Continuous Treatment on Invasive Ductal Carcinoma

Óscar J. Escobar, Joseph Humpherys, Henry Fetzer, Clifton Langley

Abstract

Breast cancer is the most common cancer in the US. Most breast cancers are specifically located in the milk ducts [Ins24]. Treatments are based on decision tree algorithms that can consider tumor size, degree of spread, and other patient information. Applying optimal control theory to the treatment, based on dynamical cancer models, provides the benefit of *in silico* analysis of various treatments while also providing a personalized treatment to the patient. Our research creates a model for breast cancer that incorporates immunological responses and cell competition and then applies optimal control theory to derive an optimal continuous treatment. We show that an optimal control can be found, but our results are bang-bang solutions that may not be physically safe. However, there is hope that by better incorporating dosing constraints, the optimal control may become more nuanced but our initial work appears to be aligned with current theoritical results.

1 Motivation & Background

Mathematical Oncology (MO) is described as a "a bridge between . . . the biologist and practicing clinician " [SR19]. Tumor modeling is an area of MO whose objective is to create and understand models that can help clinicians understand properties that govern tumor growth, the effect of treatments, and other factors like drug toxicity or interactions between the immune system and the cancerous tumor.

Most therapeutic regimes are administered by decision-tree algorithms that incorporate current tumor size, how much the tumor has spread, and other patient information [ADE+20]. Although these are good, the dosing of

the administered regime is not necessarily optimized for individual patients since it is difficult to estimate the efficacy of the treatment distribution and the resulting drug toxicity. There are models for better determining doses as well as models that can help analyze toxicity risk, but these are limited in number and not commonly used. Moreover, depending on the severity of the cancer, there may not be time to use these algorithms to find a good regime. In these cases, a quicker approach is to choose a particular method and test it *in silico* (see Appendix A for a definition).

Optimal control theory (OTC) seeks to find the best possible control strategy for a dynamical system by optimizing a performance criterion while following the constraints and modeling the dynamics of the system. [ADE+20]. OTC, applied to cancer, can help us derive controls on different therapeutic regimes, such as chemotherapy, which can be specified to fit certain constraints, such as dosing limits. Moreover, we can also apply OTC to specific individuals given an initial estimate of the cancer cells in their body. This would help us see *in-silico* how the tumor would respond to certain dosages or other treatments. Thus, we would not have to wait to see how a patient might respond to a specific treatment. The OTC control and model can help us predict how the tumor will respond to the treatment.

The topic of using OTC to find optimal cancer treatments is an existing field of study. We have both seen it used for in-class examples this semester, as well as in research journals. For example, in [SNK23], authors build on existing research of chemotherapy OTC by using a new chemotherapy function and applying it to the known form of chemotherapy OTC. They state, "[it] is has been proven that the optimal protocol has a piecewise constant structure with no more than one switch". Thus, finding a model that could mimic this piecewise form would essentially be promising results in our research.

In [ABA+24], the focus was to derive an optimal control that incorporates immunotherapy and chemotherapy for any given tumor that grows logistically. They applied numerous OTC techniques including Pontryagin's Maximum Principle (PMP), interior point, and the approximate sequence Riccatti equation. Their work was particularly enlightening for how to make a tumor model that also considered Natural Killer (NK) cells and Cytotoxic T-Lymphocytes ($CD8^+$) interactions. However, they did not incorporate any competition terms between the tumor and normal healthy cells (excluding immune cells). On the other hand, in [GKGK21], the authors focused on analyzing competition terms between cancer cells and normal healthy cells

in cases of blood cancer. They were particularly interested in a differential equation that describes the change in the concentration of the chemo-drug being administered in a Lokta-Volterra style competition model.

Our research is similar to the aforementioned papers because we use OTC to derive an optimal therapeutic regime to minimize tumor size. However, we also incorporate competition interactions and focus on the optimal control for a specific cancer treatment, the Adriamycin-Cytoxan treatment. We also build on the existing research of optimal control theory applied to chemotherapy, but with some lesser-explored modeling choices and assumptions, stated below.

2 The Problem

The American Cancer Society estimates that about 310,000 new breast cancer diagnoses will be made in 2025 alone. Of these, 80% will be invasive ductal carcinoma (IDC; see A). For patients with IDC, the most common treatment in the early stages is the Adriamycin-Cytoxan (AC) treatment, which is comprised of both doxorubicin and cyclophosphamide (see A). Seeing that IDC is quite common and has a well-defined chemo regime, we decided to focus on IDC.

Both doxorubicin and cyclophosphamide are commonly administered in $\frac{mg}{m^2}$, which is based on the patient's body surface area (BSA). The use of body surface area for chemotherapeutic drugs was initially proposed by Donald Pinkel in 1958 [Pin58]. Pinkel would later help pioneer the first "Total Treatment" cancer treatment, which achieved a 50% success rate in 1968. Since BSA is still one of the most prevalent ways of dosing chemotherapeutic drugs, we use it to determine the bounds on our doses.

2.1 Modeling

This research is a continuation of our breast cancer model from last semester. In particular, we continued to describe the growth of IDC with a Gompertz ODE term and follow NK and $CD8^+$ cells. But, as we learned in our literature review and from last semester, we also needed to model the population of normal healthy cells that compete for space and nutrients with cancerous cells. Although there is evidence that tumor cells kill normal cells, we opted

to model competition between them like DePillis so that, in a sense, normal epithelial cells also kill off IDC in the struggle for resources and nutrients [LR03].

Furthermore, we know that NK and $CD8^+$ cells are more active than other immune cells in fighting breast cancer, so we opted to only model the populations of these immune cells [ABZ21]. We consider all the interactions between NK, $CD8^+$, tumor, and normal cells in our model. Our work is similar to this [GKGK21] since we model the competition between normal cells and cancerous cells. We also added the Michaelis-Menten kinetics (given by nonlinear growth terms) expressions that allow us to incorporate immune cell recruitment due to tumor lysis and debris as well as the death due to interactions. Since IDC starts off localized, we decided to consider both the natural death of immune cells as well as some constant rate at which these flow into the region around the milk ducts. Lastly, we also added in death rates that either come naturally or through interaction with other cells.

Thus, define the state vector as $\mathbf{x}(t) = \begin{bmatrix} T(t) & N(t) & CD(t) & NK(t) \end{bmatrix}^\intercal$ where $T(t)$ is the IDC burden at time t, measured in days, $N(t)$ is the cell count of normal epithelial cells, $CD(t)$ is the $CD8^+$ cell count, and $NK(t)$ is the count of NK cells. Our state elements evolve according to

$$\frac{\mathrm{d}T(t)}{\mathrm{d}t} = g_T T \ln\Big(\frac{T_{\max}}{T}\Big) - a_1 NT - a_2 NKT - a_3 CDT - \Big(\frac{4}{5}E_c D_c + E_d D_d\Big)T, \tag{1}$$

$$\frac{\mathrm{d}N(t)}{\mathrm{d}t} = g_N N \ln\Big(\frac{N_{\max}}{N}\Big) - k_N N - a_0 NT, \tag{2}$$

$$\frac{\mathrm{d}CD(t)}{\mathrm{d}t} = r_{CD} - k_{CD} CD + \frac{\rho_0 CDT^i}{\alpha_0 + T^i} - a_4 CDT - b_{CD} D_c CD, \tag{3}$$

$$\frac{\mathrm{d}NK(t)}{\mathrm{d}t} = r_{NK} - k_{NK} NK + \frac{\rho_1 NKT^j}{\alpha_1 + T^j} - a_5 NKT \tag{4}$$

We set our inital state to $\mathbf{x}(0) = \begin{bmatrix} 1 \times 10^6 & 1 \times 10^7 & 1.21 \times 10^3 & 1.185 \times 10^3 \end{bmatrix}^\intercal$.

Our assumption was that epithelial cells would be around 96% of their carrying capacity and the tumor would comprise about 10% of the total epithelial cell population.

2.2 The Control

Cancer is very unique to each person and so are the doses of chemotherapy drugs administered to the patient. Thus, to work on a general case of breast cancer would be quite difficult. Instead, we assume that we our patient is a woman of age 65 whose is 161.8 cm tall and weighs 61.89 kg. The age we chose is within the range when breast cancer is most likely to develop according to the American Cancer Society. The height and weight are from looking at statistics of the average weight and height for a 65 year old woman.

Normally, the AC treatment is administered for a total of 4-6 doses spaced every 2-3 weeks. However, the dosage concentration, as mentioned earlier, can depend on various factors. One of the commonly used metrics to determine the dose is *body surface area*. The formula we used to calculate BSA in this project is Mosteller's Formula, which was developed by biostatistician Ralph Mosteller in 1987.

Mosteller's Formula: $\frac{\sqrt{(height(cm))\cdot(weight(kg))}}{60}$

We utilized this formula because it is known for it's simplicity while also obtaining an acceptable degree of accuracy, which is the reason it has become increasingly popular over time. In 2002 the Journal of Oncolocy Pharmacy Practice (OCPP) published an article comparing various methods of calculating BSA for cancer prescriptions and concluded that Mosteller's formula was the optimal choice [Vu02]. Later in 2016, OCPP published another article comparing BSA formulas for curative cancer treatment and concluded that the Mosteller formula was the best method for calculating BSA [Fan16].

Given this, denote the control vector as $\mathbf{u}(t) = \begin{bmatrix} D_c(t) & D_d(t) \end{bmatrix}^\intercal$ where D_d is the amount of doxorubicin and D_c the amount of cyclophosphamide in the body of the patient. For our 65 year old patient, the dosage given at any time t, as measured by BSA, must adhere to

$$1394 \leq D_c \leq 1,700\text{mg} \tag{5}$$

$$104.55 \leq D_d \leq 127.5\text{mg} \tag{6}$$

These constraints were found by inputting the height and weight discussed previously into Mosteller's Formula and then multiplying the result by the maximum prescribed doses found for doxorubicin and cyclophosphamide, $75\frac{mg}{m^2}$[Pfi24] and $1000\frac{mg}{m^2}$[Med24] respectively. We thus obtain the maximum

bounds of $1,700$mg for cyclophosphamide and 127.5mg for doxorubicin. For the lower bounds, we found multiple sources that advise going no lower than 85% of the maximum dose, as studies have shown that doing so causes a decrease in patient survivability. Acknowledging this, we made our lower bound 82% of our maximum dosage.

2.3 The Cost Functional

To further simplify our problem, we assumed that the normal discrete AC treatment would be administered to our patient every 3 weeks for a total of 6 doses. That is, the AC treatment lasts for a total of 126 days. Hence, to see if our optimal control as well as our dynamics are correct, we opted to model for $t \in [0, 156]$. The first 126 days correspond to the hypothetical, but realistic, AC treatment window. We model the last 30 days to monitor the effectiveness of our treatment. Assuming the treatment is effective, either the cancer will be eradicated after 126 days or the remaining cancer will be killed off by the immune cells within a month's time.

Our cost functional takes the form of

$$J[\mathbf{u}] = \int_0^{126} \mathbf{u}(t)^\intercal Q\mathbf{u}(t) + T^2(t)\,\mathrm{d}t + \xi_{126}T^2(126) \tag{7}$$

where $Q = \begin{bmatrix} \xi_c & 0 \\ 0 & \xi_d \end{bmatrix}$ denotes the positive weights for each of the elements of $\mathbf{u}$ and ξ_{126} the final weight for the final tumor population at the end of 126 days. The Lagrangian

$$L(t, \mathbf{x}(t), \mathbf{u}(t)) = \mathbf{u}(t)^\intercal Q\mathbf{u}(t) + T^2(t) \tag{8}$$

helps us take into consideration that we do not want to overdose a person as we administer the chemo, but we also minimize the number of IDC cells present throughout the treatment.

3 Solving the Problem

During a meeting with Dr. Barker, we realized that working with hard constraints on the optimal control would require KKT conditions and would make the problem harder to solve. He suggested we use soft constraints by adding our inequality constraints into our cost functional.

3.1 1st Attempt

In order to incorporate these hard constraints, we added in the following equations

$$\frac{\gamma_1}{1+(D_d-127.5)^{\lambda_1}} \tag{9}$$

$$\frac{\gamma_2}{1+(D_d-104.55)^{\lambda_2}} \tag{10}$$

$$\frac{\gamma_3}{1+(D_c-1,700)^{\lambda_3}} \tag{11}$$

$$\frac{\gamma_4}{1+(D_c-1394)^{\lambda_4}} \tag{12}$$

These allow us to transform the hard constraints into soft constraints. We left these with subscripts based on the assumption that not all our constants for these should be the same.

