Package ‘GPoM’

July 26, 2018

Type Package

Title Generalized Polynomial Modelling

Version 1.2

Date 2018-07-26

Author Sylvain Mangiarotti [aut],
Flavie Le Jean [ctb],
Malika Chassan [ctb],
Laurent Drapeau [ctb],
Mireille Huc [cre]

Maintainer Mireille Huc <mireille.huc@cesbio.cnes.fr>

Description Platform dedicated to the Global Modelling technique. Its aim is to obtain ordinary differential equations of polynomial form directly from time series. It can be applied to single or multiple time series under various conditions of noise, time series lengths, sampling, etc. This platform is developped at the Centre d'Etudes Spatiales de la Biosphere (CESBIO), UMR 5126 UPS/CNRS/CNES/IRD, 18 av. Edouard Belin, 31401 TOULOUSE, FRANCE. The developments were funded by the French program Les Enveloppes Fluides et l'Environnement (LEFE, MANU, projets GloMo, SpatioGloMo and MoMu). The French program Defi InFiNiTi (CNRS) and PNTS are also acknowledged (projects Crops'IChaos and Musc & SlowFast).

License CeCILL-2

LazyData TRUE

RoxygenNote 6.0.1

Depends R (>= 2.10), deSolve, rgl

Suggests testthat, knitr, rmarkdown, signal, float

VignetteBuilder knitr

Encoding UTF-8

NeedsCompilation no

Repository CRAN

Date/Publication 2018-07-26 16:20:23 UTC

R topics documented:

GPoM-package *GPoM package: Generalized Polynomial Modelling*

Description

GPoM is a platform dedicated to the Global Modelling technique. Its aim is to obtain deterministic models of Ordinary Differential Equations from observational time series. It applies to single and to multiple time series. With single time series, it can be used: to detect low-dimnesional determinism and low-dimensional (deterministic) chaos. It can also be used to characterize the observed behavior, using the obtained models as a proxy of the original dynamics, as far as the model validation could be checked. With multiple time series, it can be used: to detect couplings between observed variables, to infer causal networks, and to reformulate the original equations of the observed system (retro-modelling). The present package focuses on models in Ordinary Differential Equations of polynomial form. The package was designed to model weakly predictable dynamical behaviors (such as chaotic behaviors). Of course, it can also apply to more of fully predictable behavior, either linear or nonlinear. Several vignettes are associated to the package which can be used as a tutorial, and it also provides an overlook of the diversity of applications and at the performances of the tools. Users are kindly asked to quote the corresponding references when using the package (see hereafter).

Note

FOR USERS
This package was developped at Centre d'Etudes Spatiales de la Biosphere (Cesbio, UMR 5126, UPS-CNRS-CNES-IRD, http://www.cesbio.ups-tlse.fr). An important part of the developments were funded by the French program Les Enveloppes Fluides et l'Environnement (LEFE, MANU, projets GloMo, SpatioGloMo and MoMu). The French program Défi InFiNiTi (CNRS) and PNTS are also acknowledged (projects Crops'IChaos and Musc & SlowFast).

If you apply this package to single time series, please quote [6]. If you apply it to multivariate time series, please quote [10]. If you apply it to infer couplings among time series, please quote [8]. If you apply it to classification, please quote [11].

HISTORICAL BACKGROUND
The global modelling technique was initiated during the early 1990s [1-3]. It takes its background from the Theory of Nonlinear Dynamical Systems. Earlier investigations can also be found in the fields of Electrical Engineering and Statistics but these mainly focused on linear problems [4]. The approach became applicable to the analysis of real world environmental behaviours by the end of the 2000s [5-7]. Recent works have shown that the approach could be applied to numerous other dynamical behaviors [8-10]. Global modelling aims to obtain deterministic models directly from observed time series.

Author(s)

Sylvain Mangiarotti, Flavie Le Jean, Malika Chassan, Laurent Drapeau, Mireille Huc.

Maintainer: M. Huc <mireille.huc@cesbio.cnes.fr>

References

[1] J. P. Crutchfield and B. S. McNamara, 1987. Equations of motion from a data series, Complex Systems. 1, 417-452.
[2] Gouesbet G., Letellier C., 1994. Global vector-field reconstruction by using a multivariate polynomial L2 approximation on nets, Physical Review E, 49 (6), 4955-4972.
[3] C. Letellier, L. Le Sceller, E. Marechal, P. Dutertre, B. Maheu, G. Gouesbet, Z. Fei, and J. L. Hudson, 1995. Global vector field reconstruction from a chaotic experimental signal in copper electrodissolution, Physical Review E, 51, 4262-4266.
[4] L. A. Aguirre & C. Letellier, Modeling nonlinear dynamics and chaos: A review, Mathematical Problems in Engineering, 2009, 238960.
C. Letellier, L. Le Sceller, E. Marechal, P. Dutertre, B. Maheu, G. Gouesbet, Z. Fei, and J. L. Hudson, 1995. Global vector field reconstruction from a chaotic experimental signal in copper electrodissolution, Physical Review E 51, 4262-4266.
[5] J. Maquet, C. Letellier, and L. A. Aguirre, 2007. Global models from the Canadian Lynx cycles as a first evidence for chaos in real ecosystems, Juornal of Mathematical Biology. 55(1), 21-39.
[6] Mangiarotti S., Coudret R., Drapeau L., & Jarlan L., 2012. Polynomial search and global modeling : Two algorithms for modeling chaos, Physical Review E, 86, 046205.
[7] Mangiarotti S., Drapeau L. & Letellier C., 2014. Two chaotic models for cereal crops observed from satellite in northern Morocco. Chaos, 24(2), 023130.
[8] Mangiarotti S., 2015. Low dimensional chaotic models for the plague epidemic in Bombay (1896-1911). Chaos, Solitons and Fractals, 81A, 184-186.
[9] Mangiarotti S., Peyre M. & Huc M., A chaotic model for the epidemic of Ebola Virus Disease in West Africa (2013-2016). Chaos, 26, 113112, 2016.
[10] Mangiarotti S., 2014. Modelisation globale et Caracterisation Topologique de dynamiques environnementales - de l'analyse des enveloppes fluides et du couvert de surface de la Terre a la caracterisation topolodynamique du chaos. Habilitation to Direct Research, University of Toulouse 3, France.
[11] Mangiarotti S., Sharma A.K., Corgne S., Hubert-Moy L., Ruiz L., Sekhar M., Kerr Y., Can the global modelling technique be used for crop classification? Chaos, Solitons & Fractals, in press.

`allMod_nVar3_dMax2 data set`

Numerical description of a list of eighteen three-dimensional chaotic sytems (see vignette `7_Retro-Modelling`*)*

Description

A list named `allMod_nVar3_dMax2` of matrix providing the numerical description of eighteen three-dimensional chaotic systems:
Lorenz-1963 (`$L63`), Rössler-1976 (`$R76`), Burke & shaw 1981 (`$BS81`), Lorenz-1984 (`$L84`), Nosé & Hooer 1986 (`$NH86`), Genesio & Tosi 1992 (`$GT92`), Spott systems 1994 (`$SprF`, `$SprH`, `$SprK`, `$SprO`, `$SprP`, `$SprG`, `$SprM`, `$SprQ`, `$SprS`), Chlouverakis & Sprott 2004 (`$CS2004`), Li 2007 (`$Li2007`) and the Cord system by Aguirre & Letellier 2012 (`$Cord2012`). Each dynamical system is provided as a matrix: each column corresponds to one equation, each lines to the polynomial coefficients which order is following the convetion defined by function `poLabs(nVar = 3, dMax = 2)`.

Usage

```
allMod_nVar3_dMax2
```

Format

An object of class `list` of length 18.

Author(s)

Sylvain Mangiarotti, Mireille Huc.

References

All the references are provided in vignette `7_retro-modelling`.

`allToTest` *A list providing the description of six models tested by the function* `autoGPoMoTest`.

Description

List of 6 models available for tests (by `autoGPoMoTest`). Each model (`$mToTest1`, `$mToTest2`, etc.) is provided as a matrix of dimension 10 * 3. Each column corresponds to one equation. The order of the coefficients follows the conventions defined by `poLabs(nVar = 3, dMax = 2)`.

Usage

```
allToTest
```

Format

An object of class `list` of length 6.

