Radiobiological Modeling with Mathematica®

A Voxel-Based Functional Programming Approach

William Kassing, PhD
University of Cincinnati

Copyright William Kassing 2024

Note: This is just for educational and investigational purposes, not for clinical decision making.

To Jan

Preface

In this book we use *Mathematica* to model the 5Rs of radiobiology at the voxel level using functional programming, where programs are seen as compositions of functions from beginning to end. We also use Python/Pydicom in an imperative manner for input and output tasks such as accessing and manipulating the DICOM RT Dose and RT Structure set files.

We begin by developing a convolution method for calculating the BED in the voxels given any dose-rate input function whether discrete and pulse-like or exponentially varying. We then add to that a repopulation model, a model to account for cell cycle effects, a reoxygenation model, and a model accounting for intrinsic tissue radiosensitivity. Finally, we allow for variation in any of the model parameters using a normal or a lognormal distribution.

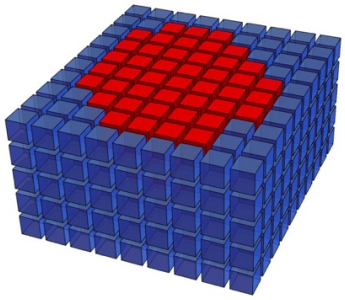
Thus, under one framework, the 5Rs of radiobiology can be modeled at the voxel level, and the functions describing these models can be combined as compositions of functions to account for a wide range of biological effects. The functions developed are fairly simple, fundamental models and could be expanded upon in the future. One of the benefits of functional programming is since everything is seen as a function these functions can be easily modified and expanded. Also, we limit our scope in this book to tumor voxels and do not include normal tissue effects.

We begin by accessing the DICOM RT Dose and RT Structure set files to determine which voxels lie within the contours of interest. We also determine the doses in each of the voxels. Knowing the dose and dose-rate input functions (e.g., conventional fractionation, Gamma Knife radiosurgery, LDR, HDR, brachytherapy, radionuclide therapy, etc.) we can develop dose-rate input functions for each voxel. Then a convolution between these absorbed dose-rate functions and the DNA repair functions gives the BED in each voxel. This is the starting point to which the other 4Rs of radiobiology (repopulation, redistribution, reoxygenation, radiosensitivity) can be included as well. The effects on the tumor TCP can be explored under variations of these conditions. Finally, the BEDs calculated in *Mathematica* can be exported back into the DICOM RT Dose file to display the BED on the treatment planning computer.

The purpose of this book is to introduce functional programming as a method to use for problems of this type. Functional programming has some unique capabilities not seen in traditional imperative languages, and "functional thinking" in general comes naturally to those in the medical physics community. Some of the commands may be new (e.g., Map, Nest, Fold, Part, Apply, and so on) and the list structure for representing arrays may seem unfamiliar at first, but when one compares the size of the resulting programs to those of a comparable imperative programs, the efficiency is apparent.

Once within *Mathematica,* everything is seen as a function, and as we will see functions in functional programming are treated as first-class citizens, meaning they can be passed around as arguments, returned from other functions, and assigned to variables, which allows for the creation of higher-order functions, which take one or more functions as arguments. We will see that this is very useful in creating higher-order composite functions.

One of the datasets that is used in this book as an example is an 11x10x9 set of voxels. A region through those voxels is shown below. In red are tumor voxels while dark blue represents normal tissue. Within each voxel we will know the absorbed dose-rate input function (from the treatment planning computer) along with many other characteristics of the voxels. Some of the voxels may be repopulating, some may be distributed through various phases of the cell cycle while others may be more stationary, and some of the of the voxels may be more hypoxic than neighboring voxels.



These are some of the things we can investigate using this functional approach. And the process is dynamic as well, with the voxel characteristics changing over time, the tumor either growing or shrinking with time.

This book represents a modest attempt to explore radiobiological modeling for investigational and educational purposes using techniques of functional programming. Much could be added to improve upon the models. It is hoped that some investigators or students may find some of the methods useful and may even contribute to their further development. Some may find it useful to see how DICOM files are manipulated and processed as in the Appendices.

Mathematica and other files from this book can be found at www.kassing.com.

William Kassing, 2024

Acknowledgements

I would like to thank Dr. Ronald Warnick and Mr. John Thaman for providing anonymized data files of actual Gamma Knife radiosurgery cases that were used in developing the methodology described in this book. Some of this data and the results obtained from it are shown in the book including the figure on the cover.

I also thank Dr. Derek Moyer and Dr. Michael Platt, former graduate students at the University of Cincinnati, for providing assistance with Pydicom Python programming.

Before his untimely death, Dr. Howard Elson was actively involved in this project and offered valuable advice and encouragement that was much appreciated and is greatly missed.

Finally, I would like to extend my gratitude to the following individuals who have, in various ways, contributed to the creation of this book: Albert Balcells, Richard Bozian, Roy Eckart, Richard Gass, Dan Ionascu, Randy Janke, Robert Janke, Michael Lamba, Frank Pinski, Ronald Sachs, Henry Spitz, Stephen Thomas, David Roesener, Norma Wagoner, and former colleagues and students at the Barrett Cancer Center in Cincinnati.

About the Author

Dr. William Kassing retired from the University of Cincinnati Medical Center as a Senior Medical Physicist in 2021. He is currently Adjunct Associate Professor in the Department of Radiation Oncology at the University of Cincinnati where he has taught radiation biology for over 25 years.

Table of Contents

Appendices

1. Introduction

Technological advances in radiation therapy in recent years have made it possible to deliver radiation dose to a patient with a higher level of accuracy and precision than ever before. Ongoing developments and refinements in radiation dose delivery, image guidance, and motion management have been responsible for these advances, allowing for a high level of control of the radiation dose distribution in a patient. With such impressive advances in dose delivery, many investigators have set their sights on another frontier in radiotherapy, the biological frontier.

Biological information obtained from individual patients through functional or molecular imaging studies, or by using predictive assays and other biomarkers, gives the promise of more individualized or patient-specific radiation treatments. This is the goal of personalized or precision medicine in general and will be an ongoing challenge and active area of investigation in the field of radiation therapy.

Patient-specific information that can be used to better characterize an individual's radiation response include that of tumor and normal tissue radiosensitivity, DNA repair rates and kinetics, proliferative (tissue repopulation) response characteristics, tissue oxygen concentration, and tumor burden (clonogen density). Using this and other biological information, radiobiological models of tumor control probability (TCP) and normal tissue complication probability (NTCP) can be personalized allowing radiation treatments to be better tailored to individual patients.

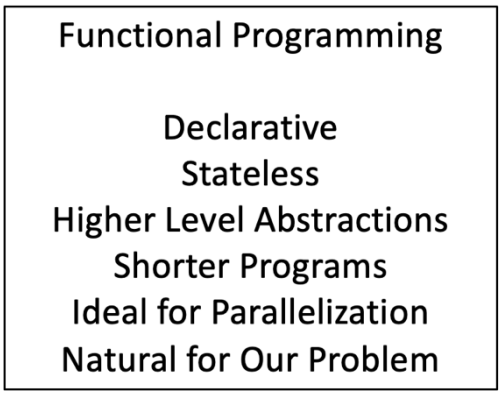
This approach has been variously referred to as biologically optimized treatment planning, biologically conformal radiotherapy, theragnostic imaging and dose painting, and biologically guided radiation therapy (BGRT). Guided by biological information at the voxel level, customized treatment plans with non-uniform physical dose distributions can be generated that yield improved biological dose distributions. Such an approach can lead to patient-specific TCP maps and (much more challenging to model) NTCP maps which give the potential for improving the therapeutic ratio in individual patients.

This emerging area of investigation has promise to benefit radiotherapy patients in the future. In this book we explore biological modeling at the voxel level using functional programming with *Mathematica*. We model the 5Rs of radiobiology and show that functional programming is a style of computer programming that provides a natural, powerful, and elegant approach for use in investigations of this type.

2. Functional Programming

Functional programming is not as well-known as the more traditional and mainstream imperative programming, but it has been growing in popularity in recent years and this trend is likely to continue. Functional programming is fascinating in its own right, its foundations in the lambda calculus played an important role in the history of computer science, and it has unique characteristics that make it a powerful programming paradigm for certain types of problem solving, including those explored in this book. Appendix A has a more detailed explanation of functional programming, here we give just a brief overview emphasizing those aspects that are most useful in voxel-based radiobiological modeling.

Functional programming is essentially mathematical programming, using functions and functional thinking to accomplish our goal. This is a very natural type of programming for those in the medical physics community where functions are common and familiar. We translate what we want to accomplish into a composition of mathematical functions in a declarative manner (rather than imperatively; see Appendix A) producing functional programs that are both elegant and powerful, and often more concise and easier to read than comparable imperative programs.



Important Properties of Functional Languages

Functions in functional programming are treated as first-class citizens, meaning they can be passed around as arguments, returned from other functions, and assigned to variables. This allows for the creation of higher-order functions, which take one or more functions as arguments. An example of a higher-order function is the Map function described below. It takes as arguments a given function and a collection of elements (which in *Mathematica* is a list) and returns a new list with the function applied to each element of the list. The Map function is both a higher level abstraction and a declarative command. With this function we declare to the program that we want to apply a mapping of a function over a list of data, and we leave the details of the implementation (such as explicit looping) to the program. Shorter and easier to read programs result, and sometimes what would take many lines of code in an imperative language can be done with a single line of code (called a one-liner) in a functional language.

Another benefit of functional programming is that the internal state of the system does not change during the computation, which implies that there are no side effects as seen in imperative

programs, as well as no strict evaluation order of the functions making up the overall program. For these reasons, functional programs are more easily parallelized for multicore computations than are imperative programs, which is a major reason for the increase in popularity of functional programming. In this book, a variant of the Map function called ParallelMap is used which automatically distributes the computation among the available kernels and processors of the computer being used. With large datasets containing many voxels, it is this ability to easily distribute the computation over multiple processors that makes voxel-based radiobiological modeling feasible.

Higher-order functions, such as Map and ParallelMap, as we have seen, accept other functions as arguments and hide the details of the machinery of the underlying algorithms that accomplish their tasks. Other very general and useful higher-order functions that are used in this book include Flatten, Partition, Apply, Part, Select, Nest, NestList, Fold, and FoldList. Thinking about a problem in terms of functions such as these (i.e., functional thinking) allows us to approach a problem at a higher level of abstraction than we normally would when we program with traditional imperative languages. *Mathematica* includes all the well-known mathematical functions as well, and these functions can be used in constructing specialized radiobiological modeling functions. Mathematical functions used in this book include Integrate, Exp, UnitStep, RandomChoice, RandomVariate, NormalDistribution, and LogNormalDistribution.

We want to map radiobiological functions over the voxels of interest taking into account the specific characteristics of the individual voxels. The voxels themselves are represented as elements in a list. Lists are the primary data structures in functional languages, and nested lists are used to represent the arrays of imperative languages. Lists are actually a special type of function themselves, since in functional programming everything is seen as a function, even if just a constant function. In *Mathematica*, a collection of objects is grouped together in a list by using the List function

List[*voxel1*, *voxel2*, *voxel3*, …]

or equivalently by enclosing the objects in curly brackets

{*voxel1*, *voxel2*, *voxel3*, …}

where *voxel1*, *voxel2*, …, are lists themselves, containing information about the individual voxels, such as the dose-rate input function, the DNA repair rate function, voxel type including the α/β ratio, radiosensitivity, proliferative capacity, cell cycle phase, tissue oxygen concentration, and other characteristics used in the radiobiological modeling. The voxels above are grouped together in a one-dimensional list, but they can also be grouped together in a higher-dimensional nested list representing the slices, rows, and columns of the medical imaging dataset. In *Mathematica*, the functions Partition and Flatten are used to convert back and forth between one-dimensional and multi-dimensional lists.

Radiobiological functions can be mapped over these lists in a very efficient manner using the Map function introduced above. The complete list data structure is passed into the Map function as an argument and the radiobiological function of interest is applied to all of the elements in the list and an updated list is returned containing the transformed values. For example, here we show the mapping of a general radiobiological function called RadiobiologicalFunction over the

voxels to produce the biologically effective dose (BED) in each voxel. We give the voxel list data structure the name *Voxels*

$$Voxels = \{voxel1, voxel2, voxel3, \ldots\}$$

and we map the radiobiological function over this list as follows

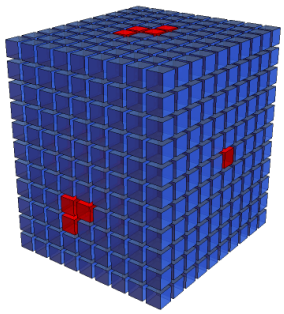
$$\text{Map[RadiobiologicalFunction[}Voxels\text{]]}$$

to produce another list containing the voxel BEDs which we call *VoxelBEDs*

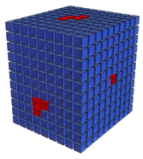
$$VoxelBEDs = \{BED1, BED2, BED3, \ldots\}$$

These BED values can be modified further, for example, by converting them into TCPs as demonstrated later. The mapping is performed in a functional manner, where there is simply an input list of voxels containing information about each voxel, the evaluation of a composite function over this list, and the resulting output list of BED values in each voxel.

A graphical representation of an imaging dataset that will be used later in this book from a Gamma Knife radiosurgery case consisting of 990 voxels is shown below. The red voxels are tumor and the blue voxels are surrounding normal tissue.



In essence, what we will be doing is mapping a RadiobiologicalFunction over these voxels as shown here

Map[RadiobiologicalFunction, ]

where the RadiobiologicalFunction is itself a composition of other functions. That is,

$$\text{RadiobiologicalFunction} = \text{Function[Function[Function[Function}\ldots\text{]]]}.$$

This is functional programming in a nutshell, the program is simply a composition of functions from beginning to end.

A graphic from another imaging dataset of a larger tumor is shown below in Figure 1. This shows some hypothetical examples of what can be done with functional programming. Tumor voxels are shown in red, while normal tissue voxels are shown in dark blue. Also displayed are hypothetical cases showing hypoxic voxels (Fig. 1b), voxels containing proliferating cells (Fig. 1c), and voxels with a dose deficit (Fig. 1d), demonstrating various clinical situations that can be investigated at the voxel level by mapping functions over the voxels. These regions could grow or shrink with time as well.

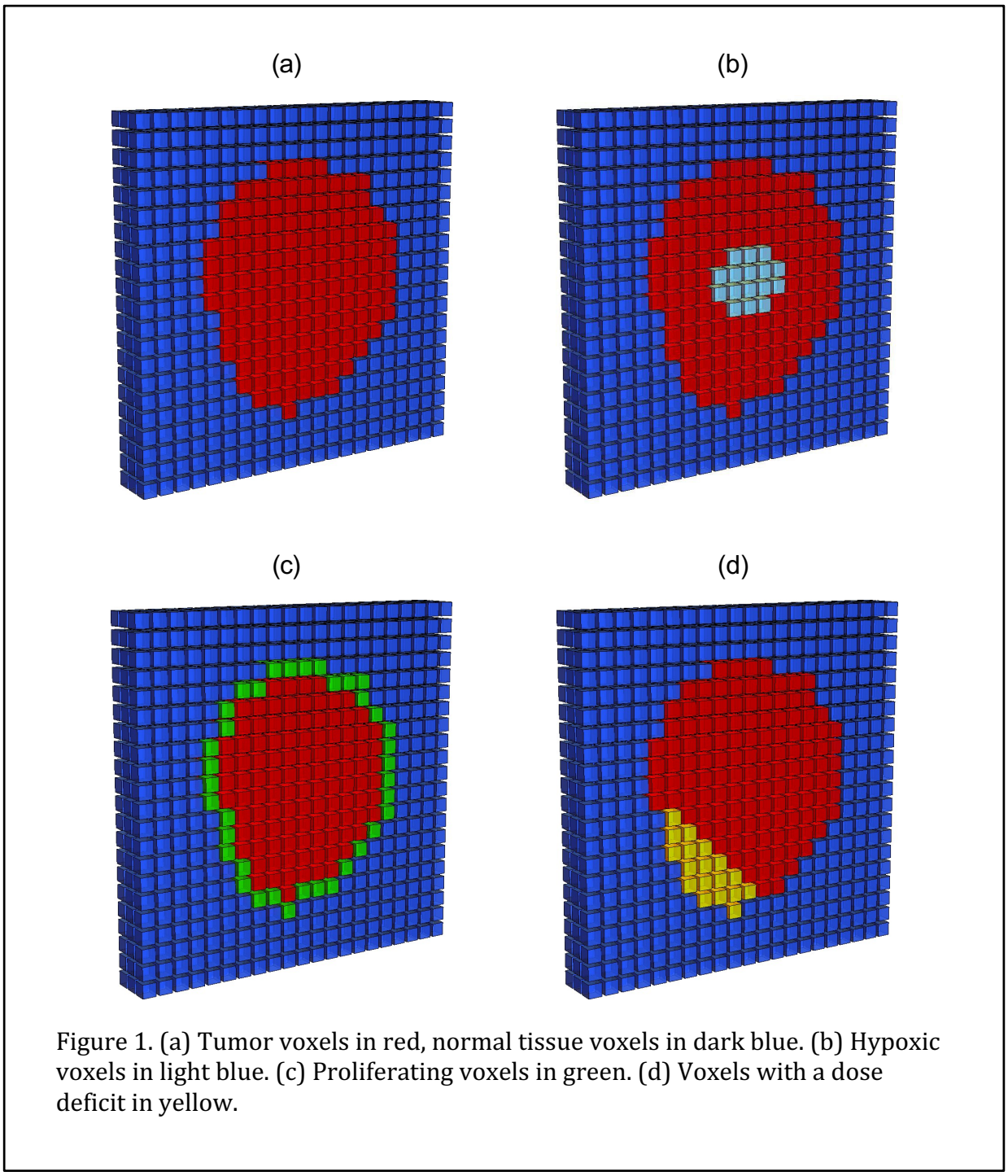


Figure 1. (a) Tumor voxels in red, normal tissue voxels in dark blue. (b) Hypoxic voxels in light blue. (c) Proliferating voxels in green. (d) Voxels with a dose deficit in yellow.

3. Voxel-Based Radiobiological Modeling

Our goal is to map radiobiological functions over the voxels in a radiotherapy treatment plan. We can do so in both tumor and normal tissue voxels for a variety of purposes. In this book we limit our scope to tumor voxels, calculating the BED in the individual voxels, and from these values and an estimate of the clonogen number per voxel, we calculate the TCP in the voxels and thence in the tumor as a whole. We develop models for the 5Rs of radiobiology (repair, repopulation, redistribution, reoxygenation, radiosensitivity) and we investigate their effects on the TCP of the tumor. We also allow for variation in any of the parameters used in the modeling, accounting for heterogeneity seen in patient populations.

3.1 The Linear Quadratic Model

To model cell survival, we begin with the well-known and widely used linear quadratic model, where for a single instantaneous dose of radiation D, the cell surviving fraction (SF) is given by

$$SF = \exp(-\alpha D - \beta D^2) \tag{1}$$

where α and β are tissue specific radiosensitivity parameters. A mechanistic basis for the linear quadratic model has been proposed, and though certainly only an approximation of the true mechanism involved, it is a conceptually useful tool. In this model, DNA damage leading to double-strand breaks and thence cell killing can occur by a single-hit (or single track or cluster) of radiation, or by a double-hit (or double track or cluster) of radiation. The single-hit lesions are always lethal in this model, while the double-hit lesions may be lethal or may be repaired, depending on the rate and time course of the irradiation (Bodey et al. 2004).

Single-hit DNA lesions are governed by the α term of the linear quadratic equation and are considered irrepairable. These lethal lesions are often thought of as double strand DNA breaks that lead to chromosomal aberrations (dicentrics, rings, and anaphase bridges) which lead to cell death when the cell attempts mitosis (mitotic catastrophe). Radiation induced apoptosis can lead to cell death as well, the amount dependent on the tumor and tissue type, and this can also be modeled, but will not be considered in this book. The α parameter gives the natural logs of irrepairable cell kill per unit dose of radiation. Double-hit DNA lesions are governed by the β term of the linear quadratic equation and are considered repairable. These lesions are thought of as sublethal DNA breaks, and depending on the dose-rate and time course of the irradiation, may be either correctly repaired or converted into lethal damage, as discussed below. The β parameter gives the natural logs of repairable cell kill per unit dose squared of radiation. Two units of dose are involved because this component of cell killing is made up of the interaction of two different particles (or tracks or clusters) of radiation (Brenner 2008, Brenner et al. 1998).

Taking the natural logarithm of both sides of the linear quadratic equation and rearranging gives

$$-\ln SF = \alpha D + \beta D^2 \tag{2}$$

The term $-\ln SF$ is called the logarithmic cell kill or biological effect E. The larger the biological effect, the more cells are killed by the radiation. Dividing the biological effect by the radiosensitivity parameter α gives

$$-\frac{\ln SF}{\alpha} = D + \frac{D^2}{\alpha/\beta} \tag{3}$$

The term on the left-hand side of this equation is defined as the biologically effective dose (BED), giving

$$BED = D + \frac{D^2}{\alpha/\beta} \tag{4}$$

or

$$BED = D\left[1 + \frac{D}{(\alpha/\beta)}\right] \tag{5}$$

for a single instantaneous dose of radiation. In the formulation of BED in equation (4) and in those that follow below, the BED is seen to be made up of two terms. The first term is the total physical dose and the second term involves the interaction of two units of dose and is modified by the α/β value. It is this second term that gives rise to fractionation and protraction effects, and tissues with lower α/β values are influenced more by these effects than are tissues with higher α/β values. This fact gives rise to the well-known phenomenon of late responding normal tissue, with a lower α/β value, being influenced more by fractionation or protraction (i.e., lengthening the treatment) than are tumor or early responding normal tissue with a higher α/β value. In equation (5) the BED is written in terms of the total dose multiplied by what is called the relative effectiveness (RE). The relative effectiveness is a measure of how "effective" the treatment is, again, with tissues with a lower α/β value giving rise to higher BED values than tissues with a higher α/β value, the RE therefore being higher for those tissues with lower α/β values.

If instead of giving one single dose D of radiation, we give n instantaneous doses of size d, with time for full repair of sublethal damage between fractions, the initial part of the cell survival curve is repeated with each fraction, and the survival fraction can be written

$$SF = [\exp(-\alpha d - \beta d^2)]^n \tag{6}$$

or

$$SF = \exp[n(-\alpha d - \beta d^2)] \tag{7}$$

Taking the natural logarithm of both sides of the equation, rearranging and dividing by the radiosensitivity parameter α gives

$$-\frac{\ln SF}{\alpha} \equiv BED = nd + \frac{nd^2}{(\alpha/\beta)} \tag{8}$$

or

$$BED = nd\left[1 + \frac{d}{(\alpha/\beta)}\right] \tag{9}$$

which is the familiar equation for BED for fractionated radiation. This equation is applicable for instantaneous doses *d* of equal size with full repair of sublethal damage between doses.

A more general BED equation where the size of the dose per fraction may vary fraction to fraction is given by

$$BED = \sum_{k=1}^{n} d_k + \frac{1}{\alpha/\beta} \sum_{k=1}^{n} d_k^2 \tag{10}$$

Here *n* instantaneous fractions are given, with sufficient time between fractions for complete repair of sublethal damage.

These equations for BED for fractionated dose delivery rely on the assumptions that the fraction durations are short and the inter-fraction times are long, compared to the rate of repair. This assures that the repair during the fraction is negligible and that the repair between fractions is complete. If these assumptions do not hold, then incomplete repair during treatment will alter the BED from that predicted by the above formulas.

3.2 Repair Model

A common way to interpret incomplete repair during treatment is with the binary misrepair model, where there are competing processes of correct repair of the DNA lesions and the incorrect interaction (misrepair) of the two (binary) lesions leading to lethality. As the dose-rate is lowered or when fractionating the dose delivery, the two independent radiation induced lesions are likely to occur at different times during irradiation, allowing for repair of the first DNA lesion before it can undergo binary misrepair with the second lesion. This phenomenon gives rise to the fractionation and protraction effects of the linear quadratic model (Brenner 2008).

Lea and Catcheside (1942) proposed a dimensionless time-protraction function, called *G*, to account for incomplete repair during treatment. In this model, the survival fraction is written as

$$SF = \exp(-\alpha D - \beta G D^2) \tag{11}$$

and the biological effect is given by

$$-\ln SF = \alpha D + \beta G D^2 \tag{12}$$

where

$$G = 2/D_T^2 \int_0^T \dot{D}(t)dt \int_0^t \dot{D}(w)\, e^{-\mu(t-w)}dw \tag{13}$$

In this equation, D_T is the total dose, $\dot{D}(t)$ is a function describing the variation in dose-rate over the entire course of the treatment, and μ is the repair rate constant. The term after the second integral sign can be thought of as the first of a pair of DNA lesions, which decays away exponentially with rate constant μ, while the term after the first integral sign refers to the second

DNA lesion, which can interact with what remains of the first lesion after repair (Brenner 2008). G acts only on the quadratic, repairable part of the linear quadratic equation.

The BED equation (5) can be written as

$$BED(T) = D_T \left[1 + \frac{D_T \cdot G(T)}{(\alpha/\beta)}\right] \tag{14}$$

or

$$BED(T) = D_T + \frac{D_T[D_T \cdot G(T)]}{(\alpha/\beta)} \tag{15}$$

where D_T is the total dose, and $G(T)$ can be thought of as the fraction of primary lesions that remain to interact with secondary lesions over the course of the treatment.

G takes on values from 0 to 1. If G = 0, we have full repair of sublethal damage, there is no interaction of lesions, and

$$BED(T) = D_T \tag{16}$$

In this case, the BED is equal to the total physical dose. If G = 1, we have no repair of sublethal damage, there is complete interaction and misrepair of the lesions, and

$$BED(T) = D_T + \frac{D_T^2}{\alpha/\beta} \tag{17}$$

which is equivalent to the BED for a single instantaneous dose of radiation. The case with incomplete repair has a G value somewhere between these two extremes.

Gustafsson et al. (2013) showed that the equation for BED can be written very generally as

$$BED(T) = \int_0^\infty \dot{D}(t)\mathrm{d}t + \frac{2}{\alpha/\beta}\int_0^\infty \dot{D}(t)[\dot{D}(t) \otimes R(t)]\mathrm{d}t \tag{18}$$

where the final term includes a convolution of the dose-rate input function $\dot{D}(t)$ with the repair function $R(t)$. (Note: Gustafsson uses $R_T(t)$ and $I(t)$ where we use $\dot{D}(t)$ and $R(t)$, respectively.) In this formulation of BED, the repair function is not limited to an exponential function, and can be very general.

This equation can also be written as

$$BED(T) = D_T + \frac{2}{\alpha/\beta}\int_0^T \dot{D}(t)[\dot{D}(t) \otimes R(t)]\mathrm{d}t \tag{19}$$

where T is the total treatment time, D_T is the total physical dose, and the second term of the equation depends on how much DNA repair occurs during treatment, which can be no repair, full repair, or incomplete repair, depending on the time course of the treatment.

Millar and Canney (1993) showed that when the repair function is an exponential, as is the case in this book, the BED equation can be written as

$$BED = D_T + \frac{1}{\alpha/\beta}[\psi(\Xi, \mu)] \tag{20}$$

where

$$\psi(\Xi, \mu) = 2\int_0^T \dot{D}(t)dt \int_0^t \dot{D}(w)\, e^{-\mu(t-w)}\mathrm{d}w \tag{21}$$

The symbol Ξ signifies that the integral depends on the specifics of the treatment, that is, it depends on the total irradiation protocol including, for example, the fractional dose, start time of the fraction, dose rate, etc. This treatment specific protocol is described by the absorbed dose rate function, $\dot{D}(t)$, and the symbol μ is again the sublethal DNA repair rate constant.

Millar and Canney (1993) also showed that for fractionated protracted irradiation, and assuming an exponential repair function, the psi term can be written

$$\psi(\Xi, \mu) = 2\sum_{j=1}^{N}\int_{t_j}^{t_j+\delta t_j} \dot{D}_j\, e^{-\mu t}\mathrm{d}t \times \left[\int_{t_j}^{t} \dot{D}_j\, e^{\mu w}\mathrm{d}w + \sum_{i=1}^{i=j-1}\int_{t_i}^{t_i+\delta t_i} \dot{D}_\iota\, e^{\mu w}\mathrm{d}w\right] \tag{22}$$

where N is the number of fractions, each starting at time t_j and having duration $t_j + \delta t_j$, and where δt_j is such that $t_j + \delta t_j \leq t_{j+1}$. The final term in this equation represents the contributions due to incomplete repair from previous fractions and does not contribute to the first fraction when j = 1. This is the form of the equation we implement in our functional program. It handles any fractionated or protracted dose-rate input function that can be written as a mathematical function. With N = 1, this equation reduces to the form used for continuous radiation, for example, as in brachytherapy or radionuclide therapy. Figure 2 shows examples of dose-rate input functions that can be implemented in this equation for psi.

For the case of a bi-exponential repair function, with a fast and a slow component of repair (see Figure 3a), there will be two psi terms, $\psi_1$and ψ_2, corresponding to the two repair rate constants $\mu_1$and μ_2. In this case, the BED equation is written

$$BED = D_T + \frac{1}{\alpha/\beta}[\mathrm{a}\psi_1(\Xi, \mu_1) + \mathrm{b}\psi_2(\Xi, \mu_2)] \tag{23}$$

where a and b are the fractional contributions of the fast and slow components of repair, with a+b=1. This equation is often written using a partition coefficient c as follows

$$BED = D_T + \frac{1}{\alpha/\beta}\left[\frac{\psi_1(\Xi, \mu_1) + \mathrm{c}\psi_2(\Xi, \mu_2)}{1 + \mathrm{c}}\right] \tag{24}$$

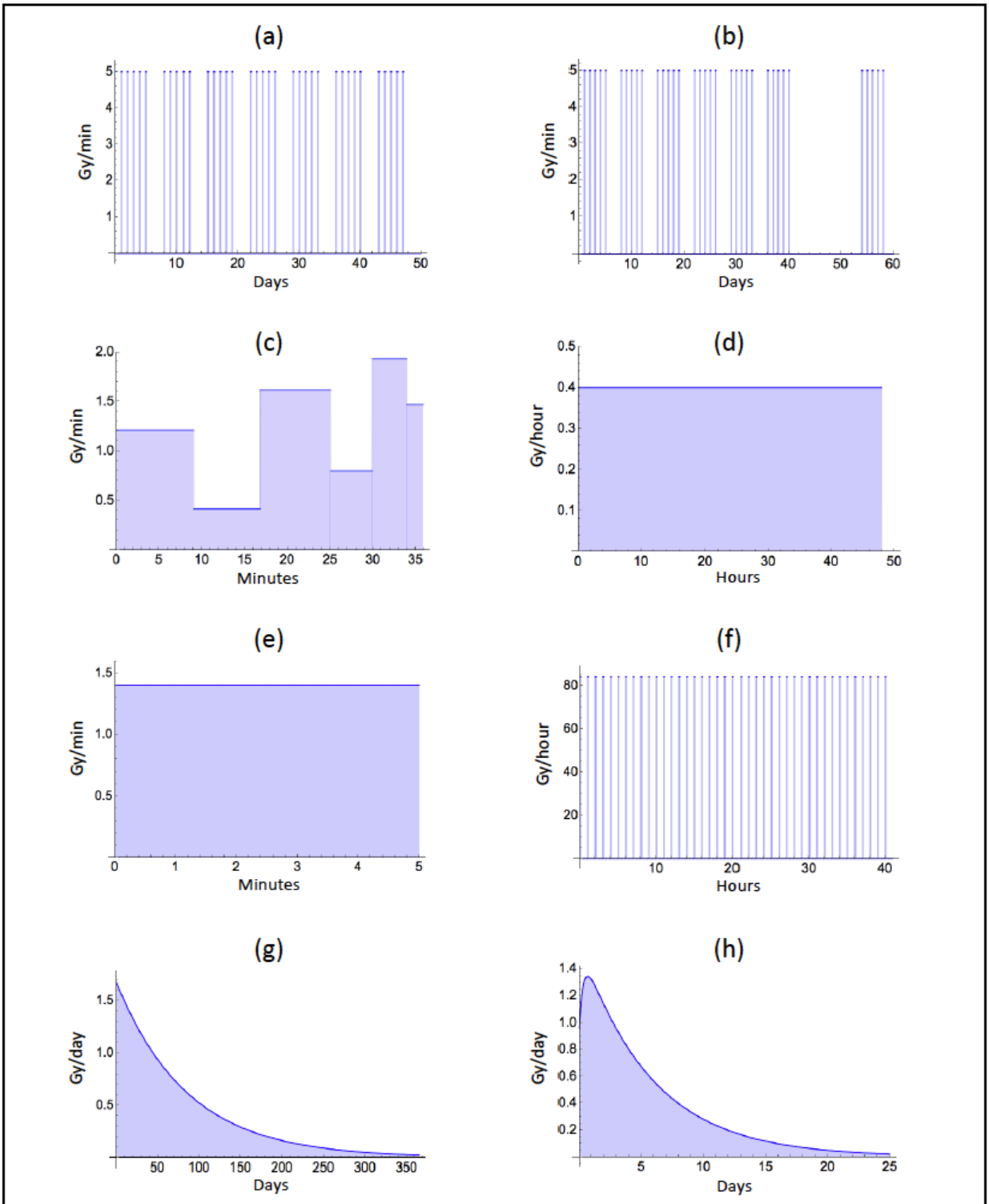


Figure 2. Dose-rate input functions. (a) Conventional fractionation with weekend breaks. (b) Conventional fractionation with weekend breaks and a two-week break. (c) Gamma Knife treatment with six shots. (d) Low dose-rate treatment. (e) High dose-rate treatment. (f) Pulsed dose-rate treatment. (g) I-125 permanent implant treatment. (h) Radionuclide treatment.

The BED equations above are very general and can be used with external beam radiotherapy, brachytherapy, radionuclide therapy, or any combination of these. As shown in section 3.8, from the BED in the voxels, and an estimate of the number of clonogens per voxel N, the surviving clonogen number is given by

$$Surviving\ Clonogens = N \cdot \exp(-\alpha \cdot BED) \tag{25}$$

and the TCP by

$$TCP = \exp\big(-N \cdot \exp(-\alpha \cdot BED)\big). \tag{26}$$

3.3 Repopulation Model

Tumors may repopulate (proliferate) during treatment decreasing the BED in the voxels where this occurs. Given an initial number of clonogenic cells N_0, if the effective doubling time of these cells is T_{eff}, then at time t there will have been t/T_{eff} cell doublings and the number of cells as a function of time can be written as

$$N(t) = N_0 2^{t/T_{\text{eff}}} \tag{27}$$

and the fractional increase in the cell population with time is given by

$$\frac{N(t)}{N_0} = 2^{t/T_{\text{eff}}} \tag{28}$$

Noting that

$$2^{t/T_{\text{eff}}} = \exp\left(\ln 2 \frac{t}{T_{\text{eff}}}\right) \tag{29}$$

the survival fraction equation (11), including the effects of repopulation, can be written as

$$SF = \exp(-\alpha D - \beta G D^2) \cdot \exp\left(\ln 2 \frac{t}{T_{\text{eff}}}\right) \tag{30}$$

and the BED equation (14) above, including the effects of repopulation, can now be written as

$$BED(T) = D_T + \frac{D_T[D_T \cdot G(T)]}{(\alpha/\beta)} - \frac{\ln 2 \cdot T}{\alpha \cdot T_{\text{eff}}} \tag{31}$$

or equivalently, using BED equation (20) above, as

$$BED(T) = D_T + \frac{1}{\alpha/\beta}[\psi(\Xi, \mu)] - \frac{\ln 2 \cdot T}{\alpha \cdot T_{\text{eff}}} \tag{32}$$

where

$$\psi(\Xi,\mu) = 2\int_0^T \dot{D}(t)dt \int_0^t \dot{D}(w)\, e^{-\mu(t-w)}\mathrm{d}w \tag{33}$$

The equations above assume that T_{eff} is a constant. T_{eff} will generally be a function of time and is given by the equation

$$T_{\text{eff}}(t) = \frac{T_{\text{c}}}{GF \cdot (1-\phi(t))} \tag{34}$$

where T_{c} is the cell cycle time, GF is the growth fraction (the fraction of cells in cycle), and $\phi(t)$ is the cell loss factor. Here we assume that T_{c} and GF are constants, while $\phi(t)$ is a function of time as described below. If the cell loss $\phi(t)$ is zero, T_{eff} becomes the potential doubling time T_{pot}, which is defined as the cell cycle time divided by the growth fraction. Thus, T_{eff} can be written as

$$T_{\text{eff}}(t) = \frac{T_{\text{pot}}}{(1-\phi(t))} \tag{35}$$

where

$$T_{\text{pot}} = \frac{T_{\text{c}}}{GF} \tag{36}$$

and

$$\phi(t) = \phi(0)\exp(-\nu \cdot t) \tag{37}$$

Here $\phi(0)$ is the pretreatment cell loss factor and ν is the cell loss rate constant. It is thought that cell loss decreases during treatment because of increased oxygenation of the tumor due to improved tissue perfusion as the treatment proceeds (Joiner and van der Kogel (2018). Therefore

$$T_{\text{eff}}(t) = \frac{T_{\text{pot}}}{(1-\phi(0)\exp(-\nu \cdot t))} \tag{38}$$

Since $T_{\text{eff}}(t)$ can be a function of time, the term for repopulation in the above BED equations, in its most general form, is written as an integral

$$BED(T) = D_T + \frac{1}{\alpha/\beta}[\psi(\Xi,\mu)] - \int_0^T \frac{\ln 2}{\alpha \cdot T_{\text{eff}}(t)}\mathrm{d}t \tag{39}$$

and when substituting the equation for $T_{\text{eff}}(t)$ above we get

$$BED(T) = D_T + \frac{1}{\alpha/\beta}[\psi(\Xi,\mu)] - \int_0^T \frac{\ln 2 \cdot \left(1-\phi(0)\exp(-\nu \cdot t)\right)}{\alpha \cdot T_{\text{pot}}}\mathrm{d}t \tag{40}$$

The loss in BED due to repopulation is given by the final term of this equation

$$\text{Loss in BED due to repopulation} = \int_0^T \frac{\ln 2 \cdot \left(1-\phi(0)\exp(-\nu \cdot t)\right)}{\alpha \cdot T_{\text{pot}}}\mathrm{d}t \tag{41}$$

Assuming T_{pot} is constant, we integrate the above equation from time 0 to time T. There are two cases. If cell loss is zero, the loss in BED due to repopulation reduces to

$$\frac{\ln 2 \cdot T}{\alpha \cdot T_{\text{pot}}} \tag{42}$$

This is the familiar case where if, for example, when α = 0.35 Gy^{-1} and T_{pot} = 3.3 day, we get a loss in BED due to repopulation of about 0.6 Gy/day. This is the classic value for BED lost due to accelerated repopulation as is thought to occur in some rapidly growing tumors, where it is often assumed that this repopulation begins after a delay or "kick-off" time of about 21-28 days. Equation (42) is referred to as the continuous repopulation model, with a time delay it is referred to as the discontinuous repopulation model. Figure 3c shows this model where after a "kick-off" time of 28 days, we get a loss in BED due to repopulation of 0.6 Gy/day.

If cell loss is not equal to zero, the loss in BED due to repopulation is given by (Dale and Jones, 2007)

$$\frac{\ln 2}{\alpha \cdot T_{\text{pot}}}\left(T - \frac{\phi(0)}{\nu}(1 - \exp(-\nu \cdot T))\right) \tag{43}$$

This is called the progressive model of repopulation, and it models a continuously changing repopulation rate, equal initially to the effective doubling time and rising during treatment as cell loss decreases to approach the much shorter T_{pot} value at the end of treatment. This progressive model of repopulation is shown in Figure 3d.

The first term of the BED equation (40) above is the total physical absorbed dose, the second term accounts for incomplete repair during treatment which will increase the BED, and the third term accounts for clonogen repopulation during treatment decreasing the BED. Thus we have one general equation for BED that is applicable to any absorbed dose rate pattern and can account for repopulation as well.

3.4 Redistribution Model

Dividing cells, such as tumor cells, progress through the cell cycle, and the different phases of the cell cycle differ in their inherent radiosensitivities and in the shape of their cell survival curves. By contrast, late responding normal tissues are thought to divide less frequently, if at all in some tissues (e.g., nervous tissue). This difference between cycling tumor cells and noncycling surrounding normal tissues is responsible for redistribution (also called reassortment) effects that can give a therapeutic advantage during fractionated or protracted radiotherapy.

Intrinsic radiosensitivity is characterized by the α parameter in the linear quadratic model, and the α/β ratio determines the shape (or amount of shoulder) of the cell survival curve which in turn determines how sensitive the tissue is to the effects of dose fractionation or protraction. Classic cell survival curves for the different phases of the cell cycle for Chinese hamster cells are shown in Figure 3e (Sinclair and Morton, 1965). This cell line was used in developing a simple

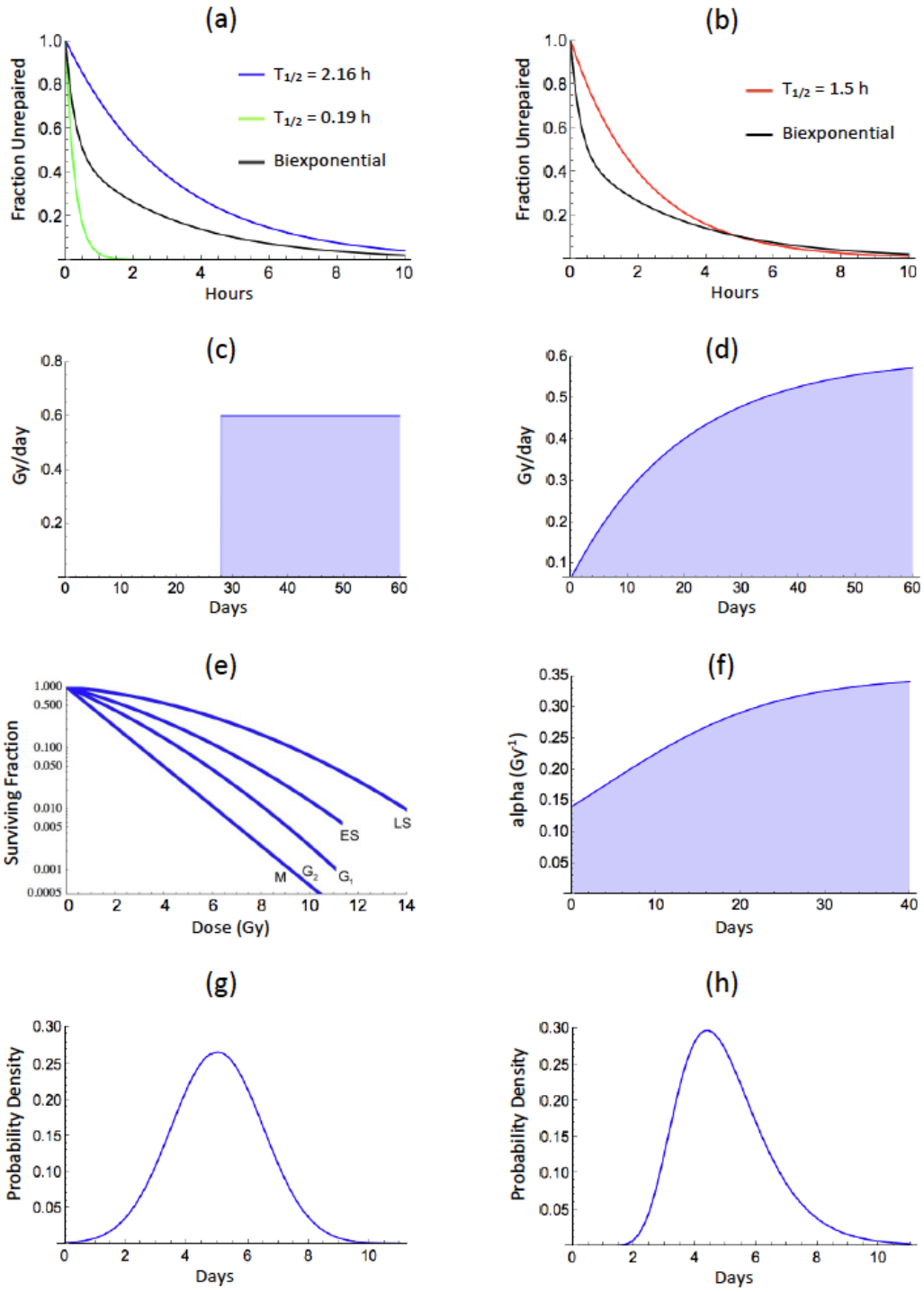


Figure 3. Radiation response modifying functions. (a) Biexponential repair function along with its short and long half-time components. (b) Mono-exponential repair function compared with the biexponential repair function shown in (a). (c) Discontinuous repopulation model function. (d) Progressive repopulation model function. (e) Cell cycle phase dependent survival fraction functions. (f) Reoxygenation function. (g) Normal distribution function. (h) Lognormal distribution function.

model of redistribution demonstrating the principle that, for fractionated or protracted radiotherapy, cycling cells have an overall lower surviving fraction than do noncycling cells. In our model, late responding normal tissues are assumed to remain in G1 phase while cycling cells in tumors are assumed to be distributed through the various phases of the cell cycle in proportion to the fractional time spent in each phase of the cell cycle as determined by cellular kinetics studies. A G0 component of cells could be included as well, where the cells are totally out of the cell cycle and are not affected by irradiation at all. The radiosensitivity parameters α for the various phases of the cell cycle were estimated from the cell survival curves in Figure 3e.

