Supporting Information for "Estimating and Assessing Differential Equation Models with Time-Course Data"

Samuel W.K. Wong, Shihao Yang, S.C. Kou

This supporting information file provides complete step-by-step code in R for running MAGI on each of the examples in the main text. For further information on the software package, we refer the reader to the usage guide for MAGI at https://arxiv.org/abs/2203.06066.

Setup

Ensure that the `magi` R package is installed and loaded:

```
install.packages("magi")
```

```
library(magi)
```

Repressilator gene regulation network

We begin by defining a function that codes the log-transformed ODEs:

```
RrepressilatorGeneRegulationLogODE <- function(theta, x, tvec) {
  m_laci = exp(x[,1])
  m_tetr = exp(x[,2])
  m_ci = exp(x[,3])
  p_laci = exp(x[,4])
  p_tetr = exp(x[,5])
  p_ci = exp(x[,6])

  alpha0 = theta[1]
  alpha = theta[2]
  n = theta[3]
  beta = theta[4]

  resultdt <-  array(0, c(nrow(x),ncol(x)))

  resultdt[,1] = -1 + (alpha / (1 + p_ci^n) + alpha0) / m_laci
  resultdt[,2] = -1 + (alpha / (1 + p_laci^n) + alpha0) / m_tetr
  resultdt[,3] = -1 + (alpha / (1 + p_tetr^n) + alpha0) / m_ci
  resultdt[,4] = (-beta*(1 - m_laci / p_laci))
  resultdt[,5] = (-beta*(1 - m_tetr / p_tetr))
  resultdt[,6] = (-beta*(1 - m_ci / p_ci))

  resultdt
}
```

Next, we provide the gradients of the ODEs with respect to the system components X and the parameters θ.

```
RrepressilatorGeneRegulationLogDx <- function(theta, x, tvec) {
  resultDx <- array(0, c(nrow(x), ncol(x), ncol(x)))

  tm_laci = x[,1]
  tm_tetr = x[,2]
  tm_ci = x[,3]
  tp_laci = x[,4]
  tp_tetr = x[,5]
  tp_ci = x[,6]

  alpha0 = theta[1]
  alpha = theta[2]
  n = theta[3]
  beta = theta[4]

  resultDx[,1,1] = -(alpha / (1 + exp(n * tp_ci)) + alpha0) * exp(-tm_laci)
  resultDx[,6,1] = alpha * exp(-tm_laci) * (-1) *
    (1 + exp(n * tp_ci))^(-2) * n * exp(n * tp_ci)
  resultDx[,2,2] = -(alpha / (1 + exp(n * tp_laci)) + alpha0) * exp(-tm_tetr)
  resultDx[,4,2] = alpha * exp(-tm_tetr) * (-1) *
    (1 + exp(n * tp_laci))^(-2) * n * exp(n * tp_laci)
  resultDx[,3,3] = -(alpha / (1 + exp(n * tp_tetr)) + alpha0) * exp(-tm_ci)
  resultDx[,5,3] = alpha * exp(-tm_ci) * (-1) *
    (1 + exp(n * tp_tetr))^(-2) * n * exp(n * tp_tetr)

  resultDx[,1,4] = beta * exp(tm_laci - tp_laci)
  resultDx[,4,4] = -beta * exp(tm_laci - tp_laci)
  resultDx[,2,5] = beta * exp(tm_tetr - tp_tetr)
  resultDx[,5,5] = -beta * exp(tm_tetr - tp_tetr)
  resultDx[,3,6] = beta * exp(tm_ci - tp_ci)
  resultDx[,6,6] = -beta * exp(tm_ci - tp_ci)

  resultDx
}



RrepressilatorGeneRegulationLogDtheta <- function(theta, x, tvec) {
  resultDtheta <- array(0, c(nrow(x), length(theta), ncol(x)))

  tm_laci = x[,1]
  tm_tetr = x[,2]
  tm_ci = x[,3]
  tp_laci = x[,4]
  tp_tetr = x[,5]
  tp_ci = x[,6]

  p_ci = exp(tp_ci)
  p_laci = exp(tp_laci)
  p_tetr = exp(tp_tetr)

  alpha0 = theta[1]
  alpha = theta[2]
```

```
  n = theta[3]
  beta = theta[4]

  resultDtheta[,1,1] = exp(-x[,1])
  resultDtheta[,2,1] = 1 / (1 + exp(n * tp_ci)) * exp(-x[,1])
  resultDtheta[,3,1] = alpha * exp(-x[,1]) * (-1) *
    (1 + p_ci^n)^(-2) * p_ci^n * log(p_ci)
  resultDtheta[,1,2] = exp(-x[,2])
  resultDtheta[,2,2] = 1 / (1 + exp(n * tp_laci)) * exp(-x[,2])
  resultDtheta[,3,2] = alpha * exp(-x[,2]) * (-1) *
    (1 + p_laci^n)^(-2) * p_laci^n * log(p_laci)
  resultDtheta[,1,3] = exp(-x[,3])
  resultDtheta[,2,3] = 1 / (1 + exp(n * tp_tetr)) * exp(-x[,3])
  resultDtheta[,3,3] = alpha * exp(-x[,3]) * (-1) *
    (1 + p_tetr^n)^(-2) * p_tetr^n * log(p_tetr)

  resultDtheta[,4,4] = exp(x[,1] - x[,4]) - 1
  resultDtheta[,4,5] = exp(x[,2] - x[,5]) - 1
  resultDtheta[,4,6] = exp(x[,3] - x[,6]) - 1

  resultDtheta
}
```

Define parameters and settings for the experiment and MAGI:

```
# MAGI configuration
config <- list(
  nobs = 51,
  noise = rep(0.3, 6),
  kernel = "generalMatern",
  seed = 142249801,  # example seed, or choose a random seed
  niterHmc = 10001,
  filllevel = 1,
  t.end = 300,
  modelName = "repressilator-gene-regulation-log"
)

# Parameters and initial conditions
alpha <- 240 # obtain from Fig 1b in Elowitz and Leibler (2000)
KM <- 40     # scale factor only, to convert protein number to match Fig 1c in paper
pram.true <- list(
  theta=c(0.001*alpha, alpha, 2, 1/5),  # alpha0/alpha = 0.001
  x0 = log(c(0.4, 20, 40, 0.01, 0.01, 0.01)), # initial conditions
  sigma=config$noise
)
```

Use a numerical solver to generate the true trajectories to simulate data and to compare with inference from MAGI:

```
times <- seq(0,config$t.end,length=1001)

modelODE <- function(t, state, parameters) {
  list(as.vector(RrepressilatorGeneRegulationLogODE(parameters, t(state), t)))
}
```

```
xtrue <- deSolve::ode(y = pram.true$x0, times = times,
                      func = modelODE, parms = pram.true$theta)
xtrue <- data.frame(xtrue)
```

Add multiplicative noise at the observation schedule to create simulated noisy data:

```
xtrueFunc <- lapply(2:ncol(xtrue), function(j)
  approxfun(xtrue[, "time"], xtrue[, j]))

xsim <- data.frame(time = seq(0,config$t.end,length=config$nobs))
xsim <- cbind(xsim, sapply(xtrueFunc, function(f) f(xsim$time)))

set.seed(config$seed)
for(j in 1:(ncol(xsim)-1)){
  xsim[,1+j] <- xsim[,1+j]+rnorm(nrow(xsim), sd=config$noise[j])
}

xsim.obs <- xsim[seq(1,nrow(xsim), length=config$nobs),]
```

Create the `odeModel` list, then confirm ODEs and derivatives are correct:

```
xsim <- setDiscretization(xsim.obs,config$filllevel)

dynamicalModelList <- list(
  fOde=RrepressilatorGeneRegulationLogODE,
  fOdeDx=RrepressilatorGeneRegulationLogDx,
  fOdeDtheta=RrepressilatorGeneRegulationLogDtheta,
  thetaLowerBound=rep(0, 4),
  thetaUpperBound=rep(Inf, 4)
)

testDynamicalModel(dynamicalModelList$fOde, dynamicalModelList$fOdeDx,
                   dynamicalModelList$fOdeDtheta, "dynamicalModelList",
                   data.matrix(xsim.obs[-1,-1]), pram.true$theta, xsim.obs$time[-1])
```

```
## dynamicalModelList model, with derivatives
## Dx and Dtheta appear to be correct
```

```
## $testDx
## [1] TRUE
##
## $testDtheta
## [1] TRUE
```