Author(s)

Sylvain Mangiarotti, Mireille Huc

Examples

```
###########
# example #
###########
data("allToTest")
# 6 models are available in this list:
names(allToTest)
# The parameter of their formulation (nVar and dMax)
# can be retrieved:
nVar <- dim(allToTest$mToTest6)[2]
dMax <- p2dMax(nVar = 3, pMaxKnown = dim(allToTest$mToTest6)[1])
```

```
# Their equation can be edited as follows:
visuEq(nVar, dMax, allToTest$mToTest6, approx = 2)
```

---

autoGPoMoSearch *Automatic search of polynomial Equations*

Description

This algorithm aims to get an ensemble of possible models which integrability will be tested later with function `autoGPoMoTest`. By default, all the terms are considered available (Some of the terms can be excluded intentionally using the option `filterReg`). The maximum size of the equation depends on the model dimension `nVar`, and on the maximum polynomial degree `dMax`. The algorithm removes polynomial terms one by one using a leave-one-out method.

Usage

```
autoGPoMoSearch(data, dt, nVar = nVar, dMax = dMax, weight = NULL,
  show = 0, underSamp = NULL, filterReg = NULL)
```

Arguments

`data`	Input Time series: Each column is one time series that corresponds to one variable.
`dt`	Time sampling of the input series.
`nVar`	Number of variables considered in the polynomial formulation.
`dMax`	Maximum degree of the polynomial formulation.
`weight`	A vector providing the binary weighting function of the input data series (0 or 1). By default, all the values are set to 1.
`show`	Provide (2) or not (0-1) visual output during the running process.
`underSamp`	Number of points used for undersampling the data. For `undersamp = 1` the complete time series is used. For `undersamp = 2`, only one data out of two is kept, etc.
`filterReg`	A vector that specifies the template for the equation structure (for one single equation). The convention defined by `poLabs` is used. Value is 1 if the regressor is available, 0 if it is not.

Value

A list of two matrices:

`$filtMemo` describes the selected terms (1 if the term is used, 0 if not)

`$KMemo` provides the corresponding coefficients

Author(s)

Sylvain Mangiarotti, Flavie Le Jean

See Also

autoGPoMoTest, gPoMo, findAllSets, poLabs

Examples

```
# Load data
data('RosYco')
# Search for potential models
filt = autoGPoMoSearch(RosYco[,2], nVar = 3, dMax = 2,
                       dt = 1/125, show = 1)
# As an example, the equations of the fourth line has the following terms:
poLabs(nVar = 3, dMax = 2)[filt$filtMemo[5,] != 0]
# which coefficients correspond to
cbind(filt$KMemo[5,], poLabs(nVar = 3, dMax = 2))[filt$filtMemo[5,] != 0,]
```

---

autoGPoMoTest   *Tests the numerical integrability of models and classify their dynamical regime*

Description

Tests the numerical integrability of provided models (these may have been obtained with function autoGPoMoSearch), and classify these models as Divergent, Fixed Points, Periodic or not Unclassified (potentially chaotic).

Usage

```
autoGPoMoTest(data, tin = NULL, dt = NULL, nVar = nVar, dMax = dMax,
  show = 1, verbose = 1, allKL = allKL, numValidIC = 1, weight = NULL,
  IstepMin = 10, IstepMax = 10000, tooFarThreshold = 4,
  LimCyclThreshold = 0, fixedPtThreshold = 1e-08, method = "rk4")
```

Arguments

data	Input Time series: Each column is one time series that corresponds to one variable.
tin	Input date vector which length should correspond to the input time series.
dt	Sampling time of the input time series.
nVar	Number of variables considered in the polynomial formulation.
dMax	Maximum degree of the polynomial formulation.
show	Provide (2) or not (0-1) visual output during the running process.
verbose	Gives information (if set to 1) about the algorithm progress and keeps silent if set to 0.
allKL	A list of all the models $mToTest1, $mToTest2, etc. to be tested. Each model is provided as a matrix.

numValidIC
: Line number of the first valid initial conditions, that is, such as weight is not equal to zero.

weight
: A vector providing the binary weighting function of the input data series (0 or 1). By default, all the values are set to 1.

IstepMin
: The minimum number of integration step to start of the analysis (by default `IstepMin = 10`).

IstepMax
: The maximum number of integration steps for stopping the analysis (by default `IstepMax = 10000`).

tooFarThreshold
: Divergence threshold, maximum value of the model trajectory compared to the data standard deviation. By default a trjactory is too far if the distance to the center is larger than four times the variance of the input data.

LimCyclThreshold
: Threshold used to detect the limit cycle.

fixedPtThreshold
: Threshold used to detect fixed points.

method
: The integration technique used for the numerical integration. By default, the fourth-order Runge-Kutta method (`method = 'rk4'`) is used. Other methods such as 'ode45' or 'lsoda' may also be chosen. See package `deSolve` for details.

Value

A list containing:

`$okMod` A vector classifying the models: diverging models (0), periodic models of period-1 (-1), unclassified models (1).

`$coeff` A matrix with the coefficients of one selected model

`$models` A list of all the models to be tested `$mToTest1`, `$mToTest2`, etc. and of all selected models `$model1`, `$model2`, etc.

`$tout` The time vector of the output time series (vector length corresponding to the longest numerical integration duration)

`$stockoutreg` A list of matrices with the integrated trajectories (variable `X1` in column 1, `X2` in 2, etc.) for all the models `$model1`, `$model2`, etc.

Author(s)

Sylvain Mangiarotti, Flavie Le Jean

See Also

autoGPoMoSearch, gPoMo, poLabs

Examples

```
#Example
# Load data:
data('RosYco')
```

```
# Structure choice
data('allToTest')
# Test the models
outGPT <- autoGPoMoTest(RosYco, nVar= 3, dMax = 2, dt = 1/125, show=1,
                        allKL = allToTest, IstepMax = 60)
```

---

bDrvFilt *Builds the derivative filter*

Description

Build the Savitzky-Golay derivative filter (Savitzky-Golay, 1964).

Usage

```
bDrvFilt(nDrv, tstep, winL = 9)
```

Arguments

nDrv	The number of derivatives to be computed.
tstep	Sampling time.
winL	The local window length to be used for computing the derivatives [1].

Author(s)

Sylvain Mangiarotti

References

[1] Savitzky, A.; Golay, M.J.E., Smoothing and Differentiation of Data by Simplified Least Squares Procedures. Analytical Chemistry 36 (8), 1627-1639, 1964.

---

cano2M *cano2M : Converts a model in canonical form into a matrix form*

Description

Converts the vectorial formulation of canonical models into a matrix formulation (that is, including explicitely all the equations). For both input, the list of terms follows the convention defined by poLabs.

Usage

```
cano2M(nVar, dMax, poly)
```

Arguments

nVar	The number of variables
dMax	The maximum degree allowed in the formulation
poly	A vector of coefficients corresponding to the regressor of the canonical function

Author(s)

Sylvain Mangiarotti, Mireille Huc

See Also

drvSucc, gPoMo, poLabs

Examples

```
# A vector of polynomial terms corresponding to a canonical form:
polyTerms <- c(0.2,0,-1,0.5,0,0,0,0,0,0)
# Convert this vector into a matrix formulation with all the equations:
K <- cano2M(3,2,polyTerms)
# Visualize the equations:
visuEq(3,2,K)
```

---

compDeriv *Computes the successive derivatives of a time series*

Description

Computes the successive derivatives from one single time series, with the Savitzky-Golay approach (1964).

Usage

```
compDeriv(TS, nDrv, tstep, winL = 9)
```

Arguments

TS	A single time series provided as a single vector.
nDrv	The number of derivatives to be computed from the input `series`. The resulting number of outpout time series will thus be `nVar = nDrv + 1`.
tstep	Sampling Time of the input time series `TS`.
winL	The local window length used for computing the derivatives [1-2].

Value

A matrix containing the original variable (smoothed by the filtering process) and its `nDrv` first derivatives (note that `winL` values of the original time series will be lost both at the begining and the end of the time series due to boundary effect).

Author(s)

Sylvain Mangiarotti

References

[1] Savitzky, A.; Golay, M.J.E., Smoothing and Differentiation of Data by Simplified Least Squares Procedures. Analytical Chemistry 36 (8), 1627-1639, 1964.
[2] Steinier J., Termonia Y., Deltour, J. Comments on smoothing and differentiation of data by simplified least square procedure. Analytical Chemistry 44 (11): 1906-1909, 1972.

See Also

gloMoId, gPoMo, poLabs

`concat` *Concat Concatenates separated time series*

Description

The aim of this code is to provide, from a set of multiple time series, a single concatenated time series for applying the global modeling technique to all the time time series in association.

Usage

```
concat(svrlTS, winL = 9)
```

Arguments

`svrlTS`	All separated time series.
`winL`	Total number of points used for computing the derivatives of the input time series. This parameter will be used as an input in function `drvSucc` to compute the derivatives.

Value

`concaTS` The concatenated time series.