For the case of cycling cells, the RandomChoice function in *Mathematica* was used to randomly select from a discrete distribution which α parameter to use with each fraction of radiation. The function RandomChoice has the form

$$\text{RandomChoice}[\{w_1, w_2, \dots\} \to \{e_1, e_2, \dots\}] \tag{44}$$

and gives a pseudorandom choice e_i weighted by w_i.

In section 4.4 we demonstrate the effects of redistribution for fractionated dose delivery, where with each fraction delivered the radiosensitivity parameter α is randomly selected and then the surviving clonogenic cells after a dose of radiation is given by

$$SurvivingClonogens = StartingClonogens \cdot \exp(-\alpha \cdot FractionalBED). \tag{45}$$

The *Mathematica* function Nest is used to apply this $SurvivingClonogens$ function repeatedly over the course of treatment. The Nest function has the general form

$$\text{Nest}[f, x, n] \tag{46}$$

which applies a function *f* nested *n* times to the initial argument *x*. In our case, this function takes the form

$$\text{Nest}[SurvivingClonogens, StartingClonogens, n] \tag{47}$$

where the *SurvivingClonogens* function above is applied to the number of initial clonogens, *StartingClonogens*, and this is repeated for *n* fractions, and for the case of cycling cells, the value of α is sampled at the start of each fraction using the RandomChoice function. A variant of Nest, called NestList, is used which displays all of the intermediate results during the *n* fractions.

3.5 Reoxygenation Model

The cellular oxygen concentration has long been known to modify the radiation response of the tissues, with hypoxic tissues being less radiosensitive than well oxygenated (aerobic) tissues. This effect is characterized by the oxygen enhancement ratio (OER), which is defined as the ratio of the radiation dose in hypoxic conditions to the dose in aerobic conditions to produce the same biological effect. Carlson et al. (2006) showed that the radiosensitivity parameters α for hypoxic (H) and aerobic (A) tissues are related by

$$\alpha_H = \frac{\alpha_A}{OER} \tag{48}$$

and that the α/β values are related by

$$\left(\frac{\alpha}{\beta}\right)_H = OER \cdot \left(\frac{\alpha}{\beta}\right)_A \tag{49}$$

These values can be incorporated into the equations developed earlier to account for the reduced radiosensitivity α and the increased α/β value of hypoxic tissues. This can be done on a voxel-by-voxel basis to model the effects of hypoxia on treatment outcome.

The tissues may reoxygenate during treatment. Dale and Jones (2007) have modeled this by assuming that reoxygenation occurs at an exponential rate. Using this assumption, the OER as a function of time can be expressed as

$$OER(t) = (OER_0 - 1)e^{-zt} + 1 \tag{50}$$

where OER_0 is the OER at time 0, and z is the reoxygenation time constant. See Figure 3f.

We use the *Mathematica* Fold function to model reoxygenation during fractionated radiotherapy. Fold is an extension of the Nest function, and takes a second argument at each step of the process from the successive elements of a list. Fold has the general form

$$\mathrm{Fold}[f, x, \{a, b, \ldots\}] \tag{51}$$

which is similar to the Nest function in the previous section, but in addition to applying a function f nested n times to the initial argument x, a second argument is folded into the calculation at each step, which for our application will be the time in days of the dose fraction. For our case, the Fold function takes the form

$$\mathrm{Fold}[SurvivingClonogens, StartingClonogens, \{t_1, t_2, \ldots, t_n\}] \tag{52}$$

which is similar to the Nest function above, but now the time of the dose fraction is folded into the calculation, allowing us to account for the change in OER due to reoxygenation through equation (50), which will affect the number of surviving clonogens over the course of fractionation through equations (48) and (49). This methodology is implemented in Section 4.5. A variant of Fold, called FoldList, is used which displays all the intermediate results during the n fractions.

3.6 Radiosensitivity Model

The fifth R of radiobiology, intrinsic radiosensitivity, is extremely important in determining the radiation response of the tissues (Steel et al. 1989). In our model, radiosensitivity is characterized by the α parameter of the linear quadratic equation. It is through the α parameter that the BED in the voxels in converted into clonogen surviving fraction and ultimately into the voxel TCP. We show TCP results for individual patients with different values of the radiosensitivity parameter α and we also show the population average TCP results,

demonstrating the principle that dose response curves for individual patients are much steeper than those for a population of patients.

3.7 Heterogeneity Model

Biological processes always have some inherent variability or heterogeneity, and all of the parameters used in the radiobiological models described above can be sampled from distributions. Biological variability arises from a multitude of effects acting independently of one another, and these effects can be additive or multiplicative, leading to normal or lognormal distributions, respectively (Limpert et al. 2001).

In *Mathematica*, we sample from a normal distribution with mean μ and standard deviation σ using the composite function

$$\text{RandomVariate}\left[\text{NormalDistribution}[\mu, \sigma]\right]. \tag{53}$$

Similarly, we sample from a lognormal distribution using the composite function

$$\text{RandomVariate}\left[\text{LogNormalDistribution}[\mu, \sigma]\right]. \tag{54}$$

In this case, $\text{LogNormalDistribution}[\mu, \sigma]$ represents a lognormal distribution derived from a normal distribution with mean μ and standard deviation σ. To sample from a lognormal distribution with mean m and standard deviation s we use the transformations

$$\mu = \ln\left(\frac{m^2}{\sqrt{s^2 + m^2}}\right) \tag{55}$$

$$\sigma = \sqrt{\ln\left(\frac{s^2}{m^2} + 1\right)} \tag{56}$$

These formulas give the mean μ and standard deviation σ for the normal distribution from which the lognormal distribution with mean m and standard deviation s is derived (Wicklin, 2014).

Figures 3g and 3h show normal and lognormal distributions for T_{pot} with a mean of 5 days and a standard deviation of 1.5 days.

3.8 Tumor Control Probability Model

Having obtained the BED on a voxel-by-voxel basis, the Poisson model of tumor control is used to convert voxel BED into voxel TCP. This model has its shortcomings, and has been shown to underestimate the TCP when repopulation is involved, being most severe for rapidly proliferating tumors. It is a widely used model, however, and for most clinical situations provides a reasonable approximation of TCP and is adequate for the purposes of this book.

From the defining BED equation

$$BED = -\frac{\ln SF}{\alpha} \tag{57}$$

the cell surviving fraction is

$$SF = \exp(-\alpha \cdot BED). \tag{58}$$

If the initial clonogen number is N, the number of surviving clonogens is given by

$$Surviving\ Clonogens = N \cdot SF \tag{59}$$

or

$$Surviving\ Clonogens = N \cdot \exp(-\alpha \cdot BED). \tag{60}$$

From the surviving clonogen number, using the Poisson model of tumor control, the tumor control probability (TCP) is given by

$$TCP = \exp(-Surviving\ Clonogens) \tag{61}$$

or

$$TCP = \exp\big(-N \cdot \exp(-\alpha \cdot BED)\big). \tag{62}$$

Since we are interested in TCP in the individual voxels, and since everything in our models can vary voxel-to-voxel, in Section 4.7 we will write this TCP equation as

$$VoxelTCP = \exp(-VoxelClonogens \cdot \exp(-VoxelAlpha \cdot VoxelBED)) \tag{63}$$

where each of the terms in this equation are lists of values in the individual voxels.

From the TCPs in the individual voxels, the TCP in the tumor as a whole can be found as the product of the voxel TCPs. In developing this relationship, the tumor is assumed to consist of a certain number of noninteracting and independent clonogens, cell killings are considered uncorrelated events, and the tumor is controlled if all clonogens are killed. The tumor is divided into a number of tumorlets (or subvolumes) each of which is small enough that the dose can be considered to be uniform within it. These are the individual voxels in our model. These voxels are then assumed to respond independently to irradiation. The response of a given voxel (of partial volume v_i, receiving dose d_i) can be inferred from that of the entire tumor homogeneously irradiated to that same dose, through the following relationship, derived directly from Poisson statistics (Goitein, 2008).

$$TCP(d_i, v_i) = [TCP(d_i, V)]^{\frac{v_i}{V}}. \tag{64}$$

For example, if there are 100 equal sized voxels in the total volume, the product of the TCPs of the 100 voxels gives the TCP in the whole tumor, and each individual voxel has a TCP that is scaled according to the above equation. For non-uniform irradiation, it follows that the response

of the tumor as a whole is given by the product of the individual voxel TCPs through the relationship

$$TCP = TCP(d_1, v_1) \cdot TCP(d_2, v_2) \cdot TCP(d_3, v_3) \ldots = \prod_{i=1}^{M} TCP(d_i, v_i) \tag{65}$$

where the doses d_i are the individual voxel BED values.

In our voxel-based radiobiological modeling, we first calculate the BEDs in each voxel, which are stored in a list called *VoxelBEDs*

$$VoxelBEDs = \{BED1, BED2, BED3, \ldots\} \tag{66}$$

and this list is then converted into a list of TCPs called *VoxelTCPs*

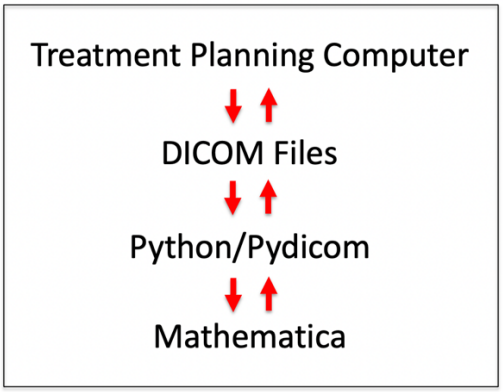
$$VoxelTCPs = \{TCP1, TCP2, TCP3, \ldots\}. \tag{67}$$

Finally, the tumor TCP as a whole is found by multiplying all the voxel TCPs together which in functional form is given by

$$TumorTCP = \text{Apply}[\text{Times}, VoxelTCPs]. \tag{68}$$

4. Functional Implementation

Now we will describe how we implement the functions described in Chapter 3 into *Mathematica* code. To perform radiobiological modeling on a voxel-by-voxel basis, we start with a patient's radiotherapy treatment plan and its associated DICOM-RT Dose file and DICOM-RT Structure Set files. From these two files, the dose in the individual voxels as well as the region of interest (ROI) that these voxels belong to can be obtained. With the voxel dose information and with knowledge of the type of dose delivery (i.e., external beam, brachytherapy, or radionuclide therapy) and the specific time course (i.e., detailed fractionation or protraction pattern) of the treatment, a dose-rate input function can be generated for each voxel. Using methods described in Section 3.2, the dose-rate input function can be convolved with a repair function to get the BED in each voxel. Other functions, including those for the remaining 4 Rs of radiobiology, can be included in the model as well. The DICOM-RT Dose file can be edited replacing the physical dose in each voxel with the biological dose, and the BED can then be displayed on the treatment planning computer in place of physical dose. Finally, with an estimate of clonogen number per voxel, voxel BEDs can be converted into voxel TCPs, and the TCP in the tumor as a whole can be computed.



Flow of Information

4.1. Input and Output

To obtain information in the voxels, the DICOM-RT Dose file and Structure Set file were accessed using the Python programming environment with the Pydicom package of functions (Pydicom version 0.9.7). Pydicom (Mason et al. 2024) is a Python package for reading from and writing to DICOM files. Python and Pydicom were used imperatively for input and output tasks, such as reading and editing DICOM files, as well as for processing the information in these files to prepare it for use in the radiobiological modeling. For the actual voxel-based radiobiological modeling, *Mathematica* was used in a functional manner.

The dose in each voxel is contained in the Pixel Data attribute of the DICOM-RT Dose file, which was accessed using the Pydicom *pixel_array* property. The contour information is contained in the Contour Data attribute of the DICOM-RT Structure Set file. In the Python/Pydicom program shown in Appendix E, the DICOM-RT Dose file and the DICOM-RT Structure Set file were used together to determine which contour or ROI a particular voxel belongs to. This program

determines which voxels in the dose array lie within the boundary of the ROI of interest, in our case the tumor ROI. For each slice of voxels along the z-axis of the imaging dataset, the ROIs are represented as closed polygons, and the vertex coordinates of these polygons are contained in the Contour Data attribute. The Matplotlib *path* module, with its *contains_point* function, was used to test if the voxel coordinates lie within the ROI polygon. Those voxels within the ROI boundary are marked by setting their dose value equal to zero.

Figure 2 on page 11 shows various dose-rate input functions that can be generated on a voxel-by-voxel basis as demonstrated in the next section. Figure 3 on page 15 shows radiation response modifying functions that are used in this book, and these functions can also vary voxel-to-voxel. The dose-rate input functions give physical dose in the voxels, the radiation response modifying functions are used to transform physical dose into biological dose.

The biological dose (BED) in the voxels can then be written back into the DICOM-RT Dose file in place of the physical dose. This was done by exporting the BED results from *Mathematica* as a flattened one-dimensional list of values (Appendix K), and then importing this list into Python, reshaping the list into the appropriate array dimension using the *reshape* function from the NumPy scientific computing library, and finally writing these BED values to the dose array using Pydicom (Appendix L). Dose values are stored in the dose array as integer data types and the DICOM-RT Dose Grid Scaling attribute is used to convert these integer values to floating-point numbers representing dose. Care must be taken in converting the BED output values from *Mathematica* to the appropriate integer values in the DICOM-RT Dose file, and the Dose Grid Scaling attribute may have to be edited if the integer representations of the BED values are larger than the maximum integer value allowed.

4.2 Repair

As shown in Section 3.2, a dose-rate input function can be convolved with a repair function to get BED, and this can be done on a voxel-by-voxel basis. There are two general types of radiation dose delivery we will consider. The first is where discrete pulses of radiation are given, such as in fractionated external beam radiation, Gamma Knife radiosurgery shots, or HDR pulses. The second type of dose delivery is one of continuous protracted radiation treatment, such as in brachytherapy or radionuclide therapy. We will demonstrate the calculation of BED for these two types of dose delivery using examples from Gamma Knife radiosurgery (Fig. 2c) and from iodine-125 brachytherapy (Fig. 2g), respectively. Similar methods can be applied to other types of dose delivery, including the other examples shown in Figure 2.

For the Gamma Knife radiosurgery case, we demonstrate the process using a small dataset that is 11x10x9 in voxel dimensions for 990 total voxels. One slice of this dataset, with the dose-rate input functions in each voxel, along with the calculated BEDs, is shown in Figure 4. The dose delivery for this case consists of six Gamma Knife shots, and each shot has a specific dose-rate and treatment time. The shot times will be the same in all the voxels and is known from the treatment plan, but the shot dose-rates will vary throughout the voxels. We find the dose-rates for the individual shots using the Python program shown in Appendix H. For each of the six Gamma Knife shots, individual DICOM-RT Dose files were generated by zeroing out the dose contributions from the other shots, so that we have voxel-by-voxel dose information for each of

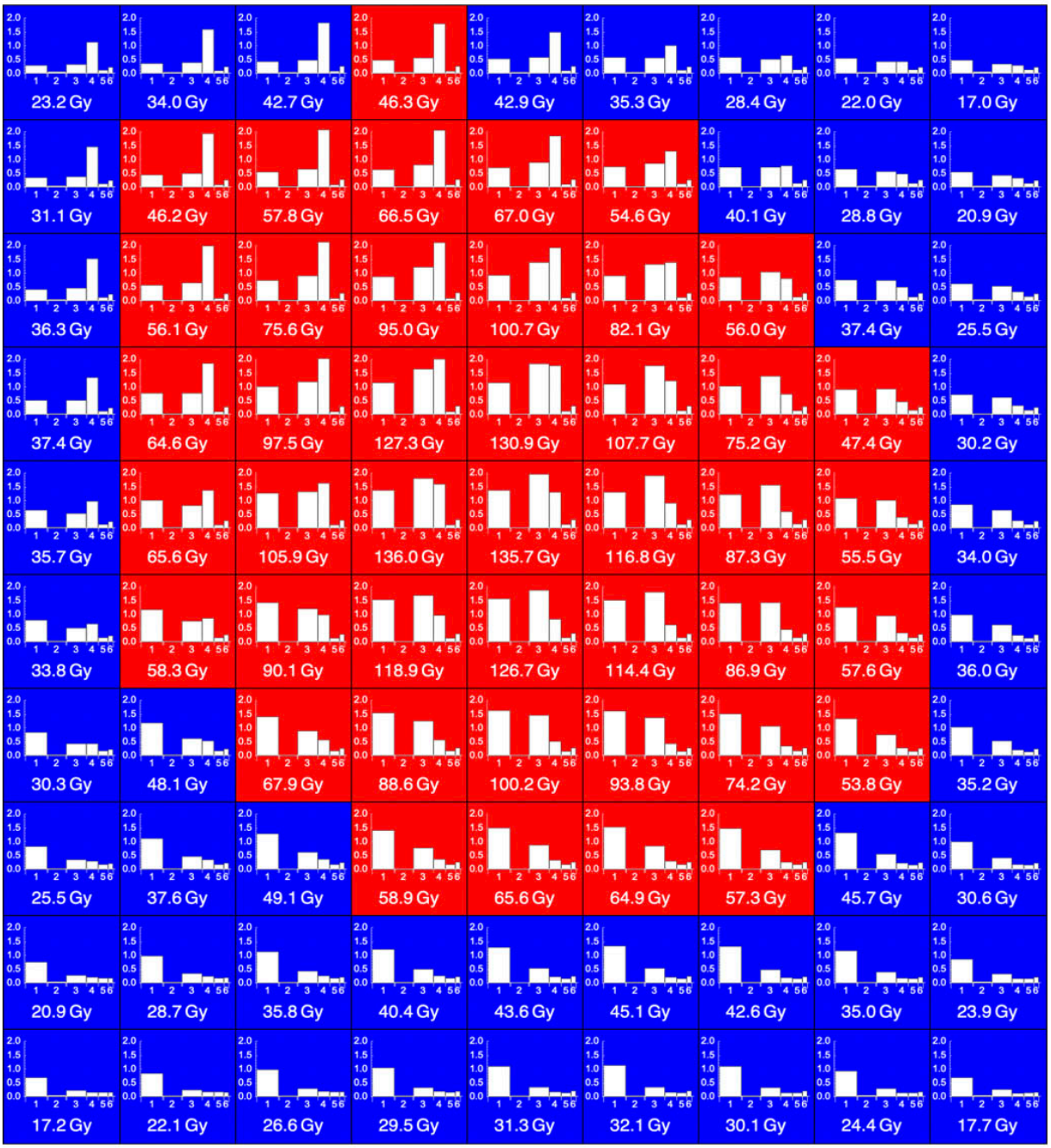


Figure 4. Gamma Knife shots in one slice of voxels along with the computed voxel BEDs. Dose rates in Gy/min are on the ordinates and shot numbers are on the abscissas of the plots. The widths of the shots are proportional to the shot times. Note that shot 2 contributes little to this slice of voxels. Red voxels = tumor; blue voxels = normal tissue.

the six individual shots. Knowing the times of each shot, the individual shot dose information can be converted into shot dose-rate information. The program in Appendix H forms a list of the Gamma Knife shot times and shot dose-rates for all the voxels in the dose array and writes this list to a text file. In this example, there are six shots, each with an associated time and dose-rate.

The complete list therefore has dimension 11x10x9x6x2. We import this multi-dimensional list into *Mathematica*, and we name it VoxelDoseRates.

Below we display a portion of the VoxelDoseRates list containing the shot times and dose-rates for the nine voxels in slice nine and row six of the dataset. This is a 9x6x2 list of values, containing information for the nine voxels, each with six shots described by two values (shot time and dose-rate). The *Mathematica* Part function is used to display the information we want. The Part function has the form shown below, here it specifies that we want to display the information in slice nine and row six of the dataset. The semicolon at the end of the command suppresses the output.

```
Part[VoxelDoseRates, 9, 6];
```

A more convenient shorthand method of applying the Part function is shown below, along with the output.

```
VoxelDoseRates[[9, 6]]
```

```
{{{9.03, 0.653157}, {7.78, 0.0267969}, {8.24, 0.525846}, {4.9, 0.975184}, {4.02, 0.136348}, {1.88, 0.234947}},
 {{9.03, 1.00739}, {7.78, 0.020746}, {8.24, 0.813584}, {4.9, 1.37902}, {4.02, 0.121439}, {1.88, 0.262241}},
 {{9.03, 1.27481}, {7.78, 0.0178203}, {8.24, 1.32263}, {4.9, 1.62214}, {4.02, 0.112715}, {1.88, 0.28238}},
 {{9.03, 1.37753}, {7.78, 0.0163907}, {8.24, 1.80645}, {4.9, 1.59219}, {4.02, 0.115122}, {1.88, 0.299836}},
 {{9.03, 1.37231}, {7.78, 0.016025}, {8.24, 1.96415}, {4.9, 1.30406}, {4.02, 0.125685}, {1.88, 0.308647}},
 {{9.03, 1.30343}, {7.78, 0.0166234}, {8.24, 1.90607}, {4.9, 0.900582}, {4.02, 0.138721}, {1.88, 0.304937}},
 {{9.03, 1.21132}, {7.78, 0.0183522}, {8.24, 1.55482}, {4.9, 0.600258}, {4.02, 0.146008}, {1.88, 0.29659}},
 {{9.03, 1.07926}, {7.78, 0.0219761}, {8.24, 1.01295}, {4.9, 0.398322}, {4.02, 0.145306}, {1.88, 0.28016}},
 {{9.03, 0.856538}, {7.78, 0.0297226}, {8.24, 0.643568}, {4.9, 0.271281}, {4.02, 0.140091}, {1.88, 0.260121}}}
```

The shot times, in minutes, are the first entries (9.03, 7.78, 8.24, 4.90, 4.02, 1.88) and the dose-rates, in Gy/min, are the second entries of each pair of values. We convert the time and dose-rate information in the VoxelDoseRates list into voxel doses with the following functional equation.

```
VoxelDoses = Apply[Plus, Apply[Times, VoxelDoseRates, {4}], {3}];
```

The resulting list is called VoxelDoses and contains the doses in the 990 voxels of the dataset. In this functional equation, the {4} specifies that we multiple the values (times and dose-rates) at level 4 of the list structure, and the {3} specifies that we add the resulting values at level 3 of the list structure. This VoxelDoses equation is an example of the concise and powerful programming techniques available in functional languages. The entire VoxelDoseRates list is passed into this equation as an argument and a new list called VoxelDoses is produced containing the dose values, and all in just one line of code. The dose results (in Gy) for the nine voxels in slice nine and row six of the dataset are shown below.

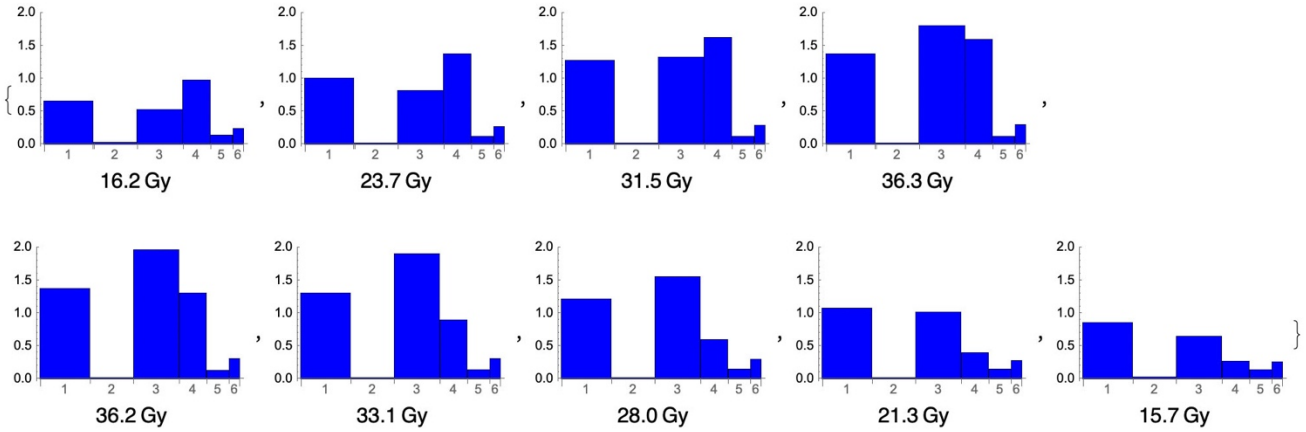
```
VoxelDoses[[9, 6]]
```

```
{16.2077, 23.7005, 31.4811, 36.28, 36.1766, 33.1491, 27.9785, 21.326, 15.6502}
```

We convert the shot times and dose-rates above into graphical form using the *Mathematica* RectangleChart function which, for each voxel, makes a rectangle chart with bars of width proportional to the shot times and height equal to the shot dose-rates (in Gy/min). We include

the total voxel dose in this display as well.

```
VoxelDoseRatesTable[[9, 6]]
```



This is a graphical representation of the shot times and dose-rates of the six shots for the nine voxels in slice nine and row six of the dataset. Similar figures are shown in Figure 4 (in the sixth row from the bottom) along with the calculated voxel BED values. Note that for this slice of voxels, shot two contributes little to the dose.

For use in the equations developed in Section 3.2, we need to convert the above information into functional form in an input dose-rate function. We use the *Mathematica* UnitStep function to do this. The cumulative times, in minutes, for the shots shown above are t1=0, t2=9.03, t3=16.81, t4=25.05, t5=29.95, t6=33.97, and t7= 35.85. For each shot we form a unit step function as follows.

```
UnitStepShot1T = UnitStep[t - t1] - UnitStep[t - t2];
UnitStepShot2T = UnitStep[t - t2] - UnitStep[t - t3];
UnitStepShot3T = UnitStep[t - t3] - UnitStep[t - t4];
UnitStepShot4T = UnitStep[t - t4] - UnitStep[t - t5];
UnitStepShot5T = UnitStep[t - t5] - UnitStep[t - t6];
UnitStepShot6T = UnitStep[t - t6] - UnitStep[t - t7];
```

For shot one, we call this function UnitStepShot1T, and similarly for the other shots. The T specifies that this is a function of the variable *t*, because we will form a similar set of functions of the variable *w*, that we will specify as UnitStepShot1W, and so on. We need to write these functions both in terms of the variables *t* and *w* because the convolution equation (22) we use to calculate BED includes dose-rate functions of both these variables.

Each of the above unit step functions are of unit height. We need to scale these step functions appropriately to correspond with the dose-rates of the shots in each voxel. We do this by mapping the above UnitStepShot functions over the VoxelDoseRates list which contains the dose-rates of all of the shots in all of the voxels. This is shown in the functional equations below. For example, Shot1VoxelDoseRateT is formed by mapping the UnitStepShot1T function over the VoxelDoseRates list and multiplying this function by the appropriate entry in the VoxelDoseRates list. The *Mathematica* Part function is used to pick out the appropriate components from the list. The VoxelDoseRates list has dimensions 11x10x9x6x2 and the Part specifier [[All, All, All, 1, 2]] indicates that for all 11 slices, all 10 rows, and all 9 columns of the

dataset, we want the first component of 6 (shot 1), and the second component of 2 (the dose rate), and similarly for the remaining five shots.

```
Shot1VoxelDoseRateT = Map[(# UnitStepShot1T) &, VoxelDoseRates, {5}][[All, All, All, 1, 2]];
Shot2VoxelDoseRateT = Map[(# UnitStepShot2T) &, VoxelDoseRates, {5}][[All, All, All, 2, 2]];
Shot3VoxelDoseRateT = Map[(# UnitStepShot3T) &, VoxelDoseRates, {5}][[All, All, All, 3, 2]];
Shot4VoxelDoseRateT = Map[(# UnitStepShot4T) &, VoxelDoseRates, {5}][[All, All, All, 4, 2]];
Shot5VoxelDoseRateT = Map[(# UnitStepShot5T) &, VoxelDoseRates, {5}][[All, All, All, 5, 2]];
Shot6VoxelDoseRateT = Map[(# UnitStepShot6T) &, VoxelDoseRates, {5}][[All, All, All, 6, 2]];
```

The individual shots are added together to form the voxel-by-voxel dose-rate input functions, in a list named VoxelDoseRateInputT.

```
VoxelDoseRateInputT = Shot1VoxelDoseRateT + Shot2VoxelDoseRateT + Shot3VoxelDoseRateT +
                      Shot4VoxelDoseRateT + Shot5VoxelDoseRateT + Shot6VoxelDoseRateT;
```

We do a similar process with *w* in place of *t*, which we name VoxelDoseRateInputW.

```
VoxelDoseRateInputW = Shot1VoxelDoseRateW + Shot2VoxelDoseRateW + Shot3VoxelDoseRateW +
                      Shot4VoxelDoseRateW + Shot5VoxelDoseRateW + Shot6VoxelDoseRateW;
```

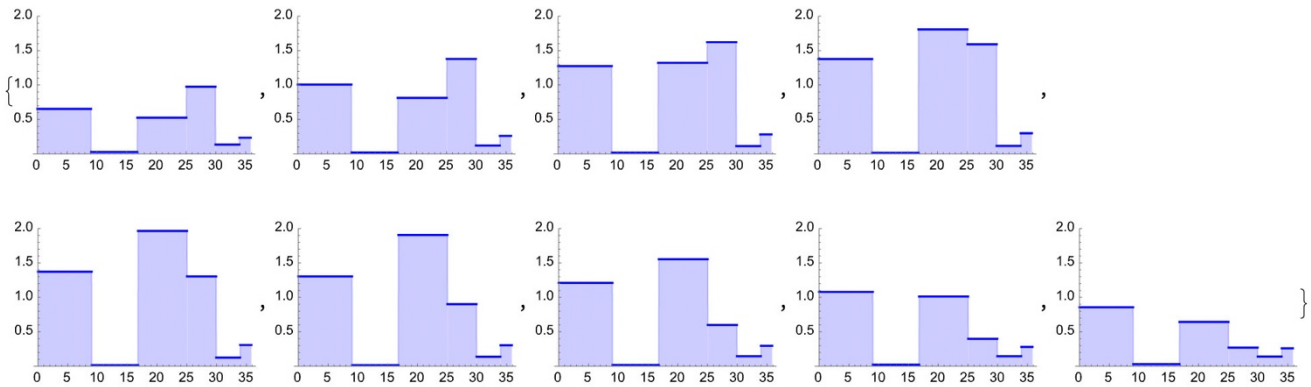
Below is shown the resulting dose-rate input function, VoxelDoseRateInputT, in the first voxel of slice nine and row six of the dataset.

```
VoxelDoseRateInputT[[9, 6, 1]]
```

```
0.234947 (-UnitStep[-35.85 + t] + UnitStep[-33.97 + t]) + 0.136348 (-UnitStep[-33.97 + t] + UnitStep[-29.95 + t]) +
 0.975184 (-UnitStep[-29.95 + t] + UnitStep[-25.05 + t]) + 0.525846 (-UnitStep[-25.05 + t] + UnitStep[-16.81 + t]) +
 0.0267969 (-UnitStep[-16.81 + t] + UnitStep[-9.03 + t]) + 0.653157 (-UnitStep[-9.03 + t] + UnitStep[t])
```

Finally, shown below is a list of the VoxelDoseRateInputT functions in graphical form for the nine voxels in slice nine, row six of the dataset. Dose-rate in Gy/min is on the ordinate and time is in minutes on the abscissa of these graphs.

```
VoxelDoseRatesInputFunctionsPlot[[9, 6]]
```



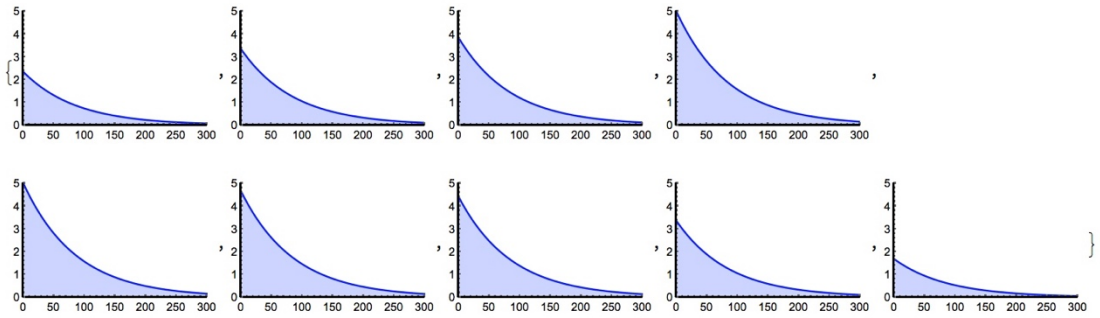
The input dose-rate functions are now in a form that can be used in equation (22) to compute the voxel BEDs.

For the I-125 case, we form the dose-rate input function VoxelDoseRateIodineT as follows, where 59.4 is the half-life of I-125 in days.

```
VoxelDoseRateIodineT = InitialDoseRateIodine * Exp[-(Log[2] / 59.4) * t];
```

VoxelDoseRateIodineT and InitialDoseRateIodine are lists containing information for all the voxels of a treatment plan. We form a similar dose-rate input function VoxelDoseRateIodineW of the variable *w*, as before.

Knowing the dose in each of the voxels from the treatment plan, we can solve for the initial dose-rate, InitialDoseRateIodine, by integrating the above equation to infinity (or a sufficiently long time) and setting the result equal to the voxel dose. For example, for a voxel dose of 144 Gy, the initial dose rate is 1.68 Gy/day. Below is a list of I-125 dose-rate input functions in graphical form for the nine voxels as before, with the last voxel receiving 144 Gy and the others receiving higher doses. An example of this type of analysis is shown in Appendix S.



The other dose-rate input functions shown in Figure 2 can be formed using similar methods, and the magnitudes of these functions can be scaled appropriately for the individual voxels, as demonstrated for the cases above. Essentially any dose-rate input function, whether discrete or continuous, can be modeled in this way, and on a voxel-by-voxel basis.

We now perform a convolution of the input dose-rate functions with a repair function to get BED using the methods of Section 3.2. We use BED equation (20) along with psi equation (22). For the Gamma Knife radiosurgery example above, we get voxel-by-voxel psi values in a list called VoxelPsi as shown.

```
VoxelPsi =
  Parallelize[2 * (
       (* shot 1 *)
      Integrate[VoxelDoseRateInputT * Exp[-VoxelMu * t] *
         (Integrate[VoxelDoseRateInputW * Exp[VoxelMu * w], {w, t1, t}]), {t, t1, t2}] +
       (* shot 2 *)
       Integrate[VoxelDoseRateInputT * Exp[-VoxelMu * t] *
         (Integrate[VoxelDoseRateInputW * Exp[VoxelMu * w], {w, t2, t}] +
           Integrate[VoxelDoseRateInputW * Exp[VoxelMu * w], {w, t1, t2}]), {t, t2, t3}] +
       (* shot 3 *)
       Integrate[VoxelDoseRateInputT * Exp[-VoxelMu * t] *
         (Integrate[VoxelDoseRateInputW * Exp[VoxelMu * w], {w, t3, t}] +
           Integrate[VoxelDoseRateInputW * Exp[VoxelMu * w], {w, t1, t3}]), {t, t3, t4}] +
       (* shot 4 *)
       Integrate[VoxelDoseRateInputT * Exp[-VoxelMu * t] *
         (Integrate[VoxelDoseRateInputW * Exp[VoxelMu * w], {w, t4, t}] +
           Integrate[VoxelDoseRateInputW * Exp[VoxelMu * w], {w, t1, t4}]), {t, t4, t5}] +
       (* shot 5 *)
       Integrate[VoxelDoseRateInputT * Exp[-VoxelMu * t] *
         (Integrate[VoxelDoseRateInputW * Exp[VoxelMu * w], {w, t5, t}] +
           Integrate[VoxelDoseRateInputW * Exp[VoxelMu * w], {w, t1, t5}]), {t, t5, t6}] +
       (* shot 6 *)
       Integrate[VoxelDoseRateInputT * Exp[-VoxelMu * t] *
         (Integrate[VoxelDoseRateInputW * Exp[VoxelMu * w], {w, t6, t}] +
           Integrate[VoxelDoseRateInputW * Exp[VoxelMu * w], {w, t1, t6}]), {t, t6, t7}])];
```

The Parallelize command distributes the computation among the available kernels and processors on the computer being used. VoxelMu is the repair rate constant μ, which could vary voxel-to-voxel, but in our examples is constant throughout the voxels.

For the I-125 case, we only need the first part of psi equation (22).

```
VoxelPsi = 2 * (Integrate[VoxelDoseRateIodineT * Exp[-VoxelMu * t] *
     (Integrate[VoxelDoseRateIodineW * Exp[VoxelMu * w], {w, 0, t}]), {t, 0, t1}]);
```

Here, t1 is the total time of the integration, for a permanent implant twenty half-lives is more than sufficient.

The above implementations of the VoxelPsi equations (22) are the most mathematically correct ways to write the equations, with the input dose-rate functions written in terms of both *t* and *w*. It was found, however, that by writing the input dose-rate functions just in terms of the variable *t*, the computation is much faster and the results are identical for the discrete pulse types of dose delivery (e.g., the Gamma Knife case or conventional fractionation) and nearly identical for the continuous types of dose delivery (e.g., the I-125 case). This is the implementation we actually used in most of this book, and this is discussed further in Chapter 5 and Appendix S and T. Using this implementation of the convolution equations for the Gamma Knife radiosurgery case, VoxelPsi is written as follows.

```
VoxelPsi =
  ParallelMap[ 2 * (
       (* shot 1 *)
       Integrate[# * Exp[-VoxelMu * t] * (Integrate[# * Exp[VoxelMu * w], {w, t1, t}]),
         {t, t1, t2}] +
       (* shot 2 *)
       Integrate[# * Exp[-VoxelMu * t] * (Integrate[# * Exp[VoxelMu * w], {w, t2, t}] +
           Integrate[# * Exp[VoxelMu * t], {t, t1, t2}]), {t, t2, t3}] +
       (* shot 3 *)
       Integrate[# * Exp[-VoxelMu * t] * (Integrate[# * Exp[VoxelMu * w], {w, t3, t}] +
           Integrate[# * Exp[VoxelMu * t], {t, t1, t3}]), {t, t3, t4}] +
       (* shot 4 *)
       Integrate[# * Exp[-VoxelMu * t] * (Integrate[# * Exp[VoxelMu * w], {w, t4, t}] +
           Integrate[# * Exp[VoxelMu * t], {t, t1, t4}]), {t, t4, t5}] +
       (* shot 5 *)
       Integrate[# * Exp[-VoxelMu * t] * (Integrate[# * Exp[VoxelMu * w], {w, t5, t}] +
           Integrate[# * Exp[VoxelMu * t], {t, t1, t5}]), {t, t5, t6}] +
       (* shot 6 *)
       Integrate[# * Exp[-VoxelMu * t] * (Integrate[# * Exp[VoxelMu * w], {w, t6, t}] +
           Integrate[# * Exp[VoxelMu * t], {t, t1, t6}]), {t, t6, t7}])
    &, VoxelDoseRateInputT];  (* a list of the voxel dose rate input functions *)
```

The function ParallelMap is used to map, in a parallel manner, the convolution equations in (22) over the list VoxelDoseRateInputT which contain all the dose-rate input functions as a function of t in all the voxels, to produce the psi values on a voxel-by-voxel basis. The dose-rate input functions are not written as a function of w in this formulation. In this equation, each element of the VoxelDoseRateInputT list is substituted in place of the # value during the mapping and a new list called VoxelPsi is produced giving the psi values in each voxel.

Similarly, for the iodine-125 case, we use the form of the equation below to compute VoxelPsi.

```
VoxelPsi = ParallelMap[ 2 * Integrate[# * Exp[-VoxelMu * t] * (Integrate[# * Exp[VoxelMu * w],
          {w, 0, t}]), {t, 0, t1}] &, VoxelDoseRateIodineT];
```

The voxel BED equation (20), containing a list of BEDs in all the voxels is written as follows.

```
VoxelBeds = VoxelDoses + (VoxelPsi / VoxelAlphaBeta);
```

VoxelDoses is a list of the voxel doses, VoxelPsi is a list of the voxel psi values, and VoxelAlphaBeta is a list of the α/β values in the voxels. VoxelBeds is thus seen to be made up of two terms, the first term is the absorbed dose and the second term accounts for incomplete repair during treatment and is modified by the α/β value.

When using a biexponential repair function, a second list of psi values called VoxelPsi2 for the second repair rate constant μ_2, is included, as in equation (23), and the VoxelBED equation is now written as shown here, with a+b=1.

```
VoxelBeds = VoxelDoses + ((a * VoxelPsi1 + b * VoxelPsi2) / VoxelAlphaBeta);
```

For the Gamma Knife example, the BED results (in Gy) for the nine voxels in slice nine and row six of the dataset are shown below. These results were calculated using a biexponential repair function with short and long repair half-times of 0.19 h and 2.16 h, respectively (Figure 3a), a partition coefficient c = 0.98 (see Eq. (24)), and an α/β value of 10 Gy.

```
VoxelBeds[[9, 6]]
```

```
{35.7375, 65.5558, 105.909, 135.985, 135.672, 116.817, 87.3034, 55.5004, 33.9667}
```

The results are shown in figure 4 and Appendix J. The methodology developed above was applied to a BED analysis of a tumor treated with two different Gamma Knife radiosurgery plans. The dataset used was of size 35x34x34 voxels. One plan was a high-efficiency plan with eight isocenters (shots) and a treatment time of 33.9 minutes, the other was a low-efficiency plan with thirteen isocenters and a treatment time of 79.3 minutes. In calculating the tumor voxel BEDs, a biexponential repair function was used, with short and long repair half-times of 0.19 h and 2.16 h, respectively, partition coefficient c=0.98 (giving a=0.5051, b=0.4949), and the α/β value was 10 Gy. For the high-efficiency plan, the effect on tumor BED of treating at half the activity (and twice the treatment time) was also investigated. The results for this case are discussed in Section 5.1.

4.3 Repopulation

Repopulation in the voxels decreases the BED in those voxels where this occurs. The loss in BED due to repopulation is given by Equation (41). The integrand of this equation is written in *Mathematica* as follows, with all of the components of this equation (VoxelTpot, VoxelAlpha, etc.) being lists of values in the individual voxels.

```
VoxelRepopulationRate[t_] = (Log[2] / (VoxelTpot * VoxelAlpha)) *
    (1 - (VoxelInitialCellLoss * Exp[-VoxelCellLossRate * t]));
```

We integrate this function to time T with the following equation, which gives the loss of BED in the voxels due to repopulation over this time.

```
VoxelRepopulation[T_] = Integrate[VoxelRepopulationRate[t], {t, 0, T}];
```

The voxel BED equation, including repopulation, is now written as shown below, and again all the terms in this equation are lists of values.

```
VoxelBED = VoxelDose + (VoxelPsi / VoxelAlphaBeta) - VoxelRepopulation;
```

The BED in the voxels is now made up of three terms: the physical dose, the contribution to BED due to unrepaired damage, and a negative term due to clonogen repopulation during treatment.

