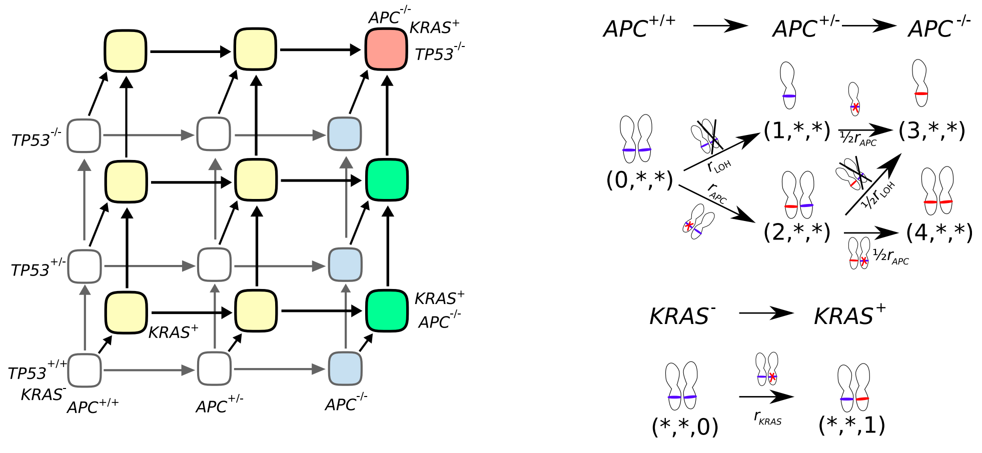
# MATHEMATICAL MODEL OF COLORECTAL CANCER INITIATION

ZIHAN CHEN, HAORAN XIANG, YUYUE YUAN, OLIVIA DING, YIRUI CHEN

Abstract. Colorectal cancer (CRC) initiation is driven by the accumulation of critical genetic alterations, including the inactivation of tumor suppressor genes (*APC* and *TP53*) and the activation of the oncogene *KRAS*. This study builds upon the foundational stochastic model of Paterson et al.[1], reformulating it into a matrix-based framework to enhance the analysis of mutation dynamics and crypt expansion. By leveraging transition and growth matrices, we model the progression of 32 genotypic states, capturing the selective growth advantages of *APC* and *KRAS* mutations. Comparative results demonstrate that while the matrix approach aligns with tau-leaping simulations, it consistently underestimates malignancy probabilities at early ages, highlighting discrepancies attributable to outflow terms and double mutation assumptions. Our findings underscore the utility of matrix methods in elucidating cancer evolution and provide insights into CRC risk modeling, aligning theoretical predictions with observed epidemiological data.

## 1. Introduction

Cancer development is a complex evolutionary process driven by the accumulation of genetic alterations. In colorectal cancer (CRC), this progression typically involves specific mutations in key genes that regulate cell growth and division. By reviewing references [2, 3], we gained a basic understanding of colorectal cancer, learning about the critical role of three driver gene mutations in the development of lung and colorectal cancers. This forms the basis for recent research[1], which has established that just three driver mutations are often sufficient for malignant transformation: the inactivation of two tumor suppressor genes (TSGs) and the activation of one oncogene.



(*A*)

Complete state space showing all possible combinations of mutations in *APC*, *TP53*, and *KRAS*.

(*B*)

Detailed mechanisms of gene inactivation through mutation and loss of heterozygosity.

Figure 1. State transition diagram of colorectal cancer evolution[1]

Paterson et al.[1], in their seminal work, developed a mathematical model of CRC initiation that focuses on three critical genetic events: the inactivation of tumor suppressors *APC* and *TP53*,

*Date*: December 11, 2024.

[1]Figure 1 is adapted from the paper by Paterson et al[1].

and the activation of the *KRAS* oncogene. Their model, illustrated in Figure 1, presents the complex network of possible evolutionary pathways to cancer. Figure 1($A$) shows the full state space of genetic alterations, where each node represents a unique combination of mutation states, and arrows indicate possible transitions between states. Figure 1($B$) details the specific mechanisms of gene inactivation, showing how *APC* and *TP53* can be inactivated through either mutation or loss of heterozygosity (LOH), while *KRAS* activation requires only a single mutation event. This comprehensive model accounts for all possible orders of mutation acquisition, allowing for a more realistic representation of cancer evolution compared to traditional linear models.

In this paper, we first analyze the two mathematical models used in Figure 2—*ODE system* and *Tau-leaping*. Then we reproduced the figure using the same method from the paper. Building upon their work, and inspired by [4], we present an alternative mathematical formulation using a *matrix-based* approach. Our method reformulates the evolutionary dynamics shown in Figure 1 as a system of linear transformations, where a transition matrix captures all possible mutation events between states, and a growth matrix represents the selective advantages conferred by different mutations. Following this theoretical foundation, we present comparative results between our matrix approach and the original methods, analyzing the similarities and differences in their predictions. For the second model, $\tau$-leaping, we drew inspiration from the simulation methods described in [5] and [6]. After comparing the strengths and weaknesses of these two approaches, we developed our own implementation of $\tau$-leaping in Python to better align with our framework.

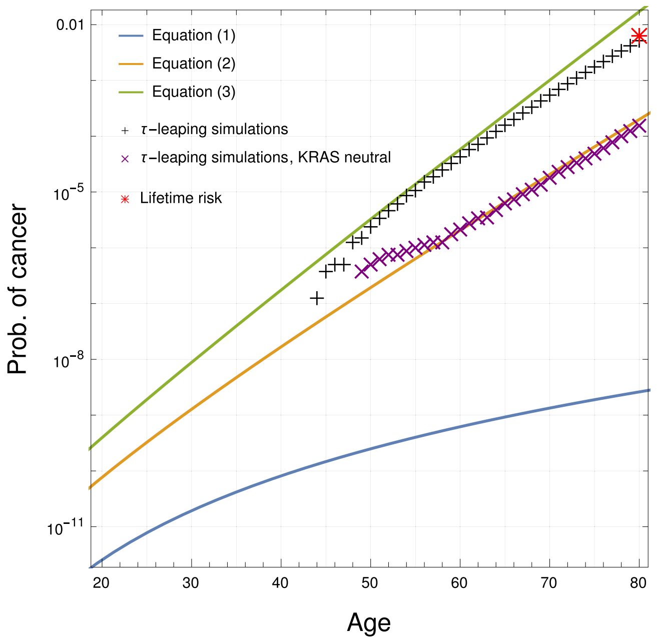


Figure 2. Age-dependent Probability of Cancer [2]

## 2. Mathematical Model

### Model Equations

Paterson et al. propose a stochastic mathematical model to describe the dynamics of driver gene mutations and crypt expansion during the initiation of CRC. This study primarily focuses on the probability of disease occurrence in humans, emphasizing the probability of cancerous crypts reaching the malignant state. The model is constructed by deriving the rate of change in the number of crypts for each genotype, combined with experimental parameter estimates from the

[2]Figure 2 is adapted from the paper by Paterson et al[1].

original study, to create a complete dynamic framework. Below are the detailed derivations and methods of the model.

2.1. **Transition Rules and Parameter Settings.** For the mutation in crypts, the driver events occur as stochastic processes. The inactivation of *APC* and *TP53* requires both alleles to be inactivated, while the activation of *KRAS* only requires a single mutation. The inactivation of tumor suppressor genes is governed by both mutations and loss of heterozygosity (LOH) which is depicted in Figure 1$(B)$. For example, in the case of *APC*:

- If the first allele is inactivated through LOH, the second allele can only be inactivated through mutation at a rate $r_{\text{APC}}/2$.
- If the first allele is inactivated through mutation, the second allele can be inactivated either through LOH at a rate $r_{\text{LOH}}/2$ or through mutation at a rate $r_{\text{APC}}/2$.

A similar mechanism applies to *TP53*. The activation of *KRAS* occurs at a rate $r_{\text{KRAS}}$ and requires only a single mutation. The mutation rate $r_{\text{APC}}$ is calculated as $r_{\text{APC}} = u \cdot n_{\text{APC}}$, and we can get $r_{\text{KRAS}}$, $r_{\text{TP53}}$ in same way. The parameter setting in the model can be found in Table 1.

| **Parameter** | **Value** | **Unit** | **Definition** |
|---|---|---|---|
| $u$ | $1.25 \times 10^{-8}$ | per year | Base pair mutation rate per year |
| $t$ | $0 - 80$ | year | The time that model predict |
| $n$ | $10^8$ | - | Total number of crypts in the colon |
| $n_{\text{APC}}$ | 604 | - | Number of driver sites in the APC gene |
| $n_{\text{P53}}$ | 73 | - | Number of driver sites in the TP53 gene |
| $n_{\text{KRAS}}$ | 20 | - | Number of driver sites in the KRAS gene |
| $b_1$ | 0.2 | per year | Division rate of APC$-/-$ crypts |
| $b_2$ | 0.07 | per year | Division rate of KRAS+ crypts |
| $r_{\text{LOH}}$ | $1.36 \times 10^{-4}$ | per year | Loss of heterozygosity rate |
| $r_{\text{TP53lost}}$ | 0.0 | per year | Division rate of TP53$-/-$ crypts |
| $r_{\text{LD}}$ | 0.0 | per year | Division rate of crypts with other mutations |
| $APC_{\text{mult1}}$ | 2.1 | - | Fixation multiplier for the first APC mutation |
| $APC_{\text{mult2}}$ | 2.8 | - | Fixation multiplier for the second APC mutation |
| $KRAS_{\text{mult}}$ | 3.6 | - | Fixation multiplier for KRAS mutation |

Table 1. Model Parameters and Definitions

The genotype of a crypt is represented as a triplet $(i, j, k)$, where:

- $i \in \{0, 1, 2, 3, 4\}$: Activation state of *APC*.
- $j \in \{0, 1, 2, 3, 4\}$: Activation state of *TP53*.
- $k \in \{0, 1\}$: Activation state of *KRAS*.