Author(s)

Sylvain Mangiarotti, Mireille Huc

References

S. Mangiarotti, F. Le Jean, M. Huc & C. Letellier, 2016. Global modeling of aggregated and associated chaotic dynamics, Chaos, Solitons & Fractals, 83, 82-96.

Examples

```
# load data
data("svrlTS")
# Concatenate the data set into a single time series
winL = 55
concaTS <- concat(svrlTS, winL = winL)
# Plot the concatenated time series
plot(concaTS$sglTS$TS[,1], concaTS$sglTS$TS[,2],
     main = 'Concatenated time series',
     xlab = 'Time (concatenated)', ylab = 'y(t)',
     type = 'l', col = 'gray')
lines(concaTS$sglTS$TS[concaTS$sglTS$W == 1,1],
      concaTS$sglTS$TS[concaTS$sglTS$W == 1,2], type = 'p', col = 'green', cex = 0.5)
lines(concaTS$sglTS$TS[concaTS$sglTS$W == 0,1],
      concaTS$sglTS$TS[concaTS$sglTS$W == 0,2], type = 'p', col = 'red', cex = 0.5)
lines(concaTS$sglTS$TS[,1], concaTS$sglTS$W, type = 'l')
## Not run:
# The concatenated data set can be used for global modelling:
GPout1 <- gPoMo(data = concaTS$sglTS$TS[,2], tin = concaTS$sglTS$TS[,1],
                dMax = 2, nS = 3, winL = winL, weight = concaTS$sglTS$W, show = 1,
                IstepMin = 10, IstepMax = 6000, nPmin = 11, nPmax = 11, method = 'rk4')

## End(Not run)
```

---

d2pMax   *Provides the number of polynomial terms* pMax *given* dMax *and* nVar

---

Description

Computes the number of polynomial terms pMax used to formulate an equation given the maximal polynomial degree dMax and the number of variables nVar following the conventions as defined by fuction poLabs.

Usage

```
d2pMax(nVar, dMaxKnown)
```

Arguments

nVar	Number of variables considered in the polynomial formulation.
dMaxKnown	The maximum polynomial degree dMax

Value

The number pMax of polynomial terms used to code a polynomial equation

Author(s)

Sylvain Mangiarotti

See Also

gloMoId, gPoMo, poLabs

Examples

```
#############
# Example 1 #
#############
# Maximum polynomial degree ?
# number of variables:
nVar <- 3
# polynomial degree:
dMax <- 3
# The maximal polynomial degree used for coding the polynomial is:
d2pMax(nVar,dMax)
```

---

data_vignetteIII data set

Output of the vignette `III_Modelling`

Description

To reduce the computation time, the outputs of the simulations presented in vignette VI have been run beforehand and saved in this file.

Usage

```
data_vignetteIII
```

Format

An object of class `list` of length 12.

Author(s)

Sylvain Mangiarotti, Mireille Huc.

`data_vignetteVI data set` *Output of the vignette* `VI_Sensitivity`

Description

To reduce the computation time, the outputs of the simulations presented in vignette VI have been run beforehand and saved in this file.

Usage

```
data_vignetteVI
```

Format

An object of class `list` of length 6.

Author(s)

Sylvain Mangiarotti, Mireille Huc.

`data_vignetteVII data set` *Output of the vignette* `VII_Retro-Modelling`

Description

To reduce the computation time, the outputs of the simulations presented in vignette VII have been run beforehand and saved in this file.

Usage

```
data_vignetteVII
```

Format

An object of class `list` of length 29.

Author(s)

Sylvain Mangiarotti, Mireille Huc.

derivODE2 — *A subfonction for the numerical integration of polynomial equations provided in a generic form following the convetion defined by function* `poLabs`.

Description

This function provides the one step integration of polynomial Ordinary Differential Equations (ODE). This function requires the function `ode` (`deSolve` package).

Usage

```
derivODE2(t, x, K, regS = NULL)
```

Arguments

`t`	All the dates for which the result of the numerical integration of the model must be provided
`x`	Current state vector (input from which the next state will be estimated)
`K`	A matrix providing the model description: each column corresponds to one equation which polynomial organisation is following the convention defined by function `poLabs`.
`regS`	Current states of each polynomial terms used in `poLabs`. These states can be deduced from the current state vector `x` (using the function `regSeries`). When available, it can be provided as an input to avoid unecessary computation.

Author(s)

Sylvain Mangiarotti

See Also

`numicano`, `numinoisy`

detectP1limCycl — *Detection of limit cycles of period-1*

Description

This algorithm aim to detect period-1 limit cycles from trajectories in the phase sapce considered in a bidimensional projection.

Usage

```
detectP1limCycl(data, LimCyclThreshold = 0.01, show = 2)
```

Arguments

`data`	A matrix of the trajectory in a 2D space (if more than two columns are provided, only the two first columns are considered)
`LimCyclThreshold`	The detection threshold
`show`	Indicates the deepness of the feedback (from 0 to 2)

Value

Indicates if a limit cycle is detected (1) or not (0)

Author(s)

Sylvain Mangiarotti

See Also

`autoGPoMoTest`

`drvSucc` *drvSucc : Computes the successive derivatives of a time series*

Description

Computes the successive derivatives from one single time series, using the Savitzky-Golay algorithm (1964).

Usage

```
drvSucc(tin = NULL, serie, nDeriv, weight = NULL, tstep = NULL,
  winL = 9)
```

Arguments

`tin`	Input date vector which length should correspond to the input time series.
`serie`	A single time series provided as a single vector.
`nDeriv`	The number of derivatives to be computed from the input time series. The resulting number of time series obtained in output will be `nDeriv + 1`.
`weight`	A vector providing the binary weighting function of the input data series (0 or 1). By default, all the values are set to 1.
`tstep`	Sampling time of the input time series. Used only if time vector `tin` is not provided.
`winL`	Number (exclusively odd number) of points of the local window used for computing the derivatives along the input time series. The Savitzky-Golay filter is used for this purpose [1,2].

Value

A list containing:

$serie The original time serie

$tin The time vector containing the dates corresponding to the original time series

$tstep The time step (assumed to be regular)

$tout The time vector of the output series

seriesDeriv A matrix containing the original time series (smoothed by the filtering process) in the first column and its `nDeriv + 1` successive derivatives in the next ones. Note that `winL` values of the original time series will be lost, that is `(winL - 1)/2` at the begining and `(winL - 1)/2` at the end of the time series due to a computation boundary effect).

Author(s)

Sylvain Mangiarotti, Mireille Huc

References

[1] Savitzky, A.; Golay, M.J.E., Smoothing and Differentiation of Data by Simplified Least Squares Procedures. Analytical Chemistry 36 (8), 1627-1639, 1964.
[2] Steinier J., Termonia Y., Deltour, J. Comments on smoothing and differentiation of data by simplified least square procedure. Analytical Chemistry 44 (11): 1906-1909, 1972.

See Also

gloMoId, gPoMo, poLabs, compDeriv

Examples

```
#############
# Example 1 #
#############
# Generate a time series:
tin <- seq(0, 5, by = 0.01)
data <- 2 * sin(5*tin)
dev.new()
par(mfrow = c(3, 1))
# Compute its derivatives:
drv <- drvSucc(tin = tin, nDeriv = 2, serie = data, winL = 5)
#
# plot original and filtered series
plot(tin, data, type='l', col = 'black', xlab = 't', ylab = 'x(t)')
lines(drv$tout, drv$seriesDeriv[,1], lty = 3, lwd = 3, col = 'green')
#
# analytic 1st derivative
firstD <- 10 * cos(5 * tin)
# plot both
plot(tin, firstD, type = 'l', col = 'black', xlab = 't', ylab = 'dx/dt')
lines(drv$tout, drv$seriesDeriv[,2], lty = 3, lwd = 3, col = 'green')
```

```
#
# analytic 2nd derivative
scdD <- -50 * sin(5 * tin)
# plot both
plot(tin, scdD, type = 'l', col = 'black', xlab = 't', ylab = 'd2x/dt2')
lines(drv$tout, drv$seriesDeriv[,3], lty=3, lwd = 3, col = 'green')

#############
# Example 2 #
#############
# load data:
data("Ross76")
tin <- Ross76[,1]
data <- Ross76[,2]

# Compute the derivatives
drvOut <- drvSucc(tin, data, nDeriv=4)
dev.new()
par(mfrow = c(3, 1))
# original and smoothed variable:
plot(drvOut$tin, drvOut$serie,
     type='p', cex = 1, xlab = 'time', ylab = 'x(t)')
lines(drvOut$tout, drvOut$seriesDeriv[,1], type='p', col='red')
lines(drvOut$tout, drvOut$seriesDeriv[,1], type='l', col='red')
# 1st derivative:
plot(drvOut$tout, drvOut$seriesDeriv[,2],
     type='p', col='red', xlab = 'time', ylab = 'dx(t)/dt')
lines(drvOut$tout, drvOut$seriesDeriv[,2], type='l', col='red')
# 2nd derivative:
plot(drvOut$tout, drvOut$seriesDeriv[,3],
     type='p', col='red', xlab = 'time', ylab = 'd2x(t)/dt2')
lines(drvOut$tout, drvOut$seriesDeriv[,3], type='l', col='red')
```

`findAllSets` *Find all possible sets of equation combinations considering an ensemble of possible equation.*

Description

For each equation to be retrieved, an ensemble of potential formulation is given. For instance, if three possible formulations are provided for equation (1), one for equation (2) and two for equation (3). In this case, six (i.e. 3*1*2) possible sets of equations can be obtained from these potential formulations. The aim of this program is to formulate all the potential systems from the individual formulations provided of the individual equations.