Figure 3c shows the voxel repopulation rate for the discontinuous repopulation model which was discussed in Section 3.3., where with VoxelTpot = 3.3 days, VoxelAlpha = 0.35 Gy^{-1}, and VoxelInitialCellLoss = 0, the loss in BED is 0.6 Gy/day after a 28 day delay. Figure 3d shows the

voxel repopulation rate for the progressive model of repopulation, with VoxelTpot and VoxelAlpha as above, but now with VoxelInitialCellLoss = 0.9 and VoxelCellLossRate = 0.05 d^{-1}. As cell loss goes to zero, the effective doubling time approaches the Tpot value, and the voxel repopulation rate increases to a value of 0.6 Gy/day loss in BED.

The effect of repopulation on the TCD-50 (50% tumor control dose) for a single voxel was investigated for a 60 Gy conventional fractionation treatment in 2 Gy fractions, using both the progressive and discontinuous models of repopulation. For the progressive repopulation model, VoxelAlpha = 0.35 Gy^{-1}, VoxelInitialCellLoss = 0.9, VoxelCellLossRate = 0.03 d^{-1}, and four different VoxelTpot values (2.5, 5, 7.5, and 10 days) were used. For the discontinuous repopulation model, VoxelAlpha = 0.35 Gy^{-1}, VoxelTpot = 3.3 days, and there was a 28 day delay before repopulation begins.

We also investigated the effect on tumor TCP of various sized aggressive spots, regions where the tumor was repopulating, as a function of VoxelTpot. We used a hypothetical case of 100 voxels, for a 60 Gy conventional fractionation treatment in 2 Gy fractions, and examined the effect on tumor TCP when 10%, 25%, or 50% of the voxels were repopulating during treatment. The progressive repopulation model, with VoxelAlpha = 0.35 Gy^{-1}, VoxelInitialCellLoss = 0.9, and VoxelCellLossRate = 0.05 d^{-1} was used, and the effect on tumor TCP was plotted as a function of VoxelTpot. A starting number of clonogens in the voxels was chosen to produce a baseline value of 80% tumor TCP for large VoxelTpot values (and little repopulation). The results are discussed in Section 5.2.

4.4 Redistribution

We developed a simple model of redistribution in the voxels using the cell cycle phase specific survival curves for Chinese hamster cells shown in Figure 3e. From these survival curves, VoxelAlpha values were estimated to be 0.75 Gy^{-1}, 0.35 Gy^{-1}, 0.25 Gy^{-1}, and 0.10 Gy^{-1}, for M/G2, G1, ES, and LS cells, respectively. From cellular kinetics studies, values for times spent in the various phases of the cell cycle are T_C = 11 h, T_M = 1 h, T_S = 6 h, T_{G1} = 1h, T_{G2} = 3 h. (Hall and Giaccia, 2019). From these data, the fraction of time that the cells spend in each phase of the cell cycle was calculated to be 0.3637, 0.0909, 0.2727, and 0.2727 for M/G2, G1, ES, and LS phases, respectively, and where S phase cells were assumed to be equally divided between ES and LS phases. As we have mentioned before a G0 component could also be included for cells totally out of cycle and unaffected by irradiation.

We used a hypothetical case of a tumor containing 100 voxels, for a 60 Gy conventional fractionation treatment in 2 Gy fractions, with VoxelAlphaBeta = 10 Gy, giving a VoxelBED value of 72 Gy. For G1 cells with VoxelAlpha = 0.35 Gy^{-1}, a BED of 72 Gy can control about 60 billion clonogens at the 50% TCP level, or about 600 million clonogens per voxel.

Equation (45) is written in *Mathematica* with the following equation, where for a starting number of clonogens, VoxelClonogens, it gives the surviving clonogens in the voxels after a single fraction of radiation with BED value VoxelFractionalBED.

```
VoxelSurvivingClonogens[VoxelClonogens_] =
   VoxelClonogens * Exp[-VoxelAlpha * VoxelFractionalBED];
```

As in Equations (46) and (47), we apply the *Mathematica* Nest function to apply this equation repeatedly over the 30 fractions. We show the results for a single voxel starting with 600 million clonogens, and we use the NestList function which displays all of the intermediate values, rather than the Nest function which gives just the final value. The results shown are for G1 cells with VoxelAlpha = 0.35 Gy^{-1}, and for this case VoxelFractionalBED = 2.4 Gy.

```
NestList[VoxelSurvivingClonogens, 600 000 000, 30]
```

$$\{600\,000\,000,\ 2.59026\times10^{8},\ 1.11824\times10^{8},\ 4.82758\times10^{7},\ 2.08412\times10^{7},\ 8.99735\times10^{6},$$
$$3.88425\times10^{6},\ 1.67687\times10^{6},\ 723\,923.,\ 312\,525.,\ 134\,920.,\ 58\,246.6,\ 25\,145.7,\ 10\,855.6,$$
$$4686.49,\ 2023.21,\ 873.441,\ 377.074,\ 162.787,\ 70.2767,\ 30.3392,\ 13.0977,\ 5.65444,$$
$$2.44108,\ 1.05384,\ 0.454954,\ 0.196408,\ 0.0847915,\ 0.0366054,\ 0.0158029,\ 0.00682229\}$$

The final value in the list is 0.0068 surviving clonogens, which gives a voxel TCP value of 99.32%. For 100 voxels, the total tumor TCP is 0.9932^{100} or approximately 50%. Similar calculations were done for the other phases of the cell cycle, using the phase-specific VoxelAlpha values above. For voxels containing cycling cells, with each fraction, we randomly sampled for VoxelAlpha from a discrete distribution, assuming the cells are distributed in each phase of the cell cycle in proportion to the times spent in these phases. We use the *Mathematica* RandomChoice function shown in equation (44) as implemented here.

```
VoxelRandomChoice :=
  Table[RandomChoice[{0.3637, 0.0909, 0.2727, 0.2727} -> {0.75, 0.35, 0.25, 0.10}],
   {i, 10}, {j, 10}, {k, 1}];
```

This function picks a random VoxelAlpha value, using the discrete distribution shown, for all the voxels in a 10x10x1 dataset represented in a list made with the Table function. The := symbol is a delayed assignment operator, which causes the VoxelRandomChoice function to be reevaluated each time it is called, and this assures that with each fraction a new set of VoxelAlpha values are selected. This is used in the equation below to randomly sample the VoxelAlpha values with each fraction delivered.

```
VoxelSurvivingClonogens[VoxelClonogens_] :=
   VoxelClonogens * Exp[-VoxelRandomChoice * VoxelFractionalBED];
```

We show the results again for a single voxel starting with 600 million cells, and we show all the intermediate results again with the NestList function.

```
NestList[VoxelSurvivingClonogens, 600 000 000, 30]
```

$$\{600\,000\,000,\ 2.59026\times10^{8},\ 4.28168\times10^{7},\ 7.07756\times10^{6},\ 1.16991\times10^{6},\ 193\,385.,$$
$$106\,132.,\ 45\,818.4,\ 25\,145.7,\ 13\,800.2,\ 7573.72,\ 5957.7,\ 984.802,\ 540.471,\ 425.149,$$
$$70.2767,\ 55.2816,\ 30.3392,\ 16.6505,\ 9.13799,\ 1.5105,\ 0.82898,\ 0.454954,\ 0.0752033,$$
$$0.0412725,\ 0.00682229,\ 0.00374415,\ 0.00205483,\ 0.00112772,\ 0.000887094,\ 0.000697813\}$$

The final value in the list is 0.000698 surviving clonogens, which gives a voxel TCP value of 99.93%, which is higher than the 99.32% voxel TCP value above for G1 cells. The total tumor TCP in this case is 0.9993^{100} = 93.2% vs. the 50% value for the noncycling G1 cells. Thus the cycling

cells are seen to be more radiosensitive giving a favorable therapeutic ratio relative to the G1 cells. Further results are shown in Section 5.3.

4.5 Reoxygenation

Using the equations developed in Section 3.5., we model the effects of hypoxia, and subsequent reoxygenation, on tumor control in the voxels. We again use a hypothetical case of a 60 Gy conventional fractionation treatment in 2 Gy fractions, with weekend breaks, for a tumor containing 100 voxels. For this treatment, with VoxelAlphaBeta = 10 Gy, the VoxelBED value is 72 Gy and we saw above that this BED, with VoxelAlpha = 0.35 Gy^{-1}, can control about 60 billion clonogens at the 50% TCP level, or about 600 million clonogens per voxel. This is our fully-oxygenated baseline case, and we explore the effects that various levels of hypoxia and reoxygenation in the voxels have on tumor TCP.

For fractionated dose delivery, we begin with the *Mathematica* equation below that we used previously. For a starting number of clonogens equal to VoxelClonogens, this functional equation gives the surviving clonogens in each voxel after a single dose of radiation with value VoxelFractionalBED.

```
VoxelSurvivingClonogens[VoxelClonogens_] =
   VoxelClonogens * Exp[-VoxelAlpha * VoxelFractionalBED];
```

In hypoxic conditions, we modify VoxelAlpha and VoxelFractionalBED using equations (48) and (49), which depend on the OER values in the voxels. We use equation (50), implemented in *Mathematica* as shown below, to account for reoxygenation in the voxels. This equation gives OER as a function of time, starting at an initial OER value of VoxelInitialOER and decreasing exponentially with rate constant VoxelReoxygenationRate to a final value of OER = 1.

```
VoxelOER[t_] = (VoxelInitialOER - 1) * Exp[-VoxelReoxygenationRate * t] + 1;
```

Combining these equations together, we get the functional equation below, which is a function of starting clonogen number VoxelClonogens and time *t*. Once again, all the components of this equation are lists of values in the voxels.

```
VoxelSurvivingClonogens[VoxelClonogens_, t_] = VoxelClonogens *
       Exp[-(VoxelAlpha / (VoxelOER[t])) * (VoxelFractionalDose +
       VoxelFractionalPsi / (VoxelOER[t] * VoxelAlphaBeta))];
```

We use the *Mathematica* Fold function to implement this equation over the course of the treatment. Below are the results for a single voxel, with starting clonogen number equal to 600,000,000, and the treatment time in days given by the list of integers.

```
FoldList[VoxelSurvivingClonogens, 600 000 000,
 {0, 1, 2, 3, 4, 7, 8, 9, 10, 11, 14, 15, 16, 17, 18, 21, 22, 23, 24, 25, 28, 29, 30, 31,
  32, 35, 36, 37, 38, 39, 42, 43, 44, 45, 46, 49, 50, 51, 52, 53, 56, 57, 58, 59, 60}]
```

$\{600\,000\,000, 4.43425\times10^{8}, 3.24511\times10^{8}, 2.35137\times10^{8}, 1.68673\times10^{8}, 1.19772\times10^{8},$
$8.24513\times10^{7}, 5.61685\times10^{7}, 3.78635\times10^{7}, 2.52562\times10^{7}, 1.66697\times10^{7}, 1.06593\times10^{7},$
$6.74478\times10^{6}, 4.22343\times10^{6}, 2.61726\times10^{6}, 1.60526\times10^{6}, 955\,077., 562\,634., 328\,221.,$
$189\,635., 108\,529., 60\,426.6, 33\,348.6, 18\,246.3, 9899.23, 5326.49, 2799.21, 1460.12,$
$756.124, 388.806, 198.564, 99.466, 49.5252, 24.5155, 12.0671, 5.90738, 2.84817, 1.36674,$
$0.65288, 0.310513, 0.147061, 0.06884, 0.032109, 0.014925, 0.0069146, 0.00319332\}$

We start with 100 tumor voxels with VoxelAlpha = 0.35 Gy^{-1}. In four voxels (4% of total) we set the initial OER, VoxelInitialOER = 2.5, which gives an initial VoxelAlpha = 0.14 Gy^{-1} using equation (48). Over 40 days, these voxels reoxygenate (VoxelReoxygenationRate = 0.05 d^{-1}) and on day 40, VoxelAlpha = 0.29 Gy^{-1}. We also explore the effects on tumor TCP of various levels of hypoxic voxels (1%, 4%, 16%, and 100%). And for 4% hypoxic voxels, we also explore the effects of various reoxygenation rates on TCP of the tumor. The results are shown in Section 5.4.

4.6 Radiosensitivity and Heterogeneity

As discussed in Section 3.6, intrinsic radiosensitivity is characterized by the α parameter in the linear quadratic model. The α parameters in the voxels are contained in a list called VoxelAlpha, or in the case where α is constant throughout the voxels, VoxelAlpha can be a scalar. The voxel TCPs are obtained with equation (63) shown here.

```
VoxelTCP = Exp[-VoxelClonogens * Exp[-VoxelAlpha * VoxelBED]];
```

VoxelClonogens is the number of clonogens per voxel, and this too can be a list if the clonogen number varies voxel-to-voxel or it can be a scalar if this number is constant throughout the voxels. For conventional fractionation in 2 Gy fractions for a tumor with α/β = 10 Gy, this equation can be written as follows, where x is absorbed dose.

```
VoxelTCP = Exp[-VoxelClonogens * Exp[-VoxelAlpha * (x * 1.2)]];
```

For this treatment, 60 Gy absorbed dose gives a BED of 72 Gy, and as we saw before, with an α value of 0.35 Gy^{-1} this BED can control about 60 billion clonogens at the 50% TCP level. These 60 billion clonogens could be in one large voxel or distributed among many smaller voxels. Since we will be exploring the effect that heterogeneity in the α parameter throughout the voxels has on the total tumor TCP, and to obtain good statistical results, we will use a rather large dataset of voxels for this part of our investigation. We introduce a 40x40x40 dataset of voxels with 64,000 total voxels with the following equation.

```
VoxelTCP = Table[Exp[-VoxelClonogens * Exp[-VoxelAlpha * (x * 1.2)]],
          {i, 40}, {j, 40}, {k, 40}];
```

For a tumor with 60 billion clonogens, this will give 937,500 clonogens per voxel and therefore VoxelClonogens = 937,500.

To get the total tumor TCP, we multiply all the voxel TCPs together as shown below, where the {0,2} specifies the correct level for the multiplication operation in the multi-dimensional list structure.

```
TumorTCP = Apply[Times, VoxelTCP, {0, 2}];
```

Alternatively, the VoxelTCP multi-dimensional list structure can first be flattened into a one-dimensional list and the terms then multiplied together directly as shown here.

```
FlatVoxelTCP = Flatten[VoxelTCP];

TumorTCP = Apply[Times, FlatVoxelTCP];
```

The final result, TumorTCP, is a function of x, which is absorbed dose, and we can plot TumorTCP as a function of dose, which we do for VoxelAlpha equal to 0.25 Gy^{-1}, 0.35 Gy^{-1}, and 0.45 Gy^{-1}. These results are for the case of VoxelAlpha values held constant throughout the voxels. We also explore the effect of variation in the VoxelAlpha values throughout the voxels. Using a normal distribution with α = 0.35 Gy^{-1} (VoxelAlphaMu=0.35), and standard deviation = 0.035 Gy^{-1} (VoxelAlphaSD=0.035), we form a 40x40x40 dataset of voxels with a distribution of α values.

```
VoxelTCP = Table[Exp[-VoxelClonogens *
          Exp[-RandomVariate[NormalDistribution[VoxelAlphaMu, VoxelAlphaSD]] *
          (x * 1.2)]], {i, 40}, {j, 40}, {k, 40}];
```

Similarly, we can use a lognormal distribution with a mean α of 0.35 Gy^{-1} and standard deviation of 0.035 Gy^{-1}, by applying the transformations of equations (55) and (56). These transformations give a value for VoxelAlphaMu of -1.055 and for VoxelAlphaSD of 0.09975.

```
VoxelTCP = Table[Exp[-VoxelClonogens *
          Exp[-RandomVariate[LogNormalDistribution[VoxelAlphaMu, VoxelAlphaSD]] *
          (x * 1.2)]], {i, 40}, {j, 40}, {k, 40}];
```

We also model a population average curve by generating individual curves with α parameters ranging from 0.29 Gy^{-1} to 0.43 Gy^{-1} (0.290, 0.300, 0.314, 0.328, 0.345, 0.360, 0.378, 0.395, 0.413, 0.430) and then taking the mean of the resulting equations (i.e., adding them together and dividing by ten). The results are shown in Section 5.5.

4.7 Clonogen Number and Dose Heterogeneity

We investigated the effect of total clonogen number and clonogen number heterogeneity on tumor TCP starting with the following equation for TCP in a voxel.

```
VoxelTCP = Exp[-VoxelClonogens * Exp[-VoxelAlpha * VoxelBED]];
```

For conventional fractionation in 2 Gy fractions for a tumor with α/β = 10 Gy, this equation can

be written as follows, where x is absorbed dose.

```
VoxelTCP = Exp[-VoxelClonogens * Exp[-VoxelAlpha * (x * 1.2)]];
```

For 100 identical voxels, the tumor TCP is obtained with the following equation.

```
TumorTCP = Exp[-VoxelClonogens * Exp[-VoxelAlpha * (x * 1.2)]] ^ 100;
```

The effect on TCP of voxel clonogen number heterogeneity can be modeled using either a normal or lognormal distribution as described in Section 3.7. For a tumor made up of 100 voxels, a list of voxel TCPs can be generated using the following equation where the mean voxel clonogen number is VoxelClonogensMu and the standard deviation is VoxelClonogensSD.

```
VoxelTCP =
     Table[Exp[-RandomVariate[NormalDistribution[ VoxelClonogensMu, VoxelClonogensSD]] *
     Exp[-VoxelAlpha * (x * 1.2)]], {i, 10}, {j, 10}, {k, 1}];
```

The individual voxel TCPs can be multiplied together as before to get the tumor TCP.

```
TumorTCP = Apply[Times, VoxelTCP, {0, 2}];
```

We plot TumorTCP versus dose for total tumor clonogen values of 60 million, 60 billion, and 60 trillion, and we discuss the effect of heterogeneity on these results in Section 5.6.

We also explored heterogeneity in dose in various volumes of a tumor. We used a hypothetical situation with 100 voxels and we decreased the dose in various fractions of these voxels. For conventional fractionation in 2 Gy fractions for an individual with VoxelAlpha = 0.35 Gy^{-1} we can control about 60 billion clonogens at the 50% TCP level, or about 600 million clonogens per voxel in this case.

For example, for a 20% volume dose deficit with VoxelClonogens = 600,000,000 and VoxelAlpha = 0.35 Gy^{-1}, we used the following equation.

```
TumorTCP = Exp[-VoxelClonogens * Exp[-VoxelAlpha * 72]] ^ 80 *
           Exp[-VoxelClonogens * Exp[-VoxelAlpha * (x * 1.2)]] ^ 20;
```

That is, we keep 80 voxels at a baseline value of BED 72 Gy, and we vary the dose in 20 of the voxels. For dose deficits of 0.1-20% less than the 60 Gy full dose, we calculated how much the TCP was reduced from the baseline of 50% TCP.

As we will see in Section 5.5, TCP curves for individual patients are quite steep, with a γ_{50} of about 8. For a population of patients, the population average TCP curve has a γ_{50} of about 2. We performed a similar analysis of the reduction in tumor TCP for various underdoses of tumor volume for a population rather than an individual by using the population average TCP curve.

4.8 Tumor Control Probability

In Section 5.7 we show the results for a hypothetical case of multiple effects on tumor TCP. We combined many of the effects modeled in previous sections into a hypothetical investigation of TCP in a tumor with 100 voxels. We again used a conventional fractionation case of 60 Gy given in 2 Gy fractions for a tumor with α/β = 10 Gy. This gives a BED of 72 Gy. With 600,000,000 clonogens per voxel and with $\alpha = 0.35\ Gy^{-1}$ the TCP per voxel is 0.993 giving a total TCP for the tumor of $0.993^{100} \approx 50\%$. This is our starting baseline TCP value. We then add variation to the α value and add hypoxia and subsequent reoxygenation to 4% of the voxels, and add proliferation to the voxels on the edges of the tumor. Finally, we add a 10% dose deficit to 3% of the voxels. And we add variation (from normal or lognormal distributions) to most of these values. The results are shown in Figure 12 of Section 5.7.

5. Functional Results

The results obtained for the convolutions used in the calculations of BED were checked against analytical formulations. In Gustafsson (2013) these analytic formulations are given by

$$BED = D_T + \frac{1}{\alpha/\beta}[\psi(\Xi, \mu)]$$

where for discrete pulse cases like the Gamma Knife case or conventional fractionation ψ takes the following form

$$\psi(\Xi, \mu) = \frac{2}{\mu}\sum_{i=1}^{n}\left\{\left[\frac{d_i}{\tau_i}\right]^2\left[\tau_i - \frac{1}{\mu}(1 - \exp\{1 - \mu\tau_i\})\right] - \frac{1}{\mu}\sum_{j=1}^{i-1}\frac{d_i d_j}{\tau_i \tau_j}\exp\left[-\mu(t_i - t_j)\right]\left[\exp(\mu\tau_j) - 1\right]\left[\exp(-\mu\tau_i) - 1\right]\right\}$$

where for the ith fraction, d_i is the absorbed dose delivered at a constant dose rate, τ_i is the fraction duration, and a fraction starts at time t_i and ends at time $t_i + \tau_i$.

For a single fraction given at an exponentially decaying absorbed dose rate (e.g., I-125), the BED is given by

$$BED(T) = D_T + \frac{D_T[D_T \cdot G(T)]}{(\alpha/\beta)}$$

where

$$G(T) = \frac{2\lambda^2}{[1 - \exp(-\lambda T)]^2}\frac{1}{\mu - \lambda}\left\{\frac{1 - \exp(-2\lambda T)}{2\lambda} - \frac{1 - \exp[-(\mu + \lambda)T]}{\mu + \lambda}\right\}$$

Here we can see that

$$\psi = D_T[D_T \cdot G(T)]$$

As we have discussed in Section 4.2, there are two ways to write the convolution integrals to calculate ψ. One form uses the full integration with the dose-rate input functions written in terms of both w and t, whereas the other form just writes the dose-rate input function in terms of t.

For the discrete pulse cases, writing the convolution integral just in terms of dose-rate input functions in t gives identical results as to when they are written in terms of both w and t and the calculation is significantly faster. For the exponentially decaying absorbed dose rate case (brachytherapy seed implants or radionuclide therapy), the results are nearly identical but there is a slight discrepancy. To review, for the discrete pulse type irradiation where the dose-rate is constant during the fractions, for a three shot (or three fraction) case, we write the ψ equation as

follows and call it VoxelPsiT. The dose-rate input function is only written as a function of t and is called VoxelDoseRateInputT. This is mapped over the convolution integral.

```
VoxelPsiT =
  ParallelMap[ 2 * (
       (* shot 1 *)
       Integrate[# * Exp[-VoxelMu * t] * (Integrate[# * Exp[VoxelMu * w], {w, t1, t}]),
        {t, t1, t2}] +
       (* shot 2 *)
       Integrate[# * Exp[-VoxelMu * t] * (Integrate[# * Exp[VoxelMu * w], {w, t2, t}] +
           Integrate[# * Exp[VoxelMu * t], {t, t1, t2}]), {t, t2, t3}] +
       (* shot 3 *)
       Integrate[# * Exp[-VoxelMu * t] * (Integrate[# * Exp[VoxelMu * w], {w, t3, t}] +
           Integrate[# * Exp[VoxelMu * t], {t, t1, t3}]), {t, t3, t4}])
    &, VoxelDoseRateInputT];
```

This is the same as writing the equation in the following way, although this form of the equation is not parallelizable like with the ParallelMap function above.

```
VoxelPsiT =
     2 * (
     (* shot 1 *)
    Integrate[VoxelDoseRateInputT * Exp[-VoxelMu * t] *
       (Integrate[VoxelDoseRateInputT *
          Exp[VoxelMu * w], {w, t1, t}]), {t, t1, t2}] +
     (* shot 2 *)
     Integrate[VoxelDoseRateInputT * Exp[-VoxelMu * t] *
       (Integrate[VoxelDoseRateInputT *
           Exp[VoxelMu * w], {w, t2, t}] +
         Integrate[VoxelDoseRateInputT * Exp[VoxelMu * t], {t, t1, t2}]),
      {t, t2, t3}] +
     (* shot 3 *)
     Integrate[VoxelDoseRateInputT * Exp[-VoxelMu * t] *
       (Integrate[VoxelDoseRateInputT * Exp[VoxelMu * w], {w, t3, t}] +
         Integrate[VoxelDoseRateInputT * Exp[VoxelMu * t], {t, t1, t3}]),
      {t, t3, t4}]);
```

As we have noted, these forms of the convolutions work accurately for discrete constant pulse-rate treatments (e.g., Gamma Knife, conventional fractionation, HDR, LDR, PDR treatments).

For the Gamma Knife case the VoxelPsiT form of the equation gives, for slice nine, row six of the voxels

```
VoxelPsiT[[9, 6]]
```

```
{103.731, 242.476, 483.51, 716.878, 783.907, 725.268, 543.017, 317.259, 167.633}
```

Whereas the VoxelPsiW form of the equation gives the exact same results, which translates into identical BEDs.

```
VoxelPsiW[[9, 6]]
```

```
{103.731, 242.476, 483.51, 716.878, 783.907, 725.268, 543.017, 317.259, 167.633}
```

So when the dose-rate is constant over the fractions (or shots in Gamma Knife terminology), VoxelPsiT gives the same results as VoxelPsiW and is much faster. It is only when the dose-rate input functions vary with time as, for example, in radioactive seed implants or radionuclide therapy that for exact results you need integration in terms of dose-rate functions written both in terms of w and t as below which we call VoxelPsiW.

```
VoxelPsiW =
     (* shot 1 *)
  Parallelize[
   (2 Integrate[VoxelDoseRateInputT * Exp[-VoxelMu * t]
        (Integrate[VoxelDoseRateInputW *
           Exp[VoxelMu * w], {w, t1, t}]), {t, t1, t2}] +
     (* shot 2 *)
     2 Integrate[VoxelDoseRateInputT * Exp[-VoxelMu * t] *
        (Integrate[VoxelDoseRateInputW * Exp[VoxelMu * w], {w, t2, t}] +
          Integrate[VoxelDoseRateInputW *
            Exp[VoxelMu * w], {w, t1, t2}]), {t, t2, t3}] +
     (* shot 3 *)
     2 Integrate[(VoxelDoseRateInputT) * Exp[-VoxelMu t] *
        (Integrate[(VoxelDoseRateInputW) * Exp[VoxelMu (w)], {w, t3, t}] +
          Integrate[(VoxelDoseRateInputW) *
            Exp[VoxelMu (w)], {w, t1, t3}]), {t, t3, t4}])];
```

When the dose-rate varies with time, as in radionuclide therapy, it makes a slight difference which form of the equation is used. As shown in Appendix S, for VoxelIodinePsiT we get

```
VoxelIodinePsiT =
 2 Integrate[VoxelDoseRateIodineT * Exp[-Mu t] *
    (Integrate[VoxelDoseRateIodineT * Exp[Mu (w)], {w, 0, t}]), {t, 0, t1}]
```

```
14.5251
```

whereas for VoxelIodinePsiW we get

```
VoxelIodinePsiW =
 2 Integrate[(VoxelDoseRateIodineT) * Exp[-Mu t] *
    (Integrate[(VoxelDoseRateIodineW) * Exp[Mu (w)], {w, 0, t}]), {t, 0, t1}]
```

```
14.5353
```

which makes a slight difference in the BED of 148.842 Gy vs. the correct value of 148.845 Gy. This is further discussed in Appendices S and T.

5.1. Repair

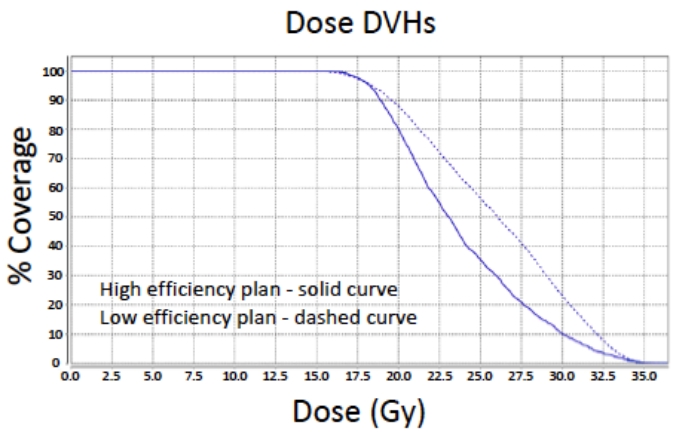
The dose-rate input functions shown in Figure 2 were all implemented with the methods described in Section 3.2, and these functions were convolved with repair functions such as those shown in Figures 3a and 3b to obtain the BEDs for each of the cases. The BED values obtained agree with those obtained with the analytical methods described above. Some of these results are shown in this chapter and in the appendices. Figure 4 shows one slice of the dataset for the Gamma Knife radiosurgery case discussed in Section 4.2, showing the dose-rate input functions in each voxel along with the computed BED in the voxels. The complete *Mathematica* code for this case is shown in Appendix J.

We analyzed another Gamma Knife case, this time from a 35x34x34 dataset shown in Figure 5. There were two cases, one was for a high efficiency (eight shot) Gamma Knife case and one for a low efficiency (thirteen shot) case. The high efficiency case had a total time of 33.9 minutes while the low efficiency case was 79.3 minutes. In calculating the tumor voxel BEDs, a biexponential repair function was used, with short and long repair half-times of 0.19 h and 2.16 h, respectively, partition coefficient c=0.98 (giving a=0.5051, b=0.4949), and the α/β value was 10 Gy. The prescription dose was 18 Gy to the 50% isodose line. One of the purposes of this exercise was to go through the complete process of determining which voxels lied within the contours (Appendix E), creating a list of shot times and shot dose-rates in each voxel (Appendix H), importing that list into *Mathematica* (Appendix I), processing the data to calculate the voxel BEDs as in Appendix J, exporting the results out of Mathematica (Appendix K), and finally converting the original DICOM dose file to a DICOM BED file (Appendix L) to display the results on the treatment planning computer.

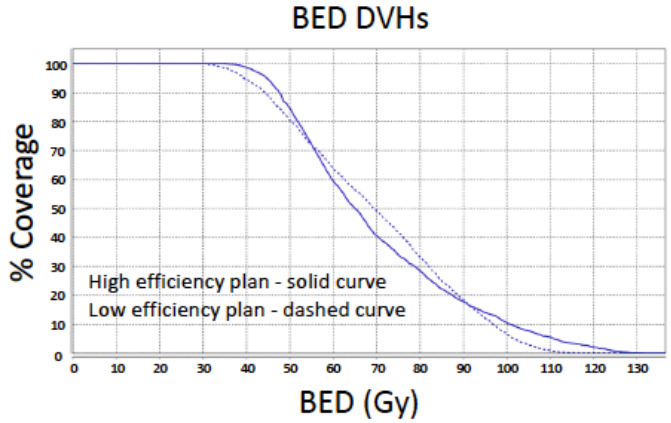
The original DVH is shown in Figure 5a for the high efficiency and low efficiency plans, while the BED DVHs are shown in Figure 5b. Because of the short repair half-time component, the longer low efficiency case the BED gets shifted to the left relative to the high efficiency case. Millar et al. (2015) show similar results for a 13 shot case relative to a 3 shot case

Finally, we explored the loss in BED for the high efficiency case when the Co-60 activity was half its previous value, and the treatment time therefore twice as long. As shown in Figure 5c, we see a decrease in BED of about 10% at the lower activity and longer treatment time due to more repair with the shorter repair half-time. Kann et al. (2016) show about a 12% decrease in BED for an 85 Gy trigeminal neuralgia case at half the Co-60 activity, although they use slightly different DNA repair half-times and an α/β of 2 Gy. From this analysis it can be seen that the BED decreases about 2% per year over the lifetime of the Co-60 source.

(a)



(b)



(c)

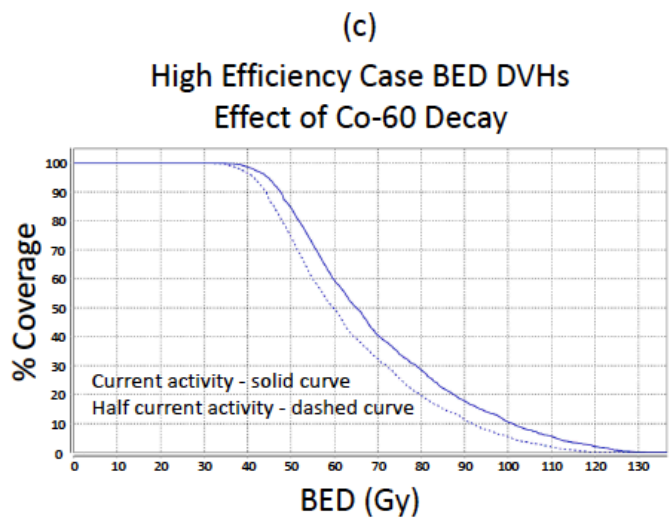


Figure 5. (a) The absorbed dose DVHs for the high efficiency (8 shot) Gamma Knife case and the low efficiency (13 shot) case. (b) The BED DVHs for the same cases. (c) BED DVHs showing loss of BED at half Co-60 activity (and therefore twice the treatment time).

5.2 Repopulation

Figure 6a shows, for an individual voxel, the TCD-50 plotted against treatment duration for the progressive and discontinuous repopulation models developed in Section 3.3. For the progressive repopulation model, results are shown for T_{pot} values of 2.5, 5, 7.5, and 10 days, the shorter the T_{pot} value, the faster the repopulation. For the discontinuous repopulation model, Figure 3c shows the voxel repopulation rate for this model which was discussed in Section 3.3,

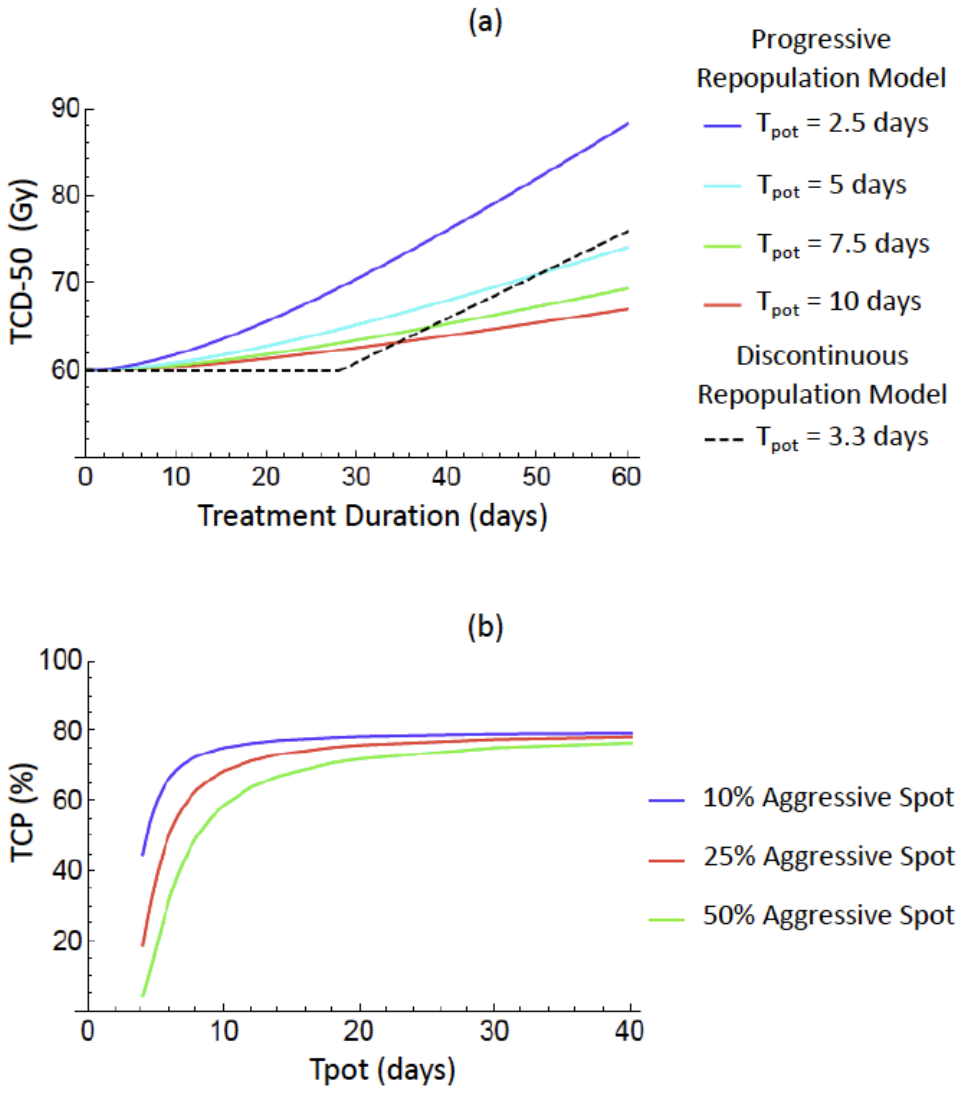


Figure 6. (a) The TCD-50 (50% tumor control dose) is plotted against treatment duration for various values of T_{pot} for the progressive repopulation model (with $\alpha = 0.35\ Gy^{-1}$; pretreatment cell loss factor $\phi_0 = 0.9$; cell loss rate constant $\nu = 0.05\ d^{-1}$), and for the discontinuous repopulation model (with $\alpha = 0.35\ Gy^{-1}$ and $T_{pot} = 3.3$ days) giving a value of 0.6 Gy/day lost to repopulation after a "kick-off" time of 28 days. (b) TCP as a function of T_{pot} for various volumes of aggressive spots starting from a baseline value of 80% TCP.

where with VoxelTpot = 3.3 days, VoxelAlpha = 0.35 Gy^{-1}, and VoxelInitialCellLoss = 0, the loss in BED is 0.6 Gy/day after a 28 day delay, which is a well-known case with accelerated repopulation.

Figure 6b shows TCP as a function of T_{pot} for various volumes of aggressive spots (where repopulation is occurring) starting from a baseline value of 80% TCP for large T_{pot} values (and little repopulation). Below a T_{pot} of about 15 days, the effect of an aggressive spot on TCP becomes substantial. Similar results are shown in Wang and Allen (2005).

5.3 Redistribution

In Figure 7 are results for the redistribution model developed in Section 3.4. The TCP of the tumor is plotted as a function of dose for the cases where the voxels contain cells exclusively in each of the various phases of the cell cycle. The case with the cells of the voxels all in G1 phase (with $\alpha = 0.35\ Gy^{-1}$) gives the baseline value of 50% TCP at 60 Gy physical dose. The case where the cell cycle phase of the individual voxels is chosen from a distribution is shown in the dashed red curve.

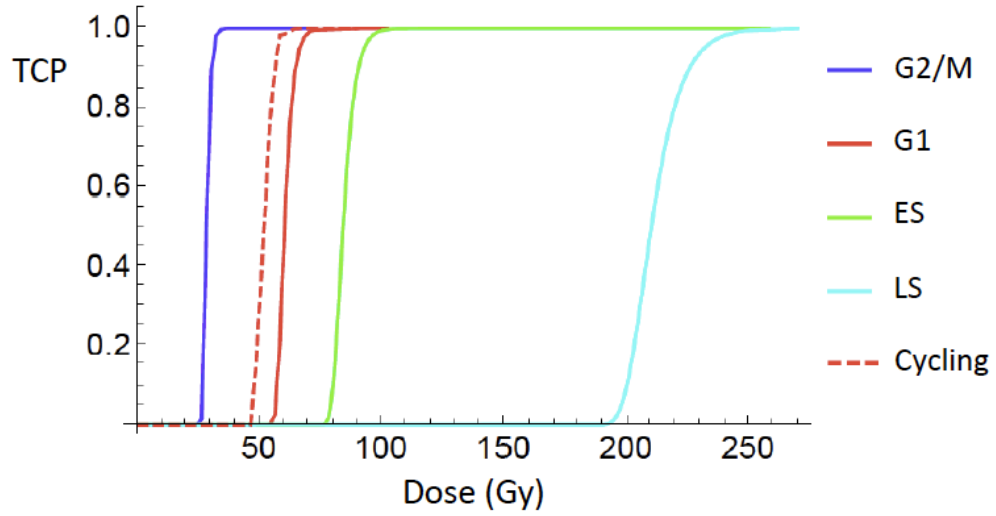


Figure 7. Effect of cell cycle on TCP for conventional fractionation in 2 Gy fractions, along with the TCP curve for cycling cells. G2/M, G1, ES (Early S), and LS (Late S) voxels have α coefficients of 0.75 Gy^{-1}, 0.35 Gy^{-1}, 0.25 Gy^{-1}, and 0.10 Gy^{-1}, respectively.

We can see that the cycling cells have a therapeutic advantage relative to cells in G1 phase, so that presumably cycling tumor cells would be more sensitive than surrounding normal tissue that is largely in G1 phase, or even out of cycle in G0. It has been noted in Perez and Brady

(Wazer et al. 2009) that the difference in sensitivity between cells in M/G2 phase and those in late S phase is greater than that between well oxygenated and hypoxic tissues, which we can see by comparing Figure 7 to Figure 9a.