Thus, our cost functional becomes

$$\begin{aligned} J[\mathbf{u}] = \int_0^{126} \mathbf{u}(t)^{\mathsf{T}} Q\mathbf{u}(t) + T^2(t) + \frac{\gamma_1}{1+(D_d-127.5)^{\lambda_1}} + \frac{\gamma_2}{1+(D_d-104.55)^{\lambda_2}} \\ + \frac{\gamma_3}{1+(D_c-1,700)^{\lambda_3}} + \frac{\gamma_4}{1+(D_c-1394)^{\lambda_4}}\,\mathrm{d}t + \xi_{126}T^2(126) \end{aligned} \tag{13}$$

This gives us the Hamiltonian

$$\begin{aligned} H &= H(t,\mathbf{x}(t),\mathbf{u}(t),\mathbf{p}(t)) = \mathbf{p}(t)\cdot\mathbf{x}'(t) - L(t,\mathbf{x}(t),\mathbf{u}(t)) \\ &= g_T p_1 T \ln\left(\frac{T_{\max}}{T}\right) - a_1 p_1 NT - a_2 p_1 NKT - a_3 p_1 CDT - p_1\left(\frac{4}{5}E_c D_c + E_d D_d\right)T \\ &+ g_N p_2 N \ln\left(\frac{N_{\max}}{N}\right) - k_N p_2 N - a_0 p_2 NT \\ &+ r_{CD} p_3 - k_{CD} p_3 CD + p_3 \frac{\rho_0 CDT^i}{\alpha_0 + T^i} - a_4 p_3 CDT - b_{CD} p_3 D_c CD \\ &+ r_{NK} p_4 - k_{NK} p_4 NK + p_4 \frac{\rho_1 NKT^j}{\alpha_1 + T^j} - a_5 p_4 NKT \\ &- \xi_c D_c^2 - \xi_d D_d^2 - T^2 - \frac{\gamma_1}{1+(D_d-127.5)^{\lambda_1}} - \frac{\gamma_2}{1+(D_d-104.55)^{\lambda_2}} \\ &- \frac{\gamma_3}{1+(D_c-1,700)^{\lambda_3}} - \frac{\gamma_4}{1+(D_c-1394)^{\lambda_4}} \end{aligned} \tag{14}$$

By Pontryagin's maximum principle (PMP), we have

$$\frac{DH}{D\mathbf{u}} = \left[\frac{\partial H}{\partial D_c} \frac{\partial H}{\partial D_d}\right] = \mathbf{0}$$

Hence,

$$0 = \frac{\partial H}{\partial D_c} = -\frac{4}{5}p_1 E_c T - p_3 b_{CD} CD - 2\xi_c D_c + \frac{\gamma_3\lambda_3(D_c - 1,700)^{\lambda_3 - 1}}{(1 + (D_c - 1,700)^{\lambda_3})^2} + \frac{\gamma_4\lambda_4(D_c - 1394)^{\lambda_4 - 1}}{(1 + (D_c - 1394)^{\lambda_4})^2}$$

$$0 = \frac{\partial H}{\partial D_d} = -E_d p_1 T - 2\xi_d D_d + \frac{\gamma_1\lambda_1(D_d - 127.5)^{\lambda_1 - 1}}{(1 + (D_d - 127.5)^{\lambda_1})^2} + \frac{\gamma_2\lambda_2(D_d - 104.55)^{\lambda_2 - 1}}{(1 + (D_d - 104.55)^{\lambda_2})^2}$$

Moreover, we also know that $\mathbf{p}' = -\frac{DH}{D\mathbf{x}}$ and that $\mathbf{p}(126) = -\frac{D\phi}{D\mathbf{x}(126)}$, where $\phi(\mathbf{x}(126)) = \xi_{126}T^2(126)$. Our costates evolve according to

$$\begin{aligned} p_1' = -\Bigg(& - g_T p_1 + g_T p_1 \ln\left(\frac{T_{max}}{T}\right) - a_1 p_1 N - a_2 p_1 NK - a_3 p_1 CD - p_1\left(\frac{4}{5}E_c D_c + E_d D_d\right) \\ & - a_0 p_2 N + \frac{(\alpha_0 + T^i)(i\rho_0 p_3 CDT^{i-1}) - (\rho_0 p_3 CDT^i)(iT^{i-1})}{(\alpha_0 + T^i)^2} - a_4 p_3 CD \\ & + \frac{(\alpha_1 + T^j)(j\rho_1 p_4 NKT^{j-1}) - (\rho_1 p_4 NKT^j)(jT^{i-1})}{(\alpha_1 + T^j)^2} - a_5 p_4 NK - 2T\Bigg) \end{aligned} \tag{15}$$

$$p_2' = -(-a_1 p_1 T + g_N p_2 \ln\left(\frac{N_{max}}{N}\right) - g_N p_2 - k_N p_2 - a_0 p_2 T) \tag{16}$$

$$p_3' = -(-a_3 p_1 T - k_{CD} p_3 - p_3 \frac{\rho_0 T^i}{\alpha_0 + T^i} - a_4 p_3 T - b_{CD} p_3 D_c) \tag{17}$$

$$p_4' = -(-a_2 p_1 T - k_{NK} p_4 - p_4 \frac{\rho_1 T^i}{\alpha_1 + T^i} - a_5 p_4 T) \tag{18}$$

with $p_1(126) = -2\xi_{126}T(126)$ and $p_2(126) = p_3(126) = p_4(126) = 0$.

Notice that our state dynamics, costates, and control are all interwoven into each other, which prevents us from solving for $\mathbf{u}$ algebraically. Thus, like we learned in our Volume 4 HIV Lab, we solved for these iteratively:

1. Take an initial guess $\mathbf{u}$ and make a copy $\bar{\mathbf{u}}$ (for a convergence check)
2. Solve the state ODEs of $\mathbf{x}'$ (i.e. Eqns 1-4) using $\mathbf{u}$
3. Solve the costate ODEs of $\mathbf{p}'$, using the found state $\mathbf{x}$

4. Using $\mathbf{x}$ and $\mathbf{p}$, find the roots of the derivatives $\frac{DH}{D\mathbf{u}}$ in order to find optimal control $\mathbf{u}$.

5. Clip $\mathbf{u}$ so that it is within its bounds

6. Check for convergence and repeat if necessary by updating $\bar{\mathbf{u}}$

In order to solve for the ODEs, we employed `solve_ivp`, from `scipy`. For the root finding, we utilized `root` also from `scipy`. To check for convergence, we utilized `allclose` from `numpy` and set `rtol=0` and `atol=1e-2`. We attach our code at the very end of our work. We also add the values of the ODE constants in Appendix B.

3.2 2nd Attempt

The equations used as soft constraints in Eqns 9-12 penalize the doses being closer to aforementioned limits for each drug. Our fear was that a maximum dosage would never be prescribed, which is common practice for chemotherapy treatment. Moreover, as shown in our results, our first attempt produced bang-bang solutions which we thought were rather interesting but not practical.

We opted to switch Eqns 9-12 out by using

$$\gamma_1(D_d - 127.5)^{\lambda_1} \tag{19}$$
$$\gamma_2(D_d - 104.55)^{\lambda_2} \tag{20}$$
$$\gamma_3(D_c - 1,700)^{\lambda_3} \tag{21}$$
$$\gamma_4(D_c - 1394)^{\lambda_4}. \tag{22}$$

These, according to our understanding, do allow the doses to be at the boundaries. However, depending on the weight, doses in between the boundaries will be penalized heavier. The further the dose is away from the boundaries of the control, the more it will be penalized.

Thus, we get

$$
\begin{aligned}
H = {} & g_T p_1 T \ln\left(\frac{T_{\max}}{T}\right) - a_1 p_1 NT - a_2 p_1 NKT - a_3 p_1 CDT - p_1\left(\frac{4}{5}E_c D_c + E_d D_d\right)T \\
& + g_N p_2 N \ln\left(\frac{N_{\max}}{N}\right) - k_N p_2 N - a_0 p_2 NT \\
& + r_{CD} p_3 - k_{CD} p_3 CD + p_3 \frac{\rho_0 CDT^i}{\alpha_0 + T^i} - a_4 p_3 CDT - b_{CD} p_3 D_c CD \\
& + r_{NK} p_4 - k_{NK} p_4 NK + p_4 \frac{\rho_1 NKT^j}{\alpha_1 + T^j} - a_5 p_4 NKT \\
& - \xi_c D_c^2 - \xi_d D_d^2 - T^2 - \gamma_1 (D_d - 127.5)^{\lambda_1} - \gamma_2 (D_d - 104.55)^{\lambda_2} \\
& - \gamma_3 (D_c - 1,700)^{\lambda_3} - \gamma_4 (D_c - 1394)^{\lambda_4} \qquad (23)
\end{aligned}
$$

Thus, by PMP, our control must satisfy

$$
\begin{aligned}
0 = \frac{\partial H}{\partial D_c} &= -\frac{4}{5} p_1 E_c T - p_3 b_{CD} CD - 2\xi_c D_c - \gamma_3 \lambda_3 (D_c - 1,700)^{\lambda_3 - 1} - \gamma_4 \lambda_4 (D_d - 104.55)^{\lambda_4 - 1} \\
0 = \frac{\partial H}{\partial D_d} &= -E_d p_1 T - 2\xi_d D_d - \gamma_1 \lambda_1 (D_d - 127.5)^{\lambda_1} - \gamma_2 \lambda_2 (D_d - 104.55)^{\lambda_2}
\end{aligned}
$$

Notice that our costates stay the same as in Eqns 15-18. Also, as in our first attempt, we solved for the optimal control using our iterative approach.

4 Results & Analysis

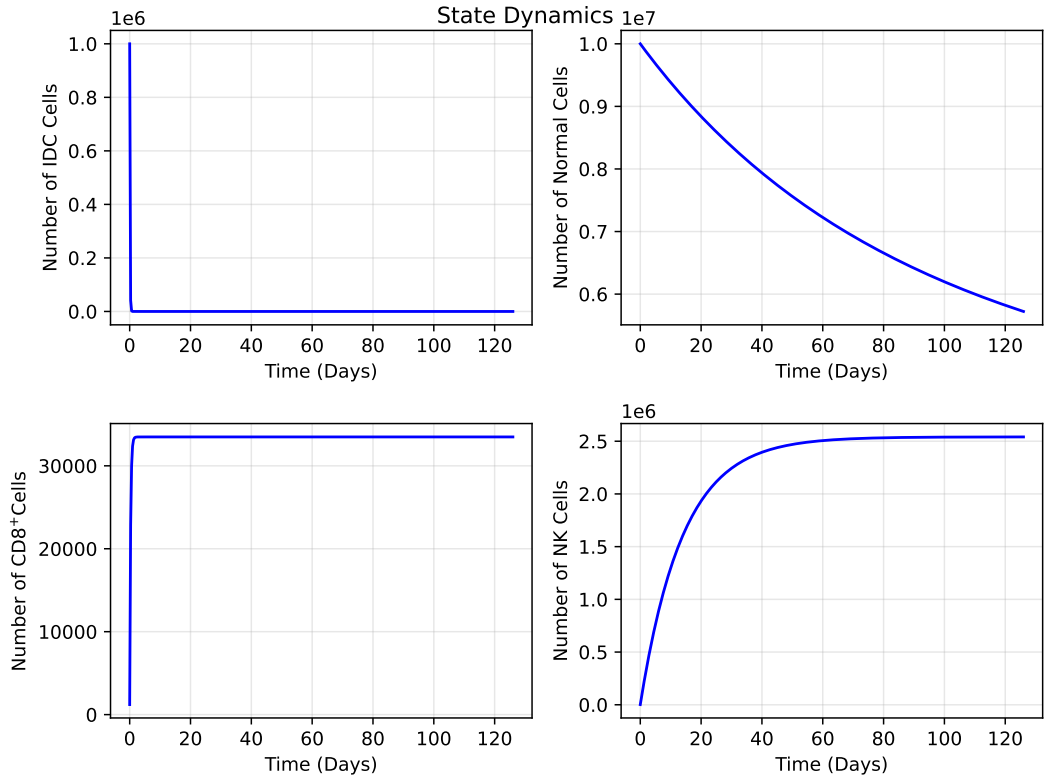


Figure 1: The state dynamics for our first method of incorporating our hard constraints as soft constraints.

Our results for the state dynamics are given in Fig. 1. The optimal control $\mathbf{u}$ can be seen in Fig. 2. As shown in the graph, the optimal control seems to be a bang-bang solution, even though our problem itself is not easily identified as a bang-bang problem.

Physically, this chemo regime cannot be possible as it would result in high toxicity levels for our patient due to the high drug concentration in their body. Moreover, as shown in Fig. 1, the IDC was completely killed within a day. This of course is impossible given the high tumor cell count as chemotherapy does not eradicate breast cancers with a single dose.

