Documentation: R and PBPK code for Drug P Model

# References for R

There is \*a lot\* out there on R from packages to base R. The first two of the following books are references I have used and found myself occasionally returning to. The third is one that I have seen many recommend but have not used myself.

Free:

The official “an introduction to R” is free and will likely always be updated

<https://cran.r-project.org/doc/manuals/R-intro.pdf>

I’ve used before:

Learning Base R is not too expensive and has online example code

https://www.amazon.com/Learning-Base-Lawrence-Mark-Leemis/dp/0982917481

<http://www.math.wm.edu/~leemis/Rcode/>

Commonly Recommended:

The R cookbook

https://www.amazon.com/Cookbook-OReilly-Cookbooks-Paul-Teetor/dp/0596809158/ref=dp\_rm\_title\_0

# Useful Syntax in R

### Assignment and data types

number = 5.0 #number or value

vector = c(1,2,3) #a vector which can be accessed by typing “b[1]” which will return the first

#element

df = data.frame(x=c(1,2,3),y=c(2,4,6),z=c(3,2,1))

#can be accessed in a variety of ways:  
## df[1,1] first row first column is returned

## df[1,] first row

## df[,1] first column OR df[,”x”] gives first column OR df$x gives first column OR df[[1]] gives first

#column

boolean = TRUE

string = “this is a string”

lst = list(number=number,vector=vector,boolean=boolean,string=string)

The above are a few of the most common data types that can be encountered in R. Numbers and vectors are commonly the building blocks of data frames which usually is a table similar to an excel spreadsheet and the thing to do data analysis on. vectors must contain data all of the same type whether that is a string, boolean, number, or other vectors of the same containing the same types.

What makes a data frame different from a vector of vectors (matrix) is that a data frame can handle names that can be used to refer to columns of the data frame and columns can be of different types for a data frame. A list is more general than a vector and can contain a bunch of different types.

The input parameters to the function that runs the pbpk model are arranged in a list partly to make it easier to call them. A final common data type is a function which will be discussed next.

### Writing functions

#How to define

f = function(input1, input2 = default\_value){

#code

#code

#code

return(output)

}

#How to call

f(in1,in2)

f(in1)

saved.output = f(in1)

Above is the syntax of a function. The first part defines a function and assigns it to “f” where f can be anything. In this case, f is defined with 2 possible inputs which will be used in the code. Possible inputs include numbers, strings, lists, data frames, and other functions. Any R object can be used as input to a function.   
  
The code of a function does not need to have to return something but if it does the last line of code is often return(output). This output can either go straight to the console or be saved by assigning it to some name. A function can also just have some side effects like plotting some things or writing files to a csv file.

### For loops

for(i in 1:10){

#code

}

The above code executes a for-loop, the first time i=1 then i=2 and so forth. The word “for” and “in” must be present along with the parenthesis and brackets but the “i” can be any letter and 1:10 can be in the form of any vector. 1:10 is a quick way of getting the integers 1 through 10 and can be used with any 2 integers to generate a vector of integers including the two integers on the ends and any in between. These integers can be going up or down.

### If/else

if(boolean){

#code

} else {

#other code

}

The above code is an if else statement where boolean = TRUE or FALSE. Specifically this can be testing to see if some value is large or small enough. The following code allows for checking of equality and inequalities for some values a and b to get boolean values.

a==b

a < b

a <= b

### setwd(“filepath”) and getwd()

getwd() tells you where you currently are in the computers file system while setwd(“filepath”) is a way to direct to where you want your working directory to be. If you have a lot of files or a lot of data, all to load into R it is a good idea to put all the files in the same folder and then put the working director their so that the filepath to the files becomes just the name of the file. In other words, if you need to load a csv file named “file.csv” from your working directory you can just type read.csv(“file.csv”).

Another way of setting the working directory in RStudio is clicking Session > Set Working Directory > Choose Directory and then selecting the folder with all of the files you need

### file.choose()

file.choose() gives us the ability to choose a file using the mouse without having to type in the file path. The return value is itself a file path that can be copied and pasted into a function such as source() or read.csv().

For the function source() another way of loading R files into R is by clicking Code > Source File or by using the keys Ctrl+Alt+G and then selecting the file you want to run.

Another way of loading data files into R other than read.csv() or a similar function is by clicking Import Dataset in the top right of RStudio and selecting the filetype you want to import. This may require installing packages into R and then loading them. R does not load packages into your working directory/workspace but into the user library which is likely located elsewhere in your computer

### help() or ?function

The help files can be called for practically any function in R from even the most basic things like c(), list() and data.frame() if you want to know more about how those data types work or if you are ever using some function you are unfamiliar with and it is behaving in some way you don’t understand. Most of the help files have descriptions of inputs, outputs, use, and a couple examples of how the function might be used. They also often have similar functions that may be useful to look at and might be better suited to a particular problem you might be trying to solve.

### Plot, lines, legend, and so forth

These are functions involved in plotting data or simulation data on a graph. I recommend using the help files and maybe consulting a book as I am somewhat familiar with using the package ggplot2 but have never been very good at graphing things in R.

# Finished Files

## DrugP Deterministic

This file loads the package deSolve and then defines the Pentamidine Model Simulation with fixed parameters. The functions within this file are:

### assign\_parameters():

roughly equivalent to INITIAL in acslx. This is where all of your constants go

### calculate\_varaiables(parameters):

Currently takes in a list of parameters from the function assign\_parameters() and returns the same list. Might be useful for defining variables that are used in solving the differential equation program but only need to be calculated once from constants and don’t need to be accessed before running the program

### pulse(t,tz,p,w):

This is the same as PULSE in acslx. W is the width of the pulse, p is the total period of how often the system repeats, and so on.

### Derivative(t,y,,parameters,…)

This is roughly equivalent to DYNAM IC in acslx and where all of the code to be solved by the differential equation goes. This function is never called except by the ode solver in run\_model()

### Run\_model(parameters)

This takes in a list of parameters and simulates your model. The output depends on what you have set your output to be within the derivative code. It returns a bunch of output that includes an object that can be treated like a data frame.

## MCPentamidine

The file is roughly the same as the deterministic model but makes use of the TruncatedNormal file to generate monte carlo simulation data.

## Data.m.R

This file loads the data sets from csv files into R, saves them to variables, and then runs the model and plots for each variable. The function model.and.plot(df) takes in a data frame as its argument. It expects that data frame to have BW and Dose as two of its columns and uses the first value of these columns to set the respective parameters after getting the parameters from the model. It then runs the model, saves those results particular to that model to a data frame, and plots the data as concentration and time and adds a line for the simulation. You can hopefully use this as a model of how to plot using an M file in asclx but this one in R.

Next, I have combined all of the data together: the function rbind() binds rows into a new data frame. Then defining a function plotCI() I can plot all of the data together along with a confidence interval simulation