Create inputs for MAGI:

```
# Set discretization level
xsim <- setDiscretization(xsim.obs,config$filllevel)

# Set some reasonable hyperparameters
phiExogenous <- rbind(rep(6, 6), rep(10, 6))

# Known noise level for mRNA
sigmaInit <- config$noise

# Remove initial conditions
xsim <- xsim[-1,]
```

```
# Set protein levels missing
xsim[,5:7] <- NA
xsim.obs[,5:7] <- NA
```

Now we are ready to run the MAGI method:

```
gpode <- MagiSolver(xsim, dynamicalModelList,
            control = list(niterHmc=config$niterHmc, phi=phiExogenous,
                           sigma=sigmaInit, useFixedSigma=TRUE))
```

Plot the noisy observations and inferred trajectories, to produce Fig S1:

```
xtrue <- xtrue[xtrue$time >= 1,] # remove initial conditions

xsampledexp <- exp(gpode$xsampled) # exponentiate to original scale
oursPostExpX <- cbind(
  apply(xsampledexp, 2:3, mean),
  apply(xsampledexp, 2:3, function(x) quantile(x, 0.025)),
  apply(xsampledexp, 2:3, function(x) quantile(x, 0.975)))

compnames <- c("m_lacI", "m_tetR", "m_cI",
               expression(paste("p_lacI (", bold("unobserved"), ")")),
               expression(paste("p_tetR (", bold("unobserved"), ")")),
               expression(paste("p_cI (", bold("unobserved"), ")")))
layout(rbind(c(1,2,3), c(4,5,6), c(7,7,7)), heights = c(8,8,1))
for (ii in 1:6) {

  par(mar = c(4, 4.5, 1.75, 0.1))
  ourEst <- oursPostExpX[,ii]
  ourEst <- exp(magi:::getMeanCurve(xsim$time, log(ourEst), xtrue[,1],
                                    t(phiExogenous[,ii]), 0,
                                    kerneltype=config$kernel, deriv = FALSE))

  ourUB <- oursPostExpX[,12+ii]
  ourUB <- exp(magi:::getMeanCurve(xsim$time, log(ourUB), xtrue[,1],
                                   t(phiExogenous[,ii]), 0,
                                   kerneltype=config$kernel, deriv = FALSE))


  ourLB <- oursPostExpX[,6+ii]
  ourLB <- exp(magi:::getMeanCurve(xsim$time, log(ourLB), xtrue[,1],
                                   t(phiExogenous[,ii]), 0,
                                   kerneltype=config$kernel, deriv = FALSE))
  plot( c(min(xtrue$time),max(xtrue$time)), c(min(ourLB), min(max(ourUB),175)),
        type='n', xlab='', ylab='')

  polygon(c(xtrue[,1], rev(xtrue[,1])), c(ourUB, rev(ourLB)),
          col = "skyblue", border = NA)

  if (ii == 1)
    title(ylab='mRNA concentration', cex.lab = 1.5)

  if (ii == 4)
    title(ylab='Protein concentration', cex.lab = 1.5)
```

```
  if (ii == 5)
    title(xlab='Time (mRNA lifetimes)', cex.lab = 1.5)

  lines(xtrue[, "time"], exp(xtrue[,ii+1]),col='red', lwd=2)
  lines(xtrue[,1], ourEst, col='forestgreen', lwd=1.5)
  mtext(compnames[ii], cex=1.25)
  if (ii <= 3) points(xsim.obs$time[-1], exp(xsim.obs[-1,ii+1]), col='black', pch=16)
}

par(mar = rep(0, 4))
plot(1, type = 'n', xaxt = 'n', yaxt = 'n',
     xlab = NA, ylab = NA, frame.plot = FALSE)

legend("center", c("truth", "inferred trajectory",
                   "95% interval", "noisy observations"),
       lty = c(1, 1, 0, 0), lwd = c(2, 2, 0, 1), bty = "n",
       col = c("red", "forestgreen", NA, "black"), fill = c(0, 0, "skyblue", 0),
       border = c(0, 0, "skyblue", 0), pch = c(NA, NA, 15, 16), horiz = TRUE, cex=1.25)
```

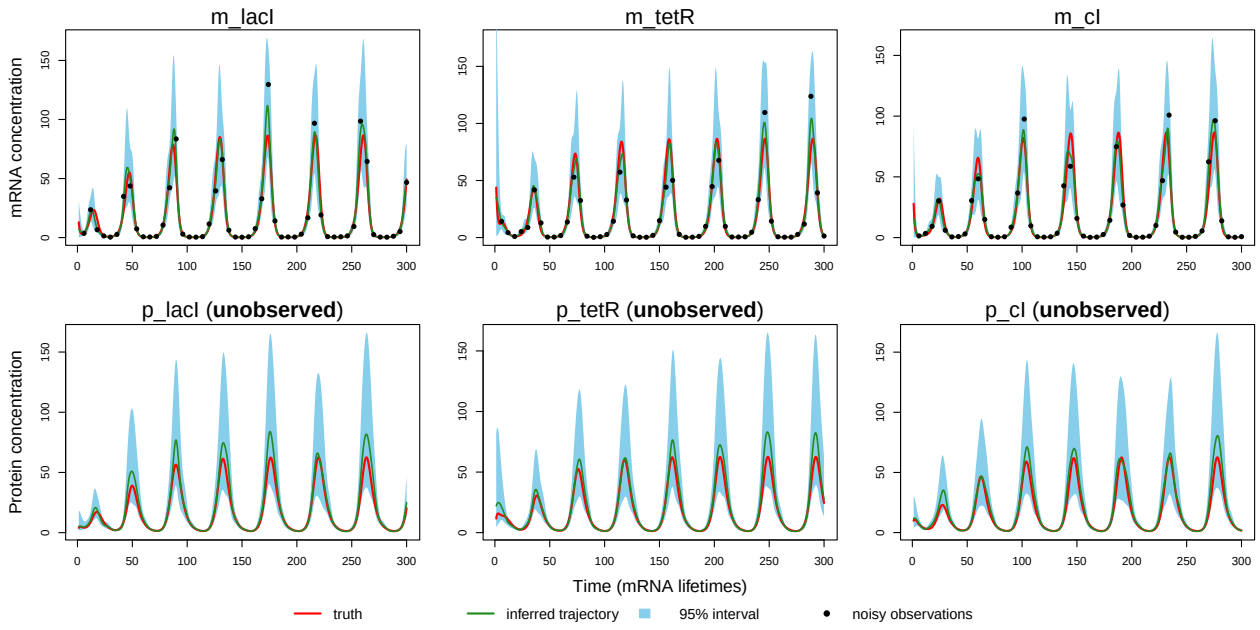

Figure S1: Inferred trajectories for a sample dataset from the repressilator gene regulation model

Plot the posterior densities of the parameters, to produce Fig S2:

```
par.names <- c( expression(alpha[0]), expression(alpha), "n", expression(beta))
par(mfrow=c(1,4))
for (ii in 1:4) {
  if (ii == 1) par(oma=c(0,1.5,0,0))
  par(mar = c(2.5, 2.5, 2, 0.75))
  den <- density(gpode$theta[,ii])
  plot(den, main='', xlab = '', ylab = '', type='n')

  value1 <- quantile(gpode$theta[,ii], 0.025)
  value2 <- quantile(gpode$theta[,ii], 0.975)
```

```
  l <- min(which(den$x >= value1))
  h <- max(which(den$x < value2))

  polygon(c(den$x[c(l, l:h, h)]),
          c(0, den$y[l:h], 0),
          col = "grey75", border=NA)
  abline(v=pram.true$theta[ii], col='red', lwd =2)

  lines(den)


  if (ii == 1) mtext(text='Posterior density',side=2,line=0,outer=TRUE)
  mtext(par.names[ii], cex=1.25)
}
```