5.4 Reoxygenation

Figure 8a shows 100 tumor voxels on the first day of treatment with four of the voxels being hypoxic (4% hypoxia) and having an α_H value of 0.14 Gy^{-1} and it also shows the voxels on day 40 of treatment where these hypoxic voxels have partially reoxygenated as described in Section 3.5,

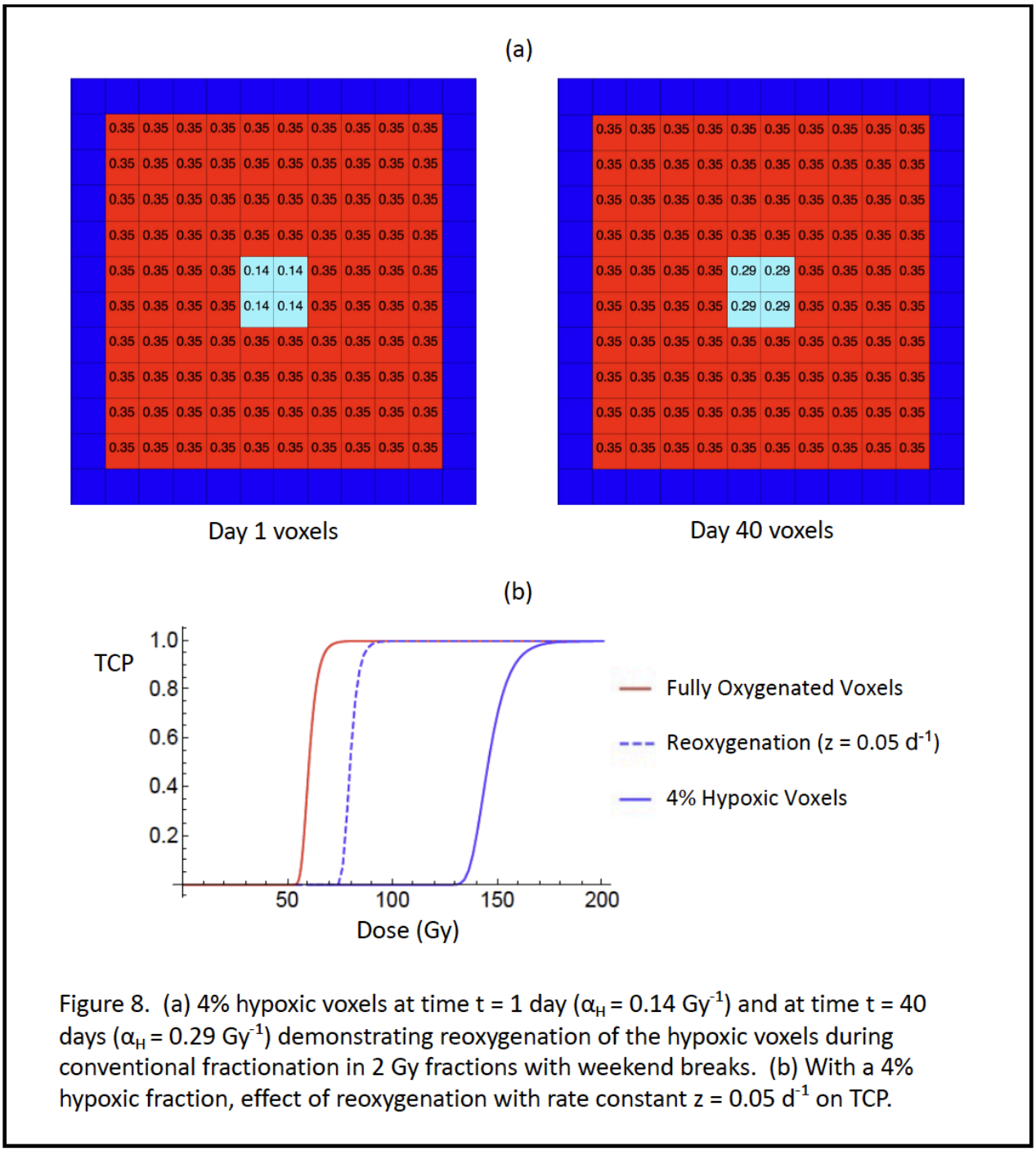


Figure 8. (a) 4% hypoxic voxels at time t = 1 day (α_H = 0.14 Gy^{-1}) and at time t = 40 days (α_H = 0.29 Gy^{-1}) demonstrating reoxygenation of the hypoxic voxels during conventional fractionation in 2 Gy fractions with weekend breaks. (b) With a 4% hypoxic fraction, effect of reoxygenation with rate constant z = 0.05 d^{-1} on TCP.

where they now have a α_H value of 0.29 Gy^{-1}. Figure 8b shows, for the case of a 4% hypoxic fraction, the effect on TCP of reoxygenation with rate constant z = 0.05 d^{-1}. Figure 9a demonstrates the effect of hypoxic fraction (1% - 100% hypoxic voxels) on TCP for conventional fractionation in 2 Gy fractions with weekend breaks. We can see how even 1% hypoxic voxels can severely effect TCP. Figure 9b shows, for a 4% hypoxic fraction, the effect of reoxygenation rate on TCP.

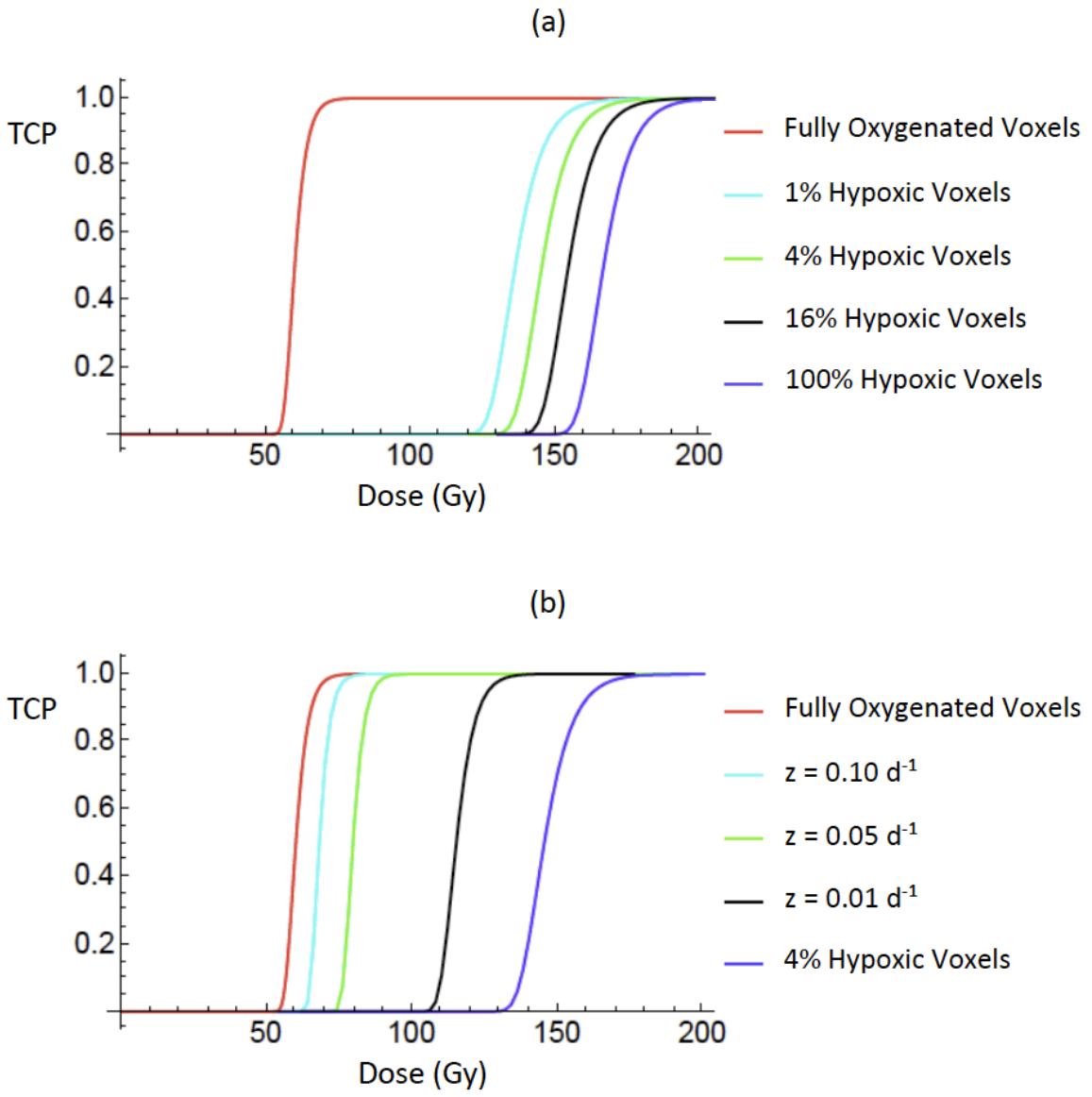


Figure 9. (a) Effect of hypoxic fraction (1% - 100% hypoxic voxels) on TCP for conventional fractionation in 2 Gy fractions with weekend breaks. (b) With a 4% hypoxic fraction, effect of reoxygenation rate z (decimal % per day) on TCP.

5.5 Radiosensitivity and Heterogeneity

Figure 10a demonstrates the effect on TCP of intrinsic radiosensitivity, which is characterized by the α coefficient in the linear quadratic model. Shown are TCP curves using α values of 0.25 Gy^{-1}, 0.35 Gy^{-1}, and 0.45 Gy^{-1}. For a mean α value of 0.35 Gy^{-1} and standard deviation of 0.035 Gy^{-1}, the effect of sampling α from a normal distribution and a lognormal distribution are also shown.

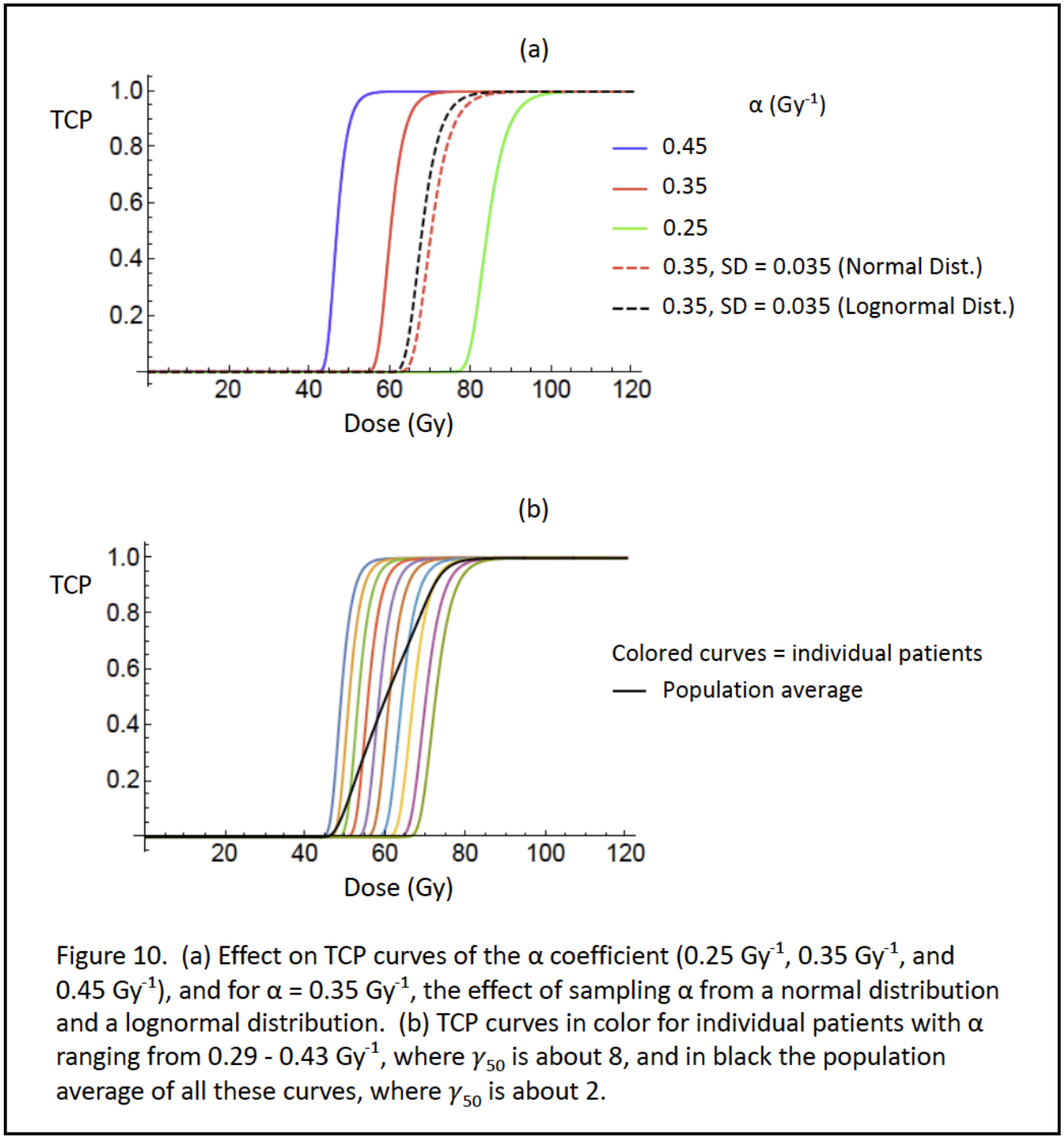


Figure 10. (a) Effect on TCP curves of the α coefficient (0.25 Gy^{-1}, 0.35 Gy^{-1}, and 0.45 Gy^{-1}), and for α = 0.35 Gy^{-1}, the effect of sampling α from a normal distribution and a lognormal distribution. (b) TCP curves in color for individual patients with α ranging from 0.29 - 0.43 Gy^{-1}, where γ_{50} is about 8, and in black the population average of all these curves, where γ_{50} is about 2.

The lognormal distribution is skewed toward the right in terms of α values and so the TCP curve is shifted further to the left than for the normal distribution case. Figure 10b shows TCP curves

for individual patients with α values ranging from 0.29 - 0.43 Gy^{-1}, and it also shows the population average of all of these curves in black. Individuals vary quite a lot in radiosensitivity and the individual curves are quite steep (γ_{50} = about 8) but when averaged over a population the average γ_{50} is about 2.

5.6 Clonogen Number and Dose Heterogeneity

Figure 11a shows the effect of clonogen number on TCP for conventional fractionation in 2 Gy fractions with α = 0.35 Gy^{-1}. The baseline curve of 50% TCP at 60 Gy physical dose (72 Gy BED) for 60 billion clonogens is shown in red. TCP curves for 60 million and 60 trillion clonogens are also shown. The clonogen number has to change by quite a bit for it to have significant influence on TCP, and clonogen heterogeneity about a mean doesn't have too much of an effect on the TCP. For α = 0.35 Gy^{-1} a factor increase of 1000 clonogens is controlled by about 20 Gy BED.

$$\frac{\ln(Factor\ Increase\ in\ Clonogens\ Controlled)}{\alpha} = \text{BED}$$

Figure 11b shows the effect on TCP of underdose of a portion of the tumor for an individual patient with an α = 0.35 Gy^{-1}. The γ_{50} is about 8 for these individuals which is reflected in a large change in TCP for an underdose.

Figure 11c shows the effects on TCP on an underdose for a population average value of γ_{50} of about 2. This is more consistent with the often quoted value of not more than 10% underdose to not more than 10% of the tumor volume, should not reduce the TCP by as much as 10% (Goitein et al. 1995, Tome and Fowler, 2000).

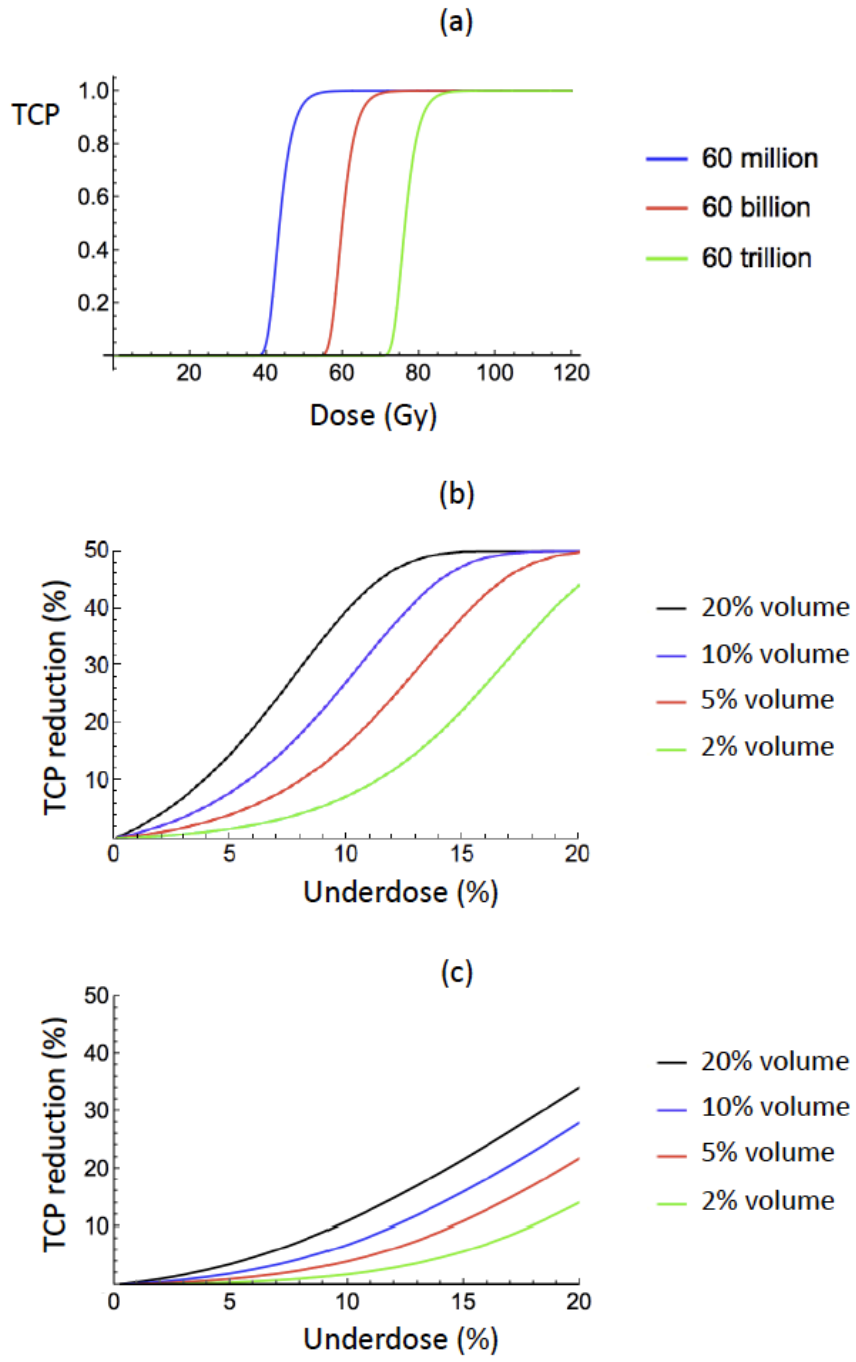


Figure 11. (a) Effect of clonogen number on TCP for conventional fractionation in 2 Gy fractions with $\alpha = 0.35\ Gy^{-1}$. (b) Decrease in TCP due to a dose deficit in various volumes of the tumor versus the level of the underdose. This is for an individual patient with $\alpha = 0.35\ Gy^{-1}$ and a γ_{50} of about 8. (c) Similar curves for a population average TCP curve having a γ_{50} of about 2.

5.7 Tumor Control Probability

We combined many of the effects modeled in previous sections into a hypothetical investigation of TCP in a tumor with 100 voxels. We again used a conventional fractionation case of 60 Gy given in 2 Gy fractions for a tumor with α/β = 10 Gy. This gives a BED of 72 Gy. With 600,000,000 clonogens per voxel and with $\alpha = 0.35\ Gy^{-1}$ the TCP per voxel is 0.993 giving a total TCP for the tumor of $0.993^{100} \approx 50\%$. This is our baseline value. We first add variation to α (using a normal distribution with standard deviation of $0.015\ Gy^{-1}$) which gives variation to the individual voxel TCP values about the mean value of 0.993, which is shown in the red voxels. We then add hypoxia and subsequent reoxygenation to 4% of the voxels shown in the light blue voxels (reoxygenation rate $z = 0.2\ d^{-1}$ sampled from a normal distribution with standard deviation of $0.02\ d^{-1}$). We next add proliferation in the green voxels using the progressive repopulation model (T_{pot} = 10 days sampled from a lognormal distribution with standard deviation of 1.5 days; cell loss rate constant $\nu = 0.06\ d^{-1}$ sampled from a normal distribution with standard deviation of $0.01\ d^{-1}$; and a pretreatment cell loss factor of 0.88 sampled from a normal distribution with standard deviation of 0.02). Finally, a 10% dose deficit in applied to the three voxels shown in yellow. This gives a final tumor TCP to 2.5%. This is the value obtained when multiplying all of the individual voxel TCPs together.

This is an example of what can be done with this functional approach to problems of this type. We can explore the effects of repair, repopulation, redistribution, reoxygenation, and radiosensitivity in individual voxels or in many voxels at the same time. By combining functions together in compositions of functions

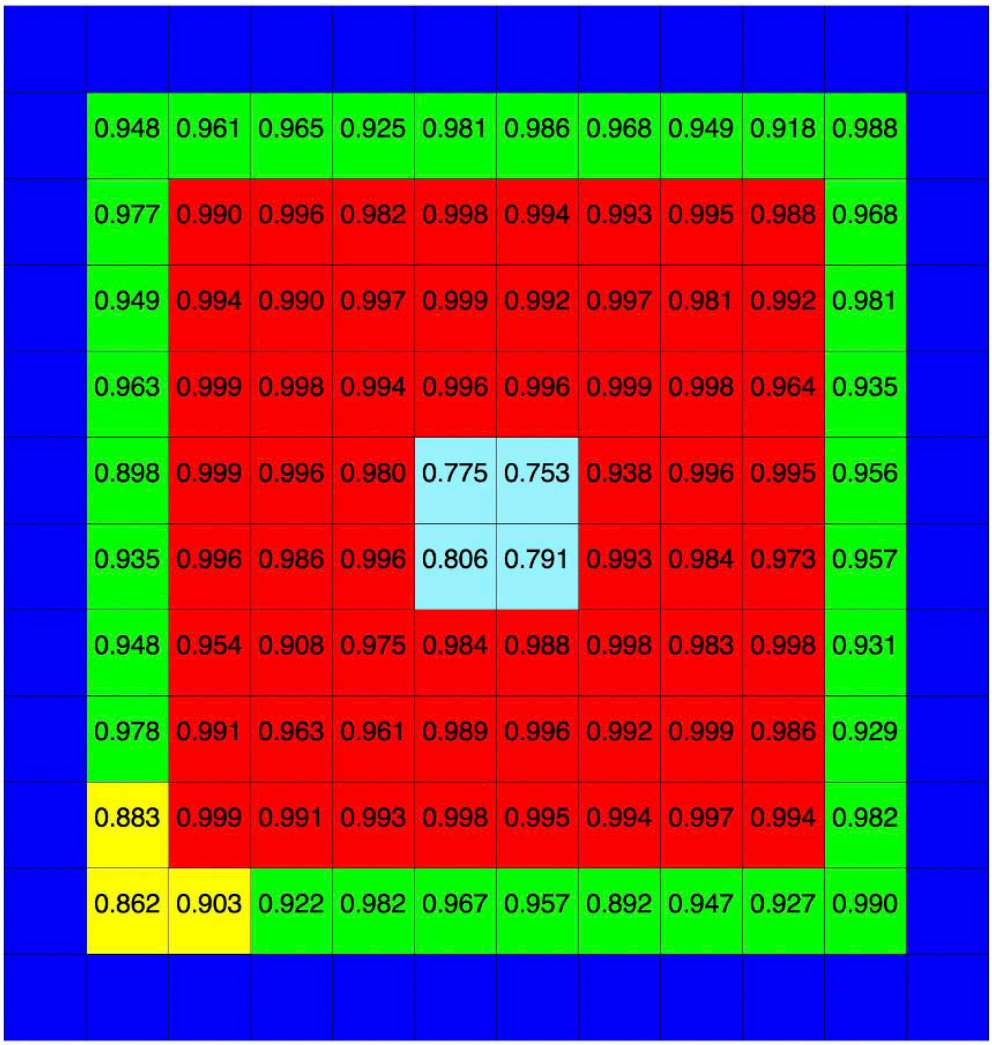


Figure 12. Hypothetical case with 100 voxels in a tumor and 600,000,000 clonogens per voxel. Conventional fractionation in 2 Gy fractions to 60 Gy gives a BED of 72 Gy. With $\alpha = 0.35\ Gy^{-1}$, the TCP per voxel is 0.993 giving a total TCP for the tumor of $0.993^{100} \approx 50\%$. Adding variation to α (using a normal distribution with standard deviation of $0.015\ Gy^{-1}$) reduces the tumor TCP to 29.3%. Voxels with these TCP values are shown in red. With hypoxia and subsequent reoxygenation in the light blue voxels (reoxygenation rate $z = 0.2\ d^{-1}$ sampled from a normal distribution with standard deviation of $0.02\ d^{-1}$) the tumor TCP drops to 11.2%. Proliferation in the green voxels using the progressive repopulation model (T_{pot} = 10 days sampled from a lognormal distribution with standard deviation of 1.5 days; cell loss rate constant ν = $0.06\ d^{-1}$ sampled from a normal distribution with standard deviation of $0.01\ d^{-1}$; and a pretreatment cell loss factor of 0.88 sampled from a normal distribution with standard deviation of 0.02), reduces the tumor TCP to 3.2%. Finally, adding a dose deficit of 10% to the voxels in yellow drops the total tumor TCP to 2.5% This is the value obtained when multiplying all of the individual voxel TCPs together.

6. Conclusions

We have demonstrated the modeling of the 5Rs of radiobiology using functional programming with *Mathematica*. Functional programming is natural for this problem, where all the models are represented as functions and these functions can be linked together as compositions of functions.

We developed a general convolution model to calculate the voxel-by-voxel BED for any absorbed dose-rate input function, and then we added to this model models for repopulation, redistribution, reoxygenation, and radiosensitivity. These models can be combined in any way to get the composite BED in the voxels and from this the voxel TCPs and thence the TCP in the tumor as a whole.

In functional programming the programmer focuses on the big picture and the results desired, and uses higher-level abstractions and mathematical reasoning in constructing a program from a composition of functions. This is a natural way to think about our problem, we want to map functions over the voxels to achieve certain goals, those goals here being to account for biological effects affecting the BED in the voxels. Functional programming is thought of as a mathematical activity, with the primary role of the programmer being to construct a function to solve a given problem, and the primary role of the computer is to act as an evaluator or calculator, its job being to evaluate expressions. (Bird, 2006).

The models developed here are rather fundamental and simple and more sophisticated models could be developed including models of normal tissue complication probability. With functional programming being essentially mathematical programming whatever can be thought of mathematically can be implemented into a functional program. This gives a lot of flexibility as to the possibilities available for both investigational and educational purposes.

Functional programming is a different programming paradigm than traditional imperative programming. One of the hallmarks of functional programming is the presence of powerful abstractions that hide many of the details of mundane operations such as iteration (Ford, 2014). This generally results in shorter, easier to read programs, and functional programming has been referred to as more of a mindset than a particular set of tools or languages (Ford, 2014). With all the unique characteristics it has, functional programming is as much a joy to use as it is a powerful tool for exploration.

References

Barendregt H, Manzonetto G, Plasmeijer R (2011). The Imperative and Functional Programming Paradigm. Preprint submitted to Alan Turing – His Work and Impact. 1-8.

Bird R, Wadler P (2006). Introduction to Functional Programming. Prentice Hall.

Bodey R, Evans M, Flux G (2004). Application of the Linear-Quadratic Model to Combined Modality Radiotherapy. *Int. J. Radiation Oncology Biol. Phys*. 59(1):228-241

Brenner D, Hlatky L, Hahnfeldt P, Huang Y, Sachs R (1998). The Linear-Quadratic Model and Most Other Common Radiobiological Models Result in Similar Predictions of Time-Dose Relationships. *Radiation Research*. 150:83-91

Brenner D (2008). The Linear-Quadratic Model Is an Appropriate Methodology for Determining Isoeffective Doses at Large Doses Per Fraction. *Semin Radiat Oncol*. 18:234-239

Carlson D, Stewart R, Semenenko A (2006). Effects of oxygen on intrinsic radiation sensitivity: A test of the relationship between aerobic and hypoxic linear-quadratic (LQ) model parameters. *Med. Phys*. 33(9):3105-3115

Dale R, Jones B (2007). Radiobiological Modelling in Radiation Oncology. The British Institute of Radiology.

Davis M (2000). The Universal Computer: The Road from Leibniz to Turing. W.W. Norton.

Ford N (2014). Functional Thinking: Paradigm Over Syntax. O'Reilly Media.

Fowler J, Dasu A, Toma-Dasu I (2014). Optimum Overall Treatment Time in Radiation Oncology. Medical Physics Publishing.

Goitein M, Niemierko A, Okunieff P (1995). The probability of controlling an inhomogeneously irradiated tumour: a stratagem for improving tumour control through partical tumour boosting. Quantitative Imaging in Oncology: Proceedings of the 19th LH Gray Conference. *British Institute of Radiology*. Chapter 2:25-39.

Goitein M (2008). Radiation Oncology: A Physicist's-Eye View. Springer.

Gustafsson J, Nilsson P, Gleisner KS (2013). On the biologically effective dose (BED)—using convolution for calculating the effects of repair: I. Analytical considerations. *Phys. Med. Biol*. 58 1507-1527.

Gustafsson J, Nilsson P, Gleisner KS (2013 II). On the biologically effective dose (BED)—using convolution for calculating the effects of repair: II. Numerical considerations. *Phys. Med. Biol*. 58 1529-1548.

Hall E, Giaccia A (2019). Radiobiology for the Radiologist. Wolters Kluwer.

Hopewell J, Millar W, Lindquist C (2012). Radiobiological Principles: Their Application to Gamma Knife Therapy. *Prog Neurol Surg*. 25:39-54.

Hu G, Hughes J, Wang M (2015). How functional programming mattered. *National Science Review*. 2:349-370.

Joiner M, van der Kogel A (2018). Basic Clinical Radiobiology. CRC Press

Kann BH, et al. (2016). The impact of cobalt-60 source age on biologically effective dose in high-dose functional Gamma Knife radiosurgery. *J Neurosurg*. 125:154-159

Lea D, Catcheside D (1942). The mechanism of the induction by radiation of chromosome aberrations in tradescantia. *J Genet*. 44:216–245.

Limpert E, Stahel W, Abbt M (2001). Log-normal Distributions across the Sciences: Keys and Clues. *BioScience*. 51(5)341-352

Mason D, et al. (2024). pydicom: An open source DICOM library.

Millar WT, Canney PA (1993). Derivation and application of equations describing the effects of fractionated protracted irradiation, based on multiple and incomplete repair processes. Part I. Derivation of equations. *Int. J. Radiat. Biol*. 64(3):275-91.

Millar W, Hopewell J, Paddick I, Lindquist C, Nordstron H, Lidberg P, Garding J (2015). The role of the concept of biologically effective dose (BED) in treatment planning and radiosurgery. *Physica* Medica 31:627-633.

Petzold C (2008). The Annotated Turing. Wiley Publishing.

Wazer D, Freeman C, Halperin E,. Prosnitz L, Brady L (2007). Principles and Practice of Radiation Oncology. Lippincott Williams & Wilkins. Chapter 2, page 15.

Sinclair WK, Morton RA (1965). X-Ray and ultraviolet sensitivity of synchronized Chinese hamster cells at various stages of the cell cycle. *Biophysical Journal* 5: 1-25.

Steel G, McMillan T, Peacock J (1989). The 5Rs of radiobiology. *Int. J Radiat Biol*. 56:1045-1048.

Stewart R, Li X (2007). BGRT: Biologically guided radiation therapy—The future is fast approaching!. *Med. Phys*. 34(10);3739-3751.

Tome W, Fowler J (2002). On cold spots in tumor subvolumes. *Med. Phys*. 29(7):1590-1598.

Webb S, Nahum A (1993). A model for calculating tumour control probability in radiotherapy including the effects of inhomogeneous distributions of dose and clonogenic cell density. *Phys. Med. Biol*. 38:653-666.

Wicklin R (2014). Simulate lognormal data with specified mean and variance. SAS: The Do Loop Blog. SAS Institute.

Wiklund K, Toma-Dasu I, Lind B (2014). Impact of Dose and Sensitivity Heterogeneity on TCP. *Computational and Mathematical Methods in Medicine*. 2014:1-7

Appendices

Appendix A – Functional Programming

Modern computer science dates back to the 1930s with the work of Alan Turing, Alonzo Church, and others in their investigations into the foundations of computability theory. These investigators were addressing David Hilbert's famous *Entscheidungsproblem*, or "decision problem." In 1936, both Church and Turing, in that order, published papers showing that a general solution to this decision problem was not possible. In doing so, it has been said that they ushered in both the modern computer and the mathematical study of the computable and the uncomputable (Petzold 2008).

As a part of these investigations, Turing invented what are now called Turing machines and Church invented the lambda calculus. Also, out of these investigations, the modern notion of an algorithm was defined, and through the Church-Turing thesis it was declared that anything that can be computed with an algorithm can be computed with a Turing machine. Furthermore, it was shown that other models of computation, including Church's lambda calculus, have equivalent power to Turing machines. Such systems that are equivalent to Turing machines in computational power are called Turing complete, and all systems that are Turing complete have exactly the same capabilities and limitations. These equivalent systems of computation may, however, have very different characteristics, making some systems more suited to particular types of problem solving than others. Functional programming languages are both Turing complete and they have very different characteristics than languages based on the Turing machine model of computation.

Computability via Turing machines gave rise to imperative programming, while computability via the lambda calculus gave rise to functional programming (Barendregt 2011). Turing machines are state-based models of computation, essentially being finite state machines that can read from and write to an infinitely long tape. They follow instructions imperatively, in a well-defined step-by-step process, and the internal state of the system changes during computation. The lambda calculus is a functional model of computation, where everything is seen as a function, and computation is seen as the evaluation of a function, and the function being evaluated is generally a composition of other functions, which may be compositions of other functions themselves, and so on. There is no internal state in this model of computation, only the evaluation of functions, and what happens inside a function is considered to be a black box, the internal workings unknown and not of interest to the programmer. This is a stateless model of computation, and this leads to many of the unique and useful properties of functional programming.

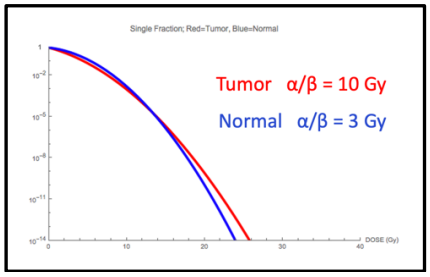
Unlike imperative programming where the programmer explicitly specifies the flow of control in a program, functional programming is declarative, meaning the programmer declares what they want to accomplish rather than providing step-by-step instructions on how to implement the task. For example, in this book a function called Map is used to map radiobiological functions over the voxels. Map is used in a declarative manner, where we declare that we want to map functions over the voxels, but we leave the details of the implementation to the functional language itself.

The first high-level imperative programming language was Fortran which was released in 1957. The first functional language was Lisp released in 1958. The name Lisp derives from the term list processor and speaks to the importance of lists as the primary data structure used in functional languages. Imperative languages matched more naturally, and were easier to implement on the von Neumann architectures and limited memory capacities of early computers, and imperative programming won out as the dominate type of computer programming. Functional languages make more demands on computer resources, and for years were largely relegated to the academic community, but have recently shown a significant increase in popularity. Advances in computer technology, particularly in multicore computing, are responsible for the increased interest in functional programming.

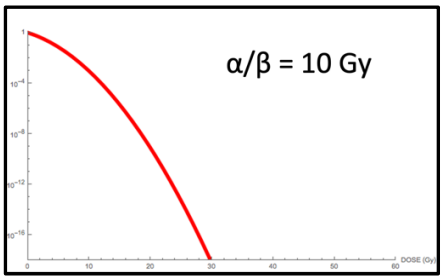
Functional languages include Lisp and Common Lisp, Haskell, Scheme, Clojure, Erlang, F#, and others. The Wolfram Language (the programming language of *Mathematica*) is a multi-paradigm programming language, but is built on a functional programming foundation, and is the functional language used in this book. Many traditional imperative languages have recently added functional extensions, such as lambda expressions, and these extensions are now found in Java, Python, Perl, and many other languages. It is becoming more common to use a hybrid approach in programming where both imperative and functional programming styles are used in the same program, and in this book Python is used imperatively for input/output tasks, while *Mathematica* is used in a functional manner for performing the actual voxel-based radiobiological modeling.

Appendix B – Linear Quadratic Modeling

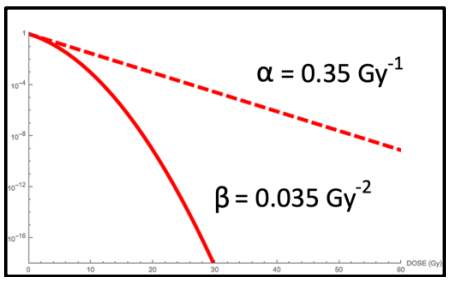
We demonstrate some basics of linear quadratic modeling.



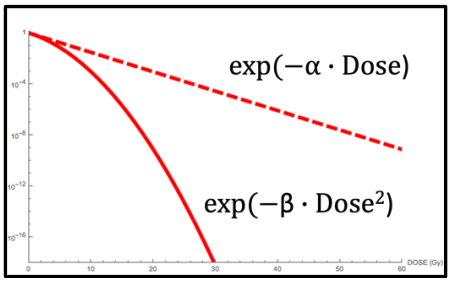
Here we show a single dose fraction to tumor with an α/β of 10 Gy and to (late responding) normal tissue with an α/β of 3 Gy.



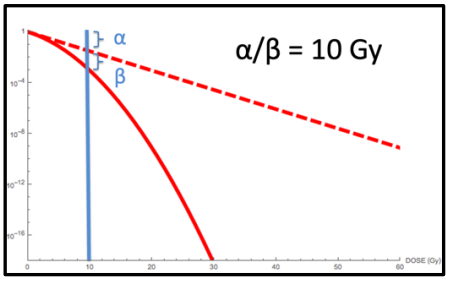
This shows the tumor curve with an α/β of 10 Gy.



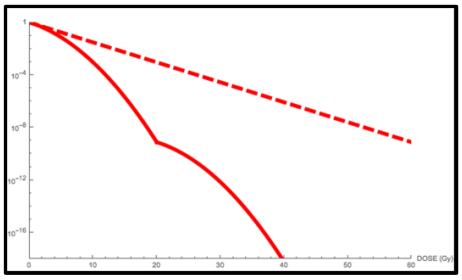
The α parameter gives the natural logs of irrepairable cell kill per unit dose of radiation. The β parameter gives the natural logs of repairable cell kill per unit dose squared of radiation. Two units of dose are involved because this component of cell killing is made up of the interaction of two different particles (or tracks or clusters) of radiation.



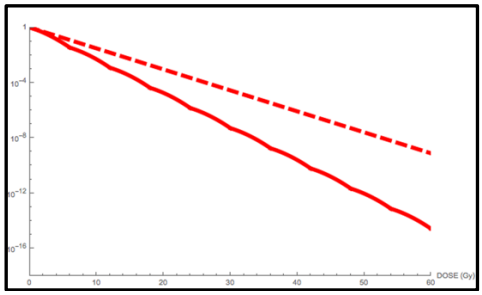
These two components form the linear quadradic equation $SF = \exp(-\alpha D - \beta D^2)$ which can also be written as $SF = \exp(-\alpha D)\exp(-\beta D^2)$.



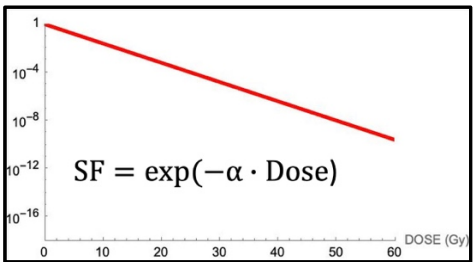
The α/β is defined as the dose where the $\exp(-\alpha D)$ component of cell killing equals that due to the $\exp(-\beta D^2)$ component. Here the α/β can be seen to be 10 Gy.



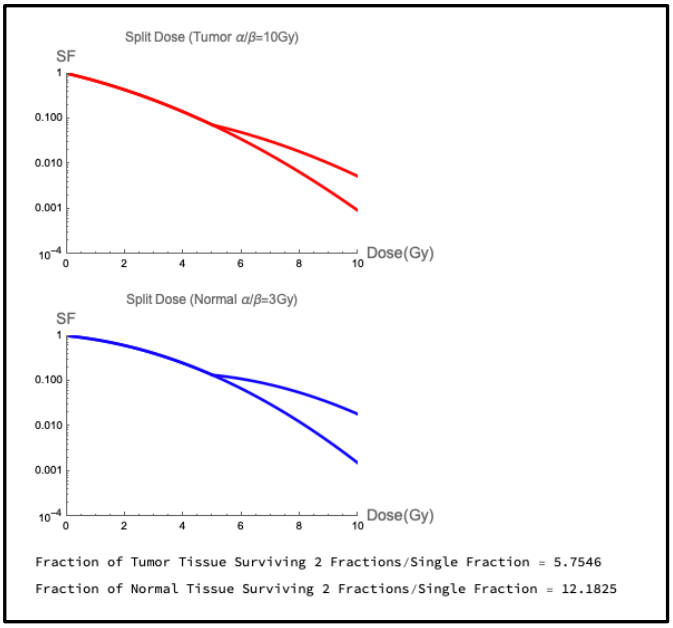
The $\exp(-\beta D^2)$ component of cell killing is considered repairable and with a lowering of dose-rate either through fractionation or going to a lower dose-rate this part of the cell killing can be repaired and removed.



And the lower the dose-rate, the more of this β component is repaired and doesn't contribute to the total cell killing.



At very low dose-rates or with extreme fractionation, this β component is completely removed and we are left with the above pure exponential curve just involving the α component $\exp(-\alpha D)$.

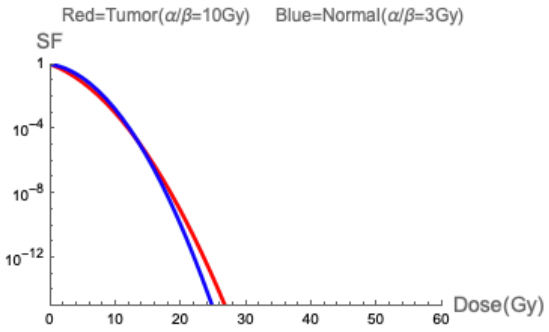


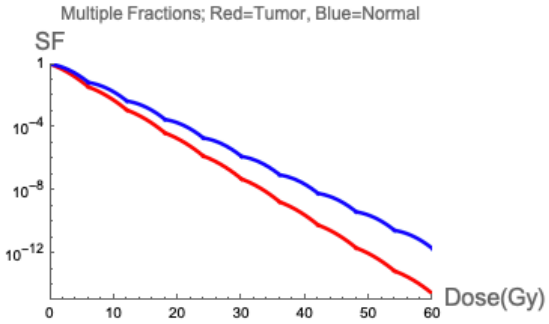
A split dose experiment can demonstrate the difference in fractionation effects between tumor (α/β = 10 Gy) and normal tissue (α/β = 3 Gy). The lower the α/β, the more sensitive the tissue is to fractionation or other dose-rate effects.