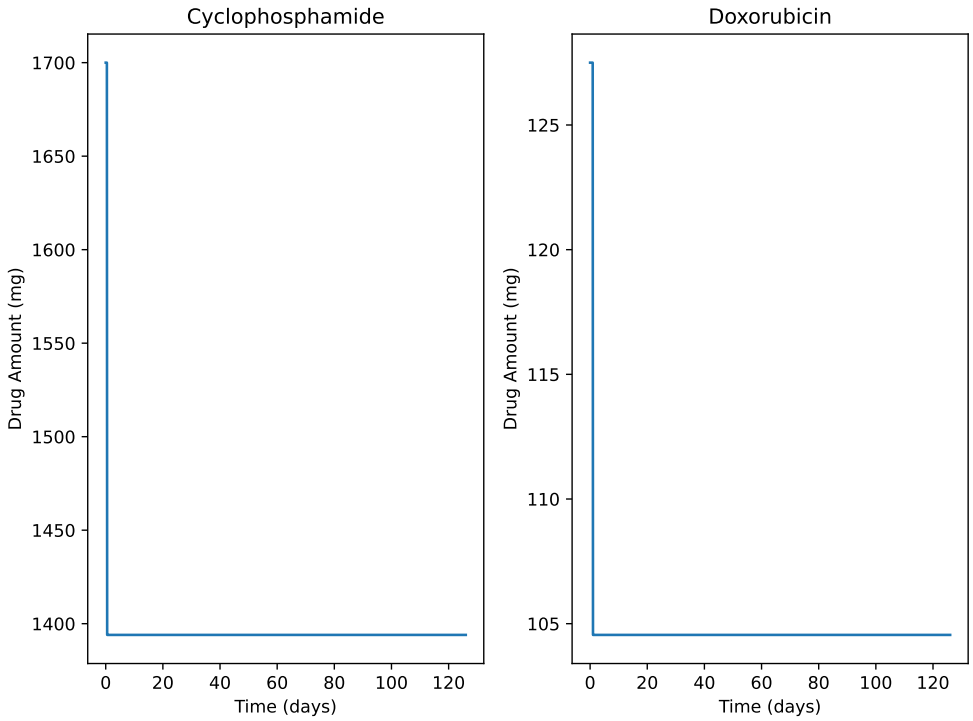


Figure 2: The optimal control for our first attempt at incorporating soft constraints.

When we evolve the state dynamics past our AC treatment window, we get the results shown in Fig. 3. The cancer is gone by the final treatment and remains that way. Though there is a spike at the end, we further expanded the final state $\mathbf{x}(126)$ and could not find any levels of IDC past 1^{-100}. Thus, we eradicated the cancer by the end of the treatment cycle as intended. Any cancer that does regrow can be taken care of by the immune cells.

One interesting thing to note is the evolution of the normal epithelial cells keeps decaying after the treatment. This can be due to inappropriate modeling in our ODEs, but another possibility is that the continual dosage has left some side effects on our patient. Though our modeling does not incorporate death of normal cells by chemo, doxorubicin has no way of knowing which cells to target. Thus, it is very possible that the death by competition in Eqn. 2 may possibly include some interaction with doxorubicin.

Seeing the bang-bang solution we obtained for using Eqns. 9 - 12, we then modeled by using Eqns. 19-22. Our results are displayed in Figs. 4 - 6.

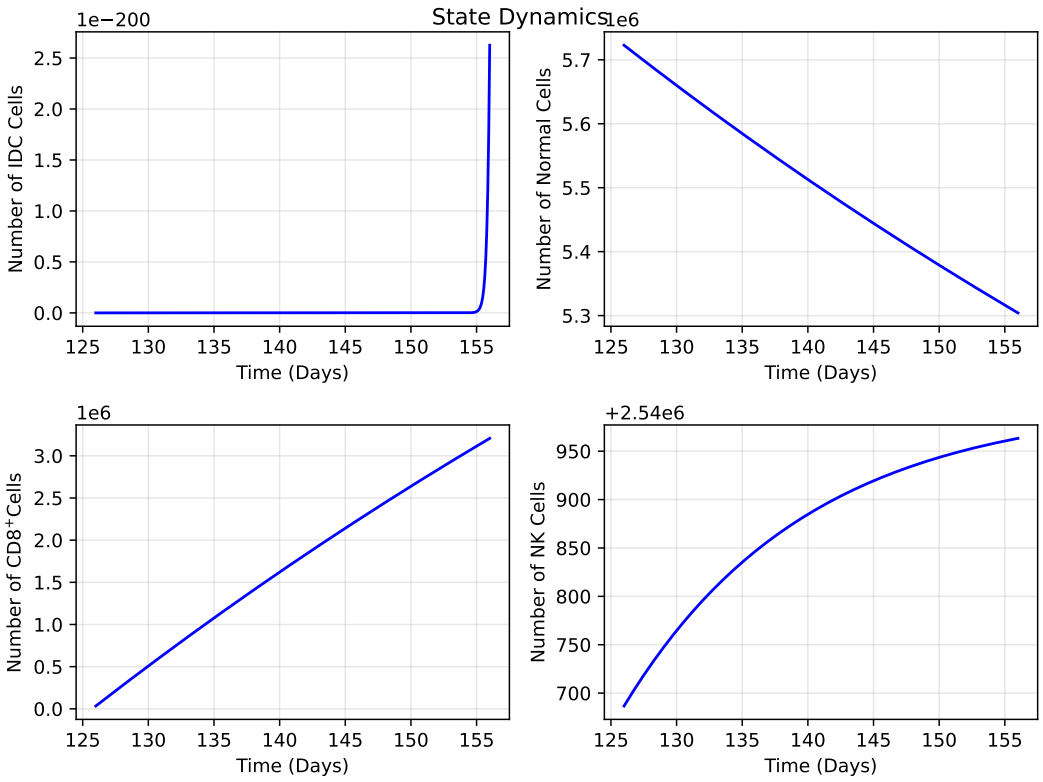


Figure 3: The state dynamics following the optimal AC treatment.

By analyzing Fig. 4, we can see that we obtain another bang-bang solution. In fact, the result is equal, if not very close to, the one we obtained in our first attempt. This seems to suggest that our answer should be bang-bang for this problem. But, as stated earlier, this solution for optimal control is not physically possible

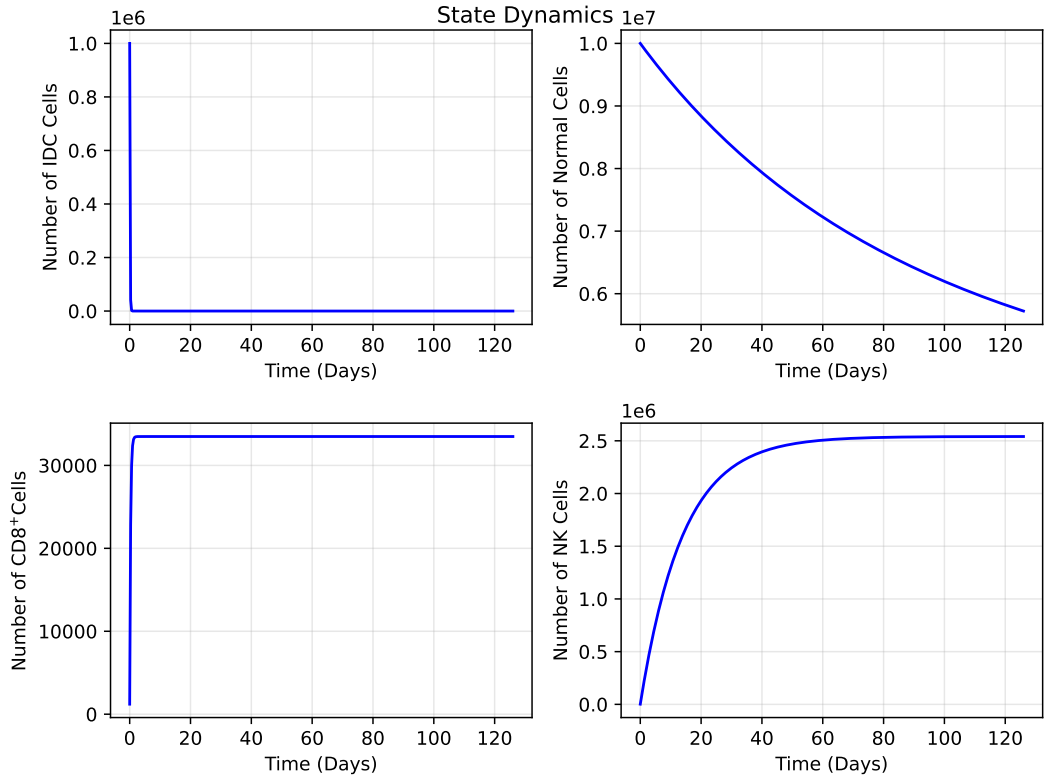


Figure 4: The state dynamics during the AC treatment window for our second attempt at incorporating constraints as soft.

In most papers that model the affect of chemotherapy on cancer, the chemotherapy decays according to some exponential term $1 - e^{M(t)}$, where $M(t)$ gives the concentration of the drug at time t (for example see [LR03] [ABA+24]). This allows the drug to decay according to its half-life. However, we thought that would constrain the optimal control to a specific form that it may not necessarily follow. Thus, it seems that our ODE set up may not be at all that correct as we do not use some sort of exponential decay. This

can explain the bang-bang solution. With an exponential, we might except the dosage to increase at some point again but then decay rapidly due to the half-life of both doxorubicin and cyclophosphamide.

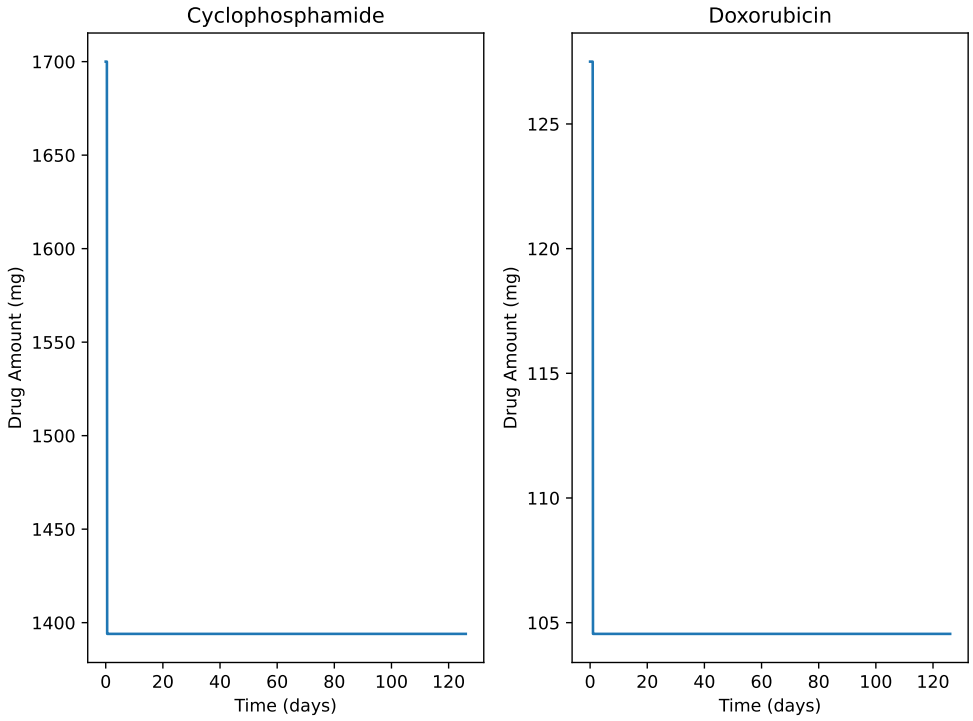


Figure 5: Optimal control for our second attempt.

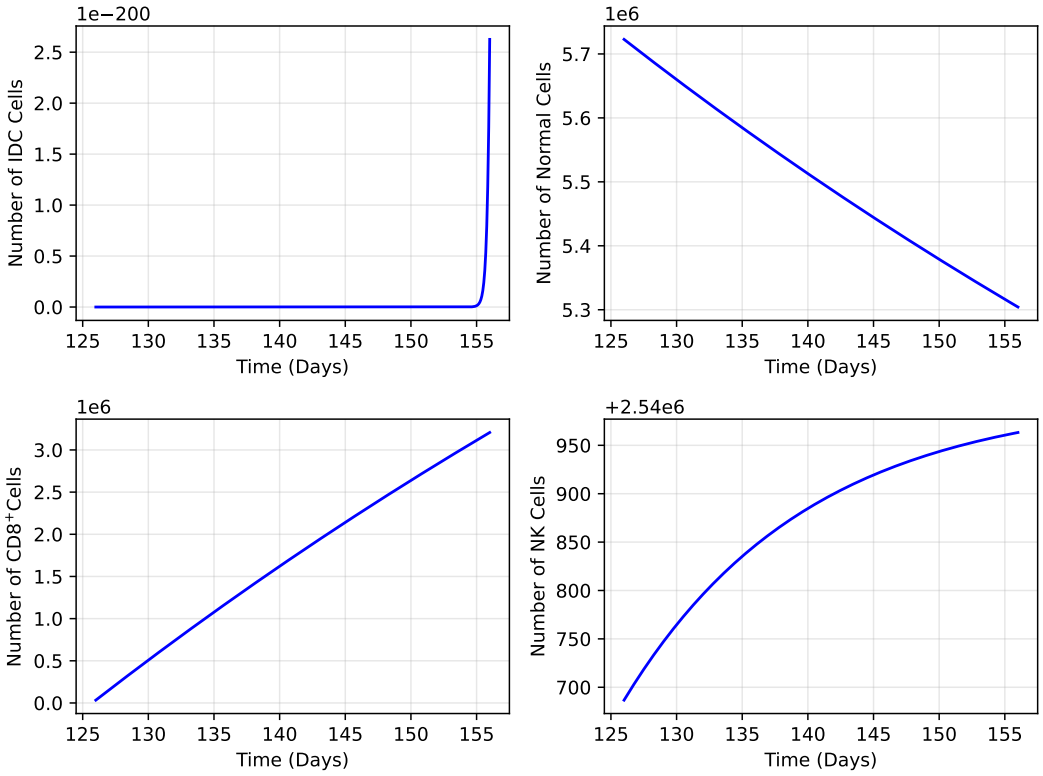


Figure 6: State dynamics post control treatment (for our second attempt).

