# bioPN manual

Roberto Bertolusso and Marek Kimmel

October 21, 2010

## Contents

# 1 Availability

Updated versions of the manual, as well as the code needed to run the examples, and the library (currently version 1.0.0), can be found on http://www.stat.rice.edu/∼mathbio/bioPN.

# 2 Functions in the Library

The library has 4 kind of functions:

- Exact stochastic simulation (4 functions)
  - `GillespieOptimDirect`: "in house" optimized version of the Gillespie SSA algorithm.
  - `GillespieDirectGB`: optimized version of the Gillespie SSA algorithm according to Gibson and Bruck [1]
  - `GibsonBruck`: Next reaction method by Gibson and Bruck [1].
  - `GillespieDirectCR`: constant-time composition/rejection algorithm [6].
- Pure deterministic integration (1 function)
  - `RungeKuttaDormandPrince45`: Runge-Kutta Dormand and Prince algorithm
- A hybrid of the above (1 function)
  - `HaseltineRawlings`: Haseltine and Rawlings method with deterministic integration of fast transitions [3]
- Dynamic re-partitioning algorithm (1 function)

  - `PartitionedLeaping`: Partitioned leaping algorithm (PLA) [2]. Transitions are dynamically re-partitioned between 4 categories (very slow, slow, medium, and fast) and simulated accordingly (SSA, $\tau$-leaping, CLE, deterministic).

## 2.1 Usage

```
GillespieOptimDirect(model, timep, delta = 1, runs = 1)
GillespieDirectGB(model, timep, delta = 1, runs = 1)
GibsonBruck(model, timep, delta = 1, runs = 1)
GillespieDirectCR(model, timep, delta = 1, runs = 1)

RungeKuttaDormandPrince45(model, timep, delta = 1, ect = 1e-09)

HaseltineRawlings(model, timep, delta = 1, runs = 1, ect = 1e-09)

PartitionedLeaping(model, timep, delta = 1, runs = 1)
```

## 2.2 Arguments

- `model`: list containing the following named elements:
  - `pre`: pre matrix, with as many rows as transitions (reactions), and columns as places (reactants). It has the stoichiometrics of the left sides of the reactions.
  - `post`: post matrix, with as many rows as transitions, and columns as places (products). It has the stoichiometrics of the right sides of the reactions.
  - `h`: list of propensity constants or functions returning the propensity (with as many elements as transitions).
  - `slow`: vector of zeros for slow transitions and ones for fast transitions. Only needed for `HaseltineRawlings`. Ignored otherwise.
  - `M`: Initial marking (state) of the system.
  - `place`: vector with names of the places.
  - `transition`: vector with names of the transitions.
- `timep`: It can be either a numeric, indicating for how long (in the same time units as the propensity constants) the process will run, or a functions (R or C), in which case can be used to change the protocol at time intervals. See below.
- `delta`: Interval time at which the state will be saved.
- `runs`: How many runs will be performed.
- `ect`: Precision for the fast reactions.

## 2.3 Returned Value

The functions return a list with the following elements:

- `place`: vector with the names of the places if supplied. If not, the function creates names as follows: P1, P2, ...
- `transition`: vector with the names of the transitions if supplied. If not, the function creates names as follows: T1, T2, ...
- `dt`: vector containing the discretized times at which the state is saved (according to delta)
- `run`: list with as many elements as runs. We will describe the first element, run[[1]], as the rest have exactly the same structure. It is also a list, with the following elements:

- `run[[1]]$M`: list with as many elements as places, each of them containing the state of the system sampled according to `delta`.
- `run[[1]]$transitions`: vector with as many elements as transitions, with the total of time each slow reaction fired.
- `run[[1]]$tot.transitions`: numeric with the summ of `run[[1]]$transitions`.

# 3 Example 1: Stochasticity in Gene Regulation

The first example is a simple model that has steady state theoretical solutions [4].

## 3.1 Reactions

$$
\begin{array}{ll}
I \xrightarrow{c} A & mRNA \xrightarrow{K} mRNA + protein \\
A \xrightarrow{b} I & mRNA \xrightarrow{1} \oslash \\
A \xrightarrow{H} A + mRNA & protein \xrightarrow{r} \oslash
\end{array}
$$

This model consists in 4 places (reactants, products) and 6 transitions (reactions). They are illustrated in the following table

| Places | Transitions |
|---|---|
| 1. inactive gene $I$ | 1. gene activation |
| 2. active gene $A$ | 2. gene inactivation |
| 3. $mRNA$ | 3. transcription |
| 4. *protein* | 4. mRNA degradation |
| | 5. translation |
| | 6. protein degradation |

## 3.2 Model Definition: First Approach

### 3.2.1 Pre Matrix

The first needed structure is the $6 \times 4$ matrix *pre*, which accounts for the stoichiometries of the left sides of the reactions.

```
pre <- matrix(c(1,0,0,0,
                0,1,0,0,
                0,1,0,0,
                0,0,1,0,
                0,0,1,0,
                0,0,0,1),
              ncol=4,byrow=T)
```

Transitions (reactions) are associated with rows, while places (reactants, products) are associated with columns.

### 3.2.2 Post Matrix

Define *post*, to account for the stoichiometries of the right side of the reactions.

```
post <- matrix(c(0,1,0,0,
                 1,0,0,0,
                 0,1,1,0,
                 0,0,0,0,
                 0,0,1,1,
                 0,0,0,0),
               ncol=4,byrow=T)
```

### 3.2.3 Propensities

Define the vector of propensities or hazards

```
h <- list(c=3, b=2, H=10, 1, K=6, r=0.25)
```

### 3.2.4 Initial Conditions

```
M <- c(1,0,0,0)
```

#### 3.2.5 Fast and Slow Transitions

Only necessary for `HaseltineRawlings`. Ignored by the other functions.

```
slow <- c(1,1,0,0,0,0)
```

This case will treat gene inactivation/activation as slow, while the rest as fast.

#### 3.2.6 Model Definition

```
model <- list(pre=pre, post=post, h=h, M=M, slow=slow)
```

#### 3.2.7 Load the Library

Load the library (can be done at any point before the call to the functions)

```
library(bioPN)
```

#### 3.2.8 Calling the Functions

```
timep <- 200
delta <- 1

## Completely deterministic run
Sim <- RungeKuttaDormandPrince45(model, timep, delta)

runs <- 10000
## Completely stochastic run
set.seed(19761111)  ## Set a seed
Sim <- GillespieOptimDirect(model, timep, delta, runs)
#Sim <- GibsonBruck(model, timep, delta, runs)
#Sim <- GillespieDirectGB(model, timep, delta, runs)
#Sim <- GillespieDirectCR(model, timep, delta, runs)

## Hybrid run
set.seed(19761111)  ## Set a seed
Sim <- HaseltineRawlings(model, timep, delta, runs)
```

### 3.3 Model Definition: Second Approach

This approach is closer to the way reactions are defined. It is more verbose than the previous one, but more suitable for large models.

#### 3.3.1 Constants

Let's start by defining the constants

```
H <- 10
K <- 6
r <- 0.25
c <- 3
b <- 2
```

#### 3.3.2 Names of the Places and Transitions

We will now define the names of the places and transitions and define variables for the amount of them

```
place <- c( "I", "A", "mRNA", "protein")

transition <- c("gene_activation", "gene_inactivation",
                "transcription", "mRNA_degradation",
                "translation", "protein_degradation")
```

```
places <- length(place)
transitions <- length(transition)
```

### 3.3.3 Variables with the Names of the Places and Transitions

The following will create variables that match the names of the places and transitions initialized to the corresponding column (place) or row (transition). We will use them when defining the transitions.

```
for (n in 1:places) {
  assign(place[n], n)
}
for (n in 1:transitions) {
  assign(transition[n], n)
}
```

### 3.3.4 Empty Matrices Pre and Post

First we create *pre* and *post* matrices with *transitions* rows and *places* columns filled with zeroes and the list of propensities

```
pre <- post <- array(0, dim <- c(transitions, places))
```

### 3.3.5 Empty List of Rate Constants

Also an empty list in which we will define the rate constants

```
h <- list()
```

### 3.3.6 Transitions: Progressive and Simultaneous Definition of $h$, *pre* and *post*

It is now time to define the transitions (reactions). The strategy is to concentrate on one transition at a time. The variable $i$ is just a temporary variable that is initialized with the name of the transition we are defining. We will only enter values for the places (columns) that are involved in this transition (row). Remember that we originally initialized the matrices with zeros. So, for the first transition, one "molecule" of inactive gene $I$ (a 1 in the I column of the gene_activation row of pre) will turn into one "molecule" of active gene $A$ (a 1 in the A column of the gene_activation row of post). The rate constant will be c.

```
## Gene Activation: I -> A   (c)
i <- gene_activation
h[[i]] <- c
pre[i, I] <- 1
post[i, A] <- 1
```

Let's do the same with the other reactions

```
## Gene Inactivation: A -> I   (d)
i <- gene_inactivation
h[[i]] <- b
pre[i, A] <- 1
post[i, I] <- 1
```

Note the following transition in which A appears in both sides of the reaction (so it is defined both in *pre* and *post*), meaning that A won't change in value when this transition fires.

```
## Transcription: A -> A + mRNA   (H)
i <- transcription
h[[i]] <- H
pre[i, A] <- 1
post[i, A] <- 1; post[i, mRNA] <- 1
```

The following transition only defines *pre* (this row of *post* is all zeros).

```
  ## mRNA Degradation: mRNA -> 0   (1)
  i <- mRNA_degradation
  h[[i]] <- 1
  pre[i, mRNA] <- 1

  ## Translation: mRNA -> mRNA + protein   (K)
  i <- translation
  h[[i]] <- K
  pre[i, mRNA] <- 1
  post[i, mRNA] <- 1; post[i, protein] <- 1

  ## Protein Degradation: mRNA -> 0   (r)
  i <- protein_degradation
  h[[i]] <- r
  pre[i, protein] <- 1
```

### 3.3.7 Initial Conditions

Define the marking at time zero (initial conditions) of the sytem (there is one inactive gene, all the other places are zero)

```
M <- rep(0, places)
M[I] <- 1
```

### 3.3.8 Number of Runs

Let's define how many runs we want to perform.

```
runs <- 10000
```

### 3.3.9 Saving the State of the System

Interval at which the state will be saved

```
delta <- 1
```

### 3.3.10 Simulation Time

```
timep <- 200
```

It is also possible to define a function to manage time (in R or C), which can be useful if the protocol requires to partition the simulation in time sections where the behaviour of the system may change. This situation will be explained in 3.6.

### 3.3.11 Model Definition

```
model <- list(pre=pre, post=post, h=h, M=M,
              place=place, transition=transition)
```

### 3.3.12 Calling the Functions

```
##############################
## Completely Deterministic ##
##############################
Sim <- RungeKuttaDormandPrince45(model, timep, delta, ect)

############################
## Completely Stochastic ##
############################
set.seed(19761111)
Sim <- GillespieOptimDirect(model, timep, delta, runs)
#Sim <- GibsonBruck(model, timep, delta, runs)
```

```
#Sim <- GillespieDirectGB(model, timep, delta, runs)
#Sim <- GillespieDirectCR(model, timep, delta, runs)

########################################################################
## Hybrid: mRNA and protein fast, gene activation/inactivation slow ##
########################################################################
slow <- rep(0, transitions)
slow[gene_activation] <- 1
slow[gene_inactivation] <- 1

model$slow <- slow

set.seed(19761111)
Sim <- HaseltineRawlings(model, timep, delta, runs, ect)
```

### 3.3.13 Example of Value Returned by the Functions

Using as an example the result from a completely stochastic run:

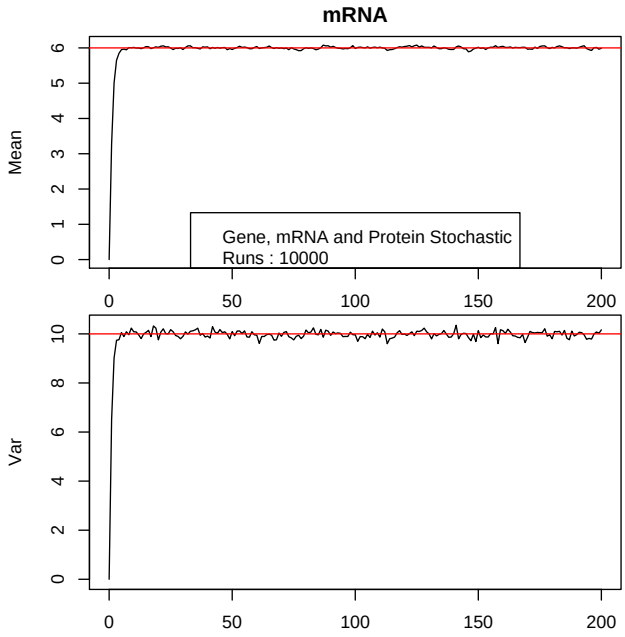
```
> str(Sim)
 $ place     : chr [1:4] "I" "A" "mRNA" "protein"
 $ transition: chr [1:6] "gene_activation" "gene_inactivation" "transcription" "mRNA_degradation" ...
 $ dt        : num [1:201] 0 1 2 3 4 5 6 7 8 9 ...
 $ run       :List of 10000
  ..$ :List of 3
  .. ..$ M             :List of 4
  .. .. ..$ : num [1:201] 1 0 1 0 0 0 0 0 0 0 ...
  .. .. ..$ : num [1:201] 0 1 0 1 1 1 1 1 1 1 ...
  .. .. ..$ : num [1:201] 0 2 2 2 7 6 10 10 6 9 ...
  .. .. ..$ : num [1:201] 0 2 18 16 48 82 105 142 147 168 ...
  .. ..$ transitions    : int [1:6] 245 245 1201 1192 6696 6540
  .. ..$ tot.transitions: int 16119
  ..$ :List of 3
  .. ..$ M             :List of 4
  .. .. ..$ : num [1:201] 1 0 0 0 0 1 1 0 1 0 ...
  .. .. ..$ : num [1:201] 0 1 1 1 1 0 0 1 0 1 ...
  .. .. ..$ : num [1:201] 0 1 2 5 6 5 4 8 10 10 ...
  .. .. ..$ : num [1:201] 0 7 18 39 47 81 88 92 103 128 ...
  .. ..$ transitions    : int [1:6] 243 242 1217 1205 7290 7156
  .. ..$ tot.transitions: int 17353

...... rest of the 10,000 runs follows...
```