Now we demonstrate some of these concepts with *Mathematica*. We start with a tumor curve with α/β = 10 Gy and normal tissue curve with α/β = 3 Gy, with 6 Gy fractions to a total dose of 60 Gy.

```
FractionSize = 6;
TotalDose = 60;
NumberofFractions = TotalDose / FractionSize;
αtumor = 0.35;
βtumor = 0.035;
αnormal = 0.15;
βnormal = 0.05;
STumor[Dose_] := Exp[-αtumor Dose - βtumor Dose^2]
SNormal[Dose_] := Exp[-αnormal Dose - βnormal Dose^2]
VerySmallNumber = 10^-15;

SingleDoseTumor = LogPlot[STumor[Dose], {Dose, 0, TotalDose},
   PlotRange → {{0, TotalDose}, {VerySmallNumber, 1}}, AxesLabel → {Style["Dose(Gy)", 14], Style["SF", 14]},
   PlotLabel → "        Red=Tumor(α/β=10Gy)      Blue=Normal(α/β=3Gy)",
   PlotStyle → {RGBColor[1, 0, 0], Thickness[.01]}];

SingleDoseNormal = LogPlot[SNormal[Dose], {Dose, 0, TotalDose},
   PlotRange → {{0, TotalDose}, {VerySmallNumber, 1}}, AxesLabel → {"DOSE (Gy)", ""},
   PlotLabel → "Single Fraction Normal", PlotStyle → {RGBColor[0, 0, 1], Thickness[.01]}];

TumorFractionatedDose =
  Table[LogPlot[ (UnitStep[(Dose - n FractionSize)]) (STumor[FractionSize])^n STumor[Dose - n FractionSize],
    {Dose, n FractionSize, (n + 1) FractionSize}, PlotRange → {{0, TotalDose}, {VerySmallNumber, 1}},
    AxesLabel → {Style["Dose(Gy)", 16], Style["SF", 16]},
    PlotLabel → "Multiple Fractions; Red=Tumor, Blue=Normal", PlotStyle → {RGBColor[1, 0, 0], Thickness[.01]}],
   {n, 0, NumberofFractions}];
```

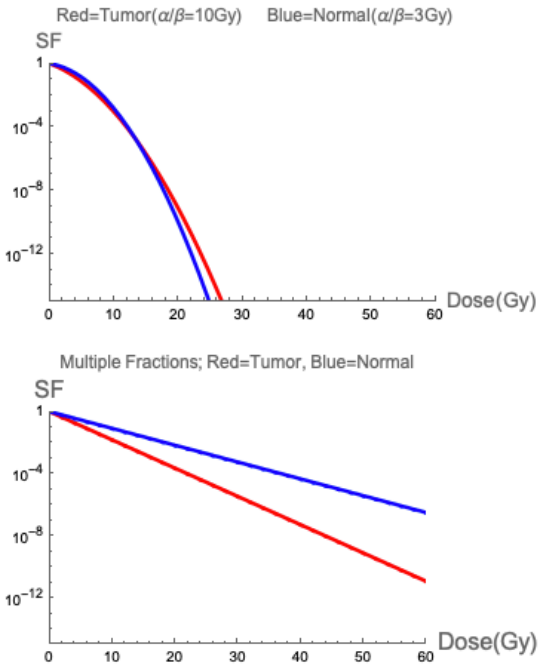
```
NormalTissueFractionatedDose =
  Table[LogPlot[ (UnitStep[(Dose - n FractionSize)]) (SNormal[FractionSize]) ^ n  SNormal[Dose - n FractionSize],
    {Dose, n FractionSize, (n + 1) FractionSize}, PlotRange → {{0, TotalDose}, {VerySmallNumber, 1}},
    PlotStyle → {RGBColor[0, 0, 1], Thickness[.01]}], {n, 0, NumberofFractions}];

Show[SingleDoseTumor, SingleDoseNormal]
Show[TumorFractionatedDose, NormalTissueFractionatedDose]
Print["Fraction of Tumor Surviving = ", (STumor[FractionSize]) ^ NumberofFractions];
Print["Fraction of Normal Tissue Surviving = ", (SNormal[FractionSize]) ^ NumberofFractions];
Print["Fraction of Normal Tissue Surviving/Fraction of Tumor Surviving = ",
 (SNormal[FractionSize]) ^ NumberofFractions / (STumor[FractionSize]) ^ NumberofFractions]
```






```
Fraction of Tumor Surviving = 2.55685×10^-15
Fraction of Normal Tissue Surviving = 1.87953×10^-12
Fraction of Normal Tissue Surviving/Fraction of Tumor Surviving = 735.095
(* here we see that with this level of fractionation 735 times more normal tissue survives than tumor *)


FractionSize = 2;
TotalDose = 60;
NumberofFractions = TotalDose / FractionSize;
αtumor = 0.35;
βtumor = 0.035;
αnormal = 0.15;
βnormal = 0.05;
STumor[Dose_] := Exp[-αtumor Dose - βtumor Dose^2]
SNormal[Dose_] := Exp[-αnormal Dose - βnormal Dose^2]
VerySmallNumber = 10^-15;

SingleDoseTumor = LogPlot[STumor[Dose], {Dose, 0, TotalDose},
   PlotRange → {{0, TotalDose}, {VerySmallNumber, 1}}, AxesLabel → {Style["Dose(Gy)", 14], Style["SF", 14]},
   PlotLabel → "      Red=Tumor(α/β=10Gy)      Blue=Normal(α/β=3Gy)",
   PlotStyle → {RGBColor[1, 0, 0], Thickness[.01]}];
```

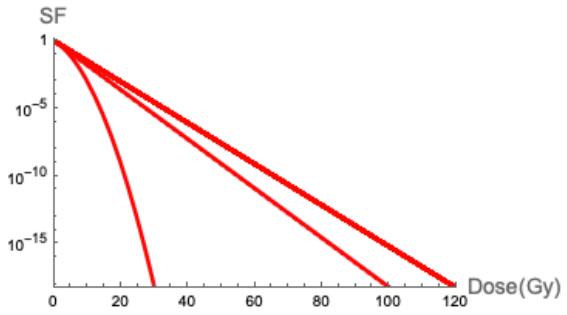
```
SingleDoseNormal = LogPlot[SNormal[Dose], {Dose, 0, TotalDose},
    PlotRange → {{0, TotalDose}, {VerySmallNumber, 1}}, AxesLabel → {"DOSE (Gy)", ""},
    PlotLabel → "Single Fraction Normal", PlotStyle → {RGBColor[0, 0, 1], Thickness[.01]}];

TumorFractionatedDose =
  Table[LogPlot[ (UnitStep[(Dose - n FractionSize)]) (STumor[FractionSize]) ^ n STumor[Dose - n FractionSize],
    {Dose, n FractionSize, (n + 1) FractionSize}, PlotRange → {{0, TotalDose}, {VerySmallNumber, 1}},
    AxesLabel → {Style["Dose(Gy)", 16], Style["SF", 16]},
    PlotLabel → "Multiple Fractions; Red=Tumor, Blue=Normal", PlotStyle → {RGBColor[1, 0, 0], Thickness[.01]}],
   {n, 0, NumberofFractions}];

NormalTissueFractionatedDose =
  Table[LogPlot[ (UnitStep[(Dose - n FractionSize)]) (SNormal[FractionSize]) ^ n SNormal[Dose - n FractionSize],
    {Dose, n FractionSize, (n + 1) FractionSize}, PlotRange → {{0, TotalDose}, {VerySmallNumber, 1}},
    PlotStyle → {RGBColor[0, 0, 1], Thickness[.01]}], {n, 0, NumberofFractions}];

Show[SingleDoseTumor, SingleDoseNormal]
Show[TumorFractionatedDose, NormalTissueFractionatedDose]
Print["Fraction of Tumor Surviving = ", (STumor[FractionSize]) ^ NumberofFractions];
Print["Fraction of Normal Tissue Surviving = ", (SNormal[FractionSize]) ^ NumberofFractions];
Print["Fraction of Normal Tissue Surviving/Fraction of Tumor Surviving = ",
 (SNormal[FractionSize]) ^ NumberofFractions / (STumor[FractionSize]) ^ NumberofFractions]
```

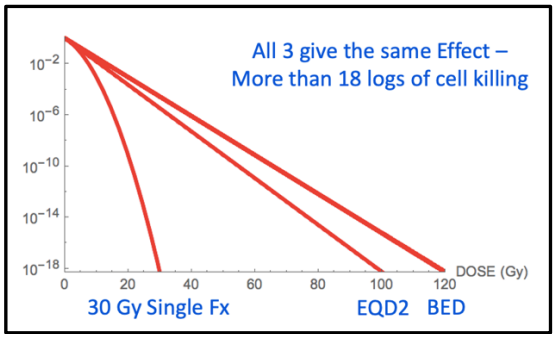



```
Fraction of Tumor Surviving = 1.13705×10^-11
Fraction of Normal Tissue Surviving = 3.05902×10^-7
Fraction of Normal Tissue Surviving/Fraction of Tumor Surviving = 26 903.2
```

```
(* here we see that with this higher level of fractionation about 27000 times more normal
 tissue survives than tumor *)


(* Now we demonstrate an example with the EQD2 and BED. This is a rather contrived example to
 demonstrate a concept. We will compare a 30 Gy single fraction with the EQD2 and BED equivalent
 doses. This is outside the range of applicability of the linear quadratic model but demonstrates
 an important concept. From the formula for EQD2 and BED for alpha/beta equal to 10 Gy we can
 calculate that the EQD2 is 100 Gy while the BED is 120 Gy *)
```

```
FractionSize = 2;
FractionSize2 = .01;
TotalDose = 120;
NumberofFractions = TotalDose / FractionSize;
NumberofFractions2 = TotalDose / FractionSize2;
αtumor = 0.35;
βtumor = 0.035;
αnormal = .15;
βnormal = .05;
STumor[Dose_] := Exp[-αtumor Dose - βtumor Dose^2]
SNormal[Dose_] := Exp[-αnormal Dose - βnormal Dose^2]

SingleDoseTumor = LogPlot[STumor[Dose], {Dose, 0, TotalDose},
   PlotRange → {{0, TotalDose}, {0.000000000000000005, 1.2}},
   AxesLabel → {Style["Dose(Gy)", 14], Style["SF", 14]}, PlotLabel → "",
   PlotStyle → {RGBColor[1, 0, 0], Thickness[.01]}];

TumorFractionatedDose =
  Table[LogPlot[ (UnitStep[(Dose - n FractionSize)]) (STumor[FractionSize]) ^ n STumor[Dose - n FractionSize],
    {Dose, n FractionSize, (n + 1) FractionSize}, PlotRange → {{0, TotalDose}, {0.000000000000000005, 1}},
    AxesLabel → {"DOSE (Gy)", ""}, PlotLabel → "", PlotStyle → {RGBColor[1, 0, 0], Thickness[.01]}],
   {n, 0, NumberofFractions}];

TumorFractionatedDose2 =
  Table[LogPlot[ (UnitStep[(Dose - n FractionSize2)]) (STumor[FractionSize2]) ^ n STumor[Dose - n FractionSize2],
    {Dose, n FractionSize2, (n + 1) FractionSize2}, PlotRange → {{0, TotalDose}, {0.000000000000000005, 1}},
    AxesLabel → {"DOSE (Gy)", ""}, PlotLabel → "", PlotStyle → {RGBColor[1, 0, 0], Thickness[.01]}],
   {n, 0, NumberofFractions2}];

Show[SingleDoseTumor, TumorFractionatedDose, TumorFractionatedDose2]
```





This shows that these three dose delivery methods give the same amount of tumor killing and this is why they are considered equivalent. The BED is given at a dose-rate where there is complete repair of the β component and that is why BED doses are always the highest.

Appendix C – DICOM RTDose File

This is the DICOM RTDose file called “composite_dose” that contains an array of the doses in each voxel. This dose array is contained in the Pixel Data attribute that we will access with Pydicom. Also important in the file below is the Image Position (Patient) which gives us the position of the first voxel in the dose array (more precisely it gives the position of the first pixel in the first slice of the dataset) and also the Dose Grid Scaling factor that will convert the dose array values to Gy. Also notice the “number of frames” or slices is 11 and that there are 10 rows and 9 columns forming an 11x10x9 dose array. Also note that the pixel spacing is 1 mm.

```
>>> composite_dose
(0008, 0000) Group Length                        UL: 380
(0008, 0005) Specific Character Set              CS: 'ISO_IR 100'
(0008, 0008) Image Type                          CS: ['DERIVED', 'SECONDARY', 'DOSE']
(0008, 0012) Instance Creation Date              DA: '20150914'
(0008, 0013) Instance Creation Time              TM: '102551'
(0008, 0016) SOP Class UID                       UI: RT Dose Storage
(0008, 0018) SOP Instance UID                    UI: 2.16.840.1.114362.1.5.2.0.11505.6304961967.400026112.612.206
(0008, 0020) Study Date                          DA: '20150817'
(0008, 0030) Study Time                          TM: '084321.468000'
(0008, 0050) Accession Number                    SH: 'SJ102332340'
(0008, 0060) Modality                            CS: 'RTDOSE'
(0008, 0070) Manufacturer                        LO: 'Elekta Instrument AB'
(0008, 0080) Institution Name                    LO: ''
(0008, 0090) Referring Physician's Name          PN: ' '
(0008, 1030) Study Description                   LO: 'HEAD^ROUTINE'
(0008, 103e) Series Description                  LO: 'composite'
(0008, 1090) Manufacturer's Model Name           LO: 'Leksell GammaPlan\xae'
(0010, 0000) Group Length                        UL: 54
(0010, 0010) Patient's Name                      PN: 'gamma BED'
(0010, 0020) Patient ID                          LO: 'ANON33848'
(0010, 0030) Patient's Birth Date                DA: ''
(0010, 0040) Patient's Sex                       CS: 'M'
(0012, 0000) Group Length                        UL: 56
(0012, 0062) Patient Identity Removed            CS: 'YES'
(0012, 0063) De-identification Method            LO: 'Limited Data Set: MIM.5.2.0.B505-05'
(0018, 0000) Group Length                        UL: 22
(0018, 0050) Slice Thickness                     DS: ''
(0018, 1020) Software Version(s)                 LO: '10.1.1'
(0020, 0000) Group Length                        UL: 360
(0020, 000d) Study Instance UID                  UI: 2.16.840.1.114362.1.5.2.0.11505.6304961967.400026074.1095.1
(0020, 000e) Series Instance UID                 UI: 2.16.840.1.114362.1.5.2.0.11505.6304961967.400026112.612.205
(0020, 0010) Study ID                            SH: 'ANON33848'
(0020, 0011) Series Number                       IS: ''
(0020, 0013) Instance Number                     IS: ''
(0020, 0032) Image Position (Patient)            DS: ['2.51927852', '19.5898048', '31.1290617']
(0020, 0037) Image Orientation (Patient)         DS: ['1', '-2.05102976e-10', '-8.26559097e-38', '2.05102976e-10', '1',
(0020, 0052) Frame of Reference UID              UI: 2.16.840.1.114362.1.5.2.0.11505.6304961967.400026074.1095.3
(0020, 1040) Position Reference Indicator        LO: ''
(0028, 0000) Group Length                        UL: 124
(0028, 0002) Samples per Pixel                   US: 1
(0028, 0004) Photometric Interpretation          CS: 'MONOCHROME2'
(0028, 0008) Number of Frames                    IS: '11'
(0028, 0009) Frame Increment Pointer             AT: (3004, 000c)
(0028, 0010) Rows                                US: 10
(0028, 0011) Columns                             US: 9
(0028, 0030) Pixel Spacing                       DS: ['1', '1']
(0028, 0100) Bits Allocated                      US: 16
(0028, 0101) Bits Stored                         US: 16
(0028, 0102) High Bit                            US: 15
(0028, 0103) Pixel Representation                US: 0
(3004, 0000) Group Length                        UL: 132
(3004, 0002) Dose Units                          CS: 'GY'
(3004, 0004) Dose Type                           CS: 'PHYSICAL'
(3004, 0006) Dose Comment                        LO: 'Dose algorithm: TMR 10'
(3004, 000a) Dose Summation Type                 CS: 'PLAN'
(3004, 000c) Grid Frame Offset Vector            DS: ['-0', '-1', '-2', '-3', '-4', '-5', '-6', '-7', '-8', '-9', '-10']
(3004, 000e) Dose Grid Scaling                   DS: '0.00055436292'
(300c, 0000) Group Length                        UL: 134
(300c, 0002)  Referenced RT Plan Sequence   1 item(s) ----
   (0008, 0000) Group Length                        UL: 106
   (0008, 1150) Referenced SOP Class UID            UI: RT Plan Storage
   (0008, 1155) Referenced SOP Instance UID         UI: 2.16.840.1.114362.1.5.2.0.11505.6304961967.400026111.770.204
   ---------
(7fe0, 0000) Group Length                        UL: 1988
(7fe0, 0010) Pixel Data                          OW or OB: Array of 1980 bytes
```

Appendix C – DICOM RTDose File (cont)

We access the dose array with the following command, "composite_dose.pixel_array." Notice the maximum array value is 65535 (2^{16}-1) and when multiplied by the Dose Grid Scaling factor above gives a dose of 36.3 Gy, which we saw was the maximum dose in the dataset.

```
>>> composite_dose.pixel_array
array([[[[11104, 13036, 15039, 16155, 16096, 14880, 12924, 11105,  9504],
         [12725, 15746, 19507, 22608, 22936, 20285, 16512, 13219, 10634],
         [14712, 19459, 26361, 31489, 32260, 28418, 21574, 16060, 12302],
         [16133, 22873, 31626, 36469, 37445, 34438, 26289, 18519, 13847],
         [16375, 23829, 32886, 37539, 38601, 36053, 28048, 19875, 14890],
         [15779, 21995, 30261, 35704, 36789, 33512, 26239, 19776, 15388],
         [14719, 19223, 24773, 29227, 30200, 27325, 22648, 18439, 15125],
         [13681, 16868, 20164, 22347, 22946, 21901, 19616, 17001, 14610],
         [12736, 14935, 17049, 18155, 18531, 18391, 17349, 15796, 14073],
         [11657, 13214, 14552, 15430, 15805, 15836, 15404, 14537, 13381]],

        [[13280, 15957, 18778, 20573, 20777, 19402, 16784, 14123, 11765],
         [15588, 19939, 25323, 29602, 30458, 27449, 22436, 17540, 13589],
         [18449, 25391, 34249, 39960, 41083, 37657, 29742, 21849, 16014],
         [20639, 30145, 40226, 44915, 45975, 43635, 35466, 25144, 17968],
         [21310, 31796, 42072, 46554, 47402, 44988, 36886, 26206, 18977],
         [20888, 30386, 40812, 45906, 46229, 42625, 33961, 25179, 19031],
         [20005, 27828, 35978, 40795, 40248, 35591, 28820, 22893, 18300],
         [18990, 25187, 30424, 32576, 31322, 28321, 24480, 20722, 17426],
         [17400, 21930, 25283, 25992, 24873, 23364, 21306, 18973, 16560],
         [15191, 18020, 20040, 20625, 20240, 19612, 18635, 17214, 15494]],

        [[15428, 18711, 22101, 24419, 25116, 24134, 21288, 17759, 14544],
         [18110, 23187, 29128, 33665, 34900, 32779, 28337, 22769, 17398],
         [21329, 29111, 38836, 45102, 46290, 42990, 35900, 28103, 20653],
         [23889, 34544, 46529, 52202, 52804, 49735, 41232, 31275, 22767],
         [25093, 37151, 49948, 55249, 55048, 51208, 41993, 31573, 23387],
         [25612, 37152, 49448, 55376, 54319, 48363, 38712, 29787, 22834],
         [25805, 35287, 44022, 49480, 48188, 41632, 33918, 27085, 21532],
         [25004, 32245, 37452, 40592, 39652, 34998, 29722, 24585, 20146],
         [22718, 28575, 32089, 33485, 32231, 29280, 25939, 22244, 18854],
         [19057, 23364, 25956, 26341, 25239, 23780, 22097, 19824, 17446]],

        [[17814, 21834, 25573, 27768, 28740, 28614, 26360, 22281, 17683],
         [20830, 26509, 31716, 34955, 36365, 35855, 33354, 28753, 22143],
         [24240, 32057, 39552, 45113, 47082, 44527, 39705, 34085, 26315],
         [26842, 37063, 47263, 54536, 56113, 51519, 43605, 36307, 28243],
         [28400, 40403, 51761, 58516, 58875, 52787, 43610, 35845, 28318],
         [29498, 41466, 52002, 57501, 56150, 49230, 41056, 34063, 27202],
         [29845, 39154, 46971, 51981, 50881, 44433, 37901, 31567, 25298],
         [28792, 35399, 40859, 45013, 44877, 39796, 34446, 28980, 23453],
         [26277, 31892, 35561, 38110, 37983, 34576, 30673, 26135, 21618],
         [21970, 26884, 29801, 30754, 30026, 28258, 25865, 22717, 19504]],

        [[20021, 24536, 27855, 29455, 30659, 31410, 30269, 26570, 20628],
         [23386, 29222, 33211, 36043, 38346, 38843, 37385, 33729, 26523],
         [27130, 34480, 39903, 44637, 46991, 46202, 43429, 38914, 31428],
         [30016, 39152, 46060, 51094, 52394, 50327, 46321, 40606, 33132],
         [31359, 42501, 50001, 54344, 54944, 51833, 46171, 39545, 32650],
         [31711, 43895, 51634, 55033, 54185, 49692, 43486, 37439, 31022],
         [31294, 41827, 49218, 52686, 50669, 44867, 39814, 35021, 28807],
         [29923, 37642, 43890, 47825, 46560, 41417, 37028, 32599, 26600],
         [27157, 33633, 38096, 41211, 40669, 37317, 33989, 29644, 24208],
         [22701, 27975, 31518, 33089, 32791, 31293, 28979, 25349, 21276]],

        [[21334, 25729, 28431, 29798, 31323, 32680, 32257, 29002, 22658],
         [24840, 30361, 33821, 37034, 39802, 40667, 39727, 36365, 29103],
         [28752, 35657, 40694, 44873, 46469, 46478, 45273, 41693, 34261],
         [31890, 40461, 46079, 48910, 49474, 49014, 47494, 43346, 35963],
         [33003, 43508, 49430, 51617, 51747, 50620, 47773, 42072, 35033],
         [32162, 44174, 51135, 53505, 53065, 50392, 45373, 39270, 32903],
         [30609, 42502, 50642, 52973, 50269, 45389, 40932, 36539, 30485],
         [29028, 38830, 46807, 49877, 46777, 42000, 38400, 34198, 28109],
         [26099, 33776, 39813, 42642, 41282, 38551, 35771, 31234, 25443],
         [22047, 27099, 31116, 33074, 33285, 32491, 30380, 26492, 22089]]],

       [[[21233, 25409, 27872, 29162, 30675, 32076, 31913, 28844, 22680],
         [24685, 29939, 33198, 36284, 39041, 40067, 39196, 36023, 28999],
         [28504, 35143, 40095, 44393, 46275, 46360, 45044, 41340, 34072],
         [31674, 39988, 45825, 49362, 50276, 49685, 47830, 43372, 35890],
         [32846, 43205, 49661, 52741, 53233, 51850, 48422, 42266, 34966],
         [31820, 43854, 51603, 54821, 54690, 51661, 45937, 39387, 32785],
         [30097, 42568, 51772, 54621, 52455, 47351, 41889, 36433, 30251],
         [28542, 39258, 48163, 51586, 49109, 43945, 39137, 33786, 27689],
         [25813, 33948, 40697, 43683, 42613, 39495, 35643, 30480, 24936],
         [22024, 27216, 31694, 33763, 33566, 32282, 29823, 25975, 21744]],

        [[19928, 23757, 26316, 27689, 28878, 29946, 29523, 26356, 20923],
         [23054, 28136, 31410, 33838, 36086, 37116, 36218, 33034, 26463],
         [26505, 33081, 37978, 42549, 45358, 45266, 42688, 38185, 31096],
         [29576, 38084, 44783, 50665, 53572, 52120, 47348, 40956, 33169],
         [31167, 42033, 50577, 57579, 60473, 57269, 49272, 40838, 32812],
         [31143, 44158, 54542, 61351, 62589, 56983, 47535, 38863, 31088],
         [30777, 43387, 53281, 59200, 59286, 52363, 43459, 35826, 28774],
         [29802, 39725, 47095, 52049, 52456, 46549, 38924, 32165, 26040],
         [27004, 35015, 40446, 43869, 43755, 39323, 33860, 28319, 23186],
         [22574, 28724, 33372, 35310, 34167, 31244, 28052, 24168, 20251]],

        [[17795, 21138, 23913, 25578, 26542, 26960, 25815, 22505, 18149],
         [20441, 25231, 29097, 31411, 32919, 33591, 32375, 28543, 22277],
         [23440, 30109, 35655, 39878, 42569, 42227, 39081, 33917, 26282],
         [26318, 35325, 43660, 51116, 54908, 52761, 45877, 37690, 28816],
         [28320, 39899, 51789, 60724, 62894, 59218, 50425, 39385, 29314],
         [29302, 42852, 56895, 65535, 65326, 59843, 50510, 38516, 28273],
         [29938, 42189, 53977, 62914, 63842, 57007, 46207, 34958, 26236],
         [29209, 38444, 46199, 52939, 54802, 48647, 38727, 30129, 23437],
         [26360, 33948, 39076, 42659, 42982, 38119, 31490, 25472, 20550],
         [21724, 27932, 32277, 34039, 32648, 29001, 25211, 21274, 17764]],

        [[15646, 18466, 21090, 22857, 23506, 23184, 21412, 18362, 14999],
         [17680, 21769, 25785, 28602, 29932, 29805, 27546, 23017, 17722],
         [20100, 25980, 32256, 37344, 39804, 38934, 34727, 28156, 20816],
         [22495, 30788, 40542, 48607, 51429, 49458, 42747, 32956, 23427],
         [24248, 34970, 48234, 56258, 57043, 54442, 48003, 35854, 24715],
         [25197, 37565, 52414, 59798, 59250, 55098, 48255, 35765, 24416],
         [25921, 37794, 50639, 58913, 58706, 53012, 44183, 32253, 22878],
         [25273, 34924, 43409, 50514, 51879, 45627, 36026, 26930, 20225],
         [22529, 30167, 35767, 39305, 38981, 33807, 27604, 21970, 17507],
         [18441, 23814, 27901, 29302, 27558, 24349, 21230, 17978, 15133]],

        [[13723, 16041, 18269, 19732, 19916, 18966, 17093, 14616, 12129],
         [15136, 18243, 21596, 24126, 24953, 24107, 21369, 17450, 13705],
         [16790, 21010, 26159, 30907, 33270, 31802, 27098, 21064, 15787],
         [18357, 24258, 32256, 40126, 43415, 41337, 34231, 25153, 17999],
         [19290, 26991, 37826, 46057, 48079, 46075, 39379, 28248, 19451],
         [19567, 28306, 40281, 47805, 48801, 46344, 40201, 28823, 19707],
         [19711, 28382, 39356, 47016, 47437, 43961, 36660, 26288, 18774],
         [18886, 26459, 34390, 40309, 40885, 36444, 29204, 21991, 16807],
         [16717, 22319, 27414, 30038, 29236, 26061, 22105, 18079, 14715],
         [14022, 17243, 20048, 21094, 20212, 18753, 17033, 14934, 12871]]]], dtype=uint16)
```

Appendix D – DICOM RTSTRUCT File

This is the DICOM RTSTRUCT file that contains the contour information in the form of polygon vertices. Three contours are shown at the bottom of the file. The first one has 12 contour (vertex) points and is at z-coordinate 31.1290617 mm. The second one has 21 contour points and is at z-coordinate 30.1290617 mm.

```
>>> rtstructfile
(0008, 0000) Group Length                       UL: 342
(0008, 0005) Specific Character Set             CS: 'ISO_IR 100'
(0008, 0012) Instance Creation Date             DA: '20150914'
(0008, 0013) Instance Creation Time             TM: '100814'
(0008, 0016) SOP Class UID                      UI: RT Structure Set Storage
(0008, 0018) SOP Instance UID                   UI: 2.16.840.1.114362.1.5.2.0.11505.6304961967.400026120.521.213
(0008, 0020) Study Date                         DA: '20150817'
(0008, 0030) Study Time                         TM: '084321.468000'
(0008, 0050) Accession Number                   SH: 'SJ102332340'
(0008, 0060) Modality                           CS: 'RTSTRUCT'
(0008, 0070) Manufacturer                       LO: 'Elekta Instrument AB'
(0008, 0080) Institution Name                   LO: ''
(0008, 0090) Referring Physician's Name         PN: ' '
(0008, 1030) Study Description                  LO: 'HEAD^ROUTINE'
(0008, 103e) Series Description                 LO: ''
(0008, 1090) Manufacturer's Model Name          LO: 'Leksell GammaPlan\xae'
(0010, 0000) Group Length                       UL: 54
(0010, 0010) Patient's Name                     PN: 'gamma BED'
(0010, 0020) Patient ID                         LO: 'ANON33848'
(0010, 0030) Patient's Birth Date               DA: ''
(0010, 0040) Patient's Sex                      CS: 'M'
(0012, 0000) Group Length                       UL: 56
(0012, 0062) Patient Identity Removed           CS: 'YES'
(0012, 0063) De-identification Method           LO: 'Limited Data Set: MIM.5.2.0.B505-05'
(0018, 0000) Group Length                       UL: 14
(0018, 1020) Software Version(s)                LO: '10.1.1'
(0020, 0000) Group Length                       UL: 238
(0020, 000d) Study Instance UID                 UI: 2.16.840.1.114362.1.5.2.0.11505.6304961967.400026074.1095.1
(0020, 000e) Series Instance UID                UI: 2.16.840.1.114362.1.5.2.0.11505.6304961967.400026120.521.212
(0020, 0010) Study ID                           SH: 'ANON33848'
(0020, 0011) Series Number                      IS: ''
(0020, 0052) Frame of Reference UID             UI: 2.16.840.1.114362.1.5.2.0.11505.6304961967.400026074.1095.3
(0020, 1040) Position Reference Indicator       LO: ''
(3006, 0000) Group Length                       UL: 34044
(3006, 0002) Structure Set Label                SH: 'ANON'
(3006, 0008) Structure Set Date                 DA: ''
(3006, 0009) Structure Set Time                 TM: ''
(3006, 0010)  Referenced Frame of Reference Sequence   1 item(s) ----
   (0020, 0000) Group Length                      UL: 68
   (0020, 0052) Frame of Reference UID            UI: 2.16.840.1.114362.1.5.2.0.11505.6304961967.400026074.1095.3
   (3006, 0000) Group Length                      UL: 21724
   (3006, 0012)  RT Referenced Study Sequence   1 item(s) ----
(3006, 0039)  ROI Contour Sequence   1 item(s) ----
   (3006, 0000) Group Length                      UL: 11902
   (3006, 002a) ROI Display Color                 IS: ['255', '0', '255']
   (3006, 0040)  Contour Sequence   12 item(s) ----
      (3006, 0000) Group Length                   UL: 554
      (3006, 0016)  Contour Image Sequence   1 item(s) ----
         (0008, 0000) Group Length                UL: 102
         (0008, 1150) Referenced SOP Class UID    UI: MR Image Storage
         (0008, 1155) Referenced SOP Instance UID UI: 2.16.840.1.114362.1.5.2.0.11505.6304961967.400026074.454.100
         ---------
      (3006, 0042) Contour Geometric Type         CS: 'CLOSED_PLANAR'
      (3006, 0046) Number of Contour Points       IS: '12'
      (3006, 0050) Contour Data                   DS: ['6.78227852', '22.593761', '31.1290617', '6.78227852', '24.420761', '31.1290617',
'6.57927852', '24.522261', '31.1290617', '5.86877852', '24.928261', '31.1290617', '5.66577852', '25.029761', '31.1290617', '5.15827852', '24.
826761', '31.1290617', '5.05677852', '24.623761', '31.1290617', '4.95527852', '24.420761', '31.1290617', '4.95527852', '23.710261', '31.12906
17', '5.05677852', '23.507261', '31.1290617', '5.25977852', '23.304261', '31.1290617', '6.37627852', '22.492261', '31.1290617']
      ---------
      (3006, 0000) Group Length                   UL: 842
      (3006, 0016)  Contour Image Sequence   1 item(s) ----
         (0008, 0000) Group Length                UL: 102
         (0008, 1150) Referenced SOP Class UID    UI: MR Image Storage
         (0008, 1155) Referenced SOP Instance UID UI: 2.16.840.1.114362.1.5.2.0.11505.6304961967.400026074.470.105
         ---------
      (3006, 0042) Contour Geometric Type         CS: 'CLOSED_PLANAR'
      (3006, 0046) Number of Contour Points       IS: '21'
      (3006, 0050) Contour Data                   DS: ['6.07177852', '25.740261', '30.1290617', '5.66577852', '25.841761', '30.1290617',
'4.54927852', '25.029761', '30.1290617', '4.44777852', '25.131261', '30.1290617', '4.24477852', '24.928261', '30.1290617', '4.14327852', '24.
725261', '30.1290617', '3.94027852', '23.913261', '30.1290617', '3.94027852', '23.608761', '30.1290617', '4.54927852', '22.086261', '30.12906
17', '4.65077852', '21.883261', '30.1290617', '5.56427852', '21.071261', '30.1290617', '5.86877852', '21.071261', '30.1290617', '6.88377852',
'20.766761', '30.1290617', '7.08677852', '20.868261', '30.1290617', '7.89877852', '21.172761', '30.1290617', '7.89877852', '21.071261', '30.1
290617', '8.10177852', '21.172761', '30.1290617', '9.01527852', '22.796761', '30.1290617', '8.60927852', '24.725261', '30.1290617', '7.594278
52', '25.537261', '30.1290617', '7.28977852', '25.537261', '30.1290617']
      ---------
      (3006, 0000) Group Length                   UL: 970
      (3006, 0016)  Contour Image Sequence   1 item(s) ----
         (0008, 0000) Group Length                UL: 102
         (0008, 1150) Referenced SOP Class UID    UI: MR Image Storage
         (0008, 1155) Referenced SOP Instance UID UI: 2.16.840.1.114362.1.5.2.0.11505.6304961967.400026074.454.99
         ---------
      (3006, 0042) Contour Geometric Type         CS: 'CLOSED_PLANAR'
      (3006, 0046) Number of Contour Points       IS: '25'
      (3006, 0050) Contour Data                   DS: ['7.69577852', '20.665261', '29.1290617', '7.89877852', '20.766761', '29.1290617',
'8.20327852', '20.766761', '29.1290617', '9.21827852', '21.781761', '29.1290617', '9.62427852', '23.710261', '29.1290617', '9.62427852', '24.
014761', '29.1290617', '8.91377852', '24.928261', '29.1290617', '8.81227852', '25.131261', '29.1290617', '7.59427852', '26.044761', '29.12906
17', '7.39127852', '26.146261', '29.1290617', '6.17327852', '26.349261', '29.1290617', '5.97027852', '26.450761', '29.1290617', '5.66577852',
'26.450761', '29.1290617', '4.65077852', '26.450761', '29.1290617', '4.34627852', '26.450761', '29.1290617', '3.53427852', '26.146261', '29.1
290617', '2.92527852', '25.334261', '29.1290617', '2.92527852', '25.029761', '29.1290617', '3.02677852', '24.826761', '29.1290617', '3.128278
52', '23.304261', '29.1290617', '3.12827852', '22.999761', '29.1290617', '3.83877852', '22.086261', '29.1290617', '4.65077852', '21.071261',
'29.1290617', '5.86877852', '20.563761', '29.1290617', '7.28977852', '20.462261', '29.1290617']
```

The way the contour vertex data is extracted from the RTSTRUCT file is shown below for two slices of the dataset. The first slice is Contours[0] (Python starts indexing with 0 while in *Mathematica* we will start indexing with 1.) Note how this data corresponds to the data in the first slice above with the 12 contour (vertex) points and z-coordinate 31.1290617 mm.

```
>>> rtstructfile.ROIContours[0].Contours[0].ContourData
['6.78227852', '22.593761', '31.1290617', '6.78227852', '24.420761', '31.1290617', '6.57927852', '24.522261', '31.1290617', '5.86877852', '24
.928261', '31.1290617', '5.66577852', '25.029761', '31.1290617', '5.15827852', '24.826761', '31.1290617', '5.05677852', '24.623761', '31.1290
617', '4.95527852', '24.420761', '31.1290617', '4.95527852', '23.710261', '31.1290617', '5.05677852', '23.507261', '31.1290617', '5.25977852'
, '23.304261', '31.1290617', '6.37627852', '22.492261', '31.1290617']
```

Contours[8] below corresponds to slice 9 in *Mathematica* and is at z-coordinate 23.1290617 mm. This is the slice with the highest dose (and BED) and we will use it in some examples.

```
>>> rtstructfile.ROIContours[0].Contours[8].ContourData
['3.17199604', '24.5818332', '23.1290617', '3.17199604', '23.958284', '23.1290617', '3.41182266', '23.1908388', '23.1290617', '3.73727852', '
22.593761', '23.1290617', '3.94027852', '22.390761', '23.1290617', '4.75227852', '21.680261', '23.1290617', '5.05677852', '21.680261', '23.12
90617', '5.25977852', '21.578761', '23.1290617', '6.17327852', '21.172761', '23.1290617', '6.47777852', '21.172761', '23.1290617', '7.8987785
2', '21.071261', '23.1290617', '9.11677852', '21.477261', '23.1290617', '9.72577852', '22.593761', '23.1290617', '9.82727852', '22.796761', '
23.1290617', '9.82727852', '23.101261', '23.1290617', '9.92877852', '24.623761', '23.1290617', '9.31977852', '26.349261', '23.1290617', '9.21
827852', '26.552261', '23.1290617', '8.10177852', '27.770261', '23.1290617', '7.89877852', '27.871761', '23.1290617', '7.08677852', '28.48076
1', '23.1290617', '6.78227852', '28.480761', '23.1290617', '5.86877852', '28.683761', '23.1290617', '5.56427852', '28.683761', '23.1290617',
'4.85377852', '28.683761', '23.1290617', '4.61095576', '28.5149898', '23.1290617', '3.93944122', '28.3231285', '23.1290617', '3.61590672', '2
8.015097', '23.1290617', '3.17199604', '27.3158567', '23.1290617', '3.17199604', '25.3492784', '23.1290617', '3.17199604', '25.2053824', '23.
1290617', '3.17199604', '25.0614865', '23.1290617']
```

Appendix E – Determining which Voxels Lie within the Contours – Imperatively

This Python/Pydicom program determines which voxels lie within the boundary of the contour of interest. Those voxels within the contour boundary are marked by setting their dose value to zero.

1. Read in the DICOM RTDose file and RTStruct file.
2. Get the x and y coordinates of the dose array origin.
3. Get number of rows and columns in dose array and determine pixel spacing.
4. Get number of slices in dose array and number of contour slices.
5. Determine which dose array slice corresponds to the first contour slice.
6. Form an array of contour polygon vertex values.
7. For each contour slice determine which voxels in the dose array lie within the contour boundary. This is done by using the Matplotlib path module and testing if the voxel coordinates lie within the contour polygon using the *contains_point* function.
8. Mark the voxels that lie within the contour boundary by setting their dose value to zero.
9. Save the modified DICOM file and save the modified array data to a list for import into *Mathematica*.

```
#this program determines which voxels lie within the boundary of the contour of interest
#those voxels within the contour boundary are marked by setting their dose value to zero
import os #imports the operating system module, needed for file and directory functions
import numpy as np #imports the NumPy scientific computing library and makes np shorthand for numpy
os.chdir("/Users/Documents/pydicom-0.9.7") #sets the directory path
import dicom #imports the pydicom package
from matplotlib import path #imports the path module needed for determining if a point lies within a polygon
rtdosefile=dicom.read_file("composite dose.dcm") #reads in the dicom rtdose file
rtstructfile=dicom.read_file("structure set.dcm") #reads in the dicom rtstruct file
x_origin=float(rtdosefile.ImagePositionPatient[0]) #gets the dose array origin x coordinate
y_origin=float(rtdosefile.ImagePositionPatient[1]) #gets the dose array origin y coordinate
pixel_spacing=float(rtdosefile.PixelSpacing[0]) #gets the pixel spacing dimension
num_columns=rtdosefile.pixel_array.shape[2] #gets the number of columns in the dose array
num_rows=rtdosefile.pixel_array.shape[1] #gets the number of rows in the dose array
num_doseslices=rtdosefile.pixel_array.shape[0] #gets the number of slices in the dose array
num_contourslices=len(rtstructfile.ROIContours[0].Contours) #gets the number of contour slices
scaling_factor=0 #sets the scaling_factor to zero, this is used to mark the voxels that lie within the contour
contour=[] #initializes a list of arrays containing the coordinates of the polygon vertices for each contour slice
#the for loop below reads in an array containing the coordinates of the polygon vertices, for each contour slice
for contour_slicenumber in range(0,num_contourslices):
        contour.append(np.array(rtstructfile.ROIContours[0].Contours[contour_slicenumber].ContourData))
#the for loop below iterates through the slices of the dose array to determine which voxels lie within the contour
#boundaries.  Care must be taken to assure that the dose_slicenumber corresponds to the correct contour_slicenumber.
#In this particular case, dose_slicenumber = contour_slicenumber.
for dose_slicenumber in range(0,num_doseslices):
        #the line below deletes the z-coordinates of the polygon vertices
        xy_coordinates = np.delete(contour[dose_slicenumber], np.arange(2, contour[dose_slicenumber].size, 3))
        #the line below reshapes xy_coordinates into an Nx2 array of vertices needed by the path function
        vertices=np.reshape(xy_coordinates, (-1, 2))
        #the for loop below iterates through each voxel (aka pixel for a single slice) of a given slice in the dose
        #array and tests whether the voxel is within the contour boundary. If it is within the contour boundary, the
        #dose in the voxel is set to zero, thereby marking the voxels that lie within the contour.
        for col in range(0,num_columns):
                for row in range(0,num_rows):
                        x=x_origin+col*pixel_spacing
                        y=y_origin+row*pixel_spacing
                        polygon = path.Path(vertices)
                        if polygon.contains_point([x,y]):
                                rtdosefile.pixel_array[dose_slicenumber,row,col]=    \
                                scaling_factor*rtdosefile.pixel_array[dose_slicenumber,row, col]
#the line below overwrites the PixelData attribute of rtdosefile
rtdosefile.PixelData=rtdosefile.pixel_array.tostring()
#the line below saves this file as "modified composite dose.dcm" which has 0 values for voxels within the contour
rtdosefile.save_as("modified composite dose.dcm")
rtdosefile_list=rtdosefile.pixel_array.tolist() #converts the rtdosefile array into a list
textfile=open("modified composite dose list.txt","w") #creates and opens a text file in write mode
print >>textfile, rtdosefile_list #writes the rtdosefile list to the text file
textfile.close() #closes the text file
```

Appendix E – Determining which Voxels Lie within the Contours – Imperatively (cont)