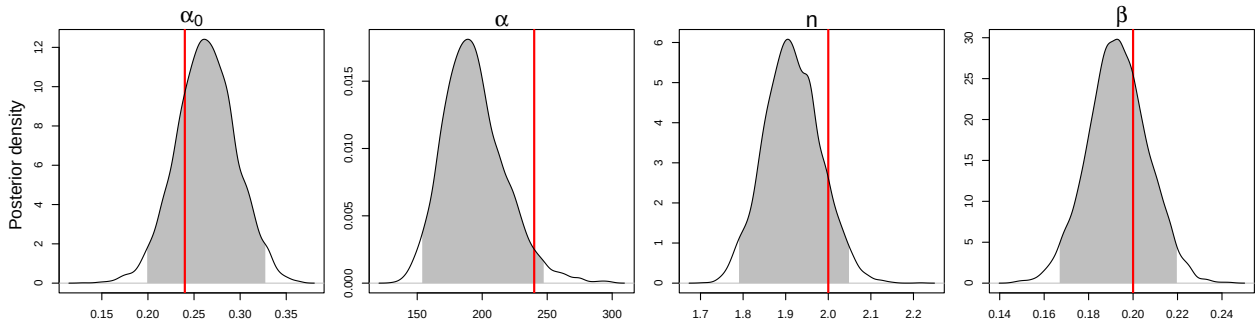

Figure S2: Bayesian posterior probability densities of system parameters for a sample dataset from the repressilator gene regulation model obtained by MAGI. The red vertical lines show the true parameter values used in the simulation. The shaded area represents the 95% interval estimate of each parameter.

Michaelis-Menten model

We begin by defining a function that codes the ODEs. Since $[E]_0 = [E] + [ES]$ is a constant, the model can be reduced to three equations.

```
RMichaelisMentenReducedODE <- function(theta, x, tvec) {
  e0 = 0.1
  e = x[,1]
  s = x[,2]
  es = e0 - e
  p = x[,3]

  resultdt <-  array(0, c(nrow(x),ncol(x)))

  resultdt[,1] = -theta[1] * e * s + (theta[2]+theta[3]) * es
  resultdt[,2] = -theta[1] * e * s + (theta[2]) * es
  resultdt[,3] = theta[3] * es

  resultdt
}
```

Next, we provide the gradients of the ODEs with respect to the system components X and the parameters θ.

```
RMichaelisMentenReducedDx <- function(theta, x, tvec) {
  resultDx <- array(0, c(nrow(x), ncol(x), ncol(x)))

  e0 = 0.1
  e = x[,1]
  s = x[,2]
  es = e0 - e
  p = x[,3]

  resultDx[,1,1] = -theta[1] * s - (theta[2] + theta[3])
  resultDx[,2,1] = -theta[1] * e

  resultDx[,1,2] = -theta[1] * s - theta[2]
  resultDx[,2,2] = -theta[1] * e

  resultDx[,1,3] = (-theta[3])

  resultDx
}

RMichaelisMentenReducedDtheta <- function(theta, x, tvec) {
  resultDtheta <- array(0, c(nrow(x), length(theta), ncol(x)))

  e0 = 0.1
  e = x[,1]
  s = x[,2]
  es = e0 - e
  p = x[,3]

  resultDtheta[,1,1] = -e * s
  resultDtheta[,2,1] = es
  resultDtheta[,3,1] = es

  resultDtheta[,1,2] = -e * s
  resultDtheta[,2,2] = es

  resultDtheta[,3,3] = es

  resultDtheta
}
```

Define parameters and settings for the experiment and MAGI:

```
# Observation times
obs.times <- c(2.5, 4.5, 7, 9.5, 11, 13.5, 15, 16, 18, 20,
               21.5, 24, 27, 29.5, 32.5, 35.5, 39.5, 45, 55, 69)

config <- list(
  nobs = length(obs.times),
  noise = c(NA, 0.02, 0.02),
  kernel = "generalMatern",
  seed = 1,
  n.iter = 5001,
  linfillspace = 0.5,
```

```
  t.start = 0,
  t.end = 70,
  phi = cbind(c(0.1, 70), c(1, 30), c(1, 30)),
  modelName = "Michaelis-Menten-Reduced"
)

pram.true <- list(
  theta=c(0.9, 0.75, 2.54),
  x0 = c(0.1, 1, 0),
  phi = config$phi,
  sigma=config$noise
)
```

Use a numerical solver to generate the true trajectories to simulate data and to compare with inference from MAGI:

```
times <- seq(0,config$t.end,length=1001)

modelODE <- function(t, state, parameters) {
  list(as.vector(RMichaelisMentenReducedODE(parameters, t(state), t)))
}

xtrue <- deSolve::ode(y = pram.true$x0, times = times,
                      func = modelODE, parms = pram.true$theta)
xtrue <- data.frame(xtrue)
```

Additive measurement noise at the observation schedule to create simulated noisy data:

```
xtrueFunc <- lapply(2:ncol(xtrue), function(j)
  approxfun(xtrue[, "time"], xtrue[, j]))

xsim <- data.frame(time = round(obs.times / config$linfillspace) * config$linfillspace)
xsim <- cbind(xsim, sapply(xtrueFunc, function(f) f(xsim$time)))
xtest <- xsim

set.seed(config$seed)
for(j in 1:(ncol(xsim)-1)){
  xsim[,1+j] <- xsim[,1+j]+rnorm(nrow(xsim), sd=config$noise[j])
}

xsim.obs <- xsim[seq(1,nrow(xsim), length=config$nobs),]
xsim.obs <- rbind(c(0, pram.true$x0), xsim.obs)
```

Create the `odeModel` list, then confirm ODEs and derivatives are correct:

```
dynamicalModelList <- list(
  fOde=RMichaelisMentenReducedODE,
  fOdeDx=RMichaelisMentenReducedDx,
  fOdeDtheta=RMichaelisMentenReducedDtheta,
  thetaLowerBound=c(0,-100,0),
  thetaUpperBound=c(Inf,Inf,Inf)
)

testDynamicalModel(dynamicalModelList$fOde, dynamicalModelList$fOdeDx,
                   dynamicalModelList$fOdeDtheta, "dynamicalModelList",
                   data.matrix(xtest[,-1]), pram.true$theta, xtest$time)
```

```
## dynamicalModelList model, with derivatives
## Dx and Dtheta appear to be correct
```

```
## $testDx
## [1] TRUE
##
## $testDtheta
## [1] TRUE
```

Create inputs for MAGI:

```
# Discretization set
xsim <- setDiscretization(xsim.obs, by = config$linfillspace)

# Linearly interpolate to initialize X, use known initial conditions
xInitExogenous <- data.matrix(xsim[,-1])
for (j in c(2,3)){
  xInitExogenous[, j] <- approx(xsim.obs$time, xsim.obs[,j+1], xsim$time)$y
  idx <- which(is.na(xInitExogenous[, j]))
  xInitExogenous[idx, j] <- xInitExogenous[idx[1] - 1, j]
}
xInitExogenous[-1, 1] <- 0.1  # fill missing E component with 0.1

# Use setSizeFactor=0 to fix initial conditions [E]=0.1, [S]=1, [P]=0
stepSizeFactor <- rep(0.01, nrow(xsim)*length(pram.true$x0) +
                        length(dynamicalModelList$thetaLowerBound) + length(pram.true$x0))
for(j in 1:3){
  for(incre in 1:1){
    stepSizeFactor[(j-1)*nrow(xsim) + incre] <- 0
  }
}
```

Run the MAGI method:

```
# Since we have manually initialized the components, skip initial optimization
gpode <- MagiSolver(xsim, dynamicalModelList,
           control = list(niterHmc=config$n.iter, stepSizeFactor = stepSizeFactor,
                          xInit = xInitExogenous, phi = pram.true$phi,
                          sigma=config$noise, useFixedSigma=TRUE,
                          skipMissingComponentOptimization=TRUE))
```

