

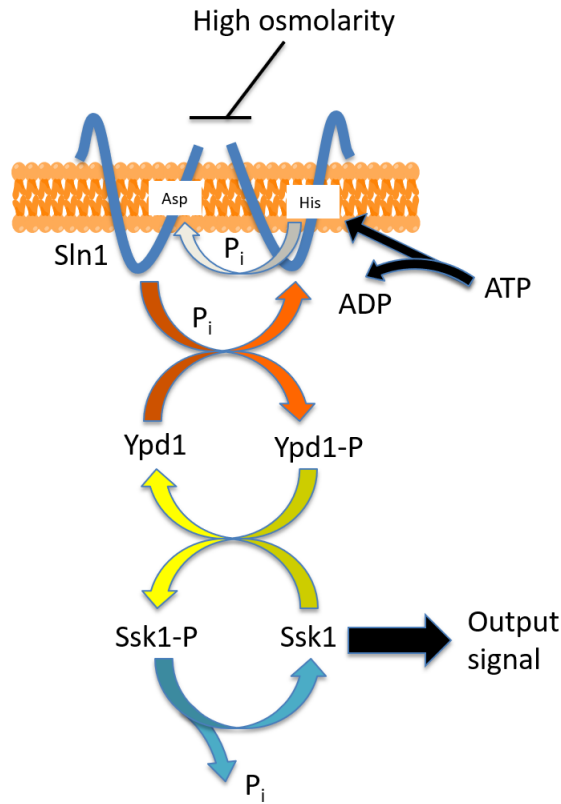
Optimus - Tutorial 4

Nicholas Andre G. Johnson and Aleksandr B. Sahakyan

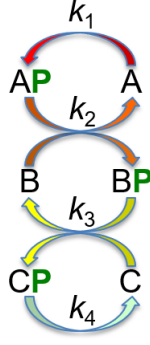
21 August 2018

Example 4: Determining Rate Constants for Coupled ODEs Modelling a Biological System

This Tutorial will demonstrate the use of Optimus to address a problem from yet another problem class. We will employ Optimus to recover the rate constants for a system of coupled ordinary differential equations (ODEs) modelling a biological pathway. Specifically, we will study a phosphorelay system from the high osmolarity glycerol (HOG) pathway in Yeast. A phosphorelay system is a network involving multiple proteins in which after an initial phosphorylation event using ATP (or an alternate phosphate donor), the phosphorylation and dephosphorylation events of proteins in the network proceed without further consumption of ATP (Klipp et al. 2009). The below diagram illustrates the phosphorelay system that will be studied in detail (Klipp et al. 2009):



Under normal circumstances, the transmembrane protein Sln1, which is present as a dimer, autophosphorylates at a histidine residue (consuming ATP). The phosphate group is then transferred to an aspartate residue of Sln1. Thereafter, the phosphate is transferred to the protein Ypd1 and finally to the protein Ssk1. Ssk1 is continuously dephosphorylated to give an output signal. The signalling pathway is inhibited by an increase in osmolarity outside of the cell (Klipp et al. 2009). If we let A represent Sln1, B represent Ypd1, C represent Ssk1 and XP represent the phosphorylated form of protein X , then the above network can be represented by the below network (Klipp et al. 2009):



where each k_i represents the rate constant for the relevant phosphorylation/dephosphorylation reaction.

The above graphic allows us to arrive at the following equations to describe the temporal behavior of the phosphorelay system:

$$\begin{aligned}\frac{d}{dt}[A] &= -k_1[A] + k_2[AP][B] \\ \frac{d}{dt}[B] &= -k_2[AP][B] + k_3[BP][C] \\ \frac{d}{dt}[C] &= -k_3[BP][C] + k_4[CP]\end{aligned}$$

Moreover, under the generally accepted assumption that the degradation and production of proteins occurs on a time scale that far exceeds that of phosphorylation events, we have the following conservation relationships (Klipp et al. 2009):

$$\begin{aligned}[A]_{total} &= [A] + [AP] \\ [B]_{total} &= [B] + [BP] \\ [C]_{total} &= [C] + [CP]\end{aligned}$$

where $[A]_{total}$, $[B]_{total}$ and $[C]_{total}$ are constants. Differentiating, we have:

$$\begin{aligned}\frac{d}{dt}[AP] &= -\frac{d}{dt}[A] \\ \frac{d}{dt}[BP] &= -\frac{d}{dt}[B] \\ \frac{d}{dt}[CP] &= -\frac{d}{dt}[C]\end{aligned}$$

Given this model of the phosphorelay system, the question we desire to answer is as follows: given initial concentrations of the three proteins $\{[A]_i, [B]_i, [C]_i\}$ and target concentrations of the three proteins $\{[A]_t, [B]_t, [C]_t\}$, what are the values $\{k_1, k_2, k_3, k_4\}$ that result in the proteins having the target concentrations at steady state when the system is allowed to equilibrate from the initial concentrations? This formulation assumes that no information is known about the rate constants and that initial and target concentrations can be determined experimentally. The problem formulation could be altered depending on what information is known or can be determined experimentally.