We use numbers to represent different states, with all possible states listed in Table 2. Since *APC* and *TP53* share the same transformation rules, the states represented by the numbers are defined using the same set of rules.

| $i$ | **Description of *APC* State** | $k$ | **Description of *KRAS* State** |
|---|---|---|---|
| 0 | Both *APC* alleles are wild-type | 0 | *KRAS* is wild-type |
| 1 | One *APC* allele lost through LOH (single loss) | 1 | *KRAS* is mutated |
| 2 | One *APC* allele inactivated by mutation (single mutation) | | |
| 3 | One *APC* allele lost (LOH), one inactivated by mutation | | |
| 4 | Both *APC* alleles inactivated by mutation (double mutation) | | |

Table 2. States of $i$ and $k$

The model allows a total genotype space of $5 \cdot 5 \cdot 2 = 50$ states, including the malignant genotypes $(3, 3, 1)$, $(4, 3, 1)$, $(3, 4, 1)$, and $(4, 4, 1)$, which correspond to crypts that have accumulated all three driver events.

In theoretical studies, it is recognized that genetic mutations can lead to malignant transformations, necessitating the simulation of this process in the model. The model simulates crypts carrying *APC* or *KRAS* driver mutations gaining a growth advantage, resulting in increased division rates, which further enhance the likelihood of crypt malignant transformation. Specifically, the study sets the division rate of $APC^{-/-}$ crypts as $b_1 = 0.2$/year, the division rate of $KRAS^{+}$ crypts as $b_2 = 0.07$/year, and the division rate of double-mutant $APC^{-/-}KRAS^{+}$ crypts as $b_{12} = b_1 + b_2 = 0.27$/year. It is assumed that genotypes without *APC* inactivation or *KRAS* activation have a division rate of 0. This assumption is based on experimental data indicating that normal crypts in adult colonic tissue divide very rarely[7].

2.2. **Probabilistic Model and Analysis.** We use the mean-field approximation to construct probabilistic models for the dynamics of crypt mutation and clonal expansion. In this approximation, the probability of double mutations is considered negligible, significantly reducing the complexity of the system. This allows the number of distinct genotypes to be reduced to 32, with the malignant state fully turned on represented by $(3, 3, 1)$.

The probabilities of crypts with different genotypes transitioning to malignant states are comprehensively modeled through various mutation and LOH pathways. When a specific mutation step involves only one allele (e.g., LOH or single-allele mutation), the likelihood of one allele mutating is factored in, halving the mutation rate to reflect this probability. To determine the probability of the first malignant crypt arising from the non-malignant crypt population, while excluding contributions from already malignant crypts, the factor $(1 - P(t))$ is included in the ODE. The rate of change of $P(t)$ is given as:

$$P'(t) = \left(\frac{1}{2} r_{\mathrm{LOH}} n_{231} + \frac{1}{2} r_{\mathrm{APC}} n_{131} + \frac{1}{2} r_{\mathrm{LOH}} n_{321} + \frac{1}{2} r_{\mathrm{TP53}} n_{311} + r_{\mathrm{KRAS}} n_{330}\right)(1 - P(t)).$$

The equation for $P(t)$ is derived by solving the differential equation $P'(t)$, which describes the probability that at least one cancerous cryptis present by time:

$$P(t) = 1 - \exp\left(-\int \left(\frac{1}{2} r_{\mathrm{LOH}} n_{231} + \frac{1}{2} r_{\mathrm{APC}} n_{131} + \frac{1}{2} r_{\mathrm{LOH}} n_{321} + \frac{1}{2} r_{\mathrm{TP53}} n_{311} + r_{\mathrm{KRAS}} n_{330}\right) dt\right).$$

This equation forms the foundation for analyzing $P(t)$, representing the cumulative probability of at least one malignant crypt by time $t$. However, solving $P(t)$ requires knowledge of the expected number of crypts for each genotype $n_{ijk}(t)$, which is derived from a system of differential equations.

The system of differential equations describes the time evolution of $n_{ijk}(t)$, the expected number of crypts with genotype $(i, j, k)$. These equations track the accumulation of driver mutations, starting from the initial state where all crypts are wild-type. The system assumes linear first-order dynamics, with "outflow" terms neglected due to their minimal impact given the small mutation rates ($r_{\mathrm{LOH}} \ll 1$)[1].

The construction of $n'_{ijk}(t)$ involves two key processes: *inflows* and *clonal expansion.* Inflows account for transitions from precursor genotypes via mutation or loss of heterozygosity (LOH), where the inflow rate is proportional to the mutation or LOH rate and the number of crypts in the precursor genotype. The clonal expansion represents the growth advantages provided by mutations, such as $b_1 = 0.2$/year for $APC^{-/-}$ crypts and $b_2 = 0.07$/year for $KRAS^{+}$ crypts, which increase the population size of these genotypes over time.

This system provides the input for evaluating $P(t)$, with solutions tailored to the mutation dynamics and growth effects in the simulated system.

2.3. **Case Analysis.** To understand the contribution of driver mutations to cancer risk, $P(t)$ is analyzed under different cases of driver growth advantages. Each case isolates specific biological scenarios, offering insights into how *APC* and *KRAS* mutations influence cancer initiation.

**Case 1.** All Drivers Neutral

- Assumption: $b_1 = b_2 = b_{12} = 0$, meaning no crypt growth advantage is provided by *APC* or *KRAS* mutations.
- $n_{000}$ is constant ($n_{000} \approx N$), and other solutions $n_{ijk}(t)$ are simple polynomial functions of $t$, derived by integrating mutation and LOH rates.
- $P(t)$ is computed as:

$$P(t) \approx N r_{\mathrm{APC}} r_{\mathrm{TP53}} r_{\mathrm{KRAS}} r_{\mathrm{LOH}}^2 \frac{t^5}{4}. \tag{1}$$

**Case 2.** *APC* Only Has Advantage

- Assumption: $b_1 > 0$, $b_2 = 0$, and $b_{12} = b_1$.
- The growth advantage of $APC^{-/-}$ crypts introduces exponential terms ($e^{b_1 t}$) in $n_{321}$, $n_{311}$, and $n_{330}$, significantly impacting $P(t)$.
- $P(t)$ is computed as:

$$P(t) \approx \frac{3}{2} N r_{\mathrm{APC}} r_{\mathrm{TP53}} r_{\mathrm{KRAS}} r_{\mathrm{LOH}}^2 \frac{t^2}{b_1^3} e^{b_1 t}. \tag{2}$$

**Case 3.** *APC* And *KRAS* Have Advantage

- Assumption: $b_1 > 0$, $b_2 > 0$, and $b_{12} = b_1 + b_2$.
- $APC^{-/-}$ and $KRAS^{+}$ crypts jointly contribute to the growth advantage. Solutions for $n_{231}, n_{131}, n_{321}, n_{311}, n_{330}$ involve $b_1$, $b_2$, and $b_{12}$, reflecting their combined impact.
- In addition to considering the proliferation rate of mutated crypts, we explicitly introduce the fixation multiplier in the case of double malignant mutations because the cumulative selective advantage and synergistic effects of double malignant mutations significantly influence the fixation probability and expansion capacity of mutated crypts[8]. We use $c = APC_{\mathrm{mult1}} \times APC_{\mathrm{mult2}} \times KRAS_{\mathrm{mult}}$ as the correction to the formula to account for the increased chance of fixation of *APC* and *KRAS* mutated in a crypt.
- The $P(t)$ is computed as:

$$P(t) \approx c N r_{\mathrm{APC}} r_{\mathrm{TP53}} r_{\mathrm{KRAS}} r_{\mathrm{LOH}}^2 \frac{t}{b_{12}^3 (b_{12} - b_1)(b_{12} - b_2)} e^{b_{12} t}. \tag{3}$$

## Tau-leaping Simulation

According to Paterson et al., two numerical simulation methods are introduced and implemented to predict the probability of CRC initiation with time: *Gillespie's Stochastic Simulation Algorithm*[5] and *$\tau$-leaping*[6]. Considering the expensive computational cost of Gillespie algorithm, we choose to use Python to reproduce the $\tau$-leaping numerical simulation.

2.4. **Gillespie's Stochastic Simulation Algorithm.** The first method is Gillespie's Stochastic Simulation Algorithm (SSA). SSA provides an exact realization of the underlying master equation by simulating each event individually. CRC initiation probability modeling in Figure 2 of Paterson et al paper, where multiple driver genes (*APC*, *TP53*, *KRAS*) can mutate at low rates (on the order of $10^{-8}$ per base pair per year) amidst a high background rate of more common events such as crypt fission, the SSA's fidelity comes at the price of immense computational expense. Mathematically, the SSA proceeds by first computing the total event rate $\Gamma = \sum_i n_i \omega_i$, where $n_i$ is the population at site $i$ and $\omega_i$ is the sum of rates $w_i^j$ for all reactions $j$ at that site. The next event time $\Delta t$ is then drawn from an exponential distribution with parameter $\Gamma$, $\Delta t \sim \mathrm{Exp}(\Gamma^{-1})$, and the specific event is selected with probability proportional to its rate. However, as $\Gamma$ becomes very large—dominated by frequent fission events rather than rare mutations—the required number of steps grows prohibitively large. The flow of the Gillespie Algorithm is shown in Figure 3 below.