Inference for k_{cat} and K_M (posterior mean, 2.5 and 97.5 percentiles):

```
# Add KM to parameters as a function of k1, k_{-1}, k2
gpode$theta <- cbind(gpode$theta, (gpode$theta[,2]+gpode$theta[,3])/gpode$theta[,1])
pram.true$theta <- c(pram.true$theta,
                     (pram.true$theta[2]+pram.true$theta[3])/pram.true$theta[1])

par.table <- function(res) {
  par.est <- apply(cbind(res$theta[,-c(1:2)]), 2,
                   function(x) c(mean(x), quantile(x, 0.025), quantile(x, 0.975)))
  colnames(par.est) <- c("k_cat", "KM")
  rownames(par.est) <- c("Mean", "2.5%", "97.5%")
  signif(par.est, 3)
}

par.table(gpode)
```

```
##       k_cat   KM
## Mean   2.47 3.49
## 2.5%   1.93 2.58
## 97.5%  3.13 4.60
```

Plot the posterior densities of the parameters k_{cat} and K_M, to produce Fig S3:

```
par.names <- c( expression('k'['cat']), expression('K'['M']))

par(mfrow=c(1,2))
for (ii in 3:4) {
  if (ii == 3) par(oma=c(0,1.5,0,0))
  par(mar = c(2.5, 2.5, 2, 0.75))
  den <- density(gpode$theta[,ii])
  plot(den, main='', xlab = '', ylab = '', type='n')

  value1 <- quantile(gpode$theta[,ii], 0.025)
  value2 <- quantile(gpode$theta[,ii], 0.975)

  l <- min(which(den$x >= value1))
  h <- max(which(den$x < value2))

  polygon(c(den$x[c(l, l:h, h)]),
          c(0, den$y[l:h], 0),
          col = "grey75", border=NA)
  abline(v=pram.true$theta[ii], col='red', lwd =2)

  lines(den)

  if (ii == 3) mtext(text='Posterior density',side=2,line=0,outer=TRUE)
  mtext(par.names[ii-2], cex=1.25)
}
```

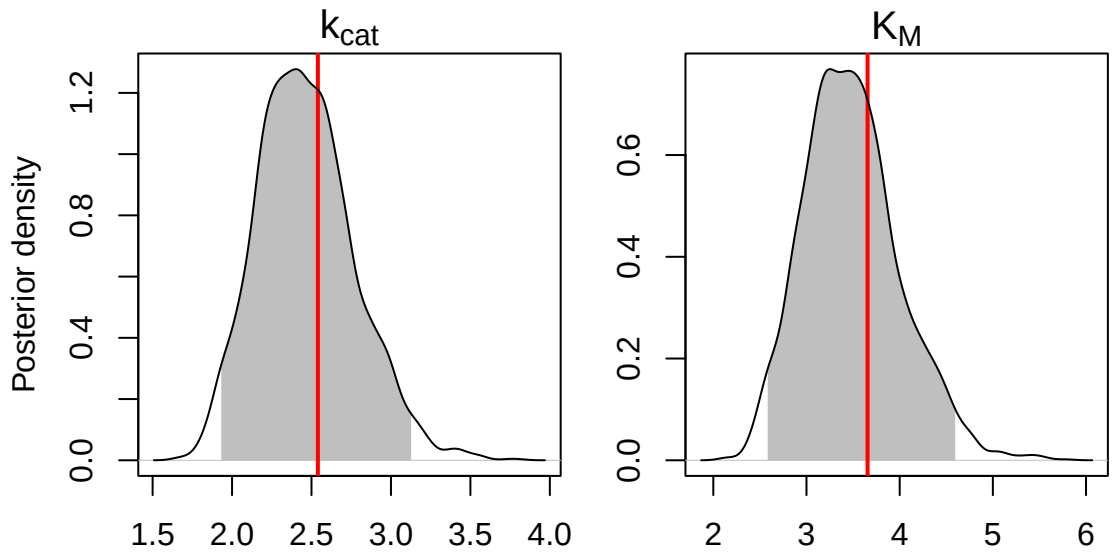

Figure S3: Bayesian posterior probability densities of k_{cat} and K_M for a sample dataset from the Michaelis-Menten model obtained by MAGI. The red vertical lines show the true parameter values used in the simulation. The shaded area represents the 95% interval estimate of each parameter.

Model for *lac* operon

We begin by defining a function that codes the ODEs:

```
RlacOperonODE <- function(theta, x, tvec) {
  ri = x[,1]
  i = x[,2]
  lactose = x[,3]
  ilactose = x[,4]
  op = x[,5]
  iop = x[,6]
  rnap = x[,7]
  rnapo = x[,8]
  r = x[,9]
  z = x[,10]

  iconstant = 1.0
  k = theta
  resultdt <-  array(0, c(nrow(x),ncol(x)))

  resultdt[,1] = k[2] * iconstant - k[13] * ri
  resultdt[,2] = k[3] * ri - k[4] * i * lactose + k[5] * ilactose -
    k[6] * i * op + k[7] * iop - k[14] * i
  resultdt[,3] = k[5] * ilactose - k[4] * i * lactose + k[15] * ilactose -
    k[12] * lactose * z
  resultdt[,4] = k[4] * i * lactose - k[5] * ilactose - k[15] * ilactose
  resultdt[,5] = k[7] * iop - k[6] * i * op - k[8] * op * rnap + (k[9] + k[10]) * rnapo
  resultdt[,6] = k[6] * i * op - k[7] * iop
  resultdt[,7] = (k[9] + k[10]) * rnapo - k[8] * op * rnap
  resultdt[,8] = k[8] * op * rnap - (k[9]+k[10]) * rnapo
  resultdt[,9] = k[10] * rnapo - k[16] * r
  resultdt[,10] = k[11]*r - k[17] * z

  resultdt
}
```

Next, we provide the gradients of the ODEs with respect to the system components X and the parameters θ.

```
RlacOperonDx <- function(theta, x, tvec) {
  resultDx <- array(0, c(nrow(x), ncol(x), ncol(x)))

  ri = x[,1]
  i = x[,2]
  lactose = x[,3]
  ilactose = x[,4]
  op = x[,5]
  iop = x[,6]
  rnap = x[,7]
  rnapo = x[,8]
  r = x[,9]
  z = x[,10]

  iconstant = 1.0
  k = theta

  resultDx[,1,1] = (-k[13])
```

```
  resultDx[,1,2] = (k[3])
  resultDx[,2,2] = -k[4] * lactose - k[6] * op - k[14]
  resultDx[,3,2] = -k[4] * i
  resultDx[,4,2] = (k[5])
  resultDx[,5,2] = -k[6] * i
  resultDx[,6,2] = (k[7])

  resultDx[,2,3] = -k[4]*lactose
  resultDx[,3,3] = -k[4]*i - k[12]*z
  resultDx[,4,3] = (k[5] + k[15])
  resultDx[,10,3] = -k[12]*lactose

  resultDx[,2,4] = k[4]*lactose
  resultDx[,3,4] = k[4]*i
  resultDx[,4,4] = (-k[5] - k[15])

  resultDx[,2,5] = -k[6] * op
  resultDx[,5,5] = -k[6] * i - k[8]*rnap
  resultDx[,6,5] = (k[7])
  resultDx[,7,5] = -k[8]*op
  resultDx[,8,5] = (k[9] + k[10])

  resultDx[,2,6] = k[6] * op
  resultDx[,5,6] = k[6] * i
  resultDx[,6,6] = (-k[7])

  resultDx[,5,7] = -k[8]*rnap
  resultDx[,7,7] = -k[8]*op
  resultDx[,8,7] = (k[9] + k[10])

  resultDx[,5,8] = k[8] * rnap
  resultDx[,7,8] = k[8] * op
  resultDx[,8,8] = (-(k[9] + k[10]))

  resultDx[,8,9] = (k[10])
  resultDx[,9,9] = (-k[16])

  resultDx[,9,10] = (k[11])
  resultDx[,10,10] = (-k[17])

  resultDx
}


RlacOperonDtheta <- function(theta, x, tvec) {
  resultDtheta <- array(0, c(nrow(x), length(theta), ncol(x)))

  ri = x[,1]
  i = x[,2]
  lactose = x[,3]
  ilactose = x[,4]
  op = x[,5]
  iop = x[,6]
```

```
  rnap = x[,7]
  rnapo = x[,8]
  r = x[,9]
  z = x[,10]

  iconstant = 1.0
  k = theta

  resultDtheta[,1,1] = (0)
  resultDtheta[,2,1] = (iconstant)
  resultDtheta[,13,1] = -ri

  resultDtheta[,3,2] = ri
  resultDtheta[,4,2] = -i*lactose
  resultDtheta[,5,2] = ilactose
  resultDtheta[,6,2] = - i*op
  resultDtheta[,7,2] = iop
  resultDtheta[,14,2] = -i

  resultDtheta[,5,3] = ilactose
  resultDtheta[,4,3] = -i*lactose
  resultDtheta[,15,3] = ilactose
  resultDtheta[,12,3] = -lactose*z

  resultDtheta[,4,4] = i*lactose
  resultDtheta[,5,4] = -ilactose
  resultDtheta[,15,4] = -ilactose

  resultDtheta[,7,5] = iop
  resultDtheta[,6,5] = -i*op
  resultDtheta[,8,5] = -op*rnap
  resultDtheta[,9,5] = rnapo
  resultDtheta[,10,5] = rnapo

  resultDtheta[,6,6] = i*op
  resultDtheta[,7,6] = -iop

  resultDtheta[,9,7] = rnapo
  resultDtheta[,10,7] = rnapo
  resultDtheta[,8,7] = -op*rnap

  resultDtheta[,8,8] = op*rnap
  resultDtheta[,9,8] = -rnapo
  resultDtheta[,10,8] = -rnapo

  resultDtheta[,10,9] = rnapo
  resultDtheta[,16,9] = -r

  resultDtheta[,11,10] = r
  resultDtheta[,17,10] = -z

  resultDtheta
}
```

