Assignment week 3

Detailed analysis of the glucocorticoid receptor dynamica model

Student: Vincent Talen

Student number: 389015

Class: BFV2

Study: Bio-Informatics

Institute: Institute for Life Science & Technology

Teacher: Tsjerk Wassenaar

Date: 2022-05-30

Contents

1 Assignment 1: Assessing Model Validity

To assess the validity of the model, the experimental data will be compared to the simulation data from previous week. The comparison will be done by plotting the experimental data and the simulation data in the same graph.

1.1 Loading in experiment data

```
# Load in data from file
data <- read.csv("MPL.csv", na.strings = "NA")
# Rename 'mRNA' and 'Free_receptor' columns to use the same name scheme
names(data)[4:5] <- c("Rm", "R")
```

The actual experiment data contains multiple values per time, if a plot is made with those raw values the plot below will result.

```
# Create plot with raw values to demonstrate the need of using medians
ggplot(data, mapping = aes(x = time, y = R, width = 2)) + geom_line() + geom_point()
```

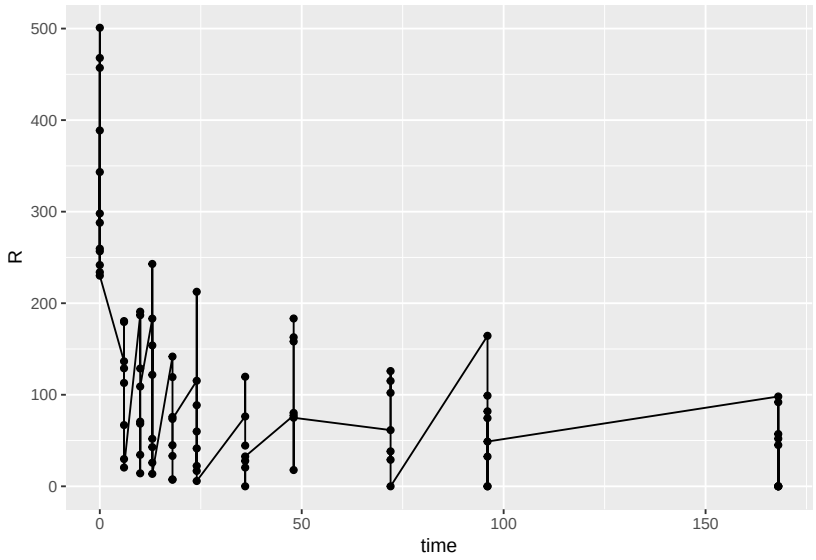


Figure 1: Demonstration of plotting raw data

As can be seen the line goes through each single point and thus is the line not that useful. So to be able to get a meaningful plot the medians are used to create a useful line of the medians that can then be used to compare the values.

```
# Create new data frame containing the medians per time
data.medians <- aggregate(data[,c("MPL_conc","Rm","R")],
                          list(data$dose,data$time), median, na.rm = T)
names(data.medians)[1:2] <- c("dose","time")
```

1.2 Implementing GRD model

Table 1: Initial Values for MPL

Parameter	Value	Unit	Explanation
Rm	4.74	$fmol/g\ liver$	concentration of receptor mRNA
R	267	$fmol/mg\ protein$	concentration of free receptor in cytosol
DR	0	$fmol/mg\ protein$	concentration of receptor complex in cytosol
DR_N	0	$fmol/mg\ protein$	concentration of receptor complex in nucleus

Table 2: Parameter Values for MPL

Parameter	Value	Unit	Explanation
k.s_Rm	2.9	$fmol/g\ liver/h$	zero-order rate constant of receptor mRNA synthesis
k.d_Rm	0.612	-	first-order rate constant receptor mRNA degradation
IC.50_Rm	26.2	$fmol/mg\ protein$	concentration of DR_N where receptor mRNA synthesis drops to 50% of base value
k.on	0.00329	$L/nmol/h$	second-order rate constant of receptor complex formation
k.T	0.63	$1/h$	first-order rate constant of translocation of receptor complex to nucleus
k.re	0.57	$1/h$	first-order rate constant of receptor 'recovery' from nucleus to cytosol
R.f	0.49	-	fraction of receptor being recycled from complexes
k.s_R	3.22	-	first-order rate constant of receptor synthesis
k.d_R	0.0572	$1/h$	first-order rate constant of receptor degradation
D	-	$nmol/L$	plasma concentration of corticosteroid

```
# Create function of basic GRD model
modelBasic <- function(time, state, parms) {
  ## Unpack the current state and the parameters for instant access
  with(as.list(c(state, parms)), {
    ## Calculate delta for each equation and return them in a list
    delta.Rm <- k.s_Rm * ( 1 - DR_N / ( IC.50_Rm + DR_N )) - k.d_Rm * Rm
    delta.R <- k.s_R * Rm + R.f * k.re * DR_N - k.on * D * R - k.d_R * R
    delta.DR <- k.on * D * R - k.T * DR
    delta.DR_N <- k.T * DR - k.re * DR_N
    return(list(c(delta.Rm, delta.R, delta.DR, delta.DR_N)))
    })
}

# Zero state values and time frame
basic.zero.state <- c(Rm = 4.74, R = 267, DR = 0, DR_N = 0)
basic.times <- seq(0, 168, by = 1)

# Create function that converts concentration of MPL from ng/mL to nmol/L
calculateD <- function(ng.ml.concentration) {return(ng.ml.concentration * 1000 / 374.471)}
# Define parameters determined for methylprednisolone (MPL)
basic.parameters <- c(k.s_Rm = 2.9, k.d_Rm = 0.612, IC.50_Rm = 26.2,
                      k.on = 0.00329, k.T = 0.63, k.re = 0.57, R.f = 0.49,
                      k.s_R = 3.22, k.d_R = 0.0572, D = calculateD(20))
```

