Package ‘magi’

January 14, 2022

Type Package

Title MAnifold-Constrained Gaussian Process Inference

Version 1.1.5

Date 2022-01-12

Encoding UTF-8

Description
Provides fast and accurate inference for the parameter estimation problem in Ordinary Differential Equations, including the case when there are unobserved system components. Implements the MAGI method
(MAnifold-constrained Gaussian process Inference) of Yang, Wong, and Kou (2021) <doi:10.1073/pnas.2020397118>.

License MIT + file LICENSE

VignetteBuilder knitr

Imports Rcpp (>= 1.0.6), gridExtra, gridBase, grid, methods, deSolve

LinkingTo Rcpp, RcppArmadillo, BH, roptim

RoxygenNote 7.1.1

Suggests testthat, mvtnorm, covr, knitr, MASS, rmarkdown, markdown

NeedsCompilation yes

Author Shihao Yang [aut, cre] (<`https://orcid.org/0000-0003-3910-4969`>),
Samuel W.K. Wong [aut],
S.C. Kou [ctb, cph] (Contributor of MAGI method development)

Maintainer Shihao Yang <`shihao.yang@isye.gatech.edu`>

Repository CRAN

Date/Publication 2022-01-14 11:52:51 UTC

R topics documented:

`calCov` *Calculate stationary Gaussian process kernel*

Description

Covariance calculations for Gaussian process kernels. Currently supports matern, rbf, compact1, periodicMatern, generalMatern, and rationalQuadratic kernels. Can also return m_phi and other additional quantities useful for ODE inference.

Usage

```
calCov(
  phi,
  rInput,
  signrInput,
  bandsize = NULL,
  complexity = 3,
  kerneltype = "matern",
  df,
  noiseInjection = 1e-07
)
```

Arguments

`phi`	the kernel hyper-parameters. See details for hyper-parameter specification for each `kerneltype`.
`rInput`	the distance matrix between all time points s and t, i.e., |s - t|
`signrInput`	the sign matrix of the time differences, i.e., sign(s - t)
`bandsize`	size for band matrix approximation. See details.
`complexity`	integer value for the complexity of the kernel calculations desired: • 0 includes C only • 1 additionally includes Cprime, Cdoubleprime, dCdphi • 2 or above additionally includes Ceigen1over, CeigenVec, Cinv, mphi, Kphi, Keigen1over, KeigenVec, Kinv, mphiLeftHalf, dCdphiCube See details for their definitions.
`kerneltype`	must be one of `matern`, `rbf`, `compact1`, `periodicMatern`, `generalMatern`, `rationalQuadratic`. See details for the kernel formulae.
`df`	degrees of freedom, for `generalMatern` and `rationalQuadratic` kernels only. Default is `df=2.01` for `generalMatern` and `df=0.01` for `rationalQuadratic`.

`noiseInjection`	a small value added to the diagonal elements of C and Kphi for numerical stability

Details

The covariance formulae and the hyper-parameters `phi` for the supported kernels are as follows. Stationary kernels have $C(s,t) = C(r)$ where $r = |s - t|$ is the distance between the two time points. Generally, the hyper-parameter `phi[1]` controls the overall variance level while `phi[2]` controls the bandwidth.

`matern` This is the simplified Matern covariance with `df = 5/2`:

$$C(r) = phi[1] * (1 + \sqrt{5}r/phi[2] + 5r^2/(3phi[2]^2)) * \exp(-\sqrt{5}r/phi[2])$$

`rbf`

$$C(r) = phi[1] * \exp(-r^2/(2phi[2]^2))$$

`compact1`

$$C(r) = phi[1] * \max(1 - r/phi[2], 0)^4 * (4r/phi[2] + 1)$$

`periodicMatern` Define $r' = |\sin(r\pi/phi[3]) * 2|$. Then the covariance is given by $C(r')$ using the Matern formula.

`generalMatern`

$$C(r) = phi[1]*2^(1-df)/\Gamma(df)*(\sqrt(2.0*df)*r/phi[2])^df*besselK(\sqrt(2.0*df)*r/phi[2], df)$$

where `besselK` is the modified Bessel function of the second kind.

`rationalQuadratic`

$$C(r) = phi[1] * (1 + r^2/(2df phi[2]^2))^(- df)$$

The kernel calculations available and their definitions are as follows:

C The covariance matrix corresponding to the distance matrix `rInput`.

Cprime The cross-covariance matrix $dC(s,t)/ds$.