Define parameters and settings for the experiment and MAGI:

```
noisefac <- 0.05
config <- list(
  noise = c(0.0236194228362474, 0.984379168607761, 0.040561105491602, 0.19224800630503,
            0.000104872420893985, 0.395628293932429, 99.3980508395468, 0.0263189487352539,
            0.00631925202022132, 0.00036869187211123)*noisefac,
  # noise level is minimum of each component * noisefac
  kernel = "generalMatern",
  seed = 115767108,
  hmcSteps = 500,
  niterHmc = 20001,
  fillinterval = 15,
  t.start = 1,
  t.end = 1201,
  modelName = "lac-operon",
  obs.times = c(seq(1,361,by=15), seq(391,601,by=30), 901, 1201)
)
config$nobs = length(config$obs.times)

pram.true <- list(
  theta=c(1, 0.02, 0.1, 0.005, 0.1, 1, 0.01, 0.1, 0.01, 0.03,
          0.1, 0.001, 0.01, 0.002, 0.002, 0.01, 0.001),
  x0 = c(0, 50, 1000, 0, 1, 0, 100, 0, 0, 0),
  sigma=config$noise
)
```

Use a numerical solver to generate the true trajectories to simulate data and to compare with inference from MAGI:

```
times <- seq(0,config$t.end,by = 0.01)

modelODE <- function(t, state, parameters) {
  list(as.vector(RlacOperonODE(parameters, t(state), t)))
}

xtrue <- deSolve::ode(y = pram.true$x0, times = times,
                      func = modelODE, parms = pram.true$theta)
xtrue <- data.frame(xtrue)
```

Additive measurement noise at the observation schedule to create simulated noisy data:

```
xtrueFunc <- lapply(2:ncol(xtrue), function(j)
  approxfun(xtrue[, "time"], xtrue[, j]))

xsim <- data.frame(time = config$obs.times)
xsim <- cbind(xsim, sapply(xtrueFunc, function(f) f(xsim$time)))

set.seed(config$seed)
for(j in 1:(ncol(xsim)-1)){
  xsim[,1+j] <- xsim[,1+j]+rnorm(nrow(xsim), sd=config$noise[j])
}
xsim.obs <- xsim
```

Create the `odeModel` list, then confirm ODEs and derivatives are correct:

```
dynamicalModelList <- list(
  fOde=RlacOperonODE,
  fOdeDx=RlacOperonDx,
  fOdeDtheta=RlacOperonDtheta,
  thetaLowerBound=rep(0, 17),
  thetaUpperBound=rep(Inf, 17),
  name="lac-operon"
)

testDynamicalModel(dynamicalModelList$fOde, dynamicalModelList$fOdeDx,
                   dynamicalModelList$fOdeDtheta, "dynamicalModelList",
                   data.matrix(xsim.obs[,-1]), pram.true$theta, xsim$time)
```

```
## dynamicalModelList model, with derivatives
## Dx and Dtheta appear to be correct
```

```
## $testDx
## [1] TRUE
##
## $testDtheta
## [1] TRUE
```

Create inputs for MAGI:

```
# Discretization set
xsim <- setDiscretization(xsim.obs, by = config$fillinterval)

# Rough hyperparameter values based on smoothness and level of each component
phiExogenous <- cbind(
  c(2.5, 600),
  c(100, 140),
  c(1000, 200),
  c(60, 100),
  c(0.01, 800),
  c(1, 200),
  c(500, 300),
  c(0.5, 300),
  c(1, 400),
  c(35, 1000)
)
```

Run the MAGI method:

```
gpode <- MagiSolver(xsim, dynamicalModelList,
            control = list(niterHmc=config$niterHmc, nstepsHmc = config$hmcSteps,
                           phi=phiExogenous, sigma=config$noise, useFixedSigma=TRUE))
```

Inference for parameters $k_1, \ldots, k_{16}$:

```
par.table <- function(res) {
  par.est <- apply(cbind(res$theta[,-1]), 2,
                   function(x) c(mean(x), quantile(x, 0.025), quantile(x, 0.975)))
  colnames(par.est) <- paste0('k', 1:16)
  rownames(par.est) <- c("Mean", "2.5%", "97.5%")
  signif(par.est, 3)
}
```

```
par.table(gpode)
```

```
##           k1     k2      k3     k4    k5      k6     k7      k8     k9    k10
## Mean  0.0198 0.0966 0.00425 0.0856 0.916 0.00919 0.0982 0.00874 0.0299 0.1000
## 2.5%  0.0193 0.0838 0.00389 0.0786 0.845 0.00842 0.0918 0.00610 0.0291 0.0998
## 97.5% 0.0203 0.1090 0.00465 0.0935 0.989 0.00999 0.1050 0.01140 0.0308 0.1000
##            k11     k12     k13     k14     k15      k16
## Mean  0.001000 0.00990 0.00192 0.00189 0.01000 0.001000
## 2.5%  0.000998 0.00961 0.00161 0.00150 0.00966 0.000992
## 97.5% 0.001000 0.01020 0.00221 0.00227 0.01030 0.001010
```

Calculate reconstructed trajectories:

```
tvecsolve <- seq(config$t.start,config$t.end,by = 0.1)
calcTraj <- function(res) {
  x0.est <- apply(res$xsampled[,1,],2,mean)
  theta.est <- apply(res$theta,2,mean)

  x <- deSolve::ode(y = x0.est, times = tvecsolve,
                    func = modelODE, parms = theta.est)
  x
}

recon <- calcTraj(gpode)
recon.obs <- subset(recon, time %in% xsim$time)
xtrue.obs <- subset(xtrue, time %in% xsim$time)[,-1]
```