Defining Optimus Inputs

Having outlined how the behaviour of the phosphorelay system can be modelled using a system of differential equations, we can now proceed with defining input parameters for Optimus. We will create a variable *state* that will be a numeric vector holding the names and initial concentration of all species in the network. For this Tutorial, we will choose $[A]_i = [B]_i = [C]_i = 100$ and $[AP]_i = [BP]_i = [CP]_i = 0$. Note that the units are arbitrary and that the total sum of units across this vector will remain constant throughout the simulation of the dynamics of the phosphorelay system.

```
state <- c(cA=100, cB=100, cC=100, cAP=0, cBP=0, cCP=0)
```

Next, we will create a variable *target* which will be a numeric vector holding the names and target concentration of all species in the network. We will arbitrarily choose target values of $[A]_t = 90$, $[B]_t = 20$, $[C]_t = 70$, $[AP]_t = 10$, $[BP]_t = 80$ and $[CP]_t = 30$. Note that the chosen target values must be consistent with the above defined conservation equations, meaning we must have $[X]_i + [XP]_i = [X]_t + [XP]_t, \forall X \in \{A, B, C\}$.

```
target <- c(cA=90, cB=20, cC=70, cAP=10, cBP=80, cCP=30)
```

In order to determine the steady state behavior of the ODE system, we will employ the function *ode()* from the R package *deSolve* (this function interfaces with the Fortran library typically used to solve systems of differential equations). This function requires as input a function that describes the dynamics of the ODE system. We will call this function *model()*. At a high level, *model()* will simply define the equations derived in the previous section that describe the network. It should contain equations that use the objects with the names specified within *state* above, and should have equations that assign the outcomes to new objects that have the same order and names as specified in *state*, but with “d” at the beginning (a more detailed description of the requirements of *model()* can be found in the documentation of *ode()*).

```
model <- function(t, state, K){  
  
  with( as.list(c(state, K)), {  
    # rate of change  
    dcA <- -k1*cA+k2*cAP*cB  
    dcB <- -k2*cAP*cB+k3*cBP*cC  
    dcC <- -k3*cBP*cC+k4*cCP  
    dcAP <- -dcA  
    dcBP <- -dcB  
    dcCP <- -dcC  
    # return the rate of change  
    list(c(dcA, dcB, dcC, dcAP, dcBP, dcCP))  
  })  
}
```

The variables *state* and *target*, and the function *model()* should be stored as entries in a list *DATA* which will be given to the functions *m()* and *u()* as inputs.

```
DATA <- NULL  
DATA$state <- state  
DATA$target <- target  
DATA$model <- model
```

We will make *K* be a numeric vector holding the set of rate constants $\{k_1, k_2, k_3, k_4\}$. We will (arbitrarily) initialize each rate constant to have value 1.0.

```
K <- c(k1=1.0, k2=1.0, k3=1.0, k4=1.0)
```

The function *m()* will take as input the vector *K* of rate constants and the list *DATA*. It will return an object *O* that contains the concentrations of the six species in the network when the system is simulated from the initial state specified in *DATA* using the *K* rate constants for 10 time steps. Note that it is not

necessarily guaranteed that the system will reach a steady state after 10 time steps; the number of time steps was chosen such that the optimisation procedure would terminate within 1-2 hours. $m()$ will call the function `ode()` from the package `deSolve`, so we must first ensure that `deSolve` is installed.

```
install.packages("deSolve")

library(deSolve)
m <- function(K, DATA){
  state <- DATA$state
  model <- DATA$model

  span = 10.0

  times <- c(0, span)
  O <- ode(y=state, times=times, func=model, parms=K)[2,2:(length(state)+1)]
  return(O)
}
```

Recall that the function $u()$ must return an energy E and a quality Q of the candidate solution. Here, $u()$ will set both E and Q to be the RMSD between the steady state concentrations of the network corresponding to the current set of rate constants K as determined by $m()$ and the target concentrations.

```
u <- function(O, DATA){
  target <- DATA$target
  RESULT <- NULL
  RESULT$Q <- sqrt(mean((O-target)^2)) # measure of the fit quality
  RESULT$E <- RESULT$Q # the pseudoenergy derived from the above measure

  return(RESULT)
}
```

The final mandatory input to `Optimus` which must be defined is the alteration function $r()$. Just as in Tutorial 1, for each snapshot of K , we shall randomly select one of its four coefficients, then either increment or decrement (chosen randomly) it by 0.0002, returning the altered set of coefficients. Since we are dealing with rate constants in this case, if ever $r()$ were to make an entry in K negative, that entry will automatically be set to 0.