Cdoubleprime The cross-covariance matrix $d^2C(s,t)/dsdt$.

dCdphi A list with the matrices $dC/dphi$ for each element of phi.

Ceigen1over The reciprocals of the eigenvalues of C.

CeigenVec Matrix of eigenvectors of C.

Cinv The inverse of C.

mphi The matrix `Cprime * Cinv`.

Kphi The matrix `Cdoubleprime -Cprime * Kinv * t(Cprime)`.

Keigen1over The reciprocals of the eigenvalues of Kphi.

Kinv The inverse of Kphi.

mphiLeftHalf The matrix `Cprime * CeigenVec`.

dCdphiCube $dC/dphi$ as a 3-D array, with the third dimension corresponding to the elements of phi.

If `bandsize` is a positive integer, additionally CinvBand, mphiBand, and KinvBand are provided in the return list, which are band matrix approximations to Cinv, mphi, and Kinv with the specified `bandsize`.

Value

A list containing the kernel calculations included by the value of `complexity`.

Examples

```
foo  <- outer(0:40, t(0:40),'-')[,1,]
r <- abs(foo)
signr <- -sign(foo)
calCov(c(0.2, 2), r, signr, bandsize=20, kerneltype="generalMatern", df=2.01)
```

`gpsmoothing` *Gaussian process smoothing*

Description

Estimate hyper-parameters `phi` and noise standard deviation `sigma` for a vector of observations using Gaussian process smoothing.

Usage

```
gpsmoothing(yobs, tvec, kerneltype = "generalMatern", sigma = NULL)
```

Arguments

`yobs`	vector of observations
`tvec`	vector of time points corresponding to observations
`kerneltype`	the covariance kernel, types `matern`, `compact1`, `periodicMatern`, `generalMatern` are supported. See `calCov` for their definitions.
`sigma`	the noise level (if known). By default, both `phi` and `sigma` are estimated. If a value for `sigma` is supplied, then `sigma` is held fixed at the supplied value and only `phi` is estimated.

Value

A list containing the elements `phi` and `sigma` with their estimated values.

Examples

```
# Sample data and observation times
tvec <- seq(0, 20, by = 0.5)
y <- c(-1.16, -0.18, 1.57, 1.99, 1.95, 1.85, 1.49, 1.58, 1.47, 0.96,
0.75, 0.22, -1.34, -1.72, -2.11, -1.56, -1.51, -1.29, -1.22,
-0.36, 1.78, 2.36, 1.78, 1.8, 1.76, 1.4, 1.02, 1.28, 1.21, 0.04,
-1.35, -2.1, -1.9, -1.49, -1.55, -1.35, -0.98, -0.34, 1.9, 1.99, 1.84)

gpsmoothing(y, tvec)
```

gpsmoothllik — *Marginal log-likelihood for Gaussian process smoothing*

Description

Marginal log-likelihood and gradient as a function of GP hyper-parameters phi and observation noise standard deviation sigma. For use in Gaussian process smoothing where values of phi and sigma may be optimized.

Usage

```
gpsmoothllik(phisig, yobs, rInput, kerneltype = "generalMatern")
```

Arguments

`phisig`	vector containing GP hyper-parameters phi and observation noise SD sigma. See `calCov` for the definitions of the hyper-parameters.
`yobs`	vector of observations
`rInput`	distance matrix between all time points of `yobs`
`kerneltype`	the covariance kernel, types `matern`, `rbf`, `compact1`, `periodicMatern`, `generalMatern` are supported. See `calCov` for their definitions.

Value

A list with elements `value` and `grad`, which are the log-likelihood value and gradient with respect to `phisig`, respectively.

Examples

```
# Suppose phi[1] = 0.5, phi[2] = 3, sigma = 0.1
gpsmoothllik(c(0.5,3,0.1), rnorm(10), abs(outer(0:9, t(0:9),'-')[,1,]))
```

MagiPosterior — *MAGI posterior density*

Description

Computes the MAGI log-posterior value and gradient for an ODE model with the given inputs: the observations Y, the latent system trajectories X, the parameters θ, the noise standard deviations σ, and covariance kernels.