5 Conclusion

[ADE+20] and [Moo18] both mention that OTC has great potential for cancer treatment optimization, especially when it comes to optimizing drug regimes. However, the potential is limited to the quality of the model. Thus, our work is limited to the model we can use to describe the dynamics of tumor-immune-chemo interactions. Our optimal control is a bang-bang solution by using soft constraints rather than hard constraints. Perchance, the problem necessitates the use of Karush-Kuhn-Tucker conditions to deal well with the hard limitations on the dosage amount.

In [GKGK], the authors incorporate some decay rate into the therapeutic function. Perchance, if we added something along those lines or through the use of the exponential decay, we might arrive at a better non bang-bang

solution. In that specific work, their Hamiltonian is linear in $\mathbf{u}$. Our original H is also linear in $\mathbf{u}$ which could possibly explain why our solution was bang-bang.

However, as stated in our introduction, [SNK23] mentions how one would expect a piecewise model as an optimal control for chemotherapy. Conceptually this is consistent with our model, as our control is the amount of AC treatment to use in a given session. In other words, our control is also a constant piecewise with a single switch (treatment on or treatment off).

Furthermore, we also theorized that we could implement a discretization using forward Euler in order to try to apply discrete PMP. Though we did not attempt this, our thought was that the discretization could allow us to mimic the discrete doses of AC. This would then allow us to also find optimal control not just in the dose amount but also in the frequency of the drug administration.

Overall, our project shows that an optimal control can be found for the AC treatment used in IDC. While not practical, our research can be used to understand how an initial dose can affect the tumor. Of course, our work has to be modified to appropriately affect the cancer over time rather than just immediately killing it. However, we can use our model to quickly determine an optimal first dose that is personalized to our patient. This would reduce the need to rely on decision algorithms that are generalized and not necessarily adapted to a specific individual.

A Definitions

- Cyclophosphamide: a chemo drug used to prevent cancer cells from dividing.
- Doxorubicin: a chemo drug used to kill cancer cells at any stage. It normally kills from the inside. A free FDA sheet is available online.
- Invasive Ductal Carcinoma (IDC): invasive breast cancer that starts in the milk ducts, the tubes that carry milk from the lobules to the nipple. Specifically, IDC begins in the epithelial cells of the milk duct.
- *In silico*: a biology experiment performed via computer simulations.

B Constants

Here we give a breakdown of the constants we used in our modeling. The majority came from the papers cited in the references by De Pillis. Whenever we encountered a parameter we did not have, we estimated by taking an average of other similar parameters and either multiplying by some constant to increase it or dividing to make it smaller. For example, we did not have the competition constant between normal cells and tumor cells but we did know those for NK and $CD8^+$, so that we estimated by using an average of those two but dividing by 50 to make the interaction weaker. We do know that normal cells cannot kill cancer directly besides competition, but the competition ought not be assumed stronger or as equally strong as that between NK and $CD8^+$. Estimated values are denoted with $*$.

Constant	Value	Units	Description
$T_{\max}$	1e8	Cells	The carrying capacity of IDC
$N^*_{\max}$	10.3e8	Cells	Normal cell carrying capacity
g_T	$\frac{1}{90}$	$\frac{1}{day}$	Growth of IDC
g_N	$\frac{1}{147}$	$\frac{1}{day}$	Growth of epithelial cells
r_{CD}	1.21e5	$\frac{cell}{day}$	The rate of $CD8^+$ influx to an area
r^*_{NK}	r_{CD}·1.5	$\frac{cell}{day}$	The rate of NK influx to an area
k_N	$\frac{1}{147}$	$\frac{1}{day}$	The turnover rate of epithelial cells
k_{CD}	9e-3	$\frac{1}{day}$	The turnover rate of $CD8^+$
k_{NK}	$\frac{1}{14}$	$\frac{1}{day}$	The turnover rate of NK
a^*_0	$\frac{a2+a3}{5}$	$\frac{1}{cell \cdot day}$	Death of normal cells by tumor
a^*_1	$\frac{a2+a3}{100}$	$\frac{1}{cell \cdot day}$	Death of tumor cells by normal
a_2	7.13e-10	$\frac{1}{cell \cdot day}$	Fractional (non)-ligand-transduced tumor cell kill by NK
a^*_3	$a2 \cdot 5$	$\frac{1}{cell \cdot day}$	Fractional (non)-ligand-transduced tumor cell kill by $CD8^+$
a_4	3.422e-10	$\frac{1}{cell \cdot day}$	Death/Inactivation of CD8+ cells by tumor
a_5	1e-7	$\frac{1}{cell \cdot day}$	Death/Inactivation of NK cells by tumor

References

[ABA+24] Ahmed J. Abougarair, Mohsen Bakouri, Abdulrahman Alduraywish, Omar G. Mrehel, Abdulrahman Alqahtani, Tariq Alqahtani, Yousef Alharbi, and Md Samsuzzaman. Optimizing cancer treatment using optimal control theory. *AIMS Mathematics*, 9(11):31740–31769, 2024.

[ABZ21] Jensen N Amens, G¨okhan Bah¸cecioglu, and Pinar Zorlutuna. Immune system effects on breast cancer. *Cell Mol Bioeng*, 14(4):279–292, Aug 2021.

[ADE+20] Jarrett AM, Faghihi D, Ii DAH Lima EABF, Virostko J, Biros G, Patt D, and Yankeelov TE. Optimal control theory for personalized therapeutic regimens in oncology: Background, history, challenges, and opportunities. *Journal of clinical medicine*, 9(5):13, 2020.

[CGG+24] Loïs Coënon, Mannon Geindreau, François Ghiringhelli, Martin Villalba, and Mélanie Bruchard. Natural killer cells at the frontline in the fight against cancer. *Cell Death & Disease*, 15, 2024.

[DeP24] Jamie DePolo. Types of breast cancer, 2024.

[Fan16] Karen Fancher. Comparison of two different formulas for body surface area in adults at extremes of height and weight. *Journal of Physilogical Anthropology*, 22:690–695, 2016.

[GBCB] Melina-Lorén Kienle Garrido1, Tim Breitenbach, Kurt Chudej, and Alfio Borzì. Modeling and numerical solution of a cancer therapy optimal control problem. *Applied Mathematics*, 9(8).

[GKGK] N. L. Grigorenko, E. N. Khailov, E. V. Grigorieva, and A. D. Klimenkova. Lotka–volterra competition model with a nonmonotone therapy function for finding optimal strategies in the treatment of blood cancers. *Proceedings of the Steklov Institute of Mathematics*, 317(1).

[GKGK21] N. L. Grigorenko, E. N. Khailov, E. V. Grigorieva, and A. D. Klimenkova. Optimal strategies in the treatment of cancers in the

lotka–volterra mathematical model of competition. *Proceedings of the Steklov Institute of Mathematics*, 313(1), 2021.

[Ins24] National Cancer Institute, 2024.

[LR03] De Pillis L.G and A Radunskaya. "the dynamics of an optimally controlled tumor model: A case study". *"Mathematical and Computer Modelling"*, 37(11):1221–1244, 2003. Modeling and Simulation of Tumor Development, Treatment, and Control.

[Med24] Medscape. Cytoxan (cyclophosphamide): Drug information, 2024. Accessed: 2025-04-07.

[Moo18] Helen Moore. How to mathematically optimize drug regimens using optimal control. *Journal of Pharmacokinetics and Pharmacodynamics*, 45, 2018.

[Ori14] Orimadegun, AE. Evaluation of Five Formulae for Estimating Body Surface Area of Nigerian Children. *Ann Med Health Sci Res.*, 2014.

[Pfi24] Pfizer, Inc. Doxorubicin: Dosage and administration, 2024. Accessed: 2025-04-07.

[Pin58] Donald Pinkel. The Use of Body Surface Area as a Criterion of Drug Dosage in Cancer Chemotherapy. *Cancer Research*, 18, 1958.

[SNK23] N. Subbotina, N. Novoselova, and E. Krupennikov. Optimal control theory and calculus of variations in mathematical models of chemotherapy of malignant tumors. *Mathematics*, 11(20):4301, 2023.

[SR19] JG Scott and RC Rockne. Introduction to mathematical oncology. *JCO clinical cancer informatics*, April 3 2019.

[U.S20] U.S. Food and Drug Administration. Doxorubicin hydrochloride label – fda, 2020. Accessed: 2025-04-07.

[VGO+20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu

Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[Vu02] Thahn Vu. Standardization of body surface area calculations. *Journal of Physilogical Anthropology*, 8:49–54, 2002.

state_modeling

April 16, 2025

[1]:
```
from IPython import display
from numpy.typing import ArrayLike
from scipy.integrate import solve_ivp
from scipy.interpolate import CubicSpline
from scipy.optimize import root
from typing import Callable
from utils import plot_state_dynamics, plot_control
import matplotlib.pyplot as plt
import numpy as np
```

1 Modeling the State Dynamics Alone

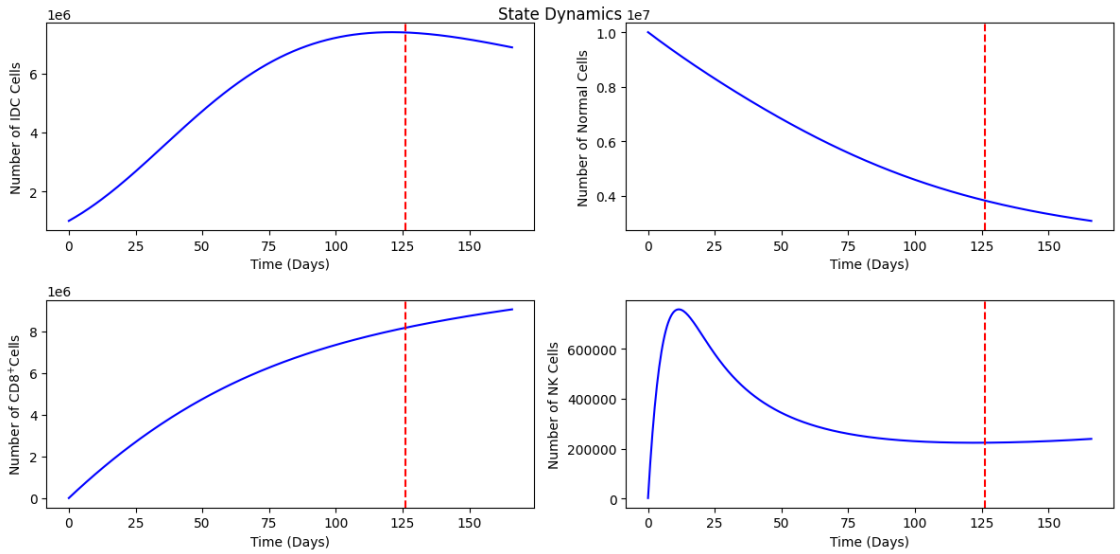
[2]:
```
Tmax = 1e8
Nmax = 10_309_278

# Growth Constants
gT = 1/90
gN = 1/147

# Rate of cell circulation (Units: cell/Day)
rCD = 1.21e5
rNK = rCD * 1.5

# Turn over rates (Units: 1/Day)
kN = 1/147
kCD = 9e-3
kNK = 1/14

# Competition constants (Units: 1/(cell * day))
a4 = 3.422e-10          # Death/Inactivation of CD8+ cells by interacting with␣
 ↪the tumor
a5 = 1e-7               # Death/Inactivation of NK cells by interacting with␣
 ↪the tumor
a2 = 7.13e-10           # Fractional (non)-ligand-transduced tumor cell kill by␣
 ↪NK cells
a3 = a2 * 5             # Fractional (non)-ligand-transduced tumor cell kill by␣
 ↪NK cells
```

```
a1 = (a2 + a3) / 100    # Death of Tumor cells by competition with Normal␣
 ↪epithelial cells
a0 = (a2 + a3) / 5


ode_constants = (Tmax, Nmax, gT, gN, rCD, rNK, kN, kCD, kNK, a0, a1, a2, a3,␣
 ↪a4, a5)
```