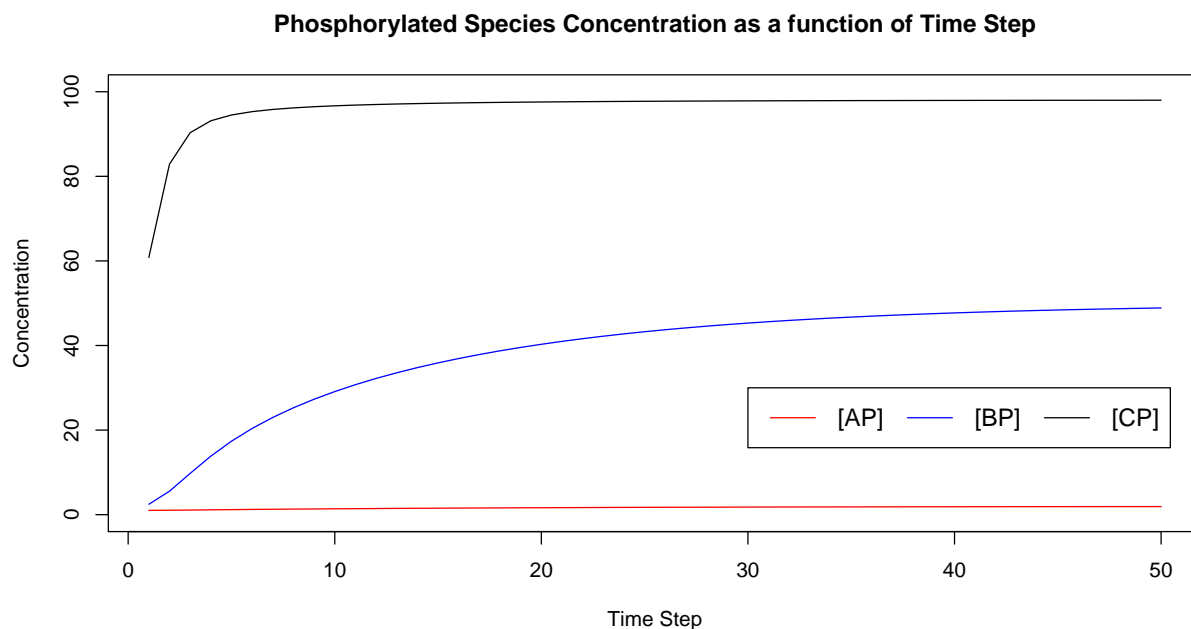
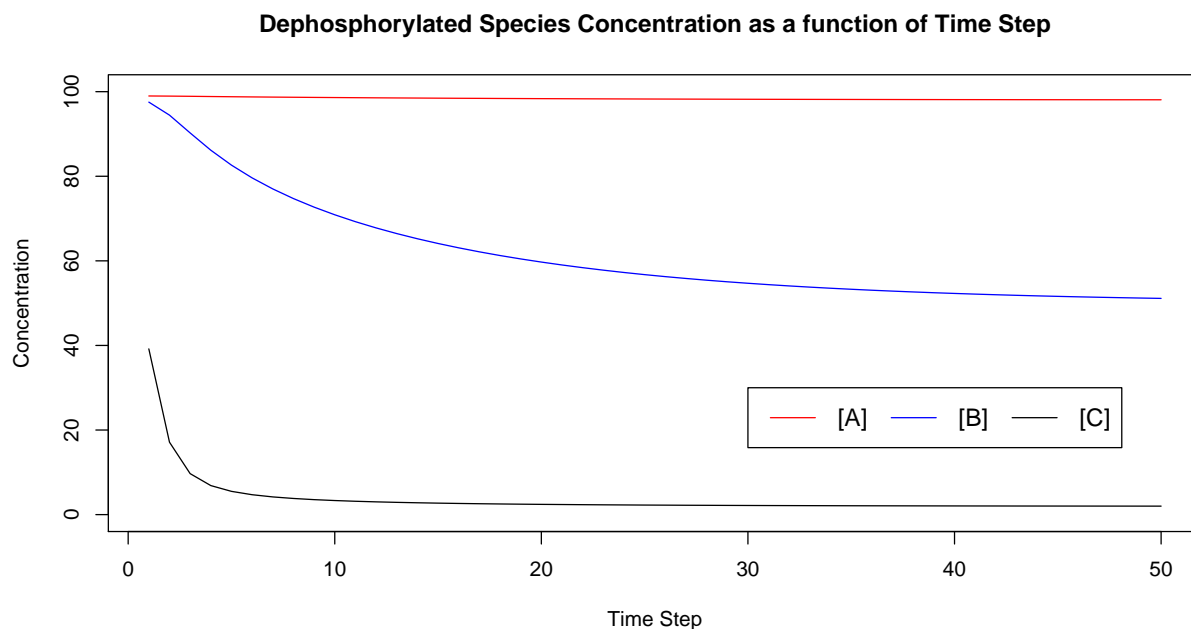
```
r <- function(K){
  K.new <- K
  # Randomly selecting a coefficient to alter:
  K.ind.toalter <- sample(size=1, x=1:length(K.new))
  # Creating a potentially new set of coefficients where one entry is altered
  # by either +move.step or -move.step, also randomly selected:
  move.step <- 0.0002
  K.new[K.ind.toalter] <- K.new[K.ind.toalter] + sample(size=1, x=c(-move.step, move.step))

  ## Setting the negative coefficients to 0
  neg.ind <- which(K.new < 0)
  if(length(neg.ind)>0){ K.new[neg.ind] <- 0 }

  return(K.new)
}
```