Usage

```
MagiPosterior(
  y,
  xlatent,
  theta,
  sigma,
  covAllDimInput,
  odeModel,
  priorTemperatureInput = 1,
  useBand = FALSE
)
```

Arguments

`y`	data matrix of observations
`xlatent`	matrix of system trajectory values
`theta`	vector of parameter values θ
`sigma`	vector of observation noise for each system component
`covAllDimInput`	list of covariance kernel objects for each system component. Covariance calculations may be carried out with `calCov`.
`odeModel`	list of ODE functions and inputs. See details.
`priorTemperatureInput`	vector of tempering factors for the GP prior, derivatives, and observations, in that order. Controls the influence of the GP prior relative to the likelihood. Recommended values: the total number of observations divided by the total number of discretization points for the GP prior and derivatives, and 1 for the observations.
`useBand`	logical: should the band matrix approximation be used? If `TRUE`, `covAllDimInput` must include CinvBand, mphiBand, and KinvBand as computed by `calCov`.

Value

A list with elements `value` for the value of the log-posterior density and `grad` for its gradient.

Examples

```
# Trajectories from the Fitzhugh-Nagumo equations
tvec <- seq(0,20,2)
Vtrue <- c(-1, 1.91, 1.38, -1.32, -1.5, 1.73, 1.66, 0.89, -1.82, -0.93, 1.89)
Rtrue <- c(1, 0.33, -0.62, -0.82, 0.5, 0.94, -0.22, -0.9, -0.08, 0.95,  0.3)

# Noisy observations
Vobs <- Vtrue + rnorm(length(tvec), sd = 0.05)
Robs <- Rtrue + rnorm(length(tvec), sd = 0.1)

# Prepare distance matrix for covariance kernel calculation
foo <- outer(tvec, t(tvec),'-')[,1,]
r <- abs(foo)
```

```
r2 <- r^2
signr <- -sign(foo)

# Choose some hyperparameter values to illustrate
rphi=c(0.95, 3.27)
vphi=c(1.98, 1.12)
phiTest <- cbind(vphi, rphi)

# Covariance computations
curCovV <- calCov(phiTest[,1], r, signr, kerneltype = "generalMatern")
curCovR <- calCov(phiTest[,2], r, signr, kerneltype = "generalMatern")

# Y and X inputs to MagiPosterior
yInput <- data.matrix(cbind(Vobs, Robs))
xlatentTest <- data.matrix(cbind(Vtrue, Rtrue))

# ODE system for Fitzhugh-Nagumo equations
fnmodelODE <- function(theta,x,t) {
  V <- x[,1]
  R <- x[,2]

  result <- array(0, c(nrow(x),ncol(x)))
  result[,1] = theta[3] * (V - V^3 / 3.0 + R)
  result[,2] = -1.0/theta[3] * ( V - theta[1] + theta[2] * R)

  result
}

# Gradient with respect to system components
fnmodelDx <- function(theta,x,t) {
  resultDx <- array(0, c(nrow(x), ncol(x), ncol(x)))
  V = x[,1]

  resultDx[,1,1] = theta[3] * (1 - V^2)
  resultDx[,2,1] = theta[3]

  resultDx[,1,2] = (-1.0 / theta[3])
  resultDx[,2,2] = ( -1.0*theta[2]/theta[3] )

  resultDx
}

# Gradient with respect to parameters theta
fnmodelDtheta <- function(theta,x,t) {
  resultDtheta <- array(0, c(nrow(x), length(theta), ncol(x)))

  V = x[,1]
  R = x[,2]

  resultDtheta[,3,1] = V - V^3 / 3.0 + R

  resultDtheta[,1,2] =  1.0 / theta[3]
  resultDtheta[,2,2] = -R / theta[3]
  resultDtheta[,3,2] = 1.0/(theta[3]^2) * ( V - theta[1] + theta[2] * R)

  resultDtheta
}

# Create odeModel list
fnmodel <- list(
  fOde=fnmodelODE,
  fOdeDx=fnmodelDx,
  fOdeDtheta=fnmodelDtheta,
  thetaLowerBound=c(0,0,0),
  thetaUpperBound=c(Inf,Inf,Inf)
)

MagiPosterior(yInput, xlatentTest, theta = c(0.2, 0.2, 3), sigma = c(0.05, 0.1),
    list(curCovV, curCovR), fnmodel)
```

`MagiSolver` *MAnifold-constrained Gaussian process Inference (MAGI)*

Description

Core function of the MAGI method for inferring the parameters and trajectories of dynamic systems governed by ordinary differential equations. See vignette for detailed usage examples.