This was the original composite_dose.pixel_array data.

```
[[11104 13036 15039 16155 16096 14880 12924 11105  9504]
 [12725 15746 19507 22608 22936 20285 16512 13219 10634]
 [14712 19459 26361 31489 32260 28418 21574 16060 12302]
 [16133 22873 31626 36469 37445 34438 26289 18519 13847]
 [16375 23829 32886 37539 38601 36053 28048 19875 14890]
 [15779 21995 30261 35704 36789 33512 26239 19776 15388]
 [14719 19223 24773 29227 30200 27325 22648 18439 15125]
 [13681 16868 20164 22347 22946 21901 19616 17001 14610]
 [12736 14935 17049 18155 18531 18391 17349 15796 14073]
 [11657 13214 14552 15430 15805 15836 15404 14537 13381]]

[[13280 15957 18778 20573 20777 19402 16784 14123 11765]
 [15588 19939 25323 29602 30458 27449 22436 17540 13589]
 [18449 25391 34249 39960 41083 37657 29742 21849 16014]
 [20639 30145 40226 44915 45975 43635 35466 25144 17968]
 [21310 31796 42072 46554 47402 44988 36886 26206 18977]
 [20888 30386 40812 45906 46229 42625 33961 25179 19031]
 [20005 27828 35978 40795 40248 35591 28820 22893 18300]
 [18990 25187 30424 32576 31322 28321 24480 20722 17426]
 [17400 21930 25283 25992 24873 23364 21306 18973 16560]
 [15191 18020 20040 20625 20240 19612 18635 17214 15494]]

[[15428 18711 22101 24419 25116 24134 21288 17759 14544]
 [18110 23187 29128 33665 34900 32779 28337 22769 17398]
 [21329 29111 38836 45102 46290 42990 35900 28103 20653]
 [23889 34544 46529 52202 52804 49735 41232 31275 22767]
 [25093 37151 49948 55249 55048 51208 41993 31573 23387]
 [25612 37152 49448 55376 54319 48363 38712 29787 22834]
 [25805 35287 44022 49480 48188 41632 33918 27085 21532]
 [25004 32245 37452 40592 39652 34998 29722 24585 20146]
 [22718 28575 32089 33485 32231 29280 25939 22244 18854]
 [19057 23364 25956 26341 25239 23780 22097 19824 17446]]

[[17814 21834 25573 27768 28740 28614 26360 22281 17683]
 [20830 26509 31716 34955 36365 35855 33354 28753 22143]
 [24240 32057 39552 45113 47082 44527 39705 34085 26315]
 [26842 37063 47263 54536 56113 51519 43605 36307 28243]
 [28400 40403 51761 58516 58875 52787 43610 35845 28318]
 [29498 41466 52002 57501 56150 49230 41056 34063 27202]
 [29845 39154 46971 51981 50881 44433 37901 31567 25298]
 [28792 35399 40859 45013 44877 39796 34446 28980 23453]
 [26277 31892 35561 38110 37983 34576 30673 26135 21618]
 [21970 26884 29801 30754 30026 28258 25865 22717 19504]]

[[20021 24536 27855 29455 30659 31410 30269 26570 20628]
 [23386 29222 33211 36043 38346 38843 37385 33729 26523]
 [27130 34480 39903 44637 46991 46202 43429 38914 31428]
 [30016 39152 46060 51094 52394 50327 46321 40606 33132]
 [31359 42501 50001 54344 54944 51833 46171 39545 32650]
 [31711 43895 51634 55033 54185 49692 43486 37439 31022]
 [31294 41827 49218 52686 50669 44867 39814 35021 28807]
 [29923 37642 43890 47825 46560 41417 37028 32599 26600]
 [27157 33633 38096 41211 40669 37317 33989 29644 24208]
 [22701 27975 31518 33089 32791 31293 28979 25349 21276]]

[[21334 25729 28431 29798 31323 32680 32257 29002 22658]
 [24840 30361 33821 37034 39802 40667 39727 36365 29103]
 [28752 35657 40694 44873 46469 46478 45273 41693 34261]
 [31890 40461 46079 48910 49474 49014 47494 43346 35963]
 [33003 43508 49430 51617 51747 50620 47773 42072 35033]
 [32162 44174 51135 53505 53065 50392 45373 39270 32903]
 [30609 42502 50642 52973 50269 45389 40932 36539 30485]
 [29028 38830 46807 49877 46777 42000 38400 34198 28109]
 [26099 33776 39813 42642 41282 38551 35771 31234 25443]
 [22047 27099 31116 33074 33285 32491 30380 26492 22089]]

[[21233 25409 27872 29162 30675 32076 31913 28844 22680]
 [24685 29939 33198 36284 39041 40067 39196 36023 28999]
 [28504 35143 40095 44393 46275 46360 45044 41340 34072]
 [31674 39988 45825 49362 50276 49685 47830 43372 35890]
 [32846 43205 49661 52741 53233 51850 48422 42266 34966]
 [31820 43854 51603 54821 54690 51661 45937 39387 32785]
 [30097 42568 51772 54621 52455 47351 41889 36433 30251]
 [28542 39258 48163 51586 49109 43945 39137 33786 27689]
 [25813 33948 40697 43683 42613 39495 35643 30480 24936]
 [22024 27216 31694 33763 33566 32282 29823 25975 21744]]

[[19928 23757 26316 27689 28878 29946 29523 26356 20923]
 [23054 28136 31410 33838 36086 37116 36218 33034 26463]
 [26505 33081 37978 42549 45358 45266 42688 38185 31096]
 [29576 38084 44783 50665 53572 52120 47348 40956 33169]
 [31167 42033 50577 57579 60473 57269 49272 40838 32812]
 [31143 44158 54542 61351 62589 56983 47535 38863 31088]
 [30777 43387 53281 59200 59286 52363 43459 35826 28774]
 [29802 39725 47095 52049 52456 46549 38924 32165 26040]
 [27004 35015 40446 43869 43755 39323 33860 28319 23186]
 [22574 28724 33372 35310 34167 31244 28052 24168 20251]]

[[17795 21138 23913 25578 26542 26960 25815 22505 18149]
 [20441 25231 29097 31411 32919 33591 32375 28543 22277]
 [23440 30109 35655 39878 42569 42227 39081 33917 26282]
 [26318 35325 43660 51116 54908 52761 45877 37690 28816]
 [28320 39899 51789 60724 62894 59218 50425 39385 29314]
 [29302 42852 56895 65535 65326 59843 50510 38516 28273]
 [29938 42189 53977 62914 63842 57007 46207 34958 26236]
 [29209 38444 46199 52939 54802 48647 38727 30129 23437]
 [26360 33948 39076 42659 42982 38119 31490 25472 20550]
 [21724 27932 32277 34039 32648 29001 25211 21274 17764]]

[[15646 18466 21090 22857 23506 23184 21412 18362 14999]
 [17680 21769 25785 28602 29932 29805 27546 23017 17722]
 [20100 25980 32256 37344 39804 38934 34727 28156 20816]
 [22495 30788 40542 48607 51429 49458 42747 32956 23427]
 [24248 34970 48234 56258 57043 54442 48003 35854 24715]
 [25197 37565 52414 59798 59250 55098 48255 35765 24416]
 [25921 37794 50639 58913 58706 53012 44183 32253 22878]
 [25273 34924 43409 50514 51879 45627 36026 26930 20225]
 [22529 30167 35767 39305 38981 33807 27604 21970 17507]
 [18441 23814 27901 29302 27558 24349 21230 17978 15133]]

[[13723 16041 18269 19732 19916 18966 17093 14616 12129]
 [15136 18243 21596 24126 24953 24107 21369 17450 13705]
 [16790 21010 26159 30907 33270 31802 27098 21064 15787]
 [18357 24258 32256 40126 43415 41337 34231 25153 17999]
 [19290 26991 37826 46057 48079 46075 39379 28248 19451]
 [19567 28306 40281 47805 48801 46344 40201 28823 19707]
 [19711 28382 39356 47016 47437 43961 36660 26288 18774]
 [18886 26459 34390 40309 40885 36444 29204 21991 16807]
 [16717 22319 27414 30038 29236 26061 22105 18079 14715]
 [14022 17243 20048 21094 20212 18753 17033 14934 12871]]
```

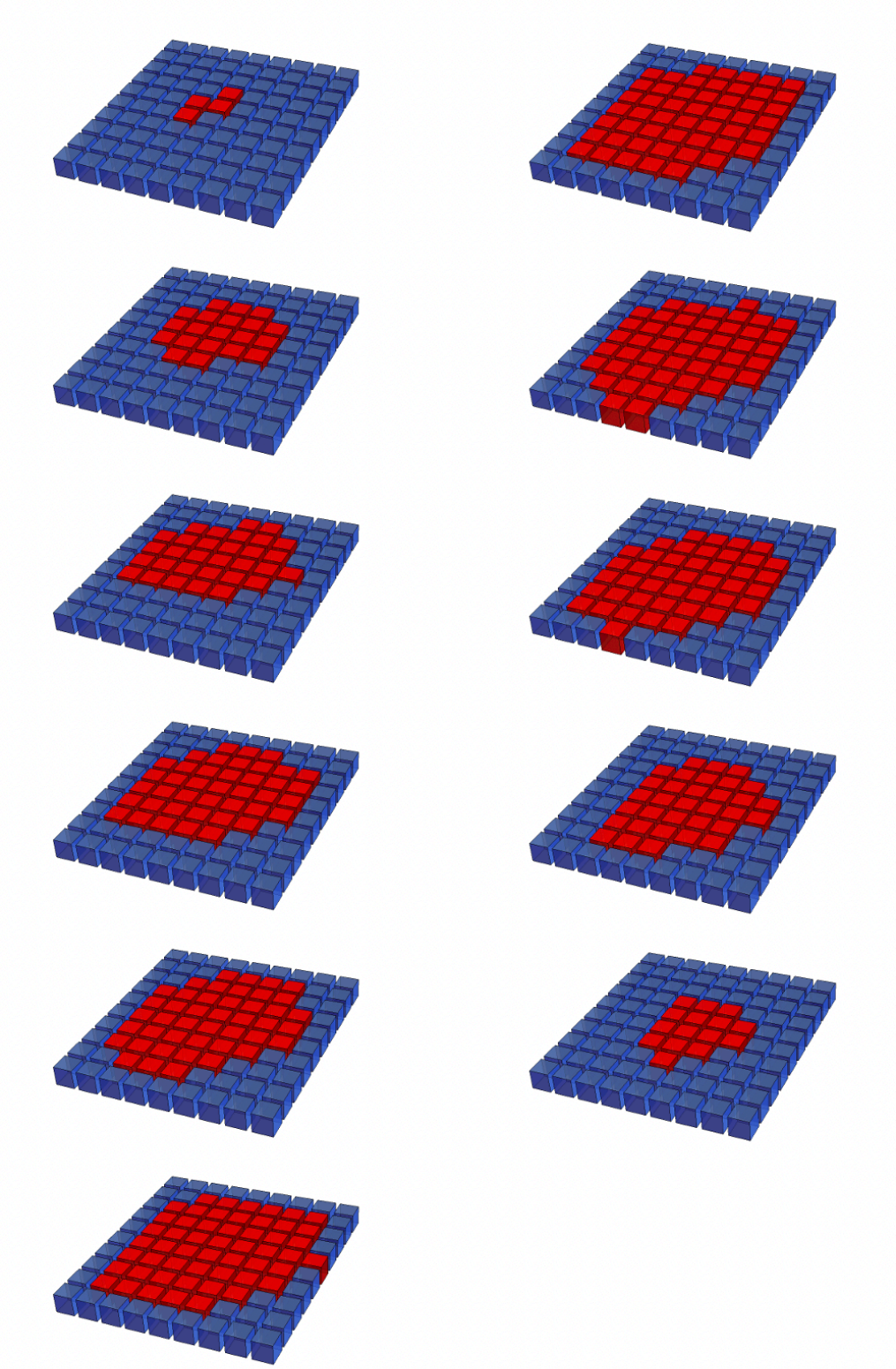
Appendix E – Determining which Voxels Lie within the Contours – Imperatively (cont)

This is the modified composite dose.pixel_array data where those voxels within the contours are set to zero.

```
[11104 13036 15039 16155 16096 14880 12924 11105  9504]
[12725 15746 19507 22608 22936 20285 16512 13219 10634]
[14712 19459 26361 31489 32260 28418 21574 16060 12302]
[16133 22873 31626 36469     0 34438 26289 18519 13847]
[16375 23829 32886     0     0 36053 28048 19875 14890]
[15779 21995 30261     0 36789 33512 26239 19776 15388]
[14719 19223 24773 29227 30200 27325 22648 18439 15125]
[13681 16868 20164 22347 22946 21901 19616 17001 14610]
[12736 14935 17049 18155 18531 18391 17349 15796 14073]
[11657 13214 14552 15430 15805 15836 15404 14537 13381]

[13280 15957 18778 20573 20777 19402 16784 14123 11765]
[15588 19939 25323 29602 30458 27449 22436 17540 13589]
[18449 25391 34249     0     0     0 29742 21849 16014]
[20639 30145     0     0     0     0     0 25144 17968]
[21310 31796     0     0     0     0     0 26206 18977]
[20888 30386     0     0     0     0     0 25179 19031]
[20005 27828 35978     0     0 35591 28820 22893 18300]
[18990 25187 30424 32576 31322 28321 24480 20722 17426]
[17400 21930 25283 25992 24873 23364 21306 18973 16560]
[15191 18020 20040 20625 20240 19612 18635 17214 15494]

[15428 18711 22101 24419 25116 24134 21288 17759 14544]
[18110 23187 29128 33665     0     0 28337 22769 17398]
[21329 29111     0     0     0     0     0 28103 20653]
[23889     0     0     0     0     0     0 31275 22767]
[25093     0     0     0     0     0     0     0 23387]
[25612     0     0     0     0     0     0 29787 22834]
[25805     0     0     0     0     0 33918 27085 21532]
[25004 32245 37452 40592 39652 34998 29722 24585 20146]
[22718 28575 32089 33485 32231 29280 25939 22244 18854]
[19057 23364 25956 26341 25239 23780 22097 19824 17446]

[17814 21834 25573 27768 28740 28614 26360 22281 17683]
[20830 26509 31716     0     0     0 33354 28753 22143]
[24240 32057     0     0     0     0     0     0 26315]
[26842     0     0     0     0     0     0     0 28243]
[28400     0     0     0     0     0     0     0 28318]
[29498     0     0     0     0     0     0     0 27202]
[29845     0     0     0     0     0     0 31567 25298]
[28792     0     0     0     0     0 34446 28980 23453]
[26277 31892 35561 38110 37983 34576 30673 26135 21618]
[21970 26884 29801 30754 30026 28258 25865 22717 19504]

[20021 24536 27855 29455 30659 31410 30269 26570 20628]
[23386 29222 33211     0     0     0     0 33729 26523]
[27130 34480     0     0     0     0     0 38914 31428]
[30016     0     0     0     0     0     0     0 33132]
[31359     0     0     0     0     0     0     0 32650]
[31711     0     0     0     0     0     0     0 31022]
[31294     0     0     0     0     0     0 35021 28807]
[29923     0     0     0     0     0 37028 32599 26600]
[27157 33633     0     0     0 37317 33989 29644 24208]
[22701 27975 31518 33089 32791 31293 28979 25349 21276]

[21334 25729 28431 29798 31323 32680 32257 29002 22658]
[24840 30361     0     0     0     0     0     0 29103]
[28752     0     0     0     0     0     0     0 34261]
[31890     0     0     0     0     0     0     0 35963]
[33003     0     0     0     0     0     0     0     0]
[32162     0     0     0     0     0     0     0 32903]
[30609     0     0     0     0     0     0     0 30485]
[29028     0     0     0     0     0     0 34198 28109]
[26099     0     0     0     0     0 35771 31234 25443]
[22047 27099 31116 33074 33285 32491 30380 26492 22089]

[21233 25409 27872 29162 30675 32076 31913 28844 22680]
[24685 29939 33198     0     0     0     0     0 28999]
[28504     0     0     0     0     0     0     0 34072]
[31674     0     0     0     0     0     0     0 35890]
[32846     0     0     0     0     0     0     0 34966]
[31820     0     0     0     0     0     0     0 32785]
[30097     0     0     0     0     0     0     0 30251]
[28542     0     0     0     0     0     0 33786 27689]
[25813     0     0     0     0     0 35643 30480 24936]
[22024 27216 31694 33763 33566 32282 29823 25975 21744]

[19928 23757 26316 27689 28878 29946 29523 26356 20923]
[23054 28136 31410 33838 36086     0 36218 33034 26463]
[26505 33081     0     0     0     0     0     0 31096]
[29576     0     0     0     0     0     0     0 33169]
[31167     0     0     0     0     0     0     0 32812]
[31143     0     0     0     0     0     0     0 31088]
[30777     0     0     0     0     0     0     0 28774]
[29802     0     0     0     0     0     0 32165 26040]
[27004 35015     0     0     0     0 33860 28319 23186]
[22574 28724 33372     0     0 31244 28052 24168 20251]

[17795 21138 23913 25578 26542 26960 25815 22505 18149]
[20441 25231 29097 31411 32919 33591 32375 28543 22277]
[23440 30109 35655     0     0     0     0 33917 26282]
[26318 35325     0     0     0     0     0     0 28816]
[28320     0     0     0     0     0     0     0 29314]
[29302     0     0     0     0     0     0     0 28273]
[29938     0     0     0     0     0     0     0 26236]
[29209     0     0     0     0     0     0 30129 23437]
[26360     0     0     0     0     0 31490 25472 20550]
[21724 27932 32277     0 32648 29001 25211 21274 17764]

[15646 18466 21090 22857 23506 23184 21412 18362 14999]
[17680 21769 25785 28602 29932 29805 27546 23017 17722]
[20100 25980 32256     0     0     0 34727 28156 20816]
[22495 30788     0     0     0     0     0 32956 23427]
[24248 34970     0     0     0     0     0     0 24715]
[25197 37565     0     0     0     0     0     0 24416]
[25921 37794     0     0     0     0     0 32253 22878]
[25273 34924     0     0     0     0     0 26930 20225]
[22529 30167     0     0     0 33807 27604 21970 17507]
[18441 23814 27901 29302 27558 24349 21230 17978 15133]

[13723 16041 18269 19732 19916 18966 17093 14616 12129]
[15136 18243 21596 24126 24953 24107 21369 17450 13705]
[16790 21010 26159 30907 33270 31802 27098 21064 15787]
[18357 24258 32256     0     0     0 34231 25153 17999]
[19290 26991 37826     0     0     0     0 28248 19451]
[19567 28306 40281     0     0     0     0 28823 19707]
[19711 28382 39356     0     0     0 36660 26288 18774]
[18886 26459 34390 40309     0 36444 29204 21991 16807]
[16717 22319 27414 30038 29236 26061 22105 18079 14715]
[14022 17243 20048 21094 20212 18753 17033 14934 12871]
```

Appendix E – Determining which Voxels Lie within the Contours – Imperatively (cont)

We can see how this data looks in *Mathematica* with tumor voxels within the contour in red.

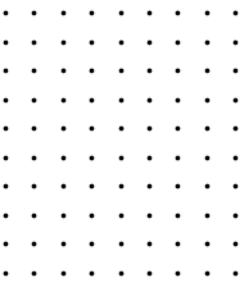


Appendix F – Determining which Voxels Lie within the Contours – Functionally

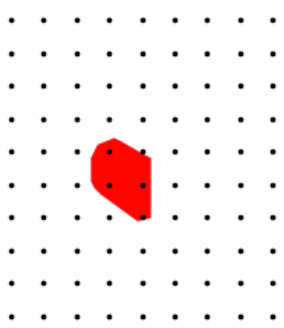
Alternatively, we can bring the contour polygon vertex data directly into *Mathematica* and determine the voxels lying within the contours in the functional manner demonstrated here. The main program here comes from Wellin's "Programming with *Mathematica*" (2013). Also, the program in Appendix G (Obtaining Vertices of Contour Polygons) is needed to obtain the "contour_slice_data_for_Mathematica."

```
ContourSliceDataRaw = ReadList["/Users/WilliamKassing/Documents/Project/contour_slice_data_for_Mathematica.txt", String];
(* this brings in the contour structue set data produced by the program in Appendix G *)

ContourSliceData = ToExpression[StringReplace[ContourSliceDataRaw, {"[" → "{", "]" → "}", "'" → ""}]];
(* converts this data into a form suitable for Mathematica *)

ContourSliceData[[1]]
(* shows the polygon vertices of the contour in slice 1 as a one-dimensional list *)

{6.78228, 22.5938, 31.1291, 6.78228, 24.4208, 31.1291, 6.57928, 24.5223, 31.1291, 5.86878, 24.9283, 31.1291,
 5.66578, 25.0298, 31.1291, 5.15828, 24.8268, 31.1291, 5.05678, 24.6238, 31.1291, 4.95528, 24.4208, 31.1291,
 4.95528, 23.7103, 31.1291, 5.05678, 23.5073, 31.1291, 5.25978, 23.3043, 31.1291, 6.37628, 22.4923, 31.1291}

Contour1 = Partition[Drop[Flatten[ContourSliceData[[1]]], {3, -1, 3}], 2]
(* drops the z coordinate from the list and partitions the remaining values into x and y coordinates *)

{{6.78228, 22.5938}, {6.78228, 24.4208}, {6.57928, 24.5223}, {5.86878, 24.9283}, {5.66578, 25.0298}, {5.15828, 24.8268},
 {5.05678, 24.6238}, {4.95528, 24.4208}, {4.95528, 23.7103}, {5.05678, 23.5073}, {5.25978, 23.3043}, {6.37628, 22.4923}}

Contour1Polygon = Graphics[{Red, Polygon[Contour1]}, ImageSize → Small]
(* plots the polygon of contour 1 *)
```

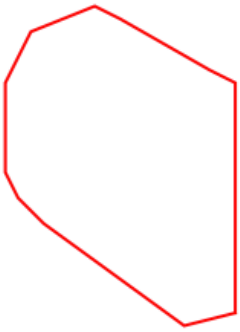
```
xorigin = 2.51927852;
yorigin = 19.5898048;
(* the x and y origins of the dataset from the DICOM file the pixel spacing is 1 mm *)

VoxelGrid = Partition[Flatten[Table[{xorigin + i, yorigin + j}, {j, 0, 9}, {i, 0, 8}]], 2];
(* forms a grid of points showing the location of the voxels *)

Graphics[{PointSize[0.02], Black, Point[VoxelGrid]}, ImageSize → Small]
(* plots these points *)
```



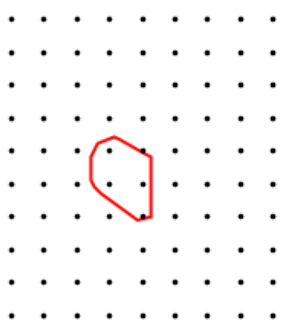
```
Show[Contour1Polygon, Graphics[{PointSize[0.02], Black, Point[VoxelGrid]}]]
(* shows the contour polygon overlying the voxel grid *)
```



```
PathPlot[coords_List] :=
 Graphics[{Red, Thick, Line[coords /. {p1_, pn__} ↦ {p1, pn, p1}]}, ImageSize → Small]
(* plots the contour polygon from the vertice coordinates from using the program in Wellin (p. 419) *)

PathPlot[Contour1]
```



```
Show[PathPlot[Contour1], Graphics[{PointSize[0.02], Black, Point[VoxelGrid]}, ImageSize → Small]]
```



```
TriangleArea[tri : {v1_, v2_, v3_}] :=
 Det[Map[PadRight[#, 3, 1] &, tri]] / 2
(* gives the triangle area, from Wellin *)

PointInPolygonQ[poly : {{_, _} ..}, pt : {x_, y_}] :=
 Module[{edges, e2, e3, e4},
  edges = Partition[poly, 2, 1, 1];
  e2 = DeleteCases[edges, {{x1_, y1_}, {x2_, y2_}} /; y1 == y2];
  e3 = DeleteCases[e2, {{x1_, y1_}, {x2_, y2_}} /;
     (Min[y1, y2] ≥ y || Max[y1, y2] < y)];
  e4 = Map[Reverse@SortBy[#, Last] &, e3]; OddQ[
   Count[TriangleArea[Join[#, {pt}]] & /@ e4, _?Positive]]]
(* from Wellin (p. 425) *)
```

```
InsideContour1 = Map[PointInPolygonQ[Contour1, #] &, VoxelGrid]
(* set the voxels within the contour polygon to True *)
```

```
{False, False, False, False, False, False, False, False, False, False, False, False, False, False, False,
 False, False, False, False, False, False, False, False, False, False, False, False, False, False, False,
 False, True, False, False, False, False, False, False, False, True, True, False, False, False, False,
 False, False, False, True, False, False, False, False, False, False, False, False, False, False, False,
 False, False, False, False, False, False, False, False, False, False, False, False, False, False, False,
 False, False, False, False, False, False, False, False, False, False, False, False, False, False, False}
```

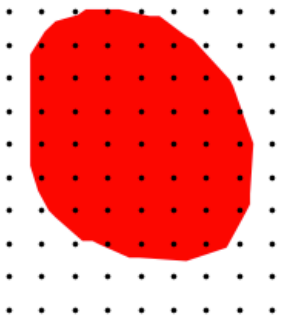
```
VoxelsInsideContour1 = Position[Partition[InsideContour1, 9], True]
(* gives the postions of the True values, this shows one at Row 4, Column 5 etc. )*)
```

```
{{4, 5}, {5, 4}, {5, 5}, {6, 4}}
```

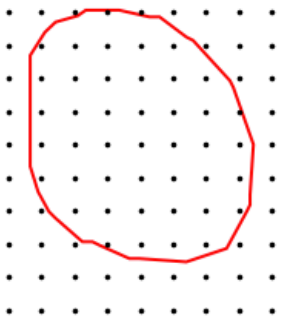
```
(* now we do the same for slice 9 *)
Contour9 = Partition[Drop[Flatten[ContourSliceData[[9]], {3, -1, 3}], 2];
(* drops the z coordinate *)
```

```
Contour9Polygon = Graphics[{Red, Polygon[Contour9]}, ImageSize → Small]
```

```
Show[Contour9Polygon, Graphics[{PointSize[0.02], Black, Point[VoxelGrid]}]]
```



```
Show[PathPlot[Contour9], Graphics[{PointSize[0.02], Black, Point[VoxelGrid]}, ImageSize → Small]]
```



Appendix F – Determining which Voxels Lie within the Contours – Functionally (cont)

```
InsideContour9 = Map[PointInPolygonQ[Contour9, #] &, VoxelGrid]
```

```
{False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False,
 False, False, False, False, True, True, True, True, False, False, False, False, True, True, True, True, True, True,
 False, False, True, True, True, True, True, True, True, False, False, True, True, True, True, True, True, True, False,
 False, True, True, True, True, True, True, True, False, False, True, True, True, True, True, True, False, False, False,
 True, True, True, True, True, False, False, False, False, False, False, True, False, False, False, False, False}
```

```
(* gives the points inside the contour *)
VoxelsInsideContour9 = Position[Partition[InsideContour9, 9], True]
```

```
{{3, 4}, {3, 5}, {3, 6}, {3, 7}, {4, 3}, {4, 4}, {4, 5}, {4, 6}, {4, 7}, {4, 8}, {5, 2}, {5, 3}, {5, 4}, {5, 5}, {5, 6},
 {5, 7}, {5, 8}, {6, 2}, {6, 3}, {6, 4}, {6, 5}, {6, 6}, {6, 7}, {6, 8}, {7, 2}, {7, 3}, {7, 4}, {7, 5}, {7, 6},
 {7, 7}, {7, 8}, {8, 2}, {8, 3}, {8, 4}, {8, 5}, {8, 6}, {8, 7}, {9, 2}, {9, 3}, {9, 4}, {9, 5}, {9, 6}, {10, 4}}
```

Appendix G – Obtaining Vertices of Contour Polygons

Gets the coordinates of the polygon vertices that make up the contours and prints them to a text file for import into *Mathematica*.

```
#this program gets the coordinates of the polygon vertices that make up the contours and prints them
#to a text file for import into Mathematica
import os #imports the operating system module, needed for file and directory functions
import numpy as np #imports the NumPy scientific computing library and makes np shorthand for numpy
os.chdir("/Users/Documents/pydicom-0.9.7") #sets the directory path
import dicom #imports the pydicom package
#the line below reads in the DICOM RTStruct file and names it rtstructfile
rtstructfile=dicom.read_file("structure set.dcm")
num_contourslices=len(rtstructfile.ROIContours[0].Contours) #gets the number of contour slices
textfile=open("contour_slice_data_for_Mathematica.txt","w") #creates a text file in write mode
#the for loop below obtains the coordinates of the contour vertices for each slice of the dataset
for contour_slicenumber in range(0,num_contourslices):
        contour_slice_data=rtstructfile.ROIContours[0].Contours[contour_slicenumber].ContourData
        textfile=open("contour_slice_data_for_Mathematica.txt","a") #opens the text file in append mode
        print >>textfile, contour_slice_data #writes the contour vertices coordinates to the text file
textfile.close() #closes the text file
```

Appendix H – Creating a List of Shot Times and Shot Dose Rates in each Voxel

This Python program forms a list of the Gamma Knife shot times and shot dose-rates for all the voxels in the dose array and writes this list to a text file. In this example there are 6 shots, each with an associated time and dose-rate. The complete list therefore has dimension 11x10x9x6x2.

1. Read in the DICOM RTDose files for each of the Gamma Knife shots.
2. Form a list of the individual shot times.
3. Iterate through the dose array and convert the shot dose to shot dose-rate.
4. Form an array of voxel shot times and voxel shot dose-rates.
5. Convert this array to a list and write this list to a text file.

```
#this program forms a list of the Gamma Knife shot times and shot dose rates for all the voxels in the dose array
#and writes this list to a text file
import os #imports the operating system module, needed for file and directory functions
import numpy as np #imports the NumPy scientific computing library and makes np shorthand for numpy
os.chdir("/Users/Documents/pydicom-0.9.7") #sets the directory path
import dicom #imports the pydicom package
shot1=dicom.read_file("shot1.dcm") #reads in the shot1 dicom rtdose file
shot2=dicom.read_file("shot2.dcm") #reads in the shot2 dicom rtdose file
shot3=dicom.read_file("shot3.dcm") #reads in the shot3 dicom rtdose file
shot4=dicom.read_file("shot4.dcm") #reads in the shot4 dicom rtdose file
shot5=dicom.read_file("shot5.dcm") #reads in the shot5 dicom rtdose file
shot6=dicom.read_file("shot6.dcm") #reads in the shot6 dicom rtdose file
shot_times=[9.03, 7.78, 8.24, 4.90, 4.02, 1.88] #list of the shot times in minutes
shot1_dosescalingfactor=float(shot1.DoseGridScaling) #shot 1 dose scaling factor for converting dose array integer to Gy
shot2_dosescalingfactor=float(shot2.DoseGridScaling) #shot 2 dose scaling factor for converting dose array integer to Gy
shot3_dosescalingfactor=float(shot3.DoseGridScaling) #shot 3 dose scaling factor for converting dose array integer to Gy
shot4_dosescalingfactor=float(shot4.DoseGridScaling) #shot 4 dose scaling factor for converting dose array integer to Gy
shot5_dosescalingfactor=float(shot5.DoseGridScaling) #shot 5 dose scaling factor for converting dose array integer to Gy
shot6_dosescalingfactor=float(shot6.DoseGridScaling) #shot 6 dose scaling factor for converting dose array integer to Gy
shot1_time=shot_times[0] #shot 1 time
shot2_time=shot_times[1] #shot 2 time
shot3_time=shot_times[2] #shot 3 time
shot4_time=shot_times[3] #shot 4 time
shot5_time=shot_times[4] #shot 5 time
shot6_time=shot_times[5] #shot 6 time
textfile=open("shot_times_and_doserates.txt","w") #creates and opens a text file in write mode
#the for loop below iterates through the voxels in the dose array and obtains the dose rates for each shot
#the shot times and the shot dose rates are then placed in a list and written to a text file
for a in range(0,11):
    for b in range(0,10):
        for c in range(0,9):
            shot1_doserate=shot1.pixel_array[a,b,c]*shot1_dosescalingfactor/shot1_time #shot1 doserate in voxel [a,b,c]
            shot2_doserate=shot2.pixel_array[a,b,c]*shot2_dosescalingfactor/shot2_time #shot2 doserate in voxel [a,b,c]
            shot3_doserate=shot3.pixel_array[a,b,c]*shot3_dosescalingfactor/shot3_time #shot3 doserate in voxel [a,b,c]
            shot4_doserate=shot4.pixel_array[a,b,c]*shot4_dosescalingfactor/shot4_time #shot4 doserate in voxel [a,b,c]
            shot5_doserate=shot5.pixel_array[a,b,c]*shot5_dosescalingfactor/shot5_time #shot5 doserate in voxel [a,b,c]
            shot6_doserate=shot6.pixel_array[a,b,c]*shot6_dosescalingfactor/shot6_time #shot6 doserate in voxel [a,b,c]
            #the line below forms a list of the shot dose rates
            shot_doserates=[shot1_doserate,shot2_doserate,shot3_doserate,shot4_doserate,shot5_doserate,shot6_doserate]
            #the line below puts the shot times and shot dose rates together in an array
            shot_times_and_shot_doserates=np.array([shot_times,shot_doserates]).transpose()
            #the line below converts the shot_times_and_shot_doserates array into a list
            shot_times_and_shot_doserates_list=shot_times_and_shot_doserates.tolist()
            textfile=open("shot_times_and_doserates.txt","a") #opens the text file in append mode
            print >>textfile, shot_times_and_shot_doserates_list #writes the shot times and dose rates to the text file
textfile.close() #closes the text file
```

Appendix I – Importing the List of Shot Times and Shot Dose Rates into *Mathematica*

This code imports the Shot Times and Shot Dose Rates produced by the Python program in Appendix H into *Mathematica*.

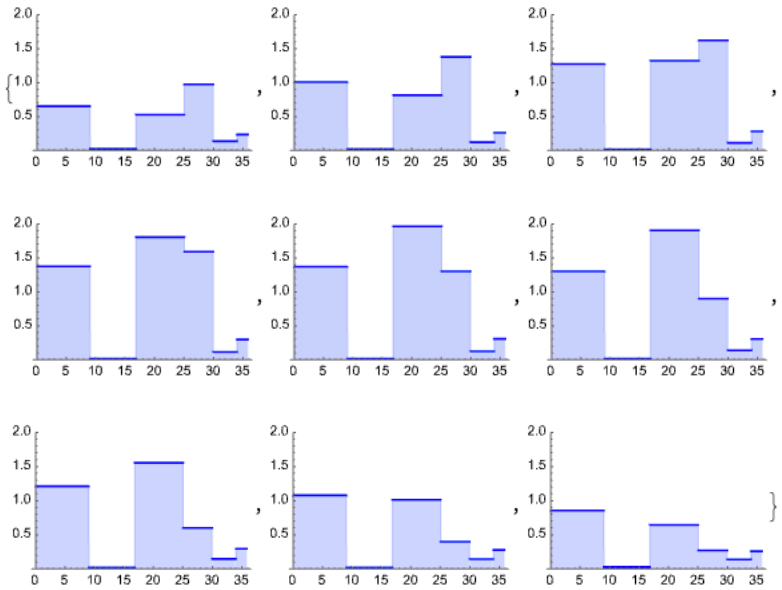
```
ShotTimesAndDoseRatesTextFile = ReadList["/Users/WilliamKassing/Documents/Project/shot_times_and_doserates.txt", String];
(* imports the text file of shot times and shot dose rates produced in Python/Pydicom (see Appendix H) *)

ShotTimesAndDoseRatesTextFileExpression = ToExpression[StringReplace[ShotTimesAndDoseRatesTextFile, {"[" → "{", "]" → "}"}]];
(* changes square brackets to curly brackets *)
(* converts a Mathematica string to a Mathematica expression *)
(* this is a list with dimensions 990x6x2 - the 990 voxels each have 6 shots with 2 values (time and dose rate) *)

Dimensions[ShotTimesAndDoseRatesTextFileExpression]
(* gives the dimension of ShotTimesAndDoseRatesTextFileExpression *)

{990, 6, 2}

VoxelDoseRates = Partition[Partition[ShotTimesAndDoseRatesTextFileExpression, 9], 10];
(* partitions the 990x6x2 list into an 11x10x9x6x2 list representing the 11x10x9 voxel array with each voxel having *)
(* 6 shots made up of 2 values (time and dose rate) *)

Dimensions[VoxelDoseRates]
(* gives the dimension of VoxelDoseRates *)

{11, 10, 9, 6, 2}
```

Appendix J – Gamma Knife Case

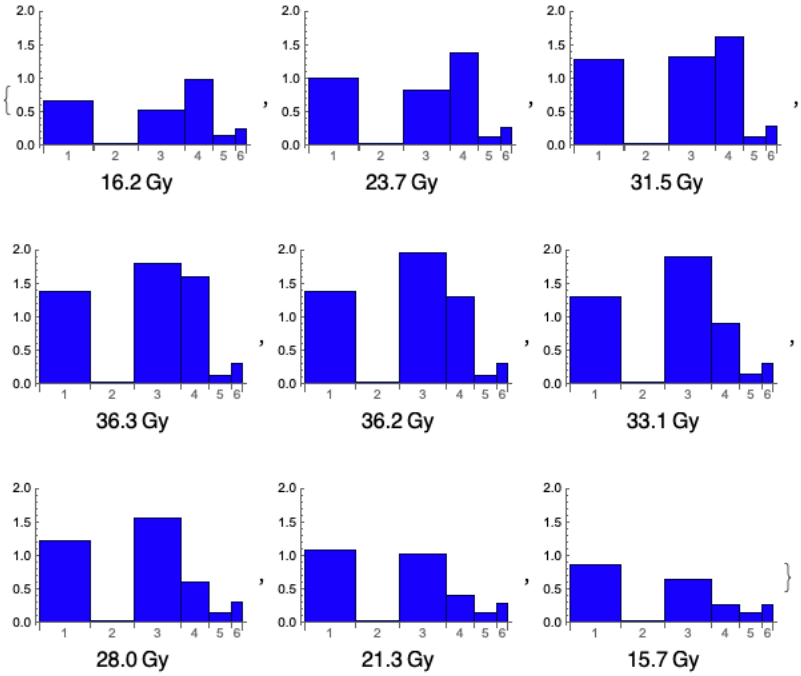
This shows the *Mathematica* code for the complete Gamma Knife case.

```
ShotTimesAndDoseRatesTextFile = ReadList["/Users/WilliamKassing/Documents/Project/shot_times_and_doserates.txt", String];
(* imports the text file of shot times and shot dose rates produced in Python/Pydicom (see Appendix H) *)

ShotTimesAndDoseRatesTextFileExpression = ToExpression[StringReplace[ShotTimesAndDoseRatesTextFile, {"[" → "{", "]" → "}"}]];
(* changes square brackets to curly brackets *)
(* converts a Mathematica string to a Mathematica expression *)
(* this is a list with dimensions 990x6x2 - the 990 voxels each have 6 shots with 2 values (time and dose rate) *)

Dimensions[ShotTimesAndDoseRatesTextFileExpression]
(* gives the dimension of ShotTimesAndDoseRatesTextFileExpression *)

{990, 6, 2}

VoxelDoseRates = Partition[Partition[ShotTimesAndDoseRatesTextFileExpression, 9], 10];
(* partitions the 990x6x2 list into an 11x10x9x6x2 list representing the 11x10x9 voxel array with each voxel having *)
(* 6 shots made up of 2 values (time and dose rate) *)

Dimensions[VoxelDoseRates]
(* gives the dimension of VoxelDoseRates *)

{11, 10, 9, 6, 2}

VoxelDoseRates[[1, 1, 1]]
(* shows the data in the first voxel of the dataset at slice 1, row 1, column 1; the first value is the shot time *)
(* (in minutes), the second value is the shot dose rate (in Gy/min) *)

{{9.03, 0.26924}, {7.78, 0.351386}, {8.24, 0.0121232}, {4.9, 0.0369929}, {4.02, 0.12993}, {1.88, 0.102849}}

VoxelDoses = Apply[Plus, Apply[Times, VoxelDoseRates, {4}], {3}];
(* we get the doses (in Gy) in each voxel by multiplying the shot time and shot dose rate for the 6 shots and *)
(* adding them together; the {4} specifies that we multiple the values (times and dose rates) at level 4 of the *)
(* list structure and the {3} specifies that we add the resulting values at level 3 of the list structure *)

VoxelDoses[[1, 1, 1]]
(* gives the dose in the first voxel of the dataset at slice 1, row 1, column 1 *)

6.16185

Max[VoxelDoses]
(* gives the maximum dose (in Gy) in the voxel dataset *)

36.28

Position[VoxelDoses, Max[VoxelDoses]]
(* gives the position of maximum dose in the voxel dataset *)

{{9, 6, 4}}

VoxelDoses[[9, 6, 4]]
(* shows the maximum dose is at slice 9, row 6, and column 4 of the voxel dataset *)

36.28

TotalTime = VoxelDoseRates[[1, 1, 1, 1, 1]] + VoxelDoseRates[[1, 1, 1, 2, 1]] + VoxelDoseRates[[1, 1, 1, 3, 1]] +
  VoxelDoseRates[[1, 1, 1, 4, 1]] + VoxelDoseRates[[1, 1, 1, 5, 1]] + VoxelDoseRates[[1, 1, 1, 6, 1]]
(* sums the times in the VoxelDoseRates list above to give the total treatment time *)
(* i.e., 9.03 + 7.78 + 8.24 + 4.9 + 4.02 + 1.88 *)

35.85

t1 = 0;
t2 = VoxelDoseRates[[1, 1, 1, 1, 1]];
t3 = VoxelDoseRates[[1, 1, 1, 1, 1]] + VoxelDoseRates[[1, 1, 1, 2, 1]];
t4 = VoxelDoseRates[[1, 1, 1, 1, 1]] + VoxelDoseRates[[1, 1, 1, 2, 1]] + VoxelDoseRates[[1, 1, 1, 3, 1]];
t5 = VoxelDoseRates[[1, 1, 1, 1, 1]] + VoxelDoseRates[[1, 1, 1, 2, 1]] + VoxelDoseRates[[1, 1, 1, 3, 1]] + VoxelDoseRates[[1, 1, 1, 4, 1]];
```

```
t6 = VoxelDoseRates[[1, 1, 1, 1, 1]] + VoxelDoseRates[[1, 1, 1, 2, 1]] + VoxelDoseRates[[1, 1, 1, 3, 1]] + VoxelDoseRates[[1, 1, 1, 4, 1]] +
    VoxelDoseRates[[1, 1, 1, 5, 1]];
t7 = VoxelDoseRates[[1, 1, 1, 1, 1]] + VoxelDoseRates[[1, 1, 1, 2, 1]] + VoxelDoseRates[[1, 1, 1, 3, 1]] +
    VoxelDoseRates[[1, 1, 1, 4, 1]] + VoxelDoseRates[[1, 1, 1, 5, 1]] + VoxelDoseRates[[1, 1, 1, 6, 1]] ;
(* sets t1 through t7 equal to the cumulative shot times *)
(* t1=0 t2=9.03 t3=16.81 t4=25.05 t5=29.95 t6=33.97 t7=35.85 *)

VoxelMu1 = 0.0608; (* Repair half-life of 0.19 h *)
VoxelMu2 = 0.0053; (* Repair half-life of 2.16 h *)
AlphaBeta = 10; (* α/β = 10 Gy *)
(* sets the repair rate constants Mu1 and Mu2 (in inverse minutes) and the α/β value (in Gy) *)

UnitStepShot1T = (UnitStep[t - t1] - UnitStep[t - t2]);
UnitStepShot2T = (UnitStep[t - t2] - UnitStep[t - t3]);
UnitStepShot3T = (UnitStep[t - t3] - UnitStep[t - t4]);
UnitStepShot4T = (UnitStep[t - t4] - UnitStep[t - t5]);
UnitStepShot5T = (UnitStep[t - t5] - UnitStep[t - t6]);
UnitStepShot6T = (UnitStep[t - t6] - UnitStep[t - t7]);
(* we use the Mathematica UnitStep function and the cumulative shot times above to form unit step functions *)
(* for each of the 6 shots *)

Shot1VoxelDoseRateT = Map[(# UnitStepShot1T) &, VoxelDoseRates, {5}][[All, All, All, 1, 2]];
Shot2VoxelDoseRateT = Map[(# UnitStepShot2T) &, VoxelDoseRates, {5}][[All, All, All, 2, 2]] ;
Shot3VoxelDoseRateT = Map[(# UnitStepShot3T) &, VoxelDoseRates, {5}][[All, All, All, 3, 2]] ;
Shot4VoxelDoseRateT = Map[(# UnitStepShot4T) &, VoxelDoseRates, {5}][[All, All, All, 4, 2]] ;
Shot5VoxelDoseRateT = Map[(# UnitStepShot5T) &, VoxelDoseRates, {5}][[All, All, All, 5, 2]] ;
Shot6VoxelDoseRateT = Map[(# UnitStepShot6T) &, VoxelDoseRates, {5}][[All, All, All, 6, 2]] ;
(* the unit step functions above are of unit height, so we need to scale these step functions so they correspond *)
(* with the dose rates of each shot in each voxel; as an example Shot1VoxelDoseRateT is formed by mapping the *)
(* UnitStepShot1T function over the VoxelDoseRates list and multiplying this function by the appropriate entry in *)
(* the VoxelDoseRates list; the Mathematica Part function is used to pick out the dose rate components from the *)
(* list; the VoxelDoseRates list has dimensions 11x10x9x6x2, and the Part specifier[[All,All,All,1,2]] indicates that *)
(* for all 11 slices, all 10 rows, and all 9 columns of the dataset, we want the first component of 6 (shot 1) *)
(* and the second component of 2 (the dose rate)and similarly for the remaining five shots, the specifier {5} *)
(* indicates that the mapping is applied at level 5 of the VoxelDoseRates list structure *)

VoxelDoseRateInputT = Shot1VoxelDoseRateT + Shot2VoxelDoseRateT + Shot3VoxelDoseRateT + Shot4VoxelDoseRateT +
    Shot5VoxelDoseRateT + Shot6VoxelDoseRateT;
(* the individual shot dose rate step functions are added together to form the voxel-by-voxel dose rate *)
(* functions in a list named VoxelDoseRateInputT *)

VoxelDoseRatesInputFunctionsPlot = ParallelTable[Plot[(VoxelDoseRateInputT[[i, j, k]]), {t, 0, TotalTime},
     Filling → Bottom, PlotStyle → Blue, PlotRange → {{All, All}, {0, 2.0}}], {i, 11}, {j, 10}, {k, 9}];
(* we make a table of plots of the voxel-by-voxel dose rate input functions *)

VoxelDoseRatesInputFunctionsPlot[[9, 6]]
(* we show these dose rate input functions for slice 9, row 6 of the dataset *)
```