1.3 Create global functions that help to make plots

Functionality is coded into functions so everything can be done easily and dynamically. The `state.info.df` helps to create a plots for state values at once in combination with the `createPlot` functions. After multiple plots are created for one question or scenario they can be passed on in a list to `printAndArrangePlots`, this function will then print the plots in a nice grid as a single image.

```
# Data frame that allows access to state value information to plot them in one fell swoop
state.info.df <- data.frame(
  name = c("Rm", "R", "DR", "DR_N"),
  unit = c("fmol/g liver", "fmol/mg protein", "fmol/mg protein", "fmol/mg protein"),
  title = c("Receptor mRNA", "Free receptor",
            "Receptor complex in cytosol", "Receptor complex in nucleus"))

# Function to print list of plots in an arranged grid with common legend at the bottom
printAndArrangePlots <- function(plot.list, grid.title) {
  my.grid <- ggarrange(plotlist = plot.list, common.legend = TRUE, legend = "bottom",
                       ncol = 2, nrow = ceiling(length(plot.list)/2))
  print(annotate_figure(my.grid, top = text_grob(grid.title)))
}
```

1.4 Create plots with experimental data

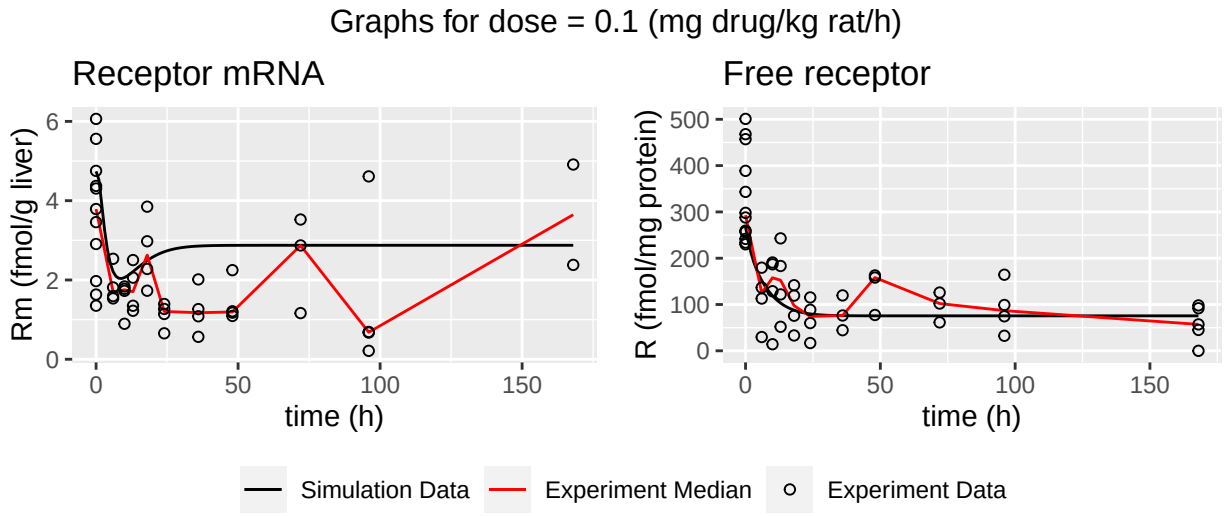
To create the plots with the experimental data, two functions are made for this assignment. A main function that collects all the data and creates subsets that can then be given to the second function that can create the plots with it.