[22]:
```
def cancer_dynamics(t:float, y:ArrayLike, ode_const:tuple) -> ArrayLike:
    """Define the cancer dynamics excluding the MMK portions.

    Parameters:
        - t (float) : the time at which to compute the derivatives.
        - y (ArrayLike) : the estimated values of the states.
        - ode_const (tuple) : a tuple containing all the constants
                              for the dynamics of the state EXCLUDING
                              MMK.

    Returns:
        - (ArrayLike) : the derivatives of the cancer dynamics.
    """

    T, N, CD, NK = y
    Tmax, Nmax, gT, gN, rCD, rNK, kN, kCD, kNK, a0, a1, a2, a3, a4, a5 =␣
 ↪ode_const

    dT = gT*T*np.log(Tmax / T) - a1*N*T - a2*NK*T - a3*CD*T
    dN = gN*N*np.log(Nmax / N) - kN*N - a0*N*T
    dCD = rCD - kCD*CD  - a4*CD*T
    dNK = rNK - kNK*NK  - a5*NK*T

    return np.array([dT, dN, dCD, dNK])
```

[4]:
```
cancer0 = np.array([1e6, 1e7, 1.21e3, 1.815e3])
t0, t1, tf = 0, 120, 156
t_span = (t0, t1)
cancer_sol = solve_ivp(fun=cancer_dynamics, t_span=t_span, y0=cancer0,␣
 ↪dense_output=True, max_step=0.1, args=(ode_constants, ))
```

[5]:
```
plot_state_dynamics(cancer_sol, tf=166)
```



2 Modeling the State Dynamics with MMK

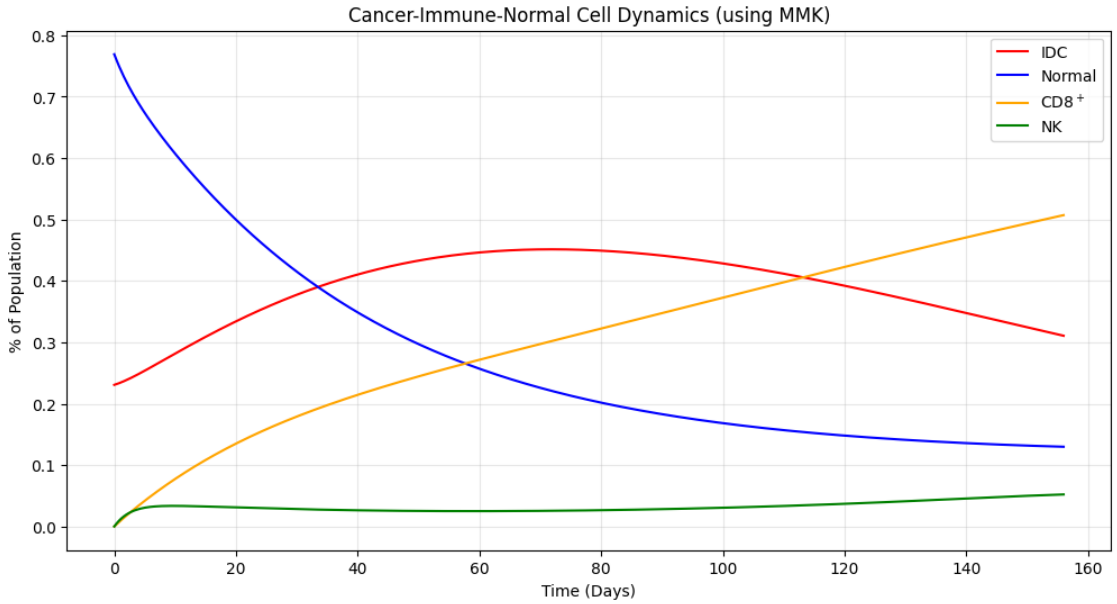
[3]:
```
# MMK for CD8+ (Computational Model by DePillis)
i = 1
rho0 = 1.245e-2   # Units 1\Day
alpha0 = 2.019e7  # Cells

# MMK for NK
# Used for IL-2 supplementation in dePillis so I estimated
j = 2
rho1 = rho0*5
alpha1 = alpha0 * 1.5

mmk_constants = (i, rho0, alpha0, j, rho1, alpha1)
```

[4]:
```
def cancer_MMK_dynamics(t:float, y:ArrayLike, ode_const:tuple, mmk_const:tuple)␣
 ↪-> ArrayLike:
    """Define the cancer dynamics incorporating the MMK portions.

    Parameters:
        - t (float) : the time at which to compute the derivatives.
        - y (ArrayLike) : the estimated values of the states.
        - ode_const (tuple) : a tuple containing all the constants
                              for the dynamics of the state EXCLUDING
                              MMK.
        - mmk_const (tuple) : a tuple containing the constants ONLY for
                              MMK.
```

```
    Returns:
        - (ArrayLike) : the derivatives of the cancer dynamics.
    """

    T, N, CD, NK = y
    Tmax, Nmax, gT, gN, rCD, rNK, kN, kCD, kNK, a0, a1, a2, a3, a4, a5 =␣
↪ode_const
    i, 0, 0, j, 1, 1 = mmk_const

    dT = gT*T*np.log(Tmax / T) - a1*N*T - a2*NK*T - a3*CD*T
    dN = gN*N*np.log(Nmax / N) - kN*N - a0*N*T
    dCD = rCD - kCD*CD  - a4*CD*T + ( 0 * CD * T**i)/( 0 + T**i)
    dNK = rNK - kNK*NK  - a5*NK*T + + ( 1 * CD * T**j)/( 1 + T**j)

    return np.array([dT, dN, dCD, dNK])
```

[14]:
```
cancer0 = np.array([3e6, 1e7, 1.21e3, 1.815e3])
t0, t1, tf = 0, 120, 156
t_span = (t0, t1)
cancer_mmk_sol = solve_ivp(fun=cancer_MMK_dynamics, t_span=t_span, y0=cancer0,␣
↪dense_output=True, max_step=0.1, args=(ode_constants, mmk_constants))
```

[15]:
```
plot_state_dynamics(cancer_mmk_sol, tf=156, title='Cancer-Immune-Normal Cell␣
↪Dynamics (using MMK)', normalize=True, use_grid=True, animate=False)
```



3 Solving for optimal control

[5]:
```
# Doxorubicin
fD = 1
Ed = 0.9

# Cyclophosphamide
fC = 4/5
Ec = 0.9
bCD = 0.5

chemo_constants = (fD, Ed, fC, Ec, bCD)
```

[6]:
```
def state_equations(t:float, y:ArrayLike, u_interp:Callable, ode_const:tuple,␣
↪mmk_const:tuple, chemo_const:tuple) -> ArrayLike:
    """Define the cancer dynamics incorporating MMK and chemotherapy.

    Parameters:
        - t (float) : the time at which to compute the derivatives.
        - y (ArrayLike) : the estimated values of the states.
        - u_interp (Callable) : a callable function to compute the
                                chemo (control) at a given time t.
        - ode_const (tuple) : a tuple containing all the constants
                              for the dynamics of the state EXCLUDING
                              MMK.
        - mmk_const (tuple) : a tuple containing the constants ONLY for
                              MMK.
        - chemo_const (tuple) : a tuple containing the constants ONLY
                                for chemo.

    Returns:
        - (ArrayLike) : the derivatives of the cancer dynamics.
    """

    T, N, CD, NK = y
    Dd, Dc = u_interp(t)
    Tmax, Nmax, gT, gN, rCD, rNK, kN, kCD, kNK, a0, a1, a2, a3, a4, a5 =␣
↪ode_const
    i, 0, 0, j, 1, 1 = mmk_const
    fD, Ed, fC, Ec, bCD = chemo_const


    dT = gT*T*np.log(Tmax / T) - a1*N*T - a2*NK*T - a3*CD*T - (fC * Ec * Dd +␣
↪fD * Ed * Dc) * T
    dN = gN*N*np.log(Nmax / N) - kN*N - a0*N*T
    dCD = rCD - kCD*CD  - a4*CD*T + ( 0 * CD * T**i)/( 0 + T**i) - bCD * Dc * CD
    dNK = rNK - kNK*NK  - a5*NK*T + + ( 1 * CD * T**j)/( 1 + T**j)
```

```
    return np.array([dT, dN, dCD, dNK])
```

[7]:
```
def costate_equations(t:float, y:ArrayLike, state_sol:Callable, u_interp:Callable, ode_const:tuple, mmk_const:tuple, chemo_const:tuple) -> ArrayLike:
    """Define the costate equations for our problem.

    Parameters:
        - t (float) : the current time being evaluated.
        - y (ArrayLike) : the values of the costates at at a time t.
        - state_sol (Callable) : a callable function to compute the
                                 values of state a time t.
        - u_interp (Callable) : a callable function to compute the
                                optimal control a time t.
        - ode_const (tuple) : a tuple containing all the constants
                              for the dynamics of the state EXCLUDING
                              MMK.
        - mmk_const (tuple) : a tuple containing the constants ONLY for
                              MMK.
        - chemo_const (tuple) : a tuple containing the constants ONLY
                                for chemo.

    Returns:
        - (ArrayLike) : the derivatives of the costate.
    """

    T, N, CD, NK = state_sol.sol(t)
    Dd, Dc = u_interp(t)
    Tmax, Nmax, gT, gN, rCD, rNK, kN, kCD, kNK, a0, a1, a2, a3, a4, a5 = ode_const
    i, 0, 0, j, 1, 1 = mmk_const
    fD, Ed, fC, Ec, bCD = chemo_const
    p1, p2, p3, p4 = y

    dp1 = - (-gT*p1 + gT*p1*np.log(Tmax/T) - a1*p1*N - a2*p1*NK - a3*p1*CD - p1*(fC*Ec*Dc + fD*Ed*Dd) - a0*p2*N
             + (( 0 + T**i) * (i* 0*p3*CD*(T**(i-1))) - ( 0*p3*CD*(T**i) *(i*(T**(i-1)))))/(( 0 + T**i)**2)
             - a4*p3*CD + (( 1 + T**j) * (j* 1*p4*NK*(T**(j-1))) - ( 1*p4*NK*(T**j) * (j*(T**(j-1)))))/(( 1 + T**j)**2)
             - a5*p4*NK - 2*T)
    dp2 = - (-a1*p1*T + gN*p2*np.log(Nmax/N) - gN*p2 - kN*p2 - a0*p2*T)
    dp3 = - (-a3*p1*T - kCD*p3 - p3*(( 0*(T**i)) /( 0 + T**i)) - a4*p3*T - bCD*p3*Dc)
    dp4 = - (-a2*p1*T - kNK*p4 - p4*(( 1*(T**j)) /( 1 + T**j)) - a5*p4*T)
    return np.array([dp1, dp2, dp3, dp4])
```

