n
  tt <- 0.0
  for(i in 1:TT){
    dn <-(lam*n)*dt+sigma*rnorm(1)
    n <-dn+n
    if(i %% 1==0){
      nn<-rbind(nn,n)
      tt<-rbind(tt,i);
    }
  }
  matplot(tt,nn,type="l",ylab="Number of Cancer cells",
xlab="Time",col="black",ylim=c(0,1e+05))
}
```

```
####
# jmmsd26ken()
####
jmmcd26ken <-function(){
dt <- 0.01
TT <-250
lam <-2.75
sigma <-9.74*sqrt(dt)
n <-17.69
nn<-n
tt <- 0.0
for(i in 1:TT){
dn <- (lam*n)*dt-sigma*rnorm(1)
n <-dn+n
if(i %% 1==0){
nn<-rbind(nn,n)
tt<-rbind(tt,i);
}
}
matplot(tt,nn,type="l",ylab="Number of Cancer cells",
xlab="Time",col="black",ylim=c(0,1e+05))
}
```

```
####
# jmmsd26plot()
####
```

```
jmmcd26plot <-function(){
jmmcd26ken()
par(new=TRUE)
jmmcd26ryu()
}
```

Codes of Monte Carlo simulations. These codes generate the plot x times that calculates the above programming.

```
####
# jmmcvgraph (x)
#Generating the function that draws jmmcvplot () x times in one sheet.
####
jmmcvgraph<-function(x){
if(x<=0){
print("Please enter one or more.")
}
```

```
}else{
for(i in 1:x){
if(i>1){
par(new=TRUE)
jmmcvplot ()
}else{
if(i==1){
jmmcvplot ()
}
}
}
}
}

####
# jmmcd24graph (x)

####
jmmcd24graph<-function(x){
if(x<=0){
print("Please enter one or more.")
}else{
for(i in 1:x){
if(i>1){
par(new=TRUE)
jmmcd24plot ()
}else{
if(i==1){
jmmcd24plot ()
}
}
}
}
}

####
# jmmcd26graph (x)

####
jmmcd26graph<-function(x){
if(x<=0){
print("Please enter one or more.")
```

```
}else{
for(i in 1:x){
if(i>1){
par(new=TRUE)
jmmcd26plot ()
}else{
if(i==1){
jmmcd26plot ()
}
}
}
}
}
```

Fig 4 code

```
10  # Simulation code of  $n_0 = 1$  and  $\lambda = 2.5$ 
    ###
    # ryu pathway
    ##

    ryu <-function(){
    dt <-0.01
    TT <-250
    lam <-2.5
    sigma <-sqrt(lam*1*2)*sqrt(dt)
    n <-1
    nn<=n
    tt <= 0.0
    for(i in 1:TT){
    dn <=(lam*n)*dt+sigma*rnorm(1)
    n <=dn+n
    if(i %% 1==0){
    nn<=rbind(nn,n)
    tt<=rbind(tt,i);
    }
    }
    matplot(tt,nn,type="l",ylab="Number of Cancer cells", xlab="Time",col="black",
    ylim=c(0,750))
    }

    ##
    #ken pathway
```

```
##
```

```
ken <-function(){  
  dt <-0.01  
  TT <-250  
  lam <-2.5  
  sigma <-sqrt(lam*1*2)*sqrt(dt)  
  n <-1  
  nn<-n  
  tt <= 0.0  
  for(i in 1:TT){  
    dn <=(lam*n)*dt-sigma*rnorm(1)  
    n <-dn+n  
    if(i %% 1==0){  
      nn<-rbind(nn,n)  
      tt<-rbind(tt,i);  
    }  
  }  
  matplot(tt,nn,type="l",ylab="Number of Cancer cells", xlab="Time",col="red",  
  ylim=c(0,750))  
}
```

```
##
```

```
# Simulation code of growing cancer clones
```

```
##
```

```
#Ryu code of the JMM CV cell lines
```

```
jmmngryucv <-function(){
```

```
  dt <= 0.01  
  TT <-250  
  lam <-2.44  
  sigma <-18.73*sqrt(dt)  
  n <-188.87
```

```
  nn<-n  
  tt <= 0.0  
  for(i in 1:TT){  
    dn <= sigma*rnorm(1)  
    n <-dn+n  
    if(i %% 1==0){  
      nn<-rbind(nn,n)  
      tt<-rbind(tt,i);  
    }  
  }  
}
```