Usage

```
findAllSets(allFilt, nS = c(3), nPmin = 1, nPmax = 14)
```

Arguments

- `allFilt` — A list with: (1) A matrix `allFilt$Xi` of possible formulations for each equation (corresponding to variable `Xi`); And (2) a vector `allFilt$Npi` providing the number of polynomial terms contained in each formulation.
- `nS` — A vector providing the number of dimensions used for each input variables (see Examples 1 and 2). The dimension of the resulting model will be `nVar = sum(nS)`.
- `nPmin` — Corresponds to the minimum number of parameters (and thus of polynomial term) allowed.
- `nPmax` — Corresponds to the maximum number of parameters (and thus of polynomial) allowed.

Author(s)

Sylvain Mangiarotti

See Also

`autoGPoMoSearch`

Examples

```
#############
# Example 1 #
#############
# We build an example
allFilt <- list()
# For equation 1 (variable X1)
allFilt$Np1 <- 1         # only one formulation with one single parameter
# For equation 2 (variable X2)
allFilt$Np2 <- c(3, 4)   # two potential formulations, with respectively three and four parameters
# For equation 3 (variable X3)
allFilt$Np3 <- c(2, 4)   # two potential formulations, with respectively two and four parameters
# Formulations for variables Xi:
# For X1:
allFilt$X1 <- t(as.matrix(c(0,0,0,1,0,0,0,0,0,0)))
# For X2:
allFilt$X2 <- t(matrix(c(0,-0.85,0,-0.27,0,0,0,0.46,0,0,
                         0,-0.64,0,0,0,0,0,0.43,0,0),
                       ncol=2, nrow=10))
# For X3:
allFilt$X3 <- t(matrix(c(0, 0.52,  0, -1.22e-05,  0, 0, 0.99, 5.38e-05, 0, 0,
                         0, 0.52, 0, 0, 0, 0, 0.99, 0, 0, 0),
                       ncol=2, nrow=10))
# From these individual we can retrieve all possible formulations
findAllSets(allFilt, nS=c(3), nPmin=1, nPmax=14)
# if only formulations with seven maximum number of terms are expected:
findAllSets(allFilt, nS=c(3), nPmin=1, nPmax=7)
```

gloMoId *Global Model Identification*

Description

Algorithm for global modelling in polynomial and canonical formulation of Ordinary Differential Equations. Univariate Global modeling aims to obtain multidimensional models from single time series (Gouesbet & Letellier 1994, Mangiarotti et al. 2012). An example of such application can be found in Mangiarotti et al. (2014) For a multivariate application, see `GPoMo` (Mangiarotti 2015, Mangiarotti et al. 2016).

Example:
For a model dimension nVar=3, the global model will read:
$dX1/dt = X2$
$dX2/dt = X3$
$dX3/dt = P(X1, X2, X3)$.

Usage

```
gloMoId(series, tin = NULL, dt = NULL, nVar = NULL, dMax = 1,
  weight = NULL, show = 1, filterReg = NULL, winL = 9)
```

Arguments

`series`	The original data set: either a single vector corresponding to the original variable; Or a matrix containing the original variable in the first column and its successive derivatives in the next columns. In the latter case, for the construction of n-dimensional model, `series` should have $nVar + 1$ columns since one more derivative will be necessary to identify the model parameters. Variable `nVar` will be set equal to n. In the former case, that is when only a single vector is provided, the derivatives will be automatically recomputed. Therefore, the dimension nVar expected for the model has to be provided.
`tin`	Input date vector which length should correspond to the input time series.
`dt`	Sampling time of the input time series.
`nVar`	Number of variables considered in the polynomial formulation.
`dMax`	Maximum degree of the polynomial formulation.
`weight`	A vector providing the binary weighting function of the input data series (0 or 1). By default, all the values are set to 1.
`show`	Provide (2) or not (0-1) visual output during the running process.
`filterReg`	A vector that specifies the template for the equation structure (for one single equation). The convention defined by `poLabs` is used. Value is 1 if the regressor is available, 0 if it is not.
`winL`	Total number of points used for computing the derivatives of the input time series. This parameter will be used as an input in function `drvSucc` to compute the derivatives.

Value

A list of five elements :

`$init` The original time series and the successive derivatives used for the modeling.

`$filterReg` The structure of the output model. Value is 1 if the regressor is available, 0 if it is not. The terms order is given by function `poLabs`.

`$K` Values of the identified coefficients corresponding to the regressors defined in `filterReg`.

`$resTot` The variance of the residual signal of the model.

`$resSsMod` The variance of the residual signal of the closer submodels.

`$finalWeight` Weighting series after boundary values were removed.

Author(s)

Sylvain Mangiarotti, Laurent Drapeau, Mireille Huc

References

[1] Gouesbet G., Letellier C., Global vector-field reconstruction by using a multivariate polynomial L2 approximation on nets, Physical Review E, 49 (6), 4955-4972, 1994.
[2] Mangiarotti S., Coudret R., Drapeau L., & Jarlan L., Polynomial search and global modeling : Two algorithms for modeling chaos, Physical Review E, 86, 046205, 2012.
[3] Mangiarotti S., Drapeau L. & Letellier C., Two chaotic models for cereal crops observed from satellite in northern Morocco. Chaos, 24(2), 023130, 2014.
[4] Mangiarotti S., Low dimensional chaotic models for the plague epidemic in Bombay (1896-1911), Chaos, Solitons & Fractals, 81(A), 184-196, 2015.
[5] Mangiarotti S., Peyre M. & Huc M., A chaotic model for the epidemic of Ebola Virus Disease in West Africa (2013-2016). Chaos, 26, 113112, 2016.

See Also

`gPoMo`, `autoGPoMoSearch`, `autoGPoMoTest`, `poLabs`

Examples

```
#############
# Example 1 #
#############
# load data
data("Ross76")
```

```
tin <- Ross76[,1]
data <- Ross76[,2:3]

# Polynomial identification
reg <- gloMoId(data[0:500,2], dt=1/100, nVar=2, dMax=2, show=0)

#############
# Example 2 #
#############
# load data
data(NDVI)

# Definition of the Model structure
terms <- c(1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1)
poLabs(3,3,terms==1)
reg <- gloMoId(NDVI [,1:1], dt=1/125, nVar=3, dMax=3,
               show=0, filterReg=terms==1)


## Not run:
#############
# Example 3 #
#############
# load data
data("Ross76")
# time vector
tin <- Ross76[1:500,1]
# single time series
series <- Ross76[1:500,3]
# some noise is added
series[1:100] <- series[1:100] + 0.01 * runif(1:100, min = -1, max = 1)
series[301:320] <- series[301:320] + 0.05 * runif(1:20, min = -1, max = 1)
# weighting function
W <- tin * 0 + 1
W[1:100] <- 0  # the first hundred values will not be considered
W[301:320] <- 0  # twenty other values will not be considered either
reg <- gloMoId(series, dt=1/100, weight = W, nVar=3, dMax=2, show=1)
visuEq(3, 2, reg$K, approx = 4)
# first weight which value not equal to zero:
i1 = which(reg$finalWeight == 1)[1]
v0 <-  reg$init[i1,1:3]

reconstr <- numicano(nVar=3, dMax=2, Istep=5000, onestep=1/250, PolyTerms=reg$K,
                     v0=v0, method="ode45")
plot(reconstr$reconstr[,2], reconstr$reconstr[,3], type='l', lwd = 3,
                       main='phase portrait', xlab='time t', ylab = 'x(t)', col='orange')
# original data:
lines(reg$init[,1], reg$init[,2], type='l',
      main='phase portrait', xlab='x', ylab = 'dx/dt', col='black')
# initial condition
lines(v0[1], v0[2], type = 'p', col = 'red')
```

```
## End(Not run)
```

---

gPoMo *Generalized Polynomial Modeling*

Description

Algorithm for a Generalized Polynomial formulation of multivariate Global Modeling. Global modeling aims to obtain multidimensional models from single time series [1-2]. In the generalized (polynomial) formulation provided in this function, it can also be applied to multivariate time series [3-4].