Exploring the System Dynamics

Before calling Optimus to solve this problem, let us first simulate the system of ODEs from the chosen initial state using a few sets of arbitrary rate constants to become familiar with how the system evolves. The below graphs illustrate the evolution of the system for 50 time steps for the rate constants $\{k_1 = 1.0, k_2 = 1.0, k_3 = 1.0, k_4 = 1.0\}$:

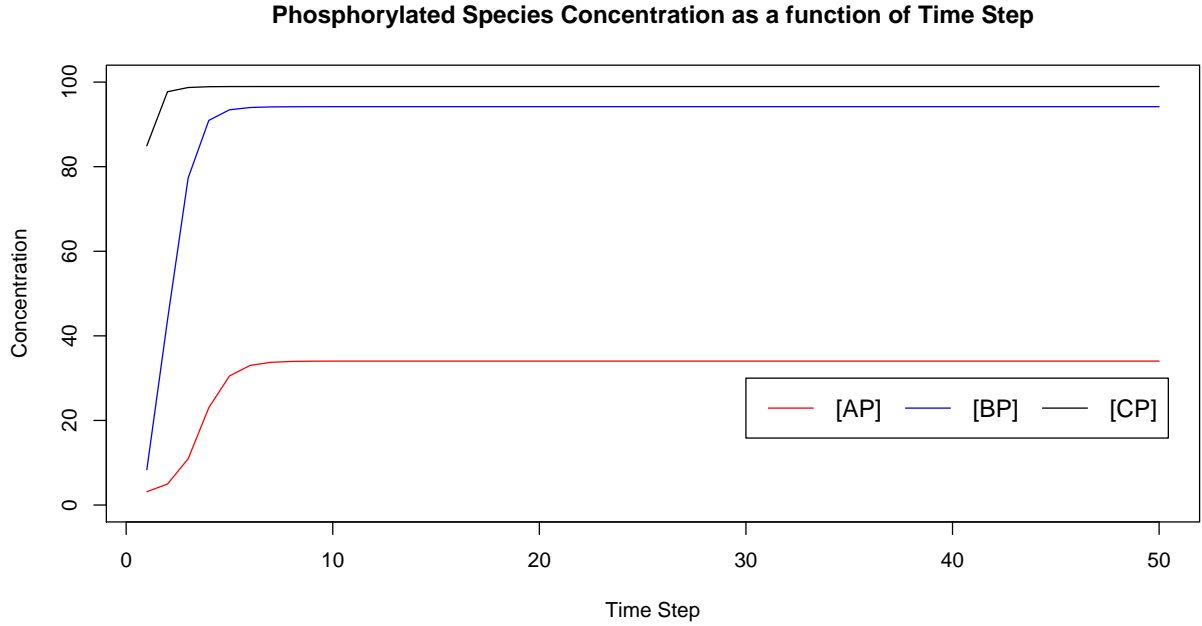
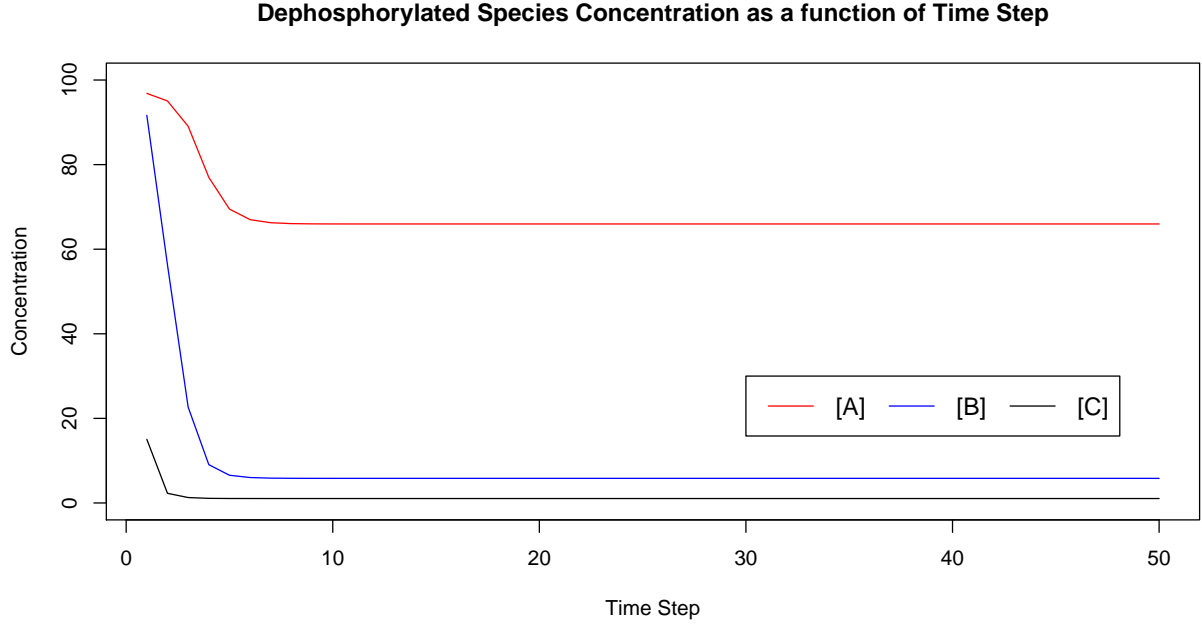