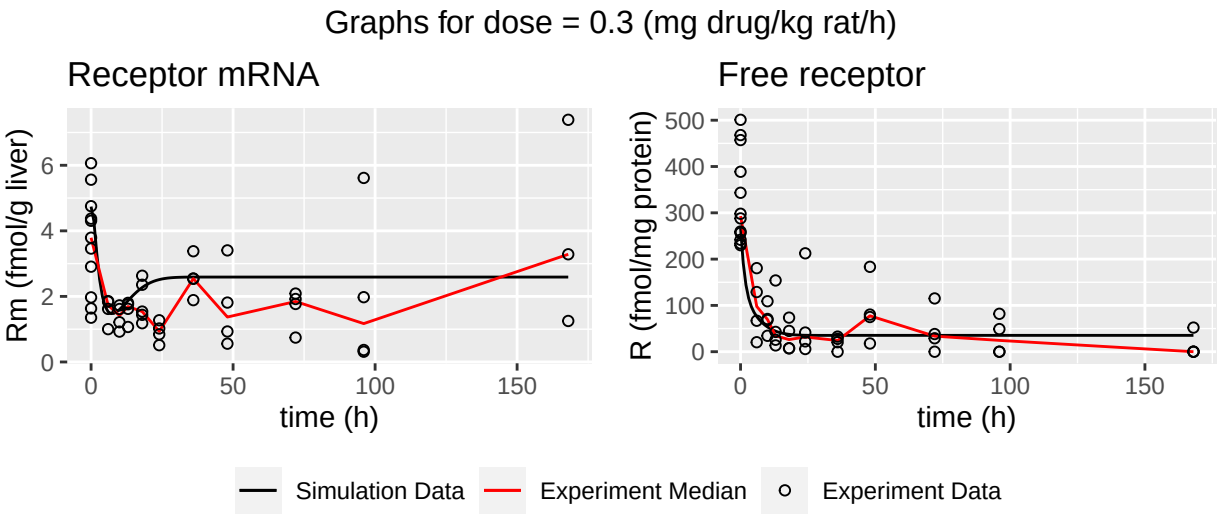
```
# Main function for assignment 1: If given a valid dose, plots will be made for it's data
a1.mainFunction <- function(current.dose) {
  ## Change D parameter to median of current dose
  cur.median.MPL_conc <- median(data$MPL_conc[data$dose == current.dose], na.rm = T)
  basic.parameters$D <- calculateD(cur.median.MPL_conc)

  ## Get subset of experiment data of current dose
  cur.data.points <- subset(data, dose == 0.0 | dose == current.dose)
  ## Get subset of experiment median data of current dose
  cur.data.medians <- subset(data.medians, dose == 0.0 | dose == current.dose)
  ## Perform ODE function with the model to get simulation data
  cur.data.simulated <- as.data.frame(ode(func = modelBasic, times = basic.times,
                                          y = basic.zero.state, parms = basic.parameters))

  ## Create plots
  plots <- lapply(c("Rm", "R"), a1.createPlot,
                  cur.data.simulated, cur.data.medians, cur.data.points)
  ## Print the plots in an arranged grid with the legend at the bottom
  printAndArrangePlots(plot.list = plots,
    grid.title = sprintf("Graphs for dose = %s (mg drug/kg rat/h)", current.dose))
}

# Function that fully creates a plot for given state variable with given data
a1.createPlot <- function(state.var, data.simulated, data.medians, data.points) {
  ## Collect current state variable from data frame that contains name, unit and title
  cur.var <- subset(state.info.df, name == state.var)
```

```
  ## Create plot for current state variable
  plot <- ggplot(data.simulated, aes(x = time, y = !!sym(cur.var[["name"]]))) +
    labs(title = cur.var["title"], x = "time (h)",
         y = sprintf("%s (%s)", cur.var["name"], cur.var["unit"])) +
    ### Add the lines and data points
    geom_line(aes(color = "Simulation Data")) +
    geom_line(aes(color = "Experiment Median"), data = data.medians) +
    geom_point(aes(color = "Experiment Data"), data = data.points, shape = 1) +
    theme(legend.position = "bottom", plot.margin = margin(0.25, 0.25, 0.25, 0.25, "cm")) +
    ### Use scale_color_manual to color the data and also set their names in the legend
    scale_color_manual(name = "",
      values = c("black","red","black"),
      limits = c("Simulation Data", "Experiment Median", "Experiment Data")) +
    ### Correct the displayed line types in the legend
    guides(color = guide_legend(
      override.aes = list(linetype = c(1, 1, NA), shape = c(NA, NA, 1))))
  return(plot)
}
```

Finally, after making all the functions, all unique doses (except 0.0) are gathered. Then the lapply function is used on the doses and the main function to create the plots for each dose.

```
# Get all unique doses present in the data except 0.0
all.doses <- unique(data$dose)
all.doses <- all.doses[!all.doses %in% (0.0)]

# Create plots for each dose by using lapply() on a1.mainFunction()
invisible(lapply(all.doses, a1.mainFunction))
```





1.5 Comparing the resulting plots

When looking the resulting plots it can be seen that the receptor mRNA is not entirely as expected like the model but the free receptor concentration in the cytosol is close to the expectations. Around 50 hours a spike can be seen in the free receptor concentration that is not expected but shortly after it decreases rather than reaching a steady state after around 25 hours like the model. The amount of receptor mRNA varies a lot more than expected as time passes, in the model a steady state is expected after around 25 hours too but the experiment is completely different and the amount of receptor mRNA keeps changing. At first it does decrease like the model but then has a little spike and after some time an even bigger spike that dips really low. After 100 hours the receptor mRNA amount keeps climbing to above the expected steady state after being below it for almost the entire time.

The reason for the different results could be that the medicine is not as effective as thought and that it thus does not really cause the receptor mRNA to reach a stable state over time. With the lower dose more variation can be seen in the amount of receptor mRNA present than in the higher dose, which is a little bit more stable. Overall it seems that the effect is highest when first administered and the effectiveness declines because of the big decrease, the high variations and the steady increase at the end.