Usage

```
MagiSolver(y, odeModel, tvec, control = list())
```

Arguments

`y`	data matrix of observations
`odeModel`	list of ODE functions and inputs. See details.
`tvec`	vector of discretization time points corresponding to rows of `y`. If missing, `MagiSolver` will use the column named ‘time‘ in `y`.
`control`	list of control variables, which may include ‘sigma‘, ‘phi‘, ‘xInit‘, ‘thetaInit‘, ‘mu‘, ‘dotmu‘, ‘priorTemperature‘, ‘niterHmc‘ ‘burninRatio‘, ‘nstepsHmc‘, ‘stepSizeFactor‘, ‘bandSize‘, ‘useFixedSigma‘. See details.

Details

The data matrix `y` has a column for each system component, and optionally a column ‘time‘ with the discretization time points. If the column ‘time‘ is not provided in `y`, a vector of time points must be provided via the `tvec` argument. The rows of `y` correspond to the discretization set I at which the GP is constrained to the derivatives of the ODE system. To set the desired discretization level for inference, use `setDiscretization` to prepare the data matrix for input into `MagiSolver`. Missing observations are indicated with `NA` or `NaN`.

The list `odeModel` is used for specification of the ODE system and its parameters. It must include five elements:

`fOde` function that computes the ODEs, specified with the form `f(theta,x,t)`. See examples.

`fOdeDx` function that computes the gradients of the ODEs with respect to the system components. See examples.

`fOdeDtheta` function that computes the gradients of the ODEs with respect to the parameters θ. See examples.

`thetaLowerBound` a vector indicating the lower bounds of each parameter in θ.

`thetaUpperBound` a vector indicating the upper bounds of each parameter in θ.

Additional control variables can be supplied to `MagiSolver` via the optional list `control`, which may include the following:

`sigma` a vector of noise levels (observation noise standard deviations) σ for each component, at which to initialize MCMC sampling. By default, `MagiSolver` computes starting values for `sigma` via Gaussian process (GP) smoothing. If the noise levels are known, specify `sigma` together with `useFixedSigma = TRUE`.

`phi` a matrix of GP hyper-parameters for each component, with two rows for `phi[1]` and `phi[2]` and a column for each system component. By default, `MagiSolver` estimates `phi` via an optimization routine.

`theta` a vector of starting values for the parameters θ, at which to initialize MCMC sampling. By default, `MagiSolver` uses an optimization routine to obtain starting values.

`xInit` a matrix of values for the system trajectories of the same dimension as `y`, at which to initialize MCMC sampling. Default is linear interpolation between the observed (non-missing) values of `y` and an optimization routine for entirely unobserved components of `y`.

`mu` a matrix of values for the mean function of the GP prior, of the same dimension as `y`. Default is a zero mean function.

`dotmu` a matrix of values for the derivatives of the GP prior mean function, of the same dimension as `y`. Default is zero.

`priorTemperature` the tempering factor by which to divide the contribution of the GP prior, to control the influence of the GP prior relative to the likelihood. Default is the total number of observations divided by the total number of discretization points.

`niterHmc` MCMC sampling from the posterior is carried out via Hamiltonian Monte Carlo (HMC). `niterHmc` specifies the number of HMC iterations to run. Default is 20000 HMC iterations.

`nstepsHmc` the number of leapfrog steps per HMC iteration. Default is 200.

`burninRatio` the proportion of HMC iterations to be discarded as burn-in. Default is 0.5, which discards the first half of the MCMC samples.

`stepSizeFactor` initial leapfrog step size factor for HMC. Default is 0.01, and the leapfrog step size is automatically tuned during burn-in to achieve an acceptance rate between 60-90%.

`bandSize` a band matrix approximation is used to speed up matrix operations, with default band size 20. Can be increased if `MagiSolver` returns an error indicating numerical instability.

`useFixedSigma` logical, set to `TRUE` if `sigma` is known. If `useFixedSigma=TRUE`, the known values of σ must be supplied via the `sigma` control variable.

Value

`MagiSolver` returns a list with the following elements:

`theta` matrix of MCMC samples for the system parameters θ, after burn-in.

`xsampled` array of MCMC samples for the system trajectories at each discretization time point, after burn-in.

`sigma` matrix of MCMC samples for the observation noise SDs σ, after burn-in.

`phi` matrix of estimated GP hyper-parameters, one column for each system component.

`lp` vector of log-posterior values at each MCMC iteration, after burn-in.

References

Shihao Yang, Samuel WK Wong, SC Kou (2021). Inference of dynamic systems from noisy and sparse data via manifold-constrained Gaussian processes. *Proceedings of the National Academy of Sciences*, 118 (15), e2020397118.

