### Clojure code examples

This page describes a basic set of demonstration scripts for using the toolkit in Clojure. The .clj files (ready for use in Clojure REPL) can be found at `demos/clojure/examples` in the svn or main distributions (from the V1.1 release at a future point in time ...).

Please see [UseInClojure](https://code.google.com/p/information-dynamics-toolkit/) for instructions on how to begin using the Java toolkit from inside clojure. Most importantly, the `project.clj` file in this directory includes a reference to the JIDT jar file hosted at `me.lizier/jidt` on the clojars.org repository:

```clojure
(defproject me.lizier/jidt-clojure-samples "1.0-SNAPSHOT"
  :description "Java Information Dynamics Toolkit (JIDT) clojure samples"
  :url "https://code.google.com/p/information-dynamics-toolkit/"
  :license
    { :name "GNU GPL v3" :
      :url "http://www.gnu.org/licenses/gpl.html"
      :distribution :repo }
  :dependencies [[org.clojure/clojure "1.6.0"] [me.lizier/jidt "LATEST"]])
```

This page contains the following code examples. They can be run as `lein repl < example1TeBinaryData.clj` (Yes, I know it's not great Clojure code, but all that's important is that shows you how to get started with JIDT in Clojure!):

- **Example 1 - Transfer entropy on binary data**
- **Example 2 - Transfer entropy on multidimensional binary data**
- **Example 3 - Transfer entropy on continuous data using kernel estimators**
- **Example 4 - Transfer entropy on continuous data using Kraskov estimators**
- More to come ...

#### Example 1 - Transfer entropy on binary data

example1TeBinaryData.clj - Simple transfer entropy (TE) calculation on binary data using the discrete TE calculator:

```clojure
; Import relevant classes:
(import infodynamics.measures.discrete.TransferEntropyCalculatorDiscrete)

(let ; Generate some random binary data.
     [sourceArray (int-array (take 100 (repeatedly #(rand-int 2))))
      destArray (int-array (cons 0 (butlast sourceArray))) ; shifts sourceArray by 1
      sourceArray2 (int-array (take 100 (repeatedly #(rand-int 2))))]

; Create TE calculator
      teCalc (TransferEntropyCalculatorDiscrete 2 1)
)

; Initialise the TE calculator and run it:
      (.initialise teCalc)
      (.addObservations teCalc sourceArray destArray)
      (println "For copied source, result should be close to 1 bit : "
               (.computeAverageLocalOfObservations teCalc))
      (.initialise teCalc)
      (.addObservations teCalc sourceArray2 destArray)
      (println "For random source, result should be close to 0 bits : "
               (.computeAverageLocalOfObservations teCalc))
)
```

#### Example 2 - Transfer entropy on multidimensional binary data

example1TeBinaryData.clj - Simple transfer entropy (TE) calculation on binary data using the discrete TE calculator:
example2TeMultidimBinaryData.clj - Simple transfer entropy (TE) calculation on multidimensional binary data using the discrete TE calculator.