Example:
Note that `nS` provides the number of dimensions used from each variable

case I
For `nS=c(2,3)` means that 2 dimensions are reconstructed from variable 1: the original variable `X1` and its first derivative `X2`), and 3 dimensions are reconstructed from variable 2: the original variable `X3` and its first and second derivatives `X4` and `X5`. The generalized model will thus be such as:
$dX1/dt = X2$
$dX2/dt = P1(X1, X2, X3, X4, X5)$
$dX3/dt = X4$
$dX4/dt = X5$
$dX5/dt = P2(X1, X2, X3, X4, X5)$.

case II
For `nS=c(1,1,1,1)` means that only the original variables `X1`, `X2`, `X3` and `X4` will be used. The generalized model will thus be such as:
$dX1/dt = P1(X1, X2, X3, X4)$
$dX2/dt = P2(X1, X2, X3, X4)$
$dX3/dt = P3(X1, X2, X3, X4)$
$dX4/dt = P4(X1, X2, X3, X4)$.

Usage

```
gPoMo(data, tin = NULL, dtFixe = NULL, dMax = 2, nS = c(3), winL = 9,
  weight = NULL, show = 1, verbose = 1, underSamp = NULL, EqS = NULL,
  IstepMin = 2, IstepMax = 2000, nPmin = 1, nPmax = 14,
  method = "lsoda")
```

Arguments

`data`	Input Time series: Each column is one time series that corresponds to one variable.
`tin`	Input date vector which length should correspond to the input time series.

dtFixe	Time step used for the analysis. It should correspond to the sampling time of the input data. Note that for very large and very small time steps, alternative units may be used in order to stabilize the numerical computation.
dMax	Maximum degree of the polynomial formulation.
nS	A vector providing the number of dimensions used for each input variables (see Examples 1 and 2). The dimension of the resulting model will be `nVar = sum(nS)`.
winL	Total number of points used for computing the derivatives of the input time series. This parameter will be used as an input in function `drvSucc` to compute the derivatives.
weight	A vector providing the binary weighting function of the input data series (0 or 1). By default, all the values are set to 1.
show	Provide (2) or not (0-1) visual output during the running process.
verbose	Gives information (if set to 1) about the algorithm progress and keeps silent if set to 0.
underSamp	Number of points used for undersampling the data. For `undersamp = 1` the complete time series is used. For `undersamp = 2`, only one data out of two is kept, etc.
EqS	Model template including all allowed regressors. Each column corresponds to one equation. Each line corresponds to one polynomial term as defined by function `poLabs`.
IstepMin	The minimum number of integration step to start of the analysis (by default `IstepMin = 10`).
IstepMax	The maximum number of integration steps for stopping the analysis (by default `IstepMax = 10000`).
nPmin	Corresponds to the minimum number of parameters (and thus of polynomial term) allowed.
nPmax	Corresponds to the maximum number of parameters (and thus of polynomial) allowed.
method	The integration technique used for the numerical integration. By default, the fourth-order Runge-Kutta method (`method = 'rk4'`) is used. Other methods such as 'ode45' or 'lsoda' may also be chosen. See package `deSolve` for details.

Value

A list containing:

`$tin` The time vector of the input time series

`$inputdata` The input time series

`$tfiltdata` The time vector of the filtered time series (boudary removed)

`$filtdata` A matrix of the filtered time series with its derivatives

`$okMod` A vector classifying the models: diverging models (0), periodic models of period-1 (-1), unclassified models (1).

`$coeff` A matrix with the coefficients of one selected model

`$models` A list of all the models to be tested `$mToTest1`, `$mToTest2`, etc. and all selected models `$model1`, `$model2`, etc.

`$tout` The time vector of the output time series (vector length corresponding to the longest numerical integration duration)

`$stockoutreg` A list of matrices with the integrated trajectories (variable `X1` in column 1, `X2` in 2, etc.) of all the models `$model1`, `$model2`, etc.

Author(s)

Sylvain Mangiarotti, Flavie Le Jean, Mireille Huc

References

[1] Gouesbet G. & Letellier C., 1994. Global vector-field reconstruction by using a multivariate polynomial L2 approximation on nets, Physical Review E, 49 (6), 4955-4972.
[2] Mangiarotti S., Coudret R., Drapeau L. & Jarlan L., Polynomial search and Global modelling: two algorithms for modeling chaos. Physical Review E, 86(4), 046205.
[3] Mangiarotti S., Le Jean F., Huc M. & Letellier C., Global Modeling of aggregated and associated chaotic dynamics. Chaos, Solitons and Fractals, 83, 82-96.
[4] S. Mangiarotti, M. Peyre & M. Huc, 2016. A chaotic model for the epidemic of Ebola virus disease in West Africa (2013-2016). Chaos, 26, 113112.

See Also

gloMoId, autoGPoMoSearch, autoGPoMoTest

autoGPoMoSearch, autoGPoMoTest, visuOutGP, poLabs, predictab, drvSucc

Examples

```
#Example 1
data("Ross76")
tin <- Ross76[,1]
data <- Ross76[,3]
dev.new()
out1 <- gPoMo(data, tin = tin, dMax = 2, nS=c(3), show = 1,
              IstepMax = 1000, nPmin = 9, nPmax = 11)
visuEq(3, 2, out1$models$model1, approx = 4)


## Not run:
#Example 2
data("Ross76")
tin <- Ross76[,1]
data <- Ross76[,3]
# if some data are not valid (vector 'weight' with zeros)
W <- tin * 0 + 1
W[1:100] <- 0
W[700:1500] <- 0
W[2000:2800] <- 0
```

```
W[3000:3500] <- 0
dev.new()
out2 <- gPoMo(data, tin = tin, weight = W,
                 dMax = 2, nS=c(3), show = 1,
                 IstepMax = 6000, nPmin = 9, nPmax = 11)
visuEq(3, 2, out2$models$model3, approx = 4)

## End(Not run)


## Not run:
#Example 3
data("Ross76")
tin <- Ross76[,1]
data <- Ross76[,2:4]
dev.new()
out3 <- gPoMo(data, tin=tin, dMax = 2, nS=c(1,1,1), show = 1,
              IstepMin = 10, IstepMax = 3000, nPmin = 7, nPmax = 8)
# the simplest model able to reproduce the observed dynamics is model #5
visuEq(3, 2, out3$models$model5, approx = 4) # the original Rossler system is thus retrieved

## End(Not run)

## Not run:
#Example 4
data("Ross76")
tin <- Ross76[,1]
data <- Ross76[,2:3]
# model template:
EqS <- matrix(1, ncol = 3, nrow = 10)
EqS[,1] <- c(0,0,0,1,0,0,0,0,0,0)
EqS[,2] <- c(1,1,0,1,0,1,1,1,1,1)
EqS[,3] <- c(0,1,0,0,0,0,1,1,0,0)
visuEq(3, 2, EqS, substit = c('X','Y','Z'))
dev.new()
out4 <- gPoMo(data, tin=tin, dMax = 2, nS=c(2,1), show = 1,
      EqS = EqS, IstepMin = 10, IstepMax = 2000,
      nPmin = 9, nPmax = 11)

## End(Not run)

## Not run:
#Example 5
# load data
data("TSallMod_nVar3_dMax2")
#multiple (six) time series
tin <- TSallMod_nVar3_dMax2$SprK$reconstr[1:400,1]
TSRo76 <- TSallMod_nVar3_dMax2$R76$reconstr[,2:4]
TSSprK <- TSallMod_nVar3_dMax2$SprK$reconstr[,2:4]
data <- cbind(TSRo76,TSSprK)[1:400,]
dev.new()
# generalized Polynomial modelling
out5 <- gPoMo(data, tin = tin, dMax = 2, nS = c(1,1,1,1,1,1),
```

```
              show = 0, method = 'rk4',
              IstepMin = 2, IstepMax = 3,
              nPmin = 13, nPmax = 13)

# the original Rossler (variables x, y and z) and Sprott (variables u, v and w)
# systems are retrieved:
visuEq(6, 2, out5$models$model347, approx = 4,
       substit = c('x', 'y', 'z', 'u', 'v', 'w'))
# to check the robustness of the model, the integration duration
# should be chosen longer (at least IstepMax = 4000)

## End(Not run)
```

GSproc *Gram-Schmidt procedure*

Description

Computes regressors coefficients using the Gram-Schmidt procedure.