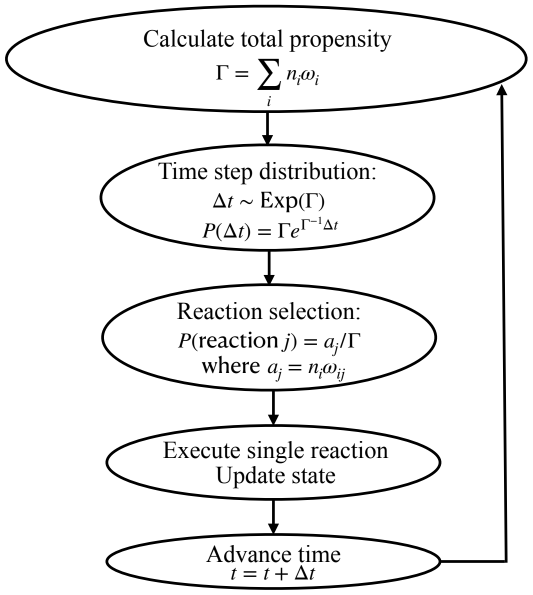

Calculate total propensity
$\Gamma = \sum_i n_i\omega_i$
Time step distribution:
$\Delta t \sim \text{Exp}(\Gamma)$
$P(\Delta t) = \Gamma e^{\Gamma^{-1}\Delta t}$
Reaction selection:
$P(\text{reaction } j) = a_j/\Gamma$
where $a_j = n_i\omega_{ij}$
Execute single reaction
Update state
Advance time
$t = t + \Delta t$

Figure 3. Gillespie Algorithm Scheme Diagram

2.5. **$\tau$-leaping.** In contrast, the tau-leaping method trades some exactness for greater efficiency. Rather than sampling one event at a time, tau-leaping fixes a time step $\tau$ and draws the number of events of each type from appropriate distributions over that interval. For example, mutation events from state $i$ to state $j$ might be incremented by a Poisson-distributed random variable $\Delta n_{i\to j} \sim \text{Poisson}(n_i r_{i\to j}\tau)$. This allows the simulation to "leap" forward by time $\tau$ without enumerating each event. Moreover, for strongly proliferative processes like crypt fission, the paper describes using a negative binomial distribution to capture the "contagious" nature of division events more realistically, rather than relying solely on Poisson increments.

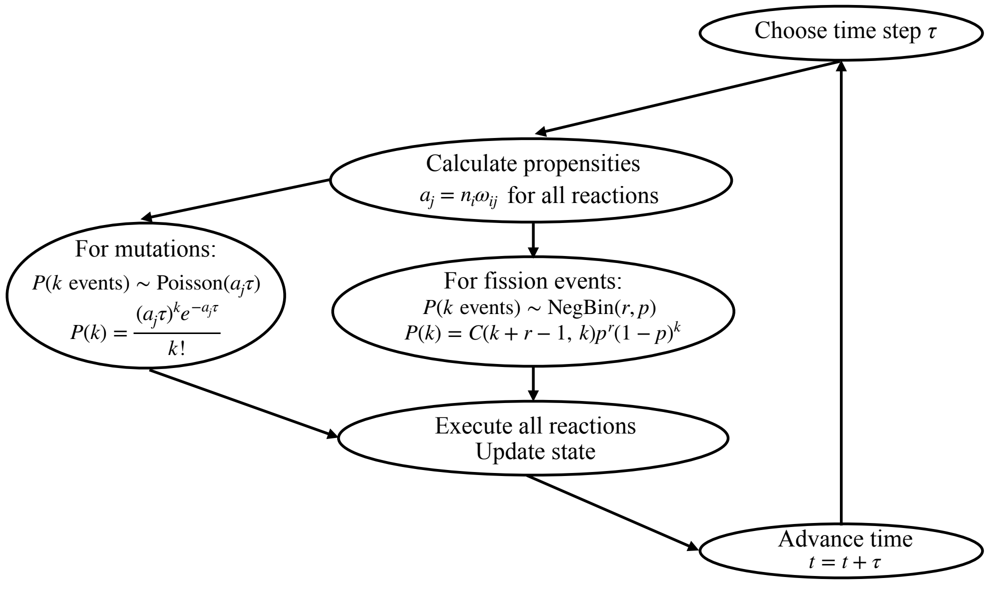

Choose time step $\tau$
Calculate propensities
$a_j = n_i\omega_{ij}$ for all reactions
For mutations:
$P(k \text{ events}) \sim \text{Poisson}(a_j\tau)$
$P(k) = \frac{(a_j\tau)^k e^{-a_j\tau}}{k!}$
For fission events:
$P(k \text{ events}) \sim \text{NegBin}(r,p)$
$P(k) = C(k+r-1, k)p^r(1-p)^k$
Execute all reactions
Update state
Advance time
$t = t + \tau$

Figure 4. Tau-leaping method Scheme Diagram

2.6. **Python Implementation.** While the Python code (Appendix B.1.2) we developed attempted to implement $\tau$-leaping foundational ideas, it did not fully replicate the recommended approach— particularly Step 3, which involves drawing the number of fission events from a negative binomial distribution $\Delta n_i \sim \mathrm{NegBinomial}(n_i, p)$ with mean $n_i b_i \tau$. Step 3 involves representing branched paths of mutation and fission events, as illustrated in the tau-leaping scheme diagram in Figure 4 above. Instead, our code employed a mixture of Poisson, binomial, and negative binomial distributions in an ad-hoc manner, failing to precisely follow the protocol and thereby reducing the accuracy of the tau-leaping approximation. The comparison of our Python implementation and the original simulation in the paper is shown in Figure 5.

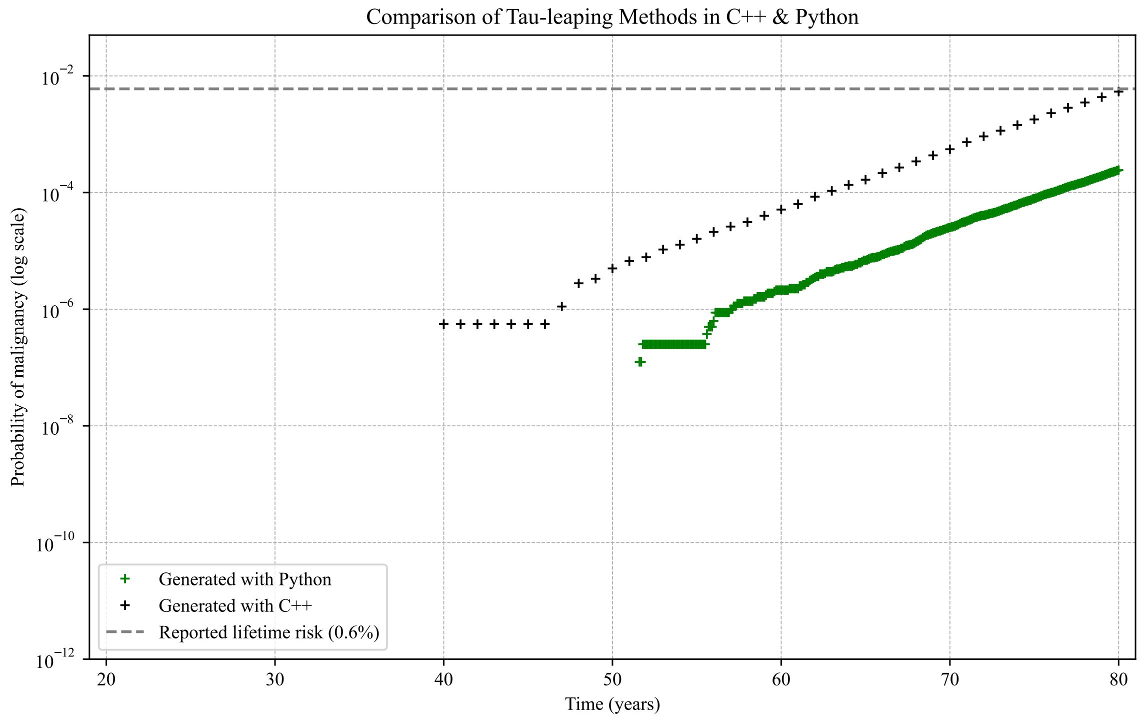


Figure 5. Different Simulation Approaches
C++ (paper)/ Python (our implementation)

Additionally, due to Python's interpreted nature and the complexity of handling large populations with numerous events, the run time of our implementation was exceedingly long. Therefore, we resorted to using the C++ implementation provided by the authors of the paper(Appendix B.1.1), as compiled languages and carefully optimized code can handle the computational load more efficiently. By doing so, we achieved simulations that more closely matched the paper's results and could be directly compared with analytically inferred probabilities and rate matrices, thereby ensuring both computational tractability and improved fidelity to the intended tau-leaping formulation.

## 3. Reproduction of Results

After performing tau-leaping simulations four times for both advantageous scenarios and two times for the *KRAS* neutral scenario, we used the equations 1, 2 and 3 outlined in Section 2.3 to reproduce the figure from the original paper, as shown in Figure 6. The analytical equations for both advantageous and *KRAS* neutral cases align well with the two tau-leaping simulation lines, and the resulting plot closely resembles Figure 2, which is the corresponding figure in the original paper. The plotting code we used is from Appendix B.2.