This example shows how to handle multidimensional arrays from Clojure to Java.

```clojure
(def relevant classes
  (import info.dynamics.measures.discrete.TransferEntropyCalculatorDiscrete)

(let
  [Create many columns in a multidimensional array (2 rows by 100 columns),
   where the next time step (row 2) copies the value of the column on the left
   from the previous time step (row 1):
   (row 1 (int-array (take 100 (repeatedly #"(rand-int 2)))))
   (row 2 (int-array (cons (aget row 1 99) (butlast row 1)))) ; shifts row 1 by 1
   twoDTimeSeriesClojure (into array (map int-array [row 1 row 2]))
   Create TE calculator
tecal (TransferEntropyCalculatorDiscrete. 2 1)]

; Initialise the TE calculator and run it:
(let
  [ initialised tecal]
  ; Add observations of transfer across one cell to the right per time step:
  (doObservations tecal tecal twoDTimeSeriesClojure)
  (println "The result should be close to 1 bit here, since we are executing copy operations of what is effectively a r
   (.computeAverageLocalOfObservations tecal))
)

Example 3 - Transfer entropy on continuous data using kernel estimators

example3TeContinuousDataKernel.clj - Simple transfer entropy (TE) calculation on continuous-valued data using the (box) kernel-estimator TE calculator.

; Import relevant classes:
(Import info.dynamics.measures.continuous.kernel.TransferEntropyCalculatorKernel)
(Import java.util.Random)
(def rg (Random.))

(let
  [numObservations 1000
   covariance 0.4
   Generate some random normalised data.
   sourceArray (double-array (take numObservations (repeatedly #"(nextGaussian rg)")))
   destArray (double-array (cons 0
     (map +
       (map partial covariance) (butlast sourceArray)))
     (map partial (- covariance 1)) (double-array (take (- numObservations 1) (repeatedly #"(nextGaussian rg)")))
     tecal (TransferEntropyCalculatorKernel. )]

; Set up the calculator
(.setProperty tecal "NORMALISE" "true")
(.initialise tecal 1 0.5) ; Use history length 1 (Schreiber k=1), kernel width of 0.5 normalised units
(.setObservations tecal sourceArray destArray)
; For copied source, should give something close to expected value for correlated Gaussians:
(print "TE result " (.computeAverageLocalOfObservations tecal)
   " expected to be close to " ($(Math/log (- 1 (- 1 (* covariance covariance))))) (Math/log 2))
   " for these correlated Gaussians but biased upward")
(.initialise tecal ) ; Initialise leaving the parameters the same
(.setObservations tecal sourceArray2 destArray)
; For random source, it should give something close to 0 bits
(print "TE result " (.computeAverageLocalOfObservations tecal)
   " expected to be close to 0 bits for these uncorrelated Gaussians but will be biased upward")

; We can get insight into the bias by examining the null distribution:
(def nullDist (.computeSignificance tecal 100))
(print "Null distribution for unrelated source and destination ",
   "(i.e. the bias) has mean " (.getMeanOfDistribution nullDist)
   " and standard deviation " (.getStdDevDistribution nullDist))
)

Example 4 - Transfer entropy on continuous data using Kraskov estimators

example4TeContinuousDataKraskov.m - Simple transfer entropy (TE) calculation on continuous-valued data using the Kraskov-estimator TE calculator.

; Import relevant classes:
(import infodynamics.measures.continuous.kraskov.TransferEntropyCalculatorKraskov)
(IMPORT java.util.Random)
(def rg (Random.))

(let
  [numObservations 1000
   covariance 0.4]
  ; Generate some random normalised data.
  sourceArray (double-array (take numObservations (repeatedly #(.nextGaussian rg))))
  destArray (double-array (cons 0
                         (map +
                          (map (partial * covariance) (butlast sourceArray))
                          (map (partial * (- covariance 1))
                               double-array (take (- numObservations 1) (repeatedly #(.nextGaussian rg))))
                         sourceArray2)
                         (double-array (take numObservations (repeatedly #(.nextGaussian rg))))
                         teCalc (TransferEntropyCalculatorKraskov.))])

; Set up the calculator
(setProperty teCalc "k" "4") ; Use Kraskov parameter K=4 for 4 nearest points
(initialise teCalc 1) ; Use history length 1 (Schreiber k=1)

; Perform calculation with correlated source:
(setObservations teCalc sourceArray destArray)
; Note that the calculation is a random variable (because the generated
; data is a set of random variables) - the result will be of the order
; of what we expect, but not exactly equal to it; in fact, there will
; be a large variance around it.
(println "TE result ".computeAverageLocalOfObservations teCalc
   " nats expected to be close to " (Math/log (/ 1 (- 1 (* covariance covariance))))
   " nats for these correlated Gaussians")

; Perform calculation with uncorrelated source:
(initialise teCalc ) ; Initialise leaving the parameters the same
(for random source, it should give something close to 0 bits
(println "TE result ".computeAverageLocalOfObservations teCalc
   " nats expected to be close to 0 nats for these uncorrelated Gaussians")

; We can also compute the local TE values for the time-series samples here:
; (See more about utility of local TE in the CA demos)
(def localTE (.computeLocalOfPreviousObservations teCalc))

(println "Notice that the mean of locals."
   (/ (reduce + localTE) (- numObservations 1))
   " nats, equals the previous result")
)

Acknowledgements

A big thank you to Matthew Chadwick for showing me how to do this and getting things up and running with clojure and leiningen.