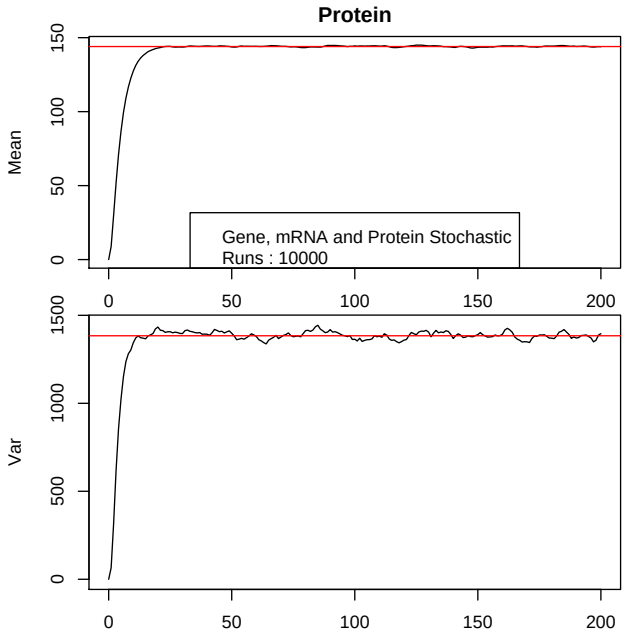
## 3.4 Plots

Let's now plot the mean and variances of the mRNA and protein and compare the results to the theoretical values

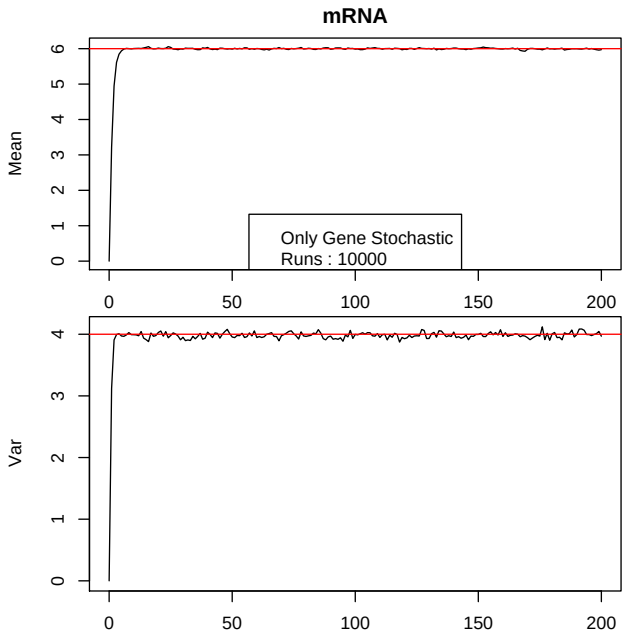
### 3.4.1 Completely Stochastic Case

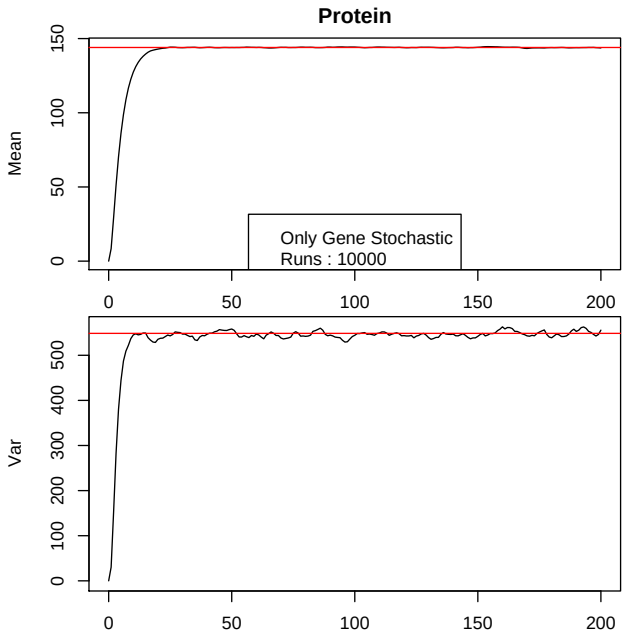
The lines in red are the theoretical values for the steady state solution.



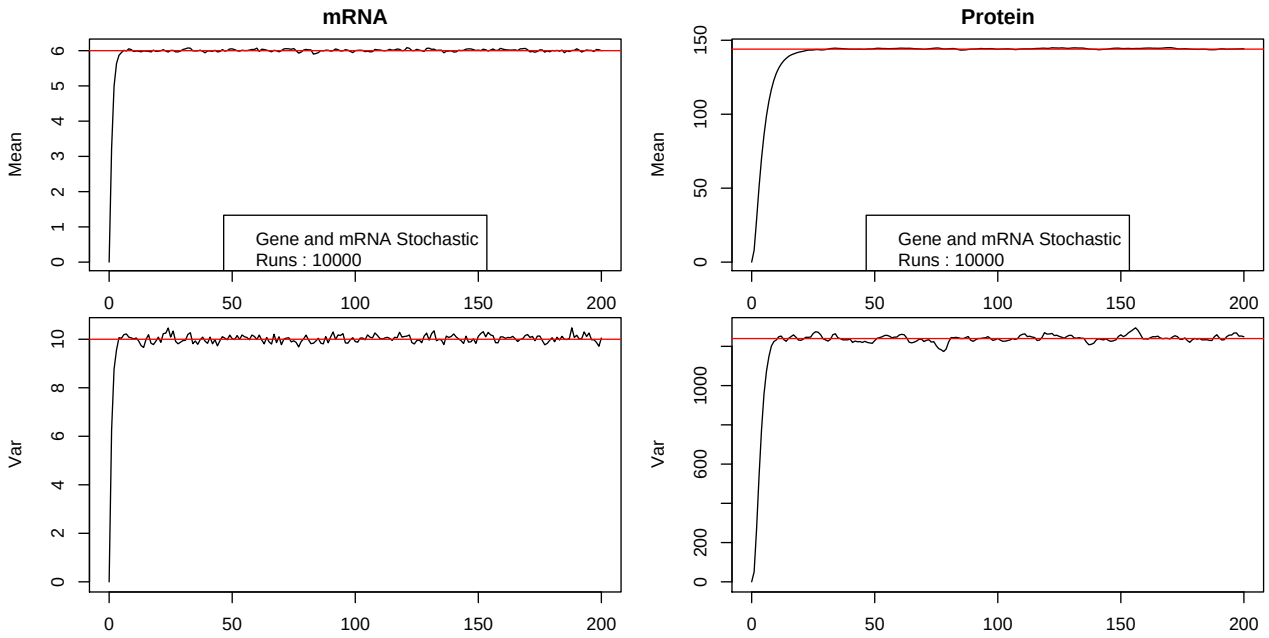




### 3.4.2 Hybrid: Only Gene Stochastic Case



### 3.4.3 Hybrid: Gene and mRNA Stochastic Case




## 3.5 How to assign a function to a propensity

All the reactions in this example are mass-action ones, so the function calculates the corresponding propensity on the fly. However, there are cases where specific propensities need to be provided. Using as an example the first reaction (gene activation) the corresponding propensity is $c \times I$.

### 3.5.1 R function

```
## Gene Activation: I -> A   (c)
i <- gene_activation
h[[i]] <- function() {c*y[I]}
pre[i, I] <- 1
post[i, A] <- 1
```

NOTE: Declaring a function in R will result in a sensible performance degradation.

### 3.5.2 C function

Create a text file called propensities.c with the following content

```
// Begin propensity.c
#define c 3.
#define I 0

double gene_activation_prop(double time, double *y) {
  return c*y[I];
}
// End propensity.c
```

Compile the file:

- R CMD SHLIB propensities.c (on Linux/Unix or Windows)

The R counterpart follows:

```
  ## Load the compiled code
  dyn.load("propensities.so") ## on Linux/Unix
  dyn.load("propensities.dll") ## on Windows


  ## Gene Activation: I -> A   (c)
  i <- gene_activation
  h[[i]] <- getNativeSymbolInfo("gene_activation_prop", PACKAGE="propensities")$address
  pre[i, I] <- 1
  post[i, A] <- 1
```

## 3.6 How to manage time with a function

If the simulation needs to be separated into different sections (for example the first 50 seconds with one behaviour, and the second 150 with another), instead of assigning T a number, it can be assigned a function (R or C).

### 3.6.1 R function

```
T <- function() {
   StartTime = round(StartTime,0)
   EndTime = 200
   if (StartTime == 0) {
      y[I] = 5
      EndTime = 50
   } else if (StartTime == 50) {
      y[I] = 10
   }
   list(EndTime,y)
}
```

### 3.6.2 C function

```
// Start of file time_protocol.c

#include <Rmath.h>

#define I 0

double manage_time(double dStartTime, double *y) {
  dStartTime = fround(dStartTime,0);
  if (dStartTime == 0) {
    y[I] = 5;
    return 50;
  } else if (dStartTime == 50) {
    y[I] = 10;
  }

  return 200;
}

// End of file time_protocol.c
```

Compile the file:

- R CMD SHLIB time_protocol.c (on Linux/Unix or Windows)

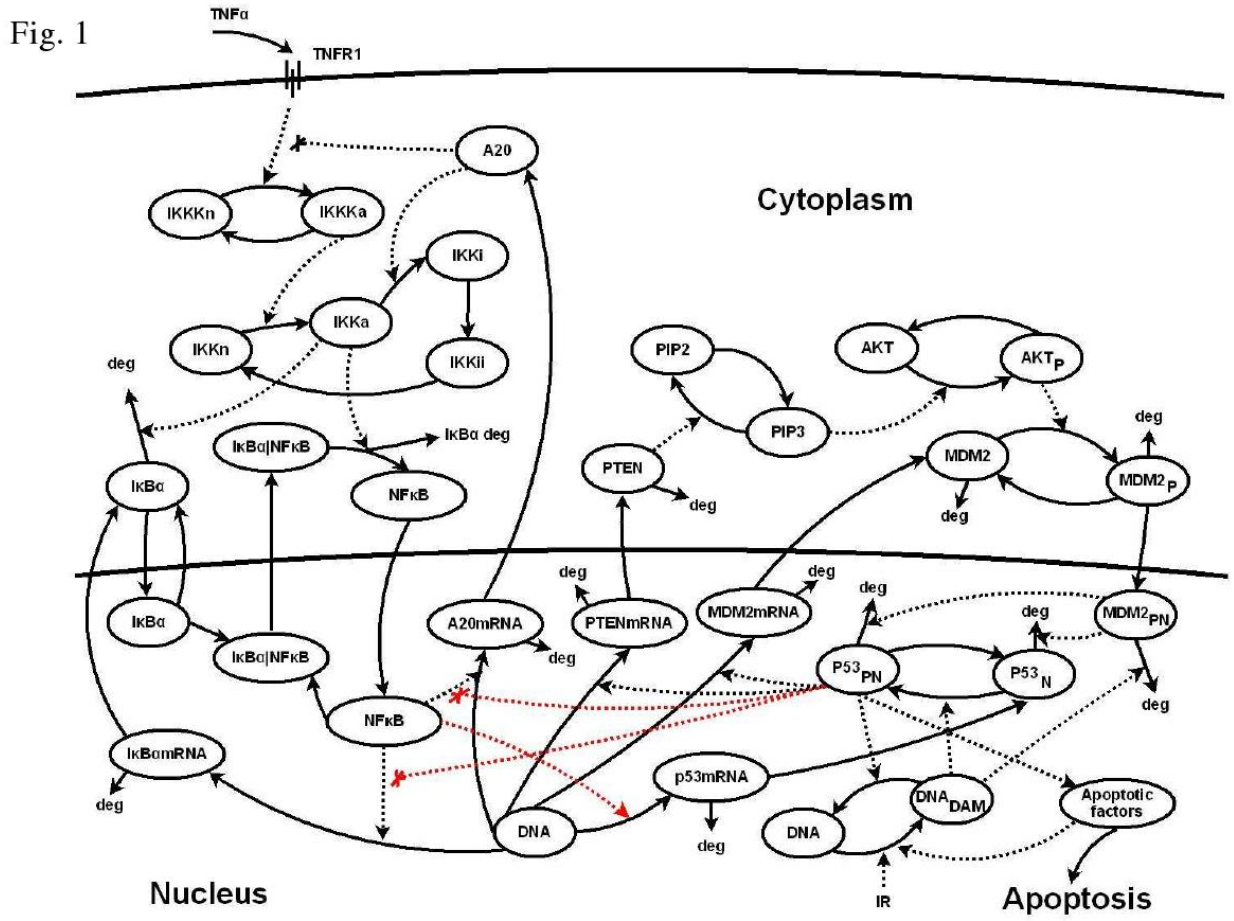
The R counterpart follows:

```
## Load the compiled code
dyn.load("time_protocol.so") ## on Linux/Unix
```

```
dyn.load("time_protocol.dll") ## on Windows

T <- getNativeSymbolInfo("manage_time", PACKAGE="time_protocol")$address
```

# 4 Example 2: Cross-talk between P53 and NFκB

The following example is based on a cross-talk model between the P53 and NFκB modules [5].




The model definition is divided into several files, that have to be on the same directory. The first is a C file that needs to be compiled. The rest are R files. The file main.R is the one that needs to be loaded into R and run. It will source the rest of the files. Complete code at the end.