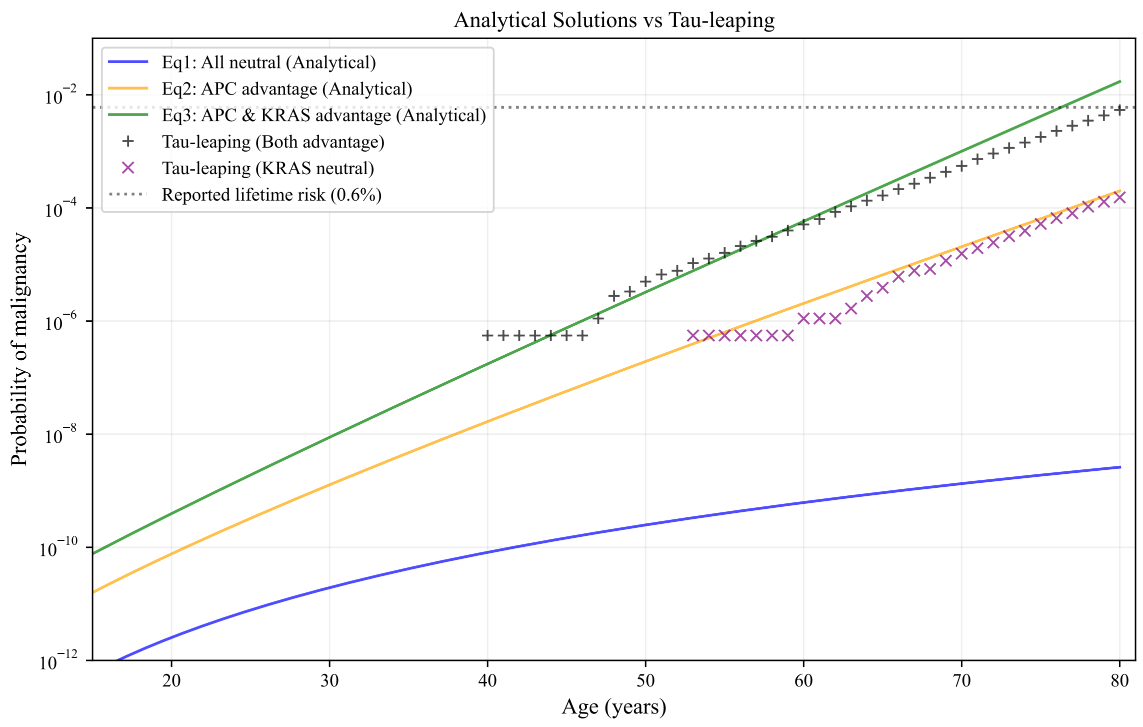


Figure 6. The analytical result with Tau-leaping

## 4. Novel Results

Building upon the biological framework established by Paterson et al., we develop a matrix-based approach to model colorectal cancer initiation, inspired by the *Matrix Models and Structured Population Dynamics* chapter in [4]. Paterson et al.'s work demonstrates that colonic crypts exhibit two fundamental behaviors: replication through fission and state transitions through genetic mutations. These characteristics naturally lend themselves to representation through a stage-based population model, where crypts can reproduce and transition between discrete genetic states. Our matrix formulation captures these dynamics through three key components: state vectors that track the population distribution across different genetic configurations, a transition matrix that governs the mutation processes between states, and a growth matrix that describes the selective proliferation of crypts with specific mutations. The setup of the linear model mainly involves three parts: The state variables that record the population compositions, the transition matrix that directs the mutation process, and the growth matrix that directs the cryptic expansion process.

4.1. **State Variables.** To model the progression of colorectal cancer, we define state variables as a combination of three key genes: *APC*, *TP53*, and *KRAS*. These state variables represent the mutational states of each gene and are denoted as a triplet $(x, y, z)$, following the convention in Section 2.1, where:

- $x \in \{0, 1, 2, 3\}$: Denotes the state of the *APC* gene.
  $x = 0$: The *APC* gene is fully functional ($APC^{+/+}$).
  $x = 1$: Loss of one *APC* allele ($APC^{+/}$).
  $x = 2$: Mutation in the remaining *APC* allele ($APC^{/+}$).
  $x = 3$: Complete functional loss of the *APC* gene ($APC^{/}$).
- $y \in \{0, 1, 2, 3\}$: Denotes the state of the *TP53* gene.
  $y = 0$: The *TP53* gene is fully functional.
  $y = 1$: Loss of one *TP53* allele (*TP53 LOH*).

$y = 2$: Mutation in the remaining *TP53* allele (*TP53 mutation*).
$y = 3$: Complete loss of *TP53* functionality (*TP53/*).

- $z \in \{0, 1\}$: Denotes the state of the *KRAS* gene.
  $z = 0$: The *KRAS* gene is in its wild-type form ($KRAS^{-}$).
  $z = 1$: The *KRAS* gene has undergone an activating mutation ($KRAS^{+}$).

Note that we only consider states 0,1,2, and 3 for *APC* and *TP53*. This is because, according to the original paper[1], losing both chromosomes for a given crypt is too unlikely to happen that the probability do not affect the calculation results. Therefore, the combination of these variables results in a total of $4 \times 4 \times 2 = 32$ distinct states. Each state is represented as a unique triplet $(x, y, z)$, corresponding to a specific mutation status for the three genes. For example, $(0, 0, 0)$ represents the initial state where all three genes are fully functional, $(0, 1, 0)$ indicates the loss of heterozygosity (LOH) in the *TP53* gene, and $(3, 3, 1)$ signifies the complete loss of *APC* and *TP53* with an activating mutation in *KRAS*. A complete listing of state variables can be found in the Appendix A for further reference. The state vector is initialized as:

$$\mathbf{v}_0 = \begin{bmatrix} 10^8 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix},$$

where $10^8$ crypts are assumed to start in the state $(0, 0, 0)$ in Section 2.1, and all other states have an initial cell population of 0. The state vector evolves over time according to a transition matrix, where each entry represents the mutation rates between different states, and the final probability of cancer can be extracted from specific states at later times.

4.2. **Transition Matrix.** The transition matrix $T$ is a key component in modeling the progression of mutations in colorectal cancer. It represents mutation rates between different states, capturing the dynamics of transitions between combinations of $APC$, $TP53$, and $KRAS$ gene mutations. Table 3 detailing the transition rules and their corresponding rates for mutations in the $APC$, $TP53$, and $KRAS$ genes.

| **Transition** | **Rate** |
|---|---|
| $APC^{+/+} \to APC^{+/}$ | $r_{\text{LOH}}$ |
| $APC^{+/+} \to APC^{/+}$ | $r_{\text{APC}}$ |
| $APC^{+/} \to APC^{/}$ | $r_{\text{APC}}/2$ |
| $APC^{/+} \to APC^{/}$ | $r_{\text{LOH}}/2$ |
| $TP53^{+/+} \to TP53^{+/}$ | $r_{\text{LOH}}$ |
| $TP53^{+/+} \to TP53^{/+}$ | $r_{\text{TP53}}$ |
| $TP53^{+/} \to TP53^{/}$ | $r_{\text{TP53}}/2$ |
| $TP53^{/+} \to TP53^{/}$ | $r_{\text{LOH}}/2$ |
| $KRAS^{-} \to KRAS^{+}$ | $r_{\text{KRAS}}$ |

Table 3. Rules for Transition Rates

The $32 \times 32$ transition matrix $T$ is constructed as follows:

- Each row corresponds to a **From State** $(x, y, z)$, and each column corresponds to a **To State** $(x', y', z')$.
- The entry $a_{ij}$ in the $i$-th row and $j$-th column represents the **rate of transition** from state $i$ (or $(x, y, z)$) to state $j$ (or $(x', y', z')$). For example:

$$T[i, j] = r$$

- $i$ is the index corresponding to the **From State** $(x, y, z)$. States are indexed sequentially based on their triplet representation (e.g., $(0, 0, 0)$ is index 1, $(0, 0, 1)$ is index 2, ..., $(3, 3, 1)$ is index 32).
- $j$ is the index corresponding to the **To State** $(x', y', z')$, following the same indexing as $i$.
- $r$ is the mutation rate that governs the probability of transitioning from state $i$ to state $j$. For example, $r_{\text{LOH}}$ represents the rate of loss of heterozygosity, $r_{\text{APC}}$ represents the mutation rate of $APC$, and so on.
- If there is no direct transition between $(x, y, z)$ and $(x', y', z')$, $T[i, j] = 0$.

We can illustrate this with a few simple examples. Suppose Transition 1 is from state $(0, 0, 0)$ to $(0, 1, 0)$, which corresponds to a loss of heterozygosity (LOH) in the *TP53* gene. The transition rate for this process is given by: $T[0, 2] = r_{\text{LOH}}$. Similarly, Transition 2 is from state $(0, 0, 0)$ to $(0, 2, 0)$, which represents a point mutation in the $TP53$ gene. The transition rate for this process is given by: $T[0, 4] = r_{\text{TP53}}$.