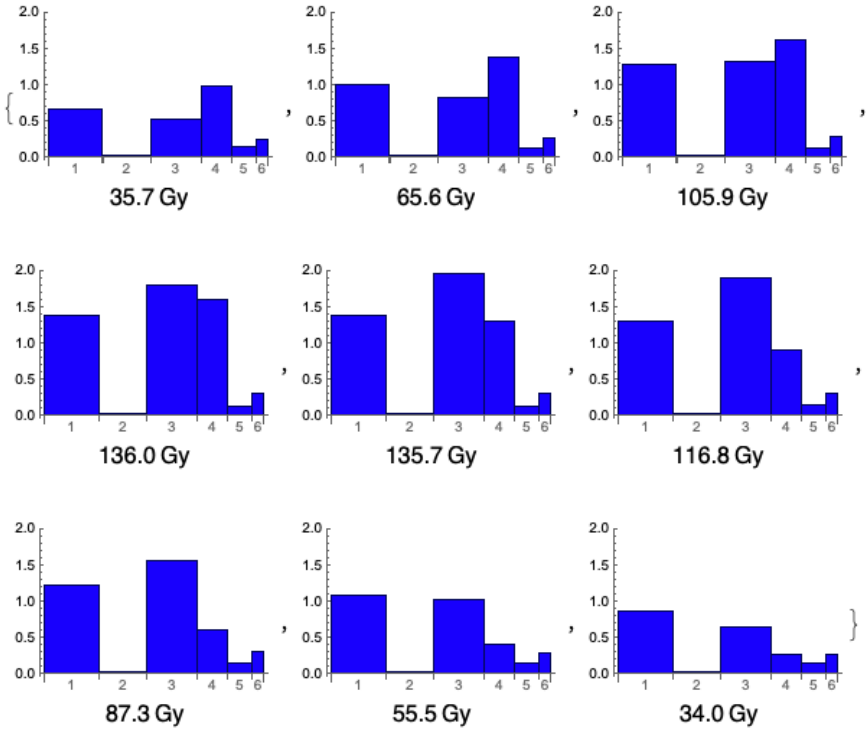


```
VoxelDoseRatesTable = Table[Table[Grid[{{RectangleChart[VoxelDoseRates[[i, j, k]], ChartLabels → {"1", "2", "3", "4", "5", "6"},
        ChartStyle → {Blue}, PlotRange → {{All, All}, {All, 2.0}}]},
      {Text[Style[NumberForm[VoxelDoses[[i, j, k]] "Gy", {4, 1}],
        16, FontFamily → "Helvetica"]]}], {k, 9}], {i, 11}, {j, 10}];
(* we make a table of plots of the actual dose rate profiles in the voxels along with the voxel dose *)

VoxelDoseRatesTable[[9, 6]]
(* we show these dose rate profiles and doses for slice 9, row 6 of the dataset and note how they match the dose *)
(* input functions above *)
```



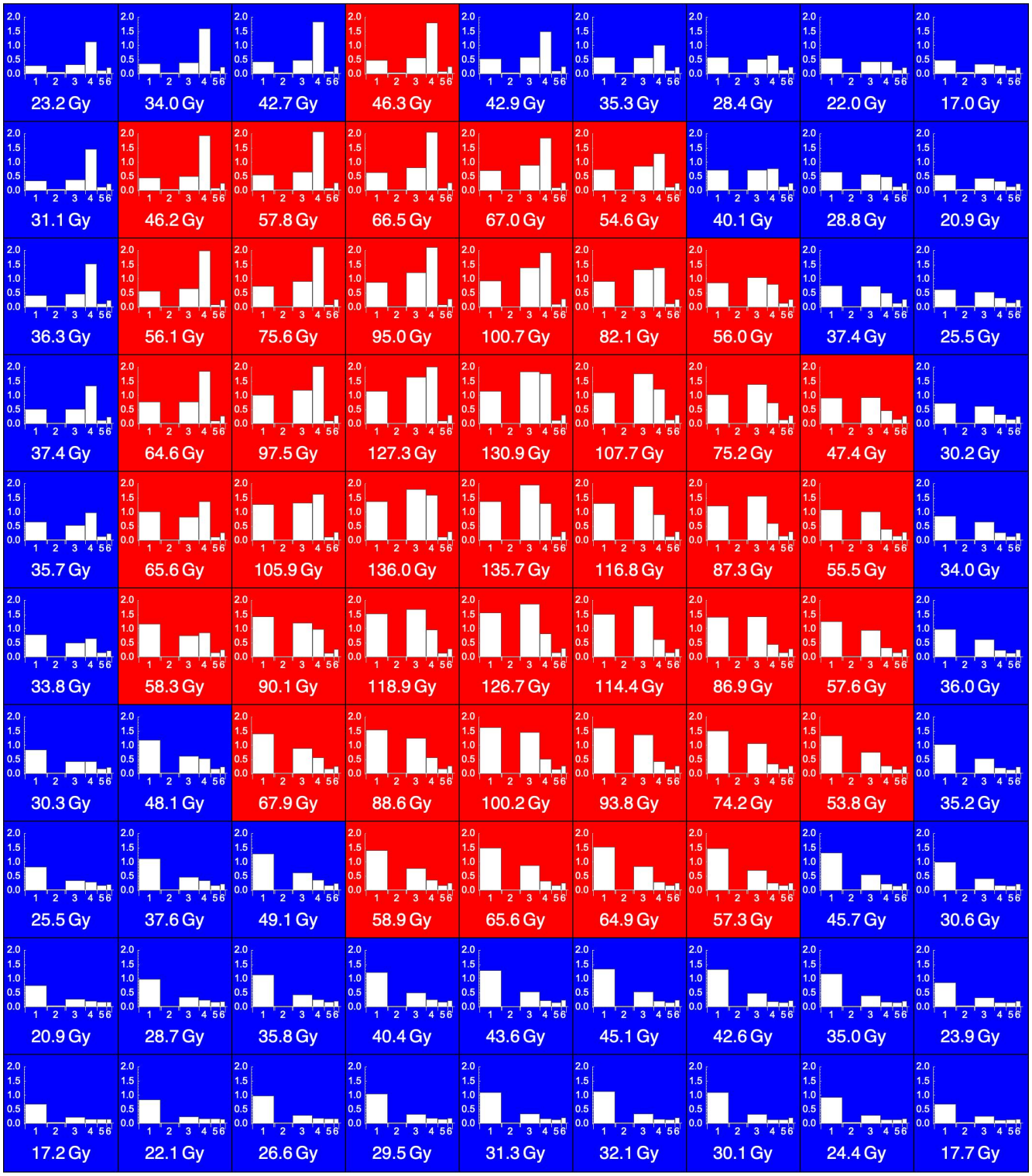

```
VoxelPsi1 =
  ParallelMap[ 2 * (
       (* shot 1 *)
       Integrate[# * Exp[-VoxelMu1 * t] * (Integrate[# * Exp[VoxelMu1 * w], {w, t1, t}]), {t, t1, t2}] +
       (* shot 2 *)
       Integrate[# * Exp[-VoxelMu1 * t] * (Integrate[# * Exp[VoxelMu1 * w], {w, t2, t}] +
           Integrate[# * Exp[VoxelMu1 * t], {t, t1, t2}]), {t, t2, t3}] +
       (* shot 3 *)
       Integrate[# * Exp[-VoxelMu1 * t] * (Integrate[# * Exp[VoxelMu1 * w], {w, t3, t}] +
           Integrate[# * Exp[VoxelMu1 * t], {t, t1, t3}]), {t, t3, t4}] +
       (* shot 4 *)
       Integrate[# * Exp[-VoxelMu1 * t] * (Integrate[# * Exp[VoxelMu1 * w], {w, t4, t}] +
           Integrate[# * Exp[VoxelMu1 * t], {t, t1, t4}]), {t, t4, t5}] +
       (* shot 5 *)
       Integrate[# * Exp[-VoxelMu1 * t] * (Integrate[# * Exp[VoxelMu1 * w], {w, t5, t}] +
           Integrate[# * Exp[VoxelMu1 * t], {t, t1, t5}]), {t, t5, t6}] +
       (* shot 6 *)
       Integrate[# * Exp[-VoxelMu1 * t] * (Integrate[# * Exp[VoxelMu1 * w], {w, t6, t}] +
           Integrate[# * Exp[VoxelMu1 * t], {t, t1, t6}]), {t, t6, t7}])
    &, VoxelDoseRateInputT]; (* VoxelDoseRateInputT is a list containing the voxel dose rate input functions *)
(* we perform the convolution of the repair function with the dose rate input functions for Mu1 *)


VoxelPsi2 =
  ParallelMap[ 2 * (
       (* shot 1 *)
       Integrate[# * Exp[-VoxelMu2 * t] * (Integrate[# * Exp[VoxelMu2 * w], {w, t1, t}]), {t, t1, t2}] +
       (* shot 2 *)
       Integrate[# * Exp[-VoxelMu2 * t] * (Integrate[# * Exp[VoxelMu2 * w], {w, t2, t}] +
           Integrate[# * Exp[VoxelMu2 * t], {t, t1, t2}]), {t, t2, t3}] +
```

```
        (* shot 3 *)
        Integrate[#*Exp[-VoxelMu2*t]*(Integrate[#*Exp[VoxelMu2*w], {w, t3, t}] +
            Integrate[#*Exp[VoxelMu2*t], {t, t1, t3}]), {t, t3, t4}] +
        (* shot 4 *)
        Integrate[#*Exp[-VoxelMu2*t]*(Integrate[#*Exp[VoxelMu2*w], {w, t4, t}] +
            Integrate[#*Exp[VoxelMu2*t], {t, t1, t4}]), {t, t4, t5}] +
        (* shot 5 *)
        Integrate[#*Exp[-VoxelMu2*t]*(Integrate[#*Exp[VoxelMu2*w], {w, t5, t}] +
            Integrate[#*Exp[VoxelMu2*t], {t, t1, t5}]), {t, t5, t6}] +
        (* shot 6 *)
        Integrate[#*Exp[-VoxelMu2*t]*(Integrate[#*Exp[VoxelMu2*w], {w, t6, t}] +
            Integrate[#*Exp[VoxelMu2*t], {t, t1, t6}]), {t, t6, t7}])
    &, VoxelDoseRateInputT]; (* VoxelDoseRateInputT is a list containing the voxel dose rate input functions *)
(* we perform the convolution of the repair function with the dose rate input functions for Mu2 *)

VoxelBeds = (VoxelDoses + (Map[(0.5051 #) &, VoxelPsi1] + Map[(0.4949 #) &, VoxelPsi2]) / (AlphaBeta));
(* we calculate the voxel BEDs as in Eq. 23 of Section 3.2 *)

VoxelBedsTable = Table[Table[Grid[{{RectangleChart[VoxelDoseRates[[i, j, k]], ChartLabels → {"1", "2", "3", "4", "5", "6"},
        ChartStyle → {Blue}, PlotRange → {{All, All}, {All, 2.0}}]},
      {Text[Style[NumberForm[VoxelBeds[[i, j, k]], "Gy", {4, 1}], 16,
        FontFamily → "Helvetica"]]}] , {k, 9}], {i, 11}, {j, 10}];
 (* we make a table of plots of the dose rate profiles in the voxels along with the voxel BEDs *)

VoxelBedsTable[[9, 6]]
(* we show these dose rate profiles and BEDs for slice 9, row 6 of the dataset *)
```




```
VoxelBedsTable = Table[Table[Grid[{{RectangleChart[VoxelDoseRates[[i, j, k]], ChartLabels → {"1", "2", "3", "4", "5", "6"},
        AxesStyle → Directive[GrayLevel[1], 10], ChartStyle → {GrayLevel[1]}, PlotRange → {{All, All}, {All, 2.0}}]},
      {Text[Style[NumberForm[VoxelBeds[[i, j, k]] ], "Gy", {4, 1}], 18, FontFamily → "Helvetica", GrayLevel[1]]]}],
    {k, 9}], {i, 11}, {j, 10}];
(* we make a table of plots of the dose rate profiles in the voxels along with the voxel BEDs, we make the *)
(* plots white using GrayLevel[1] so they show up in our next graphic *)
```

```
Slice9VoxelBEDs = Grid[{
   {VoxelBedsTable[[9, 10, 1]], VoxelBedsTable[[9, 10, 2]], VoxelBedsTable[[9, 10, 3]], Item[VoxelBedsTable[[9, 10, 4]],
     Background → Red], VoxelBedsTable[[9, 10, 5]], VoxelBedsTable[[9, 10, 6]], VoxelBedsTable[[9, 10, 7]],
    VoxelBedsTable[[9, 10, 8]],
    VoxelBedsTable[[9, 10, 9]]}, {VoxelBedsTable[[9, 9, 1]], Item[VoxelBedsTable[[9, 9, 2]], Background → Red],
    Item[VoxelBedsTable[[9, 9, 3]], Background → Red], Item[VoxelBedsTable[[9, 9, 4]], Background → Red],
    Item[VoxelBedsTable[[9, 9, 5]],
     Background → Red], Item[VoxelBedsTable[[9, 9, 6]], Background → Red], VoxelBedsTable[[9, 9, 7]], VoxelBedsTable[[9, 9, 8]],
    VoxelBedsTable[[9, 9, 9]]}, {VoxelBedsTable[[9, 8, 1]], Item[VoxelBedsTable[[9, 8, 2]], Background → Red],
    Item[VoxelBedsTable[[9, 8, 3]], Background → Red], Item[VoxelBedsTable[[9, 8, 4]], Background → Red],
    Item[VoxelBedsTable[[9, 8, 5]],
     Background → Red], Item[VoxelBedsTable[[9, 8, 6]], Background → Red], Item[VoxelBedsTable[[9, 8, 7]], Background → Red],
    VoxelBedsTable[[9, 8, 8]], VoxelBedsTable[[9, 8, 9]]},
   {VoxelBedsTable[[9, 7, 1]], Item[VoxelBedsTable[[9, 7, 2]], Background → Red], Item[VoxelBedsTable[[9, 7, 3]], Background → Red],
    Item[VoxelBedsTable[[9, 7, 4]], Background → Red], Item[VoxelBedsTable[[9, 7, 5]],
     Background → Red], Item[VoxelBedsTable[[9, 7, 6]], Background → Red], Item[VoxelBedsTable[[9, 7, 7]], Background → Red],
    Item[VoxelBedsTable[[9, 7, 8]], Background → Red], VoxelBedsTable[[9, 7, 9]]},
   {VoxelBedsTable[[9, 6, 1]], Item[VoxelBedsTable[[9, 6, 2]], Background → Red], Item[VoxelBedsTable[[9, 6, 3]], Background → Red],
    Item[VoxelBedsTable[[9, 6, 4]], Background → Red], Item[VoxelBedsTable[[9, 6, 5]], Background → Red],
    Item[VoxelBedsTable[[9, 6, 6]],
     Background → Red], Item[VoxelBedsTable[[9, 6, 7]], Background → Red], Item[VoxelBedsTable[[9, 6, 8]], Background → Red],
    VoxelBedsTable[[9, 6, 9]]}, {VoxelBedsTable[[9, 5, 1]], Item[VoxelBedsTable[[9, 5, 2]], Background → Red],
    Item[VoxelBedsTable[[9, 5, 3]], Background → Red], Item[VoxelBedsTable[[9, 5, 4]], Background → Red],
    Item[VoxelBedsTable[[9, 5, 5]],
     Background → Red], Item[VoxelBedsTable[[9, 5, 6]], Background → Red], Item[VoxelBedsTable[[9, 5, 7]], Background → Red],
    Item[VoxelBedsTable[[9, 5, 8]], Background → Red], VoxelBedsTable[[9, 5, 9]]},
   {VoxelBedsTable[[9, 4, 1]], VoxelBedsTable[[9, 4, 2]], Item[VoxelBedsTable[[9, 4, 3]], Background → Red],
    Item[VoxelBedsTable[[9, 4, 4]],
     Background → Red], Item[VoxelBedsTable[[9, 4, 5]], Background → Red], Item[VoxelBedsTable[[9, 4, 6]], Background → Red],
    Item[VoxelBedsTable[[9, 4, 7]], Background → Red], Item[VoxelBedsTable[[9, 4, 8]], Background → Red], VoxelBedsTable[[9, 4, 9]]},
   {VoxelBedsTable[[9, 3, 1]], VoxelBedsTable[[9, 3, 2]], VoxelBedsTable[[9, 3, 3]], Item[VoxelBedsTable[[9, 3, 4]], Background → Red],
    Item[VoxelBedsTable[[9, 3, 5]], Background → Red], Item[VoxelBedsTable[[9, 3, 6]], Background → Red],
    Item[VoxelBedsTable[[9, 3, 7]],
     Background → Red], VoxelBedsTable[[9, 3, 8]], VoxelBedsTable[[9, 3, 9]]},
   {VoxelBedsTable[[9, 2, 1]], VoxelBedsTable[[9, 2, 2]], VoxelBedsTable[[9, 2, 3]], VoxelBedsTable[[9, 2, 4]],
    VoxelBedsTable[[9, 2, 5]],
    VoxelBedsTable[[9, 2, 6]], VoxelBedsTable[[9, 2, 7]], VoxelBedsTable[[9, 2, 8]], VoxelBedsTable[[9, 2, 9]]},
   {VoxelBedsTable[[9, 1, 1]], VoxelBedsTable[[9, 1, 2]], VoxelBedsTable[[9, 1, 3]], VoxelBedsTable[[9, 1, 4]],
    VoxelBedsTable[[9, 1, 5]],
    VoxelBedsTable[[9, 1, 6]], VoxelBedsTable[[9, 1, 7]], VoxelBedsTable[[9, 1, 8]], VoxelBedsTable[[9, 1, 9]]}},
  Background → Blue, Frame → All, Spacings → {0.4, 1}, ItemSize → 8]
(* We use the Grid function to produce the following graphic of slice 9 of the dataset *)
```


23.2 Gy
34.0 Gy
42.7 Gy
46.3 Gy
42.9 Gy
35.3 Gy
28.4 Gy
22.0 Gy
17.0 Gy
31.1 Gy
46.2 Gy
57.8 Gy
66.5 Gy
67.0 Gy
54.6 Gy
40.1 Gy
28.8 Gy
20.9 Gy
36.3 Gy
56.1 Gy
75.6 Gy
95.0 Gy
100.7 Gy
82.1 Gy
56.0 Gy
37.4 Gy
25.5 Gy
37.4 Gy
64.6 Gy
97.5 Gy
127.3 Gy
130.9 Gy
107.7 Gy
75.2 Gy
47.4 Gy
30.2 Gy
35.7 Gy
65.6 Gy
105.9 Gy
136.0 Gy
135.7 Gy
116.8 Gy
87.3 Gy
55.5 Gy
34.0 Gy
33.8 Gy
58.3 Gy
90.1 Gy
118.9 Gy
126.7 Gy
114.4 Gy
86.9 Gy
57.6 Gy
36.0 Gy
30.3 Gy
48.1 Gy
67.9 Gy
88.6 Gy
100.2 Gy
93.8 Gy
74.2 Gy
53.8 Gy
35.2 Gy
25.5 Gy
37.6 Gy
49.1 Gy
58.9 Gy
65.6 Gy
64.9 Gy
57.3 Gy
45.7 Gy
30.6 Gy
20.9 Gy
28.7 Gy
35.8 Gy
40.4 Gy
43.6 Gy
45.1 Gy
42.6 Gy
35.0 Gy
23.9 Gy
17.2 Gy
22.1 Gy
26.6 Gy
29.5 Gy
31.3 Gy
32.1 Gy
30.1 Gy
24.4 Gy
17.7 Gy

Appendix K – Exporting the Results out of *Mathematica*

We prepare the VoxelBeds data produced in *Mathematica* for import into Python/Pydicom.

```
VoxelBedsForDicom = Round[Map[(#/0.00055436292) &, VoxelBeds]];
(* we prepare the VoxelBeds data produced in Mathematica for import into Python/Pydicom. We divide each *)
(* value in the list by the Dose Grid Scaling factor of the original "composite_dose" DICOM file and we *)
(* round the values to integers. *)

VoxelBedsForDicomFlattened = Flatten[VoxelBedsForDicom];
(* we flatten the 11x10x9 VoxelBedsForDicom list into a one-dimensional list for import into Python/Pydicom *)

Export["/Users/WilliamKassing/Documents/Project/VoxelBedsForDicom.txt", VoxelBedsForDicomFlattened, "List"];
(* we save the VoxelBedsForDicomFlattened list as a text file called VoxelBedsForDicom.txt in folder Project; *)
(* this text file can then be imported into Python/Pydicom *)
```

Appendix L – Changing the Original DICOM Dose File to a DICOM BED File

This Python program converts the original DICOM RTDose file from one containing the original dose values to one containing the BED values calculated in *Mathematica*. This allows us to display the BED dose on the treatment plan rather than the physical dose.

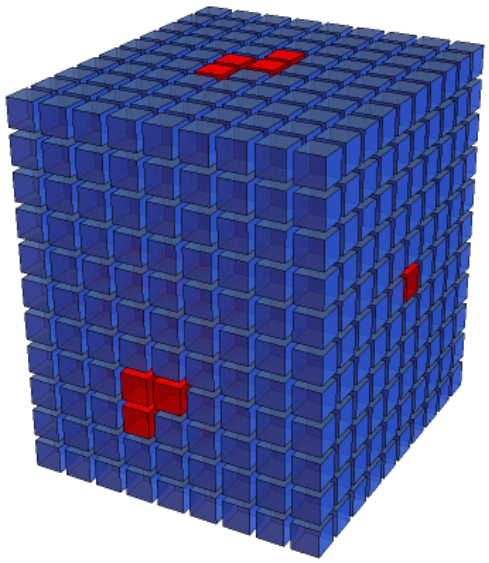
1. Read in the original "composite dose" DICOM RTDose file.
2. Read in the "VoxelBedsForDicom" text file produced in *Mathematica*.
3. Prepare the BED values by scaling to prevent overflow above 65535 ($2^{16} - 1$).
4. Overwrite the DICOM Dose Grid Scaling Factor.
5. Overwrite the dose values in the DICOM pixel_array with BED values.
6. Save the updated RTDose file as "composite beds."

```
#this programs reads in the composite dose dicom RTDose file and replaces the pixel array
#dose values with BED values calculated in Mathematica
import os #imports the operating system module, needed for file and directory functions
import numpy as np #imports the NumPy scientific computing library and makes np shorthand for numpy
os.chdir("/Users/Documents/pydicom-0.9.7") #sets the directory path
import dicom #imports the pydicom package
composite_dose_file=dicom.read_file("composite dose.dcm") #reads in the composite dose dicom file
#the line below reads in the VoxelBedsForDicom text file produced in Mathematica
beds_text_file_import=np.fromfile('VoxelBedsForDicom.txt', dtype=int,count=-1,sep=" ")
#the line below reshapes the one-dimensional text file array into an 11x10x9 array
beds_array=np.array(beds_text_file_import).reshape((11, 10,9))
max_bed=np.amax(beds_array) #finds the maximum array value
#the line below divides the maximum allowed array value (65535) by max_bed
#(decimal point needed for floating point division in Python 2)
array_scalingfactor=65535./max_bed
#the line below scales the array values to prevent overflow of values over 65535
adjusted_beds_array=beds_array*array_scalingfactor
#the for loop below iterates through the array values rounding and converting them to integers
#and replaces the values in composite_dose.pixel_array with these new values
for i in range(0,11):
    for j in range(0,10):
        for k in range(0,9):
            composite_dose_file.pixel_array[i,j,k]=int(round(adjusted_beds_array[i,j,k]))
#the line below creates a new Dose Grid Scaling factor adjusting the original Dose Grid Scaling
#factor with the array_scalingfactor
scalingfactor=float(composite_dose_file[0x3004,0x0e].value)/array_scalingfactor
#the line below rounds scalingfactor to 11 decimal places, converts it to a string
#and sets the Dose Grid Scaling factor to this value
composite_dose_file[0x3004,0x0e].value=str(round(scalingfactor,11))
#the line below overwrites the PixelData attribute of the composite_dose_file changing the original
#dose values to BED values
composite_dose_file.PixelData=composite_dose_file.pixel_array.tostring()
#the line below saves the modified "composite dose" dicom file as "composite beds"
#and thus the original RTDose file has been updated with BED values rather than dose values
composite_dose_file.save_as("composite beds.dcm")
```

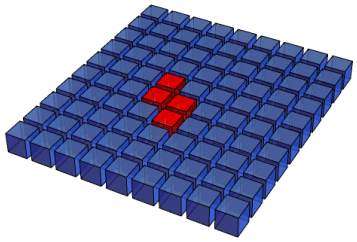
Appendix M – Voxel Graphics

Once we determine which voxels lie within the contours, either imperatively (Appendix E) or functionally (Appendix F), we can display this data in *Mathematica*.

```
RawVoxelData = ReadList["/Users/WilliamKassing/Documents/Project/modified composite dose list.txt", String];
(* imports the list of array values produced by the program in Appendix E that shows which voxels
 lie within a contour *)

VoxelData = Partition[Partition[Flatten[ToExpression[StringReplace[RawVoxelData, {"[" → "{", "]" → "}"}]]], 9], 10];
(* formats this data for Mathematica and partitions the one-dimensional list into an 11x10x9 multi-
 dimensional list *)

Dimensions[VoxelData] (* shows the dataset is 11x10x9 in dimensions *)

{11, 10, 9}

VoxelData[[1]] (* shows the values for slice one with 4 zeros marking voxels within the contour *)

{{11 104, 13 036, 15 039, 16 155, 16 096, 14 880, 12 924, 11 105, 9504},
 {12 725, 15 746, 19 507, 22 608, 22 936, 20 285, 16 512, 13 219, 10 634},
 {14 712, 19 459, 26 361, 31 489, 32 260, 28 418, 21 574, 16 060, 12 302},
 {16 133, 22 873, 31 626, 36 469, 0, 34 438, 26 289, 18 519, 13 847},
 {16 375, 23 829, 32 886, 0, 0, 36 053, 28 048, 19 875, 14 890},
 {15 779, 21 995, 30 261, 0, 36 789, 33 512, 26 239, 19 776, 15 388},
 {14 719, 19 223, 24 773, 29 227, 30 200, 27 325, 22 648, 18 439, 15 125},
 {13 681, 16 868, 20 164, 22 347, 22 946, 21 901, 19 616, 17 001, 14 610},
 {12 736, 14 935, 17 049, 18 155, 18 531, 18 391, 17 349, 15 796, 14 073},
 {11 657, 13 214, 14 552, 15 430, 15 805, 15 836, 15 404, 14 537, 13 381}}

Position[VoxelData[[1]], 0] (* shows the locations of the zero voxels *)

{{4, 5}, {5, 4}, {5, 5}, {6, 4}}

Graphics3D[
 Table[
  With[{p = {i, j, k}},
   {RGBColor[If[VoxelData[[k, j, i]] < 1, 1, .3], If[VoxelData[[k, j, i]] < 1, 0, .5], If[VoxelData[[k, j, i]] < 1, 0, 1]],
    Opacity[.75], Cuboid[p, p + .8]}], {i, 9}, {j, 10}, {k, 11}], Boxed → False]
(* a graphic showing the voxels inside the contours as red and outside the contour as blue,
in the code above if the VoxelData is less than one that voxel is colored red *)
```



```
Graphics3D[
 Table[
  With[{p = {i, j, k}},
   {RGBColor[If[VoxelData[[k, j, i]] < 1, 1, .3], If[VoxelData[[k, j, i]] < 1, 0, .5], If[VoxelData[[k, j, i]] < 1, 0, 1]],
    Opacity[.75], Cuboid[p, p + .8]}], {i, 9}, {j, 10}, {k, 1}], Boxed → False]
(* a graphic showing just one slice of the dataset *)
```



Appendix N – Thirty Fraction Case

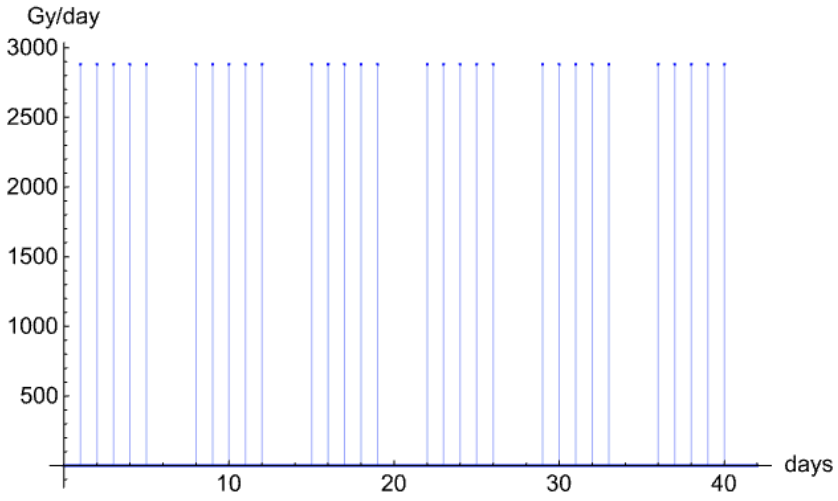
We do an analysis of a conventional thirty 2 Gy fraction treatment, 5 treatments per week with weekend breaks. We use an α/β of 10 Gy. We know the BED is 72 Gy for this case. We initially use a fraction time of 1 minute and a repair half-life of 1 hour. Then we decrease the fraction time to 10 seconds to see the effect on BED.

```
t0 = 1.;                    t30 = 22.;
t1 = 1. + delta;            t31 = 22. + delta;
t2 = 2.;                    t32 = 23.;
t3 = 2. + delta;            t33 = 23. + delta;
t4 = 3.;                    t34 = 24.;
t5 = 3. + delta;            t35 = 24. + delta;
t6 = 4.;                    t36 = 25.;
t7 = 4. + delta;            t37 = 25. + delta;
t8 = 5.;                    t38 = 26.;
t9 = 5. + delta;            t39 = 26. + delta;
t10 = 8.;                   t40 = 29.;
t11 = 8. + delta;           t41 = 29. + delta;
t12 = 9.;                   t42 = 30.;
t13 = 9. + delta;           t43 = 30. + delta;
t14 = 10.;                  t44 = 31.;
t15 = 10. + delta;          t45 = 31. + delta;
t16 = 11.;                  t46 = 32.;
t17 = 11. + delta;          t47 = 32. + delta;
t18 = 12.;                  t48 = 33.;
t19 = 12. + delta;          t49 = 33. + delta;
t20 = 15.;                  t50 = 36.;
t21 = 15. + delta;          t51 = 36. + delta;
t22 = 16.;                  t52 = 37.;
t23 = 16. + delta;          t53 = 37. + delta;
t24 = 17.;                  t54 = 38.;
t25 = 17. + delta;          t55 = 38. + delta;
t26 = 18.;                  t56 = 39.;
t27 = 18. + delta;          t57 = 39. + delta;
t28 = 19.;                  t58 = 40.;
t29 = 19. + delta;          t59 = 40. + delta;
```

```
delta = 1 / 1440; (* the fraction length *)
Mu = Log[2] 24; (* 1 hour half-life repair time in inverse days*)
AlphaBeta = 10; (*alpha/beta = 10 Gy *)
```

```
Fraction1 = (UnitStep[t - t0] - UnitStep[t - t1]);
Fraction2 = (UnitStep[t - t2] - UnitStep[t - t3]);
Fraction3 = (UnitStep[t - t4] - UnitStep[t - t5]);
Fraction4 = (UnitStep[t - t6] - UnitStep[t - t7]);
Fraction5 = (UnitStep[t - t8] - UnitStep[t - t9]);
Fraction6 = (UnitStep[t - t10] - UnitStep[t - t11]);
Fraction7 = (UnitStep[t - t12] - UnitStep[t - t13]);
Fraction8 = (UnitStep[t - t14] - UnitStep[t - t15]);
```

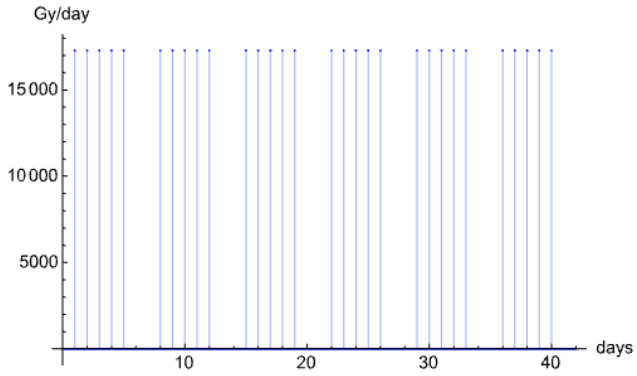
```
Fraction9 = (UnitStep[t - t16] - UnitStep[t - t17]);
Fraction10 = (UnitStep[t - t18] - UnitStep[t - t19]);
Fraction11 = (UnitStep[t - t20] - UnitStep[t - t21]);
Fraction12 = (UnitStep[t - t22] - UnitStep[t - t23]);
Fraction13 = (UnitStep[t - t24] - UnitStep[t - t25]);
Fraction14 = (UnitStep[t - t26] - UnitStep[t - t27]);
Fraction15 = (UnitStep[t - t28] - UnitStep[t - t29]);
Fraction16 = (UnitStep[t - t30] - UnitStep[t - t31]);
Fraction17 = (UnitStep[t - t32] - UnitStep[t - t33]);
Fraction18 = (UnitStep[t - t34] - UnitStep[t - t35]);
Fraction19 = (UnitStep[t - t36] - UnitStep[t - t37]);
Fraction20 = (UnitStep[t - t38] - UnitStep[t - t39]);
Fraction21 = (UnitStep[t - t40] - UnitStep[t - t41]);
Fraction22 = (UnitStep[t - t42] - UnitStep[t - t43]);
Fraction23 = (UnitStep[t - t44] - UnitStep[t - t45]);
Fraction24 = (UnitStep[t - t46] - UnitStep[t - t47]);
Fraction25 = (UnitStep[t - t48] - UnitStep[t - t49]);
Fraction26 = (UnitStep[t - t50] - UnitStep[t - t51]);
Fraction27 = (UnitStep[t - t52] - UnitStep[t - t53]);
Fraction28 = (UnitStep[t - t54] - UnitStep[t - t55]);
Fraction29 = (UnitStep[t - t56] - UnitStep[t - t57]);
Fraction30 = (UnitStep[t - t58] - UnitStep[t - t59]);

DoseRateInputFunction =
  2 / (delta) (Fraction1 + Fraction2 + Fraction3 + Fraction4 + Fraction5 + Fraction6 + Fraction7 + Fraction8 +
     Fraction9 + Fraction10 + Fraction11 + Fraction12 + Fraction13 + Fraction14 + Fraction15 + Fraction16 +
     Fraction17 + Fraction18 + Fraction19 + Fraction20 + Fraction21 + Fraction22 + Fraction23 + Fraction24 +
     Fraction25 + Fraction26 + Fraction27 + Fraction28 + Fraction29 + Fraction30);

Plot[DoseRateInputFunction, {t, 0, 42}, Filling → Bottom, PlotStyle → {Blue}, PlotPoints → 20 000,
 AxesStyle → Directive[Black, 14], AxesLabel → {Style["days", 14], Style["Gy/day", 14]}, ImageSize → 500]
```



```
Psi =
 ParallelMap[ (2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t0, t}]), {t, t0, t1}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t2, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t2}]), {t, t2, t3}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t4, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t4}]), {t, t4, t5}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t6, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t6}]), {t, t6, t7}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t8, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t8}]), {t, t8, t9}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t10, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t10}]), {t, t10, t11}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t12, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t12}]), {t, t12, t13}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t14, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t14}]), {t, t14, t15}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t16, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t16}]), {t, t16, t17}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t18, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t18}]), {t, t18, t19}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t20, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t20}]), {t, t20, t21}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t22, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t22}]), {t, t22, t23}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t24, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t24}]), {t, t24, t25}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t26, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t26}]), {t, t26, t27}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t28, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t28}]), {t, t28, t29}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t30, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t30}]), {t, t30, t31}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t32, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t32}]), {t, t32, t33}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t34, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t34}]), {t, t34, t35}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t36, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t36}]), {t, t36, t37}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t38, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t38}]), {t, t38, t39}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t40, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t40}]), {t, t40, t41}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t42, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t42}]), {t, t42, t43}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t44, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t44}]), {t, t44, t45}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t46, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t46}]), {t, t46, t47}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t48, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t48}]), {t, t48, t49}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t50, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t50}]), {t, t50, t51}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t52, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t52}]), {t, t52, t53}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t54, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t54}]), {t, t54, t55}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t56, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t56}]), {t, t56, t57}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t58, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t58}]), {t, t58, t59}] ) &,
   {DoseRateInputFunction}]

{119.539}

BED = 60 + Map[(#) &, Psi] / AlphaBeta

{71.9539}
```

Here we see the BED is 71.954 Gy, very close to the calculated value of 72 Gy. The calculated value assumes instantaneous fraction times and during our 1 minute treatment there is some repair of sublethal damage lowering the BED. In the next example we shorten the fraction time to 10 seconds and this raises the BED to 71.992 Gy. We will explore this concept further in Appendix O.

```
(* delivering the treatment in 10 seconds, the BED increases because there is less repair of sublethal damage during the treatment *)
Plot[DoseRateInputFunction, {t, 0, 42}, Filling → Bottom, PlotStyle → {Blue}, PlotPoints → 20 000, AxesStyle → Directive[Black, 14],
 AxesLabel → {Style["days", 14], Style["Gy/day", 14]}, ImageSize → 500]
```

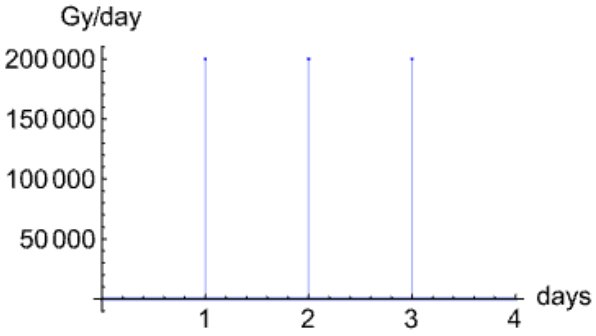



```
BED = 60 + Map[(#) &, Psi] / AlphaBeta

{71.9923}
```

Appendix O – Three Fraction Case

Here we will use a three fraction treatment to explore the assumptions that the traditional BED formula for fractionated therapy is based on the fact that the fraction durations are short and the inter-fraction times are long, compared to the rate of repair. This assures that the repair during the fraction is negligible and that the repair between fractions is complete.

```
(* We begin with a very short fraction time of 0.00001 day and a sufficiently long time
 between fractions (1 day) for complete repair between fractions *)

t0 = 1;(* fraction 1 on day 1 *)
t1 = 1.00001;(* length of fraction 0.00001 day *)
t2 = 2;(* fraction 2 on day 2 *)
t3 = 2.00001;(* length of fraction *)
t4 = 3;(* fraction 3 on day 3 *)
t5 = 3.00001;(* length of fraction *)
Mu = Log[2] 24; (* 1 hour repair half-life in day^-1 *)
AlphaBeta = 10;
(* AlphaBeta ratio = 10 Gy *)

Fraction1 = (UnitStep[t - t0] - UnitStep[t - t1]);
Fraction2 = (UnitStep[t - t2] - UnitStep[t - t3]);
Fraction3 = (UnitStep[t - t4] - UnitStep[t - t5]);

DoseRateInputFunction = (2 / 0.00001) (Fraction1 + Fraction2 + Fraction3);

Plot[DoseRateInputFunction, {t, 0, 4}, Filling → Bottom, PlotPoints → 100 000, PlotStyle → {Blue},
 AxesStyle → Directive[Black, 16], AxesLabel → {Style["days", 16], Style["Gy/day", 16]},
 ImageSize → Medium]
```



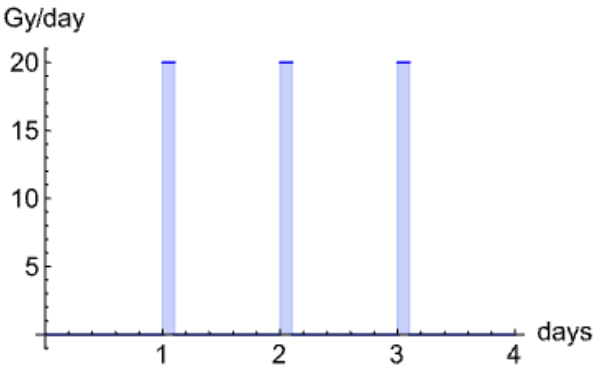

```
Psi =
 ParallelMap[ (2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t0, t}]), {t, t0, t1}] +
     2
      Integrate[(#) * Exp[-Mu t] *
        (Integrate[(#) * Exp[Mu (w)], {w, t2, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t2}]),
       {t, t2, t3}] +
     2
      Integrate[(#) * Exp[-Mu t] *
        (Integrate[(#) * Exp[Mu (w)], {w, t4, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t4}]),
       {t, t4, t5}]) &, {DoseRateInputFunction}]
{11.9993}
```

```
BED = 6 + Map[(#) &, Psi] / AlphaBeta
```

```
{7.19993}
```

```
(* we get a value very close to the calculated value of 7.2 Gy*)


(* we now increase the length of the fraction to 0.1 day (2.4 hour) *)
```

```
t0 = 1;
t1 = 1.1;(* we increase length of fraction to 0.1 day (2.4 hour) *)
t2 = 2;
t3 = 2.1;
t4 = 3;
t5 = 3.1;
Mu = Log[2] 24;
AlphaBeta = 10;

Fraction1 = (UnitStep[t - t0] - UnitStep[t - t1]);
Fraction2 = (UnitStep[t - t2] - UnitStep[t - t3]);
Fraction3 = (UnitStep[t - t4] - UnitStep[t - t5]);

DoseRateInputFunction = (2 / 0.1) (Fraction1 + Fraction2 + Fraction3);

Plot[DoseRateInputFunction, {t, 0, 4}, Filling → Bottom, PlotPoints → 10 000, PlotStyle → {Blue},
 AxesStyle → Directive[Black, 16], AxesLabel → {Style["days", 16], Style["Gy/day", 16]},
 ImageSize → Medium]
```




```
Psi =
 ParallelMap[
  (2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t0, t}]), {t, t0, t1}] +
     2
      Integrate[(#) * Exp[-Mu t] *
        (Integrate[(#) * Exp[Mu (w)], {w, t2, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t2}]),
       {t, t2, t3}] +
     2
      Integrate[(#) * Exp[-Mu t] *
        (Integrate[(#) * Exp[Mu (w)], {w, t4, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t4}]),
       {t, t4, t5}]) &, {DoseRateInputFunction}]
```

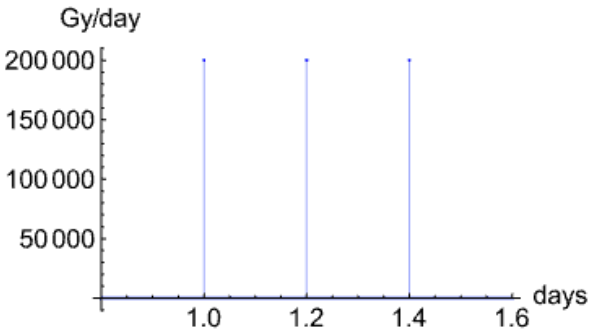
```
{7.39769}
```

```
BED = 6 + Map[(#) &, Psi] / AlphaBeta
```

```
{6.73977}
```

```
(* because of repair of sublethal damage during the fraction, the BED goes down *)
```

```
(* we now shorten the times between fractions to 4.8 hour *)

t0 = 1; (* fraction 1 on day 1 *)
t1 = 1.00001;
t2 = 1.2;(* fraction 2 on day 1.2 *)
t3 = 1.20001;
t4 = 1.4;(* fraction 3 on day 1.4 *)
t5 = 1.40001;
Mu = Log[2] 24;
AlphaBeta = 10;

Fraction1 = (UnitStep[t - t0] - UnitStep[t - t1]);
Fraction2 = (UnitStep[t - t2] - UnitStep[t - t3]);
Fraction3 = (UnitStep[t - t4] - UnitStep[t - t5]);

DoseRateInputFunction = (2 / 0.00001) (Fraction1 + Fraction2 + Fraction3);

Plot[DoseRateInputFunction, {t, 0.8, 1.6}, Filling → Bottom, PlotPoints → 100 000, PlotStyle → {Blue},
 AxesStyle → Directive[Black, 16], AxesLabel → {Style["days", 16], Style["Gy/day", 16]},
 ImageSize → Medium]
```




```
Psi =
 ParallelMap[ (2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t0, t}]), {t, t0, t1}] +

     2
      Integrate[(#) * Exp[-Mu t] *
        (Integrate[(#) * Exp[Mu (w)], {w, t2, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t2}]),
       {t, t2, t3}] +

     2
      Integrate[(#) * Exp[-Mu t] *
        (Integrate[(#) * Exp[Mu (w)], {w, t4, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t4}]),
       {t, t4, t5}]) &, {DoseRateInputFunction}]
```