Examples

```
# Setting up the Fitzhugh-Nagumo system equations, see vignette for details
# ODE system
fnmodelODE <- function(theta,x,t) {
  V <- x[,1]
  R <- x[,2]

  result <- array(0, c(nrow(x),ncol(x)))
  result[,1] = theta[3] * (V - V^3 / 3.0 + R)
  result[,2] = -1.0/theta[3] * ( V - theta[1] + theta[2] * R)

  result
}

# Gradient with respect to system components
fnmodelDx <- function(theta,x,t) {
  resultDx <- array(0, c(nrow(x), ncol(x), ncol(x)))
  V = x[,1]

  resultDx[,1,1] = theta[3] * (1 - V^2)
  resultDx[,2,1] = theta[3]

  resultDx[,1,2] = (-1.0 / theta[3])
  resultDx[,2,2] = ( -1.0*theta[2]/theta[3] )

  resultDx
}

# Gradient with respect to parameters theta
fnmodelDtheta <- function(theta,x,t) {
  resultDtheta <- array(0, c(nrow(x), length(theta), ncol(x)))

  V = x[,1]
```

```
    R = x[,2]

    resultDtheta[,3,1] = V - V^3 / 3.0 + R

    resultDtheta[,1,2] =  1.0 / theta[3]
    resultDtheta[,2,2] = -R / theta[3]
    resultDtheta[,3,2] = 1.0/(theta[3]^2) * ( V - theta[1] + theta[2] * R)

    resultDtheta
  }

  # Create odeModel list
  fnmodel <- list(
    fOde=fnmodelODE,
    fOdeDx=fnmodelDx,
    fOdeDtheta=fnmodelDtheta,
    thetaLowerBound=c(0,0,0),
    thetaUpperBound=c(Inf,Inf,Inf)
  )

  # Example noisy data observed from the FN system
  tvec <- seq(0, 20, by = 0.5)
  V <- c(-1.16, -0.18, 1.57, 1.99, 1.95, 1.85, 1.49, 1.58, 1.47, 0.96,
  0.75, 0.22, -1.34, -1.72, -2.11, -1.56, -1.51, -1.29, -1.22,
  -0.36, 1.78, 2.36, 1.78, 1.8, 1.76, 1.4, 1.02, 1.28, 1.21, 0.04,
  -1.35, -2.1, -1.9, -1.49, -1.55, -1.35, -0.98, -0.34, 1.9, 1.99, 1.84)
  R <- c(0.94, 1.22, 0.89, 0.13, 0.4, 0.04, -0.21, -0.65, -0.31, -0.65,
   -0.72, -1.26, -0.56, -0.44, -0.63, 0.21, 1.07, 0.57, 0.85, 1.04,
   0.92, 0.47, 0.27, 0.16, -0.41, -0.6, -0.58, -0.54, -0.59, -1.15,
   -1.23, -0.37, -0.06, 0.16, 0.43, 0.73, 0.7, 1.37, 1.1, 0.85, 0.23)

  # Set discretization for a total of 161 time points
  yobs <- data.frame(time=tvec, V=V, R=R)
  yinput <- setDiscretization(yobs, level=2)

  # Call MagiSolver
  # short sampler run for demo only, more iterations needed for convergence
  MagiSolver(yinput, fnmodel, control = list(nstepsHmc=5, niterHmc = 402))

  # full run with 20000 HMC iterations
  result <- MagiSolver(yinput, fnmodel, control = list(nstepsHmc=100))
```

`setDiscretization` *Set discretization level*

Description

Set the discretization level of a data matrix for input to `MagiSolver`, by inserting time points where the GP is constrained to the derivatives of the ODE system.

Usage

```
setDiscretization(dat, level, by)
```

Arguments

`dat`	data matrix. Must include a column with name ‘time‘.
`level`	discretization level (a positive integer). `2^level -1` equally-spaced points will be inserted between existing data points in `dat`.
`by`	discretization interval. As an alternative to `level`, equally-spaced spaced time points will be inserted with interval `by` between successive points.

Details

Specify the desired discretization using `level` or `by`.

Value

Returns a data matrix with the same columns as `dat`, with rows added for the inserted discretization time points.

Examples

```
dat <- data.frame(time = 0:10, x = rnorm(11))
setDiscretization(dat, level = 2)
setDiscretization(dat, by = 0.2)
```

`testDynamicalModel` *Test dynamic system model specification*

Description

Given functions for the ODE and its gradients (with respect to the system components and parameters), verify the correctness of the gradients using numerical differentiation.