4.3. **Growth Rate Matrix.** The growth rate matrix represents the growth advantages conferred by mutations in specific genes. These growth rates account for the selective advantages of different mutational states in the progression of colorectal cancer. The growth rate matrix $G$ is a $32 \times 32$ diagonal matrix, defined as follows:

$$G = \begin{bmatrix} b_{1,1} & 0 & 0 & \cdots & 0 \\ 0 & b_{2,2} & 0 & \cdots & 0 \\ 0 & 0 & b_{3,3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & b_{32,32} \end{bmatrix},$$

where $b_{i,i}$ represents the growth rate for state $i$. As we discussed in Section 4.1, each state is represented as a triplet: $(APC, TP53, KRAS) \rightarrow$ state $(x, y, z)$, where $x, y \in \{0, 1, 2, 3\}$, $z \in \{0, 1\}$. The diagonal entries $b_{i,i}$ are determined by the following sources of growth rates:

- $b_{\text{APC}}$: Growth advantage when the $APC$ gene mutation leads to complete functional loss $(x \geq 3)$,
- $b_{\text{KRAS}}$: Growth advantage when the $KRAS$ gene undergoes an activating mutation $(z = 1)$,
- $b_{\text{BOTH}}$: Combined growth advantage when both $APC$ and $KRAS$ mutations occur simultaneously $(x \geq 3$ and $z = 1)$.

Therefore, the growth rates are assigned to the diagonal entries of $G$ based on the state variables:

$$b_{i,i} = \begin{cases} b_{\text{BOTH}}, & \text{if } x \geq 3 \text{ and } z = 1, \\ b_{\text{APC}}, & \text{if } x \geq 3 \text{ and } z = 0, \\ b_{\text{KRAS}}, & \text{if } x < 3 \text{ and } z = 1, \\ 0, & \text{otherwise.} \end{cases}$$

This matrix is used in conjunction with the transition matrix to simulate the evolution of cancer probabilities over time.

4.4. **Computing the Probability.** The Leslie-like matrix combines the transition matrix $T$ and the growth rate matrix $G$ to model the progression of mutations in colorectal cancer over time. This approach allows us to calculate the probability of malignancy at a specific age $N$. The final probability is computed using the following structure:

$$\mathbf{v}_N = (G \cdot T)^N \cdot \mathbf{v}_0,$$

where:

- $G$: The growth rate matrix (diagonal matrix).

- $T$: The transition matrix, representing mutation rates between states.
- $\mathbf{v}_0$: The initial state vector, where all $10^8$ crypts start in state $(0, 0, 0)$.
- $\mathbf{v}_N$: The state vector after $N$ years, containing probabilities for all 32 states.

The resulting state vector $\mathbf{v}_N$ after $N$ years contains the expected cell population for each of the 32 states. The probability of malignancy is the last entry in $\mathbf{v}_N$ corresponding to the cancerous state $(3, 3, 1)$, denoted as $\mathbf{v}_N[32]$.

## 5. Conclusion

5.1. **Matrix Approach.** Based on the new method introduced in Section 4, the matrix approach, combined with the tau-leaping simulation results, we were able to generate Figure 7 using the code provided in Appendix B.2. By comparing it with Figure 2, we observed that the matrix approach performs exceptionally well.

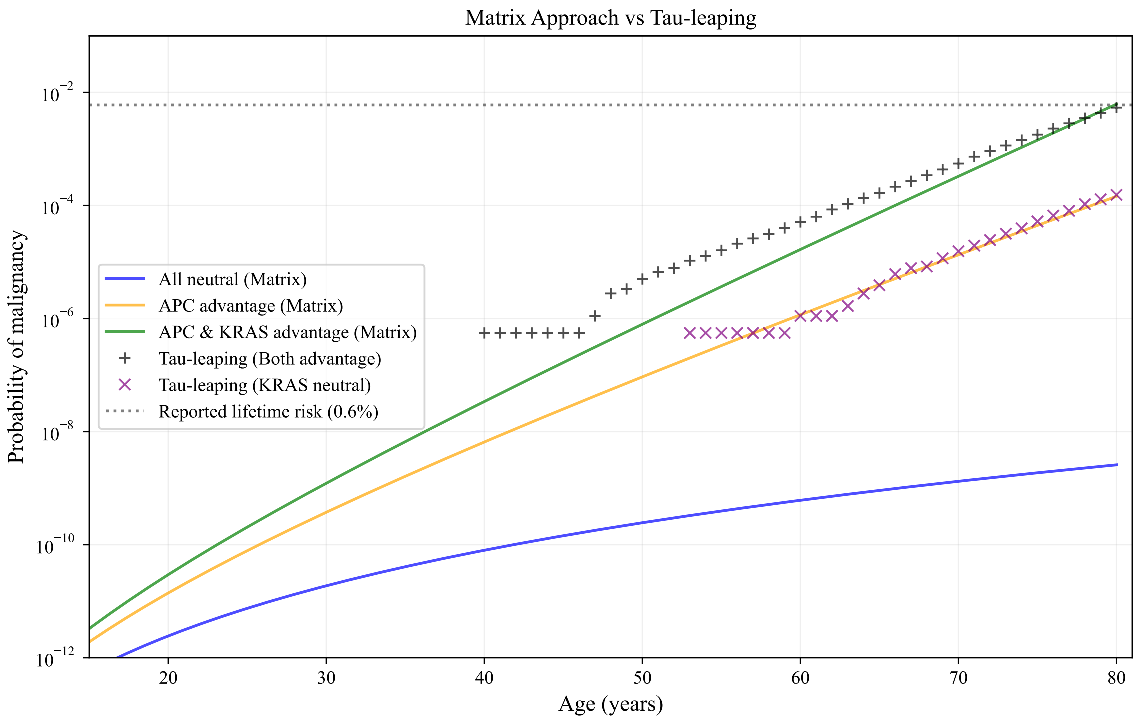


Figure 7. The Matrix Approach with Tau-leaping

5.2. **Comparison of two approaches.** To evaluate whether our matrix approach can truly replace the three equations introduced in Section 2.3, we plotted the probability curves from both the matrix and analytical methods together, as shown in Figure 8. Although the differences are very slight, they do exist. Therefore, we calculated the relative difference between the two methods, as illustrated in Figure 9.

5.3. **Error Analysis.** Based on the results shown in the figures, several key observations and potential explanations emerge regarding the differences between our matrix approach and the original analytical solutions.

The matrix approach consistently underestimates the probability of malignancy compared to the analytical solutions, as shown in Figure 8. This discrepancy is most pronounced in the "*APC* & *KRAS* advantage" case, where both mutations confer selective growth advantages. According to Paterson et al.'s supplementary materials[1], this difference may arise from their analytical approximations, which "assume that double mutations in tumor suppressor genes do not occur."

They note that this assumption is valid when "the rate of loss of heterozygosity (LOH) is much lower than the point mutation rate for each gene."

The relative difference plot (Figure 9) reveals intriguing patterns in the discrepancies between the methods. The difference is largest for the "*APC* & *KRAS* advantage" case and decreases over time in the "*APC* advantage only" scenario. This observation aligns with Paterson et al.'s claim that "the correction for double mutation in *APC* in our analytic approximation will be at most on the order of 2-3%." Our matrix method, which explicitly accounts for all possible transition pathways, including double mutations, demonstrates this effect.

Furthermore, Paterson et al. mention in their supplementary materials[1] that their analytical solutions neglect "outflow terms" in their system of equations. They state that "since the largest mutation rate, $r_{\mathrm{LOH}} = 1.36 \times 10^{-4}\,\mathrm{yr}^{-1}$, and the largest time $t = 80\,\mathrm{yr}$, it will always be the case that $r_{\mathrm{LOH}}t < 1.1 \times 10^{-2} \ll 1$." However, our matrix approach inherently includes these outflow terms, which could contribute to the observed differences.

The decreasing relative difference over time in the neutral case suggests that both methods converge in their predictions over longer time periods, particularly when no selective advantages are present. This supports the validity of both approaches while underscoring the importance of accounting for the full complexity of mutation pathways in cancer evolution modeling.

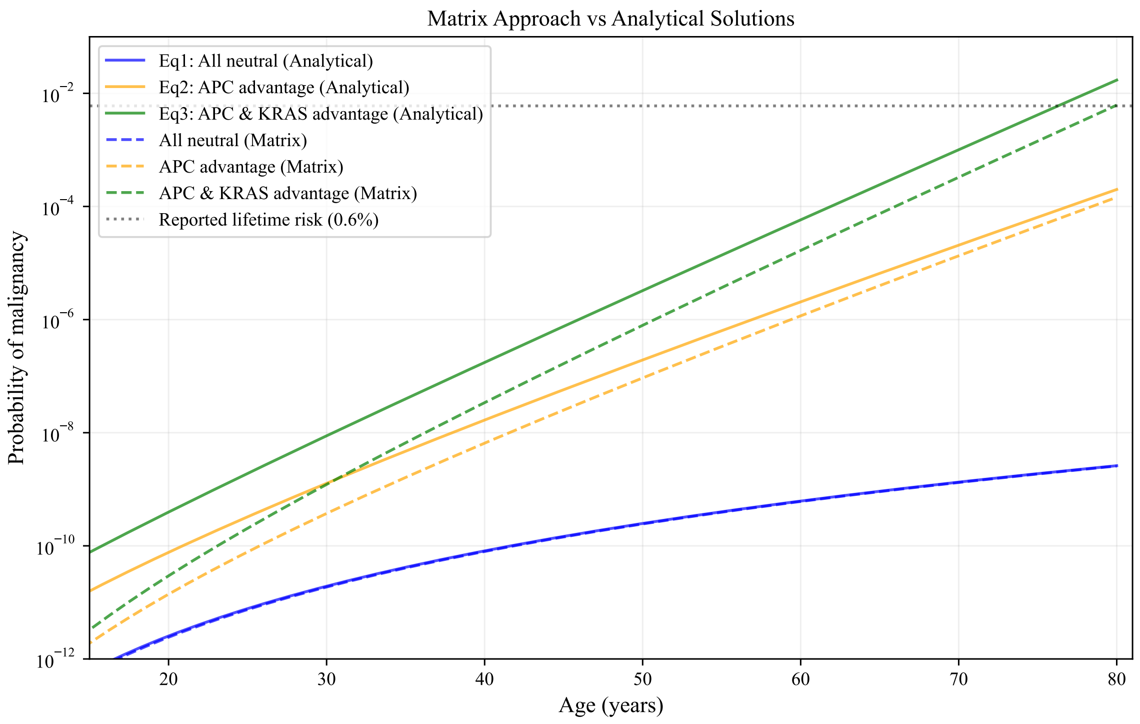


Figure 8. Matrix Approach vs Analytical Approach

## 5.4. **Conclusion.**

The original paper developed a stochastic mathematical model to describe the initiation of CRC, we reproduced the result and did an alternative matrix approach based on the Leslie Model to explain and predict CRC with observed epidemiological data. The original paper integrates mathematical modeling with experimental data to advance understanding of CRC initiation and provides a foundation for assessing therapeutic and diagnostic strategies. By comparing the results, as the method converges over a long time while it still exists underestimating, we want to emphasize the complicated of the cancer and underscore that people should treat the probability of cancer initiation seriously.