```
matplot(tt,nn,type="l",ylab="Number of Growing clones", xlab="Time", col="black",
ylim=c(0,250))
}
```

```
#Ken code of the JMM CV cell lines
```

```
jmmngkencv <-function(){
```

```
dt <- 0.01
```

```
TT <-250
```

```
lam <-2.44
```

```
sigma <-18.73*sqrt(dt)
```

```
n <-188.87
```

```
nn<-n
```

```
tt <- 0.0
```

```
for(i in 1:TT){
```

```
dn <- -sigma*rnorm(1)
```

```
n <-dn+n
```

```
if(i %% 1==0){
```

```
nn<-rbind(nn,n)
```

```
tt<-rbind(tt,i);
```

```
}
```

```
}
```

```
matplot(tt,nn,type="l",ylab="Number of Growing clones", xlab="Time", col="red",
ylim=c(0,250))
```

```
}
```

```
# JMM sh-cd26 simulations
```

```
# Ryu code
```

```
jmmcd26ngryu <-function(){
```

```
dt <- 0.01
```

```
TT <-250
```

```
lam <-2.75
```

```
sigma <-9.74*sqrt(dt)
```

```
n <-17.69
```

```
nn<-n
```

```
tt <- 0.0
```

```
for(i in 1:TT){
```

```
dn <-sigma*rnorm(1)
```

```
n <-dn+n
```

```
if(i %% 1==0){
```

```
nn<-rbind(nn,n)
```

```
tt<-rbind(tt,i);
}
}
matplot(tt,nn,type="l",ylab="Number of Growing Cancer cells", xlab="Time", col="black",
ylim=c(-10,30))
par(new=TRUE)
matplot(tt, 0*tt, type="l", lty=6, ylab="Number of Growing Cancer cells", xlab="Time",
col="blue", ylim=c(-10,30))
}

#Ken code
jmmcd26ngken <-function(){
dt <- 0.01
TT <-250
lam <-2.75
sigma <-9.74*sqrt(dt)
n <-17.69
nn<-n
tt <- 0.0
for(i in 1:TT){
dn <-sigma*rnorm(1)
n <-dn+n
if(i %% 1==0){
nn<-rbind(nn,n)
tt<-rbind(tt,i);
}
}
matplot(tt,nn,type="l",ylab="Number of Growing Cancer cells", xlab="Time", col="red",
ylim=c(-10,30))
par(new=TRUE)
matplot(tt, 0*tt, type="l", lty=6, ylab="Number of Growing Cancer cells", xlab="Time",
col="red", ylim=c(-10,30))
}

# Plotting the growing clones of the JMM sh-CD26 cell line
jmmcd26ngplt <- function(){
jmmcd26ngken()
par(new=TRUE)
jmmcd26ngryu()
}
```

Fig 2(a) and Fig 3(a) code

```
11 #Note: This graph is written by Python, matplotlib
import numpy as np
import matplotlib.pyplot as plt
JMMcvm = (1., 8.93, 23.46, 79.63)
JMMcvsd = (0., 0.15, 2.79, 57.90)
JMMsh24m = (1., 7.37, 13.47, 29.18)
JMMsh24sd = (0., 0.28, 0.43, 18.91)
JMMsh26m = (1., 4.40, 10.44, 24.04)
JMMsh26sd = (0., 0.87, 2.29, 4.57)
#the above six results are same as those listed in Table 1
n_groups = 4
fig, ax = plt.subplots()
index = np.arange(n_groups)
bar_width = 0.3
# Drawing the bar graph
JMMcvBar = plt.bar(index, JMMcvm, bar_width,
alpha=opacity,
color='blue',
yerr=JMMcvsd,
error_kw=error_config)
JMMsh24Bar = plt.bar(index + bar_width, JMMsh24m, bar_width,
alpha=opacity,
color='red',
yerr=JMMsh24sd,
error_kw=error_config)
JMMsh26Bar = plt.bar(index + 2*bar_width, JMMsh26m, bar_width,
alpha=opacity,
color='black',
yerr=JMMsh26sd,
error_kw=error_config)
plt.xlabel('Time')
plt.ylabel('Number of cancer cells')
plt.xticks(index + 2*bar_width / 3, ('Day1', 'Day4', 'Day7', 'Day10'))
plt.tight_layout()
plt.show()
```