## 4.1 How to Run the Model in Unix/Linux

To run the model in Unix/Linux, put all the file in the same folder. It can be convenient to write the following small script in bash (called here run) that has to be executable. Then, in the folder, type

```
./run <Enter>
```

### 4.1.1 run

```
#!/bin/bash

R CMD SHLIB protocols_propensities.c
R -q --vanilla < main.R > output.txt &

exit 0
```

## 4.2 How to Run the Model in Windows

It is possible to run it in Windows, but there will be needed some modifications, and the installation of Rtools (http://www.murdoch-sutherland.com/Rtools/) to be able to compile the C files. Also environment variables

need to be set (there is a pdf file from P. Rossi with detailed information on how to compile packages for Windows, with explanation on how to set the environmental variables. Google "Making R Packages Under Windows: A Tutorial"). To avoid these issues, the propensities may be defined in R (with the corresponding speed degradation due to calling interpreted code). Also the time function can be defined in R. As it is only called 5 times per run, the speed degradation in this case will be negligible.

## 4.3 Running the Model

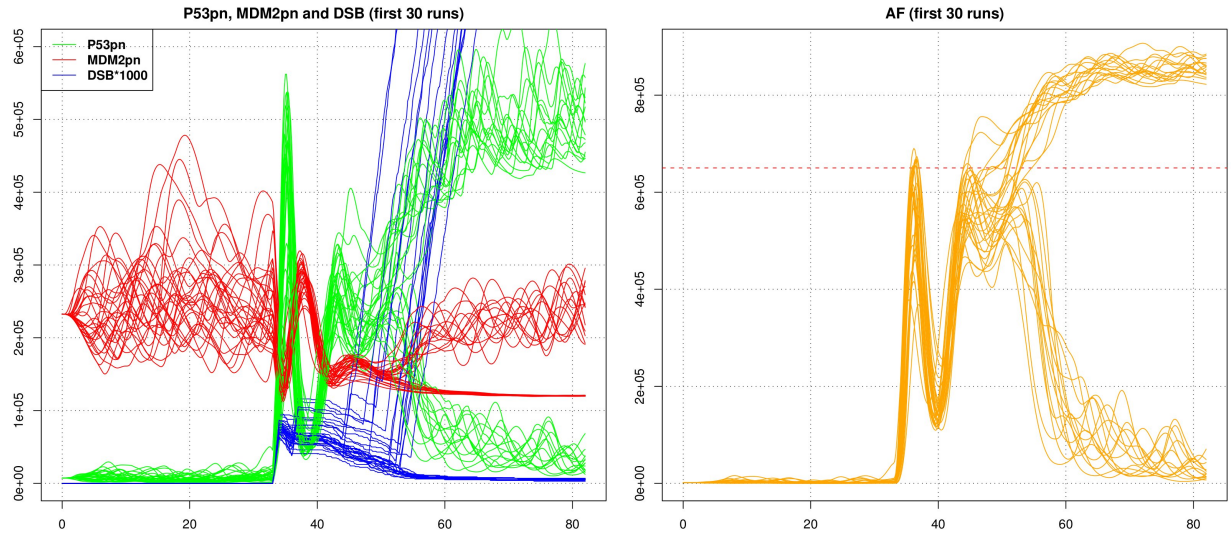
The model was run on a Dell XPS M1530 Laptop, Intel(R) Core(TM)2 Duo CPU T9500 @ 2.60GHz processor with 8GB (2GB required) and 500GB HD, running Gentoo Linux x86_64, 2.6.28-gentoo-r4 SMP PREEMPT kernel. Part of the output of the function, showing diagnostic information, was:

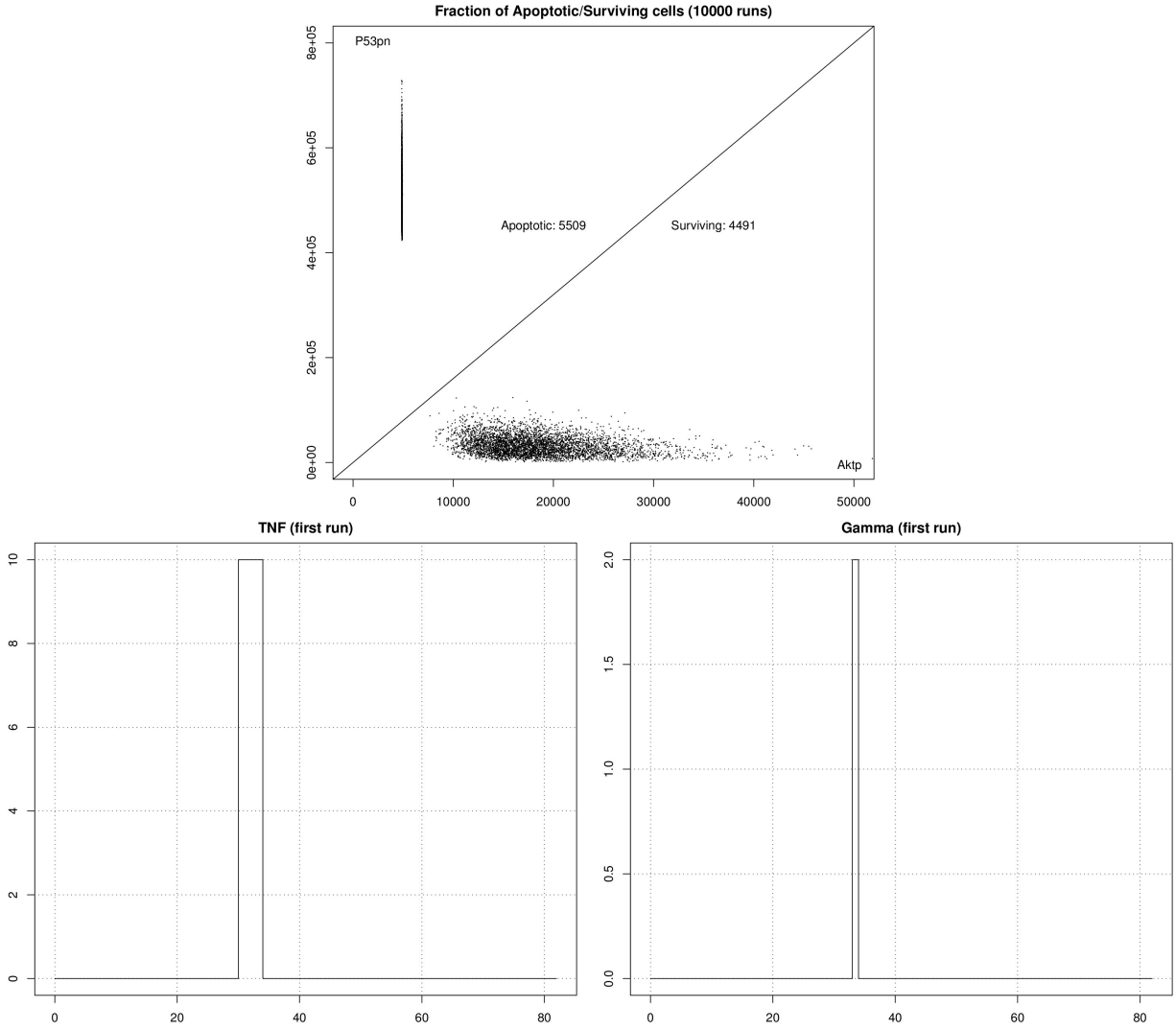
```
1 .... 3605 7214 7214 9.844409e-10 124081 119
2 .... 3859 7722 14936 9.854598e-10 123882 131
3 .... 8305 16614 31550 9.988635e-10 139470 144
4 .... 8426 16856 48406 9.946425e-10 138564 151
5 .... 8999 18002 66408 9.993794e-10 137894 124
.....
9995 .... 8750 17504 125648052 9.958226e-10 139593 161
9996 .... 8112 16228 125664280 9.940626e-10 139754 164
9997 .... 3334 6672 125670952 9.967196e-10 119067 136
9998 .... 3969 7942 125678894 9.959512e-10 125431 164
9999 .... 8852 17708 125696602 9.914101e-10 140775 122
10000 .... 7954 15912 125712514 9.856424e-10 140073 139
Elapsed CPU time: 12416.140625
```

The complete run took 3.45 hours, or about 1.24 seconds per run.

## 4.4 Plots

### 4.4.1 P53, MDM2pn, DSB, AF, Fraction of Apoptotic/Surviving Cells, TNF, and Gamma

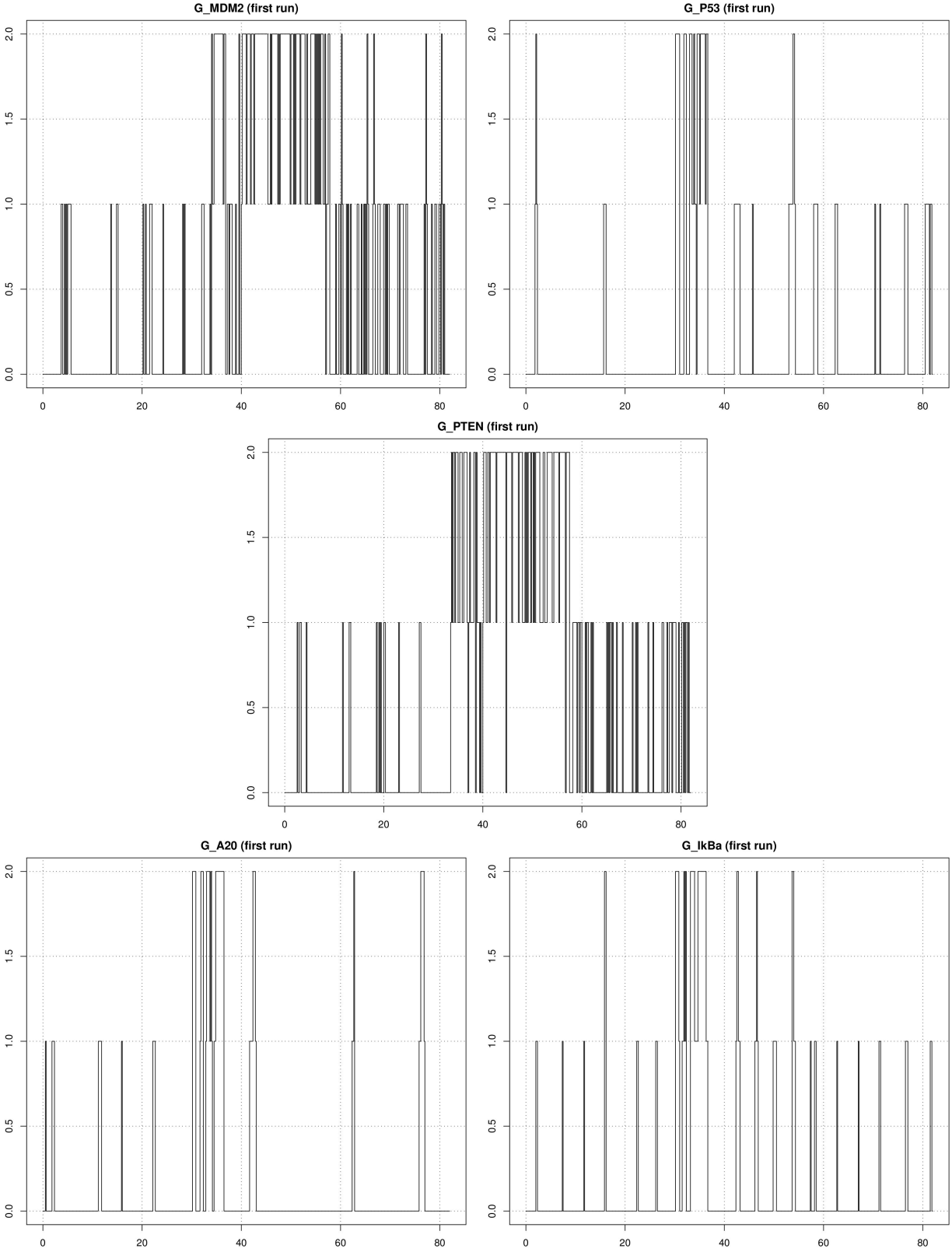




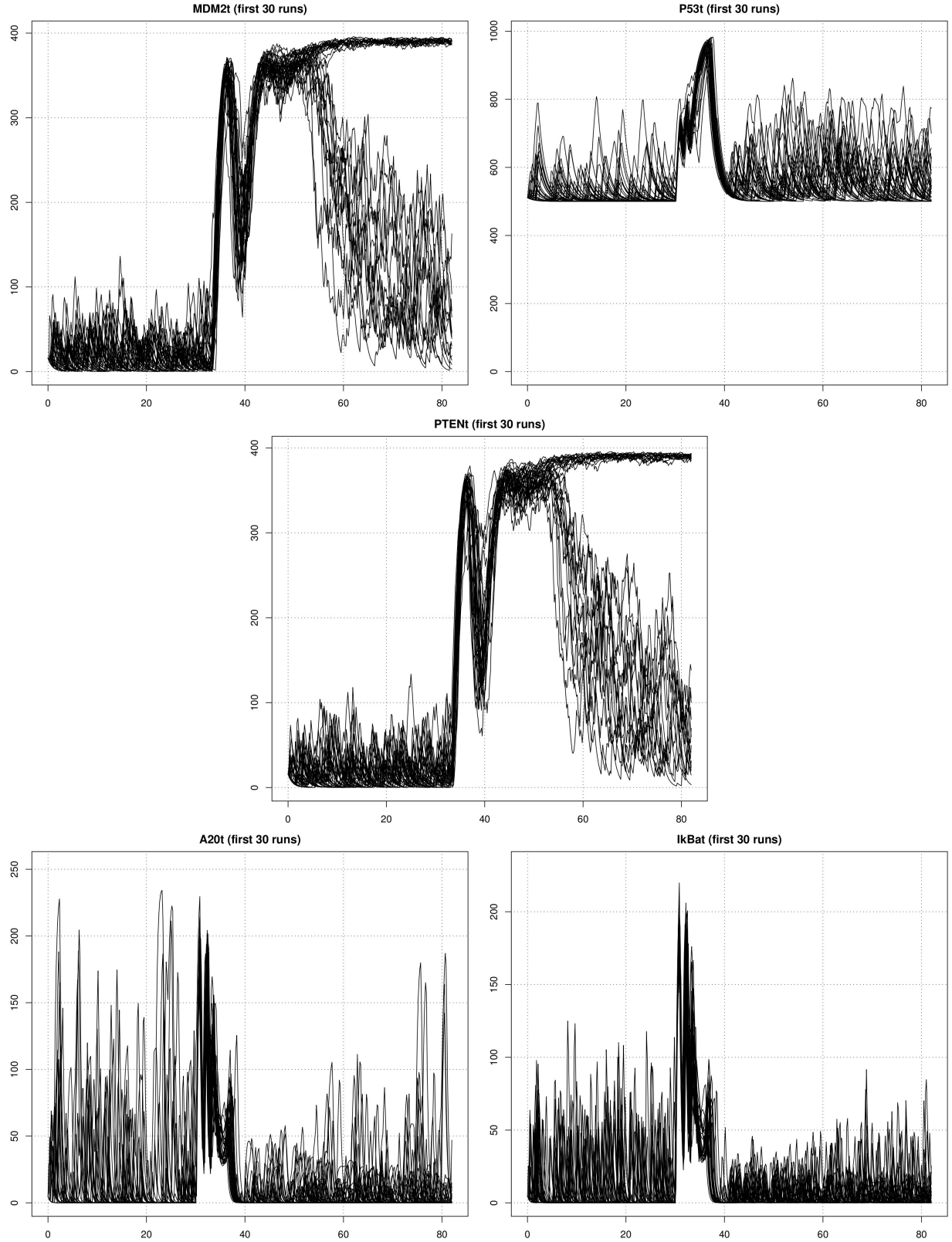

Note: TNF and Gamma are controlled only by the protocol (file protocols_propensities.c)