Usage

```
GSproc(polyK, ivec, weight = NULL)
```

Arguments

polyK	One list including $Y and $phy with: $Y a matrix for which the ith column will be used to add one orthogonal vector to the (i-1)th vectors of the current orthogonal base; and $phy such as the current orthogonal base is given by the (i-1)th first columns of matrix polyK$phy.
ivec	Defines i, the current vector of polyK$Y and the current orthogonal base of pParam$phy.
weight	The weighing vector.

Value

uNew The model parameterization, that is: The residual orthogonal vector that can be included into the current orthogonal base. If the current base is empty, uNew is equal to the input vector of $Y; if the base is complete, uNew equals 0.

Author(s)

Sylvain Mangiarotti

NDVI — *A time series of vegetation index measured from satellite*

Description

A time series of 28 years of Normalized Difference Vegetation Index measured from space by the Advanced Very High Resolution Radiometer (AVHRR) sensor from 1982 to 2008 (see reference [1] for details).

Usage

```
NDVI
```

Format

An object of class `data.frame` with 9618 rows and 4 columns.

Author(s)

Sylvain Mangiarotti, Flavie Le Jean

References

[1] Mangiarotti S., Drapeau L. & Letellier C., 2014. Two chaotic models for cereal crops observed from satellite in northern Morocco.

numicano — *Numerical Integration of models in ODE of polynomial form*

Description

Function for the numerical integration of Ordinary Differential Equations of polynomial form.

Usage

```
numicano(nVar, dMax, Istep = 1000, onestep = 1/125, KL = NULL,
  PolyTerms = NULL, v0 = NULL, method = "rk4")
```

Arguments

nVar	Number of variables considered in the polynomial formulation.
dMax	Maximum degree of the polynomial formulation.
Istep	The number of integration time steps
onestep	Time step length
KL	Matrix formulation of the model to integrate numerically
PolyTerms	Vectorial formulation of the model (only for models of canonical form)
v0	The initial conditions (a vector which length should correspond to the model dimension nVar)
method	The integration method (See package deSolve), by default method = 'rk4'.

Value

A list of two variables:

$KL The model in its matrix formulation

$reconstr The integrated trajectory (first column is the time, next columns are the model variables)

Author(s)

Sylvain Mangiarotti

See Also

derivODE2, numinoisy

Examples

```
#############
# Example 1 #
#############
# For a model of general form (here the rossler model)
# model dimension:
nVar = 3
# maximal polynomial degree
dMax = 2
# Number of parameter number (by default)
pMax <- d2pMax(nVar, dMax)
# convention used for the model formulation
poLabs(nVar, dMax)
# Definition of the Model Function
a = 0.520
b = 2
c = 4
Eq1 <- c(0,-1, 0,-1, 0, 0, 0, 0, 0, 0)
Eq2 <- c(0, 0, 0, a, 0, 0, 1, 0, 0, 0)
Eq3 <- c(b,-c, 0, 0, 0, 0, 0, 1, 0, 0)
```

```
K <- cbind(Eq1, Eq2, Eq3)
# Edition of the equations
visuEq(nVar, dMax, K)
# initial conditions
v0 <- c(-0.6, 0.6, 0.4)
# model integration
reconstr <- numicano(nVar, dMax, Istep=1000, onestep=1/50, KL=K,
                      v0=v0, method="ode45")
# Plot of the simulated time series obtained
dev.new()
plot(reconstr$reconstr[,2], reconstr$reconstr[,3], type='l',
      main='phase portrait', xlab='x(t)', ylab = 'y(t)')

## Not run:
#############
# Example 2 #
#############
# For a model of canonical form
# model dimension:
nVar = 4
# maximal polynomial degree
dMax = 3
# Number of parameter number (by default)
pMax <- d2pMax(nVar, dMax)
# Definition of the Model Function
PolyTerms <- c(281000, 0, 0, 0, -2275, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               861, 0, 0, 0, -878300, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
# terms used in the model
poLabs(nVar, dMax, PolyTerms!=0)
# initial conditions
v0 <- c(0.54, 3.76, -90, -5200)
# model integration
reconstr <- numicano(nVar, dMax, Istep=500, onestep=1/250, PolyTerms=PolyTerms,
                     v0=v0, method="ode45")
# Plot of the simulated time series obtained
plot(reconstr$reconstr[,2], reconstr$reconstr[,3], type='l',
     main='phase portrait', xlab='x', ylab = 'dx/dt')
# Edition of the equations
visuEq(nVar, dMax, reconstr$KL)

## End(Not run)
```

`numinoisy` *Generates time series of deterministic-behavior with stochatic perturbations (measurement and/or dynamical noise)*

Description

Generates time series from Ordinary Differential Equations perturbed by dynamical and/or measurement noises

Usage

```
numinoisy(x0, t, K, varData = NULL, txVarBruitA = NULL,
  txVarBruitM = NULL, varBruitA = NULL, varBruitM = NULL, taux = NULL,
  freq = NULL, variables = NULL, method = NULL)
```

Arguments

x0	The initial conditions. Should be a vector which size must be equal to the model dimension dim(K)[2] (the number of variables of the model defined by matrix K).
t	A vector providing all the dates for which the output are expected.
K	The Ordinary Differential Equations used to model the dynamics. The number of column should correspond to the number of variables, the number of lines to the number of parameters following the convention defined by poLabs(nVar,dMax).
varData	A vector of size nVar providing the caracteristic variances of each variable of the dynamical systems in ODE defined by matrix K. If not provided, this variance is automatically estimated.
txVarBruitA	A vector defining the ratio of ADDITIVE noise for each variable of the dynamical system in ODE. The additive noise is added at the end of the numerical integration process. The ratio is defined relatively to the signal variance of each variable.
txVarBruitM	A vector defining the ratio of DYNAMICAL noise for each variable of the dynamical system in ODE. This noise is a perturbation added at each numerical integration step. The ratio is defined relatively to the signal variance of each variable.
varBruitA	A vector defining the variance of ADDITIVE noise for each variable of the dynamical system in ODE. The additive noise is added at the end of the numerical integration process.
varBruitM	A vector defining the variance of DYNAMICAL noise for each variable of the dynamical system in ODE. This noise is a perturbation added at each numerical integration step.
taux	Generates random gaps in time series. Parameter taux defines the ratio of data to be kept (e.g. for $taux = 0.75$, 75 percents of the data are kept).
freq	Subsamples the time series. Parameter freq defines the periodicity of data kept (e.g. for $freq = 3$, 1 data out of 3 is kept).
variables	Defines which variables must be generated.
method	Defines the numerical integration method to be used. The fourth-order Runge-Kutta method is used by default (method = 'rk4'). Other method may be used (such as 'ode45' or 'lsoda'), see function ode from package deSolve for details.

Value

A list of two variables:

`$donnees` The integrated trajectory (first column is the time, next columns are the model variables)

`$bruitM` The level of dynamical noise

`$bruitA` The level of additive noise

`$vectBruitM` The vector of the dynamical noise used to produce the time series

`$vectBruitA` The vector of the additive noise used to produce the time series

`$ecart_type` The level standard deviation

Author(s)

Sylvain Mangiarotti, Malika Chassan

Examples

```
#############
# Example 1 #
#############
# Rossler Model formulation
# The model dimension
nVar = 3
 # maximal polynomial degree
dMax = 2
a = 0.520
b = 2
c = 4
Eq1 <- c(0,-1, 0,-1, 0, 0, 0, 0, 0, 0)
Eq2 <- c(0, 0, 0, a, 0, 0, 1, 0, 0, 0)
Eq3 <- c(b,-c, 0, 0, 0, 0, 0, 1, 0, 0)
K <- cbind(Eq1, Eq2, Eq3)
# Edit the equations
visuEq(nVar, dMax, K)
# initial conditions
v0 <- c(-0.6, 0.6, 0.4)
# output time required
timeOut = (0:1000)/50
# variance of additive noise
varBruitA = c(0,0,0)^2
# variance of multiplitive noise
varBruitM = c(2E-2, 0, 2E-2)^2
# numerical integration with noise
intgr <- numinoisy(v0, timeOut, K, varBruitA = varBruitA, varBruitM = varBruitM, freq = 1)
# Plot of the simulated time series obtained
dev.new()
plot(intgr$donnees[,2], intgr$donnees[,3], type='l',
      main='phase portrait', xlab='x(t)', ylab = 'y(t)')
```

```
dev.new()
par(mfrow = c(3, 1))
plot(intgr$donnees[,1], intgr$donnees[,2], type='l',
      main='phase portrait', xlab='x(t)', ylab = 'y(t)')
lines(intgr$donnees[,1], intgr$vectBruitM[,2]*10, type='l',
      main='phase portrait', xlab='x(t)', ylab = 'e(t)*10', col='red')
plot(intgr$donnees[,1], intgr$donnees[,3], type='l',
      main='phase portrait', xlab='x(t)', ylab = 'y(t)')
lines(intgr$donnees[,1], intgr$vectBruitM[,3]*10, type='l',
      main='phase portrait', xlab='x(t)', ylab = 'e(t)*10', col='red')
plot(intgr$donnees[,1], intgr$donnees[,4], type='l',
      main='phase portrait', xlab='x(t)', ylab = 'y(t)')
lines(intgr$donnees[,1], intgr$vectBruitM[,4]*10, type='l',
      main='phase portrait', xlab='x(t)', ylab = 'e(t)*10', col='red')
```

`odeBruitMult2` *For the numerical integration of ordinary differential equations with dynamical noise.*

Description

A subfunction for the numerical integration of Ordinary Differential Equations provided in a generic polynomial form. Model formulation follows the convention defined by function `poLabs`.