2 Assignment 2: Simulating Multiple Scenario's

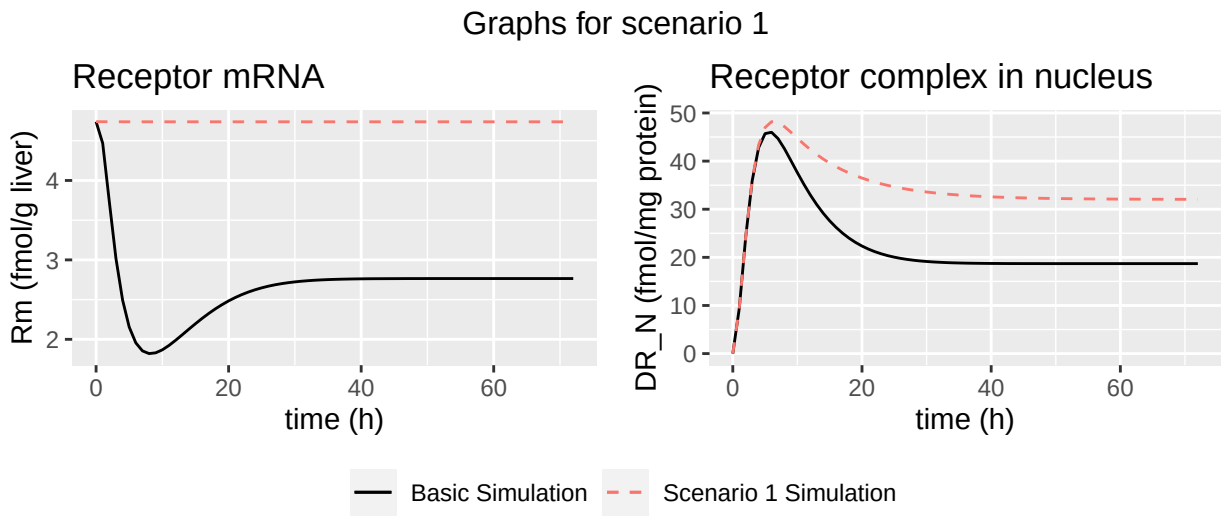
To find out about the effect and importance of different parameter values, multiple scenario's will be simulated and then compared to the basic scenario. First the basic simulation data is gathered and a function is made that creates a plot for a state variable containing a list of simulations.

```
# Set time frame for assignment 2
a2.times <- seq(0, 300, by = 1)
# Get the basic simulation data that will be used for all scenario's
basic.simulation <- as.data.frame(ode(times = a2.times, y = basic.zero.state,
                                      parms = basic.parameters, func = modelBasic))

# Function to create the plots for assignment 2
a2.createPlot <- function(state.var, max.time, sim.names, sim.data) {
  ## Collect current state variable from data frame that contains name, unit and title
  cur.var <- subset(state.info.df, name == state.var)
  ## Get line colors for scenario simulations
  extra.colors <- hue_pal()(length(sim.names))

  ## Create plot for current state variable
  plot <- ggplot(basic.simulation, aes(x = time, y = !!sym(cur.var[["name"]]))) +
    labs(title = cur.var["title"], x = "time (h)",
         y = sprintf("%s (%s)", cur.var["name"], cur.var["unit"])) +
    theme(legend.position = "bottom",
          plot.margin = margin(0.25, 0.25, 0.25, 0.25, "cm")) +
    xlim(NA, max.time) +
    ## Add lines of all the simulations
    geom_line(aes(color = "Basic Simulation")) +
    unlist(mapply(function(cur.sim.name, cur.sim.data)
      geom_line(aes(color = cur.sim.name), data = cur.sim.data, linetype = "dashed"),
      sim.names, sim.data)) +
    ## Use scale_color_manual to color the data and also set their names in the legend
    scale_color_manual(name = "", values = c("black", extra.colors),
                       limits = c("Basic Simulation", sim.names)) +
    ## Correct the displayed line types in the legend
    guides(color = guide_legend(override.aes = list(
      linetype = c(1, rep(2, length(sim.names))))))
  return(plot)
}
```

2.1 Activated receptor complex concentration without auto-regulation of glucocorticoid receptor

Auto-regulation of glucocorticoid receptor is caused by the activated receptor complexes in the nucleus (`DR(N)`). It is expressed into the parameter `IC.50_Rm` which is the concentration of DR(N) where receptor mRNA synthesis drops to 50% of it's base value. So if DR(N), which needs drugs to form, does not auto-regulate the synthesis of the receptor mRNA then IC.50_Rm can be removed from the equation. After removing `IC.50_Rm` from the equation `1 - DR_N / DR_N` remains, whose outcome is always `1-1 = 0`, so this entire part of the equation can be removed. Without the auto-regulation from DR(N) there is no effect of the drug on receptor mRNA synthesis anymore.

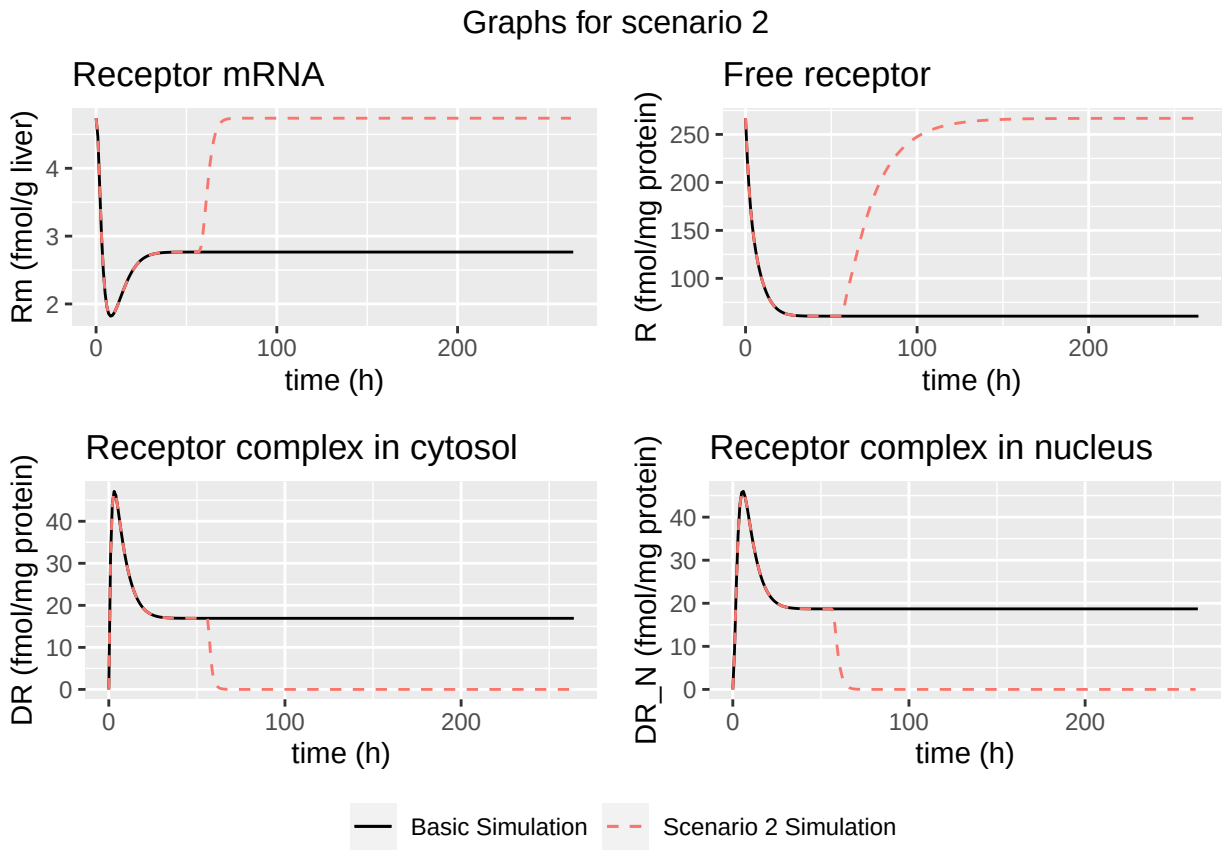
```r
# Create new model where there is no receptor mRNA auto-regulation
modelScen1 <- function(time, state, parms) {
  ## Unpack the current state and the parameters for instant access
  with(as.list(c(state, parms)), {
    ## Calculate delta for each equation and return them in a list
    delta.Rm <- k.s_Rm - k.d_Rm * Rm
    delta.R <- k.s_R * Rm + R.f * k.re * DR_N - k.on * D * R - k.d_R * R
    delta.DR <- k.on * D * R - k.T * DR
    delta.DR_N <- k.T * DR - k.re * DR_N
    return(list(c(delta.Rm, delta.R, delta.DR, delta.DR_N)))
    })
}

# Run the simulation for the no mRNA synthesis auto-regulation scenario
scen1.simulation <- as.data.frame(ode(times = a2.times, y = basic.zero.state,
                                      parms = basic.parameters, func = modelScen1))

# Create plots for state variables
plots <- lapply(c("Rm", "DR_N"), a2.createPlot, 72,
                list("Scenario 1 Simulation"), list(scen1.simulation))
# Print the plots in an arranged grid with the legend at the bottom
printAndArrangePlots(plot.list = plots, grid.title = "Graphs for scenario 1")
```