{12.584}

```
BED = 6 + Map[(#) &, Psi] / AlphaBeta
```

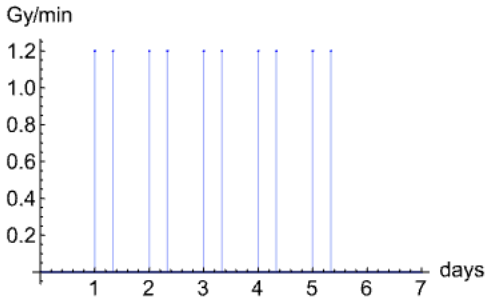
{7.2584}

```
(* because of incomplete repair of sublethal damage between treatments, BED goes up *)

(* This demonstrates how the traditional BED equation for fractionated dose delivery relies
  on the assumptions that the fraction durations are short and the inter-fraction times are long,
compared to the rate of repair assuring that the repair during the fraction is negligible
 and that the repair between fractions is complete. *)
```

Appendix P – Hyperfractionation

Here we explore a hyperfractionation treatment described in Jack Fowler's book "Optimal Overall Treatment Time in Radiation Oncology." This is RTOG HFX (Fu et al. 2000). This treatment gives 81.6 Gy in 68 fractions BID. Here I will do one week of treatment and multiply the results by 6.8.

```
(* Hyperfractionation *)
(* 81.6 Gy at 1.2 Gy per fx BID *)
(* I do 1 week and multiply by 6.8 *)

(* starting treatment on day 1 *)
t0 = 1; (* day 1 treatment 1 *)
t1 = 1 + 1.0 / 1440; (* treatment length of 1 minute *)
t2 = 1 + 480 / 1440; (* day 1 treatment 2 *)
t3 = 1 + 481 / 1440;
t4 = 2; (* day 2 treatment 1 *)
t5 = 2 + 1.0 / 1440;
t6 = 2 + 480 / 1440; (* day 2 treatment 2 *)
t7 = 2 + 481 / 1440;
t8 = 3; (* day 3 treatment 1 *)
t9 = 3 + 1.0 / 1440;
t10 = 3 + 480 / 1440; (* day 3 treatment 2 *)
t11 = 3 + 481 / 1440;
t12 = 4; (* day 4 treatment 1 *)
t13 = 4 + 1.0 / 1440;
t14 = 4 + 480 / 1440; (* day 4 treatment 2 *)
t15 = 4 + 481 / 1440;
t16 = 5; (* day 5 treatment 1 *)
t17 = 5 + 1.0 / 1440;
t18 = 5 + 480 / 1440; (* day 5 treatment 2 *)
t19 = 5 + 481 / 1440;
Mu = Log[2] 24; (* 1 hour half life repair time in day^-1 *)
AlphaBetaEarly = 10; (* alpha/beta = 10 Gy for tumor and early responding tissue *)
AlphaBetaLate = 3; (* alpha/beta = 3 Gy for late responding tissue *)

Fraction1 = UnitStep[t - t0] - UnitStep[t - t1];
Fraction2 = UnitStep[t - t2] - UnitStep[t - t3];
Fraction3 = UnitStep[t - t4] - UnitStep[t - t5];
Fraction4 = UnitStep[t - t6] - UnitStep[t - t7];
Fraction5 = UnitStep[t - t8] - UnitStep[t - t9];
Fraction6 = UnitStep[t - t10] - UnitStep[t - t11];
Fraction7 = UnitStep[t - t12] - UnitStep[t - t13];
Fraction8 = UnitStep[t - t14] - UnitStep[t - t15];
Fraction9 = UnitStep[t - t16] - UnitStep[t - t17];
Fraction10 = UnitStep[t - t18] - UnitStep[t - t19];

DoseRateInputFunction =
  1.2 / (1.0 / 1440) (Fraction1 + Fraction2 + Fraction3 + Fraction4 + Fraction5 + Fraction6 + Fraction7 + Fraction8 +
     Fraction9 + Fraction10);

Plot[DoseRateInputFunction / 1440, {t, 0, 7}, Filling → Bottom, PlotStyle → {Blue}, PlotPoints → 4000,
 AxesStyle → Directive[Black, 16], AxesLabel → {Style["days", 16], Style["Gy/min", 16]}, ImageSize → Medium]
```




```
Psi =
 ParallelMap[ (2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t0, t}]), {t, t0, t1}] +
       2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t2, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t2}]),
        {t, t2, t3}] +
       2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t4, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t4}]),
        {t, t4, t5}] +
       2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t6, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t6}]),
        {t, t6, t7}] +
       2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t8, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t8}]),
        {t, t8, t9}] +
       2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t10, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t10}]),
        {t, t10, t11}] +
       2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t12, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t12}]),
        {t, t12, t13}] +
       2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t14, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t14}]),
        {t, t14, t15}] +
       2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t16, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t16}]),
        {t, t16, t17}] +
       2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t18, t}] + Integrate[(#) * Exp[Mu (t)], {t, t0, t18}]),
        {t, t18, t19}]) &, {DoseRateInputFunction}]
{14.4011}

BEDperWeek = 12 + Map[(#) &, Psi] / AlphaBetaEarly
{13.4401}

TotalBED = 6.8 * BEDperWeek
{91.3928}

AcuteBED =
 TotalBED -
  Log[2] (45 - 7) / (0.35 * 2.5) (* The overall treatment time is 45 days with an early tissue repopulation
     kickoff time of 7 days, alpha = 0.35 Gy^-1 and Tpot = 2.5 days *)
{61.2904}

(* Fowler's value = 61.3 Gy *)

TumorBED =
 TotalBED -
  Log[2] (45 - 21) / (0.35 * 3) (* The overall treatment time is 45 days with a tumor repopulation kickoff time of 21 days,
    alpha = 0.35 Gy^-1 and the tumor Tpot = 3 days *)
{75.5494}

(* logs of cell kill =(alpha x BED)/(2.303) *)
LogsCellKill = 0.35 * TumorBED / 2.303
{11.4817}

(* Fowler's value = 11.48 *)
```

```
LateBEDperWeek = 12 + Map[(#) &, Psi] / AlphaBetaLate
```

```
{16.8004}
```

```
LateBED = 6.8 * LateBEDperWeek
```

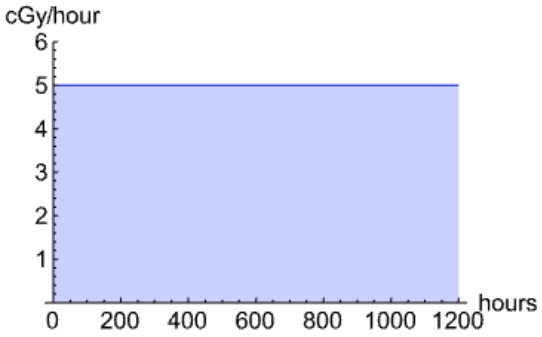
```
{114.243}
```

```
(* Fowler's value = 114 Gy *)
```

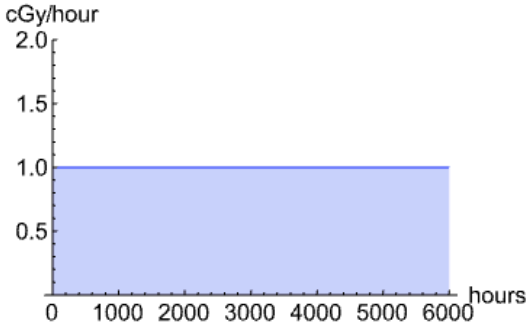
Appendix Q – LDR and HDR

Now we will look at LDR and HDR treatments. It is stated in Joiner and van der Kogel's Basic Clinical Radiobiology (Joiner, 2018) that for full repair of sublethal damage to occur during treatment the dose-rate must be less than about 5 cGy/hour. We will start by examining this.

```
cGyperhr = 5; (* 5 cGy/hour *)

doserate = cGyperhr (-UnitStep[-1200 + t] + UnitStep[t])

5 (-UnitStep[-1200 + t] + UnitStep[t])

Plot[doserate, {t, 0, 1200}, Filling → Bottom, AxesStyle → Directive[Black, 16], PlotStyle → {Blue, Thickness[.003]},
 PlotRange → {0, 6}, AxesLabel → {Style["hours", 16], Style["cGy/hour", 16]}, ImageSize → Medium]
```




```
totaldose = NIntegrate[doserate, {t, 0, 1200}] (* we give a total dose of 6000 cGy *)

6000.

t1 = 1200; (* time = 1200 hour *)
Mu = Log[2] / 1.0 ; (* 1 hour repair halftime *)
alphabeta = 1000; (* alpha/beta = 1000 cGy *)

psi =
 ParallelMap[ (2 Integrate[ (#) * Exp[-Mu t] * (Integrate[ (#) * Exp[Mu (w)], {w, 0, t}]), {t, 0, t1}]) &, {doserate}]

{86 457.6}

bed = (totaldose + (Map[ ( #) &, {psi}] ) / (1000 ))

{{6086.46}}

(* this value depends on the repair halftime; with repair halftime of 2 hour the value goes up to 6173 cGy;
with repair halftime of 0.5 hour the value goes down to 6043 cGy *)


(* now going to 1 cGy/hr *)

cGyperhr = 1; (* 1 cGy/hour *)

doserate = cGyperhr (-UnitStep[-6000 + t] + UnitStep[t])

-UnitStep[-6000 + t] + UnitStep[t]

Plot[doserate, {t, 0, 6000}, Filling → Bottom, AxesStyle → Directive[Black, 16], PlotStyle → {Blue, Thickness[.003]},
 PlotRange → {0, 2}, AxesLabel → {Style["hours", 16], Style["cGy/hour", 16]}, ImageSize → Medium]
```




```
totaldose = NIntegrate[doserate, {t, 0, 6000}] (* total dose of 6000 cGy *)
```

```
6000.
```

```
t1 = 6000; (* time = 6000 hour *)
Mu = Log[2] / 1.0; (* repair halftime of 1 hour *)
alphabeta = 1000; (* alpha/beta = 1000 cGy *)
```

```
psi =
 ParallelMap[ (2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, 0, t}]), {t, 0, t1}]) &, {doserate}]
```

```
{17308.2}
```

```
bed = (totaldose + (Map[( #) &, {psi}] ) / (1000 ))
```

```
{{6017.31}}
```

```
(* so at 1 cGy/hour we get very close to full repair of sublethal damage and the BED is very close
 to the physical dose *)
```

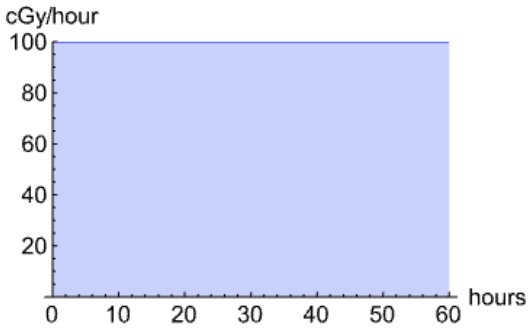
```
(* 100 cGy/hour *)
```

```
cGyperhr = 100; (* 100 cGy/hour *)
```

```
doserate = cGyperhr (-UnitStep[-60 + t] + UnitStep[t])
```

```
100 (-UnitStep[-60 + t] + UnitStep[t])
```

```
Plot[doserate, {t, 0, 60}, Filling → Bottom, AxesStyle → Directive[Black, 16], PlotStyle → {Blue, Thickness[.003]},
 PlotRange → {0, 100}, AxesLabel → {Style["hours", 16], Style["cGy/hour", 16]}, ImageSize → Medium]
```




```
totaldose = NIntegrate[doserate, {t, 0, 60}] (* total dose of 6000 cGy *)
```

```
6000.
```

```
t1 = 60; (* time = 60 hour *)
Mu = Log[2] / 1.0 ; (* 1 hour repair halftime *)
alphabeta = 1000; (* alpha/beta = 1000 cGy *)
```

```
psi =
 ParallelMap[ (2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, 0, t}]), {t, 0, t1}]) &, {doserate}]
```

$\{1.68961 \times 10^6\}$

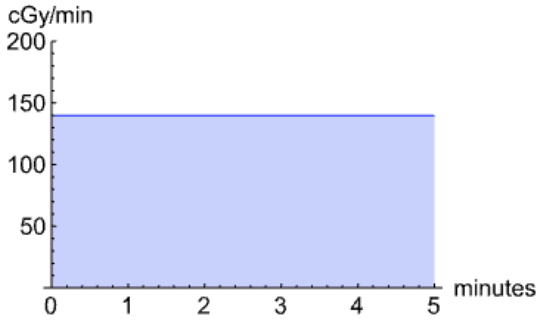
```
bed = (6000 + (Map[( #) &, {psi}] ) / (1000 ))
```

{{7689.61}}

```
(* there is a rule of thumb that 100 cGy per hour is equivalent to traditional fractionation which
 in 2 Gy fractions would have a BED of 7200 cGy compared to the 7690 cGy we see here. If we lower
 the repair halftime to 0.75 hour we get a BED of 7275 cGy or if we keep the repair halftime at 1
 hour but lower the doserate to 75 cGy/hour we get a BED of 7275 cGy as well  *)


(* Now we will look at an HDR treatment, say a treatment of 7 Gy over 5 minutes or 1.4 Gy/min or 140 cGy/min *)

cGypermin = 140;  (* 140 cGy/min *)

doserate = cGypermin (-UnitStep[-5 + t] + UnitStep[t])
```

140 (-UnitStep[-5 + t] + UnitStep[t])

```
Plot[doserate, {t, 0, 5}, Filling → Bottom, AxesStyle → Directive[Black, 16], PlotStyle → {Blue, Thickness[.003]},
 PlotRange → {0, 200}, AxesLabel → {Style["minutes", 16], Style["cGy/min", 16]}, ImageSize → Medium]
```




```
totaldose = NIntegrate[doserate, {t, 0, 5}] (* total dose of 700 cGy *)
```

700.

```
t1 = 5; (* time = 5 minutes *)
Mu = Log[2] / 60.0 ;(* 60 minute repair halftime *)
alphabeta = 1000;
(* alpha/beta = 1000 cGy *)

psi =
 ParallelMap[ (2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, 0, t}]), {t, 0, t1}]) &, {doserate}]
```

{480 700.}

```
bed = (totaldose + (Map[( #) &, {psi}] ) / (1000 ))
```

{{1180.7}}

```
(* we see this HDR treatment of 700 cGy has a BED of 1181 cGy *)
```

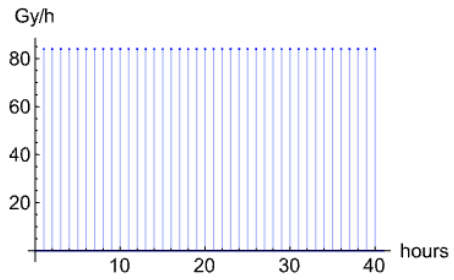
Appendix R – PDR

Here we explore a PDR treatment, where the goal is to use HDR to mimic an LDR treatment in terms of BED. We will deliver 0.5 Gy every hour for 40 hours for a total dose of 20 Gy.

```
(* Here we demonstate a PDR treatment giving 50 cGy each hour for 40 hours at a dose-rate of 140 cGy/min (84 Gy/hr) *)

pulse = 0.5/84 (* 0.5 Gy/84 Gy/hr gives this pulse length in hours *)

0.00595238

t0 = 1.0; (* first fraction at hour 1 *)
t1 = 1.0 + pulse; (* first fraction duration is 0.00595 hour long delivering 50 cGy each hour *)
t2 = 2;
t3 = 2 + pulse;
t4 = 3;
t5 = 3 + pulse;
t6 = 4;
t7 = 4 + pulse;
t8 = 5;
t9 = 5 + pulse;
t10 = 6;
t11 = 6 + pulse;
t12 = 7;
t13 = 7 + pulse;
t14 = 8;
t15 = 8 + pulse;
t16 = 9;
t17 = 9 + pulse;
t18 = 10;
t19 = 10 + pulse;
t20 = 11;
t21 = 11 + pulse;
t22 = 12;
t23 = 12 + pulse;
t24 = 13;
t25 = 13 + pulse;
t26 = 14;
t27 = 14 + pulse;
t28 = 15;
t29 = 15 + pulse;
t30 = 16;
t31 = 16 + pulse;
t32 = 17;
t33 = 17 + pulse;
t34 = 18;
t35 = 18 + pulse;
t36 = 19;
t37 = 19 + pulse;
t38 = 20;
t39 = 20 + pulse;
t40 = 21;
t41 = 21 + pulse;
t42 = 22;
t43 = 22 + pulse;
t44 = 23;
t45 = 23 + pulse;
t46 = 24;
t47 = 24 + pulse;
t48 = 25;
t49 = 25 + pulse;
t50 = 26;
```

```
t51 = 26 + pulse;
t52 = 27;
t53 = 27 + pulse;
t54 = 28;
t55 = 28 + pulse;
t56 = 29;
t57 = 29 + pulse;
t58 = 30;
t59 = 30 + pulse;
t60 = 31;
t61 = 31 + pulse;
t62 = 32;
t63 = 32 + pulse;
t64 = 33;
t65 = 33 + pulse;
t66 = 34;
t67 = 34 + pulse;
t68 = 35;
t69 = 35 + pulse;
t70 = 36;
t71 = 36 + pulse;
t72 = 37;
t73 = 37 + pulse;
t74 = 38;
t75 = 38 + pulse;
t76 = 39;
t77 = 39 + pulse;
t78 = 40;
t79 = 40 + pulse;
Mu = Log[2] / 1.; (* repair half-time of 1 hour *)
AlphaBeta = 10;

Fraction1 = (UnitStep[t - t0] - UnitStep[t - t1]);
Fraction2 = (UnitStep[t - t2] - UnitStep[t - t3]);
Fraction3 = (UnitStep[t - t4] - UnitStep[t - t5]);
Fraction4 = (UnitStep[t - t6] - UnitStep[t - t7]);
Fraction5 = (UnitStep[t - t8] - UnitStep[t - t9]);
Fraction6 = (UnitStep[t - t10] - UnitStep[t - t11]);
Fraction7 = (UnitStep[t - t12] - UnitStep[t - t13]);
Fraction8 = (UnitStep[t - t14] - UnitStep[t - t15]);
Fraction9 = (UnitStep[t - t16] - UnitStep[t - t17]);
Fraction10 = (UnitStep[t - t18] - UnitStep[t - t19]);
Fraction11 = (UnitStep[t - t20] - UnitStep[t - t21]);
Fraction12 = (UnitStep[t - t22] - UnitStep[t - t23]);
Fraction13 = (UnitStep[t - t24] - UnitStep[t - t25]);
Fraction14 = (UnitStep[t - t26] - UnitStep[t - t27]);
Fraction15 = (UnitStep[t - t28] - UnitStep[t - t29]);
Fraction16 = (UnitStep[t - t30] - UnitStep[t - t31]);
Fraction17 = (UnitStep[t - t32] - UnitStep[t - t33]);
Fraction18 = (UnitStep[t - t34] - UnitStep[t - t35]);
Fraction19 = (UnitStep[t - t36] - UnitStep[t - t37]);
Fraction20 = (UnitStep[t - t38] - UnitStep[t - t39]);
Fraction21 = (UnitStep[t - t40] - UnitStep[t - t41]);
Fraction22 = (UnitStep[t - t42] - UnitStep[t - t43]);
Fraction23 = (UnitStep[t - t44] - UnitStep[t - t45]);
Fraction24 = (UnitStep[t - t46] - UnitStep[t - t47]);
Fraction25 = (UnitStep[t - t48] - UnitStep[t - t49]);
Fraction26 = (UnitStep[t - t50] - UnitStep[t - t51]);
Fraction27 = (UnitStep[t - t52] - UnitStep[t - t53]);
Fraction28 = (UnitStep[t - t54] - UnitStep[t - t55]);
Fraction29 = (UnitStep[t - t56] - UnitStep[t - t57]);
Fraction30 = (UnitStep[t - t58] - UnitStep[t - t59]);
Fraction31 = (UnitStep[t - t60] - UnitStep[t - t61]);
```

```
Fraction32 = (UnitStep[t - t62] - UnitStep[t - t63]);
Fraction33 = (UnitStep[t - t64] - UnitStep[t - t65]);
Fraction34 = (UnitStep[t - t66] - UnitStep[t - t67]);
Fraction35 = (UnitStep[t - t68] - UnitStep[t - t69]);
Fraction36 = (UnitStep[t - t70] - UnitStep[t - t71]);
Fraction37 = (UnitStep[t - t72] - UnitStep[t - t73]);
Fraction38 = (UnitStep[t - t74] - UnitStep[t - t75]);
Fraction39 = (UnitStep[t - t76] - UnitStep[t - t77]);
Fraction40 = (UnitStep[t - t78] - UnitStep[t - t79]);

DoseRateInputFunction =
  84 (Fraction1 + Fraction2 + Fraction3 + Fraction4 + Fraction5 + Fraction6 + Fraction7 + Fraction8 + Fraction9 + Fraction10 + Fraction11 +
     Fraction12 + Fraction13 + Fraction14 + Fraction15 + Fraction16 + Fraction17 + Fraction18 + Fraction19 + Fraction20 + Fraction21 + Fraction22 +
     Fraction23 + Fraction24 + Fraction25 + Fraction26 + Fraction27 + Fraction28 + Fraction29 + Fraction30 + Fraction31 + Fraction32 + Fraction33 +
     Fraction34 + Fraction35 + Fraction36 + Fraction37 + Fraction38 + Fraction39 + Fraction40);

totaldose = Integrate[DoseRateInputFunction, {t, 0, 100}]

20.

Plot[DoseRateInputFunction, {t, 0, 41}, Filling → Bottom, PlotPoints → 10 000, AxesStyle → Directive[Black, 16], PlotStyle → {Blue},
 AxesStyle → Directive[Black, 16], AxesLabel → {Style["hours", 16], Style["Gy/h", 16]}, ImageSize → Medium]
```




```
Psi =
 ParallelMap[ (2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t0, t}]), {t, t0, t1}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t2, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t2}]), {t, t2, t3}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t4, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t4}]), {t, t4, t5}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t6, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t6}]), {t, t6, t7}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t8, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t8}]), {t, t8, t9}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t10, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t10}]), {t, t10, t11}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t12, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t12}]), {t, t12, t13}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t14, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t14}]), {t, t14, t15}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t16, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t16}]), {t, t16, t17}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t18, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t18}]), {t, t18, t19}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t20, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t20}]), {t, t20, t21}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t22, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t22}]), {t, t22, t23}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t24, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t24}]), {t, t24, t25}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t26, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t26}]), {t, t26, t27}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t28, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t28}]), {t, t28, t29}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t30, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t30}]), {t, t30, t31}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t32, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t32}]), {t, t32, t33}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t34, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t34}]), {t, t34, t35}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t36, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t36}]), {t, t36, t37}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t38, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t38}]), {t, t38, t39}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t40, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t40}]), {t, t40, t41}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t42, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t42}]), {t, t42, t43}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t44, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t44}]), {t, t44, t45}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t46, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t46}]), {t, t46, t47}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t48, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t48}]), {t, t48, t49}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t50, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t50}]), {t, t50, t51}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t52, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t52}]), {t, t52, t53}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t54, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t54}]), {t, t54, t55}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t56, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t56}]), {t, t56, t57}] +
```

```
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t58, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t58}]), {t, t58, t59}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t60, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t60}]), {t, t60, t61}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t62, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t62}]), {t, t62, t63}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t64, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t64}]), {t, t64, t65}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t66, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t66}]), {t, t66, t67}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t68, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t68}]), {t, t68, t69}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t70, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t70}]), {t, t70, t71}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t72, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t72}]), {t, t72, t73}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t74, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t74}]), {t, t74, t75}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t76, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t76}]), {t, t76, t77}] +
      2 Integrate[(#) * Exp[-Mu t] * (Integrate[(#) * Exp[Mu (w)], {w, t78, t}] + Integrate[(#) * Exp[Mu (t)], {t, 0, t78}]),
       {t, t78, t79}]) &, {DoseRateInputFunction}]
```

```
{28.9863}
```

```
BED = 20 + Map[# &, Psi] / AlphaBeta
```

```
{22.8986}
```

```
(* This BED of 22.9 Gy is very similar to that of tradiational fractionation of 20 Gy in 2 Gy fractions where the BED
 would be 24 Gy with an alpha/beta of 10 Gy *)
```

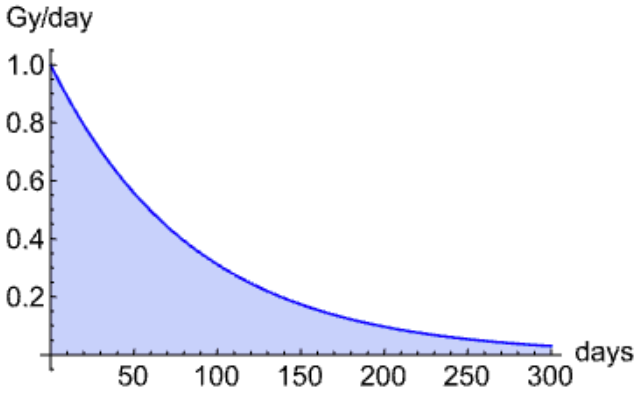
Appendix S – I-125 and Pd-103 Treatment

We will now explore I-125 and Pd-103 dosimetry. Here we will see that the form of the convolution equation we use as explained in Section 4.2 makes a slight difference in the results.

```
HalfLifeIodine = 59.4; (* half-life of Iodine-125 in days *)

UnitDoseRateIodine = 1; (* set the dose-rate initially to 1 Gy/day *)

(* form and plot a function for exponential decay of I-125 starting with unit dose-rate *)
UnitVoxelDoseRateIodine = UnitDoseRateIodine * Exp[-(Log[2] / HalfLifeIodine) t]
```

$$e^{-0.0116691\,t}$$

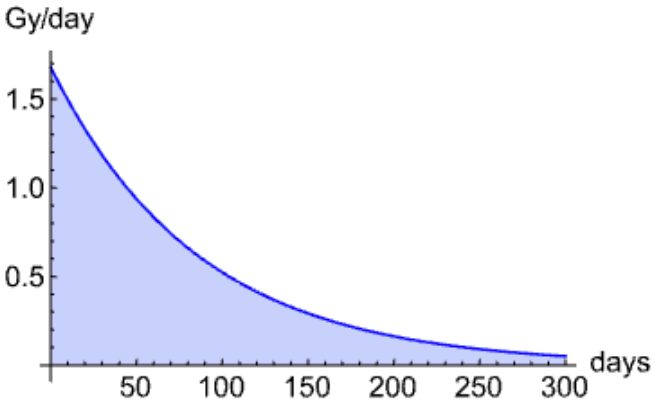
```
Plot[UnitVoxelDoseRateIodine, {t, 0, 300}, Filling → Bottom, PlotStyle → {Blue},
 AxesStyle → Directive[Black, 16], AxesLabel → {Style["days", 16], Style["Gy/day", 16]},
 ImageSize → Medium]
```




```
(* calculate the total dose from decay of the function above *)
TotalDoseIodineFromUnitDoseRate = Integrate[UnitVoxelDoseRateIodine, {t, 0, 10 000}]

85.6961

(* calculate the initial dose-rate to deliver a total dose of 144 Gy *)
InitialDoseRateIodine = 144 / TotalDoseIodineFromUnitDoseRate

1.68036

(* confirm that this is correct *)
TotalVoxelDoseIodine = NIntegrate[InitialDoseRateIodine Exp[-(Log[2] / HalfLifeIodine) t],
  {t, 0, 10 000}]

144.

(* form and plot a function of the voxel dose rate given the above initial dose rate -
 in terms of t *)
VoxelDoseRateIodineT = InitialDoseRateIodine * Exp[-(Log[2] / HalfLifeIodine) t]
```

$$1.68036\, e^{-0.0116691\,t}$$

```
Plot[VoxelDoseRateIodineT, {t, 0, 300}, Filling → Bottom, PlotStyle → {Blue},
 AxesStyle → Directive[Black, 16], AxesLabel → {Style["days", 16], Style["Gy/day", 16]},
 ImageSize → Medium]
```




```
(* write the above equation in terms of w (rather than t) for use in the PsiW
 equation below *)
VoxelDoseRateIodineW = InitialDoseRateIodine * Exp[-(Log[2] / HalfLifeIodine) w]
```

$$1.68036\, e^{-0.0116691\, w}$$

```
t0 = 0; (* we will integrate between 0 and 5,000 days *)
t1 = 5000;
AlphaBeta = 3; (* alpha/beta = 3 Gy *)
Mu = Log[2] / (1 / 24) (* 1 hour half-life *);

(* perform the convolution with just t rather than both t and w as explained in
 section 4.2 and call this VoxelIodinePsiT *)
VoxelIodinePsiT =
 2 Integrate[VoxelDoseRateIodineT * Exp[-Mu t] *
    (Integrate[VoxelDoseRateIodineT * Exp[Mu (w)], {w, 0, t}]), {t, 0, t1}]
```

14.5251

```
(* calculate the BED and call it BEDIodineT *)
BEDIodineT = TotalVoxelDoseIodine + VoxelIodinePsi / AlphaBeta
```

148.842

```
(* perform the full convolution with both w and t and call this VoxelIodinePsiW *)
VoxelIodinePsiW =
 2 Integrate[(VoxelDoseRateIodineT) * Exp[-Mu t] *
    (Integrate[(VoxelDoseRateIodineW) * Exp[Mu (w)], {w, 0, t}]), {t, 0, t1}]
```

14.5353

```
(* calculate the BED with the second complete form of the Psi equation VoxelIodinePsiW
 integrating over both t and w and call this BEDIodineW *)
BEDIodineW = TotalVoxelDoseIodine + VoxelIodinePsiW / AlphaBeta
```

148.845

```
(* we see the results for BEDIodineW are slightly higher than for BEDIodineT *)
```

```
(* we can check our results with an analytic equation using a G term found in many
 references such as Gustafsson (2013)*)

G = ((2 Lambda ^ 2) / (1 - Exp[-Lambda Time]) ^ 2) * (1 / (Mu - Lambda)) *
   (((1 - Exp[-2 Lambda Time]) / (2 Lambda)) - ((1 - Exp[-(Mu + Lambda) Time]) / (Mu + Lambda)))
```

$$\frac{2\,\text{Lambda}^2\left(\frac{1-e^{-2\,\text{Lambda}\,\text{Time}}}{2\,\text{Lambda}}-\frac{1-e^{(-\text{Lambda}-\text{Mu})\,\text{Time}}}{\text{Lambda}+\text{Mu}}\right)}{\left(1-e^{-\text{Lambda}\,\text{Time}}\right)^2\,(-\text{Lambda}+\text{Mu})}$$

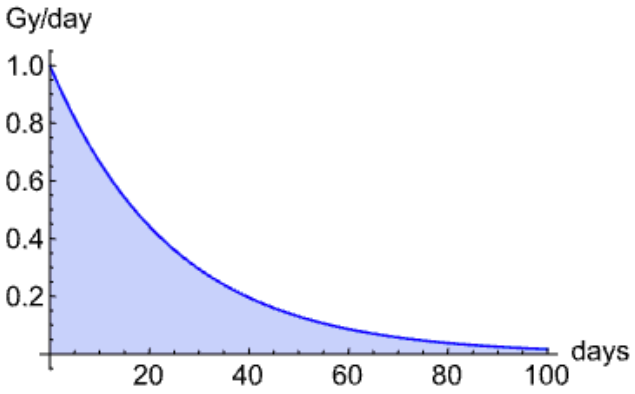
```
(* setting all exponential terms to zero for infinite time *)
G = Simplify[((2 Lambda ^ 2) / (1 - 0) ^ 2) * (1 / (Mu - Lambda)) *
    (((1 - 0) / (2 Lambda)) - ((1 - 0) / (Mu + Lambda)))]
```

$$\frac{\text{Lambda}}{\text{Lambda}+\text{Mu}}$$

```
(* Then the BED can be calculated according to this formula *)
BEDIodine = VoxelDoseIodine (1 + (VoxelDoseIodine G) / (AlphaBeta))

t0 = 0;
Time = t1 = 5000;(* we will use time 5000 days as before *)
AlphaBeta = 3; (* alpha/beta = 3 Gy *)
Mu = Log[2] / (1 / 24); (* 1 hour half-life *)
Lambda = (Log[2] / HalfLifeIodine );

G = ((2 Lambda ^ 2) / (1 - Exp[-Lambda Time]) ^ 2) * (1 / (Mu - Lambda)) *
    (((1 - Exp[-2 Lambda Time]) / (2 Lambda)) - ((1 - Exp[-(Mu + Lambda) Time]) / (Mu + Lambda)));

BEDIodine = VoxelDoseIodine (1 + (VoxelDoseIodine G) / (AlphaBeta))

148.845

(* We see the results with the complete convolution BEDIodineW is the correct one
 however the other simpler convolution using VoxelIodinePsiT is faster and quite
 accurate as well and is the form primarily used in this book *)



(* Now we'll compare the BED for I-125 to that of an equal dose of Palladium-103 *)

HalfLifePalladium = 17.0 (* half-life of Palladium-103 in days *);

UnitDoseRatePalladium = 1 (* set the dose-rate initially to 1 Gy/day *);

(* form and plot a function for exponential decay of Pd-103 starting with unit dose-rate *)
UnitVoxelDoseRatePalladiumT = UnitDoseRatePalladium Exp[-(Log[2] / HalfLifePalladium ) t]
```

$$e^{-0.0407734\,t}$$

```
Plot[UnitVoxelDoseRatePalladiumT, {t, 0, 100}, Filling → Bottom, PlotStyle → {Blue},
 AxesStyle → Directive[Black, 16], AxesLabel → {Style["days", 16], Style["Gy/day", 16]},
 ImageSize → Medium]
```




```
(* calculate the total dose from decay of the function above *)
TotalDosePalladiumFromUnitDoseRate =
 NIntegrate[UnitDoseRatePalladium Exp[-(Log[2] / HalfLifePalladium) t], {t, 0, 10 000}]
```

```
24.5258
```

```
(* calculate the initial dose-rate to deliver a total dose of 144 Gy *)
InitialDoseRatePalladium = 144 / TotalDosePalladiumFromUnitDoseRate
```

```
5.87136
```

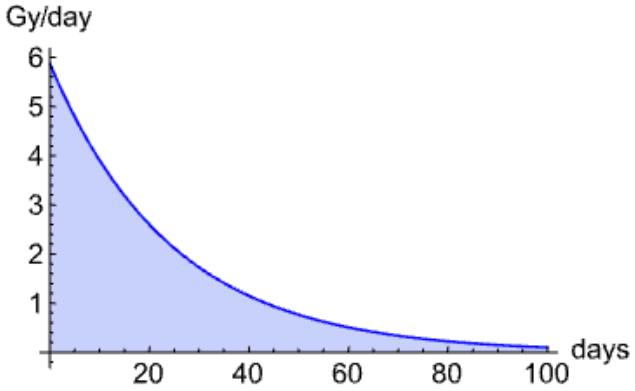
```
(* confirm that this is correct *)
TotalVoxelDosePalladium =
 NIntegrate[InitialDoseRatePalladium Exp[-(Log[2] / HalfLifePalladium) t], {t, 0, 10 000}]
```

```
144.
```

```
(* form and plot a function of the voxel dose rate given the above initial dose rate *)
VoxelDoseRatePalladiumT = InitialDoseRatePalladium * Exp[-(Log[2] / HalfLifePalladium) t]
```

$$5.87136\, e^{-0.0407734\, t}$$

```
Plot[VoxelDoseRatePalladiumT, {t, 0, 100}, Filling → Bottom, PlotStyle → {Blue},
 AxesStyle → Directive[Black, 16], AxesLabel → {Style["days", 16], Style["Gy/day", 16]},
 ImageSize → Medium]
```




```
(* write the above equation in terms of w (rather than t) for use in the PsiW
 equation below *)
VoxelDoseRatePalladiumW = InitialDoseRatePalladium * Exp[-(Log[2] / HalfLifePalladium) w]
```

$$5.87136\, e^{-0.0407734\, w}$$

```
t0 = 0; (* we will integrate between 0 and 5,000 days *)
t1 = 5000;
AlphaBeta = 3; (* alpha/beta = 3 Gy *)
Mu = Log[2] / (1 / 24) (* 1 hour half-life *);

(* perform the convolution with just t rather than both t and w as explained in
 section 4.2 and call this VoxelPalladiumPsiT *)
VoxelPalladiumPsiT =
 2 Integrate[VoxelDoseRatePalladiumT * Exp[-Mu t] *
    (Integrate[VoxelDoseRatePalladiumT * Exp[Mu (w)], {w, 0, t}]), {t, 0, t1}]
```

50.5756

```
(* perform the full convolution with both w and t and call this VoxelPalladiumPsiW *)
VoxelPalladiumPsiW =
 (2 Integrate[(VoxelDoseRatePalladiumT) * Exp[-Mu t] *
     (Integrate[(VoxelDoseRatePalladiumW) * Exp[Mu (w)], {w, 0, t}]), {t, 0, t1}])
```

50.6993

```
(* calculate the BED using VoxelPalladiumPsiT *)
BEDPalladiumT = TotalVoxelDosePalladium + VoxelPalladiumPsiT / AlphaBeta
```

160.859

```
(* calculate the BED using VoxelPalladiumPsiW which again is the correct one *)
BEDPalladiumW = TotalVoxelDosePalladium + VoxelPalladiumPsiW / AlphaBeta
```

160.9

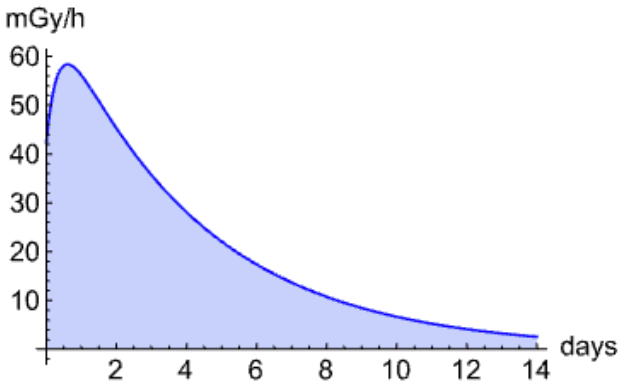
```
(* we see the BED for Pd-103 is greater than that of I-125 because of its shorter half-
 life and thus greater dose rate. The RBE of Pd-103 is also higher than the RBE of I-
 125 also accounting for the different doses used in treatment *)
```

Appendix T – Radionuclide Therapy

We will explore radionuclide therapy based on a case involving Lu-177 octreotate described in a paper by Gustafsson (2013 II).

```
(* this is a biexponential function and plot similar to Figure 8 in Gustafsson (2013 II) *)
LutetiumDoseRate = 73.3 Exp[-0.24 t] - 31 Exp[-3 t]
Plot[LutetiumDoseRate, {t, 0, 14}, Filling → Bottom, PlotStyle → {Blue},
 AxesStyle → Directive[Black, 16], AxesLabel → {Style["days", 16], Style["mGy/h", 16]},
 ImageSize → Medium]
```

$$-31\, e^{-3\,t} + 73.3\, e^{-0.24\,t}$$

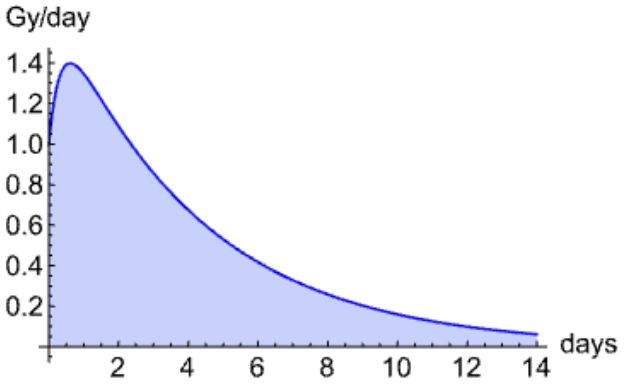



```
(* shows the dose-rates at three data points in the Gustafsson paper, matches pretty well *)
LutetiumDoseRate /. t → 1
LutetiumDoseRate /. t → 4
LutetiumDoseRate /. t → 7
```

```
56.1164

28.0659

13.6612
```

```
(* we plot this as Gy/day vs day *)
LutetiumDoseRate = 1.76 Exp[-0.24 t] - 0.76 Exp[-3 t] (*this is close Gy/day vs day *)
Plot[LutetiumDoseRate, {t, 0, 14}, Filling → Bottom, PlotStyle → {Blue},
 AxesStyle → Directive[Black, 16], AxesLabel → {Style["days", 16], Style["Gy/day", 16]},
 ImageSize → Medium]
```

$$-0.76\, e^{-3\,t} + 1.76\, e^{-0.24\,t}$$



```
(* writing the function with respect to w, rather than t, for use in PsiW below *)
LutetiumDoseRateW = 1.76 Exp[-0.24 w] - 0.76 Exp[-3 w]
```

$$-0.76\, e^{-3 w} + 1.76\, e^{-0.24 w}$$

```
(* get the total dose from complete decay *)
LutetiumTotalDose = NIntegrate[LutetiumDoseRate, {t, 0, 300}]
```

```
7.08
```

```
t1 = 300; (* integrate to 300 days *)
Mu = Log[2] / (2.8 / 24); (* repair rate constant 2.8 hours in days *)
AlphaBeta = 2.6;
```

```
(* perform the convolution with just t rather than both t and w as explained in
 section 4.2 and call this VoxelLutetiumPsiT *)
VoxelLutetiumPsiT =
  2 Integrate[LutetiumDoseRate * Exp[-Mu t] * (Integrate[LutetiumDoseRate * Exp[Mu (w)], {w, 0, t}]),
    {t, 0, t1}]
```

```
1.84625
```

```
(* perform the full convolution with both w and t and call this VoxelLutetiumPsiW *)
VoxelLutetiumPsiW =
 2 Integrate[LutetiumDoseRate * Exp[-Mu t] * (Integrate[LutetiumDoseRateW * Exp[Mu (w)], {w, 0, t}]),
    {t, 0, t1}]
```

```
1.88364
```

```
(* the BED using VoxelLutetiumPsiT *)
BEDLutetiumT = LutetiumTotalDose + VoxelLutetiumPsiT / AlphaBeta
```

```
7.7901
```

```
(* the BED using VoxelLutetiumPsiW *)
BEDLutetiumW = LutetiumTotalDose + VoxelLutetiumPsiW / AlphaBeta
```

```
7.80448
```

```
(* again there are slightly different results and we can validate which is correct
   by using an analytical calculation for G from Gustafsson (2013) Eq. 24 for a1*Exp[-λ1*t]+
 a2*Exp[-λ2*t] *)
a1 = -.76;
a2 = 1.76;
λ1 = 3;
λ2 = .24;
Mu = Log[2] / (2.8 / 24);
```

```
G12 =
 (((a1 ^ 2) / (λ1 (Mu + λ1))) + ((2 a1 a2) / ((λ1 + λ2) (Mu + λ1))) + ((2 a2 a1) / ((λ2 + λ1) (Mu + λ2))) +
    ((a2 ^ 2) / (λ2 (Mu + λ2)))) / ((a1 / λ1) + (a2 / λ2)) ^ 2
```

```
0.0375778
```

```
(* the analytical solution for BED using the G12 factor above *)
BEDLutetiumAnalytic = LutetiumTotalDose (1 + ((LutetiumTotalDose G12) / AlphaBeta))
```

```
7.80448
```

```
(* This compares favorably to a value of 7.79 Gy given in the Gustafsson paper *)

(* so we see again the correct results in this case of a continuous changing dose
 rate comes from using the full convolution with both w and t which we called
 VoxelLutetiumPsiW *)
```

Index