The table below summarizes the initial and final concentrations of the various species when the system is simulated for 50 time steps using the rate constants $\{k_1 = 1.0, k_2 = 1.0, k_3 = 1.0, k_4 = 1.0\}$:

Table 1: System Summary for $k_1 = k_2 = k_3 = k_4 = 1.0$

	[A]	[B]	[C]	[AP]	[BP]	[CP]
Initial	100.00000	100.00000	100.000000	0.000000	0.00000	0.00000
Final (after 50 time steps)	98.08145	51.12118	2.004924	1.918549	48.87882	97.99508

If instead we use the set of rate constants $\{k_1 = 1.5, k_2 = 0.5, k_3 = 1.0, k_4 = 1.0\}$, the system evolves as follows:



The table below summarizes the initial and final concentrations of the various species when the system is

simulated for 50 time steps using the rate constants $\{k_1 = 0.5, k_2 = 1.0, k_3 = 1.0, k_4 = 1.5\}$:

Table 2: System Summary for $k_1 = 1.5, k_2 = 0.5, k_3 = k_4 = 1.0$

	[A]	[B]	[C]	[AP]	[BP]	[CP]
Initial	100.00000	100.000000	100.000000	0.00000	0.00000	0.00000
Final (after 50 time steps)	65.96628	5.814787	1.050583	34.03372	94.18521	98.94942

Acceptance Ratio Annealing Optimus Run

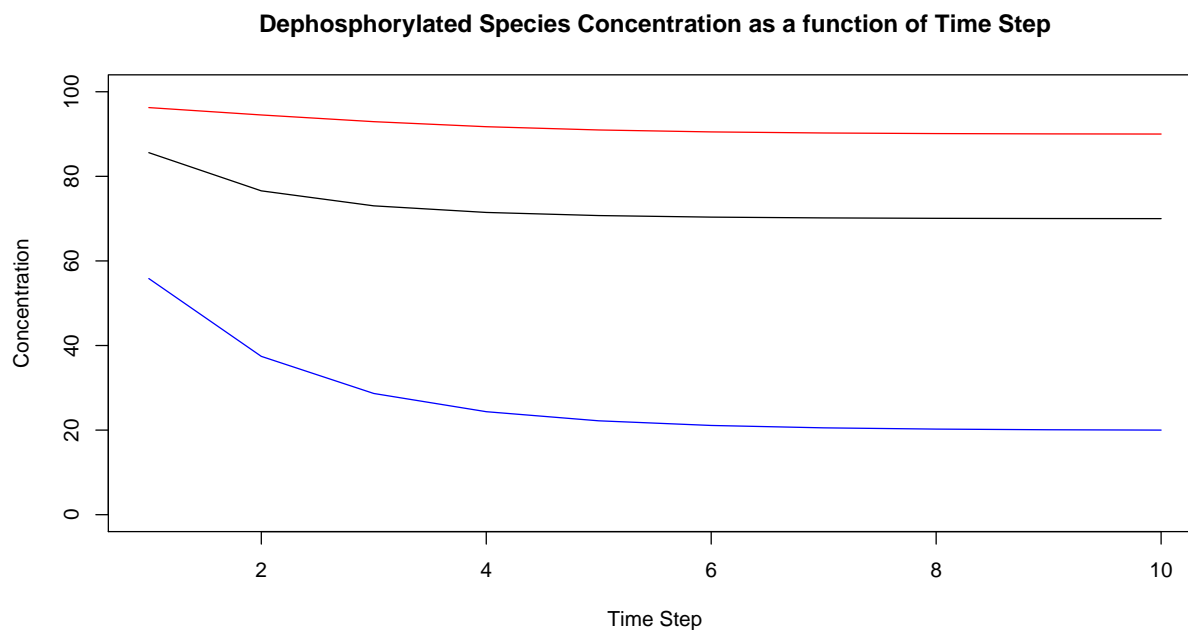
We will now call Acceptance Ratio Annealing Optimus to solve our problem. Similarly to Tutorial 2, we will execute 200 000 optimisation iterations and perform 2 annealing cycles. We will set DUMP.FREQ to have a value of 100 000.