Usage

```
testDynamicalModel(modelODE, modelDx, modelDtheta, modelName, x, theta, tvec)
```

Arguments

`modelODE`	function that computes the ODEs, specified with the form $f(theta, x, t)$. See examples.
`modelDx`	function that computes the gradients of the ODEs with respect to the system components. See examples.
`modelDtheta`	function that computes the gradients of the ODEs with respect to the parameters θ. See examples.

`modelName`	string giving a name for the model
`x`	data matrix of system values, one column for each component, at which to test the gradients
`theta`	vector of parameter values for θ, at which to test the gradients
`tvec`	vector of time points corresponding to the rows of `x`

Details

Calls `test_that` to test equality of the analytic and numeric gradients.

Value

A list with elements `testDx` and `testDtheta`, each with value `TRUE` if the corresponding gradient check passed and `FALSE` if not.

Examples

```
# ODE system for Fitzhugh-Nagumo equations
fnmodelODE <- function(theta,x,t) {
  V <- x[,1]
  R <- x[,2]

  result <- array(0, c(nrow(x),ncol(x)))
  result[,1] = theta[3] * (V - V^3 / 3.0 + R)
  result[,2] = -1.0/theta[3] * ( V - theta[1] + theta[2] * R)

  result
}

# Gradient with respect to system components
fnmodelDx <- function(theta,x,t) {
  resultDx <- array(0, c(nrow(x), ncol(x), ncol(x)))
  V = x[,1]

  resultDx[,1,1] = theta[3] * (1 - V^2)
  resultDx[,2,1] = theta[3]

  resultDx[,1,2] = (-1.0 / theta[3])
  resultDx[,2,2] = ( -1.0*theta[2]/theta[3] )

  resultDx
}

# Gradient with respect to parameters theta
fnmodelDtheta <- function(theta,x,t) {
  resultDtheta <- array(0, c(nrow(x), length(theta), ncol(x)))

  V = x[,1]
  R = x[,2]

  resultDtheta[,3,1] = V - V^3 / 3.0 + R
  resultDtheta[,1,2] =  1.0 / theta[3]
  resultDtheta[,2,2] = -R / theta[3]
  resultDtheta[,3,2] = 1.0/(theta[3]^2) * ( V - theta[1] + theta[2] * R)

  resultDtheta
}

# Example incorrect gradient with respect to parameters theta
fnmodelDthetaWrong <- function(theta,x,t) {
  resultDtheta <- array(0, c(nrow(x), length(theta), ncol(x)))

  V = x[,1]
  R = x[,2]

  resultDtheta[,3,1] = V - V^3 / 3.0 - R

  resultDtheta[,1,2] =  1.0 / theta[3]
  resultDtheta[,2,2] = -R / theta[3]
  resultDtheta[,3,2] = 1.0/(theta[3]^2) * ( V - theta[1] + theta[2] * R)

  resultDtheta
}

# Sample data for testing gradient correctness
tvec <- seq(0, 20, by = 0.5)
V <- c(-1.16, -0.18, 1.57, 1.99, 1.95, 1.85, 1.49, 1.58, 1.47, 0.96,
0.75, 0.22, -1.34, -1.72, -2.11, -1.56, -1.51, -1.29, -1.22,
-0.36, 1.78, 2.36, 1.78, 1.8, 1.76, 1.4, 1.02, 1.28, 1.21, 0.04,
-1.35, -2.1, -1.9, -1.49, -1.55, -1.35, -0.98, -0.34, 1.9, 1.99, 1.84)
R <- c(0.94, 1.22, 0.89, 0.13, 0.4, 0.04, -0.21, -0.65, -0.31, -0.65,
 -0.72, -1.26, -0.56, -0.44, -0.63, 0.21, 1.07, 0.57, 0.85, 1.04,
 0.92, 0.47, 0.27, 0.16, -0.41, -0.6, -0.58, -0.54, -0.59, -1.15,
 -1.23, -0.37, -0.06, 0.16, 0.43, 0.73, 0.7, 1.37, 1.1, 0.85, 0.23)

# Correct gradients
testDynamicalModel(fnmodelODE, fnmodelDx, fnmodelDtheta,
    "FN equations", cbind(V,R), c(.5, .6, 2), tvec)

# Incorrect theta gradient (test fails)
testDynamicalModel(fnmodelODE, fnmodelDx, fnmodelDthetaWrong,
    "FN equations", cbind(V,R), c(.5, .6, 2), tvec)
```

Index