20231223_PKAmodel_2nd

2023-12-23

to do: phosphorylation levels should be at steady state before t = 0

```r
rm(list = ls())
set.seed(20230106)

library('deSolve')
library('tidyverse')
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

Simulation

Modeling the phosphorylation of different substrates by the same kinase and dephosphorylation by the same phosphatase Substrates have different Km for kinase, but the same for PPTase The kinase changes its activity over time We assume the we know the total amount of each substrate

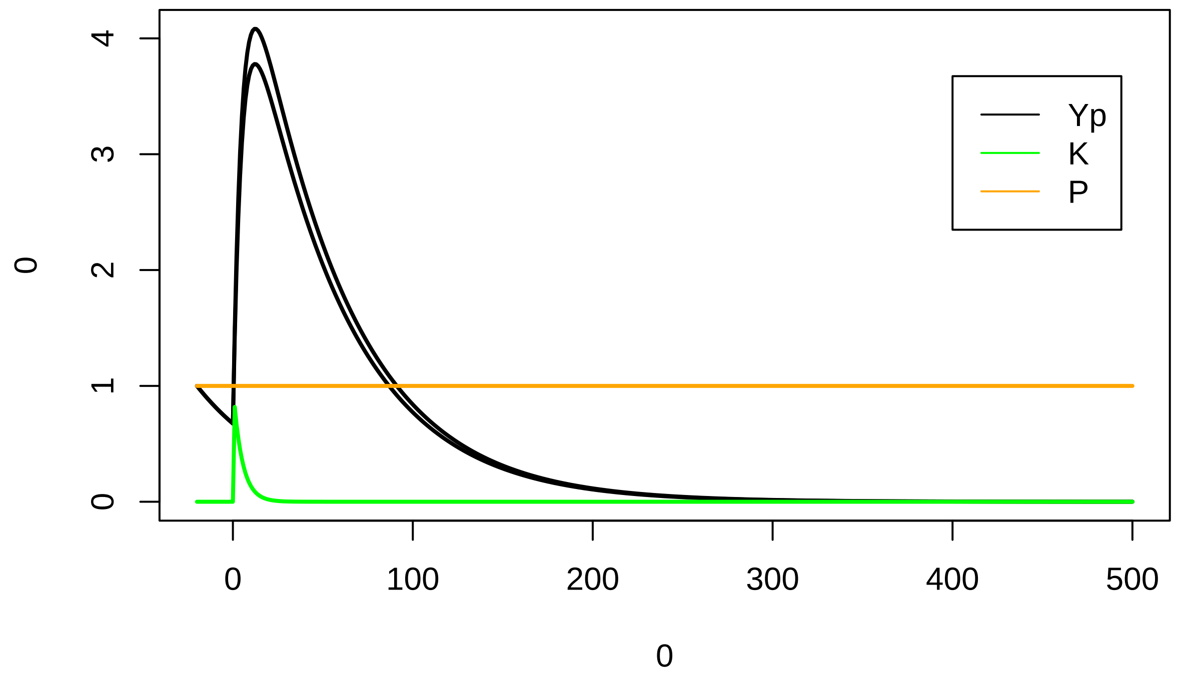
dYp_dt = K * (Ytot-Yp)/[(Ytot-Yp)+Km_K] - P * Yp/(Yp+Km_P) K: kinase activity P: PPTase activity Ytot: total substrate concentration Yp: phosphorylated substrate concentration Km_K: Michaelis constant of kinase Km_P: Michaelis constant of PPTase

```r
Yp_fun = function(t, Yp, pars) {
 # ODE of phosphorylation change
  # Yp: vector of concentrations of phosphorylated substrates
  # pars: named list of Ytot, Km_K, Km_P
    with(pars, {
      K = K_fun(t)
      P = P_fun(t)
      dYp_dt = K * (Ytot-Yp)/((Ytot-Yp)+Km_K) - P * Yp/(Yp+Km_P)

      return(list(Yp = dYp_dt))
  })
}
```

```r
# Define kinase activity (K) and PPTase activity (P) as functions of time
K_fun = function(t){
  #K = 0.02 * t
  #K = ifelse(t > 0, 0.02 * t, 0)
  K = ifelse(t > 0, exp(-0.2 * t), 0)
  return(K)
}

P_fun = function(t){
  P = rep(1, length(t))
  return(P)
}
```

```r
# initial condition
Yp_init = c(1, 1)  # initial does not necessarily mean t == 0