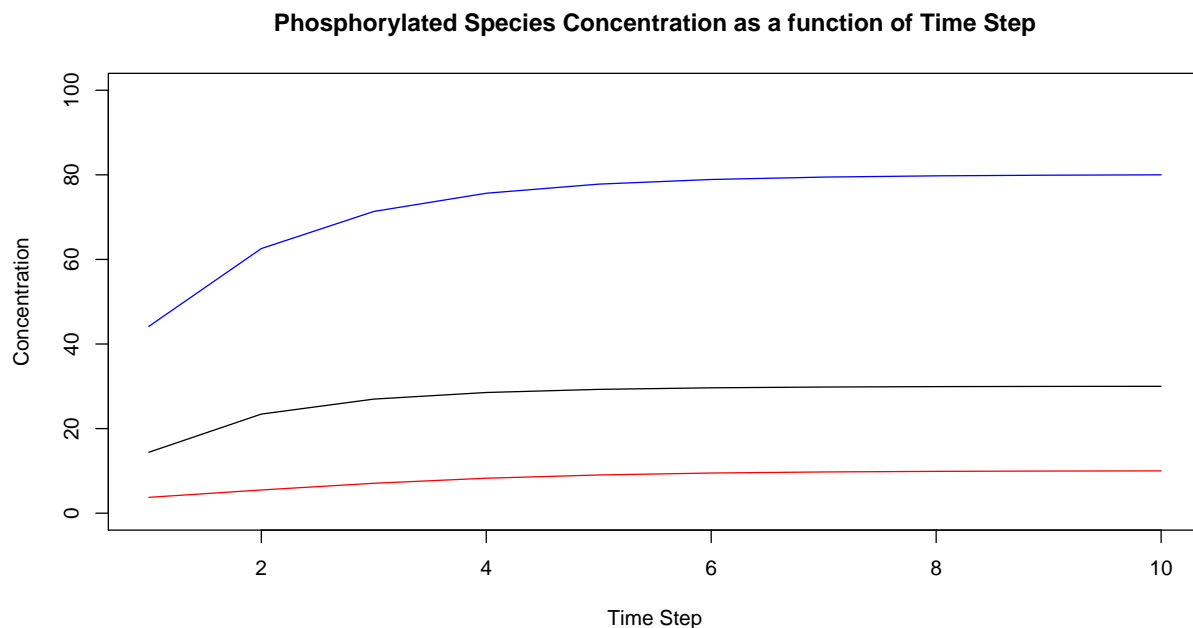
```
Optimus(NCPU = 4, K.INITIAL = K, rDEF = r, mDEF = m, uDEF = u, OPT.TYPE = "SA",
        OPTNAME = "DE_4_SA", DATA = DATA, NUMITER = 2e+05, CYCLES = 2, DUMP.FREQ = 1e+05,
        LONG = FALSE)
```

Table 3: 4 Core Acceptance Ratio Annealing Optimus Run Results

	E (RMSD)	K1	K2	K3	K4
Processor 1	0.0012516	0.7974	0.3586	0.0128	2.3886
Processor 2	0.0017556	0.7850	0.3532	0.0126	2.3512
Processor 3	0.0013625	0.8098	0.3642	0.0130	2.4262
Processor 4	0.0012516	0.7974	0.3586	0.0128	2.3886

Of the 4 optimisation replicas, Processors 1 and 4 find the best set of rate constants, $\{k_1 = 0.7974, k_2 = 0.3586, k_3 = 0.0128, k_4 = 2.3886\}$. This set of rate constants results in an RMSD (after 10 iterations) of 0.0012516. Let us simulate how the system evolves according to these rate constants for 10 time steps:





The table below summarizes the initial and final concentrations of the various species when the system is simulated for 10 time steps using the rate constants $\{k_1 = 0.7974, k_2 = 0.3586, k_3 = 0.0128, k_4 = 2.3886\}$:

Table 4: System Summary for $k_1 = 0.7974$, $k_2 = 0.3586$, $k_3 = 0.0128$, $k_4 = 2.3886$

	[A]	[B]	[C]	[AP]	[BP]	[CP]
Initial	100.00000	100.00000	100.00000	0.00000	0.00000	0.00000
Final (After 50 time steps)	89.99814	20.00081	69.99924	10.00186	79.99919	30.00076

As can be seen in the above table, the concentration of the species in the system after 10 time steps are very close to the target values $[A]_t = 90$, $[B]_t = 20$, $[C]_t = 70$, $[AP]_t = 10$, $[BP]_t = 80$ and $[CP]_t = 30$. As alluded to earlier, it is possible that the system not have reached a steady state after 10 time steps, however the above graphs suggest that the concentrations after 10 steps are already extremely close to if not equal to steady state values. The optimisation process could be reexecuted after increasing the value of the parameter *span* in the function *m()* to simulate the system for a larger number of time steps.