### 4.4.2 Genes

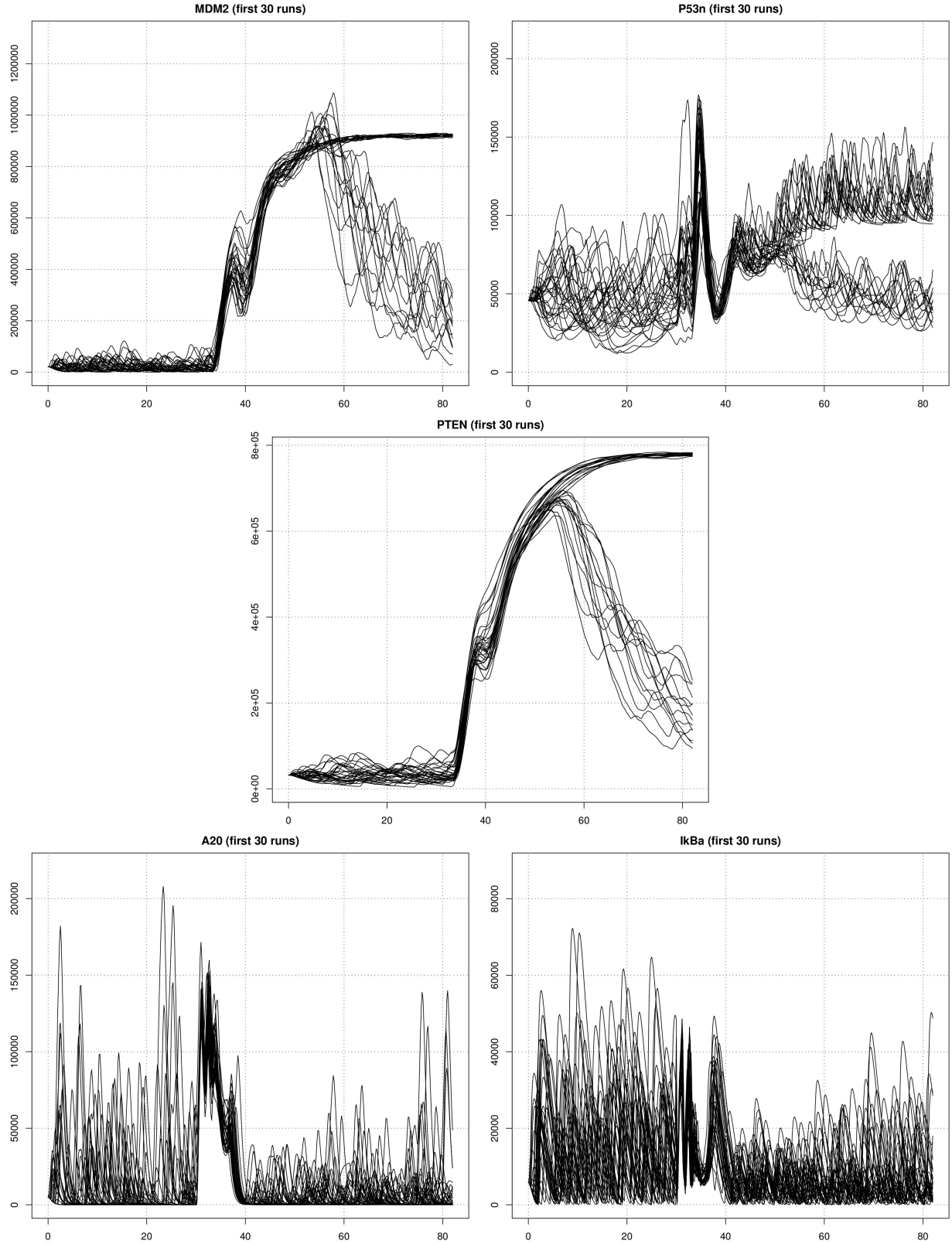
Only the first run is plotted, to avoid cluttering



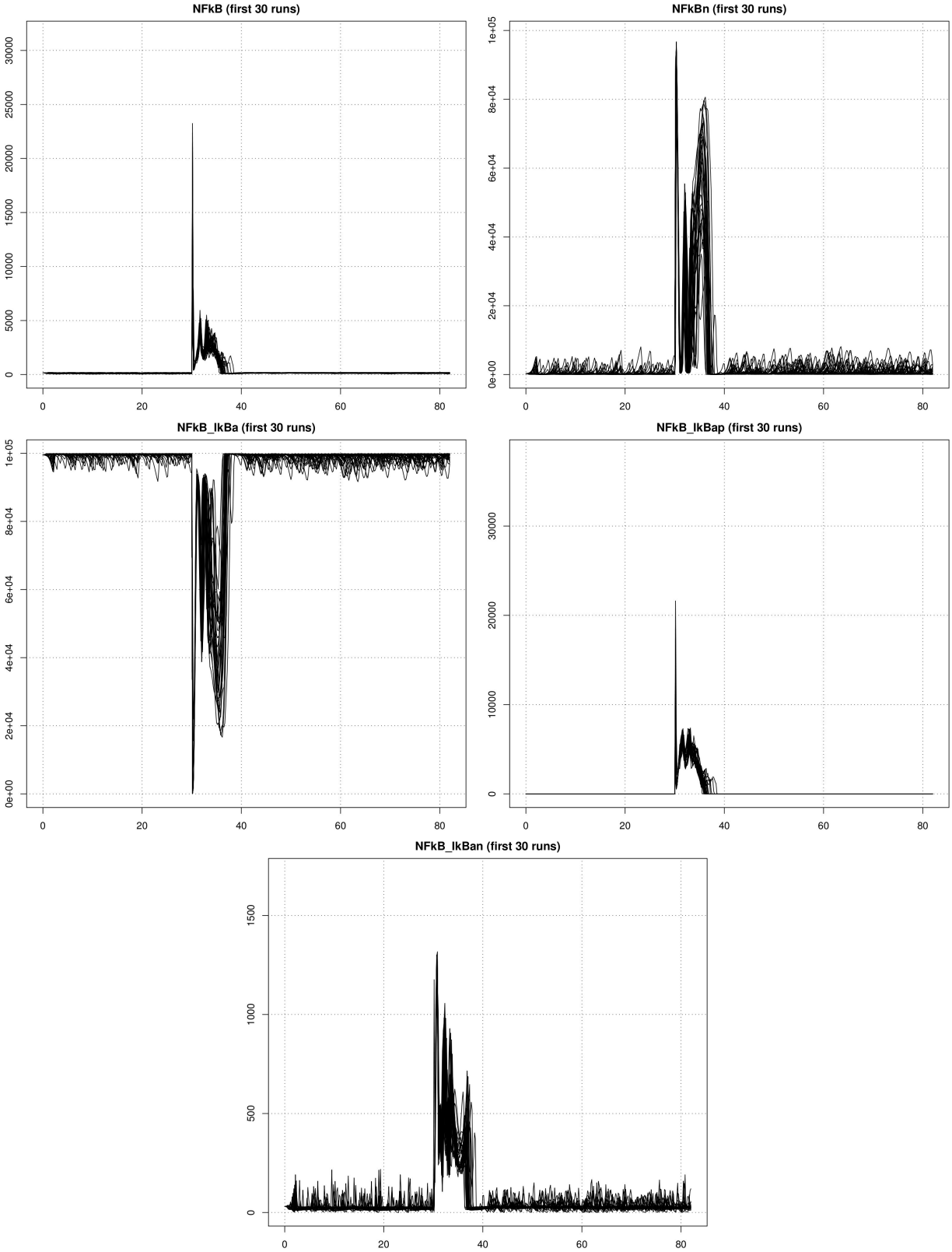
### 4.4.3 mRNAs


MDM2t (first 30 runs)
P53t (first 30 runs)
PTENt (first 30 runs)
A20t (first 30 runs)
IkBat (first 30 runs)