3.1 Using first Boundary Equations

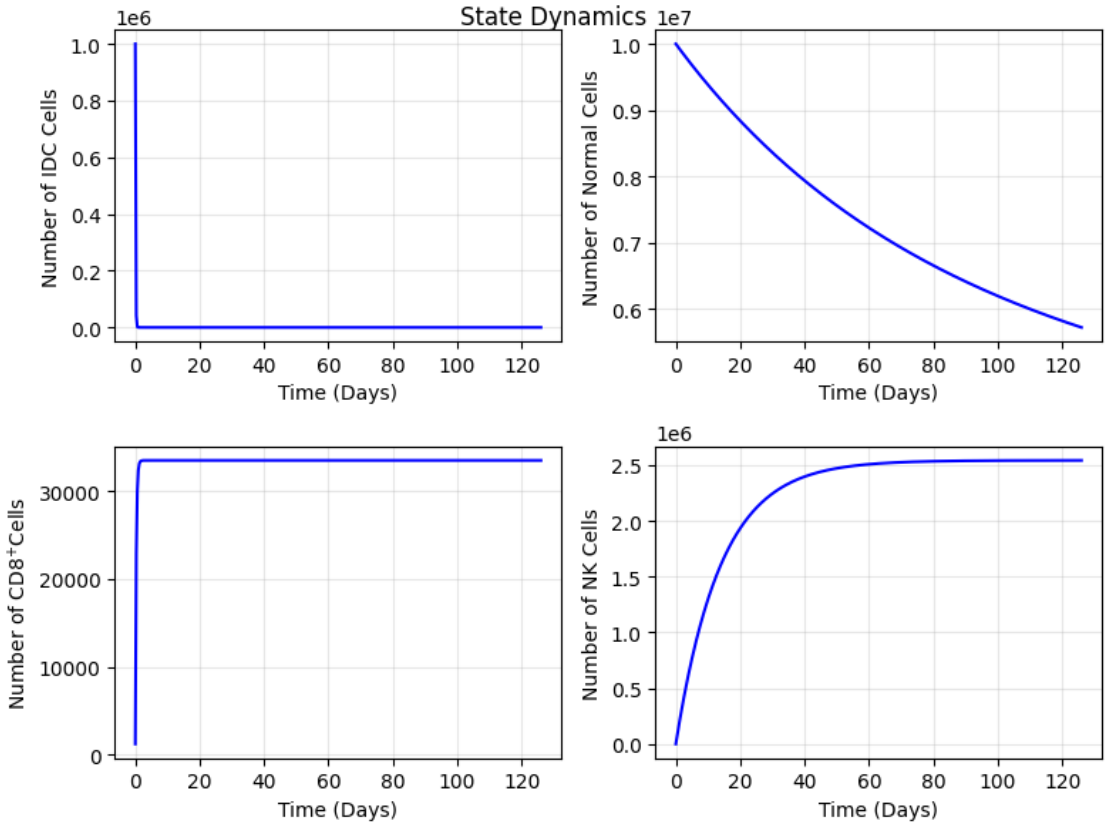
[8]:
```
#epsilon = 0.01
#test = epsilon + 1
test = True
max_step = 0.1
 126 = 80
 c = 75
 d = 75

state0 = np.array([1e6, 1e7, 1.21e3, 1.815e3])
t0, t1, tf = 0, 126, 156
t_span1 = (t0, t1)
t_span1b = (t1, t0)
costate126 = np.zeros(shape=4)
costate126[0] = -  126 * 1

num_vals = 2500
time1 = np.linspace(t0, t1, num=num_vals)
time1b = np.linspace(t1, t0, num=num_vals)
control = np.zeros((2, num_vals))                                    # [Doxo,
                                                                     #  Cyclo]

doxo_lb, doxo_hb = 104.55, 127.5
cyclo_lb, cyclo_hb = 1394, 1_700
cyclo_kill_rate = 0.97
doxo_kill_rate = 0.97
cyclo_lower, cyclo_max = np.log(cyclo_kill_rate * cyclo_lb), np.
 ↪log(cyclo_kill_rate * cyclo_hb)
doxo_lower, doxo_max = np.log(doxo_kill_rate * doxo_lb), np.log(doxo_kill_rate␣
 ↪* doxo_hb)
control[0], control[1] = doxo_max, cyclo_max

iterations = 0
max_iters = 5_000
```

[9]:
```
while test and (iterations < max_iters):

    old_control = control.copy()
    control_interp = CubicSpline(time1, y=control, axis=1)

    # Solve state forwards (t0 -> t1) [T N CD NK]
    state_solu = solve_ivp(fun=state_equations, t_span=t_span1, t_eval=time1,␣
 ↪y0=state0, max_step=max_step,
                           args=(control_interp, ode_constants, mmk_constants,␣
 ↪chemo_constants), dense_output=True)
```

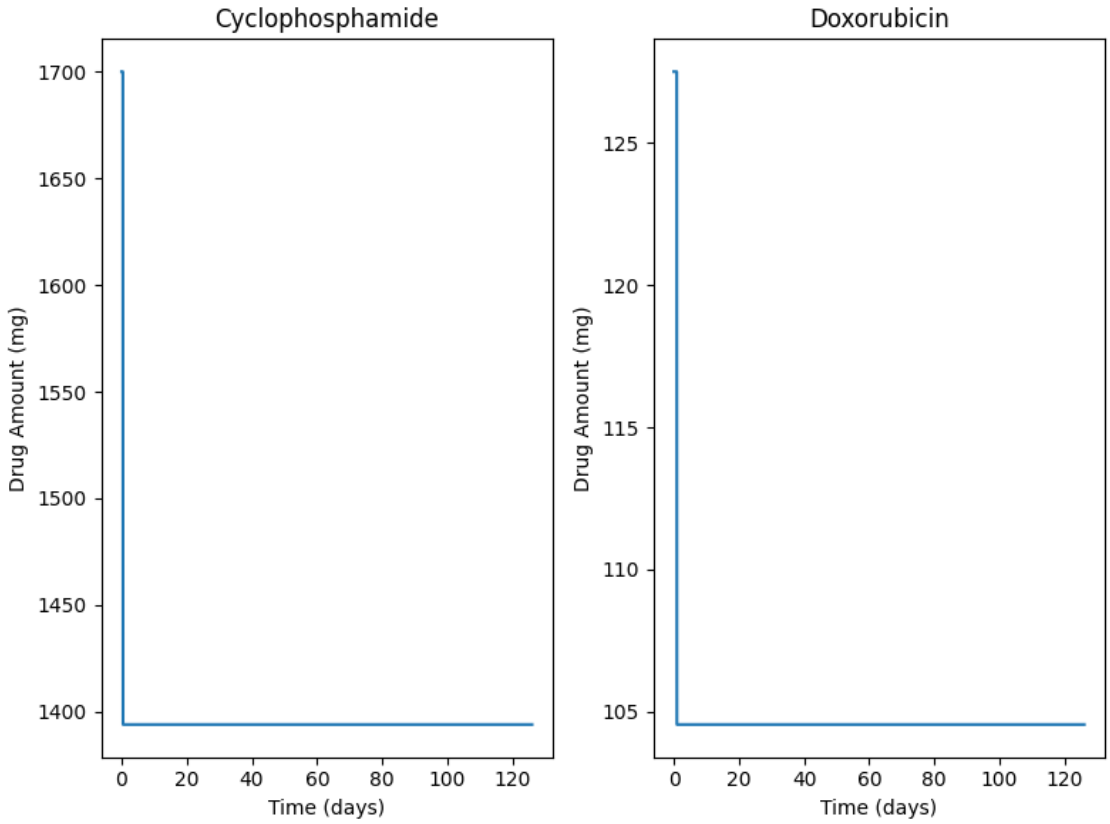
29

```
    # Solve costate backward (t1 -> t0)
    costate_solu = solve_ivp(fun=costate_equations, t_span=t_span1b,␣
↪y0=costate126, t_eval=time1b, max_step=max_step,
                             args=(state_solu, control_interp, ode_constants,␣
↪mmk_constants, chemo_constants), dense_output=True)

    states = state_solu.y
    costates = costate_solu.y[:, ::-1]

     1,  1,  2,  2 = 50, 2, 50, 2
     3,  3,  4,  4 = 50, 2, 50, 2

    def rootu(u):

        u = u.reshape((2, num_vals))

        DH_Du1 = -(4/5) * costates[0] * Ec - costates[2] * bCD * states[2] - 2 *␣
↪ c * u[1] +\
                  ( 3 *  3 * (u[1] - cyclo_max)**( 3 - 1) / (1 + (u[1] -␣
↪cyclo_max)** 3)**2) +\
                  ( 4 *  4 * (u[1] - cyclo_lower)**( 4 - 1) / (1 + (u[1] -␣
↪cyclo_lower)** 4)**2)
        DH_Du2 = - Ed * costates[0] * states[0] - 2 *  d * u[0] +\
                   ( 1 *  1 * (u[0] - doxo_max)**( 1 - 1) / (1 + (u[0] -␣
↪doxo_max)** 1)**2) +\
                   ( 2 *  2 * (u[0] - doxo_lower)**( 2 - 1) / (1 + (u[0] -␣
↪doxo_lower)** 2)**2)

        return (np.array([DH_Du1, DH_Du2])).flatten()

    root_control = root(fun=rootu, x0=control.flatten())
    control = root_control.x
    control = control.reshape((2, num_vals))
    control[0] = np.clip(control[0], doxo_lower, doxo_max)
    control[1] = np.clip(control[1], cyclo_lower, cyclo_max)

    if np.allclose(a=old_control, b=control, rtol=0, atol=1e-2):
        test = False

    iterations += 1
```

[10]:
```
plot_state_dynamics(state_solu, tf=126, use_grid=True, fig_name='./imgs/
↪state_dynamics_bd1.pdf', save_fig=True)
```



[11]:
```
plot_control(np.exp(control), save_fig=True, path="./imgs/optimal_control_bdy1.
↪pdf")
```

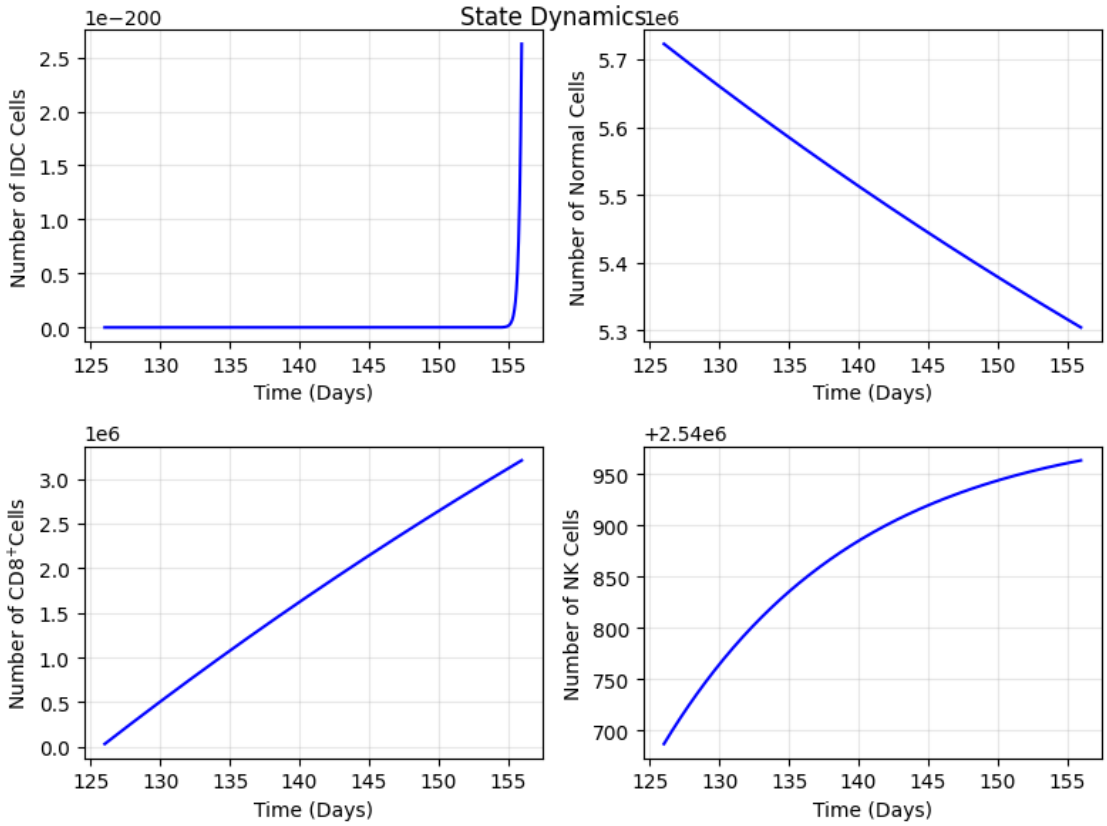


3.1.1 Evolve the state past the end of chemo

[12]:
```
state1 = state_solu.sol(126)
```

[13]:
```
state_sol2 = solve_ivp(cancer_MMK_dynamics, t_span=(126, 156), t_eval=np.
↪linspace(126, 156, 400), y0=state1, max_step=max_step,
                       args=(ode_constants, mmk_constants), dense_output=True)
```

[14]:
```
plot_state_dynamics(state_sol2, t0=126, tf=156, use_grid=True, fig_name="./imgs/
↪state_dynamics_bd1_post.pdf", save_fig=True)
```



3.2 Switching the soft constraints

[15]:
```
#epsilon = 0.01
#test = epsilon + 1
test = True
max_step = 0.1
 126 = 80
 c = 75
 d = 75

state0 = np.array([1e6, 1e7, 1.21e3, 1.815e3])
t0, t1, tf = 0, 126, 156
t_span1 = (t0, t1)
t_span1b = (t1, t0)
costate126 = np.zeros(shape=4)
costate126[0] = -  126 * 1

num_vals = 2500
time1 = np.linspace(t0, t1, num=num_vals)
time1b = np.linspace(t1, t0, num=num_vals)
```