Visualize the reconstructed trajectories together with the observations, as shown in Fig S4:

```
compnames <- c("r_I", "I", "Lactose", "ILactose", "Op",
               "IOp", "RNAP", "RNAPo", "r", "Z")
par(oma=c(0,1.5,0,0))
layout(rbind(c(1:5), c(6:10), c(11,11,11,11,11)), heights = c(8,8,1))

for (i in 1:10) {
  par(mar = c(4, 2.5, 1.75, 0.1))
  plot(c(min(tvecsolve), max(tvecsolve)),
       c(min(c(recon[,i+1], xtrue[xtrue$time >=1,i+1] - config$noise[i]*5)),
         max(c(recon[,i+1], xtrue[xtrue$time>=1,i+1] + config$noise[i]*5))),
       type = 'n', ylab='', xlab='')
  mtext(compnames[i])

  lines(xtrue$time[xtrue$time >=1], xtrue[xtrue$time >=1,i+1], col="red", lwd=2)
  lines(tvecsolve, recon[,i+1], col="forestgreen", lwd=1.5)

  points(xsim[,1], xsim[,i+1], pch=16)

  if (i == 8) title(xlab='Time (sec)', cex.lab = 1.5)
  if (i == 1) mtext("       Concentration (arb. unit)",side=2,line=0,outer=TRUE)
}

par(mar=rep(0,4))
plot(1,type='n', xaxt='n', yaxt='n', xlab=NA, ylab=NA, frame.plot = FALSE)

legend("center",  c("truth", "reconstructed trajectory", "observations"),
```

```
       lty = c(1, 1, 0), lwd = c(2, 2, 0), bty = "n",
       col = c("red", "forestgreen", "black"), fill = c(0, 0, 0),
       border = c(0, 0, 0), pch = c(NA, NA, 16), horiz = TRUE, cex = 1.25)
```

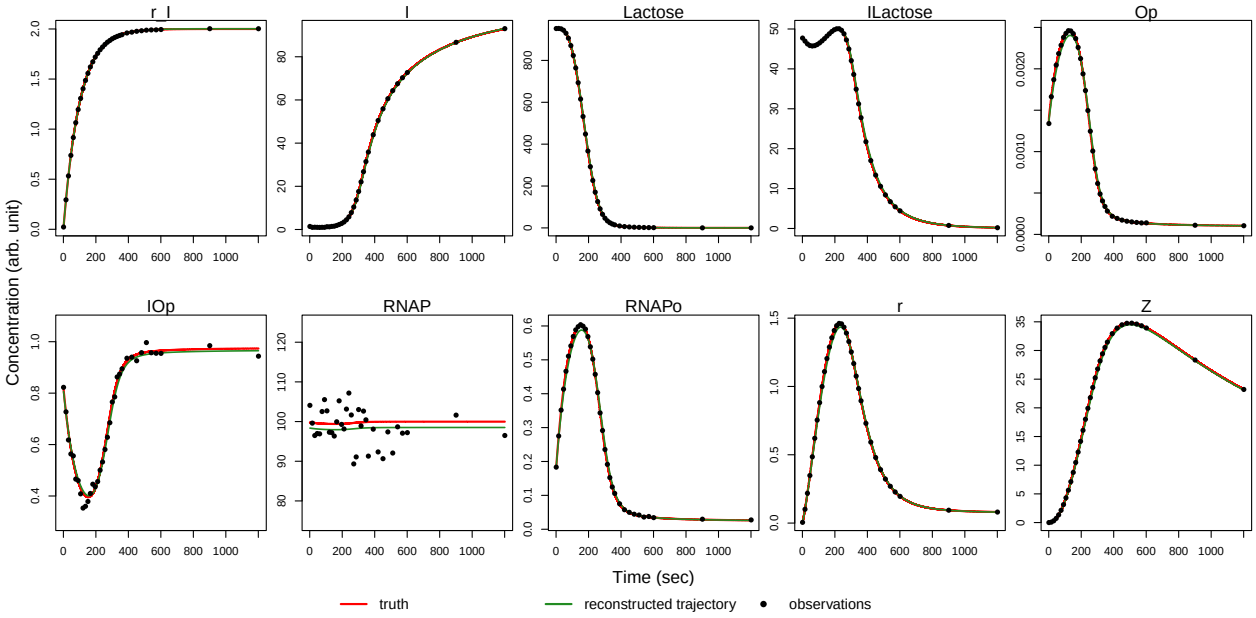

Figure S4: Reconstructed trajectories for a sample dataset simulated from the *lac* operon model.

Model comparison

We begin by defining a function that codes the ODEs for the inhibitor model:

```
RMichaelisMentenInhibitor6ODE <- function(theta, x, tvec) {
  resultdt <-  array(0, c(nrow(x),ncol(x)))

  e0 = 0.1
  e = x[,1]
  s = x[,2]
  p = x[,3]
  i = x[,4]
  ei = x[,5]
  es = x[,6]

  resultdt[,1] = -theta[1] * e * s + (theta[2]+theta[3]) * es -
    theta[4] * i * e + theta[5] * ei
  resultdt[,2] = -theta[1] * e * s + (theta[2]) * es
  resultdt[,3] = theta[3] * es
  resultdt[,4] = -theta[4] * i * e + theta[5] * ei
  resultdt[,5] = theta[4] * i * e - theta[5] * ei
  resultdt[,6] = theta[1] * e * s  - (theta[2]+theta[3]) * es

  resultdt
}
```

Next, we provide the gradients of the ODEs with respect to the system components X and the parameters θ.

```
RMichaelisMentenInhibitor6Dx <- function(theta, x, tvec) {
  resultDx <- array(0, c(nrow(x), ncol(x), ncol(x)))

  e0 = 0.1
  e = x[,1]
  s = x[,2]
  p = x[,3]
  i = x[,4]
  ei = x[,5]
  es = x[,6]

  resultDx[,1,1] = -theta[1] * s - theta[4] * i
  resultDx[,2,1] = -theta[1] * e
  resultDx[,4,1] = -theta[4] * e
  resultDx[,5,1] = (theta[5])
  resultDx[,6,1] = (theta[2]+theta[3])

  resultDx[,1,2] = -theta[1] * s
  resultDx[,2,2] = -theta[1] * e
  resultDx[,6,2] = (theta[2])

  resultDx[,6,3] = (theta[3])

  resultDx[,1,4] = -theta[4] * i
  resultDx[,4,4] = -theta[4] * e
  resultDx[,5,4] = (theta[5])

  resultDx[,1,5] = theta[4] * i
  resultDx[,4,5] = theta[4] * e
  resultDx[,5,5] = (-theta[5])

  resultDx[,1,6] = theta[1] * s
  resultDx[,2,6] = theta[1] * e
  resultDx[,6,6] = (-(theta[2]+theta[3]))

  resultDx
}

RMichaelisMentenInhibitor6Dtheta <- function(theta, x, tvec) {
  resultDtheta <- array(0, c(nrow(x), length(theta), ncol(x)))

  e0 = 0.1
  e = x[,1]
  s = x[,2]
  p = x[,3]
  i = x[,4]
  ei = x[,5]
  es = x[,6]

  resultDtheta[,1,1] = -e * s
  resultDtheta[,2,1] = es
  resultDtheta[,3,1] = es
  resultDtheta[,4,1] = -i * e
```

```
  resultDtheta[,5,1] = ei

  resultDtheta[,1,2] = -e * s
  resultDtheta[,2,2] = es

  resultDtheta[,3,3] = es

  resultDtheta[,4,4] = -i * e
  resultDtheta[,5,4] = ei

  resultDtheta[,4,5] = i * e
  resultDtheta[,5,5] = -ei

  resultDtheta[,1,6] = e * s
  resultDtheta[,2,6] = -es
  resultDtheta[,3,6] = -es

  resultDtheta
}
```

Define parameters and settings for the experiment:

```
# Train time points (used for fitting)
obs.times <- c(2.5, 4.5, 7, 9.5, 11, 13.5, 15, 16, 18, 20)
# Test time points (not used for fiting)
test.times <- c(21.5, 24, 27, 29.5, 32.5, 35.5, 39.5, 45, 55, 69)

config <- list(
  nobs = length(obs.times),
  noise = c(NA, 0.02, 0.02, NA, NA, NA),
  kernel = "generalMatern",
  seed = 669097609,
  n.iter = 20001,
  linfillspace = 0.5,
  t.end = 70,
  modelName = "MM-Inhibitor"
)

pram.true <- list(
  theta=c(0.9, 0.75, 2.54, 1, 0.5),
  x0 = c(0.1, 1, 0, 0.2, 0, 0),
  phi = cbind(c(0.1, 70), c(1, 30), c(1, 30), c(1, 70), c(1, 70), c(1, 70)),
  sigma=config$noise
)
```