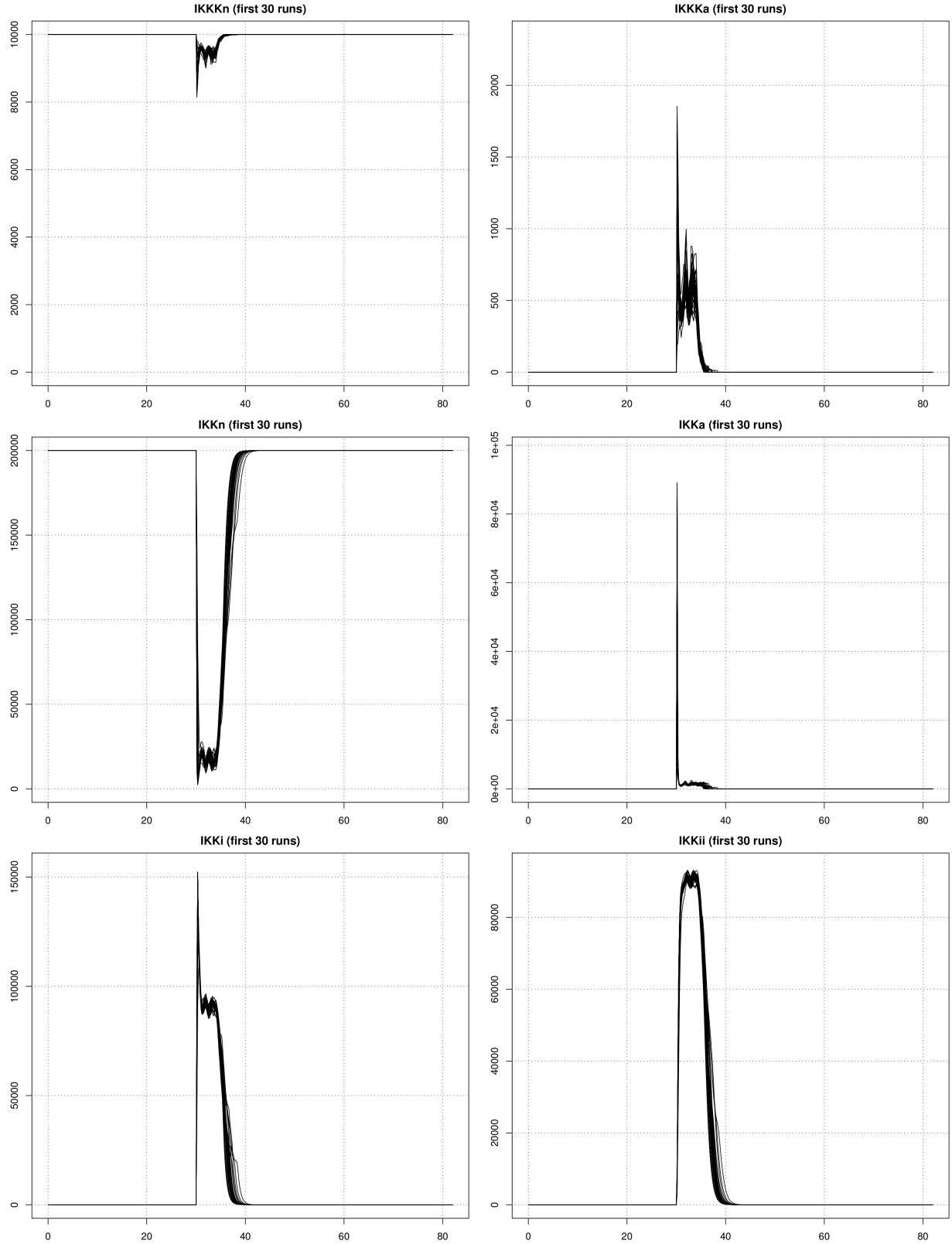
### 4.4.4 Proteins



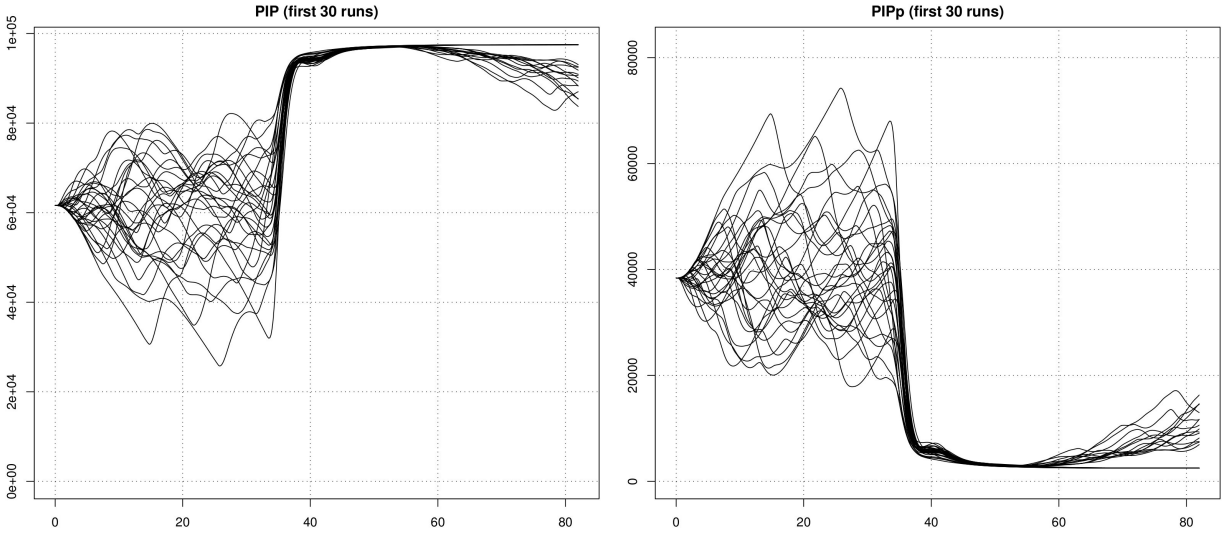
### 4.4.5 NFκB alone and bound to IκBα



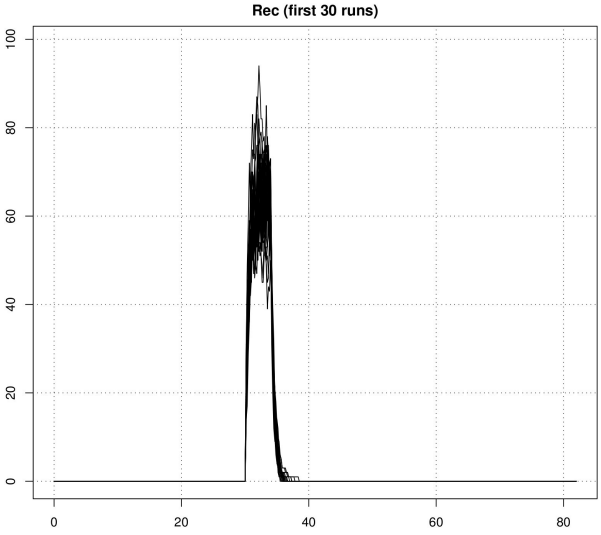
### 4.4.6 IKKK and IKK



### 4.4.7 PIP




### 4.4.8 Rec



# 5 References

## References

[1] Gibson, M. and J. Bruck: 2000, 'Efficient exact stochastic simulation of chemical systems with many species and many channels'. *Journal of Physical Chemistry A* **104**(9), 1876–1889.

[2] Harris, L. and P. Clancy: 2006, 'A "partitioned leaping" approach for multiscale modeling of chemical reaction dynamics'. *J. Chem. Phys.* **125**, 144107.

[3] Haseltine, E. L. and J. B. Rawlings: 2002, 'Approximate simulation of coupled fast and slow reactions for stochastic chemical kinetics'. *The Journal of Chemical Physics* **117**(15), 6959–6969.

[4] Paszek, P.: 2007, 'Modeling stochasticity in gene regulation: characterization in the terms of the underlying distribution function.'. *Bull Math Biol* **69**(5), 1567–601.

[5] Puszynski, K., R. Bertolusso, and T. Lipniacki: 2009, 'Crosstalk between p53 and nuclear factor-$\kappa$B systems: Pro- and anti-apoptotic functions of NF-$\kappa$B'. *IET Systems Biology* **3**(5), 356–367.

[6] Slepoy, A., A. P. Thompson, and S. J. Plimpton: 2008, 'A constant-time kinetic Monte Carlo algorithm for simulation of large biochemical reaction networks.'. *J Chem Phys* **128**(20), 205101.

# 6 Complete Code

## 6.1 First Example

```
    ########################################################
###### First approach, minimum requirements to run a model ######
    ########################################################
library(bioPN)
model <- list(pre = matrix(
                c(1,0,0,0, 0,1,0,0, 0,1,0,0,
                  0,0,1,0, 0,0,1,0, 0,0,0,1), ncol=4,byrow=T),
              post =  matrix(
                c(0,1,0,0, 1,0,0,0, 0,1,1,0,
                  0,0,0,0, 0,0,1,1, 0,0,0,0), ncol=4,byrow=T),
              h = list(c=3, b=2, H=10, 1, K=6, r=0.25),
              slow <- c(1,1,0,0,0,0),
              M = c(1,0,0,0))
timep <- 200
delta <- 1

## Completely deterministic run
Sim <- RungeKuttaDormandPrince45(model, timep, delta)

runs <- 10
## Completely stochastic run
set.seed(19761111)  ## Set a seed
Sim <- GillespieOptimDirect(model, timep, delta, runs)

## Hybrid run
set.seed(19761111)  ## Set a seed
Sim <- HaseltineRawlings(model, timep, delta, runs)

    ########################################################
###### Second approach, more suitable for large models ######
    ########################################################
library(bioPN)

timep <- 200
delta <- 1
ect <- 1e-9

place <- c( "I", "A", "mRNA", "protein")

transition <- c("gene_activation", "gene_inactivation",
                "transcription", "mRNA_degradation",
                "translation", "protein_degradation")

places <- length(place)
for (n in 1:places) {
  assign(place[n], n)
}
transitions <- length(transition)
for (n in 1:transitions) {
  assign(transition[n], n)
}
pre <- post <- array(0, dim <- c(transitions, places))
h <- list()

M <- rep(0, places)
M[I] <- 1
```

```
## Constants for Prokariotes
H <- 10
K <- 6
r <- 0.25
c <- 3
b <- 2

## Gene Activation: I -> A   (c)
i <- gene_activation
h[[i]] <- c
pre[i, I] <- 1
post[i, A] <- 1

## Gene Inactivation: A -> I   (d)
i <- gene_inactivation
h[[i]] <- b
pre[i, A] <- 1
post[i, I] <- 1

## Transcription: A -> A + mRNA   (H)
i <- transcription
h[[i]] <- H
pre[i, A] <- 1
post[i, A] <- 1; post[i, mRNA] <- 1

## mRNA Degradation: mRNA -> 0   (1)
i <- mRNA_degradation
h[[i]] <- 1
pre[i, mRNA] <- 1

## Translation: mRNA -> mRNA + protein   (K)
i <- translation
h[[i]] <- K
pre[i, mRNA] <- 1
post[i, mRNA] <- 1; post[i, protein] <- 1

## Protein Degradation: mRNA -> 0   (r)
i <- protein_degradation
h[[i]] <- r
pre[i, protein] <- 1

model <- list(pre=pre, post=post, h=h, M=M,
              place=place, transition=transition)

##############################
## Completely Deterministic ##
##############################
Sim <- RungeKuttaDormandPrince45(model, timep, delta, ect)

## Note, it could also be done as follows
## model$slow <- rep(0, transitions)
## Sim <- HaseltineRawlings(model, timep, delta, runs = 1, ect)

mRNA.run <- Sim$run[[1]]$M[[mRNA]]
protein.run <- Sim$run[[1]]$M[[protein]]

## Theoretical results (red lines in following plots)
Mean.mRNA <- c/(c+b)*H
Mean.protein <- Mean.mRNA * K/r
```

```
par(mfrow=c(1,2))
par(mar=c(2, 4, 2, 1) + 0.1)
plot(Sim$dt, mRNA.run,type="l", ylab="Mean",main="mRNA")
legend(x="bottom", paste("Deterministic run"))
abline(h=Mean.mRNA,col="red", lwd=1)
plot(Sim$dt, protein.run,type="l", ylab="Mean",main="Protein")
legend(x="bottom", paste("Deterministic run"))
abline(h=Mean.protein,col="red", lwd=1)


############################
## Completely Stochastic ##
############################
runs <- 100
set.seed(19761111)
Sim <- GillespieOptimDirect(model, timep, delta, runs)
#Sim <- GibsonBruck(model, timep, delta, runs)
#Sim <- GillespieDirectGB(model, timep, delta, runs)
#Sim <- GillespieDirectCR(model, timep, delta, runs)

## Note, it could also be done as follows
## model$slow <- rep(1, transitions)
## Sim <- HaseltineRawlings(model, timep, delta, runs, ect)

mRNA.run <- sapply(Sim$run, function(run) {run$M[[mRNA]]})
protein.run <- sapply(Sim$run, function(run) {run$M[[protein]]})

## Histograms of protein at different time points.
par(mfrow=c(2,2))
par(mar=c(2, 4, 2.5, 1) + 0.1)
hist(protein.run[Sim$dt == 1,], main="Protein Distribution at t=1sec")
hist(protein.run[Sim$dt == 2,], main="Protein Distribution at t=2sec")
hist(protein.run[Sim$dt == 10,], main="Protein Distribution at t=10sec")
hist(protein.run[Sim$dt == 200,], main="Protein Distribution at t=200sec")

## Theoretical results (red lines in following plots)
Mean.mRNA <- c/(c+b)*H
Var.mRNA <- b/(c*(1+c+b))*Mean.mRNA^2 + Mean.mRNA
Mean.protein <- Mean.mRNA * K/r
Var.protein <- r*b*(1+c+b+r)/(c*(1+r)*(1+c+b)*(r+c+b))*Mean.protein^2 +
  r/(1+r)*Mean.protein^2/Mean.mRNA + Mean.protein

if (runs > 1 ) {
  par(mfrow=c(2,2))
} else {
  par(mfrow=c(1,2))
}
par(mar=c(2, 4, 2, 1) + 0.1)
plot(Sim$dt, apply(mRNA.run,1,function(tpt) {mean(tpt)}),type="l", ylab="Mean",main="mRNA")
legend(x="bottom", paste("Gene, mRNA and Protein Stochastic\nRuns :", runs))
abline(h=Mean.mRNA,col="red", lwd=1)
plot(Sim$dt, apply(protein.run,1,function(tpt) {mean(tpt)}),type="l", ylab="Mean",main="Protein")
legend(x="bottom", paste("Gene, mRNA and Protein Stochastic\nRuns :", runs))
abline(h=Mean.protein,col="red", lwd=1)
if (runs > 1 ) {
  par(mar=c(2, 4, 0, 1) + 0.1)
  plot(Sim$dt, apply(mRNA.run,1,function(tpt) {var(tpt)}),type="l", ylab="Var")
  abline(h=Var.mRNA,col="red", lwd=1)
  plot(Sim$dt, apply(protein.run,1,function(tpt) {var(tpt)}),type="l", ylab="Var")
  abline(h=Var.protein,col="red", lwd=1)
```

```
}


########################################################################
## Hybrid: mRNA and protein fast, gene activation/inactivation slow ##
########################################################################
slow <- rep(0, transitions)
slow[gene_activation] <- 1
slow[gene_inactivation] <- 1

model$slow <- slow

runs <- 100
set.seed(19761111)
Sim <- HaseltineRawlings(model, timep, delta, runs, ect)

mRNA.run <- sapply(Sim$run, function(run) {run$M[[mRNA]]})
protein.run <- sapply(Sim$run, function(run) {run$M[[protein]]})

Mean.mRNA <- c/(c+b)*H
Var.mRNA <- b/(c*(1+c+b))*Mean.mRNA^2
Mean.protein <- Mean.mRNA * K/r
Var.protein <- r*b*(1+c+b+r)/(c*(1+r)*(1+c+b)*(r+c+b))*Mean.protein^2

pdf(file="test.pdf", width = 12, height = 7)
if (runs > 1 ) {
  par(mfrow=c(2,2))
} else {
  par(mfrow=c(1,2))
}
par(mar=c(2, 4, 2, 1) + 0.1)
plot(Sim$dt, apply(mRNA.run,1,function(tpt) {mean(tpt)}),type="l", ylab="Mean",main="mRNA")
legend(x="bottom", paste("Only Gene Stochastic\nRuns :", runs))
abline(h=Mean.mRNA,col="red", lwd=1)
plot(Sim$dt, apply(protein.run,1,function(tpt) {mean(tpt)}),type="l", ylab="Mean",main="Protein")
legend(x="bottom", paste("Only Gene Stochastic\nRuns :", runs))
abline(h=Mean.protein,col="red", lwd=1)
if (runs > 1 ) {
  par(mar=c(2, 4, 0, 1) + 0.1)
  plot(Sim$dt, apply(mRNA.run,1,function(tpt) {var(tpt)}),type="l", ylab="Var")
  abline(h=Var.mRNA,col="red", lwd=1)
  plot(Sim$dt, apply(protein.run,1,function(tpt) {var(tpt)}),type="l", ylab="Var")
  abline(h=Var.protein,col="red", lwd=1)
}
dev.off()

########################################################################
## Hybrid: protein fast, mRNA and gene activation/inactivation slow ##
########################################################################
slow <- rep(1, transitions)
slow[translation] <- 0
slow[protein_degradation] <- 0
model$slow <- slow

runs <- 100
set.seed(19761111)
Sim <- HaseltineRawlings(model, timep, delta, runs, ect)

mRNA.run <- sapply(Sim$run, function(run) {run$M[[mRNA]]})
protein.run <- sapply(Sim$run, function(run) {run$M[[protein]]})
```

```
Mean.mRNA <- c/(c+b)*H
Var.mRNA <- b/(c*(1+c+b))*Mean.mRNA^2 + Mean.mRNA
Mean.protein <- Mean.mRNA * K/r
Var.protein <- r*b*(1+c+b+r)/(c*(1+r)*(1+c+b)*(r+c+b))*Mean.protein^2 +
  r/(1+r)*Mean.protein^2/Mean.mRNA

if (runs > 1 ) {
  par(mfrow=c(2,2))
} else {
  par(mfrow=c(1,2))
}
par(mar=c(2, 4, 2, 1) + 0.1)
plot(Sim$dt, apply(mRNA.run,1,function(tpt) {mean(tpt)}),type="l", ylab="Mean",main="mRNA")
legend(x="bottom", paste("Gene and mRNA Stochastic\nRuns :", runs))
abline(h=Mean.mRNA,col="red", lwd=1)
plot(Sim$dt, apply(protein.run,1,function(tpt) {mean(tpt)}),type="l", ylab="Mean",main="Protein")
legend(x="bottom", paste("Gene and mRNA Stochastic\nRuns :", runs))
abline(h=Mean.protein,col="red", lwd=1)
par(mar=c(2, 4, 0, 1) + 0.1)
if (runs > 1 ) {
  plot(Sim$dt, apply(mRNA.run,1,function(tpt) {var(tpt)}),type="l", ylab="Var")
  abline(h=Var.mRNA,col="red", lwd=1)
  plot(Sim$dt, apply(protein.run,1,function(tpt) {var(tpt)}),type="l", ylab="Var")
  abline(h=Var.protein,col="red", lwd=1)
}
```