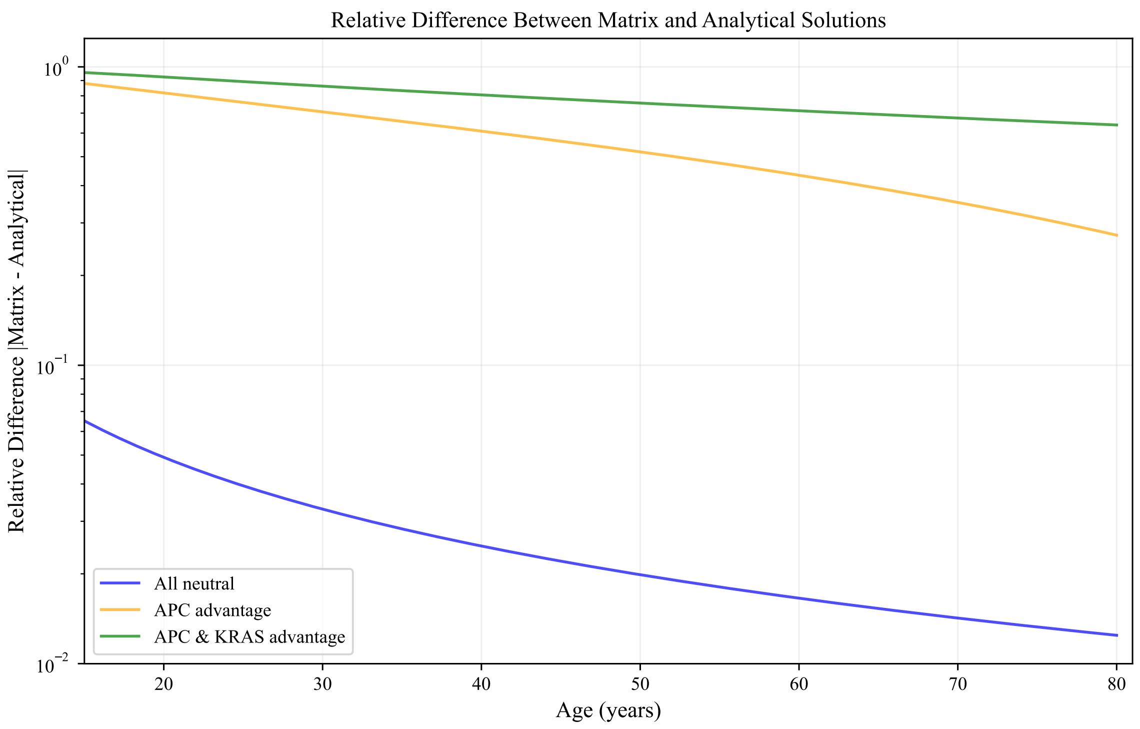


Figure 9. Relative Difference Between Matrix and Analytical Approach

The matrix approach we introduced in this paper provides several advantages. By encoding the complete state space and transition rules into matrix operations, we create a more streamlined computational framework that maintains the biological complexity of the system. Our comparative analysis shows that while the matrix method produces results that generally align with the original approach, there are notable differences that arise from the explicit inclusion of all possible transition pathways and the treatment of outflow terms.

5.5. **Future Work.** Looking forward, our matrix formulation opens up new avenues for analyzing cancer evolution using tools from linear algebra. Future work could explore the eigenvalue spectrum of the transition and growth matrices, potentially revealing characteristic timescales of cancer development and identifying critical pathways in the evolutionary process. The eigenvectors of these matrices might provide insights into the stable distributions of genetic states and the most likely paths to malignancy. Additionally, matrix perturbation theory could be applied to understand how small changes in mutation rates or growth advantages affect the overall system dynamics, with potential implications for therapeutic interventions. Furthermore, this matrix-based approach could be extended to model other types of cancer or more complex genetic networks. The framework's flexibility allows for the incorporation of additional mutations, different types of genetic alterations, or more sophisticated growth dynamics while maintaining computational traceability.

## Acknowledgments

We extend our gratitude to Prof. *Eric Shea-Brown* for invaluable guidance and tireless feedback on our project. We also thank TA *Rohin Gilman* for introducing this paper to us and consistently providing constructive suggestions for improvement. Finally, we are grateful to the Department of Applied Mathematics IT team for providing access to the AMATH servers, which made it possible for us to run Python's tau-leaping simulations effectively.

## References

[1] Chay Paterson, Hans Clevers, and Ivana Bozic. "Mathematical model of colorectal cancer initiation". In: *Proceedings of the National Academy of Sciences* 117.34 (2020), pp. 20681–20688. DOI: 10.1073/pnas.2003771117. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.2003771117. URL: https://www.pnas.org/doi/abs/10.1073/pnas.2003771117.

[2] Cristian Tomasetti et al. "Only three driver gene mutations are required for the development of lung and colorectal cancers". In: *Proceedings of the National Academy of Sciences* 112.1 (2015), pp. 118–123. DOI: 10.1073/pnas.1421839112. URL: https://doi.org/10.1073/pnas.1421839112.

[3] E. Georg Luebeck and Suresh H. Moolgavkar. "Multistage carcinogenesis and the incidence of colorectal cancer". In: *Proceedings of the National Academy of Sciences* 99.23 (2002), pp. 15095–15100. DOI: 10.1073/pnas.222118199. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.222118199. URL: https://www.pnas.org/doi/abs/10.1073/pnas.222118199.

[4] Stephen P. Ellner and John Guckenheimer. *Dynamic Models in Biology*. Princeton University Press, 2006. ISBN: 9780691125893. URL: http://www.jstor.org/stable/j.ctvcm4h1q (visited on 12/10/2024).

[5] Daniel T. Gillespie. "Exact Stochastic Simulation of Coupled Chemical Reactions". In: *The Journal of Physical Chemistry* 81.25 (1977), pp. 2340–2361. DOI: 10.1021/j100540a008. URL: https://doi.org/10.1021/j100540a008.

[6] Daniel T. Gillespie and Linda R. Petzold. "Improved Leap-Size Selection for Accelerated Stochastic Simulation". In: *The Journal of Chemical Physics* 119.16 (2003), pp. 8229–8234. DOI: 10.1063/1.1613254. URL: https://doi.org/10.1063/1.1613254.

[7] A. M. Baker et al. "Quantification of Crypt and Stem Cell Evolution in the Normal and Neoplastic Human Colon". In: *Cell Reports* 8.4 (2014), pp. 940–947. DOI: 10.1016/j.celrep.2014.07.019. URL: https://doi.org/10.1016/j.celrep.2014.07.019.

[8] Franziska Michor, Yoh Iwasa, and Martin A. Nowak. "Dynamics of Cancer Progression". In: *Nature Reviews Cancer* 4.3 (2004), pp. 197–205. DOI: 10.1038/nrc1295. URL: https://doi.org/10.1038/nrc1295.