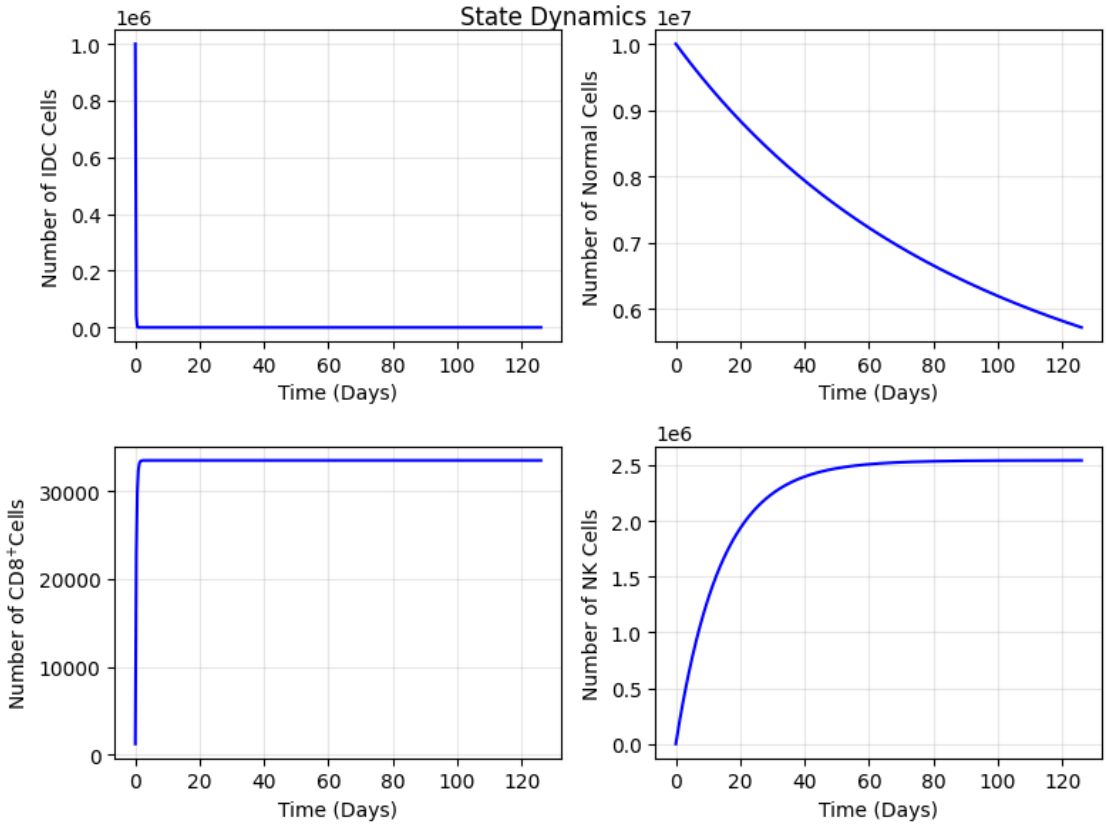
```
control = np.zeros((2, num_vals))                                    # [Doxo,
                                                                     #  Cyclo]

doxo_lb, doxo_hb = 104.55, 127.5
cyclo_lb, cyclo_hb = 1394, 1_700
cyclo_kill_rate = 0.97
doxo_kill_rate = 0.97
cyclo_lower, cyclo_max = np.log(cyclo_kill_rate * cyclo_lb), np.
 ↪log(cyclo_kill_rate * cyclo_hb)
doxo_lower, doxo_max = np.log(doxo_kill_rate * doxo_lb), np.log(doxo_kill_rate␣
 ↪* doxo_hb)
control[0], control[1] = doxo_max, cyclo_max

iterations = 0
max_iters = 5_000
```

[16]:
```
while test and (iterations < max_iters):

    old_control = control.copy()
    control_interp = CubicSpline(time1, y=control, axis=1)

    # Solve state forwards (t0 -> t1) [T N CD NK]
    state_solu = solve_ivp(fun=state_equations, t_span=t_span1, t_eval=time1,␣
 ↪y0=state0, max_step=max_step,
                           args=(control_interp, ode_constants, mmk_constants,␣
 ↪chemo_constants), dense_output=True)

    # Solve costate backward (t1 -> t0)
    costate_solu = solve_ivp(fun=costate_equations, t_span=t_span1b,␣
 ↪y0=costate126, t_eval=time1b, max_step=max_step,
                             args=(state_solu, control_interp, ode_constants,␣
 ↪mmk_constants, chemo_constants), dense_output=True)

    states = state_solu.y
    costates = costate_solu.y[:, ::-1]

     1,  1,  2,  2 = 50, 2, 50, 2
     3,  3,  4,  4 = 50, 2, 50, 2

    def rootu(u):

        u = u.reshape((2, num_vals))

        DH_Du1 = -(4/5) * costates[0] * Ec - costates[2] * bCD * states[2] - 2 *␣
 ↪c * u[1] -\
                    3 *  3 * (u[1] - cyclo_max)**( 3 - 1) -\
```

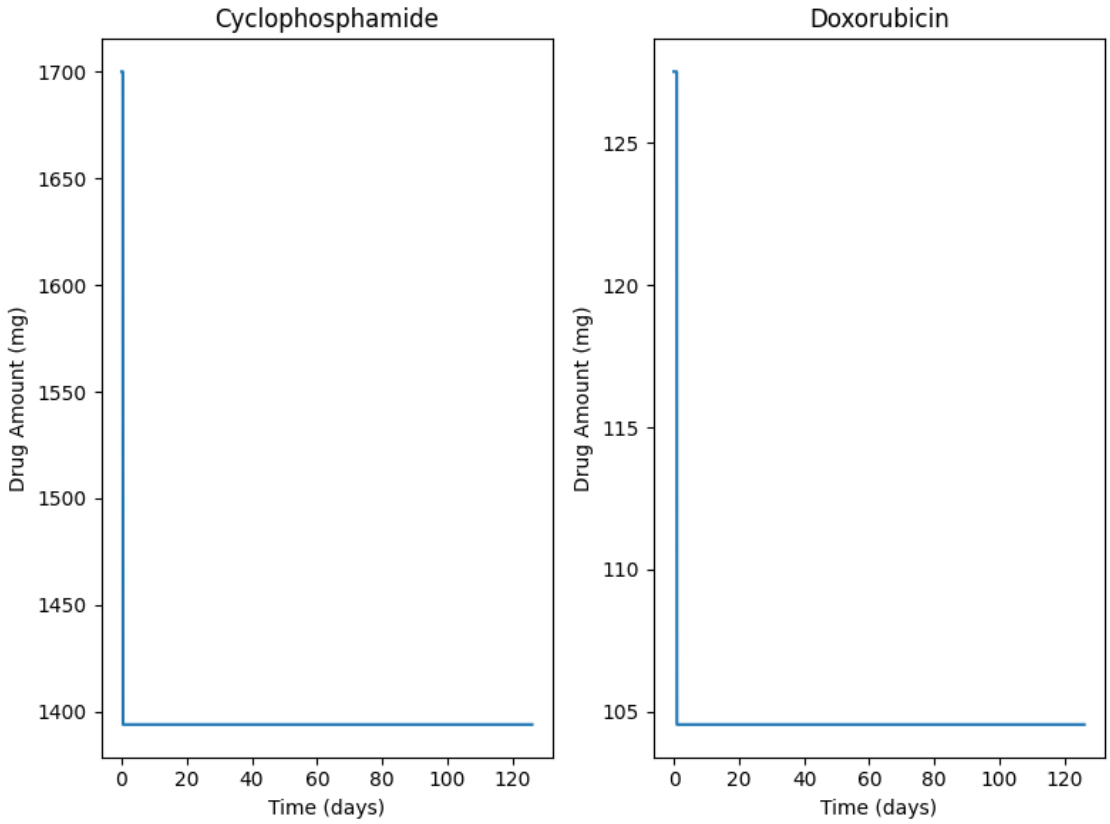
```
                4 *  4 * (u[1] - cyclo_lower)**( 4 - 1)
        DH_Du2 = - Ed * costates[0] * states[0] - 2 *  d * u[0] -\
                   1 *  1 * (u[0] - doxo_max)**( 1 - 1) -\
                   2 *  2 * (u[0] - doxo_lower)**( 2 - 1)

        return (np.array([DH_Du1, DH_Du2])).flatten()

    root_control = root(fun=rootu, x0=control.flatten())
    control = root_control.x
    control = control.reshape((2, num_vals))
    control[0] = np.clip(control[0], doxo_lower, doxo_max)
    control[1] = np.clip(control[1], cyclo_lower, cyclo_max)

    if np.allclose(a=old_control, b=control, rtol=0, atol=1e-2):
        test = False

    display.clear_output(wait=True)
    print(f"Iteration: {iterations}")

    iterations += 1
```

```
Iteration: 1
```

[17]:
```
plot_state_dynamics(state_solu, tf=126, save_fig=True, fig_name='./imgs/
↪state_dynamics_bd2.pdf', figsize=(8, 6), use_grid=True)
```



[18]:
```
plot_control(np.exp(control), save_fig=True, path="./imgs/optimal_control_bdy2.
↪pdf")
```

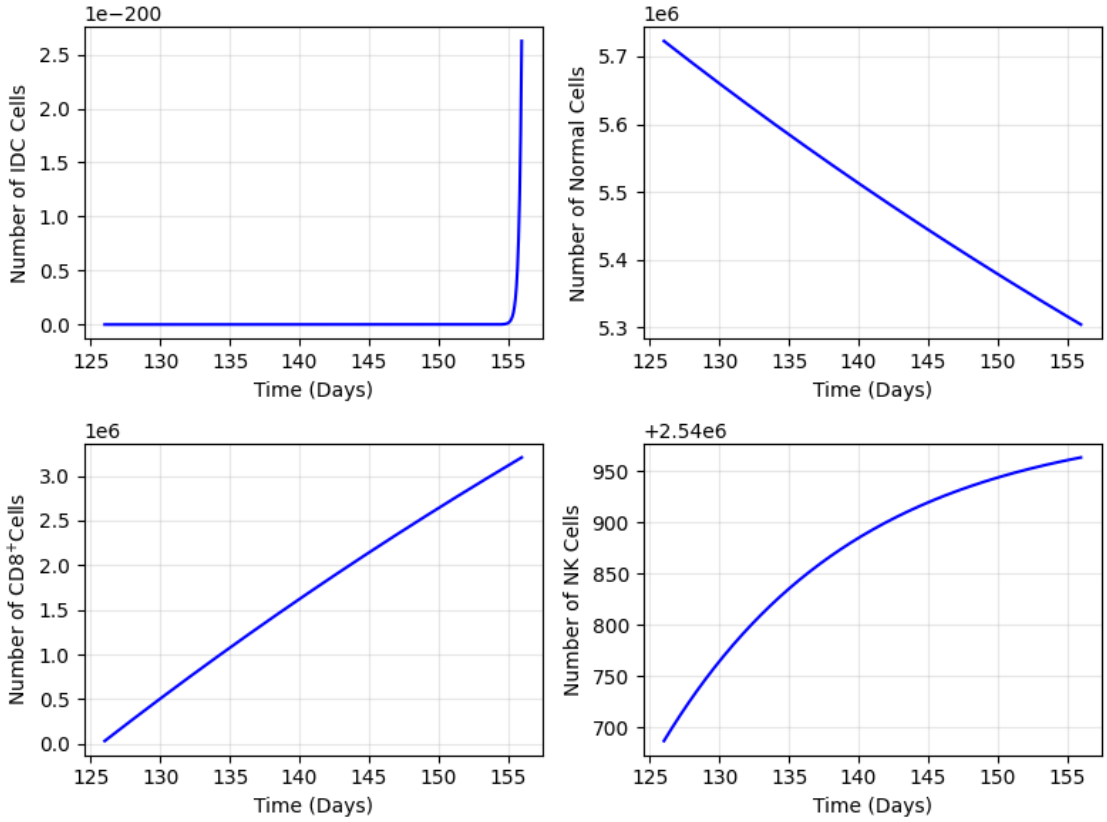


3.2.1 Evolve the state past the chemo

[19]:
```
state1 = state_solu.sol(126)
```

[20]:
```
state_sol2 = solve_ivp(cancer_MMK_dynamics, t_span=(126, 156), t_eval=np.
↪linspace(126, 156, 400), y0=state1, max_step=max_step,
                       args=(ode_constants, mmk_constants), dense_output=True)
```