## 6.2 Second Example

## 6.3 C Files

### 6.3.1 protocols_propensities.c

```
/* compile with R CMD SHLIB protocols_propensities.c or (on Windows) Rcmd SHLIB protocols_propensities.c */

#include <R.h>
#include <Rmath.h>

#define Gy 2.

typedef enum {
  G_PTEN, Gi_PTEN, PTENt, PTEN,
  PIP, PIPp,
  Akt, Aktp,
  G_MDM2, Gi_MDM2, MDM2t, MDM2, MDM2p, MDM2pn,
  G_P53, Gi_P53, P53t, P53n, P53pn,
  IKKKn, IKKKa,
  IKKn, IKKa, IKKi, IKKii,
  G_A20, Gi_A20, A20t, A20,
  G_IkBa, Gi_IkBa, IkBat, IkBa, IkBap, IkBan,
  NFkB, NFkBn,
  NFkB_IkBa, NFkB_IkBap, NFkB_IkBan,
  TNF, Gamma,
  Rec, Reci,
  DSB,
  AF
} enum_places;


/***********************************
*             Protocol
*/
double TNF_before_IR(double dStartTime, double *y) {
  dStartTime = fround(dStartTime,0);
  if (dStartTime == 0) {
    y[Gamma] = 0;
    y[TNF] = 0;
    return 30*3600;
  } else if (dStartTime == 30*3600) {
    y[TNF] = 10;
    return 33*3600;
  } else if (dStartTime == 33*3600) {
    y[Gamma] = Gy;
    return 34*3600;
  } else if (dStartTime == 34*3600) {
    y[Gamma] = 0;
    y[TNF] = 0;
  }

  return 82*3600;
}


/***********************************
*             Propensities
*/
#define d0 3e-5
#define d1 1.5e-4
```

```
#define h0 7.

double MDM2_degr(double time, double *y) {
  double DSB_sq = y[DSB]*y[DSB];
  return (d0 + d1 * DSB_sq / (h0*h0 + DSB_sq )) * y[MDM2];
}

double MDM2p_degr(double time, double *y) {
  double DSB_sq = y[DSB]*y[DSB];
  return (d0 + d1 * DSB_sq / (h0*h0 + DSB_sq )) * y[MDM2p];
}

double MDM2pn_degr(double time, double *y) {
  double DSB_sq = y[DSB]*y[DSB];
  return (d0 + d1 * DSB_sq / (h0*h0 + DSB_sq )) * y[MDM2pn];
}

#define a0 1e-4
#define a1 1e-3

double P53n_phos(double time, double *y) {
  double DSB_sq = y[DSB]*y[DSB];
  return (a0 + a1 * DSB_sq / (h0*h0 + DSB_sq )) * y[P53n];
}

#define d3 1e-4
#define d4 1e-13

double P53n_degr(double time, double *y) {
  return (d3 + d4*y[MDM2pn]*y[MDM2pn]) * y[P53n];
}

#define d5 1e-4
#define d6 1e-14

double P53pn_degr(double time, double *y) {
  return (d5 + d6*y[MDM2pn]*y[MDM2pn]) * y[P53pn];
}

#define ka 1e-4
#define ka20 1e4

double IKKKn_activ(double time, double *y) {
  //  return ka * ka20 / (ka20 + y[A20]) * y[Rec] * y[IKKKn];
  // As ka * ka20 == 1...
  return y[Rec] * y[IKKKn] / (ka20 + y[A20]);
}

#define k2 1e4
#define k3 3e-3

double IKKa_inactiv(double time, double *y) {
  return k3 * y[IKKa] * (k2 + y[A20]) / k2;
}

#define nc1 1e-1
#define h101 6e4

double A20t_transcr(double time, double *y) {
  return nc1 * h101 / (h101 + y[P53pn]) * y[G_A20];
```

```
}

double IkBat_transcr(double time, double *y) {
  return nc1 * h101 / (h101 + y[P53pn]) * y[G_IkBa];
}

#define p1 200
#define q3 2.5e-11
#define q4 1

double AF_synth(double time, double *y) {
  double tmp = q3 * y[P53pn]*y[P53pn];
  return p1 * tmp/(q4 + tmp);
}

#define q0 1e-4
#define q1 5e-13

double Gi_MDM2_activ(double time, double *y) {
  return (q0 + q1 * y[P53pn]*y[P53pn]) * y[Gi_MDM2];
}

double Gi_PTEN_activ(double time, double *y) {
  return (q0 + q1 * y[P53pn]*y[P53pn]) * y[Gi_PTEN];
}

#define a6 1e-2

#define Th 0.65

#define d9_over_p1 1e-6

double DSB_increase_AF(double time, double *y) {
  double sgn;
  double tmp = y[AF] * d9_over_p1 - Th;
  if (tmp > 0)
    sgn = 1;
  else if (tmp < 0)
    sgn = -1;
  else
    sgn = 0;

  return a6 * (sgn + 1);
}

#define Nsat 3.
#define dRep 2e-14

double DSB_decrease(double time, double *y) {
  return dRep * y[P53pn]*y[P53pn] * y[DSB]/(y[DSB] + Nsat);
}
```

## 6.4 R Files

### 6.4.1 constants.R

```
AKTtot <- 2e5
PIPtot <- 1e5
PTENmax <- 4e5
```

### 6.4.2 places_transitions.R

```
place <- c(
           "G_PTEN", "Gi_PTEN", "PTENt", "PTEN",
           "PIP", "PIPp",
           "Akt", "Aktp",
           "G_MDM2", "Gi_MDM2", "MDM2t", "MDM2", "MDM2p", "MDM2pn",
           "G_P53", "Gi_P53", "P53t", "P53n", "P53pn",
           "IKKKn", "IKKKa",
           "IKKn", "IKKa", "IKKi", "IKKii",
           "G_A20", "Gi_A20", "A20t", "A20",
           "G_IkBa", "Gi_IkBa", "IkBat", "IkBa", "IkBap", "IkBan",
           "NFkB", "NFkBn",
           "NFkB_IkBa", "NFkB_IkBap", "NFkB_IkBan",
           "TNF", "Gamma",
           "Rec", "Reci",
           "DSB",
           "AF"
           )

transition <- c(
                "PTEN_synth", "PTEN_degr",
                "PIPp_dephos", "PIP_phos",
                "Akt_phos", "Aktp_dephos",
                "MDM2_synth", "MDM2p_dephos", "MDM2_phos", "MDM2_degr",
                "MDM2p_import", "MDM2p_degr", "MDM2pn_degr", # "MDM2pn_export",
                "P53n_synth", "P53n_phos", "P53n_degr", "P53pn_degr", # "P53pn_dephos",
                "MDM2t_transcr", "MDM2t_degr",
                "PTENt_transcr", "PTENt_degr",
                "P53t_ind_transcr", "P53t_transcr", "P53t_degr",
                "IKKKn_activ", "IKKKa_inactiv",
                "IKKn_activ", "IKKa_inactiv", "IKKi_to_IKKii", "IKKii_to_IKKn",
                "IkBa_phos", "IkBap_degr",
                "NFkB_IkBa_phos", "NFkB_IkBap_to_NFkB", "NFkB_IkBa_to_NFkB",
                "NFkB_and_IkBa_to_NFkB_IkBa",
                "NFkB_import",
                "NFkBn_and_IkBan_to_NFkB_IkBan",
                "A20_synth", "A20_degr",
                "A20t_transcr", "A20t_degr",
                "IkBa_synth", "IkBa_degr", "IkBa_import", "IkBan_export",
                "IkBat_transcr", "IkBat_degr",
                "NFkB_IkBan_export",
                "AF_synth", "AF_degr",
                "Reci_activ", "Rec_inactiv",
                "Gi_A20_activ", "G_A20_inactiv",
                "Gi_IkBa_activ", "G_IkBa_inactiv",
                "Gi_P53_activ", "G_P53_inactiv",
                "Gi_MDM2_activ", "G_MDM2_inactiv",
                "Gi_PTEN_activ", "G_PTEN_inactiv",
                "DSB_increase_Gamma", "DSB_increase_AF", "DSB_decrease"
                )

places <- length(place)
for (n in 1:places) {
  assign(place[n], n)
}
transitions <- length(transition)
for (n in 1:transitions) {
  assign(transition[n], n)
}
```

```
pre <- post <- array(0, dim <- c(transitions, places))
h <- list()
```