2.2 Receptor and mRNA concentrations when stopping drug treatment at steady state

To dynamically code stopping the drug treatment at a steady state, the built in root + event arguments of the `ode` function are used. A steady state is reached when the state values do not change (much) anymore at a time, this can be determined that when the total of delta's returned by the model function are (close to) zero. This is coded into a custom root function that returns two numeric values, the first value representing the first steady state and the second is when the second steady state is reached. When the first steady state is reached, the root function then activates the custom event function, which changes the state value `D` to zero, to simulate the stopping of the drug treatment. When the second steady state is reached the root function causes the simulation to terminate. The simulation data is then plotted to see the time course concentrations of the original state values, so except D.

```r
# Zero state values now including D
scen2.zero.state <- c(Rm = 4.74, R = 267, DR = 0, DR_N = 0, D = calculateD(20))
# Define parameters determined for methylprednisolone (MPL) without D
scen2.parameters <- c(k.s_Rm = 2.9, k.d_Rm = 0.612, IC.50_Rm = 26.2, k.on = 0.00329,
                      k.T = 0.63, k.re = 0.57, R.f = 0.49, k.s_R = 3.22, k.d_R = 0.0572)

# Create new model that includes D in state deltas
modelScen2 <- function(time, state, parms) {
  ## Unpack the current state and the parameters for instant access
  with(as.list(c(state, parms)), {
    ## Calculate delta for each equation and return them in a list
    delta.Rm <- k.s_Rm * ( 1 - DR_N / ( IC.50_Rm + DR_N )) - k.d_Rm * Rm
    delta.R <- k.s_R * Rm + R.f * k.re * DR_N - k.on * D * R - k.d_R * R
    delta.DR <- k.on * D * R - k.T * DR
    delta.DR_N <- k.T * DR - k.re * DR_N
    delta.D <- 0
    return(list(c(delta.Rm, delta.R, delta.DR, delta.DR_N, delta.D)))
    })
}

# Root function: root.normal activates event and root.terminate terminates simulation
scen2.root <- function(time, state, parms) {
  # Normal root: Use model to get delta's return total delta - 0.0001
  delta.list <- unlist(modelScen2(time, state, parms))
  root.normal <- sum(abs(delta.list)) - 1e-4
  # Terminator root: if sum of normal root and D state are 0
  root.terminate <- root.normal + state["D"]
  return(c(root.normal, root.terminate))
}

# The event (stopping drug treatment) happens when root (steady state) is triggered
scen2.event <- function(time, state, parms) {
  state["D"] <- 0
  return(state)
}

# Run the simulation for the steady state drug stop scenario
scen2.out <- ode(func = modelScen2,
  times = a2.times, y = scen2.zero.state, parms = scen2.parameters,
  events = list(func = scen2.event, root = TRUE, terminalroot = 2), rootfun = scen2.root)
scen2.simulation <- as.data.frame(scen2.out)

# Create plots for state variables
plots <- lapply(c("Rm", "R", "DR", "DR_N"), a2.createPlot, attributes(scen2.out)$dim[1],
                list("Scenario 2 Simulation"), list(scen2.simulation))
# Print the plots in an arranged grid with the legend at the bottom
printAndArrangePlots(plot.list = plots, grid.title = "Graphs for scenario 2")
```

