NNX package Manual

V1.00	5/12/2019	Dongheng Jing
V1.01	5/14/2019	Dongheng Jing

Table of Contents

Abstract

NNX package is created to accomplish approximation of data using NNX method.

NNXapprox is the main function of evaluation data using NNX method. Other functions are necessary modules to successfully run NNXapprox. Users are encouraged to make modifications to NNXapprox to have optimal performance.

NNXapprox

```
[y_approx,a0,A] = NNXapprox(input_data, train_data,
n_neuron, activation_func, order)
```

Inputs:

input_data	- [m-by-t] X1 to Xm in time t
train_data	- [n-by-t] Y1 to Yn in time t
n_neuron	- number of neurons in hidden layer
activation_func	- activation function ('tansig' or 'logsig')
order	- degree that approximation reaches

Outputs:

y_approx	- approximated output of neural network given input
a0	- constant (0th order) term in polynomial
A	- coefficients of polynomial

This function returns approximated value and coefficients of polynomials.
Input_data and train_data are data inputs to train the neural net. N_neuron and activation_func are parameters of neural net. Order is user-defined, up to which degree the polynomial is to approximate the neural network model.

Example:
1 DOF underdamped spring (see example_spring.m in package)

input data: [2-by-1000] x(t-1) and v(t-1)
train data: [2-by-1000] x(t) and v(t)
n_neuron: 2
activation_func: 'tansig'
order: 3

y_approx: [2-by-1000] x_approx(t) and v_approx(t)
a0: [2-by-1] constant/ 0^{th} order term
A: [3-by-1] cell

A{1}: coefficient of 1^{st} order [2-by-2]
A{2}: coefficient of 2^{nd} order [2-by-3]
A{3}: coefficient of 3^{rd} order [2-by-4]

y_approx is given by $y_{approx} = a_0 + A_1 X^{\otimes 1} + A_2 X^{\otimes 2} + A_3 X^{\otimes 3}$ where,
A_1, A_2, A_3 are A{1}, A{2}, A{3};
$X^{\otimes n}$ is metricized cross term of X in nth degree (to find them, see varCrossMat (pg.5) or varCrossVec (pg.6)).

findCoef

```
[a0,A] = findCoef(net,activation_func,order)
```

Inputs:

net	- neural network model
activation_func	- activation function ('tansig' or 'logsig')
order	- degree that approximation reaches

Outputs:

a0	- constant (0th order) term in polynomial
A	- coefficients of polynomial

This function finds coefficients of polynomial given trained neural network model.
net contains trained model of a neural net. It includes necessary info such as weight and bias. Activation_func is parameter of neural net. Order is user-defined, up to which degree the polynomial is to approximate the neural network model.

Example:
1 DOF underdamped spring (see example_findcoef.m in package)

net: trained neural net model of 1 DOF underdamped spring case
activation_func: 'tansig'
order: 3

a0: [2-by-1] constant/ 0^{th} order term
A: [3-by-1] cell

A{1}: coefficient of 1^{st} order [2-by-2]
A{2}: coefficient of 2^{nd} order [2-by-3]
A{3}: coefficient of 3^{rd} order [2-by-4]

iwgenerate

```
iw = iwgenerate(IW,order)

Inputs:
     IW        - input weights of a neural net
     order     - up to which degree approximation is to
Outputs:
     iw   - metricized iw to order'th degree
```

This function finds metricized input weights up to nth degree. The NNX method uses $IW^{\otimes n}$ to find coefficients of approximation polynomial. The output of this function is a cell of [order-by-1], each of which is a metricized $IW^{\otimes n}$.

Example:
See example_iwgenerate.m in package

IW: net.IW{1};
order: 3;

iw: [3-by-1] cell

iw{1}: $IW^{\otimes 1}$ [2-by-2]
iw{2}: $IW^{\otimes 2}$ [2-by-3]
iw{3}: $IW^{\otimes 3}$ [2-by-4]

varCrossMat

```
     crossTerm = varCrossMat(X,order)

Inputs:
     X          - data of dim m [m-by-t]
     order      - up to which order
Outputs:
     crossTerm - metricized tensor of cross term of X
```

This function finds metricized cross term up to nth order. The cross terms are denoted as $X^{\otimes n}$. It is a tensor of [m-by-m-…-by-m-by-t] (number of ms = n). The function metricizes each cross term in tensor form and returns a cell of [n-by-1], each of which is the matricized tensor of corresponding order.

Example:
See example_varCrossMat.m in package

X: [2-by-1000]
order: 3;

crossTerm: [3-by-1] cell

crossTerm{1}: $X^{\otimes 1}$ [2-by-1000]
crossTerm{2}: $X^{\otimes 2}$ [3-by-1000]
crossTerm{3}: $X^{\otimes 3}$ [4-by-1000]

varCrossVec

```
crossTerm = varCrossVec(varargin)

Inputs:
     Xs         - vector [1-by-t]
     order      - up to which order
Outputs:
     crossTerm - metricized tensor of cross term of X
```

This function is similar to varCrossMat. It takes m+1 input. The last one is order. The first m inputs are all [1-by-t] vector representing Xn. X is a combination X1 to Xm.

Example:
See example_varCrossVec.m in package

X1: [1-by-1000]
X2: [1-by-1000]
order: 3;

crossTerm: [3-by-1] cell

crossTerm{1}: $X^{\otimes 1}$ [2-by-1000]
crossTerm{2}: $X^{\otimes 2}$ [3-by-1000]
crossTerm{3}: $X^{\otimes 3}$ [4-by-1000]

outerprod

```
comb = outerprod(xdim,order)

Inputs:
     xdim       - dimension of data input X (m)
     order      - up to which order
Outputs:
     comb       - structure with two elements
     comb.mult            - multiplicity of a certain combination
     comb.listofComb      - combination
```

This function finds coefficients unique combinations of X to nth order. X is denoted by $X_1, X_2, \dots X_m$. This function returns coefficients and its corresponding combination.

Example:
See example_outerprod.m in package

xdim: 2
order: 3

comb: struct
comb.mult:

$$\begin{matrix} 1 \\ 3 \\ 3 \\ 1 \end{matrix}$$

comb.listofComb:

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 2 & 2 \\ 2 & 2 & 2 \end{matrix}$$

Representing:
$(X_1 + X_2)^3 = 1 \times X_1X_1X_1 + 3 \times X_1X_1X_2 + 3 \times X_1X_2X_2 + 1 \times X_2X_2X_2$
or
metricized tensor $X^{\otimes 3}$, where X is [X1, X2].

logsigDerivative

```
     de = logsigDerivative(x,n)

Inputs:
     x    - value at which function logsig be evaluated
     n    - to nth order derivative
Output:
     de   - nth order derivative of logsig evaluated at x
```

This function finds nth order derivative of logsig function ($\frac{1}{1+e^{-x}}$).

Example:

x: $\begin{bmatrix} 1.1 \\ 0.2 \end{bmatrix}$

n: 3

de: $\begin{bmatrix} -0.0233 \\ -0.1201 \end{bmatrix}$

tansigDerivative

```
     de = tansigDerivative(x,n)

Inputs:
     x    - value at which function tansig be evaluated
     n    - to nth order derivative
Output:
     de   - nth order derivative of tansig evaluated at x
```

This function finds nth order derivative of tansig function (tanh).

Example:

x: $\begin{bmatrix} 1.1 \\ 0.2 \end{bmatrix}$

n: 3

de: $\begin{bmatrix} 0.6627 \\ -1.6974 \end{bmatrix}$