### 6.4.3 model.R

```
#### (1) (Cytoplasmic) PTEN

## PTEN sysnthesis: PTENt -> PTENt + PTEN   (t1)
i <- PTEN_synth
h[[i]] <- 1e-1 # t1
pre[i, PTENt] <- 1
post[i, PTENt] <- 1; post[i, PTEN] <- 1

## PTEN degradation: PTEN -> 0   (d2)
i <- PTEN_degr
h[[i]] <- 5e-5 # d2
pre[i, PTEN] <- 1


#### (2) Active form of PIP

## PIPp dephosphorylation: PTEN + PIPp -> PTEN + PIP   (c0)
i <- PIPp_dephos
h[[i]] <- 1e-3/PTENmax # c0
pre[i, PTEN] <- 1; pre[i, PIPp] <- 1
post[i, PTEN] <- 1; post[i, PIP] <- 1

## PIP phosphorylation: PIP -> PIPp   (a2)
i <- PIP_phos
h[[i]] <- 5e-5 # a2
pre[i, PIP] <- 1
post[i, PIPp] <- 1


#### (3) Active Akt

## Akt phosphorylation: PIPp + Akt -> PIPp + Aktp   (a3)
i <- Akt_phos
h[[i]] <- 2e-4/PIPtot # a3
pre[i, PIPp] <- 1; pre[i, Akt] <- 1
post[i, PIPp] <- 1; post[i, Aktp] <- 1

## Aktp dephosphorylation: Aktp -> Akt   (c1)
i <- Aktp_dephos
h[[i]] <- 2e-4 # c1
pre[i, Aktp] <- 1
post[i, Akt] <- 1


#### (4) Cytoplasmic Mdm2

## MDM2 synthesis: MDM2t -> MDM2t + MDM2   (t0)
i <- MDM2_synth
h[[i]] <- 5e-1 # t0
pre[i, MDM2t] <- 1
post[i, MDM2t] <- 1; post[i, MDM2] <- 1

## MDM2p dephosphorylation: MDM2p -> MDM2   (c2)
i <- MDM2p_dephos
h[[i]] <- 1e-4 # c2
pre[i, MDM2p] <- 1
```

```
post[i, MDM2] <- 1

## MDM2 phosphorylation: Aktp + MDM2 -> Aktp + MDM2p   (a4)
i <- MDM2_phos
h[[i]] <- 1.5e-3/AKTtot # a4
pre[i, Aktp] <- 1; pre[i, MDM2] <- 1
post[i, Aktp] <- 1; post[i, MDM2p] <- 1

## MDM2 degradation: DSB + MDM2 -> DSB   ( (d0 + d1*DSB^2/(h0^2+DSB^2))*MDM2 )
i <- MDM2_degr
if (onlyR) {
  d0 <- 3e-5
  d1 <- 1.5e-4
  h0 <- 7.

  h[[i]] <- function() {(d0 + d1*y[DSB]^2/(h0^2+y[DSB]^2))*y[MDM2]}
} else {
  h[[i]] <- getNativeSymbolInfo("MDM2_degr", PACKAGE="protocols_propensities")$address
}
pre[i, DSB] <- 1; pre[i, MDM2] <- 1
post[i, DSB] <- 1


#### (5) Cytoplasmic phosphorylated Mdm2

## MDM2p import: MDM2p -> MDM2pn   (i0)
i <- MDM2p_import
h[[i]] <- 5e-4 # i0
pre[i, MDM2p] <- 1
post[i, MDM2pn] <- 1

## MDM2pn export: MDM2pn -> MDM2p   (e0)
##i -> MDM2pn_export
##h[[i]] <- 0e-4 # e0
##pre[i, MDM2pn] <- 1
##post[i, MDM2p] <- 1

## MDM2p degradation: DSB + MDM2p -> DSB   ( (d0 + d1*DSB^2/(h0^2+DSB^2))*MDM2p )
i <- MDM2p_degr
if (onlyR) {
  h[[i]] <- function() {(d0 + d1*y[DSB]^2/(h0^2+y[DSB]^2))*y[MDM2p]}
} else {
  h[[i]] <- getNativeSymbolInfo("MDM2p_degr", PACKAGE="protocols_propensities")$address
}
pre[i, DSB] <- 1; pre[i, MDM2p] <- 1
post[i, DSB] <- 1


#### (6) Nuclear phosphorylated Mdm2

## MDM2pn degradation: DSB + MDM2pn -> DSB   ( (d0 + d1*DSB^2/(h0^2+DSB^2))*MDM2pn )
i <- MDM2pn_degr
if (onlyR) {
  h[[i]] <- function() {(d0 + d1*y[DSB]^2/(h0^2+y[DSB]^2))*y[MDM2pn]}
} else {
  h[[i]] <- getNativeSymbolInfo("MDM2pn_degr", PACKAGE="protocols_propensities")$address
}
pre[i, DSB] <- 1; pre[i, MDM2pn] <- 1
post[i, DSB] <- 1
```

```
#### (7) Inactive (nuclear) p53
#### (8) Active (nuclear) p53

## P53n synthesis: P53t -> P53t + P53n   (t101 t2)
i <- P53n_synth
h[[i]] <- 5e-1 # t101
pre[i, P53t] <- 1
post[i, P53t] <- 1; post[i, P53n] <- 1

## P53n phosphorylation: DSB + P53n -> DSB + P53pn   ( (a0 + a1*DSB^2/(h0^2+DSB^2))*P53n )
i <- P53n_phos
if (onlyR) {
  a0 <- 1e-4
  a1 <- 1e-3

  h[[i]] <- function() {(a0 + a1*y[DSB]^2/(h0^2+y[DSB]^2))*y[P53n]}
} else {
  h[[i]] <- getNativeSymbolInfo("P53n_phos", PACKAGE="protocols_propensities")$address
}
pre[i, DSB] <- 1; pre[i, P53n] <- 1
post[i, DSB] <- 1; post[i, P53pn] <- 1

## P53n degradation: MDM2pn + P53n -> MDM2pn   ( (d3 + d4*MDM2pn^2)*P53n )
i <- P53n_degr
if (onlyR) {
  d3 <- 1e-4
  d4 <- 1e-13

  h[[i]] <- function() {(d3 + d4*y[MDM2pn]^2)*y[P53n]}
} else {
  h[[i]] <- getNativeSymbolInfo("P53n_degr", PACKAGE="protocols_propensities")$address
}
pre[i, MDM2pn] <- 1; pre[i, P53n] <- 1
post[i, MDM2pn] <- 1;

## P53pn dephosphorylation: P53pn -> P53n   (c3)
##i <- P53pn_dephos
##h[[i]] <- 0 # c3
##pre[i, P53pn] <- 1
##post[i, P53n] <- 1

## P53pn degradation: MDM2pn + P53pn -> MDM2pn   ( (d5 + d6*MDM2pn^2)*P53pn )
i <- P53pn_degr
if (onlyR) {
  d5 <- 1e-4
  d6 <- 1e-14

  h[[i]] <- function() {(d5 + d6*y[MDM2pn]^2)*y[P53pn]}
} else {
  h[[i]] <- getNativeSymbolInfo("P53pn_degr", PACKAGE="protocols_propensities")$address
}
pre[i, MDM2pn] <- 1; pre[i, P53pn] <- 1
post[i, MDM2pn] <- 1;


#### (9) Mdm2 transcript

## MDM2t transcription: G_MDM2 -> G_MDM2 + MDM2t   (s0)
i <- MDM2t_transcr
```

```
h[[i]] <- 6e-2 # s0
pre[i, G_MDM2] <- 1
post[i, G_MDM2] <- 1; post[i, MDM2t] <- 1

## MDM2t degradation: MDM2t -> 0  (d7)
i <- MDM2t_degr
h[[i]] <- 3e-4 # d7
pre[i,  MDM2t] <- 1


#### (10) PTEN transcript

## PTENt transcription: G_PTEN -> G_PTEN + PTENt   (s1)
i <- PTENt_transcr
h[[i]] <- 6e-2 # s1
pre[i, G_PTEN] <- 1
post[i, G_PTEN] <- 1; post[i, PTENt] <- 1

## PTENt degradation: PTENt -> 0  (d8)
i <- PTENt_degr
h[[i]] <- 3e-4 # d8
pre[i,  PTENt] <- 1


#### (11) P53 transcript

## P53t independent transcription: 0 -> P53t   (2*s3)
##    s3=0.05;        %0.1 p53 mRNA synthesis independend from NFkB,
i <- P53t_ind_transcr
h[[i]] <- 2 * 5e-2 # 2*s3
post[i, P53t] <- 1

## P53t transcription: G_P53 -> G_P53 + P53t   (s2)
##    s2=0.05;        %0.1 p53 mRNA synthesis NFkB inducible,
i <- P53t_transcr
h[[i]] <- 5e-2 # s2
pre[i, G_P53] <- 1
post[i, G_P53] <- 1; post[i, P53t] <- 1

## P53t degradation: P53t -> 0  (d10)
##    d10=2*10^-4;    %(5*10^-4 JTB) (7.5*10^-4 IET); %p53 transcript degradation,
i <- P53t_degr
h[[i]] <- 2e-4 # d10
pre[i, P53t] <- 1


#### (12) IKKK in active state

## IKKKn activation: Rec + A20 + IKKKn -> Rec + A20 + IKKKa
## ( ka*ka20/(ka20+A20)*Rec*IKKKn )
i <- IKKKn_activ
if (onlyR) {
  ka <- 1e-4
  ka20 <- 1e4

  h[[i]] <- function() {ka*ka20/(ka20+y[A20])*y[Rec]*y[IKKKn]}
} else {
  h[[i]] <- getNativeSymbolInfo("IKKKn_activ", PACKAGE="protocols_propensities")$address
}
pre[i, Rec] <- 1; pre[i, A20] <- 1; pre[i, IKKKn] <- 1
```

```
post[i, Rec] <- 1; post[i, A20] <- 1; post[i, IKKKa] <- 1

## IKKKa inactivation: IKKKa -> IKKKn   (ki)
i <- IKKKa_inactiv
h[[i]] <- 1e-2 # ki
pre[i, IKKKa] <- 1
post[i, IKKKn] <- 1


#### (13) IKK in the natural state
#### (14) IKK in the active state
#### (15) IKK in the inactive state

## IKKn activation: IKKKa + IKKn -> IKKKa + IKKa   (k1)
i <- IKKn_activ
h[[i]] <- 5e-6  # k1
pre[i, IKKKa] <- 1; pre[i, IKKn] <- 1
post[i, IKKKa] <- 1; post[i, IKKa] <- 1

## IKKa inactivation: A20 + IKKa -> A20 + IKKi   ( k3*IKKa*(k2 + A20)/k2 )
i <- IKKa_inactiv
if (onlyR) {
  k2 <- 1e4
  k3 <- 3e-3

  h[[i]] <- function() {k3*y[IKKa]*(k2 + y[A20])/k2}
} else {
  h[[i]] <- getNativeSymbolInfo("IKKa_inactiv", PACKAGE="protocols_propensities")$address
}
pre[i, A20] <- 1; pre[i, IKKa] <- 1
post[i, A20] <- 1; post[i, IKKi] <- 1

## IKKi to IKKi: IKKi -> IKKii   (k4)
i <- IKKi_to_IKKii
h[[i]] <- 5e-4  # k4
pre[i, IKKi] <- 1
post[i, IKKii] <- 1

## IKKii to IKKn: IKKii -> IKKn   (k4)
i <- IKKii_to_IKKn
h[[i]] <- 5e-4  # k4
pre[i, IKKii] <- 1
post[i, IKKn] <- 1


#### (16) Phospho-IkBa

## IkBa phosphorylation: IKKa + IkBa -> IKKa + IkBap  (na2)
i <- IkBa_phos
h[[i]] <- 1e-7  # na2
pre[i, IKKa] <- 1; pre[i, IkBa] <- 1
post[i, IKKa] <- 1; post[i, IkBap] <- 1

## IkBap degradation: IkBap -> 0  (ntp)
i <- IkBap_degr
h[[i]] <- 1e-2  # ntp
pre[i, IkBap] <- 1


#### (17) Phospho-IkBa complexed to NF-kB
```

```
## NfKb_IkBa phosphorylation: IKKa + NFkB_IkBa -> IKKa + NFkB_IkBap (na3)
i <- NFkB_IkBa_phos
h[[i]] <- 5e-7  # na3
pre[i, IKKa] <- 1; pre[i, NFkB_IkBa] <- 1
post[i, IKKa] <- 1; post[i, NFkB_IkBap] <- 1

## NFkB_IkBap to NFkB: NFkB_IkBap -> NFkB  (ntp)
i <- NFkB_IkBap_to_NFkB
h[[i]] <- 1e-2  # ntp
pre[i, NFkB_IkBap] <- 1
post[i, NFkB] <- 1


#### (18) Free cytoplasmic NF-kB

## NFkB_IkBa to NFkB: NFkB_IkBa -> NFkB  (nc6a)
i <- NFkB_IkBa_to_NFkB
h[[i]] <- 2e-5  # nc6a
pre[i, NFkB_IkBa] <- 1
post[i, NFkB] <- 1

## NFkB and IkBa to NFkB_IkBa: NFkB + IkBa -> NFkB_IkBa  (na1)
i <- NFkB_and_IkBa_to_NFkB_IkBa
h[[i]] <- 5e-7  # na1
pre[i, NFkB] <- 1; pre[i, IkBa] <- 1
post[i, NFkB_IkBa] <- 1

## NFkB import: NFkB -> NFkBn  (ni1)
i <- NFkB_import
h[[i]] <- 1e-2  # ni1
pre[i, NFkB] <- 1
post[i, NFkBn] <- 1

#### (19) Free nuclear NF-kB

## NFkBn and IkBan to NFkB_IkBan: NFkBn + IkBan -> NFkB_IkBan  (na1*kv)
i <- NFkBn_and_IkBan_to_NFkB_IkBan
h[[i]] <- 5e-7 * 5 # na1 * kv
pre[i, NFkBn] <- 1; pre[i, IkBan] <- 1
post[i, NFkB_IkBan] <- 1


#### (20) A20 protein

## A20 sysnthesis: A20t -> A20t + A20   (nc4)
i <- A20_synth
h[[i]] <- 5e-1 # nc4
pre[i, A20t] <- 1
post[i, A20t] <- 1; post[i, A20] <- 1

## A20 degradation: A20 -> 0   (nc5)
i <- A20_degr
h[[i]] <- 5e-4 # nc5
pre[i, A20] <- 1


#### (21) A20 transcript

## A20t transcription: G_A20 + P53pn -> G_A20 + P53pn + A20t
```

```
## ( nc1 * h101 / (h101 + P53pn) * G_A20 )
i <- A20t_transcr
if (onlyR) {
  nc1 <- 1e-1
  h101 <- 6e4

  h[[i]] <- function() {nc1 * h101 / (h101 + y[P53pn]) * y[G_A20]}
} else {
  h[[i]] <- getNativeSymbolInfo("A20t_transcr", PACKAGE="protocols_propensities")$address
}
pre[i, G_A20] <- 1; pre[i, P53pn] <- 1
post[i, G_A20] <- 1; post[i, P53pn] <- 1; post[i, A20t] <- 1

## A20t degradation: A20t -> 0  (nc3)
i <- A20t_degr
h[[i]] <- 7.5e-4 # nc3
pre[i,  A20t] <- 1


#### (22) Free cytoplasmic IkBa protein
#### (23) Free nuclear IkBa protein

## IkBa sysnthesis: IkBat -> IkBat + IkBa   (nc4)
i <- IkBa_synth
h[[i]] <- 5e-1 # nc4
pre[i, IkBat] <- 1
post[i, IkBat] <- 1; post[i, IkBa] <- 1

## IkBa degradation: IkBa -> 0   (nc5a)
i <- IkBa_degr
h[[i]] <- 1e-4 # nc5a
pre[i, IkBa] <- 1

## IkBa import: IkBa -> IkBan  (ni1a)
i <- IkBa_import
h[[i]] <- 2e-3  # ni1a
pre[i, IkBa] <- 1
post[i, IkBan] <- 1

## IkBan export: IkBan -> IkBa  (ne1a)
i <- IkBan_export
h[[i]] <- 5e-3  # ne1a
pre[i, IkBan] <- 1
post[i, IkBa] <- 1


#### (24) IkBa transcript

## IkBat transcription: G_IkBa + P53pn -> G_IkBa + P53pn + IkBat
##( nc1 * h101 / (h101 + P53pn) * G_IkBa )
i <- IkBat_transcr
if (onlyR) {
  nc1 <- 1e-1
  h101 <- 6e4

  h[[i]] <- function() {nc1 * h101 / (h101 + y[P53pn]) * y[G_IkBa]}
} else {
  h[[i]] <- getNativeSymbolInfo("IkBat_transcr", PACKAGE="protocols_propensities")$address
}
pre[i, G_IkBa] <- 1; pre[i, P53pn] <- 1
```

```
post[i, G_IkBa] <- 1; post[i, P53pn] <- 1; post[i, IkBat] <- 1

## IkBat degradation: IkBat -> 0  (nc3)
i <- IkBat_degr
h[[i]] <- 7.5e-4 # nc3
pre[i,  IkBat] <- 1


#### (25) Cytoplasmic IkBa-NF-kB complexes
#### (26) Nuclear IkBa-NF-kB complexes

## NFkB_IkBan export: NFkB_IkBan -> NFkB_IkBa (ne2a)
i <- NFkB_IkBan_export
h[[i]] <- 5e-2  # ne2a
pre[i, NFkB_IkBan] <- 1
post[i, NFkB_IkBa] <- 1


#### (27) Apoptotic factor AF

## AF synthesis: P53pn -> P53pn + AF   ( p1 * q3*P53pn^2/(q4 + q3*P53pn^2) )
i <- AF_synth
if (onlyR) {
  p1 <- 200
  q3 <- 2.5e-11
  q4 <- 1

  h[[i]] <- function() {p1 * q3*y[P53pn]^2/(q4 + q3*y[P53pn]^2)}
} else {
  h[[i]] <- getNativeSymbolInfo("AF_synth", PACKAGE="protocols_propensities")$address
}
pre[i, P53pn] <- 1
post[i, P53pn] <- 1; post[i, AF] <- 1

## AF degradation: AF -> 0  (d9)
##     d9=2*e-4;
i <- AF_degr
h[[i]] <- 2e-4 # d9
pre[i,  AF] <- 1


#### (28) Receptor activation/inactivation

## Rec activation: TNF + Reci -> TNF + Rec   (kb)
i <- Reci_activ
h[[i]] <- 4e-6  # kb
pre[i, TNF] <- 1; pre[i, Reci] <- 1
post[i, TNF] <- 1; post[i, Rec] <- 1

## Rec inactivation: Rec -> Reci   (kf)
i <- Rec_inactiv
h[[i]] <- 6e-4  # kf
pre[i, Rec] <- 1
post[i, Reci] <- 1


#### (29) Activation of genes dependent on NFkB
####      and inactivation on IkBan

## G_A20 activation: NFkBn + Gi_A20 -> NFkBn + G_A20   (nq1)
```

```
i <- Gi_A20_activ
h[[i]] <- 1.5e-7  # nq1
pre[i, NFkBn] <- 1; pre[i, Gi_A20] <- 1
post[i, NFkBn] <- 1; post[i, G_A20] <- 1

## G_A20 inactivation: IkBan + G_A20 -> IkBan + Gi_A20   (nq2)
i <- G_A20_inactiv
h[[i]] <- 1e-6  # nq2
pre[i, IkBan] <- 1; pre[i, G_A20] <- 1
post[i, IkBan] <- 1; post[i, Gi_A20] <- 1

## G_IkBa activation: NFkBn + Gi_IkBa -> NFkBn + G_IkBa   (nq1)
i <- Gi_IkBa_activ
h[[i]] <- 1.5e-7  # nq1
pre[i, NFkBn] <- 1; pre[i, Gi_IkBa] <- 1
post[i, NFkBn] <- 1; post[i, G_IkBa] <- 1

## G_IkBa inactivation: IkBan + G_IkBa -> IkBan + Gi_IkBa   (nq2)
i <- G_IkBa_inactiv
h[[i]] <- 1e-6  # nq2
pre[i, IkBan] <- 1; pre[i, G_IkBa] <- 1
post[i, IkBan] <- 1; post[i, Gi_IkBa] <- 1

## G_P53 activation: NFkBn + Gi_P53 -> NFkBn + G_P53   (nq1)
i <- Gi_P53_activ
h[[i]] <- 1.5e-7  # nq1
pre[i, NFkBn] <- 1; pre[i, Gi_P53] <- 1
post[i, NFkBn] <- 1; post[i, G_P53] <- 1

## G_P53 inactivation: IkBan + G_P53 -> IkBan + Gi_P53   (nq2)
i <- G_P53_inactiv
h[[i]] <- 1e-6  # nq2
pre[i, IkBan] <- 1; pre[i, G_P53] <- 1
post[i, IkBan] <- 1; post[i, Gi_P53] <- 1


#### (30) Activation of genes dependent on P53pn
####      and independent inactivation

## G_MDM2 activation: P53pn + Gi_MDM2 -> P53pn + G_MDM2   ( (q0 + q1 * P53pn^2) * Gi_MDM2 )
i <- Gi_MDM2_activ
if (onlyR) {
  q0 <- 1e-4
  q1 <- 5e-13

  h[[i]] <- function() {(q0 + q1 * y[P53pn]^2) * y[Gi_MDM2]}
} else {
  h[[i]] <- getNativeSymbolInfo("Gi_MDM2_activ", PACKAGE="protocols_propensities")$address
}
pre[i, P53pn] <- 1; pre[i, Gi_MDM2] <- 1
post[i, P53pn] <- 1; post[i, G_MDM2] <- 1

## G_MDM2 inactivation: G_MDM2 -> Gi_MDM2   (q2)
i <- G_MDM2_inactiv
h[[i]] <- 3e-3  # q2
pre[i, G_MDM2] <- 1
post[i, Gi_MDM2] <- 1

## G_PTEN activation: P53pn + Gi_PTEN -> P53pn + G_PTEN   ( (q0 + q1 * P53pn^2) * Gi_PTEN )
i <- Gi_PTEN_activ
```

```
if (onlyR) {
  h[[i]] <- function() {(q0 + q1 * y[P53pn]^2) * y[Gi_PTEN]}
} else {
  h[[i]] <- getNativeSymbolInfo("Gi_PTEN_activ", PACKAGE="protocols_propensities")$address
}
pre[i, P53pn] <- 1; pre[i, Gi_PTEN] <- 1
post[i, P53pn] <- 1; post[i, G_PTEN] <- 1

## G_PTEN inactivation: G_PTEN -> Gi_PTEN   (q2)
i <- G_PTEN_inactiv
h[[i]] <- 3e-3  # q2
pre[i, G_PTEN] <- 1
post[i, Gi_PTEN] <- 1


#### (31, 32, 33) Double strand brakes DSB

## DSB increase due to Gamma: Gamma -> Gamma + DSB  (sort of Tbreaks...)
i <- DSB_increase_Gamma
DNAGy <- 40
tp0 <- 1*3600
h[[i]] <- DNAGy/tp0
pre[i, Gamma] <- 1
post[i, Gamma] <- 1; post[i, DSB] <- 1

## DSB increase due to AF: AF -> AF + DSB  ( a6*( (sign((AF*d9/p1)-Th))+1) )
i <- DSB_increase_AF
if (onlyR) {
  a6 <- 1e-2
  Th <- 0.65
  d9 <- 2e-4

  h[[i]] <- function() {a6*( (sign((y[AF]*d9/p1)-Th))+1)}
} else {
  h[[i]] <- getNativeSymbolInfo("DSB_increase_AF", PACKAGE="protocols_propensities")$address
}
pre[i, AF] <- 1
post[i, AF] <- 1; post[i, DSB] <- 1

## DSB decrease: P53pn + DSB -> P53pn  ( dRep * P53pn^2 * DSB/(DSB + Nsat) )
i <- DSB_decrease
if (onlyR) {
  Nsat <- 3.
  dRep <- 2e-14

  h[[i]] <- function() {dRep * y[P53pn]^2 * y[DSB]/(y[DSB] + Nsat)}
} else {
  h[[i]] <- getNativeSymbolInfo("DSB_decrease", PACKAGE="protocols_propensities")$address
}
pre[i, P53pn] <- 1; pre[i, DSB] <- 1
post[i, P53pn] <- 1
```