Use a numerical solver to generate the true trajectories under the inhibitor model:

```
times <- seq(0,config$t.end,length=1001)

modelODE <- function(t, state, parameters) {
  list(as.vector(RMichaelisMentenInhibitor6ODE(parameters, t(state), t)))
}

xtrue <- deSolve::ode(y = pram.true$x0, times = times,
                      func = modelODE, parms = pram.true$theta)
```

```
xtrue <- data.frame(xtrue)

xtrueFunc <- lapply(2:ncol(xtrue), function(j)
  approxfun(xtrue[, "time"], xtrue[, j]))

xsim <- data.frame(time = round(c(obs.times,test.times) / config$linfillspace) *
                     config$linfillspace)
xsim <- cbind(xsim, sapply(xtrueFunc, function(f) f(xsim$time)))
xtestDS <- xsim
```

Create simulated noisy data, and then divide into train/test parts:

```
set.seed(config$seed)
for(j in 1:(ncol(xsim)-1)){
  xsim[,1+j] <- xsim[,1+j]+rnorm(nrow(xsim), sd=config$noise[j])
}

# Divide into train/test
xtest <- xsim[xsim$time %in% test.times,]
xsim <- xsim[xsim$time %in% obs.times,]

xsim.obs <- rbind(c(0, pram.true$x0), xsim) # tack on initial conditions
```

Create the `odeModel` list, then confirm ODEs and derivatives are correct:

```
dynamicalModelList <- list(
  fOde=RMichaelisMentenInhibitor6ODE,
  fOdeDx=RMichaelisMentenInhibitor6Dx,
  fOdeDtheta=RMichaelisMentenInhibitor6Dtheta,
  thetaLowerBound=c(0,-100,0,0,-100),
  thetaUpperBound=c(Inf,Inf,Inf,Inf,Inf)
)

testDynamicalModel(dynamicalModelList$fOde, dynamicalModelList$fOdeDx,
                   dynamicalModelList$fOdeDtheta, "dynamicalModelList",
                   data.matrix(xtestDS[,-1]), pram.true$theta, xtestDS$time)
```

```
## dynamicalModelList model, with derivatives
## Dx and Dtheta appear to be correct
```

```
## $testDx
## [1] TRUE
##
## $testDtheta
## [1] TRUE
```

Create inputs to MAGI for inhibitor model:

```
# Discretization set
xsim <- setDiscretization(rbind(xsim.obs, c(config$t.end, rep(NaN,ncol(xsim)-1))),
                          by=config$linfillspace)
xsim[1,5] <- NaN  # do not observe initial I value, other initial conditions known

# Use setSizeFactor=0 to fix initial conditions
# [E]=0.1, [S]=1, [P]=0, [ES]=0, except for unknown I
stepSizeFactor <- rep(0.01, nrow(xsim)*length(pram.true$x0) +
                        length(dynamicalModelList$thetaLowerBound) + length(pram.true$x0))
```

```
for(j in c(1,2,3,5,6)){
  for(incre in 1:1){
    stepSizeFactor[(j-1)*nrow(xsim) + incre] <- 0
  }
}

# Initialize X matrix for HMC sampling with some naive values
xInitExogenous <- matrix(NA, nrow=nrow(xsim[,-1]), ncol=ncol(xsim[,-1]))
xInitExogenous[,1] <- 0.1
xInitExogenous[,2] <- 1
xInitExogenous[,3] <- 0
xInitExogenous[,4] <- 0.1
xInitExogenous[,5] <- 0
xInitExogenous[,6] <- 0
```

Run the MAGI method under the inhibitor model:

```
# Use the option positiveSystem = TRUE since all components are non-negative
gpode <- MagiSolver(xsim, dynamicalModelList,
            control = list(xInit = xInitExogenous, niterHmc=config$n.iter,
                           stepSizeFactor = stepSizeFactor, positiveSystem = TRUE,
                           skipMissingComponentOptimization = TRUE,
                           phi = pram.true$phi, sigma=config$noise, useFixedSigma=TRUE))
```

Calculate sum of square errors for fitting and prediction for inhibitor model:

```
xMean <- apply(gpode$xsampled, c(2, 3), mean)
fit_inhib <- xMean[gpode$tvec %in% xsim.obs[,"time"],2:3]
pred_inhib <- xMean[gpode$tvec %in% xtest[,"time"],2:3]
sse_train_inhib <- sum((fit_inhib - xsim.obs[,3:4])^2)
sse_test_inhib <-  sum((pred_inhib - xtest[,3:4])^2)

ourEst_inhib <- apply(gpode$xsampled, c(2, 3), mean)
ourLB_inhib <- apply(gpode$xsampled, c(2, 3), function(x) quantile(x, 0.025))
ourUB_inhib <- apply(gpode$xsampled, c(2, 3), function(x) quantile(x, 0.975))
```

Create inputs to MAGI for fitting with Michaelis-Menten model:

```
dynamicalModelListReduced <- list(
  fOde=RMichaelisMentenReducedODE,
  fOdeDx=RMichaelisMentenReducedDx,
  fOdeDtheta=RMichaelisMentenReducedDtheta,
  thetaLowerBound=c(0,-100,0),
  thetaUpperBound=c(Inf,Inf,Inf)
)

# Use setSizeFactor=0 to fix initial conditions [E]=0.1, [S]=1, [P]=0
stepSizeFactor <- rep(0.01, (nrow(xsim))*(length(pram.true$x0)-3) +
    length(dynamicalModelListReduced$thetaLowerBound) + length(pram.true$x0) - 3)
for(j in 1:3){
  for(incre in 1:1){
    stepSizeFactor[(j-1)*nrow(xsim) + incre] <- 0
  }
}
```

Run the MAGI method under the Michaelis-Menten model:

```
gpode <- MagiSolver(xsim[,c(2,3,4)], dynamicalModelListReduced, xsim$time,
            control = list(xInit = xInitExogenous[,1:3], niterHmc=config$n.iter,
                           stepSizeFactor = stepSizeFactor, positiveSystem = TRUE,
                           skipMissingComponentOptimization = TRUE,
                           phi = pram.true$phi[,1:3], sigma=config$noise[1:3],
                           useFixedSigma=TRUE))
```

Calculate sum of square errors for fitting and prediction for Michaelis-Menten model:

```
xMean <- apply(gpode$xsampled, c(2, 3), mean)
fit_vanil <- xMean[gpode$tvec %in% xsim.obs[,"time"],2:3]
pred_vanil <- xMean[gpode$tvec %in% xtest[,"time"],2:3]
sse_train_vanil <- sum((fit_vanil - xsim.obs[,3:4])^2)
sse_test_vanil <-  sum((pred_vanil - xtest[,3:4])^2)

ourEst_vanil <- apply(gpode$xsampled, c(2, 3), mean)
ourLB_vanil <- apply(gpode$xsampled, c(2, 3), function(x) quantile(x, 0.025))
ourUB_vanil <- apply(gpode$xsampled, c(2, 3), function(x) quantile(x, 0.975))
```

Summary of fitting and prediction errors for each model:

```
print(paste0("Inhibitor: SSE(train) = ", round(sse_train_inhib,3),
             ", SSE(test) = ", round(sse_test_inhib,3)))
```

```
## [1] "Inhibitor: SSE(train) = 0.009, SSE(test) = 0.009"
```

```
print(paste0("M-M: SSE(train) = ", round(sse_train_vanil,3),
             ", SSE(test) = ", round(sse_test_vanil,3)))
```

```
## [1] "M-M: SSE(train) = 0.016, SSE(test) = 0.035"
```