Replica Exchange Optimus Run

Let us now examine how replica exchange Optimus using 12 cores performs on this task. We will use 200 000 optimisation iterations and set STATWINDOW to equal, similarly to Tutorials 2 and 3.

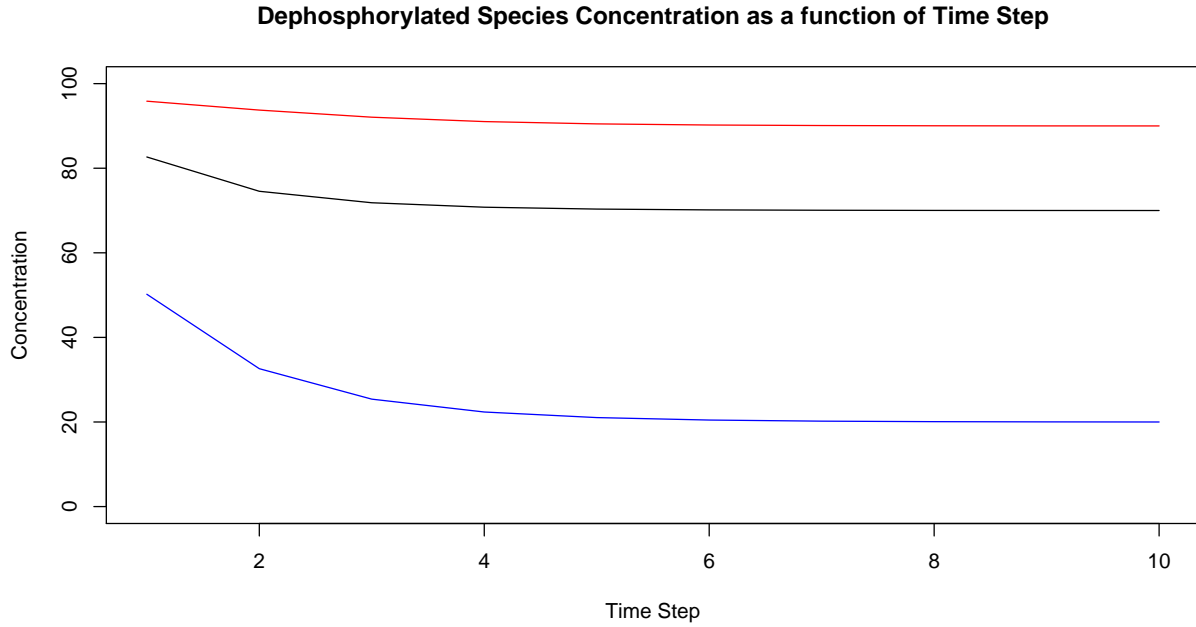
```
ACCRATIO <- c(90, 82, 74, 66, 58, 50, 42, 34, 26, 18, 10, 2)
```

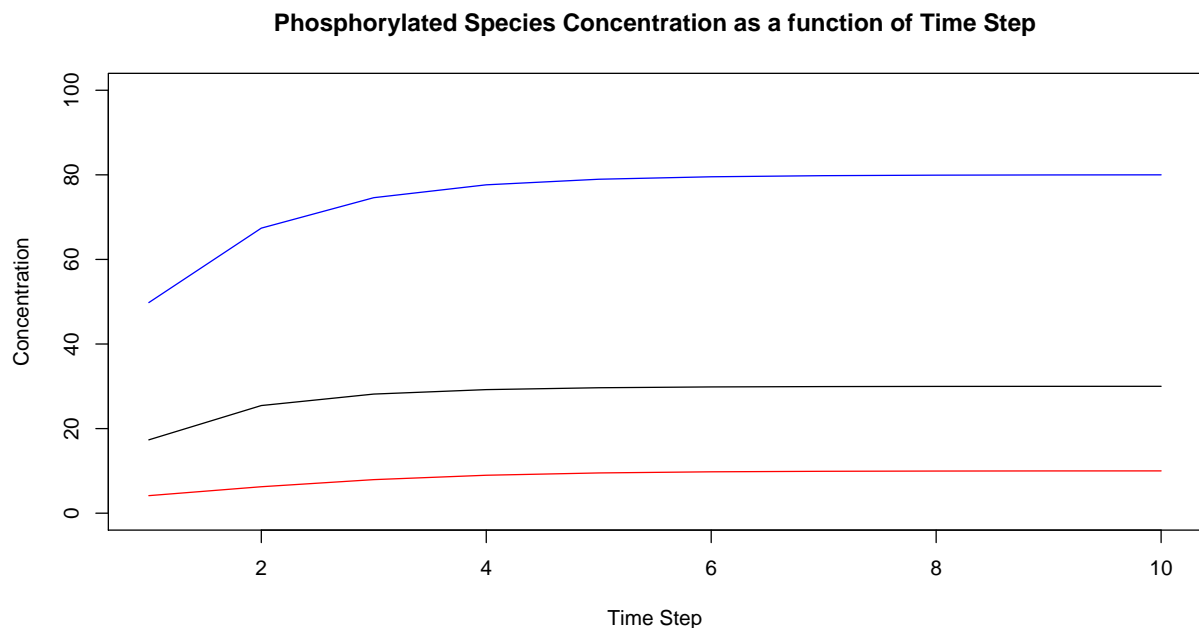
```
Optimus(NCPU = 12, K.INITIAL = K, rDEF = r, mDEF = m, uDEF = u, ACCRATIO = ACCRATIO,
  OPT.TYPE = "RE", DATA = DATA, OPTNAME = "DE_12_RE", NUMITER = 2e+05, STATWINDOW = 50,
  DUMP.FREQ = 1e+05, LONG = FALSE)
```