## Appendix A. State Variables

| State | Description |
|---|---|
| (0, 0, 0) | Normal crypt, no mutations. |
| (0, 0, 1) | KRAS activated, others normal. |
| (0, 1, 0) | TP53 lost one allele, others normal. |
| (0, 1, 1) | TP53 lost one allele + KRAS activated. |
| (0, 2, 0) | TP53 mutated one allele, others normal. |
| (0, 2, 1) | TP53 mutated one allele + KRAS activated. |
| (0, 3, 0) | TP53 mutated+lost, others normal. |
| (0, 3, 1) | TP53 mutated+lost + KRAS activated. |
| (1, 0, 0) | APC lost one allele, others normal. |
| (1, 0, 1) | APC lost one allele + KRAS activated. |
| (1, 1, 0) | APC lost one allele + TP53 lost one allele. |
| (1, 1, 1) | APC lost one allele + TP53 lost one allele + KRAS activated. |
| (1, 2, 0) | APC lost one allele + TP53 mutated one allele. |
| (1, 2, 1) | APC lost one allele + TP53 mutated one allele + KRAS activated. |
| (1, 3, 0) | APC lost one allele + TP53 mutated+lost. |
| (1, 3, 1) | APC lost one allele + TP53 mutated+lost + KRAS activated. |
| (2, 0, 0) | APC mutated one allele, others normal. |
| (2, 0, 1) | APC mutated one allele + KRAS activated. |
| (2, 1, 0) | APC mutated one allele + TP53 lost one allele. |
| (2, 1, 1) | APC mutated one allele + TP53 lost one allele + KRAS activated. |
| (2, 2, 0) | APC mutated one allele + TP53 mutated one allele. |
| (2, 2, 1) | APC mutated one allele + TP53 mutated one allele + KRAS activated. |
| (2, 3, 0) | APC mutated one allele + TP53 mutated+lost. |
| (2, 3, 1) | APC mutated one allele + TP53 mutated+lost + KRAS activated. |
| (3, 0, 0) | APC mutated+lost, others normal. |
| (3, 0, 1) | APC mutated+lost + KRAS activated. |
| (3, 1, 0) | APC mutated+lost + TP53 lost one allele. |
| (3, 1, 1) | APC mutated+lost + TP53 lost one allele + KRAS activated. |
| (3, 2, 0) | APC mutated+lost + TP53 mutated one allele. |
| (3, 2, 1) | APC mutated+lost + TP53 mutated one allele + KRAS activated. |
| (3, 3, 0) | APC mutated+lost + TP53 mutated+lost. |
| (3, 3, 1) | APC mutated+lost + TP53 mutated+lost + KRAS activated (Malignant state). |

## Appendix B. Main Code

### B.1. tau-leaping simulations.

#### B.1.1. *Code for tau-leaping simulations in C++.*

The C++ code used for tau-leaping simulations was provided by the original paper and can be found in the Data Availability section on the website.[1].

#### B.1.2. *Code for tau-leaping simulations in Python.*

```python
import numpy as np
import matplotlib.pyplot as plt
from dataclasses import dataclass
from typing import Dict, Set, Tuple
import random

@dataclass
class SimParams:
    # Base mutation rate per base pair per year
    u: float = 1.25e-8

    # Number of driver positions in each gene
    n_APC: int = 604     # Number of APC driver positions
    n_TP53: int = 73     # Number of TP53 driver positions
    n_KRAS: int = 20     # Number of KRAS driver positions

    # Derived mutation rates per year
    r_APC: float = 604 * 1.25e-8    # APC mutation rate
    r_TP53: float = 73 * 1.25e-8    # TP53 mutation rate
    r_KRAS: float = 20 * 1.25e-8    # KRAS mutation rate
    r_LOH: float = 1.36e-4          # Rate of loss of heterozygosity

    # Division rates per year
    b_APC: float = 0.2     # Division rate for APC-/- crypts
    b_KRAS: float = 0.07   # Division rate for KRAS+ crypts
    b_BOTH: float = 0.27   # Division rate for APC-/-,KRAS+ crypts
                           # (b_APC + b_KRAS)

    # Time parameters
    dt: float = 0.01       # Time step size (tau)
    t_max: float = 80.0    # Maximum simulation time

    # Initial conditions
    N_crypts: int = 10**8  # Number of initial crypts

class State:
    """Represents the genetic state of a crypt"""
    def __init__(self, APC: int, TP53: int, KRAS: int):
        # APC states:
        # 0: Wild type
        # 1: First hit LOH
        # 2: First hit mutation
        # 3: Second hit after LOH or mutation
        # 4: Double mutation
        self.APC = APC

        # TP53 states: similar to APC
        self.TP53 = TP53

        # KRAS states:
        # 0: Wild type
        # 1: Activated
        self.KRAS = KRAS

    def __hash__(self):
        return hash((self.APC, self.TP53, self.KRAS))

    def __eq__(self, other):
        return (self.APC, self.TP53, self.KRAS) ==
               (other.APC, other.TP53, other.KRAS)

    def is_malignant(self):
        """Check if state represents a malignant crypt"""
        # Malignant states: (3,3,1), (3,4,1), (4,3,1), (4,4,1)
        return ((self.APC >= 3) and (self.TP53 >= 3) and (self.KRAS == 1))

def get_transition_rate(source: State, target: State, params: SimParams)
    -> float:
    """Calculate transition rate between states based on Suppl. Mater."""
    # APC transitions
    if target.APC > source.APC:
        if source.APC == 0:
            # First hit can be LOH or mutation
            return params.r_LOH if target.APC == 1 else params.r_APC
        elif source.APC == 1:
            # After LOH, only mutation possible
            return params.r_APC / 2
        elif source.APC == 2:
            # After mutation, both LOH and mutation possible
            return params.r_LOH / 2 if target.APC == 3 else params.r_APC/2

    # TP53 transitions (similar to APC)
    if target.TP53 > source.TP53:
        if source.TP53 == 0:
            return params.r_LOH if target.TP53 == 1 else params.r_TP53
        elif source.TP53 == 1:
            return params.r_TP53 / 2
        elif source.TP53 == 2:
            return params.r_LOH / 2 if target.TP53==3 else params.r_TP53/2

    # KRAS activation
    if target.KRAS > source.KRAS:
        return params.r_KRAS

    return 0.0

def get_division_rate(state: State, params: SimParams) -> float:
    """Get division rate for a given state based on Suppl. Mater."""
    if state.APC >= 3:       # APC-/-
```

```
        if state.KRAS == 1:  # Also KRAS+
            return params.b_BOTH
        return params.b_APC
    elif state.KRAS == 1:    # Only KRAS+
        return params.b_KRAS
    return 0.0

def get_neighbors(state: State) -> Set[State]:
    """Get all possible next states from current state"""
    neighbors = set()

    # APC transitions
    if state.APC == 0:
        neighbors.add(State(1, state.TP53, state.KRAS)) # LOH
        neighbors.add(State(2, state.TP53, state.KRAS)) # Mutation
    elif state.APC == 1:
        neighbors.add(State(3,state.TP53,state.KRAS)) # Mutation after LOH
    elif state.APC == 2:
        neighbors.add(State(3, state.TP53, state.KRAS)) # LOH after
           mutation
        neighbors.add(State(4, state.TP53, state.KRAS)) # Second mutation

    # TP53 transitions
    if state.TP53 == 0:
        neighbors.add(State(state.APC, 1, state.KRAS))  # LOH
        neighbors.add(State(state.APC, 2, state.KRAS))  # Mutation
    elif state.TP53 == 1:
        neighbors.add(State(state.APC,3,state.KRAS))  # Mutation after LOH
    elif state.TP53 == 2:
        neighbors.add(State(state.APC, 3, state.KRAS))  # LOH after
           mutation
        neighbors.add(State(state.APC, 4, state.KRAS))  # Second mutation

    # KRAS activation
    if state.KRAS == 0:
        neighbors.add(State(state.APC, state.TP53, 1))

    return neighbors

def tau_leaping_simulation(params: SimParams, n_runs: int = 1000) -> np.
   ndarray:
    """Run multiple tau-leaping simulations following Suppl. Mater."""
    time_points = np.arange(0, params.t_max + params.dt, params.dt)
    malignant_counts = np.zeros(len(time_points))

    for run in range(n_runs):
        if run % 100 == 0:
            print(f"Run {run}/{n_runs}")

        # Initialize population dictionary
        population = {State(0,0,0): params.N_crypts}
        had_malignant = False

        for t_idx, t in enumerate(time_points):
```

```python
            if had_malignant:
                malignant_counts[t_idx:] += 1
                break

            # Process each populated state
            new_events = {}
            for state, count in list(population.items()):
                if count == 0:
                    continue

                #Check for division events(negative binomial distribution)
                division_rate = get_division_rate(state, params)
                if division_rate > 0:
                    p = np.exp(-division_rate * params.dt)
                    n_divisions = np.random.negative_binomial(count, p)
                    if n_divisions > 0:
                        new_events[state] = new_events.get(state, 0) + n_divisions

                # Check for transitions to neighbor states
                for neighbor in get_neighbors(state):
                    rate = get_transition_rate(state, neighbor, params)
                    if rate > 0:
                        # Use Poisson approximation for small rates
                        if rate * params.dt < 0.01:
                            n_transitions = np.random.poisson(rate * params.dt * count)
                        else:
                            # Use binomial for larger rates
                            p = 1 - np.exp(-rate * params.dt)
                            n_transitions = np.random.binomial(count, p)

                        if n_transitions > 0:
                            new_events[state] = new_events.get(state, 0) - n_transitions
                            new_events[neighbor] = new_events.get(neighbor, 0) + n_transitions

                            if neighbor.is_malignant():
                                had_malignant = True
                                break

            # Update population
            for state, delta in new_events.items():
                population[state] = population.get(state, 0) + delta
                if population[state] < 0:  # Sanity check
                    population[state] = 0

            if had_malignant:
                malignant_counts[t_idx:] += 1
                break

    return malignant_counts / n_runs

