Introduction

The code associated with the research for the paper titled "Improving Mathematical Models of Cancer Through Game-Theoretic Modelling: A Study in Non-Small Cell Cancer" is intricate. This complexity stems from multiple factors, including the contributors' diverse coding styles and experience levels. The extensive use of legacy code further adds to the complexity. The intricacy inherent to the code makes it challenging to comprehend. This document is crafted to elucidate the code more comprehensively.

The primary reason for the numerous branches in the code is due to computational discrepancies identified during the research. These branches were retained for historical transparency but may have individual differences owing to computational variations.

The computational discrepancies predominantly arise from the use of GEKKO, specifically its backend, APMonitor. When GEKKO-based scripts are run on a Windows system, they operate in a 32-bit environment. Conversely, on MacOS or Linux, the scripts run in a 64-bit environment. This leads to differences in data and results contingent upon the system's architecture. For certain analyses, the scripts were specifically executed in a 64-bit environment to align with the required computational settings. Despite these complexities, certain elements like the use of GEKKO and APMonitor remain constant due to their integral role in the research.

Code notes:

The legacy code essentially consists of a large for loop that has been modified over time as necessary. Although it may not be the most optimal or the cleanest solution, the complexity of the code makes it challenging to alter. Therefore, the decision has been made to retain the current code and provide a detailed explanation of it.

The scripts 'Cleanup_script.py' and 'Parallel_GEKKO.py' are integral to our system. 'Parallel_GEKKO.py' is used for parallel processing with the GEKKO library, significantly reducing computation time. This script generates a substantial number of temporary files. To manage these files and prevent system crashes due to space limitations, 'Cleanup_script.py' is employed. It checks the directory for GEKKO's temporary files, removing all but the 15 most recent ones to avoid any potential overflow issues. Ensuring the correct path is set for these scripts is vital for their proper functioning.

These two scripts were created and executed in Ubuntu (Linux). The total group output associated with certain characteristics (Size, trend, increasing or decreasing) gets taken and hardcoded in the necessary functions. The function PR_separate_by_size_chemo_vs_immuno gives the parameters to the corresponding group in other scripts, reducing the computation time for when these parameters are needed again.

FitFunctions_model.py: This script executes the following functions: Exponential, Logistic, Classic Bartalanffy, General Bartalanffy, Gompertz, and General Gompertz. While the script is pretty much straightforward, the output itself is not. Namely, there is something stochastic about the output of the script. Some functions, such as Gompertz, can result in predictions that lead to a negative R^2; this output is not influenced by seed or such. This was determined by executing the script with the same seed and even a different changing seed; however, it leads to either prediction causing a negative R^2 or a positive R^2. The R^2 for that category (being positive or negative) was the same for each time it was in that category. However, the predictions leading to positive R^2 were observed more often. Thus, the predictions were chosen from the iteration from the script that gave predictions that led to a positive R^2. The output of FitFunctions_model.py gets stored in Pickle files, and these Pickle files are then analysed with Result_Heatmap_values.py.

GEKKO_model_Prediction.py: This is the script that uses the parameters created by Parralel_GEKKO.py; however, like the Parralel_GEKKO.py script, this one is heavenly influenced by the environment; for this reason, the script in question is very extensive with multiple switches allowing for more additional control of printing and saving parameters that are redundant but interesting if you want to have a closer look to difference. The switches were also built because this script contains a lot of legacy work; thus, the switches helped me understand the code better. Furthermore, the older parameters in the code (legacy) were questioned because of it being legacy and thus determined with an older version of GEKKO than the current parameters, combined with the uncertainty of the 32-bit and 64-bit environment, was it chosen to redo this for certainty. It is advised to read this script carefully because of the amount of legacy work and the switches leading to it being very complex. The output of this script can be analysed with Result_Heatmap_Gekko.py. However, it is important to mention that this script was intended to work in combination with the old legacy results. Thus, some small tweaks need to be made to replicate the results.

R-squared and mean-squared: The switch to R-squared and mean-squared was done manually by just using the acquired functions.

FirstDim_Simulation.py: This script simulates the trajectory based on the first few points. Some manual changes would be needed to only correspond to a specific arm (aka treatment); furthermore, this script contains a lot of legacy code left in there for transparency reasons. I only made a few small tweaks to this so that all plots were created during the run instead of a select few. This could lead to some unnecessary plots; however, it was more efficient this way.

Alternative_treatment.py and Optimized_and_constant_treatment.py: These two scripts do what their names suggest. Alternative_treatment.py switches the parameters chemo becomes immuno, and Immuno becomes chemo. The optimized_and_constant_treatment.py is about the hypothetical, optimised, and constant treatments. Both scripts need manual attention because of how they are coded; therefore, it is advised to read the script carefully.

Nopars: a variable related to the number of parameters for the function in question; this is needed for some calculations (e.g., Akaike calculation).

The unmentioned scripts primarily consist of functions utilized by other scripts. Although critical for execution, they contain redundant code—some of which is obsolete or was only relevant during the experimental phase. It's worth noting that a few of these scripts have become completely unnecessary in our current context. However, they were integral during the research process for various objectives like understanding concepts, experimenting, or problem-solving. To ensure full transparency in our methods and historical development, these scripts have been included in our documentation.