Table 5: 12 Core Replica Exchange Optimus Run Results

	Replica Acceptance Ratio	E (RMSD)	K1	K2	K3	K4
Processor 1	90	0.0026225	0.9088	0.4090	0.0146	2.7246
Processor 2	82	0.0026343	0.8346	0.3754	0.0134	2.5012
Processor 3	74	0.0019724	0.7234	0.3252	0.0116	2.1644
Processor 4	66	0.0020945	0.9586	0.4312	0.0154	2.8748
Processor 5	58	0.0029243	0.9338	0.4200	0.0150	2.8002
Processor 6	50	0.0025811	0.8964	0.4034	0.0144	2.6872
Processor 7	42	0.0028685	0.8592	0.3866	0.0138	2.5752
Processor 8	34	0.0025121	0.8840	0.3978	0.0142	2.6500
Processor 9	26	0.0011265	0.9834	0.4424	0.0158	2.9492
Processor 10	18	0.0025811	0.8964	0.4034	0.0144	2.6872
Processor 11	10	0.0012516	0.7974	0.3586	0.0128	2.3886
Processor 12	2	0.0017556	0.7850	0.3532	0.0126	2.3512

Of the 12 optimisation replicas, Processor 9 (26% acceptance ratio) finds the best set of rate constants, $\{k_1 = 0.9834, k_2 = 0.4424, k_3 = 0.0158, k_4 = 2.9492\}$. This set of rate constants results in an RMSD (after 10 iterations) of 0.0011265, which is lower than the RMSD of the solution found by acceptance ratio annealing Optimus (0.0012516). Let us simulate how the system evolves according to these rate constants for 10 time steps:





The table below summarizes the initial and final concentrations of the various species when the system is simulated for 10 time steps using the rate constants $\{k_1 = 0.9834, k_2 = 0.4424, k_3 = 0.0158, k_4 = 2.9492\}$:

Table 6: System Summary for $k_1 = 0.9834$, $k_2 = 0.4424$, $k_3 = 0.0158$, $k_4 = 2.9492$

	[A]	[B]	[C]	[AP]	[BP]	[CP]
Initial	100.00000	100.00000	100.00000	0.00000	0.00000	0.00000
Final (After 50 time steps)	89.99807	19.99983	70.00022	10.00193	80.00017	29.99978

Here again, we see that the species concentrations after 10 time steps are remarkably close to the target values $[A]_t = 90$, $[B]_t = 20$, $[C]_t = 70$, $[AP]_t = 10$, $[BP]_t = 80$ and $[CP]_t = 30$ and the graphs suggests that these concentrations have either converged or are very close to converging to steady state.

Summary

We have seen how Optimus can be employed to recover rate constants for a systems of coupled ODEs that describe a biological pathway. Given an initial state of the system and a target state, both Acceptance Ratio Annealing Optimus and Replica Exchange Optimus found a set of rate constants that resulted in the desired system behaviour upon simulation of the system. Replica Exchange Optimus retrieved a solution that resulted in a lower RMSD after 10 time steps (0.0011265) than the solution found by Acceptance Ratio Annealing Optimus (0.0012516).

Table 7: Summary of Species' Concentrations after 10 Time Steps

	[A]	[B]	[C]	[AP]	[BP]	[CP]
Target	90.00000	20.00000	70.00000	10.00000	80.00000	30.00000
Optimus (Acceptance Ratio Annealing)	89.99814	20.00081	69.99924	10.00186	79.99919	30.00076
Optimus (Replica Exchange)	89.99807	19.99983	70.00022	10.00193	80.00017	29.99978

Table 8: Summary of Recovered Rate Constants

	E (RMSD)	K1	K2	K3	K4
Optimus (Acceptance Ratio Annealing)	0.0012516	0.7974	0.3586	0.0128	2.3886
Optimus (Replica Exchange)	0.0011265	0.9834	0.4424	0.0158	2.9492

Klipp, Edda, Wolfram Liebermeister, Cristoph Wierling, Axel Kowald, Hans Lehrach, and Ralf Herwig. 2009. *Systems Biology*. Wiley-VCH.