```

```python
def plot_results(time_points: np.ndarray, probabilities: np.ndarray):
    """Plot the probability of malignancy over time"""
    plt.figure(figsize=(10, 6))
    plt.semilogy(time_points, probabilities, 'b-', label='Tau-leaping
        simulation')
    plt.xlabel('Age (years)')
    plt.ylabel('Probability of malignancy')
    plt.title('Probability of Colorectal Cancer Development')
    plt.grid(True)
    plt.legend()
    plt.show()

from tqdm import tqdm

def tau_leaping_simulation_with_progress(params: SimParams, n_runs: int =
   1000, checkpoints: list = None) -> np.ndarray:
    """Run multiple tau-leaping simulations with intermediate
    probability output and a progress bar"""
    time_points = np.arange(0, params.t_max + params.dt, params.dt)
    malignant_counts = np.zeros(len(time_points))

    if checkpoints is None:
        checkpoints = []  # Default: no checkpoints

    # Initialize tqdm progress bar
    with tqdm(total=n_runs, desc="Simulation Progress") as pbar:
        for run in range(n_runs):
            # Update progress bar
            pbar.update(1)

            # Checkpoints output
            if (run + 1) in checkpoints:
                probabilities = malignant_counts / (run + 1)
                print(f"Checkpoint {run + 1}/{n_runs}:")
                print(f"Probabilities: {probabilities}")

            # Initialize population dictionary
            population = {State(0, 0, 0): params.N_crypts}
            had_malignant = False

            for t_idx, t in enumerate(time_points):
                if had_malignant:
                    malignant_counts[t_idx:] += 1
                    break

                # Process each populated state
                new_events = {}
                for state, count in list(population.items()):
                    if count == 0:
                        continue

                    # Check for division events(negative binomial distri.)
                    division_rate = get_division_rate(state, params)
                    if division_rate > 0:
                        p = np.exp(-division_rate * params.dt)
                        n_divisions = np.random.negative_binomial(count, p)
                        if n_divisions > 0:
                            new_events[state] = new_events.get(state, 0) + n_divisions

                    # Check for transitions to neighbor states
                    for neighbor in get_neighbors(state):
                        rate = get_transition_rate(state,neighbor,params)
                        if rate > 0:
                            # Use Poisson approximation for small rates
                            if rate * params.dt < 0.01:
                                n_transitions = np.random.poisson(rate * params.dt * count)
                            else:
                                # Use binomial for larger rates
                                p = 1 - np.exp(-rate * params.dt)
                                n_transitions=np.random.binomial(count,p)

                            if n_transitions > 0:
                                new_events[state] = new_events.get(state, 0) - n_transitions
                                new_events[neighbor] = new_events.get(neighbor, 0) + n_transitions

                                if neighbor.is_malignant():
                                    had_malignant = True
                                    break

                # Update population
                for state, delta in new_events.items():
                    population[state] = population.get(state, 0) + delta
                    if population[state] < 0:  # Sanity check
                        population[state] = 0

                if had_malignant:
                    malignant_counts[t_idx:] += 1
                    break

    return malignant_counts / n_runs

# Run the simulation with a progress bar
params = SimParams(
    dt=0.1,           # Time step size
    t_max=80.0,       # Maximum time
    N_crypts=10**8    # Number of initial crypts
)

n_runs = 8000000
checkpoints = [500000, 1000000, 1500000, 2000000, 2500000, 3000000,
               3500000, 4000000, 4500000, 5000000, 5500000, 6000000,
               6500000, 7000000, 7500000, 8000000]

```

```
time_points = np.arange(0, params.t_max + params.dt, params.dt)
probabilities = tau_leaping_simulation_with_progress(params, n_runs=n_runs
    , checkpoints=checkpoints)

# Plot the results
plot_results(time_points, probabilities)
```

B.2. **Code for reproducing plots.**

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
np.set_printoptions(threshold=np.inf)
plt.rc('font', family='Times New Roman')
# Define parameters
class Parameters:
    def __init__(self):
        # Base mutation rate per base pair per year
        self.u = 1.25e-8

        # Number of driver positions in each gene
        self.n_APC = 604
        self.n_TP53 = 73
        self.n_KRAS = 20

        # Mutation rates per year
        self.r_APC = self.n_APC * self.u
        self.r_TP53 = self.n_TP53 * self.u
        self.r_KRAS = self.n_KRAS * self.u
        self.r_LOH = 1.36e-4

        # Growth rates per year
        self.b_APC = 0.2    # APC-/- growth rate
        self.b_KRAS = 0.07  # KRAS+ growth rate
        self.b_BOTH = 0.27  # Combined APC-/-,KRAS+ growth rate

        # Initial number of crypts
        self.N_crypts = 10**8

        # Correction factors
        self.c1 = 5.88  # APC fixation advantage
        self.c2 = 3.6   # KRAS fixation advantage
        self.c = self.c1 * self.c2  # Combined correction
```

B.2.1. *Matrix Approach.*

```
# Matrix Method
def create_transition_matrix(params):
    """Create 32x32 transition rate matrix"""
    n = 32
    T = np.zeros((n, n))

    def state_to_idx(apc, tp53, kras):
        return apc * 8 + tp53 * 2 + kras

    # APC transitions
    for tp53 in range(4):
        for kras in range(2):
            # APC: (0) -> (1) or (2)
            T[state_to_idx(0, tp53, kras), state_to_idx(1, tp53, kras)] =
                params.r_LOH
            T[state_to_idx(0, tp53, kras), state_to_idx(2, tp53, kras)] =
                params.r_APC

            # APC: (1) -> (3)
            T[state_to_idx(1, tp53, kras), state_to_idx(3, tp53, kras)] =
                params.r_APC/2

            # APC: (2) -> (3)
            T[state_to_idx(2, tp53, kras), state_to_idx(3, tp53, kras)] =
                params.r_LOH/2

    # TP53 transitions
    for apc in range(4):
        for kras in range(2):
            # TP53: (0) -> (1) or (2)
            T[state_to_idx(apc, 0, kras), state_to_idx(apc, 1, kras)] =
                params.r_LOH
            T[state_to_idx(apc, 0, kras), state_to_idx(apc, 2, kras)] =
                params.r_TP53

            # TP53: (1) -> (3)
            T[state_to_idx(apc, 1, kras), state_to_idx(apc, 3, kras)] =
                params.r_TP53/2

            # TP53: (2) -> (3)
            T[state_to_idx(apc, 2, kras), state_to_idx(apc, 3, kras)] =
                params.r_LOH/2

    # KRAS transitions
    for apc in range(4):
        for tp53 in range(4):
            # KRAS: (0) -> (1)
            T[state_to_idx(apc, tp53, 0), state_to_idx(apc, tp53, 1)] =
                params.r_KRAS

    return T

def create_growth_matrix(params):
    """Create 32x32 diagonal growth rate matrix"""
    n = 32
    G = np.zeros((n, n))

    def state_to_idx(apc, tp53, kras):
        return apc * 8 + tp53 * 2 + kras

    # Fill diagonal with growth rates
    for apc in range(4):
        for tp53 in range(4):
            for kras in range(2):
                idx = state_to_idx(apc, tp53, kras)

                if apc >= 3:  # APC-/- states
                    if kras == 1:  # APC-/- + KRAS+
                        G[idx, idx] = params.b_BOTH
                    else:  # Only APC-/-
                        G[idx, idx] = params.b_APC
                elif kras == 1:  # KRAS+ only
                    G[idx, idx] = params.b_KRAS

    return G

def simulate_evolution(params, t_max=80, dt=0.1):
    """Simulate the evolution of crypts using matrix approach"""
    # Create matrices
    T = create_transition_matrix(params)
    G = create_growth_matrix(params)

    # # Output the transition matrix T and growth matrix G
    # print("Transition Matrix (T):")
    # print(T)
    # print("\nGrowth Matrix (G):")
    # print(G)

    # Time points
    times = np.arange(0, t_max + dt, dt)

    # Initialize state vector (all crypts in state 000)
    v = np.zeros(32)
    v[0] = params.N_crypts

    # Store results
    results = np.zeros(len(times))
    malignant_idx = 3*8 + 3*2 + 1  # Index for state (3,3,1)

    # Evolution
    for i, t in enumerate(times):
        # Store malignant probability
        # results[i] = v[malignant_idx] / params.N_crypts
        results[i] = v[malignant_idx]

        # Update state vector
        v = v + (np.matmul(v, T) + np.matmul(v, G)) * dt

    return times, results
```

B.2.2. *Equation Approach.*

```
# equation method
def neutral_solution(t, params):
    """Equation (1): All mutations are neutral"""
    P = (params.N_crypts * params.r_APC * params.r_TP53 *
```

```python
            params.r_KRAS * params.r_LOH**2 * t**5) / 4
    return P

def APC_advantage_solution(t, params):
    """Equation (2): Only APC provides growth advantage"""
    P = (3 * params.N_crypts * params.r_APC * params.r_TP53 *
            params.r_KRAS * params.r_LOH**2 * t**2 *
            np.exp(params.b_APC * t)) / (2 * params.b_APC**3)
    return P * params.c1

def both_advantage_solution(t, params):
    """Equation (3): Both APC and KRAS provide advantage"""
    b12 = params.b_BOTH
    b1 = params.b_APC
    b2 = params.b_KRAS

    P = (params.N_crypts * params.r_APC * params.r_TP53 *
            params.r_KRAS * params.r_LOH**2 * t * np.exp(b12 * t) *
            (1/(b12**3 * (b12 - b1)) + 1/(b12**3 * (b12 - b2)) +
             1/(b12**2 * (b12 - b2)**2)))
    return P * params.c
```

B.2.3. *Tau-leaping.*

```python
# Consider tau-leaping data
data1 = pd.read_csv('test1.1.csv', header=None)
data2 = pd.read_csv('test2.1.csv', header=None)

def process_csv(file_name):
    """Read and process a CSV file to calculate cancer probabilities."""
    df = file_name
    years = df[0]
    state_331 = df[38]  # (3,3,1) state
    state_341 = df[40]  # (3,4,1) state
    state_431 = df[48]  # (4,3,1) state
    state_441 = df[50]  # (4,4,1) state
    cancer_prob = state_331 + state_341 + state_431 + state_441
    print(cancer_prob)
    return years, cancer_prob
```