[21]:
```
plot_state_dynamics(state_sol2, t0=126, tf=156, use_grid=True, title="",␣
↪save_fig=True, fig_name='./imgs/state_dynamics_bdy2_post.pdf')
```



utils.py

[ ]:
```
from numpy.typing import ArrayLike, NDArray
from typing import Callable
import matplotlib.animation as animation
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp

def drug_concentration(time:float|ArrayLike, doses:ArrayLike|list, decay_rate:
 ↪float, duration:int=6,
                       dose_interval:int=21) -> float|ArrayLike:
    """Compute the function of a drug quantity for a given time using the model␣
 ↪chemo
    model proposed by Ophir Nave in
    https://pmc.ncbi.nlm.nih.gov/articles/PMC9065634/ .

    Parameters:
        - time (float or ArrayLike)
        - doses (float) : an array or list containing the the quantities␣
 ↪administered
```

```
                            for the duration of the chemo treatment.
        - decay_rate (float) : the terminal half life of the drug measured in␣
↪(1/days).
        - duration (int) : the duration of chemo treatment which is the total␣
↪number of
                            doses the patient will receive. Defaulted to 6.
        - dose_interval (int) : the number of days between administered doses.␣
↪Defaulted
                                to 21.

    Returns:
        - f_t (float or ArrayLike) : the generated concentration of the drug␣
↪over time.
    """

    time = np.asanyarray(time)
    f_t = np.zeros_like(time, dtype=float)

    for i in range(duration):
        delay = i * dose_interval
        step = np.heaviside(time - delay, 1.0)            # Unit step
        decay = np.exp(-np.log(2)/decay_rate * (time - delay))
        f_t += doses[i] * step * decay

    return f_t

def plot_drugs(time:ArrayLike, drug_vals:dict[str, ArrayLike]|ArrayLike, colors:
↪list[str]=None,
               xlabel='Time (days)', ylabel:str='Concentration␣
↪($\\frac{mg}{mL}$)',
               figtitle:str='Chemotherapy Concentration.') -> None:
    """Plot a generated concentration of one drug or several of them over time.

    Parameters:
        - time (ArrayLike) : the time values at which the drug is measured.
        - drug_vals (dict or ArrayLike) : either an ArrayLike or dictionary␣
↪containing the
                                          values of the drug or drugs to plot␣
↪over time.
                                          If drug_vals is an np.array, it is␣
↪assumed that only
                                          one drug will be plotted. If␣
↪drug_vals is a dictionary,
                                          then several drugs will be plotted.␣
↪In this last case,
```

```
                                        the dictionary must have as keys␣
↪strings denoting the
                                        name of the drugs and the values are␣
↪arrays with the
                                        concentrations of the drugs over time.
        - colors (list) : a string or a list of strings denoting the color or␣
↪colors to use to
                          plot the given drug or drugs. Defaulted to None.
        - xlabel (str) : the label to give the x-axis of the generated plot.␣
↪Defaulted to 'Time (days)'.
        - ylabel (str) : the label to give the y-axis of the generated plot.␣
↪Defaulte to
                         r'Concentration ($\frac{\text{mg}}{\text{mL}}$)'.
        - figtitle (str) : the title to give the generated plot. Defaulted to␣
↪'Chemotherapy Concentration'.

    Returns:
        - None
    """

    if isinstance(drug_vals, dict):
        multiple_drugs = True
    elif isinstance(drug_vals, np.array):
        multiple_drugs = False

    ax = plt.subplot(111)

    if multiple_drugs:

        drugs = list(drug_vals.keys())
        vals = list(drug_vals.values())

        if colors is None:
            for drug, val in zip(drugs, vals):
                ax.plot(time, val, label=drug)
        else:
            for drug, val, color, in zip(drugs, vals, colors):
                ax.plot(time, val, label=drug, color=color)

        ax.set_xlabel(xlabel)
        ax.set_ylabel(ylabel)
        ax.set_title(figtitle)
        ax.legend()

    else:
```

```
        if colors is None:
            ax.plot(time, drug_vals, color=colors)
        else:
            ax.plot(time, drug_vals)

        ax.set_xlabel(xlabel)
        ax.set_ylabel(ylabel)
        ax.set_title(figtitle)

    plt.show()

def make_ani(time:ArrayLike, normalized_states:NDArray, tf:float|int, useGrid:
 ↪bool, title:str,
             xlabel:str) -> None:
    """Function to animate the normalized state dynamics over time.

    Parameters:
        - time (ArrayLike) : the time of the system evolutions.
        - states (NDArray) : the system states evolved through time.
        - tf (float|int) : the final time of the system. Used to set up the␣
 ↪x-limits
        - useGrid (bool) : whether to use a grid or not.
        - title (str) : a str to use to plot the title of the figure.
        - xlabel (str) : the labeling to use for the x-axis.

    Returns:
        - None:
    """

    fig = plt.figure(figsize=(12, 6))
    ax = fig.add_subplot(111)
    colors = ['red', 'blue', 'orange', 'green']
    labels = ['IDC', 'Normal', 'CD8$^+$', 'NK']
    lines = [ax.plot([], [], color=color, label=label)[0] for color, label in␣
 ↪zip(colors, labels)]

    ax.set_xlim(-0.001, tf+0.01)
    ax.set_ylim(0, 1.01)
    ax.set_xlabel(xlabel)
    ax.set_ylabel('% of Population')

    if useGrid:
        ax.grid(True, alpha=0.3)

    def update(idx) -> plt.axes:
        """Function to update the fig object.
```

```
        Parameters:
            - idx (int) : the index to plot on the lines
        Returns:
            - (plt.axes) : the updated line objects
        """

        for line, state in zip(lines, normalized_states):
            line.set_data(time[:idx+1], state[:idx+1])

        return lines

    ax.legend()
    ax.set_title(title)
    animate = animation.FuncAnimation(fig=fig, func=update,␣
↪frames=range(len(time)), interval=50)
    animate.save('./imgs/system_dynamics.mp4')
    plt.close()

def plot_state_dynamics(state_sol:Callable, t0:int=0, t1:int=126, tf:int=156,␣
↪linecolor:str='blue',
                        xscale:str='linear', yscales:
↪list[str]=['linear','linear','linear','linear'],
                        xlabel:str='Time (Days)',
                        ylabels:list[str]=['Number of IDC Cells','Number of␣
↪Normal Cells', 'Number of CD$8^{+}$Cells', 'Number of NK Cells'],
                        title:str='State Dynamics', normalize:bool=False,␣
↪plot_end_chemo:bool=False,
                        use_grid:bool=False, animate:bool=False, save_fig:
↪bool=False,
                        fig_name:str='state_dynamics.pdf', figsize:
↪tuple[int]=(8, 6)) -> None:
    """Plot the state dynamics in 2x2 grid. Plots are arranged in the following
    order (for unnormalized):

    Tumor Normal
    CD    NK

    Parameters:
        - state_sol (Callable) : a callable object from scipy.integrate that␣
↪will
                                 allow the computation of the states.
        - t0 (int) : the starting time for the modeling problem. Defaulted to 0.
                     This corresponds to the lowertime  bound of the cost␣
↪functional.
        - t1 (int) : the upper time bound of the cost functional. Defauled to␣
↪126.
```

```
                    This is used to plot a vertical line to indicate this time.
        - tf (int) : the final time for the modeling problem. Defaulted to 156.␣
↪This
                    can be bigger than the t1. This is used to continue␣
↪evolving
                    the system past t1.
        - linecolor (str) : the line color to use on the plot. Defaulted to␣
↪'blue'.
        - xscale (str) : the scale to use on the x-axis for plotting. Defaulted␣
↪to
                        'linear'.
        - yscales (list) : a list of the scales to use on the y-axis for␣
↪plotting for
                          each generated subplot. Defaulted to
                          ['linear', 'linear', 'linear', 'linear']. Note that␣
↪the
                          first value corresponds to the first state component␣
↪and
                          successively.
        - xlabel (str) : the label to use on the x-axis. Defaulted to 'Time␣
↪(Days)'.
        - ylabels (list) : a list of string containing the labels to use on the␣
↪y-axis
                          of each plot. Note that the first value corresponds␣
↪to the
                          first state component and successively. Defaulted to
                          ['Number of IDC Cells','Number of Normal Cells',
                          'Number of CD$8^{+}$Cells', 'Number of NK Cells'].
        - title (str) : the title to give the overall produced image. Defaulted␣
↪to
                       'State Dynamics'.
        - normalize (bool) : whether to normalize the states or not. Defauled␣
↪to False.
                             If True, only one plot is produced.
        - plot_end_chemo (bool) : whether to plot a vertical line indicating␣
↪the end
                                  of AC treatment. Defaulted to False.
        - use_grid (bool) : whether to use a grid or not. Defaulted to False.
        - save_fig (bool) : a boolean specifying whether or not to save the␣
↪generated
                            figure. Defaulted to False.
        - fig_name (bool) : the name to give the generated figure. Defaulted to
                            './imgs/state_dynamics.pdf'
        - figsize (tuple) : a tuple of two ints that describe the size of the␣
↪image. Defaulted
                            to (8, 6).
```

```
    Returns:
        - None
    """


    if normalize:
        fig, ax = plt.subplots(figsize=figsize)
        time = np.linspace(t0, tf, num=400)
        states = state_sol.sol(time)

        total_pop = states.sum(axis=0)
        states_norm = states / total_pop
        ax.plot(time, states_norm[0], color='red', label='IDC')
        ax.plot(time, states_norm[1], color='blue', label='Normal')
        ax.plot(time, states_norm[2], color='orange', label='CD8$^+$')
        ax.plot(time, states_norm[3], color='green', label='NK')
        ax.set_xlabel(xlabel)
        ax.set_ylabel('% of Population')
        ax.set_title(title)
        ax.legend()

        if use_grid:
            ax.grid(True, alpha=0.3)

        if plot_end_chemo:
            ax.axvline(x=t1, ymin=0, ymax=0, linestyle='dashed',␣
↪color='purple', label='End of Chemo')

        if save_fig:
            plt.savefig(fig_name, format='pdf')


        if animate:
            make_ani(time=time, normalized_states=states_norm, tf=tf,␣
↪useGrid=use_grid, title=title, xlabel=xlabel)

        plt.show()

    else:
        fig, ax = plt.subplots(nrows=2, ncols=2, figsize=figsize)
        time = np.linspace(t0, tf, num=400)
        states = state_sol.sol(time)

        ax[0, 0].plot(time, states[0], color=linecolor)
        ax[0, 0].set_xlabel(xlabel)
        ax[0, 0].set_ylabel(ylabels[0])
```

```
        ax[0, 0].set_xscale(xscale)
        ax[0, 0].set_yscale(yscales[0])

        ax[0, 1].plot(time, states[1], color=linecolor)
        ax[0, 1].set_xlabel(xlabel)
        ax[0, 1].set_ylabel(ylabels[1])
        ax[0, 1].set_xscale(xscale)
        ax[0, 1].set_yscale(yscales[1])

        ax[1, 0].plot(time, states[2], color=linecolor)
        ax[1, 0].set_xlabel(xlabel)
        ax[1, 0].set_ylabel(ylabels[2])
        ax[1, 0].set_xscale(xscale)
        ax[1, 0].set_yscale(yscales[2])

        ax[1, 1].plot(time, states[3], color=linecolor)
        ax[1, 1].set_xlabel(xlabel)
        ax[1, 1].set_ylabel(ylabels[3])
        ax[1, 1].set_xscale(xscale)
        ax[1, 1].set_yscale(yscales[3])

        if plot_end_chemo:
            ax[0, 0].axvline(x=t1, ymin=0, ymax=states[0].max(),␣
 ↪linestyle='dashed', color='red', label='End of Chemo')
            ax[0, 1].axvline(x=t1, ymin=0, ymax=states[1].max(),␣
 ↪linestyle='dashed', color='red', label='End of Chemo')
            ax[1, 0].axvline(x=t1, ymin=0, ymax=states[2].max(),␣
 ↪linestyle='dashed', color='red', label='End of Chemo')
            ax[1, 1].axvline(x=t1, ymin=0, ymax=states[3].max(),␣
 ↪linestyle='dashed', color='red', label='End of Chemo')

        if use_grid:
            ax[0, 0].grid(True, alpha=0.3)
            ax[1, 0].grid(True, alpha=0.3)
            ax[1, 1].grid(True, alpha=0.3)
            ax[0, 1].grid(True, alpha=0.3)

        plt.tight_layout()
        plt.suptitle(title, va='top')

        if save_fig:
            plt.savefig(fig_name, format='pdf')

        plt.show()

def plot_control(u:NDArray, cyclo_kill_rate:float=0.97, doxo_kill_rate:float=0.
 ↪97, t0:int=0, t1:int=126,
```

```
                figsize:tuple[int]=(8, 6), save_fig:bool=False, path:
↪str='optimal_control.pdf') -> None:
    """Plot the given optimal control.

    Parameters:
        - u (NDArray) : the optimal control. Must have shape of (N, 2).
        - cyclo_kill_rate (float) : the kill rate used for cyclophosphamide.
                                    Defaulted to 0.97.
        - doxo_kill_rate (float) : the kill rate used for doxorubicin. Also
                                   defaulted to 0.97.
        - t0 (int) : the starting time of the control. Defaulted to 0.
        - t1 (int): the ending time of the control. Defaulted to 126.
        - figsize (tuple) : a tuple of two ints containing the size of the␣
↪image.
        - save_fig (bool) : whether to save the figure or not. Defaulted to␣
↪False.
        - path (str) : path, containing the name, of where to store the image.
                       Defaulted to 'optimal_control.pdf'

    Returns:
        - None
    """

    time = np.linspace(t0, t1, num=u.shape[1])
    fig, ax = plt.subplots(nrows=1, ncols=2, figsize=figsize)

    ax[0].plot(time, u[1]/cyclo_kill_rate)
    ax[0].set_title('Cyclophosphamide')
    ax[0].set_xlabel('Time (days)')
    ax[0].set_ylabel('Drug Amount (mg)')

    ax[1].plot(time, u[0]/doxo_kill_rate)
    ax[1].set_title('Doxorubicin')
    ax[1].set_xlabel('Time (days)')
    ax[1].set_ylabel('Drug Amount (mg)')

    plt.tight_layout()

    if save_fig:
        plt.savefig(path, format='pdf')

    plt.show()
```