### 6.4.4 fast_slow.R

```
slow <- rep(0, transitions)
slow[Reci_activ] <- slow[Rec_inactiv] <-
  slow[Gi_A20_activ] <- slow[G_A20_inactiv] <-
  slow[Gi_IkBa_activ] <- slow[G_IkBa_inactiv] <-
  slow[Gi_P53_activ] <- slow[G_P53_inactiv] <-
  slow[Gi_MDM2_activ] <- slow[G_MDM2_inactiv] <-
  slow[Gi_PTEN_activ] <- slow[G_PTEN_inactiv] <-
```

```
  slow[DSB_increase_Gamma] <- slow[DSB_increase_AF] <-
  slow[DSB_decrease] <- 1
```

### 6.4.5 initcond.R

```
AB <- 1

M <- rep(0, places)

M[PIPp] <- 3.8392e4
M[PIP]  <- 6.1608e4
M[Aktp] <- 5.5480e4
M[Akt]  <- 1.4452e5
M[MDM2] <- 2.1110e4
M[MDM2p] <- 1.3944e4
M[PTEN] <- 3.2094e4
M[MDM2pn] <- 2.3239e5
M[P53n] <- 4.5603e4
M[P53pn] <- 7.1248e3

M[MDM2t] <- 16.0468
M[PTENt] <- 16.0468

M[AF] <- 1.2674e3

M[IKKKn] <- 1e4
M[IKKn] <- 2e5

M[NFkB] <- 153.8276
M[NFkBn] <- 301.1542
M[A20] <- 5.1560e+003*AB
M[A20t] <- 5.1560*AB

M[IkBa] <- 5.8771e3
M[IkBan] <- 2.0432e3
M[IkBat] <- 5.1560

M[NFkB_IkBa] <- 9.9515e4
M[NFkB_IkBan] <- 30.7655

M[P53t] <- 510.8155

M[G_MDM2] <- 0.0802
M[Gi_MDM2] <- 2 - M[G_MDM2]
M[G_PTEN] <- 0.0802
M[Gi_PTEN] <- 2 - M[G_PTEN]

M[G_P53] <- 0.0433
M[Gi_P53] <- 2 - M[G_P53]
M[G_A20] <- 0.0433
M[Gi_A20] <- 2 - M[G_A20]
M[G_IkBa] <- 0.0433
M[Gi_IkBa] <- 2 - M[G_IkBa]

M[Reci] <- 1000
```

### 6.4.6 main.R

```
library("bioPN")

onlyR <- F
```

```
if (!onlyR) {
  # NOTE: change extension .so to .dll if in Windows
  dyn.load("protocols_propensities.so")
}

source("constants.R")
source("places_transitions.R")
source("model.R")
source("fast_slow.R")
source("initcond.R")
#cat("\nInitial conditions are rounded to zero decimals to start\n")
#cat("as some quantities have to be integers for the stochastic simulation.\n\n")
print(M <- round(M,0))

ect <- 1e-9
delta <- 10*60

runs <- 10

runs.to.plot <- min(30,runs)

if (onlyR) {
  Gy <- 2
  timep <- function() {
    StartTime = round(StartTime,0)
    EndTime = 82*3600
    if (StartTime == 0) {
      y[Gamma] = 0
      y[TNF] = 0
      EndTime = 30*3600
    } else if (StartTime == 30*3600) {
      y[TNF] = 10
      EndTime = 33*3600
    } else if (StartTime == 33*3600) {
      y[Gamma] = Gy
      EndTime = 34*3600
    } else if (StartTime == 34*3600) {
      y[Gamma] = 0
      y[TNF] = 0
    }
    list(EndTime,y)
  }
} else {
  timep <- getNativeSymbolInfo("TNF_before_IR", PACKAGE="protocols_propensities")$address
}

model <- list(pre=pre, post=post, h=h, slow=slow, M=M,
              place=place, transition=transition)
set.seed(19761111)
Sim <- HaseltineRawlings(model, timep, delta, runs, ect)

nLast <- length(Sim$dt)
#ApopSurv <- t(sapply(Sim$run, function(run) {as.matrix(run$M[nLast,c("Aktp","P53pn")])}))
ApopSurv <- t(sapply(Sim$run, function(run) {as.data.frame(run$M)[nLast,c(Aktp,P53pn)]}))

colnames(ApopSurv) <- c("Aktp", "P53pn")
pdf(file="second_example.pdf", width=8, height=7)
par(mar=c(2, 2, 2, 1) + 0.1)
plot(ApopSurv,type="p", pch=".",
```

```
     main=paste("Fraction of Apoptotic/Surviving cells (", runs," runs)", sep=""),
     ylim=c(0,8e5),xlim=c(0,5e4))
legend("topleft", "P53pn", bty="n")
legend("bottomright", "Aktp  ", bty="n")
b <- 8e5/5e4
abline(a=0,b=b)
text(1.9e4,4.5e5, paste("Apoptotic:", sum(ApopSurv[,"P53pn"] >= as.numeric(ApopSurv[,"Aktp"]) * b)))
text(3.6e4,4.5e5, paste("Surviving:", sum(ApopSurv[,"P53pn"] < as.numeric(ApopSurv[,"Aktp"]) * b)))

plot(Sim$dt/3600,Sim$run[[1]]$M[[P53pn]],type="l",col="green",ylim=c(0,6e5),
     xlab="",ylab="", main=paste("P53pn, MDM2pn and DSB (first", runs.to.plot, "runs)"),
     panel.first=grid(col="black"))
lines(Sim$dt/3600,Sim$run[[1]]$M[[MDM2pn]],type="l",col="red")
lines(Sim$dt/3600,(Sim$run[[1]]$M[[DSB]])*1000,type="l",col="blue")
if (runs.to.plot > 1) {
  for (n in 2:runs.to.plot) {
    lines(Sim$dt/3600,Sim$run[[n]]$M[[P53pn]],type="l",col="green")
    lines(Sim$dt/3600,Sim$run[[n]]$M[[MDM2pn]],type="l",col="red")
    lines(Sim$dt/3600,Sim$run[[n]]$M[[DSB]]*1000,type="l",col="blue")
  }
}
old.par <- par(font=2)
legend("topleft",c("P53pn", "MDM2pn", "DSB*1000"),
       col=c("green", "red", "blue"), lty=1)
par(old.par)

plot(Sim$dt/3600,Sim$run[[1]]$M[[AF]],type="l",col="orange",ylim=c(0,9e5),
     xlab="",ylab="", main=paste("AF (first", runs.to.plot, "runs)"),
     panel.first=grid(col="black"))
abline(h=6.5e5, lty=2, col="red")
if (runs.to.plot > 1) {
  for (n in 2:runs.to.plot) {
    lines(Sim$dt/3600,Sim$run[[n]]$M[[AF]],type="l",col="orange")
  }
}

which.places <- c(
                  TNF, Gamma,
                  G_PTEN,
                  G_MDM2,
                  G_P53,
                  G_A20,
                  G_IkBa
                  )
for (plc in which.places) {
  plot.type = "s"

  plot(Sim$dt/3600,Sim$run[[1]]$M[[plc]],type=plot.type,main=paste(place[plc], "(first run)"),
       xlab="",ylab="",ylim=c(0, max(as.numeric(lapply(Sim$run, function(run) {max(run$M[[plc]])})))),
       panel.first=grid(col="black"))
}
which.places <- c(
                  PTENt, PTEN,
                  PIP, PIPp,
                  Akt, Aktp,
                  MDM2t, MDM2, MDM2p, MDM2pn,
                  P53t, P53n, P53pn,
                  IKKKn, IKKKa,
                  IKKn, IKKa, IKKi, IKKii,
                  A20t, A20,
```

```
                  IkBat, IkBa, IkBap, IkBan,
                  NFkB, NFkBn,
                  NFkB_IkBa, NFkB_IkBap, NFkB_IkBan,
                  Rec,
                  DSB
                  )
for (plc in which.places) {
  plot.type = "l"

  plot(Sim$dt/3600,Sim$run[[1]]$M[[plc]],type=plot.type,main=paste(place[plc],
                                                          "(first", runs.to.plot, "runs)"),
       xlab="",ylab="",ylim=c(0, max(as.numeric (lapply(Sim$run, function(run) {max(run$M[[plc]])})))),
       panel.first=grid(col="black"))

  if (runs.to.plot > 1) {
    for (n in 2:runs.to.plot) {
      lines(Sim$dt/3600,Sim$run[[n]]$M[[plc]],type=plot.type)
    }
  }

}
dev.off()
```