Visualizations of the observations, model fits and predictions, as shown in Fig S5:

```
compnames <- c("", "[S]", "[P]")

layout(cbind(c(1,1,6,6),c(2,2,4,4),c(3,3,5,5)))
par(mar = c(4, 4.5, 1.75, 0.1))

matplot(xtrue[, "time"], (xtrue[, -1]), type="n", lty=1, col=0, xlab='', ylab='mM')
title(xlab="Time (min)", line=2, cex.lab=1)
abline(v = max(obs.times), col="grey", lty=2, lwd=2)
matplot(xsim.obs$time, (xsim.obs[,3:4]), type="p", col=c(1,2), pch=19, add = TRUE)
matplot(xtest$time, xtest[,3:4], type="p", col=c(1,2), pch=5, add = TRUE)

mtext('observations', line = 0.3)

for (ii in 3:2) {

  par(mar = c(4, 4.5, 1.75, 0.1))
  ourEstp <- magi:::getMeanCurve(xsim$time, ourEst_inhib[,ii], xtrue[,1],
                                 t(pram.true$phi[,ii]), 0,
                                 kerneltype=config$kernel, deriv = FALSE)

  ourUBp <- magi:::getMeanCurve(xsim$time, ourUB_inhib[,ii], xtrue[,1],
                                t(pram.true$phi[,ii]), 0,
                                kerneltype=config$kernel, deriv = FALSE)
```

```
  ourLBp <- magi:::getMeanCurve(xsim$time, ourLB_inhib[,ii], xtrue[,1],
                                t(pram.true$phi[,ii]), 0,
                                kerneltype=config$kernel, deriv = FALSE)
  plot( c(min(xtrue$time),max(xtrue$time)), c(min(ourLBp), min(max(ourUBp),175)),
        type='n', xlab='', ylab='mM')
  title(xlab="Time (min)", line=2, cex.lab=1)
  abline(v = max(obs.times), col="grey", lty=2, lwd=2)

  polygon(c(xtrue[xtrue[,1] <= max(obs.times),1],
            rev(xtrue[xtrue[,1] <= max(obs.times),1])),
          c(ourUBp[xtrue[,1] <= max(obs.times)],
            rev(ourLBp[xtrue[,1] <= max(obs.times)])), col = "skyblue", border = NA)
  polygon(c(xtrue[xtrue[,1] > max(obs.times),1],
            rev(xtrue[xtrue[,1] > max(obs.times),1])),
          c(ourUBp[xtrue[,1] > max(obs.times)],
            rev(ourLBp[xtrue[,1] > max(obs.times)])), col = "peachpuff", border = NA)

  lines(xtrue[,1], ourEstp, col='forestgreen', lwd=1.5)
  mtext(paste(compnames[ii], "inferred from inhibitor model"), line = 0.3)

  if(compnames[ii] == "[P]"){
    point_col = "red"
  }else{
    point_col = "black"
  }
  points(xsim$time, xsim[,ii+1], col=point_col, pch=16)
  points(xtest$time, xtest[,ii+1], col=point_col, pch=5)
}

for (ii in 3:2) {

  par(mar = c(4, 4.5, 1.75, 0.1))
  ourEstp <- magi:::getMeanCurve(xsim$time, ourEst_vanil[,ii], xtrue[,1],
                                 t(pram.true$phi[,ii]), 0,
                                 kerneltype=config$kernel, deriv = FALSE)

  ourUBp <- magi:::getMeanCurve(xsim$time, ourUB_vanil[,ii], xtrue[,1],
                                t(pram.true$phi[,ii]), 0,
                                kerneltype=config$kernel, deriv = FALSE)

  ourLBp <- magi:::getMeanCurve(xsim$time, ourLB_vanil[,ii], xtrue[,1],
                                t(pram.true$phi[,ii]), 0,
                                kerneltype=config$kernel, deriv = FALSE)
  plot( c(min(xtrue$time),max(xtrue$time)), c(min(ourLBp), min(max(ourUBp),175)),
        type='n', xlab='', ylab='mM')
  title(xlab="Time (min)", line=2, cex.lab=1)
  abline(v = max(obs.times), col="grey", lty=2, lwd=2)

  polygon(c(xtrue[xtrue[,1] <= max(obs.times),1],
            rev(xtrue[xtrue[,1] <= max(obs.times),1])),
          c(ourUBp[xtrue[,1] <= max(obs.times)],
            rev(ourLBp[xtrue[,1] <= max(obs.times)])), col = "skyblue", border = NA)
  polygon(c(xtrue[xtrue[,1] > max(obs.times),1],
```

```
            rev(xtrue[xtrue[,1] > max(obs.times),1])),
          c(ourUBp[xtrue[,1] > max(obs.times)],
            rev(ourLBp[xtrue[,1] > max(obs.times)])), col = "peachpuff", border = NA)

  lines(xtrue[,1], ourEstp, col='forestgreen', lwd=1.5)
  mtext(paste(compnames[ii], "inferred from M-M model"), line = 0.3)

  if(compnames[ii] == "[P]"){
    point_col = "red"
  }else{
    point_col = "black"
  }
  points(xsim$time, xsim[,ii+1], col=point_col, pch=16)
  points(xtest$time, xtest[,ii+1], col=point_col, pch=5)

}

par(mar=rep(0,4))
plot(1,type='n', xaxt='n', yaxt='n', xlab=NA, ylab=NA, frame.plot = FALSE)

oos_bg_col = "peachpuff"

legend("center", c("observed noisy [S] for training", "observed noisy [S] for prediction",
                   "observed noisy [P] for training", "observed noisy [P] for prediction",
                   "inferred trajectory", "95% interval in training",
                   "95% interval in prediction"),
       lty=c(0,0,0,0,1,0,0), lwd=c(0,1,0,1,3,0,0),
       col = c(1,1,"red","red", "forestgreen", NA, NA),
       fill=c(0,0,0,0, 0,"skyblue",oos_bg_col),
       border=c(0,0,0,0, 0, "skyblue",oos_bg_col), pch=c(19,5,19,5, NA, 15, 15), cex=1.7)
```

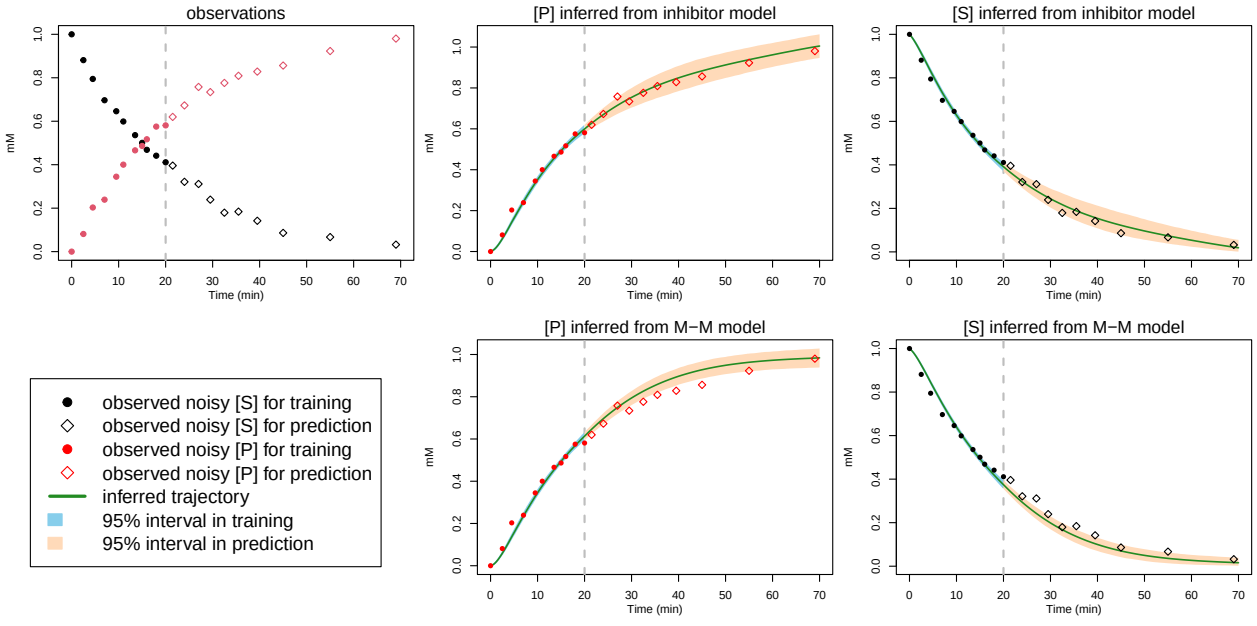

Figure S5: Michaelis-Menten model comparison for a sample dataset simulated with inhibitor.