Usage

```
odeBruitMult2(x0, t, K, varData = NULL, txVarBruitM = NULL,
  varBruitM = NULL, method = NULL)
```

Arguments

`x0`	Initial conditions
`t`	All the dates for which the result of the numerical integration of the model must be provided
`K`	A matrix providing the model description: each column corresponds to one equation which polynomial organisation is following the convention defined by function `poLabs`.
`varData`	A vector of size `nVar` providing the caracteristic variances of each variable of the dynamical systems in ODE defined by matrix `K`. If not provided, this variance is automatically estimated.
`txVarBruitM`	A vector defining the ratio of DYNAMICAL noise for each variable of the dynamical system in ODE. This noise is a perturbation added at each numerical integration step. The ratio is defined relatively to the signal variance of each variable.
`varBruitM`	A vector defining the variance of DYNAMICAL noise for each variable of the dynamical system in ODE. This noise is a perturbation added at each numerical integration step.

method	Numerical method used in the integration process. (see ode function in deSolve package for details).

Author(s)

Sylvain Mangiarotti, Malika Chassan

See Also

numinoisy

---

p2dMax *p2dMax : provides the maximum polynomial degree* dMax *given the number of variables* nVar *and the number of possible polynomial terms* pMax.

Description

Find the maximum polynomial degree dMax given the number of polynomial terms pMax and the system dimension nVar.

Usage

```
p2dMax(nVar, pMaxKnown)
```

Arguments

nVar	Number of variables considered in the polynomial formulation.
pMaxKnown	The number of polynomial terms

Value

dMax The maximum polynomial degree

Author(s)

Sylvain Mangiarotti, Laurent Drapeau

See Also

gloMoId, gPoMo, poLabs

Examples

```
#############
# Example 1 #
#############
# Maximum polynomial degree ?
# number of variables:
nVar <- 3
# size of the polynomial vector:
pMax <- 10
# The maximal polynomial degree used for coding the polynomial is:
p2dMax(nVar,pMax)

#############
# Example 2 #
#############
# for pMax = 462 and nVar = 6, then dMax is:
p2dMax(6,462)
# indeed:
length(poLabs(nVar=6, dMax=5))
```

`paramId` *For parameter Identification*

Description

Estimate the polynomial coefficients.

Usage

```
paramId(allForK, drv, weight)
```

Arguments

`allForK`	The list of input parameters
`drv`	The derivative (on the equation left hand)
`weight`	The weighting series

Value

`allForK` The initial list completed with the model parameters.

Author(s)

Sylvain Mangiarotti

poLabs *Polynomial labels order*

Description

Defines the order of the polynomial labels given the number of variables nVar and the maximum polynomial degree dMax.

Usage

```
poLabs(nVar, dMax, findIt = NULL, Xnote = "X")
```

Arguments

nVar	The number of variables
dMax	The maximum degree allowed in the formulation
findIt	A vector of selected terms.
Xnote	Enables to defines the notation used for the variable, by default Xnote = 'X'.

Value

lbls A vector of characters. Each element is the expression of one polynomial term, such as $X_1^2 X_3 X_4$

Author(s)

Sylvain Mangiarotti

See Also

visuEq

Examples

```
#Regressor order for three variables \eqn{(X1,X2,X3)} (nVar = 3) for a maximum
#polynomial degree equal to 2 (dMax = 2): poLabs(3,2)
#and for two variables only : poLabs(2,2)

# For a quadratic equation of two variables,
# the polynomial \deqn{P(X1,X2) = 0.5 + 0.3 X1 -0.25 X1 X2}
# could thus be written as a vector Pvec such as:
Pvec = c(0.5, 0, 0, 0.3, -0.25, 0)
# considering the convention corresponding to
poLabs(2,2)
# Indeed:
poLabs(2, 2, findIt = Pvec!=0)
# An alternative notation can be used with parameter Xnote
poLabs(2, 2, findIt = Pvec!=0, Xnote = 'w')
```

```
# or also
poLabs(2, 2, findIt = Pvec!=0, Xnote = c('x','y'))
```

---

predictab — *Estimate the models performance obtained with* `GPoMo` *in term of predictability*

Description

The algorithm aims to estimate automatically the forecasting performances of the models obtained with `gPoMo`.

Usage

```
predictab(ogp, fullt = NULL, fulldata = NULL, hp = NULL, Nech = 50,
  show = 1, selecmod = NULL, id = 1, selV = 1, na.rm = FALSE)
```

Arguments

`ogp`	The output list obtained from function `gPoMo`.
`fullt`	Time vector of the data set for which predictability will be tested
`fulldata`	Data set for which predictability will be tested
`hp`	Time vector of the horizon of prediction
`Nech`	Number of simulations
`show`	Provide (2) or not (0-1) visual output during the running process.
`selecmod`	A vector of the model selected.
`id`	The type of model to identify. `id = 1` corresponds to unidentified models, that is, potentialy chaotic.
`selV`	Selected variable for the analysis
`na.rm`	Indicates if the `NA` should be removed (`na.rm = TRUE`) or not (`na.rm = FALSE`).

Value

`ErrmodAll` A list of matrix `$Predmod1`, `$Predmod2`, etc. and `$Errmod1`, `$Errmod2`, etc. providing respectively the forecasting and the forecasting error of models 1, 2, etc. Each column corresponds to one simulation starting from a specific initial condition. Each line corresponds to one horizon of prediction. Vectors corresponding to the initial condition time `tE` and the horizon of prediction `hpE` are also provided in `$tE` and `$hpE`, respectively.

Author(s)

Sylvain Mangiarotti, Mireille Huc

Examples

```
# load data
data("Ross76")
# time vector
tin <- Ross76[seq(1, 3000, by = 8), 1]
# single time series
data <- Ross76[seq(1, 3000, by = 8), 3]
# dev.new()
# plot(tin, data, xlab = 'time', ylab = 'y(t)')

# global modelling
# results are put in list outputGPoM
outputGPoM <- gPoMo(data[1:300], tin = tin[1:300], dMax = 2, nS=c(3),
                    show = 0, method = 'rk4',
                    nPmin = 10, nPmax = 12,
                    IstepMin = 150, IstepMax = 151)
#
visuOutGP(outputGPoM)

###########################
# and test predictability #
###########################
outpred <- predictab(outputGPoM, hp = 15, Nech = 30)

# manual visualisation of the outputs (e.g. for model 1):
dev.new()
image(outpred$tE, outpred$hpE, t(outpred$Errmod1),
xlab = 't', ylab = 'hp', main = 'Errmod1')
```

---

regOrd — *Generate the conventional order for polynomial terms in a the polynomial formulation*

Description

Generate the conventional order of the polynomial terms for the polynomial description. It is formulated as a matrix of exponents: Each column of the matrix (a,b,c, ...) corresponds to a product of the `nVar` available variables X1, X2, X3, etc., that is, $X1^a X2^b X3^c$, etc.

Usage

```
regOrd(nVar, dMax)
```

Arguments

`nVar`	The number of variables
`dMax`	The maximum degree allowed in the formulation

Value

A matrix of exponents. Each column corresponds to one polynomial term. Each line correspond to the exponent of one variable. For example, a column of three exponents `(0,2,1)` corresponds to the monomial `X1^0 * X2^2 * X3^1`, that is $X2^2X3$.

Author(s)

Sylvain Mangiarotti

See Also

`poLabs`

`regSeries` *Estimates the monomial time series*

Description

Creates time series by multiplying given time series among them.

Usage

```
regSeries(nVar, dMax, series, pReg = NULL)
```

Arguments

`nVar`	Number of variables considered in the polynomial formulation.
`dMax`	Maximum degree of the polynomial formulation.
`series`	A matrix containing the original time series from which the monomials are built. Each column corresponds to one given variable.
`pReg`	A matrix filled, for each column, with powers of time series used to create.