# parameters
p = list(Ytot = c(100, 100), Km_K = c(10, 20), Km_P = c(50, 50))
```

```r
# Simulation
t = seq(-20, 500, 1)
ODEout = as.data.frame(ode(y = Yp_init, times = t, func = Yp_fun, parms = p))

myPal = c('black', 'green', 'orange')
palette(myPal)
plot(type = 'n', 0, 0, xlim = c(min(t), max(t)),
  ylim= c(0, max(ODEout[, 2:dim(ODEout)[2]])) )
for(i in 2:dim(ODEout)[2]){
  lines(ODEout$time, ODEout[, i], col = 1, lwd = 2)
}
lines(t, K_fun(t), col = 2, lwd = 2)
lines(t, P_fun(t), col = 3, lwd = 2)
legend(x= 0.8 * max(t), y = 0.9 * max(ODEout[, 2]),
       legend = c('Yp', 'K', 'P'), col = myPal[1:3], lty = 1)
```



Assumed observed data

```r
t_obs = c(0, 10, 20, 40, 80, 160, 320, 500)
Yp_obs = ODEout[match(t_obs, ODEout$time),]
```

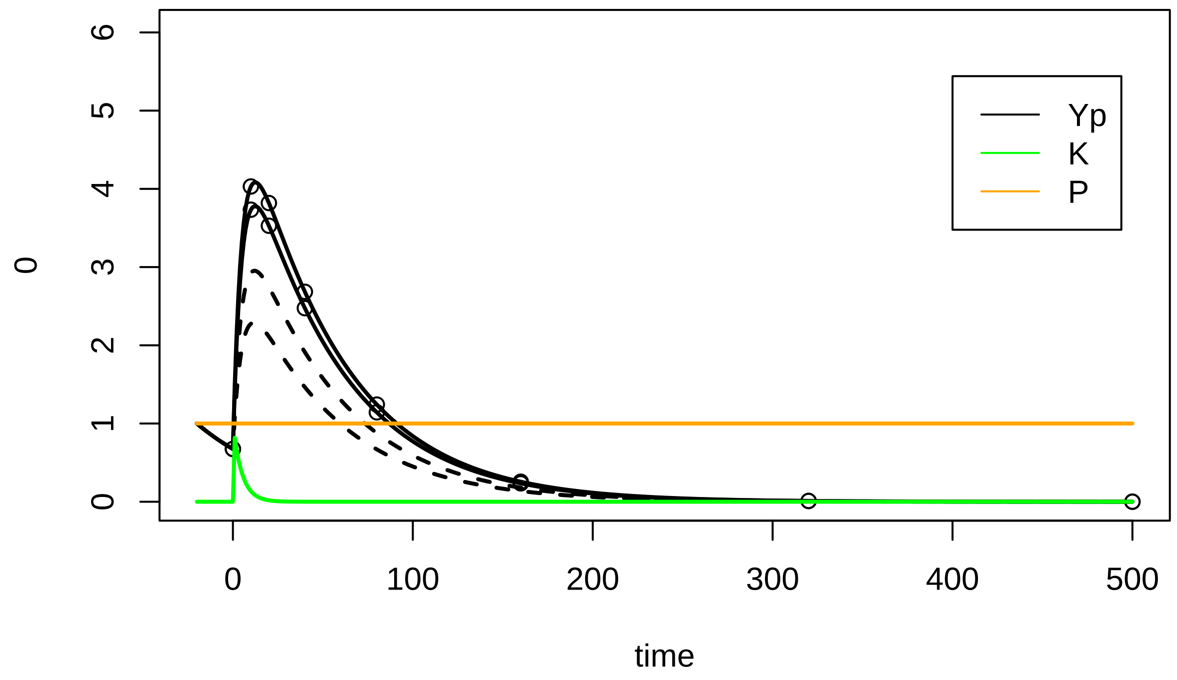
to do: phosphorylation levels should be at steady state before t = 0

Make a function that solves the ODEs at different parameter choices

```r
ODE_fun = function(Yp_init, times, func, p_list) {
  # Function to solve ODEs given in func at each of the parameter combinations
   # listed in p_list
   # Each element of p_list is a list of vectors, where each vector element
   # corresponds to a substrate

  ODEout_loDF = list() # list of data.frames

  # loop over parameter combinations
  for(i in 1:length(p_list)){
    p = p_list[[i]]
    ODEout_loDF[[i]] = as.data.frame(ode(y = Yp_init, times = times, func = func, parms = p))
  }

  return(ODEout_loDF)
}
```

Test different parameter choices

```r
p_toTest = list(
  p1 = list(Ytot = c(100, 100), Km_K = c(10, 20), Km_P = c(50, 50)),
  p2 = list(Ytot = c(100, 100), Km_K = c(60, 120), Km_P = c(50, 50))
)

fits_loDF = ODE_fun(Yp_init = Yp_init, times= t, func = Yp_fun, p_list = p_toTest)

myPal = c('black', 'green', 'orange')
palette(myPal)
Y_expand = 1.5 # factor by which to expand Y-axis
plot(type = 'n', 0, 0, xlim = c(min(t), max(t)),
  ylim= c(0, Y_expand * max(Yp_obs[, 2:dim(Yp_obs)[2]])), xlab = 'time' )
for(i in 2:dim(Yp_obs)[2]){
  points(Yp_obs$time, Yp_obs[,i])
}