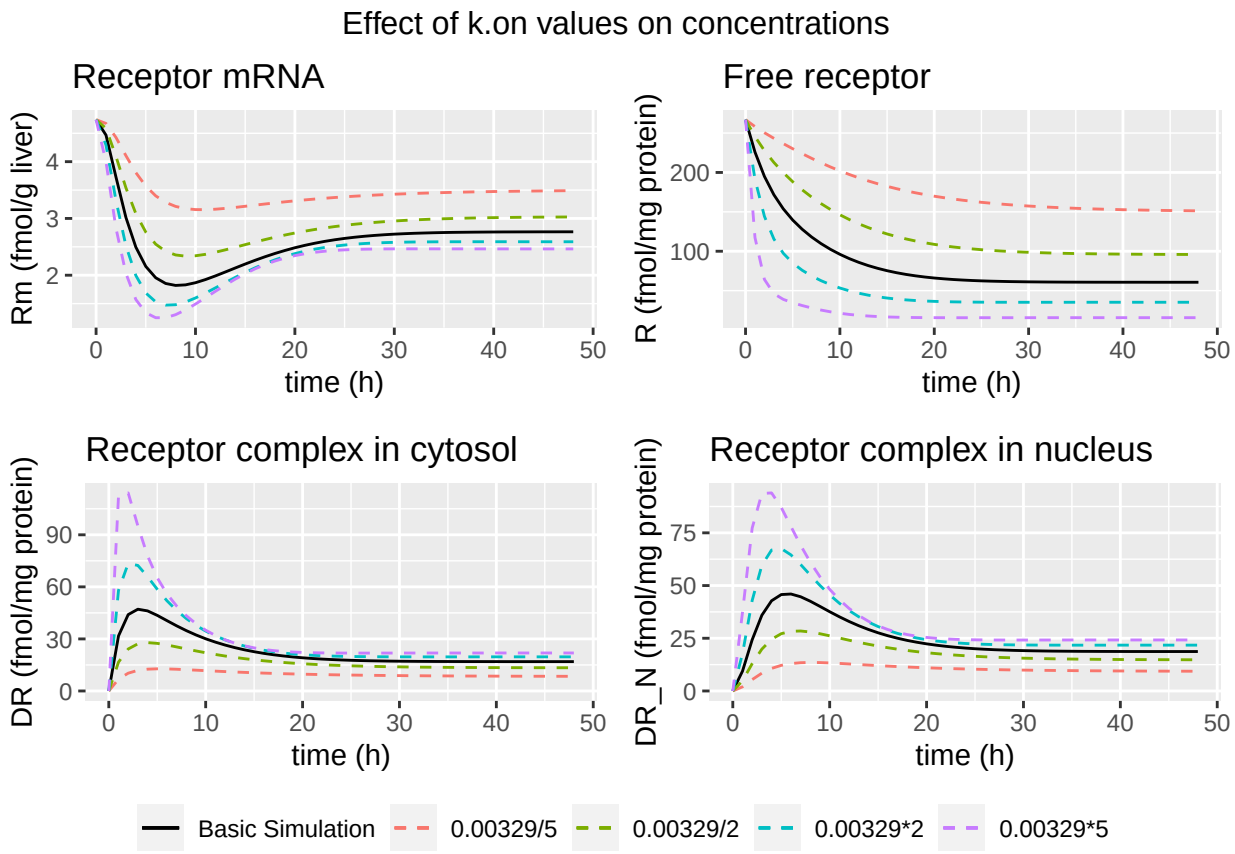


2.3 Effect of different k.on and k.re values on receptor and mRNA dynamics

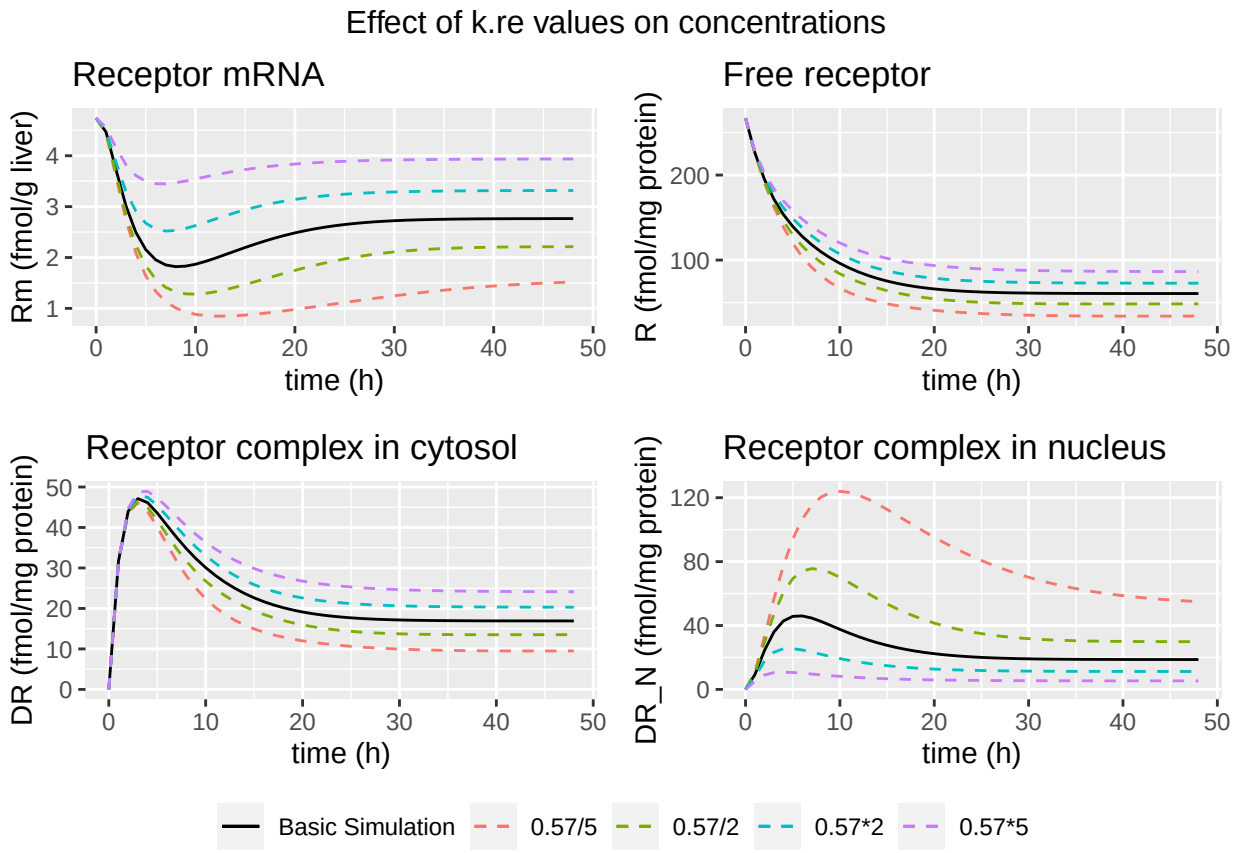
```
# Main function for assignment 2 to easily change parameters
a2.mainFunction <- function(new.values, plot.title, wanted.state.vars) {
  ## Get simulation names and data for each altered parameter scenario
  sim.names <- lapply(new.values, function(x) paste(unname(x), collapse=", "))
  sim.data <- lapply(new.values, a2.simulateNewParameters)

  ## Create plots for state variables
  plots <- lapply(wanted.state.vars, a2.createPlot, 48, sim.names, sim.data)
  ## Print the plots in an arranged grid with the legend at the bottom
  printAndArrangePlots(plot.list = plots, grid.title = plot.title)
}

# Function that takes new parameter values and gets simulation data with them
a2.simulateNewParameters <- function (new.values.string) {
  ## Turn the new values from strings of expressions to numeric values
  new.values.num <- sapply(new.values.string, function(x) eval(parse(text=x)))

  ## Replace the parameters with the new values
  cur.scen.parameters <- basic.parameters
  cur.scen.parameters[names(new.values.num)] <- new.values.num

  ## Run the simulation and return data
```

```
  cur.scen.out <- ode(func = modelBasic, times = basic.times,
                      y = basic.zero.state, parms = cur.scen.parameters)
  cur.scen.simulation <- as.data.frame(cur.scen.out)
  return(cur.scen.simulation)
}
```

```
# Create plots of simulations with changed k.on values
new.k.on.values <- list(c(k.on = "0.00329/5"),
                        c(k.on = "0.00329/2"),
                        c(k.on = "0.00329*2"),
                        c(k.on = "0.00329*5"))

a2.mainFunction(new.k.on.values, "Effect of k.on values on concentrations",
                c("Rm", "R", "DR", "DR_N"))
```