Value

`rpFull` A matrix of time series. Each column corresponds to one regressor such as $X_1^2X_3X_4$

Author(s)

Sylvain Mangiarotti

Examples

```
data(TSallMod_nVar3_dMax2)
sprottK <- as.matrix(TSallMod_nVar3_dMax2$SprK$reconstr)[,2:4]
dMax <- 2
nVar <- dim(sprottK)[2]

#Example 1
polySeries2 <- regSeries(nVar, dMax, sprottK)

#Example 2
p <- c(1,3,1)
polySeries2 <- regSeries(nVar, dMax, sprottK, pReg=p)
```

`Rossler-1976 data set` *Time series of the Rossler-1976 system*

Description

The Rössler system is the 3-dimensional chaotic system
$dx/dt = -y - z$
$dy/dt = x + ay$
$dz/dt = b + z(x - c)$,
discovered by Otto E. Rössler in 1976 [1]. The following parameters and initial conditions were used to produce the present data set:
a = 0.520, b = 2, c = 4
and (x0, y0, z0) = (-0.04298734, 1.025536, 0.09057987).
The following four columns are provided:
(1) time t, (2) x(t), (3) y(t) and (4) z(t).
For this parameterization, the Rössler system produces a chaotic behavior characterized by a regime non-coherent in phase (oscillations duration can be very different from one oscillation to another).

Usage

```
Ross76
```

Format

An object of class `deSolve` (inherits from `matrix`) with 4000 rows and 4 columns.

Author(s)

Sylvain Mangiarotti, Flavie Le Jean, Malika Chassan, Laurent Drapeau, Mireille Huc.

References

[1] O. Rössler, 1976. An Equation for Continuous Chaos, Physics Letters, 71A, 2-3, 155-157.

`RosYco` *Twelve Rossler-1976 time series (exclusively variable y)*

Description

Twelve independant Rossler-1976 time series (variable y). The parameters used to generate the time series correspond to a phase coherent behavior. Details can be found in [1]

Usage

```
RosYco
```

Format

An object of class `matrix` with 3000 rows and 12 columns.

Details

Another set of time series of the Rossler-1976 chaotic system

Author(s)

Sylvain Mangiarotti, Flavie Le Jean.

References

[1] Mangiarotti S., Le Jean F., Huc M. & Letellier C., Global Modeling of aggregated and associated chaotic dynamics. Chaos, Solitons and Fractals, 83, 82-96.

`svrlTS` *A data set for the global modeling of time series in association*

Description

This data set aims to test the global modelling technique when several time series of different sizes are available. Four time series are provided, all derived from the Rössler-1976 system.

Usage

```
svrlTS
```

Format

An object of class `list` of length 4.

Author(s)

Sylvain Mangiarotti, Mireille Huc.

References

S. Mangiarotti, F. Le Jean, M. Huc & C. Letellier, 2016. Global modeling of aggregated and associated chaotic dynamics, Chaos, Solitons & Fractals, 83, 82-96.

TSallMod_nVar3_dMax2 data set

Time series of three-dimensional chaotic sytems (for vignette `VII_Retro-Modelling`*)*

Description

A list of matrix providing the time series in a list named `TSallMod_nVar3_dMax2` of eighteen three-dimensional chaotic systems: Lorenz-1963 ($L63), Rössler-1976 ($R76), Burke & shaw 1981 ($BS81), Lorenz-1984 ($L84), Nosé & Hooer 1986 ($NH86), Genesio & Tosi 1992 ($GT92), Spott systems 1994 ($SprF, $SprH, $SprK, $SprO, $SprP, $SprG, $SprM, $SprQ, $SprS), Chlouverakis & Sprott 2004 ($CS2004), Li 2007 ($Li2007) and the Cord system by Aguirre & Letellier 2012 ($Cord2012). Time series are provided in a matrix in which each column corresponds to one variable of the dynamical systems.

Usage

```
TSallMod_nVar3_dMax2
```

Format

An object of class `list` of length 18.

Author(s)

Sylvain Mangiarotti, Mireille Huc.

References

References for the systems are provided in vignette ‘VII_retro-modelling‘.

visuEq *Displays the models Equations*

Description

Displays the model equations for a polynomial model which description is provided as a matrix `K`, each column corresponding to one equation. The coefficients of the polynomial terms are given following the order defined by function `poLabs`.

Usage

```
visuEq(nVar, dMax, K, substit = 0, approx = FALSE)
```

Arguments

`nVar`	The number of variables
`dMax`	The maximum degree allowed in the formulation
`K`	A matrix providing the model description: each column corresponds to one equation which polynomial organisation is following the convention defined by function `poLabs`.
`substit`	Applies subtitutions to the default values: for `substit = 0` (default value), variables are chosen as `X1`, `X2`, ... for `substit = 1`, variable `X1`, `X2`, ... will be replaced by `x`, `y`, `z`, ... for `substit = 2`, the codes provides a LaTex-like formulation of the model. The variables name can also be defined explicitely as follows: for `substit = c('x', 'H', 'T1')`, variables `X1`, `X2`, `X3` ... will be replaced by `x`, `H` and `T1`.
`approx`	The number of extra digits to be used: for `approx = FALSE` (default value) digits are edited with double precision; for `approx = TRUE`, only the minimum number of digits is edited (in order to have all the terms different from 0) for `approx` = 1, 2, etc. then respectively 1, 2, etc. digits are added to the minimum number of digits corresponding to `approx = TRUE`.

Author(s)

Sylvain Mangiarotti

Examples

```
#EQUATIONS VISUALISATION
# number of variables:
nVar <- 3
# maximum polynomial degree:
dMax <- 2
# polynomial organization:
poLabs(nVar,dMax)
# model construction
KL = matrix(0, ncol = 3, nrow = 10)
```

```
KL[1,1] <- KL[2,2] <- 1
KL[4,1] <- -1
KL[5,3] <- -0.123456789
# Equations visualisation:
# (a) by default, variables names X1, X2, X3 are used
visuEq(nVar, dMax, KL)
# (b) for susbstit=1, variables names x, y, y are used instead
visuEq(nVar, dMax, KL, approx = TRUE, substit=1)
# (c) the name of the variables can also be chosen manualy
visuEq(nVar, dMax, KL, approx = 3, substit=c('U', 'V', 'W'))

# A canonical model can be provided as a single vector
polyTerms <- c(0.2,0,-1,0.5,0,0,0,0,0,0)
visuEq(3,2,KL)
```

---

visuOutGP *visuOutGP : get a quick information of gPoMo output*

Description

The algorithm aims to get a quick information about the outputs obtained with gPoMo.

Usage

```
visuOutGP(ogp, selecmod = NULL, id = 1, prioMinMax = "data",
  opt3D = "TRUE")
```

Arguments

ogp	The output list obtained from gPoMo.
selecmod	A vector of the selected model. Maximum 24 models can be presented at the same time.
id	The type of model to identify. `id = 1` corresponds to the unidentified models, that is, potentialy chaotic models).
prioMinMax	Gives the priority for the plots among: `"data"`, `"model"`, `"dataonly"` and `"modelonly"`.
opt3D	Provides a 3D plot (x,y,z) when `opt = 'TRUE'` (the `rgl` library is required).

Value

A Matrix describing the terms composing each model by row. The first row corresponds to the model detection (1 unclarified, 2 diverging, 0 is fixed point, -n with n an integer, is period-n cycle')

Author(s)

Sylvain Mangiarotti

Examples

```
# load data
data("Ross76")
# # time vector
tin <- Ross76[seq(1, 3000, by = 8), 1]
# single time series
data <- Ross76[seq(1, 3000, by = 8), 3]
dev.new()
plot(tin, data, type = 'l')
# global modelling
# results are put in list outputGPoM
outputGPoM <- gPoMo(data, tin=tin, dMax = 2, nS=c(3), show = 0,
                    nPmin = 9, nPmax = 12, method = 'rk4',
                    IstepMin = 200, IstepMax = 201)
visuOutGP(outputGPoM)
```

---

wInProd *Weighted inner product*

Description

Computes weighted inner products.

Usage

```
wInProd(A1, A2, weight = NULL)
```

Arguments

A1	The input matrix 1.
A2	The input matrix 2.
weight	The weighting vector.

Value

`inP` The weighted inner product.

Examples

```
############
#Example 1 #
############
A1 = c(0,1,2,0,1,3)
A2 = c(1,2,0,0,4,1)
wInProd(A1, A2)

############
```

```
#Example 2 #
############
A1 = c(0,1,2,0,1,3)
A2 = c(1,2,0,0,4,1)
w = c(0,0,0,1,1,1)
wInProd(A1, A2, weight = w)
```

Index