for(mIdx in 1:length(fits_loDF)){
  ODEout = fits_loDF[[mIdx]]
  for(i in 2:dim(ODEout)[2]){
    lines(ODEout$time, ODEout[, i], col = 1, lwd = 2, lty = mIdx)
  }
}
lines(t, K_fun(t), col = 2, lwd = 2)
lines(t, P_fun(t), col = 3, lwd = 2)

legend(x= 0.8 * max(t), y = 0.9 * Y_expand * max(Yp_obs[, 2:dim(Yp_obs)[2]]),
       legend = c('Yp', 'K', 'P'), col = myPal[1:3], lty = 1)
```



Function to evaluate model fit using likelihood and assuming fixed relative SD

```
loglikFUN = function(obsData, modelData){
  # Function to calculate log-likelihood of observed data for given model,
   # assuming fixed relative SD
  #ARGS:
   # obsData: data.frame of observed data with first column named "time"
   # modelData: data.frame of modelled data with first column "time" that contains the
    # time values of 'obsData'
  # RETURNS:
   # vector with log-likelihood for each substrate

  relSD = 0.1

  # Observed data may have multiple entries per TP; these are treated as INDEPENDENT replicates
  obsData_tLengths = aggregate(. ~ time, data = obsData, FUN = length)
  if(any(obsData_tLengths[, colnames(obsData_tLengths) != 'time'] > 1)) {
    print('WARNING: Observed data contain multiple entries per TP! - The are treated as indpendent repli
  }

  # Model data should not have duplicates in time-variable
  if(sum(duplicated(modelData$time)) > 0){
    stop('ERROR: Duplicated TPs in model!')
  }

  modelData_atObsTP = modelData[match(obsData$time, modelData$time), ]
  if (any(is.na(modelData_atObsTP$time))){
    stop('Error: Not all observed time points found in modelled data!')
  }

  logLik = rep(NA, dim(obsData)[2] - 1)
  # for each substrate:
  for (i in 2:dim(obsData)[2]){
    logLik[i - 1] = sum(log(
      dnorm(obsData[, i], mean = modelData_atObsTP[, i], sd = relSD * obsData[, i])))
  }

  return(logLik)
}
```

Test log-likelihood function

```
relNoise = 0.1
# data from the same parameters as observed Data
noise1 = matrix( rnorm(n = dim(fits_loDF[[1]])[1] * (dim(fits_loDF[[1]])[2] - 1), mean = 0, sd = relNoi
modelData1 = cbind.data.frame(time = fits_loDF[[1]]$time, fits_loDF[[1]][,2:3] +
  noise1)

# data from the different parameters than observed Data
noise2 = matrix( rnorm(n = dim(fits_loDF[[2]])[1] * (dim(fits_loDF[[2]])[2] - 1), mean = 0, sd = relNoi
modelData2 = cbind.data.frame(time = fits_loDF[[2]]$time, fits_loDF[[2]][,2:3] +
  noise2)

loglikFUN(obsData = Yp_obs, modelData = modelData1)
```

```
## [1] 17.89718 18.25304
```

```
loglikFUN(obsData = Yp_obs, modelData = modelData2)
```

```
## [1]  -7.453841 -43.239435
```

```
Yp_obs
```

```
##     time            1            2
## 21     0 0.6746951751 0.6746951233
## 31    10 4.0304103525 3.7336150290
## 41    20 3.8188768740 3.5284745343
## 61    40 2.6840878010 2.4750558136
## 101   80 1.2421406389 1.1428849193
## 181  160 0.2557803743 0.2349727810
## 341  320 0.0104774489 0.0096212742
## 521  500 0.0002863413 0.0002629382
```

```
modelData1[match(Yp_obs$time, modelData1$time),]
```

```
##     time            1           2
## 21     0 0.7535723763 0.663278021
## 31    10 3.6160729211 3.770580344
## 41    20 4.2956875569 3.565658622
## 61    40 2.7888852777 2.280477199
## 101   80 1.3904123222 1.207316676
## 181  160 0.2835258337 0.223300708
## 341  320 0.0113250090 0.008964346
## 521  500 0.0002908432 0.000328177
```

```
modelData2[match(Yp_obs$time, modelData2$time),]
```

```
##     time           1            2
## 21     0 0.653705200 0.6573787394
## 31    10 2.753486870 2.5254232125
## 41    20 2.622373167 1.9458475106
## 61    40 1.999787953 1.5951418439
## 101   80 0.863239697 0.6574588003
## 181  160 0.201547407 0.1396141267
## 341  320 0.006460229 0.0043898825
## 521  500 0.000235395 0.0001355535
```