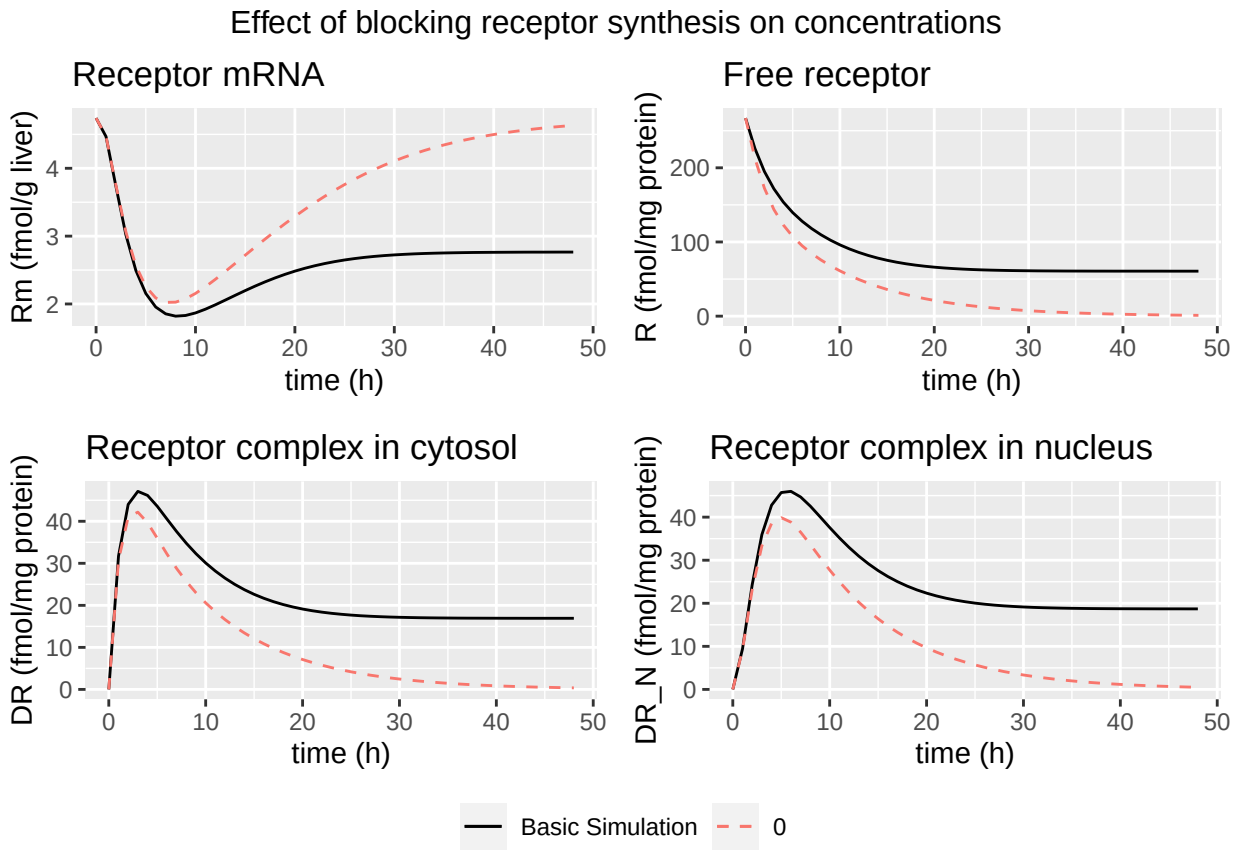



```
# Create plots of simulations with changed k.re values
new.k.re.values <- list(c(k.re = "0.57/5"),
                        c(k.re = "0.57/2"),
                        c(k.re = "0.57*2"),
                        c(k.re = "0.57*5"))

a2.mainFunction(new.k.re.values, "Effect of k.re values on concentrations",
                c("Rm", "R", "DR", "DR_N"))
```



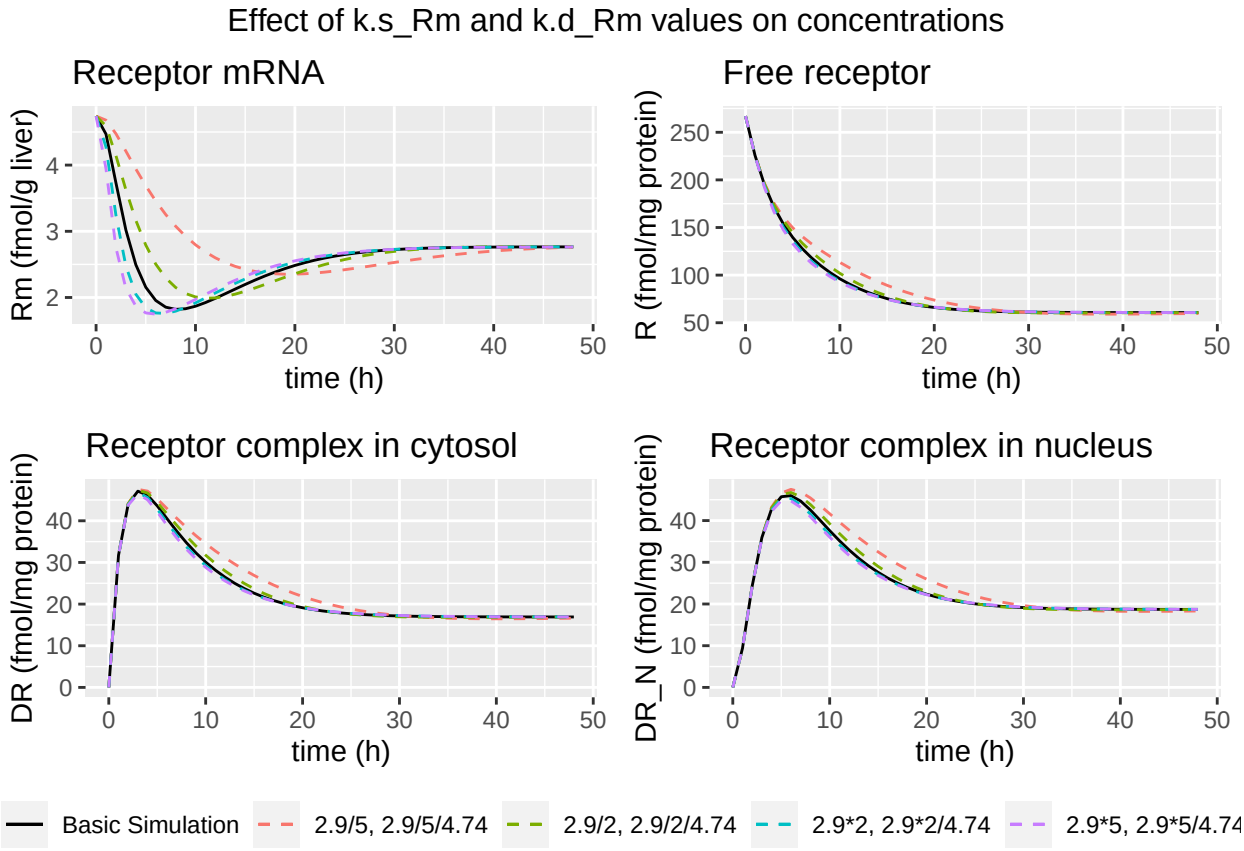
2.4 Effects of completely blocking receptor synthesis

```
# Create plots of simulations with changed k.s_R values
new.k.s_R.values <- list(c(k.s_R = "0"))

a2.mainFunction(new.k.s_R.values,
                "Effect of blocking receptor synthesis on concentrations",
                c("Rm", "R", "DR", "DR_N"))
```



2.5 System dynamics with increased or decreased baseline mRNA production

```
# Create plots of simulations with changed k.s_Rm and k.d_Rm values
new.mrna.prod.values <- list(c(k.s_Rm = "2.9/5", k.d_Rm = "2.9/5/4.74"),
                             c(k.s_Rm = "2.9/2", k.d_Rm = "2.9/2/4.74"),
                             c(k.s_Rm = "2.9*2", k.d_Rm = "2.9*2/4.74"),
                             c(k.s_Rm = "2.9*5", k.d_Rm = "2.9*5/4.74"))

a2.mainFunction(new.mrna.prod.values,
                "Effect of k.s_Rm and k.d_Rm values on concentrations",
                c("Rm", "R", "DR", "DR_N"))
```


Effect of k.s_Rm and k.d_Rm values on concentrations
Receptor mRNA
Rm (fmol/g liver)
time (h)
Free receptor
R (fmol/mg protein)
time (h)
Receptor complex in cytosol
DR (fmol/mg protein)
time (h)
Receptor complex in nucleus
DR_N (fmol/mg protein)
time (h)
Basic Simulation
2.9/5, 2.9/5/4.74
2.9/2, 2.9/2/4.74
2.9*2, 2.9*2/4.74
2.9*5, 2.9*5/4.74

References

[1] Barnes, P.J. (2011), *Glucocorticosteroids: current and future directions*, British Journal of Pharmacology, 163: 29-43, https://doi.org/10.1111/j.1476-5381.2010.01199.x (accessed May 11, 2022).